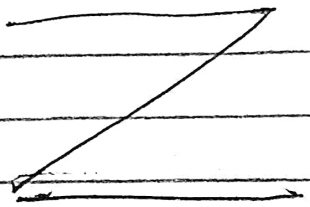
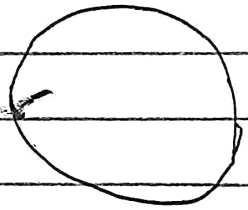
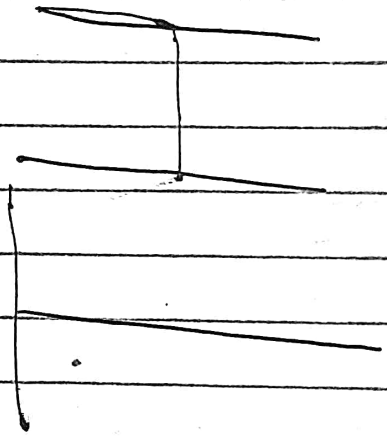
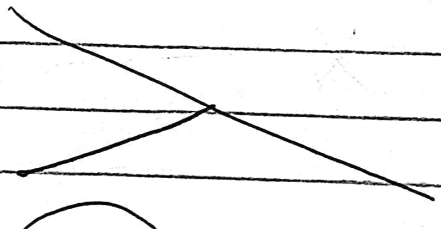








DATE
PAGE

Day 3

Modules and Pip

Pip is used to download modules.
Modules are precompiled code.

types of modules
- → Built in modules
- → external modules

Day 4

Print

input	output
Print ("Hello world")	Hello world
Print (69)	7
Print ("Hello", 6)	Hello 6
Print (17 * 13)	221

Day 5

Escape sequence characters (\) backslash character
- → To insert character that cannot be used in a string.
- → New line character : \n
- → More Eg: \', \"

Comments

This is python
↳ by using this compiler will ignore that line (Single line comment)
- → To make multiple line comments we can use ctrl + / after selecting the lines (only in modern IDEs)

DATE
PAGE

```
'''
Hi, I am chandan
Nice to meet you
'''
```

(OR)

```
"""
Hi, I am chandan
Nice to meet you
"""
```

These are ways to make multiline comments.

Seperator

input

```
Print("Hey", 6, 7)
Print("Hey", 6, 7, sep="~")
```

↳ separator

output

```
Hey 6 7
Hey~6~7
```

Space is the default seperator.

end

→ To specify what to print at the end of a line

input

```
Print("Hey", 6, 7, sep="~", end="00?")
Print("Chandan")
```

output

```
Hey~6~700?Chandan
```

\n is the default end.

Day 6

Variables and Datatypes

↳ containers ↳ Type of value in container

Eg:

```
a = 1
b = True
c = "Bye"
d = None
e = 1.34
```

datatypes

int

bool

(String) str

Nonetype

Float

Boolean data consists of True and false

DATE
PAGE

→ To know datatype of a variable type() is used

Eg: input | output

Print(type(a)) | <class 'int'>

No need to predefine datatypes of variables in python like C/C++.

Python doesn't have double datatype

Every thing in Python is an object.

Day 7

Operators

+, -, *, /, % are same as C and C++

→ / will give float value without typedef or using float values/variables.

→ // this is floor division this gives integer value only after division.

Eg: Print(17//3) | output: ~~5.66~~ 5, just int value

→ ** this exponential this used to find exponential value.

Eg: input | output

Print(5**3) | 125

this means 5^3 (Pow function in C)

Day 9

Typecasting

Conversion of datatype into other.

Eg:

input

a = "1"

b = "2"

Print(a+b)

output

12

~~Error~~ # this is bcz a and b are string, hence concatination is being done.

DATE
PAGE

types of type casting:
- Explicit typecasting
- Implicit typecasting

Explicit

Eg:
```
a = "1"
b = "2"
Print(int(a) + int(b))
```
output: 3

→ When we typecaste externally.

Implicit

→ when compiler automatically typecaste by itself. converts lower to higher datatype.

Eg:
```
a = 1.9
b = 8
Print(a+b)
```
output: 9.9
b is changed to float by compiler.

Day 10

User input

input() is used as scanf or Cin in python

Eg:
```
a = input()
Print(a)
```
output: Chandan / Chandan

```
a = input("Enter name:")
Print(a)
```
Enter name : Chandan
Chandan

input function takes every thing as a string input.

Eg:
```
a = input()
b = input()
Print(a+b)
```
output: 36 / 5 / 365 # concatination

#=# Python excutes a code line by line until an error comes.

DATE
PAGE

-) # We have to typecast that at the time of input or print to deal with the string input problem.

Eg:

```
input
a = int(input())
b = int(input())
Print(a+b)
   (OR)
a = input()
b = input()
Print(int(a)+int(b))
```

output

```
36
5
41
```

Unlike scanf and cin it takes spaces also as input like getline.

