Access Modifiers

In this lesson, you will learn about the private, public, and protected data members in Python.

In Python, we can impose access restrictions on different data members and member functions. The restrictions are specified through **access modifiers**. Access modifiers are tags we can associate with each member to define which parts of the program can access it directly.

There are two types of access modifiers in Python. Let's take a look at them one by one.

Public Attributes

> **Public attributes** are those that be can be accessed inside the class and outside the class.

Technically in Python, **all** methods and properties in a class are publicly available by default. If we want to suggest that a method should not be used publicly, we have to declare it as private explicitly.

Below is an example to implement public attributes:

```python
class Employee:
    def __init__(self, ID, salary):
        # all properties are public
        self.ID = ID
        self.salary = salary

    def displayID(self):
        print("ID:", self.ID)


Steve = Employee(3789, 2500)
Steve.displayID()
print(Steve.salary)
```

In the code above, the properties `ID` and `salary`, and the method `displayID()` are *public* as they can be accessed in the class as well as the outside the class.
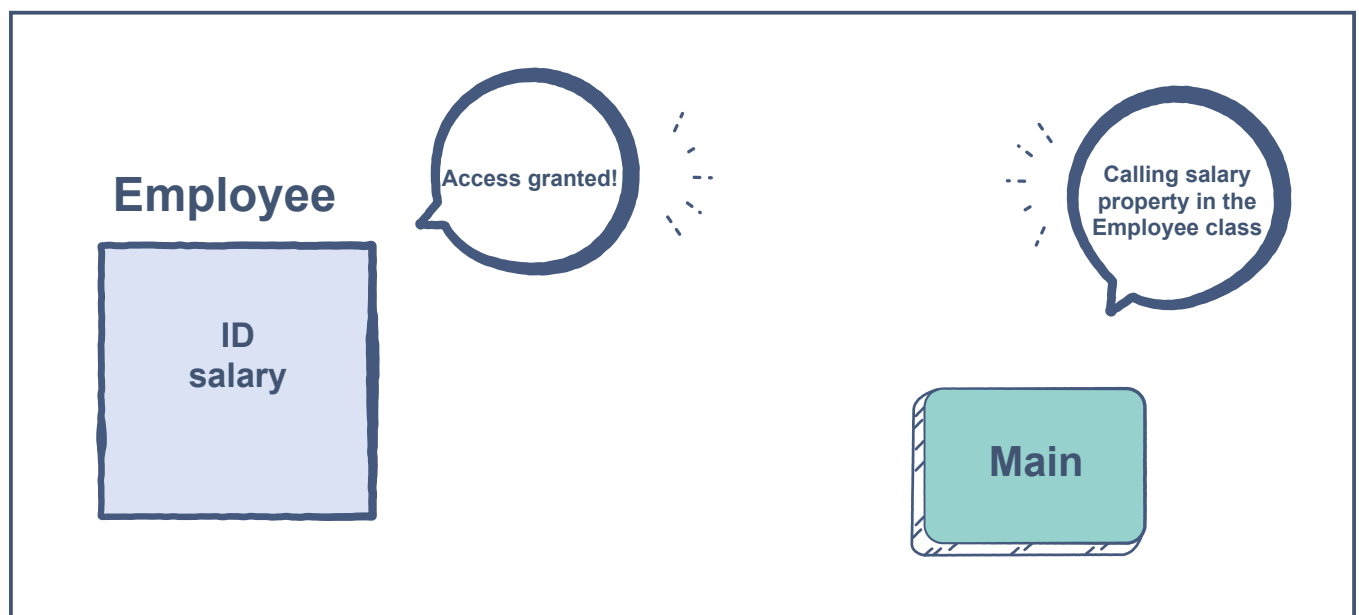
class

Private Attributes

> **Private attributes** cannot be accessed directly from outside the class but can be accessed from inside the class.

The aim is to keep it hidden from the users and other classes. Unlike in many different languages, it is not a widespread practice in Python to keep the data

members private since we do not want to create hindrances for the users. We can make members private using the double underscore `__` prefix

Trying to access private attributes in the main code will generate an *error*. An example of this is shown below:

Private Properties

Let's see a code example for implementing private properties:

```python
class Employee:
    def __init__(self, ID, salary):
        self.ID = ID
        self.__salary = salary  # salary is a private property


Steve = Employee(3789, 2500)
print("ID:", Steve.ID)
print("Salary:", Steve.__salary)  # this will cause an error
```

Code Explanation

- In the code above, `ID` is a *public* property but `__salary` is a *private* property, so it cannot be accessed outside the class.

