Hands-on Beginning Python & Drones

@__mharrison__

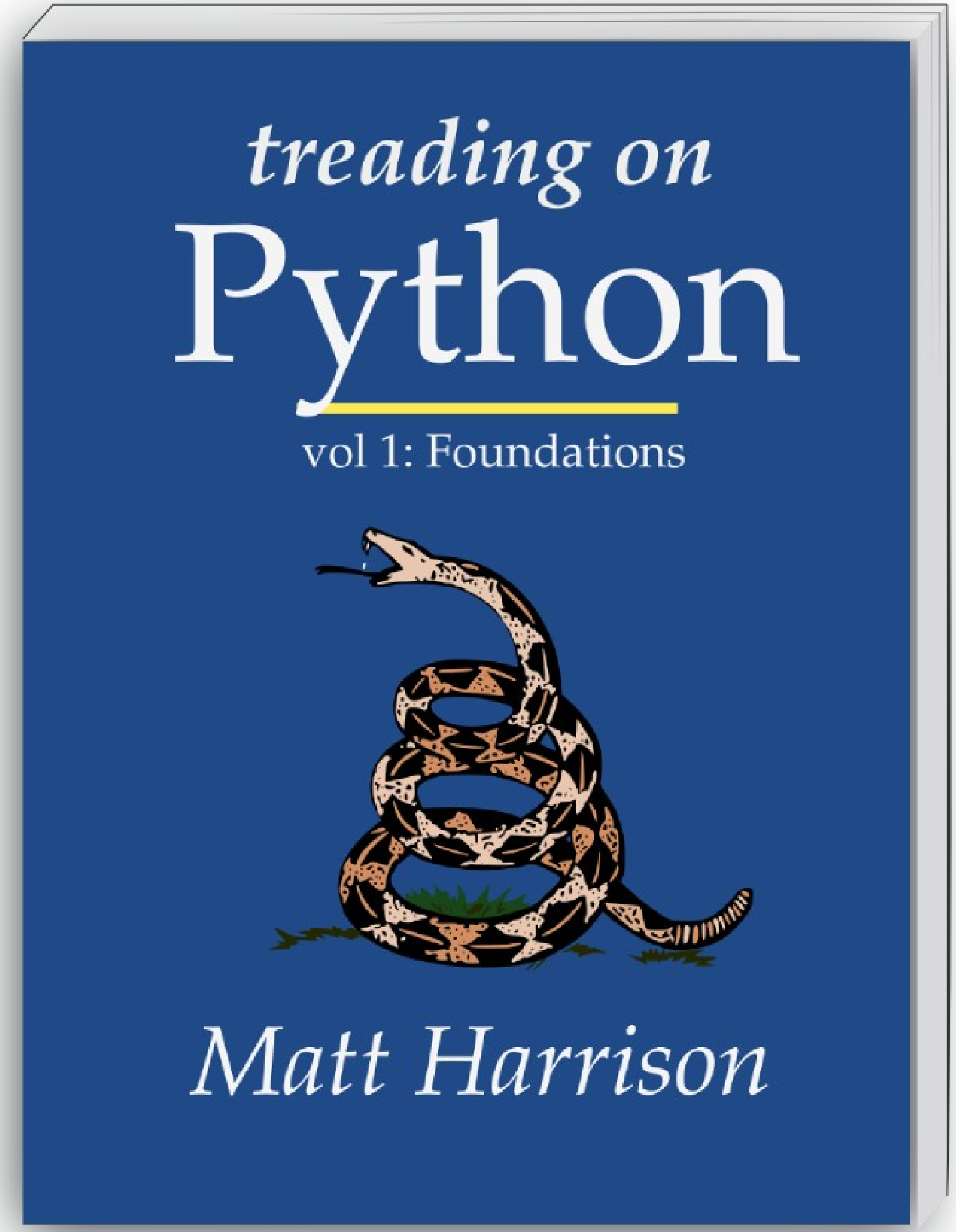
METASNAKE

About Me

Co-chair Utah Python. Consultant with 14 years Python experience across Data Science, BI, Web, Open Source Stack Management, and Search.

http://metasnake.com/

METASNAKE



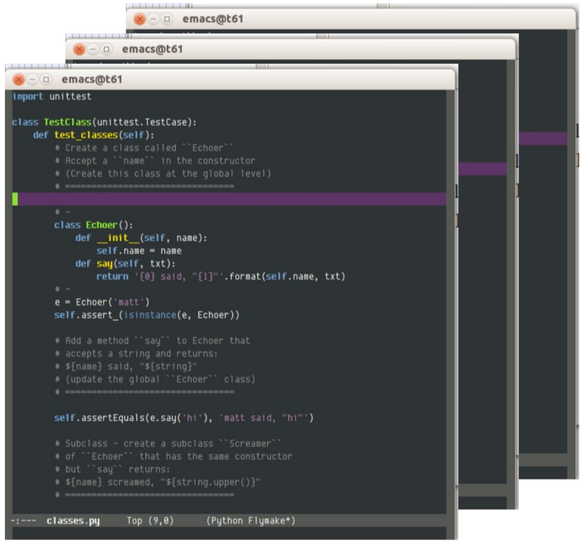
treading on

Python

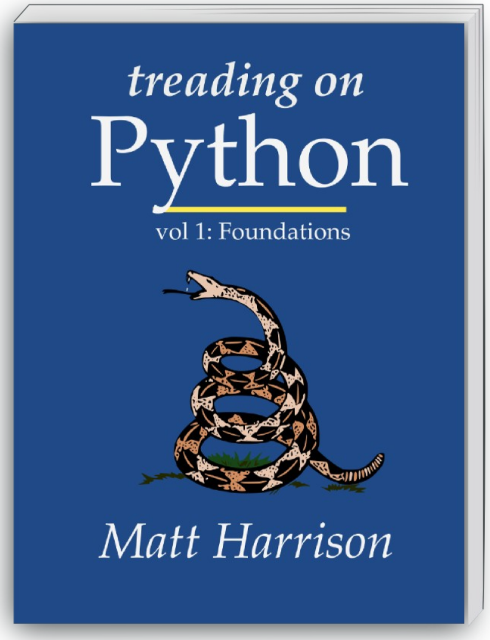
vol 1: Foundations

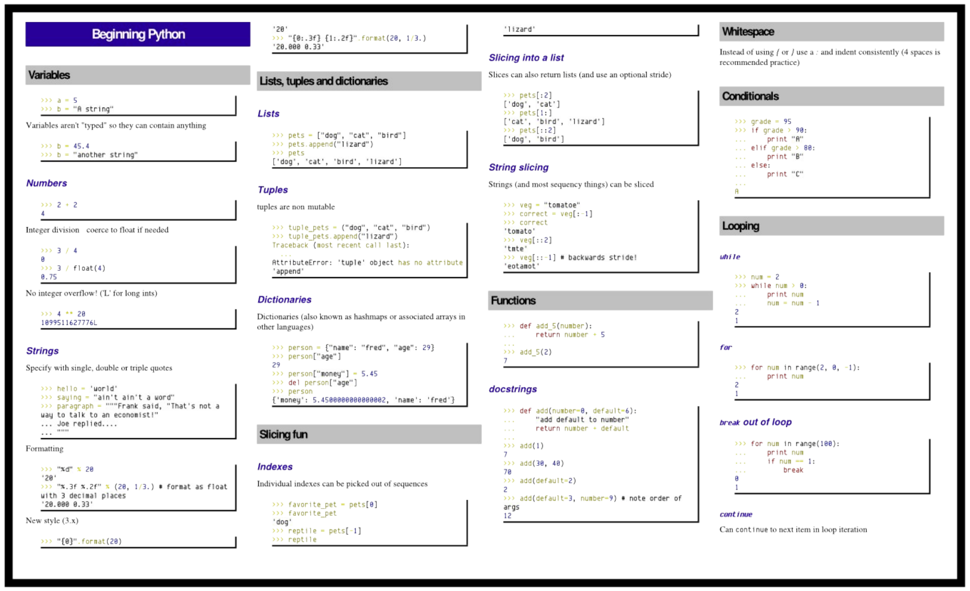
Matt Harrison

METASNAKE



```
emacs@t61
import unittest

class TestClass(unittest.TestCase):
    def test_classes(self):
        # Create a class called ``Echoer``
        # Accept a ``name`` in the constructor
        # (Create this class at the global level)
        # ================================

        # -
        class Echoer():
            def __init__(self, name):
                self.name = name
            def say(self, txt):
                return '{0} said, "{1}"'.format(self.name, txt)
        # -
        e = Echoer('matt')
        self.assert_(isinstance(e, Echoer))

        # Add a method ``say`` to Echoer that
        # accepts a string and returns:
        # ${name} said, "${string}"
        # (update the global ``Echoer`` class)
        # ================================

        self.assertEquals(e.say('hi'), 'matt said, "hi"')

        # Subclass - create a subclass ``Screamer``
        # of ``Echoer`` that has the same constructor
        # but ``say`` returns:
        # ${name} screamed, "${string.upper()}"
        # ================================

-:---  classes.py     Top (9,0)      (Python Flymake*)
```





Beginning Python

Variables

```
>>> a = 5
>>> b = "A string"
```

Variables aren't "typed" so they can contain anything

```
>>> b = 45.4
>>> b = "another string"
```

Numbers

```
>>> 2 + 2
4
```

Integer division   coerce to float if needed

```
>>> 3 / 4
0
>>> 3 / float(4)
0.75
```

No integer overflow! ('L' for long ints)

```
>>> 4 ** 20
1099511627776L
```

Strings

Specify with single, double or triple quotes

```
>>> hello = 'world'
>>> saying = "ain't ain't a word"
>>> paragraph = """Frank said, "That's not a
way to talk to an economist!"
... Joe replied....
... """
```

Formatting

```
>>> "%d" % 20
'20'
>>> "%.3f %.2f" % (20, 1/3.) # format as float
with 3 decimal places
'20.000 0.33'
```

New style (3.x)

```
>>> "{0}".format(20)
'20'
>>> "{0:.3f} {1:.2f}".format(20, 1/3.)
'20.000 0.33'
```

Lists, tuples and dictionaries

Lists

```
>>> pets = ["dog", "cat", "bird"]
>>> pets.append("lizard")
>>> pets
['dog', 'cat', 'bird', 'lizard']
```

Tuples

tuples are non mutable

```
>>> tuple_pets = ("dog", "cat", "bird")
>>> tuple_pets.append("lizard")
Traceback (most recent call last):
  ...
AttributeError: 'tuple' object has no attribute
'append'
```

Dictionaries

Dictionaries (also known as hashmaps or associated arrays in other languages)

```
>>> person = {"name": "fred", "age": 29}
>>> person["age"]
29
>>> person["money"] = 5.45
>>> del person["age"]
>>> person
{'money': 5.4500000000000002, 'name': 'fred'}
```

Slicing fun

Indexes

Individual indexes can be picked out of sequences

```
>>> favorite_pet = pets[0]
>>> favorite_pet
'dog'
>>> reptile = pets[-1]
>>> reptile
'lizard'
```

