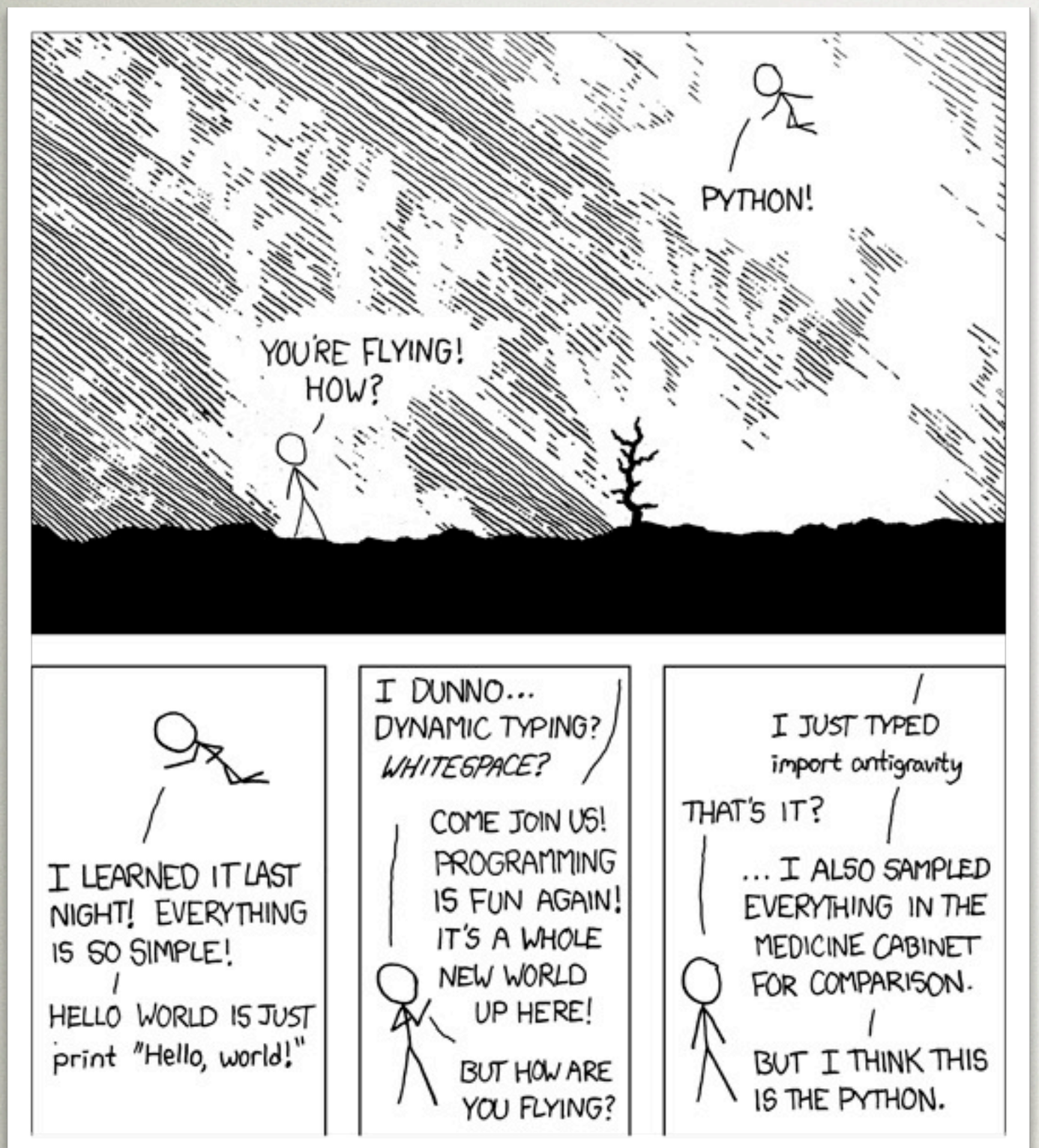


http://xkcd.com/353/

Programming in Python

University of Washington Professional and Continuing Education

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

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

Bree Overly
UW PC&E
boverly@pce.uw.edu

Introductions, What to expect, What is Python?

