PyCon 2015 - Python by Immersion

Overview

This tutorial, presented at PyCon 2015 in Montreal by Stuart Williams, is intended for software developers who want a fast introduction to a lot Python.

You'll learn by seeing and doing. We'll mostly use the interactive Python interpreter prompt, aka the Read Eval Print Loop (REPL). I'll be using Python version 3.4 but most of this will work identically in earlier versions.

Most exercise sections start out simple but increase quickly in difficulty in order to give more advanced students a challenge, so don't expect to finish all the exercises in each section. I encourage you to revisit them later.

I encourage you to *not* copy and paste code from this document when you do the exercises. By typing the code you will learn more. Also pause before you hit the Enter key and try to predict what will happen.

Python by Immersion

Python by Immersion Tutorial
Presented at PyCon 2015
April 9th, 2015
Montreal, Quebec, Canada

Stuart Williams
`stuart@swilliams.ca`
`@ceilous`

Exercises: `http://bit.ly/1Iuu9PR`
Tutorial Survey: `http://bit.ly/1HSYfci`

Objects

```
>>> # Create objects via literals                                    0
>>> 1                                                               1
>>> 3.14                                                            2
>>> 'hello'                                                         3
>>> (1, 2, 3)                                                       4
>>> [1, 2, 3]                                                       5
```

Everything in Python (at runtime) is an object and has:

- a single *value*,
- a single *type*,
- some number of *attributes*,
- one or more *base classes*,
- a single *id*, and
- (zero or) one or more *names* (in one or more namespaces).

```
>>> # Object have types                                            6
>>> type(1)                                                        7
>>> type(3.14)                                                     8
>>> type('hello')                                                  9
>>> type((1, 2, 3))                                               10
>>> type([1, 2, 3])                                               11
```

```
>>> # Objects have attributes
>>> True.__doc__
>>> 'hello'.upper
>>> callable('hello'.upper)
>>> 'hello'.upper()

>>> # Objects have base classes
>>> import inspect
>>> inspect.getmro(type('hello'))
>>> inspect.getmro(type(True))

>>> # Every object has one id (memory address in CPython)
>>> id(3)
>>> id(3.14)
>>> id('hello')

>>> # Create objects by calling an object (function, method, class)
>>> abs
>>> callable(abs)
>>> abs(-3)
>>> abs(3)
>>> int
>>> callable(int)
>>> int(3.14)
```

Names

```
>>> # We can add names to refer to objects
>>> dir()
>>> def __names():
...     return dict([(k, v) for (k, v) in globals().items()
...     if not k.startswith('__')])

>>> __names()
>>> a
>>> a = 300
>>> __names()
>>> a
>>> a = 400
>>> __names()
>>> a
>>> b = a
>>> b
>>> a
>>> __names()
>>> id(a)
>>> id(b)
>>> a is b
>>> a = 'hello'
>>> a
>>> b
>>> del a
>>> __names()
>>> del b

>>> # 'is' checks identity (via 'id'), not equality
>>> i = 10
>>> j = 10
>>> i is j

>>> i = 500
>>> j = 500
>>> i is j
```

Exercises: Objects and Names

Restart Python to unclutter the local namespace.

```
>>> i
```

```
>>> dir()
>>> i = 1
>>> i
>>> dir()
>>> type(i)
>>> j = i
>>> i is j

>>> m = n = [1, 2, 3]
>>> m
>>> n
>>> m is n
>>> dir()
>>> m[1] = 'two'
>>> m
>>> n
```

Numbers, etc.

```
>>> 1
>>> -1
>>> 1-
>>> 1 = 2
>>> 1 == 2
>>> 1 != 2
>>> 1 < 2
>>> 1 <= 1
>>> 1 > 2
>>> 1 * 2
>>> 1 + 2
>>> 1 / 2.0
>>> 1 / 2
>>> 1 // 2.0
>>> 1 // 2
>>> from __future__ import division
>>> 1 / 2.0
>>> 1 / 2
>>> 1 // 2.0
>>> 1 // 2
>>> 9 % 3
>>> 10 % 3

>>> int
>>> callable(int)
>>> int(2)
>>> int(2.0)
>>> int(2.1)
>>> int(2.9)
>>> int('2')
>>> int('2.0')
>>> int('four')

>>> float(2)
>>> float('2')
>>> float('2.9')

>>> 1 / 0

>>> 1 + 1.0

>>> dir()
>>> a = 1
>>> dir()
>>> a
>>> a = a + 1
>>> a
>>> a += 1
>>> a
>>> a++

>>> abs(-5)
>>> abs(5)
```

```
>>> sin(pi / 2)
>>> dir()
>>> import math
>>> sin(pi / 2)
>>> dir()
>>> dir(math)
>>> math.sin(math.pi / 2.0)
>>> from math import sin, pi
>>> sin(pi / 2)
>>> help(math.sin)
```

Exercises: Numbers

Now it's your turn. Use the Python interpreter to see what happens (and what you can learn) when you type in the following expressions. Try to predict what will be displayed.

```
>>> 1
>>> 1 + 2
>>> 1 + 2 == 3
>>> 1 + 2 < 3
>>> 1 + 2 <= 3

>>> 2 ** 8
>>> 2 ** 16

>>> round(1.01)
>>> round(1.99)
>>> round(1.50)
>>> third = 1/3.0
>>> third

>>> round(third)
>>> round(third, 1)
>>> round(third, 2)
>>> round(third, 3)

>>> round(1234.56, -1)
>>> round(1234.56, -2)
>>> round(1234.56, -3)

>>> temp = 20
```

Write an expression that evaluates to `True` if `temp` is greater than zero and less than 100.

