

Python 101

Overview

Course: Python 101 tutorial, PyCon 2010, Atlanta

Presenter: Stuart Williams (stuart@swilliams.ca)

Intended audience: Programmers who want a fast introduction to the basics of Python.

Tutorial format: Frequently alternating presentation of concepts and exercise sets. Each pair of concepts and exercises ranges in length from 5 minutes on simple topics, to 20 minutes on more involved topics.

Requirements: A laptop computer with Python 2.6 (or 3.1) installed.

License: This PyCon 2010 *Python 101* Tutorial by Stuart Williams is licensed under a Creative Commons Attribution-Share Alike 2.5 Canada License (<http://creativecommons.org/licenses/by-sa/2.5/ca/>).

Strategy

You'll learn by seeing and doing.

The interactive Python interpreter is used very heavily.

I'll demo using Python 2.6 but with two "import `__future__`" features from Python 3.1.

Early exercises use exploration via the interpreter. Later sections will have more traditional "do this" exercises.

In order to be engaging for a wide range of levels of experience and ability there are some examples and exercises that some or most of you won't figure out or complete in the time we have today. Don't be discouraged! If you follow 50% you're doing great and you can try the harder ones after the course. Very few will follow 100%.

For each topic I'll demonstrate with many examples.

Expect some examples to fail. This is intentional to help you learn how Python handles errors and to learn some of its boundaries.

I am not providing the text of these exercises online because by typing them yourselves you will learn more.

Feel free to interrupt with questions. Be patient if I delay my answer to a question based on my planned outline.

Numbers, etc.

```
>>> 1 0
>>> -1 1
>>> 1- 2
>>> 1 = 2 3
>>> 1 == 2 4
>>> 1 != 2 5
>>> 1 < 2 6
>>> 1 <= 1 7
>>> 1 > 2 8
>>> 1 < 3 < 5 9
>>> 1 < 3 and 3 < 5 10
>>> 1 * 2 11
>>> 1 + 2 12
>>> 1 / 2 13
>>> 1 / 2.0 14
>>> 1 // 2 15
>>> 1 // 2.0 16
>>> from __future__ import division 17
>>> 1 / 2 18
>>> 1 / 2.0 19
>>> 1 // 2 20
>>> 1 // 2.0 21
>>> 9 % 3 22
>>> 10 % 3 23

>>> int 24
>>> int(2) 25
>>> int(2.0) 26
>>> int(2.1) 27
>>> int(2.9) 28
>>> int('2') 29
>>> int('2.0') 30
>>> int('four') 31

>>> float(2) 32
>>> float('2') 33
>>> float('2.9') 34

>>> 1 / 0 35

>>> 1 + 1.0 36

>>> 2 & 4 37
>>> 2 | 4 38
>>> ~2 39
>>> 2 << 1 40
>>> 2 << 2 41
```

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.

>>> abs(4)	42
>>> abs(-4)	43
>>> pow(2, 8)	44
>>> 2**8	45
>>> 2 ** 16	46
>>> 2 ** 32	47
>>> 2 ** 31	48
>>> 2 ** 30	49
>>> int(2 ** 31 - 1)	50
>>> int(2 ** 31 - 1) + 1	51
>>> round(1.01)	52
>>> round(1.99)	53
>>> round(1.50)	54
>>> 1/3.0	55
>>> third = 1/3.0	56
>>> round(third)	57
>>> round(third, 1)	58
>>> round(third, 2)	59
>>> round(third, 3)	60
>>> round(1234.56, -1)	61
>>> round(1234.56, -2)	62
>>> round(1234.56, -3)	63
Advanced exercises	
>>> type(int)	64
>>> callable(int)	65
>>> int()	66
>>> 0 == int()	67
>>> 0 is int()	68
>>> type(int)	69
>>> type(int())	70
>>> int(4.3)	71
>>> int('4')	72
>>> int('four')	73
>>> int('z')	74
>>> int('c', 16)	75
>>> 1 < 2 and 2 < 3	76
>>> 1 < 2 and not (2 < 3)	77
>>> 1 < 2 and True	78
>>> 1 < 2 and False	79

Strings

```
>>> 'hello' 80
>>> "hello" 81
>>> 'today's the day' 82
>>> 'today\'s the day' 83
>>> "today's the day" 84
>>> 'A quote (") mark' 85
>>> 'hello' 86
>>> """hello
... there""" 87

>>> """today's the "day""" 88
>>> '''today's the "day''' 89
>>> 'hello\nthere\n' 90
>>> 'h' in 'hello' 91
>>> 'h' not in 'hello' 92
>>> 'hello'[0] 93
>>> s = 'hello' 94
>>> s 95
>>> s[0] = 'j' 96
>>> s 97
>>> s = 'j' + 'ello' 98
>>> s 99
```

Exercises: Strings

```
>>> r'hello' 100
>>> r'hello' is 'hello' 101

>>> r'hello\n' 102
>>> r'hello\n' == 'hello\n' 103
>>> len(r'hello\n') 104
>>> len('hello\n') 105

>>> 2 * 'hello' 106
>>> 2 + 'hello' 107
>>> '2' + 'hello' 108
>>> 'hello' 'there' 109

>>> type('hello') 110
>>> u'hello' 111
>>> type(u'hello') 112
```

String Methods

>>> len('hello')	113
>>> min('hello')	114
>>> max('hello')	115
>>> sorted('hello')	116
>>> 'hello'.startswith('h')	117
>>> 'hello'.startswith('he')	118
>>> 'hello'.endswith('lo')	119
>>> 'hello'.upper()	120
>>> 'HELLO'.lower()	121
>>> ' hello '.strip()	122
>>> ' hello '.rstrip()	123
>>> ' hello '.lstrip()	124
>>> 'Jan Feb Mar'.split()	125
>>> 'one, two, three'.split(',')	126

Exercises: String Methods

>>> sorted('hello')	127
>>> sorted('hello', reverse=True)	128
>>> reversed('hello')	129
>>> list(reversed('hello'))	130
>>> 'hello'.upper()	131
>>> 'HELLO'.isupper()	132
>>> 'hello'.title()	133
>>> 'Hello'.istitle()	134
>>> 'hello world'.title()	135
>>> 'hello world'.title().swapcase()	136
>>> '!' in '.?!'	137

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

Print and String Formatting

>>> 3	138
>>> print 3	139
>>> print(3)	140
>>> print(3, 2)	141
>>> from __future__ import print_function	142
>>> print 3	143
>>> print(3)	144
>>> print(3, 2)	145
>>> print('three', 4)	146
>>> 'hello\n'	147
>>> print('hello\n')	148
>>> print('hello\nthere\n')	149
>>> print('%d good reasons' % 3)	150
>>> print('{0} good reasons'.format(3))	151
>>> '{0} good reasons'.format(3)	152
>>> 'Hello'.format()	153
>>> 'Hello {0}'.format()	154
>>> 'Hello {0}'.format('Stu')	155
>>> '{0} {1}'.format('Hi', 'Stu')	156
>>> '{1} {0}'.format('Hi', 'Stu')	157
>>> '{0} {1}, {0}!'.format('Hi', 'Stu')	158
>>> '{0:d}'.format(99)	159
>>> '{0:10d}'.format(99)	160
>>> '{0:>10d}'.format(99)	161
>>> '{0:<10d}'.format(99)	162
>>> '{0:^10d}'.format(99)	163
>>> '{greet} {who}'.format(... greet='Hi', ... who='Stu')	164

Exercises: Print and String Formatting

>>> print 3	165
>>> from __future__ import print_function	166
>>> print 3	167
>>> print(3)	168
>>> print(3, 4, 5, sep=':', end='\$\n')	169
>>> 'Take {0} or {1}'.format(3, 4)	170
>>> 'Take {1} or {0}'.format(3, 4)	171

```

>>> v = 1/3.0 172
>>> '{0:f}'.format(v) 173
>>> '{0:4.2f}'.format(v) 174
>>> '{0:7.2f}'.format(v) 175
>>> '{0:7.4f}'.format(v) 176

>>> '{0:b}'.format(2) 177
>>> '{0:b}'.format(15) 178
>>> '{0:b}'.format(16) 179
>>> '{0:x}'.format(65535) 180
>>> '{0:o}'.format(65535) 181

>>> '{0:%}'.format(0.35) 182
>>> '{0:5.2%}'.format(0.35) 183

>>> '{0:10.{1}f}'.format(v, 3) 184
>>> '{0:10.{1}f}'.format(v, 5) 185

