

INTRODUCTION TO PYTHON

What is Programming?

Just like we use Hindi or English to communicate with each other, we use a Programming language to communicate with the computer.

Programming is a way to instruct the computer to perform various tasks.

What is Python?

Python is a simple and easy-to-understand language that feels like reading simple English. This Pseudo code nature of Python makes it easy to learn and understandable for beginners.

Features of Python:

- Easy to understand = Less development time
- Free and open source
- High-level language
- Portable – works on Windows/Linux/Mac
- + Fun to work with :)

Modules, Comments & Pip

Let's write our very first python program.

Create a file called hello.py and paste the below code into it

```
print("Hello World")          # print is a function (more later)  
Copy
```

Execute this file (.py file) by typing `python hello.py`, and you will see Hello World printed on the screen.

Modules

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs.

Pip

Pip is a package manager for python. You can use pip to install a module on your system.

E.g., pip install flask (It will install flask module in your system)

Types of modules

There are two types of modules in Python:

1. Built-in modules – Pre-installed in Python
2. External modules – Need to install using pip

Some examples of built-in modules are os, abc, etc.

Some examples of external modules are TensorFlow, flask, etc.

Using Python as a Calculator

We can use python as a calculator by typing “python” + **TO DO** on the terminal. [It opens REPL or read evaluation print loop]

Comments

Comments are used to write something which the programmer does not want to execute.

Comments can be used to mark the author's name, date, etc.

Types of Comments:

There are two types of comments in python,

1. Single line comments – Written using # (pound/hash symbol)
2. Multi-line comments – Written using ''' Comment ''' or """ Comment """.

Variables and Data Types

A variable is a name given to a memory location in a program. For example

```
a=30
```

```
b="Harry"
```

```
c=71.22
```

Copy

Variable - Container to store a value

Keywords - Reserved words in Python

Identifiers - class/function/variable name

Data Types:

Primarily there are the following data types in Python:

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a = 71                                #Identifies a as  
class<int>  
  
b = 88.44                             #Identifies b as class<float>  
  
name = "Harry"                       #Identifies name as class<Str>  
Copy
```

Rules for defining a variable name: (Also applicable to other identifiers)

- A variable name can contain alphabets, digits, and underscore.
- A variable name can only start with an alphabet and underscore.
- A variable can't start with a digit.
- No white space is allowed to be used inside a variable name.

Examples of few valid variable names,

Harry, harry, one8, _akki, aakash, harry_bro, etc.

Operators in Python

The following are some common operators in Python:

1. Arithmetic Operators (+, -, *, /, etc.)
2. Assignment Operators (=, +=, -=, etc.)

3. Comparison Operators (==, >=, <=, >, <, !=, etc.)
4. Logical Operators (and, or, not)

type() function and Typecasting

type function is used to find the data type of a given variable in Python.

```
a = 31

type(a)                                #class<int>

b = "31"

type(b)                                #class<str>
Copy
```

A number can be converted into a string and vice versa (if possible)

There are many functions to convert one data type into another.

```
Str(31)                                # "31" Integer to string conversion

int("32")                              # 32 String to int conversion

float(32)                              #32.0 Integer to float conversion
Copy
```

... and so on

Here "31" is a string literal, and 31 is a numeric literal.

input() function

This function allows the user to take input from the keyboard as a string.

