6 MONTHS AGO • 6 MIN READ

Python Dunder (Special, Magic) Methods List with Tutorial

#Python

There is a reason for Python's popularity, it is the consistency of this language. Python allows you to support your custom objects for inbuilt functions like `len()`, `abs()`, `sorted()` and many others. You just need to implement some special methods (dunder methods) to your class.

In this tutorial I have used words like dunder methods, special methods and magic methods, all have the same meaning.

Before diving deep into the tutorial I want to share a fun fact about these names.

If you know object-oriented programming in Python then you will be familiar to this `__init__` method. These have name dunder method because those `__` double underscores are called dunder in Python.

There are magic methods in Ruby which serve the same purpose that’s why some Python developers use this name in Python language.

Mostly I call them Special Methods because this is what we see in Python documentation.

That is the reason if you see dunder methods or Magic Methods always assume them as Special Methods in Python.

What are Special methods in Python?

The Special methods have a predefined syntax which is available to implement in our class.

The core data structures also implement these Special methods internally which allow them to be compatible with many inbuilt functions.

To understand this better, here is an example of a Python list. We can easily use `len()` function to find the length of any list but here is an alternative to confuse you.

```
nums = [1, 2, 3, 4, 5]
print(len(nums)) #5
print(nums.__len__()) #5
```

Here `__len__` is the magic method that does all the magic. Both syntaxes is the same for Python.

Whenever we use an inbuilt function, it tries to find a predefined method that does the task, like `len()` function finds `__len__` method in that object.

The reason behind this was to build a common API which other classes can implement to do the same task. This removes the burden to learn methods of each class to do that task.

This is the reason I love Python. I remember the Java days of my coding when I spent most of my time thinking what is the method name to do any specific task. It was really frustrating.

Now I can think that you have a basic understanding of Special methods in Python.

List of all Special Methods in Python

There are very different types of Special methods and they are in large number. It is not possible to explain them all in a single post.

Here are the most popular and widely used list of Special or dunder methods in Python.

Basic Customizations

- `__new__(self)` return a new object (an instance of that class). It is called before `__init__` method.
- `__init__(self)` is called when the object is initialized. It is the constructor of a class.
- `__del__(self)` for `del()` function. Called when the object is to be destroyed. Can be used to commit unsaved data or close connections.
- `__repr__(self)` for `repr()` function. It returns a string to print the object. Intended for developers to debug. Must be implemented in any class.
- `__str__(self)` for `str()` function. Return a string to print the object. Intended for users to see a pretty and useful output. If not implemented, `__repr__` will be used as a fallback.
- `__bytes__(self)` for `bytes()` function. Return a byte object which is the byte string representation of the object.
- `__format__(self)` for `format()` function. Evaluate formatted string literals like `%` for percentage format and 'b' for binary.
- `__lt__(self, anotherObj)` for < operator.
- `__le__(self, anotherObj)` for <= operator.
- `__eq__(self, anotherObj)` for == operator.
- `__ne__(self, anotherObj)` for != operator.
- `__gt__(self, anotherObj)`for > operator.
- `__ge__(self, anotherObj)`for >= operator.

Arithmetic Operators

- `__add__(self, anotherObj)` for + operator.
- `__sub__(self, anotherObj)` for – operation on object.
- `__mul__(self, anotherObj)` for * operation on object.
- `__matmul__(self, anotherObj)` for @ operator (numpy matrix multiplication).
- `__truediv__(self, anotherObj)` for simple / division operation on object.
- `__floordiv__(self, anotherObj)` for // floor division operation on object.

Type Conversion

- `__abs__(self)` make support for `abs()` function. Return absolute value.
- `__int__(self)` support for `int()` function. Returns the integer value of the object.
- `__float__(self)` for `float()` function support. Returns float equivalent of the object.

- `__complex__(self)` for `complex()` function support. Return complex value representation of the object.
- `__round__(self, nDigits)` for `round()` function. Round off float type to 2 digits and return it.
- `__trunc__(self)` for `trunc()` function of math module. Returns the real value of the object.
- `__ceil__(self)` for `ceil()` function of math module. The ceil function Return ceiling value of the object.
- `__floor__(self)` for `floor()` function of math module. Return floor value of the object.

Emulating Container Types

- `__len__(self)` for `len()` function. Returns the total number in any container.
- `__getitem__(self, key)` to support indexing. LIke container[index] calls `container.__getitem(key)`explicitly.
- `__setitem__(self, key, value)` makes item mutable (items can be changed by index), like `container[index] = otherElement`.
- `__delitem__(self, key)` for `del()` function. Delete the value at the index `key`.
- `__iter__(self)` returns an iterator when required that iterates all values in the container.

Here is a practical use case of `__init__`special method in Python.

Example use of init dunder method.

How to use Dunder or Special Magic Methods?

Now you know what are dunder methods in Python. It is very easy to use them in your custom created classes. They will extend the functionality of the class by making them compatible with inbuilt python function.