Day 11

Strings

-> It is ~~similar~~ like an ~~to~~ array of characters

-> For multi line string ''' or """ like multicomments are used

Eg:

```
a = """ Hi, my name
        is chandan """
Print(a)
```

output

```
Hi, my name
        is chandan
```

```
input
a = "Chandan"
Print(a[0])
Print(a[1])
Print(a[7])
```

output

```
C
h
Index Error
```

Printing characters one by one using for loop:

```
for character in a
    Print(character)
```

indentation is imp in python.

output

```
C
h
a
n
d
a
n
```

DATE
PAGE

Day 12

String slicing

→ Eg:

```
input
names = "Chandan, Shikansh"
Print (names[0:7])
Print(len(names))
# len() is like strlen() in C
a = "Mango"
Print(a[0:4])
Print(a[:4])  # Python automatically puts 0 at initial pos.
Print(a[1:4])
Print(a[:])  # puts len() at final pos.
Print(a[0:-3])
Print(a[0:len(a)-3])  # Python takes them as same.
Print(a[-1:-3])  # no output as its 4 to 2
Print(a[-3:-1])
```

```
output
Chandan
15
Mang
Mang
ang
Mango
Ma
Ma

ng
```

Day 13

String methods

strings are immutable.

```
input
a = Chandan
Print (a)
Print (a.upper())
# makes a copy of 'a' in upper case
Print (a.lower())
# makes a copy of 'a' in lower case
```

```
output
Chandan
CHANDAN
chandan
```

DATE
PAGE

```
a = !! Chandan !!!
Print(a.rstrip("!"))
# removes trailing characters
Print(a.replace("Chandan","Jaadu"))
# replaces 1st characters to 2nd.
.
b = "Nishchay Jaadu Vanisha"
Print(b.split())
# splits array into List
c = hi My naMe is ChanDan
Print(c.capitalize())
# Capitalizes first letter and
changes other into lower case.
d = "Welcome to the console!!!"
Print(d.center(50))
# aligns to center as per parameters
Print(len(d))
Print(len(d.center))
Print(d.count("co")) # counts chars
Print(d.endswith("!!!"))
# checks if string ends with given
  string chars and give bool return
Print(d.endswith("to",4,10))
  # checks after slicing
Print(d.find("to"))
# gives initial index of string
  if it is present else returns -1
Print(d.index("too"))
#Same as find but gives errror when
  substring not found.
e = "Vanisha9"
Print(e.isalnum())
# Checks if string only has A-Z, 0-9, a-z
  and returns in bool value.
```

```
!! Chandan
!! Jaadu !!!

['Nishchay', 'Jaadu', 'Vanisha']
Hi my name is chandan
    Welcome to the centre!!!
25
50
2
True
True
8
Value errror: Substring not found.

True
```

```
Print(c.isalpha())                                  False
# Same as isalnum() but doesn't                     False
  include 0-9.                                      False
Print(c.islower())                                  False
# checks for lowercase                              True
f = "Aditya \n"                                     False
Print(f.isprintable())                              True
# checks for printable char                         aDITYA  (one line space)
Print(f.isspace())                                  Hi My Name Is Deepak
# checks for space or tab
Print(f.istitle())
# checks if every first char of
  a word is capitalized.
Print(f.isupper())
# same as is lower() but for
  uppercase.
Print(f.startswith("Ad"))
# same as endswith.
Print(f.swapcase())
# changes upper to lower and vice-a-versa

g = "Hi my name is deepak"
Print(g.title())
# converts first char of every
  word to upper case and
  others to lower case.
```

Day - 14

If - Else conditional statement

→ Same as C but indentation is imp

```
if( ):
else( ):
```

DATE
PAGE

→ elif ( ): is ~~else~~ like else if ~~in~~ in C.

Day 15

time

```
import time
t = int(~~strftime~~ time.strftime("%H:%M:%S"))
Print (t)
```

output

23:10:27

Day 16

Match case

→ Same as switch case in C/c++ but break is not needed after case

Syntax:

```
Match a:
    ~~case~~ case 1:
        ——— body
    case 2:
        ——— body
    (default case) case _:
        ——— body
```

Day 17

For loops

→ It can iterate over a sequence of iterable objects like string. (Day 11 eg:)

range()

Syntax:

```
For i in range(start no, stop no, step no.)   # if only one number is given by default start will be 0 and that no will be considered as stop.
    Print (i)                                 # commas are used unlike C/c++.
```

→ It will print from start to stop - 1

step no. is the no. added to start, ~~by~~ its ~~by~~ default value is 1.

DATE
PAGE

Day 18

While loop

it same as in C/C++

Syntax:

```
i = 0
while(i<=10):
    Print(i)
    i = i+1    # i++ doesn't work here
```

→ We can use else statement with while also when cursor will not go in while loop else will be executed.

Day 19

Break, continue

→ Break is used to come out of a loop ~~giving~~ same as in C/C++.

→ Continue is used to skip a iteration

Eg:
```
for i in range(11):
    if(i>4):
        break
    Print(i)
```
output
0
1
2
3
4

Eg:
```
for i in range(5):
    if(i==4)
        Continue
    Print(i)
```
output
0
1
2
3
5

Day 20, 21

Functions → user defined
Same as in C/C++.
Syntax:

```
def Functionname (arguments):
    function-Content
```

→ We can also use 'Pass' to write a function later.

Eg:

```
def sum (a=3, b):
    Print (a+b)
sum (16, 4)
def Mul (a,b):
    Pass
```

default argument

output

```
20
```

Day 22

Lists

→ It is like an array in C/C++, it is a data type.

