~~index.html~~

1) Classes & objects
2) Encapsulation
3) Abstraction
4) Inheritance
5) Polymorphism

OOP → Object → Primitive datatype

1) int
2) float
3) char
4) Boolean
5) Double

} Simple data

why we need OOP ?

Mario Game :

1) player
2) position
3) Size, Health.

What is an object ? (instance of class)

templete for object.

class	object
Home Blueprint	Actual Home.

Class & object - Mario Example

Enemy

Class

size

position

move()

dead

objects

Enemy 1

Enemy 2

Enemy 3

1) Encapsulation :-

Encapsulation is grouping data with methods in a class.

Hiding data. Prevent direct access from outside.

Access through methods

Getter

Retrieve Info

Setter

Modify Info.

Check correctness

2) Abstraction

only show essential detail

Hide all other detail

keyboard Touchpad

Monitor USB

Hide Ram Hard disk

Battery CPU

Interface

Show outside class

keyboard

Mouse

monitor

Implementation

Hidden from outside

CPU

Hard

RAM

Mario

only get position

Code

3) Inheritance:

Allow classes to use data & method from another class.

~~Take all~~

Superclass

Subclass 1, Subclass 2, Subclass 3

Subclass 4 & 5 Subclass inherit from superclass

Inheritance - Access Modifiers.

1) Public
2) Private
3) Protected

4) Polymorphism : (Same time)

1) Dynamic (Both superclass & subclass have same method).
2) Static

If you write a subclass for object it will work

2) Static Polymorphism

Occurs during compile time

many methods with same name & different argument in the same class

Function overloading
Method overloading

argument

move (main. location)

OBJECT ORIENTED PROGRAMMING :

1) Classes and objects
2) Encapsulation
3) Abstraction
4) Inheritence
5) Polymorphism

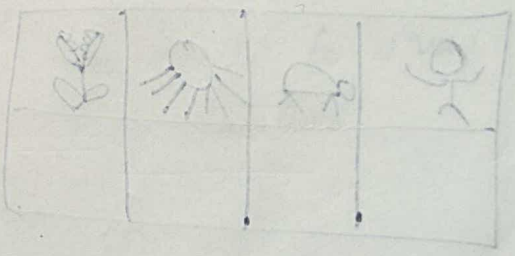
OOPS → Object

Primitive data type :

Complex data : Group similar primitive data together.

class & object :

class	Objects
size	Enemy 1
position	Enemy 2
move()	Enemy 3
dead	



class → Template (Home blueprint)

object → Actual home (used for object) (Template)

Object → object is an instance of class

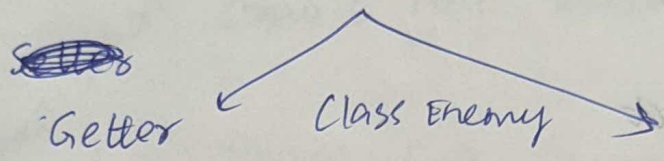
class → is a template for object

Encapsulation:

Encapsulation is grouping data with methods in a class.

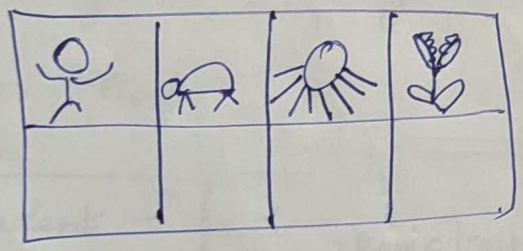
Hiding data. Prevent direct access from outside.

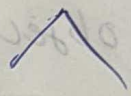
Access through methods



Retrieve Information
(Enemy position)

Modify Information
(Change Enemy position)





Check correct or not

class Enemy

class Player

class name

class Item

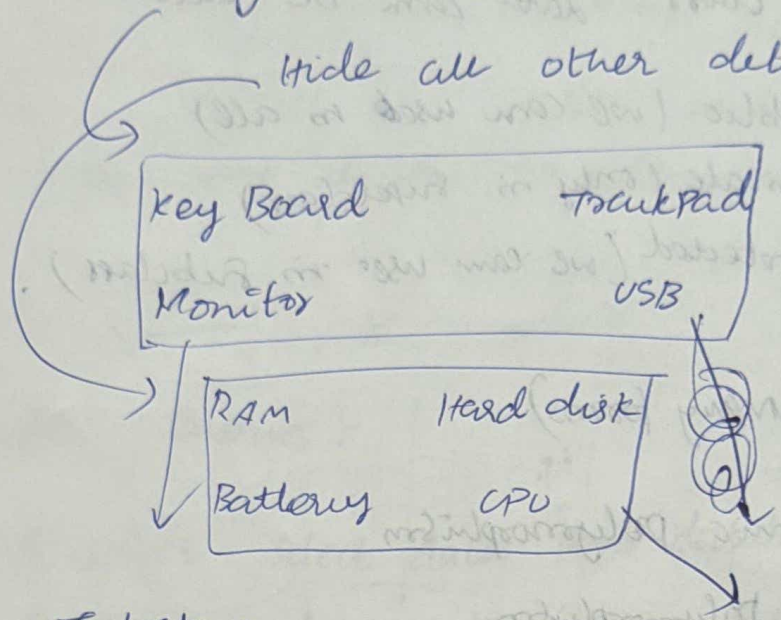
Class Background.

We can find error easily. Through encapsulation

Abstraction:

only show essential detail

Hide all other detail



Interface

Shown outside class

Implementation

Hidden from outside

class Enemy :

get position()

no need to show Implementation

Inheritance:

Allows classes to use data & method from another class

1) Single Inheritance
2) Multiple
3) Multilevel
4) Hierarchical
5) Hybrid

class enemy

class monkey class Octopus class dragon

Super Class

Sub class 1 Sub Class 2 Subclass 3

Subclass 4

Access modifiers :

change where class data can be accessed from.

1) Public (we can used in all)
2) private (only in superclass)
3) protected (we can use in subclass).

Polymorphism : (Many forms)

1) Dynamic polymorphism
2) Static Polymorphism

1) occurs during run time

Both superclass and subclass have same method

Class Enemy move()

Class Monkey

Class Octopus (move())

one more in Octopus

Static Polymorphism.

occurs during Complile-time

1) Many methods with same name & different arguments in the same class

called as method or function overloading

class - Enemy

move()

move(mario_location):

detects the location and have to move differ towards mario to attack

SQL Queries :

1. select - select data from database
2. From - which table we are pulling from
3. where - Condition we can specify a condition (filter away to match condition)
4. as = alias - Rename column or table with an alias
5. join - add two or more rows from different tables
6. And - Both conditions must be meant
7. Or - only one condition is enough
8. Limit - Limit rows returned
9. in - Specified values when we use "delete"
10. case - Return value of the row specified condition
11. Is Null - Row value as null value
12. Like - search for pattern in column
13. Commit - write transaction to database
14. Rollback - undo the transaction block

Method Overloading

table
database
column
rows

ESC

Object Oriented Programming with Python :-

1) Abstract class
2) Attributes & objects
3) polymorphism
4) Abstraction
5) Encapsulation
6) Method kinds

Methods → "def functions" inside class are Methods.

Formated string :-

f"A instance created : {name}" → formated string.

assert → defines wheather positive or negative or ranges the value.

Argument → ?

instances → ?

decorators → ?

Class → Design

Objects → Instances

```
Class Item:
    def __init__(self, name: str, price: float, quantity=0):
        assert price >= 0, f"Price {price} is not greater
                            than or equal to zero!"
        assert Quantity >= 0 f" Quantity {quantity} is not
                            greater on equal to zero! "

        self.name = name
        self.price = price
        self.Quantity = Quantity

    # calculation for price (function or Method).

    def calculate_total_price(self):
        return self.price * self.Quantity

item1 = item ("phone," 100, 1)
item2 = item ("Laptop", 1000, 3)

Print (item1.calculate_Total_price())
print (item2.calulate_Total_prue())
```

Next to 20% discount :-

Magic attribute :-

```
Print(item1.__dict__) # All attributes for class level
Print(item1.__dict__) # All attributes for Instance level
```

__repr__ → to show the attributes and instances

csv → comma separated value

x

```
class softwareengineer :
    # class attribute
  alias = "keyboard Magician"

def __init (self, name, age, level, salary) :
    self.name = name # instance attributes
    self.age = age
    self.level = level
    self.salary = salary

  # instance (object)
  sel = softwareengineer ("Max", 20, "Junior", 5000)
  print (sel.name, age)
```

1) instance can't be used in the whole class

2) But, class attribute can be used in whole class

Function in classes :- (method) or (instance method)

```
def add (a, b) :        → parameters
    return a+b

add (2,3)
  → Arguments
```

def __str__(self): Information of object

d under method

def __eq (self, other): Compare two objects

decorator → ?

@ staticmethod

Inheritence :-

one class takes the attributes and methods of other class. The newly created class is called as child class and pre-existing one is parent class.

inherit, override, extend.

super().__init__(name, age, Salary)

↳ used to override the inheritance. extends.

inherit, override, extends

class employee:

def __init__(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

def work(self):

print(f"{self.name} is working...")

```
class SoftwareEngineer(Employee):
    def __init__(self, name, age, salary, level):
        super().__init__(name, age, salary)
        self.level = level

    def work(self):
        print(f"{self.name} is coding...")

    def debug(self):
        print(f"{self.name} is debugging...")

class Designer(Employee):

    def work(self):
        print(f"{self.name} is designing...")

    def draw(self):
        print(f"{self.name} is drawing...")

se = SoftwareEngineer("Max", 25, 6000, "Junior")
se.work()

d = designer("Philipp", 27, 7000)
d.work()
```

Polymorphism :- (Many forms)

Polymorphism gives us a way to use a class exactly as a parent class, but still each child class keeps its method.

1) inheritance : child class (Base class)
(Parent class)

2) inherit, override, extend

3) super().__init__() when we change from Base class

4) Concept of Polymorphism.

Encapsulation :-

1) Mechanism of Hiding the data implementation.

2) instance methods are kept private.

3) Single underscore (_) to create private instance attribute

4) double underscore (__) for no attribute or completely private.

↳ you never see this in real often.

Single underscore (protected)

double underscore (private)

Bank details are valid

"Abstraction" is the natural extension of the "Encapsulation."

they don't care about a internal calculation of the salary. they only care about the output.

```
class SoftwareEngineer():
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self._salary = None # No need to define in parameters
        self._num_bugs_solved = 0

    def code(self):
        self._num_bugs_solved += 1

    # getter
    def get_salary(self):
        return self._salary

    # setter
    # check values, constraints enforcement, check details.
    def set_salary(self, base_value):
        self._salary = self._calculate_salary(base_value)

    def _calculate_salary(self, base_value):
        if self._num_bugs_solved < 100:
            return base_value
        if self._num_bugs_solved < 100:
            return base_value * 2
        return base_value * 3
```

```
se = SoftwareEngineer()
Print (se.age, se.name)

for i in range (70):
    se.code()

se.set_salary (6000)
Print (se.get_salary())
```

——— X ——— decorators

```
@property
def salary (self):
    return self.salary

@salary.setter
def salary (self, value):
    self.salary = value

@salary.deleter
def salary (self):
    del self.salary
```

——— X ←———

Getter & Setter are only way to access the private and protected values in code.

1) getter → @property
2) setter → @x.setter

OOP's PRINCIPLES :-

Inheritance :-

Inheritance is the process of by which one class takes on the attributes and methods of others. Newly formed classes are called child class, and the classes that child classes are derived from are called parent classes.

child classes inherit all of the parent's attributes and methods but can also extend and overrites attributes and methods that are unique to themselves.

Polymorphism :-

" Many shapes"

we can write a code that works on the superclass and it coill work with any subclass type as well

Gives us a way to use a class exactly like its parent but each child class keeps it's own method as they are.

Encapsulation :-

Encapsulation is the mechanism of hiding of data implementation.

Instances variables are kept private and accessor methods are made Public to achieve this with this, we restrict access to Public methods (getter / setter).

Instance methods can also kept private

Abstraction :-

Abstraction can be thought of as an natural extension of encapsulation. Applying abstraction means that each object should only expose a high-level mechanism for using it.

The mechanism should hide internal implementation details. It should only reveal operations relevant for the other objects.
