Intro to Python

by Daniel Greenfeld

Intro to Python

a.k.a. 20 cool things you can do with Python

Tons of content

- Please hold your questions until the end
- Latest slides will be made available
- Special thanks to:
  - Raymond Hettinger
  - Audrey Roy

Daniel Greenfeld

http://www.flickr.com/photos/pydanny/4442245488/

- pydanny
  - twitter.com/pydanny
  - github.com/pydanny
  - pydanny.blogspot.com
  - pydanny-event-notes.rtfd.org
- Python/Django developer at Cartwheel Web
- Capoeira
- Los Angeles
- Fiancee is Audrey Roy

Who uses Python?

NASA
Canonical
Ubuntu
Google
Dropbox
Facebook
Rackspace
Bitly
Disqus
Mozilla

CCP Games
Freshbooks
Quora
Github
Bitbucket
Cars.com
Evite
EventBrite

Blender and Maya
Dreamworks
Walt Disney Animation Studios
Industrial Light and Magic
WETA

Truecar.com
Grindr
grove.io
reddit
Lots of newspapers

What is Python

- Over 20 years old
- Dynamic, strongly typed scripted language
- A multi-paradigm programming language
- Named after Monty Python

Python is similar to...

- Perl
- Ruby
- Lisp
- Java

Python is different than...

- Perl
- Ruby
- Lisp
- Java

Python Core Concepts

Whitespace!

```
""" whitespace.py """
from random import randrange

def numberizer():
    # Generate a random number from 1 to 10.
    return randrange(1, 11)

number = numberizer()
if number > 5:
    print("This number is big!")

class RandomNumberHolder(object):
    # Create and hold 20 random numbers using numberizer

    def __init__(self):
        self.numbers = [numberizer(x) for x in range(20)]

random_numbers = RandomNumberHolder()
```

Whitespace!

```
""" whitespace.py """
from random import randrange

def numberizer():
    # Generate a random number from 1 to 10.
    return randrange(1, 11)

number = numberizer()
if number > 5:
    print("This number is big!")

class RandomNumberHolder(object):
    # Create and hold 20 random numbers using numberizer

    def __init__(self):
        self.numbers = [numberizer(x) for x in range(20)]

random_numbers = RandomNumberHolder()
```

Philosophy of Core Developers

- Conservative growth
- Aim for a simple implementation
- “We read Knuth so you don’t have to”

Zen of Python

Zen of Python

```
>>> import this
```

Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Which Python?

For learning and simple scripting...

Use what is on your system by default.

Don't have Python yet?

Download 3.2

Unless you are working with a tool
with a specific Python dependency
(e.g. Django requires Python 2.7)

20 cool things you can do with Python

Run it anywhere

Linux	Windows	
FreeBSD	Mac OS X	JVM
OpenBSD	Solaris	.NET
NetBSD	HP-UX	Android
BSD	OS/2	

http://en.wikipedia.org/wiki/CPython#Supported_platforms

Learn it fast

Learn it fast

Python is easy to learn but powerful.

Learn it fast

Python is easy to learn but powerful.

Experienced developers get up to speed in days.

Introspect

Introspect

a.k.a Introducing the String type

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
>>> spam = 'eggs'
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
>>> spam = 'eggs'
>>> fun = 'spam and EGGS   '
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
>>> spam = 'eggs'
>>> fun = 'spam and EGGS   '
>>> dir(fun)
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
>>> spam = 'eggs'
>>> fun = 'spam and EGGS   '
>>> dir(fun)
['__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',
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
>>> spam = 'eggs'
>>> fun = 'spam and EGGS   '
>>> dir(fun)
['__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', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper','zfill']
```

Introspect

a.k.a Introducing the String type

```
>>> foo = 'bar'
>>> spam = 'eggs'
>>> fun = 'spam and EGGS   '
>>> dir(fun)
['__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', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper','zfill']
```

dir() is a Python built-in function

Introspect

Introspect

```
>>> fun
```

Introspect

```
>>> fun
'spam and EGGS   '
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
>>> help(str)
```

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
>>> help(str)
```

Line comments start with '# '

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
>>> help(str)
```

Line comments start with '# '

help() is a Python built-in

Introspect