Note that it is not necessary and sometimes not possible to implement each and every Special method in any class. For example, you cannot divide two bank account together so there is no need to implement `__truediv__` Special method in `Account` class.

Here is an example that explains the use of all Special methods mentioned above.

```
class Account:
    # def __new__(self):
    #     print("__new__ called A new account is Created.")
    def __init__(self, name, balance = 0):
        self.name = name
        self._balance = balance
        print("(__init__) Account Created with {} balance".format(self._balar
    def getBalance(self):
        return self._balance
    def __repr__(self):
        return "(__repr__) Account({0}, {1})".format(self.name, self._balance
    def __str__(self):
        return "(__str__) Account Name = {0}, Account Balance = {1}".format(s
    def __lt__(self, otherObj):
        try:
            return self._balance<otherObj.getBalance()
        except:
            return "Cannot Be compared."
    def __del__(self):
        self._balance = 0
        print("(__del__) Account Deleted")

digvijay_ac = Account("Digvijay Singh", 500)
saket_ac = Account("Digvijay Singh", 700)

print(str(digvijay_ac))
print(repr(digvijay_ac))
print(digvijay_ac<saket_ac)
```

The Output of the above code is:

```
(__init__) Account Created with 500 balance
(__init__) Account Created with 700 balance
(__str__) Account Name = Digvijay Singh, Account Balance = 500
(__repr__) Account(Digvijay Singh, 500)
True
(__del__) Account Deleted
(__del__) Account Deleted
```

The only thing that is annoying in the output is the message Account Deleted. We don't

call `del()` function on the object still it executes.

This is the intelligence of Python. It deletes the objects as soon as the interpreter exits. The `__del__()` is executed implicitly.

Why use Dunder methods?

I have already leaked the main reason to implement dunder methods in your class. There are hundreds of Special methods available to implement in Python.

They give the ability to create classes that behave like native data structures like lists, tuples, dictionary, set etc.

The special methods provide a common API that allows developers to create interactive classes which are very useful. The benefit of special methods in Python is that we do not need to remember individual methods of any object to do any task.

For example how to calculate the length of an array in Java? Is it `.len()` method or `.length()` method?

Python is consistent in its design. The `len()` function works with the list, dictionary, tuples and other inbuilt data structures and you can also make your own object compatible with by implementing suitable dunder method (__len__() in this case).

In simple words, we can say that Dunder or Magic Methods make Python Consistent making it easy to learn and use.

Conclusion

Now, you have a complete understanding of dunder methods in Python. They are very useful to make our class interactive.

To have a better understanding of this you need to implement this on your own. There are certain things which you will learn only when you write code.

Here is the summary of the points.

- Dunder Methods, Special Methods and Magic methods are the same thing in Python.
- Dunder Methods makes our class compatible with inbuilt functions like`abs()`, `len()`, `str()` and many more.
- Users don’t need to remember each and every method to do a certain task, just use inbuilt function and pass the object with required parameters.

- They make Python consistent with its syntax.

Share the Post ;)

Digvijay Singh
I am a computer science student from Lucknow, India. I love coding web apps and reading self-help books. I spend most of the time working on side projects.

Liked the article?

Get more like this directly in your inbox.

EMAIL

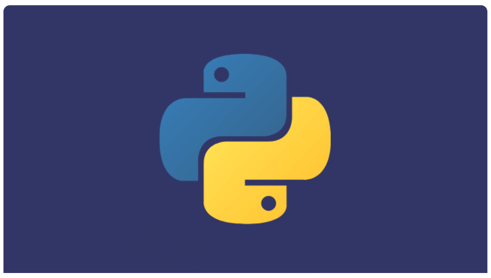
yourmail@email.com

Get In

Types of Data structures with examples

Python program to remove the key from a dictionary

Related Posts

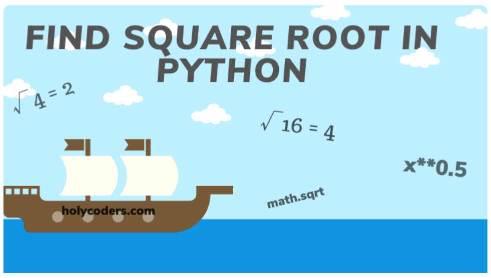


• 2 min read

Python Program to count even and odd numbers in a list

This is a simple python program to count even and odd numbers is a list using for loop, lambda function and list comprehension feature of Python....

Digvijay Singh • 22 Jun 2020



• 2 min read

Find Square Root of a number in Python 3

Problem Statement: Find square root of any number in Python with and without using inbuilt function...

Digvijay Singh • 8 Jan 2020

• 4 min read

Nested List Comprehension in Python: If-else, loop Examples

The beauty of python lies in its simplicity and solving complex problems in less code. List compreh...

Digvijay Singh • 14 Mar 2020

About Contact