MY-PYTHON-JOURNEY

BASIC TO ADVANCED

CONFIDENTIAL - PROPERTY OF VENOM

- VARIABLES
- VARIABLES
- LOOPS & ITERATION
- LOOPS & ITERATION
- FUNCTIONS
- LOOHS & TRDCTCREGINES
- OFJECT-INE CRATIHES
- OBJECT-ORIRIRT & PRSGRMMING
  STREAHOD SOCIENT & DATD
- DATA STRUCTIUNS
- PYTHNS PRDGAMMING

- OBJECT-DRISNT HITHLING
  HMSINEL RESTION WEEENT
- CGTUS TUCICTING
  TAUNC PILOBREES
- OBJEUG LN OKTIAMMING

DARS STRUCTUREX

FUNCTJBNS

PYTHOM EHO-IE CHTED

- BONEE HVA FROM
- FE GHHOLL CORHE
- LUSE DAD STHOIRAS

ED IDRAS CONUSE
ATHIAD-TRORC FHES
BAET LONGE ANDNTERS

PRYTHON STEECHONS

PYTHON

01/10/25

PYTHON

Programming: Programming is a way to instruct the computer to perform various tasks.

PYTHON: It is a popular, general-purpose programming language known for its simple syntax, making it easy to read and learn like English. DEVELOPED BY "GUIDO VAN ROSSUM"

Features:

* Easy to understand = Less development time
* Free and open source
* High level language
* Portable: Works on Linux/Windows/Mac
* Fun to work with.

Installation

It can be downloaded from python.org. When the file is downloaded intall it simply on your system.

CHAPTER=1 MODULES, COMMENTS & PIP

Lets write our first code.

```
print ("Hello Guys")
```

After executing it will return "Hello Guys.

MODULES

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

PIP

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

```
pip install flask # Installs Flask module.
```

Types of Modules

1. Built in Modules (Pre-installed in python)
2. External Modules (Need to install using pip)

Some examples of Built in modules are os, random, etc.

" " external modules are tenserflow, flask, etc.

Using Python as Calculator

We can use python as a calculator by typing "python"+↵ on terminal.

This opens REPL or Read Evaluate Print Loop.

COMMENTS

Comments are used to write something which progammer doesn't want to execute. This can be used to mark author name, date etc.

Single line Comment: To write a single line comment just add a '#' at start of the line.

```
# This is a Comment
```

Multiline Comment: To write a multi line comment we can use '#' at each line or we can use the multiline string (""" """).

```
"""
This is a
multi-line comment
"""
```

CHAPTER=2 Variables and Datatype

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

```
a= 10
b= "Shiv"
c= 71.22
```

Data Types

Primarily these are the following data types in python:

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

Python automatically identifies the type of data for us.

Rules for Choosing an Identifier

* A variable can contain alphabets, digits, and underscores.
* A variable name can only start with an alphabet and underscores.
* A variable name can't start with a digit.

* No while space is allowed to be used inside a variable name.

Example:- shivu, one8, seven, _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 AND 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)  # class <str>
```

A number can be converted into string and vice versa (if possible)

There are many functions to convert one data type into another.

```
str(31)      =>  "31"     # int to string
int("31")    =>  31       # string to int
float(31)    =>  31.0     # int to float
```

.... and so on

Here "31" is a string literal and 31 a numeric literal.

INPUT() FUNCTION

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

```
A= input("enter name")
```

It is important to note that the output of a string is always a string (even if a number is entered).

CHAPTER - 03

STRINGS

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

```
a = 'shivu'       # single quoted string
b = "shivu"       # double quoted string
c = '''shivu'''   # Triple quoted strings
```

STRING SLICING

A string in python can be sliced for getting a part of the string.

Consider the string.

```
Name = "shivu"  => Length = 5
       0 1 2 3 4
       | | | | |
      (-5)(-4)(-3)(-2)(-1)
```

figure1

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

(Inclusive) (Exclusive)

sl = name[ind_start : ind_end]

First index included ↓ ↳ Last index is not included.

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

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

[3(end) is excluded]

Negative Indices

Negative indexes can also be used as shown in the figure1. -1 corresponds to the (length-1), index -2 to (length-2).

SLICING WITH SKIP VALUE

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

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

Other advanced slicing techniques:

```
Word = "amazing"
Word = [:7)  # word[0:7] - 'amazing'
Word = [0:]  # Word[0:7] - 'amazing'
```

STRING FUNCTION

Some of the commonly used function to perform operations on or manipulate strings are as

follows. Let us assume there is a string 'str' as follows:

```
str = 'shivu'
```

Now when operated on this string 'str', these functions do the following:

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

```
str = "shivu"
print(len(str))   # Output: 5
```

2. string.endswith("ivu") - This function tells whether the variable string "ivu" or not. If string is "shivu", it returns true for "ivu" since shivu ends with ivu.

