

Python for Data Science, AI & Development

17/04/2024

Course Overview

- This course introduces the basics of Python 3, including conditional execution and iteration as control structures, and strings and lists as data structures.
- The course is for you if you're a newcomer to Python programming, if you need a refresher on Python basics, or if you may have had some exposure to Python programming but want a more in-depth exposition and vocabulary for describing and reasoning about programs.

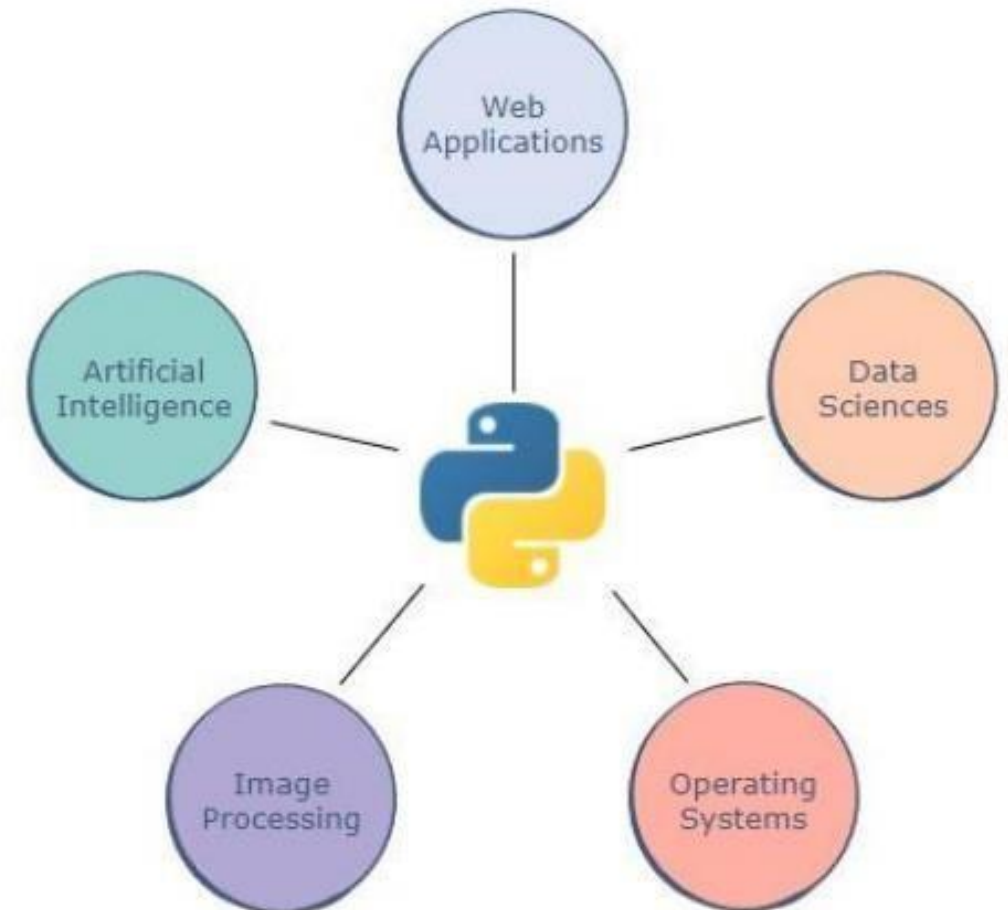
Course Content

- Intro to python
- 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
- OOP



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.



Python History

- Back in the late 80's
- Implemented and started at 1989 by " Guido Van Rossum "
- Developed at CWI Netherlands
- Capable of exception handling and interfacing
- Python 2.0 on October 2000
- Supported unicode & memory management
- Released python 3 by the community backed process
- C provided some of it's syntax



What is Programming Languages?



interpreter



From python code to machine language



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 to Learn Python?

