Lecture 1

- Introduction to Data Science
- Getting Started with Git and Python Demonstrations
- Introduction to Python
- Random Experiments and Simulations

Instructor: Shreya Saxena

Assistant Professor @ ECE, UF

- Office: NEB 433
- Email: shreya.saxena@ece.ufl.edu

About Me

- My pronous are she/her/hers

- PhD in Electrical and Computer Engineering @ MIT

- Postdoc in the Zuckerman Institute for Mind, Brain, Behavior @ Columbia University

- Research interests: Neuro-AI (Neuroscience meets Artificial Intelligence)

- I have lived in India, Switzerland, and the U.S. for approximately 1/3 of my life each.

- I speak English, French, Hindi, and a little Spanish.

Why should you care about Data Science?

According to **Glassdoor**'s report 50 Best Jobs in America for 2022:

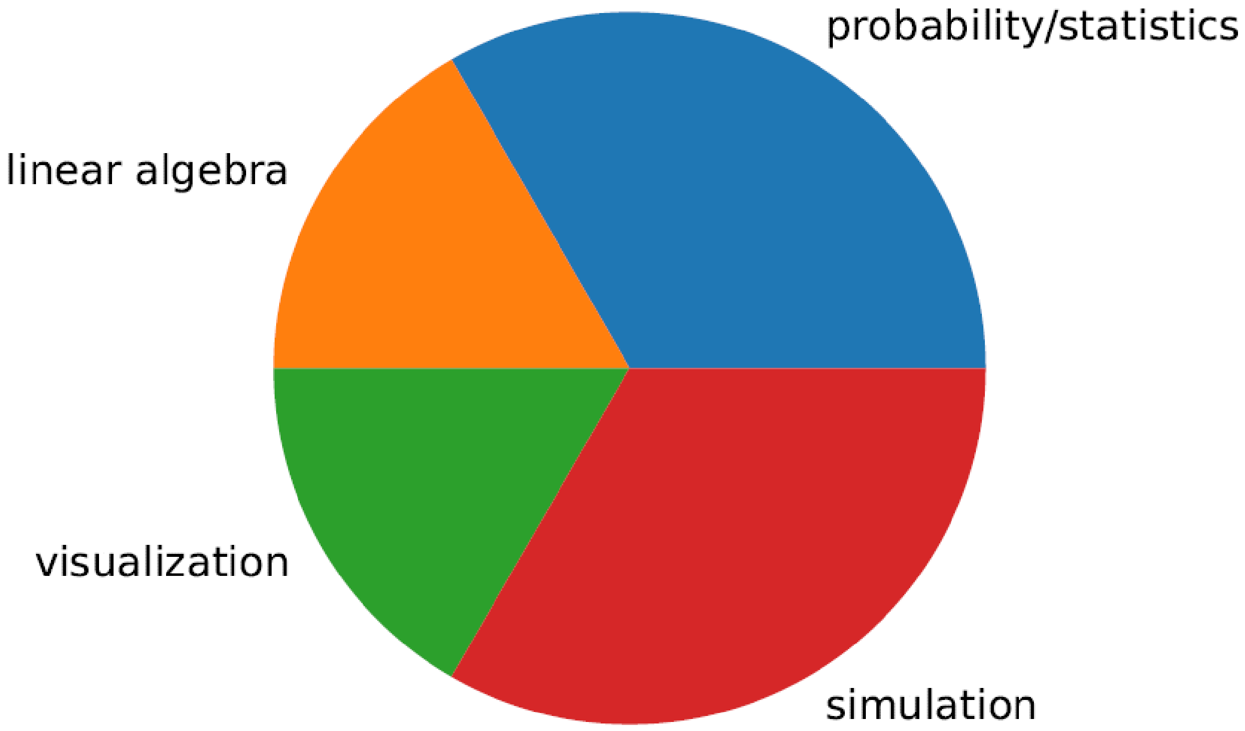
Data Scientist ranks #3 Job Satisfaction: 4.1/5 Median Base Salary: \$120,000

Why Data Science is Still a Top Job

What tools will we use to perform data science?

In [1]:
```
from IPython.display import Image
Image('figures/tools.png', width=700)
```

```
Out[1]:
```

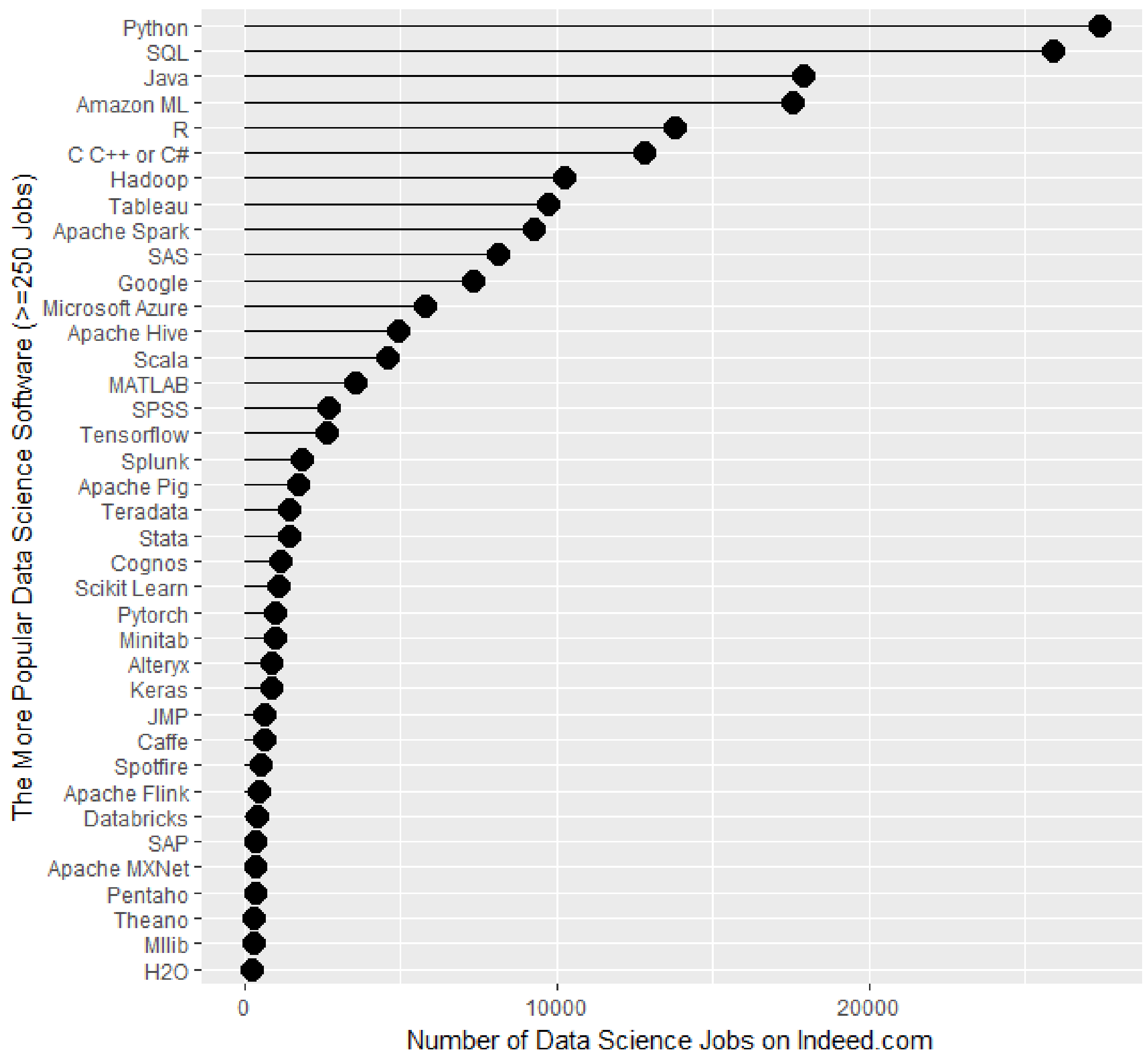


EEL 3850 Data Science for ECE

Course Description: (4 credits) Analysis, processing, simulation, and reasoning of data. Includes data conditioning and plotting, linear algebra, statistical methods, probability, simulation, and experimental design.

