Infinite-Instruct: Synthesizing Scaling Code instruction Data with **Bidirectional Synthesis and Static Verification**

Anonymous ACL submission

Abstract

Traditional code instruction data synthesis methods suffer from limited diversity and poor logic. We introduce Infinite-Instruct, an auto-005 mated framework for synthesizing high-quality question-answer pairs, designed to enhance the code generation capabilities of large language models (LLMs). The framework focuses on improving the internal logic of synthesized problems and the quality of synthesized code. First, "Reverse Construction" transforms code snippets into diverse programming problems. Then, through "Backfeeding Construction," keywords in programming problems are structured into 015 a knowledge graph to reconstruct them into programming problems with stronger internal logic. Finally, a cross-lingual static code analysis pipeline filters invalid samples to ensure data quality. Experiments show that on mainstream code generation benchmarks, our finetuned models achieve an average performance improvement of 21.70% on 7B-parameter models and 36.95% on 32B-parameter models. Using less than one-tenth of the instruction finetuning data, we achieved performance comparable to the Qwen-2.5-Coder-Instruct. Infinite-Instruct provides a scalable solution for LLM 028 training in programming. We open-source the datasets used in the experiments, including both unfiltered versions and filtered versions via static analysis. The data are available at https://anonymous.4open.science/r/ Infinite-Instruct-dataset-DF90

1 Introduction

004

007

012

017

023

027

034

042

Recent advances in Large Language Models (LLMs) have revolutionized code generation capabilities through instruction tuning(Zhang et al., 2023). The effectiveness of this paradigm fundamentally depends on high-quality instructionresponse pairs that enable models to comprehend and execute diverse programming tasks(Chen et al., 2021). However, scaling the acquisition of

such instruction data presents a significant bottleneck(Wang et al., 2024), as conventional manual annotation methods are both resource-intensive and constrained in their coverage.

043

045

047

049

051

054

055

057

060

061

062

063

064

065

066

067

068

069

070

071

072

073

074

075

077

078

079

Prior research has explored various approaches to automatic instruction synthesis. Early attempts leveraging Self-Instruct(Wang et al., 2022) demonstrated the potential of utilizing existing LLMs in a teacher-student framework for instruction synthesis. Despite their initial success, these approaches exhibited inherent limitations due to biases inherited from both the foundation models and the limited seed examples(Yu et al., 2023). While subsequent developments like Evol-Instruct(Luo et al., 2023)(Taori et al., 2023) introduced more sophisticated instruction evolution mechanisms, they remained bounded by predetermined evolution rules. OSS-Instruct(Wei et al., 2023) achieved a significant advancement by pioneering the utilization of real-world code snippets as inspiration sources. However, this approach lacked robust quality assurance mechanisms and structural validation protocols.

To address these limitations, we present Infinite-Instruct, a novel bidirectional framework for codeoriented data synthesis that extends the capabilities of OSS-Instruct. The framework operates through two complementary mechanisms: Reverse Construction and Backfeeding Construction. The Reverse Construction component transforms diverse code fragments into comprehensive programming tasks by analyzing code structure and functionality. The Backfeeding mechanism establishes vocabularies of tasks, instructions, and knowledge points through a complex knowledge graph architecture, transforming phrases into programming tasks. This bidirectional approach enables the framework to infinitely utilize code fragments and maintained vocabularies, continuously leveraging the model to generate unlimited high-quality training data.

Our methodology incorporates four key inno-

vations with specific implementation details: (1) 084 a bidirectional code-prompt evolution framework that combines structural code analysis with se-086 mantic understanding. (2) a knowledge-graphenhanced backfeeding mechanism that creates a closed-loop optimization by extracting and refining keywords from synthetic prompts. (3) a compre-090 hensive seven-dimensional quality assessment protocol implemented through a cross-language static code analysis pipeline, which effectively filters invalid samples and ensures instruction validity. (4) systematic knowledge integration through structured vocabulary tables that maintain consistent terminology and conceptual relationships.

2 Related Work

100

101

104

105

106

108

109

110

111

112

113

114

115 116

117

118

119

121

122

123

124

125

126

127

129

130

131

132

133

2.1 Advances in Automated Instruction Synthesis

Automated instruction synthesis has made breakthroughs in recent years, contributing to the development of open-source models, represented by Nemotron-4(Adler et al., 2024). (Wang et al., 2023) pioneered Self-Instruct, which synthesizes instruction prompts by bootstrapping from a small seed set of human-written examples. This method allowed LLMs to generate both new instructions and corresponding instances, reducing dependence on manual annotation. However, Self-Instruct's synthesized prompts often lacked complexity and diversity compared to expert-created instructions. (Xu et al., 2023) addressed this limitation with Evol-Instruct in their work WizardLM, introducing evolutionary synthesis method that systematically increase complexity through operations like adding constraints, deepening context, and increasing reasoning steps. While Evol-Instruct successfully generated more challenging prompts, it operated primarily in a direction from code snippets to prompts, evolving existing instructions without systematically creating novel prompt types.

For code domains, (Luo et al., 2023) developed WizardCoder, adapting evolutionary instruction synthesis specifically for programming tasks with code-specific constraints and debugging scenarios. (Wei et al., 2023) proposed OSS-INSTRUCT, which uniquely leverages open-source code snippets as inspiration for generating diverse coding problems, mitigating LLM bias through real-world code references. Despite these advances, existing prompt synthesis approaches remain limited by their evolution patterns, insufficient coverage across knowledge domains, limited quality assurance beyond basic validity checks, and lack of closed-loop optimization mechanisms.

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

152

153

154

155

156

157

158

159

160

161

162

163

166

167

169

170

171

172

173

174

175

176

177

178

179

180

181

182

2.2 Static Code Analysis

Static code analysis is a method that identifies potential issues without executing the program, with linter systems being its powerful implementation. These systems support multiple programming languages including JavaScript, Python, Ruby, C/C++, and Java, detecting syntax errors, code smells, and style deviations, while sometimes identifying performance bottlenecks and security vulnerabilities.

In recent years, with the increasingly widespread application of large language models (LLMs) in code generation and comprehension, static code analysis has evolved to evaluate and enhance the quality of AI-generated code. For instance, the Llama 3 series (Grattafiori et al., 2024) employs parsers and linters to guarantee syntactic accuracy in all generated code, detecting errors such as syntax mistakes, use of uninitialized variables, nonimported functions, code style issues, and typing errors. Similarly, Qwen2.5-Coder (Hui et al., 2024) incorporates abstract syntax tree parsing to filter out code snippets containing parsing errors. It indicates that LLMs for code generation are increasingly integrating multilingual static code analysis capabilities during their datas construction phase to ensure code quality across multiple programming languages.

3 Method

We propose Infinite-Instruct, a code-oriented bidirectional instruction synthesis method that generates high-quality and diverse instruction data. It revolves the inter-evolution between code and problems, forming a closed-loop optimization generation system through the reverse construction ("Code \rightarrow Problem") and the backfeeding construction ("Keyword \rightarrow Problem"). The complete data synthesis process can be seen in Figure 1.

During Reverse Construction, we leverage a large language model (LLM) to analyze numerous code snippets and generate diverse programming problems. These problems undergo complexity adjustment and textual rewriting to ensure quality and variety. During BackFeeding Construction, a knowledge graph is established to present the semantic relationships among keywords extracted from the synthesized problems. At last, the gener-



Figure 1: Automated Prompt Synthesis Technology Roadmap.

ated instruction data is evaluated and filtered by a seven-dimensional quality assessment framework.

183

184

186

190

191

192

193

194

196

199

204

207

210

212

3.1 Reverse Construction: Deriving Problems from Code

We extract multilingual code from Magicoder-OSS-Instruct-75K (Wei et al., 2023) as our seed repository, it reduces the need of data cleaning. We randomly extract code snippets of 5-20 lines multiple times, ensuring these snippets maintain functional completeness and independence after deduplication.

We design specialized prompt templates for different types of problems (see A.1 for details). By analyzing the characteristics of each code snippet, we guide GPT-40-0806 to consider possible application scenarios, identify valuable problem points, and improve question quality accordingly. To increase the difficulty of questions, we rewrite problems from six directions, including adding constraints, depth extension, concretization, reasoning refinement, input enhancement, and innovative changes(see A.2). It formulates questions with richer constraints and contextual information that more closely resemble real-world application scenarios.

We also discover that synthesized questions usually lack expression diversity (e.g., most code generation questions begin with "Please help me write a..."). To address this issue, we rewrite the text by analyzing sentence logical relationships and restructuring paragraph structures, making the questions more aligned with real users' questioning styles(refer to A.3).

213

214

215

216

217

218

219

221

222

224

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

Through this process, We synthesize approximately 30K problems. The advantage of this approach is that it breaks through the limitations of predefined datasets and can generate more diverse questions. At the same time, the generated questions are closely related to the code, ensuring their solvability.

3.2 Backfeeding Construction: Closed-Loop Optimization from Keywords to Problems

While generating programming problems based on randomized code snippets yields diversity, it often lacks logical structure and pedagogical focus.

We define three types of keywords for a programming problem(tasks, instructions, and knowledge points). Tasks and instructions are extracted from "reverse" synthesis problems (refer to A.4), and after deduplication, we construct a vocabulary containing keywords from all three categories and generate random combinations. Initial knowledge points sourced from online programming tutorials (like "Runoob Tutorial") and official SQL documentation (including MySQL, Hive, SQLite, PostgreSQL, Oracle, etc.), supplements with secondary tags from the field of computer science. Each programming language has approximately 10K words per dimension on average, with about 20K secondary tags collected from various fields

247

251

254

256

261

265

270

272

276

277

278

281

293

of computer science (such as algorithms, software engineering, etc.).

To improve the effectiveness of the problems, we use the concept of knowledge graphs to construct prompts as follows: First, we extract entities from the keywords and categorize them into tasks, instructions, and knowledge points, forming a list of nodes. Second, we analyze the relationships between these nodes and construct triples to represent the connections between them, then extract keyword combinations with clear semantic associations and rigorous logical structures from these triples. We deduplicate the final keyword groups using cosine similarity (threshold 0.8) and use these groups to synthesize seven types of "backfeed" problems, including Code Generation, Code Understanding, Knowledge-based Question, Code Completion, Code Optimization, Debug, and Modify Code as required. We use GPT-40-0806 model as the engine to drive the entire process, ultimately synthesize approximately 20K instructions. The prompt template for "backfeed" problem synthesis can be found in A.5 and A.6.

3.3 Prompt Filtering

For both "Reverse" and "Backfeeding" construction, we introduce seven quality metrics of Arena-Hard-Auto (Li et al., 2024) to filter out high-quality question, including the metrics of specificity, domain knowledge, complexity, problem solving ability, creativity, technical accuracy, and practical application. We use GPT-40-0806 to evaluate the questions that the system generates. For each question, we conduct three assessments, and the final score is the average of the three assessments. We filter out data with scores less than 6 points. See A.7 for specific cue words. Finally, we filter out 10K "reverse" data and 10K "backfeed" data.

Three datasets are included in our analysis: Reverse, Backfeed, and OSS-Instruct. Each dataset is assessed using a complexity score that ranges from 1 to 10.A detailed explanation of the scoring system can be found in A.8. To ensure consistency, 10,000 samples are randomly selected from each dataset for evaluation. As shown in Figure 2, the results indicate:

Backfeed: highest complexity (mean 6.83, median 8.0), standard deviation 1.54, difficulty is more concentrated at high levels.

OSS-Instruct: lowest complexity (mean 3.17, median 3.0), suggesting that open source instructions are generally simpler and more straightfor-

ward.

Reverse: moderate complexity (mean 5.85, median 6.0), largest standard deviation (1.95), a more balanced distribution of instruction difficulty. 294

295

296

299

300

301

302

303

304

305

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

329

330

331

These differences are important guidelines for training strategies: Backfeed is suitable for training advanced reasoning ability, OSS is suitable for basic function training, and Reverse is suitable for comprehensive training due to its balanced characteristics. Taken together, the complexity differences of different datasets can provide guidance for hybrid training strategies, choosing the appropriate ratio of dataset combinations according to the needs of different application scenarios, such as mixing Backfeed and Reverse to optimize model performance.

3.4 Response Quality Enhancement Based on Static Syntax Analysis

We generate corresponding responses using GPT-40-0816 for the above prompts. To ensure the quality of generated data, we employ language-specific static analysis tools as shown in Table 1. Each tool is configured with customized rule settings categorized into three levels: disabled, error, and info. Disabled rules represent unnecessary checks based on our experience (e.g., @typescript-eslint/quotes in ESLint, which enforces consistent quote style). Error-level rules identify critical issues according to each linter's severity classifications, while remaining checks are set to info level. When static analysis detects error-level issues in generated code, we remove the entire prompt-response pair from our dataset rather than attempting to fix them automatically. This strict filtering approach ensures that only syntactically valid and high-quality code samples remain in our final dataset, while still accommodating acceptable stylistic variations flagged at the info level.

Programming Language	Static Analysis Tool
Python	PyLint
JavaScript	ESLint
Java	Checkstyle
C/C++	Clang-Tidy
SQL	SQLFluff

Table 1: Static analysis tools used for different programming languages

Our screening identifies a substantial number 332 of responses with syntax errors—approximately 333



Figure 2: Complexity Score Comparison

10,000 out of 100,000 generated responses contained syntax issues, including incorrect indentation, non-code content mixed within code blocks, and erroneous class instantiation methods (detailed cases are provided in Appendix B).

4 Experiments

335

336

337

341

342

343

344

345

347

348

353

357

364

368

4.1 Experimental Settings

We select Qwen2.5-Coder-7B-Base and Qwen2.5-Coder-32B-Base models as baseline to investigate the effectiveness of our proposed dataset. We transfer the SFT data to the same type of schema, which is a one-prompt and one-response format. We set the loss mask parameter of prompt to 0.0 and the response loss mask to 1.0. When training, we use the Mariana (Zou et al., 2014) platform to implement the SFT training stage. Meanwhile, we make the training process parallelized on 4 machines, each equipped with 8 Nvidia A100-SXM-80GB GPUs. We set the global batchsize to a fixed integer 512, and run for 3 global epochs for each model to reach the endpoint. Also, we set the initial learning rate at 2×10^{-5} , and simultaneously we set the learning rate to decay in a cosine annealing manner.

In the evaluation phase, we adapt a greedy generation strategy, specifically setting the temperature to 0 and the Top-K value to 1, ensuring deterministic outputs during each evaluation. For each evaluation instance across all models, we perform exactly one sampling.

We prepare multiple datasets as follows. The original Magicoder-OSS-Instruct-75K dataset (generated by ChatGPT 3.5 turbo 1106) and its distilled version using the newer GPT-4o-0806 model, Magicoder-OSS-Instruct-75K-GPT-40 (to eliminate performance differences caused by model upgrades).Reverse-100K, a dataset of 100,000 coding task instances that we create using our reverse construction method, and its filtered version Reverse-90K that contains 90,000 instances after static code analysis. Backfeed-100K, a 100,000instance coding dataset that we build through our backfeeding method, and its filtered version Backfeed-90K that contains 90,000 instances. 369

370

371

372

373

374

376

377

378

381

382

383

384

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

We select the following benchmarks to evaluate the coding ability of the models, MultiPL-E HumanEval (Cassano et al., 2023), MBPP (Austin et al., 2021), MBPP+ (Liu et al., 2023), Aider, BigCodeBench (Zhuo et al., 2024), Live-CodeBench (Jain et al., 2024). These benchmarks include mostly all the mainstream programming languages and cover a wide range of coding problems from easy to hard.

At the same time, in order to observe the impact on general problem solving abilities while monitoring changes in the model's code capabilities, we use Arena hard (Li et al., 2024). , a high-quality and reliable benchmark to test our trained models.

4.2 Experimental Results and Analysis

4.2.1 Improvements in Code Generation Capabilities

The experimental results reveal several key insights regarding the effectiveness of different data construction and enhancement strategies.

Effectiveness of the Reverse Construction. Table 2 demonstrates that, for the 7B model, the Reverse Construction method significantly improves performance across all benchmarks. Reverse 90K achieves the highest score of 45.11 on the Aider dataset, and both Reverse 90K and Reverse 100K outperform the baseline OSS-75K, with average improvements of 18.81% and 18.37%, respec-

Table 2: Experimental Results Based on Qwen2.5-Coder-7B-Base

Model	MBPP	MBPP plus	MPL-E Human	Aider	BigCode Bench	LiveCode Bench	Arena Hard
OSS-75K	74.80	70.63	61.93	43.61	42.02	32.12	8.96
OSS-75K-GPT-4o	73.40	70.90	64.77	42.86	48.42	34.50	42.85
Reverse 100K	76.40	72.49	67.05	44.36	49.74	32.82	48.09
Reverse 90K	75.80	71.69	67.28	45.11	50.61	34.22	47.76
Backfeed 100K	71.60	67.99	55.24	42.11	50.61	35.34	47.70
Backfeed 90K	75.80	73.54	65.80	41.35	50.09	34.08	49.56
Rev+Back 200K	76.60	72.75	67.22	42.11	50.79	36.47	46.94
Rev+Back 180K	77.80	73.81	67.35	42.86	50.96	37.03	49.59
Qwen-2.5-Coder-7B-Instruct	83.50	71.7	76.5	55.6	41.0	18.2	-

Table 3: Experimental Results Based on Qwen2.5-Coder-32B-Base

Model	MBPP	MBPP plus	MPL-E Human	Aider	BigCode Bench	LiveCode Bench	Arena Hard
OSS-75K	81.40	74.34	67.58	48.87	52.46	34.22	19.71
OSS-75K-GPT-4o	82.60	76.98	76.12	57.14	55.35	50.49	67.40
Reverse 100K	83.20	76.46	76.48	57.89	57.28	49.79	66.95
Reverse 90K	83.40	75.13	76.65	59.40	57.72	49.09	67.38
Backfeed 100K	80.80	76.98	76.08	57.89	56.93	51.19	68.23
Backfeed 90K	81.20	75.40	76.55	57.14	56.84	51.19	67.93
Rev+Back 200K	82.40	76.46	76.55	56.39	56.67	50.49	70.17
Rev+Back 180K	83.20	75.13	76.65	56.39	56.32	49.51	67.97
Qwen-2.5-Coder-32B-Instruct	90.2	75.1	79.4	60.9	49.6	31.4	-

Note: MPL-E Human = MultiPLE-E Humaneval. Rev+Back 200K = Reverse 100K + Backfeed 100K. Rev+Back 180K = Reverse 90K + Backfeed 90K. Bolded numbers represent the highest scores of all models.

tively. Table 3 highlights similar trends for the 32B model, where Reverse 90K achieves state-ofthe-art results on MPL-E Human (76.65), Aider (59.40), BigCode Bench (57.72), and Arena Hard (67.38). The method delivers an average improvement of 36.64% over OSS-75K, further demonstrating its effectiveness in enhancing model performance across diverse evaluation datasets.

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

Effectiveness of the Backfeeding Construction. On the Qwen2.5-Coder-7B-Base model, as shown in Table 2, the Backfeed strategy shows notable improvements over the OSS-75K.For the Qwen2.5-Coder-32B-Base model, Backfeed 100K also demonstrates significant advantages, as seen in Table 3. It achieves state-of-the-art results on multiple datasets, including MBPP plus (76.98) and LiveCode Bench (51.19), as shown in Table 3. Notably, it outperforms the larger Qwen2.5-Coder-32B-Instruct model on BigCode Bench (56.93 vs. 49.6) and achieves comparable performance on Aider (57.89 vs. 60.9), highlighting its efficiency and robustness even with smaller training datasets. These results demonstrate the scalability and versatility of Backfeed Construction across diverse

evaluation scenarios.

