Date 27 11 24

Chapter 1: Modules, comments & pip.

print ("Hello world!") - print is the function

Execute this file (.py) by typing 'python index.py' and you will see 'Hello World!' printed on the screen.

Modules: A module is a file containing code written by somebody else which can be imported and used in our programs.

pip: It is the package manager for python you can use pip to install a module on your system.

'pip install flask' installs flask modules.

Types of modules:

There are two 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:
os, abc, etc.

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

Using python as a calculator:
We can use python as a calculator by typing 'python' on terminal
(This will open REPL: Read Evaluate Print Loop)

Comments: Comments are used to write something which the programmer does not want to execute. (can be used to mark author name, date, etc).

Types of comments:
There are two types of comments of Python:
1. Single line comments: written using #
2. Multi line comments: written using '''___'''

Chapter 2: Variables and Datatypes.

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

```
a = 30
b = 'John'
c = 71.22
```

Variables: container to store value.
Keywords: reserved words in python.
identifiers: class / functions / variable name.

Datatypes:
Primarily, there are following datatypes in python:

1. Integers
2. Floating point numbers.
3. Strings.
4. Booleans
5. None.

Python automatically identifies the type of data.

a = 71 class <int>

b = 88.44 class <float>

c = "John" class <str>

Rules for defining a variable name / identifiers:

- a variable name can contain alphabets, digits and 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.

Examples of a few variable names:
john, one2, seven, -seven, etc.

Operators in python.
Following are common operators in python:
arithmetic + , - , * , /
assignment = , += , -= , etc

Comparison ==, >, >=, <, <=, !=, !=, etc.
Logical and, or, not.

type() function and type casting: It is used to find the datatype of a given variable.

a=31
b="31"
type(a) → class<int>
type(b) → class<str>

A number can be converted into a string and vice versa. There are many functions to convert one data type into another.

str(31) → "31" <str>
int("32") → 32 <int>
float(32) → 32.0 <float>

input() function: This function allows the user to take input from the keyboard as a string.

a = input("Enter name")
(Even if a number is entered, it is considered as a string).

Chapter 3: strings.

String is a datatype in python.
It is a sequence of characters enclosed in quotes.
We can primarily write a string in these three ways:

1. Single quoted strings. ' '
2. Double quoted strings " "
3. Triple quoted strings ''' '''

String slicing: A string in python can be sliced for getting a part of the string. Consider:

name = " John " length = 4.

	J	o	h	n
	0	1	2	3
	-4	-3	-2	-1

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

sl = name [start : end]
first index ↗ ↘ not the last index

sl [0:3] → "Joh" 0 → 3
sl [1:3] → "oh" 1 → 3

Negative indices: Negative indices can also be used as shown in the figure. -1 corresponds to (len-1) index -2 to (len-2)

Slicing with skip value: We can provide a skip value as a part of our slice like this:

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

Other advanced slicing techniques:

```
word = 'amazing'
word [0:]  → word [0:7] → 'amazing'
word [:6] → word [0:6] → 'amazin'
```


String functions:
some of the mostly used functions to perform operations on or manipulate strings are:

1. len() function: This function returns the length of the string

```
len("Joh")  → 3
```

2: string.endswith("oh"): This function tells whether the variable string ends with 'oh' or

not. If string is 'Joh', it reaches to true for 'oh' since 'Joh' ends with 'oh'

3. String.count('c'): counts the total number of occurrence of any character.

4. String.capitalize(): capitalizes the ~~first~~ first character of a given string.

5. String.find(word): This function finds a word and returns the index of first occurrence of that word in the string.

6. String.replace(oldword, newword): This function replaces the old word with new word in the entire string.

Escape sequence characters:
sequence of characters after backslash '\':
It comprises of more than one characters but represents one character when used within the strings.

\n	newline
\t	tab
\'	single quote.
\\	backslash.

Chapter 11: Lists and Tuples.

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

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

(can store value of any datatype)

List indexing: a list can be indexed just like a string.

