Programming in Python

Week 2

Syllabus:
`http://briandorsey.github.com/uwpython_onsite/`

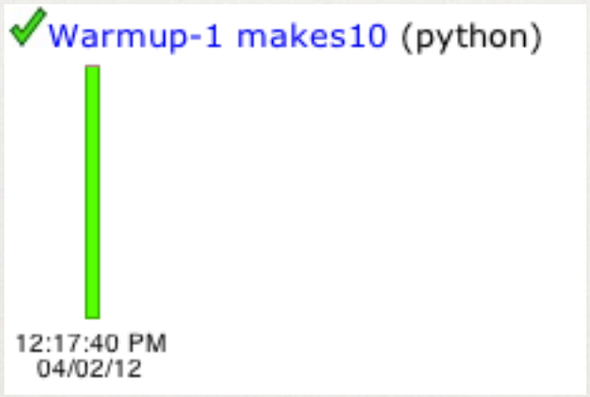
Brian Dorsey
brian@dorseys.org
http://briandorsey.info

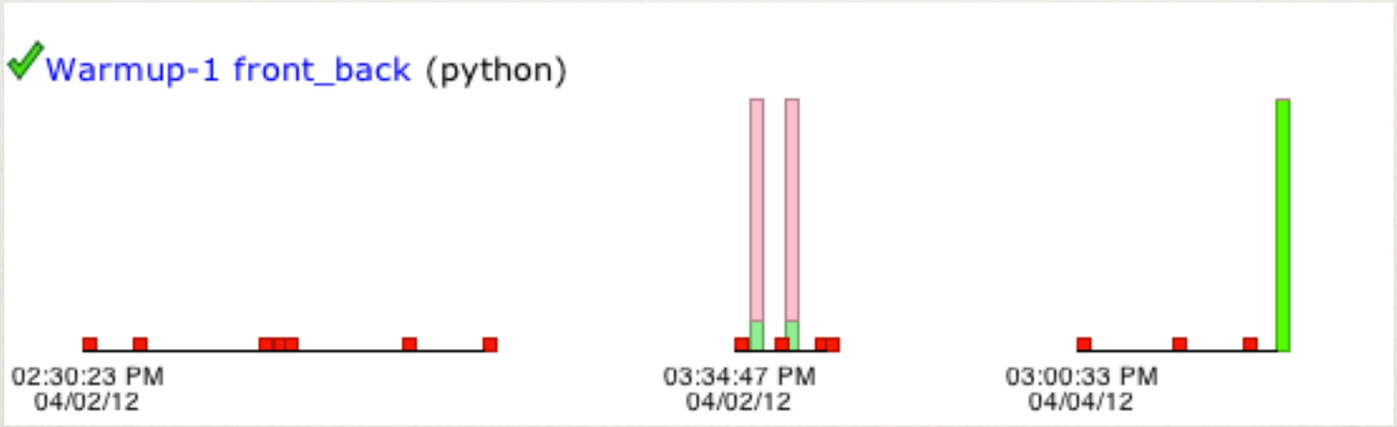
Introductions, comments (...)

about me

homework review

codingbat?


Warmup-1 makes10 (python)
12:17:40 PM
04/02/12


Warmup-1 front_back (python)
02:30:23 PM
04/02/12
03:34:47 PM
04/02/12
03:00:33 PM
04/04/12

overwhelmed?

assign weeks for lightning talks

review

different ways to run programs

values

variables

oh!

