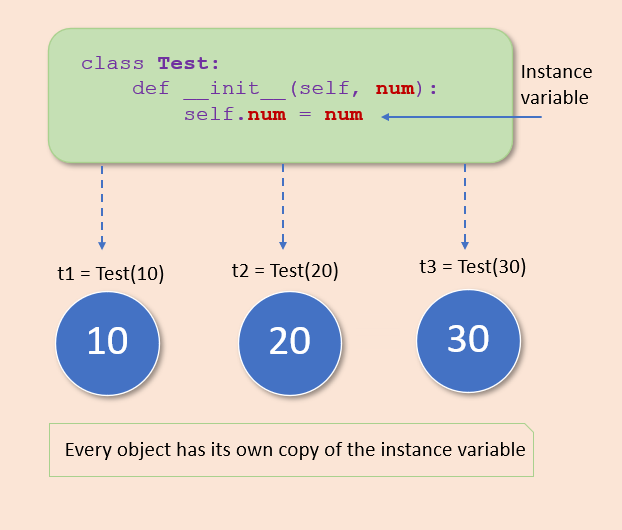
## Instance Variable in Python

**If the value of a variable varies from object to object, then such variables are called instance variables**. For every object, a separate copy of the instance variable will be created.

Instance variables are not shared by objects. Every object has its own copy of the instance attribute. This means that for each object of a class, the instance variable value is different.



Instance variables are declared inside a method using the self keyword. We use a constructor to define and initialize the instance variables.

Example 4:

# In this example, we are creating two instance variable name and age in the Student class.

class Student:

    # constructor

    def \_\_init\_\_(self, name, age):

        # Instance variable

        self.name = name

        self.age = age

# create first object

s1 = Student("Jessa", 20)

# access instance variable

print('Object 1')

print('Name:', s1.name)

print('Age:', s1.age)

# create second object

s2= Student("Kelly", 10)

# access instance variable

print('Object 2')

print('Name:', s2.name)

print('Age:', s2.age)

We can modify the value of the instance variable and assign a new value to it using the object reference.

**Note**: When you change the instance variable’s values of one object, the changes will not be reflected in the remaining objects because every object maintains a separate copy of the instance variable.

Example 5:

# Modify Values of Instance Variables

class Student:

    # constructor

    def \_\_init\_\_(self, name, age):

        # Instance variable

        self.name = name

        self.age = age

# create object

stud = Student("Jessa", 20)

print('Before')

print('Name:', stud.name, 'Age:', stud.age)

# modify instance variable

stud.name = 'Emma'

stud.age = 15

print('After')

print('Name:', stud.name, 'Age:', stud.age)

Example 6:

# Access instance variable in the instance method

class Student:

    # constructor

    def \_\_init\_\_(self, name, age):

        # Instance variable

        self.name = name

        self.age = age

    # instance method access instance variable

    def show(self):

        print('Name:', stud.name, 'Age:', stud.age)

# create object

stud = Student("Jessica", 20)

# call instance method

stud.show()

Example 7:

**getattr**(Object, 'instance\_variable')

Pass the object reference and instance variable name to the getattr() method to get the value of an instance variable.

# Access instance variable using getattr()

class Student:

    # constructor

    def \_\_init\_\_(self, name, age):

        # Instance variable

        self.name = name

        self.age = age

# create object

stud = Student("Jessa", 20)

# Use getattr instead of stud.name

print('Name:', getattr(stud, 'name'))

print('Age:', getattr(stud, 'age'))

Example 8:

# Delete Instance Variable

# In Python, we use the del statement and delattr() function to delete the attribute of an object. Both of them do the same thing.

# Using the del statement

class Student:

    def \_\_init\_\_(self, roll\_no, name):

        # Instance variable

        self.roll\_no = roll\_no

        self.name = name

# create object

s1 = Student(10, 'Jessa')

print(s1.roll\_no, s1.name)

# del name

del s1.name

# Try to access name variable

print(s1.name)

Example 9:

**delattr() function**

The delattr() function is used to delete the named attribute from the object with the prior permission of the object. Use the following syntax.

delattr(object, name)

class Student:

    def \_\_init\_\_(self, roll\_no, name):

        # Instance variable

        self.roll\_no = roll\_no

        self.name = name

    def show(self):

        print(self.roll\_no, self.name)

s1 = Student(10, 'Jessa')

s1.show()

# delete instance variable using delattr()

delattr(s1, 'roll\_no')

s1.show()

Example 10:

In this example, the engine is an instance variable of the Vehicle class. We inherited a Vehicle class to access its instance variables in Car class

# Access Instance Variable From Super Class

class Vehicle:

    def \_\_init\_\_(self):

        self.engine = '1500cc'

class Car(Vehicle):

    def \_\_init\_\_(self, max\_speed):

        # call parent class constructor

        super().\_\_init\_\_()

        self.max\_speed = max\_speed

    def display(self):

        # access parent class instance variables 'engine'

        print("Engine:", self.engine)

        print("Max Speed:", self.max\_speed)

# Object of car

car = Car(240)

car.display()

Example 11:

We can get the list of all the instance variables the object has. Use the \_\_dict\_\_ function of an object to get all instance variables along with their value.

The \_\_dict\_\_ function returns a [dictionary](https://pynative.com/python-dictionaries/) that contains variable name as a key and variable value as a value

# List all Instance Variables of a Object

class Student:

    def \_\_init\_\_(self, roll\_no, name):

        # Instance variable

        self.roll\_no = roll\_no

        self.name = name

s1 = Student(10, 'Jessica')

print('Instance variable object has')

print(s1.\_\_dict\_\_)

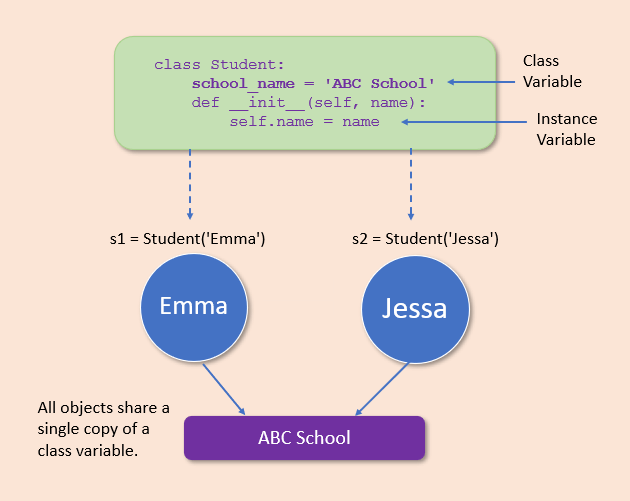
# Get each instance variable

for key\_value in s1.\_\_dict\_\_.items():

    print(key\_value[0], '=', key\_value[1])

# Python Class Variables

* **Instance variables**: If the value of a variable varies from object to object, then such variables are called instance variables.
* **Class Variables**: A class variable is a variable that is declared inside of class, but outside of any instance method or \_\_init\_\_() method.
* If the **value of a variable is not varied from object to object**, such types of variables are called class variables or static variables.
* Class variables are **shared by all instances of a class**. Unlike instance variable, the value of a class variable is not varied from object to object,



Example 12:

# Creating class variable

class Student:

    # Class variable

    school\_name = 'ABC School '

    def \_\_init\_\_(self, name, roll\_no):

        self.name = name

        self.roll\_no = roll\_no

# create first object

s1 = Student('Emma', 10)

print(s1.name, s1.roll\_no, Student.school\_name)

# access class variable

# create second object

s2 = Student('Jessa', 20)

# access class variable

print(s2.name, s2.roll\_no, Student.school\_name)

NOTE:

* We can access the class variable in the following places
* Access inside the constructor by using either self parameter or class name.
* Access class variable inside instance method by using either self of class name
* Access from outside of class by using either object reference or class name.