Slicing into a list

Slices can also return lists (and use an optional stride)

```
>>> pets[:2]
['dog', 'cat']
>>> pets[1:]
['cat', 'bird', 'lizard']
>>> pets[::2]
['dog', 'bird']
```

String slicing

Strings (and most sequency things) can be sliced

```
>>> veg = "tomatoe"
>>> correct = veg[:-1]
>>> correct
'tomato'
>>> veg[::2]
'tmte'
>>> veg[::-1] # backwards stride!
'eotamot'
```

Functions

```
>>> def add_5(number):
...     return number + 5
...
>>> add_5(2)
7
```

docstrings

```
>>> def add(number=0, default=6):
...     "add default to number"
...     return number + default
...
>>> add(1)
7
>>> add(30, 40)
70
>>> add(default=2)
2
>>> add(default=3, number=9) # note order of
args
12
```

Whitespace

Instead of using { or } use a : and indent consistently (4 spaces is recommended practice)

Conditionals

```
>>> grade = 95
>>> if grade > 90:
...     print "A"
... elif grade > 80:
...     print "B"
... else:
...     print "C"
...
A
```

Looping

while

```
>>> num = 2
>>> while num > 0:
...     print num
...     num = num - 1
2
1
```

for

```
>>> for num in range(2, 0, -1):
...     print num
2
1
```

break out of loop

```
>>> for num in range(100):
...     print num
...     if num == 1:
...         break
0
1
```

continue

Can `continue` to next item in loop iteration

METASNAKE

Beginning Python - Get code

`beg_python.zip`

- Thumbdrive has it

Unzip it somewhere (`unzip beg_python.zip`)

Begin

METASNAKE

Warning

- Starting from zero
- Hands on
  - (short) lecture
  - (short) code
  - repeat until time is gone

Why Python?

- Used (almost) everywhere
- Fun
- Concise

Introduction

Installation

Depends on Platform

Unix (Mac OSX, Linux)

Probably already installed

Windows

Download from http://www.python.org Installs to `C:\Python34\` or `C:\Python27\`

Windows

- Supports multiple version
- Need to update `PATH`

Windows

Add `C:\Python27\;C:\Python27\Scripts\` to `PATH` ie:

- run `[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27\;C:\Python27\Scripts\", "User")` in powershell
- Vista: My Computer > Properties > Advanced > Environment Variables
- Windows 7: Right click Computer > Properties > Advanced System Settings (on left-hand side) > Advanced Tab > Environment Variables

Windows

Type `echo %PATH%` in DOS prompt to verify. (Also `python` should work)

Python 2 or 3?

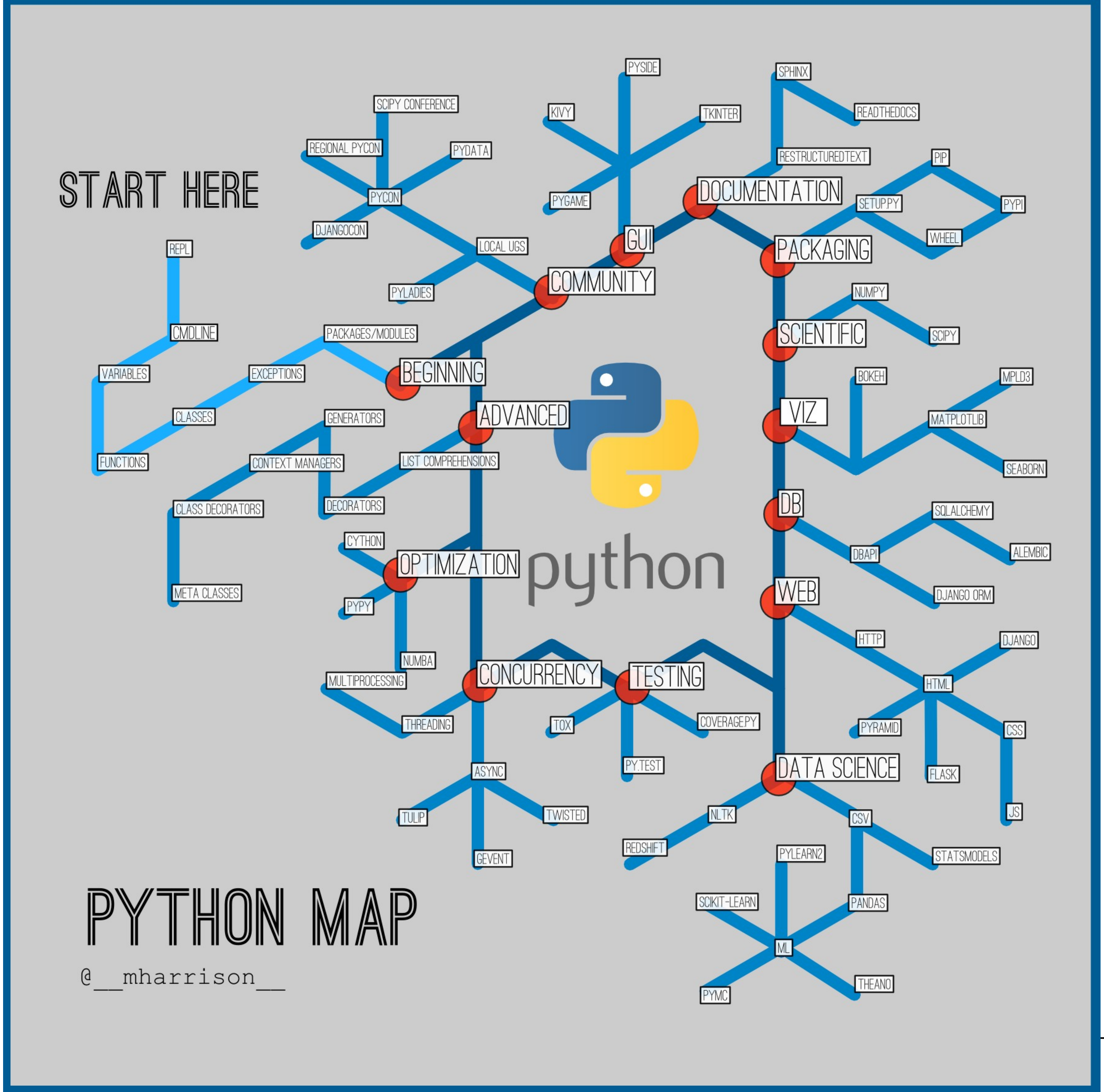
Most of this is agnostic. I'll note the differences, but use 2.x throughout

