Python Built-In Functions

Gaurav Kr. suman

MIT5

Python Built-In Functions

1. abs()

The abs() is one of the most popular Python built-in functions, which returns the absolute value of a number. A negative value's absolute is that value is positive.

```
>>> abs(-7)
```

7

```
>>> abs(7)
```

7

```
>>> abs(0)
```

2. all()

The all() function takes a container as an argument. This Built in Functions returns True if all values in a python iterable have a Boolean value of True. An empty value has a Boolean value of False.

```
>>> all({'*','',''})
```

False

```
>>> all([' ',' ',' '])
```

True

3. any()

Like all(), it takes one argument and returns True if, even one value in the iterable has a Boolean value of True.

```
>>> any((1,0,0))
```

True

```
>>> any((0,0,0))
```

False

4. ascii()

It is important Python built-in functions, returns a printable representation of a **python object** (like a string or a **Python list**). Let's take a Romanian character.

```
>>> ascii('ş')
```

“‘\\u0219’”

Since this was a non-ASCII character in python, the interpreter added a backslash (\) and escaped it using another backslash.

```
>>> ascii('uşor')
```

“‘u\\u0219or'”

Let’s apply it to a list.

```
>>> ascii(['s','ş'])
```

“[‘s’, ‘\\u0219’]”

5. bin()

bin() converts an integer to a binary string. We have seen this and other functions in our article on Python Numbers.

```
>>> bin(7)
```

‘0b111’

We can’t apply it on floats, though.

```
>>> bin(7.0)
```

Traceback (most recent call last):

File “<pyshell#20>”, line 1, in <module>

bin(7.0)

TypeError: ‘float’ object cannot be interpreted as an integer

6. bool()

bool() converts a value to Boolean.

```
>>> bool(0.5)
```

True

```
>>> bool('')
```

False

```
>>> bool(True)
```

True

7. bytearray()

bytearray() returns a python array of a given byte size.

```
>>> a=bytearray(4)
>>> a
```

bytearray(b’\x00\x00\x00\x00′)

```
>>> a.append(1)
>>> a
```

bytearray(b’\x00\x00\x00\x00\x01′)

```
>>> a[0]=1
>>> a
```

bytearray(b’\x01\x00\x00\x00\x01′)

```
>>> a[0]
```

1

Let’s do this on a list.

```
>>> bytearray([1,2,3,4])
```

bytearray(b’\x01\x02\x03\x04′)

8. bytes()

bytes() returns an immutable bytes object.

```
>>> bytes(5)
```

b’\x00\x00\x00\x00\x00′

```
>>> bytes([1,2,3,4,5])
```

b’\x01\x02\x03\x04\x05′

```
>>> bytes('hello','utf-8')
```

b’hello’

Here, utf-8 is the encoding.

Both bytes() and bytearray() deal with raw data, but bytearray() is mutable, while bytes() is immutable.

```
>>> a=bytes([1,2,3,4,5])
>>> a
```

b’\x01\x02\x03\x04\x05′

```
>>> a[4]=
3
Traceback (most recent call last):
File "<pyshell#46>", line 1, in <module>
a[4]=3
TypeError: 'bytes' object does not support item assignment
```

Let's try this on bytearray().

```
>>> a=bytearray([1,2,3,4,5])
>>> a
bytearray(b'\x01\x02\x03\x04\x05')
>>> a[4]=3
>>> a
bytearray(b'\x01\x02\x03\x04\x03')
```

9. callable()

callable() tells us if an object can be called.

```
>>> callable([1,2,3])
False
>>> callable(callable)
True
>>> callable(False)
False
>>> callable(list)
True
```

A function is callable, a list is not. Even the callable() python Built In function is callable.

