CIS1100.py — Spring 2025 — Final Exam

Name: _____ PennID (e.g. 12345): _____

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

_____ _____

Signature Date

- Do not open this exam until told by the proctor.
- You will have exactly 120 minutes to take this exam. There are nine questions plus a bonus.
- Make sure your phone is turned OFF (not on vibrate!) before the exam starts.
- Food and gum are not permitted—don't be noisy or messy.
- You may not use your phone or open your bag for any reason, including to retrieve or put away pens or pencils, until you have left the exam room.
- This exam is closed-book, closed-notes, and closed computational devices.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All code must be written in proper Python format.
- Do not separate the exam pages. Do not take any exam pages with you. The entire exam packet must be turned in as is.
- There are blank pages at the end of the exam if you need extra space for any graded answers.
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to you.
- When you turn in your exam, you may be required to show your PennCard. If you forgot to bring your ID, talk to an exam proctor immediately.
- We wish you the best of luck!

Q1. Types [18 points]

In the column marked **"Type,"** choose the type (`int`, `float`, `bool`, `str`, `list`, `set`, `tuple`, `range`, `dict`) for the expression, or write **"error"** if there is an error in the expression. You do not need to write the value of the expression. In cases where multiple answers are possible, any of them will be accepted.

Code	**Type**
`[3, 4, 5, 2, 1, 2][:3]`	
`set("alvvays")`	
`15 / 4`	
`15 // 4`	
`{7: [8, -8], 9: [10]}[8]`	
`5 > 3 and 2 < 1`	
`"19" + 96`	
`int("19") + 96`	
`ord(chr(68))`	
`[i + 1 for i in range(3, 6)]`	

Q2. Values [14 points]

Write the value that gets printed, or write **"error"** if there is an error during the execution of these lines of the program. **Hint:** remember that **None** is a value.

Question 2.1

```
x = 25 // 7
y = 25 % 7
print((x * 7) + y)
```

Answer:

Question 2.2

```
a = [5, 8, 3, 1]
a[2] = a[1] - a[3]
print(a[a[0] - 2])
```

Answer:

Question 2.3

```
x = 5
print(f"{x} * 3 = {x*3}")
```

Answer:

Question 2.4

```
print("cat" in "concatenate")
```

Answer:

Question 2.5

```
print(["a", "b", "c"] in ["a", "b", "c", "d"])
```

Answer:

Question 2.6

```
nums = [12, 13, 14, 15]
result = {n * n for n in nums if n % 3 == 0}
print(len(result))
```

Answer:

Question 2.7

```
def mystery(word, times):
   if times <= 0:
       return ""
   return word + mystery(word, times - 1)
print(mystery("ha", 3))
```

Answer:

Q3. Debug the Code: Airline Tickets [X points]

An airline ticket is represented by a tuple with three elements. The first element is the name on the ticket, the second is the destination, and the third is a boolean that indicates whether the ticket is still valid (`True`) or has already been used (`False`).
For example, `("Molly", "SFO", True)` represents a valid ticket where Molly is going to San Francisco.

Your job is to help debug a function called `process_ticket(...)` that:

1. returns `True` if there is a `valid` ticket matching that person's name and destination, or returns `False` otherwise.

2. replaces a matching and valid ticket's tuple with an equivalent ticket that is no longer valid.

As an example, here is how the function will be called in a script called `tickets.py`:

```
tickets = [("Molly", "SFO", True),
           ("Zwe", "LAX", False),
           ("Jared", "CDG", True)]

people_in_line = [("Molly", "SFO"),
                  ("Zwe", "LAX"),
                  ("Jared", "CDG"),
                  ("Harry", "FLV")]

for info in people_in_line:
    name = info[0]
    dest = info[1]
    if process_ticket(name, dest, tickets):
        print(f"{name} -> {dest}: allowed")
    else:
        print(f"{name} -> {dest}: denied")
print(f"Resulting ticket list: {tickets}"
```

And here's what would be printed to the terminal when run:

```
$ python tickets.py
Molly -> SF: allowed
Zwe -> LAX: denied
Jared -> CDG: allowed
Harry -> FLV: denied
Resulting ticket list: [("Molly", "SF", False),
                        ("Zwe", "LAX", False),
                        ("Jared", "CDG", False)]
```

There are **six** buggy lines in the program below. Use the table to identify the line number, explain the issue, and provide a corrected version. The first bug is already done for you.

```
1  def process_ticket(name, dest, tickets -> bool):
2      for i in range(tickets):
3          ticket = tickets[i]
4          this_name = ticket[0]
5          this_dest = ticket[i]
6          if this_name == name or this_dest == dest:
7              if ticket[2]:
8                  tickets[i][2] = False
9                  return True
10     return
```

Bug #	Line #	Issue/Fix
Bug #1	1	the type hint is wrong, -> bool goes outside the)
Bug #2		
Bug #3		
Bug #4		
Bug #5		
Bug #6		

Q4. Complete the Program: Essay Recovery [24 points]

You wrote an essay, but when you reopened the file, many words had been corrupted with numbers and symbols. Fortunately, the sentence structure remains intact so you only need to filter out the corrupted words.

