Measuring Coding Challenge Competence With APPS — Supplementary Material

Dan Hendrycks* UC Berkeley
Steven Basart* UChicago
Saurav Kadavath UC Berkeley
Mantas Mazeika UIUC
Akul Arora UC Berkeley
Ethan Guo UC Berkeley
Collin Burns UC Berkeley
Samir Puranik UC Berkeley
Horace He Cornell
Dawn Song UC Berkeley
Jacob Steinhardt UC Berkeley

	Hearthstone	Django	NAPS	APPS
Programming Language	Python	Python	UAST	Python
Test Cases	✗	✗	✓	✓
Number of Programs	665	18,805	17,477	232,421
Lines per Program (Avg.)	7.7	1	21.7	18.0
Number of Exercises	665	18,805	2,231	10,000
Text Input	Card Text	Comment	Pseudocode	Problem Descriptions

Table 1: Further comparisons of APPS with previous datasets.

	Top-5 Test Case Average				Top-5 Strict Accuracy			
Model	Introductory	Interview	Competitive	Average	Introductory	Interview	Competition	Average
GPT-2 0.1B	13.81	10.97	7.03	10.75	2.70	0.73	0.00	1.02
GPT-2 1.5B	16.86	13.84	9.01	13.48	3.60	1.03	0.00	1.34
GPT-Neo 2.7B	19.89	13.19	9.90	13.87	5.50	0.80	0.00	1.58

Table 2: Top-5 performance of GPT-2 models and GPT-Neo. Taking the best of five candidate solutions markedly improves performance.

A Checklist Information

Legal Compliance. In APPS, we scrape question text, ground-truth solutions, and test cases from various coding challenge websites. These websites are AtCoder, CodeChef, Codeforces, Codewars, HackerRank, Kattis, and LeetCode. In all cases, we only scrape public-facing data. For instance, we avoid scraping data from paywalled portions of sites. In the case of Kattis, all problems we scrape are under the CC BY-SA 3.0 license (https://creativecommons.org/licenses/by-sa/3.0/). For other websites, some content may be copyrighted. In these cases, we abide by Fair Use §107: "the fair use of a copyrighted work, including such use by ... scholarship, or research, is not an infringement of copyright", where fair use is determined by "the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes", "the amount and substantiality of the portion used in relation to the copyrighted work as a whole", and "the effect of the use upon the potential market for or value of the copyrighted work." The APPS dataset is

*Equal Contribution.

Submitted to the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks. Do not distribute.

noncommercial and is likely to have no effect on the value of the original problems. Moreover, for all problem sources, we only scrape a fraction of the available problems and ground-truth solutions.

Regarding international copyright laws, the websites that we scrape from are based in the United States, Japan, India, and Russia, all of which are contracting parties to the WIPO Copyright Treaty. In the United States, the WIPO Copyright Treaty is implemented by the Digital Millenium Copyright Act (DMCA). Since APPS was made in the United States, the DMCA is the relevant legislation that we must comply with. Notably, DMCA §1201 states, "No person shall circumvent a technological measure that effectively controls access to a work protected under this title." We do not circumvent access controls when creating APPS and hence abide by §1201. Fair Use extends to content protected by the DMCA, for which we refer readers to the previous paragraph.

Although GDPR only applies in the European Union, some of the ground-truth solutions in APPS may have been written by EU citizens. GDPR is chiefly concerned with the protection of personal data gathered by entities engaging in economic activity. The only personally linked information in APPS is the problem solutions written by individuals and published under aliases to public websites. In some cases, these solutions contain identifying information in comments, which we remove to preserve privacy. We comply with GDPR, because our processed solutions remove identifiers, and we are compliant because we collect the data for academic research purposes.

Author Statement and License. We bear all responsibility in case of violation of rights. The APPS data is licensed under CC BY-SA 3.0 in accordance with the Kattis problem licenses and the ShareAlike terms. Our code is open sourced under the MIT license.

