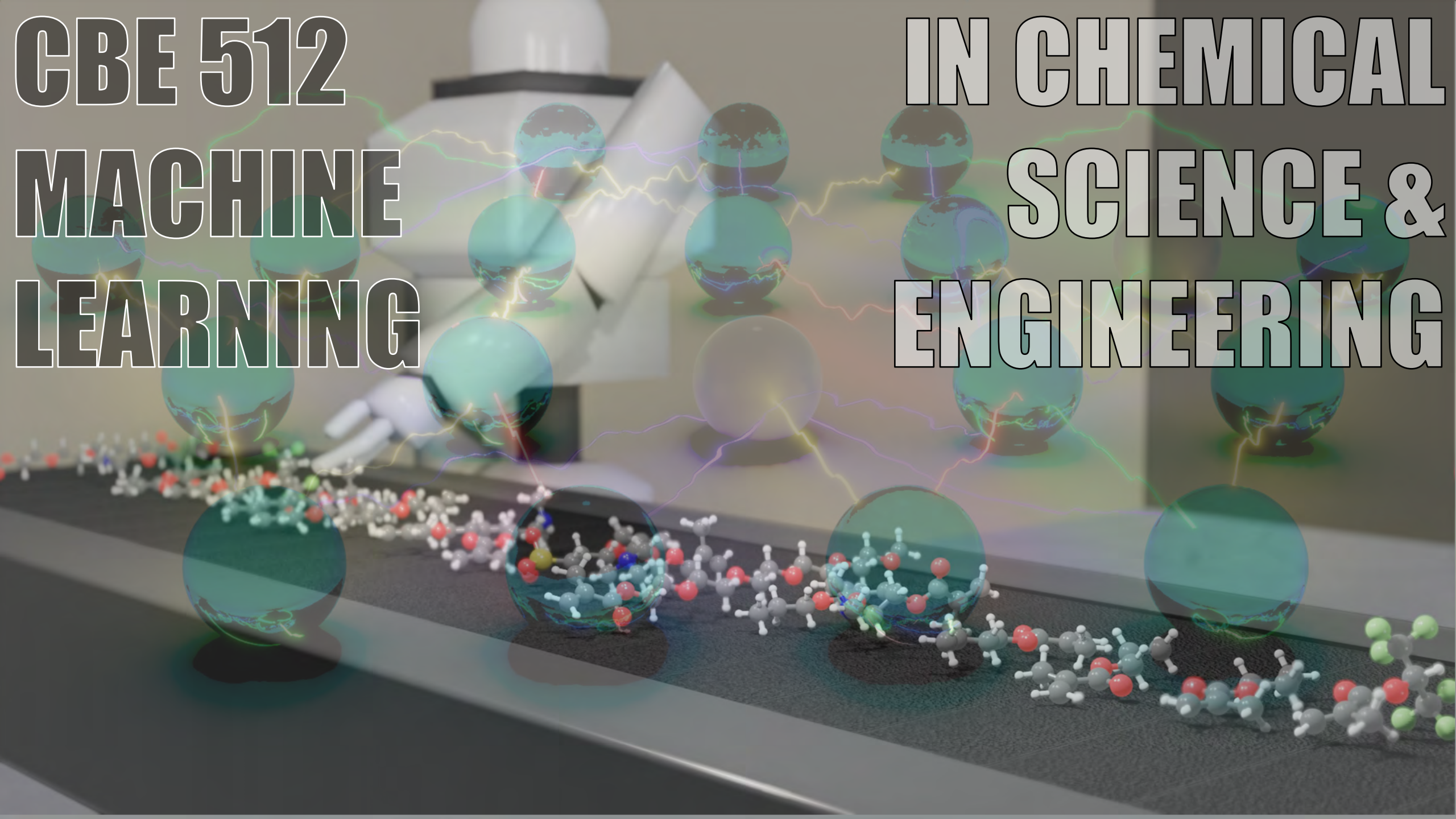

CBE 512
MACHINE
LEARNING
IN CHEMICAL
SCIENCE &
ENGINEERING

So, what's the deal with Machine Learning (in science and engineering)?

*via imprecise searching on Web of Science

AI4Science to empower the fifth paradigm of scientific discovery

Published July 7, 2022

By Christopher Bishop, Technical Fellow and Director, Microsoft Research AI4Science

Research Area

Artificial intelligence

8000

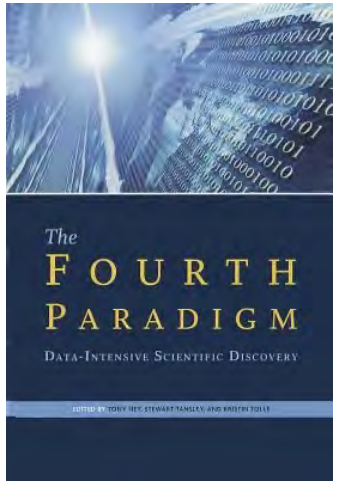
Michael Webb suggests the sixth paradigm of scientific discovery