10. chr()

chr() Built In function returns the character in python for an ASCII value.

```
>>> chr(65)
'A'
>>> chr(97)
```

```
'a'
>>> chr(9)
'\t'
>>> chr(48)
'0'
```

11. classmethod()

classmethod() returns a class method for a given method.

```
>>> class fruit:
def sayhi(self):
print("Hi, I'm a fruit")

>>> fruit.sayhi=classmethod(fruit.sayhi)
>>> fruit.sayhi()
Hi, I'm a fruit
```

When we pass the method sayhi() as an argument to classmethod(), it converts it into a python class method one that belongs to the class. Then, we call it like we would call any static method in python without an object.

12. compile()

compile() returns a Python code object. We use Python in built function to convert a string code into object code.

```
>>> exec(compile('a=5\nb=7\nprint(a+b)','','exec'))
12
```

Here, 'exec' is the mode. The parameter before that is the filename for the file form which the code is read.
Finally, we execute it using exec().

13. complex()

complex() function creates a complex number. We have seen this is our article on Python Numbers.

```
>>> complex(3)
(3+0j)
```

```
>>> complex(3.5)
(3.5+0j)
>>> complex(3+5j)
(3+5j)
```

14. delattr()

delattr() takes two arguments- a class, and an attribute in it. It deletes the attribute.

```
>>> class fruit:
size=7

>>> orange=fruit()
>>> orange.size
7
>>> delattr(fruit,'size')
>>> orange.size
Traceback (most recent call last):
File “<pyshell#95>”, line 1, in <module>
orange.size
AttributeError: ‘fruit’ object has no attribute ‘size’
```

15. dict()

dict(), as we have seen it, creates a python dictionary.

```
>>> dict()
{}
>>> dict([(1,2),(3,4)])
{1: 2, 3: 4}
```

16. divmod()

divmod() in Python built-in functions, takes two parameters, and returns a tuple of their quotient and remainder. In other words, it returns the floor division and the modulus of the two numbers.

```
>>> divmod(3,7)
```

```
(0, 3)
>>> divmod(7,3)
(2, 1)
```

If you encounter any doubt in Python Built-in Function, Please Comment.

17. enumerate()

This Python Built In function returns an enumerate object. In other words, it adds a counter to the iterable.

```
>>> for i in enumerate(['a','b','c']):
print(i)
(0, ‘a’)
(1, ‘b’)
(2, ‘c’)
```

18. eval()

This Function takes a string as an argument, which is parsed as an expression.

```
>>> x=7
>>> eval('x+7')
14
>>> eval('x+(x%2)')
8
```

19. exec()

exec() runs Python code dynamically.

```
>>> exec('a=2;b=3;print(a+b)')
5
>>> exec(input("Enter your program"))
Enter your programprint(2+3)
5
```

20. filter()

Like we’ve seen in python Lambda Expressios, filter() filters out the items for which the condition is True.

```
>>> list(filter(lambda x:x%2==0,[1,2,0,False]))
[2, 0, False]
```

21. float()

This Python Built In function converts an int or a compatible value into a float.

```
>>> float(2)
2.0
>>> float('3')
3.0
>>> float('3s')
Traceback (most recent call last):
File “<pyshell#136>”, line 1, in <module>
float(‘3s’)
ValueError: could not convert string to float: ‘3s’
>>> float(False)
0.0
>>> float(4.7)
4.7
```

22. format()

We have seen this Python built-in function, one in our lesson on Python Strings.

```
>>> a,b=2,3
>>> print("a={0} and b={1}".format(a,b))
a=2 and b=3
>>> print("a={a} and b={b}".format(a=3,b=4))
a=3 and b=4
```

23. frozenset()

frozenset() returns an immutable frozenset object.

```
>>> frozenset((3,2,4))
```

frozenset({2, 3, 4})

Read Python Sets and Booleans for more on frozenset.