B Datasheets

We follow the recommendations of Gebru et al. (2018) and provide a datasheet for the ETHICS dataset in this section.

B.1 Motivation

For what purpose was the dataset created? Was there a specific task in mind? Was there a specific gap that needed to be filled? Please provide a description. The APPS dataset was created to track the progress of code generation models on the task of generating arbitrary Python code from complex natural language specifications, a challenging setting that had no rigorous benchmark before our work.

Who created the dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)? Refer to the main document.

Who funded the creation of the dataset? If there is an associated grant, please provide the name of the grantor and the grant name and number. There is no associated grant.

Any other comments? No.

B.2 Composition

What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)? Are there multiple types of instances (e.g., movies, users, and ratings; people and interactions between them; nodes and edges)? Please provide a description. The instances are coding challenge problems posed in natural language, each of which consists of question text, ground-truth solutions, and test cases. Please refer to the main document for more detail.

How many instances are there in total (of each type, if appropriate)? APPS contains 10,000 problems, 232,421 ground-truth solutions, and 131,777 test cases.

Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how

this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable). APPS contains a subset of all possible test cases for its problems. These test cases are written by problem designers to cover important functionality.

What data does each instance consist of? "Raw" data (e.g., unprocessed text or images) or features? In either case, please provide a description. Each instance consists of text and numerical data.

Is there a label or target associated with each instance? If so, please provide a description. Each instance is associated with test cases, which provide a ground-truth signal for functional correctness.

Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text. No.

Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit. We remove duplicate or near-duplicate problems from APPS.

Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them. We provide a training and test split. The splits were optimized for increasing the number of test cases in the test split while maintaining a fixed number of problems from each difficulty.

Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description. See Section 3 in the main paper for a discussion of test case quality.

Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? The dataset is self-contained.

Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals' non-public communications)? If so, please provide a description. No.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why. Unknown.

Does the dataset relate to people? If not, you may skip the remaining questions in this section. Yes.

Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations are identified and provide a description of their respective distributions within the dataset. No.

Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how No.

Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals racial or ethnic origins, sexual orientations, religious beliefs, political opinions or union memberships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description. No.

Any other comments? No.

B.3 Collection Process

How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how. All data was collected by scraping problems from coding challenge websites, such as Codewars, AtCoder and Kattis.

What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated? We used off-the-shelf and custom-built scrapers. We manually checked whether scraped data matched text on the websites.

If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)? Some problems we scraped were left out of APPS for various reasons, e.g. they required images to solve, they lacked ground-truth solutions and test cases, or they were duplicate problems.

Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)? All data was collected by undergraduate and graduate student authors on the paper.

Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created. Data was collected from late 2020 to early 2021 and refined for six months.

Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation No.

Does the dataset relate to people? If not, you may skip the remainder of the questions in this section. Yes.

Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)? We scraped data via websites where individuals had publicly posted problem solutions.

Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or otherwise reproduce, the exact language of the notification itself. Users who posted on the Internet were not notified of our collection, because their examples were posted publicly.

Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented. N/A

If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate). N/A

Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis) been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation. No.

Any other comments? No.

B.4 Preprocessing/Cleaning/Labeling

Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section. Yes, as described in Section 3 of the main paper.

Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data. No.

Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point. Not at this time.

Any other comments? No.

B.5 Uses

Has the dataset been used for any tasks already? If so, please provide a description. Yes, see the main paper.

Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point. No.

What (other) tasks could the dataset be used for? N/A

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? For example, is there anything that a future user might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g., stereotyping, quality of service issues) or other undesirable harms (e.g., financial harms, legal risks) If so, please provide a description. Is there anything a future user could do to mitigate these undesirable harms? We describe how our data collection is legally compliant in Appendix A.

Are there tasks for which the dataset should not be used? If so, please provide a description. N/A

Any other comments? No.

B.6 Distribution

Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created? If so, please provide a description. Yes, the dataset will be publicly distributed.