```
>>> fun
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
>>> help(str)
```

Line comments start with '# '

help() is a Python built-in

str is the Python string type object

Introspect

a.k.a Introducing the String type

```
>>> fun       
'spam and EGGS   '
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('a')
2
>>> len(fun) # built-in that gives length of object
16
>>> fun[0:5] # String slicing
'spam '
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
>>> help(str)
```

Line comments start with '# '

help() is a Python built-in

str is the Python string type object

Introspect

```
>>> help(str)
```

Introspect

```
>>> help(str)
Help on class str in module __builtin__:

class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |  If the argument is a string, the return value is the same object.
 |
 |  Method resolution order:
 |      str
 |      basestring
 |      object
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
```

Introspect

a.k.a Introducing the String type

```
>>> help(str)
Help on class str in module __builtin__:

class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |  If the argument is a string, the return value is the same object.
 |
 |  Method resolution order:
 |      str
 |      basestring
 |      object
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
```

Introspect

```
>>> help(str)
```

Introspect

```
>>> help(str)
|  capitalize(...)
|      S.capitalize() -> string
|
|      Return a copy of the string S with only its first character
|      capitalized.
|
|  center(...)
|      S.center(width[, fillchar]) -> string
|
|      Return S centered in a string of length width. Padding is
|      done using the specified fill character (default is a space)
|
|  count(...)
|      S.count(sub[, start[, end]]) -> int
|
|      Return the number of non-overlapping occurrences of substring sub
in
|      string S[start:end].  Optional arguments start and end are
interpreted
|      as in slice notation.
```

Introspect

a.k.a Introducing the String type

```
>>> help(str)
|  capitalize(...)
|      S.capitalize() -> string
|
|      Return a copy of the string S with only its first character
|      capitalized.
|
|  center(...)
|      S.center(width[, fillchar]) -> string
|
|      Return S centered in a string of length width. Padding is
|      done using the specified fill character (default is a space)
|
|  count(...)
|      S.count(sub[, start[, end]]) -> int
|
|      Return the number of non-overlapping occurrences of substring sub
in
|      string S[start:end].  Optional arguments start and end are
interpreted
|      as in slice notation.
```

##Things with Strings

```
>>> scale = 'Southern California Linux Expo'
>>> scale[0]
'S'
>>> scale[0:8]
'Southern'
>>> scale[:-5]
'Southern California Linux'
>>> scale[0:8] = 'Northern'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> scale.replace('Southern California','SoCal')
'SoCal Linux Expo'
>>> scale
'Southern California Linux Expo'
>>> scale = scale.replace('Southern California','SoCal')
>>> scale
'SoCal Linux Expo'
>>> scale.startswith('Windows')
False
>>> scale.endswith('Windows')
False
>>> scale.startswith('SoCal')
True
>>> 'Windows' in scale
False
>>> 'Linux' in scale
True
```

Strings are immutable

Basics

```
>>> x, y, z = 5, 10, 15
>>> 5 < 10
True
>>> 5 > 10
False
>>> True == False
False
>>> (5 == x) or (10 == x)
True
>>> (5 == x) and (10 == x)
False
>>> x + y - z
0
>>> 10 * 5
50
>>> 10 / 5
2
>>> 10 + 5
15
>>> 10 ** 2
100
```

Python has advanced math features that comes with the standard library.

For scientific needs, numpy is available.

String formatting

String formatting

```
>>> a = "Daniel"
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
>>> lst
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
>>> lst
['Daniel', 'Adam', 'Greenfeld']
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
>>> lst
['Daniel', 'Adam', 'Greenfeld']
>>> name =" ".join(lst)
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
>>> lst
['Daniel', 'Adam', 'Greenfeld']
>>> name =" ".join(lst)
>>> name
```

String formatting

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
>>> lst
['Daniel', 'Adam', 'Greenfeld']
>>> name =" ".join(lst)
>>> name
'Daniel Adam Greenfeld'
```