```
a = input("Enter name")                #if a is "harry", the user
entered harry
Copy
```

Note: The output of the input function is always a string even if the number is entered by the user.

Suppose if a user enters 34, then this 34 will automatically convert to "34" string literal.

Strings

The string is a data type in Python.

A string is a sequence of characters enclosed in quotes.

We can primarily write a string in three ways:

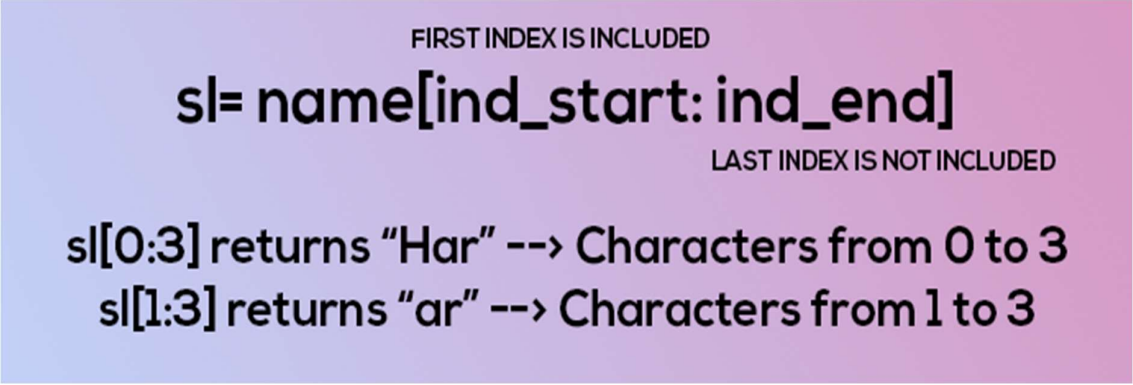
1. Single quoted strings : `a = 'harry'`
2. Double quoted strings : `b = "harry"`
3. Triple quoted strings : `c = '''harry'''`

String Slicing:

A string in Python can be sliced for getting a part of the string.

Consider the following string:

The index in a string starts from 0 to (length-1) in Python. To slice a string, we use the following syntax:



The diagram illustrates the string slicing syntax `sl = name[ind_start: ind_end]` on a blue-to-purple gradient background. Above the opening bracket, the text "FIRST INDEX IS INCLUDED" is written. Below the closing bracket, the text "LAST INDEX IS NOT INCLUDED" is written. Below the syntax, two examples are provided: `sl[0:3]` returns "Har" --> Characters from 0 to 3, and `sl[1:3]` returns "ar" --> Characters from 1 to 3.

Negative Indices: Negative indices can also be used as shown in the figure above. -1 corresponds to the (length-1) index, -2 to (length-2).

Slicing with skip value

We can provide a skip value as a part of our slice like this:

```
word = "amazing"
```

```
word[1:6:2]          # It will return 'mzn'
```

Copy

Other advanced slicing techniques

```
word = 'amazing'
```

```
word[:7] or word[0:7]      #It will return 'amazing'
```

```
word[0:] or word[0:7]      #It will return 'amazing'
```

Copy

String Functions

Some of the most used functions to perform operations on or manipulate strings are:

1. **len() function** : It returns the length of the string.

```
len('harry')                #Returns 5
```

Copy

2. **endswith("rry")** : This function tells whether the variable string ends with the string "rry" or not. If string is "harry", it returns for "rry" since harry ends with rry.
3. **count("c")** : It counts the total number of occurrences of any character.
4. **capitalize()** : This function capitalizes the first character of a given string.
5. **find(word)** : This function finds a word and returns the index of first occurrence of that word in the string.
6. **replace(oldword, newword)** : This function replaces the old word with the new word in the entire string.

Escape Sequence Characters:

Sequence of characters after backslash '\ ' [Escape Sequence Characters]

Escape Sequence Characters comprises of more than one character but represents one character when used within the string.

Examples: \n (new line), \t (tab), \' (single quote), \\ (backslash), etc.

Tuples

Python Lists are containers to store a set of values of any data type.

```
friends = ['Apple', 'Akash', 'Rohan', 7, False]
```

Copy

The list can contain different types of elements such as int, float, string, Boolean, etc. Above list is a collection of different types of elements.

List Indexing

A list can be index just like a string.

```
L1 = [7, 9, 'harry']
```

```
L1[0] - 7
```

```
L1[1] - 9
```

```
L1[70] - Error
```

```
L1[0:2] - [7,9]          (This is known as List Slicing)  
Copy
```

List Methods

Consider the following list:

```
L1 = [1, 8, 7, 2, 21, 15]  
Copy
```

1. `sort()` - updates the list to [1,2,7,8,15,21]
2. `reverse()` - updates the list to [15,21,2,7,8,1]
3. `append(8)` - adds 8 at the end of the list
4. `insert(3,8)` - This will add 8 at 3 index
5. `pop(2)` - It will delete the element at index 2 and return its value
6. `remove(21)` - It will remove 21 from the last

Tuples in Python:

A tuple is an immutable (can't change or modified) data type in Python.

```
a = ()          #It is an example of empty tuple
```

```
a = (1,)        #Tuple with only one element needs a comma
```

```
a = (1, 7, 2)   #Tuple with more than one element  
Copy
```

Once defined, tuple elements can't be manipulated or altered.

Tuple methods:

Consider the following tuple,

```
a = (1, 7, 2)
```

Copy

1. **count(1)** - It will return the number of times 1 occurs in a.
2. **index(1)** - It will return the index of the first occurrence of 1 in a.

```
a = (7, 0, 8, 0, 0, 9)
```

Copy

Dictionary and Sets

Dictionary is a collection of key-value pairs.

Syntax:

```
''' a = {"key": "value",  
"harry": "code",  
"marks" : "100",  
"list": [1,2,9]}  
a["key"]      # Prints value  
a["list"]     # Prints [1,2,9] '''
```

Copy

Properties of Python Dictionaries

1. It is unordered
2. It is mutable
3. It is indexed
4. It cannot contain duplicate keys

Dictionary Methods

Consider the following dictionary,

```
a = {"name": "Harry",  
     "from": "India",  
     "marks": [92,98,96]}
```

Copy

1. **items()** : returns a list of (key,value) tuple.
2. **keys()** : returns a list containing dictionary's keys.
3. **update({"friend": "Sam"})** : updates the dictionary with supplied key-value pairs.

4. **get("name")** : returns the value of the specified keys (and value is returned e.g., "Harry" is returned here)

More methods are available on docs.python.org

Sets in Python

Set is a collection of non-repetitive elements.

```
S= Set()           # No repetition allowed!
```

```
S.add(1)
```

```
S.add(2)
```

```
# or Set = {1,2}
```

Copy

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

Properties of Sets

1. Sets are unordered # Elements order doesn't matter
2. Sets are unindexed # Cannot access elements by index
3. There is no way to change items in sets
4. Sets cannot contain duplicate values

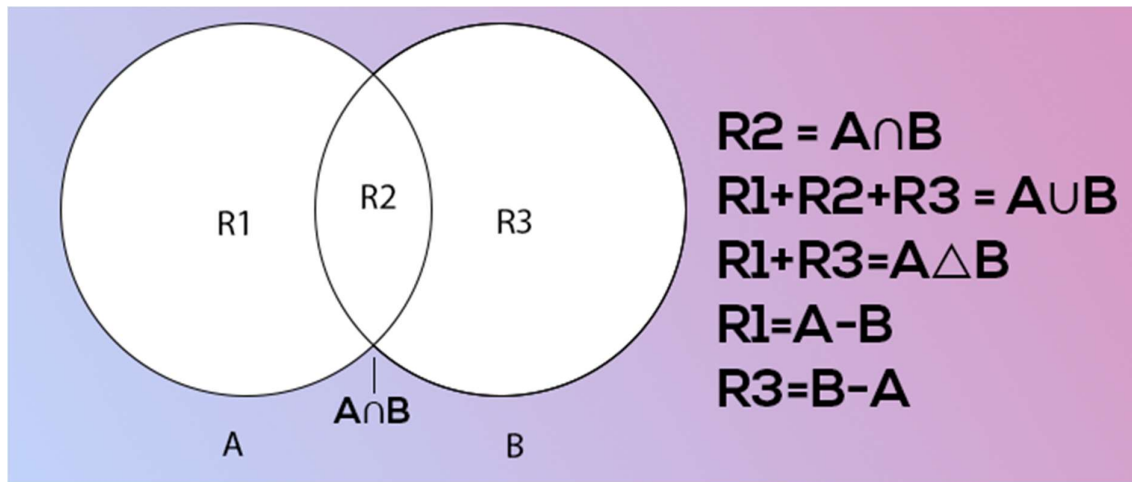
Operations on Sets

Consider the following set:

```
S = {1,8,2,3}
```

Copy

1. **Len(s)** : Returns 4, the length of the set
2. **remove(8)** : Updates the set S and removes 8 from S
3. **pop()** : Removes an arbitrary element from the set and returns the element removed.
4. **clear()** : Empties the set S
5. **union({8, 11})** : Returns a new set with all items from both sets. #{1,8,2,3,11}
6. **intersection({8, 11})** : Returns a set which contains only items in both sets. #{8}



Chapter 5 – Practice Set

1. Write a program to create a dictionary of Hindi words with values as their English translation. Provide the user with an option to look it up!
2. Write a program to input eight numbers from the user and display all the unique numbers (once).
3. Can we have a set with 18(int) and "18"(str) as a value in it?
4. What will be the length of the following set S:

```
S = Set()
```

```
S.add(20)
```

```
S.add(20.0)
```

```
S.add("20")
```

Copy

What will be the length of S after the above operations?

5. $S = \{\}$, what is the type of S?
6. Create an empty dictionary. Allow 4 friends to enter their favorite language as values and use keys as their names. Assume that the names are unique.
7. If the names of 2 friends are the same; what will happen to the program in Program 6?
8. If the languages of two friends are the same; what will happen to the program in Program 6?
9. Can you change the values inside a list which is contained in set S

```
S = {8, 7, 12, "Harry", [1, 2]}
```

Copy

Conditional Expressions

Sometimes we want to play pubg on our phone if the day is Sunday.

Sometimes we order Ice-cream online if the day is sunny.

Sometimes we go hiking if our parents allow.

All these are decisions that depend on the condition being met.

In python programming too, we must be able to execute instructions on a condition(s) being met. This is what conditions are for!

If else and elif in Python

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
...
if (condition1):           // if condition 1 is true
    print("yes")
elif (condition2):         // if condition 2 is true
    print("No")
else:                      // otherwise
    print("May be")
...
```

Copy

Code example:

```
a = 22
if (a>9):
    print("Greater")
else:
    print("lesser")
```

Copy

Relational Operators

Relational operators are used to evaluate conditions inside if statements. Some examples of relational operators are:

`=` -> equals

`>=` -> greater than/equal to

<=, etc.

Copy

Logical Operators

In python, logical operators operate on conditional statements.

Example:

and -> true if both operands are true else false

or -> true if at least one operand is true else false

not -> inverts true to false and false to true

Copy

elif clause

elif in python means [else if]. If statement can be chained together with a lot of these elif statements followed by an else statement.

```
'''
if (condition1):
    #code
elif (condition 2):
    #code
elif (condition 2):
    #code
...
else:
    #code    '''
```

Copy

- The above ladder will stop once a condition in an if or elif is met.

Important Notes:

- There can be any number of elif statements.
- Last else is executed only if all the conditions inside elifs fail.

Loops in Python

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000

Loops make it easy for a programmer to tell the computer, which set of instructions to repeat, and how!

Types of loops in Python

Primarily there are two types of loops in Python

1. While loop
2. For loop

We will look into this one by one!

While loop

The syntax of a while loop looks like this:

```
''' while condition:
    #Body of the loop '''
```

Copy

- The block keeps executing until the condition is true/false

In while loops, the condition is checked first. If it evaluates to true, the body of the loop is executed, otherwise not!

If the loop is entered, the process of condition check and execution is continued until the condition becomes false.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

An Example:

```
i = 0
while i<5:
    print("Harry")
    i = i+1
```

Copy

(Above program will print Harry 5 times)

Note: if the condition never becomes false, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

For loop

A for loop is used to iterate through a sequence like a list, tuple, or string (iterables)

The syntax of a for loop looks like this:

```
l = [1, 7, 8]
for item in l:
    print(item)
```

Copy

(Above program will print 1, 7, and 8)

Range function in Python

The range function in python is used to generate a sequence of numbers.

We can also specify the start, stop, and step-size as follows:

```
range(start, stop, step_size)
```

- step size is usually not used with range()

An example demonstrating range() function

```
for i in range(0, 7):           #range(7) can also be used
    print(i)                   #prints 0 to 6
```

Copy

For loop with else

An optional else can be used with a for loop if the code is to be executed when the loop exhausts.

Example:

```
l = [1, 7, 8]
for item in l:
    print(item)
else:
    print("Done") #This is printed when the loop exhausts!
```

Copy

Output:

1

7

8

Done

Copy

The break statement

'break' is used to come out of the loop when encountered. It instructs the program to – Exit the loop now.

Example:

```
for i in range(0, 80):  
    print(i)      #This will print 0, 1, 2 and 3  
    if i == 3:  
        break
```

Copy

The continue statement

'continue' is used to stop the current iteration of the loop and continue with the next one. It instructs the program to “skip this iteration.”

Example:

```
for i in range(4):  
    print("printing")  
    if i == 2:    #if i is 2, the iteration is skipped  
        continue  
    print(i)
```

Copy

pass statement

pass is a null statement in python. It instructs to “Do nothing.”

Example:

```
l = [1, 7, 8]  
for item in l:  
    pass          #without pass, the program will throw an error
```

Copy

Functions and Recursions

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track of which piece of code is doing what!

A function can be reused by the programmer in a given program any number of times.

Example and Syntax of a function

The syntax of a function looks as follows:

```
def func1():  
    print("Hello")
```

Copy

This function can be called any number of times, anywhere in the program.

Function call

Whenever we want to call a function, we put the name of the function followed by parenthesis as follows:

```
func1()          #This is called function call
```

Copy

Function definition

The part containing the exact set of instructions that are executed during the function call.

Types of functions in Python

There are two types of functions in Python:

1. Built-in functions #Already present in Python
2. User-defined functions #Defined by the user

Examples of built-in function includes `len()`, `print()`, `range()`, etc.

The func1() function we defined is an example of a user-defined function.

Functions with arguments

A function can accept some values it can work with. We can put these values in the parenthesis. A function can also return values as shown below:

```
def greet(name):  
    gr = "Hello" + name  
    return gr  
Copy  
a = greet("Harry") # "Harry" is passed to greet in name  
  
# a will now contain "Hello Harry"  
Copy
```

Default Parameter Value

We can have a value as the default argument in a function.

If we specify name = "stranger" in the line containing def, this value is used when no argument is passed.

For Example:

```
def greet(name='stranger'):  
    #function body  
Copy  
greet()                                #Name will be 'stranger' in function  
body(default)  
  
greet("Harry")                        #Name will be "Harry" in function body(passed)  
Copy
```

Recursion

Recursion is a function which calls itself.

It is used to directly use a mathematical formula as a function. For example:

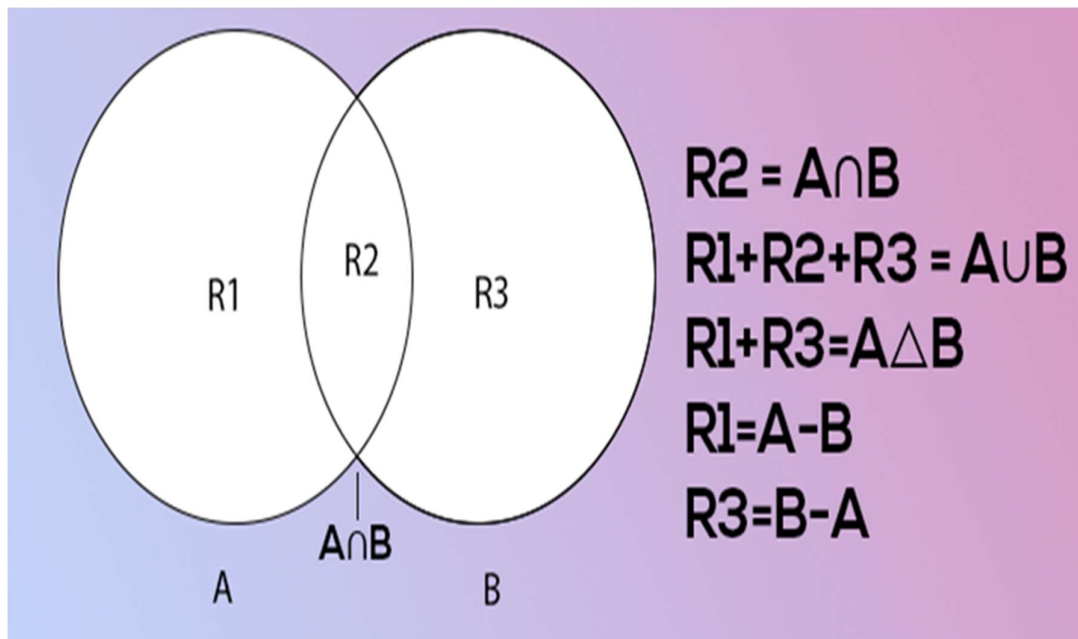
```
factorial(n) = n * factorial(n-1)  
Copy
```

This function can be defined as follows:

```
def factorial(n):  
    if i == 0 or i == 1 : #Base condition which doesn't call the  
        function any further  
        return i  
    else:  
        return n*factorial(n-1) #Function calling itself
```

Copy

This works as follows:



The programmer needs to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep calling itself.

Recursion is sometimes the most direct way to code an algorithm.

File I/O

The random access memory is volatile, and all its contents are lost once a program terminates.

In order to persist the data forever, we use files.

A file is data stored in a storage device. A python program can talk to the file by reading content from it and writing content to it.



Types of Files

There are 2 types of files:

1. Text files (.txt, .c, etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating, and deleting files.

Opening a file

Python has an `open()` function for opening files. It takes 2 parameters: filename and mode.

```
open("this.txt", "r")  
Copy
```

Here, "this" is the file name and "r" is the mode of opening (read mode)

Reading a file in Python

```
f = open("this.txt", "r")    #Opens the file in r mode  
  
text = f.read()             #Read its content  
  
print(text)                 #Print its contents  
  
f.close()                   #Close the file  
Copy
```

We can also specify the number of characters in `read()` function:

Python With CodeWithPro

```
f.read(2)          #Reads first 2 characters  
Copy
```

Other methods to read the file

We can also use `f.readline()` function to read one full line at a time.

```
f.readline()       #Reads one line from the file  
Copy
```

Modes of opening a file

`r` – open for reading

`w` – open for writing

`a` – open for appending

`+` -> open for updating

`'rb'` will open for read in binary mode

`'rt'` will open for read in text mode

Writing Files in Python

In order to write to a file, we first open it in write or append mode, after which, we use the python's `f.write()` method to write to the file!

```
f = open("this.txt", "w")  
  
f.write("This is nice")      #Can be called multiple times  
  
f.close()  
Copy
```

With statement

The best way to open and close the file automatically is the “with” statement.

```
with open("this.txt") as f:  
  
    f.read()  
Copy
```

#There is no need to write f

Object-Oriented Programming

Solving a problem by creating objects is one of the most popular approaches in programming. This is called object-oriented programming.

This concept focuses on using reusable code. (Implements DRY principle)

Class

A class is a blueprint for creating objects.



The syntax of a class looks like this:

```
Class Employee:                                [classname is written in PascalCase]
    #methods & variables
Copy
```

Object

An object is an instantiation of a class. When class is defined, a template(info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. #Abstraction & Encapsulation!

Modelling a problem in OOPs

We identify the following in our problem

Noun -> Class -> Employee

Adjective -> Attributes -> name,age,salary

Verbs -> Methods -> getSalary(), increment()

Class Attributes

An attribute that belongs to the class rather than a particular object.

Example:

```
Class Employee:
    company = "Google"    #Specific to each class
harry = Employee()    #Object instantiation
harry.company
Employee.company = "YouTube"    #changing class attribute
Copy
```

Instance Attributes

An attribute that belongs to the Instance (object)

Assuming the class from the previous example:

```
harry.name = "Harry"
harry.salary = "30K" #Adding instance attributes
Copy
```

Note: Instance attributes take preference over class attributes during assignment and retrieval.

harry.attribute1 :

1. Is attribute1 present in the object?
2. Is attribute1 present in class?

'self' parameter

self refers to the instance of the class.

It is automatically passed with a function call from an object.

Inheritance & more on OOPs

Inheritance is a way of creating a new class from an existing class.

Syntax:

```
class Emoloyee:      #Base Class
    #Code
class Programmer(Employee): #Derived or child class
    #Code
Copy
```

We can use the methods and attributes of Employee in Programmer object.

Also, we can overwrite or add new attributes and methods in the Programmer class.

Type of Inheritance

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance

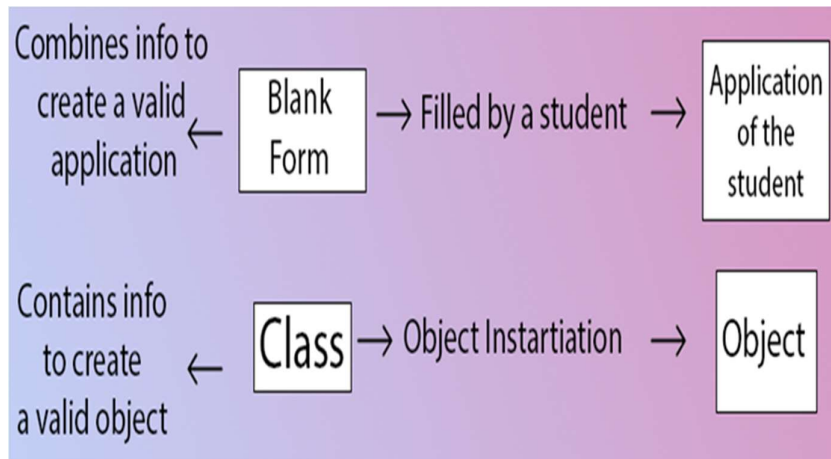
Single Inheritance

Single inheritance occurs when a child class inherits only a single parent class.

Base -> Derived

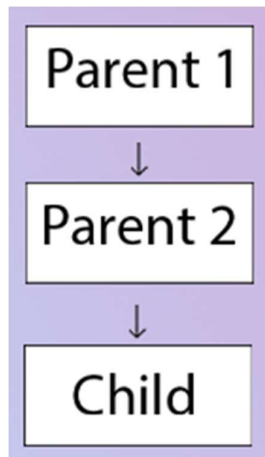
Multiple Inheritance

Multiple inheritances occurs when the child class inherits from more than one parent class.



Multilevel Inheritance

When a child class becomes a parent for another child class.



Super() method

Super method is used to access the methods of a superclass in the derived class.

```
super().__init__() #Calls constructor of the base class
```

Copy

Class methods

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method.

Syntax to create a class method:

```
@classmethod
def (cls, p1, p2):
    #code
Copy
```

@property decorators

Consider the following class

```
class Employee:
    @property
    def name(self):
        return self.ename
Copy
```

if `e = Employee()` is an object of class employee, we can print `(e.name)` to print the `ename`/call `name()` function.

@.getters and @.setters

The method name with `@property` decorator is called getter method.

We can define a function + `@name.setter` decorator like below:

```
@name.setter
def name(self, value):
    self.ename = value
Copy
```

Operator overloading in Python

Operators in python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.

Operators in python can be overloaded using the following methods:

`p1 + p2 -> p1.__add__(p2)`

`p1 - p2 -> p1.__sub__(p2)`

`p1 * p2 -> p1.__mul__(p2)`

Python With CodeWithPro

`p1 / p2 -> p1.__truediv__(p2)`

`p1 // p2 -> p1.__floordiv__(p2)`

Copy

Other dunder/magic methods in Python

`__str__()` -> used to set what gets displayed upon calling `str(obj)`

`__len__()` -> used to set what gets displayed upon calling `.__len__()`
or `len(obj)`

Copy