Tutorial Week 2

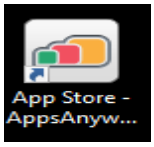
Pre-reading/viewing: Lecture 1, Lecture 2, document '**Software required for this module'**, document '**Using AppsAnywhere guid**e'

The tutorial covers the following.

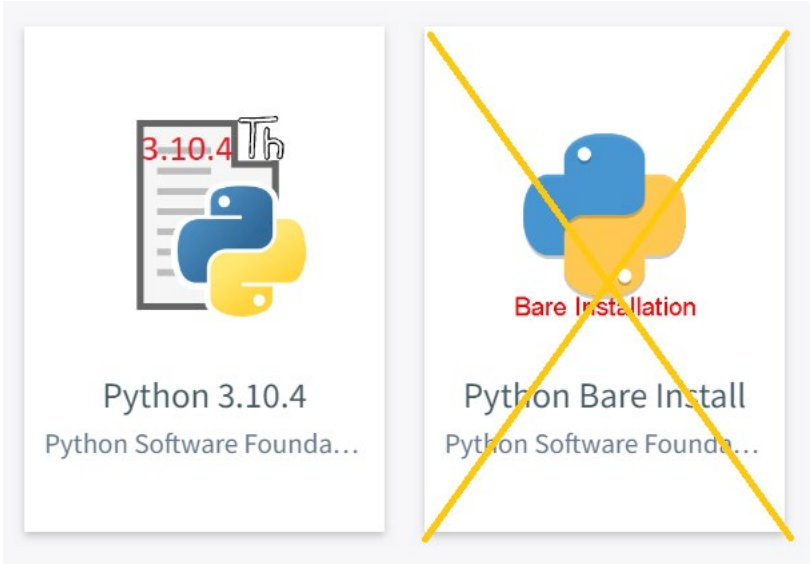
1. Launching Python/IDLE on the lab PCs.
2. Experimenting with the Python Shell
3. Running programs in Script Mode - Arithmetic Operators & printing strings
4. Keyboard input
5. Practice debugging programs errors
6. Exception Handling in Python

1: Launching Python/IDLE on the lab PCs

- The lab PC's have Python 3.10.4 (includes IDLE) available via AppsAnywhere. When logged in, click on the AppsAnywhere icon:



- Then scroll down or type 'IDLE' or 'Python' in the **Search Apps** box.
- Launch version **Python 3.10.4**



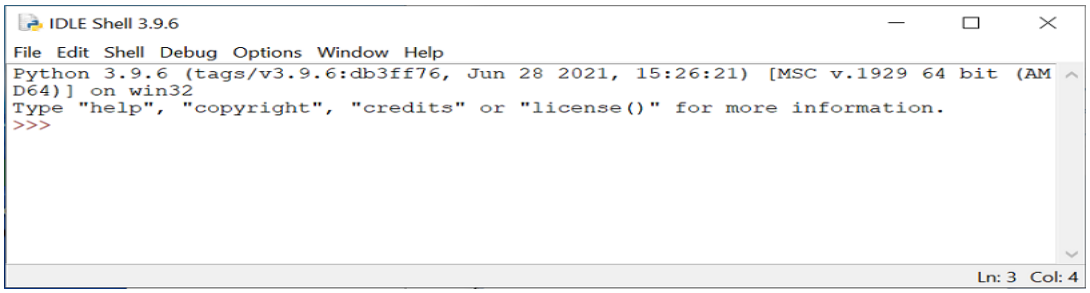
- See document '**Using AppsAnywhere guide**' for further details.

Accessing Python from home.

- See 'Guide to Using AppsAnywhere on Your Personal Computer'.
- Mac Users - AppsAnywhere does not work with Macs.
    - Download Python from: https://www.python.org/downloads/
    - Any Python 3.10.x (or Python 3.9) version will be suitable.
        - Or, for the first tutorial use an 'online Python 3 compiler'. E.g., https://www.programiz.com/python-programming/online-compiler/

2: Experimenting with the Python Shell

We can use Python in two modes - Python Shell and Script Mode. We will experiment with using the python shell. When you launch Python the Python Shell window will display as follows.



