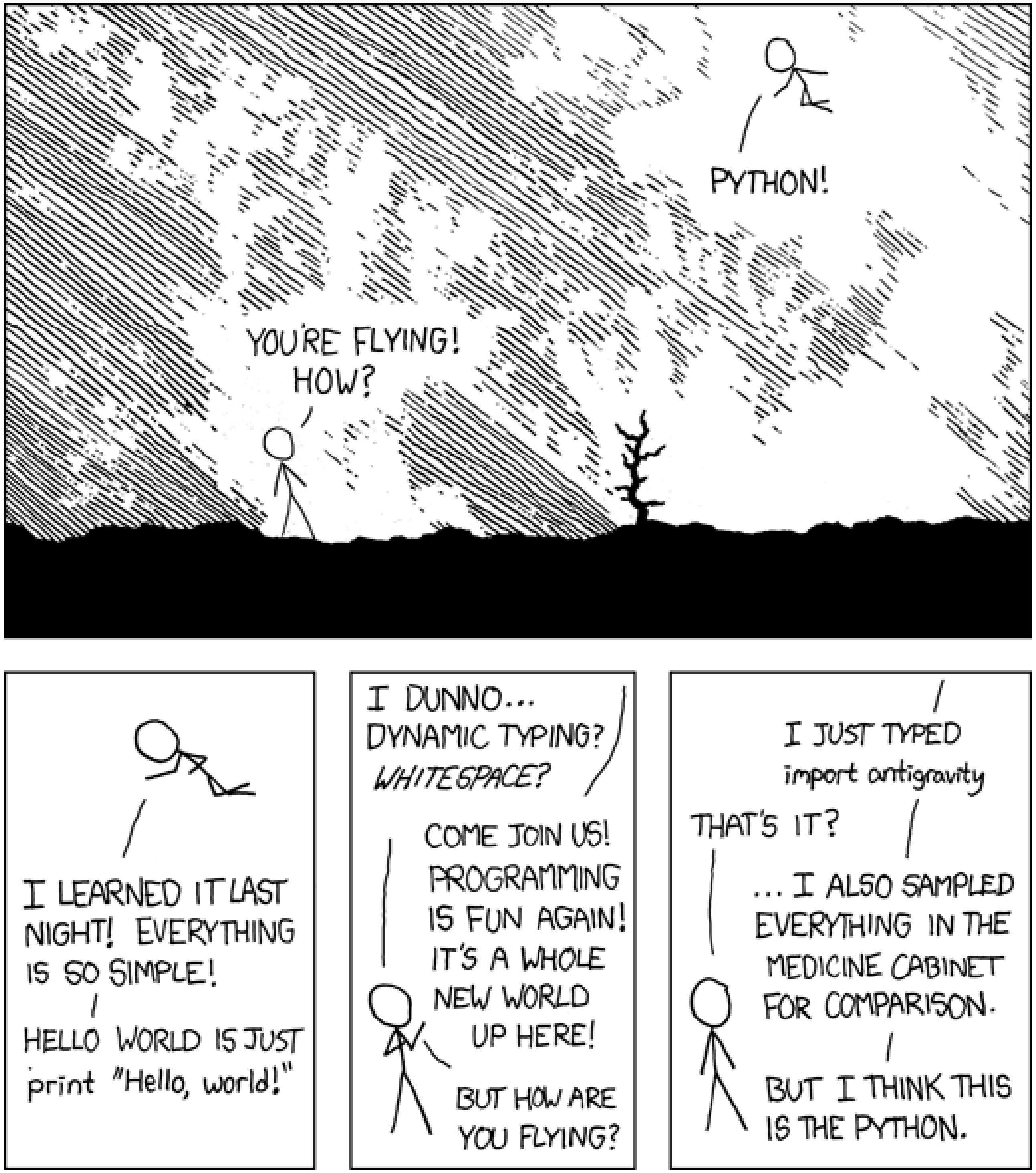

PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

xkcd comic by Randall Munroe (no relation): https://xkcd.com/353/

A Lightning Introduction to Python

Will Monroe
CS 109 tutorial
13 April 2017

Basic syntax

```python
def fizzbuzz(n):
    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            print('fizzbuzz')
        elif i % 3 == 0:
            print('fizz')
        elif i % 5 == 0:
            print('buzz')
        else:
            print(i)

fizzbuzz(100)
```

Basic syntax

~~TYPES~~

```
def fizzbuzz(n):
    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            print('fizzbuzz')
        elif i % 3 == 0:
            print('fizz')
        elif i % 5 == 0:
            print('buzz')
        else:
            print(i)

fizzbuzz(100)
```

~~SEMICOLONS~~

~~BRACKETS~~

Whitespace

- Whitespace matters in Python! Sometimes.

```
if i % 3 == 0 and i % 5 == 0:
    print('fizzbuzz')
elif i % 3 == 0:
    print('fizz')
elif i % 5 == 0:
    print('buzz')
else:
    print(i)
```

Whitespace

- Whitespace matters in Python! Sometimes.

```
if i % 3 == 0 and i % 5 == 0:
    print('fizzbuzz')
elif i % 3 == 0:
    print('fizz')
elif i % 5 == 0:
    print('buzz')
else:
    print(i)
```

INDENT = "{ }"

NEWLINE = "SEMICOLON"

- 4 spaces is a common convention.
- Don't mix spaces and tabs.

Whitespace

- Whitespace on otherwise blank lines is ignored.

```
if i % 3 == 0 and i % 5 == 0:

    print('fizzbuzz')
elif i % 3 == 0:

    print('fizz')
elif i % 5 == 0:
    print('buzz')
else:
    print(i)
```

ARE THERE SPACES HERE? DON'T KNOW, DON'T CARE.

Whitespace

- Whitespace is ignored inside all brackets/parens.

```
if (i % 3 == 0 and i % 5 == 0):

    print('fizzbuzz')
elif i % 3 == 0:

    print('fizz')
elif i % 5 == 0:
    print('buzz')
else:
    print(i)
```

Whitespace

- Whitespace is ignored inside all brackets/parens.

```
if (i % 3 == 0 and 
        i % 5 == 0):
    print('fizzbuzz')
elif i % 3 == 0:

    print('fizz')
elif i % 5 == 0:
    print('buzz')
else:
    print(i)
```

Whitespace

- Newline (+whitespace) is ignored after backslash.

```
if (i % 3 == 0 and
        i % 5 == 0):
    print('fizzbuzz')
elif i % 3 == \
         0:
    print('fizz')
elif i % 5 == 0:
    print('buzz')
else:
    print(i)
```

Dynamic typing

Variables don't have types.

Values do.

Dynamic typing

- A variable is created when you assign to it:

```
x = 3
```

- 3 is an integer. But **`x`** doesn't have to be—you can later give it a string value:

```
if x % 3 == 0:
    x = 'fizz'
```

Functions

- Arguments and return values can also be any type:

WE HOPE THIS IS AN int, BUT IT CAN BE ANYTHING!

```
def fizzbuzzify(i):
    if i % 3 == 0 and i % 5 == 0:
        return 'fizzbuzz'
    elif i % 3 == 0:
        return 'fizz'
    elif i % 5 == 0:
        return 'buzz'
    else:
        return i
```

FUNCTION DEFINITIONS START WITH def. THAT'S ALL YOU NEED!

HERE WE RETURN A STRING…

BUT HERE WE RETURN WHATEVER i IS!

A few small spelling differences

Java	C++	Python
`&&, ||`	`&&, ||`	`and, or`
`else if`	`else if`	`elif`
`true, false`	`true, false`	`True, False`
`"string"`	`"string"`	`'string', "string"`
`// comment`	`// comment`	`# comment`
`null`	`nullptr, NULL`	`None`

The interactive interpreter

Python 2 or Python 3?

- Python 3:
  - fixes some annoying design decisions
  - has a bunch of awesome new features
- but
  - some libraries might not support it

The differences aren't large, but Python 3 is not backwards compatible!

Code in this tutorial should work in both.

String operations

- Make a string:

```
first_name = 'Will'
```

- Concatenate two strings:

```
last_name = 'Monroe'
full_name = first_name + ' ' + last_name
```

- Get one character of a string

```
first_name[2]  # 'n'
```

START FROM ZERO

- Including numbers in strings

```
age = 'I am {} years old'.format(6 * 4)
```

String operations: Slicing

- "Slice" = "substring":

```
>>> full_name[3:9]
"l Monr"
```

START (INCLUSIVE)

END (EXCLUSIVE)

- Grab the first n characters:

```
>>> full_name[:3]
"Wil"
```

- ...or the last:

```
>>> full_name[-3:]
"roe"
```

NEGATIVE = COUNT FROM THE END

Containers

Java	C++ `std::`	Python
`ArrayList`	**`vector`**	**`list [1, 2]`**
`HashMap`	**`map`**	**`dict {'a': 1, 'b': 2}`**
`HashSet`	**`set`**	**`set  {1, 2}`**
(n/a)	**`tuple`**	**`tuple (1, 2)`**

List operations: Building

- Make a list:

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

- Add a single value to the end:

```
numbers.append(4)
```

- Tack another list onto the end:

```
numbers.extend([5, 6])
```

- Concatenate two lists:

```
big_numbers = [7, 8, 9, 10]
lots_of_numbers = numbers + big_numbers
```

List operations: Slicing

- Works the same way as strings:

```
>>> fruit = ['apple', 'banana', 'peach']
>>> fruit[0]
'apple' ← NO COLON: GET A SINGLE ELEMENT
>>> fruit[1:3]
['banana', 'peach']
>>> fruit[-1:]
['peach'] ← WITH A COLON: GET A LIST
```

List operations: Slicing

- Make a list:

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

- Add a single value to the end:

```
numbers.append(4)
```

- Tack another list onto the end:

```
numbers.extend([5, 6])
```

- Concatenate two lists:

```
big_numbers = [7, 8, 9, 10]
lots_of_numbers = numbers + big_numbers
```

Appendix: A recursive smudges function

```
def possible_passwords(length, smudges):
    if len(smudges) > length:
        return []
    if length == 0:
        return ['']

    passwords = []

    for i in range(0, len(smudges)):
        first = smudges[i]
        for suffix in possible_passwords(length - 1,
                                         smudges[:i] + smudges[i + 1:]):
            passwords.append(first + suffix)
        # Consider duplicates
        for suffix in possible_passwords(length - 1, smudges):
            passwords.append(first + suffix)

    return passwords
```

Set operations

- Make a set:

```
>>> cats = {'Phoebe', 'Annie'}
```

- Add a single value to the set:

```
>>> cats.add('Sylvester')
```

- Check if a value is in the set:

```
>>> 'Tweety' in cats
False
```

- Get the union of two sets:

```
>>> cats.union({'Buster', 'Fido'})
{'Buster', 'Annie', 'Phoebe', 'Fido'}
```

Set operations

Method	Operation
`a.union(b)`	a ∪ b
`a.intersection(b)`	a ∩ b
`a.difference(b)`	a - b
`a.symmetric_difference(b)`	a ∪ b - a ∩ b
`a.issubset(b)`	a ⊆ b ?
`a.isdisjoint(b)`	a ∩ b = Ø ?

```
>>> help(set)
```

File reading

```
with open('datafile.csv') as infile:
    for line in infile:
        print(line)
```

Appendix: Reading a CSV file

```
True,True,True,False,True,False
False,True,False,True,True,True
  .
  .
  .
```

Appendix: Reading a CSV file

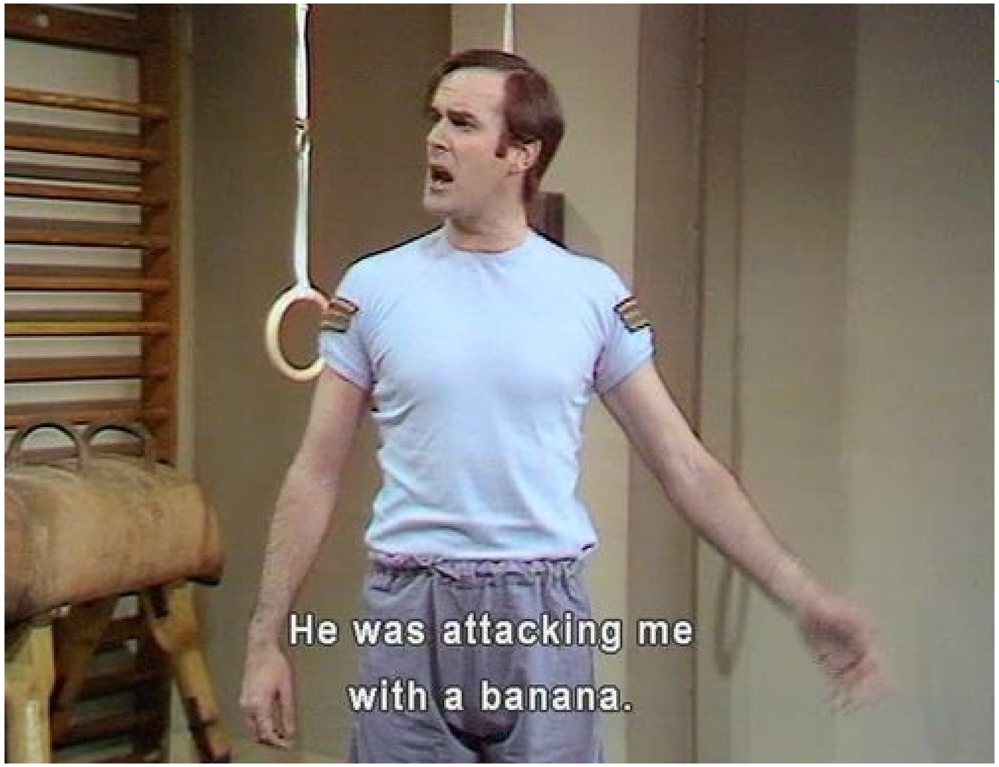
```
with open('data.csv') as infile:
    data = []
    for line in infile:
        # Strip off the final \n
        line = line[:-1]
        # line is 'True,True,False,...'
        # split(x) returns a list of
        # substrings, separated by x
        row = line.split(',')
        data.append(row)

# data is now a list of lists:
#    [['True', 'True', 'False', ...],
#     ['False', 'True', 'False', ...],
#     ...]
```

The `random` library

```
random.randint(a, b)
# integer between a and b, inclusive
random.choice(seq)
# pick one element of seq, equally likely
random.shuffle(seq)
# shuffle seq, in-place
random.sample(seq, k)
# draw k elements of seq without replacement
random.random()
# uniform float in [0, 1)
```

Appendix: How to ~~Defend Yourself Against~~ DO PROBABILISTIC SIMULATIONS WITH Fresh Fruit



GRATUITOUS MONTY PYTHON REFERENCE.

PYTHON IS NAMED AFTER THEM!

Monty Python's Flying Circus © Python (Monty) Pictures, Ltd.

Appendix: Fresh fruit

From lecture Fri 4/7: 4 mandarins, 3 grapefruits in a bag. Draw 3. P(2 grapefruits, 1 mandarin)?

```
import random

def grapefruit(num_draws):
    bag = ['mandarin', 'mandarin', 'mandarin', 'mandarin',
           'grapefruit', 'grapefruit', 'grapefruit']
    total_successes = 0
    for i in range(num_draws):
        draw = random.sample(bag, 3)
        if sorted(draw) == ['grapefruit', 'grapefruit', 'mandarin']:
            total_successes += 1
    return total_successes * 1.0 / num_draws
    # Note: * 1.0 not necessary on Python 3
```

Answer: 12/35 = 0.3428…

Running with 10M draws gave me 0.3425…

Appendix: Python classes

INHERITANCE!

```
class Mandarin(Fruit):
    def __init__(self, juiciness):
        self.juiciness = juiciness

    def peel(self, care):
        if self.juiciness - care > 9000:
            print('Squirt!')


extra_juicy = Mandarin(1000000)
extra_juicy.peel(0)
```

self IS LIKE this. YOU HAVE TO WRITE IT EXPLICITLY HERE.

BUT NO NEED TO WRITE self WHEN CALLING A METHOD.