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn.
- Python is versatile and can be used to create many different things.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

Careers with Python

- Game developer
- Web designer
- Python developer
- Full-stack developer
- Machine learning engineer
- Data scientist
- Data analyst
- Data analyst
- Data engineer
- DevOps engineer
- Software engineer
- Many more other roles
- Software engineer
- Many more other high salary

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

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, "IBM_Data_Science" in Python?

- `print IBM_Data_Science`
- `Print " IBM_Data_Science"`
- `print("IBM_Data_Science")`
- `print(IBM_Data_Science)`

Quiz

How can we print the text, "IBM_Data_Science" in Python?

- `print IBM_Data_Science`
- `Print " IBM_Data_Science"`
- `print("IBM_Data_Science")`
- `print(IBM_Data_Science)`

Comments

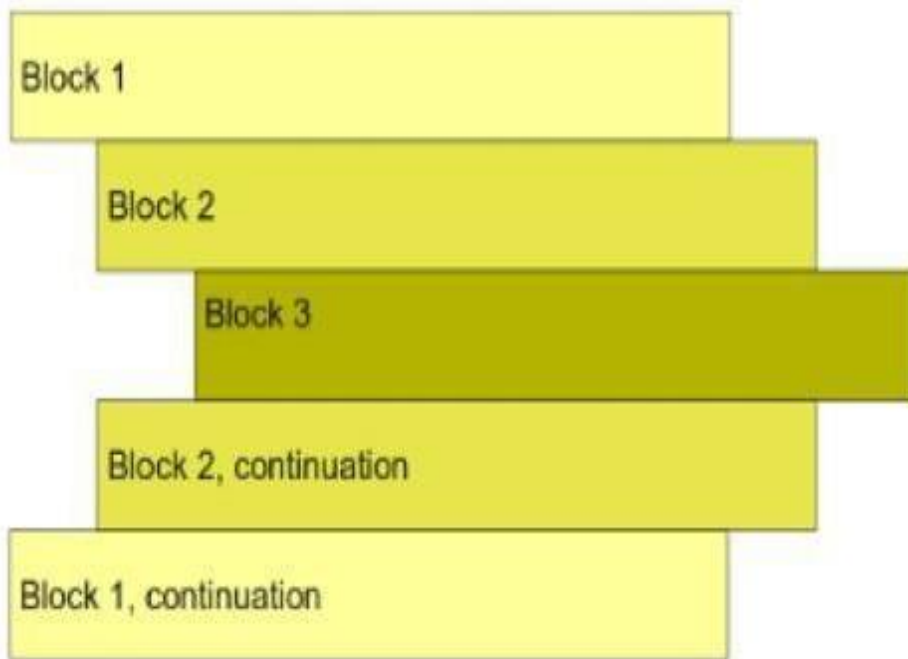
- Single comment: #
- Multiline comment/docstrings: between three double/single quotes `"""` `"""`. Or you can just write # multiple times.
- You can use Ctrl+ "/" to automatically comment a whole section.

```
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
```


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

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

Python Statements & 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

- 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
```

Variables

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

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

You can also assign multiple objects to multiple variables.

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

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

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

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


Variables

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

```
D      int x = 4
```

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

```
D      x = 4
```

It knows automatically that x is an integer.

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.

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

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 | |

- 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: $\text{<real part>} + \text{<imaginary part>}j$ (not used much in Python programming)