```

Introspection, str, repr

Note that `str` is a type, a class, a builtin, not the same as the deprecated standard library module `string`, and historically a builtin function, so don't be surprised by some obsolete references out there.

```

>>> 'hello' 186
>>> 'hello'.__class__ 187
>>> type('hello') 188
>>> 'hello'.__doc__ 189
>>> print('hello'.__doc__) 190

>>> str 191
>>> help(str) 192
>>> print(str.__doc__) 193
>>> str.strip 194
>>> help(str.strip) 195
>>> print(str.strip.__doc__) 196
>>> type(str) 197
>>> type(str.strip) 198

>>> help(dir) 199
>>> dir(str) 200
>>> dir() 201
>>> import __builtin__ 202
>>> dir() 203
>>> dir(__builtin__) 204

```

Exercises: Introspection, str, repr

```
>>> dir(str.strip) 205
>>> dir('hello') 206
>>> dir(str) == dir('hello') 207
```

```
>>> help(str) 208
>>> str(3) 209
>>> type(str(3)) 210
>>> help(repr) 211
>>> repr(3) 212
>>> float(4.3) 213
>>> str(4.3) 214
>>> repr(4.3) 215
>>> str('hello') 216
>>> repr('hello') 217
```

repr adds quotes so it's a legal Python expression (which can be eval'ed)

```
>>> '{0!s}'.format('hello') 218
>>> '{0!r}'.format('hello') 219
```

```
>>> help(eval) 220
>>> str('hello') 221
>>> eval(str('hello')) 222
>>> hello 223
>>> str('hello') == 'hello' 224
>>> repr('hello') 225
>>> eval(repr('hello')) 226
```

Tuples, Lists

```
>>> [1, 2, 3] 227
>>> type([1, 2, 3]) 228
>>> (1, 2, 3) 229
>>> type(1, 2, 3) 230
>>> n = (1, 2, 3) 231
>>> type(n) 232
>>> type(1, 2, 3) 233
>>> n 234
>>> type((1, 2, 3)) 235
>>> list((1, 2, 3)) 236
>>> tuple([1, 2, 3]) 237
>>> [1, 'b', 3] 238
>>> (1, 'b', 3) 239

>>> m = [1, 2, 3] 240
```

>>> n = (1, 2, 3)	241
>>> m[1] = 'b'	242
>>> m	243
>>> n[1] = 'b'	244
>>> n	245
>>> m + ['d']	246
>>> m	247
>>> n	248
>>> n + 'd'	249
>>> n + ('d')	250
>>> type('d')	251
>>> type(('d'))	252
>>> type(((('d'))))	253
>>> 'd'	254
>>> ('d')	255
>>> ('d',)	256
>>> tuple('d')	257
>>> type(('d',))	258
>>> n + ('d',)	259
>>> n	260
>>> n * 2	261
>>> m * 2	262
>>> tuple()	263
>>> type(tuple())	264
>>> n = ()	265
>>> n	266
>>> type(n)	267
>>> type(())	268
>>> m	269
>>> len(m)	270
>>> min(m)	271
>>> max(m)	272
>>> sorted(m)	273
>>> reversed(m)	274
>>> list(reversed(m))	275
>>> reversed('hello')	276
>>> list(reversed('hello'))	277
>>> (p, q) = (1, 2)	278
>>> p	279
>>> q	280
>>> p, q	281
>>> p, q = 3, 4	282
>>> p, q	283
>>> t1 = (1, 2, 3)	284
>>> t1	285
>>> t2 = 1, 2, 3	286
>>> t2	287
>>> t1 == t2	288