Eg:

```
marks = [3,6,9]
Print (marks)
Print (type (marks))
Print (marks [1]) # List index just like in arrays.
```

output

```
[3, 6, 9]
<class 'list'>
6
```

→ List is mutable (~~cross~~ can be changed)
→ Python allows you to store data of different datatypes in one list.
Eg l = [1, 2, 3.58, True, "Bye"]

→ negative index (just like in Day 12)

Eg: Print(marks[-2])

* this is equal to Print(marks[len(marks)-2])

So for above example it will be Print(marks[4])

output:

6

→ To find if an element exists in list or not:

Eg:

```
Input
if 7 in marks:
    Print("Yes")
else:
    Print("No")
```

output

No

```
if "6" in marks:
    Print("Yes")
else:
    Print("No")
```

No

bcz 6 exists as an int in list.

→ We can also use index like in range()

Eg:

```
Print(marks[0:3])
l = [1, 9, 10, 4, "Bye"]
Print(l[0:5:2])
```

[3, 6, 9]

[1, 10, 'Bye']

List comprehension

→ used for creating new lists from other iterables.

Eg:

Input

```
P = [i for i in range(4)]
Print(P)
Q = [i*i for i in range(10) if i%2==0]
Print(Q)
```

output

[0, 1, 2, 3]

[0, 4, 16, 36, 64]

DATE
PAGE

Day 23

List methods

Input	output

l = [11, 45, 1, 2, 11, 6]
Print (l)
l.append(7) # it adds a new element at end
Print(l)
	[11, 45, 1, 2, 4, 6]
	[11, 45, 1, 2, 4, 6, 7]

→ sort() : it sorts the list in ascending order

→ sort(reverse = True) : sorts in descending order.

→ reverse() : it reverses the list

→ index () : gives the index of the element.

→ count() : counts the no. of time an element is present in the list.

→ copy() : In python if you write

```
a = [2, 3, 6]
b = a
```

this doesn't makes a copy of a into b, but b becomes a reference to a so if you will change b, a will be changed. So, to make a copy.

```
b = a.copy()
```

→ insert (index, element) : it works same as in C++, SIL, used to insert an element at a particular index.

→ extend() : it extends the list and adds elements of other list at the end.

Eg: b.extend(a)

→ but this will change the list b, so we can concatinate also by :-

```
c = a + b   # this will concatinate b in a
c = b + a   # this will concatinate a in b
```

→ Pop() : used to remove an element.

Day 24 , 25

~~Tuple~~ Tuples

→ Tuple is same as list but is immutable (can't be changed)

Eg:

Input

```
tup = (1, 2, 76)
Print (type(tup), tup)

tup[0] = 80
```

output

```
<class 'tuple'> (1, 2, 76)

type error:
```

→ We can check element, do slicing, negative indexing, printing all can be done same as in list.

→ We can change tuple into list to edit and then back to tuple

Day 28

f strings

→ helps you to place variables in a string conveniently

old method

Syntax:

Eg:

```
variable = "My name is {} and i am from {}"
name = "chandan"
country = "India"
Print ( Variable.format(name, country))
```

output

My name is chandan and i am from India.

New method

```
Price = 897.365489
text = f" for only {Price :.2F} rupees"
       ↳ fstring
Print (text)
```

↳ This is to take 2 digits after decimal in float

DATE
PAGE

Output
For only 897.36 rupees.

→ To get rows { } in f strings we use another {}.

Input

```
age = 18
a = f"my age is {age}"
a1 = f"my age is {{age}}"
Print (a)
Print (a1)
```

output

```
my age is 18
my age is {age}
```

Day 29

Doc strings

→ these are string literals that appear right after the definations of a function, method, class or module.
→ these are like comments but are ~~given~~ not completely ignored instead they are stored in __doc__.
→ used right after the defination

Syntax:

Eg:

```
def square(n):
    """Takes in a number n, returns its square"""
    Print(n**2)          # also works with '''
square(5)
Print(square.__doc__)
```

} It will only work if docstring is ~~straight~~ after function ~~definition~~ declaration or else will be considered as comments.

output

```
5
25
Takes in a number n, returns its square.
```

DATE
PAGE

Pep 8 (Python enhancement proposal)

→ It is a document which primarily focused to improve the readability and consistancy of python code

→ The Zen of Python (easter egg)
It is a poem by Tim peters on python
It is printed on writting : Import this
Read at the time of interview.

Day 30

Recursion

→ When in a function we call the exact same function, its called recursion.
→ It is same as in C/C++.

Day 31

Sets

→ It is same as a list but value can't be repeated much like sets in maths.
→ It is unordered collection of data items, enclosed in {}
→ Sets are immutable (unchangable)
To make a empty set we cant use empty {} as it is for empty dictionary so we use set()
Eg: a = set()
→ it is an empty set.

Day 32

Set methods

→ union() :- it is same as in maths writes two sets
→ update() :- it is same as union but stores in calling set.