Essay Format

The file begins with a title line, **which should be ignored**. Every following line is a sentence. Each sentence consists of words (delimited by spaces), but some of them may now contain digits or special characters (e.g., `"m1nd$Et"`, `"divid3d"`, `"Lum0n"`). These words are *corrupted*, and your task is to remove all such corrupted words.

Note: Each line always ends with a complete word followed by punctuation (such as ".", "!", or "?"). Do not remove a word just because it ends with punctuation. You can assume no other symbols will be used.

Complete the function **`clean_essay(filename)`**, which takes a **filename** and returns a **list of lists**. Each inner list contains the valid words from one line.

Example

If the file `essay.txt` contains:

```
Severance and the Divided Self
Severance e$xplores the fr@gmentation of m3mory and ident[ty.
Workers oper@te with artificial m1ndsets and er@sed lives.
Is separ@ting work from th3 self humane or harmful?
```

Calling:

```
clean_essay("essay.txt")
```

Should return:

```
[
  ["Severance", "the", "of", "and"],
  ["Workers", "with", "artificial", "and", "lives"],
  ["Is", "work", "from", "self", "humane", "or", "harmful"]
]
```

Incomplete Program

```
def clean_essay(filename: str) -> list[list[str]]:

    file = open(filename, "r")
    lines = file.___BLANK_0___
    file.close()

    output = []
    for line in ___BLANK_1___:  # remember to skip first line.
        words = line.strip().split()
        cleaned_line = []

        # iterate over indices and words together
        for i, word in ___BLANK_2___:

            if word.isalpha(): # take only non-corrupted words
                cleaned_line.___BLANK_3___

            elif i == len(words) - 1 and len(word) > 1:
                # special case for last word
                last_is_punctuation = word[-1] ___BLANK_4___ ".?!"
                rest_is_letters = word[___BLANK_5___].isalpha()

                if last_is_punctuation and rest_is_letters:
                    cleaned_line.append(word[:-1])
        output.append(cleaned_line)

    return output
```

Fill in the table below to answer this question.

Blank	**Code**
___BLANK_0___	
___BLANK_1___	
___BLANK_2___	
___BLANK_3___	
___BLANK_4___	
___BLANK_5___	

Q5. Coding & Unit Testing: Transit Planning [30 points]

SEPTA is Philadelphia's public transit system. They have a new naming scheme for their routes:

- Any route name starting with L or B is a **SUBWAY** route (e.g. L1 or B3).
- Any route name starting with G or T is a **TROLLEY** route (e.g. G1 or T5).
- Any route name that is entirely digits without a starting letter is a **BUS** route (e.g. 9, 12, 21, or 42).

Question 5.1 `get_route_type` [10 points]

Implement a function that takes in a string representing a route's name and returns a string representing what kind of route it is: **SUBWAY**, **TROLLEY**, **BUS**. If the route name does not match any of the rules listed above, return **INVALID**. Recall the method `str.isnumeric()` that can be called on a string to determine if it represents a number.

```
def get_route_type(name: str) -> str:









_
```

Question 5.2 Testing `get_route_type`

Complete some unit tests so that they verify the behaviors specified. The first unit test is completed for you as a reference.

Write a test verifying that the function correctly identifies a valid subway route.

```
from septa import get_route_type
import unittest

class SeptaTests(unittest.TestCase):
    def test_correctly_identifies_a_subway_route(self):
        input = "B2"
        expected = "SUBWAY"
        actual = get_route_type(input)
        self.assertEqual(expected, actual)
```

Write a test verifying that the function correctly identifies a valid bus route.

```
    def test_correctly_identifies_a_bus_route(self):









_
```

Write a test verifying that the function correctly identifies an invalid route name.

```
    def test_correctly_identifies_invalid_route(self):









_
```

Question 5.3 `collect_types` [10 points]

Implement a function that takes in a list of strings representing route names dictionary mapping each of the route types to a list of the routes in the input that belong to that type. You can assume that `get_route_types` is implemented correctly. You can see an example of this function being used below:

```
>>> routes = ["L1", "B4", "40", "G", "T5", "B1", "42"]
>>> collected_by_type = collect_types(routes)
>>> print(collected_by_type)
{'SUBWAY': ['L1', 'B4', 'B1'],
 'BUS': ['40', '42'],
 'TROLLEY': ['G', 'T5']}
```

```
  def collect_types(routes: list[str]) -> dict[str, list[str]]:
  _
```

Q6. Understanding Code Snippets

Below, we have five snippets listed in the first section. Each of these snippet operates on some list `nums`. For each **original snippet**, your job is twofold:

- Select the single **alternative snippet** that produces exactly the same value of `result`. There are more alternatives than originals, so some alternatives will not be selected.
- Compute what the value of `result` would be if `nums = [5, 4, 3, 6, 7]`.

