Programming with Python

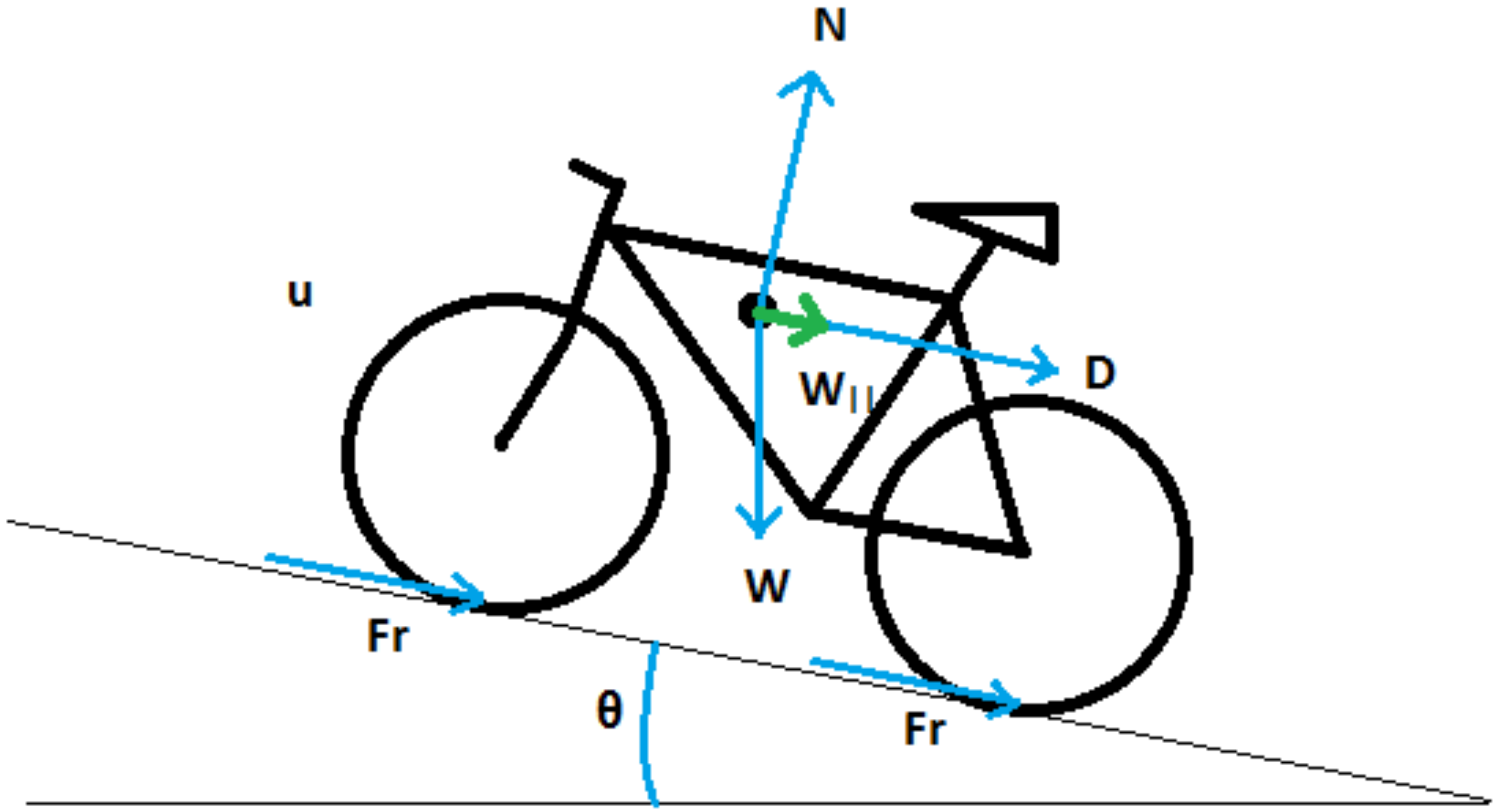
Duke UPGG Scientific Computing Bootcamp
August 12, 2019

Dan Leehr
dan.leehr@duke.edu

Dive Into Python 3
Python Programming for the absolute beginner Third Edition
Programming in Python 3 Second Edition
A Complete Introduction to the Python Language
Developer's Library
Think Python
Python Cookbook
Head First Python
A Brain-Friendly Guide
What are the best books about ... python in 2016 ?
Pro Python
Programming in Python 3
Python Programming Patterns
Python Essential Reference
Programming Computer Vision with Python
Beginning Programming with Python for Dummies
PYTHON
Fluent Python
CLEAR, CONCISE, AND EFFECTIVE PROGRAMMING
Luciano Ramalho
PYTHON CRASH COURSE
Foundations of Python Network Programming
Developer's Library
Python Essential Reference
Fourth Edition
David M. Beazley
Beginning Python Visualization
Programming Python
THE RUBY WAY
Natural Language Processing with Python
A Primer on Scientific Programming with Python
Internet Programming with Python
Introduction to Machine Learning with Python
Parallel Programming with Python
Python Standard Library
O'REILLY

What book should I read?

How many books about riding a bike did you read?



“You can be a scientist in the science of bike ride mechanics and it still won’t help you one bit to do the actual thing.”

http://twonontechies.com/bicycles-can-help-you-learn-programming/

Why Python?

- We have to use *something*
- It’s free, well-documented, and runs everywhere
- Large community among scientists
- Relatively easy to pick up, but programming is **hard**!

Goals

- Write and run programs in Python
- Understand basic data types and functions
- Work with files and libraries
- Know where to look for more help

I know, I'll use ***Python****!*

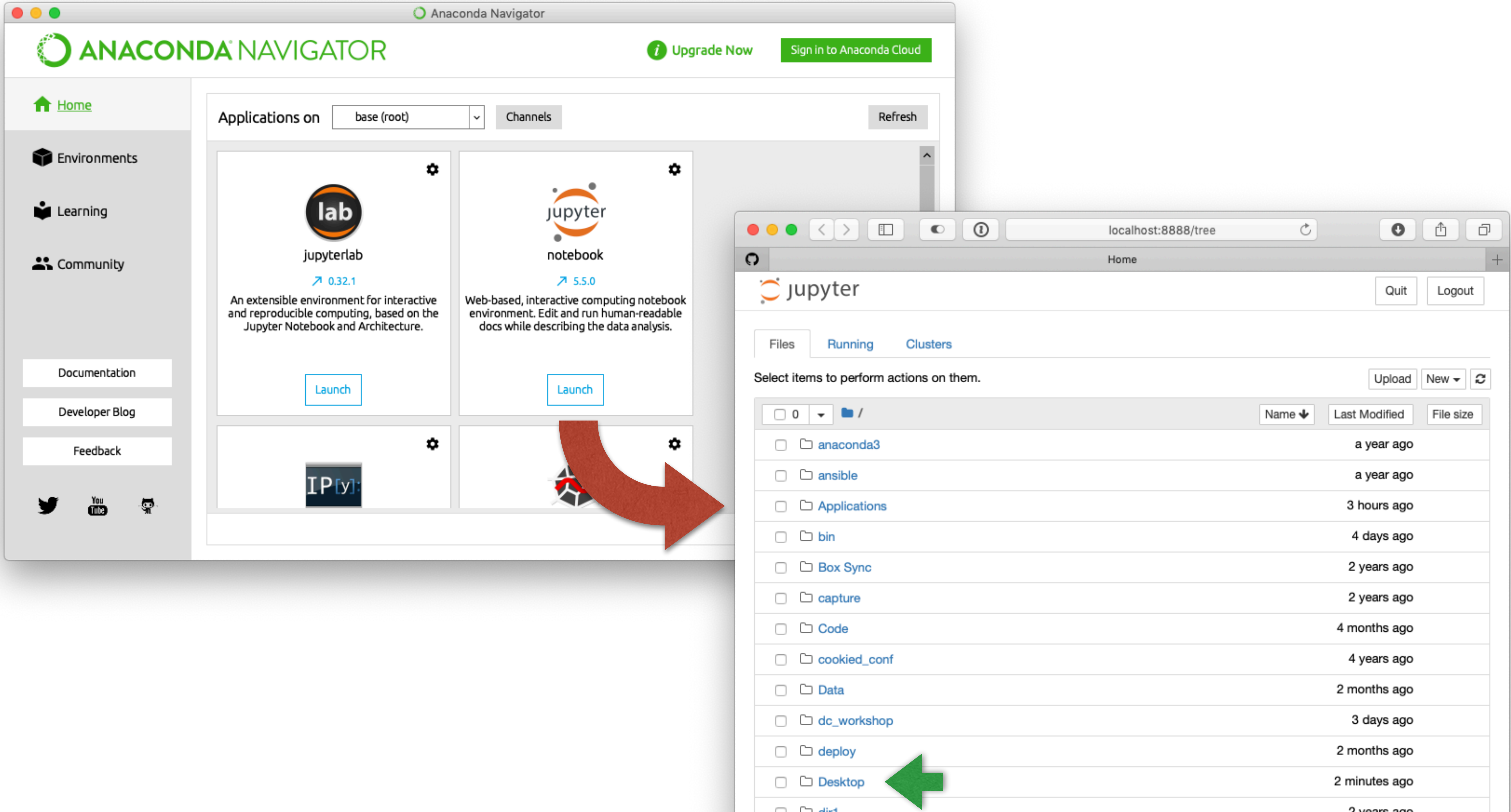
Download

- Download the **python-fasta.zip** file from the course website - **Syllabus**.
- Unzip it and place on your Desktop:

```
python-fasta/
  ae.fa
  ls_orchid.fasta
```

1. Open **Anaconda Navigator** (installed with Anaconda)

2. Click to launch **Jupyter Notebook**



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Select items to perform actions on them.

Name	Last Modified
anaconda3	a year ago
ansible	a year ago
Applications	3 hours ago
bin	4 days ago
Box Sync	2 years ago
capture	2 years ago
Code	4 months ago
cookied_conf	4 years ago
Data	2 months ago
dc_workshop	3 days ago
deploy	2 months ago
Desktop	2 minutes ago

Begin Jupyter Notebook

Data Types

- Numeric:
    - Integer: 1, 76, 400
    - Float: -1.2, 0.5, 3.1415926 (Use a decimal point)
    - Boolean: True, False
- Text:
    - Strings: ‘ACTGACAG' (Wrap in quotes)

Strings

- Strings can be created with quotes or double quotes:

```
name = 'Daniel'
```

- Access individual letters as strings with [] (starting at 0)

```
name[0] # D
name[1] # a
```

- Check if a letter exists in a string

```
'a' in name # True
'a' not in name # False
```

Variables

- Assign variables with equals

```
x = 3
```

- Access variables by name

```
print x # 3
```

- Variables work like sticky notes, they’re just a label on top of a value

What do we know?

- Our sequence is a string, in **seq10**
- Strings are sequences of characters, each at a numbered position (starting from 0)
- We can extract characters as strings with square brackets **[ ]**
- We can combine strings together with **+**

Exercise: Reverse

- Write some code that **reverses** the sequence in seq.
- It should

1. Create an empty string variable **rev**

   ```
   rev = ''
   ```

2. Loop over the items in **seq**, adding these to rev in reversed order

3. Print the contents of **rev**

Loops

- Write a loop with **for** item **in** collection:

  ```
  for letter in word:
      print letter
  ```

- Always put a colon at the end of the line, indented lines are run for every item in the collection

Complementing

- We can loop over all the bases in a sequence
- Each base has a complement that we should substitute:
- We can use a **Dictionary** to store this mapping.

A	→	T
C	→	G
T	→	A
G	→	C

Dictionaries and Lists

- Create dicts with {}, lists with []

```
nucs = {'A': 5, 'C': 4, 'T': 8}
counts = [5,4,8]
```

- Both accessed with [] - dicts by key, lists by index

```
nucs['A'] # 5
counts[0] # 5

nucs['A'] = 3 # now 3
counts[0] = 3 # now 3
```

GC-content percentage

- Calculated as **(G + C) / (A + T + G + C)**
- Create a GC count variable and an ATGC count variable
    - Loop over each base in the sequence
        - If G, add 1 to GC count
        - If C add 1 to GC count
        - For everything, add 1 to ATGC count

Conditionals

```
# Test c1 for True or False
if c1:
    print "c1 was True"
# c1 was False, check c2
elif c2:
    print "c1 False but c2 True"
# All checks False
else:
    print "Both False"
```

Exercise: Functions

```
bases = 'adenine cytosine guanine thymine'
```

Write some code that:

- Makes a **list** of these bases from the string
- **Uppercases** the names (e.g. ['ADENINE', ...])
- **Reverse**s the order (e.g. ['THYMINE',...])

Hint: Use `help(str)` and `help(list)` to see what functions are available for strings and lists

Bonus: Write a for loop to print the first letter of each (e.g. A, C, ...)

Exercise

- Strings can be reversed with this special slicing notation: [::-1]

  ```
  s = 'abc'
  r = s[::-1]
  print(r)

  cba
  ```

- Update `reverse()` function to use `[::-1]` instead of a loop.

- Do we need to do anything to `complement()`?
  What about `reverse_complement()`?

Functions

- Calling functions: `length = len('abc')`
- Defining functions:

```
def double(x):
    return x * 2
```

- Composing functions:

```
def reverse_complement(seq):
    return reverse(complement(seq))
```

- Avoid using global variables in functions

Exercise

- Write a function, `read_fasta(filename)` that:
    - Takes 1 argument: `filename`
    - Reads the file line-by-line
    - Strips/combines the lines into one long line
    - Skips the line if it contains a `>`
- Hint: `if not 'i' in ‘team':`

Reading files

- Open a file with the `open()` function:

```
f = open('ae.fa')
```

- Loop over lines, and `strip()` each one

```
for line in f:
    print line.strip()
```

- Close with `f.close()`

Scripts

- Put code in a file, give it the .py extension
- Read command line-arguments from sys.argv:

```
import sys
print sys.argv[0]
print sys.argv[1]

$ python script.py hello
script.py
hello
```

- Check the length of `sys.argv` to be helpful!