```
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)
```

Common Number Functions

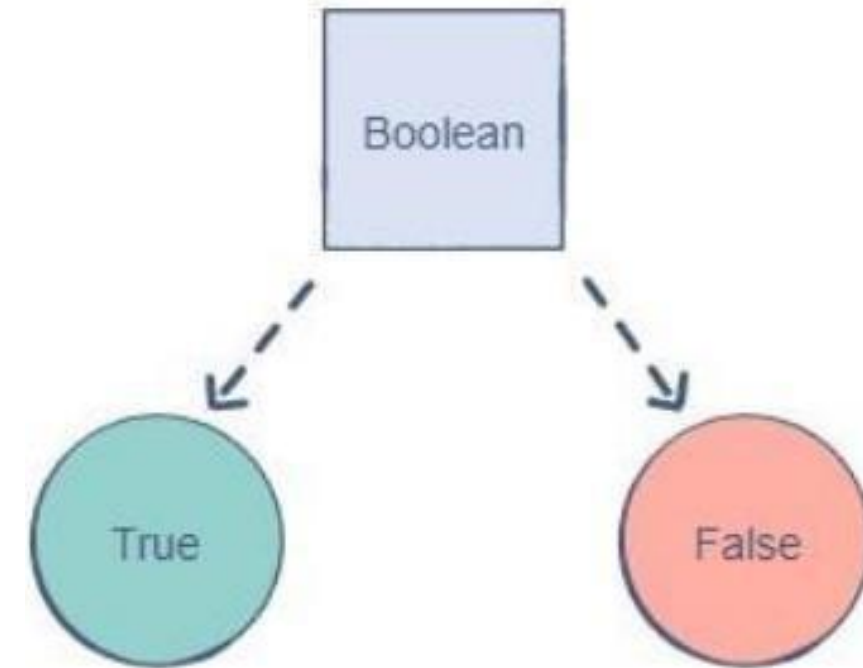
Function	Description
int(x)	to convert x to an integer
float(x)	to convert x to a floating-point number
abs(x)	The absolute value of x
cmp(x,y)	-1 if $x < y$, 0 if $x == y$, or 1 if $x > y$
sqrt(x)	The square root of x for $x > 0$
log(x)	The natural logarithm of x, for $x > 0$
pow(x,y)	The value of $x^{**}y$

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.



Strings

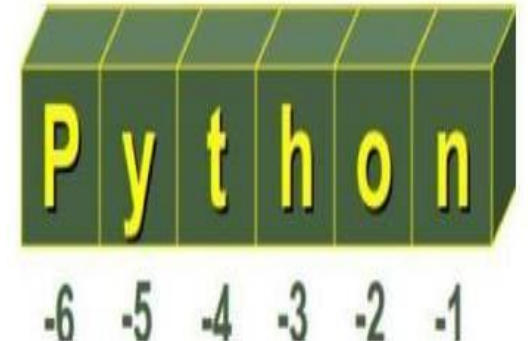
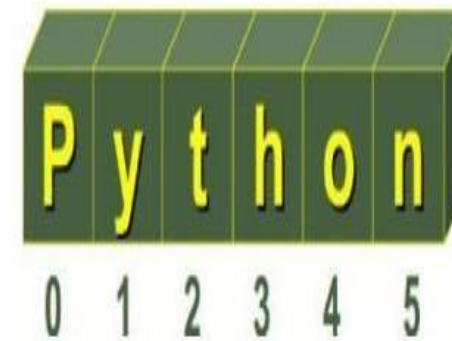
- A string is a collection of characters closed within single or double quotation marks
- (immutable).
- You can update an existing string by (re)assigning a variable to another string.
- Python does not support a character type; these are treated as strings of length one.

```
>>> 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
his 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

- D 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.
- D 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

D 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

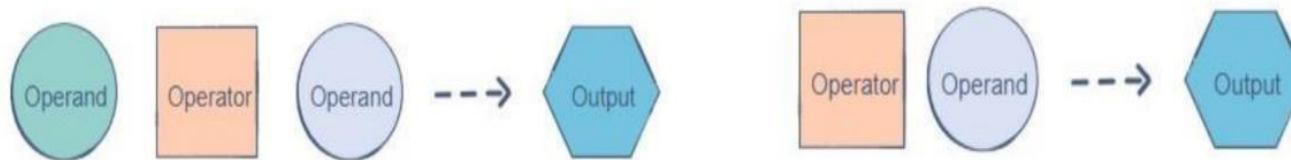
\n: New line

\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

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)	$a ** n$

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)	$a <= b$

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

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

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

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 $a = a \& 1$

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

What is the operator

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

Quiz

D
used?

What is the operator

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

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

Quiz

D What is the output of printing
'result'?

