CIS1100.py — Spring 2025 — Exam I

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 60 minutes to take this exam.
- 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, including all curly braces and semi-colons.
- 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	Q2	Q3	Q4	Q5	Q6 (Bonus)

Q1. Types [18 points]

In the column marked **"Type,"** choose the type (`int`, `float`, `bool`, `str`, `list`, `set`, `tuple`, `range`) 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**
`28 % 7`	
`not not not True`	
`"August " + 29`	
`"August " + "29"`	
`chr(68)`	
`[3, 4, 5, 2, 1, 2][3]`	
`{3, 4, 5}`	
`range(3, 6)`	
`{3 : 4, 5 : 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

```
t = 17 // 10
o = 17 % 10
print(t + o)
```

Answer:

Question 2.2

```
l = [0, 3, 2, 1]
l[3] = l[0] + l[1]
print(l[len(l) - 1])
```

Answer:

Question 2.3

```
print(f"10 + {20}")
```

Answer:

Question 2.4

```
print("peas" in "appeasement")
```

Answer:

Question 2.5

```
print(["r", "e", "a", "d"] in ["t", "r", "e", "a", "d"])
```

Answer:

Question 2.6

```
a = [10, 21, 30, 42, 55]
b = {elem / 2 for elem in a if elem > 20} # set comprehension!
print(len(b))
```

Answer:

Question 2.7

```
def celebrate(name, amount)
    msg = f"Congrats, {name}"
    for i in range(amount):
        msg = msg + "!"
    return msg
print(celebrate("buddy", 3))
```

Answer:

Q3. Debug the Code: Fancy Words [20 points]

A word is *fancy* if it contains no repeat letters and it starts with a capital (uppercase) letter. Help debug Harry's shaky attempt to write a function that returns a new list containing only the elements of the input list `words` that are *fancy*. `"Joel"` is fancy, but `"joel"` and `"Harry"` are not.

There are **six** buggy lines in the program below. Use the table below to identify the line number and issue of each buggy line and provide some replacement code that would fix the line. The first bug is solved for you.

```
1  delft fancy_words_only(words: list[str]) -> list[str]:
2      output = []
3      for word in words:
4          # no repeat letters
5          letters = set()
6          for letter in word:
7              letter.add(letters)
8          no_repeats = len(letters) == word
9
10         # capitalized
11         first_letter = word[1]
12         is_capitalized = "A" <= first_letter >= "Z"
13
14         if no_repeats and is_capitalized:
15             output.append(word)
16     return
```

Bug #	Line #	Issue/Fix
Bug #1	1	the signature is wrong, change delft to def
Bug #2		
Bug #3		
Bug #4		
Bug #5		
Bug #6		

Q4. Complete the Program: Whose Line? [24 points]

The CIS1100 Staff have written a short movie screenplay and would love to parse it using techniques learned in CIS1100. They want to demonstrate how they can manipulate files and process structured text using the concepts learned in CIS1100! *This page only contains instructions; the place for you to write will be on the next page.*

Sample Movie Script

This sample script is stored inside of file `script.txt`.

```
The Group Exam
SETTING Levine
ACTOR Adi
  Geez I wonder how long this will take?
ACTOR Molly
  Don't worry, the exam will be over soon.
ACTOR Zwe
  Are you guys hungry at all?
ACTOR Adi
  Ugh, why did we agree to do this.
```

Script Format

The first line is the title of the film. A `SETTING` indicator appears before a set of actor lines and only occurs once. An `ACTOR <name>` indicator marks the start of an actor's lines, which continue until a new `ACTOR <name>` appears, or the file ends.

Complete the function **`parse_actor_lines(filename)`**, which accepts a **filename** as input and returns a **dictionary** where each actor's name maps to a **list** of their spoken lines. You can assume that the file is correctly formatted and that the words `SETTING` and `ACTOR` do not appear within dialogue.

