Dt. 27 May '25
Pg.

Python

High level Programming language.

Has it works?

source code → compiler → Bytecode → virtual → output
machine.

- Source code is high level code with .py extension.
- Byte code is low level code with .pyc extension.

Note compiler → Byte code → virtual machine together forms Interpreter.

All of this process is automated.

Why python?

Powerful & simple.
Everything & Everywhere.
Huge community.
AI and ML.

Dt. _____

Pg. _____

Comments

It is a note in your code that does not get executed.
It is denoted by #.

Why Comments?

- Makes the code understandable.
- Increase readability
- Good practise

Types

i) Single line
denoted by #.
used for single lines.

ii) Multi line
denoted by triple quotes.
used for multiple lines.

Note: For long detailed explanation use comments above the code.

For short explanation use inline comments.

Dt. _____

Pg. _____

Print()

Built in python function that displays messages on the output screen.

Syntax

```
print("hi")
```

we can use double or single quotes.

Escape sequence

Their are a lot of characters that perform special operations.

\" : to add " to the code.
\' : to add ' to the code.
\\ : to add \ to the code.
\n : to add a new line.
\t : to add a horizontal space.

Note use triple quotes to print a text in same format.

Use case

- Helps in debugging.
- Increase readability.
- Show results

Dt. _____

Pg. _____

Variable

name you create to store a value.

Syntax

```
x = 10
name = "Devansh"
```

Note stored in the memory.

Use case

- updatable
- dynamic

Data

It can store different types of data.

- integer
- Text
- float
- boolean

Note Python allows a variable to be updated.
Their are different naming conventions.

- start with underscore or alphabets
- cannot have space.
- case sensitive
- no reserved words.

Dt. _____

Pg. _____

Input()

Built in function that helps to take input from the user.

Syntax

```
input("enter name":)
```

Note: It immediately discards the value unless stored in var.
By default it takes all the values as string.

```
name = input("Your name")
```

This stores the user input in name variable.

Data Types

These define the type of data stored in variables.

i) single value
These are primitive data Types.
- int
- float
- string
- bool

ii) Multiple value.
- list
- Tuple
- dictionary
- set

iii) No value - none.

Dt. _____

Pg. _____

Examples

a = None	none
a = 10	int
a = 3.15	float
a = "hello"	string
a = True	bool

Note Python automatically identify type of data.

Functions

independent block of code.

- function_name(value)

  eg. print("hello")

Methods

functions belonging to objects/classes.

- value.method_name()

  eg. 50.bit_length()

Dt. 1 June '25
Pg.

String functions

i. Creation and formatting

- str(obj)
  converts an object to string.
  str(123) → "123"

- format()
  insert variables into placeholders.
  "Hello, {}.format("Devansh")

- f" "
  Inline variable insertion (f string)
  f"Hello, {name}".

ii. Case Handling

- lower()
  converts all characters to lower case.
  Hello.lower() → "hello"

- upper()
  converts all characters to upper case
  Hello.upper() → 'HELLO'

- swapcase()
  swap the case of the letters.
  hELLo.swapcase() → 'HEllO'

Dt. _____

Pg. _____

- capitalize()
  convert first word to capital, rest lowercase.
  Hello world.capitalize() → Hello world.

- title()
  convert first letter to capital of each world.
  hello world.capitalize() → Hello World.

- casefold()
  like lower() but more aggressive.
  Hello.casefold() → hello

iii) Search and find.

- find()
  returns ~~of~~ index of first occurence
  returns -1 if not found.
  hello.find("e") → 1

- rfind()
  returns index of last occurence.
  returns -1 if not found.
  hello.rfind("l") → 3

- index()
  same as find() but raises error if not found.

- rindex()
  same as rfind but raises error if not found.

Dt. _____

Pg. _____

- count()
  cant has many times substring appears.
  hello.count("l") → 2

Note All these functions have an optional arguement start, end.

iv Type checking

All the functions return boolean value.

- isalpha()
  only letters.
- isdigit()
  only digits
- isalnum()
  numbers and letters
- isdecimal()
  only decimal numbers.
- islower()
  only lowercase
- isupper()
  only uppercase
- isspace()
  only whitespaces.

