PYTHON HANDWRITTEN NOTES

HAPPY LEARNING!

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("Hello World")
```
↑ 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
↳ or 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 ↵
or 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) Multi line 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.

Datatypes

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 ⇒ Identifies 'a' as class <int>
b=88.2 ⇒ Identifies 'b' as class <float>
name="Harry" ⇒ Identifies name as class <str>

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

⇒ A variable name can contain alphabets, digits and underscores.

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

Date / /

=> A 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, one8

Operators in Python

Following are some common operator 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 function to convert one datatype into another.

Str(31) ⇒ "31" ⇒ Integer to String
int("32") ⇒ 32 ⇒ String to integer
float(33) ⇒ 33.0 ⇒ 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")
↳ 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).
↳ 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:

$$\begin{array}{lccccc} \text{Name} = \text{“} & \boxed{H} & \boxed{a} & \boxed{r} & \boxed{r} & \boxed{y}\text{”} \\ & 0 & 1 & 2 & 3 & 4 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & (-5) & (-4) & (-3) & (-2) & (-1) \end{array} \Rightarrow \text{length} = 5$$

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 included   last index isn't included

sl[0:3] returns "Har" → (Characters from 0 to 3)

sl[1:3] returns "ar" → (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] → 'mzn'
```

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:

1) len() function: This function returns the length of the string.
len("Harry") → returns 5

2.) String endswith("rry") → This function tells whether the variable string ends with the string "rry" or not if string is "Harry", it returns true for "rry" since Harry ends with rry

3.) String Count("c") ⇒ Counts the total no. of occurence 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 occurence of that word in the string.

6.) String replace(oldword, newword) - This function replaces the oldword with new word 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

New line Tab Single Quote backslash

Lists and Tuples

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

Friends = ["Apple", "Mango", "Rohan", 7, False].

(str) (int) boolean

[ ] ⇒ can store value of any datatype.

List indexing

A list can be indexed just like a string.

Date / /

L1 = [7, 9, "Harry"]

- l1[0] ⇒ 7 (Result)
  L1[1] ⇒ 9 (Result)
  L1 [70] ⇒ Error (Result)
  L1[0:2] ⇒ [7, 9] ⇒ List Slicing

List methods
Consider the following list:
l1=[1, 8, 7, 2, 21, 15]

i) l1.sort(): 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 $\underbrace{\text{immutable}}$ 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 occurence of 1 in a.

Dictionary & Sets

Dictionary is a collection of key-value pairs.

Syntax:

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

a["Key"] ⇒ Prints "value"
a["list"] ⇒ Prints [1, 2, 9]