Original Snippets:

1.

```
result = []
x = 1
while x % (len(nums) - 1) != 0:
    result.append(nums[x])
    x += 1
```

2.

```
result = ""
for num in nums:
    result += f"{num}"
```

3.

```
temp = []
for num in nums:
    temp.append(num * 2)
result = []
for num in temp:
    if num % 2 == 0:
        result.append(num)
```

4.

```
result = 0
for num in nums:
    if num >= nums[0]:
        result += num
```

5.

```
result = 1
for num in nums:
    if num % 2 == 0:
        result *= num
```

See next page for Alternative Snippets.

Alternative Snippets:

A.
```
result = list(map(lambda n: n + n, nums))
```

B.
```
temp = map(str, nums)
result = reduce(lambda acc, elem: acc + elem, temp)
```

C.
```
temp = map(lambda num: num >= nums[0], nums)
result = sum(temp)
```

D.
```
result = nums[1 : len(nums) - 1]
```

E.
```
temp = [num for num in nums if num >= nums[0]]
result = reduce(lambda acc, elem: acc + elem, temp)
```

F.
```
tmp = filter(lambda x: x % 2 == 0, nums)
result = reduce(lambda x, y: x * y, tmp)
```

G.
```
tmp = reduce(lambda x, y: x * y, tmp)
result = filter(lambda x: x % 2 == 0, nums)
```

Answer Here:

Original Snippet	Matching Alternate (Select A-E)	Result for `nums = [5, 4, 3, 6, 7]` (Write the Value)
1		
2		
3		
4		
5		

Q7. RottenMovie

Below is an almost-complete definition for a class called `RottenMovie`, which represents a movie listed on Rotten Tomatoes. Your job is to fill in the missing parts so that the class works as intended. Specifically, you will complete the constructor, finish the method that computes the average rating, and correctly call that method in order to classify the movie as `"Rotten"`, `"Fresh"`, or `"Certified Fresh"` based on its reviews.

```
class RottenMovie:
    def __init__(self, title: str):
        # Missing! See Part 1

    def add_rating(self, score: int):
        if 0 <= score <= 100:
            self.ratings.append(score)

    def average_score(self) -> float:
        # Missing! See Part 2

    def classification(self) -> str:
        count = len(self.ratings)
        avg = # Missing! See Part 3

        if count < 3:
            return f"{self.title}: Not enough ratings"
        elif avg < 60:
            return f"{self.title}: Rotten"
        elif avg >= 75 and count >= 20:
            return f"{self.title}: Certified Fresh"
        else:
            return f"{self.title}: Fresh"
```

Question 7.1

Complete the constructor method `__init__` so that a new `RottenMovie` stores the `title` and can also keep track of all the `ratings` it receives in a list. Keep in mind that the `RottenMovie` object should start with zero reviews. **Write only the body of the method.** *Hint: can you see what attributes are used in the other methods?*

Answer:

Question 7.2

Complete the `average_score` method so that it returns the average of all the ratings the movie has received so far. If there are no ratings yet, it should return `0.0`. *Hint: remember that `sum` can be used to find the sum of a sequence.* **Write only the body of the method.**

Answer:

Question 7.3

In the `classification` method, complete the line that sets the value of `avg` by calling the correct method on the current object; that is, the object that the `classification` method has been called on. **Write only the line that sets the `avg` variable.**

Answer:

Question 7.4

Finally, fill in the unit test below that verifies that `classification` behaves correctly for a `"Fresh"` movie, which has at least three ratings of an average of at least 60. Your job will be to initialize a new `RottenMovie` object and add enough ratings of a high enough quality so that the classification returned is `"Fresh"`.

```
import unittest
from tomato import RottenMovie
class TomatoTests(unittest.TestCase):
    def test_fresh_movie_classification(self):
        # TODO: initialize and set up a RottenMovie object
        #       with enough positive reviews









        actual = ____________________ # TODO!
        expected = "Fresh"

        self.assertEqual(expected, actual)
```

Q8. Pandas: Spread Bagelry [X points]

You may find the appendix at the end of this exam helpful.

Spread Bagelry is upgrading to a digital self-order kiosk and needs your help. You are provided with a CSV file called `spread_menu.csv`, which logs popular items sold over the last two weeks. Each row represents a single menu item.