- **This course relies on both programming and math!**
- We will use **Python** almost every class, for class assignments, **and for exams**

Why Python?



Programming in Python

- You are **not** expected to already know Python
- However, you do need good basic programming skills (in any language) to do well in this course

What Python proficiency to expect to learn in this class?

I will **not** teach you **everything** you need to know about Python during class; instead, I will teach you how to use it for visualization, data processing, simulation, etc.

- I **will** provide you with lots of resources and help to get going in Python
- The teaching team **will** be available during office hours to help you with Python issues
- You should expect to be at an intermediate proficiency by the end of the course, assuming you put in the work

Analytical work

I will also be doing demonstrations and solving analytical exercises using probability and statistics concepts, which will require you to be proficient in Calculus.

- I have uploaded to GitHub (Lecture 0) a set of notes with review material up to Calculus II (equivalent to MAC 2312).
- No prior knowledge of probability or statistics is needed

Pre-requisite: MAC 2312 (Calculus 2) and EEL 3134 (Programming 1). Students are expected to bring a portable computer to class meetings.

So... what is Data Science?

Data science is a multidisciplinary approach to extracting actionable insights from the large and ever-increasing volumes of data collected and created by today's organizations. Data science encompasses preparing data for analysis and processing, performing advanced data analysis, and presenting the results to reveal patterns and enable stakeholders to draw informed conclusions.

Data preparation can involve cleansing, aggregating, and manipulating it to be ready for specific types of processing. Analysis requires the development and use of algorithms, analytics and AI models. It's driven by software that combs through data to find patterns within to transform these patterns into predictions that support business decision-making. The accuracy of these predictions must be validated through scientifically designed tests and experiments. And the results should be shared through the skillful use of data visualization tools that make it possible for anyone to see the patterns and understand trends.

As a result, data scientists require computer science and pure science skills beyond those of a typical data analyst. A data scientist must be able to do the following:

- Apply mathematics, statistics, and the scientific method
- Use a wide range of tools and techniques for evaluating and preparing data
- Extract insights from data using predictive analytics and artificial intelligence (AI), including machine learning and deep learning models
- Write applications that automate data processing and calculations
- Tell—and illustrate—stories that clearly convey the meaning of results to decision-makers and stakeholders at every level of technical knowledge and understanding
- Explain how these results can be used to solve business problems

This combination of skills is rare, and it's no surprise that data scientists are currently in high demand

The Data Science Lifecycle

The data science lifecycle—also called the data science pipeline—includes anywhere from five to sixteen (depending on whom you ask) overlapping, continuing processes. The processes common to just about everyone’s definition of the lifecycle include the following:

1. **Capture**: This is the gathering of raw structured and unstructured data from all relevant sources via just about any method—from manual entry and web scraping to capturing data from systems and devices in real time.
2. **Prepare and maintain**: This involves putting the raw data into a consistent format for analytics or machine learning or deep learning models. This can include everything from cleansing, deduplicating, and reformatting the data, to using ETL (extract, transform, load) or other data integration technologies to combine the data into a data warehouse, data lake, or other unified store for analysis.
3. **Preprocess or process**: Here, data scientists examine biases, patterns, ranges, and distributions of values within the data to determine the data's suitability for use with predictive analytics, machine learning, and/or deep learning algorithms (or other analytical methods).
4. **Analyze**: This is where the discovery happens—where data scientists perform statistical analysis, predictive analytics, regression, machine learning and deep learning algorithms, and more to extract insights from the prepared data.
5. **Communicate**: Finally, the insights are presented as reports, charts, and other data visualizations that make the insights—and their impact on the business—easier for decision-makers to understand. A data science programming language such as Python (see below) includes components for generating visualizations such as Jupyter Notebooks; alternatively, data scientists can use dedicated visualization tools.

Some examples of data science problems we have studied in previous years:

- Do gun laws reduce firearms mortality?
- If we look at the effect of state laws on firearms mortalities, what other factors might be responsible for any observed differences? Do urban and rural states have different firearms mortalities? How about richer or poorer states?
- Do males score higher on standardized high school math and science tests than females?
- Do first pregnancies last longer than other pregnancies?
- How should we measure the effectiveness of medical tests, including risks with false identification and missed detection of disease?

- Has the temperature been increasing over the last decades?

How does a typical lecture looks like?

- We will use Jupyter Notebooks such as this one
- I will publish the class notes **before** class
- I will do live coding, you can follow along with your laptops
- I will share the notebook with edits after class, and also push its pdf
- I will use my iPad as a virtual whiteboard
- I will share the handwritten whiteboard notes after class

All lectures will be pushed to GitHub

Course Homepage

In this course we will use two main organizations:

1. Canvas page: announcements, grades, send/receive emails, participation assignments through discussion boards, question and answering on assignment issues through discussion boards.

2. GitHub Organization: I will post all lecture notes in this private organization. You will complete all assignments in a private repository and send its URL to Canvas as the assignment submission.

    - In order to receive an invitation to join the organization, join the GitHub Classroom first by accepting and creating your repository for Short Assignment 0 repository.
    - **Clone the "Lectures" repository to your local machine and pull from that repository before class**
    - Be sure to download Git
    - Complete one (or a few) introductory tutorials:
        - Git bootcamp: https://help.github.com/categories/bootcamp/
        - Tutorials: https://www.atlassian.com/git/tutorials/
        - Interactive Introduction: https://try.github.io/

Intro to GitHub

Basic git commands:

- `git status`
- `git pull`
- `git add`
- `git commit`

- `git push`

Git Demonstration

Install and download Git and GitHub Desktop.

1. How to `clone` a repository

You can use **Git Bash** to clone a repo or use GitHub Desktop. **I will demonstrate how to do it using Git Bash.** But all can also be easily performed using the interface GitHub Desktop. (Download and install GitHub Desktop)

For example, let's create "Short Assignment 0" repository and clone it.

2. Getting the latest edits from a repository - use `git pull`

To `pull` from a repository, simply call `git pull` using Git Bash.

3. How to manage files within a repo

The 3 most used Git commands are: `git pull`, `git add`, `git commit` and `git push`. You can call these commands directly on the **Git Bash** console within the cloned repository on your machine.

This should be sufficient to get you started with Git and GitHub in this course. To learn more, watch the tutorials below:

- Git bootcamp: https://help.github.com/categories/bootcamp/
- Tutorials: https://www.atlassian.com/git/tutorials/
- Interactive Introduction: https://try.github.io/

The Curious git is also a great resource.

Course Objectives (as time allows)

Upon completion of this course, the student will be able to:

1. Implement, debug, and deploy Python code
2. Generate visualizations to expose meaning in data
3. Generate and understand the meaning and uses of summary statistics of data
4. Model random phenomena using random variables
5. Generate random variables with specified densities or distributions
6. Conduct hypothesis tests using simulations and analysis
7. Understand and use conditioning to simplify problems

8. Estimate parameters of distributions from samples
9. Understand dependence and independence among random phenomena
10. Use statistical tests to determine or characterize dependence among random phenomena
11. Design experiments to understand random phenomena
12. Understand the difference between Bayesian statistics and classical statistics
13. Use simulation to calculate Bayesian statistics
14. Apply linear algebra for data processing and statistical calculations

The main goal of this course is to equip the students with a data science mindset for successful practical implementations, in particular: understand, analyze, and design an approach to work with a data science or electrical engineering problem.

Contribution of course to meeting the professional component

4 credits of Engineering Science

Relationship of course to program outcomes:

1. An ability to identify, formulate, and solve engineering problems by applying principles of engineering, science, and mathematics. $\Rightarrow$ High

2. An ability to apply both analysis and synthesis in the engineering design process, resulting in designs that meet desired needs. $\Rightarrow$ High

3. An ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions. $\Rightarrow$ High

Time commitment

This is an estimate:

Work	Hours/Week
Attend lectures, ask questions	3.33
Study/Read ~20 pages of lecture notes and code	4
Reading assignments	1.2
Homework exercises	3
Total	**11.5**

Software Required

1. **Anaconda Distribution**

- with Python 3.8

- It includes all libraries, modules and tools we will use: Jupyter notebooks, `NumPy`, `Matplotlib`, `SciPy`, `Pandas`, `scikit-learn`, `random`
- Download and install it before next class

Some popular libraries in Anaconda

You have 2 options to manage your packages and virtual environment/s:

1. using `pip`. System that manages Python packages.
2. using `conda`. System that manages packages that may be written in any programming language.

Since we will use Python packages, you can use either one of these systems to manage your virtual environment. Which one to use typically comes with your specific needs. I typically use `conda` and that would be sufficient for this course.

Find help for creating and managing your **virtual environments**:

- using `conda` to manage virtual environments.
- using `pip` to manage virtual environments.

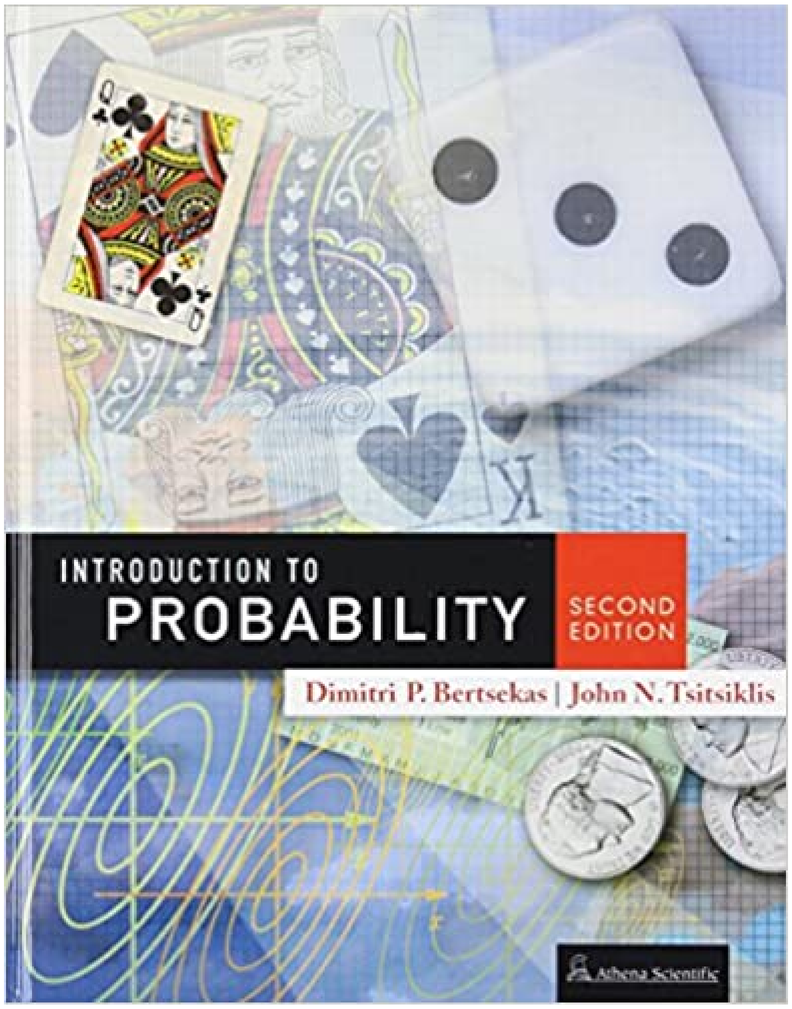
Textbooks - Required

1. **Introduction to Probability**

- Author: Dimitri P. Bertsekas, John N. Tsitsiklis
- Edition: 2nd

- Publisher: Athena Scientific
- Year: 2008
- ISBN: 978-1-886529-23-6

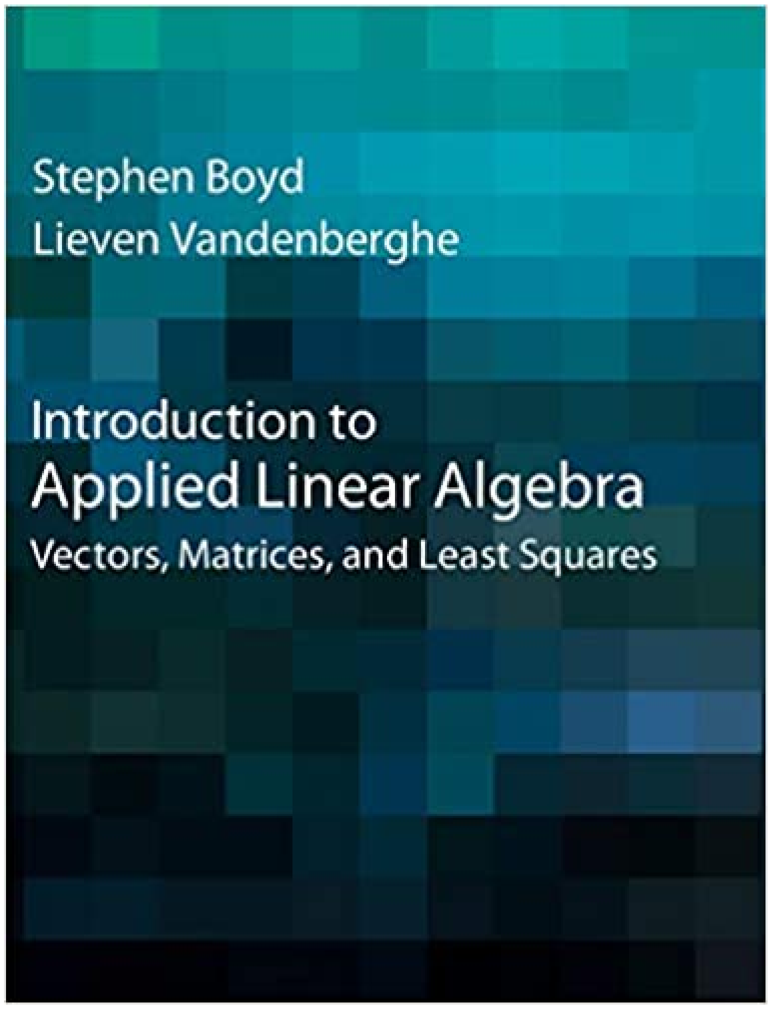
An **e-book version** will be cheaper and is perfectly fine for this course. The authors created an instructional digital version of the book and you can download it here



1. **Introduction to Applied Linear Algebra - Vectors, Matrices, and Least Squares**

- Author: Stephen Boyd
- Edition: 1st
- Publisher: Cambridge University Press
- Year: 2018
- ISBN: 978-1-886529-23-6

An **e-book version** is freely available and is perfectly fine for this course: you can download it here


Stephen Boyd
Lieven Vandenberghe
Introduction to
Applied Linear Algebra
Vectors, Matrices, and Least Squares

Library Course Reserves

- Additional readings will be listed in our Canvas page
- All reading materials are available on Course Reserves with both hard copy (at Marston't library) and electronic access

Course Schedule

A complete course schedule can be found in our Syllabus.