→ intersection() :- same as in maths
→ intersection_update() :- same as update but for intersection.
→ Symmetric_difference() :- (A ∪ B) - (A ∩ B)
→ Symmetric_difference_update() :- same.
→ difference() :- A - B
→ difference_update() :- same
→ isdisjoint() :- checks if sets are disjoint
→ issuperset() :- checks if calling set is superset of argument set.
→ issubset() :- checks if calling set is subset of argument set.
→ add() :- same as append() in list.
→ remove() :- it removes the element but gives error if the element is not present in set.
→ discard() :- same as remove but doesn't gives error.
→ pop() :- removes last element but as sets are unordered it becomes random.
→ del :- deletes the whole set
Eg: del names
→ clear() :- deletes all the elements of the set

Day 33

Dictionaries

→ These are ordered collection of data items. They are same as maps in STL in C++.
→ These were unordered but after Python 3.7 it became ordered collection.

Some basic functions:-

→ get() :- use to get values for a given key.
it is similar to index method but that give error if the key is not present while get() returns 'None'.
Eg: Print(dic[92])
Print(dic.get(92))

DATE
PAGE

→ keys() :- To get all keys
→ values() :- To get all values
→ items() :- To get keys and values in pair.
We can use for loops using these functions to get key, values and both separately like array.

Day 34
Dictionary methods

→ update() :- same as sets
→ clear() :- same as sets
→ pop() :- removes item by given key.
→ popitem() :- removes last item.
→ del :- deletes whole dictionary
can also be used like pop()
Eg:- del ep 3[222]
this is same as ep 3.pop(222)

Day 35
else with for loop

→ It is same as else with while loop
else is executed after the loop is ended.
NOTE: else is not executed on break but is executed on continue after the end of loop.

Day 36
(Essay) Exceptional handling

→ same as exception handling in C++ but more like error handling as there is no 'Throw' keyword. exeption is thrown when an error occurs, except is used like catch, we can use different

multiple excepts for different types of errors.

Syntax:

```
try:
    "code having threat of error"

except:
    "code to handle the error"
```

Day 37
Finally keyword

→ It is also a part of exception handling. It is always executed irrespective of try ... except ~~if~~ block outcomes. generally used for doing the concluding ~~tasks~~ like closing file resources or ending a program with a helpful message.

Syntax:

```
try:
    # code that could get error
Except:
    # to solve errors
Finally:
    # will be executed in any situation.
```

→ more likely used in a function, so that ~~code~~ code in finally block gets executed even after return.

Day 38
Raising custom errors

→ This is the real "Exception" handling

→ ~~Raise~~ raise keyword is like throw in C++ we can use it to throw an exception (error) by ourself

DATE
PAGE

Syntax:-

if (condition):

raise type of error ("statement to be ~~displayed~~")

→ We can give predefined error types or can create our own error type class.

Day 41

Shorthand if else statements

→ one liner for if else statements like in C/C++.

→ Increases the readability in simple codes.

→ Reduces the readability in complex codes.

→ hence Recommended for less complex codes only.

Syntax:-

if — elif

expression1 if condition1 else expression2 if condition2 else expression3

else.

Eg: ~~a = 20~~

~~Print (a)~~

output:-

35

Input:-

Eg:- a = 20

b = 35

Print (a) if a>b else Print (b) if a<b else Print ("a = b")

DATE
PAGE

Day 42

Enumerate function

→ It is a built-in function in python that allows you to loop over a sequence (list, tuple, string) and get the index and value of each element in the sequence at the same time.

→ enumerate function returns a tuple but by writing two variables in for loop, tuple is unpacked into index and value.

→ We can also give starting value of index by using start = value in enumerate.

Eg:

```
for m,n in enumerate (marks, start=1) :
```

Day 43

Virtual Environment

→ It is a tool used to isolate specific python environments on a single machine, allowing you to work on multiple projects with different dependencies and packages without conflict.

→ mainly used when a group of people is working on different systems having different versions of packages but working for same project.

→ To create a virtual environment:

```
python -m venv myenv
```

↳ can be any name.

→ To activate it in windows

```
myenv\Scripts\activate.bat
```

↳ ps1 → for Powershell

→ once virtual environment is activated, any packages you install will be installed in the virtual environment

→ To deactivate:
deactivate

Requirement.txt file

→ It makes a txt file of version of all packages used, automatically. Hence making easy for other user to install same packages

→ pip freeze :- shows all packages and version

→ pip freeze > requirements.txt :-
this will make Requirements.txt file containing the info of packages.

→ To directly install all packages from requirements.txt we can use :-
pip install -r requirements.txt

Day 44
Import

→ It is the process of loading a module into the current script, allows you to use the functions and variables defined in the module in your current script as well as module on which imported module may depend upon

Eg: Input:
import math # Now we can use function as math.function()

→ You can also import certain function from module.

Eg: Input:
```
from math import sqrt, pi
result = sqrt(9) * pi
print(result)
```
output
9.424777968...

when importing certain functions we can directly use function that module name or object.

DATE
PAGE

→ We can also use "from math import *" to import everything in math to script but it is not recommended as it can lead to confusion when using multiple modules.

→ we can use as keyword to use a short form (object) for the module of ~~or~~ specific function.

Eg :- Input :-

```
import math as m
print(m.sqrt(9))
```

output: 3

OR

```
from math import sqrt as s
print(s(9))
```

output: 3

→ dir function :- It used to ~~use~~ view the names of all functions in a module.

Eg :- ~~print~~ print(~~dir~~ dir(math))

→ We can import classes, functions and variables from one program to another.

