# Python Instance Methods Explained With Examples

## What is Instance Methods in Python

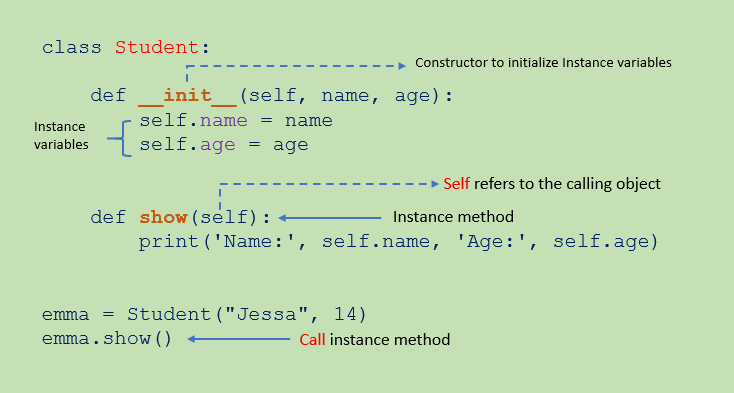
If we use instance variables inside a method, such methods are called instance methods. The instance method performs a set of actions on the data/value provided by the instance variables.

A instance method is bound to the [object](https://pynative.com/python-classes-and-objects/) of the class.

It can access or modify the object state by changing the value of a instance variables

When we create a class in Python, instance methods are used regularly. To work with an instance method, we use the self keyword. We use the self keyword as the first parameter to a method. The self refers to the current object.

Any method we create in a class will automatically be created as an instance method unless we explicitly tell Python that it is a class or static method.

Instance method in Python

## Define Instance Method

[Instance variables](https://pynative.com/python-instance-variables/) are not shared between objects. Instead, every object has its copy of the instance attribute. Using the instance method, we can access or modify the calling object’s attributes.

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

Use the def keyword to define an instance method in Python.

Use self as the first parameter in the instance method when defining it. The self parameter refers to the current object.

## Using the self parameter to access or modify the current object attributes.

You may use a [variable](https://pynative.com/python-variables/) named differently for self, but it is discouraged since self is the recommended convention in Python.

Let’s see the example to create an instance method show() in the Student class to display the student details.

Example:

class Student:

# constructor

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

# Instance variable

self.name = name

self.age = age

#### # instance method to access instance variable

def show(self):

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

Calling An Instance Method

We use an object and dot (.) operator to execute the block of code or action defined in the instance method.

First, create instance variables name and age in the Student class.

Next, create an instance method display() to print student name and age.

Next, create object of a Student class to call the instance method.

et’s see how to call an instance method show() to access the student object details such as name and age.

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:', self.name, 'Age:', self.age)

# create first object

print('First Student')

emma = Student("Jessa", 14)

# call instance method

emma.show()

# create second object

print('Second Student')

kelly = Student("Kelly", 16)

# call instance method

kelly.show()

### Output:

First Student

Name: Jessa Age: 14

Second Student

Name: Kelly Age: 16

Note:

Inside any instance method, we can use self to access any data or method that reside in our class. We are unable to access it without a self parameter.

An instance method can freely access attributes and even modify the value of attributes of an object by using the self parameter.

By Using self.\_\_class\_\_ attribute we can access the class attributes and change the class state. Therefore instance method gives us control of changing the object as well as the class state.

Modify Instance Variables inside Instance Method

Let’s create the instance method update() method to modify the student age and roll number when student data details change.

class Student:

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

# Instance variable

self.roll\_no = roll\_no

self.name = name

self.age = age

# instance method access instance variable

def show(self):

print('Roll Number:', self.roll\_no, 'Name:', self.name, 'Age:', self.age)

# instance method to modify instance variable

def update(self, roll\_number, age):

self.roll\_no = roll\_number

self.age = age

# create object

print('class VIII')

stud = Student(20, "Emma", 14)

# call instance method

stud.show()

# Modify instance variables

print('class IX')

stud.update(35, 15)

stud.show()

Output:

class VIII

Roll Number: 20 Name: Emma Age: 14

class IX

Roll Number: 35 Name: Emma Age: 15

Create Instance Variables in Instance Method

Till the time we used [constructor](https://pynative.com/python-constructors/) to create instance attributes. But, instance attributes are not specific only to the \_\_init\_\_() method; they can be defined elsewhere in the class. So, let’s see how to create an instance variable inside the method.

Example:

class Student:

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

# Instance variable

self.roll\_no = roll\_no

self.name = name

self.age = age

# instance method to add instance variable

def add\_marks(self, marks):

# add new attribute to current object

self.marks = marks

# create object

stud = Student(20, "Emma", 14)

# call instance method

stud.add\_marks(75)

# display object

print('Roll Number:', stud.roll\_no, 'Name:', stud.name, 'Age:', stud.age, 'Marks:', stud.marks)

Output:

Roll Number: 20 Name: Emma Age: 14 Marks: 75

Dynamically Add Instance Method to a Object

Usually, we add methods to a class body when defining a class. However, Python is a dynamic language that allows us to add or delete instance methods attime. Therefore, it is helpful in the following scenarios.

When class is in a different file, and you don’t have access to modify the class structure

You wanted to extend the class functionality without changing its basic structure because many systems use the same structure.

Let’s see how to add an instance method in the Student class attime.

Example:

We should add a method to the object, so other instances don’t have access to that method. We use the types module’s MethodType() to add a method to an object. Below is the simplest way to method to an object.

import types

class Student:

# constructor

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

self.name = name

self.age = age

# instance method

def show(self):

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

# create new method

def welcome(self):

print("Hello", self.name, "Welcome to Class IX")

# create object

s1 = Student("Jessa", 15)

# Add instance method to object

s1.welcome = types.MethodType(welcome, s1)

s1.show()

# call newly added method

s1.welcome()

Output:

Name: Jessa Age: 15  
Hello Jessa Welcome to Class IX

Dynamically Delete Instance Methods

We can dynamically delete the instance method from the class. In Python, there are two ways to delete method:

By using the del operator

By using delattr() method

By using the del operator

The del operator removes the instance method added by class.

Example:

In this example, we will delete an instance method named percentage() from a Student class. If you try to access it after removing it, you’ll get an Attribute Error.

class Student:

# constructor

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

self.name = name

self.age = age

# instance method

def show(self):

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

# instance method

def percentage(self, sub1, sub2):

print('Percentage:', (sub1 + sub2) / 2)

emma = Student('Emma', 14)

emma.show()

emma.percentage(67, 62)

# Delete the method from class using del operator

del emma.percentage

# Again calling percentage() method

# It will raise error

emma.percentage(67, 62)

Output:

Name: Emma Age: 14

Percentage: 64.5

File "/demos/oop/delete\_method.py", line 21, in <module>

del emma.percentage

AttributeError: percentage

By using the delattr() method

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

delattr(object, name)

object: the object whose attribute we want to delete.

name: the name of the instance method you want to delete from the object.

Example:

In this example, we will delete an instance method named percentage() from a Student class.

emma = Student('Emma', 14)

emma.show()

emma.percentage(67, 62)

# delete instance method percentage() using delattr()

delattr(emma, 'percentage')

emma.show()

# Again calling percentage() method

# It will raise error

emma.percentage(67, 62)