The semester will have 2 parts:

1. **Probability and Statistics.** Main textbook is "Introduction to Probability". Week 1-8
2. **Linear Algebra.** Main textbook is "Introduction to Applied Linear Algebra". Week 9-15

Course Policies

Please read the syllabus carefully.

1. **How to get help:** office hours, email (via Canvas or at shreya.saxena@ufl.edu), telephone, or Slack.
    - Slack channel: https://uf-eel3850-fall2022.slack.com/
    - Office Hours (Saxena): R 5 (11:45am-12:35pm) NEB 433
    - Office Hours (Hellwege): MW 12:30pm-1:20pm NEB 501
    - Office Hours (Santiago): MW 4:30pm-5:20pm NEB 501
2. **Attendance:** attendance is not required though summative and cumulative assessments may happen during class time. I will prepare course materials with the expectation that students will attend class synchronously. If any schedule conflict comes up, please email me so I'm informed and we can work together for alternative arrangements.
3. **Grading:** make sure your submissions are carefully completed with clean and well documented code. Make full use of Jupyter features, such as markdown text. Individual assignments will **not** be curved. Final grades **will** be curved.
4. **Late Work:** I will accept all assignment submissions as long as solutions have not yet been released, but you will lose the **on-time** points listed in the rubric. Solutions will typically be released 1 week after the assignment is due.
5. **Make-Up Policy:** If you feel that any assignment needs to be re-graded, you must discuss this with me within 1 week of grades being posted. If approved, the entire assignment will be subject to complete evaluation. Excused absences must be consistent with university policies in the graduate catalog and require appropriate documentation
6. **Collaboration:** healthy collaboration is encouraged. If another student contributes substantially to your understanding of a problem, you should cite this student. You will not be negatively judged for citing another student.
7. **Cheating and Plagiarism:** you are expected to submit your own work. If you are suspected of dishonest academic activity, I will invite you to discuss it further in private. Academic dishonesty will likely result in grade reduction, with severity depending on the nature of the dishonest activity. I am obligated to report on academic misconduct with a letter to the department, college and/or university leadership. Repeat offenses will be treated with significantly greater severity.

Grading

Grading will be based on:

Assignment	Total	Percentage Final Grade
Exams	3	20% each

Assignment	Total	Percentage Final Grade
Homework	~ 6	15%
Short Assignments	~ 8	15%
Participation	~5	10%

Homeworks will have 2 parts: (1) analytical exercises, typically solved on paper. (2) practical problems to be implemented in Python.

Exams will be drawn from lectures and readings. Practice exams will be provided.

Short Assignments will typically consist of short problems (with shorter turnaround time) to help consolidate and retain the information learned in class.

Participation will be in the form of discussion boards participation and class participation

Exams!

- Exam 1: September 26th @ 8:20 PM - 10:10 PM (tentative)
- Exam 2: November 2nd @ 8:20 PM - 10:10 PM (tentative)
- Exam 3: December 15th @ 10:00 AM - 12:00 PM

Grading Scale

Individual assignments will not be curved. **Final grades will be curved**, participation points will be used in grade boosting.

Percent	Grade	Grade Points
93.4 - 100	A	4.00
90.0 - 93.3	A-	3.67
86.7 - 89.9	B+	3.33
83.4 - 86.6	B	3.00
80.0 - 83.3	B-	2.67
76.7 - 79.9	C+	2.33
73.4 - 76.6	C	2.00
70.0 - 73.3	C-	1.67
66.7 - 69.9	D+	1.33
63.4 - 66.6	D	1.00
60.0 - 63.3	D-	0.67
0 - 59.9	E	0.00

Students Requiring Accommodations

Students with disabilities who experience learning barriers and would like to request academic accommodations should connect with the disability Resource Center by visiting https://disability.ufl.edu/students/get-started/.

- Please make sure you share your accommodation letter with me as soon as you have it, so we can discuss your access needs.

Course Evaluations

You are expected to provide professional and respectful feedback on the quality of instruction in this course by completing the course evaluations online via GatorEvals.

- The University is using a relatively new evaluation system, and evaluation results are now publicly available here: https://gatorevals.aa.ufl.edu/public-results/
- Guidance on how to give feedback in a professional and respectful manner is available at https://gatorevals.aa.ufl.edu/students/.
- You will be notified when the evaluation period opens, and can complete evaluations through the email you receive from GatorEvals, in the Canvas course menu under GatorEvals, or via https://ufl.bluera.com/ufl/.

University Honesty Policy

All UF students are bound by The Honor Pledge which states:

We, the members of the University of Florida community, pledge to hold ourselves and our peers to the highest standards of honor and integrity by abiding by the Honor Code. On all work submitted for credit by students at the University of Florida, the following pledge is either required or implied: "On my honor, I have neither given nor received unauthorized aid in doing this assignment."

The Honor Code specifies a number of behaviors that are in violation of this code and the possible sanctions. Furthermore, you are obligated to report any condition that facilitates academic misconduct to appropriate personnel. If you have any questions or concerns, please consult with the instructor or TAs in this class.

Commitment to a Safe and Inclusive Learning Environment

The Herbert Wertheim College of Engineering values broad diversity within our community and is committed to individual and group empowerment, inclusion, and the elimination of discrimination. It is expected that every person in this class will treat one another with dignity and respect regardless of gender, sexuality, disability, age, socioeconomic status, ethnicity, race, and culture.

If you feel like your performance in class is being impacted by discrimination or harassment of any kind, please contact your instructor or any of the following:

- Your academic advisor or Graduate Program Coordinator
- Robin Bielling, Director of Human Resources, 352-392-0903, rbielling@eng.ufl.edu
- Curtis Taylor, Associate Dean of Student Affairs, 352-392-2177, taylor@eng.ufl.edu
- Toshikazu Nishida, Associate Dean of Academic Affairs, 352-392-0943, nishida@eng.ufl.edu

Software Use

All faculty, staff, and students of the University are required and expected to obey the laws and legal agreements governing software use. Failure to do so can lead to monetary damages and/or criminal penalties for the individual violator. Because such violations are also against University policies and rules, disciplinary action will be taken as appropriate. We, the members of the University of Florida community, pledge to uphold ourselves and our peers to the highest standards of honesty and integrity.

Student Privacy

There are federal laws protecting your privacy with regards to grades earned in courses and on individual assignments. For more information, please see: https://registrar.ufl.edu/ferpa.html

Health and Wellness

U Matter, We Care:

Your well-being is important to the University of Florida. The U Matter, We Care initiative is committed to creating a culture of care on our campus by encouraging members of our community to look out for one another and to reach out for help if a member of our community is in need. If you or a friend is in distress, please contact umatter@ufl.edu so that the U Matter, We Care Team can reach out to the student in distress. A nighttime and weekend crisis counselor is available by phone at 352-392-1575. The U Matter, We Care Team can help connect students to the many other helping resources available including, but not limited to, Victim Advocates, Housing staff, and the Counseling and Wellness Center. Please remember that asking for help is a sign of strength. In case of emergency, call 9-1-1.

Counseling and Wellness Center: http://www.counseling.ufl.edu/cwc, and 392-1575; and the University Police Department: 392-1111 or 9-1-1 for emergencies.

Sexual Discrimination, Harassment, Assault, or Violence If you or a friend has been subjected to sexual discrimination, sexual harassment, sexual assault, or violence contact the Office of Title IX Compliance, located at Yon Hall Room 427, 1908 Stadium Road, (352) 273-1094, title-ix@ufl.edu

Sexual Assault Recovery Services (SARS), Student Health Care Center, 392-1161.

University Police Department at 392-1111 (or 9-1-1 for emergencies), or http://www.police.ufl.edu/.

Academic Resources

E-learning technical support, 352-392-4357 (select option 2) or e-mail to Learning-support@ufl.edu. https://lss.at.ufl.edu/help.shtml.

