“Data science” has completely changed our world
Course goals

- Establish a broad background in Python programming
- Prepare you for the inevitable coding interview
- Survey popular tools in academia/industry for data analysis and exploration
- Learn how to read documentation and quickly get new tools up and running
- Learn basic distributed computing frameworks

These tools will be obsolete some day...

...but not your ability to learn new frameworks and solve problems!
Course structure

Unit 1: Introduction to Python
   Data types, functions, Jupyter, classes, objects, functional programming

Unit 2: Numerical Computing and Data Visualization
   numpy, scipy, matplotlib

Unit 3: Dealing with structured data
   regular expressions, retrieving web data, SQL, Python pandas, APIs

Unit 4: Big data and parallel programming
   Basics of the UNIX command line, ssh, Hadoop, Spark, TensorFlow

Schedule (tentative) and other information available on course webpage:
   www.umich.edu/~klevin/teaching/Winter2019/STATS507
Prerequisites

I assume that you have *some* background in programming and statistics.

Come speak to me if:
- this is your first programming course
- you have never taken a probability or statistics course

This course is probably not for you if:
- you have no programming background
Course information

**Instructor:** Keith Levin
- Email: klevin@umich.edu
- Office: 272 WH
- OH: TBA or by appointment

**GSI:** Roger Fan
- Email: rogerfan@umich.edu
- OH: TBA

**Textbook:** None
- Readings posted to the website

**Grading:** 10-12 HWs,
- Weighted approximately equally
- No midterm, no final
- No class project
- Late days (see syllabus)

See syllabus on Canvas or at umich.edu/~klevin/teaching/Winter2019/STATS507/syllabus.pdf
A Note on Enrollment and the Waitlist

This is an immensely popular course...
... which is excellent, but it means that there are... a lot of you.

Waitlist:

The waitlist is handled by the statistics office. I have no control of it!
Please do not email me asking for overrides. I cannot grant them.

Please direct all enrollment questions to the statistics office: stat-um@umich.edu
Before we continue...

Readings:
For the first half of the course, readings will be given in both
  Allen B. Downey’s *Think Python* and
  Charles Severance’s *Python for Everybody*
You can do the readings out of either one, whichever you prefer!

Later, we’ll make exclusive use of Severance
A Note on Readings

I will post weekly readings throughout the course

I would prefer if you do the readings before lecture...
   ...but I recognize this is not always possible...
   ...and if you find that you learn better seeing lecture first, then that’s fine.

Some of the readings consist of technical documentation
   It is a goal of this course to get you comfortable reading docs!
   Read and understand what you can, google terms you don’t understand…
   ...and it’s okay to set things aside to come back to later!
Policies

Don’t plagiarize!
● You may discuss homeworks with your fellow students...
● ...but you must submit your own work
● Disclose in your homework whom (if anyone) you worked with

Late homeworks are not allowed!
● Instead, we have “late days”, of which you get 7
● One late day extends HW deadline by 24 hours
● **Note:** homework deadlines may not be extended beyond 11:59pm on the scheduled day of the final (Thursday, May 2nd).

Refer to the syllabus for details.
Survey time!

1. Raise your hand if you have used Python before.
2. Raise your hand if you have used jupyter/iPython in the past.
3. Raise your hand if you have used the UNIX/Linux command line.
4. Raise your hand if you have used the Python \texttt{matplotlib} package.
5. Raise your hand if you prefer Canvas over a course webpage.
Things to do very soon:

Pick an editor/IDE for python
  or just use a text editor, or just write directly in jupyter

Familiarize yourself with jupyter:
  https://jupyter.readthedocs.io/en/latest/content-quickstart.html

Get a flux/fladoop username
  Fill out form here: http://arc-ts.umich.edu/hpcform/
  List me (Keith Levin, klevin@umich.edu) as your “advisor”
  Include a note that you are in STATS507 and need access to Fladoop

Note: we will use only Python 3 in this course. Check that you have Python 3 installed on your machine and that it is running properly.
Other things

HW1 is posted to canvas and the website. Get started now!

