COM4013 – Introduction to Software Development
Week 5 – For Loops, Debugger, and Review

*Always select "**Enable Editing**" if prompted by Microsoft Word*

Lab Exercises

The exercises in this lab follow the topics in the lecture notes. Refer to the lecture notes for examples and guidance.

There are two "stages" marked in the lab sheet. Stage 1 is the *absolute minimum* point you should reach. Hopefully, you can reach it in the lab session. Reaching stage 2 suggests you are on target for a top-class mark for the module. Set your goals sensibly - do not just aim for the minimum or you may struggle to pass the module.

I use the term integer, string and float, and Boolean to infer what the input/output looks like. Unlike many other languages Python does not expect a data type before the variable names.

So if I ask that you declare an integer num1 to 1, then do as follows:

```
num1 = 1
```

For declaring a string and assigning the value hello

```
greeting = "hello"
```

For floats we can declare it like this

```
Money = 2.50
```

For Booleans (bool) we can declare it like this

```
isHeavy = True
```

Password Protection Program

- Create a new Jupyter Notebook project called **Lab 5 Exercises**. Refer to Week 1's lecture/worksheet/video if you have forgotten how to create a new project/file/program or use the shell code.
- Initialise a variable called attempts with a value of 3. This variable will keep track of how many attempts the user has left.

- Create a for loop that will run three times, using the range() construct. This will ensure that the program allows three password input attempts.
- Inside the loop, set the correct password as password = "password123".
- Prompt the user to enter their password and store their input in a variable called passwordInput.
- Check whether the passwordInput is equal to the correct password.
- If the passwordInput is not correct, decrement the attempts variable by 1.
    - If the user has no attempts left, then print a message indicating that the user is locked out of the system and should contact the administrator for more information.
    - Else print the number of attempts they have left.
- If the passwordInput is correct, print a message indicating that the user has successfully logged on.
    - If the for loop is incorrectly written then you may find that the user is asked for a password input again (fix you for loop to only allow 3 loops, the start is inclusive, and the end is exclusive).
    - You MUST also break out of the loop at this point as it is likely that you still have some attempts left

Improved Password Protection Program (optional)

- As an **OPTIONAL TASK** *(do not ask me about this)* - First Create a new cell and copy the code from above into it.
- you can improve the logic of your password protection program by editing the for loop. Instead of using the attempts variable, we can use the counting variable in the for loop itself to track the attempts. This change will also involve counting backward.

Nested 'for' Loops

- Create a new cell in Jupyter Notebook (use the shell code).
- Try to do these on your own first...
- Write a 'for' loop that displays 10 X's in a row. **The code to do this is shown in the lecture notes**.

```
XXXXXXXXXX
```

♦ Add a newline print (in the same cell), ***nest*** this 'for' loop in another 'for' loop to display a 10x20 box of X's. **Again, this is shown in the lecture notes**.

Note: You must use nested for loops in all these exercises. ***Do not try to solve these problems with many repeated print() statements!***

```
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
```

- Add a newline print (in the same cell), change the size of the box so it is 10 high and 5 wide:

```
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
```

- Add a newline print (in the same cell) and try to draw different shapes with 'for' loops. Copy and update your code to show an L shape as shown below 4*15.
- This is easy when you see the shape is actually made of two different sized boxes. So, you need two copies of the code you have already with different dimensions. The boxes are shown in different colours below (we won't show colour in the print though), count the X's and make sure you get the box sizes correct:

```
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
```

♦ Add a newline print (in the same cell). Now change your program to draw a T shape. This has the same two boxes swapped around. Also, the tall thin box is pushed to the right. Do this by displaying extra spaces before each line for this box. Think carefully where to put that code, but also experiment - you will learn by trying out different ideas:

```
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
      XXXXX
```

NestedNumberGrid

- Create a new cell in Jupyter Notebook (use the shell code).
- Write with the code to draw a box of X's, 7 wide and 5 high (copy one of your nested loops from the previous exercise).

Next, we want to update the code to display a grid of numbers as shown below 7*5. There should be a space between each number and a blank line between each row.
The next few steps will give you instructions on how to do this.

```
01 02 03 04 05 06 07
08 09 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
```

♦ To draw the number grid, we need to replace each "X" with a number. Also, the numbers must have two digits, small numbers like 7 need to be displayed 07.

