OOPS Concept

Procedural Programming Language

- Structure Programming Language based on concept of Functions.
- Follows Top to Bottom approach
- Data can be changeable/modified – So less secure
- No Inheritance/ No Code reusability/ No Data Hiding
- Divides the whole code into parts called Functions.

Need of OOPS

- In order to build huge applications with repetitive functionality
- Apply same type of properties to multiple cases
- Protect the data – More security

Object Oriented Programming Language

- Programming model that deals with objects and its data and functionality.

Objects – a physical entity/an entity that exists

- Collection of Variables & functions. Ex: Lists, Dictionaries, Strings.
- Everything is an object in python and it has its own memory address.

Classes – a logical entity that creates objects/template/ blue print structure

- Attributes (Collection of Variable & Functions) defined inside class and accessed by objects.
- Also have methods where we can define/modify object's state or functionality.

Advantages:

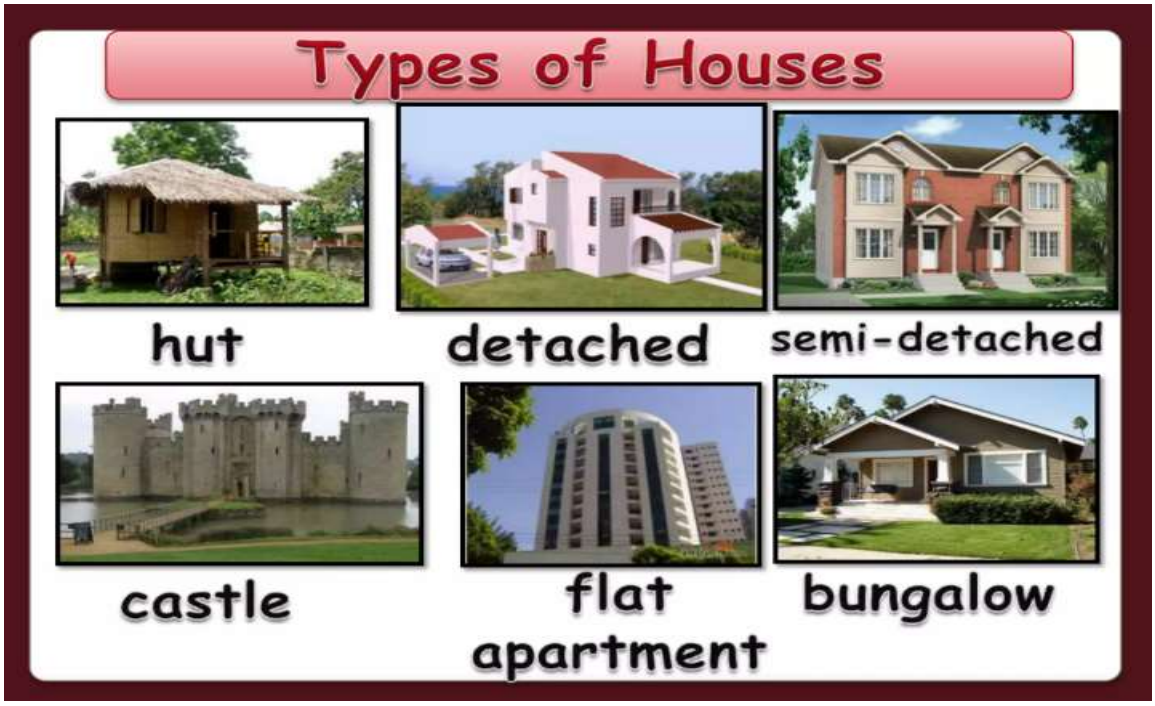
Bind Data members/ Grouping according to similarity.

Enable inheritance to utilize properties of other class.

Ability to reuse code to make program efficient.

Provides clean structure to the complex applications.

Real time example: Types of Houses



Creating a class:

```
class ClassName:
        pass
```

Creating Object of a class

```
class Employee:
   id = 10
   name = "Devansh"
   def display(self):
      print(self.id,self.name)
e1 = Employee()
e1.display()
```

self

- 1st argument to methods defined in class.
- Reference to class current instance/object.
- Used to access class variables.

Python Constructor

- __init__ is constructor of Python defined in form of method.
- This method is provided with “self” as first parameter to access attributes and methods of class.
- Can pass any number of arguments to this constructor.
- By default it is called when an object is created/class in instantiated.

Count no. of objects created

```
class Student:
        count = 0
  def __init__(self):
        Student.count += 1

s1=Student()
s2=Student()
s3=Student()
```

```
class Student:
  def __init__(self, name, dept, roll_no):
    self.name = name
    self.dept = dept
    self.roll_no = roll_no

  def show(self):
    print(self.name + " bearing roll number " + self.roll_no + " studying in " +
self.dept)

s1 = Student("Karthik", "ECE", "100")
s2 = Student("Arjun", "CSE", "101")
s1.show()
s2.show()

print(getattr(s1,"name"))
setattr(s1,"name","Rahul")
print(getattr(s1,"name"))
print(hasattr(s1,"roll_no"))
delattr(s1,"dept")
print(s1.roll_no)
```