- False
- -21
- 5
- 5.25

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

Quiz

D 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()`

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

Task

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

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'

DICTIONARY Notes & Methods

- Dictionaries can be deleted using the del function in python.
- Duplicate keys are not allowed. Last key will be assigned while others are ignored.

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 list, Tuple, Set, Dictionary :

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 list, Tuple, Set, Dictionary :

- 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.

IF, 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")
```

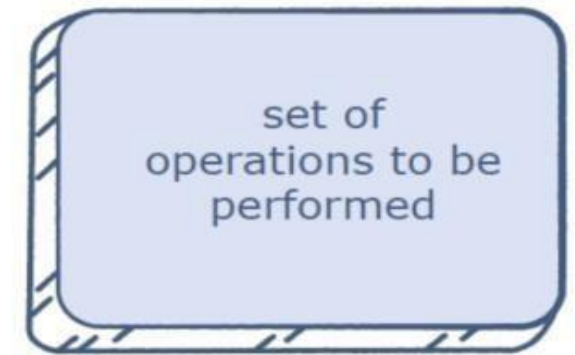
Loops

- 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

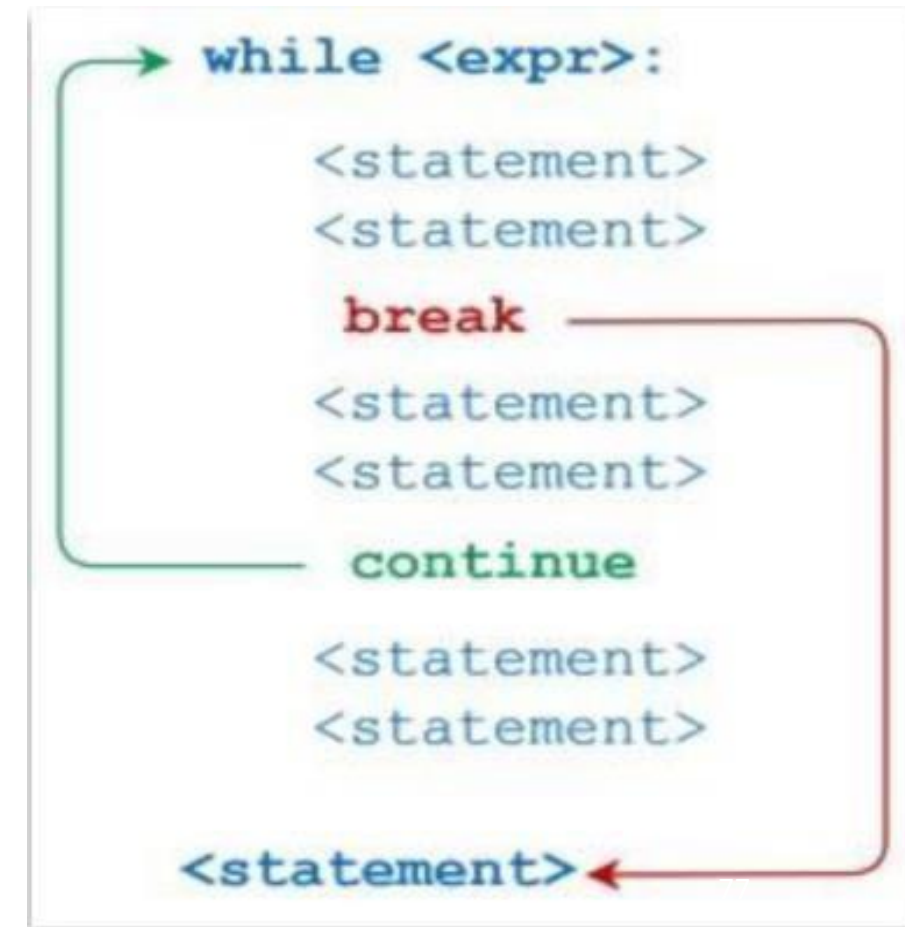
While Loops

- 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)
```