Lists

```
>>> my_list = [1, 2, 3]
>>> my_list.append(4)
>>> my_list
[1, 2, 3, 4]
>>> my_list.insert(2, 'dog')
>>> my_list
[1, 2, 'dog', 3, 4]
>>> my_list.extend([5, 6])
>>> my_list
[1, 2, 'dog', 3, 4, 5, 6]
>>> my_list.append([7, 8])
>>> my_list
[1, 2, 'dog', 3, 4, 5, 6, [7, 8]]
>>> my_list.pop(2)
'dog'
>>> my_list
[1, 2, 3, 4, 5, 6, [7, 8]]
>>> my_list.reverse()
>>> my_list
[[7, 8], 6, 5, 4, 3, 2, 1]
```

Lists are mutable

Lists + Functional Programming

```
>>> def divisible_by_2(x):
...     return x % 2 == 0
...
>>>
>>> def cube(x):
...     return x ** 3
...
>>>
>>> numbers = [1, 2, 3, 4, 6, 31]
>>>
>>> filter(divisible_by_2, numbers)
>>> [2, 4, 6]
>>>
>>> map(cube, numbers)
>>> [1, 8, 27, 64, 216, 29791]
```

Filter constructs a list from those elements of an iterable for which the specified function returns True.

Map applies the specified function to every item of the iterable and returns the results.

List Comprehensions

List Comprehensions

Remember this from the beginning?

```
""" whitespace.py """
from random import randrange

def numberizer():
    # Generate a random number from 1 to 10.
    return randrange(1, 11)

number = numberizer()
if number > 5:
    print("This number is big!")

class RandomNumberHolder(object):
    # Create and hold 20 random numbers using numberizer

    def __init__(self):
        self.numbers = [numberizer(x) for x in range(20)]

random_numbers = RandomNumberHolder()
```

List Comprehensions

Remember this from the beginning?

```
""" whitespace.py """
from random import randrange

def numberizer():
    # Generate a random number from 1 to 10.
    return randrange(1, 11)

number = numberizer()
if number > 5:
    print("This

class RandomNumberHolder(object):
    # Create and hold 20 random numbers using numberizer

    def __init__(self):
        self.numbers = [numberizer(x) for x in range(20)]

random_numbers = RandomNumberHolder()
```

List Comprension!

List Comprehensions

List Comprehensions

```
>>> items = [x for x in range(20)]
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> # Fizzbuzz solved using Python's List Comprehension
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> # Fizzbuzz solved using Python's List Comprehension
>>> lst = [(x, 'Fizz', 'Buzz', 'FizzBuzz') \
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> # Fizzbuzz solved using Python's List Comprehension
>>> lst = [(x, 'Fizz', 'Buzz', 'FizzBuzz') \
...     [(not x % 3) | (not x % 5) << 1] for x in range(20)]
```

List Comprehensions

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> # Fizzbuzz solved using Python's List Comprehension
>>> lst = [(x, 'Fizz', 'Buzz', 'FizzBuzz') \
...     [(not x % 3) | (not x % 5) << 1] for x in range(20)]
```

Backslash can be used to break up long statements. Please use sparingly!

Generators

```
>>> def countdown(n):
...     print("Counting down from {0}".format(n))
...     while n > 0:
...         yield n
...         n -= 1

>>> x = countdown(10)
>>> x
<generator object at 0x58490>
>>> x.next()
Counting down from 10
10
>>> x.next()
9
>>> x.next()
8
>>> x.next()
7
```

http://www.dabeaz.com/generators/Generators.pdf

Generators

```
>>> def countdown(n):
...     print("Counting down from {0}".format(n))
...     while n > 0:
...         yield n
...         n -= 1

>>> x = countdown(10)
>>> x
<generator object at 0x58490>
>>> x.next()
Counting down from 10
10
>>> x.next()
9
>>> x.next()
8
>>> x.next()
7
```

A generator evaluates only when the iterable is at that iteration. This is really powerful, especially when working with large iterables.

http://www.dabeaz.com/generators/Generators.pdf

Generators

```
>>> def countdown(n):
...     print("Counting down from {0}".format(n))
...     while n > 0:
...         yield n
...         n -= 1

>>> x = countdown(10)
>>> x
<generator object at 0x58490>
>>> x.next()
Counting down from 10
10
>>> x.next()
9
>>> x.next()
8
>>> x.next()
7
```

A generator evaluates only when the iterable is at that iteration. This is really powerful, especially when working with large iterables.

A billion iterations for a generator is NOTHING.

http://www.dabeaz.com/generators/Generators.pdf

Generator Expressions

