CIS 1100 Fall 2026 Practice 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.
- 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 a dark 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 [9 points]

What is the type of the following expression? Choose `error` if evaluating it raises an error.

Question 1.1 [1 points]

`"c" + 67` Type: Error

Question 1.2 [1 points]

`"3"` Type: String

Question 1.3 [1 points]

`"3" in [3]` Type: Bool

Question 1.4 [1 points]

`28 % 7 == 1` Type: Bool

Question 1.5 [1 points]

`[4] + 5` Type: Error

Question 1.6 [1 points]

`[4] + [5]` Type: List

Question 1.7 [1 points]

`3` Type: Int

Question 1.8 [1 points]

`len("hello!")` Type: Int

Question 1.9 [1 points]

`input("prithee")` Type: String

Q2. Values [8 points]

What is printed when the following program runs? Write `error` if the program raises an error. If the output spans several lines, write it on one line separated by spaces.

Question 2.1 [1 points]

```
h = "bird"
h = "plane"
print(f"It's a {h}! It's a {h}!")
```

Answer: It's a plane! It's a plane!"

Question 2.2 [1 points]

```
k = [7, "tree", True]
for thing in range(k):
    print(thing)
```

Answer: Error

Question 2.3 [1 points]

```
print(10 + len("10"))
```

Answer: 12

Question 2.4 [1 points]

```
x = [4, 5, 6, 7]
x[4] = 8
print(x)
```

Answer: Error

Question 2.5 [1 points]

```
t = 18 // 9
r = 18 % 9
print(t - r)
```

Answer: 2

Question 2.6 [1 points]

```
def mystery(start, text):
    msg = ""
    while start > 0:
        msg = msg + str(start) + "..."
        start -= 1
    return msg + f"{text}"

print(mystery(3, "Go!"))
```

Answer: 3...2...1...Go!

Question 2.7 [1 points]

```
j = []
j.append([4, 5, 6])
j.append(5)
print(j)
```

Answer: [[4, 5, 6], 5]

Question 2.8 [1 points]

```
k = ["to", "be", "or"]
for thing in range(len(k)):
    print(thing)
```

Answer: 0 1 2

Q3. Debugging [1 points]

Question 3.1 [1 points]

A shipping company calculates costs as follows:

- **Weight brackets:** base cost is $5 under 5 lbs, $8 for 5–20 lbs (inclusive), $12 over 20 lbs.
- **Distance:** add $0.10 per mile for the first 100 miles, then $0.05 per mile beyond 100.
- **Fragile:** if `fragile`, add a flat $3 fee (before the express multiplier).
- **Express:** if `express`, double the entire total.
- **Discount:** if the package is both over 20 lbs *and* travelling over 200 miles, apply a 10% discount last.

The program below has seven bugs. Bug #1 is the missing colon on line 1, which is given to you.

```
1  def calc_shipping(weight: float, miles: int, fragile: bool, express:
       bool) -> float
2      if weight < 5:
3          base = 5
4      if weight <= 20:
5          base == 8
6      else:
7          base = 12
8
9      if 100 <= miles:
10         distance_cost = miles * 0.10
11     else:
12         distance_cost = 100 * 0.10 + (miles - 100) * 0.05
13
14     total = base - distance_cost
15     if fragile:
16         total = total + 3
17     if express:
18         total * 2
19     if weight > 20 and weight > 200:
20         total = total * 0.90
21     return total
```

For each remaining bug, give the line number it is on, describe the issue briefly, and write a correction for that line. *Line numbers are graded automatically; your descriptions and fixes are read by a human.*

Bug	Line #	Issue (describe briefly)	Fix (add/replace just what's necessary)
#2	4	Conditions should be mutually exclusive	elif weight <= 20:
#3	5	Compares instead of assigning	base = 8
#4	9	Condition is backwards	if miles <= 100:
#5	14	total cost is base + distance_cost	total = base + distance_cost
#6	18	value of total should be updated	total = total * 2
#7	19	need to check weight and miles	if weight > 20 and miles > 200:

Q4. Complete the Program [1 points]

Question 4.1 [1 points]

Maintenance staff surveyed every dorm room and stored the results in `survey.txt`:

```
2
HARRISON 3
101 3 SAFE
102 2 FLOODED
103 4 SAFE
RODIN 2
2401 2 FLOODED
2301 4 FLOODED
```