Superior Performance of Combined Strategies. The combination of Reverse 90K + Backfeed 90K performs best on the 7B model. Table 2 shows that the combined strategy after static analysis surpasses Qwen2.5-Coder-7B-Instruct on MBPP plus, BigCode Bench, LiveCode Bench, and Arena Hard, making it the strongest performing model among all. On other evaluation sets, the combined strategy also demonstrates better scores than OSS-75K-GPT-4o.This is sufficient to demonstrate that combined strategies exhibit stable performance across all tests, reducing the volatility associated with single strategies. 429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

Impact of Static Analysis Filtering. As can be seen from Table 4 and Table 5,Datasets filtered through syntax checking (linter) generally perform better than their unfiltered versions. On the 7B model, Backfeed 90K shows significantly greater improvement than Backfeed 100K (19.30% vs 8.75%). On the 32B model, the trends in performance are similar(9.56% vs -1.54%).It demonstrates the important influence of high-quality instruction data.Based on the statistics from the



Figure 3: Benchmark with MBPP

MBPP and MBPP+ evaluation datasets, data that has undergone static syntax checking tends to achieve higher scores than data that has not been checked. The specific performance improvement effects can be referred to in Figure 3 and Figure 4.

 Table 4: Average Performance Improvement Compared

 to the original OSS-Instruct dataset

Model	Based on Qwen2.5-coder-7B	Based on Qwen2.5-coder-32B
Reverse 100K	18.37%	33.66%
Reverse 90K	18.81%	36.64%
Backfeed 100K	8.75%	32.84%
Backfeed 90K	19.30%	36.95%
Rev+Back 200K	21.17%	36.82%
Rev+Back 180K	21.70%	34.95%

Table 5: Average Performance Improvement Compared to OSS-75K-GPT-40 dataset

Model	Based on	Based on
	Qwen2.5-coder-7B	Qwen2.5-coder-32B
Reverse 100K	7.62%	1.48%
Reverse 90K	9.59%	2.71%
Backfeed 100K	-1.54%	0.91%
Backfeed 90K	9.56%	2.59%
Rev+Back 200K	12.45%	2.62%
Rev+Back 180K	11.92%	2.18%

Note:

458

459

460

461

462

Rev+Back 200K = Reverse 100K + Backfeed 100K

Rev+Back 180K = Reverse 90K + Backfeed 90K Bolded numbers represent the highest scores of all models.

4.2.2 Comparative Analysis with Benchmark Methods

Enhanced Diversity. Our instruction data covers various code-related tasks, as shown in Figure 5. The diversities enable stable model performance

Score Changes Before and After Linter



Figure 4: Benchmark with MHPP

across different evaluation sets, especially in the complex MultiPle-EHumaneval.

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480



Figure 5: Percentage of topic types

Quality Improvement. Magicoder-OSS-Instruct-75K-GPT-40, shows improvements over the original Magicoder-OSS-Instruct-75K dataset in most benchmarks. It indicates that as the performance of the LLM used to generate responses improves, the quality of the dataset tends to also increase. Datasets filtered through static analysis (Reverse 90K and Backfeed 90K) further enhance performance, indicating that code quality filtering is crucial for training effectiveness.

Greater Efficiency. According to the official report released by the Qwen team (Hui et al., 2024), the Qwen2.5-coder-instruct model was fune-tuned on Qwen2.5-coder-Base model with millions of high-quality prompts. After that, they also employed mixed tuning and Direct Preference Opti-

mization (DPO) during post-training. In contrast 481 to the millions of instruction samples and complex 482 post-training methods utilized by the Qwen team, 483 our approach achieve comparable or even supe-484 rior performance using at most 200K data samples. 485 This outcome further validates the effectiveness 486 of our data construction strategies. Our synthetic 487 strategies perform exceptionally well on the 32B 488 model, outperforming the Qwen2.5-coder-instruct 489 model in multiple tests. In the MBPP+ test, the 490 Backfeed 100K strategy enable the 32B model to 491 reach 76.98, higher than Qwen2.5-coder-instruct's 492 75.1. On BigCodeBench and LiveCodebench, all 493 synthetic data strategies achieve evaluation results 494 superior to Qwen2.5-coder-instruct, and obtain 495 very close performance on other tests. 496

> **Model Scale Effects** Experiments clearly show that the 32B model can better utilize high-quality synthetic data compared to the 7B model.On the 32B model, the average improvements compare to OSS-Instruct-GPT-40 is smaller (1-3%), but this is still significant considering OSS-Instruct-GPT-40 already gets high-performance on benchmarks.The 7B model is more sensitive to data quality, with appropriate strategy combinations bringing over 10% improvement.

5 Conclusion

497

498

499

500

501

504

507

510

511

512

513

516

517

518

519

522

524

Existing data generation methods for training large language models (LLMs) in code generation are constrained by high costs and limited diversity, hindering scalability and effectiveness. To address this, we propose Infinite-Instruct, a framework that automates high-quality instruction data synthesis through Reverse Construction and Backfeeding Construction, while leveraging cross-language static analysis to ensure data quality by filtering invalid samples. Our experiments show that Infinite-Instruct achieves 21.70% performance gains on 7B models and 34.95% on 32B models, with static analysis improving MBPP test accuracy from 74.80% to 77.80% on 7B models. These results demonstrate the scalability of automated data generation, the critical role of static analysis, and the ability to achieve state-of-the-art performance with less training data.

6 Limitations

527Despite the significant advantages our method528demonstrates in generating code SFT data, several529noteworthy limitations remain:

 Simplistic Difficulty Adjustment Mechanism: The current method relies primarily on adjusting the number of keywords to increase problem complexity, which may lead to unpredictable complexity outcomes.

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573 574

575

576

577

578

- 2. Inherent Limitations of Static Analysis: Static analysis techniques cannot comprehensively capture all types of code issues, particularly logical errors, algorithmic complexity problems, and architectural design flaws. These limitations constrain the upper bound of code quality improvement and may result in certain obscure yet serious problems being overlooked during the screening process.
- 3. Imbalanced Multi-language Support: Although our method supports multiple programming languages, the depth of support varies across different languages. Mainstream languages (such as Python and JavaScript) receive more comprehensive support, while some niche yet important domain-specific languages have weaker support.

Future research will focus on four key directions: enhancing chain of thoughts generation to construct code question-answer pairs containing detailed reasoning steps; integrating dynamic and static analysis to comprehensively assess code quality; developing an adaptive difficulty tuning mechanism to match the needs of different proficiency models; and enhancing cross-language knowledge transfer for efficient multilingual data generation. These explorations will improve the quality and efficiency of code SFT data and lay a more solid foundation for the application of large-scale language models in the code domain.

References

- Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, and 1 others. 2024. Nemotron-4 340b technical report. *Computing Research Repository*, arXiv:2406.11704.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *Computing Research Repository*, arXiv:2108.07732.
- Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney,

580

- 589

- 596
- 598

- 610 611

613

615 616 617

- 618 619
- 621
- 624

625

- 627

633

- Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, and 1 others. 2023. Multipl-e: a scalable and polyglot approach to benchmarking neural code generation. IEEE Transactions on Software Engineering, 49(7):3675–3691.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. Computing Research Repository, arXiv:2107.03374.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. Computing Research Repository, arXiv:2407.21783.
 - Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. Computing Research Repository, arXiv:2409.12186.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. Computing Research Repository, arXiv:2403.07974.
- Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2024. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. Computing Research Repository, arXiv:2406.11939.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in Neural Information Processing Systems, 36:21558–21572.
- Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. Computing Research Repository, arXiv:2306.08568.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.
- Ke Wang, Jiahui Zhu, Minjie Ren, Zeming Liu, Shiwei Li, Zongye Zhang, Chenkai Zhang, Xiaoyu Wu, Qiqi Zhan, Qingjie Liu, and 1 others. 2024. A survey on data synthesis and augmentation for large language models. Computing Research Repository, arXiv:2410.12896.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. Computing Research Repository, arXiv:2212.10560.

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

- Yufei Wang, Jiayi Zheng, Can Xu, Xiubo Geng, Tao Shen, Chongyang Tao, and Daxin Jiang. 2022. Knowda: All-in-one knowledge mixture model for data augmentation in low-resource nlp. *Computing* Research Repository, arXiv:2206.10265.
- Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Empowering code generation with oss-instruct. Computing Research Repository, arXiv:2312.02120.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. Computing Research Repository, arXiv:2304.12244.
- Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. 2023. Large language model as attributed training data generator: A tale of diversity and bias. Advances in Neural Information Processing Systems, 36.
- Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and 1 others. 2023. Instruction tuning for large language models: A survey. Computing Research Repository, arXiv:2308.10792.
- Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, and 1 others. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. Computing Research Repository, arXiv:2406.15877.
- Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, and Bin Xiao. 2014. Mariana: tencent deep learning platform and its applications. Proc. VLDB Endow., 7(13).

Prompts Α

Integrated Reverse Question Generation A.1 Template

1 # Task 678 2 As a senior full-stack engineer, you 679 need to design a high-quality [PROBLEM_TYPE] programming problem. You need to draw inspiration from 682 input random code snippets to create 683 problems that fit real-world 684 scenarios. 685 686 4 # Characteristics of [PROBLEM_TYPE] 687 Problems 688 689 ## From the problem requirements:

```
690
          6 1. [PROBLEM_TYPE specific requirement
                description]
                                                         46
               [Common question patterns or keywords
          7 2.
               ٦
694
          8 3. [Assessment focus]
                                                         47
         9
696
         10 ## From the code requirements:
         11 1. [Whether code must be provided]
697
                                                         48
         12 2. [Code feature requirements]
                                                         49
         13 3. [Relationship requirements with
                                                         50
700
               original code snippet]
701
         14
                                                         51
         15 # Workflow
703
         16 1. Code Snippet Feature Analysis:
                                                         52
704
         17 Analyze the programming language used,
                core functionality, implementation
705
                methods, technical characteristics
706
                                                         53
707
                and difficulty level, understand the
708
                code design thinking and
                application scenarios.
710
                                                         54
         18
711
         19 2. Inspiration Brainstorming:
712
         20 Based on code characteristics,
713
                brainstorm [PROBLEM_TYPE related
714
                brainstorming focus].
715
         21
716
         22 3. Initial Problem Design:
717
         23 Design the initial problem framework
718
               based on previous analysis and
719
                thinking, combined with #
720
                Characteristics of [PROBLEM_TYPE]
721
                Problems. The problem framework
722
                includes problem description and [
723
                related code requirements].
724
         24
725
         25 4. Problem Review and Optimization:
726
         26 Review initial problems based on the
797
               following points:
               - [PROBLEM_TYPE specific review
728
         27
729
                   points]
730
               - Whether the code meets requirements
         28
               - Whether the code is identical to
731
         29
732
                   the original code snippet, if so,
733
                    it needs modification
734
               - Whether the problem difficulty/
         30
735
                   logic is appropriate
736
         31
737
         32 5. Problem Improvement and Revision:
738
         33 Modify and improve based on issues found
739
                in review, optimize problem
740
                description and code.
741
         34
742
         35 6. Formal Problem Output: Strictly
743
                output problems according to the
744
                format below:
         36 [Programming Language]: [Programming
745
746
                language that should be used for the
747
                 answer]
748
         37 [Problem Description]: [Describe your
749
                created problem in easy-to-
                understand language]
750
751
         38
752
         39 # Output Format
753
         40 1. Code Snippet Feature Analysis:
754
         41 2. Inspiration Brainstorming:
755
         42 3. Initial Problem Design:
756
         {\scriptstyle 43} 4. Problem Review and Optimization:
         44 5. Problem Improvement and Revision:
         45 6. Formal Problem Output: Strictly
759
               output problems according to the
```

	format below: [Programming Language]: [Programming	760 761
	language that should be used for the	762
,	answer] [Problem Description]: [Describe your	763 764
	created problem in easy-to-	765
	understand language]	766
)	# Notes	767 768
	1. Do not provide any solution ideas or	769
	hints in the problem. 2. Do not show any content in the	770 771
	problem that might suggest answers.	772
2	3. Problems should be written in clear	773
	and concise language that is easy to understand.	774 775
	4. Written problems need to draw	776
	inspiration from input random code snippets but **cannot directly use	777 778
	the input random code snippets**.	779
ŀ	[PROBLEM_TYPE specific additional	780
	notes]	781
	When using the integrated template, replace the	782
	placeholders with the specific requirements for	783
	each problem type:	784
	1. Code Generation	785
	Characteristics from problem requirements:	786
	Characteristics from problem requirements.	100
	• Problems must be solved by writing code	787
	• Problems can have multiple solutions	700
	• Problems can have multiple solutions	788
	• Keywords: implement, develop, build, write	789
	Characteristics from code requirements:	790
	• No code should be provided in the problem	791
	Inspiration Brainstorming focus:	792
	• Possible application scenarios and problem	793
	directions	794
	• Different implementation angles	795
	Valuable problem points	796
	Problem Review additional points:	797
	• Unreasonable: Directly using the input ran-	798
	dom code snippet as the answer to the problem	799
	• Not complex enough: Problem requirements	800
	are too simple, lacking divergent thinking	801
	Additional notes:	802
	• Problems must demonstrate being solved	803
	through code writing, but do not provide any	804
	form of code examples	805

806	2. Code Understanding	Pros and cons analysis	844
807	Characteristics from problem requirements:	Inspiration Brainstorming focus:	845
808 809	 Typically starts with verbs such as "Explain", "Analyze", "Describe", "Clarify", etc. 	• Identify one core concept as the test point	846
810	• Sometimes requires explaining specific pro-	Problem Review additional points:	847
811	gramming concepts or technical details	 Verify avoidance of code implementation ten- dency 	848 849
812 813	• Some questions may include the code's exe- cution output, asking to explain the reasoning	• Whether it forms complete knowledge context	850
814	behind the output		
815	Characteristics from code requirements:	4. Code Completion Characteristics from problem requirements:	851 852
816 817	• Not allowed to be identical to the provided random code snippet	• Words such as "complete", "fill in", "perfect", "supplement" will be used to describe the re-	853 854
818	• Followed by a complete code block, which	quirements	855
819 820	ideally contains multiple complete functions Inspiration Brainstorming focus:	• Focuses on examining the ability to under- stand existing code structure and interfaces	856 857
821 822	• Potential application scenarios and problem directions	• Maintains consistency by following code style and optimizes code structure to improve exe- cution efficiency	858 859 860
823	• Different implementation perspectives		
824	Valuable question points	Characteristics from code requirements:	861
825	Problem Review additional points:	• Code must be provided where there are gaps in the logic	862 863
826 827	• Does the initial question framework provide the code to be explained?	• There should be gaps in the code for comple- tion	864 865
828 829	• Whether the code to be explained is consistent with random code snippets	Inspiration Brainstorming focus:	866
830	• Is the logic of the code to be explained too	• Select a single and focused functional area	867
831	simple?	• Determine the difficulty level of the question	868
832 833	3. Knowledge-based Question Characteristics from problem requirements:	• Design the core algorithm or data structure	869
834 835	 Use words like "explain", "analyze", "com- pare" to describe requirements 	• Plan the location and scope of the code to be completed	870 871
836	• Organize related concepts into coherent ques-	Problem Review additional points:	872
837	tions	• Does the initial problem provide the code to be completed?	873 874
838 839	• Each test point should focus on concept un- derstanding	• Are there gaps in the code of the initial prob- lem?	875 876
840	Assessment Requirements:		
841	Conceptual accuracy	• Is the code to be completed consistent with the random code snippet?	877 878
842 843	 Depth of principle understanding Practical application scenarios	• Is the logic of the code to be completed too simple?	879 880

881	5. Code Optimization	• The code may be incomplete or contain errors
882	Characteristics from problem requirements:	• The code may include errors such as:
883	• The problem should have multiple possible	– Syntax errors
884	optimization directions	 Logical flaws
885	• Consider algorithm complexity, code struc-	 Algorithm efficiency issues
886	ture, and implementation details	 Boundary condition handling
887	• Typically described using terms like "opti-	 Incorrect use of data structure
888	mize", "improve", "refactor", etc.	• The code to be fixed should not be identical to the provided random code snippet
889 890	• The problem should be concise and compre- hensive	Inspiration Brainstorming focus:
891	Characteristics from code requirements:	• Design multiple error insertion strategies
892	• Not allowed to be identical to the provided	Problem Review additional points:
893	random code snippet	• Does it provide code to be fixed?
894	• Should have clear efficiency, readability, or	• Is the code to be fixed identical to the provided
895	structural issues	random code snippet?
896	Inspiration Brainstorming focus:	• Does the code to be fixed contain multiple
897	• Design an initial code framework	errors?
898	• Identify multiple core optimization points to	• Is the error logic too simple?
899	focus on	Additional notes:
900	Problem Review additional points:	• Do not allow the content of possible problems
901	• Does it provide code to be optimized?	to be solved
902	• Is the code to be optimized identical to the	• Do not provide error point comments in the code to be fixed
903	provided random code snippet?	
904	• Is the code optimization logic too simple?	7. Modify Code as required Characteristics from problem requirements:
905	Additional notes:	
906	• The final problem description output must in-	• The problem should have multiple modifica- tion requirements
907	clude the code	-
908	6. Debug	• The problem typically uses verbs like "refac- tor" or "modify" etc, followed by specific re-
909	Characteristics from problem requirements:	quirements
910	• The problem should contain multiple errors	Characteristics from code requirements:
911	• Problem sentence patterns may include:	• The code to be modified must be provided, with clear functionality but space for optimiza-
912	- "The following code attempts to imple-	tion
913	ment"	Inspiration Brainstorming focus:
914 915	 "An error occurs when running the fol- lowing code" 	
915	 – "Please identify and fix the errors in the 	• Design the core functionality of the initial code
917	code"	
918	Characteristics from code requirements:	• Plan multiple specific aspects that need modi- fication

958	Problem Review additional points:	18	- Completeness Verific
959	• Does it provide the code to be modified?	10 4	all code blocks ar integrated] ‡ Output Format
960 961	• Is the code to be modified identical to the given random code snippet?	20 1 21 2	
962 963	• Is the logic of the code to be modified too simple?	24 5	 Selection Rationale: Complexity Results: Prompt Section:
964	Additional notes:	26 27	- Code Section: - Completeness Verifica
965 966	• Do not include modification comments in the code to be modified		 Important Constraints Code Completeness: Fully preserve all constraints including language
967 968	• The final problem description output must in- clude the code	21	indentation, line b comments
969	A.2 Complicate Prompt	31	- Maintain original po blocks in complexi
970 971 972 973 974 975 976 977 978 977 978 979 980 981	 1 # Task 2 You will act as a prompt complexity expert, rewriting given prompts into more challenging versions that pose greater challenges to AI systems like ChatGPT. The rewrite must maintain human comprehensibility and executability. 3 # Complexity Methods (randomly select one) 4 1. Constraint Addition - Introduce additional restrictions or 	34 35 36	 Prohibited from modicontent and format Prompt Requirements: New content limited Ensure readability and executability Special marker words Task Boundaries: Only complexify promprovide solution approvide solution
982 983	requirements 5 2. Depth Extension - Extend inquiry	A	A.3 Text Rewrite Prompt
984	depth and expand scope		-
985 986 987 988 989 990 990	 6 3. Concrete Specification - Replace abstract concepts with more specific expressions 7 4. Reasoning Refinement - Transform simple questions into forms requiring multi-step reasoning 8 5. Input Enhancement - Add data or code 	2) 3 /	# Task You are a text rewriting task is to completely **textual description original question whil all non-textual elemer All the output must be in
992 993	in specific formats, using question forms	4 5 ‡	ŧ Workflow
994 995 996	9 6. Innovation Variation - Maintain domain and difficulty while creating more unique new prompts		 Paragraph Structure An- outline the abstract s paragraphs)
997 998	10 # Workflow 11 1. Understanding Given Prompt: [Analyze	7 4	2. Logical Flow Analysis: function of each sente
999 1000 1001 1002 1003	theme, goals, difficulty, constraints, and domain] 12 2. Code Identification and Extraction: [Identify and fully extract all code blocks from original prompt, skip	8	logical relationships) 3. Paragraph Structure Br brainstorm 3 different restructure paragraph rewriting methods)
1004 1005	this step if original prompt contains no code]	94	 Sentence Structure Bra brainstorm 3 different
1006 1007 1008 1009	 3. Selected Method: [Choose appropriate complexity method based on understanding] 14 4. Selection Rationale: [Explain method 	10	reconstruct sentences maintaining question is 5. Selected Approach: (sp approaches from steps
1010 1011	selection rationale based on given prompt and chosen method]	11 6	use) 5. Rewritten Question: (p
1012 1013 1014 1015	 15 5. Complexity Results: 16 - Prompt Section: [Show complexified text content] 17 - Code Section: [Insert extracted 	12	rewritten question whi all code, tables, and textual elements)
1016 1017 1018	code blocks at original positions and formats, output "None" if none]	14	# Output format I. Paragraph Structure An 2. Logical Flow Analysis:

- Completeness Verification: [Confirm	1019
all code blocks are correctly	1020
integrated]	1021
# Output Format	1022
1. Understanding Given Prompt:	1023
2. Code Block Extraction:	1024
3. Selected Method:	1025
4. Selection Rationale:	1026
5. Complexity Results:	1027
- Prompt Section:	1028
- Code Section:	1029
- Completeness Verification:	1030
# Important Constraints	1031
1. Code Completeness:	1032
- Fully preserve all code blocks,	1033
including language markers,	1034
indentation, line breaks, and	1035
comments	1036
- Maintain original positions of code	1037
blocks in complexity results	1038
 Prohibited from modifying code 	1039
content and format	1040
2. Prompt Requirements:	1041
- New content limited to 10-20 words	1042
 Ensure readability and 	1043
executability	1044
- Special marker words prohibited	1045
3. Task Boundaries:	1046
- Only complexify prompts, do not	1047
provide solution approaches	1048
- Use only one complexity method per	1049
time	1050

A.3 Text Rewrite Prompt

1	# Task	1052
2	You are a text rewriting expert. Your	1053
	task is to completely transform the	1054
	<pre>**textual description** in the</pre>	1055
	original question while preserving	1056
	all non-textual elements.	1057
3	All the output must be in English.	1058
4		1059
5	# Workflow	1060
6	1. Paragraph Structure Analysis: (1061
	outline the abstract structure of	1062
	paragraphs)	1063
7	2. Logical Flow Analysis: (identify the	1064
	function of each sentence and their	1065
	logical relationships)	1066
8	3. Paragraph Structure Brainstorming: (1067
	brainstorm 3 different approaches to	1068
	restructure paragraphs using	1069
	rewriting methods)	1070
9	4. Sentence Structure Brainstorming: (1071
	brainstorm 3 different approaches to	1072
	reconstruct sentences while	1073
	maintaining question integrity)	1074
10	5. Selected Approach: (specify which	1075
	approaches from steps 3 and 4 you'll	1076
	use)	1077
11	6. Rewritten Question: (present the	1078
	rewritten question while preserving	1079
	all code, tables, and other non-	1080
	textual elements)	1081
12		1082
	# Output format	1083
	1. Paragraph Structure Analysis:	1084
15	2. Logical Flow Analysis:	1085

1086	16 3. Paragraph Structure Brainstorming:	13 6. Result Formatting: Format and output
1087	17 4. Sentence Structure Brainstorming:	the task type keyword according to
1088	18 5. Selected Approach:	the given output example.
1089	19 6. Rewritten Question:	14
1090	20	15 # Output Format
1091	21 # Rewriting Methods	<pre>16 [Task]:[keyword]</pre>
1092	22 1. Alternative sentence ordering	17
1093	23 2. Different syntactic structures (e.g.,	18 # Examples Given
1094	inversions, passives)	19 Example 1:
		•
1095	24 3. Various writing techniques (e.g.,	20 Input:
1096	concise phrasing, strategic omission	21 {Given a string s containing just the
1097)	characters '(', ')', '{', '}', '[',
1098	25 4. Alternative paragraph organization (e	']', determine if the input string
1099	.g., purpose-first vs. background-	is valid.
1100	first)	22
1101	26	23 An input string is valid if:
1102	27 # Quality Standards	24
1103	₂₈ 1. Avoid structural similarity – the	25 Open brackets must be closed by the same
1104	rewritten version should differ	type of brackets.
1105	significantly in paragraph and	26 Open brackets must be closed in the
1106	sentence patterns	correct order.
1107	29 2. Present only the final question	27 Every close bracket has a corresponding
1108	without explanatory content	open bracket of the same type.}
1109	30 3. Maintain a neutral tone and	28
1110	professional, concise style without	29 Output:
1111	colloquialisms	30 [Task]:[String Validation]
1112	31 4. Preserve all code, tables, and non-	31
1113	textual elements exactly as they	32 Example 2:
1114	appear	33 Input:
1115	32 5. Exclude solution hints or guidance	34 {I have an employee payroll table with
1116	33 6. Avoid content that might suggest	headers including employee ID, name,
	5 55	
1117	answers	gender, age, department, daily wage
		, attendance days, and allowance.
		Please help me write a Python
1110	A.4 Prompts for keyword extraction	program to add a salary column to
1118	A.4 I Tompts for Keyword extraction	
		this table (Salary = daily wage *
		this table. (Salary = daily wage *
1119	A.4.1 Extract Task	attendance days + allowance)}
1119	A.4.1 Extract Task	attendance days + allowance)} 35
1119 1120	A.4.1 Extract Task	attendance days + allowance)} 35 36 Output:
1120	ı # Task	attendance days + allowance)} 35
1120 1121	1 # Task 2 Please deeply analyze the provided	attendance days + allowance)} 35 36 Output:
1120 1121 1122	 # Task Please deeply analyze the provided programming problems, extract the [attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing]
1120 1121 1122 1123	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and</pre>	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3:
1120 1121 1122 1123 1124	1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input:
1120 1121 1122 1123	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and</pre>	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose
1120 1121 1122 1123 1124	1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input:
1120 1121 1122 1123 1124 1125 1126	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must</pre>	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence
1120 1121 1122 1123 1124 1125 1126 1127	1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English.	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose
1120 1121 1122 1123 1124 1125 1126 1127 1128	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3</pre>	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task</pre>	attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence 42 A valid additive sequence must contain at least three numbers. Except for
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it.</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application.</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise.</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content.</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise.</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main</pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. </pre>	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.}</pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141	<pre>1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. </pre>	 attendance days + allowance)} ³⁵ ³⁶ Output: ³⁷ [Task]:[Data Processing] ³⁸ ³⁹ Example 3: ⁴⁰ Input: ⁴¹ {An additive number is a string whose digits can form an additive sequence ⁴² A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. ⁴³ Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. ⁴⁴ Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.}
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or 	 attendance days + allowance)} ³⁵ ³⁶ Output: ³⁷ [Task]:[Data Processing] ³⁸ ³⁹ Example 3: ⁴⁰ Input: ⁴¹ {An additive number is a string whose digits can form an additive sequence . ⁴² A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. ⁴³ Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. ⁴⁴ Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.}
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 	 attendance days + allowance)} ³⁵ ³⁶ Output: ³⁷ [Task]:[Data Processing] ³⁸ ³⁹ Example 3: ⁴⁰ Input: ⁴¹ {An additive number is a string whose digits can form an additive sequence ⁴² A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. ⁴³ Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. ⁴⁴ Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.}
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this 	 attendance days + allowance)} ³⁵ ³⁶ Output: ³⁷ [Task]:[Data Processing] ³⁸ ³⁹ Example 3: ⁴⁰ Input: ⁴¹ {An additive number is a string whose digits can form an additive sequence . ⁴² A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. ⁴³ Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. ⁴⁴ Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.}
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a concise keyword. 	 attendance days + allowance)} ³⁵ ³⁶ Output: ³⁷ [Task]:[Data Processing] ³⁸ ³⁹ Example 3: ⁴⁰ Input: ⁴¹ {An additive number is a string whose digits can form an additive sequence . ⁴² A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. ⁴³ Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. ⁴⁴ Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.}
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a concise keyword. 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a concise keyword. 12 5. Keyword Validation: Ensure the chosen keyword accurately reflects the 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] 41 # Task </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a concise keyword. 12 5. Keyword Validation: Ensure the chosen keyword accurately reflects the core of the content, avoiding overly 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] 1 # Task 2 Please deeply analyze the provided </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a concise keyword. 12 5. Keyword Validation: Ensure the chosen keyword accurately reflects the 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence . 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] 41 # Task </pre>
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150	 1 # Task 2 Please deeply analyze the provided programming problems, extract the [Task] keywords from them, and summarize them in concise keyword form. You need to ensure the accuracy of the keywords.Output must be in English. 3 4 # Definition of Task 5 **[Task]** is defined as categorizing the main theme or operation of the input content into a broad domain or operation type for effective handling and application. 6 7 # Workflow 8 1. Input Reception: Receive and prepare to analyze input content. 9 2. Content Understanding: Carefully read the input content to grasp its main theme or operation. 10 3. Domain Categorization: Categorize the content into a broad domain or operation type. 11 4. Keyword Extraction: Describe this domain or operation type with a concise keyword. 12 5. Keyword Validation: Ensure the chosen keyword accurately reflects the core of the content, avoiding overly 	<pre>attendance days + allowance)} 35 36 Output: 37 [Task]:[Data Processing] 38 39 Example 3: 40 Input: 41 {An additive number is a string whose digits can form an additive sequence 42 A valid additive sequence must contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the two numbers before it. 43 Given a string containing only digits '0'-'9', write a Java algorithm to determine if the given input is an additive number. Return true if it is, and false otherwise. 44 Note: Numbers in the additive sequence cannot start with 0, except for the number 0 itself, so there won't be cases like 1, 2, 03 or 1, 02, 3.} 45 46 Output: 47 [Task]:[Sequence Validation] 1 # Task 2 Please deeply analyze the provided </pre>

1220	The second secon
	Instruction] keywords from them, and
1221	summarize them in concise keyword
1222	form. You need to ensure the
1223	accuracy of the keywords.Output must
1224	be in English.
1225	3
1226	4 # Definition of Instructions
1227	5 1. [Instructions] are explicit
1228	requirements or constraints
1229	extracted from the input, used to
1230	guide task execution, output format,
1231	or operation steps.
1232	6 2. **Instructions do not include
1233	programming languages specified in
1234	the problem**
	•
1235	7
1236	8 # Workflow
1237	9 1. Input Reception: Receive complete
1238	
	description of the programming
1239	problem.
1240	10 2. Problem Understanding: Read the
1241	problem in detail to understand core
1242	requirements and objectives.
1243	n 3. Instruction Identification: Carefully
1244	analyze the problem description to
1245	identify explicit [Instruction]
1246	content.
1247	12 4. Instruction Extraction: Extract the
1248	identified instructions from the
1249	
	problem description.
1250	13 5. Instruction Validation: Review the
1251	extracted instructions according to
1252	the definition of [Instructions],
1253	you need to ensure your output
1254	complies with the content in #
1255	Definition of Instructions.
1256	14 6. Result Formatting: Organize and
1257	output the extracted instructions
1258	according to the given output
1259	example format.
1260	15
1261	
1262	16 # Output Format
	<pre>16 # Output Format 17 [Instructions]:[keyword1] [keyword2]</pre>
1263	
1263	<pre>17 [Instructions]:[keyword1] [keyword2] 18</pre>
1263 1264	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given</pre>
1263 1264 1265	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1:</pre>
1263 1264	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given</pre>
1263 1264 1265	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input:</pre>
1263 1264 1265 1266 1267	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?}</pre>
1263 1264 1265 1266 1267 1268	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23</pre>
1263 1264 1265 1266 1267 1268 1269	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output:</pre>
1263 1264 1265 1266 1267 1268	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23</pre>
1263 1264 1265 1266 1267 1268 1269	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output:</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2:</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input:</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2:</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission,</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the third and fourth digits.}</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the third and fourth digits.}</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the third and fourth digits.} 31 32 Output:</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the third and fourth digits.} 31 32 Output: 33 [Instructions]:[Implement digit</pre>
1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286	<pre>17 [Instructions]:[keyword1] [keyword2] 18 19 # Examples Given 20 Example 1: 21 Input: 22 {How to compare two lists in Python?} 23 24 Output: 25 [Instructions]:[] 26 27 Example 2: 28 Input: 29 {Please implement in Java: A company uses public telephone to transmit data, the data is a four-digit integer, which is encrypted during transmission, 30 The encryption rules are as follows: add 7 to each digit, then replace the digit with the remainder of the sum divided by 3, then swap the first and second digits, and swap the third and fourth digits.} 31 32 Output:</pre>

A.4.3 Extract Knowledge

1	# Task	1291
	Please deeply analyze the provided	1292
2		
	programming problems, extract the [1293
	Knowledge Points] keywords from them	1294
	, and summarize them in concise	1295
	keyword form. You need to ensure the	1296
	accuracy of the keywords.Output	1297
	must be in English.	1298
3		1299
4	<pre># Definition of Knowledge Points</pre>	1300
5	<pre>**[Knowledge Points]**: Core concepts,</pre>	1301
	basic principles, or key operational	1302
	steps necessary to solve specific	1303
	programming problems, usually	1304
	presented in the form of concise	1305
	keywords.	1306
6		1307
	# Waal 61	
	# Workflow	1308
8	 Input Reception: Receive and prepare 	1309
	to analyze specific programming	1310
	problems.	1311
0	2. Problem Understanding: Read the	1312
7		1313
	problem thoroughly to grasp its core	
	requirements and objectives.	1314
10	3. Core Concept Identification: Identify	1315
	basic theoretical knowledge and key	1316
	programming principles needed to	1317
		1318
	solve the problem.	
11	Operation Step Extraction: Analyze	1319
	main steps and necessary programming	1320
	techniques for problem-solving.	1321
12	5. Knowledge Point Condensation:	1322
12	Transform identified concepts and	1323
	steps into concise keywords.	1324
13	6. Key Point Validation: Review	1325
	extracted knowledge points to ensure	1326
	their necessity, completeness, and	1327
	conciseness.	1328
14	7. Result Formatting: Format and output	1329
	key points according to the given	1330
	output example.	1331
15		1332
	# Output Format	1333
	[Knowledge Points]:[keyword1] [keyword2	1334
1/		
]	1335
18		1336
19	# Examples Given	1337
	Example 1:	1338
	Input:	1339
	•	
22	{Please implement in Java: A company	1340
	uses public telephone to transmit	1341
	data, the data is a four-digit	1342
	integer, which is encrypted during	1343
	transmission,	1344
23	The encryption rules are as follows: add	1345
	7 to each digit, then replace the	1346
	digit with the remainder of the sum	1347
	divided by 3, then swap the first	1348
	and second digits, and swap the	1349
	third and fourth digits.}	1350
24		1351
25	Output:	1352
26	[Knowledge Points]:[Modulo Operation][1353
	Number Processing][Data Swapping]	1354
27		1355
	Evennle 2.	
	Example 2:	1356
	Input:	1357
30	{Given an integer n, return the number	1358

1359	of strings of length n that consist	8 - **Instruction**: Specific functions,	1425
1360	only of vowels (a, e, i, o, u) and	effects or operations that need to	1426
1361	are lexicographically sorted. A	be implemented in the task, usually	1427
1362		low-level, concrete steps.	1428
	string s is lexicographically sorted		
1363	if for all valid i, s[i] is the	Instructions are specific means to	1429
1364	same as or comes before s[i+1] in	accomplish tasks, such as "query	1430
1365	the alphabet.}	optimization" or "table structure	1431
1366	31	design".	1432
1367	32 Output:	9	1433
1368	33 [Knowledge Points]:[String Generation][<pre>10 # Node Extraction Rules:</pre>	1434
1369	Vowels][Lexicographical Order][11 1. Preserve all identified **Task** and	1435
1370	Combination Counting]	<pre>**Knowledge Point** entities.</pre>	1436
1371	34	12 2. Only preserve **important	1437
1372	35 Example 3:	instructions** directly related to	1438
1373	36 Input:	the task's core objectives, remove	1439
1374	37 {Please provide a solution in Java code	minor or unimportant instructions.	1440
1375	for the following problem: There are	13 3. Each `Node` object should include:	1441
1376	some spherical balloons taped onto	 14 - Unique identifier (`id`) 	1442
1377	a flat wall that represents the XY	15 - Entity type (`type`)	1443
1378	plane. The balloons are represented	16	1444
1379	as an integer array points where	17 # Output Format:	1445
1380	points[i] = [xstart, xend] denotes a	18 - **Only output structured text** in the	1446
1381	balloon whose horizontal diameter	following format:	1447
1382	stretches between xstart and xend.	19 Node(id="entity name", type="entity type	1448
1383	You do not know the exact y-	")	1449
1384	coordinates of the balloons.	20 - **Ďo not generate code**, only output	1450
1385	38 An arrow can be shot up exactly	the `Node` list in text form.	1451
1386	vertically from different points	21	1452
1387	along the x-axis. A balloon with	22 # Work Steps:	1453
	-		
1388	xstart and xend bursts by an arrow	23 1. Read through the input content,	1454
1389	shot at x if xstart x xend.	identify and classify entities.	1455
1390	There is no limit to the number of	24 2. Analyze logical relationships between	1456
1391	arrows that can be shot. An arrow	entities to determine their	1457
1392	once shot keeps traveling up	categories.	1458
1393	infinitely.	25 3. Output extracted `Nodes` in specified	1459
1394	39 Given an array points, return the	format.	1460
1395	minimum number of arrows that must	26	1461
1396	be shot to burst all balloons.}	27 # Notes:	1462
1397	40	28 - **Important instructions**:	1463
1398	41 Output:	Instructions directly related to	1464
1399	42 [Knowledge Points]:[Array Operations][task core objectives.	1465
1400		29 - **Unimportant instructions**:	1466
	Conditional Logic][Mathematical		
1401	Logic]	Instructions with minor or secondary	1467
		impact on task core objectives.	1468
		30 - If there is no clear task topic in the	1469
1402	A.5 Keyword combination and filtering	input, analyze logical	1470
-	· · · · · · · · · · · · · · · · · · ·	relationships between entities to	1471
1403	A.5.1 Description of keyword types	infer appropriate task type.	1472
		31 - Task type should be unique and clear.	1473
1/0/	1 # Task:	32	1474
1404		33 # Example:	1475
1405	2 Extract information from user input and	34	1476
1406	structure it into `Node` objects.	35 **Input**:	1477
1407	Only output structured text, no code	36 [open website], [develop an e-commerce	1478
1408	generation needed.Output must be in		
1409	English.	website], [HTML], [CSS], [JavaScript	1479
1410	3], [implement user registration], [1480
1411	4 # Entity Type Description:	shopping cart functionality], [mouse	1481
1412	5 Identify entities in the input and	operations]	1482
1413	classify them into the following	37 **Expected Output**:	1483
1414	types:	38 Node(id="open website", type="	1484
1415	6 - **Task**: High-level goals or topics	instruction")	1485
1416	that typically require multiple	39 Node(id="develop an e-commerce website",	1486
1417	steps or operations to complete.	type="task")	1487
		40 Node(id="HTML", type="knowledge point")	1488
1418	Tasks are ultimate goals, such as "	41 Node(id="CSS", type="knowledge point")	1489
1419	develop an e-commerce website" or "	42 Node(id="JavaScript", type="knowledge	1490
1420	design a database system".	point")	1491
1421	<pre>7 - **Knowledge Point**: Computer science</pre>	43 Node(id="implement user registration",	1492
1422	knowledge required for tasks or		
1423	instructions, such as "HTML", "	<pre>type="instruction") w Node(id="sharping_cost_functionality"</pre>	1493
1424	Python" or "MySQL".	<pre>44 Node(id="shopping cart functionality",</pre>	1494

