PYTHON PROGRAMMING

Python is a simple & easy to understand language which feels like reading simple english. The pseudo code nature of python makes it easy to learn & understand for beginners.

Features of Python

→ Easy to understand
→ Free & open source
→ High level language
→ Works on Windows / Linux / 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.

Ch-1: Modules, Comments & Pip

Let's write our very first python program create a file called hello.py & paste the below code in it

print("Hello World") → print is a function (More later)

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

Module

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

Pip

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

→ pip install flask installs flask module

Types of Modules

There are 2 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 as, abc etc.

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

To append To concatenate a string use:

```
print(t1[i]," ", end=" ")
```

Date
Page

Using Python as Calculator

We can use python as 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 mask author name, date etc.

Types of Comments

There are 2 types of coments in Python
1) Single line coniment → Written using #

2) Multi line comments → Written using
"""
Comment
"""

Ch-1 Practice Set

Q) Write a program to print Twinkle Twinkle little star poem in python

Code:
```
print("""Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
like a diamond in the sky.""")
```

Output
Twinkle, twinkle ..... diamond in the sky

Note: When there are more than one line, you need to use triple quotes.

Q2) Install an external module & use it to perform an operation of your interest

Type in terminal
pip install playsound
Collecting playsound
Successfully installed playsound-1.2.2

Code: from playsound import playsound
playsound ('D:\\MyData\\Business\\
code playground\\Python course with
Notes\\1-
Chapter 1\\play.mp3')

Output
The mp3 sound gets played

Q3) Write a program to print the contents of a directory using OS module. Search online for the function which does that

Code: import os
print(os.listdir())

Output: You get there the list of programs you have made

['.vscode', '01_hello.py', '02_pr_01_twinkle.py
... etc.

Ch-2 - Variables & Datatypes

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

a = 30
b = "Harry"

→ Variables = Containers to store a value
→ Keywords = Reserved words 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 → True or False
5) None

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

a = 77 ⇒ Identifies a as class <int>
b = 88.44 ⇒ Identifies b as class <float>
name = "Harry" ⇒ Identifies name as class <Str>
d = None ⇒ Identifies d as class <NoneType>
~~Rules for defining a variable name~~
e = True ⇒ Identifies e as class <bool>

Rules for defining a variable name → Also applies to other identifiers

→ A variable name can contain alphabets, digits & underscores.
→ A variable name can only start with an alphabet and underscore.
→ A variable name can't start with a digit.
→ No white space is allowed to be used inside a variable name.
→ Variable names are case sensitive.

Examples of a few variable names are:- one, _one, two_P, 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 & 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) ⇒ type <str>

A number can be converted into a string & vice-versa (if possible)
There are many functions to convert one data type to 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 & 31 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

It is important to note that ⇒ If a is "34" even though the output of input is always entered 34 a string (even if the number is entered)

Ch:2 Practice set

Q1) Write a program to add two no.s.

Code:

```
a = 34
b = 5
print("The sum of a & b is", a+b)
```

Output:

The sum of a & b is 39

~~(Q2) The sum of~~

(Q2) Write a python program to find remainder when a no. is divided by 2.

Code:

```
a = 45
b = 15

print("The remainder when a is divided by b is", a % b)
```

Output:

The remainder when a is divided by b is 0

(Q3) Use comparison operators to find out whether a given variable a is greater than 'b' or not. (Take a = 34 & b = 80)

Soln.