Exercises: Tuples, Lists

>>> m = [1, 2, 3]	289
>>> m	290
>>> m += 'd'	291
>>> m	292
>>> m.append('e')	293
>>> m	294
>>> m.append(5, 5, 6, 6, 7)	295
>>> m.append([5, 5, 6, 6, 7])	296
>>> m	297
>>> del m[-1]	298
>>> m	299
>>> m.extend([5, 5, 6, 6, 7])	300
>>> m	301
>>> 5 in m	302
>>> 10 not in m	303
>>> not 10 in m	304
>>> [5, 6] in m	305
>>> m	306
>>> m.append([5, 6])	307
>>> m	308
>>> [5, 6] in m	309
>>> n = [1, 2, 4]	310
>>> m < n	311
>>> p, q = 1, 2	312
>>> p, q	313
>>> p, q = q, p	314
>>> p, q	315
>>> x, y, z = (1, 2, 3)	316
>>> x, y, z	317
>>> x, y, z = 1, 2, 3	318
>>> x, y, z	319
>>> x, y, z = [1, 2, 3]	320
>>> x, y, z	321
>>> x, y, z = 'xyz'	322
>>> x, y, z	323
>>> r = 'one two three'.split()	324
>>> r	325
>>> ' '.join(r)	326
>>> ', '.join(r)	327
>>> m.reverse()	328

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 329
>>> m.sorted() 330
>>> m 331
>>> sorted(m) 332
>>> m 333
>>> m.sort() 334
>>> m 335
>>> m.sort(reverse=True) 336
>>> m 337

```

Sequence Indexing, Slicing

```

>>> m = ['jan', 'feb', 'mar', 338
...      'apr', 'may']

>>> m[0] 339
>>> m[3] 340
>>> m[-1] 341
>>> m[-2] 342
>>> m[0:1] 343
>>> m[0:2] 344
>>> m[0:-1] 345
>>> m[0:100] 346
>>> m2 = m[:] 347
>>> m2 348
>>> m2 == m 349
>>> m2 is m 350
>>> id(m2), id(m) 351
>>> help(id) 352
>>> del m2[0] 353
>>> m 354
>>> m2 355

>>> m[0] = 'January' 356
>>> m 357
>>> m[-1] = m[-1].capitalize() 358
>>> m 359
>>> del m[2] 360
>>> m 361
>>> m = range(10) 362
>>> m 363

```

Exercises: Sequence Indexing, Slicing

```

>>> m = [0, 1, 2] 364

```

```

>>> m[1] = [10, 20] 365
>>> m 366
>>> m = [0, 1, 2] 367
>>> m[1:2] = [10, 20] 368
>>> m 369

>>> range(10, 20) 370
>>> range(10, 20, 3) 371
>>> range(0, 100, 10) 372
>>> range(100, 0, -10) 373
>>> range(100)[100::-10] 374
>>> range(101)[-1:1:-10] 375

```

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

List Comprehensions

```

>>> range(8) 376
>>> [e for e in range(8)] 377
>>> [2 * e for e in range(8)] 378
>>> [2 + e for e in range(8)] 379
>>> [e for e in range(8) 380
...     if e % 2 == 0]

>>> ['{0} * 2 == {1}'.format(e, 2 * e) 381
...     for e in range(8)]

>>> ['{0} * 2 == {1}'.format(e, 2 * e) 382
...     for e in range(8)
...     if e % 2 == 0]

>>> [e for e in range(8) if e % 3 == 0] 383

```

Exercises: List Comprehensions

```

>>> [(a, b) for a in range(3) 384
...     for b in 'Jan Feb Mar'.split()]

>>> help(enumerate) 385
>>> [(10 * n, c) for (n, c) in 386
...     enumerate(['a', 'b', 'c'])]

>>> help(zip) 387
>>> zip(['Jan', 'Feb', 'Mar'], 388
...     (1, 2, 3))

```

```
>>> zip(['Jan', 'Feb', 'Mar'],
...      (1, 2, 3, 4))
```

389

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

390

Decorate, Sort, Undecorate (DSU) Idiom

```
>>> months = [
...     ('Jan', 1, 31), ('Feb', 2, 28),
...     ('Mar', 3, 31), ('Apr', 4, 30)]
```

391

```
>>> sorted(months)
>>> dsu = [(days, (name, order, days))
...        for (name, order, days) in months]
```

392

393

```
>>> dsu
>>> dsu.sort()
>>> dsu
>>> [ b for (a, b) in dsu ]
```

394

395

396

397

```
>>> from operator import itemgetter
>>> sorted(months, key=itemgetter(0))
```

398

399

Exercises: Decorate, Sort, Undecorate (DSU) Idiom

Use the DSU idiom to sort `months` alphabetically.

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

Objects and Variables

Restart python to empty the local namespace.