1495		type="instruction")
496	45	Node(id="mouse operations", type="
497		instruction")
		,
498		A.5.2 Relationship analyses
499	1	# Task:
1500	2	Build relationship object triples from
501		input nodes.Output must be in
1502		English.Here's what needs to be done
1503		:
1504	3	
1505	4	<pre># Relationship Extraction:</pre>
1506		- You should identify relationships
1507	5	between Nodes extracted from the
1508		input content.
509	6	- Create a relationship object for each
1510		relationship.
1511	7	- A relationship object should have a
1512		subject (subj) and an object (obj),
1513		which are Node objects representing
514		the entities involved in the
1515		relationship.
1516	0	- Each relationship should also have a
	8	
1517		type (type) and, where applicable,
1518		other attributes (such as weight,
1519		direction, etc.).
1520	9	
1521	10	<pre># Node Type and Relationship Type</pre>
522		Mapping:
1523	11	- **Instructions** and **Knowledge
524		Points** may have a "displays"
1525		relationship.
526	12	- **Tasks** and **Instructions** may
1527	12	have a "requires" relationship.
1528	10	
	13	- **Tasks** and **Knowledge Points** may
1529		have a "contains" relationship.
1530	14	- If there is no logical connection
1531		between two nodes and they clearly
1532		belong to completely different
1533		domains, use an "unrelated"
1534		relationship.
1535	15	
1536	16	<pre># Relationship Building Process:</pre>
1537		1. Parse and understand the id and type
538		from input nodes.
1539	18	2. Think deeply about the inherent
1539	10	connections between different nodes,
1541		combining computer knowledge.
1542	19	3. Find relationships that exist between
543		different nodes.
1544	20	Mark as "unrelated" if no
1545		relationship exists between nodes
1546		and they clearly belong to
1547		completely different domains.
1548	21	· · ·
549	22	# Work Steps:
1550		- Read through the provided content.
1551		- Identify relationships between input
1552	24	Nodes.
	25	
1553	25	- Provide extracted relationships in
1554		specified format.
1555	26	
556		# Output Format:
1557	28	 Extracted relationships should be
1558		formatted as instances of the
1559		provided relationship class.
1560	29	- Ensure extracted data conforms to the
1561		class definition structure.

```
30 - Your output format should be: subject
                                                       1562
      (subj) + type + object (obj).
                                                       1563
                                                       1564
31
32 # Examples:
                                                       1565
                                                       1566
33
34 ## Example 1:
                                                       1567
35 Node:
                                                       1568
36 Node(id="data visualization analysis",
                                                       1569
      type='instruction')
                                                       1570
37 Node(id='HTML line chart', type='
                                                       1571
      knowledge point')
                                                       1572
38 Node(id='business analysis report', type
                                                       1573
      ='task')
                                                       1574
                                                       1575
39
40 Expected Output:
                                                       1576
                                                       1577
41 data visualization analysis displays
      HTML line chart
                                                       1578
42 business analysis report requires data
                                                       1579
      visualization analysis
                                                       1580
43 business analysis report contains HTML
                                                       1581
      line chart
                                                       1582
44
                                                       1583
45 ## Example 2:
                                                       1584
                                                       1585
46 Node:
47 Node(id="physical acceleration", type='
                                                       1586
      task')
                                                       1587
48 Node(id="navigation bar", type='
                                                       1588
      instruction')
                                                       1589
                                                       1590
49
50 Expected Output:
                                                       1591
51 physical acceleration unrelated
                                                       1592
      navigation bar
                                                       1593
                                                       1594
52
53 ## Example 3:
                                                       1595
                                                       1596
54 Node:
55 Node(id="user login", type='task')
                                                       1597
56 Node(id="password verification", type='
                                                       1598
      instruction')
                                                       1599
57 Node(id="security", type='knowledge
                                                       1600
      point')
                                                       1601
                                                       1602
58
59 Expected Output:
                                                       1603
_{\rm 60} user login requires password
                                                       1604
      verification
                                                       1605
                                                       1606
61 password verification contains security
62
                                                       1607
63 ## Example 4:
                                                       1608
64 Node:
                                                       1609
65 Node(id="artificial intelligence", type
                                                       1610
                                                       1611
      ='task')
66 Node(id="psychology", type='knowledge
                                                       1612
      point')
                                                       1613
67 Node(id="ethics", type='knowledge point
                                                       1614
                                                       1615
      ')
                                                       1616
68
69 Expected Output:
                                                       1617
70 artificial intelligence contains
                                                       1618
      psychology
71 artificial intelligence contains ethics
                                                       1620
72
73 ## Example 5:
                                                       1622
74 Node:
                                                       1623
75 Node(id="quantum computing", type='task
                                                       1624
                                                       1625
       ')
76 Node(id="artificial intelligence", type
                                                       1626
      ='task')
                                                       1627
77 Node(id="blockchain", type='knowledge
                                                       1628
      point')
                                                       1629
78 Node(id="cryptography", type='knowledge
      point')
                                                       1631
```

1600	-			۴Ar
1632	79		11 2. *:	[¢] AI
1633	80	Expected Output:	t	he
1634	91	quantum computing contains cryptography	ŀ	et
1635	82	artificial intelligence contains	t	he
1636		blockchain	12 3. *:	۲a
1637	02	quantum computing unrelated artificial		el
	0.0			
1638		intelligence	1	01
1639	84	artificial intelligence unrelated	k	íno
1640		cryptography		ns
		ci yptography	e	
1641	85			гe
1642	86	<pre># Complex Relationship Handling:</pre>	13	
		· · · · ·		
1643	87	- If multiple relationships exist	14 # Ou	•
1644		between nodes, list each	15 - Ex	tra
1645		relationship separately.		sh
1646	0.0			th
	88	- If nested relationships exist (where		
1647		the subject or object in a	C	com
1648		relationship is itself a	16 - En:	sur
1649				:1a
		relationship), clearly mark them.		
1650	89	- If temporal relationships exist	17 - You	ır
1651		between nodes (such as "Task A" must	ā	icc
1652				
		be completed before "Task B"), note		pri
1653		this in the relationship type.	t)et
1654	90		18	
		# Error Handling.		- - -
1655		# Error Handling:	19 # Wo	
1656	92	- If input node format is incorrect,	20 1. *:	×Re
1657		return error message: "Input node	(on
1658		format incorrect".	č	nd
1659	93	- If unable to determine relationships	21 2. *:	۶De
1660		between nodes, return: "Unable to	F	le1
1661		determine relationship".	5	ec
			i	np
			k	no
1662		A.5.3 Associative phrase construction	22 3. *:	×Re
		•	V	ıor
			1	og
1663		# Task Description:		-
1004				ıor
1664	2	Extract reasonable word combinations		
	2	Extract reasonable word combinations		۰ Or
1665	2	from user-input triples (subject +	23 4. *:	
	2	from user-input triples (subject + type + object). Reasonable word	23 4. * : r	el
1665	2	from user-input triples (subject +	23 4. *: r f	el ol
1665 1666 1667	2	from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant	23 4. *: r f	el
1665 1666 1667 1668	2	from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that	23 4. *: r f	el ol oi
1665 1666 1667 1668 1669	2	from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and	23 4. *: f 24 5. *:	°el °ol °oi ⊁Fc
1665 1666 1667 1668	2	from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that	23 4. *: f 24 5. *:	el ol Foi Foi
1665 1666 1667 1668 1669 1670	2	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output</pre>	23 4. *: f 24 5. *:	°el °ol °oi ⊁Fc
1665 1666 1667 1668 1669 1670 1671		from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and	23 4. *: f 24 5. *: c	el ol Foi Foi
1665 1666 1667 1668 1669 1670 1671 1672	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English.</pre>	23 4. *: f 24 5. *: 25	rel Poi ∀Fc rea
1665 1666 1667 1668 1669 1670 1671	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English.</pre>	23 4. *** f 24 5. *** 25 26 # No	rel Yoi YFC YEA
1665 1666 1667 1668 1669 1670 1671 1672 1673	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions:</pre>	23 4. *** f 24 5. *** 25 26 # No	rel Poi ∀Fc rea
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. ***	rel Poi Fo rea cc tes
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on</pre>	23 4. *: 1 24 5. *: 25 26 # No 27 1. *: 6	el ol Foi Foi cc tes Se
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. *** 6	rel fol foi rea ncc tes ses ses set set
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. *** 28 2. ***	rel Yoi Yoi Yoi Yea Icc tes Yea Yea Yea Yea Yea Yea Yea Yea Yea Yea
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677	3	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. *** 28 2. ***	rel fol foi rea ncc tes ses ses set set
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task.</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. *** 28 2. ***	<pre>>el Yoi Yoi Yoi Yea Yea Yea Yea Yea Yea Yea Yea Yea Yea</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. *** 28 2. *** 5	<pre>>el >ol >oi >rea >rcc >tes >xt :le >xt :le >xt ;ho pr</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit</pre>	23 4. *** 1 f 24 5. *** 25 26 # No 27 1. *** 28 2. *** 29 3. ***	rel Fol Poi Foi rea tes Stor Stor Stor Stor Stor Stor Stor Stor
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints</pre>	23 4. *** 1 f 24 5. *** 25 26 # No 27 1. *** 28 2. *** 29 3. ***	<pre>>el >ol >oi >rea >rcc >tes >xt :le >xt :le >xt ;ho pr</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to</pre>	23 4. *** 1 24 5. *** 25 26 # No 27 1. *** 28 2. *** 29 3. *** 29 3. ***	<pre>>el Yoi Yoi Yoi Yoi Yoi Yoi Yoi Yoi Yoi Yoi</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format,</pre>	23 4. *** 1 f 24 5. *** 25 26 # No 27 1. *** 28 2. *** 29 3. *** 29 3. ***	rel Fol Poi Foi rea tes Stor Stor Stor Stor Stor Stor Stor Stor
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format,</pre>	23 4. *** 1 24 5. *** 25 26 # No ² 27 1. *** 28 2. *** 29 3. *** 29 3. *** 29 3. ***	rel fol fol fol fol fol fol fol fol fol fo
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, "</pre>	23 4. *** 1 f 24 5. *** 25 26 # No 27 1. *** 28 2. *** 29 3. *** 29 3. ***	rel fol fol fol fol fol fol fol fol fol fo
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is</pre>	23 4. *** 1 f 24 5. *** 25 26 # No 27 1. *** 28 2. *** 29 3. *** 29 3. *** 29 3. *** 29 3. ***	rel fol fol fol fol fol fol fol fol fol fo
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685	3 4 5	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction.</pre>	23 4. *: 1 24 5. *: 25 26 # No 27 1. *: 28 2. *: 29 3. *: 30 31 # Ex. 32	<pre>>el >oi >Fc >ea >Fc >Fc >ea >Fc >Fc >Fc >Fc >Fc >Fc >Fc >Fc >Fc >Fc</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction.</pre>	23 4. *** 1 1 24 5. *** 25 26 # No ² 27 1. *** 28 2. *** 29 3. *** 29 3. *** 30 31 # Ex. 32 33 ## E	<pre>>el Yoi Yoi Yoi Yea Yea Yea Yea Yea Yea Yea Yea Yea Yea</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts,</pre>	23 4. *: 1 24 5. *: 25 26 # No 27 1. *: 28 2. *: 29 3. *: 30 31 # Ex. 32	<pre>>el Yoi Yoi Yoi Yea Yea Yea Yea Yea Yea Yea Yea Yea Yea</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational</pre>	23 4. *** 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	<pre>>ell >ol >ol >ol >col >col >col >col >col</pre>
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific</pre>	23 4. *** f f 24 5. *** 25 26 # Noo 27 1. *** 28 2. *** 29 3. *** 29 3. *** 30 31 # Ex. 32 33 ## E: 34 creations	elloi Poi tese tese tese tese tese tese tese tes
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific</pre>	23 4. *** 1 f 24 5. *** 25 26 # Noo 27 1. *** 28 2. *** 29 3. *** 29 3. *** 30 31 # Ex. 32 33 ## E* 34 creations	el ol ol ol ol ol ol ol ol ol o
1665 1666 1667 1668 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, "</pre>	23 4. *** f f 24 5. *** 25 26 # Noo 27 1. *** 28 2. *** 29 3. *** 29 3. *** 30 31 # Ex. 32 33 ## E: 34 creations	el ol ol ol ol ol ol ol ol ol o
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a</pre>	23 4. *** f f f f f f f f	ell oli voi tesete
1665 1666 1667 1668 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, "</pre>	23 4. *** r f F 24 5. *** 25 26 # No ² 27 1. *** 6 28 2. *** 28 29 3. *** 30 31 # Ex. 33 33 ## E: 34 creations 35 creations 36 creations 37 creations 37 creations 37 creations 37 creations 37 creations 37 creations 37 creations 38 creations 37 creations 38 creations 37 creations 38 creations 37 creations 38 creations 38 creations 39 creations 30 creations 30 creations 30 creations 30 creations 31 creations 32 creations 33 creations 34 creations 35 creations 36 creations 37 creations 37 creations 37 creations 38 creations 38 creations 39 creations 30 creations 30 creations 30 creations 30 creations 30 creations 30 creations 31 creations 31 creations 32 creations 33 creations 34 creations 37 creations	ell oli Poi tesete
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690	3 4 5 6	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a</pre>	 23 4. *: 24 5. *: 26 # Noi 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 30 31 # Ex. 33 ## E: 34 crea: 35 crea: 36 crea: 37 crea: 	el Soli Poi reactes ster ster ster ster ster ster ster s
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692	3 4 5 6 7 8	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point.</pre>	23 4. *** r f F 24 5. *** 25 26 # No ² 27 1. *** 6 28 2. *** 28 29 3. *** 30 31 # Ex. 33 33 ## E: 34 creations 35 creations 36 creations 37 creations 37 creations 37 creations 37 creations 37 creations 37 creations 37 creations 38 creations 37 creations 38 creations 37 creations 38 creations 37 creations 38 creations 38 creations 39 creations 30 creations 30 creations 30 creations 30 creations 31 creations 32 creations 33 creations 34 creations 35 creations 36 creations 37 creations 37 creations 37 creations 38 creations 38 creations 39 creations 30 creations 37 creations	el Soli Poi reactes ster ster ster ster ster ster ster s
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693	3 4 5 6 7 8	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable</pre>	 23 4. *: 24 5. *: 25 26 # Noi 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 30 31 # Ex. 33 ## E: 34 creation 35 creation 36 creation 38 creation 	ell fol fol kFeac tesetello for stello for for stello for for stello for stello for stello for stello for stello for stello for stello for for stello for for stello for for for for for for for for for fo
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1685 1685 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable Associated Word Combinations:</pre>	 23 4. *: 24 5. *: 25 26 # No 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 20 3. *: 30 31 # Ex. 33 ## E: 33 ## E: 34 creation of crea	elloice loice
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable</pre>	 23 4. *: 24 5. *: 25 26 # Noi 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 20 3. *: 21 1. *: 22 2. *: 23 ## E: 33 ## E: 33 ## E: 34 creation 35 creation 36 creation 37 creation 38 creation 39 creation 	elloicace setellopstcs and the setellopstcs tellopstcs
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1685 1685 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable Associated Word Combinations: 1. **Remove Irrelevant Words**: Delete</pre>	 23 4. *: 24 5. *: 25 26 # Noi 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 20 3. *: 21 1. *: 22 2. *: 23 ## E: 33 ## E: 33 ## E: 34 creation 35 creation 36 creation 37 creation 38 creation 39 creation 	elloice loice
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable Associated Word Combinations: 1. **Remove Irrelevant Words**: Delete words that have no semantic or </pre>	 23 4. *: 24 5. *: 25 26 # Noi 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 20 3. *: 21 1. *: 22 2. *: 23 ## E: 33 ## E: 34 creation 35 creation 36 creation 37 creation 38 creation 39 creation 	elloicac settellors an ane steriet
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable Associated Word Combinations: 1. **Remove Irrelevant Words**: Delete words that have no semantic or logical relationship with other</pre>	23 4. *** 1 *** 24 5. *** 25 26 26 # No 27 1. *** 28 2. *** 29 3. *** 30 31 31 # Ex. 33 ## Ex. 34 creat 35 creat 38 creat 39 creat 30 creat 31 # Creat 32 *** 34 creat 35 creat 36 creat 37 creat 38 creat 40 creat	elloiter test test test test test test test t
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable Associated Word Combinations: 1. **Remove Irrelevant Words**: Delete words that have no semantic or </pre>	 23 4. *: 24 5. *: 25 26 # Noi 27 1. *: 28 2. *: 29 3. *: 29 3. *: 20 3. *: 20 3. *: 21 1. *: 22 2. *: 23 ## E: 33 ## E: 34 creation 35 creation 36 creation 37 creation 38 creation 39 creation 	elloiter test test test test test test test t
1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697	3 4 5 6 7 8 9	<pre>from user-input triples (subject + type + object). Reasonable word combinations refer to task-relevant keywords, phrases, or sentences that clearly express the core logic and relationships of the task.Output must be in English. # Word Type Definitions: 1. **Task**: Categorized into broad domains or operation types based on the main theme or operation of the input. For example, "create game" is a task. 2. **Instruction**: Explicit requirements or constraints extracted from the input, used to guide task execution, output format, or operation steps. For example, " function to detect cookie value" is an instruction. 3. **Knowledge Point**: Core concepts, basic principles, or key operational steps needed to solve specific programming problems. For example, " using HTML, CSS, JavaScript" is a knowledge point. # Rules for Extracting Reasonable Associated Word Combinations: 1. **Remove Irrelevant Words**: Delete words that have no semantic or logical relationship with other</pre>	23 4. *** 1 *** 24 5. *** 25 26 26 # No 27 1. *** 28 2. *** 29 3. *** 30 31 31 # Ex. 33 ## Ex. 34 creat 35 creat 38 creat 39 creat 30 creat 31 # Creat 32 *** 34 creat 35 creat 36 creat 37 creat 38 creat 40 creat	elloiter test test test test test test test t

2. **Analyze Relationships**: Analyze	1699
the subject-object relationships	1700
between remaining words, clarifying	1701
their logical connections.	1702
3. **Task Logic**: Organize	1703
relationships between words	1704
following the logic of "Task +	1705
Knowledge Point + Instruction",	1706
ensuring clear primary and secondary	1707
relationships.	1708
	1709
# Output Format:	1710
 Extracted reasonable word combinations 	1711
should be formatted as instances of	1712
the provided reasonable word	1713
combination class.	1714
- Ensure extracted data conforms to the	1715
class definition structure.	1716
- Your output should be arranged	1717
according to the task logic or	1718
primary-secondary relationships between words.	1719 1720
between words.	1721
# Working Steps:	1722
1. **Read Input**: Carefully read input	1723
content, understand its core theme	1723
and logic.	1725
2. **Determine Primary-Secondary	1726
Relationships**: Clarify primary-	1727
secondary relationships between	1728
input words, distinguish tasks,	1729
knowledge points, and instructions.	1730
3. **Remove Irrelevant Words**: Delete	1731
words that have no semantic or	1732
logical relationship with other	1733
words.	1734
4. **Organize Logic**: Organize	1735
relationships between words	1736
following the "Task + Knowledge	1737
Point + Instruction" logic. 5. **Format Output**: Output extracted	1738 1739
reasonable word combinations	1740
according to specified format.	1741
	1742
# Notes:	1743
1. **Semantic Association**: Ensure	1744
extracted word combinations have	1745
clear semantic associations.	1746
2. **Clear Logic**: Output content	1747
should have clear logic and distinct	1748
primary-secondary relationships.	1749
3. **Standard Format**: Output strictly	1750
according to specified format,	1751
ensuring consistent data structure.	1752
[1753
# Examples:	1754
## Example 1:	1755 1756
create game contains create interface	1757
create game needs function to detect	1758
cookie value	1759
create game needs record time	1760
create game needs ensure clear code	1761
structure	1762
create game based on using HTML, CSS,	1763
JavaScript	1764
create interface needs function to	1765
detect cookie value	1766
create interface needs record time	1767
create interface needs ensure clear code	1768

1769	structure
1770	42 create interface based on using HTML,
1771	CSS, JavaScript
1772	43 create interface unrelated to physical
1773	acceleration
1774	44
1775	45 Expected output:
1776	46 create game, using HTML, CSS, JavaScript
1777	, create interface, function to
1778	detect cookie value, record time,
1779	ensure clear code structure
1780	47
1781	48 ## Example 2:
1782	49 programming parameter definition
1783	unrelated to print character
1784	50 programming parameter definition
1785	unrelated to default primary key
1786	field
1787	51 handle missing values unrelated to print
1788	character
1789	52 handle missing values unrelated to
1790	default primary key field
1791	53 train test set unrelated to print
1792	character
1793	54 train test set unrelated to default
1794	primary key field
1795	55 investigate outliers unrelated to print
1796	character
1797	56 investigate outliers unrelated to
	-
1798	default primary key field
1799	57 data analysis unrelated to print
1800	character
1801	58 data analysis unrelated to default
1802	primary key field
1803	59 problem solving unrelated to print
1804	character
1805	60 problem solving unrelated to default
1806	primary key field
1807	61 data splitting unrelated to print
1808	character
1809	62 data splitting unrelated to default
1810	primary key field
1811	63
1812	64 Expected output:
1813	65 No relevance
1011	A (Intermeted Dealsfording Origination
1814	A.6 Integrated Backfeeding Question
1815	Generation Template
C101	Generation Template
1016	1 # Role
1816	
1817	2 As an examiner specialized in designing
1818	[QUESTION_TYPE] programming
1819	questions, your task is to create a
1820	high-quality question based on
1821	keywords provided by users. These
1822	questions should [QUESTION_PURPOSE].
1823	Output must be in English.
1824	3
1825	4 # Characteristics of [QUESTION_TYPE]
1826	Questions
1020	Questions . ## Enom question requirements.

