EECS 1015 Lab 3

Goal

- Be able to write a script that involve string
- Learn computational thinking

Tasks

1. Guide you through the process of planning to write a script using computational thinking
2. Learn to debug a script with bool
3. Learn to write a script with string and bool
4. Learn to write a script with string and bool
5. Learn to write a script with string
6. Learn to write a script with string
7. Learn to write a script with string

Total credit: 100 pts

Deadline: by the end of your lab

Cut-off date (Extended deadline): Sunday, October 6, 23:59:59

Therefore, no accommodations will be made if you do not meet the deadline. Furthermore, three lowest marks will be dropped.

You are very welcome to ask clarification questions to TAs. But please read the document carefully before you ask questions.

It is an academic offense to copy code from other students, provide code to other students, let other people (including the teaching staff) debug your code, or debug other students' code.

We will make check code similarities at the end of the semester.

Questions 3 – 5 may not be arranged by the difficulty level. You are highly recommended to take a look and decide the order of completing the questions. You will experience something similar and it is important to learn how to triage your tasks.

Task 1: Follow the Steps (30 pts)

In the previous lab, you learned about how to write a simple program. In this task, you will see a demonstration on how to design a more complicated script. This is your first encounter of computational thinking. It is a very important skill as our program gets more and more complicated.

Let's go through this example together. Suppose we define a new operation # between two natural numbers x # y such that it results in $x^2 - y^2$. For example:

$$2 \# 3 = 2^2 - 3^2 = 4 - 9 = -5$$

Our goal is to write a script that prompts the user to give us expressions in the a # b format and our script will calculate the value and print the result. For example:

```
Please provide an expression in the format of x # y: 2 # 3
-5
```

You can take a few minutes to think of how you would approach this problem, or even write some code yourself.

Now let's do it together. If you are able to finish the script yourself, awesome! But please still along as I will introduce computational thinking so that it will benefit you in the future if we have more complicated tasks. If you are not able to finish the script yourself, no worries! Let's see how to approach this problem in a more systematic manner.

So how do we write a script to achieve the functionality? Recall that one of the principles we mentioned in the first lecture is that we use building blocks to build systems. Let's break down the problem. Please note that you can use the same procedure for complicated problems in the future.

Recall that in the previous lab, your code usually has the following the structure:

- Take the input
- Do some calculation with the input
- Show the result

Similarly, our script, or most of the scripts, also follow this structure. The challenging part of this problem is how to further break down the calculation part. Pause here before you read on, and think about what steps are needed.

Now let's move on.

Let's use an example "2 # 3" and the steps needed are:

1. Find 2 and 3 in "2 # 3"
2. Calculate the result based on the definition of #

You are highly recommended to attempt the questions yourself before you look at the solution as a way to learn.

Q1.1 How would you break down the first step?

Solution:

In the string “2 # 3”, how we find the numbers is to see what are the numbers before and after the ‘#’. So here are the steps we need:

1. Find 2 and 3 in “2 # 3”
    a. Find where ‘#’ is
    b. The number before ‘#’ is 2
    c. The number after ‘#’ is 3
2. Calculate the result based on the definition of ‘#’

(Note: it is ok if you have something slightly different, as long as it makes sense.)

Now let’s put it all together, the structure of our script should look like if our example is “2 # 3”:

1. Take the input “2 # 3”
2. Do some calculation with the input
    - Find 2 and 3 in “2 # 3”
        - Find where ‘#’ is
        - The number before ‘#’ is 2
        - The number after ‘#’ is 3
    - Calculate the result based on the definition of ‘#’
        - $2 \# 3 = 2^2 – 3^2 = 4 – 9 = -5$
3. Show the result -5

Our goal here is to write a script that can handle any expressions in this format, not just “2 # 3”. So let’s generalize our structure:

1. Take the input in the format of “x # y”
2. Do some calculation with the input
    2.1 Find x and y in “x # y”
        i. Find where ‘#’ is
        ii. Find what the number is before ‘#’ and let it be x
        iii. Find what the number is before ‘#’ and let it be y
    2.2 Calculate the result based on the definition of ‘#’
        i. $x \# y = x^2 – y^2$
3. Show the result

Q1.2 Is there anything special about the example “2 # 3”?

