

Python

Python: Python is a simple and easy to understand language which feels like reading simple English.

Features

- ⇒ Easy to understand i.e it takes less development time.
- ⇒ Free and open source.
- ⇒ High level language
- ⇒ It works in Linux/Windows/Mac.

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.

- * Install Visual Studio Code and the recommended Python extension.

Modules, Comments & pip

lets write our very first python program. Create a file called `hello.py` and paste the below code in it

```
print("HelloWorld")
```

↑ Print is a function

Execute this file (.py file) by typing `python hello.py` and you will see Hello World printed on 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 the package manager for python. You can use pip to install a module on your system

Eg. `pip install flask` (installs flask 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 be install using pip.

* Some example of built in module are os, abc etc.

To import use: Import os, abc...

* Some example of external module are tensorflow, flask etc.

Eg. • Import flask (command)

Using Python as a calculator

We can use python as a calculator by typing "python" + ↲ on terminal

This opens REPL ↲

on Read Evaluate Print loop.

Comments

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

- * Comments are used to mark author name, date etc.

Types of comments

There are two types of comment in Python.

- i) Single line comments: Written using '#'
- ii) Multiline comments: Written using "Comment", "

Variables and datatypes

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

a = 30

b = "Harry"

c = 71.22

- * Variable: Container to store a value.
- * Keywords: Reserved words in Python.
↳ Can't be use as variable.

Identifier: Class / Function / Variable Name.

Data types

Primarily there are following data types in Python.

- i) Integer
- ii) Floating point numbers
- iii) String
- iv) Boolean
- v) None.

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

a = 71 \Rightarrow Identifies 'a' as class (int)

b = 88.2 \Rightarrow Identifies 'b' as class (float)

name = "Harry" \Rightarrow Identifies name as class (str)

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

\Rightarrow A variable name can contain alphabets, digits and underscores.

\Rightarrow A variable name can only start with an alphabet & underscore.

- No variable name can't start with a digit.
- ⇒ No space is allowed to be used inside a variable name.

Examples of a few variable name are:-
a, b, Harry, seven, -seven, ones

Operations in Python

Following are some common operators in python.

- i) Arithmetic operators ⇒ +, -, *, / etc
- ii) Assignment operators ⇒ =, +=, -=, etc
- iii) Comparison operators ⇒ ==, >, >=, <, etc
- iv) Logical operators ⇒ and, or, not

Type() function and Typecasting
Type function is used to find the datatype of a given variable in Python.

a = 21

type(a) ⇒ Class <int>

b = "21"

type(b) ⇒ Class <str>

- * A number can be converted into a string and vice versa (if possible). There are many functions to convert one datatype into another.

`str(31) => "31" \Rightarrow Integer to String
int("32") => 32 \Rightarrow String to integer
float(33) => 33.0 \Rightarrow Integer to float
... 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")`

\hookrightarrow If a is "Harry" the user entered Harry.

- * Note : It is important to note that the output of `input` is always a string (even if the number is entered).

\hookrightarrow If a is "34" user enter 34.

String

- ⇒ String is a datatype in Python.
 - ⇒ String is a sequence of characters enclosed in quotes.
- * We can primarily, write a String in these three ways -
- i) Single quoted string → $a = 'Name'$
 - ii) Double quoted string → $b = "Name"$
 - iii) Triple quoted string → $c = """Name"""$

String

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

Name = "H a n g y" $\Rightarrow \text{length} = 5$

0 1 2 3 4
↓ ↓ ↓ ↓
(-5) (-4) (-3) (-2) (-1)

The index in a string starts from 0 to (length - 1) in python. In order to slice a string, we use the following syntax.

$sl = \text{name}[ind_start : ind_end]$

First index included. Last index isn't included

`str[0:3]` returns "Hai" → (Characters from 0 to 3)

`str[1:3]` returns "ai" → (Character from 1 to 3)

Negative Indices

Negative indices can also be used as shown in the figure above.

-1 corresponds to (`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] → "maz"`

Other advanced Slicing techniques

`word = "amazing"`

`word[:7] → word[0:7] → "amazing"`

`word[0:] → word[0:7] → "amazing"`

String function

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

- 7) `len()` function: This function returns the length of the string.
`len("Harry")` → returns 5
- 2) `String.endswith("ny")` → This function tells whether the variable string ends with the string "ny" or not if string is "Harry", it returns true for "ny" since Harry ends with ny.
- 3) `String.Count("c")` → Counts the total no. of occurrence of any character
- 4) `String.capitalize()` → This function capitalize 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 of more than one character but represents one character when used within the strings.