- Questions
- 5 ## From question requirements:
 - 6 1. [QUESTION_TYPE specific requirement description]
 - ${\scriptscriptstyle 7}$ 2. [Common question patterns or keywords]

```
8 3. [Assessment focus]
9
```

1828

1829

1830

1831

1832

1833

1834

10 ## From code requirements:

11	1. [Whether code must be provided -	1835
	specific requirements]	1836
12	2. [Code feature requirements]	1837
13		1838
14	# Output Example	1839
15	1. Consider logical relationships	1840
	between keywords: [List meanings of	1841
	keywords and their logical	1842
	relationships]	1843
16	2. Understand question characteristics:	1844
	[Analyze characteristics of [1845
	QUESTION_TYPE] questions, provide	1846
	elements you think must be included]	1847
17	3. Consider how to organize keywords	1848
17	into questions: [Through	1849
	brainstorming, think divergently	1850
	about how keywords can work together	1851
	to form a programming question]	1852
18	4. Output initial question: [Combine	1853
10	above thoughts to propose initial	1854
	question]	1855
19	5. Review initial question: [Identify	1856
1)	unreasonable or specific areas for	1857
	improvement in initial question and	1858
	propose modification examples]	1859
20	6. Propose new question: [Fix question	1860
20	based on modification suggestions]	1861
21	7. Repeat above steps, review and modify	1862
21	again until question meets	1863
	requirements	1864
22	8. Final question output: [Output final	1865
22	question without any guiding words (1866
	like "Question:") or any symbols]	1867
23	like question.) of any symbols	1868
	# Question Standards	1869
	1. Can hide emphasis on which	1870
20	programming language to use, letting	1871
	students derive related knowledge	1872
	themselves	1873
26	2. Please strictly follow the format in	1874
20	# Output Example to give your	1875
	thinking process for each step, but	1876
	don't directly output the content in	1877
	[], and the last step must be the	1878
	final question output	1879
27	3. Don't provide any solution ideas or	1880
2,	hints	1881
28	4. Don't show any content that might	1882
20	suggest answers	1883
29	5. [QUESTION_TYPE specific additional	1884
27	standards]	1885
	A.7 Prompt Filtering	1000
	A. Trompt Futching	1886
-	# Task	1887
2	Your task is to evaluate how input	1888
	prompts enhance the capabilities of	1889
	advanced AI assistants. For each	1890
	input prompt, analyze it according	1891
	to the following 7 criteria.	1892

- 3 4 # Standards
- 5 1. Specificity: Does the prompt request specific, clear outputs without ambiguity? This allows AI to demonstrate its ability to follow instructions and generate precise, targeted responses. 6 2. Domain Knowledge: Does the prompt test AI's knowledge and

1892 1893

1894

1895

1896

1897

1898

1899

1900

1901

1903	understanding in specific domain(s)?	tl
1904	The prompt must require strong	C
1905	prior knowledge or mastery of domain	C
1906	-specific concepts, theories, or	
1907	principles.	ai
1908	7 3. Complexity: Does the prompt contain	ai
1908		- 50000
	multiple components, variables, or	7 Score
1910	depth and nuance? This evaluates AI'	r
1911	s ability to handle complex,	ус
1912	multifaceted problems beyond simple	8
1913	queries.	9 Scori
1914	8 4. Problem-Solving: Does the prompt	10 1 poi
1915	require active problem-solving:	11 - Bas
1916	analyzing and clearly defining	b
1917	problems, then systematically	12 - No
1918	developing and implementing	р
1919	solutions? Note that active problem-	13 - Typ
1920	solving goes beyond reciting facts	14 -
1921	or following fixed instruction sets.	
1922	9 5. Creativity: Does the prompt require	
1923	creative approaches or solutions?	15 -
1924	This tests AI's ability to generate	
1925	novel ideas tailored to the specific	
1925	needs of the request or current	16 -
1920	problem.	10
1927	10 6. Technical Accuracy: Does the prompt	
1928		
1929	require answers with high technical	17
	accuracy, correctness, and precision	17 -
1931	? This evaluates the reliability and	
1932	truthfulness of AI outputs.	18 -
1933	11 7. Real-World Application: Does the	
1934	prompt relate to real-world	19 -
1935	applications? This tests AI's	
1936	ability to provide practical	20
1937	information that can be implemented	21 2 poi
1938	in real-life scenarios.	22 - Sui
1939	12	u
1940	13 # Output Example	23 - Typ
1941	14 Evaluation Process: (Ensure explanation	24 -
1942	before determining if input meets	
1943	each criterion)	
1944	15 Standards Met: (List standard numbers	
1945	met in Python array format, e.g.,	25 -
1946	[1, 2, 4, 6, 7])	
1947	A.8 Instruction Complexity Assessment	
	- ·	26 -
1948	1 You are an expert at evaluating the	
1949	difficulty of programming questions.	27 -
1950	Your responsibility is to assess	
1950	various types of questions,	
1951	including QA, multiple-choice,	28 -
1952		
	debugging tasks, code explanations,	29 -
1954	and more. Your goal is to assign a	
1955	difficulty score ranging from 1 (
1956	easiest) to 10 (most difficult).	30
1957	2 Note that you are tasked with evaluating	
1958	user-submitted programming	31 3 poi 32 - Sui
1959	questions rather than answering them	
1960	3	pi Tur
1961	4 Steps:	33 - Typ
1962	5 Think and Understand: First, read and	34 -
1963	think carefully to ensure that you	
1964	fully comprehend the questions	
1965	intent. Focus on what the question	35 -
1966	is asking and what skills or	
1967	knowledge are required to solve it.	
1967	6 Analysis: Based on your understanding,	36 -
	use the scoring criteria to assess	
1969	use the scoring criteria to assess	

	the difficulty of the question.	1970
	Consider factors such as the	1971
	complexity of the task, the level of	1972 1973
	programming experience required, and whether specialized knowledge is	1973
	needed.	1975
7	Score: Assign a score between 1 and 10,	1976
	reflecting the difficulty based on	1977
	your analysis.	1978
8	Cooning Critoria	1979
	Scoring Criteria: 1 points - Very Easy	1980 1981
	- Basic questions that programming	1982
	beginners can easily answer.	1983
12	- No specialized knowledge or prior	1984
	programming experience is required.	1985
	- Typical tasks include:	1986
14	 Simple syntax corrections (e.g., missing semicolons or 	1987 1988
	parentheses).	1989
15	- Basic input/output operations (e.g	1990
	., printing "Hello World" or	1991
	reading user input).	1992
16	- Basic variable assignments or	1993
	arithmetic operations (e.g.,	1994 1995
	assigning a value to a variable or adding two numbers).	1995
17	- Fixing a simple typo in a piece of	1997
	code.	1998
18	- Simple logical conditions (e.g.,	1999
	writing an if-else statement).	2000
19	- Basic loops (e.g., a for-loop to	2001
20	iterate over an array).	2002 2003
20	2 points – Basic Programming Task	2003
22	- Suited for beginners who have	2005
	undergone a short learning period.	2006
	- Typical tasks include:	2007
24	- Arrays and basic list	2008
	manipulations (e.g., accessing array elements, adding elements)	2009 2010
		2011
25	- Elementary software configuration	2012
	tasks (e.g., installing a	2013
	library, setting up an IDE,	2014
	configuring environment	2015
26	variables). - Writing basic functions that take	2016 2017
20	input and return output.	2018
27	- Basic debugging, such as finding	2019
	and fixing simple runtime errors	2020
		2021
28	- Basic file I/O (e.g., reading from	2022
29	and writing to a file). - Writing functions that involve	2023 2024
29	loops, conditionals, and data	2025
	manipulation.	2026
30		2027
	3 points - Common Programming Task	2028
32		2029
33	programming experience. - Typical tasks include:	2030 2031
33 34	- Basic use of common data	2032
	structures like lists or	2033
	dictionaries.	2034
35	- Simple algorithms like sorting (e.	2035
	g., bubble sort) and linear	2036
21	search. - Software development tasks like	2037
36	 Software development tasks like basic database operations (e.g., 	2038 2039
		2000

2040			inserting or querying data from	
2041			a database).	67
2042	37		 Implementing basic math functions 	
2043			(e.g., finding the greatest	
2044			common divisor).	68
			· · · · · · · · · · · · · · · · · · ·	08
2045	38		- Basic error handling (e.g., using	
2046			try-catch blocks).	
2047	39		- Introduction to object-oriented	
2048			programming (e.g., creating	
2049			classes and objects).	69
2050	40		5 <i>,</i>	
2051		1	points - Entry-Level	
				=0
2052	42	-	Suitable for developers just starting	70
2053			out in software development.	
2054	43	-	51	
2055	44		- Data structures like linked lists,	
2056			hash tables, stack and queue (e	71
2057			.g., implementing a singly	
2058			linked list).	
2059	45		- Algorithms like binary search,	
2060			insertion sort.	72
2061	46		- Simple server-side programming (e.	73 7
2062			g., writing a basic HTTP server)	74 -
2063				
2064	47		– Designing and implementing basic	75 -
2065	·τ /		APIs.	
				76
2066	48		- Debugging and testing small	
2067			codebases (e.g., writing unit	
2068			tests).	77
2069	49			
2070	50	5	points - Lower Intermediate	
2071		_	• • • • • • • • • • • • • • • • • • • •	
2072	51		of experience.	
			•	=0
2073		-	Typical tasks include:	78
2074	53		- Complex algorithms (e.g.,	
2075			improving the time complexity of	
2076			sorting from O(n) to O(n log	
2077			n)).	79
2078	54		- Complex data structures (e.g.,	
2079	51		binary tree, heap).	
2080	55			
	55		- Object-oriented programming with	
2081			inheritance, polymorphism, and	80
2082			encapsulation (e.g., designing a	
2083			class hierarchy).	
2084	56		- Basic functional programming	
2085			concepts (e.g., lambda	
2086			expressions, higher-order	81
2087			functions).	01
2088	57		- Code debugging and performance	
2089			optimization (e.g., optimizing a	82
2090			recursive function).	83 8
2091	58		 Development of small-scale systems 	84 -
2092			, such as building a RESTful API	
2093			or optimizing a database query.	85 -
2094	59		- Implementing simple design	86
2094	59		patterns (e.g., Singleton,	00
2096			Factory).	
2097	60		- Using version control systems like	87
2098			Git for basic collaboration	
2099			tasks.	
2100	61			88
2101		6	points - Intermediate	
2102	63	_	Suitable for developers with 3-4 years	
	05			9.0
2103			of experience.	89
2104	64	-	Typical tasks include:	
2105	65		- Involvement with multi-module	
2106			projects, such as writing	
2107			modular and reusable code across	90
2108			different components.	
2109	66		- More complex data algorithms like	
	00			

	greedy and backtracking.	2110
	- Performance optimizations (e.g.,	2111
	improving the time complexity of	2112
	algorithms).	2113
	 Designing and implementing moderately complex API 	2114 2115
	interfaces (e.g., handling	2115
	authentication and rate limiting	2117
).	2118
	- Service integration (e.g.,	2119
	integrating a third-party API	2120
	into a project).	2121
)	- Developing small-to-medium-sized	2122
	system modules (e.g., creating a caching layer for an	2123 2124
	application).	2124
	- Concurrency control in programming	2126
	(e.g., handling race conditions	2127
	in multi-threaded environments)	2128
		2129
-	· · · · · · · · · ·	2130
/	points - Upper Intermediate	2131 2132
-	Suitable for developers with 5-6 years of experience.	2132
-	Typical tasks include:	2133
,	- Complex system designs, requiring	2135
	architectural understanding of	2136
	multi-tier applications.	2137
r	- Working with more complex data	2138
	structures like balanced trees (2139
	e.g., AVL trees) and graphs (e.g ., BFS, DFS) and algorithms (e.g	2140 2141
	. dynamic programming).	2141
:	- Tackling advanced multithreading	2143
	and synchronization issues (e.g	2144
	., handling deadlock in	2145
	concurrent programming).	2146
)	- Distributed system design and	2147
	implementation (e.g., designing a distributed file storage	2148 2149
	system).	2149
)	- Building and optimizing high-	2151
	concurrency models (e.g.,	2152
	designing a system to handle	2153
	millions of simultaneous	2154
	requests).	2155
	- Designing and implementing	2156
	advanced networked applications (e.g., web crawlers).	2157 2158
,	(e.g., web clawiels).	2150
8	points - Advanced	2160
_	Suitable for developers with 7-10	2161
	years of experience.	2162
-	Typical tasks include:	2163
0	- Advanced dynamic programming	2164
	problems (e.g., solving longest	2165
,	common subsequence problems). - Complex graph algorithms (e.g.,	2166 2167
	implementing Dijkstras or A*	2167
	algorithms).	2169
	- Working with complex technical	2170
	stacks that span multiple	2171
	platforms and languages.	2172
)	- Solving distributed system	2173
	challenges (e.g., ensuring data	2174
	consistency across a distributed database).	2175 2176
)	- Advanced performance optimization	2173
	tasks (e.g., reducing latency in	2178
	real-time systems).	2179

	91	 Complex concurrency models and 	115
2180	91		115
2181		synchronization across multiple	
2182		threads or processes.	
2183	92	- System performance tuning at scale	116
2184	12		110
		(e.g., profiling and optimizing	
2185		system performance for millions	
2186		of users).	
2187	93	- Cross-domain integrations (e.g.,	
	15		
2188		integrating machine learning	
2189		models into production-level	
2190		systems).	
2191	94		
2192		9 points - Expert Level	
2193	96	- Suitable for developers with over 10	
2194		years of experience.	1
2195	97	- Typical tasks include:	
2196	98	- Designing and implementing domain-	
	20		
2197		specific languages (DSLs).	
2198	99	- Developing low-level hardware	
2199		drivers (e.g., writing a device	
2200		driver in C).	
2201	100	- Real-time system design (e.g.,	
	100		
2202		building systems for high-	
2203		frequency trading platforms).	
2204	101	- Working with highly specialized	
2205		technologies (e.g., cryptography	
2206		, embedded systems).	
2207	102	- Building complex, large-scale	
2208		distributed systems that require	
2209		deep expertise (e.g., designing	
2210			
		a global load balancer for a	1
2211		cloud platform).	
2212	103	– Handling complex issues in system	
2213		architecture (e.g., data	
2214		sharding and replication across	
2215		geographically distributed	
2216		servers).	
2217	104		
2218	105	10 points – Academic Research and	
2219		Innovation	
2220	106	- Reserved for top-tier experts,	
	100		
2221		typically in academia or cutting-	2
2222		edge research. These task usually	3
2223		requires deep expertise in system	4
2224		architecture and understanding of	+
2225		limitations in computing, as well as	5
2226			6
2226		potentially coming up with an	6 7
2227		potentially coming up with an innovative or theoretical solution	
2227 2228		potentially coming up with an	7
2227 2228		potentially coming up with an innovative or theoretical solution that goes beyond conventional	7
2227 2228 2229	107	potentially coming up with an innovative or theoretical solution that goes beyond conventional programming.	7 8
2227 2228 2229 2230	107	potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include:	7
2227 2228 2229 2230 2231	107 108	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e.</pre>	7 8
2227 2228 2229 2230		potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include:	7 8 9
2227 2228 2229 2230 2231		<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e.</pre>	7 8
2227 2228 2229 2230 2231 2232 2233		<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms).</pre>	7 8 9
2227 2228 2229 2230 2231 2232 2233 2233	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions.</pre>	7 8 9
2227 2228 2229 2230 2231 2232 2233 2233 2234 2235	108	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that</pre>	7 8 9
2227 2228 2229 2230 2231 2232 2233 2233 2234 2235 2236	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative</pre>	7 8 9 10
2227 2228 2229 2230 2231 2232 2233 2233 2234 2235	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve</pre>	7 8 9 10 11 12
2227 2228 2229 2230 2231 2232 2233 2233 2234 2235 2236	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative</pre>	7 8 9 10 11 12 13
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2236 2237 2238	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g</pre>	7 8 9 10 11 12 13 14
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2236 2237 2238 2239	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to</pre>	7 8 9 10 11 12 13
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2239 2240	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems).</pre>	7 8 9 10 11 12 13 14
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2239 2240 2241	108 109	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in</pre>	7 8 9 10 11 12 13 14 15
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2239 2240	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields.</pre>	7 8 9 10 11 12 13 14 15 16 17
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2239 2240 2241	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields.</pre>	7 8 9 10 11 12 13 14 15 16 17 18
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that</pre>	7 8 9 10 11 12 13 14 15 16 17 18 19
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that require significant technical</pre>	7 8 9 10 11 12 13 14 15 16 17 18
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2243	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that require significant technical and academic expertise (e.g.,</pre>	7 8 9 10 11 12 13 14 15 16 17 18 19
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2245 2246	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that require significant technical and academic expertise (e.g., creating a new programming</pre>	7 8 9 10 11 12 13 14 15 16 17 18 19 20
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2243	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that require significant technical and academic expertise (e.g.,</pre>	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2245 2246	108 109 110	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that require significant technical and academic expertise (e.g., creating a new programming</pre>	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2244 2245 2246 2247	108 109 110 111 112	<pre>potentially coming up with an innovative or theoretical solution that goes beyond conventional programming. - Typical tasks include: - Designing brand new algorithms (e. g., quantum computing algorithms). - Creating breakthrough solutions. - Solving complex challenges that require original, creative thinking and often involve interdisciplinary knowledge (e.g ., combining AI with robotics to solve novel problems). - Research and development in bleeding-edge fields. - Leading complex projects that require significant technical and academic expertise (e.g., creating a new programming</pre>	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

 Please adhere strictly to the provided output format in the few -shot examples. Your response should consist of three essential sections: Thinking Steps, Analysis, Json Output. 	2250 2251 2252 2253 2254 2255 2256
B Static Analysis Examples	2257
B.1 Python	2258
Prompt	2259

Extend the given function to support	
multiple regular expression patterns	
. Each pattern may involve various	
named groups, and multiple string	
lists should be processed. Each	
string across these lists must be	
parsed twice using different	
configurations. Successful matches	
should have their results compiled	
into dictionaries. Address handling	
of edge cases, specifically focusing	
on empty strings and non-matches,	
while verifying processing between	
different lists and pattern sets.	