General Advice

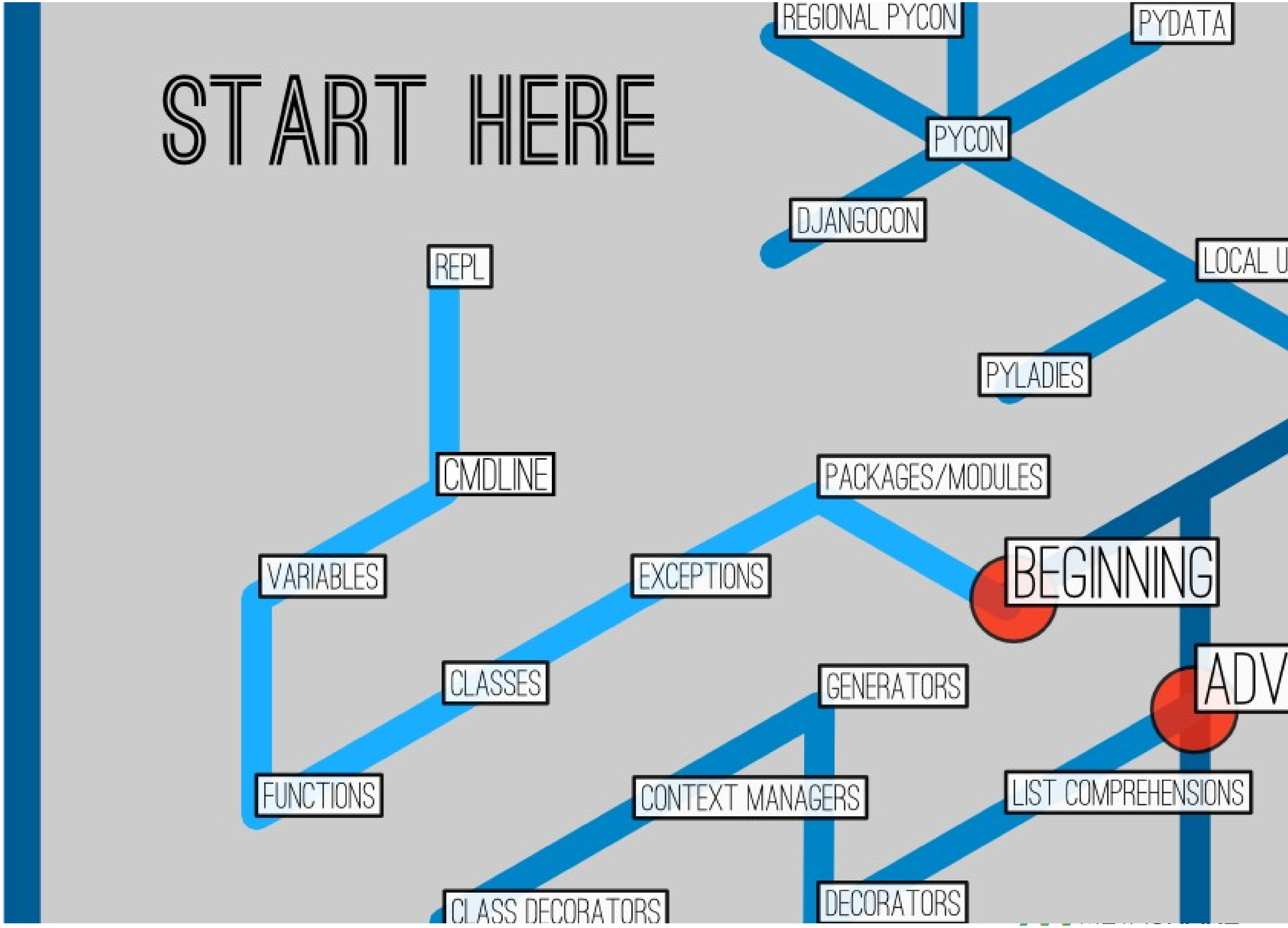
Most deployments are Python 2, but Python 3 is becoming more popular