- When it is tried to be accessed outside the class, the following error is generated:

  ```
  'Employee' object has no attribute '__salary'
  ```

- To ensure that no one from the outside knows about this *private* property, the error does not reveal the identity of it.
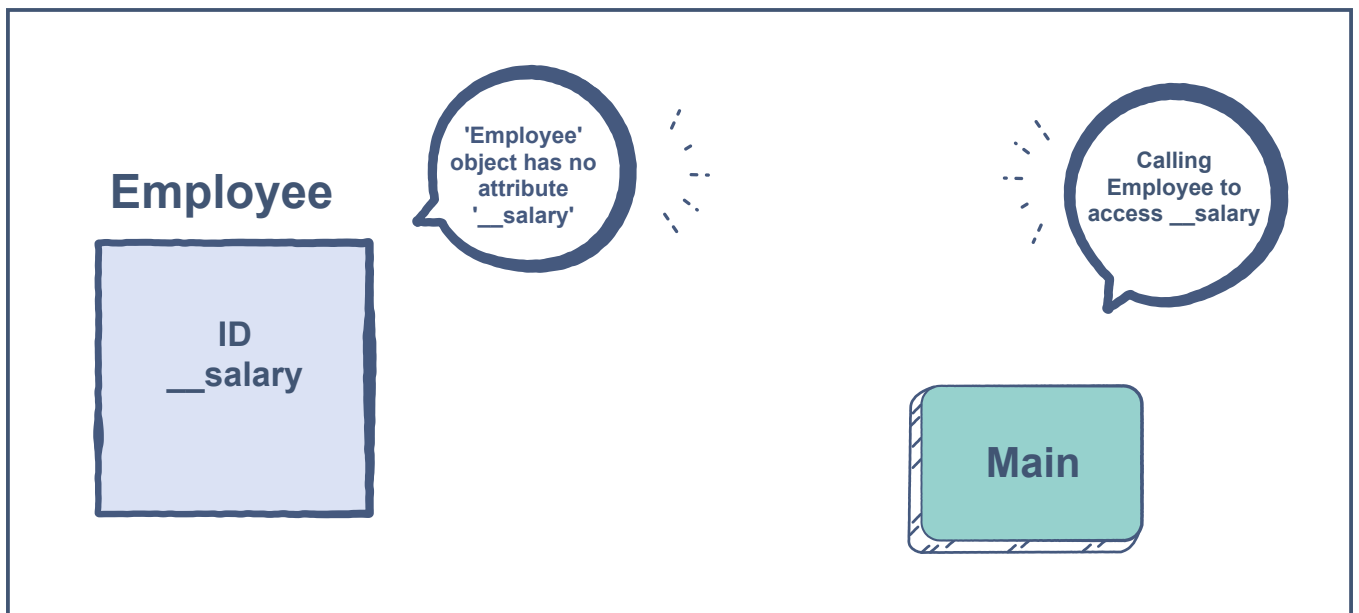
Class

Private Methods

Let's see a code example for implementing private methods:

private_methods

```python
class Employee:
    def __init__(self, ID, salary):
        self.ID = ID
        self.__salary = salary  # salary is a private property

    def displaySalary(self):  # displaySalary is a public method
        print("Salary:", self.__salary)

    def __displayID(self):  # displayID is a private method
        print("ID:", self.ID)


Steve = Employee(3789, 2500)
Steve.displaySalary()
Steve.__displayID()  # this will generate an error
```

Code Explanation

- `ID` is a *public* property, so it can be accessed from outside and inside the class.

- `__salary` is a *private* property, so it cannot be accessed from outside the class but can be accessed from inside the class.

- `displaySalary()` method is a *public* method, so it can be accessed from outside the class. This method can access the *private* property `__salary` as well.

- `__displayID()` method is a *private* method, so it cannot be accessed from outside the class.

