Python

* print('Hello') OR print("Hello World")

Modules

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

Pip

Pip is a package manager for python. we can use pip to install a module

ex - pip install ____ → name ex - flask etc

Types of modules

1.) Built in modules → pre installed in python
2.) External modules → Need to install using pip
   → ex - tensorflow, flask etc
   → ex - os, abc, etc

using python as calculator

* we can use python as a calculator by typing "python" in windows powershell or cmd
* This will open REPL
  or Read evaluate print loop
* ex - type 1+2
  = 3
  print("Hello") → Hello

Comments

is used to comment any line

''' Hell ''' → multiline comments

Datatypes

* integer
* floating point numbers
* Strings
* Booleans
* None

a = ''' ashish ''' is valid

* Python is case sensetive
* Comparison operator return bool type
* logical operator the (not) will reverse the things
* type() function

It is used to find the data type of a given variable in python

- Typecasting() function
  - A number can be converted into a string and vice a versa (if possible)
  - There are many ways 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

* input() function

This input allows the user to take input from the keyboard as a String

String

* The index in a string starts from 0 to (length-1) in python

```
a = name[0:3]
```

a	s	h	i	s	h
0	1	2	3	4	5
(-6)	(-5)	(-4)	(-3)	(-2)	(-1)

* Sometimes to skip the value

```
word = "amazing"
word[1:6:2] → 'mzn'
```

* String function

a = Hello

- len(a) function – returns length
- a.endswith("o") → True

  the word Hello is ending with o
- a.count("e") → 1

  Total no of e is 1

  Print(a.count("e"))

  works for word also
- a.capitalize() → Amazing

  makes 1 letter capital
- a.find("z") → 3

  tells at what no the z is
- a.replace("m","c") – acting replace m with c

* Escape sequence

\n - newline
\t - Tab
\' - single quote
\\ - backslash
** - use for power (2**3) = 8

List & Tuples

* Python list are containers to store a set of values of any data type

friend = [1, 2, 5, 75, 63]

[] - for List

We can → c = [int, String, Bool, float]

ex - [45, "ash", True, 7.9]

List Methods

- L = [1, 8, 7, 2, 21, 15]

① - L.sort() : update the list to [1, 2, 7, 8, 15, 21]
② - L.reverse() - " " " " [15, 21, 2, 7, 8, 1]
③ - L.append(8) - add 8 at the end of List
④ - L.insert(3, 8) - This will add 8 at 3 index
⑤ - L.pop(2) - will delete element at index 2 & return its value
⑥ - L.remove(21) - remove 21 from list

a = L.reverse().
Print(a) will not work in list

Tuple

() - for tuples

- A tuple is an immutable data type in python
  ↳ cannot change

a()
a(1,) - tuple with only 1 elements needs a comma
a(1,2,3)

- once defined a tuple cannot be altered or manipulated

- Methods

a = (1, 7, 2)

- a.count(1) - it will return number of times 1 occurs in a
- a.index(1) - it will return the index of first occurence of 1 in a

b = a.count(1)
Print(b) will work in tuples

Dictonary

{} - for dictionary

* It is a collection of key-value pairs

Syntax : a = { "key" : "value",
"ashish" : 'code',
"marks" : "100",
"list" : [1,2,9] }

a["key"] = print "value"
a["list"] = prints [1,2,9]

* properties

① - It is unordered
② - It is immutable
③ - It is indexed
④ - cannot contain duplicate keys

* Methods

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

```
a = {"name": "ashish",
     "from": "India",
     "marks": [10,19,24]}
```

④ - a.get("name") - return the value of specified keys ex. ashish will returned

* for more methods visit docs.python.org

Sets

* set is a collection of non-repetitive elements

```
a: {1,2,3}
```

* tuple can be added inside the set because we change the value of tuple but list cannot be added inside the set

```
S = set()
```

* properties of sets

- Sets are unordered
- Sets are unindexed
- There is no way to change items in sets
- Sets cannot contain duplicate value

* opesations s = {1,2,3,4}

① - len(s) - return 4, as length is 4

② - s.remove(4) - remove 4 from the set

③ - s.pop - removes an asbitary element from the set & returns the element removed

④ - s.clear - empties the set s

⑤ - s.union - ({8,11}) - return a new set with all items from both sets {11,8,2,3,4}

⑥ - s.intersection({4}) - return ~~contains~~ a set which contains only items in both sets - {4}

conditional Expression

* If, else & elif in python

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

```
if (condition):
    print("Yes")
elif (condition 2):
    print("No")
else:
    print("Error")
```

* Relational operators

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

== → equals
>= → greater than/equal to
<= → etc

* logical operator

In python logical operators operate on conditional statements - example

and - true if both operands are true
or - true if at least 1 operand is true
not - inverts true to false & false to true

c = [1, 12, 18]

Print(17 in c) → true if 17 found in list

loops

2 - types

- While loop
- for loop

* While loop

```
While condition :
    # body of loop
```

The block keeps executing until the condition is true

* for loop

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

* Range function

the range function in python is used to generate a sequence of numbers.
We can also specify the start, stop & step-size as follows:

```
range (start, stop, step-size)
```

* Pass statement

pass is a null statement in python
It instructs to 'do nothing'

```
Print ( i, end = "")
```

→ don't put next line.

