PYTHON

Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:

print("Five is greater than two!")
```

Variables

Creating Variables

Example

```
x = 5
y = "John"
print(x)
print(y)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Get the Type

You can get the data type of a variable with the type() function.

Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Remember that variable names are case-sensitive

Condition for creating variable name

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alphanumeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

```
myvar = "John"
my_var = "John"
```

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John”
```

Multi Words Variable Names

- Variable names with more than one word can be difficult to read.
- There are several techniques you can use to make them more readable:

Camel Case

- Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

- Each word starts with a capital letter:

```
MyVariableName = "John"
```

Snake Case

- Each word is separated by an underscore character:

```
my_variable_name = "John”
```

Many Values to Multiple Variables

- Python allows you to assign values to multiple variables in one line:

```
#Example
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Local Variable

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

```
x = 300

def myfunc():

  print(x)

myfunc()

print(x)
```

Print Statement

- The Python print statement is often used to output variables.
- To combine both text and a variable, Python uses the ‘+’ character

```
#Example

x = "awesome"

print("Python is " + x)
```

User Defined Input

```
username = input("Enter username:")
print("Username is: " + username)
```

Data Types

Data Types

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Data Types Example

Data Type Keyword	**Example**
str	x=“Hello World”
int	x=3
float	x=3.2
complex	x=1j
list	x=[1,2,3,4,5]
tuple	x=(1,2,”Raghul”,’a’)
set	x={1,2,3,4}
dict	x={“name”:”Raghul”,”age”:24}
bool	x=True

Data Types Example with pre defining

Data Type Keyword	**Example**
str	x=str(“Hello Wrold”)
int	x=int(3)
float	x=float(3.2)
complex	x=complex(1j)
list	x=list((1,2,3,4,5))
tuple	x=tuple((1,2,3,4))
set	x=set((1,2,3,4))
dict	x=dict(“name”:”Apple”,”Quantity”:20)

Int datatype Type Conversion

Example
Convert from one type to another:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
#convert from int to float:
a = float(x)
#convert from float to int:
b = int(y)
#convert from int to complex:
c = complex(x)
print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Random Number

- Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

```
#Example

#Import the random module, and display a random number between 1 and 9:

import random

print(random.randrange(1, 10))
```

If condition

```
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")
```

Nested IF

```
x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

One line if else statement, with 3 conditions:

```
a = 330

b = 330

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

While Loop

```
i = 1

while i < 6:

  print(i)

  i += 1
```

Break statement

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Continue

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

For Loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

For range

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)
```

- range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):
  print(x)
```

- range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

```
for x in range(2, 30, 3):
  print(x)
```

Nested For

```
adj = ["red", "big", "tasty"]

fruits = ["apple", "banana", "cherry"]


for x in adj:

  for y in fruits:

    print(x, y)
```

Operators

Operators

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

Arithmetic Operator

Operation	Operator	Description
Addition	+	Addition two number
Subtraction	-	Subtraction of two number
Multiply	*	Multiplication of two number
Modulus	%	Remainder
Division	/	Quotient
Floor Division	//	Quotient rounds down to the nearest whole number
Exponentiation	**	Power

Assignment Operator

Operator	**Example**	**Operation**
=	x=3	Assign a value
+=	x+=3	x=x+3
-=	x-=3	x=x-3
*=	x*=3	x=x*3
/=	x/=3	x=x/3

Comparison Operator

Operator	**Description**
==	Equal
!=	Not Equal
<	Less Than
>	Greater Than
>=	Less Than or Equal to
<=	Greater Than or Equal to

Logical Operator

Operator	**Description**
and	Return true if both statement are True
or	Return True if one statement is True
not	Result will be opposite to input given

Identity Operator

is	Return True if Both variable are Same
is not	Return True if Both variable are not Same

Membership Operator

in	Return True if sequence with specified value is present
not in	Return True if sequence with specified value is not present

Bitwise Operator

Operator	Operation	Description
&	AND	Single bit AND operation
|	OR	Single bit OR operation
^	XOR	Single bit XOR operation
~	NOT	Changing bit 1 to 0 and 0 to 1
<<	Left Shift	Left shifting
>>	Right Shift	Right shifting

Strings

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

```
#Example

print("Hello")

print('Hello')
```

