# CIS 1100 — Fall 2024 — Practice Exam

Full Name: ___________________________________

PennID (e.g. 12345678): ____________________________

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

_______________________ ____________________

Signature Date

Instructions are below. Not complying will lead to a 0% score on the exam.

- 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 semicolons.
- Do not separate the exam pages. Do not take any exam pages with you. The entire exam packet must be turned in as is.
- Only answers on the FRONT of pages will be graded. There are two 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 Fill In The Blank

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.

| **Statement** | **Type** |
|---|---|
| `"yes" + "45"` | |
| `5 > 8 > 6` | |
| `ord("apple"[0]) - 45` | |
| `True and 3 < 4 and not "yes" == "no"` | |
| `sys.argv` | |
| `"pearson"[2:7]` | |
| `[""]` | |
| `range(10)[10]` | |

## Q2. Values Fill In the Blank

Write the value that gets printed, or write **"error"** if there is an error during the execution of these lines of the program.

### Question 2.1

```
print(int("4") % 3)
```

Answer:

### Question 2.2

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

Answer:

### Question 2.3

```
s = {1, 2, 3}
print(s[3])
```

Answer:

### Question 2.4

```
print("10" + "20")
```

Answer:

### Question 2.5

```
numbers = [4, 2, -1]
x = 1
for num in numbers:
      x = x * numbers
print(x)
```

Answer:

## Q3. Tracing

Here's a python file that features a few functions.

```
def func_three(x, y):
      print(f"func_three arguments: x={x}, y={y}")
      result1 = x * 3
      result2 = y * 2
      final_result = result1 + result2
      print(f"func_three returning: {final_result}")
      return final_result

def func_two(num):
      print("func_two argument: num=" + str(num))
      result = num * 2
      print("funct_two returning: " + str(result))
      return result

def func_one(a, b):
      print(f"func_one arguments: a={a}, b={b}")
      result1 = a + b
      result2 = func_two(result1)
      result3 = func_three(a, b)
      final_result = result2 - result3
      print(f"func_one returning: {final_result}")
      return final_result

def main():
      print("Starting main")
      x = 10
      y = 5
      z = func_one(y, x)
      print("The final result is: " + str(z))

if __name__ == "__main__":
      main()
```

When the program is run as `python tracing_exercise.py`, eight lines are printed. For each of the following lines, fill in the blanks in the **"Printed Line"** column (on the next page) to show what values the variables have when they are printed out. Also, mark the order in which they are printed in the **"Order"** column starting at 1. The order for the first line is marked for you.

| **Printed Line** | **Order** |
|---|---|
| `Starting main` | 1 |
| `The final result is: ______` | |
| `func_one arguments: a=_____, b=______` | |
| `func_one returning: ________` | |
| `func_two argument: num=________` | |
| `func_two returning: ________` | |
| `func_three argument: x=______, y=______` | |
| `func_three returning: ________` | |

## Q4. Complete the Program: *clock.py*

The following program is supposed to print out a time readout every fifteen minutes over the course of one day, starting with 12:00 AM, then 12:15 AM, then 12:30 AM and so on. Times in the second half of the day should be printed using the 12-hour clock, meaning that the full set of printed times should include 96 timestamps and should match the format & pattern shown in the following example:

```
12:00 AM
12:15 AM
12:30 AM
12:45 AM
1:00 AM
...       // ellipses are not printed literally;
11:30 AM  // they are here only for abbreviation.
11:45 AM
12:00 PM
12:15 PM
...
11:45 AM
```

Help write this clock program by filling in the blanks in the program on the next page.

```
for hour in range(_BLANK_0_):
    for minute in range(0, 60, _BLANK_1_):
        if _BLANK_2_ or hour == 12:
            hour_to_print = "12"
        else:
            hour_to_print = str(hour % _BLANK_3_)
        if minute == 0:
            minute_to_print = ":00"
        else:
            minute_to_print = ":" + str(minute)
        if _BLANK_4_:
            suffix = "AM"
        else:
            suffix = "PM"
        print(hour_to_print + minute_to_print + " " + suffix)
```

| Blank # | Code |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

## Q5. Coding: Password Hygiene

A password can be represented by a `str`. A password is said to be **strong** if it contains an uppercase letter (a letter between 'A' and 'Z'), a lowercase letter (a letter between 'a' and 'z'), and a digit (a letter between '0' and '9'). A password is said to be **valid** if it does not contain any forbidden characters: ' ' (a space) or '-' (a hyphen). Finally, a password is **good** if it is both strong and valid. Write the following three functions: `is_strong`, `positions_of_invalid_chars`, and `is_good`. Note the function headers & signatures for each that describe how they should behave: your functions need to return values of the correct types to receive credit! Your implementation of `is_good` must be completed in one line to receive credit, although you can still get credit for `is_good` even if your other functions are not correctly implemented.

### Question 5.1

```
def is_strong(password):
    """
    Input: a string representing the password to test
    Output: True if the password is considered "strong"
    and false otherwise.

    A password is "strong" if it contains an uppercase
    letter, and a lowercase letter, and a digit character.
    """
```

## Question 5.2

```
def positions_of_invalid_chars(password):
    """
    Input: a string representing the password to test
    Output: a list containing the indices of all invalid characters,
            which are either ' ' or '-'.
    positions_of_invalid_chars("finePassword") → []
    positions_of_invalid_chars("harry-smith") → [5]
    positions_of_invalid_chars("i love-you") → [1, 6]
    positions_of_invalid_chars("- - -") → [0, 1, 2, 3, 4]
    """
```

## Question 5.3

```
def is_good(password):
    """
    Input: a string representing the password to test
    Output: true if the password is both strong and valid, false otherwise.

    A password is good if it is both strong and valid. Use the previous two
    functions you wrote to complete this one in **one line only!**
    """
```