Programming in Python

Week 3

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

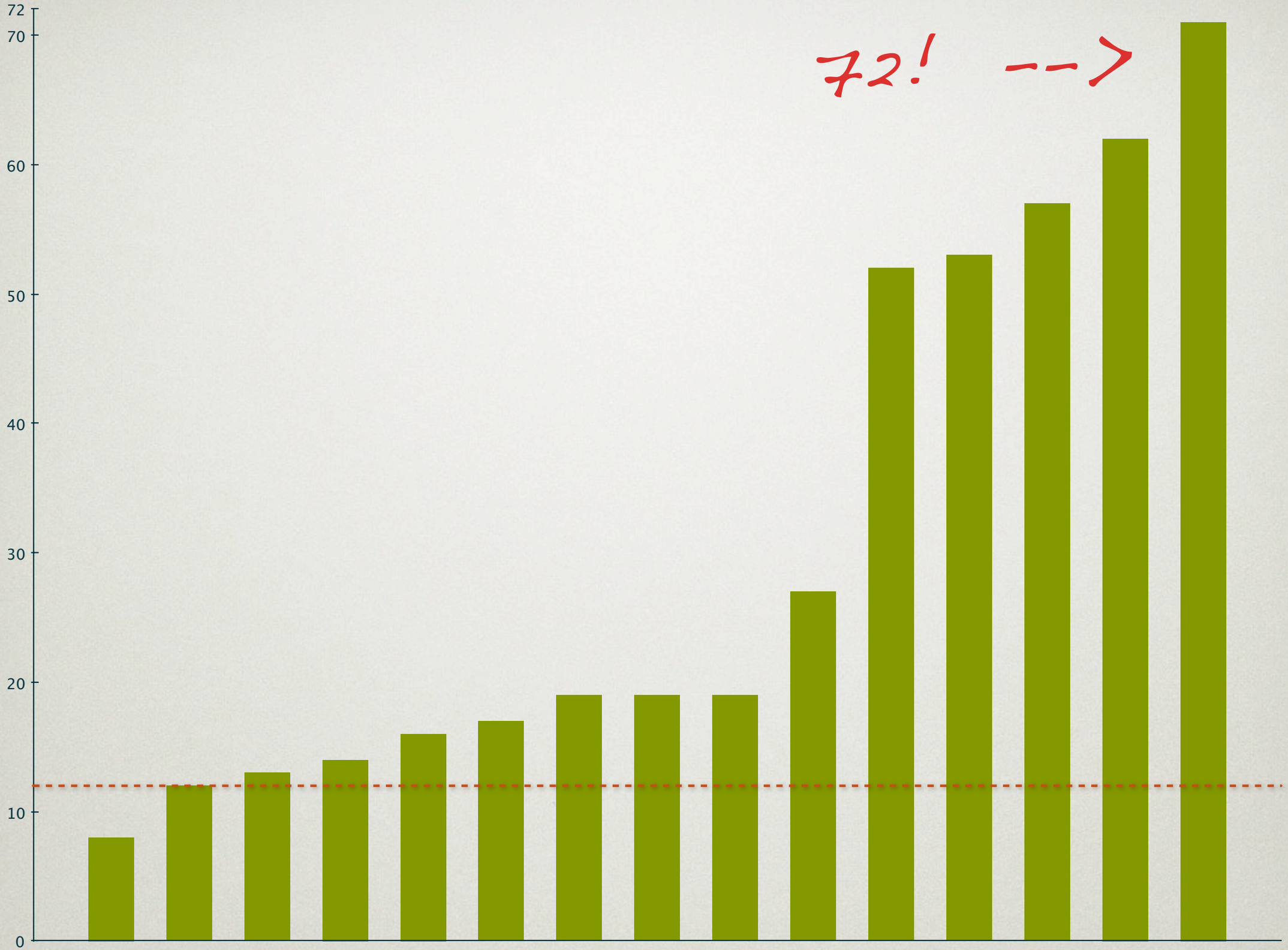
Brian Dorsey
brian@dorseys.org
http://briandorsey.info

Introductions, comments (...)

readings this week

questions?