- The **Shell window** allows you to type Python code at the prompt string and see the result immediately. It is known as **interactive mode**.
- >>> is the prompt string.
- In the Python shell, we could enter the following calculations line by line and hit enter to get the result.

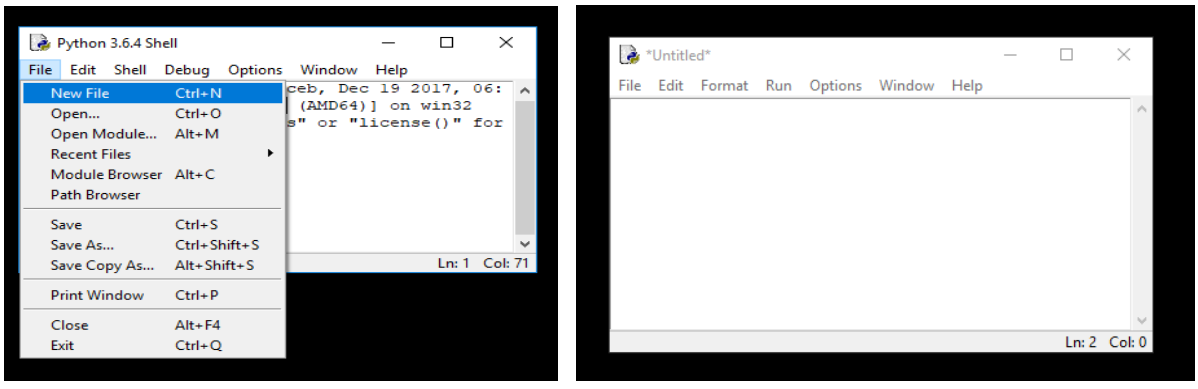
```
>>> 88 + 4
92
>>> 45 * 4
180
>>> 17 / 3
5.666666666666667
```

- Try the above in the Python shell.
- Python Shell is good for testing small amounts of code however the statements are not saved anywhere. If you want to execute a set of statements multiple times use script mode.

3: Run programs in Script Mode

The interactive Shell window allows us to open a program editing window.

- From the Python Shell window, select New File from the File menu. You will see a window entitled "Untitled".



- From the **File** menu, select **Save As**, and select a folder to save your Python program file.
  - On the University PCs you need to **save to your H drive.** In the **File name:** text box, type: test1.py. Then click on the **Save** button.
- You will then see a blank editor window ready for you to type in the following Python programs.

a) Arithmetic Operators
- Write down the output of the following program.
- Then create, save and run the program.
- Check that you get the right values for all of them?

```
num = 5
print(num)          #Output?
num = num + 2
print(num)          #Output?
num = num // 3 * 6
print(num)          #Output?
print(7 + 15 % 4)   #Output?
num = 24 // 3 // 4
print(num)          #Output?
```

b) Printing strings

i. Create, save and run the following program.

```
a = "Hello out there."
print(a)
b = 'Hello again.'
print(b)
```

Modify the program so that it concatenates (joins) variables a and b and prints the result.

ii. Why would the following produce an error?

```
total = 10
greet =  'Hello'
both = total + greet
print(both)
```

- Note that you cannot concatenate a string and an integer.
- Use the built-in str() function (converts to string) to fix the error.

iii. What will be displayed by the following code? Type and run the code to check your answer.

```
print("A", end = ' ')
print("B", end = ' ')
print("C", end = ' ')
print("D", end = ' ')
```

iv. What is the output of the following program? Test the program to check your answer.

```
a = '10'
b = '99'
c = a + b
print(c)
```

- Modify the above program so that it prints out the value **109**

4: Keyboard input

i. Try the following program:

```
name = input('Please enter your name: \n')
print('Hello', name)
```

- Remember, \n within quote marks forces a new line to be printed.
- When you run the program, you should see the message "Please enter your name:" in the shell window. Type in your name, followed by the ENTER key. The program will greet you.

ii. Extend the program to get the user's age, and print out "your age is" (with the response).

iii. How would you print the following so that it displays as shown: test\test2\answers.txt

iv. Type the following. Check the differences between the three print statements.

```
the_text = input('Enter some text.\n')    # get some text!

#print - version 1
print('This is what you entered: ')
print(the_text)

#print – version 2
print('This is what you entered:', the_text)

#print - version 3 - Supress printing of a new line, use end=' '
print('This is what you entered:', end=' ')
print(the_text)
```

v. **Lecture 2 Self-Check Questions (2-4)**

Type and run programs discussed in the Lecture 2. Partial possible solutions are shown.

