* OOPs Concepts *

Obj-oriented Programming	Procedural Programming
① Prob-solving approach & used where computa^n is done by using objects.	① Uses list of istr^n to do computa^n step-by-step.
② Makes development & maintenance easier.	② Not easy to maintain codes when proj lengthy.
③ Simulate real world entity. can solve real world problems easy.	③ No real world simulate. works on step-by-step instr^n divided into small parts cla functions.
④ Provide data hiding. secure than POP. Cannot access private data.	④ No data hiding, less secure
⑤ Ex- C++, Java, .Net, Python, C#.	⑤ Ex - C, Fortan, Pascal, VB.
⑥ Bottom-up approach.	⑥ Top-down approach.
⑦ Overloading possible.	⑦ Overloading not possible.
⑧ Data is more IMP than functions.	⑧ Fun^ more IMP than the data.
⑨ Suitable designing medium sized programs.	⑨ Suitable designing large & complex programs.

Python OOPs concepts:-

↳ By OO approach, can design program using classes & objects.

↳ Focus on writing reusable code.

↳ Widespread technique to solve prob by creating objects.

↳ Major principles of OOP System:

1. Class
2. Object
3. Method
4. Inheritance
5. Polymorphism
6. Data Abstraction
7. Encapsulation

- **Classes:-** Class is defined as coll^n of object.

↳ Logical entity that has some specific attributes & methods.

↳ Class contains blueprints / prototype from which objects are being created.

↳ Classes are created by 'class' keyword.

↳ Attributes are var that belong to a class.

↳ Attr are always public & can accessed using dot (.) operator.

Ex. class.attr.

- **Syntax –**

```
class   ClassName:
    # Stmt - 1
    .
    .
    .
    # Stmt - 2
```

- **Creating Empty Class:**

```
>>> class  Employee:
>>>     pass
```

- **Example:** An employee class contain attributes & methods like- email, name, age, salary etc.

- **Objects:-**

↳ object is entity that has state (Variables) & behaviour (Methods) associated with it.

↳ Example - Real-world entities like - pen, pencil, car, employee, account etc.

↳ Everything has objects & methods.

↳ All fun^n have built-in attr --doc--, which returns docstring defined in fun^n source code.

↳ After defining class, needs to create object to allocate the memory.

↳ Also cla Instance - copy of class with actual values.

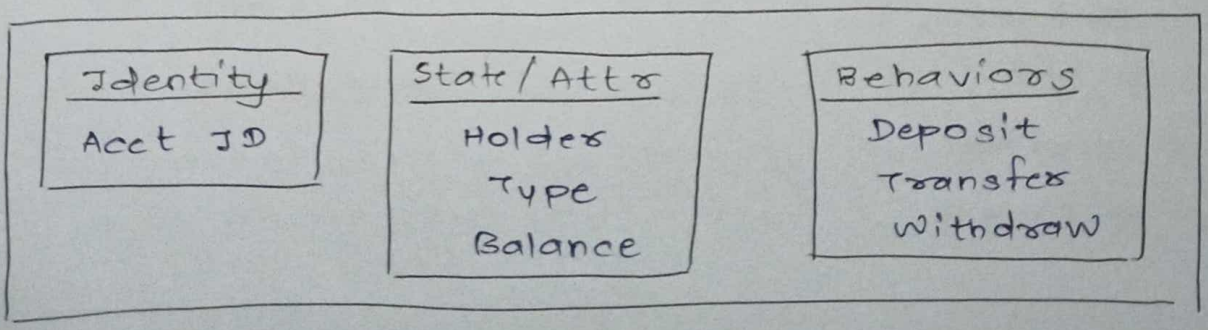
- **An Object consist of:**

① State: Repr by attr of an object. Reflect props of obj.

② Behavior: Repr by methods of an obj.
↳ Reflect obj response to other obj.

③ Identity: Give unique name to object & enable obj interⁿ.

- **Example:** For Account obj:



⊙ self Parameter :-

↳ When we are referencing the class meth/attr in that obj itself, then that can be refered by using 'self' keyword instead of obj-name.

↳ We can give any other name than 'self', but it is std practice.

↳ 'self' in Python, serves the same as 'this' keyword in Java.

```
>>> class Employee :
>>>     def __init__(self, id, name):
>>>         self.id = id
>>>         self.name = name
>>>     def show(self):
>>>         print(self.id, self.name)
```

self.name = name } These are params. So, self distinguish bet^n these names.

Attr for object

By this, we can refer to attr of the current obj.

```
>>>     def greet(self):
>>>         self.show()
```

(Must be the 1st one) Have to pass self as param to each meth.

can also call meth of same obj.

↳ self is similar to this ptr in C++ & this reference in Java.

The __init__() method:- can also refer as a constructor.

↳ Built-in method by the Python.

↳ present in all classes, always executed when the class is being executed.

↳ Used to assign values to obj props / do other operⁿ needs to be performed at object creation.

> • Note: The __init__() funⁿ / meth always called automatically, every time the class being used to create an object.

↳ Similar to constructors in C++ / Java.

↳ Initializes instance of the class.

• Example -

```
>>> class Account:
>>>     def __init__(self, acct:int, name:str)->None:
>>>         self.acct = acct
>>>         self.name = name
>>>         self.bank = 'BOI'
>>>         self.branch = 'Sarvd'
```

1st param must be always 'self'

Then, follows list of params.

Return type

Initialize attrs with values passed.

can also do initializⁿ with some default values.

↳ Can also invoke a meth.

> • Note:- Constructor overloading not allowed in Python.

- **Advantages of Constructors :-**

① **Obj Initialization** – Initialize objs of a class.
↳ Allow set default values to attrs & allow to initialize obj with some provided custom data.

② **Easy to implement** – Defined using __init__() meth very easily.

③ **Better Readability** – Improve readability by making clear, what & how values being initialized.

④ **Encapsulaⁿ** – Ensure correct & controlled obj initializaⁿ.

- **Disadvantages of Constructors :-**

① **Overloading not supported** – Does not support constructor overloading.

② **Limited functionality** – Limited in functionality compared to other lang constr.
Ex – No access modifiers like public, private & protected.

③ **May be unnecessary** – (In some cases), as attrs default values may be sufficient.
↳ Here, constr add unnecessary complexity.

⊙ Destructors in Python :-

↳ Destructors are called when object gets destroyed.

↳ Not needed as much as in C++, bcz Python has garbage collector that handle memory management automatically.

↳ __del__() method kla desto meth in Python.

↳ Called when all references to obj have been deleted i.e. when an obj is garbage collected.

- Syntax –

```
>>> def __del__(self):
>>>     # body of destructor
```

- Note :- Obj reference also deleted, when object goes out of reference or when prog ends.

① Use "del" keyword :-

```
>>> class Employee:
>>>     def __init__(self, name: str) -> None:
>>>         self.name = name
>>>     def __del__(self):
>>>         print('object deleted')
>>> emp1 = Employee('Sn')
>>> del emp1   # Automatically call the desto.
```

(constructor) (Destructor)

O/p → object deleted

- Note : Using 'del' keyword, when we deleted all the references of object, so desto invoked automatically.

Built-in Class Attributes in Python :-

↳ Provide us info about the class.

↳ Using dot (.) operator, access built-in class attr

Attributes	Description
--dict--	Dictionary containing class namespace.
--doc--	Documentation strnt present in the class. If not present, return None.
--name--	class name.
--module--	Module name, in which class defined. ↳ This attr is "--main--" in interactive mode.
--bases--	Possibly empty tuple containing base classes, in order of their occurrence in the base class list.
--setattr(key, val)-- str object	set the attributes, key (name of attr) & val as the value.
--getstate()--	Get object, which containes dictionary of keys as the attr names & their corresponding values.

◉ __str__() Function :-

↳ This fun^n controls what should be returned when the class obj is represented as a string.

```
>>> class Account:
>>>     def __init__(self, acct, name):
>>>         self.acct = acct
>>>         self.name = name
>>>     def __str__(self)->str:
>>>         return f'{self.acct) : {self.name}'
>>> acct1 = Account(1, 'snehal')
>>> print(acct1)
```

O/p → 1 : Snehal

```
>>> acct2 = Account(2, 'Kal')
>>> print(acct2)
```

O/p → 2 : Kal

◉ Delete Object properties :-

```
>>> acct1 = Account(1, 'Snehal')
>>> del acct1.acct
>>> print(acct1.__getstate__())
```

O/p → {'name' : 'Snehal'}

Built-in Class Functions:-

Function	Description
getattr(obj, name, default) OR obj.__getattribute__(name)	Access value of an attr from the object
obj.__getstate__()	Get instance state, i.e. values associated with all the attr of instance.
setattr(obj, name, ~~default~~ value) OR obj.__setattr__(name, val)	Set particular value to specific attr of an obj.
delattr(obj, name) OR obj.__delattr__(name) OR del obj.attr_name	Delete specific attr from the instance.
hasattr(obj, name)	check, if inst has an attr. Return True ⇒ if contains attr. otherwise False.

Class and Instance Variables :-

- **Instance variables** – specific to each instance.
  - ↳ Data, unique to each instance.
  - ↳ These values are assigned inside a const s/met with the self keyword.
    - ↳ Also refered as static variables.
- **Class Variables** – specific to the class only.
  - ↳ Attr & methods shared by all instances of class.
  - ↳ Variables whose value assigned in the class, outside of a constr / method.
  - ↳ These are independent of any object / inst & may be accessed th' use of class name.

```
>>> class Account:
>>>     count = 0                          } class / static variables
>>>     acct-type = 'Saving'
>>>     def __init__(self, acct:int, name:str) -> None:
>>>         self.acct = acct               } Instance variables
>>>         self.name = name
>>>     def __str__(self) -> str:
>>>         st = f'{Account.count}, {Account.acct-type}'   # class var refer by class name
>>>         inst = f'{self.acct}, {self.name}'
>>>         return '{st}, {inst}'          # Instance var refer by the Instance.
>>> acct1 = Account(11, 'Snehal')
>>> print(acct1)
```

O/p → (1) Saving, 11, Snehal

static variable / class var go on ↑ing

```
>>> acct2 = Account(13, 'Kal')
>>> print(acct2)
```

O/p → (2) Saving, 13, Kal

Advantages of Class/ static Variables :-

(1) Memory efficient - shared among all instances, save meory avoid mult copies.

(2) Shared State - All instances allowed to access & modify the same data.

(3) Easy to access - can access by class name, inst not needed. More convenient access & modify.

(4) Initialization - when class defined, ensure var has default value.

(5) Readability - clearly indicate data shared among all inst of class.

Disadvantages :-

(1) Inflexibility - Bcz shared, so cannot have diff values for diff inst.

(2) Hidden dependencies - can create hidden dependencies betn diff parts of code, diffi under & modify.

(3) Thread safety - Introduce race condin & synchroniza issues if not properly synchronized.

(4) Namespace Pollution - static var add to namesp of class, cause conflict & hard to maintain code.

(5) Testing - static var can make more diffi to write effective unit tests, as var state may affect behaviour of class & its methods.

Class method v/s Static method :-

- Method - Fun^n that is associated with an object

Class Method in Python:

↳ @classmethod decorator is built-in fun^n decorator that is an expr^n that gets evaluated after your fun^n is defined.

↳ Receive class as 1st arg, just like inst meth.

- Syntax -

```
class C(object):
    @classmethod
    def fun(cls, arg1, arg2, ....):
        .....
```

fun: Fun^n to be converted into cls meth.

returns: a class meth for fun^n.

↳ Meth bound to class & not the object.

↳ Have access to state of class as takes class param that points to class & not obj inst.

↳ Can modify class state that would apply across all class inst.

For ex: can modify a class var, that will be applicable to all inst.

Static Method in Python :-

↳ Does not receive 1st arg.

↳ Meth bound to class & not object.

↳ Can't access / modify class state.

↳ It is present in a class, bcz it makes sense for the meth to be present in class.

- Syntax:

```
class C(object):
    @staticmethod
    def fun(arg1, arg2, ....):
        .....
```

returns – static meth for fun° fun.

Instance Method in Python :-

↳ Bound to class inst & operate on the object (inst) variables.

↳ If use inst var inside methods, these methods cla inst methods.

↳ can modify object state.

↳ self keyword is 1st param to work with inst method & self refer to current object.

- Syntax:

```
class C(object):
    def fun(self, arg1, arg2, .....):
        .....
```

Action	Class	Static	Instance
Method call	Class & obj name.	cls & obj name.	Using obj / inst
Modification	Class behavior, reflects to the entire class i.e. all inst.	Perform task in isolation. No interaⁿ with class / inst methods.	Modify behavior of inst variables.
Attribute Access	can access the class variables.	can't access class & static var. can't change behavior of cls / inst.	can access class & inst variables.
Class Bound & Inst Bound	Bound to class. So, good to use by cls name.	Bound to the class. So, good to use by cls name.	Bound with obj, so can access using object.

⊙ Imp Notes :-

① Inst meth : Use self param to access class inst.

② Class meth : Don't require class inst.
↳ Use cls param instead of self param.

③ Static meth : Uses neither self nor cls.
↳ Act as regular funⁿ, but access th' class name.