Below is a preview of the file:

```
Item Name,Category,Price
The Standard,All Day Breakfast Sandwiches,9.25
Frittata Scramble,All Day Breakfast Sandwiches,9.75
Egg White,All Day Breakfast Sandwiches,9.75
Nova Scramble,All Day Breakfast Sandwiches,10.25
Avocado Toast,All Day Breakfast Sandwiches,9.00
The Cure,All Day Breakfast Sandwiches,13.50
The Classic,Signature Sandwiches,14.00
Roast Chicken BLT,Signature Sandwiches,11.00
Wood Oven Tuna Melt,Signature Sandwiches,10.50
Bagel Grilled Cheese,Signature Sandwiches,10.25
Roast Chicken Cheesesteak,Signature Sandwiches,10.50
Wood Oven Beef Brisket,Signature Sandwiches,12.50
Hot Carnegie Deli Pastrami,Signature Sandwiches,13.50
The Carnegie Reuben,Signature Sandwiches,13.50
The Balboa,Signature Sandwiches,11.00
The Garden Veggie,Signature Sandwiches,7.00
Montreal Pizza Bagels,Signature Sandwiches,7.00
```

Question 8.1

Write one line of code that reads in the CSV as a DataFrame named **`df`**.

Answer:

Question 8.2

Extract the `"Item Name"` column from `df` **as a Series** and assign it to a variable called **`items`**.

Answer:

Question 8.3

Some item names are too long to fit properly on kiosk buttons. Write one line of code that returns a Series of **`True`** or **`False`** values depending on whether each item name is longer than 15 characters and save the result in a variable called **`long_name_filter`**. You may use the **`items`** or **`df`** variables from the previous parts.

Answer:

Question 8.4

Write one line of code that filters the DataFrame to contain **only the items whose names are longer than 15 characters** and save it in a variable called **`long_names_df`**. These are the ones the kiosk team may need to rename. You may use the **`items`** or **`df`** variables from the previous parts, and may assume the result from the previous question **(8.3)** is stored in a variable called **`long_name_filter`**.

Answer:

Question 8.5

To analyze pricing patterns and see if changing those long names might impact sales, compute the average price of all items whose names are longer than 15 characters. You may use the **`items`** or **`df`** variables from the previous parts, and may assume the filtered result is stored in a variable called **`long_names_df`**, if helpful.

Answer:

Q9. Who's Amy Gutmann? [12 points]

You're writing a script to scrape content from `https://www.WhoIsAmyG.com`, a page celebrating the academic and diplomatic accomplishments of Amy Gutmann. She's not just the person after whom the building is named!

That site includes the following HTML:

```
<html>
  <head> <title>Amy Gutmann</title> </head>
  <body>
    <h1>Career Highlights</h1>
    <ul>
      <li class="role">
        8th President of the University of Pennsylvania (and longest
            in office)
      </li>
      <li class="award">
        Named one of the world's 50 most powerful leaders by Forbes
      </li>
      <li class="role">U.S. Ambassador to Germany (2022-2024)</li>
      <li class="award">
        Recipient of the Harvard Centennial Medal</li>
    </ul>
  </body>
</html>
```

Question 9.1

Write a line of code that requests the page from the correct *url* and stores the HTML contents of that page ***as a string*** in a variable called `html`.

Answer:

Question 9.2

Extract a list of all text from `<li>` elements with class `"award"` only. Your result should be a list of strings. You can assume you have already parsed the html as follows: `soup = BeautifulSoup(html, "html.parser")`.

Answer:

Q10. Bonus

Reflect: what would you tell your past self from August 2024? Or, if you prefer, you can create a piece of art or tell a joke! All exams will get full credit for this question even if you leave it blank.

Answer:

Appendix: Pandas

Note here that **pd** refers to the **pandas** library, **df** refers to a **DataFrame**, and **s** refers to a **Series**.

Function	Description
df.col	Accesses the column named col in df as a Series.
df['col']	Accesses the column named col in df as a Series.
df[['col1', 'col2', ...]]	Accesses a list of columns (here, ['col1', 'col2', ...]) as a DataFrame.
df.min(axis=0)	Finds the minimum of each row or column.
df.max(axis=0)	Finds the maximum of each row or column.
df.mean(axis=0)	Finds the mean/average of each row or column.
df.sum(axis=0)	Finds the sum of each row or column.

Remember that you can use comparison operators with values on Series. For example,

- s > 0
- s == "Hello"
- s <= 8

Extra Work Space

You may use this page for additional space for answers; keep it attached to this exam. Clearly note on the original question page that your answer is on this extra page, and clearly note on this page what question you are answering.

Answer: