Useful Builtins

zip

In [1]:
```
zip([1,2,3], ['a', 'b', 'c'])
```

Out[1]:
```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

In [3]:
```
x = [[ 1, 2, 3 ],
     [ 4, 5, 6 ],
     [ 7, 8, 9 ] ]
```

In [5]:
```
zip(x[0], x[1], x[2])
```

Out[5]:
```
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

In [6]:
```
zip(*x)
```

Out[6]:
```
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

enumerate

In [7]:
```
lst = [ 'foo', 'bar', 'baz' ]
```

In [8]:
```
# Don't do this:
for x in range(len(lst)):
    print x, lst[x]
```

```
0 foo
1 bar
2 baz
```

In [11]:
```
# Do this instead:
for index, x in enumerate(lst):
    print index, x
```

```
0 foo
1 bar
2 baz
```

eval

In [18]:
```
eval('5+5')
```

Out[18]:
```
10
```

In [19]:
```
x = [1,2,3]
r_x = repr(x)
eval(r_x)
```

Out[19]: [1, 2, 3]

dir

In [23]:
```
x = 5
dir(x)
```

Out[23]:
```
['__abs__',
 '__add__',
 '__and__',
 '__class__',
 '__cmp__',
 '__coerce__',
 '__delattr__',
 '__div__',
 '__divmod__',
 '__doc__',
 '__float__',
 '__floordiv__',
 '__format__',
 '__getattribute__',
 '__getnewargs__',
 '__hash__',
 '__hex__',
 '__index__',
 '__init__',
 '__int__',
 '__invert__',
 '__long__',
 '__lshift__',
 '__mod__',
 '__mul__',
 '__neg__',
 '__new__',
 '__nonzero__',
 '__oct__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdiv__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rlshift__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__rpow__',
 '__rrshift__',
 '__rshift__',
 '__rsub__',
 '__rtruediv__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 '__trunc__',
 '__xor__',
 'bit_length',
 'conjugate',
 'denominator',
 'imag',
 'numerator',
```

In [24]:
```
x.bit_length()
```

Out[24]: 3

In [35]:
```
x = 'Foo'
dir(x)
```

Out[35]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__getslice__',
 '__gt__',
 '__hash__',
 '__init__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '_formatter_field_name_split',
 '_formatter_parser',
 'capitalize',
 'center',
 'count',
 'decode',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'index',
 'isalnum',
 'isalpha',
 'isdigit',
 'islower',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'partition',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
```

In [37]:
```
help(x.partition)
```

```
Help on built-in function partition:

partition(...)
    S.partition(sep) -> (head, sep, tail)

    Search for the separator sep in S, and return the part before it,
    the separator itself, and the part after it.  If the separator is not
    found, return S and two empty strings.
```

`ord` and `chr`

In [38]:
```
ord('a')
```

Out[38]: 97

In [39]:
```
chr(97)
```

Out[39]: 'a'

`map` and `filter`

In [41]:
```
x = map(ord, 'Foo')
x
```

Out[41]: [70, 111, 111]

In [42]:
```
map(chr, x)
```

Out[42]: ['F', 'o', 'o']

In [43]:
```
def is_even(num):
    return num % 2 == 0
filter(is_even, range(10))
```

Out[43]: [0, 2, 4, 6, 8]

`sum`, `max`, `min`, and `len`

In [44]:
```
sum(range(10))
```

Out[44]: 45

In [45]:
```
max(range(10))
```

Out[45]: 9

In [46]:
```
min(range(10))
```

Out[46]: 0

In [47]:
```
len(range(10))
```

Out[47]: 10

repr

In [48]:
```
repr(1)
```

Out[48]: '1'

In [49]:
```
repr('foo')
```

Out[49]: "'foo'"

In [50]:
```
repr([1,2,4])
```

Out[50]: '[1, 2, 4]'

Basic types

In [51]:
```
int('5')
```

Out[51]: 5

In [52]:
```
int('ff', base=16)
```

Out[52]: 255

In [53]:
```
float('5')
```

Out[53]: 5.0

In [54]:
```
float(5)
```

Out[54]: 5.0

In [57]:
```
list(), list([1,2,3])
```

Out[57]: ([], [1, 2, 3])

In [58]:
```
tuple([1,2,3])
```

Out[58]: (1, 2, 3)

In [61]:
```
keys = range(4)
values = ['a', 'b', 'c', 'd']
dct = dict(zip(keys, values))
dct
```

Out[61]:
```
{0: 'a', 1: 'b', 2: 'c', 3: 'd'}
```

In [62]:
```
dict(foo=1, bar=2, baz=3)
```

Out[62]:
```
{'bar': 2, 'baz': 3, 'foo': 1}
```

In [63]:
```
unicode('abcd')
```

Out[63]:
```
u'abcd'
```

Exercises

- Given that you have a list of keys and a list of values, how would you create a `dict` containing the key/value pairs
- Write a function that converts a list of ASCII values to a string. Test it on the string `[86, 77, 87, 97, 114, 101]`