```
>>> id(1)
>>> help(id)
```

400

401

Everything in Python is an object and has:

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

```

>>> id([]) 402
>>> [] 403
>>> dir([]) 404
>>> type([]) 405
>>> i = [] 406
>>> j = i 407
>>> id(i), id(j) 408
>>> id(i) == id(j) 409
>>> i is j 410
>>> type([]).__bases__ 411

>>> id('xyz') 412
>>> id('xyz2') 413
>>> type('xyz') 414
>>> 'xyz' 415
>>> s = 'xyz' 416
>>> dir() 417
>>> s[1] = 'b' 418
>>> id('xyz') 419
>>> id(s) 420
>>> t = s 421
>>> dir() 422
>>> id(t) 423
>>> id(s) == id(t) 424
>>> t = 'xyz' 425
>>> id(s) == id(t) 426

>>> type([]) 427
>>> class SubList(list): 428
...     pass

>>> slist = SubList() 429
>>> type(slist) 430
>>> slist.__class__ 431
>>> slist.__class__.__bases__ 432
>>> slist.__class__.__bases__[0] 433
>>> slist.__class__.__bases__[0].__bases__ 434

```

Exercises: Objects and Variables

It is suggested you restart python to empty the local namespace.

```
>>> dir() 435
>>> i = 1 436
>>> i 437
>>> type(i) 438
>>> id(i) 439
>>> j = 1 440
>>> id(j) 441

>>> m = [1, 2, 3] 442
>>> m 443
>>> n = m 444
>>> n 445
>>> id(m) == id(n) 446
>>> m[1] = 'two' 447
>>> m 448
>>> n 449
```

Dictionaries

```
>>> int_to_month_list = [ 450
...     None, 'Jan', 'Feb', 'Mar']

>>> int_to_month_list[2] 451

>>> int_to_month = { 452
...     1: 'Jan', 2: 'Feb', 3: 'Mar'}

>>> int_to_month[2] 453

>>> month_to_int = { 454
...     'Jan': 1, 'Feb': 2, 'Mar': 3 }

>>> month_to_int 455

>>> month_to_int['Feb'] 456
>>> month_to_int['Apr'] 457
>>> month_to_int['Apr'] = 4 458
>>> month_to_int['Apr'] 459

>>> month_to_int.has_key('Feb') 460
>>> 'Feb' in month_to_int 461
>>> del month_to_int['Feb'] 462
>>> 'Feb' in month_to_int 463
```

```

>>> help(dict.fromkeys) 464
>>> list('mississippi') 465
>>> d = dict.fromkeys('mississippi', 1) 466
>>> d 467
>>> d.keys() 468

>>> import collections 469
>>> dd = collections.defaultdict(int) 470
>>> int() 471
>>> for c in 'mississippi': 472
...     dd[c] += 1

>>> dd.items() 473
>>> dd 474

```

Exercises: Dictionaries

```

>>> d = {'Jan': 1, 'Feb': 2, 'Mar': 3} 475
>>> d['Feb'] 476
>>> d['Apr'] = 4 477
>>> d.keys() 478
>>> d.values() 479
>>> d.items() 480
>>> help(d.items()) 481
>>> help(d.items) 482

>>> dict([(k, v + 1) for v, k in enumerate( 483
...     'Jan Feb Mar Apr'.split())])

>>> dict(Jan=1, Feb=2, Mar=3, Apr=4) 484

```

Blocks, for loops

```

>>> print('hello') 485
>>> print('there') 486

>>> i = 0 487
>>> while i < 5: 488
...     i += 1
...     print(i)

```

```

>>> temp = 15 489
>>> if temp <= 0: 490
...     print('Freezing')
...     elif temp < 10:
...         print('Cold')
...     elif temp < 20:
...         print('Temperate')
...     else:
...         print('Warm')