L1 = [7, 9, 'John']

L1[0] = 7, L1[1] = 9, L1[0:2] = [7, 9]

List methods: consider the following list:

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

1. L1.sort(): updates the list to [1, 2, 7, 8, 15, 21]
2. L1.reverse(): updates list to [15, 21, 2, 7, 8, 1]
3. L1.append(8): adds 8 to the end of list.
4. L1.insert(3, 8): This will add 8 at 3rd index.
5. L1.pop(2): delete element at index 2 and return its value.
6. L1.remove(21): will remove 21 from the list.

Tuple: A tuple is an immutable data type in python.

a = ( ) (empty tuple)
a = (1,) (Tuple with only one element needs a coma,)
a = (1, 7, 2) (Tuple with more than one element)

Once defined a tuple's element cannot be altered or manipulated.

Tuple methods: consider the following tuple:
a = (1, 7, 2)

1. a.count(1) : It will return the no. of times 1 occur in a.
2. a.index(1) : will return the index of first occurrence of 1 in a.

Chapter 5: Dictionary and Sets.

Dictionary is a collection of key value pairs.

Syntax :
```
a = { "key" : "value",
      "marks" : "100",
      "name" : "Sohan",
      "list" : [1, 2, 9] }
```

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

Properties of dictionaries:
1. unordered.
2. mutable.
3. indexed
4. cannot contain duplicate keys.

Dictionary methods: Consider the following dict.

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


1. a.items() : returns a list of (key, value) tuples.
2. a.keys() : returns a list containing dictionaries keys
3. a.update({"friend" : "Sam"}) : update the dictionary with supplied with key-value pairs.
4. a.get("name") : returns the value corresponding to specified key.

more methods are available on python.org (docs)

Sets in Python: A collection of non repetitive elements. They are datatypes containing unique value.

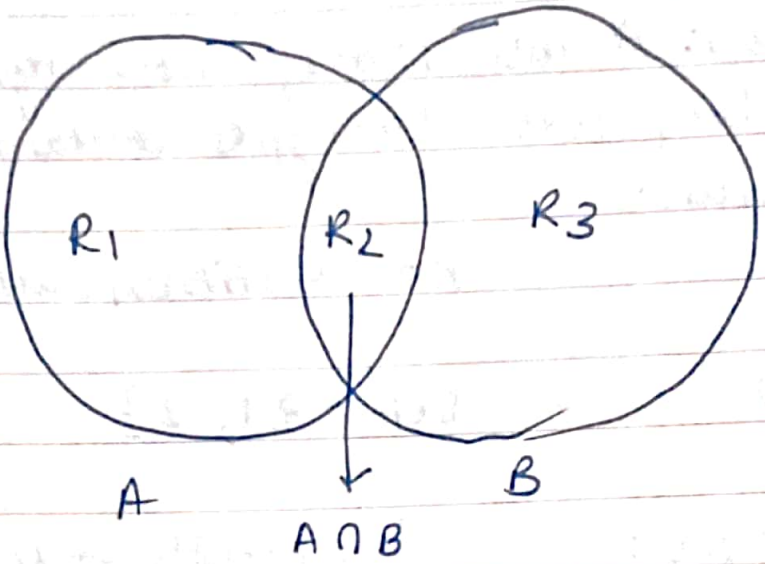
```
S = Set()                    no repetition allowed.
S.add(1)
S.add(2)                     set = {1, 2}
```

Properties of sets: [Element's order doesn't matter. Cannot access elements by index]

1. sets are unordered
2. unindexed
3. immutable.
4. cannot contain duplicate values.

Operations on sets:
Consider the following set: S = {1, 8, 2, 3}

1. len(S) : Returns 4, length of set.
2. S.remove(8) : updates S and removes 8.
3. S.pop() : removes the arbitrary element from the set and returns the element removed.
4. S.clear() : Empties the set S.
5. 8.union({8, 11}) : Returns a new set with all items from both sets {1, 8, 2, 3, 11}
6. 8.intersection({8, 11}) : Returns a set which contains only items in both set: {8}



$R_2 \rightarrow A \cap B$

$R_1 + R_2 + R_3 \rightarrow A \cup B$

$R_1 + R_3 \rightarrow A \Delta B$

$R_1 \rightarrow A - B$

$R_3 \rightarrow B - A$

Chapter 6: Conditional Expressions.

IF, else and ~~elsei~~ elif in python: It is a multiway decision taken by program due to certain conditions in our code.