(...)

introductions

you

Brian Dorsey

brian@dorseys.org

this class

What have you gotten into?

fast paced introduction class - with a few extra topics
in a month, we'll have learned enough to write a webserver from scratch

syllabus:

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

questions?

book

official text book

Think Python

Best of the Python introductions -- really explains what is actually going on
Axiomatic: simple explanations and examples build up understanding incrementally
Axiomatic approach starts from zero, assumes nothing, omits nothing
Axiomatic approach requires patience with "trivial" examples -- humor him!

alternate text book

Learn Python the Hard Way

http://learnpythonthehardway.org/

Pragmatic, exercise driven, learn by doing, repetition.
Closely matches real-world problem solving.
You *must* type the programs in by hand, or the style doesn't work.

- talks: review, highlights, experience
  not comprehensive:
  you must read & do assignments

- approximately 50/50 talks/lab
  (except today)

- no explicit break times

- interrupt me

- starting at the beginning,
  but move pretty fast

lightning talks

labs

assignments

grading

Class is P/F, show up, put in effort.
once a week isn't enough, purpose of the assignments is to get you *using* python between classes. show me that you're working on something and you'll get credit.

wide range of backgrounds

assignments & labs

when multiple options, pick the first one which looks like a strech

diversity in students

getting help

class mailing list

email me

some thoughts...

Embarrassment of riches

We’re all standing on the shoulders of giants here.
Each of you has the power to create amazing new things... as a hobby!
With this leverage, even a couple of hours a week is amazing.

Beware the trappings of authority.

I’m not the smartest one in the room. But this room is designed to make it look and feel that way. You’re trapped. Apologies. This is what our culture thinks ‘learning’ looks like.
One thing we can do, is make this more like a discussion than a lecture - ask questions anytime.

my role:

curation
share known good paths
unblocking!

Collaboration

Real learning happens when doing.
Real learning happens when teaching.
Software is crazy specialized, *nobody* knows it all, lets all help each other get better at it.

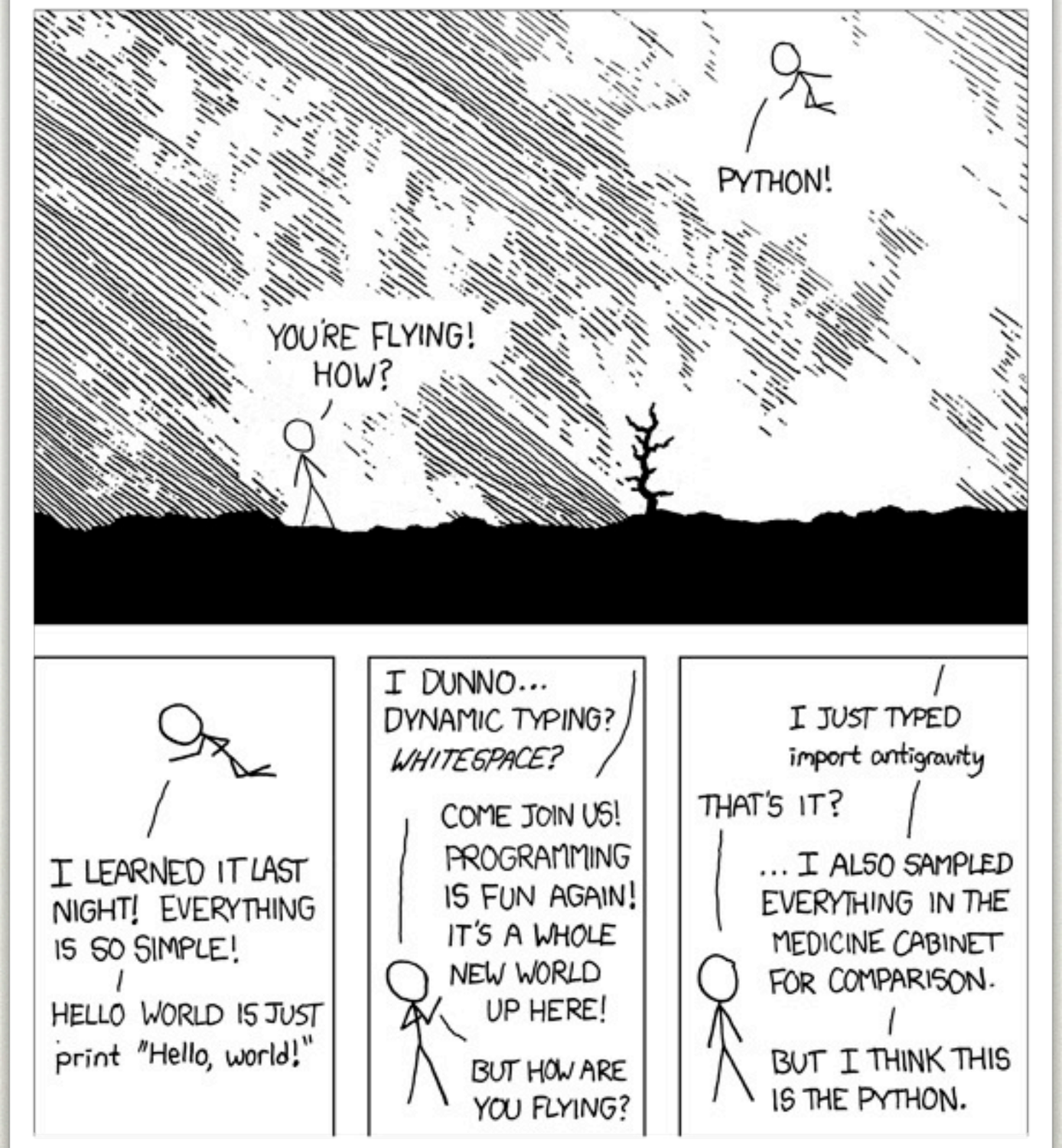
"One of the things I find that's remarkable about Python is that it has a very even learning curve. Maybe it's not even a curve, It's kind of a straight line."