Response

Certainly! Let's extend a given function
to support parsing multiple lists
of strings using multiple regular
expression patterns. Each pattern
will use named groups, and we will
process each string twice using
different configurations. Successful
matches will be compiled into
dictionaries. We will also address
handling of edge cases like empty
strings and non-matches.
Here's a Python implementation:
```Python

```
import re
6 from typing import List, Dict, Any
 {\tt def parse\_strings\_with\_patterns(patterns})
 : List[str], string_lists: List[List
 [str]]) -> List[Dict[str, Any]]:
 # Compile all regex patterns
 beforehand for efficiency
 compiled_patterns = [re.compile(
 pattern) for pattern in patterns
]
 # Storage for the parsed results
 results = []
 # Iterate over each list of strings
```

```
for strings in string_lists:
 # Process each string
 for s in strings:
 # Skip empty strings
 if not s:
 continue
```

```
Prepare a dictionary to
 hold results for this
 particular string
```

2315	24	<pre>string_results = {'string':</pre>	append({	2385
	24			
2316		s, 'matches': []}	49 <b>'config': '</b>	2386
2317	25		findall	2387
			, inddit	
2318	26	<pre># Parse each string using</pre>	Ϊ,	2388
2319		each pattern twice with	50 'groups': {	2389
2320		different configurations	list(	2390
2321	27	for pattern in	pattern.	2391
2322	_ /		•	
		compiled_patterns:	groupindex	2392
2323	28	# First attempt: direct	. keys())	2393
2324		match	[0]:	2394
2325	29	direct_match = pattern.	match}	2395
2326		match(s)	for	2396
2327	30	if direct_match:	match in	2397
2328	31	# Save matched		2398
	51			
2329		groups	all_matches	2399
2330	32	string_results['		2400
	02			
2331			51 })	2401
2332		({	52	2402
2333	22		53 # Handle non-matches	2403
	33	-		
2334		direct_match	54 if not string_results['	2404
2335		1	matches ']:	2405
		, ,		
2336	34	'groups':	55 string_results['matches	2406
2337		direct_match	'].append({	2407
		—	- 11 (6	
2338		.groupdict()	56 'config': 'nomatch',	2408
2339	35	})	57 'groups': None	2409
			8 1	
2340	36		58 })	2410
2341	37	<pre># Second attempt: match</pre>	59	2411
2342			60 # Append the result for this	2412
2343	38	all_matches = pattern.	string	2413
2344		findall(s)	results.append(	2414
		. ,		
2345	39	if all_matches:	string_results)	2415
2346	40	# The result format	62	2416
	40			
2347		of findall	63 return results	2417
2348		differs, we	64	2418
2349		handle	65 # Example usage	2419
2350		conversion here	66 patterns = [	2420
2351	41	if isinstance(	67 r'(?P <word>\w+)',</word>	2421
	41	•		
2352		all_matches[0],	68 r'(?P <number>\d+)',</number>	2422
2353		tuple):	69	2423
2354	42	for match in	70	2424
2355		all_matches:	71 string_lists = [	2425
2356	10			0.406
	43	string_results		2426
2357		['	73 ["NoMatch", "789"]	2427
2358			74 ]	2428
2359		'].	75	2429
2360		append ({	<pre>76 results = parse_strings_with_patterns(</pre>	2430
2361	44	'config	patterns, string_lists)	2431
2362		· · · · · · · · · · · · · · · · · · ·	77 for result in results:	2432
2363		findal	18 print(result)	2433
		i Illuat.		
2364		',	79	2434
2365	45	'groups	80 ### Explanation:	2435
		0 1	•	
2366			81	2436
2367		k: v	82 1. **Multiple Patterns**: We accept a	2437
2368		for	list of patterns. Each pattern is	2438
2369		k,	expected to have named groups.	2439
2370			83	2440
2371		zip	84 2. **Multiple String Lists**: We accept	2441
2372		(	multiple lists, each containing	2442
2373		patter	n strings to be parsed.	2443
2374		<u>.</u>	85	2444
		•	nday we Compiling Dottomouture All motore	
2375		groupi	mdðx **Compiling Patterns**: All patterns	2445
2376			are compiled up front for better	2446
2377		keys	performance.	2447
			periormance.	
2378		(),	87	2448
2379			88 4. **Processing Each String**: For each	2449
2380		)}	string in the lists:	2450
2381	46	})	89 - Skip empty strings.	2451
2382	47		90 - Attempt to match using each	2452
2383	48	string_results['	compiled pattern twice:	2453
2384			· · ·	2454
2304		matches ].	91 - Direct match using `pattern.match	2434

2455		
2456	92 - Find all matches using `pattern.	
2457	findall()` which returns tuples	1
2458	for multiple groups.	1
2459	93 - Extract named groups for successful	
2460	matches and append them to the	
2461	results.	
2462	94	
2463	95 5. **Handling Edge Cases**:	
2464	96 – Empty strings are skipped.	
2465	97 - If no matches are found for a	
2466	string, an entry is added with `	
2467	config set to `nomatch`.	
2468	98	
2469	99 6. **Compiling Results**: All successful	
2470	matches and non-matches are	
2471	collected in a structured format	
2472	into a dictionary list that	
2473	maintains all relevant information	2
		3
2474	for further processing or analysis.	4
2475	100	5
2476	101 This extended function systematically	
2477	processes a potentially complex set	0
2478	of inputs and configurations.	7
	, , , , , , , , , , , , , , , , , , ,	8
2479	providing a robust solution for	9
2480	varied parsing needs.	10
		11
2481	Static analysis result	
0.100	·	12
2482	1 {	13
2483	2 "issues": [	14
2484	3 {	15
2485	<pre>4 "rule_name": "E0001:syntax-error",</pre>	16
2486	5 "message": "Parsing failed: '	17
2487	invalid syntax (current file,	18
2488	line 46)'",	19
2489	<pre>6 "position": {"start_line": 51, "</pre>	20
2403		
		21
2490	end_line": 51},	21
2490 2491	end_line": 51}, 7	22
2490	end_line": 51}, 7 "severity": "error", 8 "content": "	
2490 2491	end_line": 51}, 7	22
2490 2491 2492	end_line": 51}, 7 "severity": "error", 8 "content": "	22
2490 2491 2492 2493 2494	end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern.	22 23
2490 2491 2492 2493 2494 2495	end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match}	22
2490 2491 2492 2493 2494 2495 2495 2496	end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches",	22 23
2490 2491 2492 2493 2494 2495 2496 2497	end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python"	22 23
2490 2491 2492 2493 2494 2495 2495 2496	end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches",	22 23
2490 2491 2492 2493 2494 2495 2496 2497	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" }</pre>	22 23 24
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499	<pre>end_line": 51}, 7 "severity": "error", 8 "content": "</pre>	22 23 24 25
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500	<pre>end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python" 10 } 11 ], 12 "status": "fail"</pre>	22 23 24 25 26
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499	<pre>end_line": 51}, 7 "severity": "error", 8 "content": "</pre>	22 23 24 25
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500	<pre>end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python" 10 } 11 ], 12 "status": "fail"</pre>	22 23 24 25 26 27
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501	<pre>end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python" 10 } 11 ], 12 "status": "fail" 13 }</pre>	22 23 24 25 26
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500	<pre>end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python" 10 } 11 ], 12 "status": "fail"</pre>	22 23 24 25 26 27
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } 11 ], 12 "status": "fail" 13 } B.2 C/C++</pre>	22 23 24 25 26 27 28
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501	<pre>end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python" 10 } 11 ], 12 "status": "fail" 13 }</pre>	22 23 24 25 26 27 28
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2502	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } 1], 2] "status": "fail" B.2 C/C++ Prompt</pre>	22 23 24 25 26 27 28 29
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504	<pre>end_line": 51}, 7 "severity": "error", 8 "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", 9 "language": "Python" 10 } 11 ], 12 "status": "fail" 13 } B.2 C/C++ Prompt 1 Create a thread-safe task manager class</pre>	222 23 24 25 26 27 28 29 30
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to</pre>	22 23 24 25 26 27 28 29 30 31
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], "status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to :</pre>	222 23 24 25 26 27 28 29 30 31 32
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to</pre>	22 23 24 25 26 27 28 29 30 31
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], "status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to :</pre>	222 23 24 25 26 27 28 29 30 31 32
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } "language": "Python" } J, "status": "fail" B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 3 - Add new tasks with associated priority</pre>	222 23 24 25 26 27 28 29 30 31 32 33 34
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } "language": "Python" } J, "status": "fail" B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number</pre>	222 233 244 255 266 277 288 299 300 311 322 333 344 355
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } "language": "Python" } J, "status": "fail" B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number of tasks running concurrently.</pre>	222 233 244 255 266 277 288 299 300 311 322 334 355 366
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511	<pre>end_line": 51},     "severity": "error",     "content": "     groups': {list(pattern.     groupindex.keys())[0]: match}     for match in all_matches",     "language": "Python"     }      B.2 C/C++     Prompt     Create a thread-safe task manager class     in C++. This class should be able to     :     2     - Add new tasks with associated priority     levels, while limiting the number     of tasks running concurrently.     4 - Mark tasks as completed, ensuring that </pre>	222 233 244 255 266 277 288 299 300 311 322 333 344 355
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number of tasks running concurrently. 4 - Mark tasks as completed, ensuring that tasks with dependencies only</pre>	222 233 244 255 266 277 288 299 300 311 322 334 355 366
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511	<pre>end_line": 51},     "severity": "error",     "content": "     groups': {list(pattern.     groupindex.keys())[0]: match}     for match in all_matches",     "language": "Python"     }      B.2 C/C++     Prompt     Create a thread-safe task manager class     in C++. This class should be able to     :     2     - Add new tasks with associated priority     levels, while limiting the number     of tasks running concurrently.     4 - Mark tasks as completed, ensuring that </pre>	222 233 244 255 266 277 288 299 300 311 322 334 355 366
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number of tasks running concurrently. 4 - Mark tasks as completed, ensuring that tasks with dependencies only</pre>	222 233 244 255 266 277 288 299 300 311 322 334 355 366
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number of tasks running concurrently. 4 - Mark tasks as completed, ensuring that tasks with dependencies only proceed once dependencies are completed.</pre>	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2507 2508 2509 2510 2511 2512 2513 2514 2515	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number of tasks running concurrently. 4 - Mark tasks as completed, ensuring that tasks with dependencies only proceed once dependencies are completed. 5 - Reset the completion status of all</pre>	222 233 244 255 266 277 288 299 300 311 322 334 356 377
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2507 2508 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516	<pre>end_line": 51}, "severity": "error", "content": " groups': {list(pattern. groupindex.keys())[0]: match} for match in all_matches", "language": "Python" } ], ], ? status": "fail" ] B.2 C/C++ Prompt Create a thread-safe task manager class in C++. This class should be able to : 2 - Add new tasks with associated priority levels, while limiting the number of tasks running concurrently. 4 - Mark tasks as completed, ensuring that tasks with dependencies only proceed once dependencies are completed. 5 - Reset the completion status of all tasks.</pre>	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517	<pre>end_line": 51},     "severity": "error",     "content": "     groups': {list(pattern.     groupindex.keys())[0]: match}     for match in all_matches",     "language": "Python"     }     ],     "status": "fail"     B.2 C/C++     Prompt     Create a thread-safe task manager class     in C++. This class should be able to     :     2     - Add new tasks with associated priority     levels, while limiting the number     of tasks running concurrently.     Mark tasks as completed, ensuring that     tasks with dependencies only     proceed once dependencies are     completed.     - Reset the completion status of all     tasks.     - Guarantee multi-threaded safety and </pre>	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518	<pre>end_line": 51},     "severity": "error",     "content": "     groups': {list(pattern.     groupindex.keys())[0]: match}     for match in all_matches",     "language": "Python"     }     ],     "status": "fail"     B.2 C/C++     Prompt     Create a thread-safe task manager class     in C++. This class should be able to     :     2     - Add new tasks with associated priority     levels, while limiting the number     of tasks running concurrently.     Mark tasks as completed, ensuring that     tasks with dependencies are     completed.     - Reset the completion status of all     tasks.     - Guarantee multi-threaded safety and     manage task execution in accordance </pre>	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517	<pre>end_line": 51},     "severity": "error",     "content": "     groups': {list(pattern.     groupindex.keys())[0]: match}     for match in all_matches",     "language": "Python"     }     ],     "status": "fail"     B.2 C/C++     Prompt     Create a thread-safe task manager class     in C++. This class should be able to     :     2     - Add new tasks with associated priority     levels, while limiting the number     of tasks running concurrently.     Mark tasks as completed, ensuring that     tasks with dependencies only     proceed once dependencies are     completed.     - Reset the completion status of all     tasks.     - Guarantee multi-threaded safety and </pre>	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518	<pre>end_line": 51},     "severity": "error",     "content": "     groups': {list(pattern.     groupindex.keys())[0]: match}     for match in all_matches",     "language": "Python"     }     ],     "status": "fail"     B.2 C/C++     Prompt     Create a thread-safe task manager class     in C++. This class should be able to     :     2     - Add new tasks with associated priority     levels, while limiting the number     of tasks running concurrently.     Mark tasks as completed, ensuring that     tasks with dependencies are     completed.     - Reset the completion status of all     tasks.     - Guarantee multi-threaded safety and     manage task execution in accordance </pre>	22 22 22 22 22 22 22 22 22 22 22 22 22

# Response

1 To create a thread-safe task manager in	2522
C++, we will use several components	2523
from the C++ Standard Library,	2524
including `std::mutex`, `std::	2525
<pre>condition_variable`, and `std::</pre>	2526
thread`. We will use a priority	2527
	2528
queue to manage task priorities and	
a mechanism to track task	2529
dependencies. This example will	2530
focus on providing core	2531
functionalities and thread safety,	2532
but it's important to know that	2533
fleshing it out for production use	2534
might require additional error	2535
checking and optimization.	2536
2	2537
³ Here's a basic implementation:	2538
4 ```cpp	2539
5 #include <iostream></iostream>	2540
6 #include <queue></queue>	2541
<pre>7 #include <unordered_map></unordered_map></pre>	2542
<pre>8 #include <unordered_set></unordered_set></pre>	2543
<pre>9 #include <vector></vector></pre>	2544
10 #include <thread></thread>	2545
<pre>m #include <mutex></mutex></pre>	2546
<pre>12 #include <condition_variable></condition_variable></pre>	2547
<pre>13 #include <functional></functional></pre>	2548
14 #include <atomic></atomic>	2549
	2550
15	
16 // Task structure	2551
17 struct Task {	2552
18 int id;	2553
19 int priority;	2554
<pre>20 std::function<void()> execute;</void()></pre>	2555
	2556
<pre>std::vector<int> dependencies;</int></pre>	
22	2557
23 Task(int id, int priority, std::	2558
function <void()> f, std::vector&lt;</void()>	2559
int> deps = {})	2560
24 : id(id), priority(priority),	2561
execute(std::move(f)),	2562
<pre>dependencies(std::move(deps)</pre>	2563
) {}	2564
25 };	2565
26	2566
27 // Comparator for priority queue (higher	2567
priority runs first)	2568
28 struct TaskCompare {	2569
29 bool operator()(const Task& t1,	2570
const Task& t2) {	2571
30 return t1.priority < t2.priority	2572
:	2573
31 }	2574
32 };	2575
33	2576
34 class TaskManager {	2577
35 public:	2578
36 TaskManager(int maxConcurrentTasks)	2579
37 : maxConcurrentTasks(	2580
maxConcurrentTasks),	2581
<pre>activeTasks(0) {}</pre>	2582
38	2583
39 void addTask(int id, int priority,	2584
<pre>std::function<void()></void()></pre>	2585
<pre>taskFunction, std::vector<int></int></pre>	2586
dependencies = {}) {	2587
40 std::unique_lock <std::mutex></std::mutex>	2588
	2J00
lock(mutex);	2589

2590	41	tasks[id] = Task(id, priority,	88	
2591		<pre>std::move(taskFunction), std</pre>	89 pri	vate:
2592		::move(dependencies));	90	<pre>int maxConcurrentTasks;</pre>
2593	42	if (dependencies.empty()) {	91	<pre>std::atomic<int> activeTasks;</int></pre>
2594	43	<pre>taskQueue.push(tasks[id]);</pre>	92	<pre>std::unordered_map<int, task=""> tasks;</int,></pre>
2595	44	<pre>cv.notify_one();</pre>	93	<pre>std::unordered_set<int></int></pre>
2596	45	}		<pre>completedTasks;</pre>
2597	46	}	94	<pre>std::unordered_set <int> waitingTasks</int></pre>
2598	47	5		;
2599	48	<pre>void markTaskCompleted(int id) {</pre>	95	<pre>std::priority_queue<task, pre="" std::<=""></task,></pre>
2600	49	<pre>std::unique_lock<std::mutex></std::mutex></pre>		vector <task>, TaskCompare&gt;</task>
2601		lock(mutex);		taskQueue;
2602	50	completedTasks.insert(id);	96	std::mutex mutex;
2603	51	for (auto& it : tasks) {	97	<pre>std::condition_variable cv;</pre>
2604	52	Task& task = it.second;	98	
2605	53	if (!completedTasks.count(	99	<pre>void workerThread() {</pre>
2606	55	task.id) && !		while (true) {
2607		waitingTasks.count(task.	100	
			101	Task task(0, 0, []{}, {});
2608		id)) {	102	{
2609	54	auto& deps = task.	103	<pre>std::unique_lock<std::< pre=""></std::<></pre>
2610		dependencies;		<pre>mutex&gt; lock(mutex);</pre>
2611	55	if (std::all_of(deps.	104	cv.wait(lock, [this] {
2612		<pre>begin(), deps.end(),</pre>		return !taskQueue.
2613		[this](int dep) {		empty()
2614		return		<pre>activeTasks &gt; 0; });</pre>
2615		completedTasks.count	105	
2616		(dep); })) {	106	if (taskQueue.empty()) {
2617	56	<pre>taskQueue.push(task)</pre>	107	break;
2618		;	108	}
2619	57	waitingTasks.erase(	109	-
2620		task.id);	110	<pre>task = taskQueue.top();</pre>
2621	58	cv.notify_one();	111	taskQueue.pop();
2622	59	}	112	++activeTasks;
2623	60	}		}
2624			113	}
-	61	}	114	11 Execute the teck
2625	62	}	115	// Execute the task
2626	63		116	<pre>task.execute();</pre>
2627	64	<pre>void resetCompletion() {</pre>	117	<pre>markTaskCompleted(task.id);</pre>
2628	65	<pre>std::unique_lock<std::mutex></std::mutex></pre>	118	_
2629		<pre>lock(mutex);</pre>	119	{
2630	66	<pre>completedTasks.clear();</pre>	120	<pre>std::unique_lock<std::< pre=""></std::<></pre>
2631	67	while (!taskQueue.empty())		<pre>mutex &gt; lock(mutex);</pre>
2632		taskQueue.pop();	121	activeTasks;
2633	68	waitingTasks.clear();	122	<pre>cv.notify_all();</pre>
2634	69	for (auto& it : tasks) {	123	}
2635	70	<pre>if (it.second.dependencies.</pre>	124	}
2636		empty()) {	125	}
2637	71	<pre>taskQueue.push(it.second</pre>	126 };	
2638		);	127	
2639	72	} else {		main() {
2640	73	waitingTasks.insert(it.	129	TaskManager tm(3); // Allow 3 tasks
2641		second.id);		to run concurrently
2642	74	}	130	
2643	75	}	130	tm.addTask(1, 10, [] {
2644	75	cv.notify_all();	1.71	"Running Task 1\n"; });
2077		-	132	
26/5			1.17	
2645 2646	77	}	102	<pre>tm.addTask(2, 20, [] { std::cout &lt;&lt;     "Pupping Task 2)n": });</pre>
2646	78			"Running Task 2\n"; });
2646 2647	78 79	void run() {	133	"Running Task 2\n";
2646 2647 2648	78		133	"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout << "Running Task 3\n"; });
2646 2647 2648 2649	78 79 80	<pre>void run() {     std::vector<std::thread> threads     ;</std::thread></pre>		<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt;</pre>
2646 2647 2648 2649 2650	78 79	<pre>void run() {     std::vector<std::thread> threads     ;     for (int i = 0; i &lt;</std::thread></pre>	133	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2});</pre>
2646 2647 2648 2649 2650 2651	78 79 80	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {</std::thread></pre>	133	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt;</pre>
2646 2647 2648 2649 2650	78 79 80	<pre>void run() {     std::vector<std::thread> threads     ;     for (int i = 0; i &lt;</std::thread></pre>	133 134	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2});</pre>
2646 2647 2648 2649 2650 2651	78 79 80 81	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {</std::thread></pre>	133 134	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt;</pre>
2646 2647 2648 2649 2650 2651 2652	78 79 80 81	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {       threads.emplace_back(&amp;</std::thread></pre>	133 134 135	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt;</pre>
2646 2647 2648 2649 2650 2651 2652 2653	78 79 80 81	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {       threads.emplace_back(&amp;       TaskManager::</std::thread></pre>	133 134 135 136	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt; "Running Task 5\n"; }, {3});</pre>
2646 2647 2648 2649 2650 2651 2652 2653 2654	78 79 80 81 82	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {       threads.emplace_back(&amp;       TaskManager::          workerThread, this);    } </std::thread></pre>	133 134 135 136	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt; "Running Task 5\n"; }, {3}); // Run tasks, observing priorities</pre>
2646 2647 2648 2650 2651 2652 2653 2654 2655	78 79 80 81 82 83	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {       threads.emplace_back(&amp;        TaskManager::            workerThread, this);    }    for (auto&amp; thread : threads) {</std::thread></pre>	133 134 135 136 137	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt; "Running Task 5\n"; }, {3});</pre>
2646 2647 2648 2650 2651 2652 2653 2654 2655 2655	78 79 80 81 82 83 84	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {       threads.emplace_back(&amp;       TaskManager::          workerThread, this);    }    for (auto&amp; thread : threads) {       thread.join();    } </std::thread></pre>	133 134 135 136 137 138	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt; "Running Task 5\n"; }, {3}); // Run tasks, observing priorities and dependencies tm.run();</pre>
2646 2647 2648 2650 2651 2652 2653 2654 2655 2656 2656 2657	78 79 80 81 82 83 84 85	<pre>void run() {    std::vector<std::thread> threads    ;    for (int i = 0; i &lt;       maxConcurrentTasks; ++i) {       threads.emplace_back(&amp;        TaskManager::            workerThread, this);    }    for (auto&amp; thread : threads) {</std::thread></pre>	133 134 135 136 137 138 139	<pre>"Running Task 2\n"; }); tm.addTask(3, 15, [] { std::cout &lt;&lt; "Running Task 3\n"; }); tm.addTask(4, 30, [] { std::cout &lt;&lt; "Running Task 4\n"; }, {1, 2}); tm.addTask(5, 25, [] { std::cout &lt;&lt; "Running Task 5\n"; }, {3});</pre>

2730	142		23
2731	143		24
2732		This implementation includes.	
2132	144	This implementation includes:	25
2733	145		26
2734	146	- **Task Structure**: Defines properties	27
2735		of a task, including its ID,	
2133			
2736		priority, execution function, and	
2737		dependencies.	
2738	147	- **Task Manager Class**: Manages tasks,	
	147		
2739		constraints, and execution using a	
2740			
		priority queue, mutex, and condition	1
2741		variables.	1
2742	148	- **Add Task Method**: Adds tasks,	
2743		specifying priorities and optional	
2744		dependencies.	
2745	1.40	- **Mark Task Completed Method**: Marks	
2143	149		
2746		tasks as completed and checks if	
0747		•	
2747		dependent tasks can proceed.	
2748	150	- **Reset Method**: Resets all tasks'	
	1.50		
2749		completion status and restarts	
2750		execution.	
2751	151	- **Worker Threads**: Manage concurrent	
2752		execution of tasks while observing	
2753		their priorities and dependencies.	
2754	150	· ·	
	152		
2755	153	This setup allows for scalable and	
		officient tack management in a	
2756		efficient task management in a	
2757		multithreaded context, accommodating	
2758		dependency resolution and priority-	
2759		based execution. More features and	
2760		error-handling might be necessary	
2761		for a production system.	
2101		for a production system.	1
2762		Static analysis result	
		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	
0=00		_	
	1	1	
2763	1	{	
2763	1 2	{ "issues": [	
2764	2	"issues": [	
		"issues": [ {	
2764	2	"issues": [ {	
2764 2765 2766	2 3 4	"issues": [ { "rule_name": "error",	
2764 2765 2766 2767	2 3	"issues": [ { "rule_name": "error", "message": "'all_of' is not a	
2764 2765 2766	2 3 4	"issues": [ { "rule_name": "error", "message": "'all_of' is not a	
2764 2765 2766 2767 2768	2 3 4	"issues": [ { "rule_name": "error", "message": "'all_of' is not a member of 'std'\n if (std::	
2764 2765 2766 2767 2768 2769	2 3 4	"issues": [ { "rule_name": "error", "message": "'all_of' is not a member of 'std'\n if (std:: all_of(deps.begin(), deps.end	
2764 2765 2766 2767 2768	2 3 4	"issues": [ { "rule_name": "error", "message": "'all_of' is not a member of 'std'\n if (std:: all_of(deps.begin(), deps.end	
2764 2765 2766 2767 2768 2769 2770	2 3 4	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return</pre>	
2764 2765 2766 2767 2768 2769 2770 2771	2 3 4	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))</pre>	2
2764 2765 2766 2767 2768 2769 2770	2 3 4	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))</pre>	2
2764 2765 2766 2767 2768 2769 2770 2771 2772	2 3 4 5	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",</pre>	3
2764 2765 2766 2767 2768 2769 2770 2771	2 3 4	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {</pre>	
2764 2765 2766 2767 2768 2769 2770 2771 2772	2 3 4 5	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {</pre>	3 4
2764 2765 2766 2767 2768 2769 2770 2771 2772 2773 2774	2 3 4 5 6 7	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~",         "position": {             "start_line": 56,         }         }     } }</pre>	3 4 5
2764 2765 2766 2767 2768 2769 2770 2771 2772 2773	2 3 4 5	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {</pre>	3 4
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775	2 3 4 5 6 7 8	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         "end_line": 56         "         "         "istart_line": 56         "         "         "</pre>	3 4 5 6
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776	2 3 4 5 6 7 8 9	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },     } }</pre>	3 4 5 6 7
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775	2 3 4 5 6 7 8	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",</pre>	3 4 5 6
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777	2 3 4 5 6 7 8 9 10	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",</pre>	3 4 5 6 7 8
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778	2 3 4 5 6 7 8 9	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         ])         [             "start_line]         ])         [             "severity": "error",         "content": " if (std::all_of(deps.         ])         [         [         [</pre>	3 4 5 6 7 8 9
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777	2 3 4 5 6 7 8 9 10	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",</pre>	3 4 5 6 7 8
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779	2 3 4 5 6 7 8 9 10	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.             begin(), deps.end(), [this](</pre>	3 4 5 6 7 8 9
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780	2 3 4 5 6 7 8 9 10	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return</pre>	3 4 5 6 7 8 9 10 11
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779	2 3 4 5 6 7 8 9 10	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.             begin(), deps.end(), [this](</pre>	3 4 5 6 7 8 9
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781	2 3 4 5 6 7 8 9 10	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))</pre>	3 4 5 6 7 8 9 10 11 12
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782	2 3 4 5 6 7 8 9 10 11	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         ""         ""         ""</pre>	3 4 5 6 7 8 9 10 11
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781	2 3 4 5 6 7 8 9 10	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))</pre>	3 4 5 6 7 8 9 10 11 12
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783	2 3 4 5 5 6 7 8 9 100 111 12	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.             begin(), deps.end(), [this](             int dep) { return             completedTasks.count(dep); }))         {"         return         completedTasks.count(dep); }))         {"         return         completedTasks.count(dep); }))         {"",         "language": "C++"         "         "allaguage": "C++"         "         "         return         returun         return         returun</pre>	3 4 5 6 7 8 9 10 11 12 13
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783 2784	2 3 4 5 6 7 8 9 10 11	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.             begin(), deps.end(), [this](             int dep) { return             completedTasks.count(dep); }))         {"         return         completedTasks.count(dep); }))         {"         return         completedTasks.count(dep); }))         {"",         "language": "C++"     },     } }</pre>	3 4 5 6 7 8 9 10 11 12 13 14
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783	2 3 4 5 5 6 7 8 9 100 111 12	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.             begin(), deps.end(), [this](             int dep) { return             completedTasks.count(dep); }))         {"         return         completedTasks.count(dep); }))         {"         return         completedTasks.count(dep); }))         {"",         "language": "C++"         "         "allaguage": "C++"         "         "         return         returun         return         returun</pre>	3 4 5 6 7 8 9 10 11 12 13
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2781 2782 2783 2784 2785	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.             begin(), deps.end(), [this](             int dep) { return             completedTasks.count(dep); }))         {",         "language": "C++"     },     { }</pre>	3 4 5 6 7 8 9 10 11 12 13 14
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2781 2782 2783 2784 2785 2786	2 3 4 5 5 6 7 8 9 100 111 12 12 13	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completeTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "rule_name": "error",         "         "rule_name": "error",         "         "         "</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2781 2782 2783 2784 2785	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completeTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "rule_name": "error",         "         "rule_name": "error",         "         "         "</pre>	3 4 5 6 7 8 9 10 11 12 13 14
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2781 2782 2781 2782 2783 2784 2785 2786 2787	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function     }         "untegentering": "no matching function     }     }     }     } } </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2776 2777 2778 2779 2780 2781 2782 2783 2784 2783 2784 2785 2786 2787 2788	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         "and this it all of it or it is it of it or it is it or it or it or it is it or it or it or it is it or it is it or it is it or it is it or it or it is it or it or it is it or</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2781 2782 2781 2782 2783 2784 2785 2786 2787	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         "and this it all of it or it is it of it or it is it or it or it or it is it or it or it or it is it or it is it or it is it or it is it or it or it is it or it or it is it or</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2776 2777 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2786 2787 2788 2789	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completeTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2776 2777 2778 2779 2780 2781 2782 2783 2784 2783 2784 2785 2786 2787 2788	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2)) </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2776 2777 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2786 2787 2788 2789	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [ {     "rule_name": "error",     "message": "'all_of' is not a     member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",     "position": {         "start_line": 56,         "end_line": 56     },     "severity": "error",     "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2)) </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     } }</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2781 2782 2783 2784 2783 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781 2782	2 3 4 5 5 6 7 8 9 10 11 12 12 13 14 15	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {     } } </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {     } } </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781 2782 2783	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {         "start_line": 1673,         "settime": 1673,         "         "settime": 1673,         "settime: 1673,         "settime": 1673,         "se</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2781 2782 2783 2784 2783 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781 2782	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {     } } </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2776 2777 2778 2780 2781 2782 2783 2784 2785 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781 2782 2783 2784 2785 2784 2785 2785 2786 2787 2788 2789 2790 2791 2792 2793 2794	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {         "start_line": 1673,         "end_line": 1673     } } </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2784 2785 2786 2787 2788 2789 2780 2791 2792 2793 2794 2795	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {         "start_line": 1673,         "end_line": 1673     },     </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 3
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2776 2777 2778 2780 2781 2782 2783 2784 2785 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781 2782 2783 2784 2785 2784 2785 2785 2786 2787 2788 2789 2790 2791 2792 2793 2794	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {         "start_line": 1673         },         "file_path": "/usr/include/c++/8/     } }</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2784 2785 2786 2787 2788 2789 2780 2781 2782 2783 2784 2785 2784 2785 2786 2787 2788 2789 2790 2791 2792 2793 2794 2795 2796	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {         "start_line": 1673         },         "file_path": "/usr/include/c++/8/     } }</pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
2764 2765 2766 2767 2768 2770 2771 2772 2773 2774 2775 2776 2777 2778 2778 2779 2780 2781 2782 2783 2784 2785 2784 2785 2784 2785 2786 2787 2788 2789 2780 2791 2792 2793 2794 2795	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	<pre>"issues": [     {         "rule_name": "error",         "message": "'all_of' is not a         member of 'std'\n if (std::         all_of(deps.begin(), deps.end         (), [this](int dep) { return         completedTasks.count(dep); }))         {\n ^~~~~~",         "position": {             "start_line": 56,             "end_line": 56         },         "severity": "error",         "content": " if (std::all_of(deps.         begin(), deps.end(), [this](         int dep) { return         completedTasks.count(dep); }))         {",         "language": "C++"     },     {         "rule_name": "error",         "message": "no matching function         for call to 'Task::Task()'\n         second(std::forward&lt;_Args2&gt;(         std::get&lt;_Indexes2&gt;(tuple2))        )\n ^",     "position": {         "start_line": 1673,         "end_line": 1673     },     </pre>	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 3

"language": "C++"	2799
}	2800
],	2801
"status": "fail"	2802
	2803
	} ],

# B.3 Java

Prom	pt

	Destan a Taur alars tealed with	2806
1	Design a Java class tasked with executing and managing multiple	2800
	tasks concurrently. Leverage the	2808 2809
	Builder pattern to configure the	2809
	system, ensuring safe threading and	
	efficient resource use. Support for	2811
	executing SQL queries on specified	2812
	databases should be included. The	2813
	class must handle dependencies	2814
	through priority levels, include	2815
	comprehensive exception handling,	2816
	and provide features for scheduling	2817
	and canceling tasks. Additionally,	2818
	define metrics to assess performance	2819
	and consider the need for	2820
	distributed task execution to	2821
	address dynamic management	2822
	challenges in real-world scenarios.	2823
	Desmanas	0004
	Response	2824
1	Creating a Java class that executes and	2825
	manages multiple tasks concurrently	2826
	using the builder pattern involves a	2827
	structured approach to design not	2828
	only for functionality but also for	2829
	extensibility and maintainability.	2830
	Here's a conceptual design of the	2831
	class, including the necessary	2832
	details such as exception handling,	2833
	task dependencies, scheduling, and	2834
	metrics for performance assessment.	2835
2	``Java	2836
	<pre>import java.sql.Connection;</pre>	2837
	<pre>import java.sql.DriverManager;</pre>	2838
	<pre>import java.sql.ResultSet;</pre>	2839
	<pre>import java.sql.Statement;</pre>	2840
7	<pre>import java.sql.SQLException;</pre>	2841
8	<pre>import java.util.ArrayList;</pre>	2842
	<pre>import java.util.List;</pre>	2843
	<pre>import java.util.PriorityQueue;</pre>	2844
11	<pre>import java.util.concurrent.Callable;</pre>	2845
12	<pre>import java.util.concurrent.Executors;</pre>	2846
	import java.util.concurrent.	2847
15	ScheduledExecutorService;	2848
14	<pre>import java.util.concurrent.Future;</pre>	2849
15	<pre>import java.util.concurrent.TimeUnit;</pre>	2850
16		2851
17	// Define a Task class	2852
18	<pre>class Task implements Comparable<task> {</task></pre>	2853
19	private final int priority;	2854
20	private final Callable callable;	2855
20	private final String description;	2856
21	private rinar string description,	2857
22	public Task(int priority, Callable	2858
23	<pre></pre>	2859
	) {	2860
24	this.priority = priority;	2861
24 25	this.callable = callable;	2862
	this.description = description;	2863
26	this.description - description;	2003

	_		
2864	27 }		Executors.
2865	28		newScheduledThreadPool(
2866	<pre>29 public int getPriority() {</pre>		numThreads);
2867	30 return priority;	78	return new TaskManager(
2868	31 }		executorService,
2869	32		taskQueue);
2870	<pre>33 public Callable<?> getCallable() {</pre>	79	}
2871	34 return callable;	80	}
2872	35 }		J
_	-	81	public usid everyta() throws
2873	36	82	<pre>public void execute() throws</pre>
2874	<pre>37 public String getDescription() {</pre>		<pre>InterruptedException {</pre>
2875	38 return description;	83	while (!taskQueue.isEmpty()) {
2876	39 }	84	Task task = taskQueue.poll()
2877	40		; // get highest
2878	41 @Override		priority
2879	42 public int compareTo(Task o) {	85	Future future =
2880	43 return Integer.compare(o.		executorService.submit
2881	getPriority(), this.priority		(() -> {
2882	);	86	try {
2883			return task.
	44 }	87	
2884	45 }		getCallable().
2885	46		call();
2886	47 // Task manager class	88	<pre>} catch (Exception e) {</pre>
2887	48 public class TaskManager {	89	<pre>exceptions.add(e);</pre>
2888	49 private final	90	// Log or handle
2889	ScheduledExecutorService		exception
2890	executorService;	91	}
2891	50 private final PriorityQueue <task></task>	92	});
2892	taskQueue;	93	<pre>activeTasks.add(future);</pre>
2893			
	-	94	}
2894	activeTasks;	95	<pre>executorService.shutdown();</pre>
2895	52 private final List <exception></exception>	96	executorService.awaitTermination
2896	exceptions;		(Long.MAX_VALUE, TimeUnit.
2897	53		NANOSECONDS);
2898	54 // Private constructor	97	}
2899	55 private TaskManager(	98	
2900	ScheduledExecutorService	99	<pre>public void scheduleTask(Task task,</pre>
2901	executorService, PriorityQueue<		long delay, TimeUnit timeUnit) {
2902	Task> taskQueue) {	100	executorService.schedule(() -> {
2903		101	try {
2904	executorService;	102	<pre>task.getCallable().call</pre>
2905	57 this.taskQueue = taskQueue;		();
2906	58 this.activeTasks = new ArrayList	103	<pre>} catch (Exception e) {</pre>
2907	<>();	104	<pre>exceptions.add(e);</pre>
2908	59 this.exceptions = new ArrayList	105	}
2909	<>();	106	<pre>}, delay, timeUnit);</pre>
2910	60 }	107	}
2911	61	107	-
2912	62 public static class Builder {	108	<pre>public void cancelAllTasks() {</pre>
2912			for (Future task :
		110	•
2914	64 private final PriorityQueue <task< td=""><td></td><td>activeTasks) {</td></task<>		activeTasks) {
2915	> taskQueue = new	111	<pre>task.cancel(true);</pre>
2916	<pre>PriorityQueue&lt;&gt;();</pre>	112	}
2917	65	113	}
2918	66 public Builder setNumThreads(int	114	
2919	numThreads) {	115	<pre>public List<exception> getExceptions</exception></pre>
2920	67 this.numThreads = numThreads		() {
2921	· · · ·	116	return exceptions;
	, naturn thia.		
2922	68 return this;	117	}
2923	69 }	118	
2924	70	119	// SQL Task Support
2925	71 public Builder addTask(Task task	120	public static Callable <resultset></resultset>
2926	) {		<pre>createSQLTask(String query,</pre>
2927	<pre>72 taskQueue.add(task);</pre>		String url, String user, String
2928	73 return this;		password) {
2929	74 }	121	return () -> {
2930	74 <b>J</b> 75	121	try (Connection connection =
2930		1 22	DriverManager.
	76 public TaskManager build() {		0
2932	77 ScheduledExecutorService		<pre>getConnection(url, user,</pre>
2933	executorService =		password);