>>> for i in [1, 2, 3]: 491
...     print(i)
...     print(i * 2)

>>> for i in range(3): 492
...     for j in range(3):
...         print((i, j))

>>> from __future__ import print_function 493

>>> for i in range(3): 494
...     for j in range(3):
...         print((i, j))

>>> for i in range(3): 495
...     for j in range(3):
...         print(i, j, sep=', ', end='')
...     print()

>>> for i in (1, 2, 3): 496
...     print(i)

>>> d = {'zero': 0, 'one' : 1, 'two' : 2} 497
>>> for k, v in d.items(): 498
...     print('{0} -> {1}'.format(k, v))

>>> for k, v in d.iteritems(): 499
...     print('{0} -> {1}'.format(k, v))

>>> for t in d.iteritems(): 500
...     print('{0} -> {1}'.format(k, v))

>>> months = 'jan feb mar apr may'.split() 501
>>> for m in reversed(months): 502
...     print(m)

```

Exercises: Blocks, for loops

>>> if []:	503
... print('list non-empty')	
>>> if [None]:	504
... print('list non-empty')	
>>> if '':	505
... print('string non-empty')	
>>> if 'False':	506
... print('string non-empty')	
>>> d = dict(one=1, two=2, three=3)	507
>>> for k in d:	508
... print(k)	
>>> for k in sorted(d):	509
... print(k)	
Advanced exercises	
>>> for k in reversed(d):	510
... print(k)	
>>> for k in reversed(d.iteritems()):	511
... print(k)	
>>> for k in reversed(d.items()):	512
... print(k)	
>>> range(10)	513
>>> help(range)	514
>>> range(5, 15)	515
>>> range(5, 15, 3)	516
>>> range(15, 5, -3)	517

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`.

• *Note:* `iter(callable, sentinel)` behaves differently.

```

>>> m = [1, 2, 3] 518
>>> reversed(m) 519
>>> it = reversed(m) 520
>>> type(it) 521
>>> dir(it) 522
>>> it.next() 523
>>> it.next() 524
>>> it.next() 525
>>> it.next() 526
>>> it.next() 527
>>> it.next() 528

>>> for i in m: 529
...     print(i)

>>> m.next() 530
>>> it = iter(m) 531
>>> it.next() 532
>>> it.next() 533
>>> it.next() 534
>>> it.next() 535

>>> m.__getitem__(0) 536
>>> m.__getitem__(1) 537
>>> m.__getitem__(2) 538
>>> m.__getitem__(3) 539

>>> it = reversed(m) 540
>>> it2 = it.__iter__() 541
>>> hasattr(it2, 'next') 542