Functions

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

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

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

```
func1()
```

* Types

- Built in functions - already present in pyth
- User defined functions - defined by user

* function with arguments.

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

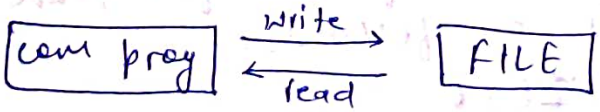
* if name = "stranger" then stranger will work as a default argument when there will be no parameter passed to name

Recursion

```
def factorial(n):
    if i==0 or i == 1
        return 1
    else
        return n* factorial(n-1)
this(" ")
use - Print(this.strip())
```

File I/O

The random access memory is volatile & all its contents are lost once program terminates. Inorder to persist the ~~lost~~ data forever, we use files



* Types

2 - types

① - Text files (.txt, c etc)

② - Binary files (jpg, dat, etc)

python has a lot of function for reading, updating and deleting files

* opening a file

python has an Open() function for opening files. It takes 2 parameters: filename & mode

```
Open ("this.txt","r")
```

"this.txt" → file name

"r" → mode of opening (read mode)

Open is a built-in function

```
f(open ("this.txt",'r')
text = f.read()
print(text)
f.close()
```

we can also specify the number of characters in read() function: f.read(2)

↳ read first 2 characters

f.readline() → reads one line from the 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 good ")
f.close()
```

* with statement

The best way to open & close a file automatically is the with statement

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

OOP (object oriented programming)

This concept focuses on using reusable code.
Ls implement DRY princip
(do not repeat yourself)

* Class Employee :
# methods & Vasicables

↓

follows PascalCase

- EmployeeName → pascal case
- camelCase → isNumeric, ~~isFloat~~ isFloatorInt
- snake case → is_numeric

* Object

An Object is an instantiation of a class. When class is defined, a tempelate (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

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

* Class Attribute

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

ex. Class Employee:
Company = "Google" → [Specific to each class]

```
harry = Employee()          → object instantiation
harry.company
Employee.company = "Youtube"
                    ↓
                changing class attribute
```

* Instance Attributes

An attributes that belongs to the Instance (object). Assuming the class from the previous example:

```
Ashish.name = "Ashish"
Ashish.salary = "30K"
        ↳ adding instance attribute
```

> Note - Instance attributes take preference over class attributes during assignment & retrival

* 'self' parameter

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

```
harry.getsalary() → here self is harry
        ↳ equivalent to Employee.get salary(harry)
```

```
class Employee :
    company = "Google"
    def get_salary(self)
        print("salary is not these)
```

* 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")
```

→ decorator to make greet as a static method

* __init__() Constructor

~~initi~~ __init__() is a special method

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

It ~~was~~ takes self argument & can also take further argument.

for ex -

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

    def getsalary(self)
        ...

harry = Employee("Harry")
```

— object can be instantiated using constructor like this

* Inheritance

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

Syntax

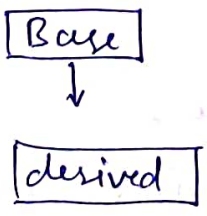
```
Class Employee :                    → base class
    # code
    ...

Class programer(Employee) :         → derived class
    # code
```

we can use methods & attributes of Employee in programmer object
Also we can overwrite or add new attributes & methods in programmer class

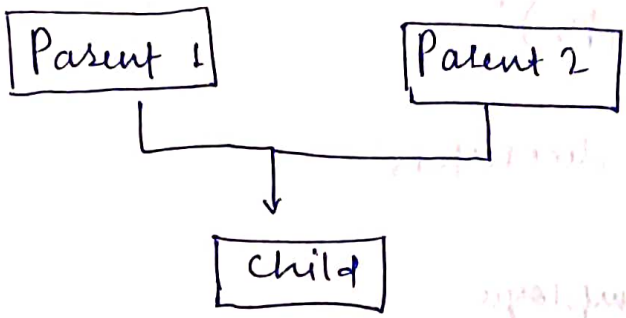
- Types

① - single inheritance
② - Multiple inheritance
③ - Multilevel inheritance

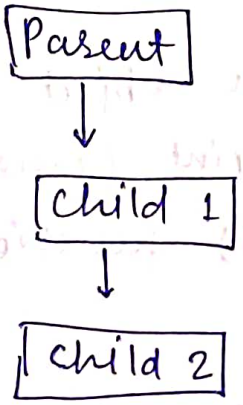
① - single inheritance



② - Multiple inheritance



③ Multilevel inheritance



- Super Methods

This method is used to access the methods of a super class in the derived class

super().__init__()

↳ calls constructor of the base class

super().mem-fun-name()

↳ calls the mem fun of base class

- Class methods

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

- @classmethod decorator is used to create a class method

Syntax

