Class Methods as Alternative Constructors in Python | Python Tutorial - Day #70

Day 70 - Class Methods as Alternative Constructors

In object-oriented programming, the term "constructor" refers to a special type of method that is automatically executed when an object is created from a class. The purpose of a constructor is to initialize the object's attributes, allowing the object to be fully functional and ready to use.
However, there are times when you may want to create an object in a different way, or with different initial values, than what is provided by the default constructor. This is where class methods can be used as alternative constructors.
A class method belongs to the class rather than to an instance of the class. One common use case for class methods as alternative constructors is when you want to create an object from data that is stored in a different format, such as a string or a dictionary. For example, consider a class named

"Person" that has two attributes: "name" and "age". The default constructor for the class might look like this:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

But what if you want to create a Person object from a string that contains the person's name and age, separated by a comma? You can define a class method named "from_string" to do this:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @classmethod
    def from_string(cls, string):
        name, age = string.split(',')
        return cls(name, int(age))
```

Now you can create a Person object from a string like this:

```
person = Person.from_string("John Doe, 30")
```

Another common use case for class methods as alternative constructors is when you want to create an object with a different set of default values than what is provided by the default constructor. For example, consider a class named "Rectangle" that has two attributes: "width" and "height". The default constructor for the class might look like this:

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
```

But what if you want to create a Rectangle object with a default width of 10 and a default height of 5? You can define a class method named "square" to do this:

```
class Rectangle:
  def __init__(self, width, height):
    self.width = width
    self.height = height

  @classmethod
  def square(cls, size):
    return cls(size, size)
```

Now you can create a square rectangle like this:

```
rectangle = Rectangle.square(10)
```

Previous

Next

CodeWithHarry

Copyright © 2023 CodeWithHarry.com