Day 45

If __name__ == "__main__"

→ This idiom is a common pattern used in python scripts to determine whether the script is being run directly or being imported as a module into another script.

→ For more detail check from program in whatsapp.

DATE
PAGE

Day 46
OS Module

→ It is a module that helps us to do changes in our operating system like creating, deleting, renaming files and much more

→ Some basic functions are :-
- os.mkdir()
- os.chdir()
- os.listdir()
- os.getcwd()
- os.rename()

→ Watch video again for detail/better understanding

Day 48
Local vs Global variables

→ A local variable is a variable defined within a function and is only accessible within that function. it is created when function is called and destroyed when function returns

→ A global variable is a variable defined outside of a function and is accesible from within any function in your code.

global keyword

→ It is used to specify that the variable is in global scope with in the function

→ helps to change global values in from function.

→ It is recommended to avoid using it.

DATE
PAGE

Day 49

File handeling

→ Opening a file :-

Syntax :

f = open (" file name " , " mode ")

Note : " " can be used as ' ' too in python.

→ Modes in file :

1) Read (r) :- This mode opens the file for reading only and gives an error if the file does not exists. This is the default mode if no parameter is passed.

2) Write (w) :- The mode opens the file for writting only and creates a new file if file does not exists.

3) append (a) :- Same as write mode but write mode overwrites the file when new data is entered while append mode appends the data after previous one.

4) Create (x) :- This mode creates a file and gives an error if file already exists.

5) text (t) :- t mode is used to handle text files . it is default mode when reading or writting . (Basically used if reading a binary file or we can say it is useless as it is default)

6) binary (b) :- used to handle binary files (images, pdfs etc) used as rb (reading as binary) or wb (writting as binary)

→ Reading a file :- It is done after opening a file in read mode.

Eg :- Print (f . read ())

→ Writting a file :- used in write mode.

Eg :- F . write (" text ")

→ Closing a file :
It is not mandatory but is a good practice to avoid problems.
Eg : f.close()

With statement

→ This statement automatically closes the file after your are done (indentation is ended for it).

Eg : with open("filename.txt", 'a') as f :
f.write("text")

Day 50

readline method and readlines method

readline method

→ reads a single line from the file. If we want to read multiple lines, we can use a loop.

→ readlines method reads all the lines of the file and returns them as a list of strings.

writelines method

→ It seperates an iterable like list, dictionary, etc. into lines and write them into file.
→ It does not adds newlines automatically, so we have to use newline character after every line or can use write() with loop.

DATE
PAGE

Day 51

seek() and tell() functions

→ These are used to work with file objects and their positions within a file. these are part of built in io module, which provides a consistent interface for reading and writting to various file-like objects, such as files, pipes and in-memory buffers.

→ seek() function moves the cursor in a file forward or backward by the given arguments.

→ tell() function - tells us the current position of the cursor.

truncate()

→ In write or append mode we can use truncate() function to trim the file to specific size.

Day 52

Lambda function

→ It is a small anonymous function without a name. It is defined using lambda keyword.

Syntax :-

lambda arguments : expression

→ used in situations where small function is required for a short period of time. They are commonly used as arguments for higher order functions (function as a argument to function), such as map, filter and reduce.

DATE
PAGE

Day 53

Map, filter and Reduce

→ These are built in function that allow you to apply a function to a sequence of elements and return a new sequence.

→ These are known as higher order functions as they take functions as arguments.

→ **map :-**

⇒ The map function applies a function to each element in a sequence and returns a new sequence containing the transformed elements.

Syntax :-

map(function, iterable)

→ Note :- it gives a map object so to get it as iterable we have to type caste it.

→ **Filter :-**

→ The Filter function filters a sequence based on boolean return of given function (predicate). and returns a new sequence containing only the elements that meet the condition and returns true.

Syntax :-

filter(predicate, argument)

→ same as map it gives filter object so we have to typecast.

Reduce :-

→ The reduce function applies a function to a sequence and returns a single value.

DATE
PAGE

Syntax :-
reduce (function, iterable)

→ Note :- We have to import reduce from functools before using it.

eg :→ It can be use to get sum of all elements in a list.

Day 54

'is' vs '=='

→ Both 'is' and '==' are comparision operators but
→ 'is' compares the exact location
→ '==' compares the value

Eg :

Input

```
a=[4,2,8]
b=[4,2,8]
Print (a is b)
Print (a==b)
```

Output

```
False  # it will be true if a=b=[4,2,8]
True
```

but if a=4 b=4 : which is int, which is an immutable object python points both variable to same value to store memory.

Here ends Procedural Programming

OOPS - Object oriented programing (System)

starts from here.

DATE
PAGE

Day 57

Classes and objects

→ Class is a blueprint for creating objects, providing initial values for variables or attributes (data) and implication of member functions or methods (behaviour).

→ This is same as class in C++.

Eg: class student:

name = "student_name"

age = 10

obj = student() # making object for class

Self keyword

→ It is not a keyword it is just a convention we can use any name

→ Self means that object for which the method is called.

Eg: class student:

name = "student_name"

age = 10

def info(self):

Print(f"{self.name} is of {self.age} years old")

a = student()

a.name = "Chandan"

a.age = 19

a.info()

output:

Chandan is of 19 years old.