CodingBat


72!
-->
72
70
60
50
40
30
20
10
0

How many attempted it?

FizzBuzz

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

- 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.

- 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.

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
...
```

weird little programming problem

it’s being used as a filter:

“I've discovered that people who struggle to code don't just struggle on big problems, or even smallish problems. They struggle with tiny problems.”

http://tickletux.wordpress.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding/

philosophy detour:

tacit knowledge
&
fractals

tacit |ˈtasit|

adjective

understood or implied without being stated

tacit knowledge & skills
are learned by doing

watch, copy, do, refine

experience makes us better

hard to teach, because
words are not enough

can’t become an expert
through books only

examples?

any skill with an apprenticeship probably needs lots of tacit knowledge

... at least on real computers anyway.

programming is tacit knowledge

this is the root of the FizzBuzz test

can't *actually* teach programming

problems of all sizes have the same fundamental issues

fractal nature of problems

most of the same issues come up no matter how long you have to work on the problem...

15 minutes

hour

week

month

year

that kinda sucks

it’s also awesome

need experience to get better

work on lots of small problems

lots of experience

pattern matching

low risk

all practice
CodingBat
programming katas

6 hour ‘startup’
Startup Weekend

Unit A
(25)

sequences
strings

these topics are used in nearly all Python programs

this week and next are the most important in the whole class

remember `enumerate()`?

iteration / for loops

all sequences are iterable

http://docs.python.org/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange

sequences

str

unicode

list

tuple

bytearray

buffer

xrange

strings only today

but...

lists

```
l = [1, 3.14, 'abc']
```

operators on sequences

```
>>> s = 'supercalifragilistic'
>>> t = 'expealidocious'
>>> n = 3
>>> s + t
'supercalifragilisticexpealidocious'
>>> s * n
'supercalifragilisticsupercalifragilisticsupercal
ifragilistic'
>>> s == t
False
>>> s < t
False
>>> s > t
True
```

http://en.wikipedia.org/wiki/Longest_word_in_English

`[i]` is also an operator

indexing

```
sequence[i]
```

indexing works on all sequences

0 based

```
'Python '
```

```
>>> p = 'Python '
>>> p[1]
'y'
```

```
>>> p = 'Python '
>>> p[1]
'y'
>>> p[0]
'P'
```

full disclosure: `sequence[start:end:step]`

slicing

`sequence[start:end]`

`sequence[start:end]`

from **`start`**,

up to, but *not including,* **`end`**

from start, up to but not including end

```
>>> s = 'Washington'
>>> start = 3
>>> end = 5

>>> s[0:end]
'Washi'
>>> s[:end]       # items from the beginning through (end - 1)
'Washi'

>>> s[start:len(s)]
'hington'
>>> s[start:]     # items start through the rest of the sequence
'hington'

>>> s[:]          # a copy of the whole sequence
'Washington'

>>> s[start:end]  # items start through (end - 1)
'hi'
```

```
>>> s = 'Washington'
>>> start = 3
```

```
>>> s[:start] + s[start:]
```

?

```
>>> s = 'Washington'

>>> s[-2:]       # last two items in the sequence
'on'

>>> s[:-2]       # everything except the last two items
'Washingt'
```

closed - open

```
>>> s = 'Washington'
>>> start = 3
```

```
>>> s[:start] + s[start:]
'Washington'
```

```
len(sequence)
min(sequence)
max(sequence)
```

```
any(sequence)
```

```
all(sequence)
```

x is a single object

```
x in sequence
```

```
x not in sequence
```

strings

so, all of the previous slides apply

strings are sequences

special cases

x is a single object - but for strings it *can* be more than one character

```
x in sequence
```

```
x not in sequence
```

immutable

```
>>> s = 'Washington'
>>> s[1] = 'i'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
assignment
```

if you need to do a bunch of character level modification, turn it into a list of single characters

==

>

<

+

+=

wait... aren't strings immutable? what comes back from lower()?