Open REPL

```
>>> a = 34
>>> b = 80
>>> a > b
False
>>> b > a
True
```

Q4) Write a program to find average of 2 nos entered by the user

Code:
```
a = input("Enter first number: ")
b = input("Enter second number: ")
a = int(a)
~~avg = (a+b)/2~~
b = int(b)

avg = (a+b)/2
print("The average of a & b is", avg)
```

Output:
```
Enter first number: 6
Enter second number: 4

The average of a & b is 5.0
```

~~b = Harry's~~

Note:
b = "Harry's", Here double quotes need to be used, ∵ there are single quotes in string

Ch. 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 3 ways

1) Single quoted string → a = 'Harry'
2) Double quoted string → b = "Harry"
3) Triple quoted string → c = """Harry"""

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

Consider the foll. string.

name = "H A R R Y" 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. To slice a string, we use the foll. syntax

sl = name[end_ start : end_ end]

First index is included & last ~~is~~ isn't included.

sl[0:3] returns "Har" → characters from 0 to 3

sl[1:3] returns "ar" → characters from 1 to 3

Negative indices: Neg. indices can also be used as shown in the fig. above. -1 corresponds to the (length-1) index & -2 to length(-2)

Slicing with step value

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

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

Other advanced slicing techniques

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

String functions

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

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

```
len('harry')          # Returns 5
string.endswith()
```

2) endswith() function: This func tells whether the variable string ends with the terms specified in bracket. It returns 'x'

of string ends with 'x'.
string.count()

3) ~~count()~~: It counts the total number of occurences of any character.
string.capitalize()

4) ~~capitalize()~~: This function capitalizes the first character of a given string.
string.find(word)

5) ~~find()~~: This function finds a word & returns the index of the 1st occurence of that word in the string.

6) ~~replace(oldword, newword)~~ string.replace(oldword, newword): This function repeaces the oldword with a new word in the entire string.

Escape sequence characters

Sequence of characters after backslash '\' [Escape sequence characters]
Escape sequence characters comprises of more than one character but represents one character when used within the string.

E.g. \n(new line), \t(tab), \' (single quote), \\(backslash) etc.

P3

```
Program
greeting = "Good Morning"
name = "Harry"
c = greeting + name
print (c)
```

Output
Good Morning Harry

P2

```
name = "Harry"
print(name[1])
print (name[0])
```

Output
a
H

P3

```
story = "Harry is good.
\n He is very good"

print (story)
```

Output
Harry is good
He is very good

P4

```
story = "A is good. \n He\t
        is good"
print (story)
```

Output
A is good
He is good

Ch. 3 Practice Set

Q1 Write a Python programme to display a user entered name followed by Good afternoon using input() function

A:

```
name = input ("Enter your name")
print ("Good afternoon, " + name)
```

Output
Enter your name Aduart
Good afternoon, Aduart

92) Write a program to fill in a letter template given below with name & date

```
letter = '''Dear <|NAME|>,
        You are selected!
        <|DATE|>'''
```

Ans:

```
letter = '''Dear <|NAME|>

You are selected!

Date: <|DATE|>
'''

name = input("Enter your name \n")
date = input("Enter Date \n")

letter = letter.replace("<|NAME|>", name)
letter = letter.replace("<|DATE|>", date)
print(letter)
```

Output

```
Enter your name
Adwait

Enter date
06 November
```

Dear Jackee,
You are selected!

Date: 06 November

Q3) Write a program to detect double spaces in a string
Hint: If there is a double space you get the index of double space; If there isn't a double space you get -1.

Ans: st = "This is a string with double spaces"

```
double space = st.find("  ")
print double space
print(doublespaced)
```

Ans: 28

Q4) Replace double spaces in prob-3 with single spaces
A4) st = "This is a string with double spaces ---- Addy"

```
st = st.replace("--", "-")
print (st)
```

Output: This is a string with double spaces - Addy

Qs) Write a program to format the following letter using escape sequence characters

letter = "Dear Harry, This python course is nice. Thanks!"

Ans: letter = "Dear Harry, This Python course is nice! Thanks!"

print(letter)

formatted_letter = "Dear Harry, \n\t. This Python course is nice! \n Thanks!"

Ans:

Dear Harry, This Python course is nice! Thanks!

Dear Harry,
This Python course is nice!
Thanks!