```
def print_5_lines(filename):
    """ Prints the first 5 lines of the given file. """
    my_file = open(filename)    # my_file is a generator expression
    for i in range(5):
        line = my_file.next()
        print line
```

Note 1: range in Python 3 is a generator object.

Note 11: xrange in Python 2.7 is a generator object.

Generator Expressions

```
>>> items = (x for x in range(1000000000))
```

```
def print_5_lines(filename):
    """ Prints the first 5 lines of the given file. """
    my_file = open(filename)    # my_file is a generator expression
    for i in range(5):
        line = my_file.next()
        print line
```

Note 1: range in Python 3 is a generator object.

Note 11: xrange in Python 2.7 is a generator object.

Generator Expressions

```
>>> items = (x for x in range(1000000000))
>>> items
```

```
def print_5_lines(filename):
    """ Prints the first 5 lines of the given file. """
    my_file = open(filename)    # my_file is a generator expression
    for i in range(5):
        line = my_file.next()
        print line
```

Note 1: range in Python 3 is a generator object.

Note 11: xrange in Python 2.7 is a generator object.

Generator Expressions

```
>>> items = (x for x in range(1000000000))
>>> items
<generator object <genexpr> at 0x100721460>
```

```
def print_5_lines(filename):
    """ Prints the first 5 lines of the given file. """
    my_file = open(filename)    # my_file is a generator expression
    for i in range(5):
        line = my_file.next()
        print line
```

Note 1: range in Python 3 is a generator object.

Note 11: xrange in Python 2.7 is a generator object.

Generator Expressions

```
>>> items = (x for x in range(1000000000))
>>> items
<generator object <genexpr> at 0x100721460>
```

Generator expressions are shorthand for generators.
Just like list comprehensions, but with () instead of [].

```
def print_5_lines(filename):
    """ Prints the first 5 lines of the given file. """
    my_file = open(filename)    # my_file is a generator expression
    for i in range(5):
        line = my_file.next()
        print line
```

Note 1: range in Python 3 is a generator object.
Note 11: xrange in Python 2.7 is a generator object.

Sets

```
>>> lst = [1,1,1,1,1,2,2,2,3,3,3,3,3,3]
>>> s = set(lst)
>>> s
set([1,2,3])
```

Counting unique words in the Gettysburg Address

```
>>> address = """Four score and seven years ago our fathers brought..."""
>>> for r in [',','.','-']:
...     address = address.replace(r,'')
>>> words = address.split(' ')
>>> len(words)
278
>>> unique_words = set(words)
>>> len(unique_words)
143
```

Dictionaries

```
>>> data = {
        'name':'Daniel Greenfeld',
        'nickname':'pydanny',
        'states_lived':['CA','KS','MD','NJ','VA','AD'],
        'fiancee':'Audrey Roy'
        }
>>> data['name']
'Daniel Greenfeld'
>>> data['nickname'] = 'audreyr'
>>> data['nickname']
'audreyr'
>>> data['nickname'] = 'pydanny'
>>> data.keys()
['fiancee', 'nickname', 'name', 'states_lived']
>>> data.get('fiancee')
'Audrey Roy'
>>> data.get('fiance')
None
>>> data.pop('fiancee')
'Audreyr'
>>> data
{'nickname': 'pydanny', 'name': 'Daniel Greenfeld', 'states_lived': ['CA',
'KS', 'MD', 'NJ', 'VA']}
>>> data['fiancee'] = 'Audreyr Roy'
>>> data
{'fiancee': 'Audrey Roy', 'nickname': 'pydanny', 'name': 'Daniel
Greenfeld', 'states_lived': ['CA', 'KS', 'MD', 'NJ', 'VA', 'AD']}
```

Object-Oriented Programming

```
class Animal(object):
    def __init__(self, name):
        self.name = name
    def talk(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()
```

http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming#Examples

Object-Oriented Programming

```
class Animal(object):
    def __init__(self, name):
        self.name = name
    def talk(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()
```

