

Python Fundamentals for Beginners

Great Learning Academy

Proprietary content. ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.
This file is meant for personal use by banisetti.10731577@imadtree.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Agenda

- Importance and Applications of Programming Languages
- Variables in Programming
- Decision Making Statements
- Looping Statements
- Functions in Programming
- Object Oriented Programming Concepts

Agenda

- Algorithmic Approach to solve a problem
- Introduction to Python
- Introduction to Colab
- Variables and Datatypes in Python
- Operators in Python
- Tokens in Python
- Strings in Python

Agenda

- Python Tuples
- List in Python
- Dictionary in Python
- Set in Python
- If Statement in Python
- Looping Statements in Python

Agenda

- Functions in Python
- Intro to Object Oriented Programming in Python
- Creating the First Class in Python
- Adding Parameters to a Class Method
- Creating a Class with Constructor
- Inheritance in Python
- Types of Inheritance

How do humans communicate?



hello

Holá

Bonjour

Grammar in Language

Every Language has Grammar associated with it.



I am Sam



Am Sam I

Language for Computers



Java Python
C++

Syntax for Computer Language

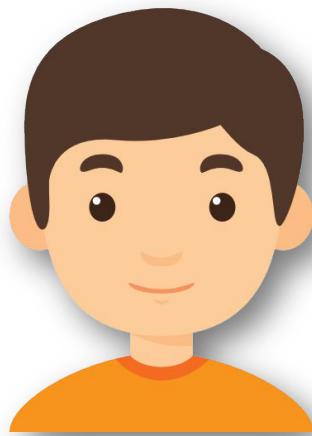
```
import pandas as  
pd
```



```
pandas import pd  
as
```

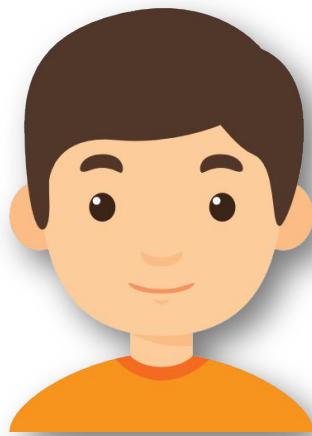


Why do we need Programming?

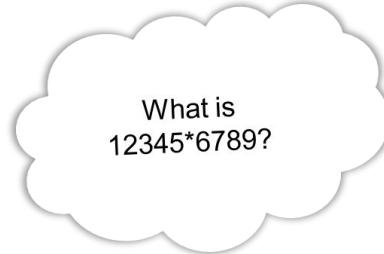
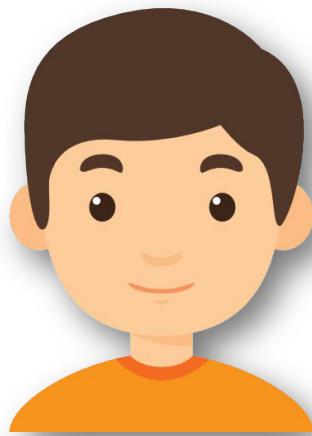


What is $2*2$?

Why do we need Programming?



Why do we need Programming?



Applications of Programming Languages



Gaming

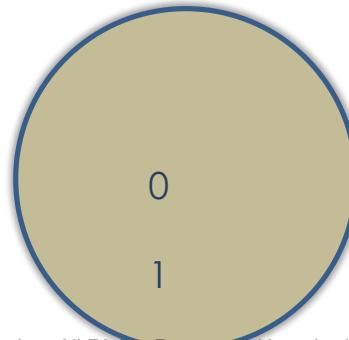
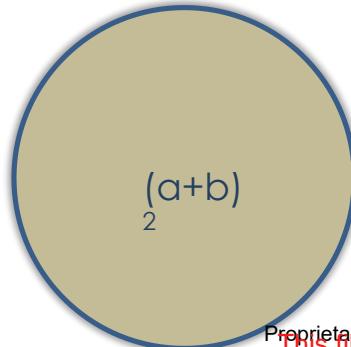
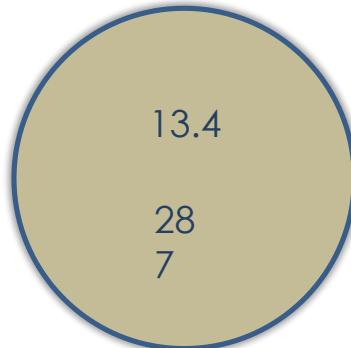


Banking



Machine Learning

What is Data?



How to Store Data?