Prog.

```
# Create a list using []
a = [1, 2, 4, 5 6, 6]

# Change the value of list using
a[0] = 98
print(a)
```

Out: [98, 2, 4, 56, 6]

Ch. 4 Lists & Tuples

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

friends = ['Appli', 'Akash', 'Rohan', 7, False]

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

List Indexing

A list can be index just like a string

L1 = [7, 9, 'harry']

L1[0] - 7 L1[70] - Error

L1[1] - 9

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

List Methods

Consider the foll. list:

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

1) list.sort() → updates the list to [1, 2, 7, 8, 15, 21]

2) list.reverse() → updates the list to [15, 21, 2, 7, 8, 1]

To avoid newline use: end=" "
e.g.
print("Statement", end=" ")

3) list.append(8) - adds 8 at the end of the list

4) list.insert(3,8) - This will add 8 at 3 index

5) list.pop(2) - It will delete the element at index 2 & return its value.

6) list.remove(21): It will remove 21 from the list.

Tuples in Python

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

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

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

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

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

Tuple methods:
Consider the following tuple,

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

To access items of a tuple you need to: just do

```
t1[i]
```

e.g.

```
print(t1[end])
```

Slicing: t1=(1,2,3,4,5)
print[t1[2:4]
o/p
(3,4,5)

Date
Page

1) a.count(1) - It will return the number of times 1 occurs in a

2) a.index(1) - It will return the index of the first occurence of 1 in a.

Prog:

```
# Create a list using []
a = [1, 2, 4, 56, 6]

# Change the value of list using
a[0] = 98
print(a)
```

Out: [98, 2, 4, 56, 6]

P5

```
l1=[1,8,7,2,21,15]
l1.sort()
print(l1)
```

Output

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

P7

```
l1=[1,8,7,21,15]
l1.append(45)
print(l1)
```

Output

[1,8,7,21,15,45]

P8

```
t1=() #Empty tuple
print(t1)
```

Output

()

```
t=(1,2,4,5,4,1,2,1,1)
print(t.count(1))
```

Output

4

P6

```
#Creating a tuple using()
t=(1,2,4,5)

#Printing the elements of
a tuple
print(t[0])

#Cannot update the values of
a tuple
t[0]=34
```

Output:

t[0]=34

Type Error: 'tuple' object does not support item assignment

```
t1=(1) #Wrong way to declare
a tuple with single
element
t1=(1,) #Tuple with single
element
print(t1)
```

Output

(1,)

Ch. 4 Practice set

Q1 Write a program to store 2 fruits in a list entered by the user

Ans:
```
f1 = input("Enter fruit Number 1:")
f1 = input("Enter fruit Numbers 2:")

myFruitList = [f1, f2, f3, f4, f5, f6, f6, f7]
print(myFruitList)
```

Output:
```
Enter fruit Number 1: Apple
Enter fruit Numbers 2: Banana

[Apple, Banana]
```

Q2) Write a program to accept marks of students and display them in a sorted manner

```
m1 = int(input("Enter marks for student 1: "))
m2 = int(input("Enter marks for student 2: "))
m3 = int(input("Enter marks for student 3: "))

myTest = [m1, m2, m3]
print(
myTest.sort()
print(myTest)
```

Output:

Enter marks for student 1 : 55
—11— 2 : 23
—11— 3 : 21

[21, 23, 55]

Q3 Check that a tuple cannot be changed in python.

Q4) Write a program to sum a list with 4 numbers

Ans:

```
a = [2, 4, 56, 7]
print(a[0] + a[1] + a[2] + a[3])
print(sum(a))
```

Output 69
69

Ch. 5 - Dictionary & Sets

Dictionary is a collection of key-value pairs

Syntax

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

a["key"]   # Prints value.

a["list"] # Prints [1,2,9]'''
```

Properties of Python Dictionaries.

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

Consider the following dictionary

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

1) items(): returns a list of (key, value) tuple

2) keys(): returns a list containing dictionary's keys.

3) update({"friend":"Sam"}): updates the dictionary with supplied key value pairs

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

Sets in Python

Set is a collection of non-repetitive elements