pass statement

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

Required Output

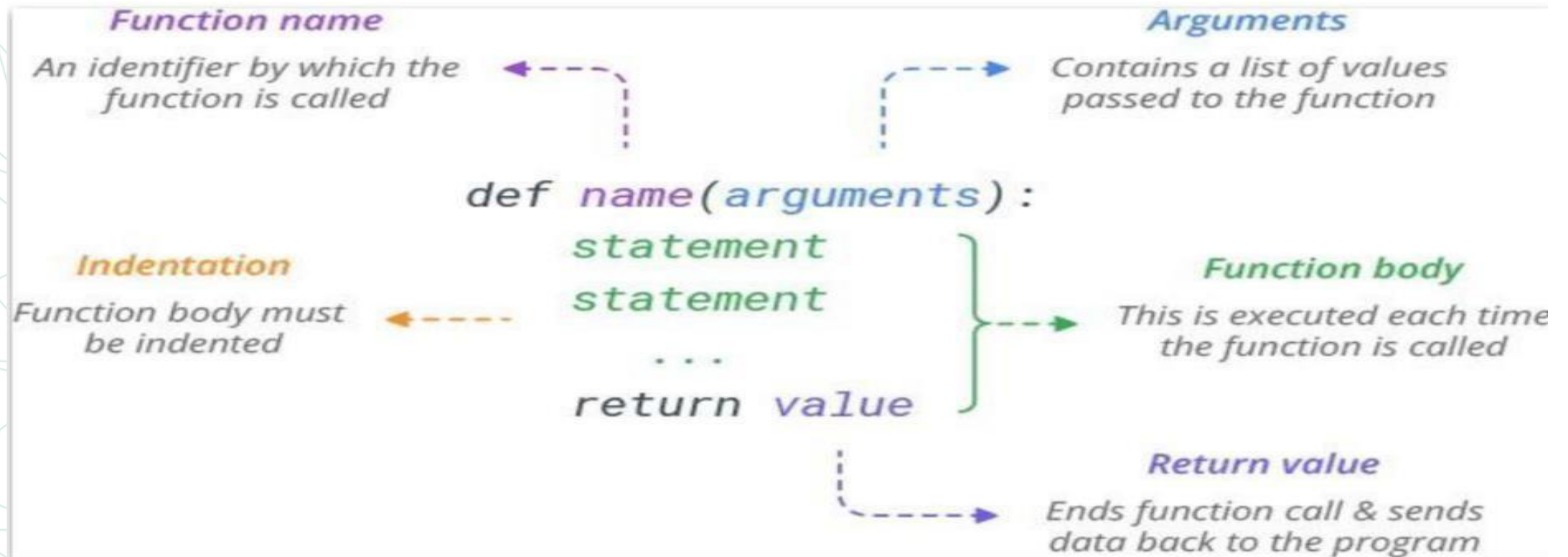
Quiz

- D 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)
```

Functions

A block of code that is used more than once.



Function 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.

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. Argument

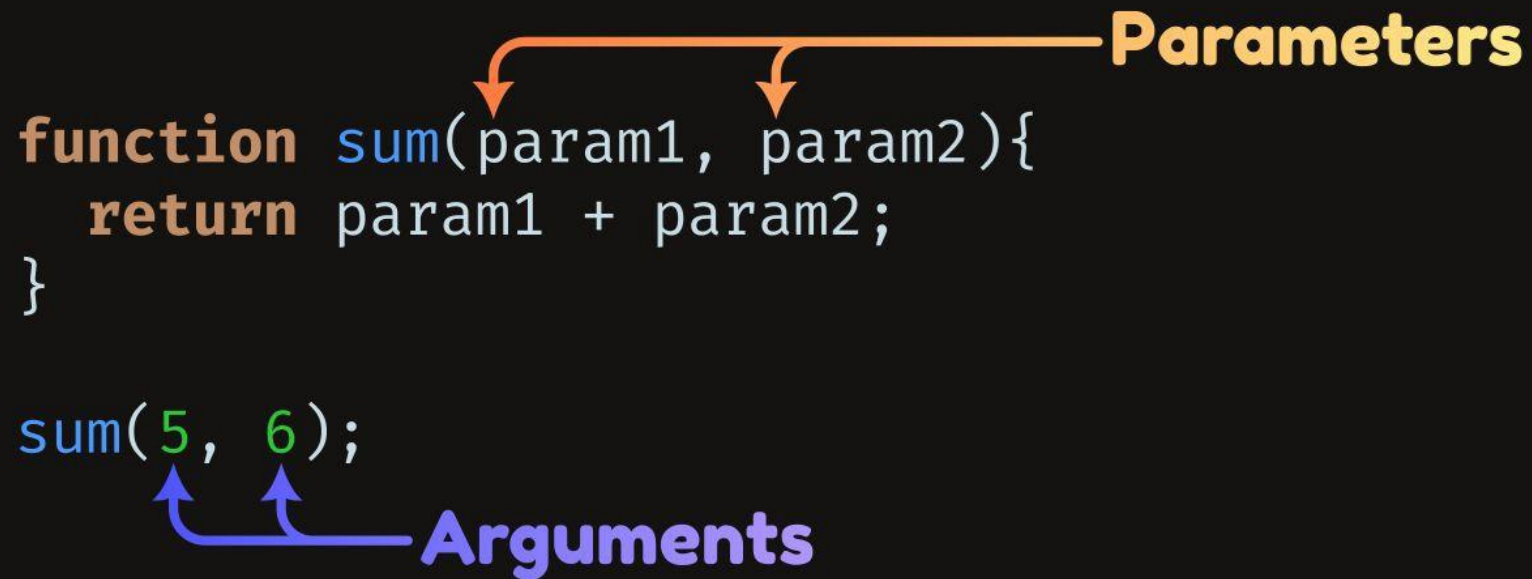
- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that are sent to the function when it is called.

```
function sum(param1, param2){  
    return param1 + param2;  
}
```

Parameters

```
sum(5, 6);
```

Arguments



Quiz

D 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))
```

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
```

Try, Except

- 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 Error Handling

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


File Operations

- 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 Operations

- 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

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

Modules

- 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 *`

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'