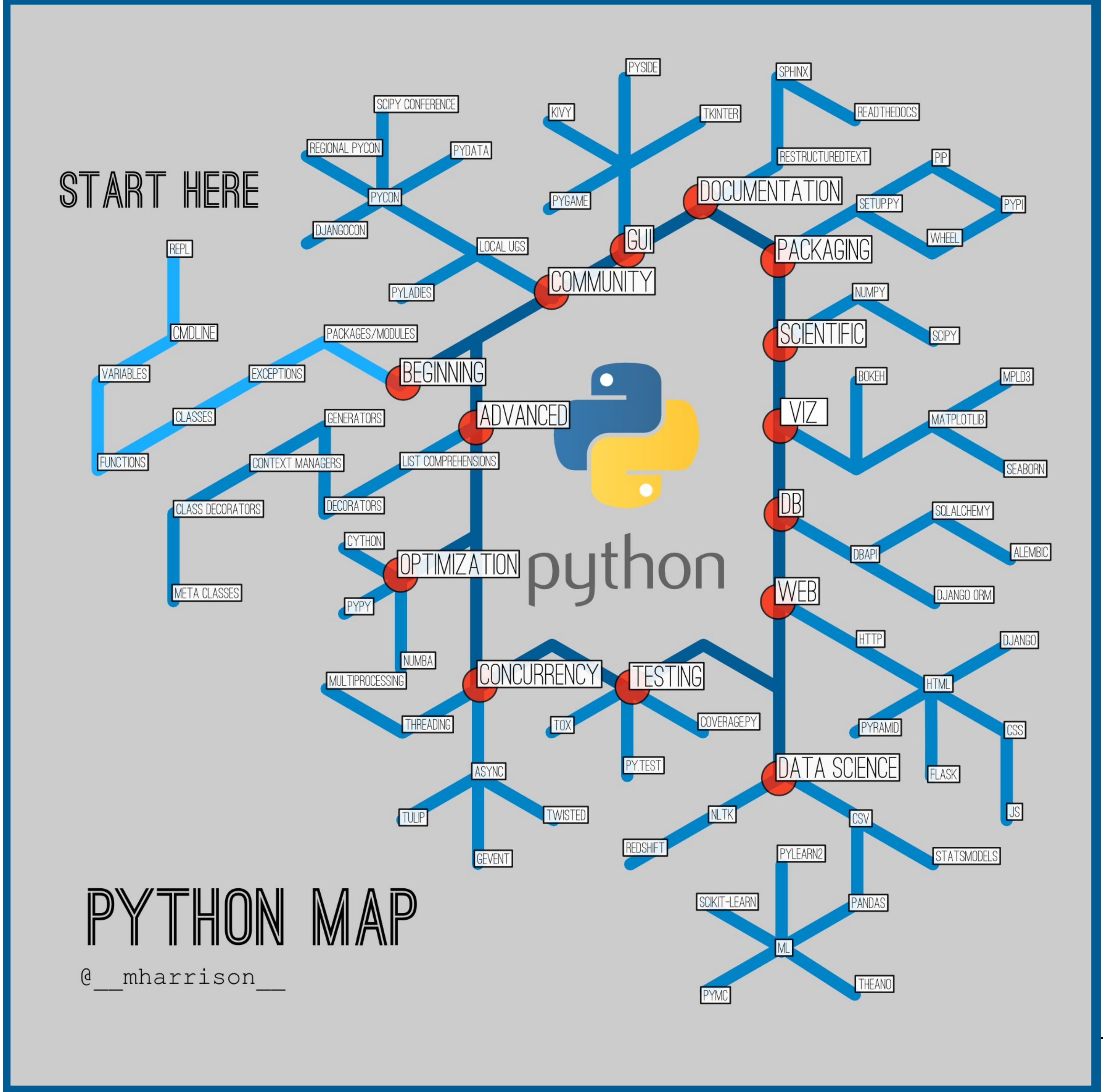
Content

METASNAKE


START HERE
PYTHON MAP
@__mharrison__
python
REPL
CMDLINE
VARIABLES
FUNCTIONS
CLASSES
EXCEPTIONS
PACKAGES/MODULES
BEGINNING
ADVANCED
GENERATORS
CONTEXT MANAGERS
LIST COMPREHENSIONS
DECORATORS
CLASS DECORATORS
META CLASSES
OPTIMIZATION
CYTHON
PYPY
NUMBA
CONCURRENCY
MULTIPROCESSING
THREADING
ASYNC
TULIP
TWISTED
GEVENT
COMMUNITY
SCIPY CONFERENCE
REGIONAL PYCON
PYDATA
PYCON
DJANGOCON
LOCAL UGS
PYLADIES
GUI
PYSIDE
KIVY
TKINTER
PYGAME
DOCUMENTATION
SPHINX
READTHEDOCS
RESTRUCTUREDTEXT
PACKAGING
PIP
SETUP.PY
PYPI
WHEEL
SCIENTIFIC
NUMPY
SCIPY
VIZ
BOKEH
MPLD3
MATPLOTLIB
SEABORN
DB
SQLALCHEMY
ALEMBIC
DBAPI
DJANGO ORM
WEB
HTTP
DJANGO
HTML
PYRAMID
CSS
FLASK
JS
TESTING
TOX
COVERAGE.PY
PY.TEST
DATA SCIENCE
NLTK
REDSHIFT
CSV
STATSMODELS
PANDAS
PYLEARN2
SCIKIT-LEARN
ML
PYMC
THEANO

ASNAKE


START HERE
REGIONAL PYCON
PYDATA
PYCON
DJANGOCON
LOCAL U
PYLADIES
REPL
CMDLINE
PACKAGES/MODULES
VARIABLES
EXCEPTIONS
BEGINNING
CLASSES
GENERATORS
ADV
FUNCTIONS
CONTEXT MANAGERS
LIST COMPREHENSIONS
CLASS DECORATORS
DECORATORS