Properties of a Python Dictionaries

i) It is unordered
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}
```

Page No.
Date / /

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

Properties of Sets

1. Sets are unordered ⇒ Element's 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}

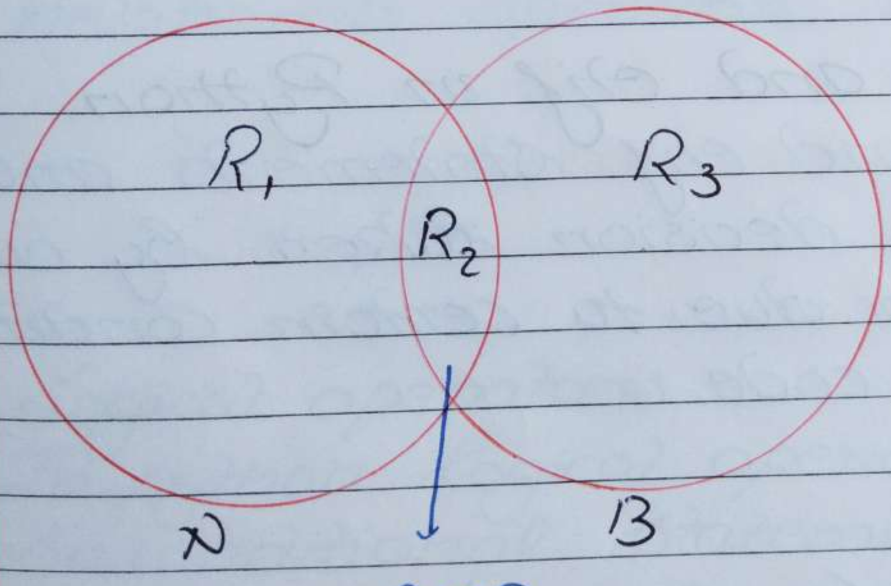
i) len(s): Returns 4, the length of the set.
ii) s.remove(8): Updates the set s and remove 8 from s.

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 ⇒ {1,8,2,3,11}

vi) S.intersection({8,11}) : Returns a set which contains only items in both sets ⇒ {8}



$R_2 = A \cap B$

$R_1 + R_2 + R_3 \Rightarrow$ ~~[illegible]~~ $A \cup B$

$R_1 + R_3 \Rightarrow A \triangle B$

$R_1 = A - B$, $R_3 = B - A$

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):          ⇒ if (1) is true
    print("Yes")
elif (Condition2):        ⇒ if 2nd is true
    print("No")
```

```
Else:                    ⇒ 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 relationals operators are:

- `= =` ⇒ Equals
- `> =` ⇒ Greater than/equal to
- `< =`, etc.

Logical operators

In python logical operators operate on conditional statement. Eg.

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

```
if (condition1):
        #Code
Elif (condion2):
        # 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 condition inside elif fail.

Date / /

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 programm-
-er 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        ⇒ 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

Date / /

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)   → 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()

```
For i in range(0,7):   → range(7) can also be use.
    print(i)
```

Result: This prints 0 to 6.

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.
Eg.

```
l = [1, 7, 8]
For item in l:
        print(item)
    Else:
        print("Done")
          ↳ This is printed when the
            loop exhaust
```

Output:

```
1
7
8
Done
```

The break statement
'break is used to come out of the loop when encountered. It instruct the program to - Exit the loop now.

Date / /

Example.

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

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
     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:      → Without pass, the
    pass              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 ⇒ 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.

Date / /

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

  factorial(n) = n x factorial (n-1)

This function can be defined as:

```
def factorial(n):
    if i==0 or i==1: ⇒ Base condition
       return 1          which doesnt call
  Else                   function any further
     return n* factorial(n-1)
                  ↳Function calling itself
```

This works as follows:

```
Factorial (4)
  ⇓      ↳[Function called]
 4x Factorial(3)
 4 x [3x factorial(2)]
 4 x 3x [2x factorial(1)]
 4x3x2x1 [Function returned]
```

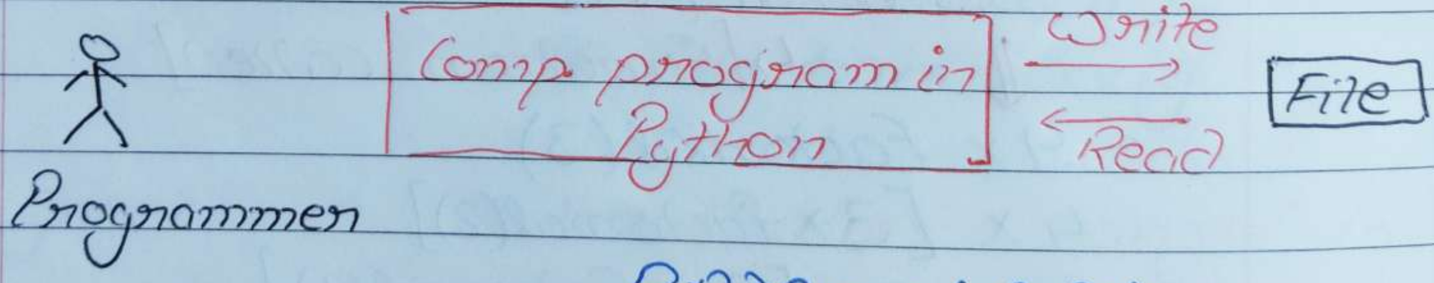
Programmer need to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep

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 parameter: 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") ⇒ Open file in r mode.

text = f. read() ⇒ Read its content
print(text) ⇒ Print its Content
f. close() ⇒ Close file

We can also specify the number of characters in read() function:
f. read(2)
↳ Read first 2 Character

Date / /

Other methods to read file
We can also use f. readline ()
function to read on 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

$$\begin{tikzcd}
\text{Combine info to create a valid application} & \boxed{\text{Blank form}} \arrow[l] \arrow[r, Rightarrow] & \text{Filled by a Student} \arrow[d, Rightarrow] \\
 & & \boxed{\text{Application of Student}}
\end{tikzcd}$$

$\boxed{\text{Class}}$ $\Rightarrow$ Object instantiation $\Rightarrow$ $\boxed{\text{Object}}$
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 ⇒ Abstraction & Encapsulation!

Modelling a problem in OOPs

We identify the following in our problem.

Noun → Class → Employee
Adjective → Attributes → name, age, salary
Verbs → Methods → get salary(), increment()

Class Attributes

An attributes 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.

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 & retreival.

Harry.attribute1 → (i) Is attribute 1 present in object?
(ii) Is attribute 1 present in Class?

'Self' parameter

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

Harry.get salary() ⇒ Here self is Harry.
↳ Equivalent to Employee.get salary (Harry)

The function get salary is defined as:

```
Class Employee:
    Company = "Google"
  def get salary (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
  def greet()           greet as a static
      print("Hello user")     method
```

-- init --() Constructor
* --int--() is a special method which is first run as soon as the object is created.
* --int--() method is also known

Date / /

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        ⇒ Base Class
    #Code
        ...
```

```
class programmer(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.

$$\begin{tikzcd}
\boxed{\text{Base}} \arrow[d, Rightarrow] \\
\boxed{\text{Derived}}
\end{tikzcd}$$

Date / /

Multiple Inheritance

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

$$\begin{tikzcd}
\text{Parent 1} \arrow[dr, dash] & & \text{Parent 2} \arrow[dl, dash] \\
& \text{Child} &
\end{tikzcd}$$

Multilevel Inheritance

When a child class becomes a parent for another child loss

$$\begin{tikzcd}
\boxed{\text{Parent}} \arrow[d] \\
\boxed{\text{Child 1}} \arrow[d] \\
\boxed{\text{Child 2}}
\end{tikzcd}$$

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.ename = 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:

$p_1 + p_2$	$\rightarrow$ 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--() on 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              → code which
except exception as e:    might throw
    print(e)                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:
      # somecode (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 ran.

Date / /

* 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.

```
list 1 = [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 interpretor 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 ~~coatching~~ 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)

"{1} is a good {0}".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

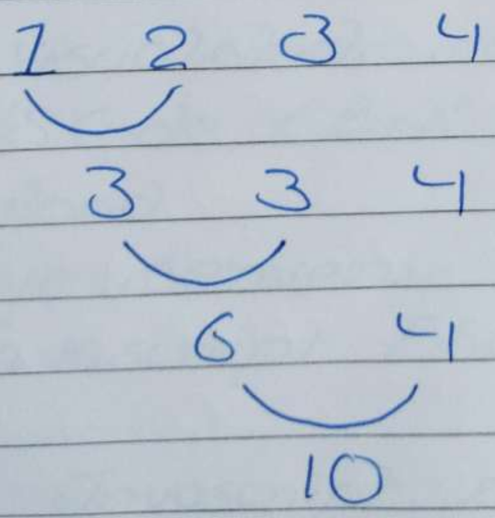
Filter creates a list of items for which the function returns true.

list(Filter(function))
↳ Can be a lambda function

Reduce applies a rolling computation of sequential pair of elements

From functools import reduce
val = reduce(function, list1)
↳ Can be a lambda function

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



⇒ Sequential Computation.