Made up September 5, 2023

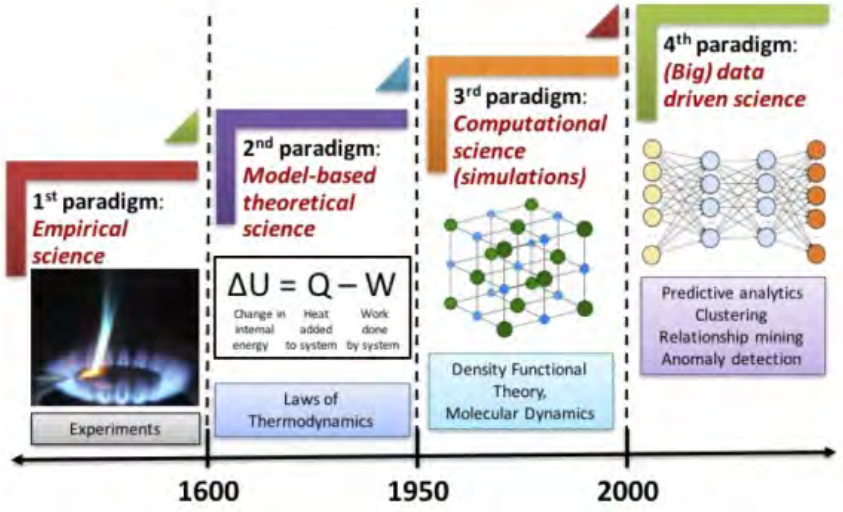
By Michael Webb, CBE 512 Instructor, Princeton University

Research Area

Artificial intelligence



Microsoft Research (2004)



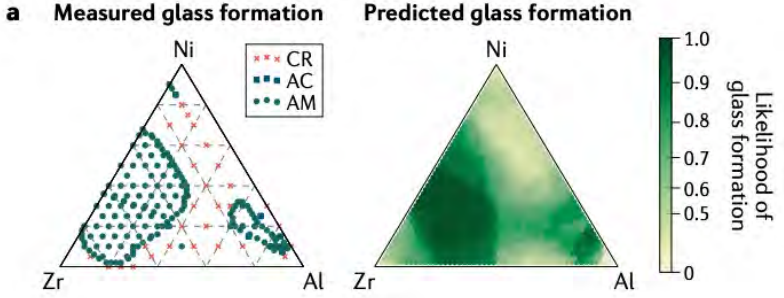
Agrawal & Choudary. *APL Mater.* **4** (2016)

Rise of the Machines: Materials Science

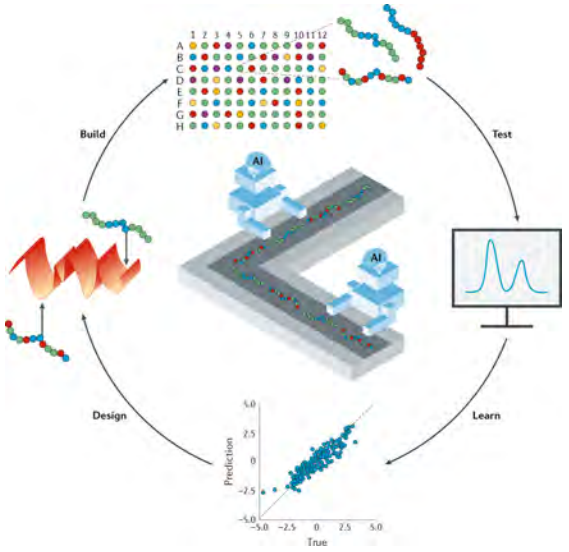
COLLECTION | 17 AUGUST 2021

Machine learning in materials science

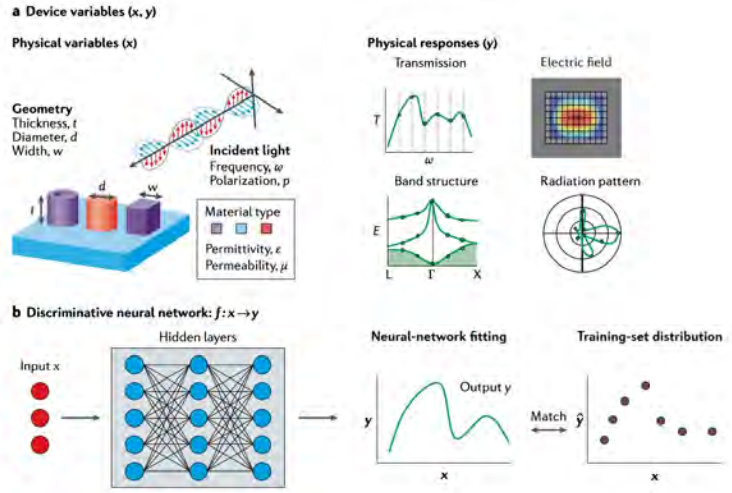
Machine learning is a powerful tool in materials research. Our collection of articles looks in depth at applications of machine learning in various areas of materials science.