```
3004
 Statement statement =
 using message brokers like
3005
 connection.
 Kafka, RabbitMQ).
 - Database for task tracking,
3006
 createStatement())
 11
3007
 retries, and fault tolerance.
 {
3008
 11
 - Task execution engines that
 124
 return statement.
 176
3009
 can scale beyond what is
 executeQuery(query);
 } catch (SQLException e) {
 available in one JVM,
3010
 125
 potentially leveraging cloud
3011
 126
 throw new
 177
 11
3012
 services like AWS Lambda for
 RuntimeException("
3013
 SQL Task Failed", e)
 serverless execution.
3014
 ;
 178 }
 - - - -
3015
 }
 179
 127
3016
 };
 128
 180
3017
 129
 }
 181 ### Key Points:
3018
 130
 182
3019
 183 1. **Concurrency Management**: Uses `
 // Metrics to assess performance
 131
 public static class Metrics {
 ScheduledExecutorService` to handle
3020
 132
3021
 private final long startTime;
 concurrent task execution and
3022
 134
 private long endTime;
 scheduling.
3023
 private int totalTasks;
 135
 184
3024
 private int completedTasks;
 185 2. **Task Queuing with Priority**:
 136
3025
 Utilizes a `PriorityQueue` to manage
 137
 tasks based on priority levels.
3026
 public Metrics() {
 138
3027
 this.startTime = System.
 139
 186
3028
 currentTimeMillis();
 187 3. **Builder Pattern**: Allows flexible
3029
 configuration of `TaskManager` with
 140
 }
3030
 a specified number of threads and
 141
3031
 142
 public void taskCompleted() {
 tasks
3032
 143
 completedTasks++;
 188
3033
 189 4. **SOL Execution**: Implements a
 144
 }
3034
 method to create SQL tasks with a
 145
3035
 public void setTotalTasks(int
 specified query and database
 146
3036
 totalTasks) {
 connection details.
3037
 this.totalTasks = totalTasks 190
 147
3038
 191 5. **Exception Handling**: Robust
 ;
3039
 approach to capturing and managing
 148
 }
3040
 149
 exceptions during task execution.
 public void end() {
3041
 150
 192
 this.endTime = System.
 193 6. **Performance Metrics**: Provides a
3042
 151
3043
 currentTimeMillis();
 mechanism to measure execution time,
3044
 completion rate, and count of
 152
 }
3045
 153
 completed tasks.
3046
 154
 public long getTotalTime() {
 194
3047
 return endTime - startTime;
 195 7. **Task Scheduling and Cancellation**:
 155
3048
 Offers functionalities to schedule
 }
 156
3049
 157
 tasks with delays and cancel running
3050
 public int getCompletedTasks() {
 158
 tasks.
3051
 159
 return completedTasks;
 196
3052
 160
 3
 197 8. **Distributed Execution
3053
 Considerations**: While the current
 161
 design is for a single JVM, it
3054
 public int getTotalTasks() {
 162
3055
 return totalTasks:
 mentions the necessities for scaling
 163
3056
 }
 towards distributed execution
 164
3057
 environments.
 165
3058
 public double getCompletionRate
 166
 Static analysis result
3059
 () {
 167
 return (double)
 1 {
3061
 completedTasks /
 "issues": [
 2
3062
 totalTasks * 100;
 3
 {
3063
 168
 }
 "rule_name": "error",
 4
3064
 169
 }
 "message": "no suitable method
 5
3065
 170
 found for submit(()->{ try
3066
 // Considerations for Distributed
 [...]; })\n Future<?> future
3067
 Task Execution:
 = executorService.submit(() ->
3068
 // To expand this design to a
 {\n ^\n method ExecutorService
3069
 distributed system, additional
 .<T#1>submit(Callable<T#1>) is
3070
 components and mechanisms should
 not applicable\n (cannot infer
 be
 type-variable(s) T#1\n (
 // integrated, like:
3072
 argument mismatch; bad return
 // - Distributed Task Queues (e.g.,
3073
 174
 type in lambda expression\n
```