If you run into trouble, come to office hours for help
  ● But also please post to the discussion board on Canvas
  ● If you’re having trouble, at least one of your classmates is, too
  ● You’ll learn more by explaining things to each other than by reading stackexchange posts!

Email policy:
  I will not provide tech support over email!
  If you are having trouble, post to the discussion board and/or come to OHs!
STATS 507
Data Analysis in Python

Lecture 1: Introduction to Python
Python: Overview

Python is a **dynamically typed, interpreted** programming language
Created by Guido van Rossum in 1991
Maintained by the Python Software Foundation

Design philosophy: simple, readable code

Python syntax differs from R, Java, C/C++, MATLAB
whitespace delimited
limited use of brackets, semicolons, etc
Python: Overview

Python is a dynamically typed, interpreted programming language. Created by Guido van Rossum in 1991 and maintained by the Python Software Foundation. Design philosophy: simple, readable code. Python syntax differs from R, Java, C/C++, MATLAB: whitespace delimited, limited use of brackets, semicolons, etc. In many languages, when you declare a variable, you must specify the variable’s type (e.g., int, double, Boolean, string). Python does not require this.
Python: Overview

Python is a **dynamically typed**, **interpreted** programming language. Created by Guido van Rossum in 1991. Maintained by the Python Software Foundation.

Design philosophy: simple, readable code.

Python syntax differs from R, Java, C/C++, MATLAB:
- Whitespace delimited
- Limited use of brackets, semicolons, etc.

Compiled vs interpreted languages: compiled languages are (generally) faster than interpreted languages, typically at the cost of being more complicated.

Some languages (e.g., C/C++ and Java) are **compiled**: we write code, from which we get a runnable program via **compilation**. In contrast, Python is **interpreted**: A program, called the **interpreter**, runs our code directly, line by line.
Running Python

Several options for running Python on your computer

- Python interpreter
  - Jupyter: https://jupyter.org/
  - PythonAnywhere: https://www.pythonanywhere.com/
  - Suggestions from Allen Downey: http://www.allendowney.com/wp/books/think-python-2e/

Your homeworks must be handed in as Jupyter notebooks

But you should also be comfortable with the interpreter and running Python on the command line


Note: Jupyter recommends Anaconda: https://www.anaconda.com/
  I mildly recommend against Anaconda, but it’s your choice

Image credit: https://www.python.org/community/logos/
Python Interpreter on the Command Line

```
keith@Steinhaus:~/demo$ python3
Python 3.6.3 (default, Oct  4 2017, 06:09:05)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
keith@Steinhaus:~/demo$ python
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016, 23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>>
```
Python Interpreter on the Command Line

keith@Steinhaus:~/demo$ python3
Python 3.6.3 (default, Oct 4 2017, 06:09:05)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
keith@Steinhaus:~/demo$ python
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016, 23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>>

The prompt indicates that the system is waiting for your input.

I have Python 2 running inside Anaconda, by default.
Python Interpreter on the Command Line

Write Python commands (code) at the prompt
Python in Jupyter

Creates “notebook files” for running Julia, Python and R

Example notebook:

https://nbviewer.jupyter.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb

Clean, well-organized presentation of code, text and images, in one document


Good tutorials:

https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook
Running Jupyter

Jupyter provides some information about its startup process, and then...
Running Jupyter

```
keith@Steinhaus:~/demo$ jupyter notebook
[I 17:11:41.129 NotebookApp] Serving notebooks from local directory:/Users/keith/Dropbox/Academe/Teaching/STATS507/Lecs/L1_AdminIntro
[I 17:11:41.129 NotebookApp] 0 active kernels
[I 17:11:41.129 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/?token=452d6d4b227f306f5bb57e72f5d4722fcbadf47d1d794441
[I 17:11:41.129 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[C 17:11:41.132 NotebookApp]
```

...Jupyter opens a browser window in which you can launch a new notebook or open an existing one.
Creates a new notebook file running Python 2.

Creates a new notebook file running Python 3.

Creates a new notebook file running R.