Example

Calling:

```
parse_actor_lines("script.txt")
```

Should return the following dictionary:

```
{
    "Adi": ["Geez I wonder how long this will take?",
            "Ugh, why did we agree to do this."],
    "Molly": ["Don't worry, the exam will be over soon."],
    "Zwe": ["Are you guys hungry at all?"]
}
```

Incomplete Program

```
def parse_actor_lines(filename: str) -> dict[str, list[str]]:
    # open the file and read all lines
    file_handler = open(___BLANK_0___, "r")
    lines = file_handler.readlines()
    file_handler.close()

    actor_lines = {}
    current_actor = ""
    lines = lines[2:] # skip the title and setting
    for line in lines:
        line = line.strip()
        if ___BLANK_1___ in line:  # Detect actor name
            # read <name> from "ACTOR <name>"
            current_actor = line.split()___BLANK_2___

            if current_actor not in actor_lines:
              actor_lines[current_actor] = ___BLANK_3___

        else:
            ___BLANK_4___.append(line)

    return ___BLANK_5___
```

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: Price Tracker [30 points]

You're writing a price tracker to be used to save money while shopping on the internet. To do this, you'll write a few functions.

Question 5.1 `lowest_in_k_days` [10 points]

Implement a function `lowest_in_k_days` that calculates the number of days since a product's price has been at or below the given `threshold`. To do this, the function will also take in a list of previous prices called `history` wherein the first element (`history[0]`) is the price one day ago, the second element represents the price two days ago, and so on. If the price has never been below or equal to the given `threshold`, return the number of days in `history`. First, study the unit tests below that verify the correct behavior. Then, implement the function.

```
  def test_lowest_in_one_day(self):
      history = [3, 6, 9]
      threshold = 4
      actual = lowest_in_k_days(threshold, history)
      self.assertEqual(1, actual)
  def test_lowest_in_three_days(self):
      history = [10, 8, 4, 10]
      threshold = 4
      actual = lowest_in_k_days(threshold, history)
      self.assertEqual(3, actual)
  def test_new_lowest(self):
      history = [3, 6, 9, 12]
      threshold = 1
      actual = lowest_in_k_days(threshold, history)
      self.assertEqual(4, actual)
```

```
  def lowest_in_k_days(threshold: int, history: list[int]) -> int:
      return
```

Question 5.2 `is_rare` [16 points]

First, implement a function `is_rare` that takes in today's `price` and a list of previous prices called `history` and returns `True` if the fraction of days on which the item cost *l*ess than `price` is less than 50%. Otherwise, return `False`.

For example, for a history of `[3, 4, 6]`, the fraction of days where the price is below 5 is 66.6%, so 5 is not a rare price. 4 is a rare price because 33.3% of days have prices less than 4.

```
def is_rare_price(price: int, history: list[int]) -> bool:









    return
```

Then, complete the following two unit tests so that they correctly represent the behavior of `is_rare`.

```
def not_rare_price(self):
    history = [3, 4, 5, 6, 7]
    price = ________________ # TODD!

    # same as self.assertEqual(False, is_rare(price, history))
    self.assertFalse(is_rare(price, history))

def actually_rare_price(self):
    history = [3, 4, 5, 6, 7]
    price = 0
    # TOOD: below, write an assertion to make
    #       the test pass with correct behavior

    ____________________________________________________________
```

Question 5.3 `should_buy_now` [6 points]

Finally, write a function that determines whether or not a user should buy an item at the current `price`. They should do so if it has been at least 3 days since the price has been at or below this level and this is a rare price. You should do this in two or three lines by calling functions you've already written! *(You do not need to have answered the previous questions correctly in order to do this.)*

```
def should_buy_now(price: int, history: list[int]) -> bool:









    return
```

Q6. Bonus [2 points]

Create a piece of art (e.g. drawing, poem, short program, anything you like)! If you are unsure of what to make, select one member of the course staff and make art about that person. All exams will get full credit for this question even if you leave it blank. And no, it does not have to be PennDraw!

Answer:

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: