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)

out put 7

>>> abs(7)

out put 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({'*','',''})

out put False

>>> all([' ',' ',' '])
out put 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))

out put True

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

out put 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, shall we!

1. >>> ascii('ș') (disclaimer thats not S)

Output “‘\\u0219′”
Since this was a non-ASCII character in python, the interpreter added a backslash (\) and escaped it using another backslash.

```
>>> ascii('ușor')
```

Output “‘u\\u0219or'”
Let’s apply it to a list.

```
>>> ascii(['s','ș'])
```

Output “[‘s’, ‘\\u0219’]”

5. bin()

bin() converts an integer to a binary string.

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

Output ‘0b111’
We can’t apply it on floats, though.

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

```
Output 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)
```

Output True

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

Output False

>>> bool(True)

Output True

7. bytearray()

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

>>> a=bytearray(4)

>>> a

Output bytearray(b'\x00\x00\x00\x00')

>>> a.append(1)

>>> a

Output bytearray(b'\x00\x00\x00\x00\x01')

>>> a[0]=1

>>> a

Output bytearray(b'\x01\x00\x00\x00\x01')

>>> a[0]

Let's do this on a list.

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

bytearray(b'\x01\x02\x03\x04')

8. bytes()

bytes() returns an immutable bytes object.

>>> bytes(5)

Output b'\x00\x00\x00\x00\x00'

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

```
Output b’\x01\x02\x03\x04\x05′
```

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

```
Output 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

Output b’\x01\x02\x03\x04\x05′

Output >>> 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().

1. >>> a=bytearray([1,2,3,4,5])
2. >>> a

```
bytearray(b’\x01\x02\x03\x04\x05′)
```

1. >>> a[4]=3
2. >>> a

```
Output bytearray(b’\x01\x02\x03\x04\x03′)
```

9. callable()

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

```
>>> callable([1,2,3])
```

```
Output False
```

```
>>> callable(callable)
```

Output True

```
>>> callable(False)
```

Output False

```
>>> callable(list)
```

Output 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)
```

Output ‘A’

```
>>> chr(97)
```

Output ‘a’

```
>>> chr(9)
```

Output ‘\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'))
```

Output 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)
```

Output (3+0j)

```
>>> complex(3.5)
```

Output (3.5+0j)

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

Output (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
```

Output 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()
```

Output {}

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

Output {1: 2, 3: 4}
This was about dict() Python Built In function

16. dir()

dir() returns an object's attributes.

```
>>> class fruit:
size=7
shape='round'
>>> orange=fruit()
>>> dir(orange)
```

Output ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'shape', 'size']

17. 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)
```

Output (0, 3)

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

Output (2, 1)
If you encounter any doubt in Python Built-in Function, Please Comment.

18. enumerate()

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

1. >>> for i in enumerate(['a','b','c']):
2. print(i)

```
Output (0, ‘a’)
       (1, ‘b’)
       (2, ‘c’)
```

19. eval()

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

```
>>> x=7

>>> eval('x+7')

Output 14

>>> eval('x+(x%2)')

Output 8
```

20. exec()

exec() runs Python code dynamically.

```
>>> exec('a=2;b=3;print(a+b)')

Output 5

>>> exec(input("Enter your program"))

Enter your programprint(2+3)
Output 5
```

21. 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]))
```

```
Output [2, 0, False]
```

22. float()

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

```
>>> float(2)
```

```
Output 2.0
```

```
>>> float('3')
```

```
Output 3.0
```

```
>>> float('3s')
```

```
Output Traceback (most recent call last):
File "<pyshell#136>", line 1, in <module>
float('3s')
ValueError: could not convert string to float: '3s'
```

```
>>> float(False)
```

```
Output 0.0
```

```
>>> float(4.7)
```

```
Output 4.7
```

23. format()

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

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

```
Output a=2 and b=3
```

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

Output a=3 and b=4

24. frozenset()

frozenset() returns an immutable frozenset object.

1. >>> frozenset((3,2,4))

frozenset({2, 3, 4})
Read **Python Sets and Booleans** for more on frozenset.

25. getattr()

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

1. >>> getattr(orange,'size')

Output 7

26. globals()

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

1. >>> globals()

Output {‘__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}

27. hasattr()

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

>>> hasattr(orange,'size')

Output True

>>> hasattr(orange,'shape')

Output True

>>> hasattr(orange,'color')

Output False

28. hash()

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

>>> hash(orange)

Output 6263677

>>> hash(orange)

Output 6263677

>>> hash(True)

Output 1

>>> hash(0)

Output 0

>>> hash(3.7)

Output 644245917

>>> hash(hash)

Output 25553952
This was all about hash() Python In Built function

29. help()

To get details about any module, keyword, symbol, or topic, we use the help() function.

1. >>> help()
2.

Output Welcome to Python 3.6's help utility!

3.
4. If this is your first time using Python, you should definitely check out the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.
5.

```
Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

help> map
Help on class map in module builtins:
class map(object)
| map(func, *iterables) --> map object
|
| Make an iterator that computes the function using arguments from
| each of the iterables. Stops when the shortest iterable is exhausted.
|
| Methods defined here:
|
| __getattribute__(self, name, /)
| Return getattr(self, name).
|
| __iter__(self, /)
| Implement iter(self).
|
| __new__(*args, **kwargs) from builtins.type
| Create and return a new object. See help(type) for accurate signature.
|
| __next__(self, /)
| Implement next(self).
|
| __reduce__(...)
| Return state information for pickling.
help> You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)". Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>>
```

30. hex()

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

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

Output ‘0x10’

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

Output ‘0x0’

31. id() Function

id() returns an object’s identity.

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

Output 100218832

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

Output True

32. input()

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

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

Output 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"))
```

Output Enter a number7
7

33. int()

int() converts a value to an integer.

>>> int('7')

Output 7

34. 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)

Output False

>>> isinstance(orange,fruit)

Output True

35. issubclass()

This Python Built In function takes two arguments- two **python classes**. If the first class is a subclass of the second, it returns True. Otherwise, it returns False.

>>> issubclass(fruit,fruit)

Output True

>>> class fruit:

Output

1. pass
2. >>> class citrus(fruit):
3. Output pass

1. >>> issubclass(fruit,citrus)

Output False

36. iter()

Iter() Python built-in functions, returns a **python iterator** for an object.

> >>> for i in iter([1,2,3]):
>
> Output print(i)

```
1
2
3
```

37. len()

We’ve seen len() so many times by now. It returns the length of an object.

```
>>> len({1,2,2,3})
```

Output 3
Here, we get 3 instead of 4, because the set takes the value ‘2’ only once.

38. list()

list() creates a list from a sequence of values.

```
>>> list({1,3,2,2})
```

Output [1, 2, 3]

39. locals()

This function returns a dictionary of the current local symbol table.

```
>>> locals()
```

Output {‘__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’: 3, ‘x’: 7, ‘b’: 3, ‘citrus’: <class ‘__main__.citrus’>}

40. map()

Like filter(), map() Python built-in functions, takes a function and applies it on an iterable. It maps True or False values on each item in the iterable.

```
>>> list(map(lambda x:x%2==0,[1,2,3,4,5]))
```

Output [False, True, False, True, False]

41. max()

A no-brainer, max() returns the item, in a sequence, with the highest value of all.

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

Output 4

```
>>> max([3,5,4])
```

Output 5

```
>>> max('hello','Hello')
```

Output ‘hello’

42. memoryview()

memoryview() shows us the memory view of an argument.

1. >>> a=bytes(4)
2. >>> memoryview(a)

<memory at 0x05F9A988>

1. >>> for i in memoryview(a):
2. print(i)

43. min()

min() returns the lowest value in a sequence.

```
>>> min(3,5,1)
```

Output 1

```
>>> min(True,False)
```

Output False

44. next()

This Python Built In function returns the next element from the iterator.

```
>>> myIterator=iter([1,2,3,4,5])

>>> next(myIterator)
```

Output 1

```
>>> next(myIterator)

Output 2

>>> next(myIterator)

Output 3

>>> next(myIterator)

Output 4

>>> next(myIterator)

Output 5
```

Now that we’ve traversed all items, when we call next(), it raises StopIteration.

```
>>> next(myIterator)

Output Traceback (most recent call last):
File “<pyshell#392>”, line 1, in <module>
next(myIterator)
StopIteration
```

45. object()

Object() Python built-in functions, creates a featureless object.

```
>>> o=object()

>>> type(o)

<class ‘object’>

>>> dir(o)
```

Output [‘__class__’, ‘__delattr__’, ‘__dir__’, ‘__doc__’, ‘__eq__’, ‘__format__’, ‘__ge__’, ‘__getattribute__’, ‘__gt__’, ‘__hash__’, ‘__init__’, ‘__init_subclass__’, ‘__le__’, ‘__lt__’, ‘__ne__’, ‘__new__’, ‘__reduce__’, ‘__reduce_ex__’, ‘__repr__’, ‘__setattr__’, ‘__sizeof__’, ‘__str__’, ‘__subclasshook__’] Here, the function type() tells us that it’s an object. dir() tells us the object’s attributes. But since this does not have the __dict__ attribute, we can’t assign to arbitrary attributes.

46. oct()

oct() converts an integer to its octal representation.

>>> oct(7)

Output ‘0o7’

>>> oct(8)

Output ‘0o10’

>>> oct(True)

Output ‘0o1’

47. open()

open() lets us open a file. Let’s change the current working directory to Desktop.

1. >>> import os
2. >>> os.chdir('C:\\Users\\lifei\\Desktop')

Now, we open the file ‘topics.txt’.

1. >>> f=open('topics.txt')
2. >>> f

<_io.TextIOWrapper name=’topics.txt’ mode=’r’ encoding=’cp1252′>

>>> type(f)

<class ‘_io.TextIOWrapper’>
To read from the file, we use the read() method.

1. >>> print(f.read())
2. DBMS mappings
3. projection
4. union
5. rdbms vs dbms
6. doget dopost
7. how to add maps
8. OOT
9. SQL queries
10. Join
11. Pattern programs

Output

Default constructor in inheritance

48. ord()

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

>>> ord('A')

Output 65

>>> ord('9')

Output 57
This is complementary to chr().

>>> chr(65)

Output ‘A’

49. pow()

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

>>> pow(3,4)

Output 81

>>> pow(7,0)

Output 1

>>> pow(7,-1)

Output 0.14285714285714285

>>> pow(7,-2)

Output 0.02040816326530612

50. print()

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

1. >>> print("Okay, next function, please!")

Okay, next function, please!

51. property()

The function property() returns a property attribute. Alternatively, we can use the syntactic sugar @property.we can learn about this in detail in the class.

52. range()

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

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