- When `displayID()` tried to be accessed from outside the class, the following error is generated:

  `'Employee' object has no attribute '__displayID()'`

- To ensure that no one from the outside knows about this *private* property, the error does not reveal the identity of it.
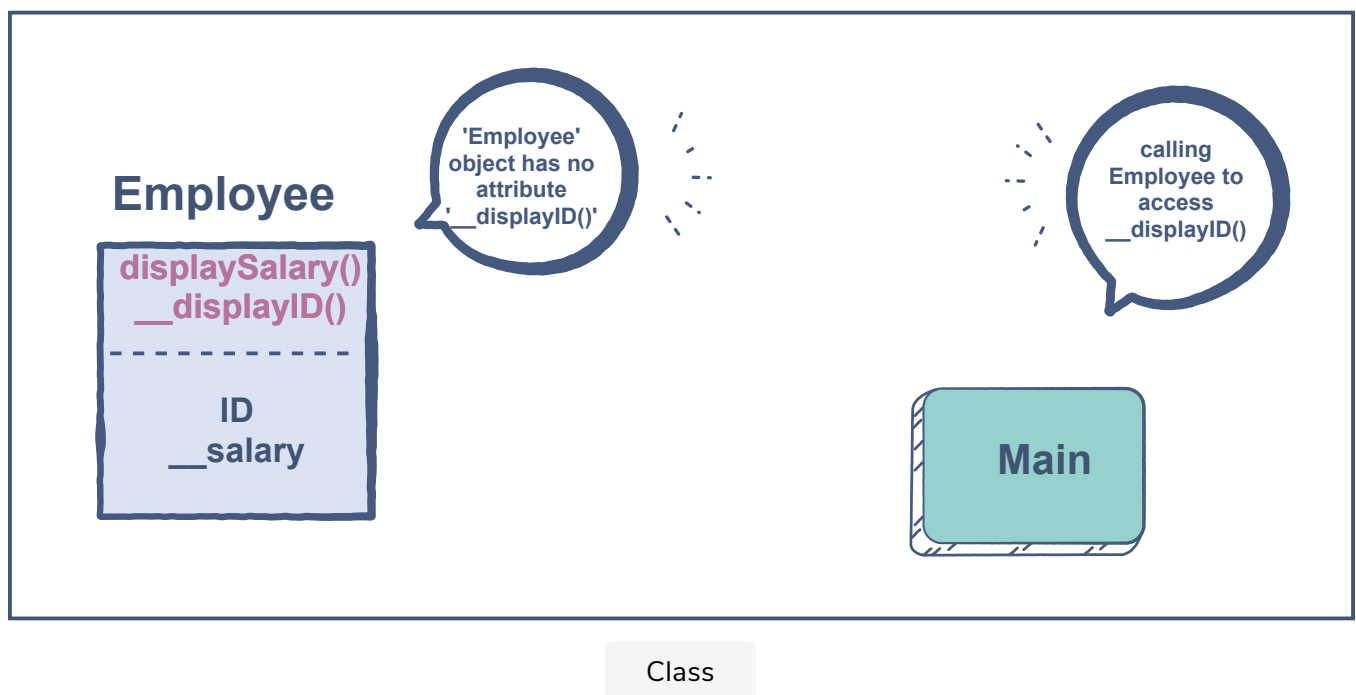
Class

> **Note:** Methods are **usually public** since they provide an interface for the class properties and the main code to interact with each other.

Accessing Private Attributes in the Main Code

As discussed above, it is not common to have private variables in Python.

Properties and methods with `__` prefix are usually present to make sure that the user does not *carelessly* access them. Python allows for a free hand to the

user to avoid any future complications in the code. If the user believes it is

absolutely necessary to access a private property or a method, they can access it using the `_<ClassName>` prefix for the property or method. An example of this is shown below:

```python
class Employee:
    def __init__(self, ID, salary):
        self.ID = ID
        self.__salary = salary  # salary is a private property


Steve = Employee(3789, 2500)
print(Steve._Employee__salary)  # accessing a private property
```

Not So Protected

Protected properties and methods in other languages can be accessed by classes and their subclasses which will be discussed later in the course. As we have seen, Python does not have a hard rule for accessing properties and methods, so it **does not** have the protected access modifier.

Now, let's test your knowledge with a quick quiz!