Career Resource Center, Reitz Union, 392-1601. Career assistance and counseling. https://www.crc.ufl.edu/.

Library Support, http://cms.uflib.ufl.edu/ask. Various ways to receive assistance with respect to using the libraries or finding resources.

Teaching Center, Broward Hall, 392-2010 or 392-6420. General study skills and tutoring. https://teachingcenter.ufl.edu/.

Writing Studio, 302 Tigert Hall, 846-1138. Help brainstorming, formatting, and writing papers. https://writing.ufl.edu/writing-studio/.

Student Complaints Campus: https://care.dso.ufl.edu.

On-Line Students Complaints: http://www.distance.ufl.edu/student-complaint-process.

Any Questions?

Jupyter Notebooks

The Jupyter notebook is a browser-based graphical interface to the **IPython** shell, and builds on a rich set of dynamic display capabilities.

As well as executing Python/IPython statements, the notebook allows the user to include formatted text, static and dynamic visualizations, mathematical equations, and much more. Furthermore, these documents can be saved in a way that lets other people open them and execute the code on their own systems.

Though the IPython notebook is viewed and edited through your web browser window, it must connect to a running Python process in order to execute code. This process (known as a "kernel") can be started by running one of the following ways:

Python Environment Demonstration

Step 1: Download and install Anaconda with Python 3.9 (default). If you are installing Anaconda for the first time, this will create a *base* environment with all the Anaconda libraries installed and **ready to run**.

Step 2: (optional) Create a new environment and install all libraries.

> *conda create --name eel3850*
>
> *conda activate eel3850*
>
> *conda install anaconda*

Step 3: (optional) Customize your environment by installing Jupyter Notebook extensions and RISE (Jupyter Notebook slideshow extensions).

> *conda install -c conda-forge jupyter_contrib_nbextensions*
>
> *conda install -c conda-forge rise*

Launching Jupyter Notebooks

Using Conda command line:

> *conda activate eel3850*
>
> *jupyter notebook*

Or launch it with Anaconda navigator interface.

How to open Jupyter

I will do a quick demonstration on how to launch Jupyter Notebook (and other IDEs) in a specific environment.

Creating new files & Jupyter Environment

I will demonstrate how to create new files with Python 3 kernels and review Jupyter Environment.

Getting Extensions (optional)

I have installed extra Jupyter functionalities supported by Nbextensions. It includes a wide list of functionalities that you may find useful.

I also have the slideshow extension for Jupyter Notebooks called RISE.

Introduction to Python

Python is an interpreted language

- Programs do not need to be compiled before they are executed
- Allows for rapid development and exploration (important for statistics/data science)

You can run the Python interpreter directly from the command line by calling ''python'' or ''python3''

We will instead use Juypter Notebook with a Python 3 kernel

A Jupyter notebook is divided into cells

Cells may be subdivided into an input cell and an output cell

Input cells generally either contain code or text

- The Markdown language can be used to *format* text
- LaTeX can be used to input math

Cell type defaults to "Code" and can be changed to "Markdown" using the Cell->Cell Type menu or a keyboard shortcut (ESC-m in Windows)

```
In [ ]:
```

Heading level 2

- Item 1
    - Item 2
        1. Item 3

$$\int_0^\infty e^x \, dx$$

```
2+2
def sum(a,b):
    out = a+b
    return out
```

You can use *italic* and **bold** with the * and ** sign, respectively, and mathematical equations with $ sign, in front and at the end of the equation: $x = 1$.

- You can also type in LateX using $ sign. Inline equation: $x^2 + y^2 = \alpha^2$
- Centered equation with double $ sign:

$$x^2 + y^2 = \alpha^2$$

Cells also may contain "magics", which are commands to the Jupyter notebook server

- For instance, to determine which directory you are in, you can use the "%pwd" (print working directory) magic:

```
In [2]: %pwd
```

```
Out[2]: '/Users/shreyasaxena/Dropbox (UFL)/Documents/Labwork/UF/Courses/Data Science f
or ECE/Fall 2022/Lectures/Lecture01'
```

You can use the "%cd" magic to change your directory:

```
In [3]: %cd ..
```

```
/Users/shreyasaxena/Dropbox (UFL)/Documents/Labwork/UF/Courses/Data Science fo
r ECE/Fall 2022/Lectures
```

~ is short your home directory

Hello World!

To print output in Python, use the print() function. It knows how to output many data types without providing explicit formatting (like printf in C)

```
In [4]: print('Hello world!')
```

```
Hello world!
```

```
In [5]: print("Hello World!")
```

```
Hello World!
```

```
In [6]: print(3859)
```

```
3859
```

- Code cells may contain multiple Python statements.
- All statements in a cell will be run sequentially when the cell is run
- Cells may be run using the "play" button, via the Cell-> Run Cells button or by using a keyboard command (usually shift-Enter)

```
In [7]: print('Hello')
        print('World!')
```

```
Hello
World!
```

In [8]:
```
print('Hello!')
print(44)
```

```
Hello!
44
```

In [9]:
```
print('Hello!')
34+43
```

```
Hello!
```

Out[9]:
```
77
```

Anything that follows a # (hash) symbol is a comment:

In [10]:
```
# This piece of code prints Hello world
print('Hello world')
```

```
Hello world
```

There is not really a multi-line comment in Python (like / / in C). One way to make a multiline comment is to just make a multi-line string that is not assigned to any variable. Multi-line strings are delimited by triple-ticks ('''):

```
This is a
multi-line
string
```

When we make functions, a multi-line string right after the function definition serves as the docstring (documentation string) for that function.

In [11]:
```
def myfun(x):
    '''This describes the function
    while taking two lines!'''
    return x
```

In [12]:
```
help(myfun)
```

```
Help on function myfun in module __main__:

myfun(x)
    This describes the function
    while taking two lines!
```

In [13]:
```
myfun?
```

Python is a dynamically typed language

Variable types are determined when they are assigned values

In [14]:
```
x=10
```

```
In [15]: print(x)
```

```
10
```

```
In [16]: type(x)
```

```
Out[16]: int
```

```
In [17]: x=9.0
```

```
In [18]: print(x)
```

```
9.0
```

```
In [19]: x='Hello world!'
         print(x)
```

```
Hello world!
```

```
In [20]: type(x)
```

```
Out[20]: str
```

Python will usually do the *right thing* based on the type of the variable

```
In [21]: a=3
         b=4
         print(a+b)
```

```
7
```

```
In [22]: a='Hello '
         b='World!'
         print(a+b)
```

```
Hello World!
```

Just be careful of the implications of this:

```
In [23]: a='3'
         b='4'
         print(a+b)
```

```
34
```

You can add color to your text by using the code `<font color=blue|red|green|pink|yellow>Text</font>`

You can also add hyperlinks to your text like this, e.g. Canvas page

Sometimes I will use *Bootstrap alerts* to emphasize new concepts or convey new information:

Bootstrap alerts All of these markdown rules can also be used when you are editing READ ME files on GitHub!

All of these markdown rules can also be used when you are editing READ ME files on GitHub!

