OSU Oregon State University

Intro to Python

Peter Krenesky
Lead Software Engineer

OSL OPEN SOURCE LAB

Slides: http://bit.ly/nXwnzQ

Why Python?

- Easy to learn.
- Easy to read.
- Large selection of stable libraries.

Much easier to build and maintain complex system infrastructure than bash scripts.

The Basics

- Python is a **dynamic** language

- **primitives**: int, str, float
- but even literals are **objects**

```
>>> x = 1
>>> print type(x)
<type 'int'>
```

Nulls in Python

- All variables must have a value
- **None** is python's equivalent of a **Null**

```
>>> x = None
>>> print x is None
True
```

Strings

```
x = “a string”
z = str(1)


# string concatenation
“, ”.join([x, y])


# string prefixes (types)
u”A Unicode String”
r”A raw string, ignores / escapes!”
```

Strings: **Formatting**

```
# Single variable
“formated: %s” % x

# tuple of multiple variables
“%s formated %s” % (x, y)

# dictionary of variables
“%(foo)s formated %(foo)s” % {'foo':1}

#format string
format = ”format: %s”
format % x
```

Basic Data Structures

- **`tuple:`** `(1, 2, 3)`
- **`list:`** `[1, 2, 3]`
- **`dict:`** `{“a”:1, “b”:2, “c”:3}`

Data Structures: **dict**

```
# create
d = {'a':1, 'b':2}
d = dict(a=1, b=2)

# add / update
d['a'] = 2
d['c'] = 3

# remove items
del d['a']

# iterators of keys, values, or both
d.keys()
d.values()
d.items()
```

Data Structures: **tuple**

```
# create a tuple (immutable list)
t = ()
t = (1, 2, 3)
t = tuple(iterable)

# indexable
x = t[1]

# implicit tuple creation
t = 1, 2, 3

# unpacking
a, b, c = t
```

Data Structures: **list**

```
# create
x = [1, 2, 3]
x = list(iterable)

# add items
x.append(item)
x.insert(position, item)

# remove items
x.remove(item)
x.pop()
```

Oregon State University

Data Structures: **slices**

```
>>> x = [0, 1, 2, 3, 4]
>>> print x[1:3]
[1, 2]

>>> print x[1:]
[1, 2, 3, 4]

>>> print x[:2]
[0, 1]

>>> print x[0:5:2]
[0, 2, 4]

>>> print x[-1]
4
```

Data Structures: **etc.**

- **Queue** – for making queues
- **Deque** – double ended queues
- **OrderedDict** – dictionary that maintains order
- **Named Tuples** – tuples with named fields

- **DefaultDict** – tools for building dictionaries
- **Itertools** – tools for quickly iterating lists

Classes

```
class Vehicle(object):
    “”” something that can move “””

    x = 0
    y = 0
    z = 0
```

Classes: **methods**

```
class Vehicle(object):
    “”” something that can move “””

    x = 0
    y = 0
    z = 0

    def move(self, x, y, z):
        “”” makes the vehicle move “””
        self.x = x
        self.y = y
        self.z = z

        return x, y, z
```

Classes: **Inheritance**

```
class Car(Vehicle):

    def move(self, x, y):
        super(Car, self).move(x, y, self.z)
```

Classes: **initializers**

```
class Car(Vehicle):

    def __init__(self, x, y):
        “”” init a car “””
        super(Car, self).__init__(x, y, 0)
```

Classes: “magic” methods

__getitem__ makes a class **indexable**
__setitem__

__iter__ makes a class **iterable**

__call__ makes a class **callable**

__add__ math functions
__sub__

Oregon State University

Doc Strings

```
class Vehicle(object):
    “”” something that can move “””

    X = 0
    Y = 0

    def move(self, x, y):
        “”” makes the vehicle move “””
        self.x = x
        self.y = y
```

Doc Strings: **help()**

```
>>> help(Vehicle)

Help on class Vehicle in module __main__:

class Vehicle(__builtin__.object)
 |  something that can move
 |
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if
defined)
 |
 |  __weakref__
 |      list of weak references to the object (if
defined)
```

Methods: **default args**

```
def move(self, x, y, z=0, rate=1):
    self.x = x
    self.y = y
    self.z = z
```

```
>>> # move just x and y
>>> move(1, 1)

>>> # move z too
>>> move(1, 1, 1)

>>> # custom rate, but no z movement
>>> move(1, 1, rate=42)
```

Methods: **args & kwargs**

```
def foo(*args, **kwargs):
    print args
    print kwargs
```

```
>>> foo(1, 2, 3, a=4, b=5)
(1, 2, 3)
{'a':4, 'b':5}
```

Methods: **kwargs common use**

unknown set of arguments

```
>>> print dict(a=1, b=2, c=3)
{“a”:1, “b”:2, “c”=3}
```

OSU Oregon State University

Methods: **arg & kwarg unpacking**

```
def foo(a, b, c, d, e):
    print a, b, c, d, e
```

```
>>> t = (1,2,3)
>>> d = {'d':4, 'e':5}
>>> foo(*t, **d)

(1, 2, 3, 4, 5)
```

Methods: **kwargs common use**

Method overloading

```
class Vehicle():
  def __init__(self, x, y, z=0):
        self.x, self.y, self.z = x, y, z


class TimeMachine(Vehicle):
    def __init__(self, ts, *args, **kw):
        super(Car, self).__init__(*args, **kw)
        self.ts = ts
```

```
>>> from datetime import datetime
>>> ts = datetime.now()
>>> delorean = TimeMachine(ts, 1, 2, 3)
>>> print delorean.x, delorean.y, delorean.z
1, 2, 3
```

Oregon State University

If statements

```
if 5 in list:
   print '5 was found'

elif 10 in list:
   print '10 was found'

else:
    print 'no 5 or 10'



# ternary (in-line)
five = True if 5 in list else False
```

Identity vs. Value

```
>>> foo = None
>>> print foo is None
True


>>> car1 = Car(id=123)
>>> car2 = Car(id=123)
>>> print car1 == car2
True

>>> print car1 is car2
False
```

Sequences as booleans

```
>>> empty = []
>>> full = [1, 2, 3]
>>> print bool(empty)
False

>>> print bool(full)
True


>>> print bool(“”) or {}
False
```

Oregon State University

__contains__

```
>>> foo = [1, 2, 3]
>>> print 2 in foo
True



>>> bar = dict(a=1, b=2, c=3)
>>> print “d” in bar
False
```

Oregon State University

Iteration

```
for i in iterable:
    print i



for i in range(10):
    if i == 10:
        break
    elif i == 5:
        continue
    print i
```

Iteration: **dicts**

```
# iterating a dict lists its keys
for key in dict:
    print key


# items returns a tuple
# which can be unpacked during iteration
for key, value in dict.items():
   print key, value
```

Exceptions

```
try:
    raise Exception('intentional!')

except Exception, e:
    print 'handle exception'

else:
    print 'no exception occurred'

finally:
    print 'this always runs'
```

Oregon State University

List Comprehensions

```
>>> [i**2 for i in range(3)]
[0, 1, 4]



>>> [i**2 for i in range(3) if i > 0]
[1, 4]


>>> (i for i in range(3))
<generator object <genexpr> at 0xf717d13>
```

Generators

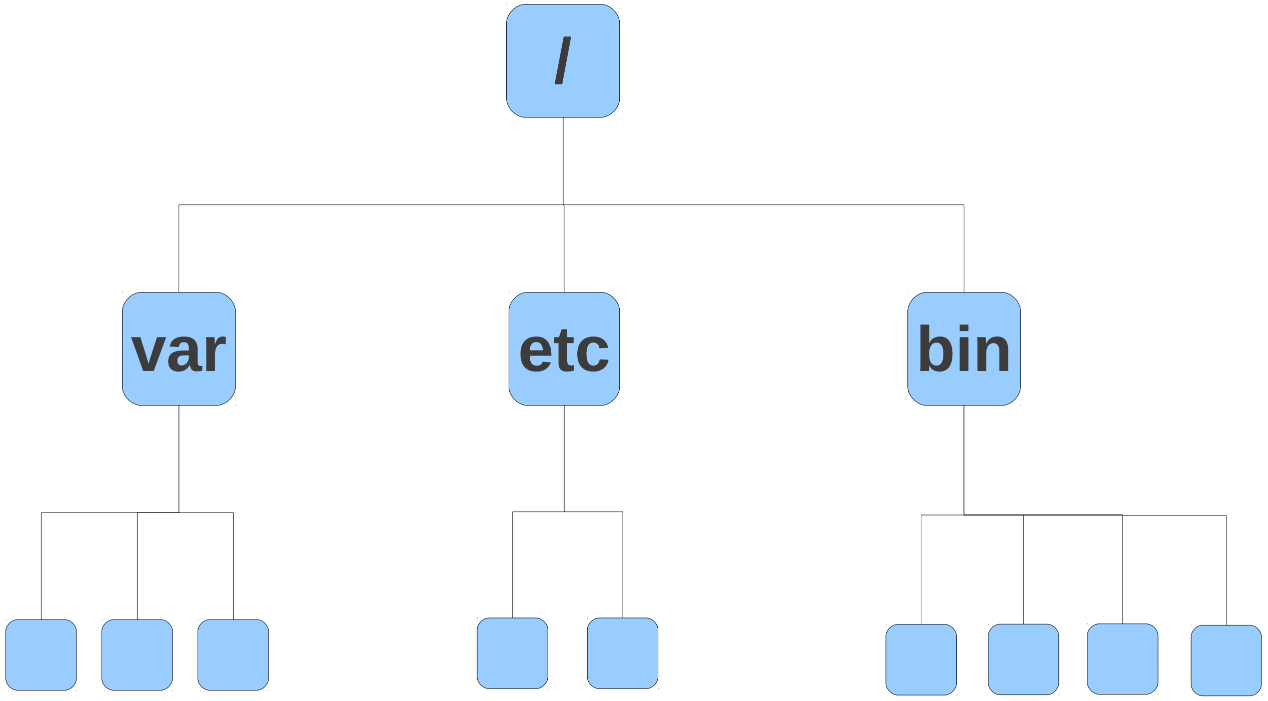
```
def foo():
    for i in range(2):
        yield i
```

```
>>> gen = foo()
>>> gen.next()
0

>>> gen.next()
1

>>> gen.next()
StopIteration
```

Generators: **more useful example**



Generators: **more useful example**

```
def dfs(tree):
    “”” depth first traversal “””
    yield tree

    if tree.children:
        for child in tree.children:
            for node in dfs(child):
                yield node

def search(tree, value):
    for node in dfs(tree)
        if value == node.value:
            return True

    return False
```

Oregon State University

Scopes

```
GLOBAL_VARS = [1, 2]
print GLOBAL_VARS


class Foo():
    class_scope = [3, 4]

    def bar(self, class_scope_too=[5, 6]):
        local_scope = [7, 8]

        class_scope_too += local_scope
        print class_scope_too
```

Scopes: **imports are global scope**

```
from foo import GLOBAL_ARGS
```

Questions?

Slides:
http://bit.ly/nXwnzQ

Peter Krenesky
Email: **peter@osuosl.org**
twitter: @kreneskyp