- Start by replacing the "X " in your print statement with your *inner* loop variable. So, for example, if your inner loop uses a variable called j, then display j instead of "X." (remember you don't use quotation marks to display variables)

```
Print(j, end='')
```

- Now we can make sure that each number has two digits with some special Python formatting. You may have seen this if you reached the later stages of last week's lab - the `0:02` part makes sure numbers are displayed with at least 2 digits. Replace the line you just wrote with this, *but make sure you use the correct variable name for your program*:

```
print("{0:02}  ".format(j), end='')
```

You should see a result like this:

```
01 02 03 04 05 06 07
01 02 03 04 05 06 07
...
```

This is close, but not quite what we want. The number should not reset back to 01 on each row, but that is what the loop variable is doing. The loops are working OK, so we don't want to change their variables. So, we need another variable:

- Declare a new integer at the start of your program and initialise it to 1. Inside the *inner* loop, increment this variable. Display this variable in your print statement instead of the loop variable. You should get the correct result:

```
01 02 03 04 05 06 07
08 09 10 11 12 13 14
15 16 17 18 19 20 21
...
```

When writing loops, you can use the loop variables themselves or you can declare new variables that work independently. Exactly what you need depends on the problem you're trying to solve.

Virtually every loop you write will need to use variables, but always think carefully about the correct variable to use, or if you need to create a new variable.

You now have a choice of tasks:

- There are some **Practice** questions below, which reinforce the key programming skills you should have gained so far. Practice is vital to learn programming, so this section is recommended. Solutions are provided so you can check your progress.

- However, if you feel very comfortable with your skills so far, then you may skip the practice questions and try the **Advanced** tasks later in the worksheet.

Practice

The solutions to these questions may at some future junctures be provided on the module webpage so you can check your work. **Attempt the exercises before looking at the answers or you will gain no skill (I would have released them in advance if I didn't think most of you would peek)**.

♦ Create a new cell in Jupyter Notebook (use the shell code).

You don't need to create a new project for every answer, you can put multiple answers in the same project. Create a new project if you feel your current project is getting too full though. Separate each answer with these lines (same as a previous worksheet);

```
print('\n')
```

```
Or we can comment the code out by highlighting earlier sections and
pressing ctrl+/
```

Practice 1
Follow these simple instructions exactly and when you are finished your code should match the solution *precisely, line for line*. This tests whether you understand the programming terms used:

Declare an integer called fileCount and initialise it to 0.

On the next line declare an integer called fileSize but do not initialise it (you will have to assign it a value though - N**e).

On the next line using the shorthand operators, increment fileCount

On the next line assign 1000 to fileSize

On the final line display a message in this format (using the fileCount and fileSize variables to display the numbers in red). Your output should be *identical* except for the colours:

```
The number of files is 1, the file size is [1000].
```

Practice 2
Ask the user for their age. Display a different message depending on whether they are a child, adult or pensioner. You can decide the age ranges.

Add some input validation to the integer value to prevent a ValueError.

Practice 3
Write a number guessing game program. You can use the word guessing game from the week 3 and 4 lectures as a basis for your code:

The program selects a number - you can just choose a number and type it in the program (e.g., 231), or better you can get a random number using the method shown:

```
import random # You can use this for your assignment


# Generate a random integer between a specified range (inclusive)
randomNumber = random.randint(1, 100)
# You don't need this... It's just here to print these numbers...
print("Random integer:", random_number)
```

The user guesses this number. If they are wrong the program say so *and tells them whether the correct number is higher or lower than their guess.*

The guessing game continues until the user guesses correctly - a winning message is displayed.

You may as well ensure that the morons who are using your programme can't break it by entering an emoji etc. Gracefully handle the errors.

Practice 4
Use a for loop to display the 12 "times-table" up to 12 x 12:

```
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
...
```

A little tricky: Update your program to show all the "times-tables" from 1 to 12. You will need to nest your loop.

Practice 5
Display a menu with 5 choices (you decide what the choices are...). Ask the user for their choice that must be in the range 1-5. Fully validate their answer:
If they type a choice outside the valid range, or if they type something that isn't a number (like words), then an error message is displayed, and they are prompted to enter their choice again.

Also, for cleanliness, use the `clear_output()` function so that we do not have multiple copies of the menu. Use the library below to get the clear_ouput function (it works in Jupyter – I do not know if it works elsewhere).