```
str = "shivu"
print(str.endswith("ivu"))  # Output: True
```

3. string.count("c") - Counts the total number of occurance of any character.

```
str = "shivu"
count = str.count("v")
print(count)  # Output: 1
```

4. The first character of a given string

```
str1 = "shivu"
capitalized_string = str1.capitalize()
print(capitalized_string) # Output: "Shivu"
```

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

```
str1 = "shivu"
index = str1.find("iv")
print(index) # output = 2
```

6. string.replace(old word, new word) -

This function replaces the old word with new word in the entire string.

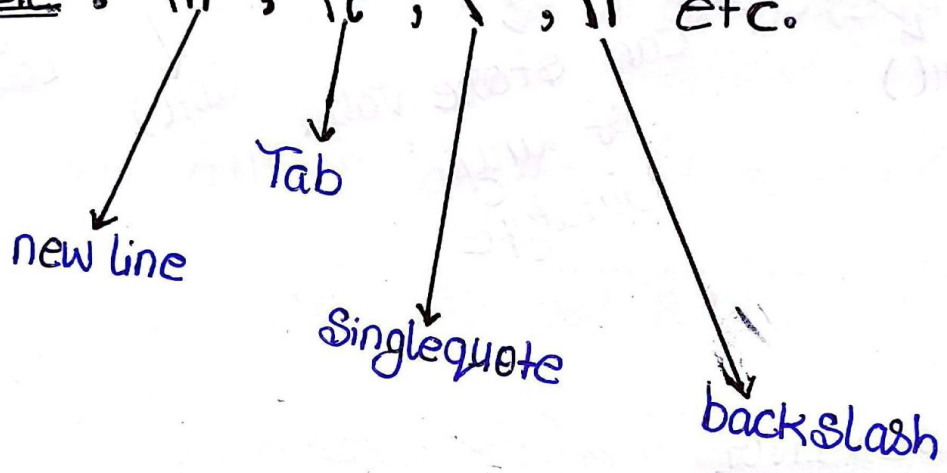
```
str1 = "shivu"
replaced_string = str1.replace("i", "1")
print(replaced_string) # Output : "sh1vu"
```

Escape Sequence Characters

Sequence of characters after backslash "\" → Escape Sequence Characters.

Escape sequence characters comprises of more than one character but represent one character when used within the strings.

Example : \n , \t , \' , \\ etc.

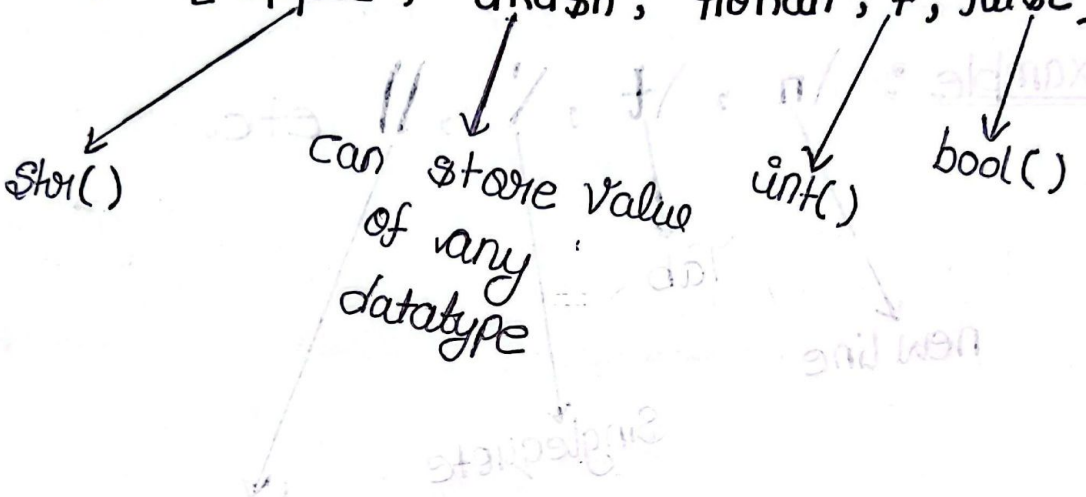


CHAPTER-04 LISTS AND TUPLES

List:

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

```
friends = ["apple", "akash", "mohan", 7, false]
```



LIST INDEXING

A list can be indexed just like a string.

```
l1 = [7, 9, "shivu"]
l1[0]   #7
l1[1]   #9
l1[70]  # error
l1[0:2] # [7, 9]  # list slicing
```

LIST METHODS

Consider the following list:

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

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

* ll.reverse() : updates the list to [15, 21, 2, 7, 8, 1]
  → this flips the list first in last, 2nd one in second last and vice versa.

* ll.append(8) : adds 8 at the end of the list.

* ll.insert(3, 8) : This will add 8 at 3 index.

* ll.pop(2) : Will delete element at index 2 and returns its value.

* ll.remove(21) : Will remove 21 from list.

TUPLES IN PYTHON

A tuple in Python is a built-in, ordered collection of items, much like a list, The key difference is that tuples are immutable, meaning once a tuple is created, you cannot change its size or the values of its elements.

```
a=()   # empty tuple
a = (1,)  # tuple with only one element needs a comma
a= (1, 7,2) # tuple with more than one element
```

TUPLE METHODS

Consider the following tuple.

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

* a.count(1):

a count(1) will return number of times 1 occurs in 'a'.

* a.index(1)

will return the index of first occurrence 1 in a.

CHAPTER=05 DICTIONARY AND SETS

Dictionary is a collection of keys-value pairs.

Syntax:

```
a= {
      "key": "value",
      "shivu": "code",
      "marks": "100",
      "list" : [1, 2, 9]
   }
print(a["key"])   # Output : "value"
print(a["lists"]) # Output : [1, 2, 9]
```

Properties of Python Dictionaries

1. It is unordered. [For recent versions since 3.7 => ordered)
2. It is mutable. [changeable]
3. It is indexed
4. Cannot contain duplicate keys.
5. Values can be anything (heterogeneous)
   - ↳ Strings
   - ↳ Numbers
   - ↳ Lists
   - ↳ Other Dictionaries (Nested dictionaries)
   - ↳ Tuples
   - ↳ Any Python object!

DICTIONARY METHODS

Consider the following dictionary.

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

* a.items(): Returns a list of (key, value) tuples.

* a.keys(): Returns a list containing dictionary's keys.

* a.update({"friends":3}): Updates the dictionary with supplied key-value pairs.

* a.get("name") : Returns the value of the specified keys (and value is returned e.g. "Shivu" is returned here).

More methods are available on docs.python.org

SETS IN PYTHON

Set is a collection of non-repetive elements.

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

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

Properties of Sets

1. Sets are unordered ⇒ Element's order doesn't matter.
2. Sets are unindexed ⇒ Cannot access elements by index.
3. There are no way to change items in sets.
4. Sets cannot contain duplicate values.

Operations on Sets

Consider the following set:

$S = \{1, 0, 2, 3\}$

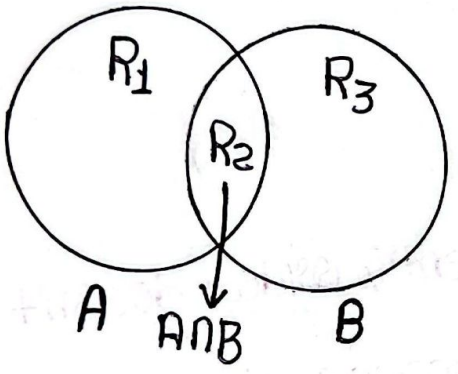
* Len(S): Returns 4, the length of the set.
* S.remove(0): Updates the set S and removes 0 from S.

* S.pop(): Removes an arbitrary element from the set and return the element removed.

* S.clear: empties the set S.

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

* S.intersection({8,11}): Return a set which contains only item in both sets {8}.



R2 = A∩B
R1+R2+R3 = A∪B
R1+R3 = A Δ B
R1 = A-B
R3 = B-A

CHAPTER=06 CONDITIONAL EXPRESSION

A conditional expression in Python, often called a Ternary Operator, is a compact way to write a simple `if-else` statement on a single line. It allows you to a variable based on a condition.

* Goal: To pick one of two values based on whether a condition is True or False.

* Format: It follows the structure: `value_if_true if condition else value_if_false.`

* Assignment: The result of the expression is immediately assigned to a variable.

* Readability: It's used for simple, clear decision to keep code concise

Python Code

