[Hindi] Magic/Dunder Methods In Python? | Object Oriented Programming Using Python Tutorial #8

[Hindi] Magic/Dunder Methods In Python? | Object Oriented Programming Using Python Tutorial #8

What is a Dunder Method?

Dunder method also called a magic method is the method having two prefixes and suffix underscores in the method name. Dunder here means “Double Under (Underscores)”.

Simplifying the statement. This simply means **__methodName__**

- To learn the Dunder Method in depth. Let us consider the following code snippet...

```
class Employee:
    increment = 1.5

    def __init__(self,fname,lname,salary):
        self.fname=fname
        self.lname=lname
        self.salary=salary
        self.increment=1.4

harry = Employee("harry", "jackson", 99000)
```

```
rohan = Employee("rohan", "das", 9)
```

In this snippet, **harry** and **rohan** are the objects of class **Employee**.
__init__ is the special method, which tends to auto-execute when the class is called.
As there are different types of Dunder Methods. Let's consider some of these...

1. **Dunder __add__ method:**

Take a look at the below code:

```
print(3+2)
print('rohan'+'2')
```

Output:

```
5
rohan2
```

In the above block, an addition operation is performed. The addition of two string variables may also be referred to as a concatenation operation.
Since we work with Dunder Method, both above addition operations can be performed with this method as follows:

```
a=3
#Addition
print(a.__add__(2))
#Multiplicaton
C= 'rohan'
print(a.__add__('2'))
```

Output:

```
5
Rohan2
```

General Syntax of Dunder Method:

The general syntax of the Dunder method may be memorized as,

"variable1. __methodName__(variable2)"

Based on the above scenarios, we learned that the addition (add) has the same features as the Dunder method (__add__)

Now you're probably thinking about, if the "add" method works correctly then why should we use the "__add__" Dunder method.

Suppose we wanted to add the salaries of the two items. Assuming the answer is straightforward.

We may conduct:

```
print(harry+rohan)
```

In this operation, to get our result, we simply add the two items of the Employee class. This will usually provide us with an error because we did not specify anything.

But, we want the above program to return us the addition of salaries. To obtain the desired outcome of this condition, the Dunder method can be considered.

Dunder Method substitutes for the "add" inbuild method. The process is also called overriding. The addition operation does not differ from the __add__ method. Add method calls the inbuilt __add__ method, which is our Dunder method.

A method called __add__ may be created within our class. Reason for creating the method within the class as both variables are objects of the same class.

Creating Method:

```
def __add__(self,other):
    return self.salary + other.salary
#Printing the values
print(harry+rohan)
```

Here:

- self = harry,
- other = rohan

Output:

```
99009
```

This will simply return the addition of salaries of harry and rohan.

2. **Dunder method __repr__**

__repr__ is a special method used to represent a class's objects as a string. It returns us the official string or syntax representation of the object from which it is made up.
__repr__ can be examined with the help of the following method.
Method:

```
def __repr__(self):
    return "Employee({},{},{})".format(self.fname,self.lname,self.salary)
print(harry)
```

Note: ". format" is used to handle complex strings.
Output:

```
Employee(harry,jackson,99000)
```

3. **Dunder method __str__:**

Another Dunder method similar to __repr__ is**__str__**
Both __repr__ and __str__ have the same features and both may be used in different ways.
To understand __str__ , we create a method that will return the object fname or first name. Like simple string output.
This can be achieved by:

```
def __str__(self):
    return f"The name of the employee is {self.fname}"
print(str(harry)
```

Output:

```
The name of the employee is harry
```

This is all about the dunder method. In the next article, we are going to discuss Decorators.

Previous

Next

CodeWithHarry