COSC 101, Practice Exam for Midterm #1
September 2023

Name: _____

Please write your name above. Do not start the exam until instructed to do so.

You have 50 minutes to complete this exam.

There are 5 questions and a total of 36 points available for this exam. Don't spend too much time on any one question.

Since indentation is important in Python, please be sure that your use of indentation is obvious for any code you write.

If you want partial credit, show as much of your work and thought process as possible.

If you run out of space for answering a question, you can continue your answer on one of the blank pages at the end of the exam. If you do so, be sure to indicate this in two places: (1) below the question, indicate which blank page contains your answer, and (2) on the blank page, indicate which question you are answering.

The last page of the exam contains documentation for string and list methods.

Question	Points	Score
1	8	
2	6	
3	4	
4	10	
5	8	
Total:	36	

1. (8 points) Assume that the following statements have already been executed:

```
x = 3
y = "4"
z = 8.4
s = "Raiders"
```

For each of the following expressions, evaluate the expression and write the resulting value, or describe the error in the code that would prevent it from running.

(a) `int(y) * 2 ** x`

Solution:

```
32
```

(b) `x ** (z // x)`

Solution:

```
9.0
```

(c) `s[0:4:2]`

Solution:

```
Ri
```

(d) `y + x * str(x)`

Solution:

```
'4333'
```

(e) `s[7]`

Solution:

```
IndexError: string index out of range
```

(f) `list(range(0,18,3)`

Solution:

```
[0, 3, 6, 9, 12, 15]
```

(g) `int(z)*x`

Solution: 24

(h) `'o' * x + y`

Solution:

```
'ooo4'
```

(i) `input='hello world'`

Solution:

```
Syntax error: variables can not be named as a reserved word
(input is a built-in function)
```

2. (6 points) Consider this program:

   What is printed by the following program?

```
def loop() -> None:
    i = 0
    j = 0

    for num in range(10, 40, 10):
        i = i + 1
        print(i, j, num, num*i)
        j = i ** 2
loop()
```

Solution:

```
1 0 10 10
2 1 20 40
3 4 30 90
```

3. (4 points) What is the output of the following program?

```
def function(b:str) -> str:
     a = b * 2
     b = b + 'corn'
     print(f"A = {a}")
     return b

def main() -> None:
    a = 'corn'
    b = 'candy'
    b = function('pop')
    print(f"a = {a}")
    print(f"b = {b}")

main()
```

Solution:

```
A = poppop
a = corn
b = popcorn
```

4. (10 points) For this problem, select one line of code from each of the pairs of lines of code below to solve the following problem:

   *W*rite a program that asks for student grades and prints the highest score and the index of the student with the highest score in the list.

```
A1 scores_list = get_scores(num_scores)
A2 scores_list = get_scores()

B1 max_index = scores_list.index(max_score)
B2 max_index = scores_list[max_score]

C1 max_score = max(scores)
C2 max_score = max(scores_list)

D1 for i in range(num_students):
D2 for i in num_students:

E1 print("Student 3 has the highest score of 100")
E2 print("Student",stats[0],"has the highest score of",stats[1])

F1       score = int(input(f"Score for Student {i}: "))
F2       score = int(scores_list[i])

G1 return [max_index,max_score]
G2 return max_score[max_index]

H1       scores_list[i] = score
H2       scores_list.append(score)

I1 stats = get_max(int(scores))
I2 stats = get_max(scores)

J1 scores_list = ["90", "99", "94", "100"]
J2 scores_list = []
```

   Select only 10 lines of code from above, and **only one line from each pair**. You may write only the line identifiers (*e.g.*, E2) below, or write out the code. Your selections should *only* go on numbered lines below (see next page).

```
def get_scores(num_students: int) -> list:
    '''This function receives an int and returns a list of scores. Scores are added to the list using a for loop which asks users for a score. Each score in the return list must be a whole number. Usage example with 90,99,94, and 100 as the user inputs: get_scores(4) would return [90,99,94,100]'''
```

