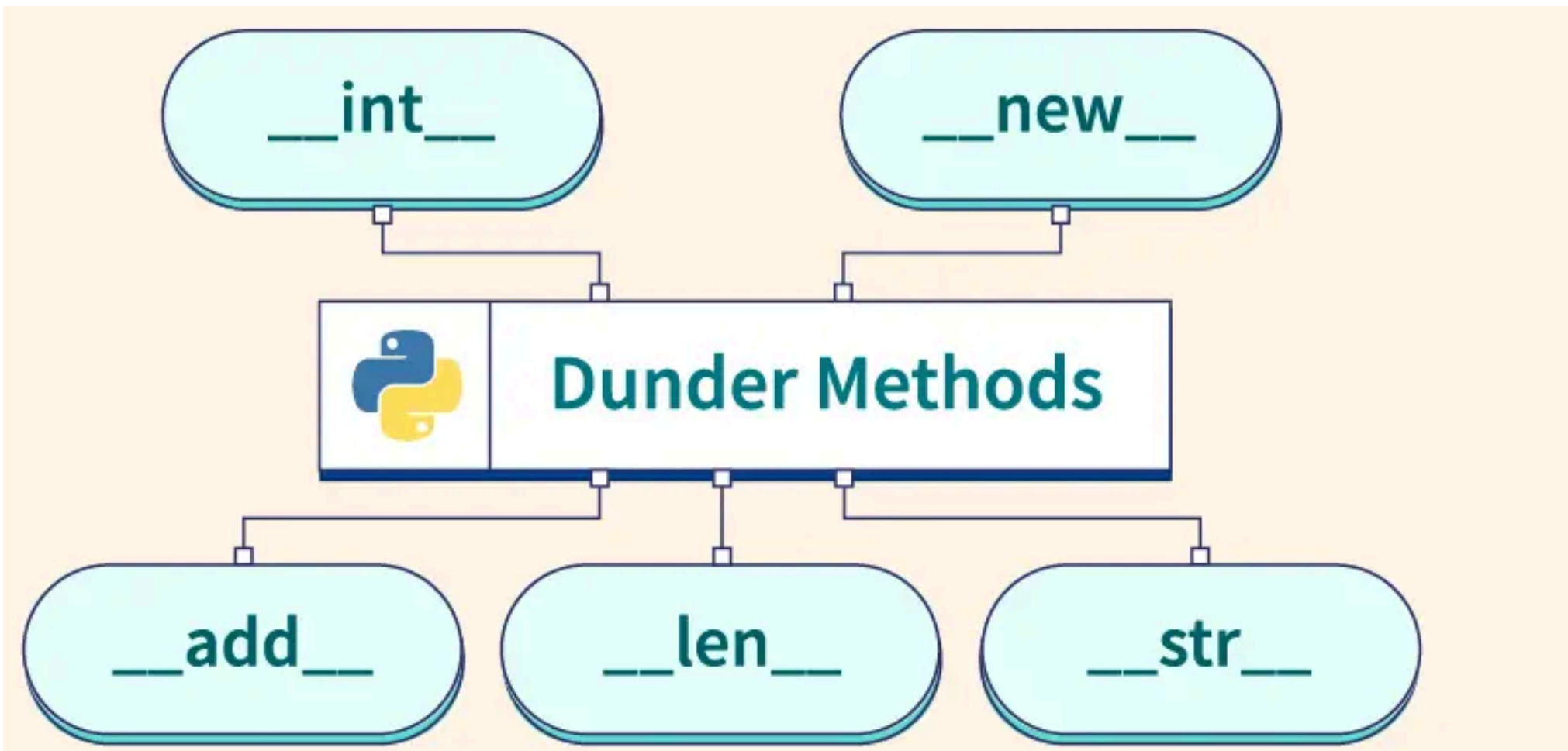
Dunder or magic methods in Python



Ganesh R

Senior Data Engineer

Magic methods in Python are the special methods that start and end with the double underscores. They are also called dunder methods. Magic methods are not meant to be invoked directly by you, but the invocation happens internally from the class on a certain action. For example, when you add two numbers using the + operator, internally, the __add__() method will be called.