- Bruce Eckel

http://www.artima.com/intv/prodperfP.html

(yes, this is an appeal to authority... two sides after cautioning against it.)

(imagine graph here)



http://xkcd.com/353/

The flat learning curve leads us here.

this is a safe place to fail

we're going to do a lot of it

this is a safe place to

break stuff

we're going to do a lot of it

Unit A (25)

overview

Why Python?

embarrassment of riches
easily connect to anything
good at almost the entire spectrum of programming tasks

What **is** Python?

general purpose programming language


```
 CS education (this course!)
 Application scripting (GIS, GNU Radio, ...)
 Systems administration and "glue"
 Web applications (Django etc. etc. etc.)
 Scientific/technical computing (a la MATLAB, Mathematica, also BioPython etc. ..)
 Software tools (automated software testing, distributed version control, ...)
 Research (natural language, graph theory, distributed computing, ...)

An unusually large number of niches -- versatile
```

used by...

Beginners

Professional computer specialists:
software developers, system administrators, test engineers

Professionals **other than** computer specialists:
biologists, urban planners, GIS, 3d modeling, etc...

interpreted

No compile, build, simple process.
Python code is just a text file. Needs an interpreter to run.

dynamically, strongly typed

automatic memory management

multiple implentations

cPython

PyPy

Jython

IronPython

• • •

python is glue

What can you **do** with Python?

almost anything within the spectrum of programming tasks

What **can't** you do with Python?

hard real-time
drivers
operating systems

an aside about testing

http://wiki.python.org/moin/
PythonTestingToolsTaxonomy

Lab A
(...)

groups of 4ish if possible, pairs if not

Lab A

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

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

- log into VMs
  check out course materials
- run **`hello.py`**
  ```
  $ cd uwpython
  $ cd week01
  $ python hello.py
  ```
- **`http://learnpythonthehardway.org/book/ex1.html`**
- **`http://learnpythonthehardway.org/book/ex3.html`**
- bonus: write a program which prints **`sys.argv`** and os.getcwd(). Run it from various directories, and pass it different parameters. Why did I recommend these two?

Unit B
(25)

Orientation

demo:
the interpreter

demo: use as a calculator
what works as you expect?
anything suprising? (integer division?)
help(), dir(), _, up arrow, dir(__builtins__), abs(), ord(), chr()
() is ‘call’ syntax

running programs

Python programs are just text files with python code in them.
Running Python code *always* needs an interpreter.

```
$ python filename
```


familiar with the $ convention?
this is actually the only way to run python scripts, all of the rest of the things I'm about to show are actually shortcuts to this.

