## Object Oriented Programming (OOP):

- Object Oriented Programming, or OOP, is the method of structuring a program by bundling related properties and behaviors into individual objects.
- An object contains data.

### What is a Class?

Primitive data structures – like numbers, strings, and lists – are designed to represent simple things, such as the cost of something, the name of a poem, and a list of your favorite colors, respectively.

For example,

Let's say you wants to track employees in an organization. You need to store some basic information about each employee, such as their name, age and the year they started working.

One way to do this is the represent each employee as a list.

```
>>> Wasim = ["Wasim Akram", 55, "Bowler", 1984]
>>> Aamer = ["Aamer Sohail", 45, "Opening Batsman", 1990]
>>> Afridi = ["Shahid Afridi", "All-rounder", 1996]
```

There are many number of issues with this approach.

1. This approach require many line of codes, Let's say the organization have 1000 employee.
2. When you reference Wasim[0] after the Wasim is declared. Will you remember that the $0^{th}$ element of the list is the employee's name?
3. What if not every employee has the same number of elements in the list?

   In the Afridi list above, the age is missing, so Afridi[1] will return "All-rounder" instead of Shahid Afridi's age.

A great way to handle this type of code more manageable and more maintainable is to use classes.

***Classes*** *are used to create* ***user-defined data structures****. Classes also have special functions, called* ***methods****, that defines behaviors and actions that an object created from the class can perform with its data.*

*A* ***class*** *is a* ***blueprint*** *for how something should be defined. It doesn't actually provide any real content itself.*

*An* ***instance*** *is an object built from a class that contains real data.*

*Example:*

*A class is like a form or questionnaire, It defines the needed information. After you fill out the form, your specific copy is an instance of the class. It contains actual information relevant to you. You can fill out multiple copies of a form to create many different instances, but without the form as a guide, you would be lost, not knowing what information is required.*

*In short, before you can create individual instances of an object, you must first specify what is needed, by defining a class.*

- **Class (real-life Analogy):**

Example 1: Car Class Analogy

- Class = Blueprint of a car (like Toyota Corolla design).
- Object (Instance) = Actual car you buy and drive.
- Instance Attributes = Color, model, engine power (different for each car).
- Class Attribute = Company name (same for all cars).
- Instance Methods = Start car, accelerate, brake.
- Class Methods = Change company name for all cars.
- Static Methods = Utility functions like checking if a speed is overspeeding (doesn't depend on a specific car).

Example 2: StudentRecords Class Analogy

- Class = Blueprint for all students.
- Object (Instance) = Actual student (Ali, Ahsan, Sara).
- Instance Attributes = Name, roll_no, marks (unique for each student).
- Class Attribute = Institute name (same for all students).
- Instance Methods = Show details, check pass/fail.
- Class Methods = Change institute for all students.
- Static Methods = Utility like calculating grade (independent of a specific student).

## Class Definition Syntax:

```
class Employee:
    # body of class
    pass
```

Note: The convention for naming class in Python is to use **CamelCase** notation. For e.g. creating a class for **Saylani Mass IT employees**, would be written as **SaylaniMassEmployee.**