...and the interactive interpreter eats **`None`** values

this *might* be relevant for the labs....

next few classes:

you need to read the text

no complete review in class:

highlights,

gotchas,

easy to miss points

Unit A
(25)
Functions

minimal function

```
def <name>():
    <statement>
```

pass statement

```
def <name>():
    pass
```

def is a statement

it is executed

it creates a local variable

functions are just variables pointing at a `<type 'function'>` value

function **`def`**s must be executed before the functions can be called

functions call functions

this makes a stack

that's all a trace back is

traceback

```
def exceptional():
    print "I am exceptional!"
    print 1/0

def passive():
    pass

def doer():
    passive()
    exceptional()

if __name__ == '__main__':
    doer()
```

```
----------------------------------------

I am exceptional!
Traceback (most recent call last):
  File "functions.py", line 15, in <module>
    doer()
  File "functions.py", line 12, in doer
    exceptional()
  File "functions.py", line 5, in exceptional
    print 1/0
ZeroDivisionError: integer division or modulo by zero
```

Anything after a return statement will never get run.

Only one return statement will ever be executed.

Ever.

This is useful when debugging!

3.12 Why functions?

1. Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read and debug.

2. Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.

3. Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.

4. Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.

Lab A

(20)

Lab A

```
git clone https://github.com/briandorsey/uwpython_onsite.git uwpython
```

```
svn checkout https://github.com/briandorsey/uwpython_onsite/trunk uwpython
```

- write print_() function which takes a string and prints it
- what is returned when a function has no return statement? (like the above print_() function)
- write a function which takes two arguments, adds them together and returns the result. Test with pairs of int, float and strings... then pass an int and a string.
- bonus - write a version of print_() which takes any number of arguments and prints with a space between them. (requires research and topics we haven't covered)

Unit B (25)

Block structure.
Scope.
Modules and import.
Truthiness.

Scopes

http://www.flickr.com/photos/jronaldlee/5566380424/

all top level statements are executed on import...

including def

arguments are evaluated before calling

only once

variable names passed to functions calls are unrelated to name inside function

```
def print_twice(bruce):
  print bruce
  print bruce
```

```
brian = 'brian'

print_twice('phil')
print_twice(brian)
```

remember:

it's OK to have two names pointing to the same value

local variables are local
can't use them from outside

also: locals()

also,

DRY

also,

DRY

Don’t Repeat Yourself

def

=

each add a name
to the local scope

There are others as well...

all variables are locally scoped by default

not the only choice:

JavaScript is
global by default

good discussion of scopes:

http://docs.python.org/tutorial/classes.html#python-scopes-and-namespaces

Recursion, max stack depth, function call overhead.

Because of these two(?), recursion isn't used *that* often in Python.

Boolean expressions.

`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**

http://docs.python.org/library/stdtypes.html

```
a or b or c or d
```

```
a and b and c and d
```

what is truth?

thruthiness

determining thruthiness

```
bool(something)
```

False

- 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 orbool value False.

http://docs.python.org/library/stdtypes.html

avoid

```
if xx == True:
```

use

```
if xx:
```

```
True == 1
False == 0
```

ick

```
True = 42
```

good answer, but... ick

Lab B

(20)

Lab B

```
git clone https://github.com/briandorsey/uwpython_onsite.git uwpython

svn checkout https://github.com/briandorsey/uwpython_onsite/trunk uwpython
```

- take some code with functions, add this to each function:

```
for name, value in locals().items():
    print name, value
```

- inside a function add:

```
import sys
```

both inside and out, add:

```
print sys.version_info
```

what does this tell you about import?

- Play with ways to import, what do they do to locals?

```
import sys
from sys import version, version_info
from sys import *
import sys as name
name = __import__('sys')
```

- Short answer: always use import.
- As usual, there are a number of exceptions to this rule:
- The Module Documentation Tells You To Use from-import. The most common example in this category is Tkinter, which is carefully designed to add only the widget classes and related constants to your current namespace. Using import Tkinter only makes your program harder to read; something that is generally a bad idea.
- You're Importing a Package Component. When you need a certain submodule from a package, it's often much more convenient to write from io.drivers import zip than import io.drivers.zip, since the former lets you refer to the module simply as zip instead of its full name. In this case, the from-import statement acts pretty much like a plain import, and there's not much risk for confusion.
- You Don't Know the Module Name Before Execution. In this case, use __import__(module) where module is a Python string. Also see the next item.
- You Know Exactly What You're Doing. If you think you do, just go ahead and use from-import. But think twice before you ask for help ;-)

`http://effbot.org/zone/import-confusion.htm`

Unit C

(25)

Conditionals,
control flow,
iteration,
recursion,
reading and writing programs.

if

```
if a:
    print 'a'
elif b:
    print 'b'
elif c:
    print 'c'
else:
    print 'that was unexpected'
```

if

```
if a:
    print 'a'
elif b:
    print 'b'

########

if a:
    print 'a'
if b:
    print 'b'
```

no switch / case in Python

use if..elif..elif..else

...or dictionary lookups... but I'll talk about that in a few weeks.

`if` is the only choice for branching logic

OK, there are other options, exceptions, dictionary of functions, generator tricks, etc... it is programming after all.

if

```
if a:
    print 'a'
elif b:
    print 'b'
elif c:
    print 'c'
else:
    print 'that was unexpected'
```

for

```
for x in s:
    print x
```

```
>>> for i in range(5):
...     print i
...
0
1
2
3
4
```

```
>>> for i in range(1, 5):
...     print i
...
1
2
3
4
```

```
>>> for i in range(1, 6):
...     print i
...
1
2
3
4
5
```

Python doesn't have
a three part for loop:

JavaScript:

```
for(var i=0; i<arr.length; i++) {
  var value = arr[i];
  alert(i =") "+value);
}
```

use range()?

```
>>> letters = 'Python '
>>> for i in range(len(letters)):
...     c = letters[i]
...     print c
...
P
y
t
h
o
n
```

if [i] looks funny, no worries, we haven't talked about it yet... it's called 'indexing' in Python

more Pythonic

```
>>> letters = 'Python '
>>>
>>> for c in letters:
...     print c
...
P
y
t
h
o
n
```

for loops:
never index in normal cases

if index is needed,
enumerate or slice

slice? that’s in two weeks.s

enumerate()

```
>>> letters = 'Python '
>>>
>>> for i, c in enumerate(letters):
...     print i, c
...
0 P
1 y
2 t
3 h
4 o
5 n
6
```

while

```
# executes body as long as p is true
while p:
    ...
```

that's it

while is more general
than **`for`**

can always express **`for`**
as **`while`**,
but not always vice-versa

while is more error-prone,
requires some care to terminate

loop body must make progress,
so p can become False

potential error: infinite loops

```
letters = 'Python '

i = 0
while i < len(letters):
    print letters[i]
    i += 1

# vs:

letters = 'Python '

for c in letters:
    print c
```

what if len(letters) == 0 ?

```
# Shortcut: recall number 0
# or empty collection is False

while x:     # terminates if x >= 0 on entry
    ...      # do something with x
    x -= 1   # make progress toward 0
```

continue

break

`else` again

```
# else executes at end of sequence, or when while
condition is false

for x in s:
    ... # do something
else:
    ... # do something


while p:
    ... # do something
else:
    ... # do something
```

continue will hit else, break skips it!

`while` vs. `for`

clarity

recursion

"If you try to follow the flow of execution here, even for fairly small values of n, your head explodes."

pg. 58

most useful with data structures like trees or graphs

temporary data
is in the call stack

Lab C (20)

Lab C

```
git clone https://github.com/briandorsey/uwpython_onsite.git uwpython
```

```
svn checkout https://github.com/briandorsey/uwpython_onsite/trunk uwpython
```

- Look up the % operator. What do these do?

  ```
  10 % 7 == 3
  14 % 7 == 0
  ```

- Write a program that prints the numbers from 1 to 100 inclusive.
  But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”.
  For numbers which are multiples of both three and five print “FizzBuzz” instead.

FizzBuzz

http://www.codinghorror.com/blog/2007/02/why-cant-programmers-program.html

WRAPUP

Assignment

- Continue CodingBat exercises,
  6 more by next week.