```
S=Set()        # No repetition allowed
S.add(1)
S.add(2)
# or set={1,2}
```

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 on 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

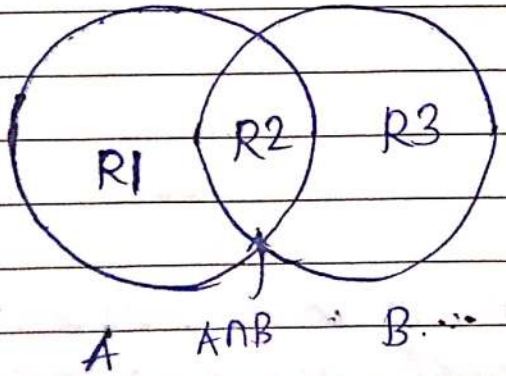
2) remove(8): Updates the set S & remove 8 from S.

3) pop(): Removes an arbitrary element from the set & returns the element removed

4) clear(): Empties the set S

5) union({8, 11}): Returns a new set with all items from both sets. #{1, 8, 2, 3, 11}

6) intersection({8, 11}): Returns a set which contains only items on both sets. #{8}



R2 = A ∩ B
R1 + R2 + R3 = A ∪ B
R1 + R3 = A Δ B
R1 = A − B
∴ R3 = B − A

P1

```
myDict = {
    "Fast": "In a Quick Manner",
    "Harry": "A Coder"
    }

print(myDict['Fast'])
print(myDict['Harry'])
```

Output:

In a Quick Manner
A Coder

P2

```
My Dict = {
    "Marks": [1, 2, 5]
    "Harry": "A Coder"
    "anotherdict": {'harry': 'Player'} }
MyDict['Marks'] = [45, 78]

print(MyDict['Marks'])
print(MyDict['anotherdict']['Harry'])
```

Output

[45, 78]

Player

Ch-6 Conditional Expressions

Sometimes we want

If else & elif in Python

If else & 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")
```

E.g.

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

Relational operators

Relational operators are used to evaluate conditions inside of statements. Some e.g. of relational operators are:

== → equals

>= → greater than or equal to

<= → less than or equal to

same as c

Logical Operators

In Python logical operators work on conditional statements

and → true if both operands are true else false

or → true if atleast one is true

not → inverts true to false & vice versa

Elif clause

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

```
if (condition 1):
    # code
elif (condition 2):
    # code
elif (condition 3):
    # code
```

```
else:
    # code
```

The above condition will stop once a condition in an ~~elif is~~ met. If or elif is met.

Imp. Notes:

→ There can be any number of elif statements

→ Last else is executed only if all the conditions inside elifs fail.

Ch. 7 Loops in Python

Sometimes we want to repeat a set of statements in our program. For instance: Printing 1000 times or asking correct password from user. Loops make it easy for a programmer to tell the computer, which set of instructions to repeat, & how!

Types of loops in Python

1) While loop 2) For loop

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 / false.

In while loops, the condition is checked first. If it evaluates to true, the body of the loop is executed, otherwise not!
If the loop is entered, the process of condition check & ~~condition~~ execution is continued until the condition becomes false.

E.g. i=0
while i<5:
⟷ print("Harry")
i = i+1

Leave space here very imp.
Above program will print harry 5 times.

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

For loop

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

The syntax of a for loop looks like this

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

Range function in Python

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)
```

- step size is usually not used with range()

An example demonstrating range() function

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

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.

E.g.

```
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.

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

The continue statement

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

E.g.

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

pass statement

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

E.g.

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

Ch: 8 Functions & Recursions

A function is a group of statements performing a specific task.
When a program gets bigger in size & its complexity grows, it gets difficult for a programmer to keep track of which piece of code is doing what!
A function can be reused by the programmer in a given program any number of times.

