Turing

Let’s Solve Some Puzzles

It’s mandatory to go through this entire deque in detail before attempting the coding challenge

Turing

Overview

We're collecting data to train AI models to implement programming problems and debug their own solutions. Programmers are expected to solve programming problems with optimal solution and along with proper plan and reasoning throughout the process of implementing a solution. In general, we want to train the AI on correctly solved tasks as well as the debugging process to arrive at the final correct solution. The correctness of the solution would be determined by whether it passes a hidden set of unit tests that check for edge cases.

The ideal candidate will be comfortable with Python programming and will have strong problem solving skills. The ideal candidate will also be familiar with an IPython/Jupyter notebook interface, or can be quickly taught how to use the interface.

Guidelines

The final submission will be evaluated on the following dimensions:

Plan/Approach	Explanation of your plan/approach of the solution. (Please start with Plan section as a comment describing the overall steps you will be using to implement. It is ok, if the initial plan didn't work, you can subsequently write the improved plan in the next section)
Reasoning	Your thought process for choosing a specific approach. (You can use detailed comment to describe the reasoning behind some important code snippet)
Test Cases	Test Inputs and Outputs (Make sure to give reasoning for some of the test inputs and the expected outputs. Also make sure to cover any edge cases)
Solution	The actual executable code.

As the process **does not involve any further rounds of interview**, it is important to communicate both the approach as well as the solution code of the problem, proactively in this section. You're allowed to execute any code you'd like to test out various ideas and function calls.

Instructions

You'll find a prompt describing a function to be implemented. We want to see how you plan, implement, and debug code. For that reason, we have three important instructions:

1. **Before starting an implementation, write a plan.** Although not required, we encourage you to provide example inputs and outputs for each step of the plan. Test each step of your plan and make a comment of which step you are working on. If you want to change your plan (feel free to!), write a new plan in a new comment section with a header “Plan #”.

2. **Instead of changing existing code in place, please copy paste the full code block, comment out the previous block and start working on copied code block, whenever you're making any changes to code.** This is important because we can't see the history of the code blocks in the IDE. This is mainly for AI models to get trained and understand the mistakes and how to recover from it.

3. **Before you write code, explain what you’re about to do in a comment.** For example, if a code block has an error that you want to fix, explain what caused the error and how you plan to fix it.

Turing

Example Problem and writing solution following the instructions

Problem:

Given a list of emails and URLs, return a dictionary, where each key is a URL and the value is how many emails have the same domain. Note that the domains begin with www. whereas the emails do not, and that emails with domains not in the list of urls should be ignored.

count_email_domains(

emails=['foo@a.com', 'bar@a.com', baz@b.com', 'qux@d.com'],

urls=['www.a.com', 'www.b.com', 'www.c.com'],

)

Turing

Example of writing a Plan/Approach:

We want to see how you **plan, implement, and debug** code. For that reason, we have **two important Instructions**:

- Before starting an implementation, write a plan.
- Adding comments for all approaches and solution iterations + test cases. Add multi-line comments using """operators in the IDE.

Choose a programming language: Python 3

```
""" Approach

1.Find the domain of each email

input: ['foo@a.com' , 'bar@a.com','baz@b.com','qux@d.com']
outputs: ['a.com','b.com','c.com]

2.Find the domains of each URL

inputs: ['www.a.com','www.b.com','www.c.com']
outputs: ['a.com','b.com','c.com']

3.Count the number of times each URL while ignoring the ones that do not matter

inputs: ['a.com','a.com','b.com','d.com'], ['a.com','b.com','c.com']
outputs: ['a.com':2,'b.com':1,'c.com':0]

4.Return the dictionary with the 'www' attached

inputs:['a.com':2,'b.com':1,'c.com':0]
outputs:['www.a.com':2,'www.b.com':1,'www.c.com':0]

"""
```

Making changes to solution code

Instead of changing existing code in place, please comment out the current code and add a new comment below explaining shortcomings in the previous code and your new approach.
This is important because we can't see the history of tested solutions in the IDE. Having meaningful iterations will provide a signal to recruiters of your ability to improve existing code/logic.

Original Code

```
""" Testing step 2"""

x = ['www.a.com','www.b.com','www.c.com']
y = []

for i in x:
    y.append(i.split('.')[1])
print(y)

```

Changed Code

```
""" Testing step 2
x = ['www.a.com','www.b.com''www.c.com']
y = []
for i in x:
    y.append(i.split('.')[1])
print(y)

This is not correct, I want to keep the entire domain including the '.com'. Rewriting
this step.
"""

"""Testing step 2 - iteration 2"""
x = ['www.a.com','www.b.com','www.c.com']
y = []
for i in x:
    y.append(i.split('.')[1] + '.' + i.split('.')[2])
print(y)

```

Turing

Summary of rules

1.	Test your code continuously and thoroughly (e.g. print statements, example input-outputs, etc.)
2.	When starting a new code section and/or fixing a bug, write a comment explaining your thought process. Avoid skipping any steps. When in doubt, show more detail.
3.	To edit existing code after executing it, comment out the existing code, add the explanation for your new approach in the comment. Successively add the new code below.
4.	Spend the last 25% of your time double checking your work for errors. You should be 100% confident that your code will pass all of our hidden unit tests.
5.	Avoid using the internet for debugging, unless to look up common documentation.

Handling Corner Cases. If a function specifies a precondition (e.g., “given a list of integers”), you can assume that this precondition will be satisfied (e.g., you do not need to check if the elements in the list are integers). However, you will still want to check valid corner-cases (e.g., an empty list of integers).

Instructions

1. The examination will consist of **2 questions.**
2. **Total time** to complete the coding challenge (both questions) is 300**mins.** Please allocate your time accordingly.
3. Please make sure you have a good and stable internet connection and power source. The coding challenge **cannot be paused** once it begins.
4. We strongly recommend using a programming language which is more suitable for the role you are applying for.
5. Please write your code **within the function provided.** The code editor also includes *Template Code*. DO NOT EDIT that section of the code, doing so will fail your submission.
6. You can create custom test cases and **test your code** for different edge cases an **unlimited number of times** before submission.
7. You cannot edit the code once you click **Submit.**
8. **Please take the practice test** before your formal attempt. The practice test has been allotted more time than the actual test so that you can familiarize yourself with the provided coding environment and challenge format.

Note: We will test your code against hidden test cases after your submission, therefore please make sure to test your solutions carefully before submitting. Please also remove any print statements before submitting your solution.