Syntax:

```
if (condition1):
    print ("Yes")
elif (condition2):
    print ("Maybe")
else:
    print ("No")
```

Relational operators: It is used to evaluate conditions inside the if statements.

==	equals
<=	less than equal to
>=	greater than equal to

Logical operators: operate on conditional statements.

and: true if both operands are true else false.

or: true if atleast one operand is true else false.

not: Inverts true to false and vice versa.

elif clause: [elseif]: can be chained together with lot of elif followed by else statements.

Note:

- There can be any number of elif statements.
- Last else is executed if all the conditions fail

Chapter 7: Loops in python.

Types of loops in Python:
- while loop
- for loop: →

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

- while loop:

Syntax: while condition:
    # code

condition is checked first. If it is true, the body of loop will be executed else not. If the loop is entered, process of execution continues till the condition is false.

If condition never becomes false, the loop will iterate forever.

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

range (start, stop, step-size)
not usually used.

Eg: for i in range (0, 7): [prints 0 to 6]
    print (i)

for loop with else: used when loop exhaust.

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

Break statement: used to come out of the loop when encountered.

```
for i in range(0, 80):
    print(i)
    if i == 3:
        break
```


Continue statement: used to stop the current iteration of the loop and continue with the next one. It instructs to 'skip this iteration'.

```
for i in range(4):
    print("printing")
    if i == 2:
        continue
    print(i)
```

pass statement: null statement saying 'Do nothing'

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

→ if not, it will show error.

Chapter 8: Functions & Recursion:

A function is a group of statements performing a specific task.
It can be reused by the programmer in a given program any number of times.

Example and syntax:

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

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

Function call: func1()

Types of functions in python:
1. Built in functions (already present in python)
2. User defined functions (defined by users)

Examples:
len(), print(), range(), etc are built in functions
func1() is user defined function.

Function with arguments: A function can accept some values it can work with: we can

put these values in the parenthesis. A function can also return values as shown below:

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

Default Parameter value: Suppose if we specify name = "stranger" in the line containing def, then this value will be displayed in case no argument is passed during function call.

Recursion: It is a function which calls itself repeatedly. It is used to directly use a mathematical formula as a function.

Example: factorial(n) = n × factorial(n-1)

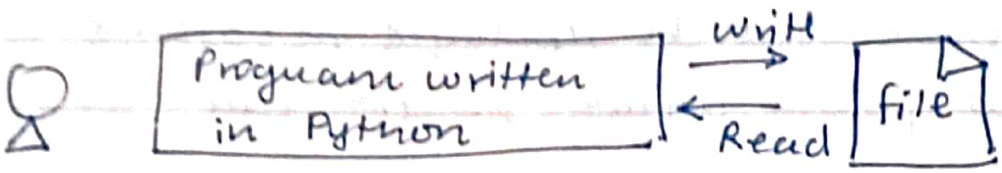
```
def factorial(n)
    if i == 0 or i == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Note that the function doesn't keep calling itself infinitely and the base condition is specified.

Chapter 9: File I/O

The Random Access Memory (RAM) is volatile and all its contents are lost once a program terminates. In order to persist the data forever, we use files.

A file is a data stored in a storage device. A Python program can talk to the file by reading content from it and writing content to it



RAM → volatile

HDD → non volatile.

Types of files

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

Opening a file: open() function is used for opening a file. It takes two parameters: filename and mode.

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

We can specify the number of character in read() function: f.read(2)

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

Other methods to read the file: f.readline()

Methods of opening a file:

- r → open for reading
- w → open for writing
- a → open for appending
- + → open for updating
- rb → open read in binary mode.
- rt → open read in text mode.

Writing files in python: In order to write a file in python, we first open it in write or append mode after which, we use the python's f.write() method to write the file.