>>> m = [2 * i for i in range(3)] 543
>>> m 544
>>> type(m) 545

>>> mi = (2 * i for i in range(3)) 546
>>> mi 547
>>> type(mi) 548
>>> hasattr(mi, 'next') 549
>>> dir(mi) 550
>>> help(mi) 551
>>> mi.next() 552
>>> mi.next() 553
>>> mi.next() 554
>>> mi.next() 555

```

Exercises: Iterables, Generator Expressions

>>> m = [1, 2, 3]	556
>>> it = iter(m)	557
>>> it.next()	558
>>> it.next()	559
>>> it.next()	560
>>> it.next()	561
>>> for n in m:	562
... print(n)	
>>> it = iter(m)	563
>>> it2 = iter(it)	564
>>> list(it2)	565
>>> list(it)	566
>>> it1 = iter(m)	567
>>> it2 = iter(m)	568
>>> list(it1)	569
>>> list(it2)	570
>>> list(it1)	571
>>> list(it2)	572
>>> d = {'one': 1, 'two': 2, 'three':3}	573
>>> it = iter(d)	574
>>> list(it)	575
>>> mi = (2 * i for i in range(3))	576
>>> list(mi)	577
>>> list(mi)	578
>>> import itertools	579
>>> help(itertools)	580

Writing Scripts, Modules

- Start with `#!/usr/bin/env python`
- Suffix `.py` (also `.pyw` on Windows)
- Python creates `.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)

>>> help(reload)
>>> 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)
>>> 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):
...     return n % 2 == 0
599

>>> iseven(1)
600
>>> iseven(2)
601

>>> def add(x, y): return x + y
602

>>> add(1, 2)
603

>>> def plural(w):
...     if w.endswith('y'):
...         return w[:-1] + 'ies'
...     return w + 's'
604

>>> plural('word')
605
>>> plural('city')
606
>>> plural('fish')
607
>>> plural('day')
608

>>> def fact(n):
...     """factorial(n), -1 if n < 0"""
...     if n < 0:
...         return -1
...     if n == 0:
...         return 1
...     return n * fact(n - 1)
609

>>> fact.__doc__
610
>>> help(fact)
611
>>> fact(-1)
612
>>> fact(0)
613
>>> fact(1)
614
>>> fact(2)
615
>>> fact(3)
616
>>> fact(4)
617
>>> fact(10)
618
>>> fact(20)
619
>>> fact(30)
620
>>> fact(100)
621
>>> fact(500)
622
>>> fact(990)
623
>>> fact(1000)
624
```

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

625
626

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
```

627

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

628
629
630

```
>>> it = list123()
>>> it
>>> type(it)
>>> it.next()
>>> it.next()
>>> it.next()
>>> it.next()
```

631
632
633
634
635
636
637

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

638

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

639

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

640
641

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

642

```
>>> f = factorials() 643
>>> f.next() 644
>>> f.next() 645
>>> f.next() 646
>>> f.next() 647
>>> f.next() 648
>>> f.next() 649
>>> f.next() 650
>>> f.next() 651
```

Don't try this!

```
list(f)
```

```
>>> for fact in factorials(): 652
...     print fact
...     if len(str(fact)) > 6:
...         break # Quit the loop
```

Compare these two versions of evens:

```
>>> def evens1(): 653
...     n = 0
...     result = []
...     while n < 10:
...         result.append(n)
...         n += 2
...     return result
```

```
>>> def evens2(): 654
...     n = 0
...     while n < 10:
...         yield n
...         n += 2
```

Note their typical use is identical:

```
>>> for num in evens1(): 655
...     print num
```

```
>>> for num in evens2(): 656
...     print num
```

Exercises: Generators

Write a generator that generates an infinite stream of zeros.

Call by Object Reference

```
>>> def f1(i):
...     print('Old', i, end='')
...     i = i + 1
...     print('New:', i)
657

>>> j = 3
658
>>> j
659
>>> f1(j)
660
>>> j
661

>>> def f2(m):
...     print('Old:', m)
...     m.append(3)
...     print('New:', m)
662

>>> n = [0, 1, 2]
663
>>> n
664
>>> f2(n)
665
>>> n
666
```

Classes and Instances

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 (assignment, `import`, `def`, `del`) happen in the local scope.

See <http://docs.python.org/tutorial/classes.html#python-scopes-and-name-spaces>

- 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)`

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)

    __str__ = __repr__

    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__

>>> class Record(object):
...     pass

>>> r = Record()
>>> r.fname, r.lname = 'Jane', 'Doe'
>>> r.fname

```

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') 686
>>> try: 687
...     int('four')
... except:
...     print('caught it')
```

```
>>> try: 688
...     int('four')
... except Exception as e:
...     print('caught:\n -> {0}\n -> {1}'
...           .format(e, repr(e)))
...     save = e
```

```
>>> dir(save) 689
>>> save.args 690
```

Review

- Numbers, etc.
- Strings
- String Methods
- Print and String Formatting
- Introspection, `str`, `repr`
- Tuples, Lists
- Sequence Indexing, Slicing
- List Comprehensions

- Decorate, Sort, Undecorate (DSU) Idiom
- Objects and Variables
- Dictionaries
- Blocks, for loops
- Iterables, Generator Expressions
- Writing Scripts, Modules
- Defining and Calling Functions
- Generators
- Call by Object Reference
- Classes and Instances
- Exceptions