```
age =20
if age >= 18 :
  status = "Adult"
else:
   status = "Minor"
```

In python, we execute instructions on conditions being met.
This is what conditionals are for!

IF ELSE AND ELIF IN PYTHON

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

Syntax:

```
if (condition1): # if condition1 is True
    print ("yes")
elif (condition2): # if condition2 is True
    print ("no")
else :              # otherwise
    print ("maybe")
```

CODE EXAMPLE

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

RELATIONAL OPERATORS

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

==: equals.
>=: greater than/equal to
<=: lesser than / equal to

LOGICAL OPERATORS

In python logical operators operate on conditional statements. For Example:

* and - true if both operands are true else false.
* or - true if at least one operand is true or else false.
* not - inverts true to false & false to true.

ELIF CLAUSE

Elif in python means [else if]. An if statements can be chained together with a lot of these elif statements followed by an else statement.

```
if (condition1):
        # code
elif (condition2): # this ladder will stop once a
    condition is an if or elif is met.
        # code.
    elif (condition3):
            # code
    else:
        # code.
```

Important notes:

1. There can be any number of elif statements.
2. Last else is executed only if all the conditions inside elifs fail.

CHAPTER - 07 LOOPS IN PYTHON

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000.

Loops make it easy for a programmer to tell the computer which set of instructions to repeat and how!

TYPES OF LOOPS IN PYTHON

Primarily there are two types of loops in python.

* While loops
* For loops

We will look into these one by one.

WHILE LOOP

Syntax

```
while(condition): # The block keeps executing until
                  # the condition is true.
    # Body of the Loop
```

In while loops, the condition is checked first. If it evaluates to true, the body of the loop is executed otherwise not!

If the loop is entered, the process of [condition check & execution] is continued untile the condition becomes false.

Example:

```
i=0
While i<5: # print "shivu" -5 times
    print("shivu")
    i= i+1
```

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

FOR LOOP

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

Syntax

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

RANGE FUNCTION IN PYTHON

The range() function in python is used to generate a sequence of numbers.

We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
# step_size is usually not used with range()
```

An example demonstrating Range() 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.

Example:

```
l = [1,7,8]
for item in l:
    print(item)
else:
    print("done") # this is printed when 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.

Example:

```
for i in range(0,80):
print(i)
if i==3
   break
```
this will print 0,1,2 and 3.

THE CONTINUE STATEMENT

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

Example:

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

PASS STATEMENT

Pass is a null statement in python.

It instructs to "do nothing".

Example:

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

CHAPTER-08 Functions & Recursions.

FUNCTION

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

↳ It is referred to as DRY (Don't Repeat Yourself)

Example & Syntax of a function

The syntax of a function looks as follows:

```
def func1():
    print('hello')
```

This function can be called by 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 instruction which are executed during the function call.

TYPES OF FUNCTIONS IN PYTHON

There are two types of functions in python:

* Built in function (Already present in python)
* User defined functions (Defined by the user)

Examples of buit in functions includes len(), print(), range() etc.

The func1() function we defined is an example of user defined function.

Functions with Arguments

A function can accept some value it can work with. We can put these values in the parenthesis

A function can also return value as shown below:

```
def greet(name):
    gr = "hello" + name
    return gr
a = greet("shivu")
# a will now contain "hello shivu"
```

DEFAULT PARAMETER VALUE

We can have a value as default as default arguement in a function.

If we specify name = "stranger" in the line containing def, this value is used when no arguement is passed.

Example:

```
def greet(name = "stranger"):
    # function body
greet() # name will be "stranger in func body (default)
greet("shivu") # name will be "shivu" in function body (passed)
```

RECURSION

* Recursion is a function when calls itself.
* It is used to directly use a mathematical formula as function.

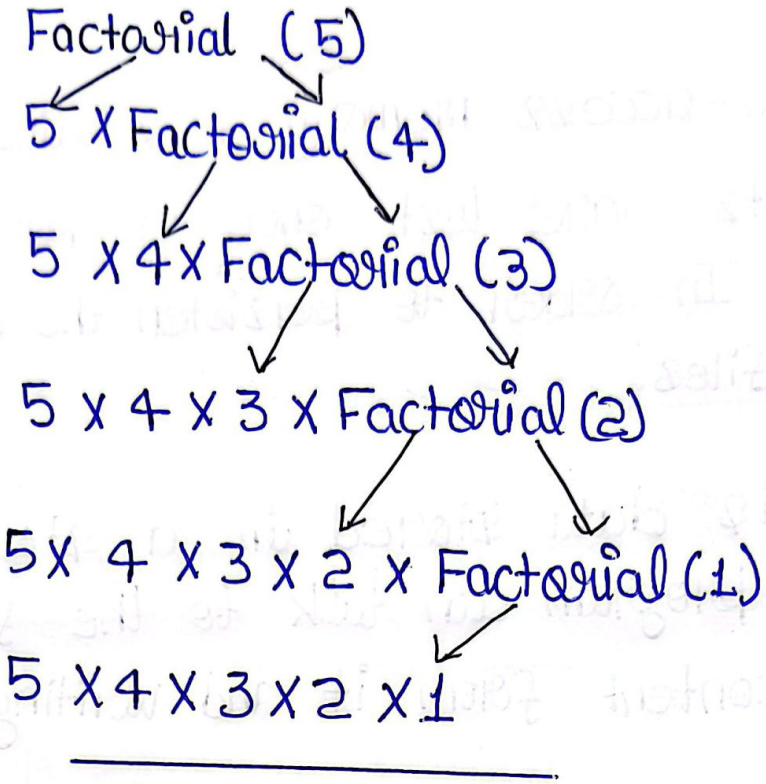
Example

$$factorial(n) = n \times factorial(n-1)$$

This function can be defined as follows:

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

This work follows:



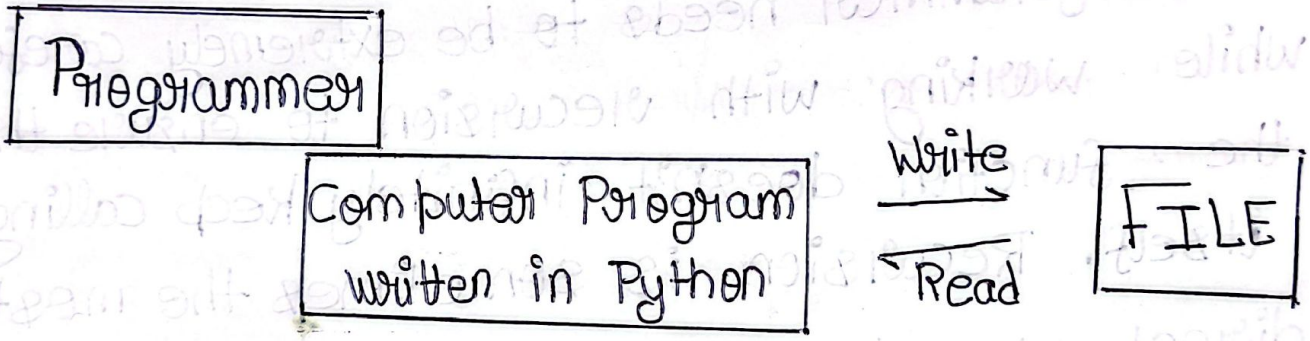
The programmer needs to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep calling itself. Recursion is sometimes the most direct way to code an algorithm.

CHAPTER-09

FILE I/O

* The random-access memory is volatile, and all its contents are lost once a program terminates. In order to persist the data forever, we use files.

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



RAM = Volatile
HDD = Non-Volatile

Type of Files

There are 2 types of files:
1. Text files (.txt, .c, etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions of reading, updating,

and deleting files.

OPENING A FILE

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

```
# open("filename", "mode of opening (read mode by default)")
  open("this.txt","r")
```

READING A FILE IN PYTHON

```
# Open the file in read mode
f = open("this.txt", "r")
# Read its contents
 text = f.read()
# Print its contents
print(text)
f.close      # close the file.
```

Other Methods to Read the file

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

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

Modes of Opening a file

r - open for reading
w - open for writing
a - open for appending
+ - open for updating

'rb' will open for read in binary mode.
'rt' will open for read in text mode.

WRITE FILES IN PYTHON

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

```
# Open the file in write mode
f = open("this.txt", "w")
# Write a string to the file
f.write("this is nice")
# Close the file
f.close()
```

With Statement

This best way to open and close the file automatically is the with statement.