Dt. _____

Pg. _____

Modify strings

- replace()
  replace old substring with new.
  replace(old, new)
  apple.replace('p', 'b') → 'abble'

- strip()
  remove whitespace from both sides.
  ' hello '.strip() → 'hello'

- lstrip()
  remove whitespace from left.
  ' hello '.lstrip() → 'hello '

- rstrip()
  remove whitespace from right.
  ' hello '.rstrip() → ' hello'

Splitting and joining.

- split()
  splits into a list based on a seperator.
  return list of string.
  "devansh Maheshwari".split() → ["Devansh", "Maheshwari"]

if no seperator is given then whitespace is used.

split(sep, maxsplit)

Dt. _____

Pg. _____

- rsplit
  same as split but from right side.
  useful with maxsplit.
  "3-2-1".rsplit("-", 1) → ["3-2", "1"]

- partition()
  split string into 3 parts.
  before, seperator, after.
  returns a tuple.

  "devansh=python".partition → ("devansh", "=", "python").

- rpartition()
  same as partition but from right

- join()
  join a iterable of strings into single string.
  "-".join(["2025", "01", "03"]) → 2025-01-03.

vi) start/end check.

- startswith()
  check if string starts with substring.
  'Hello'.startswith('He') → True.

- endswith()
  check if string end with substring.
  'Hello'.endswith('H') → false.

Dt. 2 June'25
Pg.

Number functions

These are functions that no need to be imported.

- abs()
  return the absolute value of a number
  abs(-5) → 5

- pow(x, y)
  returns the power of a number.
  pow(2, 3) → 2^3 → 8

- round()
  rand of decimal upto n places.
  rand(3.1415, 2) → 3.14

- max()
  return largest value among arguements.
  max(2, 3, 10, 30) → 30

- min()
  return min values among argument.
  min(2, 3, 10, 30) → 2

- sum()
  returns the sum of iterable passed.
  sum([1, 2, 3]) → 6

- divmod()
  return tuple of quotient and remainder.
  divmod(10, 6) → (1, 4)

Dt. _____

Pg. _____

Their are functions that use math functions.

import math

- math.sqrt()
  return square root of a number.
  math.sqrt(4) → 2

- math.floor(x)
  return the floor of the decimal number.
  math.floor(3.15) → 3.

- math.ceil(x)
  return the ceiling of decimal number.
  math.ceil(3.15) → 4

- math.factorial(x)
  return factorial of a number.
  math.factorial(5) → 120

- math.gcd()
  return the greatest common divisor.
  math.gcd(12,8) → 4.

Dt. _____

Pg. _____

import random.

- random.randint(a, b)
  generate random integer between a, b.
  a, b both are included.

  random.randint(1, 4)
  → 1, 2, 3, 4

- random.random()
  return a random number between (0.0 and 1.0)
  1.0 is excluded.

  random.random() → 0.3

- random.uniform(a, b)
  return a float in a range.
  includes a and b both

  random.uniform(10.5, 20.5)
  → 11.6

Dt. _____

Pg. _____

Python List

It is a list of items with different data types under a single name.

Properties

- Ordered
  Have a defined order or index.
- Mutable
  These are changeable.
- Accesible
  can be accessed using index.
- Duplicates
  Allow duplicates.

Syntax

a = [ ] → empty list.

num = [1, 2, 3] → list of numbers

data = [1, "My", 3.14] → mixed data type.

nest = [[1, 2], [3, 4]] → nested list

Dt. _____

Pg. _____

Note These allow negative indexing starting from -1.

Accessing Elements

- indexing
  a[0] → first element
- negative indexing
  a[-1] → last element.
- slicing
  a[1:3] → element at index 1 and 2.
- step slice -
  a[::2] → every second element.

Eg fruits = ["apple", "banana", "mango"]

f[0] → apple

f[-1] → mango.

- updating
  fruits[0] = "cherry"
- Adding
  fruits.append("peach")
- Removing elements

Dt. _____

Pg. _____

i) remove by value
fruits.remove("banana")

ii) remove by index
fruits.pop(2)
remove element at index 2.

iii) remove last item
fruits.pop()

iv) delete by index
del fruits[0]

v) remove all
fruits.clear()
remove all the item.