Advanced exercises:

```
>>> 0 == 0
>>> 0 is 0
>>> type(int)
>>> callable(int)
>>> help(callable)
>>> int()
>>> 0 == int()
>>> 0 is int()
>>> type(int)
>>> type(int())
>>> int(4.3)
>>> int('4')

>>> int('four')
>>> int('z')
>>> int('c', 16)
```

Strings

```
>>> 'hello'
>>> "hello"
>>> 'today's the day'
```

```
>>> 'today\'s the day'
>>> "today's the day"
>>> 'A quote (") mark'
>>> 'hello
>>> """hello
...  there"""

>>> """today's the "day""""
>>> '''today's the "day"'''
>>> 'hello\nthere\n'
>>> 'h' in 'hello'
>>> 'h' not in 'hello'
>>> 'hello'[0]
>>> dir()
>>> s = t = 'hello'
>>> dir()
>>> s, t
>>> s = 'j' + 'ello'
>>> s
>>> t
>>> s[0] = 'h'
>>> s

>>> s1 = 'hello'
>>> u1 = u'hello'
>>> s1 == u1
>>> s1 is u1
>>> type(s1)
>>> type(u1)
```

Exercises: Strings

```
>>> r'hello'
>>> r'hello' is 'hello'

>>> r'hello\n'
>>> r'hello\n' == 'hello\n'
>>> len(r'hello\n')
>>> len('hello\n')

>>> s = 'moby'
```

Write an expression that evaluates to `True` if `m` refers to a string with at least 3 characters.

```
>>> 2 * 'hello'
>>> 2 + 'hello'
>>> '2' + 'hello'
>>> 'hello' + 'there'
>>> 'hello' 'there'

>>> type('hello')
>>> u'hello'
>>> type(u'hello')

>>> s2 = '$'
>>> s2
>>> s3 = '\u0024'
>>> s3
>>> s4 = '\N{dollar sign}'
>>> s4

>>> s2 == s2 == s3
```

Attributes:

```
>>> i = 12
>>> dir(i)
>>> i.__doc__
>>> i.__str__
>>> i.__str__()
```

```
>>> s2                                                                  228
>>> s2.encode                                                           229
>>> s2.encode()                                                         230
>>> s2.encode(encoding='ascii')                                         231
>>> s2.encode() == '$'                                                  232

>>> type(s2)                                                            233
>>> type(s2.encode())                                                   234

>>> s5 = 'Pound sign:\N{pound sign}:'                                   235
>>> s5.encode()                                                         236
>>> s5.encode('ignore')                                                 237
>>> s5.encode(errors='ignore')                                          238
>>> s5.encode('ascii', errors='ignore')                                 239
>>> s5.encode('ascii', 'ignore')                                        240
>>> s5.encode('ascii', 'replace')                                       241
```

String Functions and Methods

```
>>> len('hello')                                                        242
>>> min('hello')                                                        243
>>> max('hello')                                                        244
>>> sorted('hello')                                                     245

>>> 'hello'.startswith('h')                                             246
>>> 'hello'.startswith('he')                                            247
>>> 'hello'.endswith('lo')                                              248

>>> 'hello'.upper()                                                     249
>>> 'HELLO'.lower()                                                     250

>>> ' hello '.strip()                                                   251
>>> ' hello '.rstrip()                                                  252
>>> ' hello '.lstrip()                                                  253

>>> 'Jan Feb Mar'.split()                                               254
>>> 'one, two, three'.split(', ')                                       255

>>> type('hello')                                                       256
>>> dir(str)                                                            257
>>> import textwrap                                                     258
>>> print(textwrap.fill(str(dir(str)), width=60))                       259

>>> dir('hello')                                                        260
>>> dir(str) == dir('hello')                                            261

>>> 'hello'.index('el')                                                 262
>>> 'hello'.index('z')                                                  263
>>> 'hello'.find('el')                                                  264
>>> 'hello'.find('z')                                                   265
>>> 'hello'.find('h')                                                   266
>>> 'hello'.find('e')                                                   267
>>> 'hello'.find('l')                                                   268

>>> 'hello'.replace('h', 'z')                                           269
>>> 'steep'.replace('e', '3')                                           270
>>> 'steep'.replace('e', '3', 1)                                        271
```

Exercises: String Functions and Methods

```
>>> sorted('hello')                                                     272
>>> sorted('hello', reverse=True)                                       273
>>> reversed('hello')                                                   274
>>> list(reversed('hello'))                                             275

>>> 'hello'.upper()                                                     276
>>> 'HELLO'.isupper()                                                   277

>>> 'hello'.title()                                                     278
>>> 'Hello'.istitle()                                                   279
```

```
>>> 'hello world'.title()
>>> 'hello world'.title().swapcase()

>>> 'he' in 'hello'
>>> '!' in '.?!'
```

Write a predicate (boolean) expression for a sentence, checking that it starts with a capital letter and ends with punctuation.

```
>>> 'yellow is mellow'.find('ow')
>>> 'yellow is mellow'.find('ow', 6)
```

What does the following expression check for?

```
s.index('e', 0, s.index(' '))
```

Hint: See `help(str.index)` and try substituting `'hello world'` for `s` in the expression.