```
@classmethod
def (cls, p1, p2):
    ....
```

- @ property decorators

```
class employe
    @propesty
    def name(self):
        return self.ename
```

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

- @ getters and @ setters

The method name with @ propesty decorator is called getter method

We can difine function + @name.setter decorator like,

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

* Operator overloading

Operator in python can be overloaded using dender methods.

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

Operators in python can be overloaded using the following methods.

$p_1 + p_2 \rightarrow p_1.\_\_add\_\_(p_2)$

$p_1 - p_2 \rightarrow p_1.\_\_sub\_\_(p_2)$

$p_1 * p_2 \rightarrow p_1.\_\_mul\_\_(p_2)$

$p_1 / p_2 \rightarrow p_1.\_\_truediv\_\_(p_2)$

$p_1 // p_2 \rightarrow p_1.\_\_floordiv\_\_(p_2)$

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)`

Exception Handling in Python

- There are many built-in exceptions 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 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 the exceptions are handled here
```

* Raising Exceptions

~~Sometimes~~ 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:
    # code
```

```
except :
    # code
else :
    # code
```

→ This is executed only if try was successful

- **try with finally**

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

```
try :
    # code
except :
    # code
finally :
    # some code
```

→ execute regardless of error

* **If __name__ == '__main__' in python**

__name__ evaluates to the name of the module in python from where the program is ran

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 & returns it

```
for i, item in list:
    print(i, item)
```

↳ print 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]
```

Advanced python 2

Virtual Environment

* An environment which is same as the system interpretor but is isolated from other python environment on the system

Installation

- To use V.E, we write

  pip install virtualenv → install the package

- We create the environment using

  Virtualenv myprojectenv → create a new venv

- Next step is to activate it after creation we can now use this virtual environment as a separate python installation

* pip freeze command

pip freeze command 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 the output of pip freeze

We can distribute this file of other users & they can recreate the same environment using:

```
pip install -r .\requirements.txt
```

* lambda functions

- function created using an expression using lambda keyword

Syntax:

```
lambda arguments : expression
```

↳ can be used as a normal function

ex-

```
square = lambda x: x*x
square(6) → returns 36

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

- bin method (strings)

Create a strings 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(b_1, b_2 ...)

↳ arguments

syntax :

" {} is a good ".format("ashish", "boy") — ①

" {1} is a good {0}".format("ashish", "boy") — ②

output for ①

Ashis is a good boy

for ②

boy is a good ashish

- Map, filter & Reduce

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

Syntax:

map(function, input_list)

↑ can be lamda function

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

list(filter(function, l))

↳ can be lambda function

- Reduce applies a rolling computation to sequential pair

list is [1,2,3,4]

1 2 3 4

3 3 4 ⇒ sequential computation

6 4

10

W3 - Python

* x, y, z = "orange", "Banana", "cherry"

```
print(x)  ⟶ orange
print(y)  ⟶ Banana
print(z)  ⟶ cherry
```

* x = y = z = "orange"

```
print(x)
print(y)   ⟶ orange
print(z)
```

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

```
x = y = z = fruits
print(x)  ⟶ apple
print(y)  ⟶ banana
print(z)  ⟶ cherry
```

* a = "my name is"

```
if "my" in a:
    print("Yes")   ⟶ Yes
if "me" not in a:
    print("No")    ⟶ No
```

* a = "hello, WORLD!"

print(a.upper()) → capitalize every letter

print(a.lower()) → makes every letter in lower case

print(a.capitalize()) → makes 1st letter capital and

otherwise letters small

print(a.strip()) → removes all the white space from beginning or the end

* txt = "\x48\x65\x6c\x6c\x6f"

print(txt) ⟶ Hello (Hex value)

txt = "\110\145\154\154\157"

print(txt) ⟶ Hello (Octal value)

* x = 200

print(isinstance(x, int)) ⟶ True

* x = 15

y = 2

print(x // y) ⟶ 7

x = 15.2

y = 15.6

print(round(x)) ⟶ 15

print(round(y)) ⟶ 16

* import math

x = math.ceil(1.4)

y = math.floor(1.4)

print(x) → round a num to upward to its nearest int

print(y) ⟶ " " " " " downward " " " "

—*—*—*—

*
```
a = 3.14
print(math.ceil(a)) ⟶ 4
print(math.floor(a)) ⟶ 3
```

*
```
name = "bro code"
first-name = name[0:3].upper()
print(first-name) ⟶ BRO
```

*
```
def add(*args):
    sum = 0
    for i in args:
        sum += i
    return sum

Print(add(1,2,3)) ⟶ 6
```

*
```
def add(**kwargs):
    print(kwargs['ashish'] + " " + kwargs['kumar'])
add(first = 'ashish', last = 'kumar')
                    ⟶ ashish kumar

    for key, value in kwargs.items():
        print(value, end=" ")
    add(first = 'ashish', last = 'kumar')
                    ⟶ ashish kumar
```

* walrus operator
```
food = list()
while food := input("...") != "quit":
    foods.append(food)
```