```
3143
 missing return value))\n method
 10
3144
 ExecutorService.<T#2>submit(
3145
 Runnable, T#2) is not applicable
 12
3146
 \n (cannot infer type-variable(
 s) T#2\n (actual and formal
3147
3148
 argument lists differ in length
 13
3149
))\n method ExecutorService.
 14
3150
 submit(Runnable) is not
3151
 applicable\n (argument mismatch
 15
3152
 ; bad return type in lambda
 16
3153
 expression\n unexpected return
 17
3154
 value)\n where T#1,T#2 are type
 18
3155
 -variables:\n T#1 extends
3156
 Object declared in method <T#1>
 19
3157
 submit(Callable<T#1>)\n T#2
 20
3158
 extends Object declared in
 21
 method <T#2>submit(Runnable,T
3159
 22
3160
 #2)\n2 errors",
 23
 "position": {
3161
 6
 24
 "start_line": 86,
3162
 7
 "end_line": 86
3163
 8
3164
 9
 },
 25
 "severity": "error",
"content": " Future<?> future =
3165
 10
 26
3166
3167
 executorService.submit(() ->
3168
 {",
 28
 "language": "Java"
3169
3170
 13
 }
],
3171
 14
 "status": "fail"
3172
 15
 29
3173
 16 }
```

```
B.4 JavaScript
```

#### Prompt

3176	When enhancing the `isCompletelyDefined`
3177	function, ensure it evaluates
3178	nested functions within objects.
3179	Special attention should be given to
3180	recursive structures. Return a
3181	boolean true when the functional
3182	properties are validated properly.

## Response

```
Enhancing the `isCompletelyDefined`
3184
3185
 function to evaluate nested
3186
 functions within objects can be done
3187
 by recursively traversing the
3188
 object, checking each property for
 undefined values, and ensuring that
3189
3190
 all functions are valid and properly
 defined. Special attention to
3191
3192
 recursive structures necessitates
3193
 keeping track of objects we've
3194
 already encountered to prevent
3195
 infinite loops.
3196
 2
3197
 <sup>3</sup> Here is a potential implementation in
3198
 JavaScript:
3199
 4
 5 ```javascript
3201
 6 function isCompletelyDefined(obj,
3202
 visited = new WeakSet()) {
 // Check for null or non-object
 7
3204
 values
3205
 if (obj === null || typeof obj !== '
 8
3206
 object') {
3207
 return true;
 9
```

11	5	3209
	// llos a Wask Cat to track wisited	
12	// Use a WeakSet to track visited	3210
	objects to handle recursive	3211
	structures	3212
13	if (visited.has(obj)) {	3213
14	return true; // Avoid infinite	3214
14	- ,	
	loops	3215
15	}	3216
16	visited.add(obj);	3217
17		3218
	// Itarata over all properties of	3219
18	<pre>// Iterate over all properties of</pre>	
	the object	3220
19	for (let key in obj) {	3221
20	if (obj.hasOwnProperty(key)) {	3222
21	<pre>const value = obj[key];</pre>	3223
	const varue = obj[key],	
22		3224
23	if (value === undefined) {	3225
24	<pre>// If a property is</pre>	3226
	explicitly undefined	3227
	, return false	3228
25	return false;	3229
26	}	3230
27	else if (typeof value === '	3231
_ /	function') {	3232
28	// If the property is a	3233
	function, assume it'	3234
	s valid if it's	3235
	defined	3236
20		3237
29	// You may add more	
	complex checks here	3238
	if necessary	3239
30	try {	3240
31	value();	3241
32	} catch (e) {	3242
33	return false; // If	3243
	function	3244
	execution	3245
	results in error	3246
	, consider it	3247
	invalid	3248
34	}	3249
35	}	3250
	else if (typeof value === '	3251
36		
	object') {	3252
37	<pre>// For nested objects,</pre>	3253
	recursively call	3254
	isCompletelyDefined	3255
38	if (!isCompletelyDefined	3256
	(value, visited)) {	3257
39	return false;	3258
40	}	3259
	}	3260
41	-	
42	}	3261
43	}	3262
44		3263
45	return true;	3264
46	}	3265
47		3266
48		3267
49	### Key Considerations:	3268
	- **Recursive Traversal**: The function	3269
50		
	recursively checks each property of	3270
	the object. If a property is an	3271
	object, it ensures those properties	3272
	are also completely defined.	3273
51		
51	- **Handling of Functions**: It's	3274
	assumed that as long as a function	3275
	can be called without causing errors	3276
	, it's properly defined. If a	3277
	, pp p p p p p	

}

3278	function property needs more complex
3279	validation, you can update the
3280	logic inside the function check.
	logic inside the function check.
3281	52 - **WeakSet for Visited Objects**: To
3282	handle recursive structures safely,
3283	the function uses a `WeakSet` to
3284	keep track of visited objects,
3285	preventing an infinite loop on
3286	recursive references, such as
3287	circular linked lists or structures.
3288	53 - **Error Handling**: Functions are
3289	simply called to check if they can
3290	execute without error, providing a
3291	basic form of validation. Depending
3292	on your context, you may want to
3293	further enhance this part.
3293	
3295	55 This implementation ensures the
3296	structure is evaluated in-depth,
3297	focuses on comprehensive validation,
3298	and maintains efficiency for large
3299	objects.
	5
3300	Static analysis result
3300	Static analysis result
3300 3301	·
3301	1 {
3301 3302	1 { 2 "issues": [
3301 3302 3303	1 { 2 "issues": [ 3 {
3301 3302 3303 3304	1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-
3301 3302 3303 3304 3305	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins",</pre>
3301 3302 3303 3304 3305 3306	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-</pre>
3301 3302 3303 3304 3305	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-</pre>
3301 3302 3303 3304 3305 3306	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-</pre>
3301 3302 3303 3304 3305 3306 3307	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.",</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { </pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { 7 "start_line": 20,</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20 9 },</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20 9 }, 10 "severity": "error", </pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20 9 }, 10 "severity": "error", 11 "content": "if (obj.</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype- builtins", 5 "message": "Do not access Object. prototype method ' hasOwnProperty' from target object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20 9 }, 10 "severity": "error", 11 "content": "if (obj.</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20, 9 }, 10 "severity": "error", 11 "content": "if (obj.         hasOwnProperty(key)) {", </pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20 9 }, 10 "severity": "error", 11 "content": " if (obj.         hasOwnProperty(key)) {", 12 "language": "JavaScript"</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317 3318	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20, 9 }, 10 "severity": "error", 11 "content": " if (obj.         hasOwnProperty(key)) {", 12 "language": "JavaScript" 13 } </pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317 3318 3319	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20, 9 }, 10 "severity": "error", 11 "content": " if (obj.         hasOwnProperty(key)) {", 12 "language": "JavaScript" 13 } 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317 3318 3319 3320	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20, 8 "end_line": 20, 9 }, 10 "severity": "error", 11 "content": " if (obj.         hasOwnProperty(key)) {", 12 "language": "JavaScript" 13 } 14 ], 15 "status": "fail"</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317 3318 3319	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20, 9 }, 10 "severity": "error", 11 "content": " if (obj.         hasOwnProperty(key)) {", 12 "language": "JavaScript" 13 } 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14 ], 14</pre>
3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317 3318 3319 3320	<pre>1 { 2 "issues": [ 3 { 4 "rule_name": "no-prototype-builtins", 5 "message": "Do not access Object.         prototype method '         hasOwnProperty' from target         object.", 6 "position": { 7 "start_line": 20, 8 "end_line": 20, 8 "end_line": 20, 9 }, 10 "severity": "error", 11 "content": " if (obj.         hasOwnProperty(key)) {", 12 "language": "JavaScript" 13 } 14 ], 15 "status": "fail"</pre>

B.5 SQL

```
Prompt
```

1 Design a comprehensive system for topic management and display in a community forum. Implement functionalities to retrieve user-created topics using efficient database management techniques. Join necessary tables to compile complex data, such as user profiles, while applying context-aware filters, like temporal scopes or regional differences. Present the resulting data as a list of dictionaries, each featuring 'title', 'creator_name', 'creation_date', and optionally ' region'. Ensure dynamic pagination to manage extensive topic volumes efficiently, prioritizing descending order by creation date and evaluate resource usage implications.

### Response

1	Designing a comprehensive system for	3344
	topic management and display in a	3345
	community forum involves a	3346
	combination of database design,	3347
	efficient query construction, and	3348
	dynamic data presentation. Here's a	3349
	proposed design and implementation	3350
	strategy:	3351
2		3352
	### **System Overview**	3353
4		3354
	1. **Database Design:**	3355
6	- **Tables:**	3356
7		3357
	information.	3358
8	- Columns: `user_id`, `name`, `	3359
	email`, `region`	3360
9	- **Topics:** Stores topics created	3361
	by users.	3362
10		3363
	`title`, `creation_date`, `	3364
	content`	3365
11	- **Regions:** (Optional) Stores	3366
	information about regions.	3367
12		3368
	region_name`	3369
13		3370
14	<pre>2. **Relations:**</pre>	3371
15	- A **User** can create multiple **	3372
	Topics** (one-to-many	3373
	relationship).	3374
16		3375
10	** (optional, many-to-one	3376
	relationship).	3377
17		3378
17 18	### **Functionality Implementation**	3378 3379
18	### **Functionality Implementation**	3379
18 19		3379 3380
18 19 20	### **Functionality Implementation** #### **Data Retrieval:**	3379 3380 3381
18 19 20 21	#### **Data Retrieval:**	3379 3380 3381 3382
18 19 20 21 22	#### **Data Retrieval:** 1. **Database Connection:**	3379 3380 3381 3382 3383
18 19 20 21	#### **Data Retrieval:** 1. **Database Connection:** - Use a robust database management	3379 3380 3381 3382 3383 3383 3384
18 19 20 21 22 23	<pre>#### **Data Retrieval:** 1. **Database Connection:**    - Use a robust database management       system like PostgreSQL or MySQL.</pre>	3379 3380 3381 3382 3383 3384 3384 3385
18 19 20 21 22	<pre>#### **Data Retrieval:** 1. **Database Connection:**    - Use a robust database management       system like PostgreSQL or MySQL.    - Implement connection pooling to</pre>	3379 3380 3381 3382 3383 3384 3385 3386
18 19 20 21 22 23	<pre>#### **Data Retrieval:** 1. **Database Connection:**    - Use a robust database management       system like PostgreSQL or MySQL.    - Implement connection pooling to       manage multiple concurrent</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3386 3387
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**    - Use a robust database management       system like PostgreSQL or MySQL.    - Implement connection pooling to</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3386 3387 3388
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**    - Use a robust database management       system like PostgreSQL or MySQL.    - Implement connection pooling to       manage multiple concurrent       accesses efficiently.</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3386 3387 3388 3389
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3389 3390
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3390 3391
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3391 3392
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ``sql </pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ``sql     SELECT</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ``sql </pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ``sql     SELECT</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3394 3395
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396 3396 3397
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396 3396 3397 3398
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396 3396 3397 3398 3399
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396 3395 3396 3397 3398 3399 3400
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t </pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396 3395 3396 3397 3398 3399 3400 3401
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t     JOIN </pre>	3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3394 3395 3396 3397 3398 3399 3400 3401 3402
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> <li>38</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**  - Use a robust database management     system like PostgreSQL or MySQL.  - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**  - Utilize SQL to retrieve topic data     with necessary joins and filters.  - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM         Topics t     JOIN         Users u ON t.user_id = u.user_id </pre>	3379 3380 3381 3382 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3394 3395 3396 3397 3398 3399 3400 3401 3402 3403
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> <li>38</li> <li>39</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t     JOIN     Users u ON t.user_id = u.user_id     ORDER BY</pre>	3379 3380 3381 3382 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3394 3395 3396 3397 3398 3399 3400 3401 3402 3403
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> <li>38</li> <li>39</li> <li>40</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t     JOIN     Users u ON t.user_id = u.user_id     ORDER BY     t.creation_date DESC </pre>	3379 3380 3381 3382 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3394 3395 3396 3397 3398 3399 3400 3401 3402 3403 3404
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> <li>38</li> <li>39</li> <li>40</li> <li>41</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t     JOIN     Users u ON t.user_id = u.user_id     ORDER BY     t.creation_date DESC     LIMIT</pre>	3379 3380 3381 3382 3383 3384 3385 3386 3386 3387 3390 3391 3392 3393 3394 3395 3396 3396 3396 3396 3396 3397 3398 3399 3400 3401 3402 3403
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> <li>38</li> <li>39</li> <li>40</li> <li>41</li> <li>42</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**     Use a robust database management     system like PostgreSQL or MySQL.     Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**     Utilize SQL to retrieve topic data     with necessary joins and filters.     Example Query:     ``sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t     JOIN     Users u ON t.user_id = u.user_id     ORDER BY     t.creation_date DESC     LIMIT     %s OFFSET %s; } </pre>	3379 3380 3381 3382 3383 3384 3385 3386 3386 3387 3390 3391 3392 3393 3394 3395 3396 3396 3396 3396 3397 3397 3398 3399 3400 3401 3402 3403 3404 3405 3406 3406 3407 3408
<ol> <li>18</li> <li>19</li> <li>20</li> <li>21</li> <li>22</li> <li>23</li> <li>24</li> <li>25</li> <li>26</li> <li>27</li> <li>28</li> <li>29</li> <li>30</li> <li>31</li> <li>32</li> <li>33</li> <li>34</li> <li>35</li> <li>36</li> <li>37</li> <li>38</li> <li>39</li> <li>40</li> <li>41</li> <li>42</li> <li>43</li> </ol>	<pre>#### **Data Retrieval:** 1. **Database Connection:**   - Use a robust database management     system like PostgreSQL or MySQL.   - Implement connection pooling to     manage multiple concurrent     accesses efficiently. 2. **Query Construction:**   - Utilize SQL to retrieve topic data     with necessary joins and filters.   - Example Query:     ```sql     SELECT     t.title,     u.name as creator_name,     t.creation_date,     u.region     FROM     Topics t     JOIN     Users u ON t.user_id = u.user_id     ORDER BY     t.creation_date DESC     LIMIT</pre>	3379 3380 3381 3382 3384 3385 3386 3386 3387 3388 3390 3391 3392 3393 3394 3395 3396 3396 3396 3396 3395 3397 3397 3397 3398 3399 3400 3401 3402 3403 3404 3405 3406 3406 3407

```
3412
 46 3. **Context-Aware Filtering:**

 Temporal Scope: Add a date
range condition (`WHERE t.

3413
 47
 93
3414
3415
 creation_date BETWEEN %s AND %s`)
 94
3416
3417
 - **Regional Differences:**
 48
 96
3418
 - Filter by region if specified (`
 49
 WHERE u.region = %s`).
3419
3420
 50
 97
3421
 51 4. **Dynamic Pagination:**
3422
 52
 - Implement pagination by calculating
 98
 `LIMIT` and `OFFSET` based on
3423
3424
 page number and page size.
3425
 - Provide API endpoints with
 53
 parameters for `page` and `
3426
3427
 page_size`.
3428
 54
 55 ### **Response Structure:**
3429
3430
 56
 57 - **Data Format:**
3431
3432
 - Return data as a list of
 58
3433
 dictionaries.
 4
3434
 59
 - Example:
 5
3435
 `python
 60
3436
 Ε
 61
3437
 62
 {
 "title": "How to Learn Python",
3438
 63
 "creation_date": "Alice",
"creation_date": "2023-10-01",
3439
 64
 8
3440
 65
 0
 "region": "North America" #
3441
 66
 10
3442
 Optional
 11
3443
 },
 67
3444
 68
 {
 12
 "title": "Javascript Tips",
3445
 69
 13
 "creator_name": "Bob",
3446
 70
 "creation_date": "2023-10-02"
3447
 71
3448
 72
 "region": "Europe" # Optional
3449
 73
 }
3450
 74
]
3451
 75
3452
 76
3453
 77 ### **Efficiency and Resource Management
3454
 :**
3455
 78
3456
 79 1. **Indexing:**
3457
 80
 - Index `creation_date` for faster
 order operations.
3458
 - Index foreign keys (`user_id`) for
3459
 81
3460
 quicker join operations.
3461
 82
 83 2. **Connection Pooling:**
3463
 - Use a connection pool library to
 84
3464
 manage database connections
3465
 efficiently.
3466
 - Libraries like SQLAlchemy (Python)
 85
 provide robust connection pooling
3467
 options.
3469
 86
3470
 87 3. **Caching Strategy:**
 - Implement caching for repetitive
3471
 88
3472
 queries using tools like Redis or
3473
 Memcached.
3474
 - Consider caching popular queries (e
 89
3475
 .g., when paginating through the
3476
 latest topics).
3477
 90
 91 4. **Load Testing:**
3478
3479
 92
 - Conduct load testing to ensure that
3480
 the system can handle extensive
3481
 volumes without performance
```

```
degradation.
 3482
 - Tools like JMeter or Locust can
 3483
 simulate concurrent users.
 3484
 3485
 3486
95 5. **Monitoring:**
 - Monitor database performance using
 3487
 built-in tools or third-party
 3488
 solutions.
 3489
 3490
 - Analyze query execution times and
 plan optimizations accordingly.
 3491
 3492
99 This design balances efficient data
 3493
 retrieval, context-aware filtering,
 and effective resource management,
 3495
 ensuring a scalable and responsive
 3496
 community forum system.
 3497
 Static analysis result
 3498
1 {
? "issues": [
 3499
 3500
 3501
3
 {
 "rule_name": "PRS:",
 3502
 "message": "Line 12, Position 1:
Found unparsable section: '
 3503
 LIMIT \\n %s OFFSET %s'",
 3505
 "position": {
 3506
6
 "start_line": 41,
 3507
7
 "end_line": 42
 3508
 3509
 },
 "severity": "error",
"content": "LIMIT \n %s OFFSET %s
 3510
 3511
 ;",
 3512
 "language": "SQL"
 3513
 3514
 }
14],
 3515
15 "status": "fail"
 3516
16 }
 3517
```