Loading Data with Pandas

- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.
- In our examples we will be using a CSV file called 'data.csv'.

```
import pandas as pd  
  
df = pd.read_csv('data.csv')
```

Working/Saving data in Pandas

- The easiest way to do this

```
df.to_csv('file_name.csv')
```

- If you want to export without the index, simply add index=False

```
df.to_csv('file_name.csv', index=False)
```

- If you get UnicodeEncodeError , simply add encoding='utf-8'

```
df.to_csv('file_name.csv', encoding='utf-8')
```

Vector/Matrix Creation

```
# Creation vector, matrix and tensor
```

```
import numpy as np
```

```
# Create one dimensional array (vector)
```

```
one_D_arr = np.array([1, 2, 3])
```

```
print(f"Vector : \n{one_D_arr}\n"  
      f"Shape : {one_D_arr.shape}")
```

Numpy ×

```
"C:\Users\Alhasan Gamal\anaconda3\python.exe"
```

```
Vector :
```

```
[1 2 3]
```

```
Shape : (3,)
```

```
# Create two-dimensional array (matrix)
```

```
two_D_arr = np.array([[1, 2, 3],  
                      [4, 5, 6],  
                      [7, 8, 9]  
                      ])
```

```
print(f"Matrix : \n{two_D_arr}\n"  
      f"Shape : {two_D_arr.shape}")
```

Numpy ×