Solution:

There is nothing so special about our specific example “2 # 3”, we do not need to make adjustment. Note that this may not be true to any arbitrary example, so you need to be careful. For example, when you implement the division operation, you may choose the example “4 / 2”. However, there are something special about this example that 4 can be perfectly divided by 2, unlike “5 / 2” which end up as a float.

Now let's convert our structure to Python code. We first write down the steps as comments in our Python script:

```
# 1.Take the input in the format of "x # y"

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is

# 2.1.2Find what the number is before '#' and let it be x

# 2.1.3Find what the number is before '#' and let it be y

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 - y2

# Show the result

```

Each line of comment shows the goal of the chunk of code below it. And now we just need to handle each mini problem. Let's work on each goal.

Q1.3 How to implement goal 1?

Solution:

This is very similar to what we did in the previous lab, and it is straightforward:

```
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is

# 2.1.2Find what the number is before '#' and let it be x

# 2.1.3Find what the number is before '#' and let it be y

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 - y2

# Show the result

```

Q1.4 How to implement goal 2.1.1?

Solution:

The data type of "x # y" is a string. You can use `help(str)` in the interactive mode to find if there is any method that is useful here. You may notice this method:

```
find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

It will give us the index of a single character or a string. This is what we need:

```
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is
pound_index = user_input.find('#')

# 2.1.2Find what the number is before '#' and let it be x

# 2.1.3Find what the number is before '#' and let it be y

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 - y2

# Show the result

```

For example, in the “2 # 3” example, it should give us 2 which is the index of ‘#’ in the string (Note: index start from 0, not 1!). You can try it in the interactive mode

```
>>> "2 # 3".find('#')
2
```

Q1.5 How to implement goal 2.1.2?

Solution:

Here we need the substring before the ‘#’ symbol, so we need to use string slicing

```
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is
pound_index = user_input.find('#')

# 2.1.2Find what the number is before '#' and let it be x
x = user_input[:pound_index]

# 2.1.3Find what the number is before '#' and let it be y

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 - y2

# Show the result

```

Note:

- We can use either `[:pound_index]` or `[0:pound_index]`. Recall that in class, we mentioned that if we omit the first one, it means the substring starts from the beginning.
- Here we use `x` as the variable name. We generally should not meaningless names such as `x, a, b, y, c,` etc. This is a special case because it mimics math formulas.

Q1.6 How to implement goal 2.1.3?

Solution:

(Note: there is actually a little bug in the code.)

```
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is
pound_index = user_input.find('#')

# 2.1.2Find what the number is before '#' and let it be x
x = user_input[:pound_index]

# 2.1.3Find what the number is before '#' and let it be y
y = user_input[pound_index:]

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 – y2

# Show the result

```

Q1.7 How to implement goal 2.1.3?

Solution:

(Note: this is another little bug in the code.)

```
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is
pound_index = user_input.find('#')

# 2.1.2Find what the number is before '#' and let it be x
x = user_input[:pound_index]

# 2.1.3Find what the number is before '#' and let it be y
y = user_input[pound_index:]

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 – y2
result = x * x - y * y

# Show the result

```

Q1.8 How to implement goal 2.1.3?

Solution:

```python
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is
pound_index = user_input.find('#')

# 2.1.2Find what the number is before '#' and let it be x
x = user_input[:pound_index]

# 2.1.3Find what the number is before '#' and let it be y
y = user_input[pound_index:]

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 – y2
result = x * x - y * y

# Show the result
print(result)
```

Let's run the script. Oops! We got an error message:

```
>>>  [evaluate lab3_task1_process.py]
     Please provide an expression in the format of x # y: 2 # 3
     Traceback (most recent call last):
       File "E:\Work_OneDriveSync\OneDrive - York University\course\EECS1015\2023-2024\Fall\Slides\Lec3\lab3_task1_process.py", line 17, in <module>
         result = x * x - y * y
     builtins.TypeError: can't multiply sequence by non-int of type 'str'
```

Q1.9 Fix this bug.

Solution:

There is a bug because we got a substring, but we need integers for the calculation.