Syntax

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

The 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 parentheses as follows:

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

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

Types of functions in Python

1) Built-in functions # Already present in Python

2) User-defined functions # Defined by the user

E.g. of built in functions includes len(), print(), range(), etc.
The func1() function we defined is an e.g. of user-defined function

Functions with arguments

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

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

a = greet("Harry") # "Harry" is passed to greet in name

# a will now contain "Hello Harry"
```

Default Parameter Value

We can have a value as the default argument in a function.
If we specify name = "stranger" in the line

containing def, this value is used when no argument is passed

For e.g.

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

greet()          # Name will be 'stranger' in func^n body (default)
greet("Harry")   # Name will be "Harry" in func^n body (passed)
```

Recursion

Recursion is a function which calls itself
It is used to directly use a mathematical formula as a func^n. For e.g.

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

This func^n can be defined as follows:

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

factorial(3)
3 x factorial(2)
2 x factorial(1)

The programmer needs to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep calling itself.
Recursion is sometimes the most direct way to code an algorithm.

Ch. 9 - File I/O

The random access memory is volatile, & all its contents are lost once a program terminates.
In order to persist the data forever, we use files.
A file is data stored on a storage device.
A python program can talk to the file by reading content from it & writing content to it.

Comp. Program Written in Python. —write→ File —Read→

RAM = Volatile
HDD = Non-Volatile

Types of files

1) ~~Text files (.txt~~

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

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

Opening a file

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

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

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

Reading a file on Python

```
f = open("this.txt", "r") # Opens the file on r mode

text = f.read() # Read its content

print(text) # Print its content

f.closed
f.close() # Close the file
```

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

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

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

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

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 ~~wit~~ 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 & close the file automatically is the "with" statement.

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

There is no need to write f.close() as it is done automatically

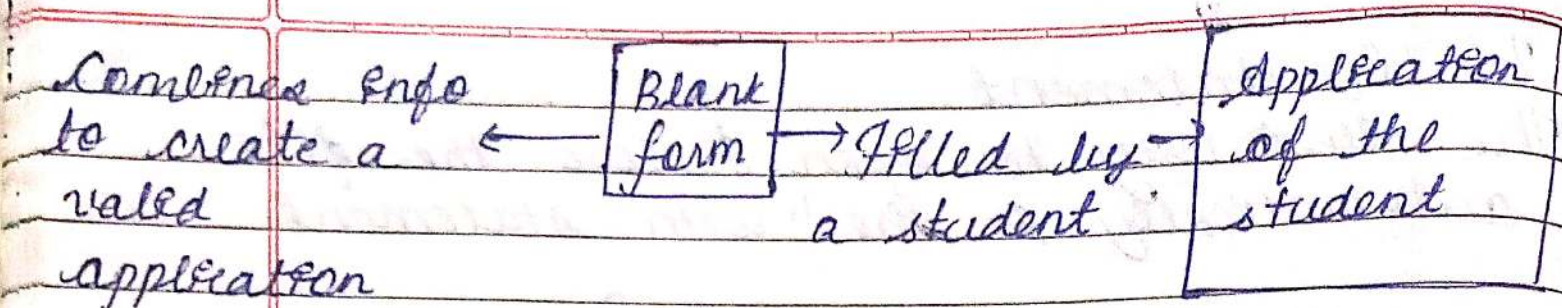
Ch.10 - Object Oriented Programming

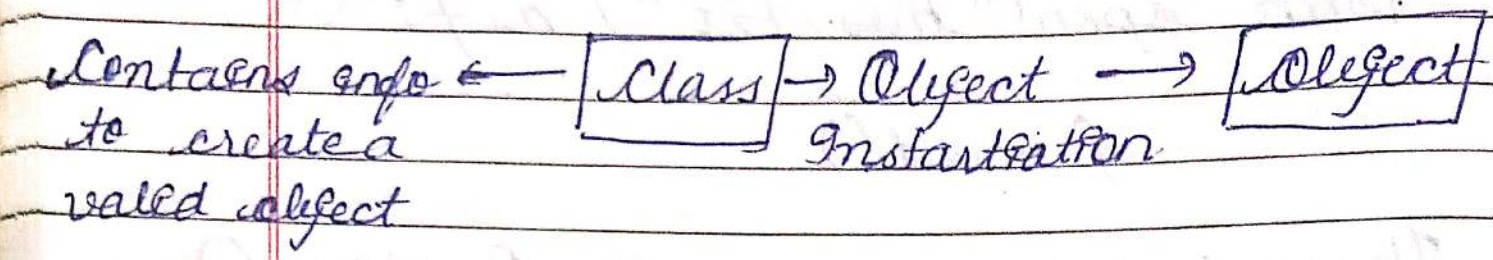
Solving a problem by creating objects is one of the most popular approaches in programming. This is called object oriented programming.

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

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
Abstraction & Encapsulation

Mood Modelling a problem in OOPs
We can 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.

```
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 = "50K" # adding instance attributes
```

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

harry.attribute 1:
1) Is attribute 1 present on the object?
2) Is attribute 1 present on the 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, & the above line of code is 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 greet as a static method

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

__init__() constructor

__init__() is a special method which runs 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 e.g.

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

    def getSalary(self):
        # Some code

harry = Employee("Harry") # Object can be instantiated using constructor like
```

Ch-11 Inheritance & more on OOPs

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

Syntax

```
class Employee:  #Base class
    #Code

class Programmer(Employee): #Derived
                            or child class
    #Code
```

We can use the methods & attributes of Employee in programmer object.
Also, we can overwrite or add new attributes & methods on the Programmer class.

Type of Inheritance

1) Single inheritance
2) Multiple inheritance
3) Multilevel inheritance

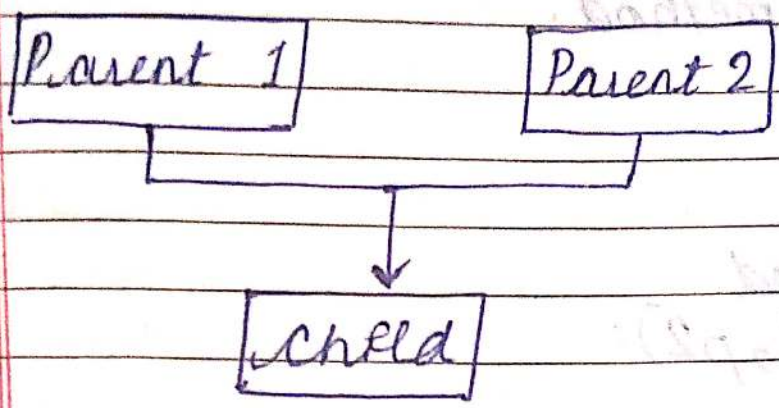
Single Inheritance:

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

Base → Derived

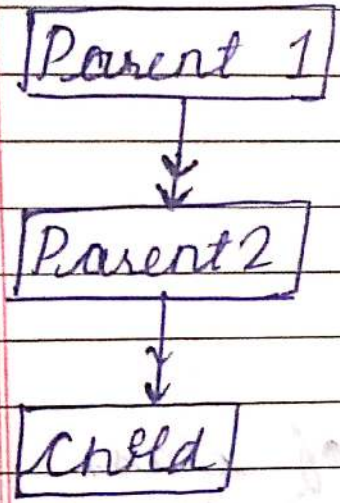
Multiple Inhesitance

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



Multilevel Inheritance

When a child class becomes a parent for another child class



Super() method

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

super().__init__() # calls constructor of the 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):
    #code
```

@property decorators

```
class Employee:
    @property

    def name(self):
        return self.ename
```

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

@getters & @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

p1 + p2 → p1.__add__(p2)

p1 - p2 → p1.__sub__(p2)

p1 * p2 → p1.__mul__(p2)

p1 / p2 → p1.__true_div__(p2)

p1 // p2 → p1.__floor_div__(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)