1 –

2

3

4

```
    return scores_list

def get_max(num_scores: int) -> list:
    '''This function receives an integer representing the number of scores to be read. It then returns the index and score of the student with highest score. Usage example: get_max(4) with user input 90,99,94, and 100 must return [3,100]'''
```

5 –

6

7

8

```
def main() -> None:

    scores = input("Number of scores: ") # user inputs: 4
```

9

10

```
    # expected output after user inputs 90,99,94,and 100 with get_max: Student 3 has the highest score of 100

main()
```

Solution:

- J2 `scores_list = []`
- D1 `for i in range(num_students):`
- F1 `score = int(input(f"Score for Student {i}: "))`
- H2 `scores_list.append(score)`
- A1 `scores_list = get_scores(num_scores)`
- C2 `max_score = max(scores_list)`
- B1 `max_index = scores_list.index(max_score)`
- G1 `return [max_index,max_score]`
- I1 `stats = get_max(int(scores))`
- E2 `print("Student",stats[0],"has the highest score of",stats[1])`

5. (8 points) Write a function called `print_name` that takes in the name of the user and prints a special greeting for them. This special greeting will say hi to the user multiple times, depending on how long their name is. Also, the number of letter 'i's in the word 'hi' and '!'s at the end will increase for each additional line. Your function's output should match these examples:

```
What is your name? Bob
Hi Bob!
Hii Bob!!
Hiii Bob!!!

What is your name? Mary
Hi Mary!
Hii Mary!!
Hiii Mary!!!
Hiiii Mary!!!!
```

Solution:

```python
def print_name(name: str) -> None:
    for x in range(len(name)):
        # compute number of 'i' and '!' to print
        count = x + 1
        # print the line (using string copies)
        print(f"H{'i'*count} {name}{'!'*count}")

def main() -> None:
    # get name from user
    name = input("What is your name? ")
    print_name(name)

main()
```

String methods

- `upper()` — Returns a string in all uppercase
- `lower()` — Returns a string in all lowercase
- `capitalize()` — Returns a string with first character capitalized, the rest lower
- `strip()` — Returns a string with the leading and trailing whitespace removed
- `lstrip()` — Returns a string with the leading whitespace removed
- `rstrip()` — Returns a string with the trailing whitespace removed
- `count(item)` — Returns the number of occurrences of item
- `replace(old, new)` — Replaces all occurrences of old substring with new
- `center(width)` — Returns a string centered in a field of width spaces
- `ljust(width)` — Returns a string left justified in a field of width spaces
- `rjust(width)` — Returns a string right justified in a field of width spaces
- `find(item)` — Returns the leftmost index where the substring item is found, or -1 if not found
- `rfind(item)` — Returns the rightmost index where the substring item is found, or -1 if not found
- `index(item)` — Like find except causes a runtime error if item is not found
- `rindex(item)` — Like rfind except causes a runtime error if item is not found
- `split(`*`separator`*`)` — Return a list of the words in the string, using (optional) separator as the delimiter string
- `join(lst)` — Return a string which is the concatenation of the strings in lst
- `isalpha()` — Return True if all characters in the string are alphabetic and there is at least one character
- `isdigit()` — Return True if all characters in the string are decimal characters and there is at least one character
- `islower()` — Return True if all cased characters in the string are lowercase and there is at least one cased character
- `isspace()` — Return True if there are only whitespace characters in the string and there is at least one character
- `isupper()` — Return True if all cased characters in the string are uppercase and there is at least one cased character

List methods

- `append(item)` — Adds a new item to the end of a list
- `insert(position, item)` — Inserts a new item at the position given
- `extend(lst)` — Extend the list by appending all the items from lst
- `pop()` — Removes and returns the last item
- `pop(position)` — Removes and returns the item at position
- `sort()` — Modifies a list to be sorted
- `reverse()` — Modifies a list to be in reverse order
- `index(item)` — Returns the position of first occurrence of item
- `count(item)` — Returns the number of occurrences of item
- `remove(item)` — Removes the first occurrence of item
- `copy()` — Return a clone of the list
- `clear()` — Remove all items from the list