



Python

Eng.Amira Fouda

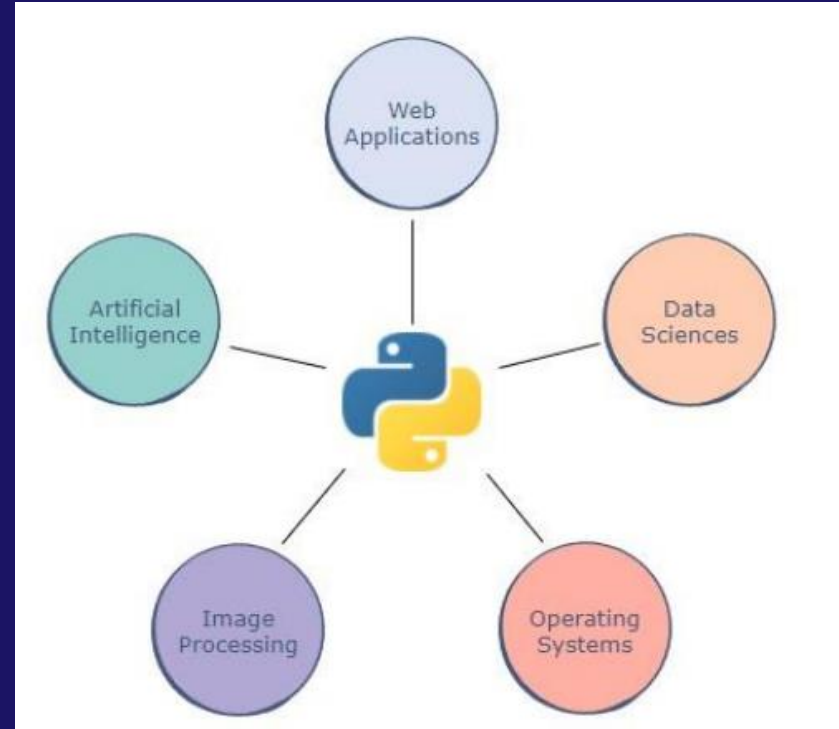
CONTENTS OF Session

- Intro to python
- Anaconda and virtual environments
- Jupyter
- Python basics syntax, comments, variables
- data types
- Print statement
- User inputs
- Strings, String methods
- Strings (cont)
- Operators (Arithmetic, Logical ...)
- List, tuple, set and Dictionary
- Control statement If Else
- Loops
- Functions
- Scopes
- Try & Except
- File Handling
- Pre-built & User Defined Modules

What is Python ?

Python is an interpreted, object- oriented, high-level and one of the most popular general-purpose programming languages in modern times.

The term “general-purpose” simply means that Python can be used for a variety of applications and does not focus on any one aspect of programming.



INTERPRETED VS. COMPILED

Compiled Language	Interpreted language
Needs an initial buildstep to convert code to machine code that CPU understands	Doesn't need a build step but it is run line by line and interpreted on the go.
Faster in execution.	Slower in execution.
More control over the hardware as memory management.	Less control over the hardware.
Examples: C++, C, C#	Examples: Python, JavaScript, Perl

Why python for Machine Learning ?

- Python is easy to understand you can read it for yourself. Its readability, non-complexity, and ability for fast prototyping make it a popular language among developers and programmers around the world.
- Python allows easy and powerful implementation With other programming languages, coding beginners or students need to familiarize themselves with the language first before being able to use it for ML or AI. This is not the case with Python. Even if you only have basic knowledge of the Python language, you can already use it for Machine Learning because of the huge amount of libraries, resources, and tools available for you.

Why python for Machine Learning ?

- Python has a great library ecosystem Having access to various libraries allows developers to perform complex tasks without the need to rewrite many code lines. Since machine learning heavily relies on mathematical optimization, probability and statistics, Python libraries help data scientists perform various studies easily.
- Huge community Python has a strong group of supporters, and it's worth knowing that if you experience a problem, there is someone out there who'll be able to offer you a helping hand.

Python installation (Windows OS)

➤ Steps:

- Go to www.python.org on your browser.
- Navigate to the “downloads” tab and click python for (Windows) OS .
- Click on “Latest Python 3 Release - Python 3.x.x”.
- Choose “Windows installer (64-bit)” at the end of the webpage
- Navigate to the downloaded files and open the installer
- It's a must to check the red box to add the python Path to the Environment variables of your system.
- In the “Advanced Options” section, check “Install for all users” option for multi users' access.

What is Anaconda ?

- Anaconda is a free and open-source distribution of the programming languages Python and R that comes with the Python interpreter and various packages related to machine learning and data science.
- Basically, the idea behind Anaconda is to make it easy for people interested in those fields to install all (or most) of the packages needed with a single installation.
- As it is managed by An open-source package and environment management system called Conda, which makes it easy to install/update packages and create/load environments.
- Anaconda also comes with several other software additions like the Anaconda Navigator, a graphical user interface.

Installing Anaconda

➤ Steps:

- Go to www.anaconda.com on your browser.
- Navigate to the “Products” tab and click “individual edition”.
- Click on “download” to download the windows installer.
- Navigate to the downloaded files open the installer.

➤ Additional Notes:

- Make sure the you have sufficient free disk space on the drive you are installing in.
- Check add Anaconda in my PATH environment variables in advanced options section.

Python Basics

➤ Print Statements:

- Since Python is one of the most readable languages out there, we can print data on the terminal by simply using the print statement.
- The text Hello World is bounded by quotation marks because it is a string or a group of characters
- Next, we'll print a few numbers. Each call to print moves the output to a new line

```
print("Hello World")
```

Hello World

```
print(50)  
print(1000)  
print(3.142)
```

50
1000
3.142

Python Basics

➤ Print Statements:

- We can even print multiple things in a single print command; we just have to separate them using commas
- By default, each print statement prints text in a new line. If we want multiple print statements to print in the same line, we can use the following code
- The value of end is appended to the output and the next print will continue from here

```
print(50, 1000, 3.142, "Hello World")
```

```
50 1000 3.142 Hello World
```

```
print("Hello", end="")  
print("World")
```

```
print("Hello", end=" ")  
print("World")
```

```
HelloWorld  
Hello World
```

Quiz:

- How can we print the text, "Array_Data_Science_Diploma" in Python?
 - `print Amit_Data_Science_Diploma`
 - `Print " Amit_Data_Science_Diploma"`
 - `Print("Array_Data_Science_Diploma")`
 - `Print(Amit_Data_Science_Diploma)`

Quiz:

- How can we print the text, "Array_Data_Science_Diploma" in Python?
 - `print Amit_Data_Science_Diploma`
 - `Print " Amit_Data_Science_Diploma"`
 - `Print("Array_Data_Science_Diploma")`
 - `Print(Amit_Data_Science_Diploma)`

Python Basics

➤ Comments:

- Single comment: #
- Multiline comment/docstrings: between three double/single quotes `""" """`. Or you can just write # multiple times.
- You can use Ctrl+ "/" to automatically comment a whole section.

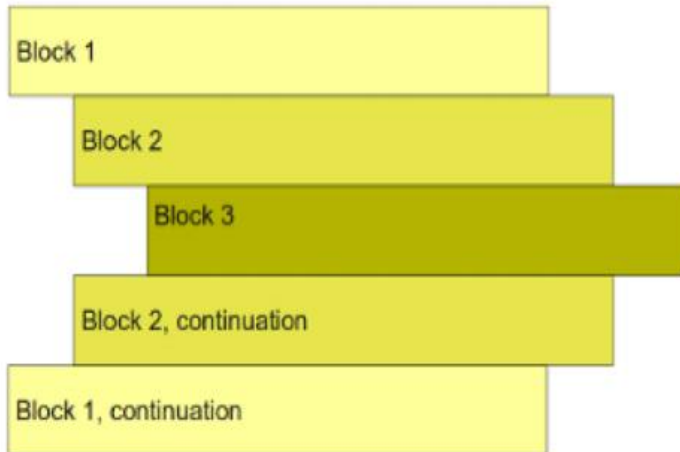
```
1 """ Docstrings are pretty cool
2 for writing longer comments
3 or notes about the code"""
4
```

```
1 print(50) # This line prints 50
2 print("Hello World") # This line prints Hello World
3
4 # This is just a comment hanging out on its own!
5
6 # For multi-line comments, we must
7 # add the hashtag symbol
8 # each time
9
```

Python Basics

➤ Indentation:

- Indentation is Important in Python. It Illustrate how code blocks are formatted.



```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
    print ("False")
```

error

Quiz:

- What will be printed by the following code?
- One
Two
Three
 - One
Three
 - Two
 - three

```
# print ("One")  
print("Two")  
# Three
```


Quiz:

- What will be printed by the following code?
 - One
Two
Three
 - One
Three
 - Two
 - three

Variables:

- A variable is simply a name to which a value can be assigned.
- Variables allow us to give meaningful names to data.
- The simplest way to assign a value to a variable is through the = operator.
- A big advantage of variables is that they allow us to store data so that we can use it later to perform operations in the code.
- Variables are mutable. Hence, the value of a variable can always be updated or replaced.

```
counter = 100          # An integer assignment
miles   = 1000.0        # A floating point
name    = "John"       # A string
z = None               # A null value
```

PYTHON STATEMENTS AND VARIABLES

Python statements means a logical line of code that can be either expression or assignment statement.

Expression means a sequence of numbers, strings, operators and objects that logically can be valid for executing.

Simple assignment statement means a statement with its R.H.S just a value-based expression or a variable or an operation.

Augmented assignment statement means a statement where the arithmetic operator is combined in the assignment.

Notes:

- A statement can be written in multi-lines by using \ character.
- Multiple statements can be written in same line with ; separator.

Variables:

- Variables can change type, simply by assigning them a new value of a different type.

```
x = 1  
x = "string value"
```

- Python allows you to assign a single value to several variables simultaneously.

```
a = b = c = 1
```

- You can also assign multiple objects to multiple variables.

```
a, b, c = 1, 2, "john"
```

- Variables are Case- Sensitive: x is different from X.

PYTHON IDENTIFIERS

Python identifiers means the user-defined name that is being given to anything in your code as the variables, function, class or any other object in your code. Guidelines for creating a python identifiers:

- 1- Use any sequence of lower case (a - z) or upper case (A - Z) letters in addition to numbers (0 - 9) and underscores (_).
- 2- No special characters or operators are allowed. (,), [,], #, \$, %, !, @, ~, &, +, =, -, /, \.
- 3 - Don't start your identifier with a number as it is not valid.
- 4- Some keywords are reserved as False, True, def, del, if, for, raise, return, None, except, lambda, with, while, try, class, continue, as, assert, elif, else, is, in, import, not, from, global, pass, finally and yield. You can't name an identifier with these words on their own however you can use them as sub-name such as True_stat or def_cat
- 5 - Make the user-defined name meaningful.

PYTHON IDENTIFIERS

6 - Python is case sensitive language so variable1 identifier is different from Variable1 identifier.

Identifiers or user-defined names has a convention in python which is as follows:

- Never use l “lower-case el” or I “Upper-case eye” or O “Upper-case Oh” as single character variable name as in some fonts they are indistinguishable

Variables:

- Do not need type definition when creating a variable. In C Language for example, to declare a variable:

```
> int x = 4
```

In python, no need for declaring the type 'int'

```
> x = 4
```

It knows automatically that x is an integer.

PYTHON DATA TYPES

Data type means the format that decides the shape and bounds of the data. In python, we don't have to explicitly predefine the variable data type but it is a dynamic typing technique. Dynamic typing means that the interpreter knows the data type of the variable at the runtime from the syntax itself.

The data types in python can be classified into:

- | | |
|------------------|--------------|
| 1 - Numbers | 2 - Booleans |
| 3 - Strings | 4 - Bytes |
| 5 - Lists | 6 - Arrays |
| 7 - Tuples | 8 - Sets |
| 9 - Dictionaries | |

PYTHON DATA TYPES

Any variable in python carries an instance of object which can be mutable or immutable. When an object is created it takes a unique object id that can be check by passing the variable to the built-in function `id()` . The type of the object is defined at runtime as mentioned before.

Immutable objects can't be changed after it is created as `int`, `float`, `bool`, `string`, `Unicode`, `tuple`.

Mutable objects can be changed after it is created as `list`, `dict`, `set` and user-defined classes.

Numbers :

- Python is one of the most powerful languages when it comes to manipulating numerical data.
- There are three built-in data types for numbers in Python:
 - Integer (int)
 - Floating-point numbers (float)
 - Complex numbers: <real part> + <imaginary part>j (not used much in Python programming)

Numbers :

- The integer data type is comprised of all the positive and negative whole numbers.
- Floating-point numbers, or floats, refer to positive and negative decimal numbers.
- Python allows us to create decimals up to a very high decimal place.
- This ensures accurate computations for precise values.
- Python also supports complex numbers, or numbers made up of a real and an imaginary part.

```
1 print(10) # A positive integer
2 print(-3000) # A negative integer
3
4 num = 123456789 # Assigning an integer to a variable
5 print(num)
6 num = -16000 # Assigning a new integer
7 print(num)
```

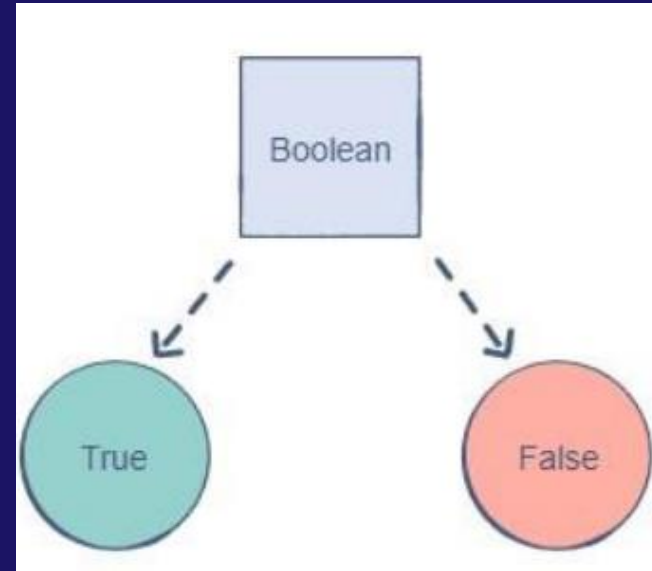
```
print(complex(10, 20)) # Represents the complex number (10 + 20j)
print(complex(2.5, -18.2)) # Represents the complex number (2.5 - 18.2j)
```

Booleans:

- The Boolean (also known as bool) data type allows us to choose between two values: true and false.
- In Python, we can simply use True or False to represent a bool

Note: The first letter of a bool needs to be capitalized in Python.

- A Boolean is used to determine whether the logic of an expression or a comparison is correct. It plays a huge role in data comparisons.



Common Number Functions:

Function	Description
int(x)	to convert x to an integer
float(x)	to convert x to a floating-point number
abs(x)	The absolute value of x
cmp(x,y)	-1 if $x < y$, 0 if $x == y$, or 1 if $x > y$
sqrt(x)	The square root of x for $x > 0$
log(x)	The natural logarithm of x, for $x > 0$
pow(x,y)	The value of $x^{**}y$

Strings:

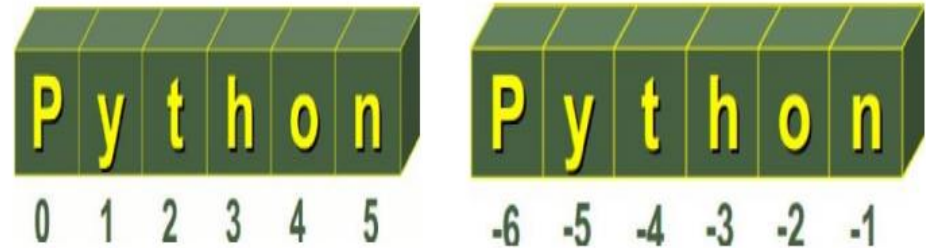
- A string is a collection of characters closed within single or double quotation marks (immutable).
- You can update an existing string by (re)assigning a variable to another string.
- Python does not support a character type; these are treated as strings of length one.

```
>>> str= "strings are immutable!"
>>> str[0]="S"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Strings:

- Python accepts single ('), double (") and triple (''' or ''') quotes to denote string literals.
- String indexes starting at 0 in the beginning of the string and working their way from -1 at the end

```
name1 = "sample string"  
name2 = 'another sample string'  
name3 = """a multiline  
string example"""
```



Strings Indexing:

You can index a string (access a character) by using square brackets:

```
1  batman = "Bruce Wayne"
2
3  first = batman[0]  # Accessing the first character
4  print(first)
5
6  space = batman[5]  # Accessing the character at index 5
7  print(space)
8
```

```
1  batman = "Bruce Wayne"
2  print(batman[-1])  # Corresponds to batman[10]
3  print(batman[-5])  # Corresponds to batman[6]
```


Strings Slicing:

- Slicing is the process of obtaining a portion (substring) of a string by using its indices. Given a string, we can use the following template to slice it and obtain a substring `String [start:end]`
 - start is the index from where we want the substring to start.
 - end is the index where we want our substring to end.
- The character at the end index in the string, **will not be** included in the substring obtained through this method.

```
1 my_string = "This is MY string!"
2 print(my_string[0:4]) # From the start till before the 4th index
3 print(my_string[1:7])
4 print(my_string[8:len(my_string)]) # From the 8th index till the end
```

Output

```
This
is
MY string!
```

Strings Slicing with a step:

- Until now, we've used slicing to obtain a contiguous piece of a string, i.e., all the characters from the starting index to before the ending index are retrieved.
- However, we can define a step through which we can skip characters in the string. The default step is 1, so we iterate through the string one character at a time.
- The step is defined after the end index

```
1 my_string = "This is MY string!"
2 print(my_string[0:7]) # A step of 1
3 print(my_string[0:7:2]) # A step of 2
4 print(my_string[0:7:5]) # A step of 5
5
```

Output

```
This is
Ti s
Ti
```

Strings Reverse Slicing:

- Strings can also be sliced to return a reversed substring. In this case, we would need to switch the order of the start and end indices.
- A negative step must also be provided The step is defined after the end index

```
1 my_string = "This is MY string!"
2 print(my_string[13:2:-1]) # Take 1 step back each time
3 print(my_string[17:0:-2]) # Take 2 steps back. The opposite of what happens in the slide above
4
```

Output

```
rts YM si s
!nrsY ish
```

Strings Partial Slicing:

- One thing to note is that specifying the start and end indices is optional.
- If start is not provided, the substring will have all the characters until the end index.
- If end is not provided, the substring will begin from the start index and go all the way to the end.

```
1 my_string = "This is MY string!"
2 print(my_string[:8]) # All the characters before 'M'
3 print(my_string[8:]) # All the characters starting from 'M'
4 print(my_string[:]) # The whole string
5 print(my_string[::-1]) # The whole string in reverse (step is -1)
6
```

Output

```
This is
MY string!
This is MY string!
!gnirts YM si siht
```

Quiz:

What is the output of the following code?

```
> my_string = "0123456789"  
> print(my_string[-2: -6: -2])
```

- 5432
- 8765
- 532
- 86

Quiz:

What is the output of the following code?

```
> my_string = "0123456789"  
> print(my_string[-2: -6: -2])
```

- 5432
- 8765
- 532
- 86

Quiz:

String indices can be floats?

- ☐ True
- ☐ False

Quiz:

String indices can be floats?

- ☐ True
- ☒ False

Common String Operators:

- Assume string variable a holds 'Hello' and variable b holds 'Python'

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e a[-1] will give o
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	'H' in a will give True

Common String Operators:

- Strings are an example of Python objects. An object contains both data (the actual string itself) and methods, which are effectively functions that are built into the object and are available to any instance of the object.
- `upper()` Return a capitalized version of string
- `lower()` Return a copy of the string converted to lowercase.
- `find()` searches for the position of one string within another
- `strip()` removes white space (spaces, tabs, or newlines) from the beginning and end of a string

Other useful methods in this [link](#)

Python User Input from Keyboard

- Python user input from the keyboard can be read using the `input()` built-in function.
- The input from the user is read as a string and can be assigned to a variable.
- After entering the value from the keyboard, we have to press the “Enter” button. Then the `input()` function reads the value entered by the user.

```
In [*]: input("please enter a value")
```

please enter a value

Python User Input from Keyboard

- The program halts indefinitely for the user input. There is no option to provide timeout value.
- The syntax of input() function is: `input(prompt)`
- The prompt string is printed on the console and the control is given to the user to enter the value. You should print some useful information to guide the user to enter the expected value.

Python User Input from Keyboard

- What is the type of user entered value?
 - The user entered value is always converted to a string and then assigned to the variable. Let's confirm this by using type() function to get the type of the input variable.
- How to get an Integer as the User Input?
 - There is no way to get an integer or any other type as the user input. However, we can use the built-in functions to convert the entered string to the integer.

```
In [12]: my_input=input("please enter a number")  
          print(type(my_input))  
  
          please enter a number10  
          <class 'str'>
```

```
In [13]: my_input=input("please enter a number")  
          my_input=int(my_input)  
          print(type(my_input))  
  
          please enter a number100  
          <class 'int'>
```

Printing Strings

- Print statement can have different formats

```
In [18]: ▶ print("my name is ", name)
```

```
my name is  ahmed
```

```
In [20]: ▶ print("my name is {} and my age is {}".format(name, age))
```

```
my name is ahmed and my age is 20
```

```
In [21]: ▶ print("my name is {0} and my age is {1}".format(name, age))
```

```
my name is ahmed and my age is 20
```

```
In [24]: ▶ print("my name is {2} and my age is {0}".format(name, age, gender))
```

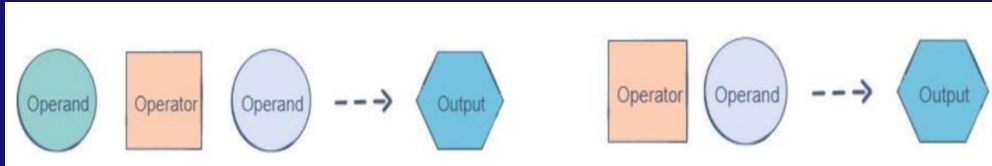
```
my name is male and my age is ahmed
```

Escape Characters

- Sometimes we want to print strings having special characters. So we add backslash before the special character as follows: "My name is \"Ahmed\" "
- Escape Characters:
 - \': single quote
 - \\: Backslash
 - \n: New line
 - \t: tab
 - \b: Backspace

Operators

- Operators are used to perform arithmetic and logical operations on data. They enable us to manipulate and interpret data to produce useful outputs.
- Python operators follow the infix or prefix notations:



- The 5 main operator types in Python are:
Arithmetic - Comparison - Assignment - Logical - Bitwise

PYTHON OPERATORS

Arithmetic		
Operator	Purpose	Example
+	Addition (Sum of two operands)	a + b
-	Subtraction (Difference between two operands)	a - b
*	Multiplication (Product of two operands)	a * b
/	Float Division (Quotient of two operands)	a / b
//	Floor Division (Quotient with fractional part)	a // b
%	Modulus (Integer remainder of two operands)	a % b
**	Exponent (Product of an operand n times by itself)	a ** n

PYTHON OPERATORS

Comparison		
Operator	Purpose	Example
>	Greater than (If left > right hence return true)	a > b
<	Less than (if left < right hence return true)	a < b
==	Equal to (if left equals right return true)	a == b
!=	Not equal to (if left not equals right return true)	a != b
>=	Greater than or equal (if left GTE right return true)	a >= b
<=	Less than or equal (if left LE right return True)	a <= b

PYTHON OPERATORS

Logical		
Operator	Purpose	Example
and	If a and b are both true hence return true	a and b
or	If either a or b is true hence return true	a or b
not	If a is true return false and vice versa	not a

PYTHON OPERATORS

Bitwise		
Operator	Purpose	Example
&	If a and b are both one in bit level return 1	a & b
	If a and b are either one or both in bit level return 1	a b
~	Invert all the bits of the passed operand	~ a
^	If a and b are either 1 but not both in bit level return 1	a ^ b
>>	Shift the bits of a to the right n times	a >> n
<<	Shift the bits of a to the left n times	a << n

PYTHON OPERATORS

- Logic Tables:

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT Truth Table

A	B
0	1
1	0

PYTHON OPERATORS

Assignment		
Operator	Purpose	Example
arithmetic=	Any arithmetic operation followed by = which apply the arithmetic operation of the left operand and put the result in it.	a += 1 equivalent to a = a + 1
bitwise=	Any bitwise operation followed by = which apply the bitwise operation of the left operand and put the result in it.	a &= 1 equivalent to a = a & 1

PYTHON OPERATORS

Identity		
Operator	Purpose	Example
is	If both operands refers to same object return True	a is b
Is not	If both operands refers to different objects return True	a is not b
Membership		
Operator	Purpose	Example
in	If a given value exists in a given sequence	“s” in [“a”, “n”, “s”]
not in	If a given value doesn’t exist in a given sequence	“s” in [“a”, “n”, “u”]

Quiz

➤ What is the operator used?

- OR
- XOR
- AND
- NOT

```
11110010
10011100
```

=

```
10010000
```


Quiz

➤ What is the operator used?

- OR
- XOR
- **AND**
- NOT

$$\begin{array}{c} 11110010 \\ 10011100 \end{array} = 10010000$$

Quiz

➤ What is the output of printing 'result'?

- False
- -21
- 5
- 5.25

```
x = 20  
y = 5  
result = (x + True) / (4 - y * False)
```

Quiz

➤ What is the output of printing 'result'?

- False
- -21
- 5
- 5.25

```
x = 20  
y = 5  
result = (x + True) / (4 - y * False)
```

Lists

- A list is a collection of mutable items in a particular order , List constants are surrounded by square brackets and the elements in the list are separated by commas.
- A list element can be any Python object - even another list
- Syntax: `list_variable = ["d", "a", 4, 5]`
- You can apply `len()`, `type()`.
- It can contain different data types.
- You can use `list()` constructor instead of the square brackets.
- Same rules of indexing and slicing strings apply here.
- You can modify an item using basic assignment: `list_variable[0] ='c'`

Lists Important Methods

- `Append(item)`: adds an element to the end of the list
- `Insert(pos, item)`: adds an element at the position `pos`.
- `Extend(another_list)`: concatenate `another_list`/any other sequence to the end
- `Remove(item)`: removes the item from a list
- `Pop(pos)`: removes the item at the position `pos`. if no `pos`, removes last.
- `Clear()`: delete items of the list but the list itself is still there.
- `Copy()`: you can't do `list1 = list2`, instead use `list1 = list2.copy()`

Search for a method to sort the items based on any metric.

Tuples

- Syntax: `Tuple_variable = ("a", "b", "c")`
- You can apply `len()`, `type()`.
- It can contain different data types.
- You can use `tuple()` constructor instead of the brackets.
- Same rules of indexing and slicing strings apply here.
- You cannot modify an item using basic assignment: `tuple_variable[0] = 'c'`

Tuples Notes & Methods

- Changing a tuple: convert into list then change then convert back into tuple.
- Join two tuples together: tuple1 + tuple2, knowing that tuple1,tuple2 > 1 element
- Multiply tuples: tuple1 * 2
- count(): Returns the number of times a specified value occurs in a tuple
- index(): Searches the tuple for a specified value and returns the position of where it was found

Sets

- Syntax: `Set_variable = {"a", "b", 6-}`
- → You can apply `len()`, `type()`.
- → It can contain different data types.
- → You can use `set()` constructor instead of the curly brackets.
- → You cannot access them the same way of indexing.
- → You cannot modify an item using basic assignment: `set_variable[0] = 'c'`

Sets Notes & Methods

- Accessing Set elements use: IN operator.
- You can add items using add()
- Same as extend() in lists you can use update() to add two sets/any other sequence
- Remove(): to remove an item from the set
- Union() == Update() but union() returns a new set. Update() modifies.
- Intersection(): get the duplicated items from two sets and return a new set.
- intersection_update() same as intersection() but updates directly.

Dictionaries

- Syntax: Dict_variable = {"name": "Merna", "age": 20, 1: [1,2,3]}
- → You can apply len(), type().
- → It can contain different data types.
- → You can use dict() constructor instead of the curly brackets.
- → You can access them using Keys.
- → You can modify an item using basic assignment: dict_variable['name']= 'Ahmed'

PYTHON DATA TYPES: DICTIONARY

Dictionaries can be deleted using the `del` function in python.

Duplicate keys are not allowed. Last key will be assigned while others are ignored. Some important built-in functions:

- `.clear()` to clear all elements of the dictionary.
- `.copy()` to copy all elements of the dictionary to another variable.
- `.fromkeys()` to create another dictionary with the same keys.
- `.get(key)` to get the values corresponding to the passed key.
- `.has_key()` to return True if this key is in the dictionary.
- `.items()` to return a list of dictionary (key, value) tuple pairs.
- `.keys()` to return list of dictionary dictionary keys.
- `.values()` to return list of dictionary dictionary values.
- `.update(dict)` to add key-value pairs to an existing dictionary.

Comparison between the rest of data types:

List	Tuple	Set	Dictionary
Ordered	Ordered	Unordered	Ordered
Changeable	Unchangeable	Changeable	Changeable
Duplicates yes	Duplicates yes	No Duplicates	No Duplicates
Indexed	Indexed	Unindexed	Indexed with key

Comparison between the rest of data types

- Ordered means that each element won't change its place until you modify it.
- Changeable means you can edit its element.
- No Duplicates means that it only contains unique values.
- Indexed means you can access each element by its index/position except dictionaries you access elements using keys.

Task2

- Write a program that accepts a sequence of comma-separated numbers from user and generates a list and a tuple with those numbers

If Else Conditional Statement

To control actions taken by the software.

Syntax:

 If (condition):

 Statements

 elif (condition):

 Statements

 else:

 Statements

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Loops

- A loop is a control structure that is used to perform a set of instructions for a specific number of times.
- Loops solve the problem of having to write the same set of instructions repeatedly. We can specify the number of times we want the code to execute.
- One of the biggest applications of loops is traversing data structures, e.g. lists, tuples, sets, etc. In such a case, the loop iterates over the elements of the data structure while performing a set of operations each time.
- There are two types of loops that we can use in Python:
 - The for loop
 - The while loop

For Loops

- To iterate over a Sequence as long It's True.
- Syntax:
 for (iterator) in (sequence):
 Statements
- The iterator is a variable that goes through the sequence.
- The in keyword specifies that the iterator will go through the values in the sequence/data structure.

for **iterator** in **sequence** :



For Loops: Range()

- In Python, the built-in range() function can be used to create a sequence of integers. This sequence can be iterated over through a loop. A range is specified in the following format: Range(start,end,step).

Range(6): [0,1,2,3,4,5]
Range(2,10): [2,3,4,5,6,7,8,9]
Range(2,10,3): [2,5,8]

```
for i in range(1, 11): # A sequence from 1 to 10
    if i % 2 == 0:
        print(i, " is even")
    else:
        print(i, " is odd")
```

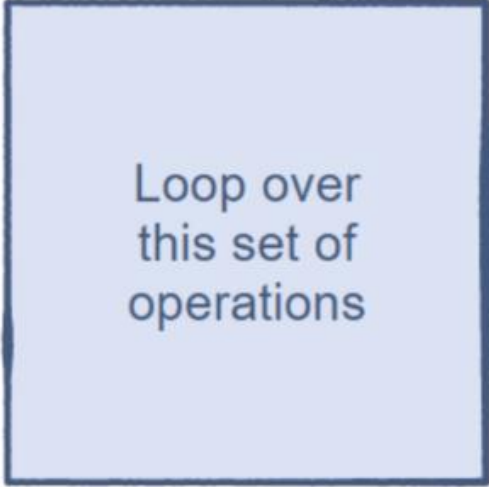
Output

```
1    is odd
2    is even
3    is odd
4    is even
5    is odd
6    is even
7    is odd
8    is even
9    is odd
```

While Loops

- To iterate over a Condition as long It's True.
- In a for loop, the number of iterations is fixed since we know the size of the sequence. On the other hand, a while loop is not always restricted to a fixed range. Its execution is based solely on the condition associated with it.

while condition is true :



Loop over
this set of
operations

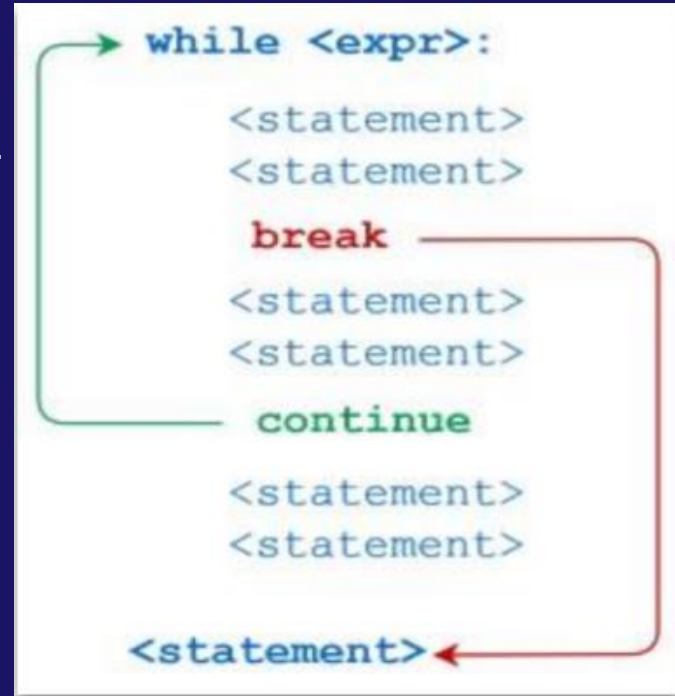
While Loops

- Be Careful, An exit condition must be provided. Exit Conditions are those where you modify the values you used in the loop condition so that the loop can be exited.

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

Continue/Break keywords

- Both can be used as exit conditions for while loops.
- Continue will cut the loop and start the next iteration.
- Break will cut the loop and exit all iterations.



Pass keyword

- Pass statement is a null operation, which is used when the statement is required syntactically. Think of it like a placeholder, until you write the code.

```
pass.py - D:/python/pass.py (3.7.4)
File Edit Format Run Options Window Help
string1 = "Stechies"

# Use of pass statement
for value in string1:
    if value == 'e':
        pass
    else:
        print("Value: ", value)
```

A red box highlights the `pass` statement, with a callout bubble labeled "pass statement" pointing to it.

```
D:\python>python pass.py
Value: S
Value: t
Value: c
Value: h
Value: i
Value: s
D:\python>
```

A red box highlights the output of the program, with a callout bubble labeled "Required Output" pointing to it.

Quiz:

Write a program that prints the integers from 1 to 100. But for multiples of three print "Fizz" instead of the number, and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

```
for num in range(1,101):  
    if num % 3 == 0 and num % 5 == 0:  
        print("FizzBuzz")  
    elif num % 3 == 0:  
        print("Fizz")  
    elif num % 5 == 0:  
        print("Buzz")  
    else:  
        print(num)
```

Quiz

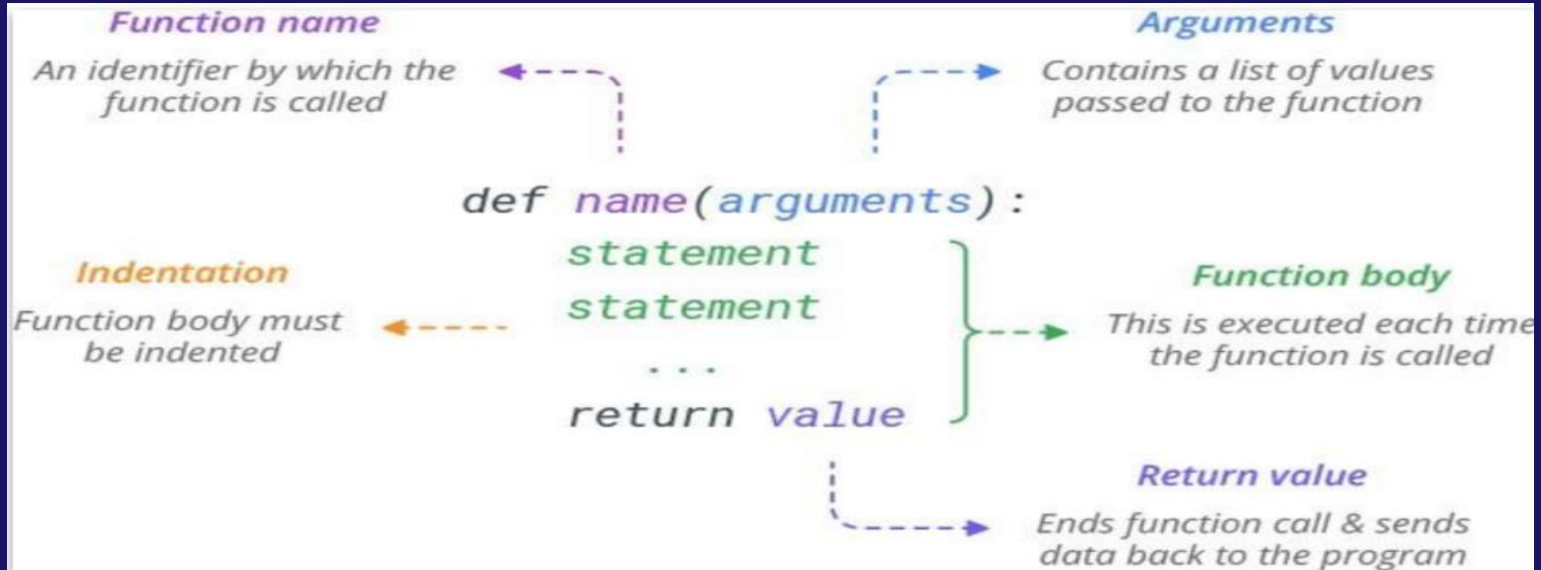
Write a program that prompts a user for a two-word string and prints True if both words begin with same letter

- ('Levelheaded Llama') --> True
- ('Crazy Kangaroo') --> False