Note: Jupyter can also run other programming languages, such as Julia, if they are installed.
Running Python 3

Notebook doesn’t have a title, yet.
I'll leave it to you to learn about the other features by reading the documentation. For now, the green-highlighted box is most important. That’s where we write Python code.
Write code in the highlighted box, then press shift+enter to run the code in that box...
Write code in the highlighted box, then press shift+enter to run the code in that box...

Note: can also run code by clicking the “run cell” button, but the shift+enter shortcut is a lot easier.
Our first function: `print`

If you haven’t already guessed, `print` takes a Python `string` and prints it. Of course, “print” here means to display a string, not literally print it on a printer!

Note: if you know Python 2, you’ll notice that `print` is a bit different in Python 3. That is because in Python 2, `print` was a `statement`, whereas in Python 3, `print` is a `function`. 

Print displays whatever is inside the quotation marks.

Can also use double quotes
Arithmetic in Python

Use `+` to add numbers.

Use `*` to multiply.

Order of operations is just like you learned in elementary school.

Python is weird in that it uses `**` for exponentiation instead of the more common `^`.

`/` for division.

`//` performs division but rounds down.

`%` is modulo. `x % y` is remainder when `x` is divided by `y`. 

1. $1+2$ = 3
2. $2*3$ = 6
3. $2*3 - 1$ = 5
4. $2**7$ = 128
Data Types

Programs work with **values**, which come with different **types**

Examples:

The value **42** is an **integer**
The value **2.71828** is a **floating point number** (i.e., decimal number)
The value “**bird**” is a **string** (i.e., a **string of characters**)

Variable’s type determines what operations we can and can’t perform
e.g., **2*3** makes sense, but what is ‘**cat’ * ‘dog’’?
(We’ll come back to this in more detail in a few slides)
Variables in Python

**Variable** is a name that refers to a value

Assign a value to a variable via **variable assignment**

```
1 mystring = 'Die Welt ist alles was der Fall ist.'
2 approx_pi = 3.141592
3 number_of_planets = 9
```

Assign values to three variables

```
1 mystring
'Die Welt ist alles was der Fall ist.'
```

Change the value of `number_of_planets` via another assignment statement.

```
1 number_of_planets = 8
2 number_of_planets
8
```
Variables in Python

**Variable** is a name that refers to a value.

Assign a value to a variable via variable assignment.

```python
1 mystring = 'Die Welt ist alles was der Fall ist.'
2 approx_pi = 3.141592
3 number_of_planets = 9
```

```python
1 mystring
'Die Welt ist alles was der Fall ist.'
```

```python
1 number_of_planets
9
```

```python
1 number_of_planets = 8
2 number_of_planets
8
```

**Note:** unlike some languages (e.g., C/C++ and Java), you don’t need to tell Python the type of a variable when you declare it. Instead, Python figures out the type of a variable automatically. Python uses what is called **duck typing**, which we will return to in a few lectures.
Variables in Python

**Variable** is a name that refers to a value

Assign a value to a variable via **variable assignment**

```python
1 mystring = 'Die Welt ist alles was der Fall ist.'
2 approx_pi = 3.141592
3 number_of_planets = 9

1 mystring
'Die Welt ist alles was der Fall ist.'

1 number_of_planets
9

1 number_of_planets = 8
2 number_of_planets
8
```

**Note:** unlike some languages (e.g., C/C++ and Java), you don’t need to tell Python the type of a variable when you declare it. Instead, Python figures out the type of a variable automatically. Python uses what is called **duck typing**, which we will return to in a few lectures.

Python variable names can be arbitrarily long, and may contain any letters, numbers and underscore (_), but may not start with a number. Variables can have any name, except for the Python 3 reserved keywords:

None continue for lambda try True def from nonlocal while and del
global not with as elif if or yield
assert else import pass break except
in raise
Variables in Python

Sometimes we do need to know the type of a variable

Python `type()` function does this for us

```python
mystring = 'Die Welt ist alles was der Fall ist.'
approx_pi = 3.141592
number_of_planets = 9

type(mystring)
```

```
str
```

```python
type(approx_pi)
```

```
float
```

```python
type(number_of_planets)
```

```
int
```

Recall that `type` is one of the Python reserved words. Syntax highlighting shows it as green, indicating that it is a special word in Python.
Variables in Python

We can (sometimes) change the type of a Python variable

Convert a float to an int:

```python
1 approx_pi = 3.141592
2 type(approx_pi)
```

```
float
```

```python
1 pi_int = int(approx_pi)
2 type(pi_int)
```

```
int
```

Convert a string to an int:

```python
1 int_from_str = int('8675309')
2 type(int_from_str)
```

```
int
```

```python
1 int_from_str
```

```
8675309
```
Variables in Python

We can (sometimes) change the type of a Python variable. Changing a variable to a different type is often called **casting** a variable to that type.