Indentation conveys meaning in Python

Use indentation to indicate code blocks that belong together

In [24]:
```
a=2
if a==2:
    print('yay!')
else:
    print('nay!')
```

```
yay!
```

In [25]:
```
for x in range(10):
    if x%2==0:
        print(x)
```

```
0
2
4
6
8
```

In [26]:
```
print(range(10))
```

```
range(0, 10)
```

In [27]:
```
list(range(10))
```

Out[27]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ranges in Python start by default at 0 and are exclusive of the end point (I.e., if started at 0, the end point is also the length of the sequence)

In [28]:
```
list(range(1,10,2))
```

Out[28]:
```
[1, 3, 5, 7, 9]
```

Main Data Types

The main data types in Python are:

- numbers (int, float, complex)
- string
- list
- tuple
- dictionary

All data types are objects. That means they have methods associated with them

In [29]:
```
x=7
```

In [30]:
```
help(x)
```

```
Help on int object:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
 |  4
 |
 |  Built-in subclasses:
 |      bool
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __bool__(self, /)
 |      self != 0
 |
 |  __ceil__(...)
 |      Ceiling of an Integral returns itself.
 |
 |  __divmod__(self, value, /)
 |      Return divmod(self, value).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __float__(self, /)
 |      float(self)
 |
 |  __floor__(...)
 |      Flooring an Integral returns itself.
 |
 |  __floordiv__(self, value, /)
 |      Return self//value.
 |
 |  __format__(self, format_spec, /)
 |      Default object formatter.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |  
 |  __getnewargs__(self, /)
 |  
 |  __gt__(self, value, /)
 |      Return self>value.
 |  
 |  __hash__(self, /)
 |      Return hash(self).
 |  
 |  __index__(self, /)
 |      Return self converted to an integer, if self is suitable for use as an
index into a list.
 |  
 |  __int__(self, /)
 |      int(self)
 |  
 |  __invert__(self, /)
 |      ~self
 |  
 |  __le__(self, value, /)
 |      Return self<=value.
 |  
 |  __lshift__(self, value, /)
 |      Return self<<value.
 |  
 |  __lt__(self, value, /)
 |      Return self<value.
 |  
 |  __mod__(self, value, /)
 |      Return self%value.
 |  
 |  __mul__(self, value, /)
 |      Return self*value.
 |  
 |  __ne__(self, value, /)
 |      Return self!=value.
 |  
 |  __neg__(self, /)
 |      -self
 |  
 |  __or__(self, value, /)
 |      Return self|value.
 |  
 |  __pos__(self, /)
 |      +self
 |  
 |  __pow__(self, value, mod=None, /)
 |      Return pow(self, value, mod).
 |  
 |  __radd__(self, value, /)
 |      Return value+self.
 |  
 |  __rand__(self, value, /)
 |      Return value&self.
 |  
 |  __rdivmod__(self, value, /)
 |      Return divmod(value, self).
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __rfloordiv__(self, value, /)
 |      Return value//self.
 |  
 |  __rlshift__(self, value, /)
 |      Return value<<self.
 |  
 |  __rmod__(self, value, /)
 |      Return value%self.
 |  
 |  __rmul__(self, value, /)
 |      Return value*self.
 |  
 |  __ror__(self, value, /)
 |      Return value|self.
 |  
 |  __round__(...)
 |      Rounding an Integral returns itself.
 |      Rounding with an ndigits argument also returns an integer.
 |  
 |  __rpow__(self, value, mod=None, /)
 |      Return pow(value, self, mod).
 |  
 |  __rrshift__(self, value, /)
 |      Return value>>self.
 |  
 |  __rshift__(self, value, /)
 |      Return self>>value.
 |  
 |  __rsub__(self, value, /)
 |      Return value-self.
 |  
 |  __rtruediv__(self, value, /)
 |      Return value/self.
 |  
 |  __rxor__(self, value, /)
 |      Return value^self.
 |  
 |  __sizeof__(self, /)
 |      Returns size in memory, in bytes.
 |  
 |  __sub__(self, value, /)
 |      Return self-value.
 |  
 |  __truediv__(self, value, /)
 |      Return self/value.
 |  
 |  __trunc__(...)
 |      Truncating an Integral returns itself.
 |  
 |  __xor__(self, value, /)
 |      Return self^value.
 |  
 |  as_integer_ratio(self, /)
 |      Return integer ratio.
 |  
 |      Return a pair of integers, whose ratio is exactly equal to the origina
l int
 |      and with a positive denominator.
 |
 |      >>> (10).as_integer_ratio()
 |      (10, 1)
 |      >>> (-10).as_integer_ratio()
 |      (-10, 1)
 |      >>> (0).as_integer_ratio()
 |      (0, 1)
 |
 |  bit_length(self, /)
 |      Number of bits necessary to represent self in binary.
 |
 |      >>> bin(37)
 |      '0b100101'
 |      >>> (37).bit_length()
 |      6
 |
 |  conjugate(...)
 |      Returns self, the complex conjugate of any int.
 |
 |  to_bytes(self, /, length, byteorder, *, signed=False)
 |      Return an array of bytes representing an integer.
 |
 |      length
 |        Length of bytes object to use.  An OverflowError is raised if the
 |        integer is not representable with the given number of bytes.
 |      byteorder
 |        The byte order used to represent the integer.  If byteorder is 'bi
g',
 |        the most significant byte is at the beginning of the byte array.  If
 |        byteorder is 'little', the most significant byte is at the end of th
e
 |        byte array.  To request the native byte order of the host system, us
e
 |        `sys.byteorder' as the byte order value.
 |      signed
 |        Determines whether two's complement is used to represent the intege
r.
 |        If signed is False and a negative integer is given, an OverflowError
 |        is raised.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  from_bytes(bytes, byteorder, *, signed=False) from builtins.type
 |      Return the integer represented by the given array of bytes.
 |
 |      bytes
 |        Holds the array of bytes to convert.  The argument must either
 |        support the buffer protocol or be an iterable object producing byte
s.
 |        Bytes and bytearray are examples of built-in objects that support th
e
 |        buffer protocol.
 |      byteorder
 |        The byte order used to represent the integer.  If byteorder is 'bi
g',
 |        the most significant byte is at the beginning of the byte array.  If
 |        byteorder is 'little', the most significant byte is at the end of th
e
 |        byte array.  To request the native byte order of the host system, us
```

```
e
 |        `sys.byteorder' as the byte order value.
 |      signed
 |        Indicates whether two's complement is used to represent the integer.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signatur
e.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  denominator
 |      the denominator of a rational number in lowest terms
 |
 |  imag
 |      the imaginary part of a complex number
 |
 |  numerator
 |      the numerator of a rational number in lowest terms
 |
 |  real
 |      the real part of a complex number
```

Methods with are designated as private, but you can still call them:

In [31]:
```
x.__mod__(3)
```

Out[31]:
```
1
```

Usually, you don't call the private methods because there are other ways of achieving the same thing that are easier to interpret:

In [32]:
```
x%3
```

Out[32]:
```
1
```

In [33]:
```
y='Hello'
y.__len__()
```

Out[33]:
```
5
```

In [34]:
```
len(y)
```

Out[34]:
```
5
```

Mutability

- Some data types in Python are **immutable**, i.e. they cannot be changed. These include numbers, strings, and tuples
- Lists and dictionaries are **mutable**, they can be changed

In [35]:
```
a=(2,3)
```

In [36]:
```
type(a)
```

Out[36]:
```
tuple
```

In [37]:
```
a=a+(4,)
print(a)
```

```
(2, 3, 4)
```

How did a change if it is immutable?

- a did not change, a new tuple was created that added 4 to the previous tuple, and a was updated to point to the new tuple. How can we tell?

In [38]:
```
a=(2,3)
b=a
a is b
```

Out[38]:
```
True
```

In [39]:
```
print(id(a),id(b))
```

```
140446403469888 140446403469888
```

In [40]:
```
# a = a+(4,)

a += (4,)
print(a)
```

```
(2, 3, 4)
```

In [41]:
```
a is b
```

Out[41]:
```
False
```

In [42]:
```
print(id(a),id(b))
```

```
140446102470976 140446403469888
```

Lists, on the other hand, are mutable:

In [43]:
```
a = [2,3] # list
b = a
a is b
```

Out[43]:
```
True
```

In [44]:
```
print(id(a),id(b))
```

```
140446102497664 140446102497664
```

In [45]:
```
a.append(4)
```

In [46]:
```
print(a,b)
```

```
[2, 3, 4] [2, 3, 4]
```

```
In [47]: print(id(a),id(b))
```

```
140446102497664 140446102497664
```

In order to leave the original list unchanged, we need to *copy* it. This will create a new list in memory:

```
In [48]: a=[2,3]
         b=a
         c=a.copy()
```

```
In [49]: a+=[(5,2)]
```

```
In [50]: print(a,b,c)
```

```
[2, 3, (5, 2)] [2, 3, (5, 2)] [2, 3]
```

You can also append to lists with +:

```
In [51]: a+=['String1','String2']
```

```
In [52]: a
```

```
Out[52]: [2, 3, (5, 2), 'String1', 'String2']
```

Lists and tuples may contain any other objects, including other lists and tuples:

```
In [53]: a[3]
```

```
Out[53]: 'String1'
```

Note that tuples and lists are ordered collections, and we can access their members directly:

```
In [54]: a[0:2]
```

```
Out[54]: [2, 3]
```

```
In [55]: a[:2]
```

```
Out[55]: [2, 3]
```

```
In [56]: a[3:]
```

```
Out[56]: ['String1', 'String2']
```

Negative indexes start from the end of the list, with -1 denoting the last member in the list:

```
In [57]: a[-1]
```

```
Out[57]: 'String2'
```

```
In [58]: a[-4:-1]
```

```
Out[58]: [3, (5, 2), 'String1']
```

Modules and Libraries

Many of the tools we will use in the class are not directly part of Python.

```
In [59]: sin(3.14)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [59], in <cell line: 1>()
----> 1 sin(3.14)

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

Instead, they are libraries or modules that provide particular functionality. These include:

- **NumPy** provides arrays, linear algebra, and math functions (many similar to the core MATLAB functions)
- **Matplotlib** provides functions to generate plots similar to those in MATLAB
- **pandas** provides tools for working with data
- **SciPy** provides many tools used in scientific computing including optimization, signal processing, and statistics
- **scikit-learn** provides various classification, regression, clustering algorithms and model implementations.

When you install the Anaconda distribution, all of these libraries will be installed in your machine!

To work with these libraries, import them:

```
In [60]: import numpy
```

```
In [61]: numpy.sin(3.14)
```

```
Out[61]: 0.0015926529164868282
```

```
In [62]: numpy.sin(numpy.pi)
```

```
Out[62]: 1.2246467991473532e-16
```

To reduce typing, you can relabel a library on import:

```
In [63]: import numpy as np
```

```
Out[63]: 1.2246467991473532e-16
```

```
In [64]: np.sin(np.pi)
```

```
Out[64]: 1.2246467991473532e-16
```

Let's take a look at how we can customize and stylize our visualizations using `Matplotlib`.

The first thing we need to do is import the library:

```
In [65]: import matplotlib.pyplot as plt
```

```
In [66]: squares = []
         vals = []

         for x in range(10,20):
             vals += [x]
             squares += [x**2]
```

```
In [70]: print(vals)
```

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [69]: print(squares)
```

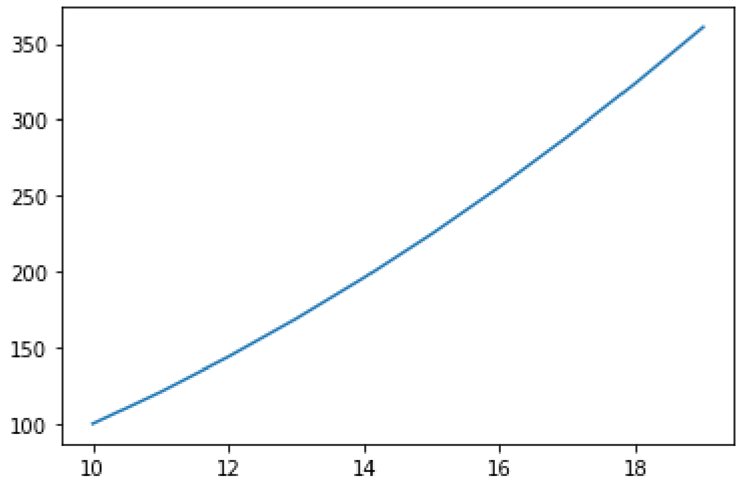
```
[100, 121, 144, 169, 196, 225, 256, 289, 324, 361]
```

On top of the normal Python syntax, IPython has *magic commands*, that are prefixed by the `%` character. There are many magic commands, these are particularly useful when using Matplotlib:

```
In [71]: %matplotlib?
```

```
In [72]: %matplotlib inline
```

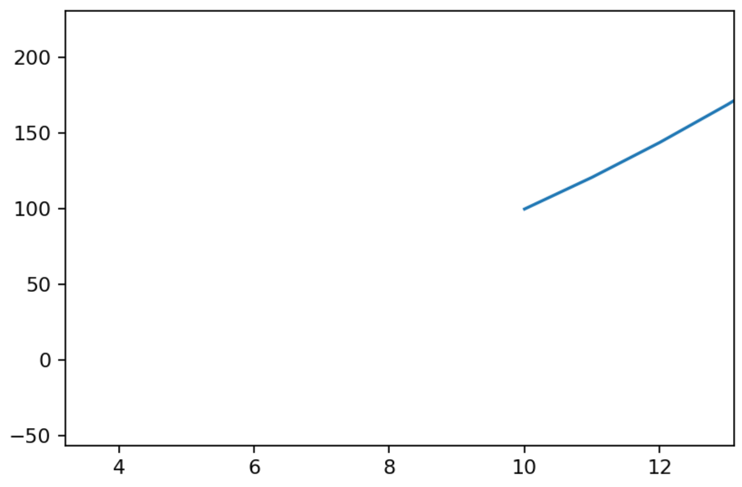
```
In [74]: plt.plot(vals,squares);
```



```
In [75]: # will create interactive plots within the notebook

         %matplotlib notebook
```

In [76]:
```
plt.plot(vals,squares);
```

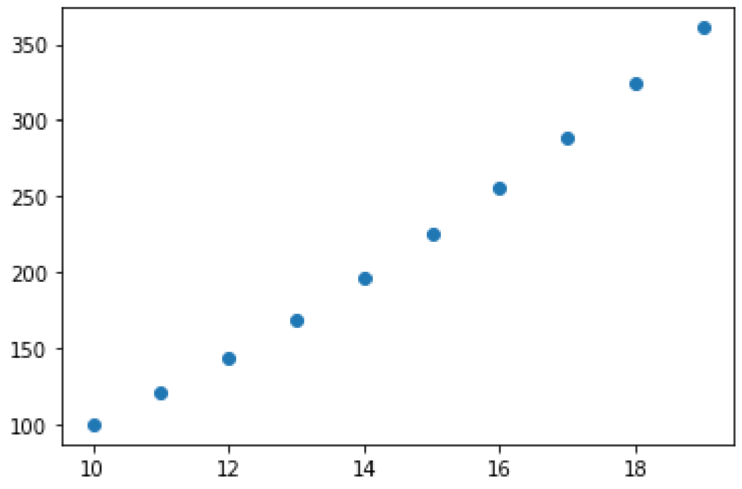


In [77]:
```
%matplotlib inline
```

Note that this `plot` interpolates among the data points. I.e., we can look at an x value of 14 and see that the corresponding y value is around 200. But those values *do not exist* in the data.