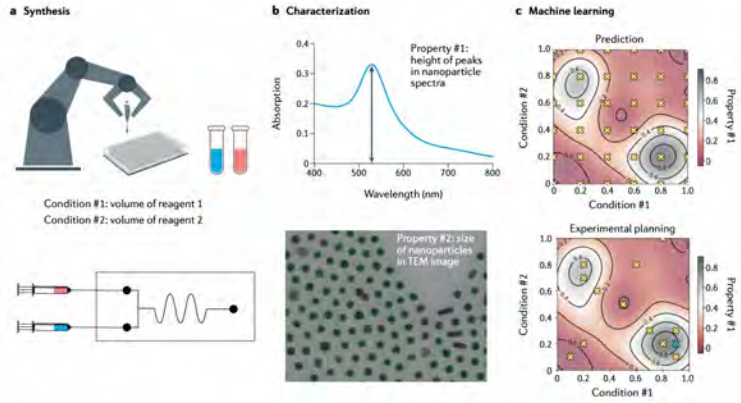
*"Machine learning for **alloys**"*
Hart, Mueller, Toher, Curtarolo



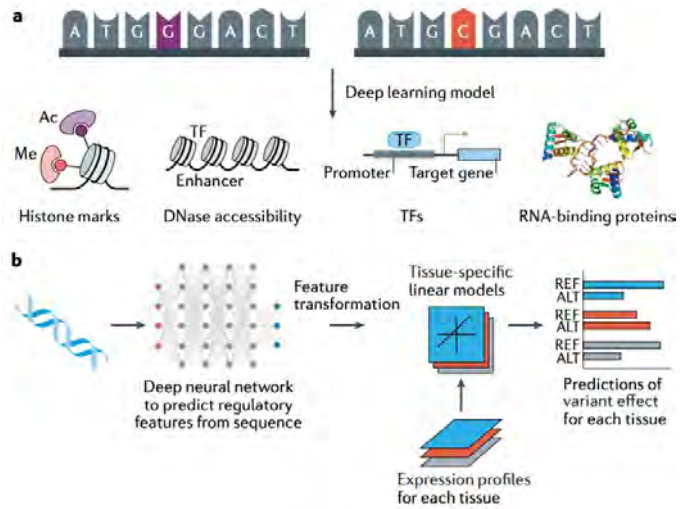
*"Machine Learning in combinatorial **polymer chemistry**"*
Gormley and Webb



*"Deep neural networks for the evaluation and design of **photonic devices**"*
Jiang, Chen, and Fan



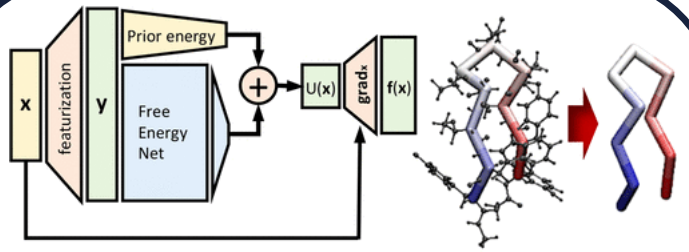
*"**Nanoparticle synthesis** assisted by machine learning"*
Tao, Wu, Aldeghi, Aspuru-Guzik, abd Kumacheva



*"Machine learning methods to model **multicellular** complexity and **tissue** specificity"*
Sealfon, Wong, and Troyanskaya

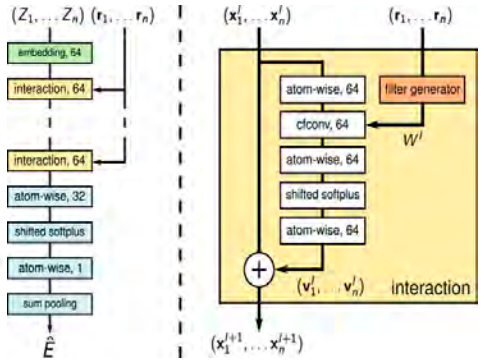
Rise of the Machines: Molecular Simulation

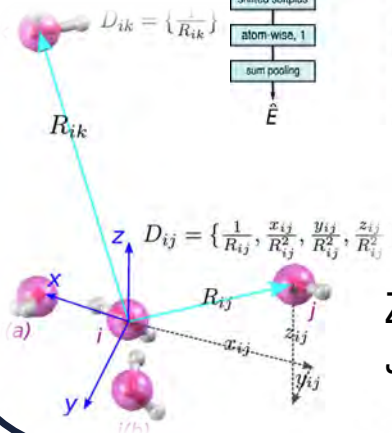
Force Field Development



Wang et al. *ACS Cent. Sci.* (2019)

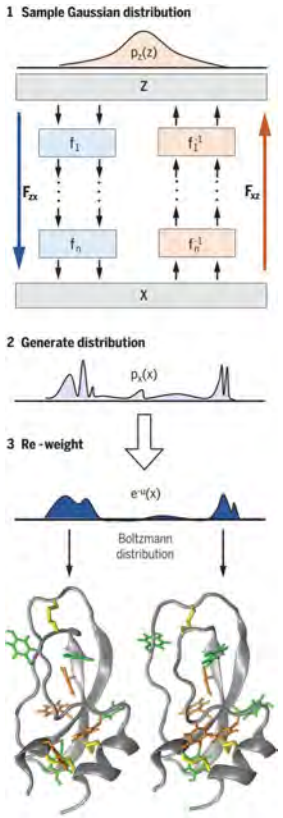
Schütt et al. *JCP* (2018)



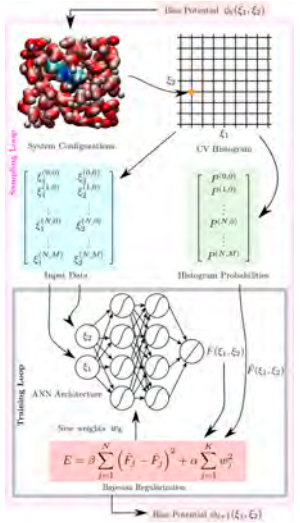


Zhang et al. *JCP* (2018)

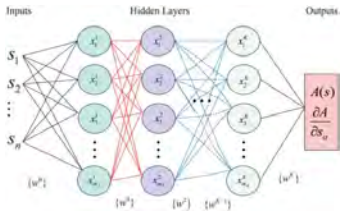
Sampling & Simulation



Noé et al. *Science* (2019)

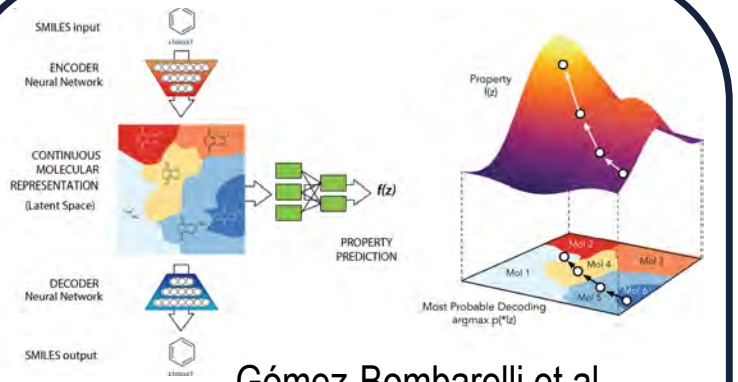


Sidky & Whitmer. *JCP* (2018)



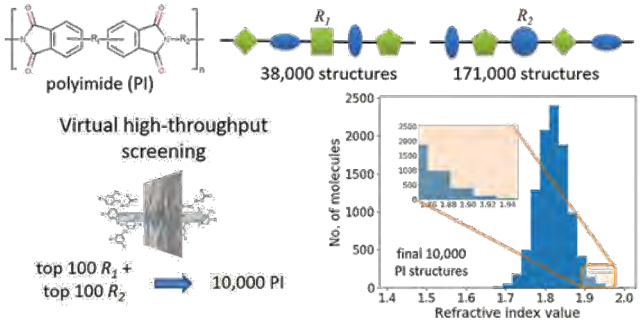
Shneider et al. *PRL* (2017)

Property Prediction & Design



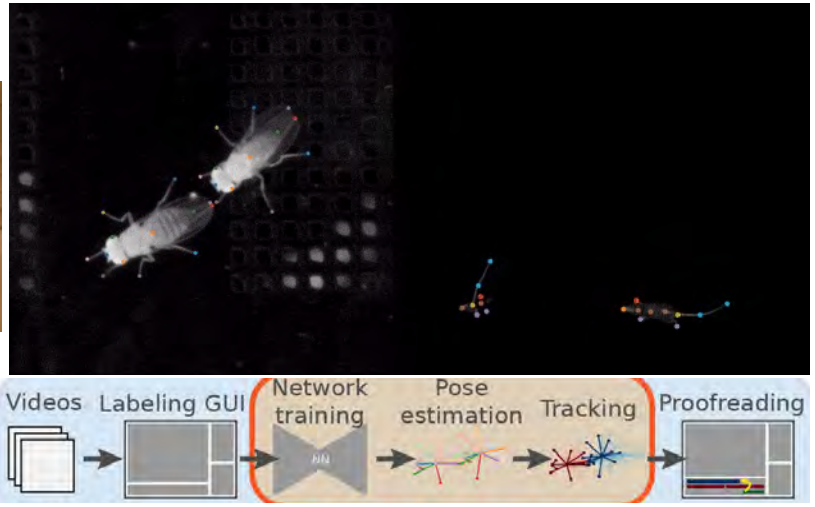
Gómez-Bombarelli et al. *ACS Cent. Sci.* (2018)