```python
# 1.Take the input in the format of "x # y"
user_input = input("Please provide an expression in the format of x # y: ")

# 2.Do some calculation with the input
# 2.1 Find x and y in "x # y"
# 2.1.1 Find where '#' is
pound_index = user_input.find('#')

# 2.1.2Find what the number is before '#' and let it be x
x = int(user_input[:pound_index])

# 2.1.3Find what the number is before '#' and let it be y
y = int(user_input[pound_index:])

# 2.2 Calculate the result based on the definition of '#'
# which is x # y = x2 – y2
result = x * x - y * y

# Show the result
print(result)
```

Note on debugging:

- It is useful to show the line numbers as the interpreter will tell us exactly which line has the error and it is easier to locate the bug. In order to show the line number in the Wing IDE, go to edit -> Show Line Numbers.

- Sometimes you may want to change the lines other than where it reported the bugs, as what we did in the example above. The error was reported on line 17, but we change line 10 and line 13.

Let's run it again. We got another bug 😟:

```
>>> [evaluate lab3_task1_process.py]
Please provide an expression in the format of x # y: 2 # 3
Traceback (most recent call last):
  File "E:\Work_OneDriveSync\OneDrive - York University\course\EECS1015\2023-2024\Fall\Slides\Lec3\lab3_task1_process.py", line 13, in <module>
    y = int(user_input[pound_index:])
builtins.ValueError: invalid literal for int() with base 10: '# 3'
```

Q1.10 Fix this bug.

Solution:

```
1   # 1.Take the input in the format of "x # y"
2   user_input = input("Please provide an expression in the format of x # y: ")
3
4   # 2.Do some calculation with the input
5   # 2.1 Find x and y in "x # y"
6   # 2.1.1 Find where '#' is
7   pound_index = user_input.find('#')
8
9   # 2.1.2Find what the number is before '#' and let it be x
10  x = int(user_input[:pound_index])
11
12  # 2.1.3Find what the number is before '#' and let it be y
13  y = int(user_input[pound_index + 1:])
14
15  # 2.2 Calculate the result based on the definition of '#'
16  # which is x # y = x2 – y2
17  result = x * x - y * y
18
19  # Show the result
20  print(result)
```

This is a very typical bug when working with strings. You should be very careful about indices. For example, when using the example "2 # 3", the index of '#' is 2. So in "2 # 3"[2:], it is a substring that starts from the # (including #).

As seen in this example, your computer will execute whatever you implement. It is exactly the same as the making sandwich video showed in the first lecture. If you cannot fix the bug by looking at the code, use the debugger to help you! The instructions can be found in the previous lab.

Now let's run it again. YAY, it works!