```
"C:\Users\Alhasan Gamal\anaconda3\python.exe"
```

```
Matrix :
```

```
[[1 2 3]
```

```
 [4 5 6]
```

```
 [7 8 9]]
```

```
Shape : (3, 3)
```

What is the API?

Application Programming Interface, is a server that you can use to retrieve and send data to using code. APIs provide essential tools in the world of artificial intelligence (AI) and data science, enabling access to vast amounts of data and powerful computing capabilities.

More specifically, APIs play a crucial role in AI and data science projects by enabling:

- Access to AI models: APIs allow developers to integrate pre-trained AI models into their applications, such as natural language processing (NLP) models from OpenAI's GPT-4 API or computer vision models from Google Cloud Vision API.
- Data acquisition: APIs provide access to large datasets required for training machine learning models. This includes public datasets like those offered by Kaggle or data from social media platforms like X (formerly Twitter) and Facebook.
- AI-powered services: Many companies offer APIs that allow developers to integrate AI capabilities into their applications without building the models themselves. Examples include sentiment analysis, entity recognition, and language translation.

Making API Requests

- If you use pip to manage your Python packages, you can install the requests library using the following command:

```
pip install requests
```

- Once you've installed the library, you'll need to import it. Let's start with that important step:

```
import requests
```

Understanding Common API Status Code

- Every request to a web server returns a status code indicating what happened with the request. Here are some common codes relevant to GET requests:
 - 200: Everything went okay, and the result has been returned (if any).
 - 301: The server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint name is changed.
 - 400: The server thinks you made a bad request. This happens when you send incorrect data or make other client-side errors.
 - 401: The server thinks you're not authenticated. Many APIs require login credentials, so this happens when you don't send the right credentials to access an API.
 - 403: The resource you're trying to access is forbidden: you don't have the right permissions to see it.
 - 404: The resource you tried to access wasn't found on the server.
 - 503: The server is not ready to handle the request.

Introduction to REST APIs

REST (Representational State Transfer):

- Architectural style for designing networked applications
- Uses a stateless communication protocol, typically HTTP

Key Principles of REST

Statelessness:

Each request from a client to a server must contain all the information needed to understand and process the request

Client-Server Architecture:

Separation of client and server concerns

Cacheability:

Responses must define themselves as cacheable or not to prevent clients from reusing stale data

Uniform Interface:

Simplifies and decouples the architecture, allowing each part to evolve independently


Layered System:

A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way

Example of a RESTful Request

Request:

http

 Copy code


POST /users HTTP/1.1

Host: api.example.com

Content-Type: application/json

Authorization: Bearer token

json


 Copy code

```
{  
  "name": "John Doe",  
  "email": "john.doe@example.com"  
}
```

Example of a RESTful Response

Response:


http

 Copy code

HTTP/1.1 201 Created

Content-Type: application/json

json

 Copy code

```
{  
  "id": 123,  
  "name": "John Doe",  
  "email": "john.doe@example.com"  
}
```

Introduction to Object-Oriented Programming (OOP)

In this presentation, we'll explore the world of Object-Oriented Programming (OOP) in Python. OOP is a powerful paradigm for structuring programs by creating objects that encapsulate data (attributes) and functionality (methods). It allows us to model real-world entities and their interactions, making code more modular, reusable, and maintainable.

OOP is based on three fundamental concepts:

- **Classes:** Blueprints that define the properties and behavior of objects.
- **Objects:** Instances of classes that encapsulate data (attributes) and functionality (methods).
- **Inheritance:** A mechanism for creating new classes (subclasses) that inherit properties and behavior from existing classes (superclasses).

Defining Classes

```
class Car:  
    def __init__(self, make, model):  
        self.make = make  
        self.model = model
```

Classes are defined using the class keyword followed by the class name and a colon.

The `__init__` method (constructor) is called when an object of the class is created. It's used to initialize the object's attributes.

Accessing Attributes

```
myCar = Car("Honda", "Civic")  
print(myCar.make) # Output: Honda
```

Once an object is created, we can access its attributes using dot notation (object_name.attribute_name).

Defining Methods

```
class Car:  
    def __init__(self, make, model):  
        self.make = make  
        self.model = model  
  
    def accelerate(self):  
        print("The car is accelerating!")
```

Methods define the behavior of objects. They are functions defined within a class. Methods can access and modify the object's attributes using self.

Calling Methods