Atif Faiz Afzal et al. *JPC C* (2019)

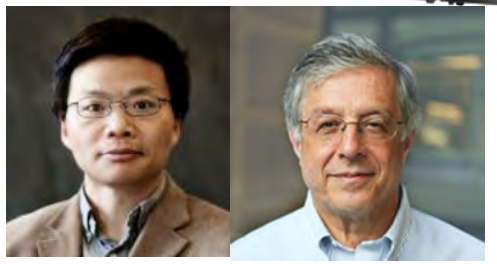


Rise of the Machines: Diverse Examples From Around Princeton

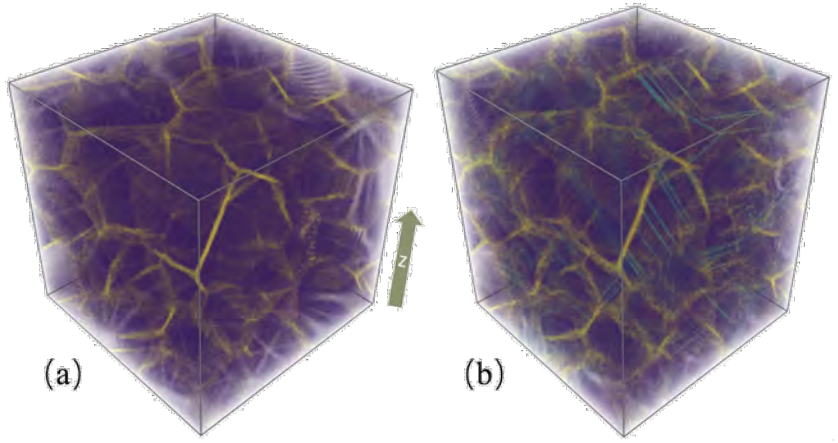
Social LEAP Estimates Animal Poses



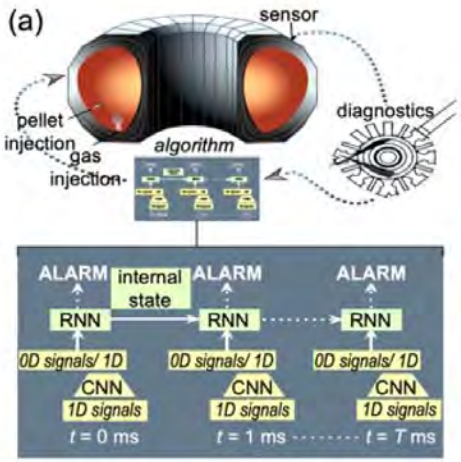
Scalable Many-body Quantum Chemical Modeling



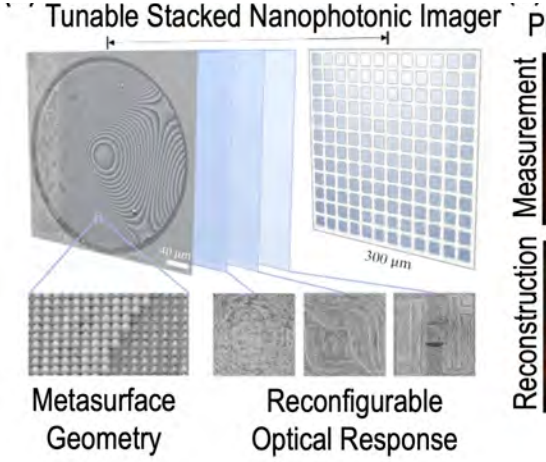
10,000,000+ atom simulation!

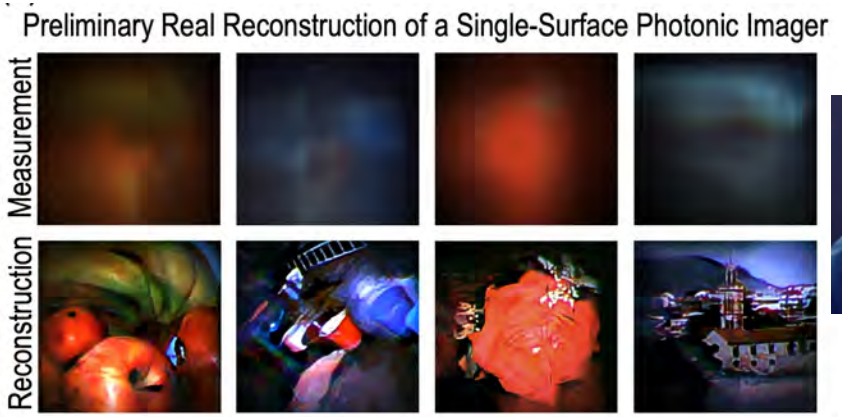


Prediction & Control for Fusion Energy



Nanophotonic Camera Design



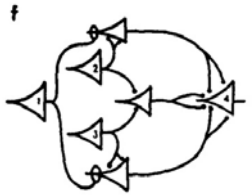


Why does CBE 512 exist? The origin story...

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS

Bulletin of Mathematical Biophysics, Vol. 5, 1943, p. 115-133



*McCulloch and Pitts publish paper establishing a mathematical network representation of nervous activity based on propositional logic (i.e., a **neural network**)*

Lighhill report

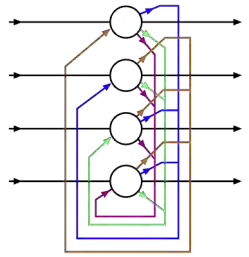
Part I Artificial Intelligence

A general survey by Sir James Lighthill FRS
Lucasian Professor of Applied Mathematics,
Cambridge University. July 1972.

"Workers entered the field around 1950, and even around 1960, with high hopes that are very far from having been realised in 1972. In no part of the field have the discoveries made so far produced the major impact that was then promised."

Hopfield network

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j + \sum_i \theta_i s_i$$



2021: AlphaFold2
2022: Dall-E 2, Stable Diffusion ChatGPT,...

you arrive in first offering CBE 512

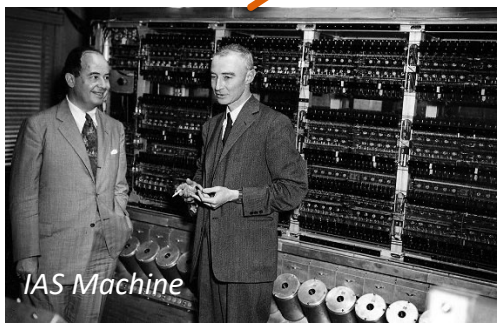
CBE 512

Big Bang

1943 — *McCulloch and Pitts publish paper establishing a mathematical network representation of nervous activity based on propositional logic (i.e., a **neural network**)*

1950 — *Alan Turing proposes Turing Test*

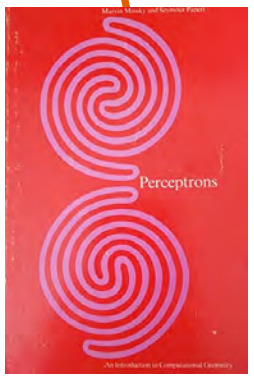
1953



Continuing advances in computing

1957 — *Frank Rosenblatt's Perceptron used for image recognition*

1969



Minsky & Papert publish Perceptrons

1972

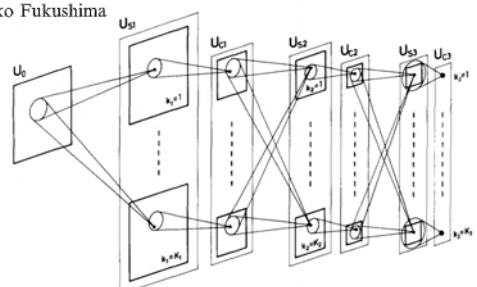
1979 — *Kunihiko Fukushima proposes multilayered ANN*

Biological Cybernetics

© by Springer-Verlag 1980

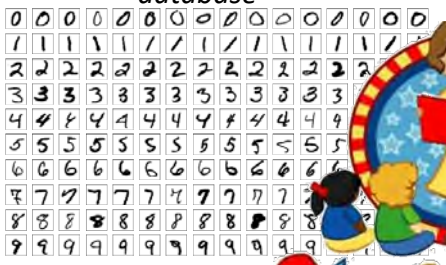
Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position

Kunihiko Fukushima



1982

1998 — *MNIST database*



1982 — *Hopfield network*

LSTMs

Random Forests + SVM

2002 — *Torch introduced*

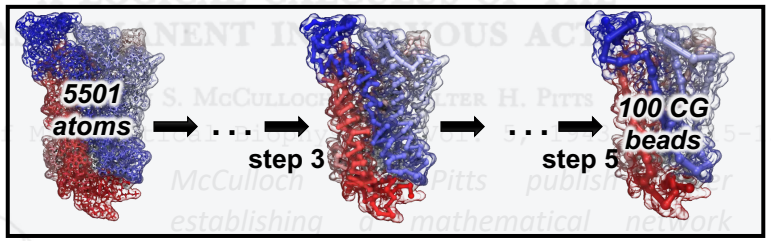
2014 — *GANs*

Explosion of interest and applications by BigTech

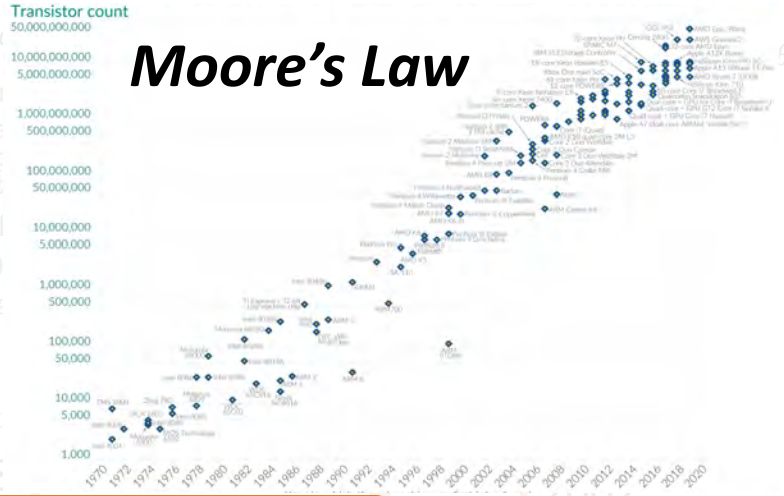
2021

2024

Why does CBE 512 exist? The origin story...



"Graph-Based Approach to Systematic Molecular Coarse-Graining" Webb et al. *JCTC* (2018)



Moore's Law

2021: AlphaFold2
2022: Dall-E 2, Stable Diffusion ChatGPT,...

you arrive in first offering CBE 512

2021

2024

Big Bang

Early attempts using AI in ChE

Building momentum in science & engineering

2018

I have a question/idea...

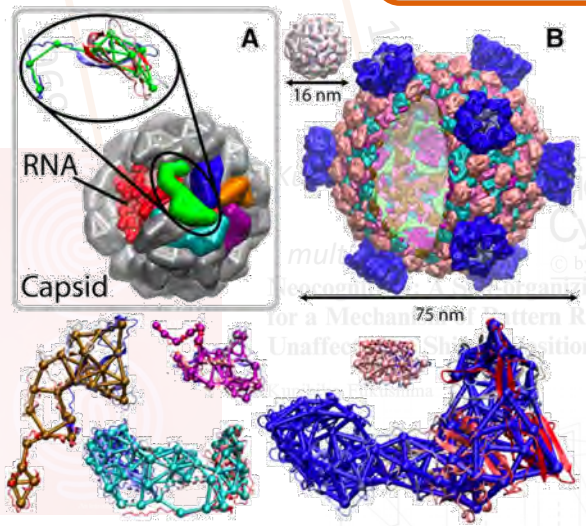
Topology Representing Networks

THOMAS MARTINETZ [1,2] AND KLAUS SCHULTEN [1]

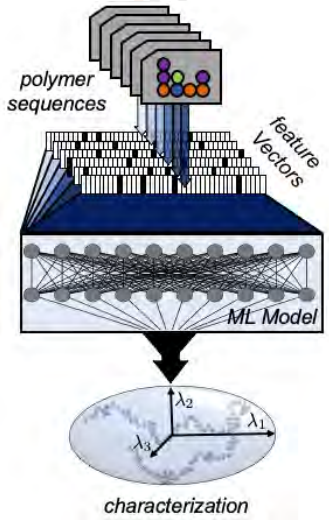
[1]University of Illinois and [2]Siemens AG

(*Received* 4 *March* 1993; *accepted* 20 *September* 1993)

"Shape-Based Coarse-graining"



"Stability and Dynamics of Virus Capsids Described by Coarse-grained Modeling" Arkhipov, Freddolino, Schulten (2006)



"Targeted Sequence Design on the Coarse-grained Polymer Genome" Webb et al. *Science Advances* (2020)

General Disclaimer and Course Guidance

- I have literally no “formal training” in Machine Learning

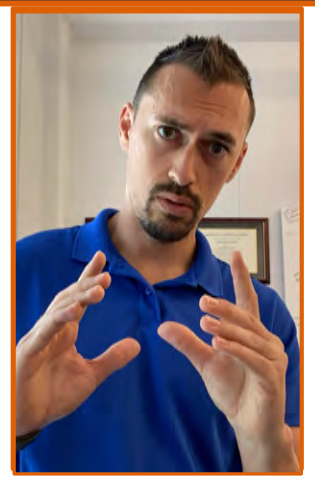
I have published numerous papers on related topics since 2018; organized multiple ML symposia at national and regional conferences; guest edited themed issues in academic journals; peer-reviewed many many articles for scholarly journals; applied for patents on ML-derived products; hosted and taught at workshops; consulted for chemical products companies

- I am going to provide candid opinions and descriptions informed by my understanding and experience. Recognize and evaluate that bias.