Write an expression to check for more spaces than non-spaces in a string.

```
>>> 'Wait...'.rstrip('.')
```

Print and String Formatting

```
>>> 3
>>> print(3)
>>> print(3, 2)
>>> print('three', 4)
>>> 'hello\n'
>>> print('hello\n')
>>> print('hello\nthere\n')

>>> print('%d good reasons' % 3)
>>> print('{0} good reasons'.format(3))
>>> '{0} good reasons'.format(3)

>>> 'Hello'.format
>>> 'Hello'.format()
>>> 'Hello'.format('there')
>>> 'Hello {0}'.format()
>>> 'Hello {0}'.format('Stu')
>>> '{0} {1}'.format('Hi', 'Stu')
>>> '{} {}'.format('Hi', 'Stu')
>>> '{1} {0}'.format('Hi', 'Stu')
>>> '{0} {1}, {0}!'.format('Hi', 'Stu')

>>> '{0:d}'.format(99)
>>> '{0:10d}'.format(99)
>>> '{0:>10d}'.format(99)
>>> '{0:<10d}'.format(99)
>>> '{0:^10d}'.format(99)

>>> '{greet} {who}'.format(
...     greet='Hi',
...     who='Stu')
```

Exercises: Print and String Formatting

```
>>> # from __future__ import print_function  # If Python 2.x
>>> print(3)
>>> print(3, 4, 5, sep=':', end='$\n')

>>> 'Take {0} or {1}'.format(3, 4)
>>> 'Take {1} or {0}'.format(3, 4)

>>> val = 1/3.0
>>> '{0:f}'.format(val)
>>> '{0:4.2f}'.format(val)
>>> '{0:7.2f}'.format(val)
```

```
>>> '{0:7.4f}'.format(val)
```

Write a string literal and call its `format` method to produce the string "The value is 0.33" using `val`.

```
>>> '{0:b}'.format(2)
>>> '{0:b}'.format(15)
>>> '{0:b}'.format(16)
>>> '{0:x}'.format(65535)
>>> '{0:o}'.format(65535)

>>> '{0:%}'.format(0.35)
>>> '{0:5.2%}'.format(0.35)

>>> '{0:10.{1}f}'.format(val, 3)
>>> '{0:10.{1}f}'.format(val, 5)

>>> import sys
>>> print(sys.version_info)
>>> 'major {0[0]}, minor {0[1]}'.format(
...     sys.version_info)

>>> sys.byteorder
>>> 'Byte order: {0.byteorder}' \
...     .format(sys)
```

Use `format` to print the `byteorder` and `maxint` from the `sys` module. Then add the 4th element of `sys.version_info` to the string.

Tuples, Lists

```
>>> m = [1, 2, 3]
>>> m
>>> t = (1, 2, 3)
>>> t
>>> list(t)
>>> m
>>> m[1] = 'b'
>>> m
>>> tuple(m)
>>> t
>>> t[1] = 'b'
>>> t
>>> t = (1, 'b', 3)
>>> t
>>> m + ['d']
>>> m
>>> t
>>> t + 'd'
>>> t + ('d')

>>> ('d')
>>> (('d'))
>>> ((('d')))
>>> tuple('d')
>>> ('d',)
>>> t + ('d',)
>>> t
>>> t * 2
>>> m * 2
>>> tuple()
>>> type(tuple())
>>> t = ()
>>> t
>>> type(t)
>>> type(())

>>> m
>>> len(m)
>>> min(m)
>>> m = [1, 2, 3]
>>> max(m)
>>> sorted(m)
```

```
>>> reversed(m)
>>> list(reversed(m))
>>> reversed('hello')
>>> list(reversed('hello'))

>>> m
>>> m.insert(0, 6)
>>> m
>>> m.remove(2)
>>> m
>>> m.pop()
>>> m

>>> (i, j) = (1, 2)
>>> i
>>> j
>>> i, j
>>> i, j = 3, 4
>>> i, j

>>> t1 = (1, 2, 3)
>>> t1
>>> t2 = 1, 2, 3
>>> t2
>>> t1 == t2

>>> t1.count(3)
>>> t1.index(3)

>>> t3, t4, *rest = [3, 4, 5, 6]
>>> t3, t4, rest

>>> t3, *rest, t6 = [3, 4, 5, 6]
>>> t3, rest, t6
```

Exercises: Tuples, Lists

Create a list containing two numbers and two strings. Give it the name `mylist`. Give it a second name `mylist2`.

```
>>> m = [1, 2, 3]
>>> m
>>> m += 'd'
>>> m
>>> m.append('e')
>>> m
>>> m.append(5, 5, 6, 6, 7)
>>> m.append([5, 5, 6, 6, 7])
>>> m
>>> del m[-1]
```

Remove the strings from your list `mylist`.

```
>>> m
>>> m.extend([5, 5, 6, 6, 7])
>>> m
>>> 5 in m
>>> 10 in m
>>> not 10 in m
>>> 10 not in m
>>> [5, 6] in m
>>> m
>>> m.append([5, 6])
>>> m
>>> [5, 6] in m

>>> n = [1, 2, 4]
>>> m < n

>>> i, j = 1, 2
>>> i, j
>>> i, j = j, i
```

```
>>> i, j
```

```
>>> i, j, k = (1, 2, 3)
>>> i, j, k
>>> i, j, k = 1, 2, 3
>>> i, j, k
>>> i, j, k = [1, 2, 3]
>>> i, j, k
>>> i, j, k = 'ijk'
>>> i, j, k
```

```
>>> i, j, k, *rest = 'ijklmnop'
>>> i, j, k, rest
```

```
>>> r = 'one two three'.split()
>>> r
>>> ' '.join(r)
>>> ', '.join(r)
```

```
>>> m = list(range(10))
>>> m
>>> m.reverse()
>>> m
```

The `reverse` and `sort` *methods* mutate a `list` and return `None`.

The `reversed` and `sorted` *functions* don't mutate a sequence, and they return a new sequence (actually an *iterator*).

```
>>> m.sorted()
>>> m
>>> sorted(m)
>>> m
>>> m.sort()
>>> m
>>> m.sort(reverse=True)
>>> m
```

Sequence Indexing and Slicing

```
>>> months = ['jan', 'feb', 'mar', 'apr', 'may']
```

```
>>> months[0]
>>> months[3]
>>> months[-1]
>>> months[-2]
>>> months[0:1]
>>> months[0:2]
>>> months[0:-1]
>>> months[0:100]
>>> months2 = months[:]
>>> months2
>>> months == months2
>>> months is months2
>>> id(months), id(months2)
>>> help(id)
>>> del months2[0]
>>> months
>>> months2
```

```
>>> months[0] = 'January'
>>> months
>>> months[-1] = months[-1].capitalize()
>>> months
>>> del months[2]
>>> months
```

```
>>> m = [ ['one', 'two', 'three'],
...       ['ONE', 'TWO', 'THREE']]
```

```
>>> m
>>> m[0]
```

```
>>> m[1]
>>> m[0][0]
```

Exercises: Sequence Indexing and Slicing

```
>>> m = [0, 1, 2]
>>> m[1] = [10, 20]
>>> m
>>> m = [0, 1, 2]
>>> m[1:2] = [10, 20]
>>> m
```

Create a list with the first element 'January' and the second element a list of the two numbers 1 and 31.

```
>>> range(10, 20)
>>> range(10, 20, 3)
>>> range(0, 100, 10)
>>> range(100, 0, -10)
>>> range(100)[100::-10]
>>> range(101)[-1:1:-10]
```

Note that indexing and slicing work on strings and tuples, too, but remember they are immutable.

List Comprehensions

```
>>> range(8)
>>> [e for e in range(8)]
>>> [2 * e for e in range(8)]
>>> [2 + e for e in range(8)]
>>> [e for e in range(8) if e % 2 == 0]

>>> ['{0} * 2 == {1}'.format(e, 2 * e)
...     for e in range(8)]

>>> ['{0} * 2 == {1}'.format(e, 2 * e)
...     for e in range(8) if e % 2 == 0]

>>> [e for e in range(8) if e % 3 == 0]
```

Exercises: List Comprehensions

```
>>> 'Jan Feb Mar'.split()

>>> [m for m in 'Jan Feb Mar'.split()]

>>> [(n, m) for n in range(3) for m in 'Jan Feb Mar'.split()]

>>> help(enumerate)
>>> [(10 * n, c) for (n, c) in enumerate(['a', 'b', 'c'])]

>>> help(zip)
>>> list(zip('Jan Feb Mar'.split(), (1, 2, 3)))

>>> list(zip('Jan Feb Mar'.split(), (1, 2, 3, 4)))

>>> list(zip('Jan Feb Mar Apr'.split(),
...     (1, 2, 3, 4),
...     (31, 28, 31, 30)))
```

Sort

```
>>> months = [ # In alphabetical order.
...     ('Feb', 2, 28),
...     ('Jan', 1, 31),
```

```
...     ('Mar', 3, 31),
...     ]
```

```
>>> months                                                                    517
```

```
>>> sorted(months)                                                            518
>>> sorted(months, reverse=True)                                              519
```

```
>>> months                                                                    520
>>> from operator import itemgetter                                           521
>>> get1 = itemgetter(1)                                                      522
>>> get1(months)                                                              523
>>> get1(months[0])                                                           524
>>> get1(months[2])                                                           525
>>> sorted(months, key=get1)                                                  526
>>> sorted(months, key=itemgetter(1))                                         527
```

Exercises: Sort

```
>>> months = list(zip(                                                        528
...     'Jan Feb Mar Apr'.split(),
...     range(1, 100),
...     [31, 28, 31, 30]))
```

```
>>> months                                                                    529
```

Use `operator.itemgetter` and `sort`'s `key` parameter to sort `months` by the number of days in the month.

Sort `months` alphabetically.

```
>>> help(list.sort)                                                           530
```

Sort `months` in reverse order.

Dictionaries and Sets

```
>>> int_to_month = [None, 'Jan', 'Feb', 'Mar']                                531
```

```
>>> int_to_month[1]                                                           532
>>> int_to_month[2]                                                           533
>>> int_to_month[3]                                                           534
```

```
>>> month_to_int = { 'Jan': 1, 'Feb': 2, 'Mar': 3 }                          535
```

```
>>> month_to_int                                                              536
```

```
>>> month_to_int['Jan']                                                       537
>>> month_to_int['Feb']                                                       538
>>> month_to_int['Mar']                                                       539
>>> month_to_int['Apr']                                                       540
>>> month_to_int['Apr'] = 4                                                   541
>>> month_to_int['Apr']                                                       542
>>> month_to_int                                                              543
```

```
>>> 'Feb' in month_to_int                                                     544
>>> month_to_int                                                              545
>>> del month_to_int['Feb']                                                   546
>>> month_to_int                                                              547
>>> 'Feb' in month_to_int                                                     548
```

```
>>> help(dict.fromkeys)                                                       549
>>> d = {}                                                                    550
>>> d['m'] = 1                                                                551
>>> d                                                                         552
>>> d['i'] = 1                                                                553
>>> d                                                                         554
>>> d['s'] = 1                                                                555
>>> d                                                                         556
>>> d['s'] = 1                                                                557
```

```
>>> d
>>> d['i'] = 1
>>> d['s'] = 1
>>> d['s'] = 1
>>> d['i'] = 1
>>> d['p'] = 1
>>> d['p'] = 1
>>> d['i'] = 1
>>> d
>>> del d

>>> list('mississippi')
>>> d = dict.fromkeys('mississippi', 1)
>>> d
>>> d.keys()

>>> import collections
>>> dd = collections.defaultdict(int)
>>> int()
>>> for c in 'mississippi':
...     dd[c] += 1

>>> dd
>>> dd.items()

>>> int_to_month
>>> int_to_month_dict = { 1: 'Jan', 2: 'Feb', 3: 'Mar'}
>>> int_to_month_dict[2]

>>> set('mississipi')
>>> set('assiniboine')
>>> set('assiniboine') ^ set('mississipi')
>>> set('assiniboine') | set('mississipi')
>>> set('mississipi') == {'i', 'm', 'p', 's'}

>>> {1: 'one', 2: 'two'}
>>> {(1,): 'one', (2,): 'two'}
>>> {[1]: 'one', [2]: 'two'}
```

Exercises: Dictionaries and Sets

```
>>> d = {'Jan': 1, 'Feb': 2, 'Mar': 3}
>>> d['Feb']
>>> d['Apr'] = 4
>>> d.keys()
>>> d.values()
>>> d.items()
>>> d.pop('Jan')
>>> d.popitem()

>>> dict(Jan=1, Feb=2, Mar=3, Apr=4)

>>> dict([(k, v + 1) for v, k in enumerate(
...     'Jan Feb Mar Apr'.split())])
```

Create a dictionary that maps all the words 'dict', 'hash', 'map', 'mapping', 'hash map', and 'associative array' to the word 'dict'. Write an expression that uses the dictionary.

```
>>> s1 = set(0, 1, 2)
>>> s1 = set([0, 1, 2])
>>> s2 = set(range(4))
>>> s1
>>> s2
>>> s1 < s2
>>> s1 <= s2
>>> s1 < s1
>>> s1 <= s1
>>> s1.symmetric_difference(set(range(1, 4)))
>>> help(s1.symmetric_difference)
```

Dictionaries Example

An example of leveraging functions as first class objects to create a simple calculator.

```
>>> 7+3
>>> import operator
>>> operator.add(7, 3)

>>> expr = '7+3'
>>> lhs, op, rhs = expr
>>> lhs, op, rhs
>>> lhs, rhs = int(lhs), int(rhs)
>>> lhs, op, rhs
>>> op, lhs, rhs
>>> operator.add(lhs, rhs)

>>> ops = {
...     '+': operator.add,
...     '-': operator.sub,
...     }

>>> ops[op] (lhs, rhs)

>>> def calc(expr):
...     lhs, op, rhs = expr
...     lhs, rhs = int(lhs), int(rhs)
...     return ops[op] (lhs, rhs)

>>> calc('7+3')
>>> calc('9-5')
>>> calc('8/2')
>>> # scope creep, need to add code
>>> ops['/'] = operator.truediv
>>> calc('8/2')
```

Blocks, for loops

```
>>> print('hello')
>>>  print('there')

>>> i = 0
>>> while i < 5:
...     i += 1
...     print(i)

>>> temp = 15
>>> if temp <= 0:
...     print('Freezing')
... elif temp < 10:
...     print('Cold')
... elif temp < 20:
...     print('Temperate')
... else:
...     print('Warm')

>>> for i in [1, 2, 3]:
...     print(i)
...     print(i * 2)

>>> for i in range(3):
...     for j in range(3):
...         print(i, j)

>>> for i in range(3):
...     for j in range(3):
...         print(i, j)

>>> # Note end=None means default
>>> for i in range(3):
...     for j in range(3):
```

```
...         print(i, j, sep=', ', end='')
...     print()
```

```
>>> for i in (1, 2, 3):
...     print(i)
```

```
>>> d = {'zero': 0, 'one' : 1, 'two' : 2}
>>> for k, v in d.items():
...     print('{0} -> {1}'.format(k, v))
```

```
>>> months = 'jan feb mar apr may'.split()
```

```
>>> for m in reversed(months):
...     print(m)
```

Exercises: Blocks, for loops

```
>>> if []:
...     print('list non-empty')
```

```
>>> if [None]:
...     print('list non-empty')
```

```
>>> if '':
...     print('string non-empty')
```

```
>>> if 'False':
...     print('string non-empty')
```

```
>>> d = dict(one=1, two=2, three=3)
>>> for k in d:
...     print(k)
```

```
>>> for k in sorted(d):
...     print(k)
```

Use the string `'Jan Feb Mar Apr'`, the `split` method on strings, and the `enumerate` function to print "1 Jan", "2 Feb", etc.

Advanced exercises:

```
>>> for k in reversed(d):
...     print(k)
```

```
>>> for k in reversed(d.items()):
...     print(k)
```

```
>>> for k in reversed(list(d.items())):
...     print(k)
```

```
>>> range(10)
>>> list(range(10))
>>> help(range)
>>> list(range(5, 15))
>>> list(range(5, 15, 3))
>>> list(range(15, 5, -3))
```

Iterables, Generator Expressions

- In a `for` loop the expression is evaluated to get an *iterable*, and then `iter()` is called to produce an *iterator*.
- The iterator's `__next__()` method is called repeatedly until `StopIteration` is raised.

- `iter(foo)`
  - If `foo.__iter__()` exists it is called.
  - Else if `foo.__getitem__()` exists, calls it starting at zero, handles `IndexError` by raising `StopIteration`.

```
>>> m = [1, 2, 3]
>>> reversed(m)
>>> it = reversed(m)
```

```
>>> type(it)
>>> dir(it)
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()

>>> for i in m:
...     print(i)

>>> m.__next__()
>>> it = iter(m)
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()

>>> m.__getitem__(0)
>>> m.__getitem__(1)
>>> m.__getitem__(2)
>>> m.__getitem__(3)

>>> it = reversed(m)
>>> it2 = it.__iter__()
>>> hasattr(it2, '__next__')

>>> m = [2 * i for i in range(3)]
>>> m
>>> type(m)

>>> mi = (2 * i for i in range(3))
>>> mi
>>> type(mi)
>>> hasattr(mi, '__next__')
>>> dir(mi)
>>> help(mi)
>>> mi.__next__()
>>> mi.__next__()
>>> mi.__next__()
>>> mi.__next__()
```

Exercises: Iterables, Generator Expressions

```
>>> m = [1, 2, 3]
>>> it = iter(m)
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()

>>> for n in m:
...     print(n)

>>> it = iter(m)
>>> it2 = iter(it)
>>> list(it2)
>>> list(it)

>>> it1 = iter(m)
>>> it2 = iter(m)
>>> list(it1)
>>> list(it2)
>>> list(it1)
>>> list(it2)

>>> d = {'one': 1, 'two': 2, 'three':3}
>>> it = iter(d)
>>> list(it)

>>> mi = (2 * i for i in range(3))
```

```
>>> list(mi)
>>> list(mi)

>>> import itertools
>>> help(itertools)
```

Writing Scripts, Modules

- Start with `#!/usr/bin/env python`
- Suffix `.py` (also `.pyw` on Windows)
- Python creates `__pycache__/*.cpython-34.pyc`
- Use lowercase and valid python identifiers

play1.py:

```
#!/usr/bin/env python
x = 3
y = 2
print(x + y)
```

```
>>> import play0
>>> import play1
>>> dir(play1)
>>> play1.x
>>> play1.y
>>> play1.z
>>> play1.z = 99
>>> play1.z
>>> dir(play1)
>>> del play1.z
>>> dir(play1)

>>> import importlib
>>> help(importlib.reload)
>>> importlib.reload(play1)
```

play2.py:

```
#!/usr/bin/env python

s = 'abc'
t = 'def'
def play():
    return s + t

play()
```

```
>>> from play2 import s, t
>>> dir(play2)
>>> dir()
>>> s, t
```

play3.py:

```
#!/usr/bin/env python

def play(args):
    pass     # Put code here.

def test_play():
    pass     # Put tests here.

if __name__ == '__main__':
    test_play()   # This doesn't run on import.
```

```
>>> from play3 import *
>>> dir()
```

Exercises: Writing Scripts, Modules

Edit your own `play.py` and load it.

Defining and Calling Functions

```
>>> def iseven(n):                                                          744
...     return n % 2 == 0

>>> iseven(1)                                                               745
>>> iseven(2)                                                               746

>>> def add(x, y): return x + y                                             747

>>> add(1, 2)                                                               748

>>> def plural(w):                                                          749
...     if w.endswith('y'):
...         return w[:-1] + 'ies'
...     return w + 's'

>>> plural('word')                                                          750
>>> plural('city')                                                          751
>>> plural('fish')                                                          752
>>> plural('day')                                                           753

>>> def fact(n):                                                            754
...     """factorial(n), -1 if n < 0"""
...     if n < 0:
...         return -1
...     if n == 0:
...         return 1
...     return n * fact(n - 1)

>>> fact.__doc__                                                            755
>>> help(fact)                                                              756
>>> fact(-1)                                                                757
>>> fact(0)                                                                 758
>>> fact(1)                                                                 759
>>> fact(2)                                                                 760
>>> fact(3)                                                                 761
>>> fact(4)                                                                 762
>>> fact(10)                                                                763
>>> fact(20)                                                                764
>>> fact(30)                                                                765
>>> fact(100)                                                               766
>>> fact(500)                                                               767
>>> fact(990)                                                               768
>>> fact(1000)                                                              769

>>> import sys                                                              770
>>> sys.getrecursionlimit()                                                 771
```

Exercises: Defining and Calling Functions

Define a function `triple(n)` in a module `triple.py` such that `triple.triple(3)` returns 9.

Import `triple` and try it out.

Extend the plural function above to handle proper nouns (that start with a capital letter) that end in 'y', for example the correct plural of "Harry" is "Harrys".

Generators

```
>>> def list123():
...     yield 1
...     yield 2
...     yield 3

>>> list123
>>> list123()
>>> list(list123())

>>> it = list123()
>>> it
>>> type(it)
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()
>>> it.__next__()

>>> for i in list123():
...     print(i)

>>> def list123():
...     for i in [1, 2, 3]:
...         yield i

>>> list123()
>>> list(list123())

>>> def factorials():
...     n = product = 1
...     while True:
...         yield product
...         product *= n
...         n += 1

>>> f = factorials()
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
```

Don't try this: `list(f)`

```
>>> # list(f)  # Don't run this

>>> for fact in factorials():
...     print(fact)
...     if len(str(fact)) > 6:
...         break  # Quit the loop
```

Compare these two versions of `evens`:

```
>>> def evens1():
...     n = 0
...     result = []
...     while n < 10:
...         result.append(n)
...         n += 2
...     return result

>>> def evens2():
...     n = 0
...     while n < 10:
...         yield n
...         n += 2
```

Note their typical use is identical:

```
>>> for num in evens1():
...     print(num)
```

```
>>> for num in evens2():
...     print(num)
```

Exercises: Generators

Write a generator that generates an infinite stream of zeros.

Write a generator that generates an infinite stream of alternating zeros and ones (i.e. 0, 1, 0, 1, ...).

Call by Reference (to Objects)

```
>>> i = j = 1
>>> i, j
>>> i = i + 1
>>> i, j

>>> m = n = [0, 1, 2]
>>> m, n
>>> m.append(3)
>>> m, n

>>> def f1(i):
...     print('Old:', i)
...     i = i + 1
...     print('New:', i)

>>> j = 3
>>> j
>>> f1(j)
>>> j

>>> def f2(m):
...     print('Old:', m)
...     m.append(3)
...     print('New:', m)

>>> n = [0, 1, 2]
>>> n
>>> f2(n)
>>> n
```

Classes and Instances

Why classes?

- Classes help us model the reality for which we're writing code.
- Object attributes are convenient, allowing us to bundle other values (objects) together.
- A class is a template for new objects that all have the same attributes.
- New instances of a class (objects) share methods (functions) from the class, and share data structure, although data values differ.

A *namespace* is a mapping from names to objects.

A *scope* is a section of Python text where a namespace is directly accessible. The namespace search order is:

1. locals, enclosing functions (or module if not in a function)
2. module, including `global`
3. built-ins

All namespaces changes (`=`, `import`, `def`, `del`, etc.) happen in the local scope.

- The `class` statement creates a new namespace and all its name assignments (e.g. function definitions) are bound to the class object.

- Instances are created by "calling" the class as in `ClassName()` or `ClassName(parameters)`

```
>>> # For Python < 3.3 use class SimpleNamespace: pass
>>> import types
>>> record = types.SimpleNamespace()
>>> record.first, record.last = 'Jane', 'Doe'
>>> record.first
>>> record.last

>>> def name(self):
...     return self.first + ' ' + self.last

>>> name(record)
```

point1.py:

```
class Point(object):
    """Example point class"""
    def __init__(self, x=0, y=0):
        # Note that self exists by now
        self.x, self.y = x, y

    def __repr__(self):
        return 'Point({0.x}, {0.y})'.format(self)

    def translate(self,
        deltax=None, deltay=None):
        """Translate the point"""
        if deltax:
            self.x += deltax
        if deltay:
            self.y += deltay
```

```
>>> from point1 import Point
>>> p1 = Point()
>>> p1
>>> p1.translate(2, 4)
>>> p1
>>> p1.translate(-3.5, 4.9)
>>> p1
>>> p2 = Point(1, 2)
>>> p2

>>> p1.__repr__()
>>> repr(p1)

>>> dir(Point)
>>> dir(p1)
>>> set(dir(p1)) - set(dir(Point))
>>> p1.__dict__
```

Exercises: Classes and Instances

Write a class `Employee` that tracks first name, last name, age, and manager.

Review: Classes

- Class creates a new namespace and a new class object, and wires them for inheritance.
- Calling the class object creates an instance.
- If attribute lookup finds a method then a method object is returned. It handles sending `self` to the function.
- Classes can be used as simple records.
- Modules and functions can also have attributes.

Exceptions

```
>>> int('four')                                                                 844
>>> try:                                                                        845
...     int('four')
... except:
...     print('caught it')

>>> try:                                                                        846
...     int('four')
... except Exception as e:
...     print('caught:\n -> {0}\n -> {1}'
...         .format(e, repr(e)))
...     save = e

>>> dir(save)                                                                   847
>>> save.args                                                                   848

>>> # https://docs.python.org/3.4/library/exceptions.html#exception-hierarchy  849
```

What was Skipped

- files and I/O
- the standard library
- subclasses
- functional programming
- more methods on `str`, `list`, `tuple`, and `dict`

Standard Library Tour

- data types: `calendar collections datetime decimal math random sets`
- file formats: `bz2 csv pickle struct zip zlib`
- text: `pprint repr textwrap`
- operating system: `os os.path shutil`
- internet: `email ftplib hashlib html web`
- programming: `difflib doctest filecmp fileinput functtools gettext glob itertools logging optparse pdb re timeit`