Examples \n , \t , \, \" etc
Newline Tab Single → backslash
Quote

Lists and Tuples

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

Friends = ["Apple", "Mango", "Rohan", 7,
(str) False].
(int)

[] → can store value of any datatype.
↳ boolean

List indexing

A list can be indexed just like a string.

$l1 = [7, 9, "Harry"]$

- $l1[0] \Rightarrow 7$ (Result)

- $l1[1] \Rightarrow 9$ (Result)

- $l1[70] \Rightarrow \text{Error}$ (Result)

- $l1[0:2] \Rightarrow [7, 9]$ \Rightarrow List Slicing

List methods

Consider the following list:

$l1 = [1, 8, 7, 2, 21, 15]$

i) $l1.append()$: updates the list to
 $[1, 2, 7, 8, 15, 21]$

ii) $l1.reverse()$: Updates the list to
 $[15, 21, 2, 7, 8, 1]$

iii) $l1.append(8)$: Adds 8 at the end of
the list.

iv) $l1.insert(3, 8)$: This will add 8 at
3 index.

v) $l1.pop(2)$: Will delete element at
index 2 and return its value

vi) $l1.remove(21)$: 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's element can't
be altered or manipulated

Tuple methods

Consider the following tuple

a = (1, 7, 2)

i) a.count(1): a.count(1) will return
number of times 1 occurs in
a.

ii) a.index(1): a.index(1) will return
the index of first occurrence
of 1 in a.

Dictionary & Sets

Dictionary is a collection of key-value pairs.

Syntax:

a = { "Key": "Values", "Harry": "Code", "marks": "100", "list": [1, 2, 9] }	↓ keys ↓ values
--	--------------------------

a["Key"] \Rightarrow Prints "Value"

a["list"] \Rightarrow Prints [1, 2, 9]

Properties of a Python Dictionaries

- i) It is unorderred
- ii) It is mutable
- iii) It is indexed
- iv) Cannot contain duplicate keys.

Dictionary methods

Consider the following dictionary

a = { "name": "Harry",
 "From": "World",
 "marks": [92, 98, 96] }

- i) `a.items()`: Returns a list of (key,value) tuples.
- ii) `a.keys()`: Returns a list containing dictionary's keys.
- iii) `a.update({ "friend": "Sam" })`: Updates the dictionary with supplied key-values pairs.
- iv) `a.get("name")`: Returns the value of the specified keys (and value is returned eg. "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)`

⇒ on `set = {1,2}`

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

Properties of Sets

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

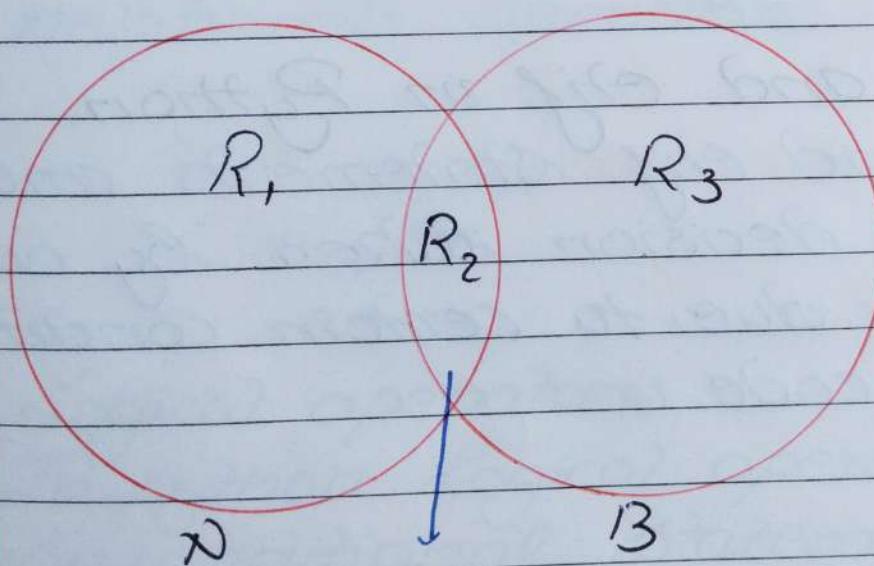
Operations on Sets.