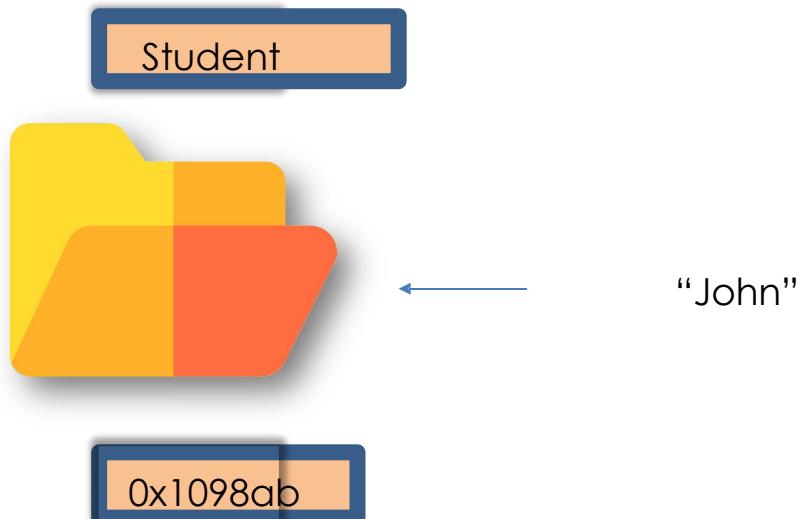
“John”

123

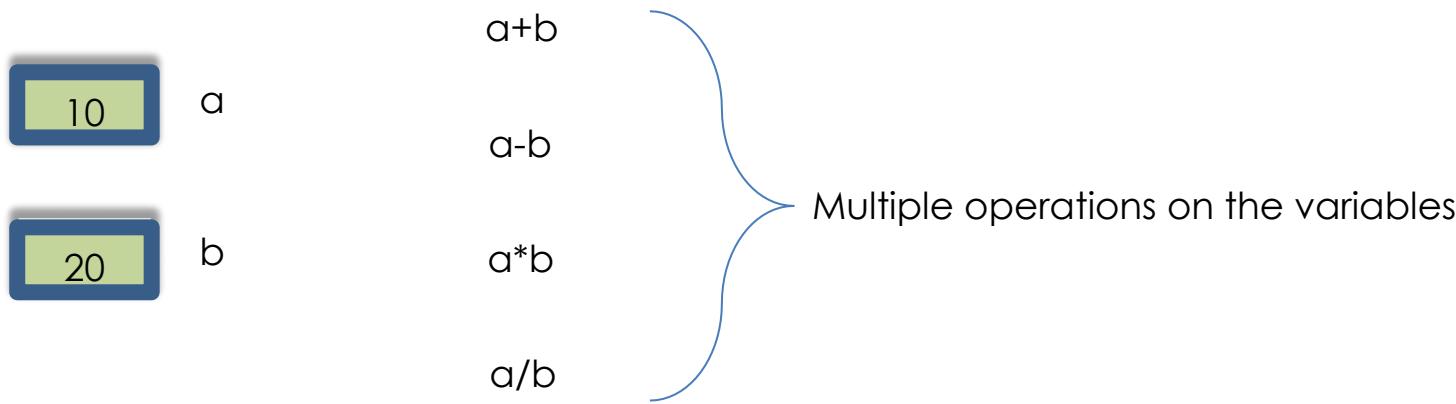
TRUE

Need of Variables

Data/Values can be stored in temporary storage spaces called variables.



Example of Variable



Decision Making Statements

If

It's raining: Sit inside

else

Go out and Play Football



Decision Making Statements

If
Marks > 70:
Get
Ice-cream



else
Give Practice
Test



If...Else Pseudo Code

```
If(condition){  
    Statements to be executed....  
}
```

```
else{  
    Statements to be executed....  
}
```

Looping Statements

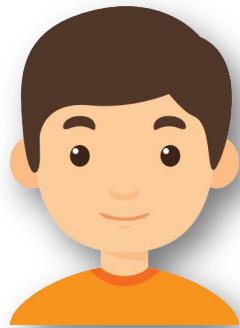
Looping statements are used to repeat a task multiple times.



Looping Statements



Looping Statements



While Loop Pseudo Code

```
while(TRUE){  
    Keep executing  
    statements....  
}
```

Functions in Real Life



Eating



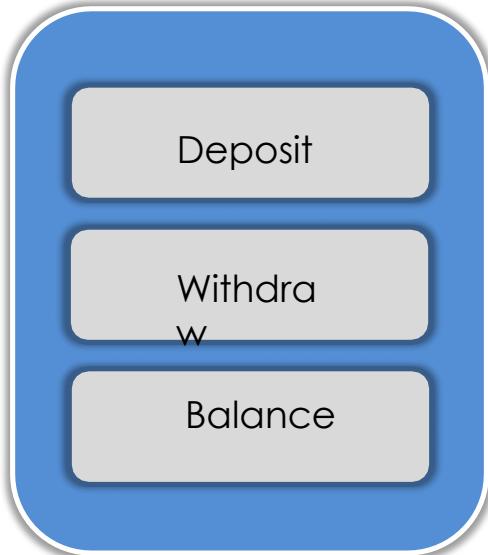
Running



Cycling

Functions in Programming World

Function is a block of code which performs a specific task.



Function to deposit money

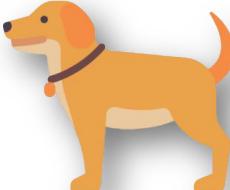
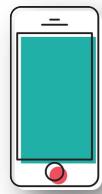
Function to withdraw money

Function to check balance

Object Oriented Programming Concepts

Importance and Applications of Programming Languages

Object Oriented Programming



Classes

Class is a template/blueprint for real-world entities.



Properties

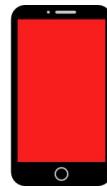
- Color
- Cost
- Battery Life

Behavior

- Make Calls
- Watch Videos
- Play Games

Objects

Objects are specific instances of a class.



Apple

Motorola

Samsung

Algorithmic Approach to Solve a Problem



Step by Step Approach

- 1 Check if you have all the ingredients
- 2 Take lemon and cut into two halves
- 3 Squeeze lemon into a glass of water
- 4 Add sugar and stir it well
- 5 Serve it cold with ice cubes

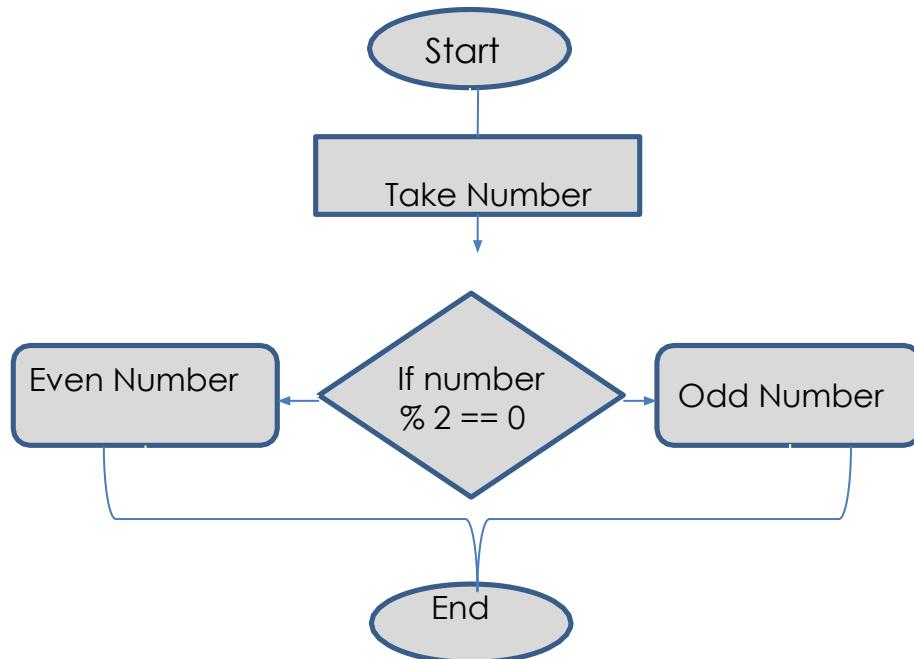


What is an Algorithm?

Step by step approach to solve a problem is known as algorithm.



Algorithm to find if number is Even/Odd



Introduction to Python

Cross-Platform
Compatible

Free & Open Source



Large Standard
Library

Object Oriented

Introduction to Google Colab

- Google Colab is a product of Google which is basically a free notebook environment that runs fully in the cloud.
- Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.
- More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.



Introduction to Google Colab

Why is it required?

- Google is quite involved in the Artificial Intelligence research and development part. One of the most popular AI frameworks was made by Google called Tensorflow and also a development tool called Collaboratory.
- Google Collaboratory was made free for public use by Google to encourage the AI enthusiast and the community well.
- Collaboratory is now publicly known as Google Colab or Colab.

Introduction to Google Colab

Features of Google Colab

- Write and execute code in Python
- Document the code which supports the mathematical equations
- Create new notebooks
- Upload the existing notebooks
- Share the notebooks with the google link
- Import data from Google drive
- Save notebooks from/to Google drive

Introduction to Google Colab

Types of runtime in Colab

- We have in total three types of runtime in Google Colab:-
- CPUs
- GPUs – these are basically optimized to train artificial intelligence and deep learning models. GPU can process multiple computations simultaneously.
- TPUs

Variables in Python



“John”

“Sam”

“Matt”

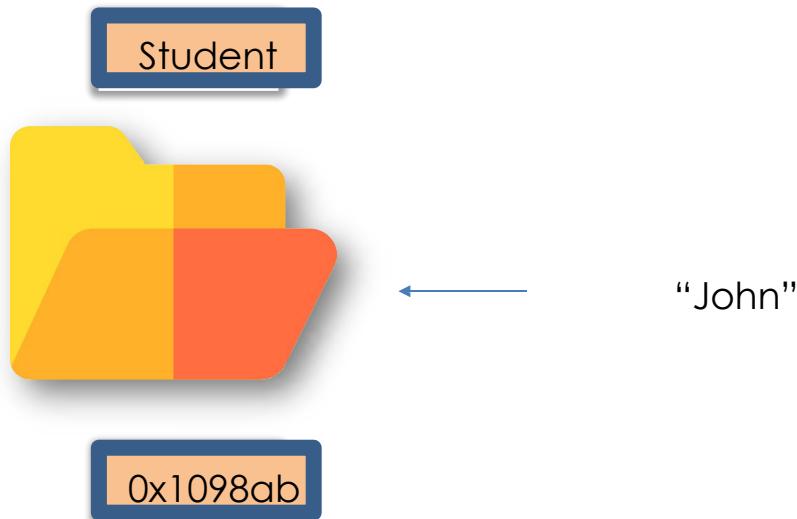
Variables in Python

Data/Values can be stored in temporary storage spaces called variables.



Variables in Python

Data/Values can be stored in temporary storage spaces called variables.



Data Types in Python

Every variable is associated with a data-type.

10, 500

3.14, 15.97

TRUE, FALSE

“Sam”, “Matt”

int

float

Boolean

String

Operators in Python

Arithmetic Operators

Relational Operators

Logical Operators/



Python Tokens

Smallest meaningful Component in a Program



Keywords

Identifiers

Literals

Operators

Python Keywords

Keywords are special reserved words.

False	class	Finally	Is	Return
None	continue	For	Lambda	Try
True	def	From	Nonlocal	While
and	del	Global	Not	With
as	elif	If	Or	Yield

Python Identifiers

Identifiers are names used for variables, functions or objects.

Rules

No special character expect (underscore)

Identifiers are case sensitive

First letter cannot be digit

Python Literals

Literals are constants in Python.



I'm a constant.
I don't change

Python Strings

Strings are sequence of characters enclosed within single quotes(' '), double quotes(" ") or triple quotes(""" """).

'Hello World'

"This is Sparta"

''' I am going to
France tomorrow'''

Extracting individual characters

```
In [23]: my_string="My name is John"
```

```
In [24]: my_string[0]
```

```
Out[24]: 'M'
```

```
In [5]: my_string="My name is John"
```

```
In [6]: my_string[-1]
```

```
Out[6]: 'n'
```

String Functions

Finding length of string

```
In [28]: len(my_string)  
Out[28]: 15
```

Converting String to uppercase

```
In [31]: my_string.upper()  
Out[31]: 'MY NAME IS JOHN'
```

Converting String to lowercase

```
In [30]: my_string.lower()  
Out[30]: 'my name is john'
```

String Functions

Replacing a substring

```
In [33]: my_string.replace('y','a')  
Out[33]: 'Ma name is John'
```

Number of occurrences of substring

```
In [7]: new_string = "hello hello world"  
In [8]: new_string.count("hello")  
Out[8]: 2
```

String Functions

Finding the index of substring

```
In [13]: s1 = 'This is sparta!!!'  
s1.find('sparta')  
  
Out[13]: 8
```

Splitting a String

```
In [15]: fruit = 'I like apples, mangoes, bananas'  
fruit.split(',')  
  
Out[15]: ['I like apples', ' mangoes', ' bananas']
```

Data Structures in Python

Tuple

List

Dictionary

Set



List in Python

List is an ordered collection of elements enclosed within [].



Lists are
mutable

```
l1=[1,'a',True]
```

Indexing the list elements

```
In [58]: l1=[1,"a",2,"b",3,"c"]  
l1[1]
```

```
Out[58]: 'a'
```

```
In [59]: l1=[1,"a",2,"b",3,"c"]  
l1[2:5]
```

```
Out[59]: [2, 'b', 3]
```

Modifying a List

Changing the element at 0th index

```
In [35]: l1=[1,"a",2,"b",3,"c"]
l1[0]=100
l1
```

```
Out[35]: [100, 'a', 2, 'b', 3, 'c']
```

Popping the last element

```
In [37]: l1=[1,"a",2,"b",3,"c"]
l1.pop()
l1
```

```
Out[37]: [1, 'a', 2, 'b', 3]
```

Appending a new element

```
In [36]: l1=[1,"a",2,"b",3,"c"]
l1.append("Sparta")
l1
```

```
Out[36]: [1, 'a', 2, 'b', 3, 'c', 'Sparta']
```

Modifying a List

Reversing elements of a list

```
In [40]: l1=[1,"a",2,"b",3,"c"]
l1.reverse()
l1
```

```
Out[40]: ['c', 3, 'b', 2, 'a', 1]
```

Sorting a list

```
In [43]: l1 = ["mango","banana","guava","apple"]
l1.sort()
l1
```

```
Out[43]: ['apple', 'banana', 'guava', 'mango']
```

Inserting element at a specified index

```
In [41]: l1=[1,"a",2,"b",3,"c"]
l1.insert(1,"Sparta")
l1
```

```
Out[41]: [1, 'Sparta', 'a', 2, 'b', 3, 'c']
```

List Basic Operations

Concatenating Lists

```
In [44]: l1 = [1,2,3]
          l2 = ["a","b","c"]
          l1+l2
```

```
Out[44]: [1, 2, 3, 'a', 'b', 'c']
```

Repeating elements

```
In [45]: l1 = [1,"a",True]
          l1*3
```

```
Out[45]: [1, 'a', True, 1, 'a', True, 1, 'a', True]
```

Tuple in Python

Tuple is an ordered collection of elements enclosed within () .



Tuples are
immutable

```
tup1=(1,'a',True)
```

Extracting individual characters

Finding index position of a tuple

```
In [20]: tup1=(1,"a",True,2,"b",False)  
tup1[0]  
  
Out[20]: 1
```

```
In [22]: tup1=(1,"a",True,2,"b",False)  
tup1[1:4]  
  
Out[22]: ('a', True, 2)
```

Slicing a tuple

```
In [21]: tup1=(1,"a",True,2,"b",False)  
tup1[-1]  
  
Out[21]: False
```

Finding the last position of a tuple

Modifying a Tuple

You cannot modify a tuple because it is immutable.

```
In [49]: tup1[2]="hello"
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-49-2fc16622751e> in <module>  
----> 1 tup1[2]="hello"  
  
TypeError: 'tuple' object does not support item assignment
```

Tuple Basic Operations

Finding Length of Tuple

```
In [24]: tup1=(1,"a",True,2,"b",False)  
len(tup1)  
  
Out[24]: 6
```

Concatenating Tuples

```
In [25]: tup1 = (1,2,3)  
tup2 = (4,5,6)  
tup1+tup2  
  
Out[25]: (1, 2, 3, 4, 5, 6)
```

Tuple Basic Operations

Repeating Tuple Elements

```
In [29]: tup1 = ('sparta',300)
tup1*3

Out[29]: ('sparta', 300, 'sparta', 300, 'sparta', 300)
```

Repeating and Concatenating

```
In [31]: tup1 = ('sparta',300)
tup2 = (4,5,6)
tup1*3 + tup2

Out[31]: ('sparta', 300, 'sparta', 300, 'sparta', 300, 4, 5, 6)
```

Tuple Functions

Minimum Value

```
In [32]: tup1=(1,2,3,4,5)  
min(tup1)
```

```
Out[32]: 1
```

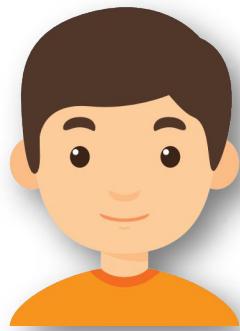
Maximum Value

```
In [33]: tup1=(1,2,3,4,5)  
max(tup1)
```

```
Out[33]: 5
```

Dictionary in Python

Dictionary is an unordered collection of key-value pairs enclosed with {}.



Dictionary is
mutable

```
Fruit={"Apple":10,"Orange":20}
```

Extracting Keys and Values

Extracting Keys

```
In [1]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}  
fruit.keys()  
  
Out[1]: dict_keys(['Apple', 'Orange', 'Banana', 'Guava'])
```

Extracting Values

```
In [70]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}  
fruit.values()  
  
Out[70]: dict_values([10, 20, 30, 40])
```

Modifying a Dictionary

Adding a new element

```
In [2]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}  
fruit["Mango"]=50  
fruit  
Out[2]: {'Apple': 10, 'Orange': 20, 'Banana': 30, 'Guava': 40, 'Mango': 50}
```

Changing an existing element

```
In [3]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40,"Mango":50}  
fruit["Apple"]=100  
fruit  
Out[3]: {'Apple': 100, 'Orange': 20, 'Banana': 30, 'Guava': 40, 'Mango': 50}
```

Dictionary Functions

Update one dictionary's elements with another.

```
In [4]: fruit1={"Apple":10,"Orange":20}  
fruit2={"Banana":30,"Guava":40}  
  
fruit1.update(fruit2)  
  
fruit1
```

```
Out[4]: {'Apple': 10, 'Orange': 20, 'Banana': 30, 'Guava': 40}
```

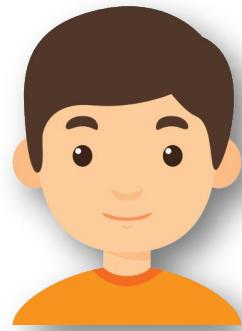
Dictionary Functions

Popping an element

```
In [6]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}
fruit.pop("Orange")
fruit
Out[6]: {'Apple': 10, 'Banana': 30, 'Guava': 40}
```

Set in Python

Set is an unordered and unindexed collection of elements enclosed with {}.



Duplicates
are not
allowed in
Set

```
s1={1,"a",True}
```

Set Operations

Update one dictionary's elements with another

```
In [7]: s1={1,"a",True,2,"b",False}  
s1.add("Hello")  
s1
```

```
Out[7]: {1, 2, False, 'Hello', 'a', 'b'}
```

Removing an element

```
In [9]: s1={1,"a",True,2,"b",False}  
s1.remove("b")  
s1
```

```
Out[9]: {1, 2, False, 'a'}
```

Updating multiple elements

```
In [8]: s1={1,"a",True,2,"b",False}  
s1.update([10,20,30])  
s1
```

```
Out[8]: {1, 10, 2, 20, 30, False, 'a', 'b'}
```

Set Functions

Union of two sets

```
In [11]: s1 = {1,2,3}  
s2 = {"a","b","c"}  
  
s1.union(s2)  
  
Out[11]: {1, 2, 3, 'a', 'b', 'c'}
```

Intersection of two sets

```
In [13]: s1 = {1,2,3,4,5,6}  
s2 = {5,6,7,8,9}  
  
s1.intersection(s2)  
  
Out[13]: {5, 6}
```

If statement in Python

If
It's raining:
Sit inside

else
Go out and Play Football



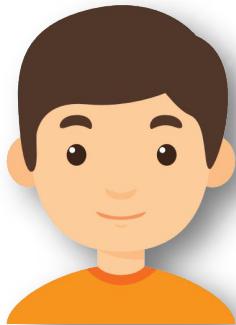
If...else Pseudo Code

```
If(condition){  
    Statements to be executed....  
}
```

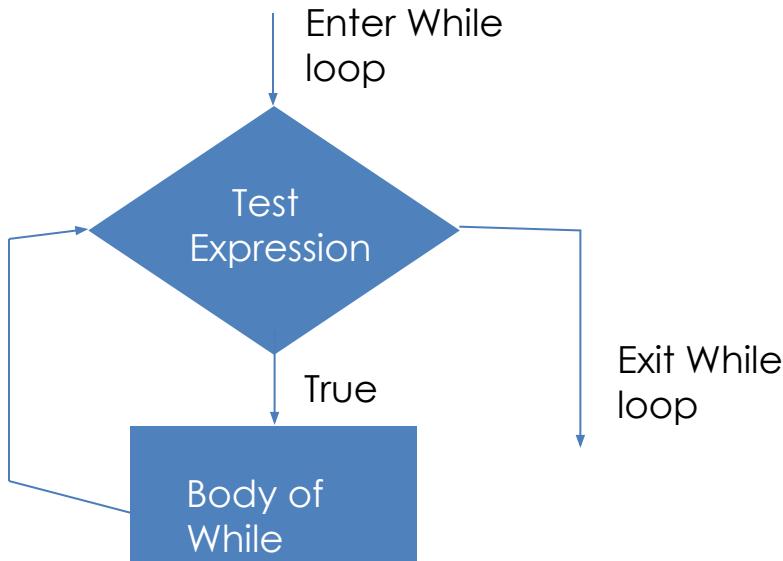
```
else{  
    Statements to be executed....  
}
```

Looping Statements in Python

Looping statements are used to repeat a task multiple times.



While Loop



Syntax

while condition:
Execute
Statements

For Loop

For Loop is used to iterate over a sequence(tuple, list, dictionary..).



This is the
syntax of for
loop

for val in sequence:
Body of for

Functions in Python

Function is a block of code which performs a specific task.



Function to deposit money

Function to withdraw money

Function to check balance

Functions Syntax

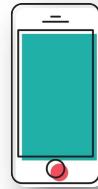
Normal Function Syntax:

```
def function_name:  
    Execute Statements
```

Lambda Function Syntax:

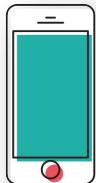
```
lambda arguments : expression
```

Python Object Oriented Programming



Classes

Class is a template/blueprint for real-world entities.



Properties

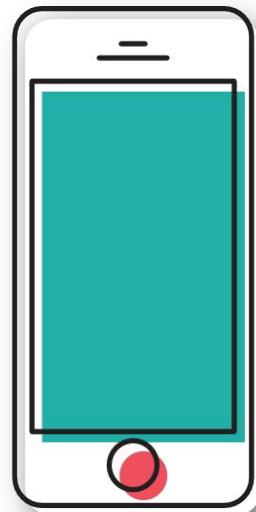
- Color
- Cost
- Battery Life

Behavior

- Make Calls
- Watch Videos
- Play Games

Class in Python

Class is a user-defined data-type.



Mobile

I am a user-defined data type

int

float

bool

str

Objects

Objects are specific instances of a class.

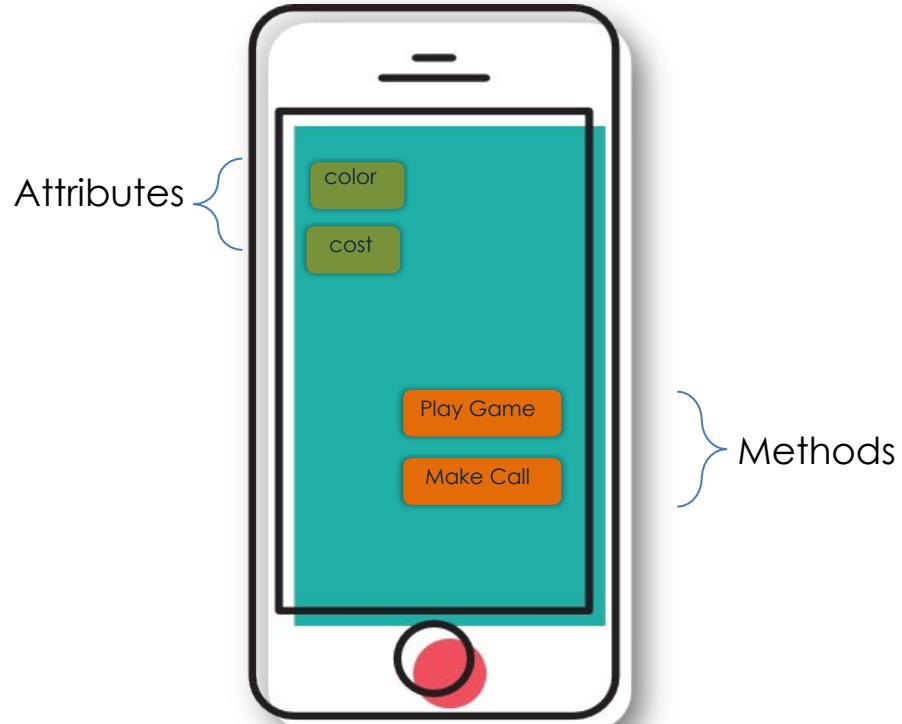


Apple

Motorola

Samsung

Attributes and Methods

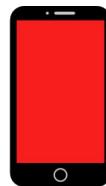


Objects in Python

Specific instances of Mobile data type.



Apple



Motorola



Samsung

a = 10

b = 20

c = 30

Creating the First Class in Python

Creating the 'Phone' class

```
In [1]: class Phone:  
  
    def make_call(self):  
        print("Making phone call")  
  
    def play_game(self):  
        print("Playing Game")
```

Instantiating the 'p1' object

```
In [38]: p1=Phone()
```

Invoking methods through object

```
In [39]: p1.make_call()  
  
Making phone call  
  
In [40]: p1.play_game()
```

Adding Parameters to the Class

Setting and Returning the attribute values.

```
' [42]: class Phone:

    def set_color(self,color):
        self.color=color

    def set_cost(self,cost):
        self.cost=cost

    def show_color(self):
        return self.color

    def show_cost(self):
        return self.cost

    def make_call(self):
        print("Making phone call")

    def play_game(self):
        print("Playing Game")'
```

Adding Parameters to a Class Method

Creating a Class with Constructor

Creating a Class with Constructor

```
In [4]: class Employee:  
    def __init__(self, name, age, salary, gender):  
        self.name = name  
        self.age = age  
        self.salary = salary  
        self.gender = gender  
  
    def employee_details(self):  
        print("Name of employee is ", self.name)  
        print("Age of employee is ", self.age)  
        print("Salary of employee is ", self.salary)  
        print("Gender of employee is ", self.gender)
```



init method acts as the constructor

Instantiating Object

Instantiating the 'e1' object



```
In [5]: e1 = Employee('Sam',32,85000,'Male')
```

Invoking the 'employee_details' method



```
In [6]: e1.employee_details()
```

```
Name of employee is Sam  
Age of employee is 32  
Salary of employee is 85000  
Gender of employee is Male
```

Inheritance in Python

With inheritance one class can derive the properties of another class.



Man inheriting features from his father

Inheritance Example

```
In [23]: class Vehicle:

    def __init__(self,mileage, cost):
        self.mileage = mileage
        self.cost = cost

    def show_details(self):
        print("I am a Vehicle")
        print("Mileage of Vehicle is ", self.mileage)
        print("Cost of Vehicle is ", self.cost)
```



Creating the base class

Inheritance Example

```
In [24]: v1 = Vehicle(500,500)  
v1.show_details()
```

```
I am a Vehicle  
Mileage of Vehicle is 500  
Cost of Vehicle is 500
```



Instantiating the object for base class

Inheritance Example

Creating the child class



```
In [25]: class Car(Vehicle):
    def show_car(self):
        print("I am a car")
```

Instantiating the object for
child class



```
In [26]: c1 = Car(200,1200)
In [27]: c1.show_details()
I am a Vehicle
Mileage of Vehicle is 200
Cost of Vehicle is 1200
```

Invoking the child class
method



```
In [28]: c1.show_car()
```

Over-riding init Method

Over-riding init method

```
In [9]: class Car(Vehicle):
    def __init__(self,mileage,cost,tyres,hp):
        super().__init__(mileage,cost)
        self.tyres = tyres
        self.hp =hp
    def show_car_details(self):
        print("I am a car")
        print("Number of tyres are ",self.tyres)
        print("Value of horse power is ",self.hp)
```

Invoking show_car_details() method from child class

```
In [12]: c1.show_car_details()
I am a car
Number of tyres are 4
Value of horse power is 300
```

Invoking show_details() method from parent class

```
In [10]: c1 = Car(20,12000,4,300)
```

```
In [11]: c1.show_details()
```

```
I am a Vehicle
Mileage of Vehicle is 20
Cost of Vehicle is 12000
```

Types of Inheritance



These are the types
of inheritance in
Python....

Single Inheritance

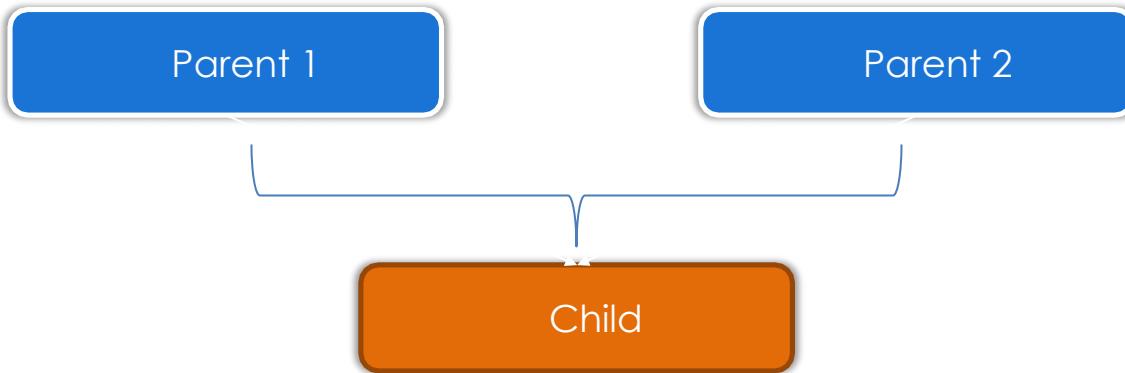
Multiple Inheritance

Multi-level Inheritance

Hybrid Inheritance

Multiple Inheritance

In multiple inheritance, the child inherits from more than 1 parent class.



Multiple Inheritance in Python

Parent Class One

```
In [35]: class Parent1():
    def assign_string_one(self,str1):
        self.str1 = str1

    def show_string_one(self):
        return self.str1
```

Child Class

```
In [40]: class Derived(Parent1, Parent2):
    def assign_string_three(self,str3):
        self.str3=str3

    def show_string_three(self):
        return self.str3
```

Parent Class Two

```
In [36]: class Parent2():
    def assign_string_two(self,str2):
        self.str2 = str2

    def show_string_two(self):
        return self.str2
```

Multiple Inheritance in Python

Invoking methods

```
In [46]: d1.show_string_one()  
Out[46]: 'one'  
  
In [47]: d1.show_string_two()  
Out[47]: 'two'  
  
In [48]: d1.show_string_three()  
Out[48]: 'three'
```

Instantiating object of child class

```
In [41]: d1 = Derived()  
  
In [42]: d1.assign_string_one("one")  
d1.assign_string_two("two")  
d1.assign_string_three("three")
```

Multi-level Inheritance in Python

In multi-level Inheritance, we have Parent, child, grand-child relationship.

Parent

Child

Grand-Child

Multi-level Inheritance in Python

Parent Class

```
In [52]: class Parent():
    def assign_name(self, name):
        self.name = name

    def show_name(self):
        return self.name
```

Grand-Child Class

```
In [54]: class GrandChild(Child):
    def assign_gender(self, gender):
        self.gender = gender

    def show_gender(self):
        return self.name
```

Child Class

```
In [53]: class Child(Parent):
    def assign_age(self, age):
        self.age = age

    def show_age(self):
        return self.age
```

Multi-level Inheritance in Python

Instantiating object of GrandChild class

```
In [55]: g1 = GrandChild()  
  
In [56]: g1.assign_name("Sam")  
          g1.assign_age(25)  
          g1.assign_gender("Male")
```

Invoking class methods

```
In [57]: g1.show_name()  
Out[57]: 'Sam'  
  
In [58]: g1.show_age()  
Out[58]: 25  
  
In [59]: g1.show_gender()  
Out[59]: 'Sam'
```

Summary

Here is a quick recap-

- We understood the basics of programming.
- We learnt what is Python programming and its advantages.
- We understood the variables, operators and data types.
- We worked on python data structures.

Thank You