You may wish to place it before the print statement in your except clause as that will ensure that you gave the neatest solution.

```
from IPython.display import clear_output
```

Advanced Tasks

Calendar Program - To Completion (You must complete the Calendar task from the last Practical worksheet to do this task)

Now, we attempt a standard calendar display. First try to make this output:

```
Enter a year: 2011

January
Mon Tue Wed Thu Fri Sat Sun
01  02  03  04  05  06  07
08  09  10  11  12  13  14
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31

February
Mon Tue Wed Thu Fri Sat Sun
01  02  03  04  05  06  07
...
```

We'll first assume that all months start on a Monday to make the first step simpler.

Let’s add a print statement below the variable to get the name of the current month:

```
print("Mon Tue Wed Thu Fri Sat Sun")
```

When we print we should see the following

```
Enter a year: 2011

January
Mon Tue Wed Thu Fri Sat Sun
```

It might be tempting to use another nested loop for each row of the output. However, that's not the simplest way. Instead create a new integer variable `weekDay` that calculates the start day of the month (this must be placed after the print statement from above in the month for loop).

```
# Calculate the starting day of the month (0 for Sunday, 1 for
Monday, etc.)
weekDay = datetime(year, month, 1).weekday()
```

Remove the current code in your day loop and switch with the following:

```
# Print day with leading 0 if it's a single-digit number
print(f" {day:02} ", end='')  # Same format as before
weekDay = (weekDay + 1) % 7  # Update the day of the week
```

Inside the day loop test 'if' the weekDay variable is equal to 7 then do an extra `print()`. This creates the grid-like effect and makes the next stage simpler.

```
Enter a year: 1996
January

Mon Tue Wed Thu Fri Sat Sun
 01  02  03  04  05  06  07
 08  09  10  11  12  13  14
 15  16  17  18  19  20  21
 22  23  24  25  26  27  28
 29  30  31
```

As you can see from above, we formatted the numbers a little: we need an extra 0 at the start of the single digit numbers. This can be done easily with Write / WriteLine:

```
print("{day:02}  ", end='')
```

The extra `:02` indicates to pad the output with 0's to at least two digits.

Final Steps…

Finally, before outputting the days for each month we can output some `--` characters to display any days missing. As we already know the starting day of the month (`weekDay` variable), we can write a for loop to print the following `--` characters as many times is required.

The final output should be (if your calendar does not look like this then there is something wrong with the for loop):

```
Enter a year: 2023
January

Mon Tue Wed Thu Fri Sat Sun
 --  --  --  --  --  --  01
 02  03  04  05  06  07  08
 09  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31
```

Make sure today's date has the correct day!

NOTE: You would be correct in thinking that I have given you most of the code to create this calendar format. I thought it would be nice for you to have something tangible to play around with in your own time.

My suggestion is that you build a simplistic calendar application using the Python PyQT library as it will enable you to graphically represent this code.

Draw more Shapes

Tasks for those who want to stretch themselves. This material is not necessary to pass the module.

Use nested for loops to draw these shapes. Keep the code for each shape in one cell so that your final cell draws all three shapes.

1)

```
X
XX
XXX
XXXX
XXXXX
XXXXXX
XXXXXXX
XXXXXXXX
XXXXXXXXX
XXXXXXXXXX
```

2)

```
         X
        XXX
       XXXXX
      XXXXXXX
     XXXXXXXXX
   XXXXXXXXXXX
  XXXXXXXXXXXXX
XXXXXXXXXXXXXXX
```

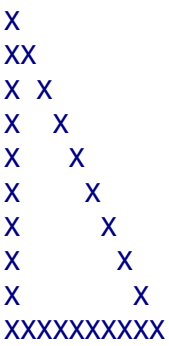
3) This is very difficult – You will need one outer loop and 3 inner loops (frankly I'd skip this as it is VERY difficult). But I thought I'd put it in here for Ollie.

```
XXXX           XXXX
 XXXX         XXXX
  XXXX       XXXX
   XXXX     XXXX
    XXXXXXXXX
```

At the start of your program allow the user to input the height of the shapes as an integer. Then update your code so each shape is drawn as the desired size. So, for example, the first triangle above is currently height 10, the second triangle is height 8. After this code change, all the shapes will be the same height - the loops extended to match the user's input.

Outlines Only

Change shape 1 and 2 drawings to show the outline only, for example:



THAT IS ALL FOR NOW

Submission

1. Upload your work to the GitHub classroom - Here
   a. Classroom Link for Week 5