Standard class methods

- `__new__`, `__init__`, `__del__`, `__repr__`, `__str__`, `__format__`
- `__getattr__`, `__getattribute__`, `__setattr__`, `__delattr__`, `__call__`, `__dir__`
- `__len__`, `__getitem__`, `__missing__`, `__setitem__`, `__delitem__`, `__contains__`, `__iter__`, `__next__`
- `__lt__`, `__le__`, `__gt__`, `__ge__`, `__eq__`, `__ne__`, `__cmp__`, `__nonzero__`, `__hash__`
- `__add__`, `__sub__`, `__mul__`, `__div__`, `__floordiv__`, `__mod__`, `__divmod__`, `__pow__`, `__and__`, `__xor__`, `__or__`, `__lshift__`, `__rshift__`, `__neg__`, `__pos__`, `__abs__`, `__invert__`, `__iadd__`, `__isub__`, `__imul__`, `__idiv__`, `__itruediv__`, `__ifloordiv__`, `__imod__`, `__ipow__`, `__iand__`, `__ixor__`, `__ior__`, `__ilshift__`, `__irshift__`
- `__int__`, `__long__`, `__float__`, `__complex__`, `__oct__`, `__hex__`, `__coerce__`
- `__radd__`, `__rsub__`, `__rmul__`, `__rdiv__`, etc.
- `__enter__`, `__exit__`

```
>>> class UpperAttr:                                                            850
...     """
...     A class that returns uppercase values
...     on uppercase attribute access.
...     """
...     def __getattr__(self, name):
...         if name.isupper():
...             if name.lower() in self.__dict__:
...                 return self.__dict__[
...                     name.lower()].upper()
...         raise AttributeError(
...             "'{}' object has no attribute {}."
...             .format(self, name))
```

```
>>> d = UpperAttr()
>>> d.__dict__
>>> d.foo = 'bar'
>>> d.foo
>>> d.__dict__
>>> d.FOO
>>> d.baz
```

Subclassing

cpoint1.py:

```
from point2 import Point
class ColorPoint(Point):
    """Example color point class"""
    def __init__(self, x=0, y=0, color=None):
        super(ColorPoint, self).__init__(x, y)
        self.color = color

    def __repr__(self):
        return ('ColorPoint({0.x}, {0.y}, {0.color})'
                .format(self))
```

```
>>> from cpoint1 import ColorPoint
>>> c1 = ColorPoint()
>>> c1
>>> c1.translate(2, 4)
>>> c1
>>> c2 = ColorPoint(9, 11, 'blue')
>>> c2
>>> c1 - c2
```

Function Parameters

```
>>> def f3(arg1, arg2, kwarg1=0, kwarg2=0):
...     print('arg1: {0}, arg2: {1}, '
...           'kwarg1: {2}, kwarg2: {3}'
...           .format(arg1, arg2, kwarg1, kwarg2))

>>> f3(1, 2)
>>> f3(1, 2, 3)
>>> f3(1, 2, kwarg2=4)
>>> f3(1, kwarg1=3)

>>> def f4(arg1, arg2, kwarg1=0, kwarg2=0,
...        *args, **kwargs):
...     print('arg1: {0}, arg2: {1}, '
...        'kwarg1: {2}, kwarg2: {3}'
...        .format(arg1, arg2, kwarg1, kwarg2))
...     if args:
...         print('args:', str(args))
...     if kwargs:
...         print('kwargs:', kwargs)

>>> f4(1, 2)
>>> f4(arg1=1, arg2=2)
>>> f4(arg2=1, arg1=2)
>>> f4(1, 2, 3)
>>> f4(1, 2, kwarg2=4)
>>> f4(1, kwarg1=3)
>>> f4(1, 2, 3, 4, 5, 6)
>>> f4(1, 2, 3, 4, keya=7, keyb=8)
>>> f4(1, 2, 3, 4, 5, 6, keya=7, keyb=8)

>>> tuple12 = 1, 2
>>> tuple12
>>> tuple56 = 5, 6
>>> tuple56
>>> dict78 = dict(keya=7, keyb=8)
```

```
>>> dict78                                                                  886
>>> f4(*tuple12)                                                            887
>>> f4(**dict78)                                                            888
>>> f4(1, 2, **dict78)                                                      889
>>> f4(1, 2, *tuple56, **dict78)                                            890
>>> f4(1, 2, 3, 4, *tuple56, **dict78)                                      891
```

Exercises: Functions

Write a better `__init__` method for your `Employee` class which can handle these calls:

- `Employee()`
- `Employee('Jane', 'Doe', 48)`
- `Employee('Jane', 'Doe')`
- `Employee(last='Doe', first='Jane')`
- `Employee('Jane Doe')`
- `Employee('Doe, Jane')`
- `Employee('Jane Doe', assistant=Employee('John Doe'))`

Files

```
>>> f = open('eg.txt')                                                      892
>>> f                                                                       893
>>> f.read()                                                                894
>>> f                                                                       895
>>> len(f.read())                                                           896
>>> f                                                                       897
>>> f = open('eg.txt')                                                      898
>>> f                                                                       899
>>> len(f.read())                                                           900
>>> f = open('eg.txt')                                                      901
>>> dir(f)                                                                  902
>>> hasattr(f, '__iter__')                                                  903
>>> hasattr(f, '__next__')                                                  904

>>> f.__next__()                                                            905
>>> f.__next__()                                                            906
>>> f.__next__()                                                            907
>>> f.__next__()                                                            908
>>> f.__next__()                                                            909

>>> with open('eg.txt') as f:                                               910
...     for line in f:
...         print(line, end='')
```

Other file operations

- `open('output.txt', 'w')`
- `open('output.txt', 'wb')`
- `f.write()`
- `f.readline()`
- `f.readlines()`

Exercises: Files (and Dictionaries)

Write code which reads a file and uses `collections.defaultdict` to produce a histogram of the frequency of each unique line in the file.

Tutorial Survey

Tutorial Survey: `http://bit.ly/1HSYfci`