```
Missy: Meow!
Mr. Mistoffelees: Meow!
Lassie: Woof! Woof!
```

http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming#Examples

Object-Oriented Programming

```
class Animal(object):
    def __init__(self, name):
        self.name = name
    def talk(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()
```

```
Missy: Meow!
Mr. Mistoffelees: Meow!
Lassie: Woof! Woof!
```

Barely scratching the surface!

http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming#Examples

Isolate Environments

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
requests==0.9.1
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
requests==0.9.1
(my_env) $ pip freeze > requirements.txt
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
requests==0.9.1
(my_env) $ pip freeze > requirements.txt
...
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
requests==0.9.1
(my_env) $ pip freeze > requirements.txt
...
(another_env) $ pip install -r requirements.txt
```

Isolate Environments

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

Pro Tip: easy_install is legacy. Use pip.

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
requests==0.9.1
(my_env) $ pip freeze > requirements.txt
...
(another_env) $ pip install -r requirements.txt
```

Warning!
Only installs
Python drivers!
Not MongoDB
or RabbitMQ

Colorize Code

How Github and Bitbucket do it

Colorize Code

How Github and Bitbucket do it

```
$ pip install pygments
```

Colorize Code

How Github and Bitbucket do it

```
$ pip install pygments
```

pygments_demo.py

```
from pygments import highlight
from pygments.lexers import get_lexer_by_name
from pygments.formatters import HtmlFormatter

if __name__ == '__main__':
    # get this file
    code = open("pygments_demo.py", "rw").read()

    # figure out the lexer
    lexer = get_lexer_by_name("python", stripall=True)

    # construct the formatter
    formatter = HtmlFormatter(linenos=False, cssclass="source")

    # style and formatting
    css = HtmlFormatter().get_style_defs('.source')
    highlighted_code = highlight(code, lexer, formatter)
    page = """
        <html>
            <head><style>{css}</style></head>
            <body>{highlighted_code}</body>
        </html>
        """.format(css=css, highlighted_code=highlighted_code)
    print(page)
```

Colorize Code

How Github and Bitbucket do it

```
$ pip install pygments
```

pygments_demo.py

```
from pygments import highlight
from pygments.lexers import get_lexer_by_name
from pygments.formatters import HtmlFormatter

if __name__ == '__main__':
    # get this file
    code = open("pygments_demo.py", "rw").read()

    # figure out the lexer
    lexer = get_lexer_by_name("python", stripall=True)

    # construct the formatter
    formatter = HtmlFormatter(linenos=False, cssclass="source")

    # style and formatting
    css = HtmlFormatter().get_style_defs('.source')
    highlighted_code = highlight(code, lexer, formatter)
    page = """
        <html>
            <head><style>{css}</style></head>
            <body>{highlighted_code}</body>
        </html>
        """.format(css=css, highlighted_code=highlighted_code)
    print(page)
```

```
$ python pygments_demo.py > text.html
```

Sunday, January 15, 2012

Output of the program

text.html

```
from pygments import highlight
from pygments.lexers import get_lexer_by_name
from pygments.formatters import HtmlFormatter

if __name__ == '__main__':
    # get this file
    code = open("pygments_demo.py", "rw").read()

    # figure out the lexer
    lexer = get_lexer_by_name("python", stripall=True)

    # construct the formatter
    formatter = HtmlFormatter(linenos=False, cssclass="source")

    # style and formatting
    css = HtmlFormatter().get_style_defs('.source')
    highlighted_code = highlight(code, lexer, formatter)
    page = """
        <html>
            <head><style>{css}</style></head>
            <body>{highlighted_code}</body>
        </html>
        """.format(css=css, highlighted_code=highlighted_code)
    print(page)
```

Persist SQL

(my_env)$ pip install django

```
from datetime import datetime

from django.contrib.auth.models import User
from django.db import models
from django.utils.translation import ugettext_lazy as _

class Post(models.Model):

    author = models.ForeignKey(User)
    title = models.CharField(_('Title'), max_length=100)
    content = models.TextField(_("Content"))
    pub_date = models.DateTimeField(_("Publication date"))

class Comment(models.Model):
    post = models.ForeignKey(Post)
    name = models.CharField(_('Title'), max_length=100)
    content = models.TextField(_("Content"))
```

Persist SQL

(my_env)$ pip install django

Internationalization!