Day 58

Constructors

DATE
PAGE

→ Constructor is almost same as in C++ but in C++ we used to make it by name of class but here we use __init__(). This is called dunder method

→ It is automatically called when an object is made just like in C++.

→ In python constructor always returns None.

~~init~~ __init__()

→ It is a reserved function used to make constructors.

→ Types of constructors :-

- parameterized - when constructor accepts arguments along with self.
- default - when constructor just takes self as only argument.

Day 59

Decorators

→ It is a function that takes another function as an argument and returns a new function that modifies the behavior of the original function.

Syntax :-

```
@ decorator_function
def function_used():
    (content)
function_used()
```

(OR)

```
def function_used():
    (content)
decoratorfunction(function_used)()
```

this is shorthand of

```
def function_used():
    (content)
function_used = decorator_function(function_used)
function_used()
```

For Function with arguments we use *args, **kwargs in decorator function

DATE
PAGE

Day 60

Getters

→ These are methods that are used to access the values of an object's properties. They are used to retrieve the value of a specific property, and are typically defined using the @property decorator.

Setters

→ Getters do not take any parameter and are cannot set the value through getter method. For that we need setter method which can be added by decorating method with @property_name.setter

→ For detail watch video again.

Day 61
Inheritance

→ The concept of inheritance is same as in C++.

Syntax:

```
class Base:
    # content
class child(Base):    # this is inherited from Base class
    # content
```

Types:

→ Single
→ Multiple
→ Multilevel
→ Hierarchical
→ Hybrid

DATE
PAGE

Day 62

Access modifiers

→ In python there is no strict concept of public, protected and private like in C++. Instead here in python, there is concept of Name mangling.

→ Every variable and method in python is public by default.

→ there is no protected access modifier but classically we use '_' (Single underscored) in preffix as a naming convention. It actually does not restrict anything and this name convention can be used as we want.

→ there is no strict concept of private in python, but we can use '__' (Double underscored) in preffix to make variable or methods private, this is known as "weak internal use indicator".

→ But we can still access those variables and method by name mangling.

→ name mangling is a technique in which we use '_' 'class name' before private variables and methods to access them.

Eg :-
```
class employee:
    def __init__(self):
        self.__name = "Chandan"
a = employee()
# print(a.__name) # this will give error
print(a._employee__name) # mangling
```

DATE
PAGE

Day 63

Random module

→ Similar to C and C++.

→ Used in games and have many other uses when you want random choice from computer.

→ import random

→ Imp methods :-

- randint() - Returns a random no. from the given rang
- choice() - Returns a random element from given seque
- random() - Returns a random float no. b/w 0 to 1.
- randrange() - same as randint but final argument no. is not included in range.
- shuffle() - Takes a sequence and returns it in random order

Day 65

Static method

→ These are functions that belong to a class rather than an instance (object) of a class.

→ They are independent of class variables and are not associated to object so, 'self' argument is not required and function can be called by object or classname.

→ This is not a replacement for normal functions. this is if we want that user of that class can use that function also.

Syntax :

```
@staticmethod
def function():
    # content
```

DATE
PAGE

Day 66

Instance variable vs Class variables

→ Instance variables are defined at instance level and are unique to each object. They are defined inside the __init__ method(constructor) they are associated to objects and can be changed using objects only. #Value changes by objects even for class variable are considered as instance variables.

→ Class variables are made outside of any method. these are variable associated to whole class instead of an object, hence stays same for every object until changed by using classname (changes for every object after this) or changed by object (as it is considered as instance variable and they are prioritized more than class variables).
Eg: Number of employees in a company.

Why we use self in non argument function?
obj.functionname() and classname.functionname(obj) are same which shows one argument is given."
If we don't use self it shows error that one argument is given when it takes 0 arguments (TypeError)
this is because obj is considered as argument.

→ If we use class variable in constructor (__init__) we use classname.variablename instead of self.variablename.

DATE
PAGE

Day 69

Class methods

→ It is a method which is associated to class not to an instance like class variables and are generally used to operate on class variables.

Syntax:

```
@ classmethod                    → this will take class name because of
def changeCompany(cls, newcompany):   classmethod decorator
    cls.company = newcompany
```

By default first argument takes instance(object) as given argument until @ classmethod is used.

→ now we can use classname.function(newdata)

Day 70

Class methods as alternate constructors

→ Some times when data is not given seperately in intented datatype like name and salary(str, int) like if it is given as st1 = "chandan-120000" a string then instead of using split string method in main code we can use it in a classmethod and then use that classmethod as constructor.

For eg:

```
@ classmethod
def fromstr(cls, string):
    return cls(string.split('-')[0], int(string.split('-')[1]))

st1 = "chandan-120000"
e1 = employee.fromstr(st1)
print(e1.name)
print(e1.salary)
```

output:

```
chandan
120000
```

DATE
PAGE

Day 71
dir, --dict-- and help method

→ They make it easy for us to understand how classes resolve various function and executes code.
→ dict-- is an attribute, dir() and help() are methods.

dir()

→ It returns a list of all attributes and methods (including dunder methods) available for an object.
Eg: x = [1, 2, 3]
print (dir(x))

--dict--

→ This attribute returns a dictionary representation of an object's attributes. useful for introspection (examine)
Eg: class Student:
def __init__(self, name, enroll):
self.name = name
self.enroll = enroll
s = Student("chandan", 92)
print (s.__dict__)

output
{'name': 'chandan', 'enroll': 92}

help()

→ It is used to get help documentation for an object, including a discription of its attributes and methods.
Eg: print (help(student))
print (help(str))

DATE
PAGE

Day 72
Super keyword

→ It is used to refer to parent class, especially useful when a class is inherited from multiple parent class.