Example 13:

class Student:

    # Class variable

    school\_name = 'ABC School '

    # constructor

    def \_\_init\_\_(self, name):

        self.name = name

        # access class variable inside constructor using self

        print(self.school\_name)

        # access using class name

        print(Student.school\_name)

# create Object

s1 = Student('Emma')

Example 14:

# Access Class Variable in Instance method and outside class

class Student:

    # Class variable

    school\_name = 'ABC School '

    # constructor

    def \_\_init\_\_(self, name, roll\_no):

        self.name = name

        self.roll\_no = roll\_no

    # Instance method

    def show(self):

        print('Inside instance method')

        # access using self

        print(self.name, self.roll\_no, self.school\_name)

        # access using class name

        print(Student.school\_name)

# create Object

s1 = Student('Emma', 10)

s1.show()

print('Outside class')

# access class variable outside class

# access using object reference

print(s1.school\_name)

# access using class name

print(Student.school\_name)

Example 15:

# Modify Class Variables

class Student:

    # Class variable

    school\_name = 'ABC School '

    # constructor

    def \_\_init\_\_(self, name, roll\_no):

        self.name = name

        self.roll\_no = roll\_no

    # Instance method

    def show(self):

        print(self.name, self.roll\_no, Student.school\_name)

# create Object

s1 = Student('Emma', 10)

print('Before')

s1.show()

# Modify class variable

Student.school\_name = 'XYZ School'

print('After')

s1.show()

# Python Class Method vs. Static Method vs. Instance Method

* Instance method performs a set of actions on the data/value provided by the instance variables. If we use instance variables inside a method, such methods are called instance methods.
* Class method is method that is called on the class itself, not on a specific object instance. Therefore, it belongs to a class level, and all class instances share a class method.
* Static method is a general utility method that performs a task in isolation. This method doesn’t have access to the instance and class variable.

## Class Method in Python

Class methods are methods that are called on the class itself, not on a specific object instance. Therefore, it belongs to a class level, and all class instances share a class method.

* **A class method is bound to the class** and not the object of the class. It can access only class variables.
* It can modify the class state by changing the value of a class variable that would apply across all the class objects.

Any method we create in a class will automatically be created as an instance method. We must explicitly tell Python that it is a class method using the @classmethod decorator or classmethod() function.

Class methods are defined inside a class, and it is pretty similar to defining a regular [function](https://pynative.com/python-functions/).

Like, inside an instance method, we use the self keyword to access or modify the instance variables. Same inside the class method, we use the cls keyword as a first parameter to access class variables. Therefore the class method gives us control of changing the class state.

Example 16:

# Create Class Method Using @classmethod Decorator

from datetime import date

class Student:

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

    @classmethod

    def calculate\_age(cls, name, birth\_year):

        # calculate age an set it as a age

        # return new object

        return cls(name, date.today().year - birth\_year)

    def show(self):

        print(self.name + "'s age is: " + str(self.age))

obj1 = Student('Jessica', 20)

obj1.show()

# create new object using the factory method

obj2 = Student.calculate\_age("Joy", 1995)

obj2.show()

Example 17:

# Create Class Method Using classmethod() function

class School:

    # class variable

    name = 'ABC School'

    # class method

    def school\_name(cls):

        print('School Name is :', cls.name)

# create class method

School.school\_name = classmethod(School.school\_name)

# call class method

School.school\_name()

Example 18:

# Access Class Variables in Class Methods

class Student:

    school\_name = 'ABC School'

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

    @classmethod

    def change\_school(cls, school\_name):

        # class\_name.class\_variable

        cls.school\_name = school\_name

    # instance method

    def show(self):

        print(self.name, self.age, 'School:', Student.school\_name)

jessa = Student('Jessa', 20)

jessa.show()

# change school\_name

Student.change\_school('XYZ School')

jessa.show()

Example 19:

# Class Method in Inheritance

class Vehicle:

    brand\_name = 'BMW'

    def \_\_init\_\_(self, name, price):

        self.name = name

        self.price = price

    @classmethod

    def from\_price(cls, name, price):

        # ind\_price = dollar \* 76

        # create new Vehicle object

        return cls(name, (price \* 75))

    def show(self):

        print(self.name, self.price)

class Car(Vehicle):

    def average(self, distance, fuel\_used):

        mileage = distance / fuel\_used

        print(self.name, 'Mileage', mileage)

bmw\_us = Car('BMW X5', 65000)

bmw\_us.show()

# class method of parent class is available to child class

# this will return the object of calling class

bmw\_ind = Car.from\_price('BMW X5', 65000)

bmw\_ind.show()

# check type

print(type(bmw\_ind))