```
word=input("please enter a two words separated by space\n")
my_words_list=word.split()
if my_words_list[0][0] == my_words_list[1][0] :
    print("True")
else:
    print("False")
```


Functions

- A block of code that is used more than once.



Functions notes

- You can define functions that take multiple arguments.
- You can define functions that takes no argument.
- You can define functions that do not return any values.
- Default Values: if a parameter is not required to be provided, assign to it a default value.
- For Example: absolute_value function is as follows:

```
def absolute_value(num):  
    """This function returns the absolute  
    value of the entered number"""  
  
    if num >= 0:  
        return num  
    else:  
        return -num
```

Function Calling

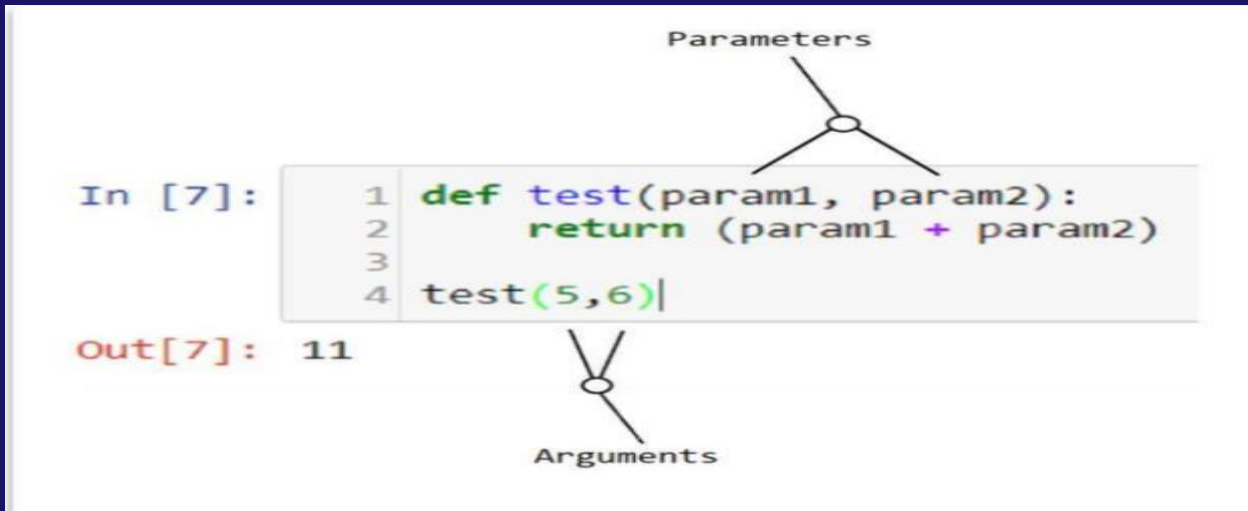
- If the function return something we call it like:
Variable=func_name(arg_1,arg_2)
- If the function doesn't return anything we call it like:
func_name(arg_1,arg_2)

```
def test_size(size):  
    if size>10:  
        print("size is within the range")  
    else:  
        print("size is outside the range")
```

```
def minimum(first, second):  
    if (first < second):  
        return first  
    return second  
  
num1 = 10  
num2 = 20  
  
result = minimum(num1, num2)  
print(result)
```

Parameters Vs Arguments

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that are sent to the function when it is called.



Quiz

- Transform the code you made for calculating the area of a circle into function that:
 - Takes the radius as inputs
 - Gives the area as an output

```
def circle_area(radius, pi_const = 3.14):  
    area = pi_const * radius * radius  
    return area  
  
print(circle_area(2))
```

*Args & **Kwargs

- (*args): You can pass undefined length of a sequence arguments using *param_name.
- (**kwargs): You can pass undefined length of key-value pairs using **param_name

```
def args(*names):  
    for name in names:  
        print(name)  
  
args('ahmed', 'mohamed', 'mostafa')
```

```
def employee_data(**employee):  
    print(employee['name'])  
    print(employee['age'])  
  
employee_data(name= 'ahmed', age='20')
```

Lambda Function

- Just like any normal python function, except that it has no name when defining it (Small anonymous function). And can only have one expression.
- It is contained in one line of code.
- Syntax: lambda arguments: expression
- In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments), like the filter() function.

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0) , my_list))

print(new_list)
```

Quiz

What is the output of the following program:

- 2
- 6
- 8
- 4

```
high_order_func = lambda x, function1: x + function1(x)
print(high_order_func(2, lambda x: x * x))
```


Quiz

What is the output of the following program:

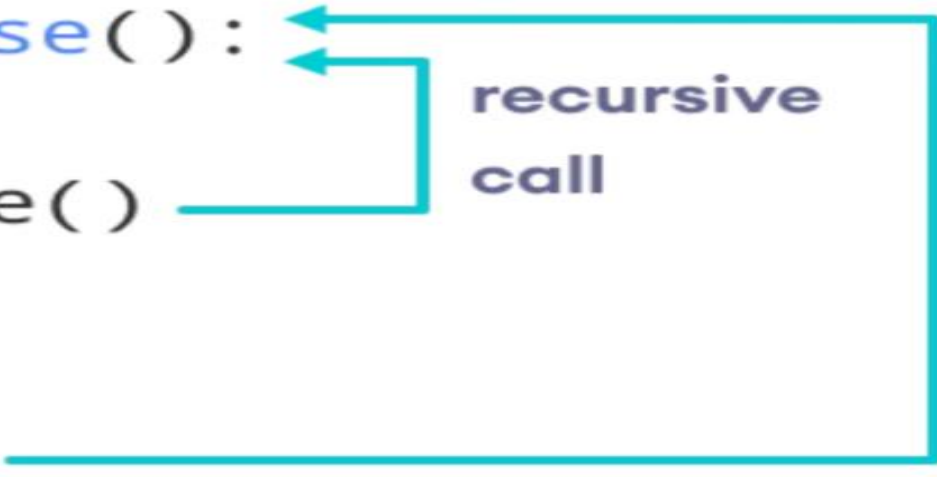
- 2
- 6
- 8
- 4

```
high_order_func = lambda x, function1: x + function1(x)
print(high_order_func(2, lambda x: x * x))
```

Recursive Function

- Recursion is the process of defining something in terms of itself.

```
def recurse():  
    ...  
    recurse()  
    ...  
  
recurse()
```



recursive call

Quiz

Using the recursion concept, write a function to calculate the factorial of a given number and returns the result:

- $1! = 1$
- $3! = 3*2*1$
- $5! = 5*4*3*2*1$

```
def factorial(x):  
    if x == 1:  
        return 1  
    else:  
        return x * factorial(x-1)
```

Global and Local Scope

- In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.
- You cannot change a global variable directly inside another scope. unless you add keyword `Global` before it.
- Local variables are defined only within a function scope. Can't be accessed outside the function.
- Nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

Global and Local Scope

```
x = "global"

def foo():
    print("x inside:", x)

foo()
print("x outside:", x)
```

```
x inside: global
x outside: global
```

```
x = "global"

def foo():
    x = "local"
    print("x inside:", x)

foo()
print("x outside:", x)
```

```
x inside: local
x outside: global
```

```
x = "global"

def foo():
    global x
    x = "changed"
    print("x inside:", x)

foo()
print("x outside:", x)
```

```
x inside: changed
x outside: changed
```

Global and Local Scope

```
def outer():  
    x = "local"  
  
    def inner():  
        x = "nonlocal"  
        print("inner:", x)  
  
    inner()  
    print("outer:", x)
```

outer()

```
inner: nonlocal  
outer: local
```

```
def outer():  
    x = "local"  
  
    def inner():  
        nonlocal x  
        x = "nonlocal"  
        print("inner:", x)  
  
    inner()  
    print("outer:", x)
```

outer()

```
inner: nonlocal  
outer: nonlocal
```

Try.. Except Clause

- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong)
- built-in exceptions
 - IOError , ValueError , ImportError , EOFError , KeyboardInterrupt ,ZeroDivisionError...
- [Link to built-in exceptions](#)

```
try:  
    print(x)  
except:  
    print("An exception occurred")
```

File Handling – opening files

- To open a file in python use `open()` and save it to a variable. You may pass it the name of the file if the python script is in the same folder, or pass the whole path.
- You may need to specify the mode you open your file with:
 - "r": read the content of the file.
 - "a": append, creates the file if not exists.
 - "w": write, creates the file if not exists, overwrite its content.
 - "x": creates the file only.

```
f = open("test.txt")  
f = open("C:/path/README.txt")  
f = open("test.txt", 'w')
```


File Handling – opening files

- The return is a file object, to read its content ,use .read() method.
- File objects need to be closed after you're done, use .close() method.
- or you can use "with" context manager.
- [Link to more file methods](#)

```
with open("test.txt", encoding = 'utf-8') as f:  
    f.write("my first file\n")  
    print(f.read(4)) # read the first 4 data
```

Quiz

- Write a Python program to read a file line by line and store it into a list

```
def file_read(fname):  
    with open(fname) as f:  
        #Content_list is the list that contains the read lines.  
        content_list = f.readlines()  
        print(content_list)  
  
file_read('test.txt')
```

Task:

- Write a function to get user info first name, last name, gender, id (3 numbers) and save it to a text file.

Modules Built-in packages

- Python as any other programming languages has built-in packages that can be imported and used without being explicitly installed. Importing a Module :
- To use the methods of a module, we must import the module into our code. This can be done using the import keyword.
 - Import modulename
 - Import modulename as md
 - From modulename import methodname, methodname
 - From modulename import *
- All built-in python packages can be found [here](#):

Used Modules: random()

- This module implements pseudo-random number generators for various distributions.

```
>>> import random
>>> random.random()
0.645173684807533
>>> random.randint(1, 100)
95
>>> random.randrange(1, 10)
2
>>> random.choice('computer')
't'
>>> numbers=[12,23,45,67,65,43]
>>> random.shuffle(numbers)
>>> numbers
[23, 12, 43, 65, 67, 45]
```

- Random.random() returns a float number between 0 and 1 randint(x,y) will return a value $\geq x$ and $\leq y$, while randrange(x,y) will return a value $\geq x$ and $< y$ (n.b. not less than or equal to y)

User-defined Modules

To import a .py file inside another make sure there are in the same folder.

- In help.py
- In main.py

```
def func(x):  
    return x*x
```

```
1  from help import func  
2  
3  print(func(3))
```

Mini-project

- Create a Calculator

Ask the user to enter a command "add","sub","mult","div" and two numbers to simulate a calculator.

All of your functions are in a separate script calc.py

Your main code is in main.py

Bonus: ask the user if he would like to make another operation or not. if yes, do it all over again in the same run. only exit the program if he says 'stop'



Thanks