A Crash Course in Python

By Stephen Saville and Andrew Lusk

SIGUnix Meeting
Mon 28 Oct 2002
8:00 pm
1310 DCL

Based on the excellent tutorial by Guido Van Rossum:
http://www.python.org/doc/current/tut/tut.html

How to Start Python

Interactive:

```
bash# python
>>> print "Hello World"
Hello World
>>>
```

From File:

```
bash# cat << EOF > myfile.py
print "Hello World\n"
EOF
bash# python myfile.py
Hello World
bash#
```

How to Start Python

Executable File:

```
bash# cat << EOF > myfile.py
#!/usr/bin/python
print "Hello World\n"
EOF
bash# chmod a+x myfile.py
bash# ./myfile.py
Hello World
bash#
```

Python Data Types

Numbers:
```
flt_num = 10.0          int_num = 25
```

Strings:
```
my_str = "Dude, why are you using perl?"
```

Lists:
```
my_list = ("yo", 24, "blah", "go away")
```

Tuples:
```
my_tup = (1, 4, 32, "yoyo", ['fo', 'moog'])
```

Dictionaries:
```
my_dict = {'a': 24.5, 'mo': 'fo', 42: 'answer'}
```

Objects:
```
my_inst = MyClass('foo')
```

Modules:
```
import myfile
```

Numbers

Integers:

```
>>> my_int = 4
>>> my_int/3
1
```

Floating Point:

```
>>> my_float = 5.5
>>> 20/my_float
3.6363636363636362
>>> 0.5-0.1
0.40000000000000002
```

Complex Numbers:

```
>>> 4+3j
(4+3j)
>>> _ - 3j
(4+0j)
>>> my_complex = complex(10,3)
```

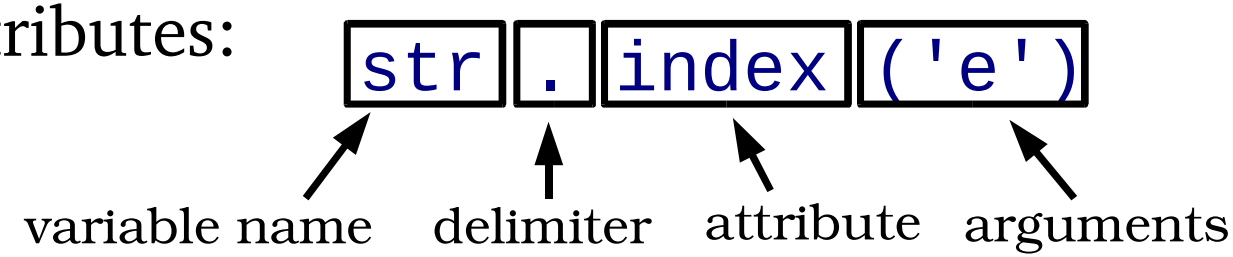
Strings

```
>>> str = "Hello, my friends, welcome to Python."
>>> str.upper()
'HELLO, MY FRIENDS, WELCOME TO PYTHON.'
>>> str.index('my')
7
>>> str[0:5]
'Hello'
>>> str + " I hope you enjoy your stay."
'Hello, my friends, welcome to Python. I hope you
enjoy your stay'
>>> print str(5) + " + " + str(3) + " = " + str(3+5)
5 + 3 = 8
>>> str.count('e')
4
>>> len(str)
37
```

one line

Everything's an Object

Object Attributes:



Attribute Peeking with dir():

```
>>> dir(str)
['capitalize', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'index', 'isalnum',
'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'replace', 'rfind', 'rindex', 'rjust',
'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper']
```

Lists

```
>>> lst = ['3', 45, 'frogger', 2]
>>> lst[2]
'frogger'
>>> del lst[2]
>>> lst
['3', 45, 2]
>>> lst.append('help')
>>> lst
['3', 45, 2, 'help']
```

List Methods:

```
  append(x) - add x to the end of the list
  extend(L) - add all items in sequence L to end of list
insert(i,x) - insert x at a position i
  remove(x) - remove first item equal to x
   pop([i]) - remove item at position i or end of list
   index(x) - return index of first item equal to x
   count(x) - count occurances of x
     sort() - sort the list
  reverse() - reverse the list
```

Tuples (sequences)

