

# Machine Learning Diploma

**Python3: Loops & Functions & File Handling**

**AMIT**

# Agenda

- Loops
- Functions
- Scopes
- Try & Except
- File Handling
- Pre-built & User Defined Modules

# 1. Loops

# 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.

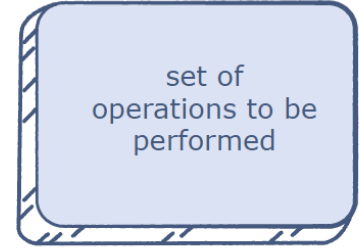
→ Syntax:

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.

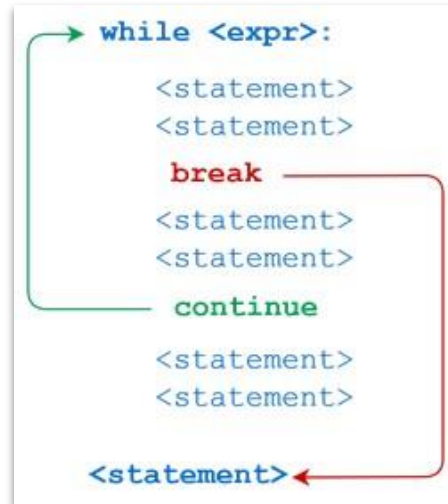
**Exit Conditions** are those where you modify the values you used in the loop condition so that the loop can be exited.

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



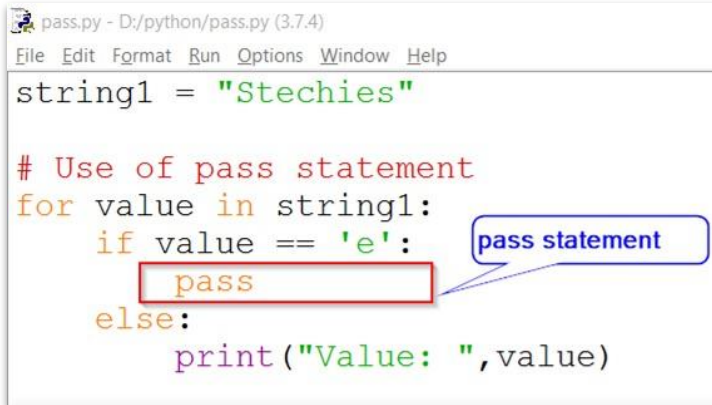
# Continue/Break keywords:

- 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 callout box labeled "pass statement" points to the `pass` keyword in the code.



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

A callout box labeled "Required Output" points to the output of the script.

## 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".

## Quiz (Solution):

→ 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

## Quiz (Solution):

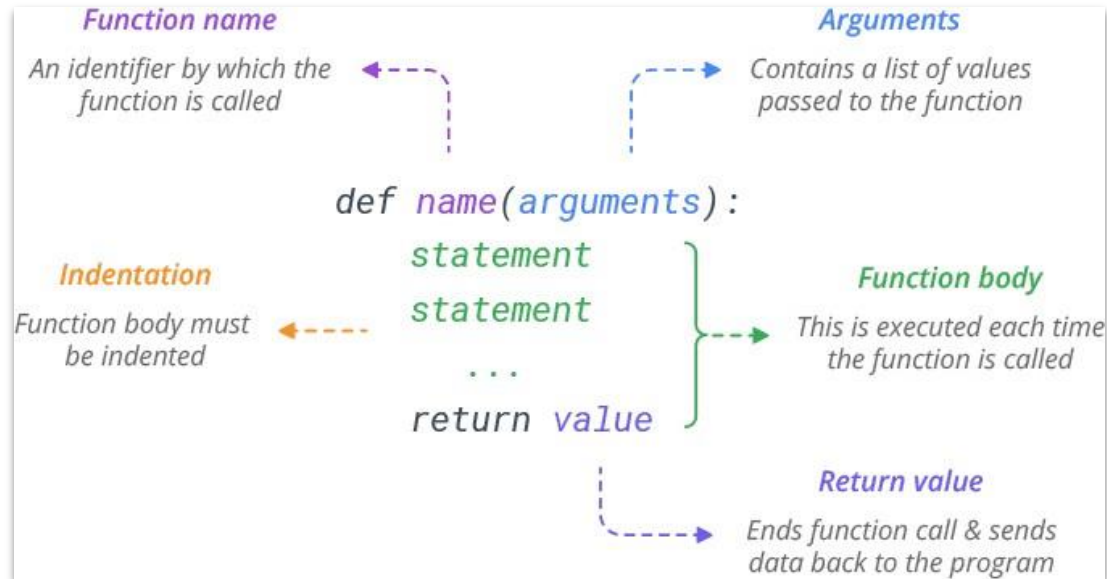
- Write a program that prompts a user for a two-word string and prints True if both words begin with same letter
- ('Levelheaded Llama') --> True
  - ('Crazy Kangaroo') --> False

```
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")
```

## 2. Functions

# 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:

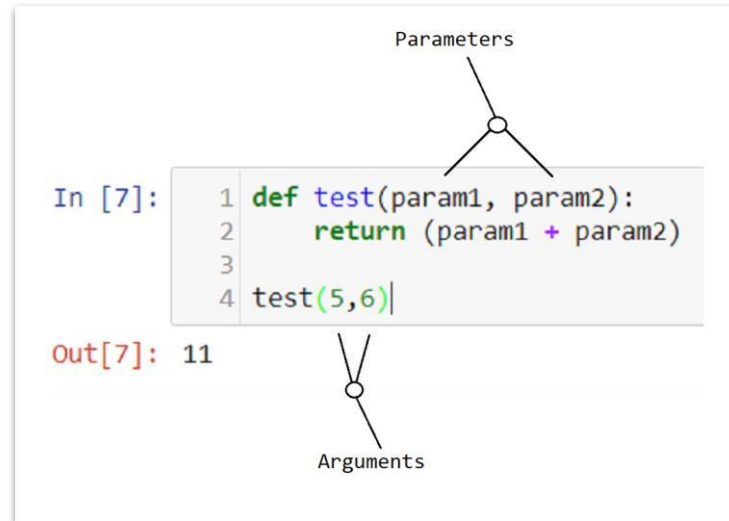
func\_name(arg\_1,arg\_2)

```
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

## Quiz (Solution):

- Transform the code you made for calculating the area of a circle into function that:
- Takes the radius and Pi value as inputs, assign a default of 3.14 for Pi
  - 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:

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

- ☐ 2
- ☐ 6
- ☐ 8
- ☐ 4



## Quiz (Solution):


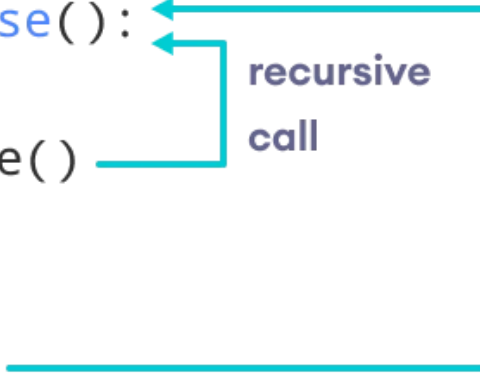
→ What is the output of the following program:

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

- ☐ 2
- ☒ 6
- ☐ 8
- ☐ 4

# 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:

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

## Quiz (Solution):

→ 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)
```

## 3. Scopes

## 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
```



## 4. Try .. Except

## 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
  - o IOError , ValueError , ImportError , EOFError , KeyboardInterrupt , ZeroDivisionError...
- [Link](#) to built-in exceptions

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

# Try.. Except Clause:

- Try: is the main code you want to handle
- Except: is the action you do if error exists
- Else: is the action you do if no error exists
- Finally: is the action you take regardless of errors or not.

```
try:  
    print(x)  
except NameError:  
    print("Variable x is not defined")  
except:  
    print("Something else went wrong")
```

```
try:  
    print("Hello")  
except:  
    print("Something went wrong")  
else:  
    print("Nothing went wrong")
```

```
try:  
    print(x)  
except:  
    print("Something went wrong")  
finally:  
    print("The 'try except' is finished")
```

## 5. File Handling

## 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.

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

[Link](#) to more file methods

## Quiz:

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

## Quiz (Solution):

→ Write a Python function 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.

## 6. 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'

Any Questions?



**THANK YOU!**

**AMIT**