How will the dataset will be distributed (e.g., tarball on website, API, GitHub)? Does the dataset have a digital object identifier (DOI)? The dataset is available at https://github.com/hendrycks/apps.

When will the dataset be distributed? The dataset is currently available.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions. The code for our experimental framework is distributed under an MIT license. Where applicable,

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions. In cases where websites that we scrape data from have copyright policies, we abide by Fair Use according to §107, and we comply with GDPR even though all our problem sources with ground-truth solutions are based in the US. See Appendix A for details.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any supporting documentation. No.

Any other comments? No.

B.7 Maintenance

Who is supporting/hosting/maintaining the dataset? Refer to the main document.

How can the owner/curator/manager of the dataset be contacted (e.g., email address)? Refer to the main document.

Is there an erratum? If so, please provide a link or other access point. Not at this time.

Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to users (e.g., mailing list, GitHub)? We plan to update the dataset with an additional JSON of test cases present in the question text for each problem. This will be available through GitHub.

If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were individuals in question told that their data would be retained for a fixed period of time and then deleted)? If so, please describe these limits and explain how they will be enforced No.

Will older versions of the dataset continue to be supported/hosted/maintained? If so, please describe how. If not, please describe how its obsolescence will be communicated to users. N/A

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to other users? If so, please provide a description. Our dataset could be extended with additional problems that follow the formatting of existing problems.

Any other comments? No.

C Additional Dataset Information

Expanded Dataset Comparisons. We compared to several datasets in the (Kulal et al., 2019; Yu et al., 2018; Raychev et al., 2016; Iyer et al., 2018; Lu et al., 2021) main paper. We continue the comparisons below. Ling et al. (2016) introduce datasets based on Hearthstone and Magic the Gathering card games for code generation. Oda et al. (2015) provide a language-to-code dataset using simple code comments. Zavershynskyi et al. (2018) introduce the NAPS dataset for converting pseudocode to code, obtained by crowdsourcing low-level descriptions of programming exercises, and apply machine translation techniques to the problem. Recent anecdotal posts on social media have demonstrated that modern Transformers can in some instances generate JSX code adhering to user requests, but our work provides precision to the discussion through quantitative evaluation. Allamanis and Sutton (2013) introduce the GitHub Java Corpus used for performing language modeling on Java code. Liu et al. (2020) do a smaller-scale analysis of code generation but with their limited language-specific training data models "fail to pass even a single predefined test case" on their 300

test problems, while with our large training set and test set, trained models can pass tens of thousands of test cases. Zelle and Mooney (1996) and Tang and Mooney (2001) precedes Yu et al. (2018) by also facilitating the synthesis of database queries, though more recent program synthesis works such as Wang et al. (2019) use Spider from Yu et al. (2018).

Table 1 compares APPS to Hearthstone (Ling et al., 2016), Django (Oda et al., 2015), and Zavershynskyi et al. (2018). 'Number of Programs' refers to the number of human-written programs or functions in the dataset, and 'Number of Exercises' refers to the number of tasks that the network must solve. These numbers can differ in datasets such as APPS with multiple human-written solutions per exercise.

Excluded Keywords. In creating the GitHub pretraining dataset, we exclude the following keywords to prevent overlap with coding challenge questions similar to those in APPS: 'atcoder', 'coderbyte', 'leetcode', 'codeforces', 'codewars', 'hackerrank', 'topcoder', 'codechef', 'checkio', 'HackerEarth', 'Programmr', 'Exercism', 'Codier', 'PyBites', 'Tynker', 'CodinGame', 'CodeCombat', 'usaco', 'IOI', 'UVA', 'ICFP', 'EPIJudge', 'SPOJ', 'UVaOJ', 'judge', 'interview', 'solution', 'coding', 'code', 'problem', 'exercise', 'challenge', 'algo', 'practice', 'competitive', 'program'.

