Exercise

Exercise 1: *Using if statements, create a variable called day, set it to "Tuesday". Check to see if day is equal to "Monday"* ***or*** *"Tuesday", and if it is, print, "Today is sunny". If it is not, print "Today it will rain"*

Answer 1:

```
day = "Tuesday"
if day == "Tuesday" or day == "Monday":
        print("Today is sunny")
else:
        print("Today it will rain")
```

Exercise List

Q. Find the greatest number among two

Q. Check whether a number is +ve, -ve or zero

Q. Check whether a number is even or odd

Q. Find the grade of a student

marks	>	80	A	grade
marks	>	60	B	Grade
marks	>	40	C	grade
marks	<	40	D	grade

Q. Find the greatest number among three

Q. Check whether a year is leap year or not

if a year is divisible by 400 its a leap year

or

if a year is divisible by 4 and not divisible by 100 its a leap year

Q. Write a Python program to convert temperatures to and from celsius, fahrenheit.

60°C is 140 in Fahrenheit $C = (5 (F - 32)) / 9$

45°F is 7 in Celsius $F = (9C + (32 * 5)) / 5$

Exercise (2)

Q. Find the greatest number among two

```
a=10
b=5

if a>b:
   print("the greatest number=", a)
else:
   print("the greatest number=",b)
```

Exercise (3)

Q. Check whether a number is +ve, -ve or zero

```
a=90
if a>0:
    print ("number is +ve=",a)
elif a<0:
    print ("number is -ve=",a)
else:
    print("zero")
```

Exercise (4)

Q. Check whether a number is even or odd

```
a=10

if a%2==0:
    print ("number is even")
else:
    print("number is odd")
```

Exercise (5)

Q. **Find the grade of a student**

marks	>	80	A	grade
marks	>	60	B	Grade
marks	>	40	C	grade
marks	<	40	D	grade

```
a= int(input("Insert Student mark: "))

if a>=80:
   print("A Grade")
elif a>=60:
   print("B Grade")
elif a>=40:
   print("C Grade")
else:
   print("D Grade")
```

Exercise (6)

Q. Find the greatest number among three

```
a= 60
b= 70
c=100

if a>b and a>c:
    print("(a) in the gratest")
elif b>a and b>c:
    print("(b) in the gratest")
else:
    print("(c) in the gratest")
```

Exercise (7)

Q. **Check whether a year is leap year or not**

if a year is divisible by 400 its a leap year

or

if a year is divisible by 4 and not divisible by 100 its a leap year

```
a= int(input("insert the year:"))

if a%400==0 or a%4==0 and a%100!=0:
    print("it is a Leap year")
else:
    print("it's not a Leap year")
```

Exercise (8)

Q. Write a Python program to convert temperatures to and from celsius, fahrenheit.

60°C is 140 in Fahrenheit C = (5 (F - 32)) / 9

45°F is 7 in Celsius F = (9C + (32 * 5)) / 5

```
temp = input("Input the temperature you like to convert? (e.g., 45F, 102C etc.) : ")
degree = int(temp[:-1])
unit = temp[-1]

if unit.upper() == "C":
        result = int(round((9 * degree) / 5 + 32))
        n_unit = "Fahrenheit"
elif unit.upper() == "F":
        result = int(round((degree - 32) * 5 / 9))
        n_unit = "Celsius"
else:
        print("Input proper convention.")
print("The temperature in", n_unit, "is", result, "degrees.")
```

Python Basics

Lesson 05—Loops




Objectives

After completing this lesson, you will be able to:

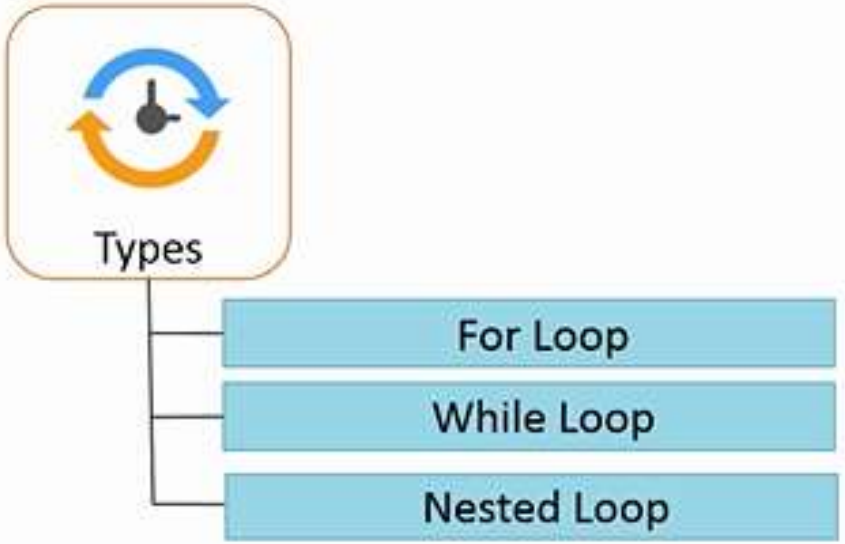
- Define loops and their types in Python
- Describe the range function
- Explain the break and continue statements in a loop

Loops in Python

Sometimes, you might want to repeat a given operation many times. Repeated executions like this are performed by (loops).

We will look at two types of loops, (for) loops and (while) loops.

A loop statement can execute a statement or group of statements many times.



Range Function

In Python, the range function is normally used with the loop statements, which provides a list of numbers, starting from zero to a given number minus one.

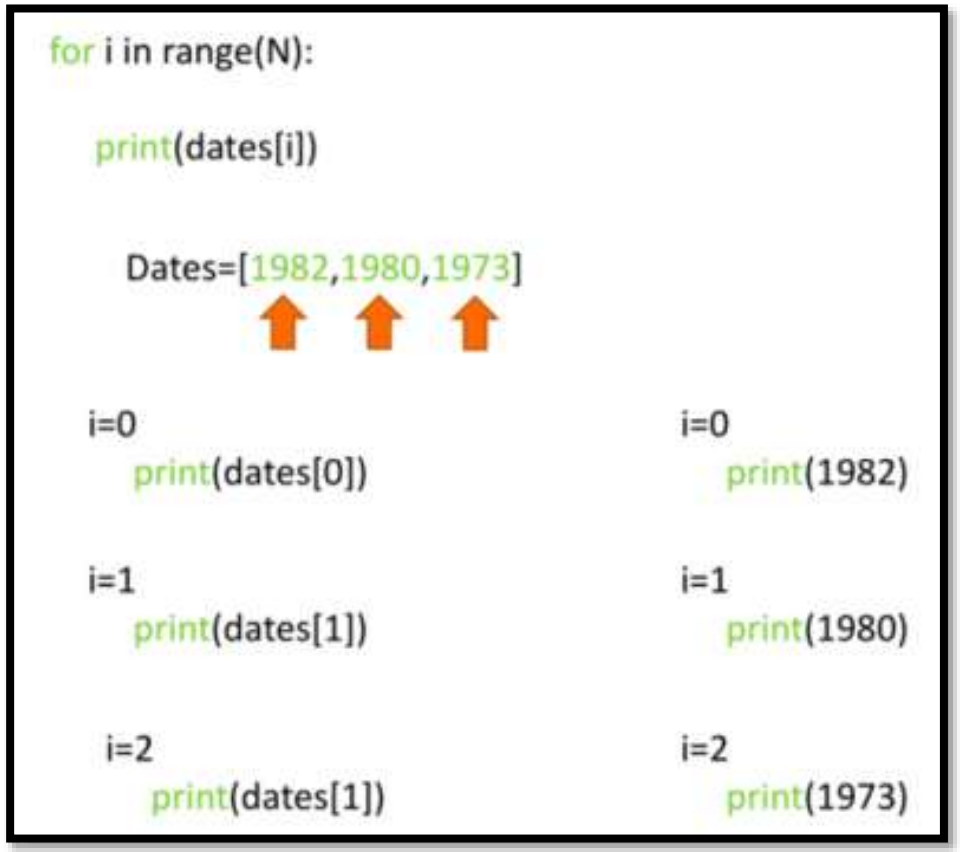
Example:

```
>>> range(5)
[0,1, 2, 3, 4]
```

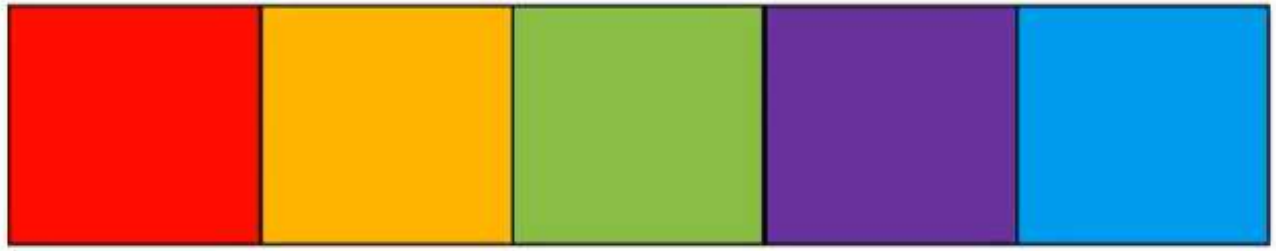
Range Function

The **for** loop enables you to execute a code block multiple times. For example, you would use this if you would like to print out every element in a list.

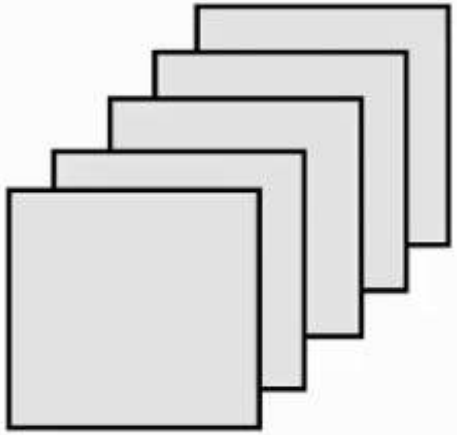
```
dates = [1982,1980,1973]
N=len(dates)

for i in range(N):

    print(dates[i])
```



Range Function



squares	red	yellow	green	purple	blue
	0	1	2	3	4



```
squares=["red", "yellow ", "green", "purple", "blue "]

for i in range(0,5):

    squares[i]="white"
```

For Loop

This type of loop runs an action repetitively. Let's view a few examples using the range function and using the "for" loop for a list.

Range Example

List Example

```
for x in range(10):
    print x
```

For Loop

As shown here, the “for” loop can also be executed for a list of strings.

Range Example | List Example

```
fruits = ['apple' , 'mango' , 'orange' , 'banana']
for fruit in fruits:
        print fruit
```

For Loops

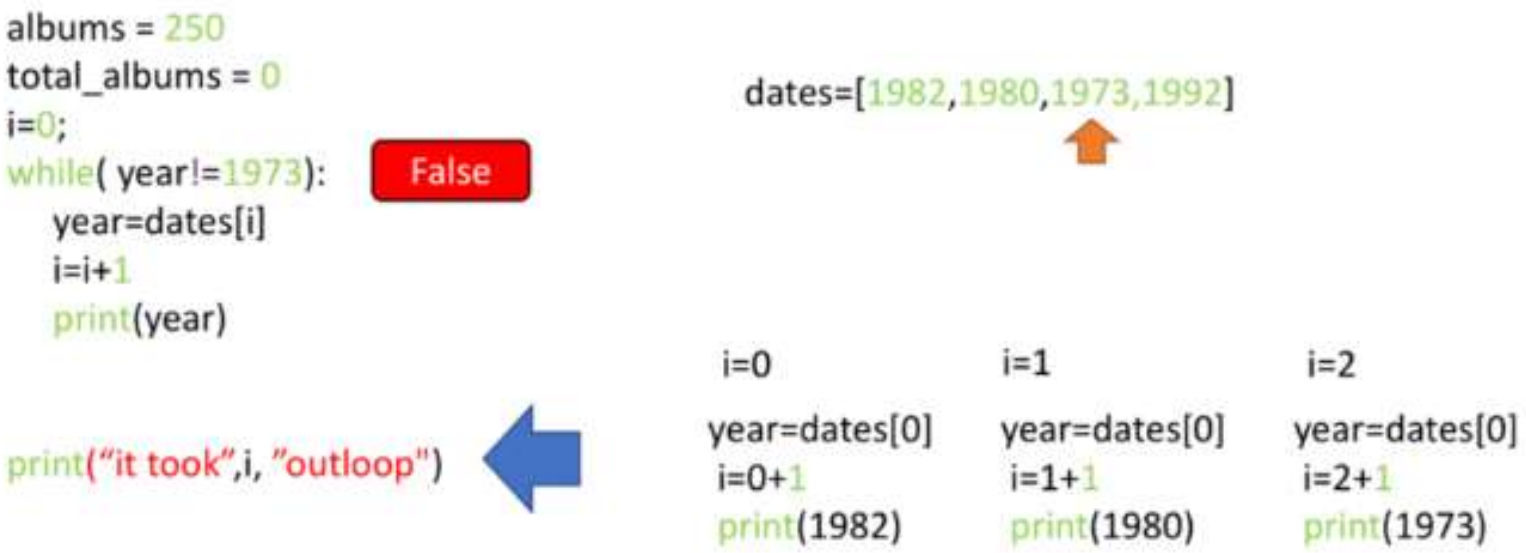
```
>>> for i in range(0,5):
        print(i)


0
1
2
3
4
>>> shoppingList = ["Milk", "Eggs", "Oranges"]
>>> for i in shoppingList:
        print(i)


Milk
Eggs
Oranges
```

```
>>> tup = (2,43,6)
>>> for i in tup:
        print(i)


2
43
6
```

While Loop

As you can see, the **for** loop is used for a controlled flow of repetition. However, what if we don't know when we want to stop the loop? What if we want to keep executing a code block until a certain condition is met? The **while** loop exists as a tool for repeated execution based on a condition. The code block will keep being executed until the given logical condition returns a **False** boolean value.

Let's say we would like to iterate through list **dates** and stop at the year 1973, then print out the number of iterations. This can be done with the following block of code:



While Loop

This type of loop keeps running a code until the base condition is satisfied.

Example:

```
x = 0
while x <= 100:
        print x
        x = x+1
```

The syntax of the while loop is: “while a condition is true, do this”.

While Loops

```
>>> counter = 5
\
>>> while counter < 10:
        print(counter)
        counter++

SyntaxError: invalid syntax
>>> while counter < 10:
        print(counter)
        counter = counter + 1


5
6
7
8
9
>>> while counter < 10:
        print(counter)


>>> counter
```

Nested For Loops

```
>>> for i in range (0,5):
        for a in range (0,5):
            print(a)


0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
>>>
```

Break Statements

Break Statements are used to exit loops.

These statements:

- are used with an “if” condition.
- are executed when they meet a condition, which is checked in the loop during every iteration.
- end the loop and execute the code written after the loop.

Example:

```
x = 1
while True:
print x
x = x+1
if x > 20:
        break
```

Continue Statements

Continue Statements help in skipping a specific iteration when a loop is being executed.

Example:

```
for x in range(100):
        if (x%2) == 0:
continue
        print x
```

Example

```
#Defult
print("normalStatement:")
n=10

for i in range(n):
    print(i)
print("the end")
print()
print("----------------------")
```

```
#continue
print("continue Statement:")
n=10

for i in range(n):
    if i==7:
        continue
    print(i)
print("the end")
print()
print("----------------------")
```

```
#Break Statement

print("break Statement:")
n=10

for i in range(n):
    if i==7:
        break
    print(i)
print("the end")
print()
print("----------------------")
```

```
#pass
print("pass Statement:")
n=10

for i in range(n):
    if i==7:
        pass
    print(i)
print("the end")
```

Quiz

Quiz 1	The output of “range (5)” will be:

- a. [0, 4]
- b. [0, 1, 2, 3, 4]
- c. [1, 2, 3, 4, 5]
- d. [1, 5]

Quiz 2	What is the purpose of a break statement?

- a. It exits the control from a loop.
- b. It helps in skipping an iteration.
- c. It slices down a loop in iterations.
- d. It holds a place keeper.

Quiz 3	What is the purpose of a continue statement?

- a. It exits the control from a loop.
- b. It helps in skipping an iteration.
- c. It slices down a loop in iterations.
- d. It holds a place keeper.

Quiz 4	How many times will the following loop run? *X =2* *while x < 100: x += 1*

- a. 50
- b. 51
- c. 49
- d. 98

Exercise

Exercise 1: *Create a loop that prints out either even numbers, or odd numbers all the way up till your age. Ex: 2,4,6,….,14*

Exercise 2: *The weight of a person on the moon is 1/6th the weight of a person standing on earth. Say that your weight on earth increases by 1 kg every year. Write a program that will print your weight on the moon every year for the next 10 years. (Your initial weight can be anything.)*

Exercise-answer

Answer 1:

```
for i in range(0,15,2):
        print(i)
```

Answer 2:

```
weight = 60.0
for i in range(0,11):
    moonweight = weight / 6
    print(moonweight)
    weight = weight + 1
```

Exercise list

Q. Find the even numbers from 1 to n

Q. Find the sum of the first n numbers

Q. Find the factors of a number

6 -> 1,2,3,6

Q. Count the vowels inside the string

a e i o u

Q. Count number of words inside the string

Technology Academy

Q. Reverse the string

Technology Academy

Q. Write a program that prints the pattern to the right using functions

```
**********
*        *
*        *
*        *
*        *
**********
>>> |
```

Exercise (2)

Q. Find the even numbers from 1 to n

```
a=1
n=10

while a<=n:
    if a%2==0:
        print(a,"number is even")
    else:
        print(a,"number is odd")
    a=a+1
```

```
n=10

for i in range(n):
    if i%2==0:
        print(a,"number is even")
    else:
        print(a,"number is odd")
```

Exercise (3)

Q. Find the sum of the first n numbers

```
a=1
n=5
s=0

while a<=n:
    s=s+a
    a=a+1

print ("Sum=",s)
```

```
n=10
s=0

for i in range(n+1):
   s=s+i

print(s)
```

Exercise (4)

Q. Find the factors of a number

6 -> 1,2,3,6

```
#factors of a number

a=int(input("Insert The Number:"))
b=1

while b<=a:
    if a%b==0:
        print (b)
    b=b+1
```

```
n=10
print(n)

for i in range(1,n+1):
    if n%i==0:
        print(i)
```

Exercise (5)

Q. Check whether a number is prime or not

```
n=5
c=0

for i in range(1,n+1):
    if n%i==0:
        print(i)
        c=c+1

if c==2:
    print( "prime num")
else: print ("not prime")
```

Exercise (6)

Q. Find the prime numbers from 1 to n

```
n= int(input ("enter the number"))
for i in range(1,n+1):
    c=0
    for j in range(1,i+1):
        if i%j==0:
            c+=1
    if c==2:
        print (i)
```

Exercise (7)

Q. Count the vowels inside the string

a e i o u

```
print("Q1:")
data="aptech computer"
print(data)
c=0

for i in data:
    if i=='a' or i=='e' or i=='i' or i=='o' or i=='u':
        c=c+1

print("number of vowels:",c)

print()
```

Exercise (8)

Q. Count number of words inside the string

Mohammed Marwan

```
print("----------------------------------------")

print("Q2:")
c=1
data = “Mohammed Marwan  Mohammed Marwan
Mohammed"
print(data)

for i in data:
    if i==' ':
        c=c+1

print("Number of words:",c)
```

Exercise (9)

Q. Reverse the string **Technology Academy**

ymedacA ygolonhceT

Exercise (9)

Q. Reverse the string

```
print("Q1")
data="mohamed marwan"
rev=""
c=-1

for i in data:
    print(data[c],end="")
    rev=rev + data[c]
    c=c-1

print()
print(rev)

print()
```

Exercise (10)

Write a program that prints the pattern to the right using functions

```
**********
*        *
*        *
*        *
*        *
**********
>>>
```

Exercise (10)

Write a program that prints the pattern to the right using functions

```
n=10

for i in range (n):

    if i ==0 or i==(n-1):
        print("********")

    else:
        print("*          *")
```

```
**********
*        *
*        *
*        *
*        *
**********
>>>
```

Python Basics

Lesson 06 : Functions




Objectives

After completing this lesson, you will be able to:

- Define a function and its advantages
- Explain the procedures to create and call a function

Introduction to Functions

- Consider the two lines of code in **Block 1** and **Block 2**: the procedure for each block is identical. The only thing that is different is the variable names and values.

```
a1=4
b1=5
c1=a1+b1+2*a1*b1-1
if(c1<0):
   c1=0
else:
   c1=5
c1
```

```
a2=0
b2=0
c2=a2+b2+2*a2*b2-1
if(c2<0):
   c2=0
else:
   c2=5
c2
```

We can replace the lines of code with a function. A function combines many instructions into a single line of code. Once a function is defined, it can be used repeatedly. You can invoke the same function many times in your program. You can save your function and use it in another program or use someone else's function.

Introduction to Functions

The lines of code in code block 1 and code block 2 can be replaced by the following function:

```
def Equation(a,b):
    c=a+b+2*a*b-1
    if(c<0):
        c=0
    else:
        c=5
    return(c)
```

This function takes two inputs, a and b, then applies several operations to return c. We simply define the function, replace the instructions with the function, and input the new values of a1,b1 and a2,b2 as inputs.

Introduction to Functions

A function is a reusable block of code which performs operations specified in the function. They let you break down tasks and allow you to reuse your code in different programs.

There are two types of functions :

1) Pre-defined functions
2) User defined functions

- Functions are an essential part of the Python programming language
- Function is a piece of code written to carry out a specified task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.
- It avoids repetition and makes code reusable.

Function Syntax

Syntax of Function

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

- **Keyword def** marks the start of function header.
- **A function name** to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
- **Parameters (arguments)** through which we pass values to a function. They **are optional**.
- **A colon (:)** to mark the end of function header.
- **Optional documentation string (docstring)** to describe what the function does.
- One or more valid **python statements** that make up the function body. Statements must have same indentation level.
- An **optional return statement** to return a value from the function.

Creating Function

keyword name Parameters

```
def add(a):
    """
    add 1 to a
    """
    b=a+1;
    print(a, " if you add one",b)
    return(b)

add(1)
```

Documentation

body

Calling Functions

Example:

>>>hello_func()

If present, pass the arguments inside brackets while keeping the original sequence. A function should be created before it is used in a program.

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

Flow of Execution with Functions

Code

```
def hello():
    print ("Hi there!")
    print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Execution with Functions

Code

```
def hello():
    print ("Hi there!")
    print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
And now we're done.
```

Arguments and Return Statement

What should be done when some information is required for a function?

In such situations, pass arguments in the function and use a "return" statement.

Example:

```
def add_numbers(x,y):
        return x+y
```

You can also set a default value of an argument.

Example:

```
def add_numbers(x,y=2):
        return x+y
```

Return Statement

- The return statement is used to exit a function and go back to the place from where it was called.
- This statement can contain expression which gets evaluated and the value is returned.
- If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

```
def MJ():
   print('Mohammed Marwan')

def MJ1():
   print('Mohammed Marwan')
   return(None)
```

Creating our own Functions

```
>>> def functionName():
        for i in range(0,5):
            print("Hi")


>>> functionName()
Hi
Hi
Hi
Hi
Hi
>>> functionName
<function functionName at 0x4b2d80c>
>>> functionName()
Hi
Hi
Hi
Hi
Hi
```

```
>>> def addNum(firstNum, secondNum):
        return (firstNum + secondNum)

>>> addNum(45, 3)
48
>>>
```

Variables

- The input to a function is called a formal parameter.

- A variable that is declared inside a function is called a **local variable**. The parameter only exists within the function (i.e. the point where the function starts and stops).

- A variable that is declared outside a function definition is a **global variable**, and its value is accessible and modifiable throughout the program.

```
def square(a):
    """
    Square the input add add 1
    """
    c=1
    b=a*a+c;
    print(a, " if you square+1 ",b)
    return(b)

x=2;
z= square(x)
```

Formal parameter

Function definition

Local variable

Main program code

Local Variable & Global Variable

- Normal variable is a global variable
- The variable inside the function can not be called as it's a local variable

```
>>> total = 10
>>> def multiply(num1, num2):
        total = num1 * num2
        return total

>>> multiply(10, 23)
230
>>> total
10
>>> def multiply(num1, num2):
        totall = num1 * num2
        return totall

>>> multiply(10, 23)
230
>>> totall
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    totall
NameError: name 'totall' is not defined
>>>
```

Pre-defined functions

There are many pre-defined functions in Python, so let's start with the simple ones.

- The **print()** function:

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
print(album_ratings)
```

- The **sum()** function adds all the elements in a list or tuple:

```
sum(album_ratings)
```

- The **length** function returns the length of a list or tuple:

```
len(album_ratings)
```

Using if/else statements and loops in functions

The **return()** function is particularly useful if you have any IF statements in the function, when you want your output to be dependent on some condition:

```
def type_of_album(artist,album,year_released):
    if year_released > 1980:
        print(artist,album,year_released)
        return "Modern"
    else:
        print(artist,album,year_released)
        return "Oldie"

x = type_of_album("Michael Jackson","Thriller",1980)
print(x)
```

Using if/else statements and loops in functions

We can use a loop in a function. For example, we can **print** out each element in a list:

```
def PrintList(the_list):
    for element in the_list:
        print(element)

PrintList(['1',1,'the man',"abc"])
```

Setting default argument values in your custom functions

You can set a default value for arguments in your function. For example, in the **`isGoodRating()`** function, what if we wanted to create a threshold for what we consider to be a good rating? Perhaps by default, we should have a default rating of 4:

```
def isGoodRating(rating=4):
    if(rating < 7):
        print("this album sucks it's rating is",rating)

    else:
        print("this album is good its rating is",rating)

isGoodRating()
isGoodRating(10)
```

Global variables

So far, we've been creating variables within functions, but we have not discussed variables outside the function. **These are called global variables.**

Let's try to see what **printer1** returns:

```
artist = "Michael Jackson"
def printer1(artist):
    internal_var = artist
    print(artist,"is an artist")

printer1(artist)
```

If we print **internal_var** we get an error.
We got a **Name Error: name 'internal_var' is not defined.** Why?
It's because all the variables we create in the function is a **local variable**, meaning that the variable assignment does not persist outside the function.

Global variables

But there is a way to create **global variables** from within a function as follows:

```
artist = "Michael Jackson"

def printer(artist):
    global internal_var
    internal_var= "Whitney Houston"
    print(artist,"is an artist")

printer(artist)
printer(internal_var)
```

Quiz

Exercise

- Write a Python function, square, that takes in one number and returns the square of that number
- Find the greatest number among three
- Find the sum of the first n numbers
- Write a function to swap (a=12,b=6)

Exercise(2)

- Find the greatest number among three

```
def greatest (a,b,c):
    ''' Find the greatest number among three'''
    if a>b and a>c:
        print("the greatest:" ,a)
    elif b>a and b>c:
        print("the greatest:" ,b)
    elif c>a and c>b:
        print("the greatest:" ,c)


greatest(x,y,z)
```

```
def greatest (a,b,c):
    ''' Find the greatest number among three'''
    if a>b and a>c:
        return(a)
    elif b>a and b>c:
        return(b)
    elif c>a and c>b:
        return(c)


Print(greatest(x,y,z))
```

Exercise(3)

- Find the sum of the first n numbers

```
def sum (a):

    '''Find the sum of the first n numbers'''

    n=a
    b=0
    c=0
    while b<n:
        b=b+1
        c=c+b
    print("sum of the first n numbers",c)

sum(5)
```

Exercise (4)

- Add items to the list

```
def add (item_list,a):
    '''Add items to the list'''
    #item_list=item_list+[a]
    item_list.append(n)


lst=[1,2,3,4,5]
n=90
print (add.__doc__)

add(lst,n)

print (lst)
```

Python Basics

Lesson 07 —Classes

Objectives

After completing this lesson, you will be able to:

- Define a class and its advantages
- Describe the method to create a class
- Describe the method to add functions to a class
- Describe the built-in class attributes

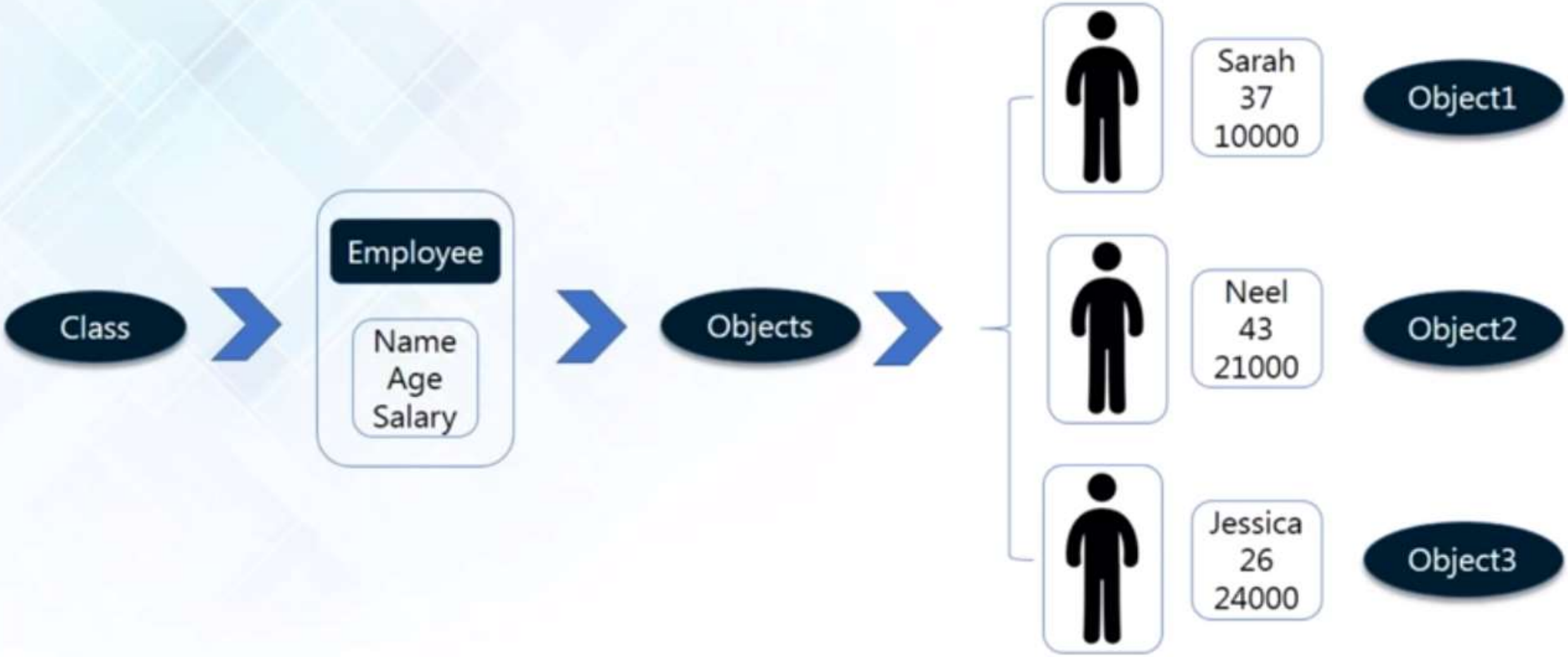
Classes

These are user-defined data types, which are:

- Used as a blueprint to create multiple objects
- Made of variables and functions
- Used to depict real-world objects in programing languages

Variables in a Class	Represent the properties of a real-world entity (such as a car); **Examples**: model number, build year, brand name
Functions in a Class	Represent the possible activities performed by an entity; **Examples**: start(), stop(), apply break()

Classes & Objects



Objects

These are variables of classes. You can create multiple objects of a class and each object can have different values for its variables.

Examples:

- *car1 = Car()*
- *car2 = Car("Test model" , "Test Brand")*

Creating a Basic Class

For example, let's learn to create a class called "Vehicle".

Example:

```
class Vehicle:
        model_name = "model x"
        brand_name =  "Brand y"
```

An object of this class will be:

```
>>>car1 = Vehicle()
```

Accessing Variable of a Class

Once an object is created, the dot operator is used to access the values stored in different variables of that object.

Print the "model name" variable of the "car1" object

Example:

>>> print car1.model_name

output : model x

Print a brand name

Example:

>>> print car1.brand_name

output : brand y

Adding Function to a Class

Functions add behavior to a class. Here's an example to add a function to a class:

Example:

```
class Vehicle:
    model_name = "model x"
    brand_name =  "Brand y"
    def start(self):
                        print "Starting Vehicle..."
    def stop(self):
                        print "Stop Vehicle..."
```

Built-in Class Attributes

Every Python class follows built-in attributes, which can be accessed using the dot operator like any other attribute. The various types of built-in attributes are:

__dict__	__doc__	__name__	__module__	__bases__
Provides a dictionary containing the class's namespace	Provides a class documentation string or nothing, if undefined	Provides a class name	Gives the module name in which the class is defined ("__main__" in interactive mode)	Returns a possibly empty tuple containing the base classes

__Init__Function

It is a constructor of a class.

It:

- Is called in a code when an object is created
- Allocates space for objects
- Initializes the variables of a class

Example of Defining and Using a Class

To understand better how to define and use a class, another example of the class "Student" is given here.

Defining a Class | Using that Class

Example:

```
class Student(object) :
        name = ""
        pass = True
        def __int__(self, name):
            self.name=name
        def is_pass(self):
        return self.pass
```

Example of Defining and Using a Class -Cont

The code below uses the class "Student".

Defining a Class | Using that Class

Example:

```
## Defining object
student1 = Student("David")
##Using the object
print student1.is_pass()
Output : True
```

Example

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

    def showname(self) :
        print self.name

    def showsalary(self) :
        print self.salary


emp1 = Employee("Harry" , 100000)
emp2 = Employee("John" , 200000)

print emp1.name
print emp2.name

emp1.showsalary()
emp2.showsalary()

emp1.showname()
emp2.showname()
```

training@Satellite-A100: ~

File Edit Tabs Help

```
training@Satellite-A100:~$ python Desktop/classes.py
Harry
John
100000
200000
training@Satellite-A100:~$ python Desktop/classes.py
Harry
John
100000
200000
Harry
John
training@Satellite-A100:~$
```

- We can use classes to define objects and attributes.

```
>>> class ClassName:
        pass

>>> instance = ClassName()
>>> class Students:
        def __init__(self, name, age, grade):
            self.name = name
            self.age = age
            self.grade = grade


>>> student1 = Students("Bob", 12, "7th")
>>> student1.name
'Bob'
>>> student1.age
12
>>> student1.grade
'7th'
\
```

```
>>> class Students:
        def __init__(self, name, age):
            self.name = name
            self.age = age
        def displayStudent(self):
            return("Student name is " + self.name + " and age is " + str(self.age))


>>> Stu = Students("Chad", 15)
>>> Stu.displayStudents()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    Stu.displayStudents()
AttributeError: 'Students' object has no attribute 'displayStudents'
>>> Stu.displayStudent()
'Student name is Chad and age is 15'
>>>
```

```
>>> class Parent:
        counter = 10
        def __init__(self):
                print("Class initialized.")
        def parentFunc(self):
                print("ParentFunc being called")
        def setCounter(self, num):
                Parent.counter = num
        def showCounter(self):
                print(str(Parent.counter))


>>> class Child:
        def __init__(self):
                print("Child class being initialized")
        def childFunc(self):
                print("Child func being called")
```

```
>>> c = Child()
Child class being initialized
>>> c.childFunch()
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    c.childFunch()
AttributeError: 'Child' object has no attribute 'childFunch'
>>> c.childFunc()
Child func being called
>>> c.parentFunc()
ParentFunc being called
>>> c.counter
10
>>> c.setCounter(20)
>>> c.showCounter()
20
>>>
```

```
>>> class Parent:
        def func(self):
            print("This is a parent function")


>>> class Child(Parent):
        def func(self):
            print("This is a child function")


>>> c = Child()
>>> c.func()
This is a child function
>>>
```

Quiz

Quiz 2	What are the features of the __init__ function? *Select all that apply.*

- [ ] a. It is a Python function that can be used outside classes.
- [ ] b. It is a class constructor.
- [ ] c. It is a private class function.
- [ ] d. It is an initializing method.

Quiz 3	Which of these shows the correct way of creating an object?

- a. Student s = new Student()
- b. s = new Student()
- c. s = Student()
- d. s = Student(s)

Quiz 4	Which of the following statements define a class correctly? *Select all that apply.*

- [ ] a. A class is an encapsulation of variables and functions.
- [ ] b. A class can only have functions.
- [ ] c. A class is a user-defined data type.
- [ ] d. A class is a real world variable.

Exercise (1)

1) Write a Python class (mail) and pass (first name) & (last name)
Ex. First name = mohammed , last name = shahin ========➔ mohammedshahin@etisalat.ae

2) Write a Python class (Rectangle) constructed by a length and width and a method which will compute the area of a rectangle.

3) Write a Python class (Circle) constructed by a radius and two methods which will compute the area and the perimeter of a circle.

Exercise (1)

Write a Python class (mail) and pass (first name) & (last name)
Ex. First name = mohammed , last name = shahin ========➔ mohammedshahin@etisalat.ae

```
class mail:
        name_first=" "
        name_last=" "

        def display(self):

                return (self.name_first + self.name_last+"@etisalat.ae")

stu1=mail()
stu1.name_first='mohammed'
stu1.name_last='shahin'
print(stu1.display())
```

```
class mail:
      def __init__ (self,name_first,name_last):
            self.name_first=name_first
            self.name_last=name_last

      def display(self):
            return (self.name_first + self.name_last+"@etisalat.ae")

stu1=mail('mohammed',"shahin")
print(stu1.display())
```

Exercise (2)

Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

```
class Circle():
        def __init__(self, r):
                self.radius = r

        def area(self):
                return self.radius**2*3.14

        def perimeter(self):
                return 2*self.radius*3.14

NewCircle = Circle(8)
print(NewCircle.area())
print(NewCircle.perimeter())
```

Exercise (3)

Write a Python class named Rectangle constructed by a length and width and a method which will compute the area of a rectangle.

```
class Rectangle():

        def __init__(self, l, w):
                self.length = l
                self.width = w
        def rectangle_area(self):
                return self.length*self.width


newRectangle = Rectangle(12, 10)
print(newRectangle.rectangle_area())
```