```
Output 7
       5
       3
```

53. repr()

repr() returns a representable string of an object.

>>> repr("Hello")

Output “‘Hello’”

>>> repr(7)

Output ‘7’

>>> repr(False)

Output ‘False’

54. reversed()

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

1. >>> a=reversed([3,2,1])
2. >>> a

<list_reverseiterator object at 0x02E1A230>

1. >>> for i in a:
2. print(i)

```
Output 1
       2
       3

>>> type(a)

<class 'list_reverseiterator'>
```

55. round()

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

```
>>> round(3.777,2)

Output 3.78

>>> round(3.7,3)

Output 3.7

>>> round(3.7,-1)

Output 0.0

>>> round(377.77,-1)

380.0
```

The rounding factor can be negative.(its in matlab as well)
Fun fact: python is being using google not php and sql and drop box too

56. set()

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

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

Output {1, 2, 3}
```

Remember, a set cannot have duplicate values, and isn't indexed, but is ordered.

57. setattr()

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

```
>>> orange.size

Output 7

>>> orange.size=8

>>> orange.size

Output 8
```

58. slice()

slice() returns a slice object that represents the set of indices specified by range(start, stop, step).

```
>>> slice(2,7,2)

Output slice(2, 7, 2)
```

We can use this to iterate on an iterable like a **string in python**.

```
>>> 'Python'[slice(1,5,2)]

Output ‘yh’
```

59. sorted()

Like we've seen before, sorted() prints out a sorted version of an iterable. It does not, however, alter the iterable.

```
>>> sorted('Python')

Output [‘P’, ‘h’, ‘n’, ‘o’, ‘t’, ‘y’]

>>> sorted([1,3,2])

Output [1, 2, 3]
```

60. staticmethod()

staticmethod() creates a static method from a function. A static method is bound to a class rather than to an object. But it can be called on the class or on an object.

```
>>> class fruit:
def sayhi():
print("Hi")
>>> fruit.sayhi=staticmethod(fruit.sayhi)
>>> fruit.sayhi()
```

Output Hi
You can also use the syntactic sugar @staticmethod for this.

```
>>> class fruit:
@staticmethod
def sayhi():
print("Hi")
>>> fruit.sayhi()
```

Output Hi

61. str()

str() takes an argument and returns the string equivalent of it.

```
>>> str('Hello')
```

Output ‘Hello’

```
>>> str(7)
```

Output ‘7’

```
>>> str(8.7)
```

Output ‘8.7’

```
>>> str(False)
```

Output ‘False’

```
>>> str([1,2,3])
```

Output ‘[1, 2, 3]’

62. sum()

The function sum() takes an iterable as an argument, and returns the sum of all values.

```
>>> sum([3,4,5],3)
```

```
Output 15
```

63. super()

super() returns a proxy object to let you refer to the parent class.

```
>>> class person:
```

```
def __init__(self):
print("A person")
>>> class student(person):
def __init__(self):
super().__init__()
print("A student")
>>> Avery=student()
```

```
A person
A student
```

64. tuple()

the function tuple() lets us create a tuple.

```
>>> tuple([1,3,2])
```

```
Output (1, 3, 2)
```

```
>>> tuple({1:'a',2:'b'})
```

```
Output (1, 2)
```

65. type()

We have been seeing the type() function to check the type of object we’re dealing with.

```
>>> type({})
```

```
<class ‘dict’>
```

```
>>> type(set())
```

<class ‘set’>

```
>>> type(())
```

<class ‘tuple’>

```
>>> type((1))
```

<class ‘int’>

```
>>> type((1,))
```

<class ‘tuple’>

66. vars()

vars() function returns the __dict__ attribute of a class.

```
>>> vars(fruit)
```

mappingproxy({‘__module__’: ‘__main__’, ‘size’: 7, ‘shape’: ’round’, ‘__dict__’: <attribute ‘__dict__’ of ‘fruit’ objects>, ‘__weakref__’: <attribute ‘__weakref__’ of ‘fruit’ objects>, ‘__doc__’: None})

67. zip()

zip() returns us an iterator of tuples.

```
>>> set(zip([1,2,3],['a','b','c']))
```

{(1, ‘a’), (3, ‘c’), (2, ‘b’)}

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

{(1, 3), (2, 4)}

```
>>> a=zip([1,2,3],['a','b','c'])
```

To unzip this, we write the following code.

```
>>> x,y,z=a
>>> x
```

(1, ‘a’)

1. >>> y

(2, ‘b’)

1. >>> z

(3, ‘c’)
Isn’t this just like tuple unpacking?