24. getattr()

getattr() returns the value of an object’s attribute.

```
>>> getattr(orange,'size')
```

7

25. globals()

This Python built-in functions, returns a dictionary of the current global symbol table.

```
>>> globals()
```

{‘__name__’: ‘__main__’, ‘__doc__’: None, ‘__package__’: None, ‘__loader__’: <class ‘_frozen_importlib.BuiltinImporter’>, ‘__spec__’: None, ‘__annotations__’: {}, ‘__builtins__’: <module ‘builtins’ (built-in)>, ‘fruit’: <class ‘__main__.fruit’>, ‘orange’: <__main__.fruit object at 0x05F937D0>, ‘a’: 2, ‘numbers’: [1, 2, 3], ‘i’: (2, 3), ‘x’: 7, ‘b’: 3}

26. hasattr()

Like delattr() and getattr(), hasattr() Python built-in functions, returns True if the object has that attribute.

```
>>> hasattr(orange,'size')
```

True

```
>>> hasattr(orange,'shape')
```

True

```
>>> hasattr(orange,'color')
```

False

27. hash()

hash() function returns the hash value of an object. And in Python, everything is an object.

```
>>> hash(orange)
```

6263677

```
>>> hash(orange)
```

6263677

```
>>> hash(True)
```

1

```
>>> hash(0)
```

0

```
>>> hash(3.7)
```

644245917

```
>>> hash(hash)
```

25553952

28. hex()

Hex() Python built-in functions, converts an integer to hexadecimal.

```
>>> hex(16)
```

'0x10'

```
>>> hex(False)
```

'0x0'

29. id() Function

id() returns an object's identity.

```
>>> id(orange)
```

100218832

```
>>> id({1,2,3})==id({1,3,2})
```

True

30. input()

Input() Python built-in functions, reads and returns a line of string.

```
>>> input("Enter a number")
```

Enter a number7
'7'

Note that this returns the input as a string. If we want to take 7 as an integer, we need to apply the int() function to it.

```
>>> int(input("Enter a number"))
```

Enter a number7

7

31. int()

int() converts a value to an integer.

```
>>> int('7')
```

7

32. isinstance()

We have seen this one in previous lessons. isinstance() takes a variable and a class as arguments. Then, it returns True if the variable belongs to the class. Otherwise, it returns False.

```
>>> isinstance(0,str)
```

False

```
>>> isinstance(orange,fruit)
```

True

33. ord()

The function ord() returns an integer that represents the Unicode point for a given Unicode character.

```
>>> ord('A')
```

65

```
>>> ord('9')
```

57
This is complementary to chr().

```
>>> chr(65)
```

‘A’

34. pow()

pow() takes two arguments- say, x and y. It then returns the value of x to the power of y.

```
>>> pow(3,4)
```

81

```
>>> pow(7,0)
```

1

```
>>> pow(7,-1)
```

0.14285714285714285

```
>>> pow(7,-2)
```

0.02040816326530612

35. print()

We don't think we need to explain this anymore. We've been seeing this function since the beginning of this article.

```
>>> print("Okay, next function, please!")
```

Okay, next function, please!

36. range()

We've taken a whole tutorial on this. Read up range() in Python.

```
>>> for i in range(7,2,-2):
print(i)
```

```
7
5
3
```

37. repr()

repr() returns a representable string of an object.

```
>>> repr("Hello")
```

"'Hello'"

```
>>> repr(7)
```

'7'

```
>>> repr(False)
```

'False'

38. reversed()

This functions reverses the contents of an iterable and returns an iterator object.

```
>>> a=reversed([3,2,1])
>>> a
<list_reverseiterator object at 0x02E1A230>
>>> for i in a:
print(i)
1
2
3
>>> type(a)
<class 'list_reverseiterator'>
```

39. round()

round() rounds off a float to the given number of digits (given by the second argument).

```
>>> round(3.777,2)
3.78
>>> round(3.7,3)
3.7
>>> round(3.7,-1)
0.0
>>> round(377.77,-1)
380.0
```

The rounding factor can be negative.

40. set()

Of course, set() returns a set of the items passed to it.

```
>>> set([2,2,3,1])
{1, 2, 3}
```

Remember, a set cannot have duplicate values, and isn't indexed, but is ordered. Read on Sets and Booleans for the same.

41. setattr()

Like getattr(), setattr() sets an attribute's value for an object.

```
>>> orange.size
7
>>> orange.size=8
>>> orange.size
8
```