D Additional Results

Top-5 Performance. Rather than allowing models to generate just one potential solution, we let models generate five and we choose the best performing solution. Full top-5 performance results are in Table 2.

GPT-3. We evaluate GPT-3 175B on APPS in a few-shot setting. A separate prompt is used for standard input and call-based questions, and each prompt includes instruction text along with two example questions and solutions from the corresponding question type. The solutions we select for each prompt are the shortest solutions passing all test cases for the corresponding problems. In preliminary experiments, we found that the original GPT-3 175B did not perform well, and the instruct series was necessary for reasonable generations. Inference on the $5{,}000$ problems in the APPS test set is possible for approximately $500.

We find that GPT-3 only solves 3 problems out of $5{,}000$: two introductory problems and one interview problem. The two introductory problems are simple interpretation tasks, such as implementing a specified algebraic expression. The interview problem requires higher-level thinking and is show in Figure 1. Legitimately solving this problem would suggest nontrivial reasoning. However, it is possible that GPT-3 memorized the solution during pretraining, or that it took a lucky guess based on heuristics in the question.

We also find that GPT-3 tends to handle syntax poorly. In Figure 2, we see a generation with improper formatting of an if-else block, which causes a syntax error. By contrast, GPT-2 0.1B has fewer syntax errors with three orders of magnitude fewer parameters, and GPT-Neo 2.7B has almost no syntax errors. These two models are fine-tuned on GitHub and the APPS training set.

More Generated Code Examples. Additional examples of generated code can be found in Figure 3, Figure 4, and Figure 5.

References

Miltiadis Allamanis and Charles Sutton. Mining source code repositories at massive scale using language modeling. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 207–216. IEEE, 2013.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumeé III, and Kate Crawford. Datasheets for datasets. *arXiv preprint arXiv:1803.09010*, 2018.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Problem

Mahmoud and Ehab play a game called the even-odd game. Ehab chooses his favorite integer n and then they take turns, starting from Mahmoud. In each player's turn, he has to choose an integer a and subtract it from n such that: $1 \leq a \leq n$. If it's Mahmoud's turn, a has to be even, but if it's Ehab's turn, a has to be odd.

If the current player can't choose any number satisfying the conditions, he loses. Can you determine the winner if they both play optimally?

—Input—

The only line contains an integer n $(1 \leq n \leq 10^9)$, the number at the beginning of the game.

—Output—

Output "Mahmoud" (without quotes) if Mahmoud wins and "Ehab" (without quotes) otherwise.

Model Output

```
n = int(input())
if n % 2 == 0:
    print('Mahmoud')
else:
    print('Ehab')
```

Figure 1: The sole interview problem solved by GPT-3. Legitimately solving this problem requires nontrivial reasoning about the proposed game's branching possibilities.

```
a, b = int(input()), int(input())
if a == b:

print(''.join(map(str, [a, b])))

else:

print(''.join(map(str, [a, b])))
```

Figure 2: Oftentimes, GPT-3 generates outputs with incorrect syntax, such as the above code. Even though the code is reasonable, bad formatting of the if-else block causes a syntax error.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

W. Ling, P. Blunsom, Edward Grefenstette, K. Hermann, Tomás Kociský, Fumin Wang, and A. Senior. Latent predictor networks for code generation. *ArXiv*, abs/1603.06744, 2016.

Hui Liu, Mingzhu Shen, Jiaqi Zhu, Nan Niu, Ge Li, and Lu Zhang. Deep learning based program generation from requirements text: Are we there yet? *IEEE Transactions on Software Engineering*, 2020.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, A. Blanco, C. Clément, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, L. Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, N. Duan, N. Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. 2021.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation. In *International Conference on Automated Software Engineering (ASE)*, 2015.

Veselin Raychev, Pavol Bielik, and Martin T. Vechev. Probabilistic model for code with decision trees. *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2016.

L. Tang and R. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*, 2001.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*, 2019.