The first line gives the number of building sections. Each section begins with a line holding the building name and how many rooms are listed for it; each room line holds the room number, the number of beds, and either SAFE or FLOODED. The program should print a list of all flooded rooms, then the total number of beds in flooded rooms.

```
survey_file = open("survey.txt", "r")
num_buildings = int(survey_file.readline())
flooded_beds = 0
flooded_rooms = ___BLANK_0___

for idx in range(num_buildings):
    building_info_line = ___BLANK_1___
    building_info_contents = building_info_line.split()
    building_name = building_info_contents[0]
    num_dorms = int(building_info_contents[1])

    for other_idx in range(___BLANK_2___):
        dorm_info_line = survey_file.readline().strip()
        dorm_info_contents = dorm_info_line.split()

        if ___BLANK_3___:
            flooded_rooms.append(f"{building_name} {___BLANK_4___}")
            flooded_beds += ___BLANK_5___

print(f"Here are the flooded rooms: {flooded_rooms}")
print(f"There are {flooded_beds} beds flooded.")
```

Fill in each blank below. Each is worth an equal share of the credit.

Blank	**Code**
____BLANK_0____	[]
____BLANK_1____	survey_file.readline().strip()
____BLANK_2____	num_dorms
____BLANK_3____	dorm_info_contents[2] == “FLOODED”
____BLANK_4____	dorm_info_contents[0]
____BLANK_5____	int(dorm_info_contents[1])

Q5. Coding: Compression [3 points]

Question 5.1 [1 points]

Compression reduces the space needed to store a message. Here is a scheme for lists of integers: replace each run of consecutive increasing integers with a `range` covering it.

Write `compress()`. Some requirements:

- The input list may be empty, but will only ever contain integers.
- Every range you produce must have a step of 1.
- Only *increasing* runs compress: `[3, 2]` becomes `range(3, 4), range(2, 3)`.

Call	Returns
`compress([])`	`[]`
`compress([100])`	`[range(100, 101)]`
`compress([1, 2, 3, 4, 5, 6, 7, 8])`	`[range(1, 9)]`
`compress([1, 2, 3, 4, 6, 7, 8])`	`[range(1, 5), range(6, 9)]`
`compress([-1, 0, 1, 3, 6, 8])`	`[range(-1, 2), range(3, 4), range(6, 7), range(8, 9)]`
`compress([-1, 0, 1, 3, 2])`	`[range(-1, 2), range(3, 4), range(2, 3)]`

Write your implementation below. This question is graded by a human, so the reference solution is shown after you submit.

```
  def compress(nums: list[int]) -> list[range]:
```

Answer:

```
def compress(nums: list[int]) -> list[range]:
    result = []

    if nums == []:
      return result

    start = nums[0]
    prev = nums[0]
    for n in nums[1:]:
        if n == prev + 1:
            prev = n
        else:
            result.append(range(start, prev + 1))
            start = n
            prev = n

    result.append(range(start, prev + 1))
    return result
```

Question 5.2 [1 points]

Now reverse the compression: given a list of ranges, return a list of all the integers in all of those ranges, in the same order. (This goes from the right column back to the left column in the examples from the previous part.)

Call	Returns
`uncompress([range(1, 5), range(6, 9)])`	`[1, 2, 3, 4, 6, 7, 8]`
`uncompress([])`	`[]`

Write your implementation below. This question is graded by a human, so the reference solution is shown after you submit.

```
def uncompress(compressed: list[range]) -> list[int]:
```

Answer:

```
def uncompress(compressed: list[range]) -> list[int]:
    result = []
    for r in compressed:
        for n in r:
            result.append(n)
    return result
```

Question 5.3 [1 points]

Compressing is not always worth it. Call a list of ints **efficient** for compression if it has 5 or more elements *and* the compressed version is at most half the length of the uncompressed version.

Write `is_efficient()`, returning `True` when a list is efficient for compression and `False` otherwise. **Do this without writing any for loops, while loops, or other conditionals.** You may assume `compress()` exists and works correctly.

Call	Returns
`is_efficient([1, 2, 3, 4, 5, 6])`	`True`
`is_efficient([1, 2, 3])`	`False`
`is_efficient([5, 1, 9, 3, 7])`	`False`

Write your implementation below. This question is graded by a human, so the reference solution is shown after you submit.

```
def is_efficient(nums: list[int]) -> bool:
```

Answer:

```
def is_efficient(nums: list[int]) -> bool:
    return len(nums) >= 5 and 2 * len(compress(nums)) <= len(nums)
```

Extra Work Space

Answer: