Python Basic For Beginners

Author: Prabin Chaudhary
(Software Engineer)

What is programming?

Just like we use Nepali or English to communicate with each other, we use a programming language like Python 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 learn, understand language which feels like reading simple english. This PseudoCode nature of python makes it easy to learn and understandable by beginners.

Features of Python?

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

Installation

Python can be easily installed from ***python.org***, when you click on the download button, Python can be installed right after you complete the setup by executing the file for your platform.

Just install it like a game ! 😊

Chapter-1: Modules, Comments & Pip

Let's write our very first Python program. Create a file called ***hello.py*** and paste the below code in it.

print("Hello World") ➔ print is a function (more later)

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 codes written by somebody else (usually) when can be imported and used in our programs.

Pip:

Pip is the package manager for Python. You can use pip to install modules on your system.

↳ ***pip install django*** (install django module)

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"*** **+↲** on the terminal. (This opens REPL or Read Evaluate Print Loop)

Comments

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

↳ **Can be used to mark another name, data etc.**

Types of Comments

There are two types of comments in Python

1. Single line comments ➔ Written using #
2. Multiline comments ➔ Written using ‘’’ comment ‘’’

Chapter-2: Variables and Datatypes

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

a = 20
b = “Praveen”
c = 12.22
➔ Variables = Container to store a value
➔ Keywords = Reserved word in Python
➔ Identifiers = Class / Function / Variable name

Data Types

Primarily there are 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 as a class <int>
b = 88.44 ➔ Identifies as a class <float>
c = “Praveen” ➔ Identifies as a class <str>

Rules for defining a variable name ➔ Also applies to other identifiers.

- A variable name can contain alphabets, digits and underscores
- A variable name can only start with an alphabet and underscores
- A variable name can start with a digit
- No white space is allowed to be used inside a variable name.

Examples of a few variable names are:- **praveen, ones, seven_, _seven etc.**

Operators in Python

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>

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 integer conversion
float(32) ➔ 32.0 =) integer to float conversion
…and so on.
Here “31” is a string literal and 31 a numeric literal.

input() function

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

a = input(“Enter name: “) ➔ if a is “Praveen”, the user entered Praveen
➔ if a is “34” user entered 34.
It is important to note that the output of input is always a string (even if the number is entered)

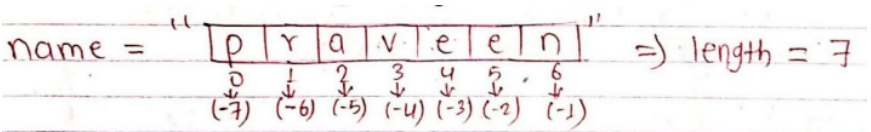
Chapter-3: Strings

String is a data type in Python. String is a sequence of characters enclosed in quotes.
We can primarily write a string in these three ways.

1. Single quoted strings ➔ a = ‘Praveen’
2. Double quoted strings ➔ b = ‘’Praveen”
3. Triple quoted strings ➔ c = ‘’’Praveen’’’

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. In order to slice a string, we use the following syntax:

Sl = name [ind_start : ind_end]

⇩ ⇩

first index include last index is not include

Sl[0:3] returns “Pra” ➔ Character from 0 to 3

Sl[1:3] returns “ra” ➔ Character 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] ➔ ‘m z n’

Other advanced slicing techniques

word = “amazing”

Word[:7] ➔ word[0:7] ➔ ‘amazing’

word[0:] ➔ word[0:7] ➔ ‘amazing’

String Slicing

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

1. len() function ➔ This function returns the length of the string
   len(“praveen”) ➔ returns 7
2. string.endswith(“een”) ➔ This function tells whether the variable string ends with the string “een” or not. If the string is “Praveen”, it returns true for “een”, since Praveen ends with een.
3. string.count(“c”) ➔ Counts the total number of occurrence of any character.
4. string.capitalize() ➔ This function capitalizes the first character of a given string.
5. string.find(word) ➔ This function finds a word and returns the index of first occurrence of that word in the string.
6. string.replace(oldword, newword) ➔ This function replaces the oldword with newword in the entire string.

Escape Sequence Character

Sequence of character after backslash ‘ \ ’ ➔ Escape sequence character.

Escape sequence character comprises more than one character but represents. One_character when used within the strings.

Examples \n , \t , \\ etc

⇩ ⇩ ⇩

new line tab backslash

Chapter-4: Lists and Tuples

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

friends = [“Apple”, “Praveen”, “Puja”, 7, False]

⇩ ⇩ ⇩

str() int() bool()

⇓

Can store value of any datatype

List Indexing

A list can be indexed just like a string

L1 = [7, 9, ”Praveen”]

L1[0] ➔ 7
L1[1] ➔ 9
L1[50] ➔ Error
L1[0:2] ➔ [7, 9] =) List slicing

List Methods

Consider the following list

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

1. L1.sort() ➔ Updates the list to [1, 2, 7, 8, 15, 21]
2. L1.reverse() ➔ Updates the list to [15, 21, 2, 7, 8, 1]
3. L1.append(8) ➔ Adds 8 at the end of the list
4. L1.insert(3, 8) ➔ This will add 8 at 3 index
5. L1.pop(2) ➔ Will delete element at index 2 and return it’s value
6. L1.remove(21) ➔ Will remove 21 from the list

Tuples in Python

A tuple is an immutable data type in Python

↳ **Cannot change**

a = () ➔ Empty tuple
a = (1,) ➔ Tuple with only one element needs a comma

a = (1, 7, 2) ➔ Tuple with more than one element

Once defined a tuple element can't be altered or manipulated.

Tuples Method

Consider the following tuple

a = (1, 7, 2)

1. a.count(1) ➔ a.count(1) will return number of times 1
2. a.index(1) ➔ a.index(1) will return the index of first occurrence of 1 in a

Chapter-5: Dictionary & Set's

Dictionary is a collection of key-value pairs.

Syntax:

```
a  = {
        "key" : "value",
        "Praveen": "code",
        "marks": 100,
        "list": [1, 2, 9]
        }
```

a["key"] ➔ prints "value"

a["list"] ➔ prints [1, 2, 9]

Properties of a Python Dictionary

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

Dictionary Methods

Consider the following dictionary

```
a = {
        "name":  "Praveen",
        "from": "Nepal",
```

"marks": [92, 98, 96]
}

1. a.items() ➔ return a list of (key, value) tuples
2. a.keys() ➔ return a list containing dictionaries keys
3. a.update({"friend": "Binod"}) ➔ updates the dictionary with supplied key-value pairs
4. a.get("name") ➔ return the value of the specified keys (and value is returned e.g. "Praveen" is returned here)

More methods are available on docs *python.org*

Sets in Python

Set is a collection of non repetitive elements

```
s = set()          =) Non repetition allowed
s.add(1)
s.add(2)           =) or set = {1, 2}
```

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

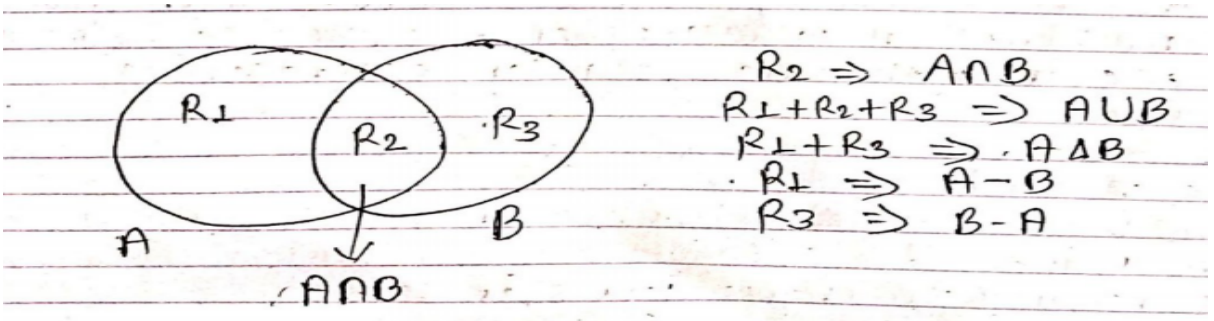
1. Sets are unordered ➔ Elements order doesn't matter
2. Sets are unindexed ➔ Cannot access elements by index
3. There is no way to change items in sets
4. Sets cannot contain duplicate values

Operations on Sets

Consider the following set:

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

1. len(s) ➔ returns 4, the length of the set
2. s.remove(s) ➔ updates the set s and remove 8 from s
3. s.pop() ➔ removes an arbitrary element from the set and returns the element removed
4. s.clear() ➔ Empties the set s
5. s.union() ➔ returns a new wet with all items from both sets =) {1, 8, 2, 3, 11}
6. s.intersection({8, 11}) ➔ returns a set which contains only items in both sets =) {8}



Chapter-6: Conditional Expressions

Sometimes we want to play PUBG on our phone if the day is Saturday.
Sometimes we order Icedream online if the day is Sunny.
Sometimes we go hiking if our parents allow it.

All these are decisions which depend on a condition being met. In Python Programming too, we must be able to execute instructions on a condition (s) being met. This is what conditionals 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 (condition 1):                =) if condition 1 is true
        print("Yes")
elif (condition 2):              =) if condition 2 is true
        print("No")
else:                            =) otherwise
        print("May be")
```

Code Example:

```
a = 22
If (a > 9):
        print("Greater")
else:
        print("Lesser")
```

Relational 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 & false to true

elif clause

elif in Python means [else if]. An if statement can be claimed together with a lot of these elif statements followed by an else statement.

```
if (condition 1):
        #code
elif (condition 2):
        #code          =) This ladder will stop once a condition in an if or elif is met
elif (condition 3):
        #code
.
.
.
else:
        #code
```

Important notes

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

Chapter-7: Loops in Python

Sometimes we want to repeat a set of statement in our program, for instance: Print 1 to 1000

Loops make it easy for a programmer to 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

1. **While loop**

The syntax of a while loop looks like this:

```
while condition:          =) The blocks keep executing until the condition is true
        Body of the loop
```

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 & execution] is continued until the condition becomes False.

An example

```
i = 0                =) prints “Praveen” 5 times
while i < 5:
        print(“Praveen”)
        i += 1
```

Note: If the condition never becomes False, the loop keeps getting executed.

2. **For loop**

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

The syntax of a for loop looks like this:

```
l = [1, 7, 8]
for item in l:
        print(item) =) 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 60
```

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 exhausted !
```

Output:

```
1
7
8
Done
```

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
```

The continue statement

continue is used to stop the current interaction of the loop and continue with the next one. It instructs the program to "skip the iteration".

Example:

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

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.
```

Chapter-8: Functions & Recursions

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

When a program gets integer 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 .

Example and syntax of a function

The syntax of a functions looks as follows

```
def func_1():
    print("Hello")
```

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:

func_1() ➔ This called function call.

Function Definition

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

Type of function 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

Example of built in functions include len(), print(), range() etc.

The func_1() function we defined is an example of user defined function.

Functions with Arguments

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

```
def greet(name):
        gr = “Hello” + name
        return gr ↱ “Praveen” is passed to greet in name
a = greet(“Praveen”)
                ↳ a will now contain “Hello Praveen”
```

Default parameter value

We can have a value as default argument in a function. If we specify name = “stranger” in line containing def, this value is used when no argument is passed.

Example:

```
def greet (name = “stranger”):
        # Function body
greet() ➔ name will be “stranger” in function body (default)
greet(“Praveen”) ➔ name will be “Praveen” in function body (passed)
```

Recursion

Recursion is a function which calls itself. It is used to directly a mathematical formula as a function. For example

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

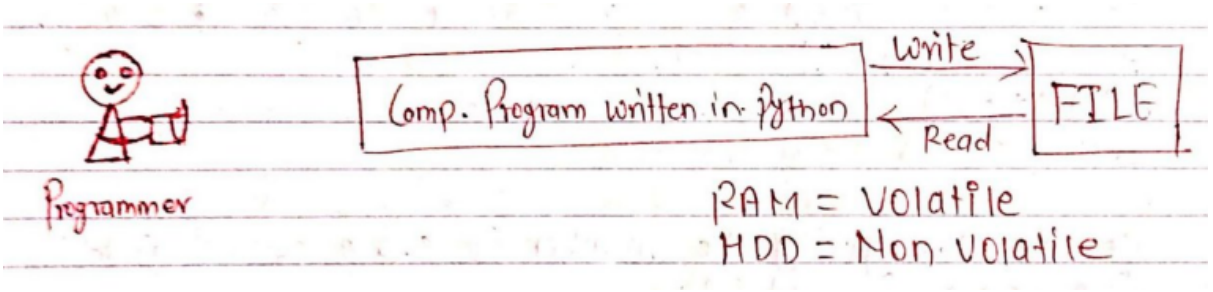
This function can be defined as follows:

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

Chapter-9: 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 two 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.

```
        ↱ filename
open("this.txt", "r")
 ⇩              ↳ mode of opening (read mode)
```

Open is a built in function

Reading file in Python

f = open("this.txt", "r") ➔ open the file in r mode
text = f.read() ➔ read its contents
print(text) ➔ print its contents
f.close() ➔ close the file

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

```
function : f.read(2)
                ↳ read first 2 characters
```

Other methods to read the file

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

f.readline() ➔ reads one line from the file

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 a file, we first open it in write or append mode, after which, we use 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()
```

with statement

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

```
with open(“this.txt”) as f:
    f.read()
```

↳ ***don’t need to write f.close() as it is done automatically***

Chapter-10: Object Oriented Programming

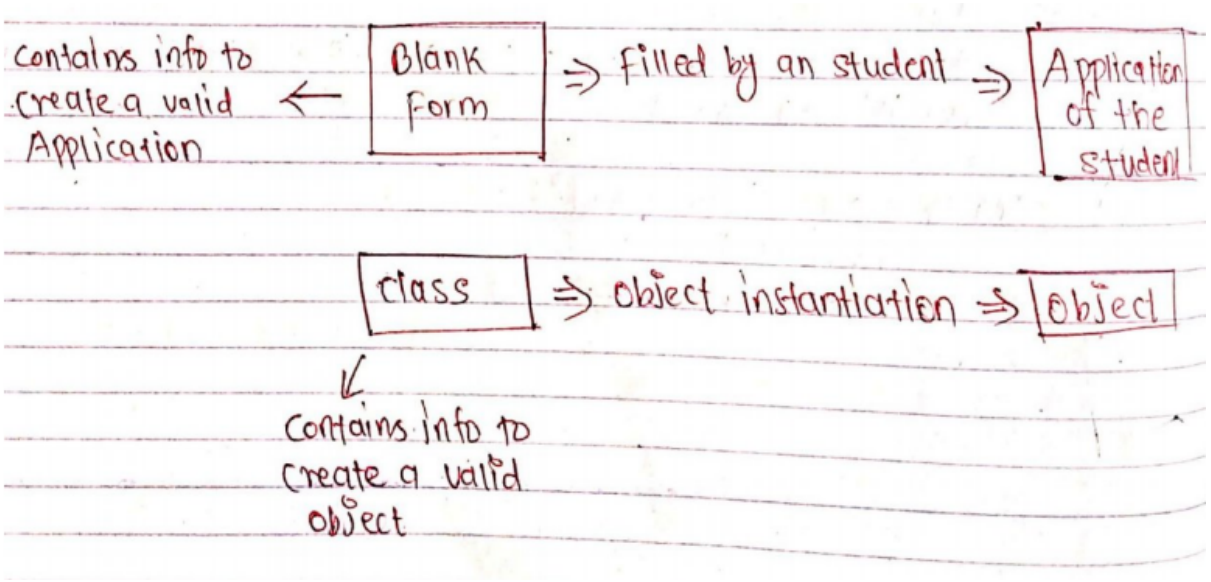
Solving a problem by creating objects is one of the most popular approaches in programming. This is called object oriented programming.

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

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. ➔ Abstract & 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]
    praveen = Employee() ➔ Object instantiation
    Praveen.company
    Employee.company = “YouTube”  ➔ Changing class attribute
```

Instance Attributes

An attribute that belongs to instance (object). Assuming the class from the previous example.

```
praveen.name = “Praveen”
praveen.salary = “47k” ➔ Adding instance attribute
```

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

‘self’ parameter

self refers to the instance of the class. It is automatically passed with a function call from an object.

```
praveen.getSalary() ➔ here self is praveen
                        ↳ equivalent to Employee.getSalary(praveen)
```

The function getSalary is defined as:

```
Class Employee:
    Company = “Google”
    def getSalary (self):
        print(“Salary is not there”)
```

Static Method

Sometimes we need a function that doesn’t use the self parameter. We can define a static method like this.

```
@staticmethod ➔ decorator to mark greet as a static method
def greet():
    print(“Hello user”)
```

__init__() constructor:

__init__() is a special method which is first run as soon as the object is created
__init__() method is also known as constructor

It takes self argument and can also take further arguments.

For example:

```
Class Employee:
    def __init__(self, name):
        Self.name = name
    def getSalary(self):
        -
        -
        -
praveen = Employee("praveen")
```

⇩

Object can be instantiated using constructor like this !

Chapter-11: Inheritance & more on Oops

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

Syntax:

```
[   Class Employee:                    ➔ Base class
        #code
    Class Programmer(Employee):        ➔ Derived or child class
        #code
```

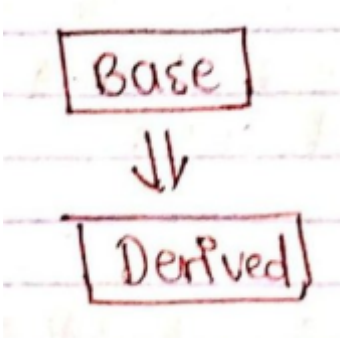
We can use the methods and attributes of the Employee in Programmer object. Also we can overwrite or add new attributes and methods in Programmer class.

Types of Inheritance

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

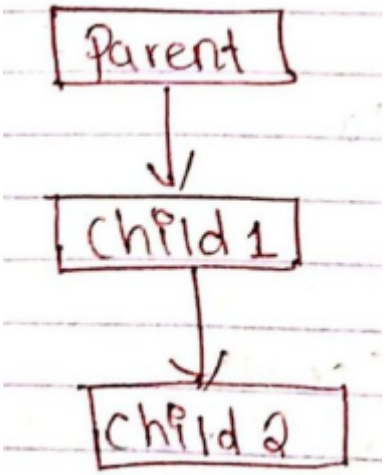
Single Inheritance

Single inheritance occurs when a child class inherits only a single 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 super class in the derived class

```
super().__init__()
```

⇩

class constructor of the base class

Class method

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 func(cls, p1, p2):
    . . .
```

@property decorators

Consider the following class

```
Class Employee:
    @property
    def name(self):
        return self.name
```

If e = Employee is an object of class employee, we can print(e.name) to print the name/ 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.name = value
```

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)
P1 / P2 =) P1__truediv__(P2)
P1 // P2 =) P1__floordiv__(P2)

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)

Chapter-12: Advanced Python 1

Exception Handling in Python

There are many built-in exceptions which are raised in Python when something goes wrong. Exceptions in Python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
    #code ➔ code which might throw Exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exception to catch like below.

```
try:
    #code
except zeroDivisionError:
    #code
except typeError:
    #code
except :
    #code ➔ All other exceptions are handled here.
```

Raising Exception

We can raise custom exceptions using the raise keyword in Python.

try with else clause

Sometimes we want to run a piece of code when try was successful.

```
try:
    #some code
except:
    #some code
else:
    #code ➔ This is executed only if the try was successful.
```

try with finally

Python offers a finally clause which ensures execution of a piece of code irrespective of the exception.

```
try:
        #some code
except:
        #some code
finally:
        #some code ➔ executed regardless of error !
```

if__name__ == '__main__' in Python

__name__ evaluates to the name of the module in Python from where the program is run.

If the module is being run directly from the command line, the __name__ is set to string "__main__". Thus, this behaviour is used to check whether the module is run directly or imported to another file.

The global keyword

global keyword is used to modify the variable outside of the current scope.

enumerate function in Python

The enumerate function adds a counter to an iterable and returns it.

```
for i, item in list1:
        print(i, item)
                ↳ prints the items of list1 with index.
```

List comprehensions

List comprehension is an elegant way to create lists based on existing lists.

```
list1 = [1, 7, 12, 11, 22]
list2 = [ i for item in list1 if item > 8 ]
```

Chapter-13: Advanced Python 2

Virtual Environment

An environment which is the same as the system interpreter but is isolated from the other Python environments on the system.

Installation

To use virtual environment, we write

pip install virtualenv ➔ install the package

We create a new environment using:

virtualenv myproject ➔ create a new venv

The next step after creating the virtual environment is to activate it. We can now use this virtual environment as a separate Python installation.

pip freeze command

pip freeze returns all the packages installed in a given python environment along with the versions.

"pip freeze > requirements.txt"

The above command creates a file named requirements.txt in the same directory containing output of pip freeze. We can distribute the file to other users and they can recreate the same environment using:

pip install -r requirements.txt

Lambda Function

Function created using an expression using lambda keyword.

Syntax:

lambda arguments : expressions

↳ ***can be used as a normal function***

Example:

```
square = lambda x : x * x
square (6) ➔ return 36

sum = lambda a,b,c : a+b+c
sum(1, 2, 3) ➔ return 6
```

bin method (strings)

Creates a string from iterable objects

```
l = [ “apple”, “mango”, “banana” ]
“,and,”.join(l)
```

The above line will return “apple, and, mango, and, banana”

format method (strings)

formats the values inside the string into a desired output.

```
template.format(P1, P2, …)
                    ↳ arguments
```

Syntax for format looks like:

```
“{} is a good {}”.format(“Praveen”, “boy”)
        Output: Praveen is a good boy
“{1} is a good {0}.format (“Praveen”, “boy”)
        Output: boy is a good Praveen
```

Map, Filter & Reduce

Map applies a function to all the items in an input_list.

```
Syntax:
    map(function, input_list)
        ↳ can be lambda function
```

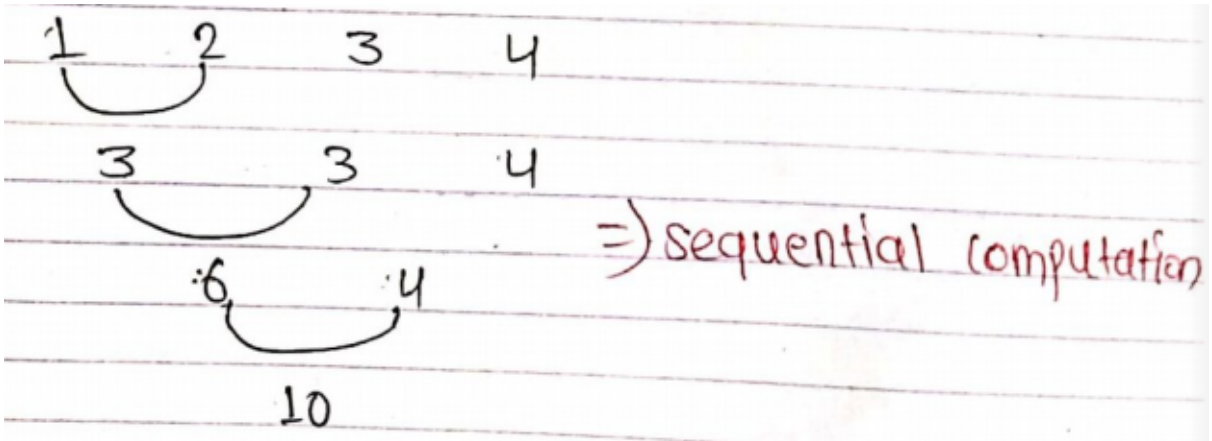
Filter creates a list of items for which the function returns true.

```
list(filter(function))
        ↳ can be a lambda function
```

Reduce applies a rolling computation to a sequential pair of elements.

```
from functions import reduce
val = reduce(function, list1)
        ↳ can be a lambda function
```

If the function computes sum of two numbers and the list is [1, 2, 3, 4]


1 2 3 4
3 3 4
6 4
10
=) Sequential computation