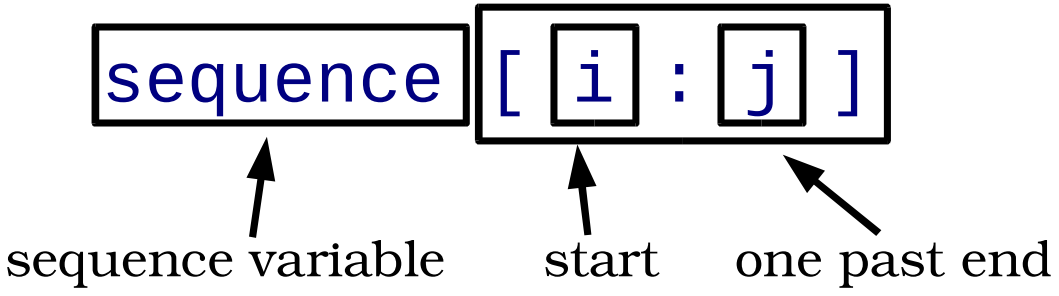
```
>>> tup = (6, 7, 'forty-two', 'question?')
>>> tup[0]
6
>>> del tup[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item deletion
>>> tup2 = ((1,2,3),[4,5,6]); tup2
((1, 2, 3), [4, 5, 6])
```

Sequence Operations (*s*, *t* sequences):

```
      x in s - test if s contains x
  x not in s - test if s does not contain x
       s + t - sequence concatenation
s * n, n * s - n shallow copies of s
        s[i] - ith element of s
      s[i:j] - slice of s
      len(s) - length of s (number of elements)
      min(s) - minimal element of s
      max(s) - maximal element of s
```

Slicing up Sequences

Slice Operator:




```
>>> seq = (0, 1, 2, 3, 4, 5, 6, 7, 8)
>>> seq[0]
0
>>> seq[-1]
8
>>> seq[1:4]
(1, 2, 3)
>>> seq[:3]
(0, 1, 2)
>>> seq[3:]
(3, 4, 5, 6, 7, 8)
>>> seq[-3:]
(6, 7, 8)
```

Dictionaries (mapping types)

```
>>> dict = {42: 'forty-two', 'naomi': 'person'}, 3: [1,2]}
>>> dict[42]
'forty-two'
>>> dict['namoi']
'person'
>>> del dict[42]; dict
{3: [1, 2], 'naomi': 'person'}
>>> dict.keys()
[3, 'naomi']
```

Mapping Operations Abridged (*d* mapping object):

len(*d***)** – number of items in *d*
*d***[***k***]** – item of *d* with key *k*
*d***[***k***] =** *x* – associate key *k* with value *x*
del *d***[***k***]** – delete item with key *k*
k **in** *d* – test if *d* has an item with key *k*
*d***.items()** – a copy of the (*key*, *value*) pairs in *d*
*d***.keys()** – a copy of the list of keys in *d*
*d***.values()** – a copy of the list of values in *d*

Control Flow Statements

If Conditionals:

```
>>> x = 2
>>> if x < 4:
...     print "x is so small\n"
... else:
...     print "x is big, yo!\n"
...
x is so small
```

For Loops:

```
>>> for x in [1,2]:
...     print x
...
1
2
```

While Loops:

```
>>> while x < 4: x++
...
>>> x
4
```

If Conditionals

```
if conditional1:
    statement1
    ...
    statementn
elif conditional2:
    statements
else:
    statements
```

For Statements

```
for name in list:
    statement1
    statement2
    ...
    statementn
```

The Range Function:

```
range([start,]stop[,step])
    make a list of integers in the range [start, stop),
    progressing by step each time.

>>> range(2,8)
[2, 3, 4, 5, 6, 7, 8
>>> range(10,2,-2)
[10, 8, 6, 4]
```

While Loops

```
while conditional:
    statement1
    statement2
    ...
    statementn
```

Breaking out of Loops

```
from random import randrange

for n in range(10):
    r = randrange(0,10) # get random int in [0,10)
    if n=r: continue    # skip iteration if n=r
    if n>r: break       # exit the loop if n>r
     print n
else:
    print "wow, you are lucky!\n"

if n<9:
    print "better luck next time\n"
```

Break, Continue and Else:

break – exit the loop immediately
continue – skip to next loop iteration
else – executed when a loop falls off the end

Pass into the Void

```
# a big, long, infinte noop

def void(): pass

if a=b: pass
for n in range(10): pass
while 1: pass

class Nada: pass
```