Built-in classes in Python define many magic methods. Use the dir() function to see the number of magic methods inherited by a class. For example, the following lists all the attributes and methods defined in the int class.

```
>>> dir(int) ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__',
                                        '__dir__',
'__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__',
                                        '__init__',
'__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

As you can see above, the int class includes various magic methods surrounded by double underscores. For example, the __add__ method is a magic method which gets called when we add two numbers using the + operator. Consider the following example.

```
>>> num=10 >>> num
    + 5 15 >>>
 num.__add__(5) 15
```

As you can see, when you do num+10, the + operator calls the __add__(10) method. You can also call num.__add__(5) directly which will give the same result. However, as mentioned before, magic methods are not meant to be called directly, but internally, through some other methods or actions.

Magic methods are most frequently used to define overloaded behaviours of predefined operators in Python. For instance, arithmetic operators by default operate upon numeric operands. This means hat numeric objects must be used along with operators like +, -, *, /, etc. The + operator is also defined as a concatenation operator in string, list and tuple classes. We can say that the + operator is overloaded.

In order to make the overloaded behaviour available in your own custom class, the corresponding magic method should be overridden. For example, in order to use the + operator with objects of a user-defined class, it should include the __add__() method.

Let's see how to implement and use some of the important magic methods.

---

https://www.linkedin.com/in/rganesh0203/

__new__() method

Languages such as Java and C# use the new operator to create a new instance of a class. In Python the __new__() magic method is implicitly called before the __init__() method. The

__new__() method returns a new object, which is then initialized by __init__().

Example: __new__()

```
class Employee:
    def __new__(cls):
        print ("__new__ magic method is called")
        inst = object.__new__(cls)
                return inst
    def __init__(self):
        print ("__init__ magic method is called")
        self.name='Satya'
```

The above example will produce the following output when you create an instance of the `Employee` class.

```
>>> emp = Employee() __new__
magic method is called __init__
magic method is called
```

Thus, the `__new__()` method is called before the `__init__()` method.

__str__() method

Another useful magic method is __str__(). It is overridden to return a printable string representation of any user defined class. We have seen str() built-in function which returns a string from the object parameter. For example, str(12) returns '12'. When invoked, it calls the

__str__() method in the int class.

```
>>> num=12
>>> str(num)
'12'
>>> #This is equivalent to
>>> int.__str__(num)
'12'
```

Let us now override the `__str__()` method in the Employee class to return a string representation of its object.

Example: __str__()

```
class Employee:
    def __init__(self):
        self.name='Swati'
        self.salary=10000
    def __str__(self):
        return 'name='+self.name+' salary=$'+str(self.salary)
```

https://www.linkedin.com/in/rganesh0203/

See how the str() function internally calls the `__str__()` method defined in the Employee class. This is why it is called a magic method!

```
>>> e1=Employee()
>>> print(e1)
name=Swati salary=$10000
```

__add__() method

In following example, a class named distance is defined with two instance attributes - ft and inch. The addition of these two distance objects is desired to be performed using the overloading + operator.

To achieve this, the magic method __add__() is overridden, which performs the addition of the ft and inch attributes of the two objects. The __str__() method returns the object's string representation.

Example: Override __add__()

```
class distance:
    def __init__(self, x=None,y=None):
        self.ft=x
        self.inch=y
    def __add__(self,x):
        temp=distance()
        temp.ft=self.ft+x.ft
        temp.inch=self.inch+x.inch
        if temp.inch>=12:
            temp.ft+=1
            temp.inch-=12
            return temp
    def __str__(self):
        return 'ft:'+str(self.ft)+' in: '+str(self.inch)
```

Run the above Python script to verify the overloaded operation of the + operator.

```
>>> d1=distance(3,10)
>>> d2=distance(4,4)
>>> print("d1= {} d2={}".format(d1, d2))
d1= ft:3 in: 10 d2=ft:4 in: 4
>>> d3=d1+d2
>>> print(d3)
ft:8 in: 2
```

__ge__() method

The following method is added in the distance class to overload the `>=` operator.

Example: __ge__()

```
class distance:
    def __init__(self, x=None,y=None):
```

https://www.linkedin.com/in/rganesh0203/

```
        self.ft=x
        self.inch=y
    def __ge__(self, x):
        val1=self.ft*12+self.inch
        val2=x.ft*12+x.inch
        if val1>=val2:
            return True
        else:
            return False
```

This method gets invoked when the >= operator is used and returns True or False. Accordingly, the appropriate message can be displayed