```
from datetime import datetime

from django.contrib.auth.models import User
from django.db import models
from django.utils.translation import ugettext_lazy as _

class Post(models.Model):

    author = models.ForeignKey(User)
    title = models.CharField(_('Title'), max_length=100)
    content = models.TextField(_("Content"))
    pub_date = models.DateTimeField(_("Publication date"))

class Comment(models.Model):
    post = models.ForeignKey(Post)
    name = models.CharField(_('Title'), max_length=100)
    content = models.TextField(_("Content"))
```

Persist NoSQL

(my_env)$ pip install mongoengine

```
from django.utils.translation import ugettext_lazy as _

import mongoengine as me

class User(me.Document):
    email = me.StringField(_('email'), required=True)
    first_name = me.StringField(_('first name'), max_length=30)
    last_name = me.StringField(_('last name'), max_length=30)

class Post(me.Document):
    title = me.StringField(_('title'), max_length=100, required=True)
    author = me.ReferenceField(User)
    content = me.StringField(_('content'))
    pub_date = me.DateTimeField(_("Publication date"))

class Comment(me.EmbeddedDocument):
    name = me.StringField(_('name'), max_length=100)
    content = me.StringField(_('content'))
```

Persist NoSQL

(my_env)$ pip install mongoengine

```python
from django.utils.translation import ugettext_lazy as _

import mongoengine as me

class User(me.Document):
    email = me.StringField(_('email'), required=True)
    first_name = me.StringField(_('first name'), max_length=30)
    last_name = me.StringField(_('last name'), max_length=30)

class Post(me.Document):
    title = me.StringField(_('title'), max_length=100, required=True)
    author = me.ReferenceField(User)
    content = me.StringField(_('content'))
    pub_date = me.DateTimeField(_("Publication date"))

class Comment(me.EmbeddedDocument):
    name = me.StringField(_('name'), max_length=100)
    content = me.StringField(_('content'))
```

Internationalization!

Message Queues

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

```
>>> from products.tasks import confirm_all_images
```

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

```
>>> from products.tasks import confirm_all_images
>>> result = confirm_all_images.delay()
```

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

```
>>> from products.tasks import confirm_all_images
>>> result = confirm_all_images.delay()
>>> result.ready()
```

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

```
>>> from products.tasks import confirm_all_images
>>> result = confirm_all_images.delay()
>>> result.ready()
False
```

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

```
>>> from products.tasks import confirm_all_images
>>> result = confirm_all_images.delay()
>>> result.ready()
False
>>> result.ready()
```

Message Queues

(my_env)$ pip install celery
(my_env)$ pip install requests

products/tasks.py

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def check_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

```
>>> from products.tasks import confirm_all_images
>>> result = confirm_all_images.delay()
>>> result.ready()
False
>>> result.ready()
True
```

##Work with JSON

```
>>> import json

>>> data = {
    'name':'Daniel Greenfeld',
    'nickname':'pydanny',
    'states_lived':['CA','KS','MD','NJ','VA','AD'],
    'fiancee':'Audrey Roy'
    }
>>> type(data)
<type 'dict'>
>>> payload = json.dumps(data)
>>> payload
'{"fiancee": "Audrey Roy", "nickname": "pydanny", "name": "Daniel
Greenfeld", "states_lived": ["CA", "KS", "MD", "NJ", "VA", "AD"]}'
>>> type(payload)
<type 'str'>
>>> restored = json.loads(payload)
>>> restored
<type 'dict'>
>>> restored
{u'fiancee': u'Audrey Roy', u'nickname': u'pydanny', u'name': u'Daniel
Greenfeld', u'states_lived': [u'CA', u'KS', u'MD', u'NJ', u'VA', u'AD'
]}
```

Work with REST

- TODO

Exception Handling

Exception Handling

```
>>> import logging
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)

Traceback (most recent call last):
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
__main__.CustomTypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
__main__.CustomTypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Zen of Python: Errors should never pass silently

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
__main__.CustomTypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Zen of Python: Errors should never pass silently

Pro Tip: Generic exceptions are the devil