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

*.py

Python launcher for Windows

http://www.python.org/dev/peps/pep-0397/

```
$ python -m modulename
```


python -m unittest
python -m turtle
python -m SimpleHTTPServer

```
$ python -i filename
```


run the script, then open an interpreter
exactly as if you had typed the entire file into the interpreter

```
$ python
>>> import hello
Hello World!
```

```
$ python -i hello.py
Hello World!
>>>
```

Documentation

http://docs.python.org/

language reference

http://docs.python.org/reference/index.html

standard library reference (modules)

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

“batteries included”

PEPs

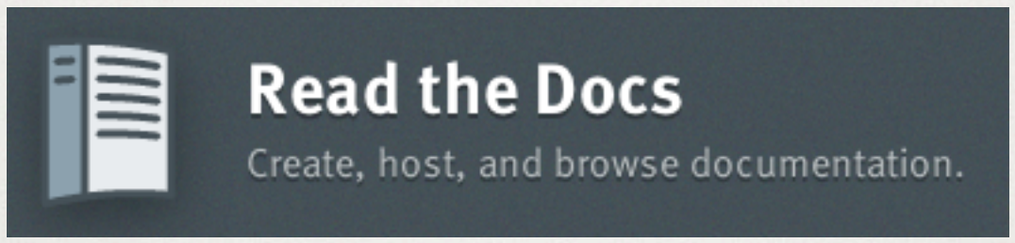
http://www.python.org/dev/peps/

```
PEP 1   PEP Purpose and Guidelines
PEP 8   Style Guide for Python Code
PEP 20  the Zen of Python
```

pydoc

http://docs.python.org/reference/index.html

doc strings



http://readthedocs.org/

google

be careful!

lots of great info out there!
Most of it is opinionated and out of date.
(might still be correct, though!)

Lab B
(20)

Lab B

- use **`python -m`** to run the **`SimpleHTTPServer`** in the checked out class syllabus directory.
  Then visit **`http://machine_name:8000`**
- Use three different techniques for viewing the documentation for the **`sys`** module.
- Start the **`pydoc`** webserver, and see which modules are installed on your machine.
  (hint: **`pydoc`** is a stand-alone program)
- bonus: Write your own simple program with doc strings and see how they appear in the pydoc webserver.

Unit C
(25)

Variables, values, expressions, and assignment.

Values

...but first...

the types mentioned in chapter two

(we'll be covering many more soon)

integers

arbitrarily long integers
i = sys.maxint
technically there is a difference between ‘int’ and ‘long’, but you can ignore it, and it’s gone in python3

float

floating point numbers

strings

arbitrary length
immutable

values

```
42
3.14
"Hello World"
```


the basic building blocks of your program
all of your data ends up as values

print

variables

variables are a name pointing at a value

pointers, ‘by reference’, by value

variables don’t have a type

values do

this is where the dynamic comes from

```
>>> type(42)
<type 'int'>
>>> type(3.14)
<type 'float'>

>>> a = 42
>>> b = 3.14
>>> c = "Hello World!"
```

```
>>> type(a)
<type 'int'>
>>> a = b
>>> type(a)
<type 'float'>
```

variables are just shorthand, so we don't have to type the values over and over, and over and over

oh... and, they allow you write logic without caring what the values are

this is basically the core of what makes programming a useful exercise, if a program cannot *vary*, we might as well just remember the answer and throw the program away

assignment

beware making new variables!

`=` assigns

`==` checks equality

`is` checks identity

`id()` queries identity

multiple assignment

```
a, b = 1, 2
```


not so readable and useful here, but very handy later

`print` again

comma, newlines
single, double, triple quotes

expressions vs. statements

expressions return a value
statements have a side effect, but no value

(In Python 3, print is an expression, not a statement! They broke Hello, world!)

(use parentheses)
(a lot)

`http://docs.python.org/reference/expressions.html#summary`

what happens to
un-assigned values?

(the end of expression
evaluation)

```
# comment

print "Hello World”

"""is
this a
comment?"""
```

Lab C

(25)

Lab C

- visit codingbat.com
  sign up for an account
  goto ‘prefs’ page
  Share To: brian@dorseys.org

- Do one exercise from CodingBat:
  Warmup-1 > monkey_trouble
  Warmup-1 > parrot_trouble
  String-1 > hello_name
  String-1 > make_abba
  or anything that looks like a bit of a challenge.

  all end with:
  `return something`

- bonus:
  http://learnpythonthehardway.org/book/ex3.html
  http://learnpythonthehardway.org/book/ex4.html
  http://learnpythonthehardway.org/book/ex5.html

WRAPUP

Assignment

- Pick something you'd like to automate.
  Do some research (1 hour max). Can it be done using Python? Can it be done with the standard library? What other modules look helpful?
- Coding is the only way to learn to code.
  Do a CodingBat exercise every day.
  Complete any 6 by next week.
- Pick one exercise from assigned chapters which looks challenging, complete and email to me.

all of these have varied difficulty, if you're already a python guru, pick something challenging