- Consider that you would like to store students records, so you need to define the class, you should start with the ***class*** keyword, which is followed by name of the class i.e. ***StudentsRecord*** and a colon.
- There are a number of properties of students such as name, roll no., course enrolled, institute location etc.
- To define the properties, or ***instance attributes,*** that all students objects must have, you need to define a special function or method called ***__init__*** - also known as **constructor.** This method is run every time a new Student object is created and tells Python what the initial values of the object's attributes.
- The first positional argument of **__init__()** *method is always a variable that reference the class instance.* This variable is universally named as ***self***. After ***self*** argument, you can specify any other arguments required to create an instance of the class.
- Function that belongs to a class are called ***instance methods.***
- While instance attributes are specific to each object, ***class attributes*** are the same for all object or instances.

```python
class StudentRecords:
    # class Attribute
    institute = "Saylani Mass IT Training Center"
    # instance Method to set initial values of instance attributes
    def __init__(self, name, roll_no, course):
        # Instance Attributes
        self.name = name
        self.roll_no = roll_no
        self.course = course
```

```
StudentRecords()
Traceback (most recent call last):
  File "<pyshell#124>", line 1, in <module>
    StudentRecords()
TypeError: StudentRecords.__init__() missing 3 required positional arguments: 'n
ame', 'roll_no', and 'course'
StudentRecords("Taha", 40564, "AI and Data Science")
<__main__.StudentRecords object at 0x0000022B1B54B520>
StudentRecords("Saqib", 40565, "AI and Data Science")
<__main__.StudentRecords object at 0x0000022B1B54B040>
```

```
student1 = StudentRecords("Huzaifa", 40570, "AI and Data Science")
student2 = StudentRecords("Saad", 50601, "Graphic Designing")

student1 == student2
False
# Note: Even though the student1 and student2 are both instances of
# the StudentRecords class and have the same attributes and methods
# but both are two distinct objects in memory



student3 = StudentRecords("Saad", 50601, "Graphic Designing")
student4 = StudentRecords("Saad", 50601, "Graphic Designing")

student3 == student4
False

type(student1)
<class '__main__.StudentRecords'>
type(student2)
<class '__main__.StudentRecords'>
```

After the ***StudentRecords*** instances are created, we can access their class or instance/object attributes by using ***dot notation.***

```
student1.name
'Huzaifa'
student1.roll_no
40570
student1.course
'AI and Data Science'
student1.institute
'Saylani Mass IT Training Center'
student2.institute
'Saylani Mass IT Training Center'

student1.institute == student2.institute
True
```

Both instance and class attributes can be modified dynamically:

```
# change instance attribute
student1.name = "owais"
# change class attribute
student1.institute = "Hajyani Campus"

student1.institute
'Hajyani Campus'
student2.institute
'Saylani Mass IT Training Center'
student1.institute == student2.institute
False
```

## Instance Methods:

```
# Tutorial on class, object and methods


class StudentRecords:
    # class attribute
    institute = "Saylani Mass IT"

    def __init__(self, name, roll_no, course):
        # instance attributes
        self.name = name
        self.roll_no = roll_no
        self.course = course

    # instance method
    def display_info(self):
        return f"{self.name} is enrolled in {self.course} course "

    # instance method
    def promote(self):
        return f"{self.name} is promoted to Advanced level"
```

```
std1 = StudentRecords("Tahir", 40555, "AI and Data Science")
std2 = StudentRecords("Osama", 40560, "Graphic Designing")

std1.display_info()
'Tahir is enrolled in AI and Data Science course '
std1
<__main__.StudentRecords object at 0x0000021553ACAB00>

std2.display_info()
'Osama is enrolled in Graphic Designing course '
std2
<__main__.StudentRecords object at 0x0000021553ACBAC0>
```

## __str__ method (dunder method)

```
class StudentRecords:
    # class attribute
    institute = "Saylani Mass IT"

    def __init__(self, name, roll_no, course):
        # instance attributes
        self.name = name
        self.roll_no = roll_no
        self.course = course

    # instance method
    def __str__(self):   # previous method name is display_info
        return f"{self.name} is enrolled in {self.course} course "

    # instance method
    def promote(self):
        return f"{self.name} is promoted to Advanced level"
```

- **Classes, Instances, Methods (Tasks)**

❑ Create a Car class with two instance attributes: i.e. color, which stores the car's color as string, and mileage, which stores the number of miles on the car as an integer. Then instantiate two Car Objects – a blue car with 20,000 miles, and a red car with 40,000 miles, and print out their colors and mileage. Your output should look like the following:

***The blue car has 20,000 miles***
***The red car has 40,000 miles***

❑ Modify the Car class with an instance method called drive() that takes a number as an argument and add that number to the mileage attribute.
*Note: Test that your solution works by instantiating a car with 0 miles, then call drive(100) and print the mileage attribute to check that it is set to 100*