Consider the following set:

$$S = \{1, 8, 2, 3\}$$

- i) `len(S)`: Returns 4, the length of the set.
- ii) `S.remove(8)`: Updates the set S and remove 8 from S.

- iii) $S.pop()$: Removes an arbitrary element from the set and returns the element removed.
- iv) $S.clear()$: Empties the set S .
- v) $S.union(\{8, 82\})$: Returns a new set with all items from both sets $\Rightarrow \{1, 8, 2, 3, 11\}$
- vi) $S.intersection(\{8, 11\})$: Returns a set which contains only items in both sets $\Rightarrow \{8\}$



$$\begin{aligned}
 R_1 &= A \cap B \\
 R_1 + R_2 + R_3 &\Rightarrow \cancel{R_1} \cup B \\
 R_1 + R_3 &\Rightarrow A \Delta B \\
 R_1 &= A - B \quad , \quad R_3 = B - A
 \end{aligned}$$

Conditional Expressions

- Sometimes we order Icecream if the day is sunny.
- Sometimes we go hiking if our parents allow.
- * All these are decisions which depends on a condition being met.

In python programming too, we must be able to execute instruction 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 (condition1): \Rightarrow if (1) is true
 print("Yes")

elif (Condition2): \Rightarrow if 2nd is true
 print("No")

Else :

\Rightarrow otherwise

print("Maybe")

Code example

a = 22

if (a > 9) :

 print("Greater")

else :

 print("Lesser")

Relational Operators

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

= = \Rightarrow Equals

> = \Rightarrow Greater than/equal to

< = , etc.

Logical operators

In python logical operators operate on conditional statement. Eg.

* and \rightarrow true if both operands are true
else false

* or \rightarrow 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 chained together with a lot of these elif statements followed by an else statement.

if (Condition1):
 #Code

elif (Condition2):
 #Code

elif (Condition3):
 #Code

:
Else:
 #Code.

* This ladder will stop once a condition in an if or elif is met.

Note

- i) There can be any number of elif statement
- ii) last Else is executed only if all conditions inside elif 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 these one by one!

While loop

The syntax of a while loop looks like this:

While condition:

```
# Body of the loop
```

- The block keeps executing until

the condition is true.

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 become false.

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

An example.

$i = 0$

while $i < 5 \Rightarrow$ Print "Harry"- 5 times!

print("Harry")

$i = i + 1$

Note: If the condition never become 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 list, tuple or string [iterables]

The syntax of a for loop looks like this:

$l = [1, 7, 8]$

for item in l:

 print(item) \Rightarrow 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()

Example demonstrating range()

For i in range(0, 7): \Rightarrow range(7) can also be used.
 print(i)

Date / /

Result: This prints 0 to 6.

For loops with else.

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

Eg.

$$l = [1, 7, 8]$$

For item in l:

 print(item)

Else:

 print("Done")

↳ This is printed when the loop exhausts

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)
 if $i == 3$
 break
 \Rightarrow This will print
 0, 1, 2 and 3.

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 $i == 2$ \Rightarrow if i is 2, the
 continue iteration is skip.
 print(i)

Pass statement

Pass is a null statement in Python. It instruct to "do nothing"

Eg.

$l = [1, 7, 8]$
 for item in l :
 pass

[Without pass, the
 program throw
 an Error.]

Functions and Recursions

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

When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track on 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")
```

Function Call

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

func1() → This is function call.

Function definition

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

Quick Quiz: Write a program to greet a user with "Good day" using function

Types of function in Python

- i) Built in function \Rightarrow Already present in .py
- ii) User defined function

* Example of built in function are len() , print() , range() etc.

* The func1() 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 the parenthesis. A function can also return values as shown below:

```
def greet(Name):
    gr = "Hello" + name
    return gr
```

*"Harry" is passed to greet
in Name
a = greet("Harry")
↳ a will now contain "Hello
Harry".*

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.

For example:

```
def greet(name = "stranger"):
    # function body
```

greet() → Name will be "stranger" in function body (default)

greet("Harry") → Name will be "Harry" in function body (passed)

Recursion

It is a function which calls itself

- It is used to directly use a mathematical formula as a function.
For Eg.

factorial(7) = 7 * factorial(7-1)

This function can be defined as:

```
def factorial(n):
```

if $i == 0$ or $i == 1$: \Rightarrow Base condition

return 1 which doesn't call

Else

function only further

return n * factorial(n-1)

↳ Function calling itself

This works as follows:

Factorial(4)

11

↳ [Function called]

\checkmark $4 \times \text{Factorial}(3)$

$$4 \times [3 \times \text{factorial}(2)]$$

$$4 \times 3 \times [2 \times \text{factorial}(1)]$$

$4 \times 3 \times 2 \times 1$ [Function returned]

Programmers need to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep

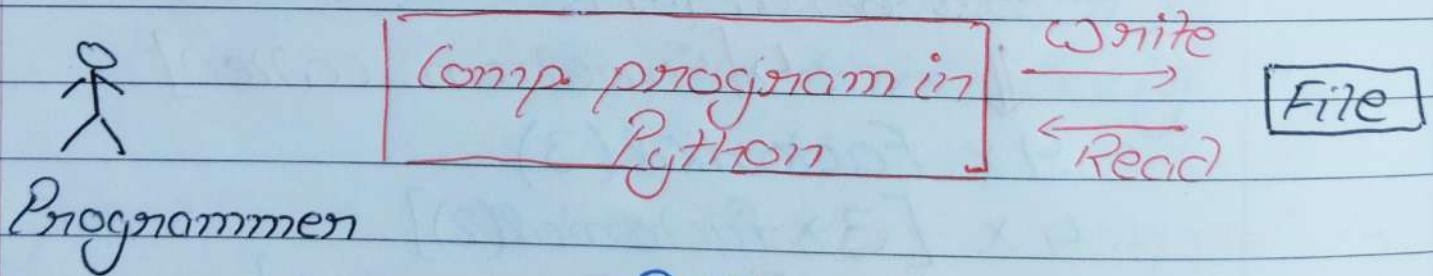
calling itself.

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.



RAM = Volatile

HDD = Non-volatile.

Types of Files

- i) Text files (.txt, .c etc)
- ii) 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")



File name

mode of opening
(Read mode)

Open is a built-in function

Reading a file in Python.

f = open("this.txt", "r") \Rightarrow Open file in
r mode

text = f.read() \Rightarrow Read its content

print(text) \Rightarrow Print its Content

f.close() \Rightarrow Close file

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

f.read(2)

\hookrightarrow Read first 2 characters

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

`f.readline()` → Read 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 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()`

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.

Object oriented programming

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

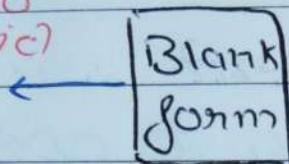
- This concept focuses on using reusable code

↳ Implements DRY Principle

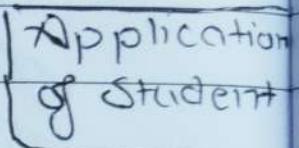
Class

A class is a Blueprint for creating object

Combine info to
create a valid
application



⇒ Filled by a Student



Class ⇒ Object instantiation ⇒ Object
contains info to
create a valid object.

The syntax of a class looks like this

Class Employee: [classname is written
method & variable in Pascal
case]

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 detail
to the user \Rightarrow Abstraction &
Encapsulation!

Modelling a problem in OOPS
We identify the following in our
problem.

Name \rightarrow Class \rightarrow Employee

Adjective \rightarrow Attributes \rightarrow name, age, salary

Verbs \rightarrow Methods \rightarrow getSalary(),
increment()

Class Attributes

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

Example:

Class Employee :

Company = "Google"

\Rightarrow Specific to each class.

Harry = Employee() \Rightarrow Object
instantiation

Harry.Company
Employee.Company = "YouTube"

⇒ Changing class attribute.

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)

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

Harry.attribute1 → i) Is attribute 1

present in object?

ii) Is attribute 1 present
in class?

'Jeff' parameter

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

Harry.getSalary() \Rightarrow Here self is Harry.
Equivalent to Employee.getSalary(Harry)

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
def greet():
    print("Hello user")
```

\Rightarrow decorator to mark greet as a static method

-- 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 Eg. Class Employee:

```
def __init__(self, name):
    self.name = name
```

```
def get salary(self)
```

...

Harry = Employee("Harry")

Object can be instantiated using
constructor like this

Inheritance & more on OOPs

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

Syntax

Class Employee \Rightarrow Base Class
Code

...

class programme (Employee):

Code

Derived or Child class

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

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

Types of inheritance

- i) Single inheritance
- ii) Multiple inheritance
- iii) Multilevel inheritance

Single Inheritance

It occurs when child class inherits only a single parent class.