```
# Open the file in read mode using 'with', which automatically closes the file
with open("this.txt", "r") as f:
    # Read the contents of the file
    text = f.read()
    # Print the contents
    print(text)
```

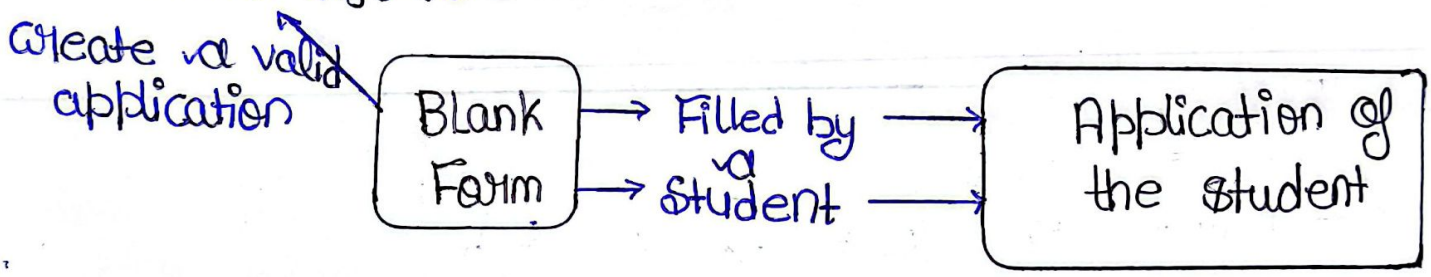
CHAPTER-10 OBJECT ORIENTED PROGRAMMING

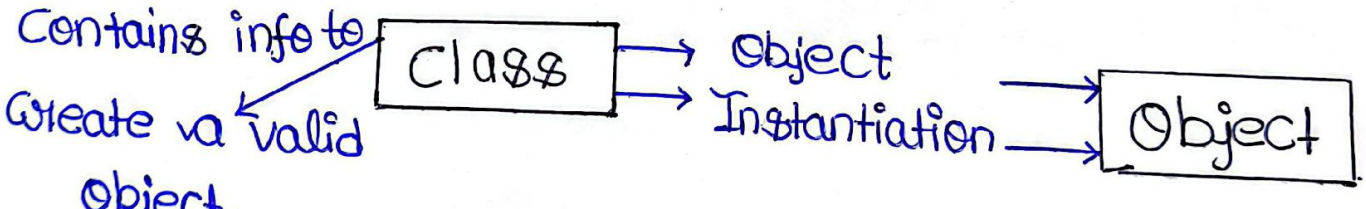
* Solving problem by creating object is one of the most popular approaches in programming. This is called object-oriented programming.

* This concept focuses on using reusable code (DRY Principle).
  ↓
  {Don't Repeat Yourself}

CLASS

A class is a blueprint for creating object.





Syntax:

```
class Employee: # Class name is written in pascal case
                # 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!

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 the class rather than a particular object.

Example:

```
class Employee:
    company = "Google" # Specific to Each Class
shivu = Employee() # Object Instantiation
shivu.company
Employee.company = "Youtube"
  #Changing Class Attribute
```

INSTANCE ATTRIBUTES

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

```
shivu.name = "shivu"
shivu.salary = "30k"  # adding instance attribute
```

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

When looking up for shivu.attribute it checks for the following:

1] Is attribute present in object?

2] Is attribute present in class?

SELF PARAMETER

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

```
shivu.getSalary() # here self is shivu
# equivalent to Employee.getSalary(shivu)
```

The function getSalary() is defined as:

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

STATIC METHOD

Sometimes we need a function that does not 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 is first run 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 Example:

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

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

Syntax:

```
class Employee : # Base class
 # code
class Programmer (Employee): # Derived for
      child class
 # Code
```

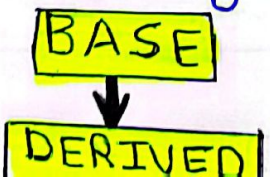
* We can use the method and attributes of 'Employee' in 'Programmer' object.
* Also, we can overwrite or add new attributes and methods in 'Programmer' class.

Types of Inheritance

* Single Inheritance
* Multiple inheritance
* Multilevel inheritance

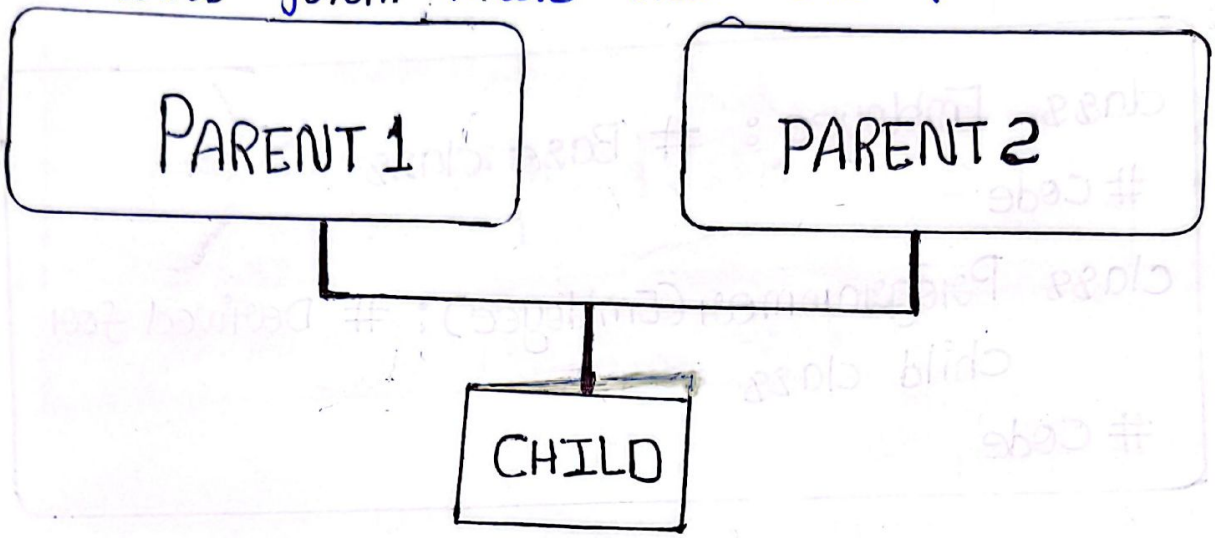
SINGLE INHERITANCE

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



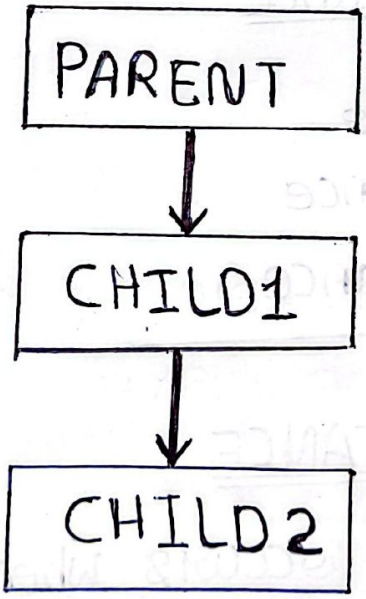
MULTIPLE INHERITANCE

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



MULTILEVEL INERITANCE

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__()
#__init__() Calls constructor of the base class.
```

CLASS METHOD

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

@classmethod decorator is used to create a class method.

Syntax:

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

@PROPERTY DECORATORS

Consider the following class:

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

If e = Employee() is an object of class employee, we can print(e.name) to print the ename by internally calling name() function.

@.GETTERS AND @.SETTERS

The method name with '@property' decorator is called getter method.
We can define via 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 method.
* 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 dunders/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)
```

ADVANCED PYTHON

NEWLY ADDED FEATURES IN PYTHON

Following are some of the newly added features in Python programming language.

Walrus Operator

The walrus operator (:=), introduced in Python 3.8 allows you to assign values to variables as part of an expression. This operator, named for its resemblance to the eyes and tusks of a walrus, is officially called the "assignment expression."

```
#Using walrus operator
if (n := len([1, 2, 3, 4, 5])) > 3:
    print(f"List is too long ({n} elements, expected <=3)")
#Output: List is too long (5 elements, expected <=3)
```

In this example, n is assigned the value of len([1,2,3,4,5]) and then used in the comparison within the if statement

TYPES DEFINITIONS IN PYTHON

Type hints were added using the colon (:) syntax for variables and the -> syntax for function return types.

```
# variable type hint
age: int = 25
# Function type hints
def greeting(name: str) -> str:
    return f"Hello, {name}!"
# usage
print(greeting("Alice"))
#Output: Hello, Alice!
```

ADVANCED TYPE HINT

Python's typing module provides more advanced types hints, such as List, Tuple, Dict and Union.

You can import List, Tuple and Dict types from the typing module like this:

```
from typing import List, Tuple, Dict, Union
```

The syntax of types looks something like this.

```
from typing import List, Tuple, Dict, Union

# List of integers
numbers: List[int] = [1,2,3,4,5]
# Tuple of a string and an integer
person: Tuple[str,int] = ("Alice",30)
# Dictionary with string keys and integer values
score: Dict[str, int] = {"Alice": 90, "Bob": 85}

# Union type for variables that can hold multiple types.
identifier: Union[int, str] = "ID123"
identifier = 12345  # Also valid.
```

This annotation help in making the code self-documenting and allow developers to understand the data structures used at a glance.

MATCH CASE

Python 3.10 introduced the match statement, which is similar to the switch statement found in other programming languages.

The basic syntax of the match statement involves matching a variable against several cases using the case keyword.

```
def http_status(status):
    match status:
        case 200:
            return "OK"
        case 404:
            return "Not Found"
        case 500:
            return "Internal Server Error"
        case _:
            return "Unknown Status"

# Usage
print(http_status(200)) # Output : OK
print(http_status(404)) # Output: Not Found
print(http_status(500)) # Output : Internal Server Error
print(http_status(403)) # Output : Unknown Status.
```

DICTIONARY MERGE & UPDATE OPERATOR

New operators | and |= allows for merging and updating dictionaries.

```
dict1 = {'a':1,'b':2}
dict2= {'b':3,'c':4}
merged = dict1 | dict2
print(merged) # Output:{'a':1,'b':3,'c':4}
```

You can now use multiple context managers in a single with statement more cleanly using the parenthesis context manager.

```
with (
    open('file1.txt') as f1,
    open('file2.txt') as f2
):
    # Process files
```

Exception Handling in Python

There are many built-in exceptions which are raised in python when something goes wrong.

Exception in python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
    #Code which might through exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exception to catch like below:

```
try:
    #Code
except ZeroDivisionError:
    #Code
except TypeError:
    # Code
except :
    #Code  # All other exceptions are handled here.
```

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:
    #SomeCode
except:
    #Somecode
else:
    #code
        # This is executed only if the try was
          successful.
```

TRY WITH FINALLY

Python offers a 'finally' clause which ensures execution of a piece of code inspective of the exception.

```
try:
    # Some Code
except
    # Some Code
finally
    # Some 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 run
* If the module is being run directly from the command line, the '__name__' is set to string "__main__". Thus, this behaviour is used to check whether the module is run directly or imported to another file.

The Global keyword

'global' keyword is used to modify the variable outside of the current scope.

ENUMERATE FUNCTION IN PYTHON

The 'enumerate' function adds counter to an iterable and returns it.

```
for i, item in list1:
    print(i, item) # Prints the items of the list 1
                   with index.
```

LIST COMPREHENSIONS

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

```
list1 = [1, 7, 12, 11, 22]
list2 = [i for item in list 1 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 virtualenv # Install the package
```

We create va new environment using:

```
virtualenv myprojectenv # Creates a new venv
```

The next step after creating the virtual environments to activate it.

We can now use this virtual environment as a seperate Python installation.

PIP FREEZE COMMAND

'pip freeze' returns all the package installed in a given python environment along with the versions.

```
pip install > 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 can recreate the same environment using:

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

LAMBDA FUNCTIONS

Function created using an expression using 'lambda' keyword.

Syntax:

```
lambda arguements : 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) # retuns 6
```

JOIN METHOD (STRING)

Creates a string from iterable objects.

```
l = ["apple", "mango", "banana"]
result = ", and, ".join(l)
print (result)
```

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...)
```

Syntax:

```
"{} is a good {}".format ("shivu", "boy") #1.
"{} is a good {0}".format ("shivu", "boy") #2.

# Output for 1:
#shivu is a good boy

# Output for 2:
# boy is a good shivu
```

MAP, FILTER & REDUCE

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

Syntax:

```
map(function, input_list)
        # the function can be lambda function
```

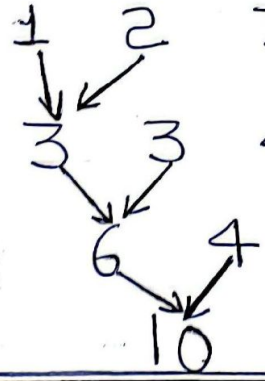
Filter creates a list of items for which the function returns true.

```
list(filter(function))
        # the function can be lambda function
```

Reduce applies a rolling computation to sequential pair elements.

```
from functools import reduce
val = reduce(function, list1)
        # the function can be lambda function.
```

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



Sequential Computation

MY-PYTHON-JOURNEY

SYMBIO-NOTEBOOK

PYTHON MASTERY ACHIEVED ACHIEVED
- BECOME ONE WITH THE CODEX

VENOM