START HERE
REPL
CMDLINE
VARIABLES
FUNCTIONS
CLASSES
EXCEPTIONS
PACKAGES/MODULES
BEGINNING
SCIPY CONFERENCE
REGIONAL PYCON
PYDATA
PYCON
DJANGOCON
LOCAL UGS
PYLADIES
COMMUNITY
KIVY
PYSIDE
TKINTER
PYGAME
GUI
SPHINX
READTHEDOCS
RESTRUCTUREDTEXT
DOCUMENTATION
PIP
SETUPPY
PYPI
WHEEL
PACKAGING
NUMPY
SCIENTIFIC
SCIPY
BOKEH
MPLD3
VIZ
MATPLOTLIB
SEABORN
SQLALCHEMY
DB
DBAPI
ALEMBIC
DJANGO ORM
WEB
HTTP
DJANGO
HTML
PYRAMID
CSS
FLASK
JS
DATA SCIENCE
NLTK
CSV
REDSHIFT
STATSMODELS
PYLEARN2
PANDAS
SCIKIT-LEARN
ML
PYMC
THEANO
GENERATORS
ADVANCED
CONTEXT MANAGERS
LIST COMPREHENSIONS
CLASS DECORATORS
DECORATORS
META CLASSES
CYTHON
OPTIMIZATION
PYPY
NUMBA
python
MULTIPROCESSING
CONCURRENCY
TESTING
THREADING
TOX
COVERAGEPY
PY.TEST
ASYNC
TULIP
TWISTED
GEVENT
PYTHON MAP
@__mharrison__

Hello World

hello world

```
print "hello world"
```

METASNAKE

from interpreter

```
$ python
>>> print "hello world"
hello world
```

REPL

```
$ python
>>> 2 + 2   # read, eval
4           # print
>>>         # repeat (loop)
```

REPL (2)

Many developers keep a REPL handy during programming

From script

Make file `hello.py` with :

```
print "hello world"
```

Run with:

```
$ python hello.py
```

IDEs

- **Editors:** Emacs, Vim, SublimeText, IDLE
- **IDEs:** PyDev (Eclipse), PyCharm (IntelliJ), Wing

(unix) script

Make file `hello` with :

```
#!/usr/bin/env python
print "hello world"
```

Run with:

```
$ chmod +x hello
$ ./hello
```

Python 3 hello world

`print` is no longer a statement, but a function :

```
print("hello world")
```

Drone Hello World

```
from turtledrone import TRDrone
drone = TRDrone()
drone.takeoff()
time.sleep(1)
drone.land()
drone.halt()
```

METASNAKE

Example Assignment

Run hello world

Assignment Notes

- Use spaces instead of tabs

Drone Commands

- takeoff
- land
- move_left, right, up, down, forward, backward
- turn_left, turn_right
- set_speed
- write (not in ardrone)

METASNAKE

Variables

```
a = 4         # Integer
b = 5.6       # Float
c = "hello"   # String
a = "4"       # rebound to String
```

Naming

- lowercase
- underscore_between_words
- don't start with numbers

See PEP 8 [1]

[1] http://legacy.python.org/dev/peps/pep-0008/

Basic Types

Math

`+`, `-`, `*`, `/`, `**` (power), `%` (modulo)

Modulo

Remainder:

```
>>> 4 % 2  # even number
0
>>> 5 % 2  # odd has remainder 1
1
```

Careful with integer division

```
>>> 3/4
0
>>> 3/4.
0.75
```

(In Python 2, in Python 3 `//` is integer division operator)

METASNAKE

Python 2/3 Division

```
>>> from __future__ import division
>>> 3/4
0.75
>>> 3//4
0
```

METASNAKE

What happens when you raise 10 to the 100th?

Long

```
>>> 10**100
10000000000000000000000000000000000000000
00000000000000000000000000000000000000000
0000000000000000000L
```

METASNAKE

Long

```
>>> import sys
>>> sys.maxint
9223372036854775807
```

Strings

```
name = 'matt'
with_quote = "I ain't gonna"
longer = """This string has
multiple lines
in it"""
```

METASNAKE

How do I print?

He said, “I’m sorry”

String escaping

Escape with \

```
>>> print 'He said, "I\'m sorry"'
He said, "I'm sorry"
>>> print '''He said, "I'm sorry"'''
He said, "I'm sorry"
>>> print """He said, "I'm sorry\""""
He said, "I'm sorry"
```

METASNAKE

Strings (2)

Escape Sequence	Output
`\\`	Backslash
`\'`	Single quote
`\"`	Double quote
`\b`	ASCII Backspace
`\n`	Newline
`\t`	Tab
`\u12af`	Unicode 16 bit
`\U12af89bc`	Unicode 32 bit
`\o84`	Octal character
`\xFF`	Hex character

METASNAKE

String formatting

c-like

```
>>> "%s %s" %('hello', 'world')
'hello world'
```

PEP 3101 style

```
>>> "{} {}".format('hello', 'world')
'hello world'
```

dir

```
>>> dir("a string")
['__add__', '__class__', ...
'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

Whats with the `'__blah__'`?

dunder methods

dunder (double under) or "special/magic" methods determine what will happen when + (`__add__`) or / (`__div__`) is called.

help

```
>>> help("a string".startswith)
```

```
Help on built-in function startswith:

startswith(...)
  S.startswith(prefix[, start[, end]]) -> bool

  Return True if S starts with the specified prefix, False
  otherwise.  With optional start, test S beginning at that
  position.  With optional end, stop comparing S at that
position.
  prefix can also be a tuple of strings to try.
```

METASNAKE

Some methods

String Method	Result
`capitalize`	Capitalize string
`endswith`	Determine if string ends with a substring
`find`	Find substring in a string (-1 not found)
`format`	Substitute objects into string
`index`	Find substring in a string (error if not found)
...	

METASNAKE

Some methods

```
>>> 'matt'.capitalize()
'Matt'

>>> 'file.xml'.endswith("xml")
True

>>> """supercalafrag""".find('frag')
9

>>> '{:.2f} {:d}'.format(1./3, 2)
'0.33 2'
```

METASNAKE

Drone Functionality

Turtle drone has a `write` method to print strings out

Assignment

Have the drone `write` `{} says hi to {}.` Fill in with your name and `drone.name`

METASNAKE

Comments

comments

Comments follow a `#`:

```
pi = 3.14  # approximation

#-----------------------
# Multi-line comment
#-----------------------

radius = 4
area = 2 * pi * radius
```

More Types

METASNAKE

None

Pythonic way of saying `NULL`. Evaluates to False:

```
>>> c = None
>>> bool(c)
False
```

None

Normally compared with `is` statement (checks identity not equality) :

```
>>> if c is None:
...     # do something
```

booleans

```
a = True
b = False
```

lists

```
>>> a = []
>>> a.append(4)
>>> a.append('hello')
>>> a.append(1)
>>> a.sort() # in place
>>> print a
[1, 4, 'hello']
```

METASNAKE

How would we find out the attributes & methods of a list?

lists

```
>>> dir([])  #doctest: +ELLIPSIS,
+NORMALIZE_WHITESPACE
['__add__', '__class__', '__contains__', ...
'__iter__', ... '__len__', ... , 'append',
'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

How would we find out documentation for a method?

METASNAKE

lists

```
>>> help([].append)
Help on built-in function append:

append(...)
    L.append(object) -- append object to end
```

METASNAKE

List methods

List Method	**Result**
`append`	Add item to end
`extend`	Add list items to end
`index`	Find item in list
`sort`	In place stable sort
...	

`in` statement

Uses `__contains__` dunder method to determine membership. (Or `__iter__` as fallback):

```
>>> 2 in [3, 4, 2]
True
```

List methods

```
>>> a = [3, 2]
>>> a.append(5)
>>> a.extend([9, 7])
>>> a.index(2)
1
>>> a.sort()
>>> a
[2, 3, 5, 7, 9]
```

Drone List

Drone stores commands in a list attribute
`_commands`

Dictionaries

dictionaries

Map *keys* to *values*. Called *hashmap* or *associative array* elsewhere:

```
>>> age = {}
>>> age['george'] = 10
>>> age['fred'] = 12
>>> age['henry'] = 10
>>> print age['george']
10
```

dictionaries (2)

Find out if `'matt'` (key) in `age`:

```
>>> 'matt' in age
False
```

.get

Get values for a key:

```
>>> print age['charles']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'charles'
>>> print age.get('charles', 'Not found')
Not found
```

deleting keys

Removing `'george'` (key) from `age`:

```
>>> del age['george']
```

(`del` is a statement, not a method. Not in `dir`. The `.pop` method is an alternative)

Some methods

Dict Method	Result
`get`	Get value for key or default
`items`	Get (key,value) pairs
`keys`	Get keys
`values`	Get values
`update`	Insert another dictionary into dict

Dict methods

```
>>> colors = {'pumpkin': 'orange', 'apple':'green'}

>>> colors.items()
[('apple', 'green'), ('pumpkin', 'orange')]

>>> colors.values()
['green', 'orange']

>>> colors.update(dict(rhubarb='red', pear='yellow'))
>>> colors
{'rhubarb': 'red', 'pear': 'yellow', 'apple': 'green',
'pumpkin': 'orange'}
```

METASNAKE

Dictionary Assignment

Pull out the `'battery'` key from the `navdata` attribute. Print it to the screen

METASNAKE

Functions

METASNAKE

functions

```
def add_2(num):
    """ return 2
    more than num
    """
    return num + 2

five = add_2(3)
```

Parts of functions

- `def`
- name
- parameters
- `:` + indent
- docstring
- body
- `return`

whitespace

Instead of { use a : and indent consistently (4 spaces)

whitespace (2)

```
/** Java **/
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}


# Python
class Hello(object):
    """Ugly translation warning!!!"""
    @staticmethod
    def main():
        print "Hello World!"

Hello.main()
```

whitespace (3)

invoke `python -tt` to error out during inconsistent tab/space usage in a file

default (named) parameters

```
def add_n(num, n=3):
    """default to
    adding 3"""
    return num + n

five = add_n(2)
ten = add_n(15, -5)
```

`__doc__`

Functions have *docstrings*. Accessible via `.__doc__` or `help`

__doc__

```
>>> def echo(txt):
...     "echo back txt"
...     return txt
>>> help(echo)
Help on function echo in module __main__:
<BLANKLINE>
echo(txt)
    echo back txt
<BLANKLINE>
```

METASNAKE

naming

- lowercase
- underscore_between_words
- don't start with numbers
- verb

See PEP 8

Assignment

write a function `draw_square` that accepts a drone as a parameter and draws a square with the drone.

Conditionals

conditionals

```
if grade > 90:
    print "A"
elif grade > 80:
    print "B"
elif grade > 70:
    print "C"
else:
    print "D"
```

METASNAKE

Remember the colon/whitespace!

Boolean tests

Supports (>, >=, <, <=, ==, !=)

```
>>> 5 > 9
False
>>> 'matt' != 'fred'
True
>>> isinstance('matt', basestring)
True
```

Boolean Operators

`and`, `or`, `not` (for logical), &, |, and ^ (for bitwise)

```
>>> x = 5
>>> x < -4 or x > 4
True
```

METASNAKE

Boolean note

Parens are only required for precedence

```
if (x > 10):
    print "Big"
```

Drone example

```python
def move_forward(self):
    """Make the drone move forward."""
    if self._state == GROUNDED:
        logging.info('Please takeoff')
    else:
        self.t.forward(100)
        self.draw_battery()
```

METASNAKE

Iteration

iteration

```
for number in [1,2,3,4,5,6]:
    print number

for number in range(1, 7):
    print number
```

range

Returns a list containing numbers from start up to but not including end:

```
>>> range(6)
[0, 1, 2, 3, 4, 5]

>>> range(2, 6)
[2, 3, 4, 5]
```