→ When a function overrides, super is used to use parent class's function in child class.

Syntax

```
super().func_name()
```

Day 73
Magic/Dunder methods

→ Magic methods, also known as "dunders" from the double underscores surrounding their name, are powerful tools that allow you to customize the behaviour of the classes.

→ __init__ is also a dunder method.

→ Most commonly used dunder methods are:

- __len__
- __str__ } both are similar and used to provide
- __repr__ string representation of an object

__str__ is informal (user-friendly) representation
__repr__ is formal (developer-friendly) representation.
if __str__ is not made but __repr__ is, then str() will also call __repr__.
We have to return string instead of using print.
We can also call __str__ by simply printing object.

- __call__ : it is used to make instances into callable methods.

Syntax: e = employee()
e()

DATE
PAGE

Day 74
Method overriding

→ It is a feature in OOPs that allows you to redefine a method in a derived class.
→ It is similer ~~to~~ as in C++.
→ The method of same name is created in child class as in parent class and then when you create an instance of derived class and call that method, the method in derived class is called.
→ super keyword is used for it.

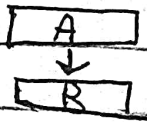
Day 77
operator overloading

→ It is same concept as in C++ but ~~so it~~ with a different way to make operator overloading method. here in python, dunder methods are used to do operator overloading like __add__, __sub__ etc.

→ operator overloading is a feature that allows developers to redefine the behaviour of mathematical and comparision operators for custom data types such as complex, vector etc.

Day 78, 79, 80, 81
Types of inheritance : ~~to~~ Same as C++

78 → Single inheritance

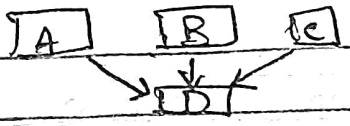


→ It is a type of inheritance where a class inherits ~~from~~ from a single parent class.

Syntax : ~~class~~ class child_class(Parent_class):
Pass

DATE
PAGE

79 → Multiple inheritance



→ It is a feature that allows a class to inherit multiple classes.

Syntax :-

```
class child-class (Parent1, Parent2, Parent3 ):
```

→ If there is method of same name in different Parent classes and is called by object of child class, (Parent1) first inherited class's method will be called.

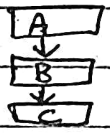
mro method (method resolution order)

Syntax :-

```
Print(child-class . mro( ))
```

→ It tells the order of classes in which method will executed if of same name.

80 → Multilevel inheritance



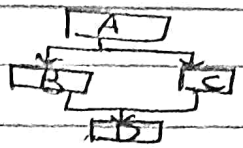
→ In this type of inheritance class inherits from another derived class, which allows you to build hierarchy of classes where on class builds upon other.

Eg :-

```
class Parent:
    Pass
class child1(Parent):
    Pass
class child2(child1):
    Pass
```

DATE
PAGE

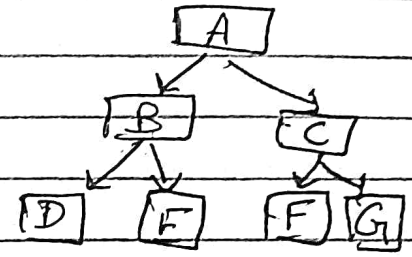
81 ⇒ Hybrid inheritance



→ Usually It is a combination of multiple inheritance and single inheritance. (combination of different inheritance)

→ here we can also see multilevel inheritance and hierarchical inheritance also.

Hierarchical inheritance



→ It is a type of inheritance where multiple subclasses inherit from a single base class (A).

→ opposite of multiple inheritance.

Day 82

PyPDF module

→ This module is used for splitting, merging, cropping and transforming the pages of PDF files.

→ It can also add custom data, viewing options, & passwords to PDF files.

Day 84

Time module

→ Time()

→ This function returns the current time as a floating point no. in seconds, starting from epoch (point in time when time module was initialized (1 Jan 1970)).

Syntax:-

```
import time
print(time.time())
```

→ sleep()

→ It delay the time of execution by the given argument in seconds

Syntax: Import time

~~time~~ Print (4)

time.sleep(3)

Print("This is printed after 3 secs")

→ strftime()

→ This function formats a time value as a string, based on specified (given) format.

Syntax: import time

t = time.localtime() # tells local time by converting from seconds.

formatted_time = time.strftime("%Y-%m-%d %H:%M:%S")

Print(formatted_time)

Day 85
Command line utilities

→ These are programs that can be given from the terminal or command line interface, and they are an essential part of many development workflows. In python, you can create your own command line utilities using the built-in 'argparse' module.

Day 86
Walrus operator

→ It is a new addition to python 3.8 onwards.

→ It allows you to assign a value to a variable within an expression; it can be used in a variety of contexts including while loops and if statements.

DATE
PAGE

Syntax:

:=

Eg: a= True
Print(a:= False)

output
False

Day 87

Shutil Module #imp

→ It is a python module that provides a higher level interface for working with file and directories (file).