The Pass Statement:

pass – do nothing but fill a syntactic hole

Functions

```
>>> def bar():
...     print "The QuuxBox if foobarred!"
...
>>> def baz():
...     print "No it's not!"
...
>>> def foo(fun):
...     fun()
...
>>> foo(bar)
The QuuxBox is foobarred!
>>> foo(baz)
No it's not!
>>> def docfoo():
...      "This foo is documented!"
```

Basic Def

```
def name([arg1, arg2, ...]):
    statement1
    ...
    statementn
    [return [expression]]
```

Returning Values:

```
return [expression]
    exit the function, optionally returning the result
    of expression to the one who invoketh the function
```

Function Namespace

```
bullets = 10;
def fire():
    print "BANG!\n"
    bullets -= 1    # error – bullets not defined

def truefire():
    global bullets  # make bullets global
    print "BANG!\n"
    bullets -= 1    # good – bullets is global
```

Global Variable Access

global *name* [...]
    tell python to interpret *name* as a global variable.
Multiple names may be globalized by listing the all
separated by commas.

The Argument List

```
def name(arg[=defval], ..., [*arglist], [**kwdict]):
    function-body
```

Default Arguments

```
def charIndex(string, char, start=0, len=-1):
    if len<0: len=len(string)
    ...
```

Keyword Arguments

```
charAt("MakeMyDay", "M", len=4)
```

Extended Argument Lists

```
def arbitraryfun(foo, *arglist):
    ...
def keywordfun(foo, **kwdict):
    ...
```

Lamba Forms

Function Objects

```
>>> def subtractor(x, y): return x-y
...
>>> sub = subtractor
>>> add = lambda(x, y): return x+y
>>> sub(5, 7)
-2
>>> add(5, 7)
12
```

lambda *arglist*: *expression*

Modules

Importing Modules

```
>>> import sys
>>> print sys.version
2.2.1 (#1, Oct  4 2002, 15:26:55)
[GCC 3.2 (CRUX)]
>>> from math import *
>>> sin(pi/2)
1.0
```

The Import Statement

```
import module
from module import name[, ...]
from module import *
from package import module
```

Standard Modules

```
     sys – python system variables, including argv
      os – generic system interface (cross-platform)
 os.path – generic filesystem interface (cross-platform)
      re – regular expressions
    time – time query and conversion
    math – basic floating-point math functions (C libm)
```

Online Module Index

http://www.python.org/doc/current/lib/modindex.html

Functional Programming (Lists)

Filter

```
def positives(list):
    return filter(lambda x: return x>0, list)
```

filter(*function*, *sequence***)**
return all items in sequence for which function is true

Map

```
def lookup(dict, kwlist):
    return map(lambda k: return dict[k], kwlist)
```

map(*function*, *sequence***)**
return the result of function applied to all items in sequence

Function Programming Contd.

List Comprehensions

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

[expression for name in sequence [if conditional] ...]
```

Reduce

```
def dot(a, b):
    if len(a) != len(b):
        raise ValueError, "sequences have inequal lengths"
    prod = [a[i]*b[i] for i in range(len(a))]
    return reduce(lamba x, y: return x+y, prod)
```

reduce(*function, sequence***)**
apply the binary function *function* to the first two items in the sequence, then on the result and the next item, ...

Classes

```
class Namespace(UserDict):
    def __init__(self, basedicts):
        self.basedicts = basedicts
    def __getitem__(self, key):
        if key in self.data:
            return self.data[key]
        for dict in basedicts:
            if key in dict:
                return dict[key]
        raise NameError, key
    def __contains__(self, key):
        if key in self.data:
            return 1
        for dict in basedicts:
            if key in dict:
                return 1
        return 0
```

Basic Syntax

```
class name(bases):
    statements
```

Class Instances

```
class Area:
    def __init__(self, x=0.0, y=0.0, w=0.0, h=0.0):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
    def pointIn(self, x, y):
        return self.x <= x <= self.x + self.w && \
               self.y <= y <= self.y + self.h
    def move(self, dx, dy):
        self.x += dx
        self.y += dy

inst = Area(1.0, 1.0, 4.0, 4.0)
```

Initializer Method

__init__(*self*[, args]**)**
    method called to initialize a new instance of the class.

Class Inheritance

```
class Rect(Area): pass
class Circle(Area):
    def __init__(self, x=0.0, y=0.0, r=0.0):
        self.x = x
        self.y = y
        self.r = r
    def pointIn(self, x, y):
        return (x-self.x)**2 + (y-self.y)**2 < self.r**2
```