**OOPS CONCEPTS - POLYMORPHISM**

**POLYMORPHISM**

Poly means multi and Morphic means shapes in the world of Object Oriented Programming.

The shape is replaced with behaviour that is if we can have different types of behaviour or different based on the data of objects that the functions are dealing with then, that is Polymorphism

Different Ways of Polymorphism

1. Duck Typing
2. Duck Typing to do Dependency Injection
3. Operator Overloading
4. Method Overriding

**DUCK TYPING**

Duck Typing is not really a feature that a programming language offers, but it comes for free, if a programming language is dynamic so is the case in Python

def callTalk(obj)

obj.talk()

Here is a method called callTalk which takes an objects and this method is going to invoke any method on the object.

The Key here is that in Python we don not specify what type of object is this.

So the dynamic nature that side effect of this dynamic nature of passing parameters is nothing but Duck Typing.

**Code:**

class **Apple**:

def **products**(*self*):

print(*"APPLE PRODUCTS"*)

print(*"=============="*)

print(*"iPhone"*)

print(*"iMac"*)

print(*"iPad"*)

print(*"Air Pod"*)

print(*"iWatch"*)

class **Samsung**:

def **products**(*self*):

print(*"SAMSUNG PRODUCTS"*)

print(*"================"*)

print(*"Samsung Phone"*)

print(*"Smarth Watch"*)

print(*"Tab"*)

print(*"Laptop"*)

print(*"Hearing"*)

def **callProducts**(obj):

obj.products()

ap=Apple()

callProducts(ap)

print()

sm=Samsung()

callProducts(sm)

**Output:**

APPLE PRODUCTS

==============

iPhone

iMac

iPad

Air Pod

iWatch

SAMSUNG PRODUCTS

================

Samsung Phone

Smarth Watch

Tab

Laptop

Hearing

**DEPENDENCY INJECTION**

DI is nothing but simply injecting an object into another object as required.

In Python, using Duck Typing we can use the DI.

**Code:**

class **Flight**:

#airlines 🡪 the value is passed dynamically

def **\_\_init\_\_**(*self*, airlines):

#USES THE VALUE AS DEPENDENCY INEJCTSION

*self*.airlines=airlines

def **airlineName**(*self*):

*self*.airlines.flightName();

class **Emirates**:

def **flightName**(*self*):

print(*"Travelling in Emirates Airways......."*)

print()

class **BritishAirways**:

def **flightName**(*self*):

print(*"Travelling in British Airways......."*)

print()

class **Qatar**:

def **flightName**(*self*):

print(*"Travelling in Qatar Airways......."*)

print()

em=Emirates()

fl=Flight(em)

fl.airlineName()

ba=BritishAirways()

f2=Flight(ba)

f2.airlineName()

qa=Qatar()

f3=Flight(qa)

f3.airlineName()

**Output:**

Travelling in Emirates Airways.......

Travelling in British Airways.......

Travelling in Qatar Airways.......

**OPERATOR OVERLOADING (+)**

The plus operator in Python is overloaded to perform multiple things so we can call it polymorphic because if you use 2 numbs plus it will add them, if you use 2 strings it will append them.

If we use 2 lists then it will come out with single list using those 2 lists.

**Code:**

x=100

y=200

print(*"USING 2 NUMBERS WITH + SYMBOL"*)

print(x+y)

print()

s1 = *"Jazlyn "*

s2 = *"Jerrick"*

print(*"USING 2 STRINGS WITH + SYMBOL"*)

print(s1+s2)

print()

lst1=[*"Augusta"*, *"Jerrick"*, *"Jazlyn"*, *"Kumar"*]

lst2 = [48,10,16,48]

print(*"USING 2 LIST WITH + SYMBOL"*)

print(lst1+lst2)

print()

**Output:**

USING 2 NUMBERS WITH + SYMBOL

300

USING 2 STRINGS WITH + SYMBOL

Jazlyn Jerrick

USING 2 LIST WITH + SYMBOL

['Augusta', 'Jerrick', 'Jazlyn', 'Kumar', 48, 10, 16, 48]

**METHOD OVERRIDING**

Method Overriding is a way to achieve runtime polymorphism.