METASNAKE

`range` (2)

Python tends to follow *half-open interval* (`[start,end)`) with `range` and *slices*:

- end - start = length
- easy to concat ranges w/o overlap (ie `range(3) + range(3,9)`)

iteration (2)

Java/C-esque style of object in array access (BAD):

```
animals = ["cat", "dog", "bird"]
for index in range(len(animals)):
    print index, animals[index]
```

iteration (3)

If you need indices, use `enumerate` (to replace `range(len(a_list))`):

```
animals = ["cat", "dog", "bird"]
for index, value in enumerate(animals):
    print index, value
```

iteration (4)

Can `break` out of nearest loop:

```
for item in sequence:
    # process until first negative
    if item < 0:
        break
    # process item
```

iteration (5)

Can `continue` to skip over items:

```
for item in sequence:
    if item < 0:
        continue
    # process all positive items
```

iteration (6)

Can loop over lists, strings, iterators, dictionaries... sequence like things:

```
my_dict = { "name": "matt", "cash": 5.45}
for key in my_dict.keys():
  # process key

for value in my_dict.values():
  # process value

for key, value in my_dict.items():
  # process items
```

pass

pass is a null operation

```
for i in range(10):
    # do nothing 10 times
    pass
```

Assignment

Write a function circle that takes a drone. In the function use a loop to repeatedly call `turn_left` and `move_forward`

Slicing

Slicing

Sequences (lists, tuples, strings, etc) can be *sliced* to pull out a single item:

```
my_pets = ["dog", "cat", "bird"]
favorite = my_pets[0]
bird = my_pets[-1]
```

METASNAKE

Negative Indexing

Proper way to think of [negative indexing] is to reinterpret `a[-X]` as `a[len(a)-X]`

@gvanrossum

Slicing (2)

Slices can take an end index, to pull out a list of items:

```
>>> my_pets = ["dog", "cat", "bird"]
>>> my_pets[0:2]
['dog', 'cat']
>>> my_pets[:2]
['dog', 'cat']
>>> my_pets[1:3]
['cat', 'bird']
>>> my_pets[1:]
['cat', 'bird']
```

METASNAKE

Slicing (3)

Slices can take a *stride*:

```
>>> my_pets = ["dog", "cat", "bird"]
>>> my_pets[0:3:2]
['dog', 'bird']
>>> range(0,10)[::3]
[0, 3, 6, 9]
```

METASNAKE

Slicing (4)

Just to beat it in:

```
>>> veg = "tomatoe"
>>> correct = veg[:-1]
>>> correct
'tomato'
>>> veg[::2]
'tmte'
>>> veg[::-1]
'eotamot'
```

METASNAKE

File IO

METASNAKE

File Input

Open a file to read from it (old style):

```
fin = open("foo.txt")
for line in fin:
    # manipulate line

fin.close()
```

File Output

Open a file using `'w'` to `write` to a file:

```
fout = open("bar.txt", "w")
fout.write("hello world")
fout.close()
```

Always remember to close your files!

METASNAKE

closing with `with`

implicit `close` (new 2.5+ style):

```
with open('bar.txt') as fin:
    for line in fin:
        # process line
```

Example: Dumping JSON

```
>>> import json
>>> with open('/tmp/data.json', 'w') as
fout:
...     data = json.dumps([1, 2, 3])
...     fout.write(data)
```

METASNAKE

Example: Dumping JSON

```
$ cat /tmp/data.json
[0, 1, 2, 3]
```

IO Assignment

Write a function that accepts a filename and a turtle and writes the `._commands` attribute into a JSON file

Classes

Classes

```
>>> class Animal(object):
...     def __init__(self, name):
...         self.name = name
...
...     def talk(self):
...         print "Generic Animal Sound"

>>> animal = Animal("thing")
>>> animal.talk()
Generic Animal Sound
```

METASNAKE

Classes (2)

notes:

- `object` (base class) (fixed in 3.X)
- *dunder* init (constructor)
- all methods take `self` as first parameter

Classes(2)

Subclassing

```
>>> class Cat(Animal):
...     def talk(self):
...         print '%s says, "Meow!"' %
(self.name)

>>> cat = Cat("Groucho")
>>> cat.talk() # invoke method
Groucho says, "Meow!"
```

Classes(3)

```
>>> class Cheetah(Cat):
...     """classes can have
...     docstrings"""
...
...     def talk(self):
...         print "Growl"
```

Classes(4)

No private attributes/methods (precede with _ to hint “don’t muck with this”)

naming

- CamelCase
- don't start with numbers
- Nouns

Assignment

Create a subclass of TRDrone that has a `move_circle` method.

Exceptions

METASNAKE

Exceptions

Can catch exceptions:

```
try:
    f = open("file.txt")
except IOError, e:
    # handle e
```

Exceptions (2)

2.6+/3 version (`as`):

```
try:
    f = open("file.txt")
except IOError as e:
    # handle e
```

METASNAKE

Exceptions (3)

Can raise exceptions:

```
raise RuntimeError("Program failed")
```

METASNAKE

Chaining Exceptions (3)

```
try:
    some_function()
except ZeroDivisionError as e:
    # handle specific
except Exception as e:
    # handle others
```

METASNAKE

re-raise

Usually a good idea to re-raise if you don't handle it. (just `raise`):

```
try:
    # errory code
except Exception as e:
    # handle higher up
    raise
```

some hints

- try to limit size of contents of `try` block.
- catch specific Exceptions rather than just `Exception`

That's all

Questions? Tweet or email me

matt@metasnake.com
@__mharrison__
http://hairysun.com

METASNAKE