Functions

- len()
count number of elements.
len(fruits) → 3

- append()
append one element at last of a list.
fruits.append("guava")

- insert()
insert item at a position.
insert(1, "guava")

Dt. _____
Pg. _____

- index()
gives the first index of the item.
index("guava")

- count()
return count of particular element.
count("guava") → 1

- sort()
sorts the list in place

- reverse()
reverse the list in place.

- copy()
return shallow copy of the list.
copy = fruits.copy()

List comprehension
provides a concise way to create lists.
replace need for loops.

To print list of squares.

```
square = [i**2 for i in range(5)]
print(square)
```

=> [0, 1, 4, 9, 16]

Dt. _____
Pg. _____

```
a=[1, 2]
b=[3,4]

print (a+b)
-> [1,2, 3,4]

print (a* 2)
=> [1, 2, 1, 2].
```

Python Tuple

collection of different data type variables under a single name. Denoted by parenthesis.

```
t1= (1, 4.5, "Hi")
```

Properties

- ordered
  maintain order of elements.
- immutable
  They are immutable. connot change once created.
- Duplicates.
  Allow duplicates.
- indexed
  can be accessed through index.

Dt. _____
Pg. _____

Creation

t1 = (1, 2, 3)
t1 = 1, 2, 3 → possible but not recommended.

Note Single element tuple can be created using comma.
t1 = (5) → integer
t1 = (5,) → tuple.

- indexing → t = (1, 2, 3)

t[1] → 2

- Negative index
t[-1] → 3

- slicing
t[:2] → (1, 2)

- concatenation
we use the '+' operator.

a = (1, 2)
b = (3, 4)
a + b → (1, 2, 3, 4)

- repitition
a * 3 → (1, 2, 1, 2, 1, 2)

Note Loops can be used to iterate.

Dt. _____

Pg. _____

Methods

- count()
  returns the cout of times x appears.
  count(1)

- index(x)
  return the index of first occurence.

Note used as keys of dictionaries.

unpacking
assigning values to variables from tuples directly.

a, b, c = (1, 2, 3)
∴ a=1, b=2, c=3

we use * to capture multiple values.

a, *b, c = (1, 2, 3, 4, 5)
a=1, b=[2, 3, 4], c=5

Note No. of elements in tuple must be equal to no. of variables.

Dt. _____

Pg. _____

Python Sets

collection of different data variables under one name.

Properties

- unorderd
  These dont have any order. Not indexed.
- mutable
  can be updated.
- No duplicates.
  Sets do not allow duplicate values.
- unindeed
  cannot be caccesed through indexes.
- iterable
  can be iterated using loops.
- Creation
  use curly brackets

Creation
use curly brackets.
a = {1, 2}.

S = set ([1, 2, 3]) → set constructor.

Dt. _____

Pg. _____

empty set can be created using set construction.

s = set() → set.
s = {} → dictionary

Note even if you store duplicate. You can not print.

a = {1, 2, 3, 3, 3}
print(a)
→ {1, 2, 3}.

Sets remove duplicate automatically.

s[0] → error.

Method And Operations

- Add.
adds one element to the set
a.add(5)

- update
used to add multiple elements
a.update([5, 6, 7])

- remove(x)
remove element from set.
raise error if not found.

Dt. _____

Pg. _____

- discard(x)
  removes element from the set.
  does nothing if not found.

- pop()
  randomly remove and return an element.

- clear()
  empties entire set.

Set operation like math

- Union
  return set containing all unique elements.
  A | B or A.union(B)

- intersection
  return elements common in both.
  A.intersection(B)

- Difference
  return element in A but not in B.
  A - B or A.difference(B)

- Symmetric difference.
  return elements in either set but not in both.
  A ^ B or A.symmetric_difference.

- Subset
  check if all element of A are in B.

Dt. _____
Pg. _____

return boolean value.
A.issubset(B)

- superset
check if A contain all elements of B or not
A.issuperset(B).

Python Dictionaries
It is a collection of key value pairs.

Properties

- key:value
data stored in key value pairs.

- Unique
keys should be unique - no duplicates allowed in them.

- Duplicates.
duplicates allowed in values.

- Mutable
we can change, add, delete items.

- Ordered.
ordered as of Python 3.7+

- immutable
keys should be of immutable type.

Dt. _____
Pg. _____

Creation

we use curly brackets.
person = {"name" : "Devansh", "age": 21}

empty dictionary
a = {}

using dict constructor.
person = dict(name = "Alice", age = 25)

Accessing elements

print(person["name"] → alice
print(person.get("age") → 25
print(person.get("salary")) → none

Note get ensures no error if not found.

Adding or updating

- person["city"] = "Delhi" → Adds
- person["age"] = 26 → update.

Removing items.

- person.pop("age")
removes key and return value.

Dt. _____

Pg. _____

- del person["age"]
  deletes key-value pair

- person.clear()
  empties dictionary.

Methods

- get
  return values or default if key doesn't exist.
  get(key, default)

- keys()
  return a list of keys.

- values()
  return list of values.

- items()
  return the list of (key, value) tuples.

- update()
  adds/updates from another dict.
  update(other_dict)

- pop()
  remove and return values of a key.

- popitem()
  removes last inserted key value pair.

Dt. _____
Pg. _____

- setdefault()
return value if key exists, else adds it
setdefault(key, default)

Note Keys must be of immutable type.

Condition Statement
conditionals allow your program to make decisions, execute block of code depending on whether condition is true or false.

- if statement
used to execute something only if condition is true.
ends with a colon.

```
if condition :
    #code.
```

- if-else statement
used when we have to execute two things based on the condition.

```
if condition :
    # code if true.
else:
    # code if false.
```

Note indentation is very important as it specifies scope.

Dt. _____
Pg. _____

- if-elif-else statements
This is used when we have multiple statements to execute.

```
if condition1:
    # code if condition1 true.
elif condition2:
    # code if condition2 true.
else:
    # code if all false.
```

Note: Only one condition gets executed.

- Nested if
we can use if condition inside another if.

```
if condition:
    if condition:
        # code
    # code
```

- Shorthand if else.
we can use this only we have one condition.

```
print("even") if x%2==0 else print("odd")
```

- Shorthand if
if condition: print("yes")

Dt. _____
Pg. _____

- Comparison Operators

These are used to compare the values.

→ ==, >=, <=, !=, >, <

- logical operators

and
return true if both are true.

or
return true if either is true.

not
if true then false, if false then true.

- identity operators.

is, is not
checks if two objects are same or not.
return boolean value.

in, not in
check for members in a iterable.

a in ['a', 'b', 'c'] → True.

Dt _____
Pg _____

Loops

loops are used to repetedly do a certain task.
either specific number of iterations or till condition is true.

Types

1. For

for loop is used when we know the number of iterations.
used upon iterables.

```
for item in iterable:
    # code.
```

- range() with for loop.

range is a built in function used to generate sequence of numbers.
it take three arguements.

range(start, end, step)

Note start and step are optional.

range(5) → 0, 1, 2, 3, 4

range(1, 5) → 1, 2, 3, 4

range(1, 5, 2) → 1, 3

Note Here end is excluded.

Dt. _____
Pg. _____

We do not need to update value of i in for loop.

- enumerate()
this gives index and value while looping through a sequence.

```
names = ['a', 'b', 'c']

for index, name in enumerate(names):
    print(index, name)
```

This returns a tuple.

enumerate(iterable, start)
→ Here start is 0 by default.

- zip()
used to combine two or more iterables element wise.

zip(iterable1, iterable2)

It returns a tuple.

```
names = ["Devansh", "Vidit", "Yash"]
ages = [22, 23, 24]

for name, age in zip(names, ages):
    print(name, age)
```

⇒ Devansh 22

Dt. _____
Pg. _____

Note if one iteroble is short in length then zip stops at it.

- Nested for loop.

```
for i in range(5):
    for j in range(5):
        print(j)
    print().
```

While loop
used when number of iterations is not known.

```
While :
    condition
```

Note we need to increment or decrement variable manually.

Eg. print from 0 to 5

```
i=0
while true:
    print(i)
    i+=1
```

loop control statements

- break
it is used to break the loop entirely.

- continue
it is used to skip an iteration.

Dt. _____

Pg. _____

- pass
  It is a placeholder.
  It does nothing.

Note: We can use conditional statements with loops.

Common Mistakes.

- Infinite while loop → make sure to update variable
- using for loop for non iterable.
- misplacing break / continue.

