

Ch03-1-Functions-Built-in

August 7, 2020

0.1 Built-in Functions

- named sequence of code that does some specific task or a function
- we'll learn how to define our own functions in [User Defined Functions chapter](#)
- some built-in functions we've used so far: `type()`, `int()`, `float()`, `str()`, `input()`, `print()`, etc.
- Python provides a list of built-in functions that are readily available for use: <https://docs.python.org/3/library/functions.html>
- below are examples of some built-in functions that may be useful to know

0.1.1 `bin(x)`

- converts an integer number to a binary string prefixed with "0b".

```
[1]: bin(3)
```

```
[1]: '0b11'
```

0.1.2 `format(value, format_spec)`

- formats the give value using format spec

```
[1]: help(format)
```

Help on built-in function format in module builtins:

```
format(value, format_spec='', /)
    Return value.__format__(format_spec)
```

`format_spec` defaults to the empty string.

See the Format Specification Mini-Language section of `help('FORMATTING')` for details.

```
[2]: # If prefix "0b is desired or not, use either of the following ways
print(format(3, '#b'))
print(format(3, 'b'))
```

```
0b11
```

```
11
```

0.1.3 chr(uniCode)

- returns the string representing a character whose Unicode code point is the integer uniCode
- inverse of ord(character)

```
[3]: print(chr(65))
      print(chr(97))
      print(chr(8364))
```

A
a
€

0.1.4 globals() and locals()

- globals() returns a dictionary representing the current global symbol table
- locals() returns a dictionary representing the current local symbol table

```
[4]: globals()
```

```
[4]: {'__name__': '__main__',
      '__doc__': 'Automatically created module for IPython interactive environment',
      '__package__': None,
      '__loader__': None,
      '__spec__': None,
      '__builtin__': <module 'builtins' (built-in)>,
      '__builtins__': <module 'builtins' (built-in)>,
      '_ih': ['',
              'bin(3)',
              '# If prefix "0b is desired or not, use either of the following
ways\nprint(format(3, \'#b\'))\nprint(format(3, \'b\'))',
              'print(chr(65))\nprint(chr(97))\nprint(chr(8364))',
              'globals()'],
      '_oh': {1: '0b11'},
      '_dh': ['/Volumes/Storage/GoogleDrive/CMU/Python/thinkpythonnotebooks'],
      '_sh': <module 'IPython.core.shadowns' from
'/Users/rbasnet/miniconda3/lib/python3.7/site-
packages/IPython/core/shadowns.py'>,
      'In': ['',
              'bin(3)',
              '# If prefix "0b is desired or not, use either of the following
ways\nprint(format(3, \'#b\'))\nprint(format(3, \'b\'))',
              'print(chr(65))\nprint(chr(97))\nprint(chr(8364))',
              'globals()'],
      'Out': {1: '0b11'},
      'get_ipython': <bound method InteractiveShell.get_ipython of
<ipykernel.zmqshell.ZMQInteractiveShell object at 0x10f035278>>,
      'exit': <IPython.core.autocall.ZMQExitAutocall at 0x10fa42668>,
      'quit': <IPython.core.autocall.ZMQExitAutocall at 0x10fa42668>,
```

```

'_': '0b11',
'__': '',
'___': '',
'_i': 'print(chr(65))\nprint(chr(97))\nprint(chr(8364))',
'_ii': '# If prefix "0b is desired or not, use either of the following
ways\nprint(format(3, \'#b\'))\nprint(format(3, \'b\'))',
'_iii': 'bin(3)',
'_i1': 'bin(3)',
'_1': '0b11',
'_i2': '# If prefix "0b is desired or not, use either of the following
ways\nprint(format(3, \'#b\'))\nprint(format(3, \'b\'))',
'_i3': 'print(chr(65))\nprint(chr(97))\nprint(chr(8364))',
'_i4': 'globals()'}

```

0.1.5 hex(x)

- convert an integer number to a lowercase hexadecimal string prefixed with “0x”

```
[5]: print(hex(42))
      print(hex(-42))
```

```

0x2a
-0x2a

```

```
[2]: # Other ways
      print(format(255, '#x'))
      print(format(255, 'x'))
      print(format(255, 'X'))
```

```

0xff
ff
FF

```

0.1.6 oct(x)

- return an octal string representation with “0o” prefix of a given integer x

```
[7]: print(oct(100))
```

```
0o144
```

```
[8]: print(format(10, '#o'))
      print(format(10, 'o'))
```

```

0o12
12

```

0.1.7 ord(c)

- return an integer representing the Unicode code of a given Unicode character

```
[9]: print(ord(' '))  
     print(ord('~'))
```

```
32  
126
```

0.1.8 id(object)

- return the ‘identity’ of an object
- guaranteed unique integer and constant throughout its lifetime

```
[10]: x = 10
```

```
[11]: id(x)
```

```
[11]: 4525196800
```

0.1.9 divmod(a, b)

- given two non-complex numbers as arguments, return a pair of numbers as tuple consisting of their quotient and remainder using integer division

```
[12]: print(divmod(7, 3)) # return (quotient, remainder)  
     quotient, remainder = divmod(7, 3)  
     print(quotient, remainder)
```

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

0.1.10 eval(expression, globals=None, locals=None)

- the expression argument is parsed and evaluated as Python expression
- syntax errors reported as exceptions

```
[13]: y = 2  
     print(eval('y**2'))  
     print(eval('y+2*3'))
```

```
4  
8
```

0.1.11 max(iterable, ...) or max(arg1, arg2, ...)

- returns the largest item in an iterable or the largest of two or more arguments

```
[14]: print('max=', max(100, 8.9, 999, 1000.5))
```

```
max= 1000.5
```

0.1.12 min(arg1, arg2, ...)

- returns the smallest of the the arguments (arg1, arg2...)

```
[15]: print('min=', min(100, 8.9, 999, 1000.5))
```

```
min= 8.9
```

0.1.13 pow(base, exponent)

- returns base to the power exponent

```
[16]: print('2 to the power 3 =', pow(2, 3))
```

```
2 to the power 3 = 8
```

0.1.14 print()

- print(*object, sep=' ', end='\n', file=sys.stdout, flush=False)
- print takes many arguments and print arbitrary number of values
- demo some print examples with different argument values

```
[17]: print('hi', 'hello', sep='', end='')
```

```
hihello
```

```
[19]: print('hi', 'hello', sep=' ', end='')
```

```
hi hello
```

```
[20]: print('hi', 'hello', sep=' ', end='')  
print('next line?')
```

```
hi hellonext line?
```

```
[21]: print('hi', 'hello', sep='\t', end='\n')  
print('next line?')
```

```
hi      hello  
next line?
```

```
[ ]:
```