```
s.lower()
s.upper()

...

s.capitalize()
s.swapcase()
s.title()
```

```
x in s
s.startswith(x)
s.endswith(x)
...
s.index(x)
s.find(x)
s.rfind(x)
```

```
s.split()
s.join(list)
...
s.splitlines()
```

string formatting

please don’t do this:

```
'Hello ' + name + '!'
```

much safer, easier to modify as code gets complicated

```
'Hello %s!' % name
```

http://docs.python.org/library/stdtypes.html#string-formatting-operations

```
dictionary = {'name': 'Joe'}

'Hello %(name)s!' % dictionary
```

:(

string formatting

!=

advanced string formatting

http://www.python.org/dev/peps/pep-3101/

```
'Hello {0}!'.format(name)
```

```
'Hello {name}!'.format(**dictionary)
```

recommendation

pick one
(probably ‘string formatting’)
get comfy with it

Lab A

(20)

http://en.wikipedia.org/wiki/ROT13

ROT13 encryption

Applying ROT13 to a piece of text merely requires examining its alphabetic characters and replacing each one by the letter 13 places further along in the alphabet, wrapping back to the beginning if necessary

GEOCACHING

You are not log

Waterfall Park Cache

GCHXVJ

Created by: nolenator

Date Hidden: 3/14/2004

Difficulty:

Terrain:

Cache Size:

Favorite Points: 37

Login to your Premium Member account or learn more about Premium Membership.

1 of 1 nearby results · < Prev | Next >

Yesler Way
Merchant's Cafe
South
Prefont
4th Ave South
Station 10
99
Boat Landing (Historical)
South Washington St
201
Alaskan Way Viaduct
Alaskan Way South
210
Garden
South Main St
South Main S
P
Union Station Sq.
SEATTLE
South Jackson St
1st Ave South
101
Occidental Ave South
2nd Ave South
2nd Ave Extension South
King Street Station Park
Union Station
South King St
3rd Ave South
Track 2
Track 1
Toll Booth
Track 3
Weller St
Railroad Way South
Track 4
Track 5
Track 6
Track 7
Gate
Downtown Seattle Transit Tunnel

Waterfall Park Cache

A cache by nolenator Hidden : 3/14/2004

Difficulty: ★★½☆☆☆
Terrain: ★★½☆☆☆

Size: (Micro)

You must be logged in with a Geocaching.com account to view the specific location of this geocache. It's free!

N/S ? ??.??? W/E ??? ??.??? In Washington, United States

Print:
5 Logs 10 Logs ·

Download: Read about waypoint downloads

Please note

Use of geocaching.com services is subject to the terms and conditions in our disclaimer.

Waterfall Park in Downtown Seatlle

This is the second in my downtown urban cache series. Waterfall Park is an oasis in the middle of an urban area. It is a great place to sit back and relax for a few moments or enjoy your brown bag lunch. The twenty-two foot waterfall is just mesmerizing. Eight Japanese stonemasons built the park in 1977 on the original location of the United Parcel Service. The park has very limited hours: 0800 to 1545 in the winter and 0800 to 1745 in the summer (after daylight savings time ends). The cache is now accessable anytime, but the waterfall only runs while the park is open. Bonus points if you take your picture in front of the waterfall. Who knows you might make the GC.com banner...

Additional Hints (Decrypt)

Zntargvp sebz bhgfvqr arne pbeare

```
Decryption Key
A|B|C|D|E|F|G|H|I|J|K|L|M
-------------------------
N|O|P|Q|R|S|T|U|V|W|X|Y|Z
(letter above equals below, and vice versa)
```

```
git pull

svn update
```

Lab A

- play with indexes
- play with slices
- play with string formatting
- write a mail merge program
- implement rot13 decoding
  decode this message:
  Zntargvp sebz bhgfvqr arne pbeare
-

Lightning Talks!

(15)

Unit B
(25)

files

secret_data is a string!