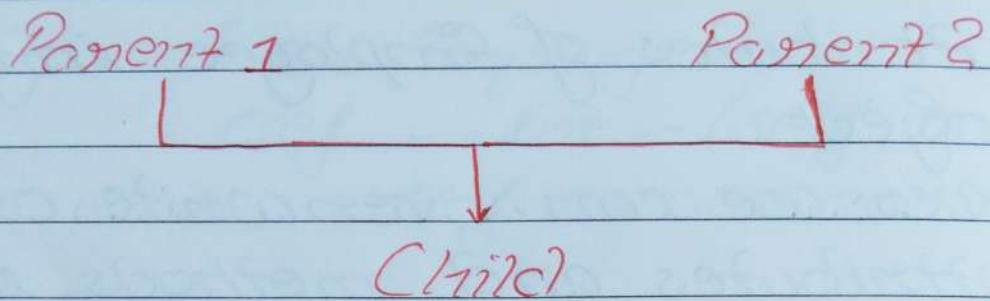
Base



Derived

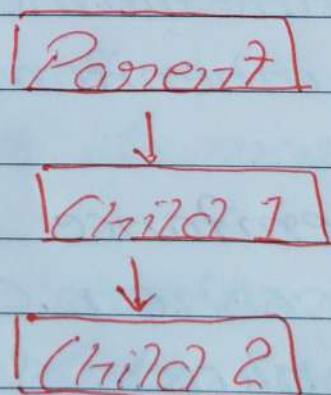
Multiple Inheritance

Multiple inheritance 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

It is used to access the methods of a super class in the derived class.

`Super().__init__()`

↳ Calls constructor of
Base class

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(p₁, p₂):

* @ property decorator

Consider the following class

class Employee

@ property

def name(self):

return self.ename

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.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.

Operations in python can be
overloaded using the following
methods:

$$P_1 + P_2 \rightarrow P_1 -- add -- (P_2)$$

$$P_1 - P_2 \rightarrow P_1 -- sub -- (P_2)$$

$$P_1 / P_2 \rightarrow P_1 -- truediv -- (P_2)$$

$$P_1 // P_2 \rightarrow P_1 -- floordiv -- (P_2)$$

$$P_1 * P_2 \rightarrow P_1 -- mul -- (P_2)$$

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)`

Advance Python 1

Exception Handling in Python

There are many built-in exception which are raised in Python when something goes wrong.

Exception in Python can be handled using a `try` statement. The code that handles the exception is written in the `except` clause.

`try:`

`# Code`

`except exception as e:`
`print(e)`

→ Code which might throw Exception

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

We can also specify the exceptions to catch like below:

try:
 #Code

except ZeroDivisionError:
 #Code

except TypeError:
 #Code

Except:

 #Code. → All other exception
 are handled here

Raising Exception

We can raise custom exception 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 success.)

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 main 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 counter to an iterable and returns it.

For `i, item` in `list 1`:

`print(i, item)`

↳ Prints the items of lists 1 with index!

List Comprehensions

List Comprehensions 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]

Advance Python - 2

Virtual Environment

An Environment which is same as the System interpreter but is isolated from the other python environments on the system.

To Installation.

To use virtual environments, we write.

pip install virtualenv

↳ install the package

We create a new environment using
virtualenv myprojectenv → Creates a
new Venv.

The next step after creating
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 create a file named `requirements.txt` in the same directory containing the output of `pip freeze`.

We can distribute this file to other users and they can recreate the same environment using:

`pip install -r requirements.txt`

Lambda function

Functions 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) → (returns 36)

Sum = lambda a, b, c: a+b+c

Sum(1, 2, 3) → (returns 6)

Join methods (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)

Format the values inside the string into a desired output.

Date / /

template.format(p₁, p₂...)

↳ arguments

Syntax for format looks like:

"{} is a good {}".format("Harry", "boy") - (i)

"{} is a good {}".format("Harry", "boy") - (ii)

Output for (i)

Harry is a good boy

Output for (ii)

Boy is a good Harry

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 of sequential pair of elements

From func tools import reduce
val = reduce(function, list)

↳ Can be a lambda function

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

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \text{(brace)} & & & \\ 3 & 3 & 4 & \\ \text{(brace)} & & & \\ 6 & 4 & & \\ \text{(brace)} & & & \\ 10 & & & \end{array}$$

⇒ Sequential computation