Convert a float to an int:

```python
approx_pi = 3.141592
```

```python
type(approx_pi)
```

```
float
```

```python
pi_int = int(approx_pi)
```

```python
type(pi_int)
```

```
int
```

Convert a string to an int:

```python
int_from_str = int('8675309')
```

```python
type(int_from_str)
```

```int
```

Test your understanding: what should be the value of `float_from_int`?
Variables in Python

We can (sometimes) change the type of a Python variable

Convert a float to an int:

```python
1 approx_pi = 3.141592
2 type(approx_pi)
```

float

```python
1 pi_int = int(approx_pi)
2 type(pi_int)
```

int

Convert a string to an int:

```python
1 int_from_str = int('8675309')
2 type(int_from_str)
```

int

```python
1 int_from_str
2 8675309
```

Note: changing a variable to a different type is often called **casting** a variable to that type.

Test your understanding: what should be the value of `float_from_int`?
Variables in Python

We can (sometimes) change the type of a Python variable

But if we try to cast to a type that doesn’t make sense...

```python
1 goat_int = int('goat')
```

```
ValueError
Traceback (most recent call last)
<ipython-input-72-6ee721a55259> in <module>()
----> 1 goat_int = int('goat')

ValueError: invalid literal for int() with base 10: 'goat'
```

`ValueError` signifies that the type of a variable is okay, but its value doesn’t make sense for the operation that we are asking for.

https://docs.python.org/3/library/exceptions.html#ValueError
Variables in Python

Variables must be declared (i.e., must have a value) before we evaluate them.

```
answer = 2*does_not_exist
```

```
NameError
<ipython-input-78-7576ff000ce0> in <module>()
----> 1 answer = 2*does_not_exist

NameError: name 'does_not_exist' is not defined
```

NameError signifies that Python can’t find anything (variable, function, etc) matching a given name. [https://docs.python.org/3/library/exceptions.html#NameError](https://docs.python.org/3/library/exceptions.html#NameError)
String Operations

Try to multiply two strings and Python throws an error.

```
1  'one' * 'two'
```

```
Traceback (most recent call last)
<ipython-input-25-168e5aba40b3> in <module>()
----> 1 'one' * 'two'

TypeError: can't multiply sequence by non-int of type 'str'
```

Python uses + to mean **string concatenation**, and defines multiplication of a string by a scalar in the analogous way.

```
1  'cat' + 'dog'

'catdog'
```

```
1  'goat'*3

'goatgoatgoat'
```

**TypeError** signifies that one or more variables doesn’t make sense for the operation you are trying to perform. [https://docs.python.org/3/library/exceptions.html#TypeError](https://docs.python.org/3/library/exceptions.html#TypeError)
Comments in Python

Comments provide a way to document your code
Good for when other people have to read your code
But also good for you!

```python
# This is a comment.
# Python doesn't try to run code that is commented out.
euler = 2.71828  # Euler's number

# Triple quotes let you write a multi-line comment like this one. Everything between the first triple-quote and the second one will be ignored by Python when you run your program

print(euler)
```

2.71828

Comments explain to a reader (whether you or someone else) what your code is meant to do, which is not always obvious from reading the code itself!
A parting note for the day...

Homework:
Start your homework early!
If you run into technical issues, you’ll want to have time to come get help!

A note on pace and difficulty
I aim to teach Python from scratch in this course, but…
...time spent on Python is time not spent on the stuff you’re really here for
So, I expect that you are willing to work hard to keep up

If I am moving too fast, or you don’t understand something, come speak to me promptly!