Initializing Objects

In this lesson, the world of initializers is explored. You'll learn why they are necessary for a class.

WE'LL COVER THE FOLLOWING

What is an Initializer?

As the name suggests, the **initializer** is used to *initialize* an object of a class. It's a special method that outlines the steps that are performed when an object of a class is created in the program. It's used to define and assign values to instance variables. We will discuss the concept of instance variables in the next lesson. For now, just focus on the syntax.

The initialization method is similar to other methods but has a pre-defined name, `__init__`.

> The double underscores mean this is a special method that the Python interpreter will treat as a special case.

The initializer is a special method because it **does not have a return type**. The first parameter of `__init__` is `self`, which is a way to refer to the object being initialized. We will discuss this later.

It is always a good practice to define it as the first member method in the class definition.

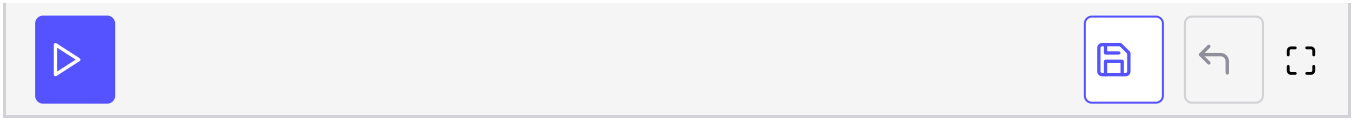
Defining Initializers

Initializers are called when an object of a class is created. See the example below:

```
class Employee:
    # defining the properties and assigning them None
    def __init__(self, ID, salary, department):
        self.ID = ID
        self.salary = salary
        self.department = department


# creating an object of the Employee class with default parameters
Steve = Employee(3789, 2500, "Human Resources")

# Printing properties of Steve
print("ID :", Steve.ID)
print("Salary :", Steve.salary)
print("Department :", Steve.department)
```



The initializer is automatically called when an object of the class is created. Now that we will be using initializers to make objects, a good practice would be to initialize all of the object properties when defining the initializer.

> It is important to define the initializer with complete parameters to avoid any errors. Similar to methods, initializers also have the provision for optional parameters.

Initializer with Optional Parameters

Similar to methods, we can also define initializers with optional parameters. As discussed previously, it's necessary to assign initial values to class properties when defining the class. So, when defining an initializer with optional parameters, it's essential to assign default values to the properties.

You can also have a **default initializer** that has all properties as optional. In this case, all the new objects will be created using the properties initialized in the initializer definition.

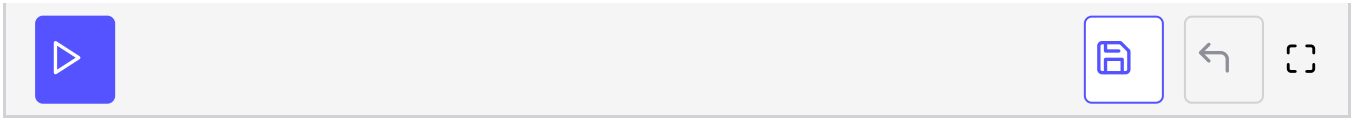
Below is an example where one `Employee` class object is created **without** the initializer parameters and one is created **with** the initializer parameters.

```
class Employee:
```

```
class Employee:
    # defining the properties and assigning None to them
    def __init__(self, ID=None, salary=0, department=None):
        self.ID = ID
        self.salary = salary
        self.department = department


# creating an object of the Employee class with default parameters
Steve = Employee()
Mark = Employee("3789", 2500, "Human Resources")

# Printing properties of Steve and Mark
print("Steve")
print("ID :", Steve.ID)
print("Salary :", Steve.salary)
print("Department :", Steve.department)
print("Mark")
print("ID :", Mark.ID)
print("Salary :", Mark.salary)
print("Department :", Mark.department)
```



In the next lesson, we will learn about the differences between class and instance variables.