Multiline Strings

- You can assign a multiline string to a variable by using three quotes:

```
#Example

#You can use three double quotes:


a = """My name is Raghul, I'm from Chennai"""

print(a)
```

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.

```
#Example
#Get the character at position 1 (remember that the first character has the position 0):

a = "Hello, World!"
print(a[1])
```

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

```
#Example
#Loop through the letters in the word "banana":

for x in "banana":
  print(x)
```

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
find()	Searches the string for a specified value and returns the position of where it was found
count()	Returns the number of times a specified value occurs in a string
split()	Splits the string at the specified separator, and returns a list
lower()	Converts a string into lower case
upper()	Converts a string into upper case
swapcase()	Swaps cases, lower case becomes upper case and vice versa

isalnum()
isalpha()
isdecimal()
isnumeric()
	Returns True

title()	Converts the first character of each word to upper case

String manipulation function

```
print(len(a)) # string length
print("free" in txt) #check the string contain the word
```

(Or)

```
txt = "The best things in life are free!"
if "free" in txt:
 print("Yes, 'free' is present.")
```

Slicing

```
print(b[2:5]) #2 to 4
print(b[:5]) #0 to 4
print(b[2:]) #2 to end
print(b[-5:-2]) #from 5th position from last till 3rd position
print(a.upper()) #upper case
print(a.lower()) #lower case
print(a.strip()) # remove unwanted spaces
print(a.replace("H", "J")) # replace H with J
print(a.split(",")) #split the two string by comma
```

Concadinate of string and integer

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))

#by using index
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Python Collection

- ❖ There are four collection data types in the Python programming language:

1. List is a collection which is ordered and changeable. Allows duplicate members.
2. Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
3. Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
4. Dictionary is a collection which is ordered** and changeable. No duplicate members.

List

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets
- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- A list can contain different data types

```
#Example

thislist = ["apple", "banana", "cherry"]

print(thislist)
```

List Methods

```
thislist = list(("apple", "banana", "cherry")) # list can also be created in this way.
print(len(thislist))
print(type(mylist))
print(thislist[1]) # index 1 print
print(thislist[-1]) # last element
print(thislist[2:5])  # index 2 ,3, 4 will print
print(thislist[:4]) # Till 3rd index
print(thislist[2:]) # index 2 to end
thislist[1] = "blackcurrant” # Changing Item at index 1
thislist[1:3] = ["blackcurrant", "watermelon"] # change index 1 and 2 not 3
thislist.insert(2, "watermelon") # insert at index 2
thislist.append("orange") # add this at end
# Extend a list
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
thislist.remove("banana") #delete an element
thislist.pop(1) # delete using index (if index not specified last element deleted)
del thislist #delete entire list (if index mentioned [1] that will alone delete)
thislist.clear() # only clear the content (empty list)
thislist.sort() # sort alphabetically / sequence
thislist.sort(reverse = True) # descending order
thislist.sort(key = str.lower) #case insensitive sort
thislist.reverse() #reverse order
mylist = thislist.copy() # copy list into new list
list3 = list1 + list2 # combining two list
# another way combining
for x in list2:
  list1.append(x)
#add list 2 at end of list 2
list1.extend(list2)
```

Method	**Description**
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Printing list

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)


#index number
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

```
#while
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
#For
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
#without list comprehensive

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = []

for x in fruits:

  if "a" in x:

    newlist.append(x)

print(newlist)
```

With list comprehensive

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

Tuple

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
- A tuple can contain different data types

```
# Example

thistuple = ("apple", "banana", "cherry")
```

Tuple Method

```
thistuple = tuple(("apple", "banana", "cherry")) #define in other way

print(len(thistuple)) # length

print(thistuple[1]) # index 1

print(thistuple[-1]) # last element

print(thistuple[2:5]) # index 2 to 4

print(thistuple[:4])

print(thistuple[2:])

print(thistuple[-4:-1])

del thistuple # delete completely

thistuple.count(5) # occurrence

thistuple.index(8) # index of the elements
```

Change Tuple Values

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi" # replace index 1
x = tuple(y)
print(x)
y.append("orange") # append
y.remove("apple") # remove tuple by changing to list
# adding 2 tuple
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

Unpacking Tuple

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)

print(yellow)

print(red)
```

Using Asterisk*

- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list

```
#Example

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)

print(yellow)

print(red)
```

Printing Tuple

```
thistuple = ("apple", "banana", "cherry")

for x in thistuple:

  print(x)

# index number.

thistuple = ("apple", "banana", "cherry")

for i in range(len(thistuple)):

  print(thistuple[i])
```

```
thistuple = ("apple", "banana", "cherry")

i = 0

while i < len(thistuple):

  print(thistuple[i])

  i = i + 1
```

Set

- Sets are used to store multiple items in a single variable.
- A set is a collection which is unordered, unchangeable*, and unindexed.

* Note: Set items are unchangeable, but you can remove items and add new items

- Sets are written with curly brackets.
- Set items are unordered, unchangeable, and do not allow duplicate values.
- A set can contain different data types

```
# Example

thisset = {"apple", "banana", "cherry"}

print(thisset)
```

Set Function

```
print(len(thisset)) # Length

thisset.add("orange") # add element

thisset.update(tropical) # add element from set 2 to set 1

thisset.remove("banana") # remove

thisset.discard("banana")  # remove

thisset.pop() # delete last index

thisset.clear() # empty set

del thisset #delete completely
```

```
set3 = set1.union(set2) # join two set

x.intersection_update(y) #keep only duplicate and store in x

z = x.intersection(y) # keep duplicate in new set

x.symmetric_difference_update(y) # keep element that are not present in both set

z = x.symmetric_difference(y) # keep element not present in both set in new set
```

Method	Description
copy()	Returns a copy of the set
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not

Printing Set

```
# loop
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
#for
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
# in function
thisset = {"apple", "banana", "cherry"}
print("banana" in thisset)
```

Dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values:
- Dictionary items are ordered, changeable, and does not allow duplicates.
- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
#Example

thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

print(thisdict["brand"])
```

Dictionary contain List as value

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

Add new item

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

Adding an item

```
thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

thisdict.update({"color": "red"})
```

Dictionary Methods

```
print(len(thisdict)) # length

x = thisdict.get("model") # get the model

x = thisdict.keys() # return list of keys

x = thisdict.values() # return values

x = thisdict.items() # return item

thisdict.pop("model") # remove model

thisdict.popitem() #last item
```

Changes in values

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.values()
print(x) #before the change
car["year"] = 2020
print(x) #after the change
```

Changing value

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
```

Copy

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Copy using dict keyword

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Check key is present

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Printing

```
# print all key
for x in thisdict:
  print(x)
# another method keys
for x in thisdict.keys():
  print(x)
# all items
for x, y in thisdict.items():
  print(x, y)
```

Printing Values

```
# all values.
for x in thisdict:
  print(thisdict[x])
# another method for value
for x in thisdict.values():
  print(x)
```

Nested dictionary

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Function

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

```
# creating function and calling
def my_function():
  print("Hello from a function")
my_function()
```

Passing Parameter

```
def rectangle(l,b):
    return l*b
print(rectangle(2,3))
```

call by reference

```
def add_more(list):
    list.append(50)
    print("Inside Function", list)

mylist = [10,20,30,40]

add_more(mylist)
print("Outside Function:", mylist)
```

call by value

```
string = "Hi"
def test(string):

    string = "Hello World!"
    print("Inside Function:", string)

test(string)
print("Outside Function:", string)
```

Array

An **array** is a special variable, which can **hold more than one value at a time.**

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"

car2 = "Volvo"

car3 = "BMW"
```

Arrays are used to **store multiple values in one single variable:**

Example

Create an array containing **car names:**

```
cars = ["Ford", "Volvo", "BMW"]                    output [ “Ford”,”Volvo” , “BMW”]

print(cars)
```

Length of an Array

- Use the len() method to return the length of an array (the number of elements in an array.

Example

Return the number of elements in the cars array:

```
cars = ["Ford", "Volvo", "BMW"]

x = len(cars)

print(x)                                        output : 3
```

Python does not have built-in support for Arrays, but Python Lists can be used instead.

Array Matrix

```
R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))
# Initialize matrix
matrix = []
print("Enter the entries rowwise:")
# For user input
for i in range(R):              # A for loop for row entries
        a =[]
        for j in range(C):      # A for loop for column entries
                a.append(int(input()))
        matrix.append(a)
# For printing the matrix
for i in range(R):
        for j in range(C):
                print(matrix[i][j], end = " ")
        print()
```

Lambda Function

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda *arguments* : *expression*

```
x = lambda a, b : a * b
print(x(5, 6))
```

Classes/Objects

- Python is an **object oriented programming language**
- Almost Everything in python is an object, with its properties and methods.

Class is like an object constructor,or a “blueprint” for creating objects.

Ex : Create a class

```
class sample:

   x,y=10,20

s=sample()  #creating Object s

print("value of x:",s.x)

print("value of y:",s.y)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

Note: The __init__() function is called automatically every time the class is being used to create a new object

Self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

```
class Person:

 def __init__(mysillyobject, name, age):

  mysillyobject.name = name

    mysillyobject.age = age

 def myfunc(abc):

  print("Hello my name is " + abc.name)

p1 = Person("John", 36)

p1.myfunc()
```

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

- **Parent class** is the class being inherited from,also called **base class**.

- **Child class** is the class that inherits from another class,also called **derived class.**

Example

To Create Parent class:

Class named **Person** ,with **Firstname** and **Lastname** properties,and a **printname** method.

```
class Person:
 def _init_(self, fname, lname):
  self.firstname = fname
  self.lastname = lname
 def printname(self):
  print(self.firstname, self.lastname)
```

Use the **Person** class to create an object, and then execute the **printname** method:

```
x=Person (“John”,”Doe”)                                   output : John Doe
x.printname()
```

Example

To Create child class :

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Use the **Student** class to create an object, and then execute the **printname** method:

```
class Person:
  def _init_(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  Pass
x=Student (“Mike”,”olsen”)                                        output :Mike Olsen
x.printname()
```

```
class Person:  #Parent
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person): #Child
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
x = Student("Mike", "Olsen")
x.printname()
```

Use the super() Function

- Python also has a `super()` function that will make the child class inherit all the methods and properties from its parent
- By using the `super()` function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

```
#init function
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

Adding Property

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)
```

Adding Method

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

Python - Algorithm

Search − Algorithm to search an item in a data structure.

Sort − Algorithm to sort items in a certain order.

Insert − Algorithm to insert item in a data structure.

Update − Algorithm to update an existing item in a data structure.

Binary Search Algorithm

- A binary search is an algorithm **to find a particular element in the list.**

- Suppose we have a list of thousand elements, and we need to get an index position of a particular element. We can find the **element's index position** very fast using the binary search algorithm.

In the **binary search algorithm,** we can find the element position using the following methods.

- **Recursive Method**
- **Iterative Method**

Recursive method:

The divide and conquer approach technique is followed by the recursive method. In this method, a f**unction is called itself again and again until it found an element in the list.**

Iterative Method:

A set of statements is repeated multiple times to find an element's index position in the iterative method. The **while** loop is used for accomplish this task.

Sorting Algorithm

Sorting algorithms denote the **ways to arrange data in a particular format.**

- Sorting ensures that data searching is optimized to a high level and that the data is presented in a readable format.

Five different types of Sorting algorithms:

- Bubble Sort
- Merge Sort
- Insertion Sort
- Shell Sort
- Selection Sort

Insertion sort

To sort the array using insertion sort below is the algorithm of insertion sort.

- Split a list in two parts - sorted and unsorted.
- Iterate from arr[1] to arr[n] over the given array.
- Compare the current element to the next element.
- If the current element is smaller than the next element, compare to the element before, Move to the greater elements one position up to make space for the swapped element.

Example Programs

```
#calculator
# This function adds two numbers
def add(x, y):
    return x + y
# This function subtracts two numbers
def subtract(x, y):
    return x - y
# This function multiplies two numbers
def multiply(x, y):
    return x * y
# This function divides two numbers
def divide(x, y):
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
```

```
while True:
    # take input from the user
    choice = input("Enter choice(1/2/3/4): ")
    # check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))
        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))
        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))
        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))
        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))
        # check if user wants another calculation
        # break the while loop if answer is no
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
          break
    else:
        print("Invalid Input")
```