GENERAL ASSEMBLY

introduction to

python™

Felix Matathias
Bloomberg LP

this presentation…

- assumes you know nothing about programming
- contains everything I consider essential to get you going as a beginner in `python`…
- lots of slides with

```
real python code
```

- **the goal**: to make you self sufficient and prepare you for the next level!
- teach you a little bit about the **culture** of python!

why python?

- python is fun!!!!
- general purpose, high Level
- rapid development
- weakly typed language
- interpreted vs. compiled:
  - work interactively with the python interpreter (let's start one!)
- tremendous wealth of reusable libraries
  - "python comes with batteries"
- clean syntax,
- lots and lots of books and documentation

what can you do with python?

- anything you can imagine…
- read and analyze tweets
- get data in and out of facebook
- interact with amazon web services
- cryptography, web security
- data analysis
- machine learning
- deep learning algorithms
- bioinformatics
- websites
- …your future killer app!

import this

the cultural aspects of python…

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!
```

the short history of python

"Python was conceived in the late 1980s[1] and its implementation was started in December 1989[2] by Guido van Rossum at CWI in the Netherlands as a successor to the ABC programming language capable of exception handling and interfacing with the Amoeba operating system.[3] Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, *Benevolent Dictator for Life* (BDFL)."

- https://en.wikipedia.org/wiki/History_of_Python

there are 2 pythons!

- **python 2**
  - current version 2.7.10
- **python 3** (2008)
  - current version 3.4.3

```
>python --version
```

- not important differences for beginners except:
  - print is a function in python 3, a statement in python 2
- not backwards compatible
- "python 2.x is legacy, python 3.x is the present and future of the language" https://wiki.python.org/moin/Python2orPython3

weakly typed languages

- require discipline in coding standards
  - or the code can get out of control and difficult to maintain
  - "spaghetti code"
  - "technological deficit"
- require good documentation
  - python provides excellent documentation tools
- exhaustive testing of all code paths
  - no surprises when the code runs in production
  - compiled languages catch most errors during compilation
- you pay a speed penalty because you do not declare types

start your engines! (…interpreters)

- the easiest way to learn Python is to write code directly in the python interpreter!

```
>which python
```

```
>python
```

```
print(“Hello World”)
```

Use exit() or Ctrl-D to exit

how to best follow the class

- editing and running
- one screen with the python interpreter
- one screen with your favorite editor, editing your script
- one screen with the command line to run your script

python scripts

- the interpreter command line is used for testing
- for real programs we create a separate file containing our code, e.g., `hello.py`
- first line of the file is called the 'shebang'
- lets the operating system know how to run it
- **`#!/usr/bin/env python`**
- to run it: `>python hello.py`
- to make the script an executable program:

```
>chmod +x hello.py:
>./hello.py
```

hello.py

```
#!/usr/bin/env python

import sys

# Define a main function
def main():

    print sys.argv

    name = sys.argv[1]
    print 'Hello ' + name
    if name.lower() == 'felix':
        print ’you must be the instructor!'
    else:
        print ‘enjoy the class!'


# This calls the main function
if __name__ == '__main__':
    main()
```

keywords,
built-ins,
libraries

every language has its keywords…

```
import keyword
```

```
keyword.kwlist
```

```
['and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else',
'except', 'exec', 'finally', 'for',
'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'not', 'or', 'pass',
'print', 'raise', 'return', 'try',
'while', 'with', 'yield']
```

```
['and', 'as', 'assert',
'break', 'class',
'continue', 'def', 'del',
'elif', 'else', 'except',
'exec', 'finally', 'for',
'from', 'global', 'if',
'import', 'in', 'is',
'lambda', 'not', 'or',
'pass', 'print', 'raise',
'return', 'try', 'while',
'with', 'yield']
```

```
['and', 'as', 'assert',
'break', 'class',
'continue', 'def', 'del',
'elif', 'else', 'except',
'exec', 'finally', 'for',
'from', 'global', 'if',
'import', 'in', 'is',
'lambda', 'not', 'or',
'pass', 'print', 'raise',
'return', 'try', 'while',
'with', 'yield']
```

and its built-in functions…

```
abs(), divmod(), input(), open(),
staticmethod(), all(), enumerate(), int(),
ord(), str(), any(), eval(), isinstance(),
pow(), sum(), basestring(), execfile(),
issubclass(), print(), super(), bin(), file(),
iter(), property(), tuple(), bool(), filter(),
len(), range(), type(), bytearray(), float(),
list(), raw_input(), unichr(), callable(),
format(), locals(), reduce(), unicode(), chr(),
frozenset(), long(), reload(), vars(),
classmethod(), getattr(), map(), repr(),
xrange(), cmp(), globals(), max(), reversed(),
zip(), compile(), hasattr(), memoryview(),
round(), __import__(), complex(), hash(), min(),
set() , delattr(), help(), next(), setattr(),
dict(), hex(), object(), slice() , dir(), id(),
oct(), sorted()
```

https://docs.python.org/2/library/functions.html

and of course its libraries...

- “python comes with batteries”
- standard Library:
- https://docs.python.org/2/library/
- “keep this under your pillow”
- let’s have a tab with this link open at all times!
- open source licensed packages:
- https://pypi.python.org/pypi
- 63,000 packages!

variables

contain all the information of your program

- variable types
  - numerical: **int**, **float**, **long**, **complex**
  - boolean: **True**, **False**
  - **None**
  - **strings**
  - Compound types (we will discuss later)
- type conversions
- let's create our first variables!

fun with variables

```
#The variable type is inferred from the assignment

a = 14          (a is a reference to an integer)

a = “gluon”     (a becomes a reference to a str)

b = 13.14159

c = True

e = “1.345”

f = float(e)     #converting types

g = None         type(g)
```

deep thoughts on variables

- where are variables stored?
- concept of **mutable** and **immutable** **objects**

```
a = “quark”
id(a)
4299832424
b = “quark”
id(b)
4299832424
a is b      # True
a == b      # True
```

the simplest things can get tricky for beginners… we will address this later

```
a = [1,2,3]

b = a       # b is another reference
            # to the object [1,2,3]

a is b      # True
a == b      # True

b = a[:]    # b is a reference to a copy

a is b      # False
a == b      # True
```

help and documentation

- your best friend
- in the previous slide someone might ask:
- is there a difference between == and ‘is’ in python?
- http://stackoverflow.com/questions/132988/is-there-a-difference-between-and-is-in-python
- invaluable resource for developers

help()

Welcome to Python 2.7! This is the online help utility.

If this is your first time using Python, you should definitely check out the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules", "keywords", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose summaries contain a given word such as "spam", type "modules spam".

dir()

Without arguments, return the list of names in the current local scope. With an argument, attempt to return a list of valid attributes for that object.

```
el = “electron”
type(el)
dir(el)
```

type(el)

Returns information about the type of the object

strings and their functions

```
str.capitalize()
str.find()
str.format() (very important)
str.isalpha()
str.isalnum()
str.isdigit()
str.isspact()
str.isupper()
str.join()
str.lower()
str.lstrip()
str.replace()
str.split()
str.strip()
str.title()
str.upper()
str.isnumeric()
str.isdecimal()
```

1. documentation: https://docs.python.org/2/library/stdtypes.html
2. help('string')
3. Reuse existing code! Do not reinvent the wheel!

string functions

```
help(‘’.lower)  # why does this work?

a = "the weather is nice”

b = a.title()

print b

'The Weather Is Nice’

# don’t reinvent the wheel,
```

important string functions

```
str1 = "this is a string example....wow!!!";
str2 = "exam";

print str1.find(str2);     #17
print str1.find(str2, 10); #17
print str1.find(str2, 40); #-1

str = "0000000this is a string example....wow!!!00000";
print str.strip('0');

s = "topeka, kansas city, wichita, olathe”
cities = s.split(‘,’)
[topeka, kansas city, wichita, olathe]

str1 = "-";
seq = ("a", "b", "c"); # This is sequence of strings.
print str1.join( seq );
a-b-c

str1 = “my name is george”
str1.replace(“george”, “jim”)
```

string formatting

```
name = “Felix”
a = "My name is {0}".format(name)
print(a)
'My name is Felix’

"My name is {0} {1}".format("Felix", "Matathias")
'My name is Felix Matathias’
```

- we are invoking the attribute format() in a str object

https://docs.python.org/2/library/string.html#format-examples

lets get two important things out of the way…

a) indentation

- in python source code, white space is significant
- the indentation at the left of your statements
- religious debate

```
      for x in range(0,3):
→TAB      print("x has value: ", x)
      CTRL-D
```

b) everything in python is an object…

- repeat 20 times every day
- …and almost everything has attributes
- ""This is so important that I'm going to repeat it in case you missed it the first few times: *everything in Python is an object*. Strings are objects. Lists are objects. Functions are objects. Even modules are objects."
  –Dive into Python
- http://www.diveintopython.net/getting_to_know_python/everything_is_an_object.html

what is an object?

- a data abstraction, the axiom in our system, the entity
- **a grouping of attributes** that define an abstraction, like an Animal, a Vehicle, a Particle
- think of them as the "nouns" in your program
- object Vehicle
- Attributes:
  - length
  - weight
  - number_of_wheels
- Methods (also attributes), the "verbs":
  - drive()
  - stop()
  - turn_lights_on()

they define the interface of an object, what it can do

```
s = "Felix”   #s is a string. A string is an object

dir(s)

['__add__', '__class__', '__contains__',
'__delattr__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', …, '__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']
```

what was that all about?

- these are the **attributes** of the string object!
- data and function attributes
- everything that has a double underscore (__dunder__ , short for double underscore) is **not** intended for general use, it is an implementation detail that could change any time
- every object has its **"signature"**, the collection of all the things it can do
- these are the "attributes" or "methods" of the object

conditionals

conditional statements

- boolean operations
  - `x or y` — if x is false, then y, else x
  - `x and y` — if x is false, then x, else y
  - `not x` — if x is false, then True, else False

- comparisons:
  - `>, >=` (less, less or equal)
  - `<, <=` (greater, greater of equal)
  - `==` (equal, caution!, 2 equal signs)
  - `!=` (not equal)

- ```
  if a > b:
      print("a is greater than b")
  else:
      print("a not greater than b")
  ```
  ```

what is `False` in python

- **`None`**
- **`False`**
- Zero of any numeric type, for example, 0, 0L, 0.0, 0j
- Any empty sequence, for example, '', (), []
- Any empty mapping, for example, {}
- Instances of user-defined classes, if the class defines a __nonzero__() or __len__() method, when that method returns the integer zero or bool value False
- **Anything else evaluates to `True`**

fun with conditional statements

```
x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42

if x < 0:
     x = 0
     print 'Negative changed to zero'
elif x == 0:
     print 'Zero'
elif x == 1:
     print 'Single'
else:
     print 'More'
...
More
```

functions

functions

- the workhorse of the python language
- “A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing” –tutorialpoints.com

```
def function_name( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

let's code some functions

- important concept: variable scope

```
def sum( arg1, arg2 ):
   """Add both the parameters and return them."""
   total = arg1 + arg2
   print "Inside the function : ", total
   return total;

# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

a subtle topic in python…

- what happens to the value of a variable that you pass to a function?

```
def add_one(a):
    a = a + 1
    return a

b = 5
add_one(b)
print b          #prints 5, didn't change
```

but, if you pass a list…

```
def add_one_element(a):
    a.append("one more")

myl = [4]
add_one_element(myl)
print myl          #prints [4, ‘one more’]
```

huh? keep this in mind, will revisit later

two important built-in functions

- range() and xrange()
- xrange() is very similar to range() but outside of the scope of this presentation

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

range(5, 10)
[5, 6, 7, 8, 9]

range(0, 10, 3)
[0, 3, 6, 9]
```

loops

loops

- an essential part of programming
- for loop through a sequence of values
- while loop as long as condition is True
- break
- continue

for loop

```
#!/usr/bin/env python

for i in range(0,10):
    if i > 7 :
        print "break at i={0}".format(i)
        break
    if i > 4 :
        print "continue at i={0}".format(i)
        continue
    print i
```

while loop

```
i = 0
while i < 10:
    if i > 7:
        print "break at i={0}".format(i)
        break
    if i > 4 :
        print "continue at i={0}".format(i)
        i = i + 1
        continue
    print i
    i += 1  #shorthand for i = i + 1
```

careful with infinite loops!

lab 1

data structures

data structures

- programming is all about algorithms and data structures
- compound data types
- ways to organize information efficiently
- which one to choose?
  - fast retrieval of what was stored
  - fast updates
  - time complexity (advanced notion)
  - best memory footprint
- we will cover:
  - list
  - dictionary

lists

```
mylist = [44, 23, 99, 7, ‘bear’]

dir(mylist)    #to see the list interface
['__add__', '__class__’,…., 'append',
'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']

help(mylist.pop)

mylist = []   #empty list
mylist.append(‘Felix’)
mylist.append(‘GA’)
mylist.append(‘14’)
print mylist
[‘Felix’, ‘GA’, 14]
```

list interface

```
append(x)       #insert at the end of the list
count(x)        #return number of occurrences of x
extend(x)       #extend list by appending items from x
index(x)        #return first index of value x
insert(i,x)     #insert object x before index i
pop()           #remove and return last item
remove(x)       #remove first occurrence of value x
reverse()       #reverse in place
sort()          #sort in place, or use sorted() function
```

list indexing and slicing

```
myl = [0, 1, 2, 3, 4, 5]
myl[0]
0
myl[6]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range

myl[3:]      #slicing!
[3,4,5]
myl[:4]
[0,1,2,3]
myl[2:4]
[2,3]
myl2 = myl[:] #this makes a copy
```

strings are sequences

```
s = "felix”
type(s)
<type 'str'>
s[0]
'f’
s[-1]      #negative index! what does it mean?
'x’
s[2:]
'lix’
s[::2]     #skip every other
’flx’
s[0]=‘g’   #does it work?
len(s)
5
h=‘python’
s+h        #adding strings?
```

dictionaries

- aka associative arrays or maps
- indexed by immutable keys not by range of numbers
- ints and strings can always be used as keys
- lists can not, they are mutable
- {}

dictionaries

```
tel = {'jack': 4098, 'sape': 4139}

tel['guido'] = 4127

tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}

tel['jack']
4098

del tel['sape']

tel['irv'] = 4127
tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}

tel.keys()
['guido', 'irv', 'jack']
'guido' in tel
True
```

loops revisited

```
animals = [“cat”, “dog”, “bird”]

for animal in animals:
    print animal

for index, value in enumerate(animals):
    print index, value

my_dict = {‘e’:1, ‘p’:32, ‘q’:2}

for key in my_dict.keys():
    print key

for value in my_dict.values():
    print value

for k, v in my_dict.items():
    print(k,v)
```

the “pythonic” way

“a term used by the Python community to describe code that follows a particular style. The idioms of Python have emerged over time through experience using the language and working with others” –Effective Python

```
i = 0
while i < mylist_length:
    do_something(mylist[i])
    i += 1
```

```
for i in range(mylist_length):
        do_something(mylist[i])
```

```
for element in mylist:
    do_something(element)
```

pythonic

lab 2

the standard library

working with the standard library

- import module
- from module import object

```
import calendar
dir(calendar)
help(calendar)
birthday = calendar.weekday(1987, 06, 14)
print calendar.day_name[birthday]
Sunday
```

- can not stress enough the depth of the libs

file input / output

```
f = open('workfile', 'w')
print f
<open file'workfile', mode 'w' at 80a0960>

f.readline()
"This is the first line of the file.\n"

for line in f:
    print line

f.write("output string")

f.close()    # don't forget this!
```

pythonic i/o

it is good practice to use the **with** keyword when dealing with file objects. This has the advantage that the file is properly closed after its suite finishes, even if an exception is raised on the way. It is also much shorter than writing equivalent try-finally blocks:

```
with open('workfile', 'r') as f:
    read_data = f.read()

f.closed
True
```

the comma separated values format and the csv format

```
import csv

with open('eggs.csv', 'rb') as csvfile:

    spamreader = csv.reader(csvfile, delimiter=‘,’)

    for row in spamreader:
          print ', '.join(row)

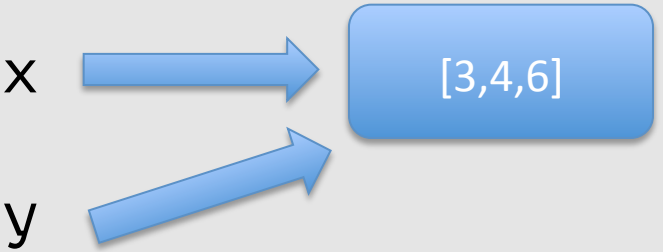
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
```

Each row read from the csv file is returned as a list of strings.

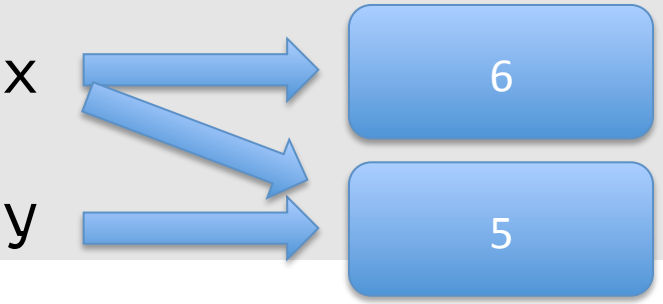
objects

python object model

```
x = []  # lists are mutable
y = x   # creates a new var referring to the same mutable obj
y.append(10)
print y
[10]
print x
[10]
```




```
x = 5       # ints are immutable
y = x
x = x + 1   # 5 can't be mutated,
            # we are creating a new object here (6)
print x
6
print y
5
```



http://jeffknupp.com/blog/2013/02/14/drastically-improve-your-python-understanding-pythons-execution-model/

pass by value,
pass by reference,
pass by assignment

```
def func2(a, b):
    a = 'new-value'          # a and b are local names
    b = b + 1                # assigned to new objects
    return a, b              # return new values

x, y = 'old-value', 99
x, y = func2(x, y)
print(x, y)                  # output: new-value 100
```

mutable objects(list, dict, set, etc) are changed within a function call
immutable objects (int, str, tuple, etc) are not changed

https://docs.python.org/2.7/faq/programming.html#why-did-changing-list-y-also-change-list-x

object identity, type, value

- Every object has an **identity**, a **type** and a **value**.
- An object's ***identity*** never changes once it has been created; you may think of it as the object's address in memory. The '**is**' operator compares the identity of two objects; the **id()** function returns an integer representing its identity.
- An object's ***type*** is also unchangeable. An object's type determines the operations that the object supports (e.g., "does it have a length?") and also defines the possible values for objects of that type.
  The **type()** function returns an object's type (which is an object itself).
- The ***value*** of some objects can change. Objects whose value can change are said to be *mutable*; objects whose value is unchangeable once they are created are called *immutable*. An object's mutability is determined by its type; for instance, numbers, strings and tuples are immutable, while dictionaries and lists are mutable.
- https://docs.python.org/2.7/reference/datamodel.html

object oriented paradigm

- Why objects?
- Encapsulation of data:
  - class ElementaryParticle
    - name
    - mass
    - spin
- Inheritance:
  - class ChargedElementaryParticls
    - Charge
- Let's write our first Python class!

class Animal

```
class Animal(object):

    def __init__(self, name):          #self is always first parameter
        self.name = name

    def talk(self):
        print "Generic Animal Sound”


animal = Animal("thing")
animal.talk()
```

inheritance

```
class Dog(Animal):

    def talk(self):
        print ’{0} says, wow!’.format(self.name)


dog = Dog(”Felix")
dog.talk()                  # invoke method dog.talk()
```

strive for elegant style!

- the PEP08 Style Guide for Python Code:
  - https://www.python.org/dev/peps/pep-0008/
- google Python Style Guide
  - https://google-styleguide.googlecode.com/svn/trunk/pyguide.html
- function names:
  - calculate_speed()
- variables:
  - number_of_particles
- constants:
  - SPEED_OF_LIGHT
- classes:
  - ChargedParticle
- white spaces, etc:
  - Read the links above

PEP = Python Enhancement Proposal

pearls of wisdom

- write short functions
  - rule of thumb: if function is more than a screen full it needs to be broken down
  - will help you think in a modular way
  - helps you write reusable code
- write short lines
  - don't pack too much in a single line of code
  - very difficult to read
  - if you write complex one-liners it doesn't make you smarter
  - next person to read your code will hate you!
- avoid global variables

lab 3

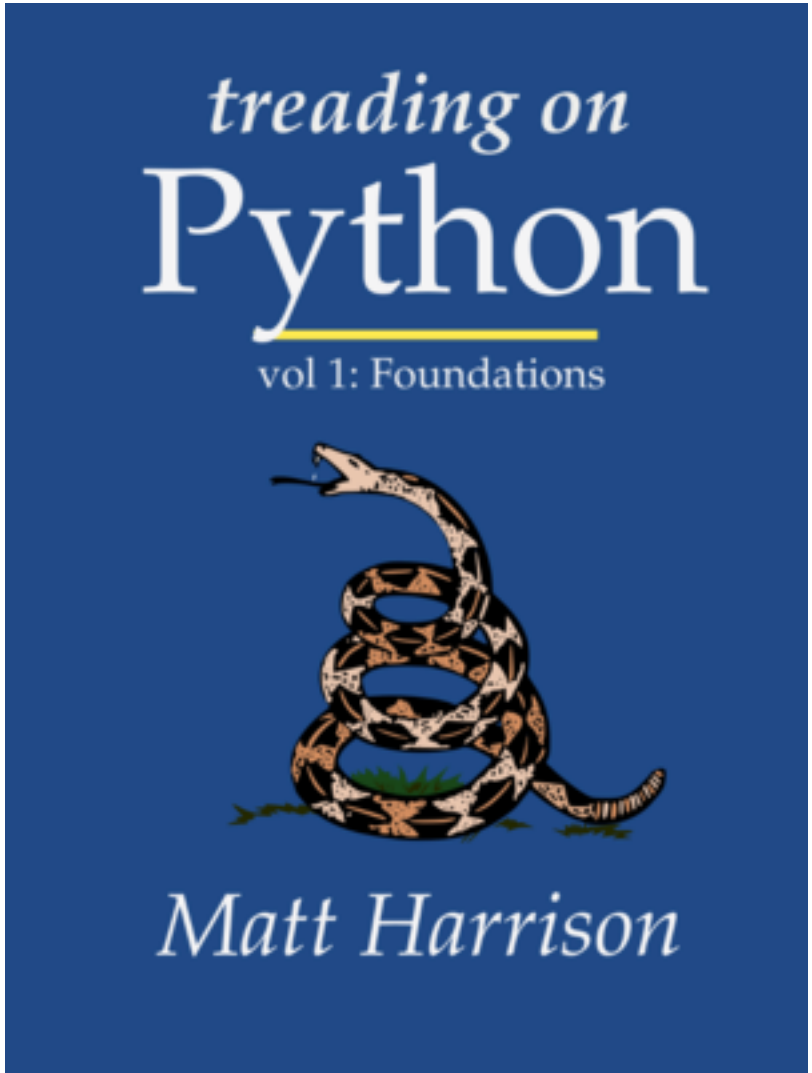
q&a

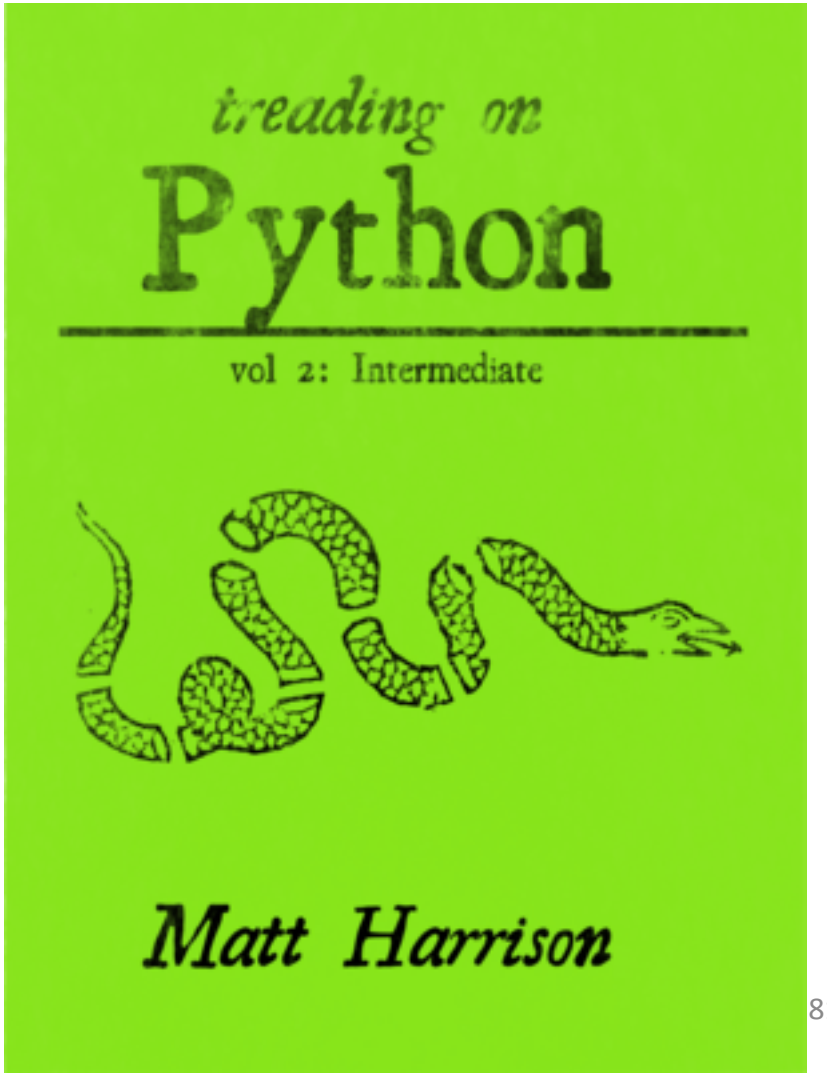
Important Resources

- The Zen of python: https://www.python.org/dev/peps/pep-0020/
- Keywords: http://www.programiz.com/python-programming/keyword-list
- Standard library: https://docs.python.org/2/library/
- Language reference: https://docs.python.org/2/reference/index.html
- Programming FAQ: https://docs.python.org/2.7/faq/programming.html
- Major python website: https://www.python.org/
- The python package index: https://pypi.python.org/pypi
- Fast access to official documentation: https://docs.python.org/2/
- Dive into python: http://www.diveintopython.net/index.html
- Google's python class: https://developers.google.com/edu/python/
- Google's exercises: https://developers.google.com/edu/python/exercises/basic

My favorite two little books

https://leanpub.com/u/mharrison





My favorite advanced book (series)

http://www.effectivepython.com