→ Shutil is short for of shell utility.

→ It provides a convinient and efficient way to automate tasks that are commonly performed on files and directories.

→ Most commonly used function in shutil module :-

- shutil.copy(src,dst) - It copies the data in src file (src - source filename) to dst (destination file name). If destination location already exists, original file will be overwritten.
- shutil.copy2(src,dst) - similar to copy() but preserves more data of original file like time stamps.
- shutil.copytree(src, dst) - same as copy() but for directories (folders), if dst already exists copied files will be merged with it.
- shutil.move(src,dst) - It moves file from src (initial location) to dst (destination location).
- shutil.rmtree(path) - It deletes the directory (folder) located at path.

there is no function is shutil module to delete a file. So we have to use os module, os.remove()

DATE
PAGE

Day 89

Request module

→ It is an HTTP library that enables developers to send HTTP requests in python.
→ useful functions:
- get() - to get html code of site.
- post()

bs4 module

→ It is used for web scraping in python.
→ useful functions:
- BeautifulSoup(html doc, 'html.parser')
- prettify()

Day 91

Generators

→ somewhat similar to dynamic memory allocation.
→ These are special type of functions that allow you to create iterable sequence of values.
→ A generator function returns a generator object, which can be used to generate the values one-by-one as you iterate over it
→ generators allows to generate the value on the fly, rather than having to create and store the entire sequence in memory.

→ **Yeild**

→ You can create generator by using yeild keyword in a function. It returns a value

from the generator and suspends the execution of the function until the next value is requested.

Generators are memory saving and are fast hence Time-saving.

→ next() - function used to execute generator for next value.

Day 92

Function caching

→ Suppose we have a complex function returning result in 5-10 sec (long time), by function caching we can store results into cache and then it won't take time for repeated values.

→ @lru_cache decorator is used by importing it from functools module.

Syntax:

```
from functools import lru_cache
@lru_cache
def func():
    pass
```

→ Used when we have repeated values and complex time taking functions only as it takes additional space.

Day 95

Regular expressions

→ Regular expression or "regex" for short, are tool for with strings and text data in python. They allow you to match and manipulate strings based on patterns.

→ To use regular expressions in python, re module is to be imported. It is a built in module.

→ Useful functions of re module :-

- re.search(pattern, text) - checks if patter is present in text or not. returns None, if pattern is not present.
- re.findall(pattern, text) - used to find all occurences.
- re.finditer(pattern, text) - used before loops to find all occurences one by one.
- re.sub(pattern, "new_text", text) - Used to change/replace a pattern by given string.

Day 96

Asyncio.

→ It is a programming pattern that allows you to run your code concurrently. (एक साथ)

→ To do async programming we have to import asyncio module and use asynchronous function.

→ It is mostly used in cases where functions are time taking seperately like functions to download imgs, vids or other files, in these cases while one function is downloading something other functions concurrently starts downloading.

→ mainly used functions and keywords :-

- async keyword - used before def to make function asynchronous.
- await keyword
- asyncio.sleep()
- asyncio.gather() - for parallel execution.
- asyncio.run()

DATE
PAGE

Day 97

Multithreading

⇒ It is a bit similar to asyncio.
⇒ It is a technique that allows multiple threads of execution to run concurrently within a single process.
⇒ To use multithreading we have to import threading module.
⇒ Steps to create a thread :-
i) import threading
ii) create a Thread() object
iii) call its start() method, which will start the execution.
iv) call its join() method, which will stop the execution.

time.perf_counter() is a similar method to time.time() from time module but time.time() uses system clock from epoch while time.perf_counter() uses the processor's performance counter to measure time.
⇒ basically we use multithreading when we have high internet speed and downloading from different urls from different servers having low server speed.

Concurrent.futures (advance, more practical (multithreading))
⇒ ThreadPoolExecutor
⇒ We will import ThreadPoolExecutor from Concurrent.futures module.

→ ThreadPoolExecutor helps you when you want to schedule a lot of tasks in bulk.

→ steps :-
i) ~~fro~~ from concurrent.futures import ThreadPoolExecutor
ii) make an object for submit() (calls the function)
iii) call result() method for that object.
(gives return value of function)

OR

→ using map function, mainly used for iterables (list).
→ steps :-
i) ~~imp~~ same as before (importing)
ii) make a map() object using list of vals or arguments as 2nd argumet - (calls the function)
iii) print its ~~element~~ return value using for loop. one-by-one ~~in~~ for the object.

Day 98

Multiprocessing

→ It is similar to multithreading.
→ threads are light weight, are into a process used for one CPU for less heavy tasks, while process is done to execute heavy tasks by implying it parallelly in different (multiple) CPU's.
→ multiprocessing allows you to take advantage of multiple cores or processors on your system to significantly improve the performance of your code.
→ steps are similar too as threading.

DATE
PAGE

→ steps :-
i) import multiprocessing
ii) make a Process() object
iii) call start() method from the object
iv) call join() method.

Concurrent.Futures

→ ProcessPoolExecutor
→ Every thing is same as for ThreadPoolExecutor. including steps for both syntax.
→ We just have to import ProcessPoolExecutor instead of ThreadPoolExecutor from concurrent.futures module.