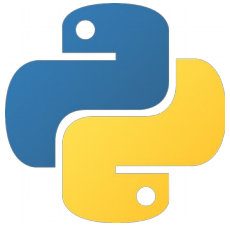
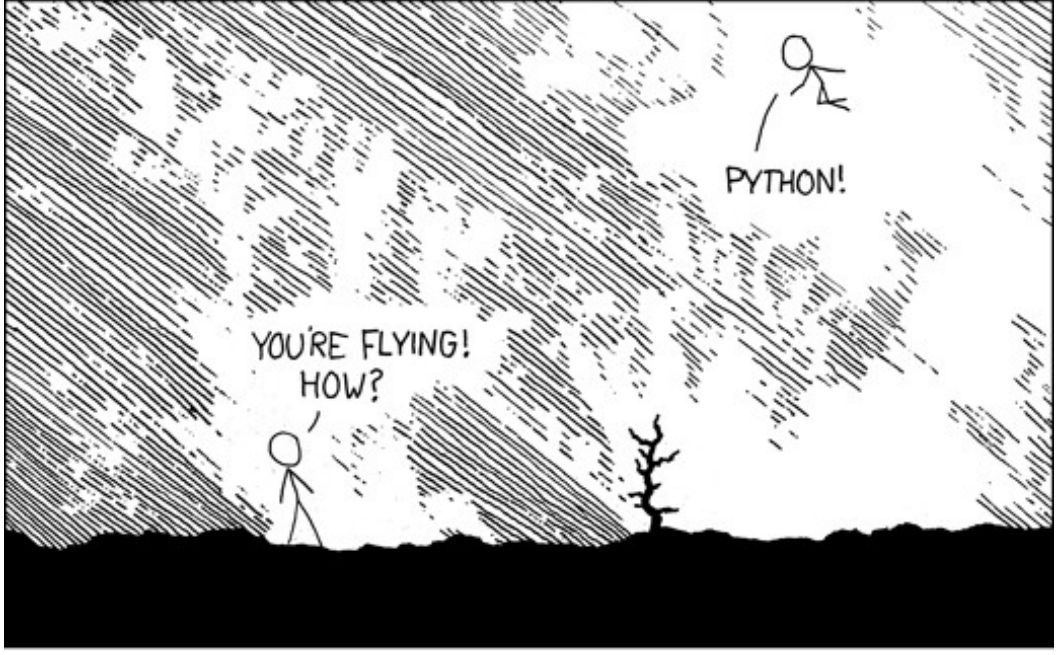


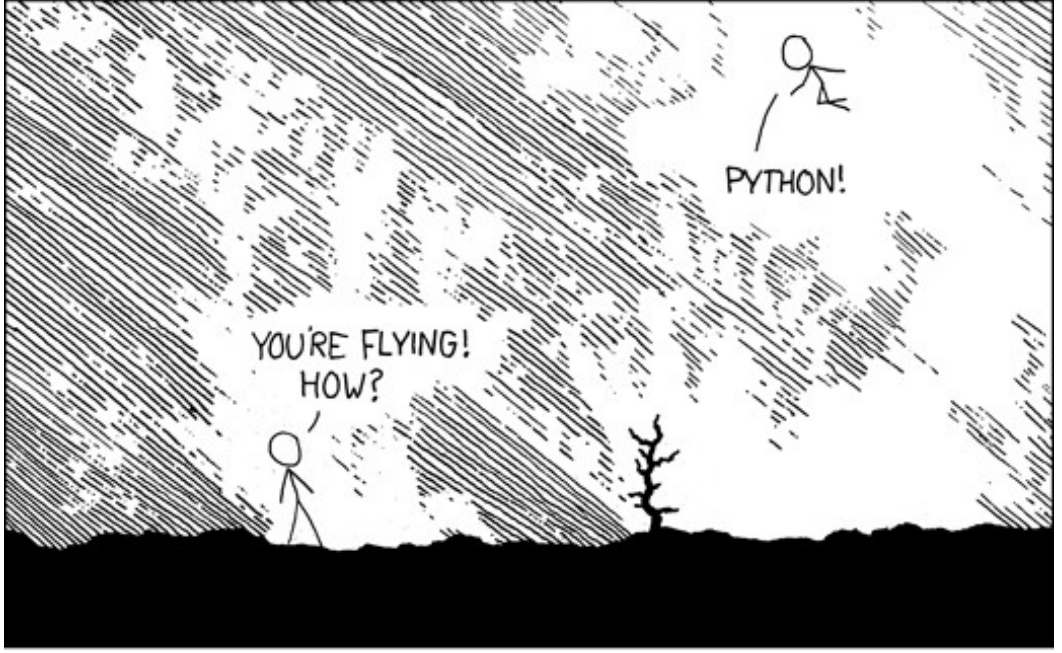
Workshop: **Introduction to Python**

Python Basics

Christian C. Luhmann
Stony Brook University

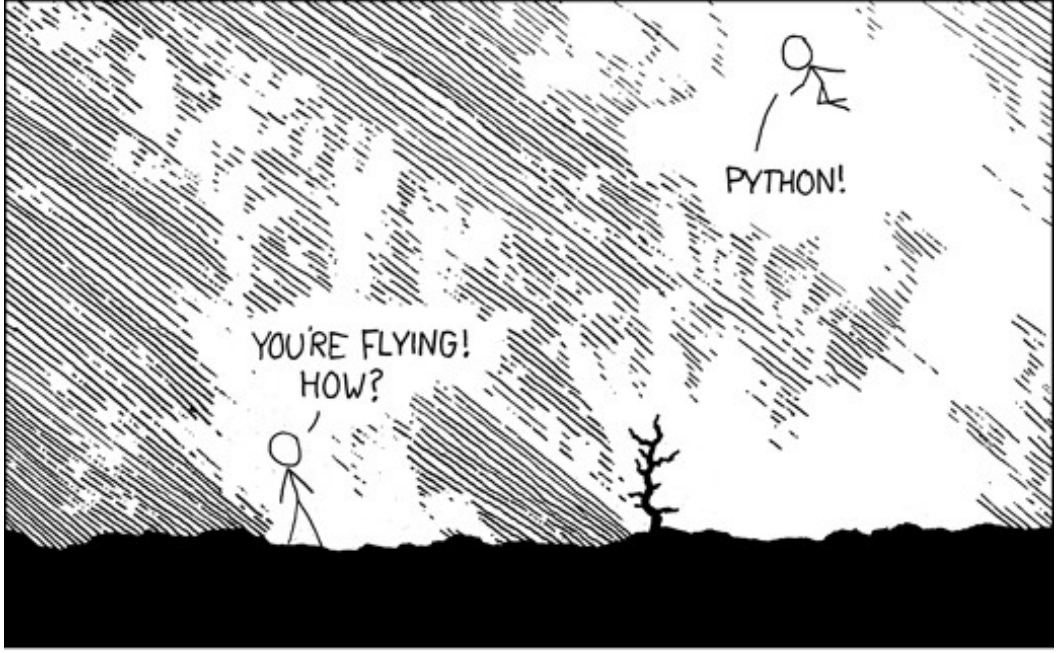


xkcd.com/353/


PYTHON!
YOU'RE FLYING!
HOW?


I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"

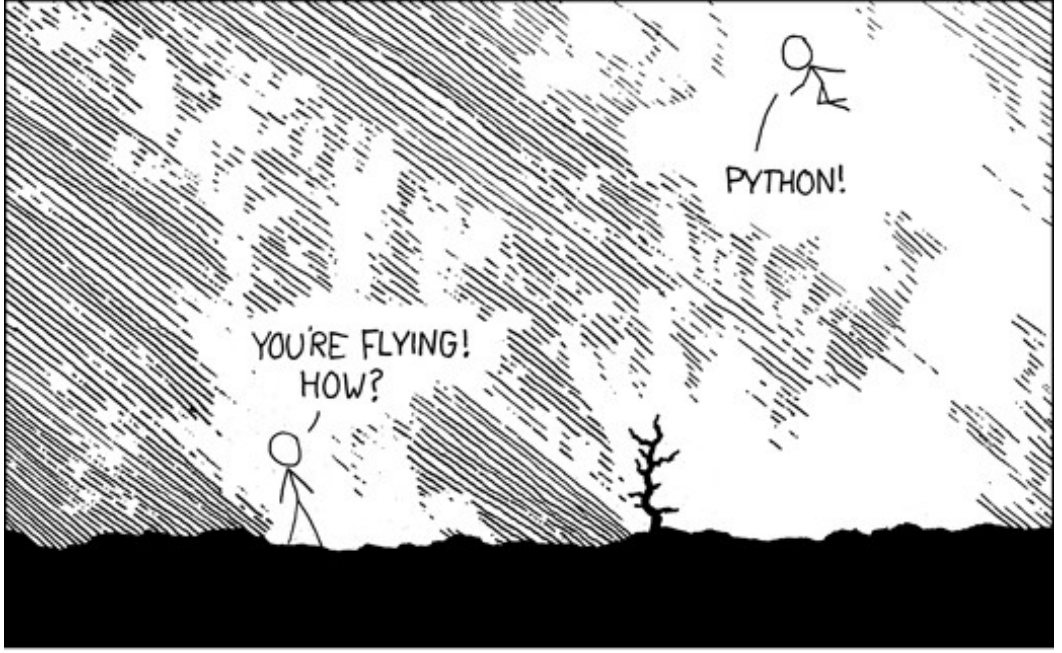
xkcd.com/353/


PYTHON!
YOU'RE FLYING!
HOW?


I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"

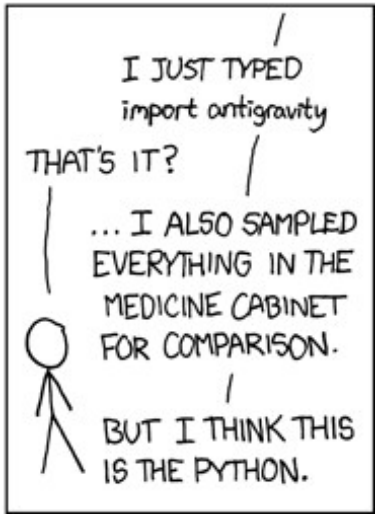

I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?

xkcd.com/353/


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.com/353/

```
>>> import this
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!
```

Python

So let’s take a closer look at the language itself

Python Language

```
>>> print('Hello world!')
Hello world!
```

Python Language

```
>>> 2 * 2
4
```

Python Language

```
>>> a = 2
>>> a * 3
6
```

Datatypes

```
>>> a = 1                  # integer
>>> b = 1.1                # float
>>> c = 'one'              # string
>>> d = [1, 2, 3]          # list
```

Lists

```
>>> my_list = [1, 3, 0, 1, 4]
>>> len(my_list)
5
>>> sum(my_list)
9
>>> my_list[0]
1
>>> my_list[-1]
4
```

Lists

```
>>> my_list = [1, 3, 0, 1, 4]
>>> my_list[0:3]
[1, 3, 0]
>>>
>>>
>>> a = ['p', 'y', 't', 'h', 'o', 'n']
>>> a[:2] + a[2:4] + a[4:]
['p', 'y', 't', 'h', 'o', 'n']
```

Dictionaries

```
>>> my_dict = {'a': 1, 'b': 2, 'c': 3}
>>> my_dict['b']
2
>>> subject_info = {'SN' : 99}
>>> subject_info['sex'] = 'M'
>>> subject_info['age'] = 21
>>> subject_info['condition'] = 'control'
```

Conditionals

```
# branch on the value of a
if a > 5:
    print(a, 'is greater than five')
else:
    print(a, 'is not greater than five')
```

Indentation

```
# branch on the value of a
if a > 5:
    print(a, 'is greater than five')
else:
    print(a, 'is not greater than five')
```

Loops

```
for item in [1, 3, 0, 1, 4]:
    print(item)
```

Loops

```
for item in range(5):
    print(item)
```

Loops

```
for item in [0, 1, 2, 3, 4]:
    print(item)
```

List Comprehensions

```
>>> my_list = [1, 3, 0, 1, 4]
>>> squares = [item * item for item in my_list]
>>> print(squares)
[1, 9, 0, 1, 16]
```

Readability counts

- Every programming language represents a tradeoff between...
  - time it takes **to write** a program (your time)
  - time it takes **to run** a program (computer's time)

- Python prioritizes **your** time

Python

xkcd.com/409/

Python

For researchers, this means...

- students are less intimidated
- collaboration is easier
- transparency is improved

Outline

1. Overview
2. Ways of using Python
3. Python basics
4. Data set overview
5. Data wrangling
6. Statistics
7. Plotting
8. Experiment creation