```
f = open("this.txt", "w")
f.write("This is nice")
f.close()
```

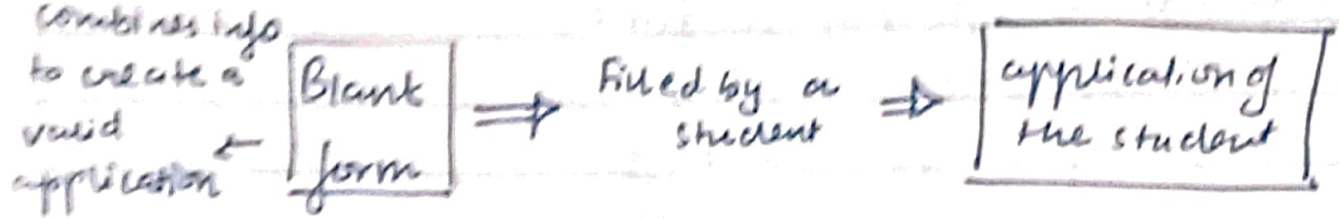
with statement: best way to open.

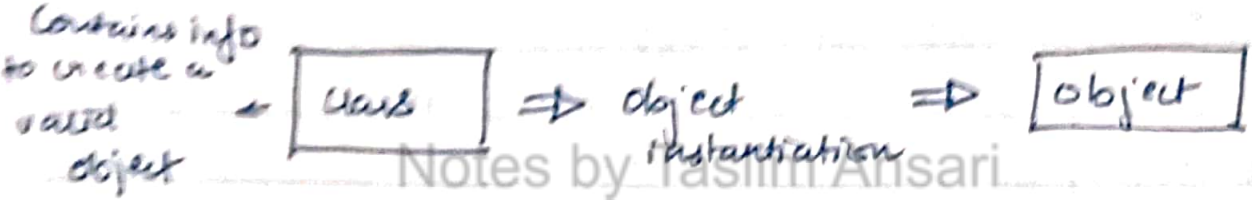
```
with open("this.txt") as f:
    f.read() -> (no need for f.close())
```

Chapter 10: Object Oriented Programming

This concept focuses on using reusable code (implements dry principle)

Class: A class is a blueprint for creating objects.





Syntax: class Employee:
    # methods and 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 needing the implementation details to the user. (Abstraction and Encapsulation)

Modelling a problem in OOPs:

We 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 a class rather than a particular object.
Eg: class Employee:
company = "Google"

john = Employee()
john.company
Employee.company = "YouTube"

Instance attribute: An attribute that belongs to the instance (object). Assuming the class from the previous eg: john.name = "John"
john.salary = "30k"

Note: Instance attributes take preference over class attributes during assignment and retrieval

'self' parameter: self refers to the instance of the class. It is automatically passed with a function call for an object.
john.getSalary() → self is john
↳ = to Employee.getSalary(john)

Function is stated as:

```
class Employee:
    company: "Google"
    def getSalary (self):
        print ("Salary is not there")
```

Static method: Sometimes we need a function that doesn't use the self parameter. We can define a static method like this:

```
@staticmethod
def greet():
    print ("Hello user!")
```

--init__() constructor: It is a special method which is first run as soon as the object is created. It is also known as constructor.
It takes self argument and can also take further arguments.

Eg:
```
class Employee:
    def __init__(self, name):
        self.name = name
    def getSalary (self):
        # code

john = Employee ("john")
```

Chapter 11: Inheritance and more on OOPS

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

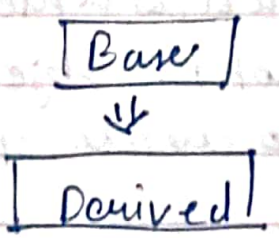
Syntax:
```
class Employee:
    # code
    :