```
>>> d1=distance(2,1)
>>> d2=distance(4,10)
>>> d1>=d2
False
  '  divmod  ',  '  doc  ',  '  eq  ',                                '  floordiv  ',
```

Important Magic Methods

The following tables list important magic methods in Python 3.

__new__(cls, other)	To get called in an object's instantiation.
__init__(self, other)	To get called by the __new__ method.
__del__(self)	Destructor method.

__pos__(self)	To get called for unary positive e.g. +someobject.
__neg__(self)	To get called for unary negative e.g. -someobject.
__abs__(self)	To get called by built-in abs() function.
__invert__(self)	To get called for inversion using the ~ operator.
__round__(self,n)	To get called by built-in round() function.
__floor__(self)	To get called by built-in math.floor() function.
__ceil__(self)	To get called by built-in math.ceil() function.
__trunc__(self)	To get called by built-in math.trunc() function.

__iadd__(self, other)	To get called on addition with assignment e.g. a +=b.
__isub__(self, other)	To get called on subtraction with assignment e.g. a -=b.
__imul__(self, other)	To get called on multiplication with assignment e.g. a *=b.
__ifloordiv__(self, other)	To get called on integer division with assignment e.g. a //=b.
__idiv__(self, other)	To get called on division with assignment e.g. a /=b.
__itruediv__(self, other)	To get called on true division with assignment
__imod__(self, other)	To get called on modulo with assignment e.g. a%=b.
__ipow__(self, other)	To get called on exponentswith assignment e.g. a **=b.

https://www.linkedin.com/in/rganesh0203/

__ilshift__(self, other)	To get called on left bitwise shift with assignment e.g. a<<=b.
__irshift__(self, other)	To get called on right bitwise shift with assignment e.g. a >>=b.
__iand__(self, other)	To get called on bitwise AND with assignment e.g. a&=b.
__ior__(self, other)	To get called on bitwise OR with assignment e.g. a|=b.
__ixor__(self, other)	To get called on bitwise XOR with assignment e.g. a ^=b.

__int__(self)	To get called by built-int int() method to convert a type to an int.
__float__(self)	To get called by built-int float() method to convert a type to float.
__complex__(self)	To get called by built-int complex() method to convert a type to complex.
__oct__(self)	To get called by built-int oct() method to convert a type to octal.
__hex__(self)	To get called by built-int hex() method to convert a type to hexadecimal.
__index__(self)	To get called on type conversion to an int when the object is used in a slice expression.
__trunc__(self)	To get called from math.trunc() method.

__str__(self)	To get called by built-int str() method to return a string representation of a type.
__repr__(self)	To get called by built-int repr() method to return a machine readable representation of a type.
__unicode__(self)	To get called by built-int unicode() method to return an unicode string of a type.
__format__(self, formatstr)	To get called by built-int string.format() method to return a new style of string.
__hash__(self)	To get called by built-int hash() method to return an integer.
__nonzero__(self)	To get called by built-int bool() method to return True or False.
__dir__(self)	To get called by built-int dir() method to return a list of attributes of a class.
__sizeof__(self)	To get called by built-int sys.getsizeof() method to return the size of an object.

__getattr__(self, name)	Is called when the accessing attribute of a class that does not exist.
__setattr__(self, name, value)	Is called when assigning a value to the attribute of a class.
__delattr__(self, name)	Is called when deleting an attribute of a class.

__add__(self, other)	To get called on add operation using + operator

https://www.linkedin.com/in/rganesh0203/

__sub__(self, other)	To get called on subtraction operation using - operator.
__mul__(self, other)	To get called on multiplication operation using * operator.
__floordiv__(self, other)	To get called on floor division operation using // operator.
__truediv__(self, other)	To get called on division operation using / operator.
__mod__(self, other)	To get called on modulo operation using % operator.
__pow__(self, other[, modulo])	To get called on calculating the power using ** operator.
__lt__(self, other)	To get called on comparison using < operator.
__le__(self, other)	To get called on comparison using <= operator.
__eq__(self, other)	To get called on comparison using == operator.
__ne__(self, other)	To get called on comparison using != operator.
__ge__(self, other)	To get called on comparison using >= operator.

Follow for more content like this

Ganesh R

Senior Data Engineer

Azure Cloud for Data

https://www.linkedin.com/in/rganesh0203/