```
myCar = Car("Honda", "Civic")  
myCar.accelerate() # Output: The car is accelerating!
```

We call methods on objects using `object_name.method_name()`.

Inheritance

Inheritance allows us to create new classes (subclasses) that inherit properties and behavior from existing classes (superclasses).

Subclasses can add new attributes and methods, or override inherited ones to provide specialized functionality.

Inheritance Example

```
class Animal:
    def __init__(self, name):
        self.name = name

    def make_sound(self):
        print("Generic animal sound")

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name) # Call superclass constructor
        self.breed = breed

    def make_sound(self):
        print("Woof!")

myDog = Dog("Buddy", "Labrador")
myDog.make_sound() # Output: Woof!
```

Polymorphism

Polymorphism is the ability of objects of different classes to respond to the same method call in different ways.

Achieved through inheritance and method overriding.

Polymorphism Example

```
class Animal:
    def make_sound(self):
        pass # Abstract method - subclasses must define it
```

```
class Dog(Animal):
    def make_sound(self):
        print("Woof!")
```

```
class Cat(Animal):
    def make_sound(self):
        print("Meow!")
```

```
def make_animal_sound(animal):
    animal.make_sound() # Polymorphic call
```

```
myDog = Dog("Buddy")
myCat = Cat("Whiskers")
```

```
make_animal_sound(myDog) # Output: Woof!
make_animal_sound(myCat) # Output: Meow!
```

Files I/O

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows:

```
Print("Python is really a great language,", "isn't it?")
```

This produces the following result on your standard screen

```
Python is really a great language, isn't it?
```


Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are

- `raw_input`
- `input`

The *raw_input* Function

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

```
str = raw_input("Enter your input: ");  
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this

```
Enter your input: Hello Python  
Received input is : Hello Python
```



The input Function

The `input([prompt])` function is equivalent to `raw_input`, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
str = input("Enter your input: ");  
print("Received input is : ", str)
```

This would produce the following result against the entered input

```
Enter your input: [x*5 for x in range(2,10,2)]  
Received input is : [10, 20, 30, 40]
```



Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.



The open Function

Before you can read or write a file, you have to open it using Python's builtin `open()` function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```



The open Function

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).



The open Function

- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).



Here is a list of the different modes of opening a file:

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.



w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.



The file Object Attributes

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
<code>file.closed</code>	Returns true if file is closed, false otherwise.
<code>file.mode</code>	Returns access mode with which file was opened.
<code>file.name</code>	Returns name of the file.
<code>file.softspace</code>	Returns false if space explicitly required with print, true otherwise.



Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

Name of the file: foo.txt

Closed or not : False

Opening mode : wb

Softspace flag : 0



The close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close();
```



Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ",
fo.name

# Close opened file
fo.close()
```

Name of the file: foo.txt



The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string –

Syntax

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file.



Example

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great
language.\nYeah its great!!\n");

# Close opened file
fo.close()
```

Python is a great language.
Yeah its great!!



The read() Method

The read() method reads a string from an open file. It is important to note that Python strings can have binary data, apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.



Example

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ",
str
# Close opened file
fo.close()
```

Read String is : Python is



Directories in Python

All files are contained within various directories, and Python has no problem handling these too.

The **os** module has several methods that help you create, remove, and change directories.



The mkdir() Method

You can use the mkdir() method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

```
#!/usr/bin/python  
import os
```

```
# Create a directory "test"  
os.mkdir("test")
```



The chdir() Method

You can use the `chdir()` method to change the current directory. The `chdir()` method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

```
import os

# Changing a directory to
"/home/newdir"
os.chdir("/home/newdir")
```



The getcwd() Method

The getcwd() method displays the current working directory.

Syntax

```
os.getcwd()
```

```
import os
```

```
# This would give location of the current directory  
os.getcwd()
```



The rmdir() Method

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

```
os.rmdir('dirname')
```

```
import os
```

```
# This would remove "/tmp/test" directory.  
os.rmdir( "/tmp/test" )
```



Questions & Answers



Thank you!