```
f = open('secrets.txt')
secret_data = f.read()
f.close()
```

f = open('secrets.txt', [mode])

r, w, a

r, w, a

r+, w+, a+

U rU

b

gotcha

w mode always clears the file

```
header_size = 4096

f = open('secrets.txt')
secret_data = f.read(header_size)
f.close()
```

```
for line in open('secrets.txt'):
print line,
```

how does secrets.txt get closed?

```
output = open('output.txt', 'w')

for line in open('secrets.txt'):
    line = do_something(line)
    output.write(line)

output.close()
```

http://docs.python.org/library/stdtypes.html#bltin-file-objects

StringIO

http://www.doughellmann.com/PyMOTW/subprocess/

```
p = subprocess.Popen(['ls'], stdout=subprocess.PIPE)
p.stdout.read()  # <-- p.stdout is 'file-like'
```

```
# new in Python 2.7
# blocks until process completes and reads all of stdout
output = subprocess.check_output(['ls'])
```

exceptions

remember when I said that if was the only branching control structure?

BTW, if this is all you're going to do... you should just let the exception bubble up

```
try:
    do_something()
    f = open('missing.txt')
    process(f)   # never called if file missing
except IOError:
    print "couldn't open missing.txt"
```

never do this

```
try:
    do_something()
    f = open('missing.txt')
    process(f)   # never called if file missing
except:
    print "couldn't open missing.txt"
```

if log is actually from the logging module, exception() will include a full traceback in the log entry

```
# log all uncaught exceptions
while True:
    try:
        do_work()
except StandardError:
    log.exception('Unexpected error')
```

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
```

<-- These are the only exceptions your program can do anything about

this will *almost* always work - it's the *almost* which will drive you insane

never do this

```
do_something()
if os.path.exists('missing.txt'):
    f = open('missing.txt')
    process(f)   # never called if file missing
```

“easier to ask forgiveness than permission”

- Grace Hopper

http://en.wikipedia.org/wiki/Grace_Hopper

Excellent talk covering Grace Hopper and forgiveness over permission in software:

http://www.youtube.com/watch?v=AZDWveIdqjY

so you have better debugging info when it fails

for simple scripts, let exceptions happen

only handle the exception if that code can and will do something about it

Unit C

(25)

unicode,
text processing

unicode

everything is actually bytes

specifically, if it's on disk or on a network, it's bytes

Python provides some abstractions while your software is running to make it easier to deal with bytes

Unicode is the biggie

most of things will work pretty much the same in 2.x & 3.x... strings are the biggest exception

strings vs unicode

python 2.x vs 3.x

strings are sequences of bytes

unicode strings are sequences of platonic characters

platonic characters cannot be written to disk or network!

ord()

chr()

unichr()

str()

unicode()

encode()

decode()

unicode demo

text processing

paths

os.getcwd()

both work fine in Python: open(), etc

relative paths

`secret.txt`

`./secret.txt`

absolute paths

`/home/brian/secret.txt`

os.path.abspath()

os.path.relpath()

os.path.split()
os.path.basename()
os.path.dirname()
os.path.join()

os.path.splitext()

os.path functions work cross-platform

with a few exceptions:

UNC paths (windows)

links depend on filesystem

os.listdir()

os.walk()

next week

string concatenation in a loop

sorting

finding things in strings

Lab C (20)

```
git pull

svn update
```

Lab C

- write a program which prints the full path to all *files* in the current directory, one per line
- write a program which copies a file from a source, to a destination
  (without using **`shutil`**, or the OS copy command)
- update mail-merge from the previous lab to write output to individual files on disk
- advanced - implement your own iterator

WRAPUP

Assignment

- CodingBat - 12 more string & list problems
  or
- write a script which does something useful (to you) and reads & writes files. Very, very small scope is good.
  Something useful at work would be great, but no job secrets!