Problem
Given is a directed graph G with N vertices and M edges. The vertices are numbered 1 to N, and the i-th edge is directed from Vertex A_i to Vertex B_i. It is guaranteed that the graph contains no self-loops or multiple edges. Determine whether there exists an induced subgraph (see Notes) of G such that the in-degree and out-degree of every vertex are both 1. If the answer is yes, show one such subgraph. Here the null graph is not considered as a subgraph.

Model Output

```
import sys

sys.setrecursionlimit(10 ** 7)
input = sys.stdin.readline
f_inf = float('inf')
mod = 10 ** 9 + 7


def resolve():
    n, m = map(int, input().split())
    edge = [[] for _ in range(n)]
    for _ in range(m):
        a, b = map(int, input().split())
        edge[a - 1].append(b - 1)
        edge[b - 1].append(a - 1)

    def dfs(u):
        stack = [u]
        seen = [False] * n
        while stack:
            u = stack.pop()
            if seen[u]:
                continue
            seen[u] = True
            for v in edge[u]:
                if seen[v]:
                    continue
                seen[v] = True
                stack.append(v)

    res = None
    for i in range(n):
        if not seen[i]:
            seen[i] = True
            res = dfs(i)
    if not res:
        print(-1)
    else:
        print(res)


if __name__ == '__main__':
    resolve()
```

Figure 3: An example generated by GPT-2 1.5B. The model can sometimes compose function calls.

Problem
Given is a tree G with N vertices. The vertices are numbered 1 through N, and the i-th edge connects Vertex a_i and Vertex b_i. Consider painting the edges in G with some number of colors. We want to paint them so that, for each vertex, the colors of the edges incident to that vertex are all different. Among the colorings satisfying the condition above, construct one that uses the minimum number of colors.

Model Output

```
import sys
from collections import deque, defaultdict, Counter
from itertools import accumulate, product, permutations, combinations
from operator import itemgetter
from bisect import bisect_left, bisect_right
from heapq import heappop, heappush
from math import ceil, floor, sqrt, gcd, inf
from copy import deepcopy
import numpy as np
import scipy as sp

INF = inf
MOD = 1000000007

n = int(input())
ab = [list(map(int, input().split())) for i in range(n - 1)]

graph = [[] for i in range(n + 1)]
for a, b in ab:
    graph[a].append(b)
    graph[b].append(a)

visited = [False] * (n + 1)
color = [0] * (n + 1)
color[0] = 1
stack = [1]
while stack:
    x = stack.pop()
    for y in graph[x]:
        if visited[y]:
            continue
        visited[y] = True
        color[y] = color[x] + 1
        stack.append(y)

ans = min(color)
print(*ans, sep="\n")
```

Figure 4: An additional generated example. Here, the model output has many import statements.

Problem
There is a special square room with mirrors on each of the four walls. Except for the southwest corner, there are receptors on each of the remaining corners, numbered 0, 1, and 2. The square room has walls of length p, and a laser ray from the southwest corner first meets the east wall at a distance q from the 0th receptor. Return the number of the receptor that the ray meets first. (It is guaranteed that the ray will meet a receptor eventually.)

Example 1:
Input: p = 2, q = 1
Output: 2
Explanation: The ray meets receptor 2 the first time it gets reflected back to the left wall.

Starter Code

```
class Solution:
    def mirrorReflection(self, p: int, q: int) -> int:
```

Model Output

```
class Solution:
    def mirrorReflection(self, p: int, q: int) -> int:
        self.p = p
        self.q = q
        self.mirrors = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        return self.mirrors[self.p - 1][self.q - 1]
```

Figure 5: An example using the call-based format with some starter code.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

Maksym Zavershynskyi, A. Skidanov, and Illia Polosukhin. Naps: Natural program synthesis dataset. *2nd Workshop on Neural Abstract Machines and Program Induction*, 2018.

J. Zelle and R. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*, 1996.