- Our choice of plot depends a lot on what we want to convey. If our data is sub-sampled from a continuous function (as in this case), then we probably want interpolation.
- If we just want to see the data points, then we can instead use scatter plot:

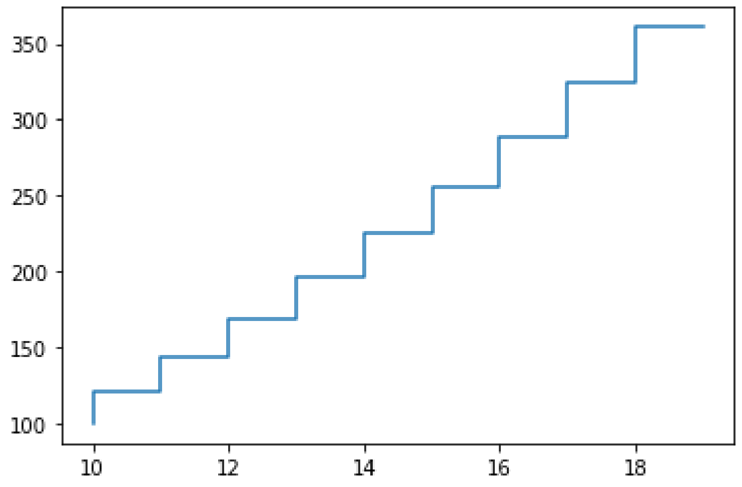
In [78]:
```
plt.scatter(vals,squares);
```



Sometimes, we may want a continuous function with the y-values held constant (known as **zero-order hold**) until the next data point.

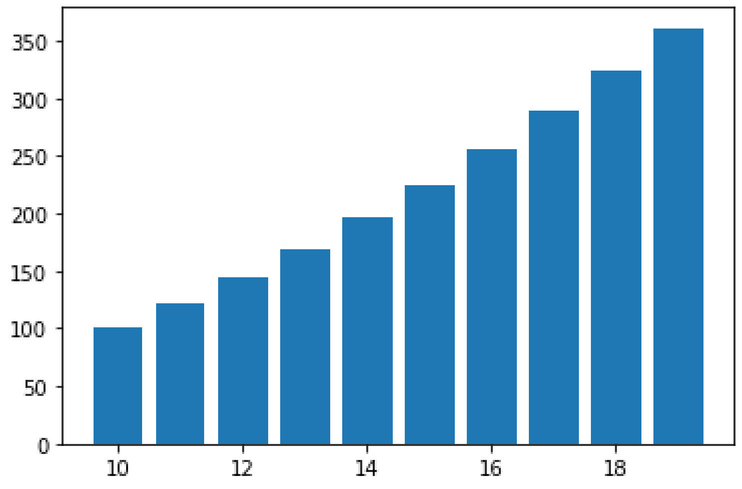
- That is a step or staircase plot:

```
In [80]: plt.step(vals,squares);
```



Bar plots are usually used to show counts of items:

```
In [82]: plt.bar(vals,squares);
```

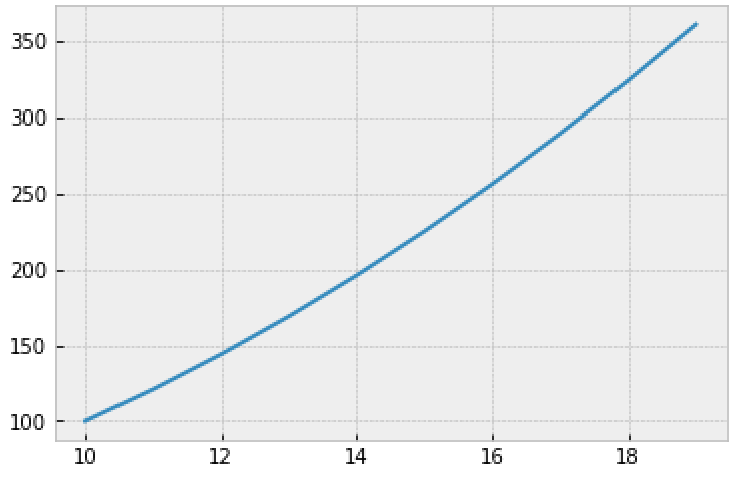


We can **style** these plots in many different ways, and we will learn about the most important ways as we go through the class.

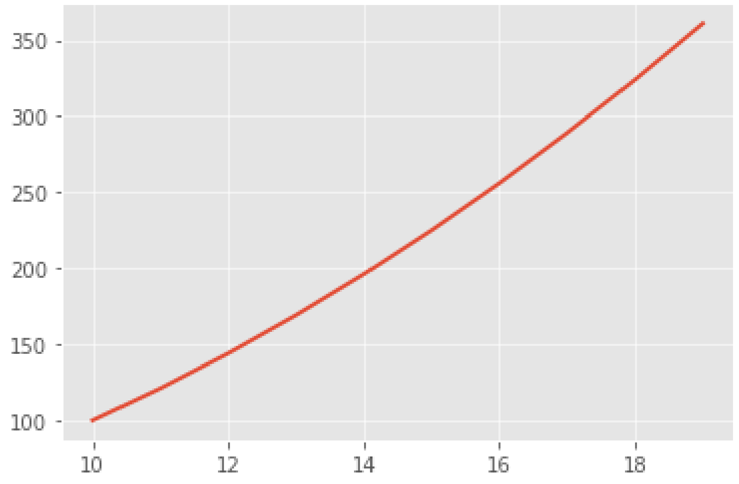
- A quick way to change your plot style is:

```
In [83]: plt.style.use('bmh')
         plt.plot(vals,squares);
```

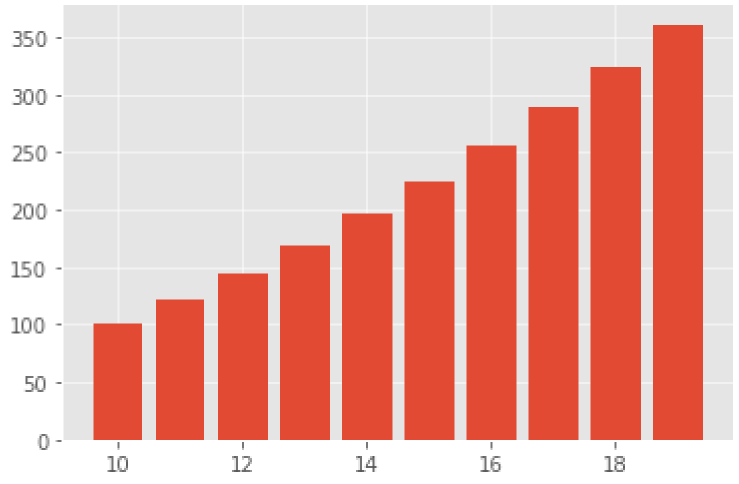
```
Out[83]: [<matplotlib.lines.Line2D at 0x7fbc08a4d4f0>]
```



In [85]:
```
plt.style.use('ggplot')
plt.plot(vals,squares);
```



In [87]:
```
plt.bar(vals,squares);
```



A style sheet reference for the latest versionn of Matplotlib can be found here: https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

Resource to learn more about Plotting with Matplotlib

Chapter 4 "Visualization with Matplotlib" from the online book Python Data Science Handbook by Jake VanderPlas is a great resource to learn more about different types of plots, customize legends and colorbars, adding text annotations and more.

To prepare

1. Download and install Anaconda
2. Download and install Git
3. (Optional) Download and install GitHub Desktop
4. Join GitHub organization
5. Create your repository for Short Assignment 0. You will be automatically added as a collaborator. We will send you an invitation to join organization as a member, only then you will have access to important repositories, such as Lectures.
6. Clone the following repositories to your machine: "Lectures" and "Assignment-Solutions".
7. Take a look at the **Modules** page in Canvas and get familiar with a typical lecture layout: it includes readings and activities to help you study. Module 1 has the jupyter notebooks (raw and in-class-edits) as well as the pdf of Thursday's lecture.

Any Questions?

Feel free to email me afterwards or come talk with me or one of the TAs during office hours.

```
In [ ]:
```