class Programmer(Employee):
    # code.
```

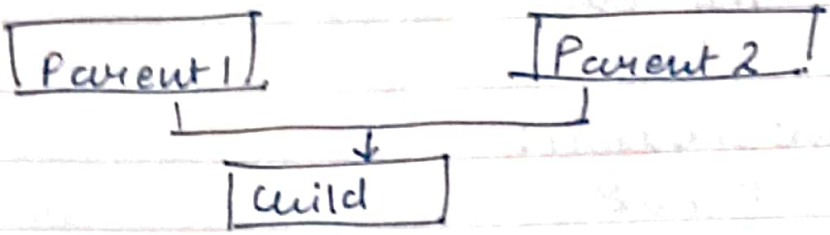
we can use the methods and attributes of Employee in Programmer object.
Also, we can overwrite or add new attributes and methods in Programmer class.

Types of Inheritance:
- Single inheritance
- multiple inheritance
- Multilevel inheritance.

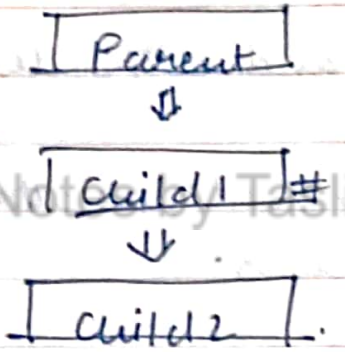
Single inheritance: It occurs when child class inherits only a single parent class



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



Multilevel inheritance: When a child becomes a parent for another child class.



Super() method: It is used to access the methods of a super class in the derived class.

Super().__init__() # calls constructor of the base class.

class methods: A class method is a method which is bound to the class and not the object of the class. '@classmethod' is used to create class method.

Syntax:

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

⇒ @property decorators: Consider:

```
class Employee:
    @property
    def name (self):
        return self.ename
```

if e = Employee() is an object of class employee, we can ~~call~~ print(e.name) or call name() to print the ename.

@.getters and @.setters:

The method name with @property decorator is called getter method.

We can define a function + @name.setter decorator like below:

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

Operator Overloading in Python:

Operators in python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.
Operators in python can be overloaded using the following methods.

p1 + p2 → p1.__add__(p2)
p1 - p2 → p1.__sub__(p2)
p1 * p2 → p1.__mul__(p2)
p1 / p2 → p1.__truediv__(p2)
p1 // p2 → p1.__floordiv__(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).

Chapter 12: Advanced Python

Exception handling in Python.

There are many built in Exceptions which are raised in Python when something goes wrong.
Exceptions in Python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
    # 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 exceptions to catch like below:

```
try:
    # code
except ZeroDivisionError:
    # code
except TypeError:
    # code
except :
    # code ( all the other exceptions)
```


Raising exceptions:
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 (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:
  # code (executed 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 the behaviour is used to check whether

the module is run directly or imported from another file.

The global keyword:
global keyword is used to modify the variable outside of the current ~~type~~ scope.

Enumerate function in python:
The enumerate function adds counter to an iterable and returns it.

```
for i, item in list1:
    print(i, item)
```


List comprehensions:
List comprehensions is an elegant way to create lists based on existing lists.

```
list1 = [1, 7, 12, 11, 22]
list2 = [ i for item in list1 if item > 8]
```

Chapter 13: Advanced Python 2.

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

Installation
To use virtual environments, we write:

```
pip install virtual env
```

we create a new environment using:

```
virtualenv myprojectenv
```

The next step after creating the virtual environment is to activate it.
We can now use this virtual environment as a separate python installation.

pip freeze command:
pip freeze returns all the packages installed in a given python environment along with the versions.


```
pip freeze > requirements.txt
```

The above command creates a file named requirements.txt in the same directory containing the output of pip freeze.
We can distribute this file to other users and they can recreate the same environment using:

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

Lambda functions

functions created using an expression using lambda keyword.

lambda arguments : expressions

↳ can be used as a normal function.

Example:

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

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


bin method (strings)

creates a string from iterable objects

```
d = ["apple", "mango", "banana"]
", and, ".join(d)
```

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(p1, p2, ...)
↳ arguments.

Syntax:

"{} is a good {}".format("John", "boy")
→ John is a good boy.

"{1} is a good {0}".format("John", "Boy")
→ Boy is a good John.

* Maps, filter and reduce:

Map applies a function to all the items in an input list.

Syntax: map(function, input_list)

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

list(filter(function))

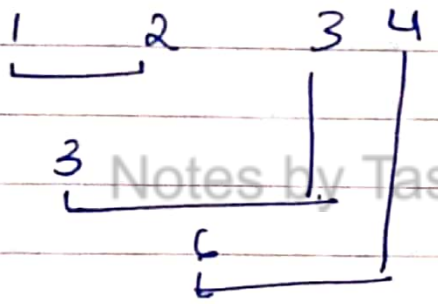
Date

Reduce function

Reduce applies a rolling computation to a sequential pair of elements.

```
from functools import reduce
val = reduce (function, list1)
```

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



Sequential computation

→ returns 10.