```
>>> [evaluate lab3_task1_process.py]
Please provide an expression in the format of x # y: 2 # 3
-5
```

Here is a summary of what we did. We first break down the problem into smaller pieces. If it is a bit challenging to do so, you can always use a specific example to guide you and then generalize your steps. However, you need to be careful and think about whether this is anything special about your example. You can then treat each step as your sub-goals and solve each mini problem. Lastly, you may need to perform some debugging.

Submission

- Go to eClass -> our course -> Week 4 -> Practice -> Lab -> Lab 3 Submission

- Copy your code to Task 1
- You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need spend some time debugging!).

Rubrics:

- You will get full mark for this question if you submit the solution (with no typo) to the required location on Prairielearn before the deadline.

Note:

- since we provide solutions to this question, it is ok to submit it as is.
- In order for the autograder to work properly:
    - You must NOT change the order of the first few lines of taking input
    - You must NOT add more print statement when submit the code
    - You must NOT change the line with the print statement
    - The print statement should be the last line of your code

Task 2: Debugging (30 pts)

In this task, you will be provided with a potential script for a problem. However, this script does not work. You need to debug the code based on what you learned in task 1 and in the lecture to make it work.

Let's define a new logical operator `xor`. It can be used between two Boolean literals `a xor b`. It is true if and only if one of a and b is True. Here is the Truth table of this logical operator.

a	b	a xor b
True	True	False
True	False	True
False	True	True
False	False	False

The code to debug (lab3_task2.py) may have syntax errors, run-time errors, and semantic errors. To help with debugging, you can use the debugger to execute the code line by line (you should fix the syntax errors before doing this).

Note: You can use `bool` to turn the str '1' to `True`, and '' (empty string) to `False`. For example, bool('1') = True

Requirement

- Your submission must be based on the provided code. There are simpler ways of doing this, but you cannot delete all the provided code and write your own.
- You can only use logical operators (e.g., `and`, `or`, `not`), assignment, bool(), and print statement. You cannot use other statements/libraries, to name a few (It's ok if you don't know what they are):
    - You are not allowed to import other libraries
    - You are not allowed to use if-statement
    - You are not allowed to use comparison operators such as == or !=
    - You are not allowed to use arithmetic operations such as +, -, etc.
    - You are not allowed to use bitwise operations

Submission

Copy the Python code to Lab 3 -> Task 2 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need spend some time debugging!).

Note:

- In order for the autograder to work properly:
    - You must NOT change the order of the first few lines of taking input
    - You must NOT add more print statement when submit the code
    - You must NOT change the line with the print statement
    - The print statement should be the last line of your code

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 7.5 pts if your script correctly provide solution True xor True, which is input 1 for a and input 1 for b
    - You will receive 7.5 pts if your script correctly provide solution True xor False, which is input 1 for a and no input for b (press enter directly)
    - You will receive 7.5 pts if your script correctly provide solution False xor True, which no input for a (press enter directly) and input 1 for b.
    - You will receive 7.5 pts if your script correctly provide solution False xor False, , which no input for a (press enter directly) and no input for b (press enter directly)

Task 3: Implementation (10 pts)

In this task, you will implement a more complicated script yourself by expanding lab3_task3.py . It determines whether a child should buy a ticket to enter a park. A child under the age of 6 (including 6) does not need to buy a ticket, but if the child is taller than 120 cm (not including 120), the child should buy a ticket regardless of the age.

If a child needs to buy a ticket, your script should show True, otherwise show False.

Requirement

- You must not use if-statement (It's ok if you don't know what it is, as it indicates that it is very unlikely for you to use it)

Submission

Copy the Python code to Lab 3 -> Task 3 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need spend some time debugging!).

Note:

- In order for the autograder to work properly:
    - You must NOT change the order of the first few lines of taking input
    - You must NOT add more print statement when submit the code
    - You must NOT change the line with the print statement
    - The print statement should be the last line of your code

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 3 pts if your script correctly provide solution to our example (i.e., age = 3, height = 100, show False)
    - You will receive 2 pts if your script correctly provide solution to our example (i.e., age = 4, height = 146, show True)
    - You will receive 1.5 pts if your script correctly provide solution to another example (e.g., another case 1)
    - You will receive 1.5 pts if your script correctly provide solution to another example (e.g., another case 2)
    - You will receive 2 pts if your script correctly provide solution to an example with random input

Task 4: Implementation (10 pts)

In this task, you will implement a script yourself by expanding lab3_task4.py . It checks whether user input a valid phone number. A phone number should satisfy these two requirements:

- It must consist of digits, not alphabets or symbols.
- It must contain 10 digits, no more, no less.