- **Lecture: Question 2.** Write the Python program using the pseudocode shown.

  INPUT num_1
  INPUT num_2
  total <- num_1 + num_2
  PRINT total

  Partial solution shown. **Update** to convert the string input to an integer.

```
num_1 = input('Enter number one: ')
num_2 = input('Enter number two: ')
total = num_1 + num_2
print(total)
```

- **Lecture: Question 3.** Write the Python program using the pseudocode shown.

  INPUT cost_of_item
  INPUT cash_paid (e.g., 10 for £10)
  CALCULATE change
  PRINT change

  Partial solution shown. **Update** to convert the string input to a float.

```
cost = input('Enter cost of item: ')
paid = input('Enter cash paid: ')
change = paid - cost
print('Change is: ', change)
```

- **Lecture: Question 4.** Program to calculate the average of three numbers (pseudocode):
  INPUT num_1
  INPUT num_2
  INPUT num_3
  average <- (num_1 + num_2 + num_3) / 3
  PRINT average

  - Test your Python solution.
  - Then experiment with **round(x,y).** Where **'x'** is what you are rounding and 'y' is how many decimal places you want to round up to. E.g., average = round(average, 2)

5: Practice debugging program errors

- This program adds the number of apples and oranges already available with the number bought.
- There are **four bugs in the code** for you to correct. The debugged program should print:

```
You have:     3 apples and 0 oranges
you buy:      4 apples and 6 oranges
you now have: 7 apples and 6 oranges
```

```
# shopping program with 4 bugs
a=3         # number of apples
o=0.0       # number of oranges
print('you have:    ',a,'oranges and',o,'apples')
buy_a=4     # apples you buy
buy_o=6     # oranges you buy
print('you buy:     ',buy_a,'apples and',buy_o,'oranges')
a=a+buy_A
o=o+buy_o
PRINT('you now have:',a,'apples and',o,'oranges')
```

Tips on debugging your program errors.

- **Syntax error** - check for missing parentheses, quotation, or commas etc.
- **Runtime error**
  - Understanding the error message will help you.
  - Using extra print() statements to display the value of your program's variables can help you figure out what is happening in your program.
  - Function type() will return a variable type – useful for debugging **TypeError.**
  - Sometimes you need to work backwards (up) from the area indicated in the error message.
- **Semantic error** - The program will run without error messages, but it will not do the right thing.

Additional Exercises

vi. Temperature Program (part 1) – To extend in a later tutorial. Write a program that will convert an input Centigrade temperature (c) into Fahrenheit (f). Formula: `f = (c * 1.8) + 32`

Pseudocode:

INPUT centigrade
CALCULATE fahrenheit
PRINT fahrenheit

h) Write a program to calculate the volume of a box. Enter values for the length, height and width.

6. Exception Handling in Python

a) Example: ask the user to enter a number using the input() method and cast the string input to an integer.

```
n = int(input("Please enter a number: "))
```

Type in the code, run it and enter 10.5 instead of a valid integer. If the input entered cannot be converted (cast) to an integer it will generate a ValueError.

ValueError: invalid literal for int() with base 10: '10.5'

With exception handling, we can write robust code for reading an integer from input:

```
n = input("Please enter an integer: ")
try:
    n = int(n)
    print(n)
except ValueError:
    print("Requires a valid integer!")
```

Example: The program accepts a value from the user. In the *try* clause the value in *n* is converted (cast) to an integer. If an exception occurs (the value cannot be converted to an integer) the *except* clause will be executed. The error, in this case a ValueError, has to match the name after except.

b) The following calculation would produce a **ZeroDivisionError**: `x = 45 / 0`

Create a specific try except construct that will give the message 'Cannot divide by zero' when appropriate.