Dt. _____
Pg. _____

Functions

Block of code that perform a specific task.

Benifits
code reuse
Modularity
Easy debugging
Better organization.

definition
we use 'def' keyword to define a function.

function call
greet()

Types

i) user defined
defined by users.
def keyword is used.

ii) Built in functions.
These are predefined in python. standard library.
Eg print(), input()

Return
return statement is used to end a function and return a value.

Dt. _____

Pg. _____

Parameters

These are the values passed in the function definition.

```
def greet (a, b)
```

Arguements

These are the values passed in function call.

```
greet (4, 5)
```

Types

- positional/required arguements

The number and order of arguement should match to that of parameters.

eg
```
def help (a, b, c)
    return a+b+c
help (2, 3, 4)  → valid
help (2, 3)  → not valid
```

- default arguements

If value is passed in parameter then arguement can be missed.

eg
```
def help (a, b, c=10)
    return a+b+c
help (5, 20)  → valid
```

Dt. _____

Pg. _____

Note Passing value to the parameter is always right to left

def help(a, b=10, c) → not valid

def help (a=5, b=6, c) → not valid

Note: Default arguements are considered only if no value is passed in function call.

- Keyword arguements
order of arguements can be changed only if writting full names with values.

```
def interest (prin, time, rate)
    return
interest (time=2, print=1000, rate=1)
```

Multiple arguement types together.
We can combine multiple types together by some rules.

- positional arguement before other.
- cannot specify value for an arguement more than once.

Return
used to exit a function.
Send back a value to the caller.

```
def hi():
    return
```

Dt. _____

Pg. _____

Properties

- exits a function
- return a value of any data type.
- can return multiple values in python

```
def sum(a, b)
    return a+b, b-a, b*a
```

- can return list, dictionaries etc.
- if function don't have return, it return None by defaut.
- can return another function.

Dt. 3 June '25
Pg.

Exception Handling

error
Mistakes in code syntax, that is caught before execution.

Types

- Syntax error
These are compile time errors
Mistake in the python code.
Detected before program execution.

eg: if true → colon missing
print ("hi")

Common syntax error

wrong python syntax
incorrect indentation.

- Exceptions
These are runtime errors.
Occur when valid syntax fails during execution.

- Arithmetic error
zero division error
Overflow error

Dt. _____

Pg. _____

ii) Value and type error
Value error - Invalid value passed
type error - wrong data type used.

iii) lookup error
Index error - list index out of range
Key error - missing key

iv) File and I/O error
File not found error - File does not exist.
Permision error - Not enough permision.

v) Import error
ModeleNotfoundError - module not found.

vi) Name and Attribute error
Name error - variable not defined
Attribute error - Missing attribute or method.

vii) Memory error.
Memory error - out of memory.

Exception Handling
It is a way to deal with errors gracefully without crashing.

Syntax
try :
    risky code
except Exception type:
    handling code.

Dt. _____
Pg. _____

eg:
```
try :
    x = 10/0
except ZeroDivisionError:
    print ("cannot divide by zero")
```

Note we can catch multiple exceptions.

else And Finally Block.

else: executes if no exception occurs.
finally: Always executes.

eg:
```
try:
    x = int(input("enter a number"))
except ValueError:
    print("Enter a number")
else :
    print("you entered", x)
finally:
    print("End of program")
```

Raise
we can raise our own exception using raise keyword.

eg:
```
a = input("enter your age : ")
if a < 0 :
    raise ValueError:
        print("age cannot be negative")
print("Age is :", a)
```

Dt. _____

Pg. _____

Catching All exceptions

```
try:
    risky code
except Exception as e:
    print("error:", e)
```

Note: Dont overyse try, except
it makes code unreadable and buggy.

Flowchart

```
try:
    Attempt code
except:
    if error, handle here
else:
    No error? Do this
Finally:
    Always use it for cleanup.
```

Dt
Pg

File Handling

This helps us to create, read, write, append or delete files

opening files

open(): built in function that allows us to work with files.

eg: file = open(filename, mode)

filename

It is the name of the file. It can be text, csv etc

eg: file = open("Devansh.txt", mode)

mode

It is the mode in which we want to open the file.

There are different modes.

- r - read (default), file must exist
- w - write, create or truncate file
- a - Appends to existing file.
- x - creates file, fails if already exist
- b - binary mode
- t - text mode (default)

eg: file = open("devansh.txt", "r")

Reading files

- read

It is used to read full content of file.

Dt. _____
Pg. _____

```
f = open("data.txt", "r")
content = f.read()
print(content)
f.close()
```

Note: we can use size argument in read to read specific no. of characters.

eg: f.read(5) → reads 5 characters.

- readline()

This helps us to read one line at a time

```
f = open("data.txt", "r")
content = f.readline()
```

- readlines()

This reads all lines as a list

```
f = open("data.txt", "r")
content = f.readlines()
```

ii) Writing to files

- write()

This helps us to write to a file.
if opened in 'w' mode then overrides file.
if opened in 'a' mode then appends to file.

Dt. _____

Pg. _____

eg:
```
f = open("data.txt", "w")
f.write("Hello")
f.close()
```

This overrides the data.txt.

eg:
```
f = open("data.txt", "a")
f.write("Devansh")
f.close()
```

This appends to the data.txt.

- writelines()

This helps us to write multiple lines.

```
line = [line1, line2]
f = open("data.txt", "w")
f.writelines()
f.close()
```

iii) Appending

We can append data to a file using append mode.
It does not override the content.

```
f = open("data.txt", "a")
f.write("hello")
f.close()
```

Dt. _____

Pg. _____

iv Deleting a file.

We can delete a file using os module.

```
import os
os.remove(filename)
```

Note it is good to check file before deleting

```
if os.path.exists(filename):
    os.remove(filename)
else:
    print("file does not exist")
```

Seek And tell

- Seek(index)

This is used to move the cursor to a specific position.

```
fseek(0) → move cursor beginning
```

- tell

This method tells us the position of the cursor.

```
f.tell()
```

- With (context manager)

This features helps us to automatically close file.

```
with open(filename, mode) as f:
    content = f.read()
# automatically closed.
```

Dt. _____
Pg. _____

Working with binary files

```
with open("image.jpg", "rb") as f:
    data = f.read()
with open("copy.jpg", "wb") as f:
    f.write(data)
```

Modules

These are used to organize or structure code.

It is a file containing python code with .py extension that can define function, classes and variables.
These help in reusability, organization etc.

Creation

- create a file with .py extension.
- write your desired code in it
- import the module in your main file.

eg: mymodule.py → module

```
import mymodule
mymodule.function()
```

Types

- Built-in modules.

These are modules provided by python.
eg: math, sys etc.

Dt. _____
Pg. _____

- user defined modules.
  These are made by the users using .py extension.

- Third party modules.
  These are modules downloaded using pip.
  Eg: pandas, numpy etc.

Importing modules

- Basic import
```
import math
print(math.sqrt(4))
```

- import with alias.
```
import pandas as pd
```

- import all
```
from math import*
```

- import specific function.
```
from math import sqrt
```

Note: To check the path of the module we use
```
import sys
print(sys.path)
```

dir()
This stands for directory.
It is used to get all methods of a module.

Dt. _____

Pg. _____

```
import math
print(dir(math))
```

- help()

This function shows the documentation of a module.

```
import random
help(random)
```

Dt. 5th June '25
Pg.

Packages

OOPs

everything in python is an object.

- class
  Blueprint of object
  defines how an object behave.

point to remember.
name of class always in pascal case.
name of methods in snake_case.
'+' denotes public, '-' denotes private.
class holds two things data and methods.

syntax
```
class Car:
    variables.
    method
```

- object
  It is an instance of the class.

syntax
```
object = classname()
```

if car is an object then.

```
wagon = Car().
```

Dt. _____

Pg. _____

Note For Built in classes we create object literal.

Eg:

```
class ATm:
    def __init__(self):
        self.pin = ""
        self.balance = 0
```

- constructor.

It is a special method of a class.
It automatically executes the code inside it when object of class is created.
__init__ → constructor.

Magic methods

special methods with double underscore on both sides.
They cannot be called by object.
These are predefined.

self

it is the object of class itself.
only object can access methods and data of class.
it is required in python to give self as parameter.

Note There are a lot of magic methods provided.
We can use them to make our own datatype.

Dt. _____
Pg. _____

Instance variable
All variable created inside constructor.
These have different values for different objects.

- Encapsulation
It is used to hide data from users.
We can hide both variables and methods.
We just use double underscore before them.

eg: self.pin → can be seen.

self.__pin → hidden.

eg: Atm() → class.

What actually happens:
When ever python encounters a variable hidden then it converts it into this format.

self._classname__variable

ie. _Atm__pin

We can still access it as.
self._Atm__pin = " "

Important
hide data members.
provide methods to access them.
This way user can access them according to your logic

Dt. _____

Pg. _____

every data member can have two methods get and set.

- Reference Variable
  we can create an object by just 'Atm()' but the problem is we cannot use it afterwards as we did not store it in a variable.

  Atm() → not recommended
  sbi = Atm() → correct way.

∴ sbi is a variable pointing to the address of the object

This is called reference variable.

Note we can pass object as an argument
we can also return the object from the function.

Pass by reference
if you pass mutable data types then it affects original data types.
∴ You can yuse clonning or immutable data types.

Objects behave as mutable data type.

Note changes in the function can cause permanent changes outside the function.

Dt. _____

Pg. _____

- static variable

a variable which has same value for all objects always created outside constructor.

It is also called class variable.

we access static variables through class name.

eg: if counter is static variable then.

Atm.counter → correct

self.counter → incorrect

static variable do not require object to be called.

static method do not take self as argument.

we define these methods by

@staticmethod

Note When their are multiple classes they show two types of relationships

i) Aggregation - has-a

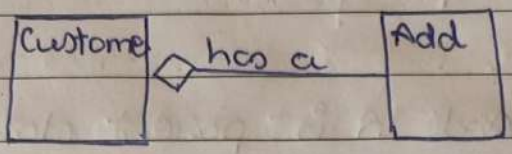
ii) Inheritance - Is-a

Dt. _____

Pg. _____

Aggregation

The classes that have has-a relationship.
These are shown in class diagram as.



- Inheritance

It is a feature of oop that helps in code reusability.
It is only in one direction.

Class A ← Class B ← class C

what do we inherit
All variables
All methods.
Constructor.

Note Private members are not inherited.

Syntax

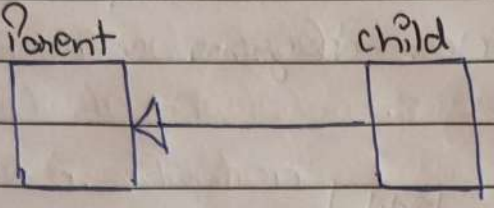
```
class A:        # parent child
    code

class B(A):     # child class.
    code.
```

Dt. _____

Pg. _____

Class Diagram



Note if B is child class and A is parent class. then.

- concept 1
  if B does not have constructor then it will search for constructor of class A.
  object can only be created of child class.

- concept 2
  child class cannot access hidden members of parent class.

- Polymorphism
  A function to take many forms.

- method overriding.
  Two functions, one in parent class and another in child class with same name and same arguements.

  when object is used to call the function then child class function overrides function of parent class.

- concept 3
  if child class has its own constructor, then constructor of parent is not called.

Dt. _____

Pg. _____

- Super

Super keyword helps us to jump to the parent class.
This does not work outside class.

We can access only two things through it:
parent class A: constructor
parent class Method.

Syntax
super().method , eg: super().bay(). [for method]

super().__init__(self,..) [for constructor.

Note Super should be first statement inside the constructor.

Types of Inheritance

i) single level
A inherited to B.

ii) multi-level
A ← B ← C ← D.
It has levels of inheritance.

iii) Hierarchal.
Single parent and multiple child.

iv) Multiple Inheritance.
Multiple parents and single child.

Dt. _____

Pg. _____

Syntax of Multiple inheritence.

class A (B, C, ...):

Note Suppose a situation.

1. Object of class A is created but it does not have a constructor.
2. But both the parent class have a constructor.
3. The class inherited first will be used for constructor.

∴ The class inherited first will have more priority.

This is called Method resolution order (MRO)

- Method overloading
  Two or more functions under same name in some class but different data types or different no. of arguements.

Note Does not exist in python.

- Operator overloading
  using operators differently for different purpose.
  This is done through magic methods.