However, the user may input a number with trailing spaces, which may appear at the beginning, or end of the phone number, or both. For example, “ 1234567890”, “1234567890 ”, “ 1234567890 ”.

Moreover, Users may also include the symbol ‘-’, ‘(‘, ‘)’ and spaces in the middle of their phone numbers, such as “(123) 456 - 0000” or “123 - 456 - 7890” or “123 456 7890”or “12- 3456-78-90” or “1)234(56 – 789(0” (although the last case looks weird, but in our case, we accept it for simplicity)

Your script should correctly identify those input are valid phone numbers (i.e., show True) and show False otherwise.

Hint:

- You may need various string methods, especially the ones in the puzzles.
- If the code does not work as you planned, you can use the debugger to figure out the problem.
- If you still cannot figure out the problem with a debugger, try to identify the problem and isolate/abstract the problem in a different script and ask for help from the TA.
- If you have trouble identifying the problem and/or isolating the problem, you can ask the TA how to do so. (Note: Your TA cannot debug the code for you)

Requirement

- You must not use the if-statement (It’s ok if you don’t know what it is, as it indicates that it is very unlikely for you to use it)

Submission

Copy the Python code to Lab 3 -> Task 4 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need spend some time debugging!).

Note:

- In order for the autograder to work properly:
  - You must NOT change the order of the first few lines of taking input
  - You must NOT add more print statement when submit the code
  - You must NOT change the line with the print statement
  - The print statement should be the last line of your code

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 1 pt if your script correctly provide solution to “ 1234567890”
    - You will receive 1 pt if your script correctly provide solution to “ 1234567890 ”
    - You will receive 1 pt if your script correctly provide solution to “(123) 456 - 0000”
    - You will receive 1 pt if your script correctly provide solution to “123 - 456 - 7890”
    - You will receive 1 pt if your script correctly provide solution to “e31235h62i”
    - You will receive 5 pts if your script correctly provide solution to other examples

Task 5: Implementation (4 pts)

In this task, you will write a script from scratch and no starter code is provided. It takes one's first name and last name, and print out the in the format of last name, comma, space, first name.

For example, my first name is *Meiying*, and last name is *Qin*. Then the script should print out:

Qin, Meiying

You script should take the following input from the user:

- First name
- Last name

Requirement

- The input must be in the order of first name, and then last name.
- Your code should not take any other input
- The print statement should be the last line of your code
- Make sure you just print the value, and no other texts
- You should remove other print statements

Submission

Copy the Python code to Lab 3 -> Task 5 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need to spend some time debugging!).

Rubrics

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 2 pts if your script correctly provide solution to our example (i.e., with the input listed above, your script should output Qin, Meiying)
- You will receive 1 pts if your script correctly provide solution to another example
- You will receive 1 pts if your script correctly provide solution to a slightly more challenging example

Task 6: Implementation (8 pts)

In this task, you will write a script from scratch and no starter code is provided. It takes one's name in the format of last name, comma, space, first name, and print out the last name.

For example, my first name is *Meiying*, and last name is *Qin*. Therefore, I should provide the script the following input:

Qin, Meiying

And the script will print out my last name:

Qin

Requirement

- The input must be in the format of last name, comma, one space, and first name and should take them as one input.
- Your code should not take any other input
- The print statement should be the last line of your code
- Make sure you just print the value, and no other texts
- You should remove other print statements

Submission

Copy the Python code to Lab 3 -> Task 6 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need to spend some time debugging!).

Rubrics

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 4 pts if your script correctly provide solution to our example (i.e., with the input listed above, your script should output Qin)
- You will receive 2 pts if your script correctly provide solution to a slightly more challenging example
- You will receive 2 pts if your script correctly provide solution to another slightly more challenging example

Task 7: Implementation (8 pts)

In this task, you will write a script from scratch and no starter code is provided. It takes one's name in the format of last name, comma, space, first name, and print out the first name.

For example, my first name is *Meiying*, and last name is *Qin*. Therefore, I should provide the script the following input:

Qin, Meiying

And the script will print out my first name:

Meiying

Requirement

- The input must be in the format of last name, comma, one space, and first name and should take them as one input.
- Your code should not take any other input
- The print statement should be the last line of your code
- Make sure you just print the value, and no other texts
- You should remove other print statements

Submission

Copy the Python code to Lab 3 -> Task 7 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need to spend some time debugging!).

Rubrics

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 4 pts if your script correctly provide solution to our example (i.e., with the input listed above, your script should output Meiying)
- You will receive 2 pts if your script correctly provide solution to a slightly more challenging example
- You will receive 2 pts if your script correctly provide solution to another slightly more challenging example