- Always be free to ask questions. There will be things I do not know. We will learn about them together!

- There may be mistakes during lectures. We will try to limit these and correct them as needed.

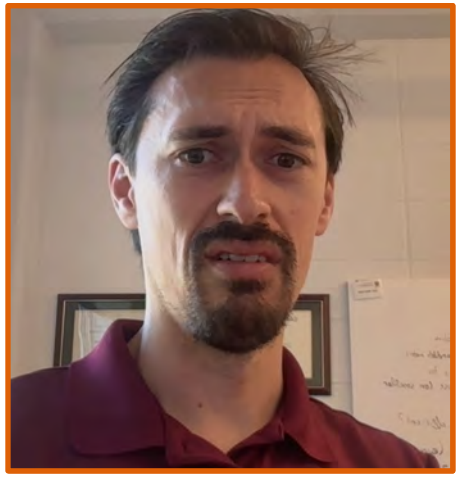
What should/will you gain from the course?



"Learning how to learn is life's ***most important skill****."*

"Sure, but learning how to ***machine learn****?*

What should/will you gain from the course?

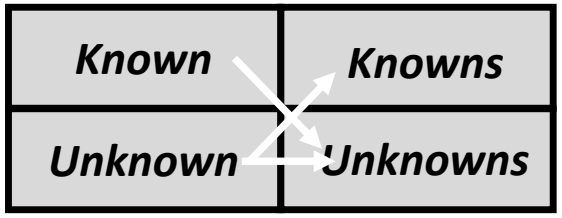


*"AI is not going to replace humans, but **humans with AI** are going to **replace humans without AI.***

*"By transitive property of... humans, **scientists and engineers who use AI** will **replace those that don't.**"*

What should/will you gain from the course?

- **Knowledge**



we will provide theoretical and practical introduction to a wide range of machine learning methods (unsupervised, supervised, classification, regression,…); this should enable you to know what you need to know/ask

we want to turn **known unknowns → known knowns**, reveal **unknown knowns → known knowns**, and convert unknown unknowns → at least known unknowns

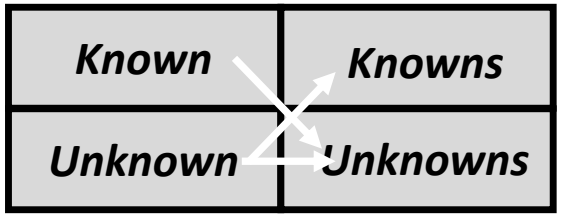
- **Skill**

you will develop gain practical experience in working with ML algorithms and their implementation in open-source software; you will become better at programming, data manipulation, and analysis)

1. This class will teach you fundamental guiding principles underlying common machine learning algorithms and strategies. This class will expose students to language and terms used in the context of machine learning. Students will be able to capably describe different branches of machine learning and provide distinguishing examples for their application and goals for their implementation.
2. This class will showcase applications in contexts that are broadly more relevant to "physical" engineers and scientists. Students will be able to more confidently approach and analyze topical literature.
3. This class will expose you to particular approaches and strategies for representing molecules and materials in the context of machine learning. Students will be able to formulate machine learning problems as structure-property relationships of interest to physical scientists and engineers.
4. The class will give you practical experience and knowledge with how to interface with software common for data science. Students will be able to prototype, debug, and test codes for data science applications. Students will have computational tools and practical knowledge needed to pursue more advanced topics in machine learning and understand their function.

What should/will you gain from the course?

- **Knowledge**



we will provide theoretical and practical introduction to a wide range of machine learning methods (unsupervised, supervised, classification, regression,…); this should enable you to know what you need to know/ask

we want to turn **known unknowns → known knowns**, reveal **unknown knowns → known knowns**, and convert **unknown unknowns →** at least **known unknowns**

- **Skill**

you will develop gain practical experience in working with ML algorithms and their implementation in open-source software; you will become better at programming, data manipulation, and analysis)

- **Perspective**

we will build around content, examples, and problems that are more topically relevant to the experience of a physical scientist or engineer

- **Opportunity**

*you will have the ability to explore **your** subject matter interests based on the previous three things*

What should/will you gain from the course?

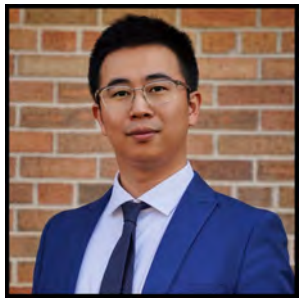
Classes	Likely Topics
Week 01	Course Introduction; Python setup/install; Math/Language Review
Week 02	Essential concepts/overview of machine learning; Breaking down Engineering Problems for ML
Week 03	Regression & Gradient Descent; Feature Scaling; Model Selection
Week 04	Model Selection; Classification & Logistic Regression; Random Forest
Week 05	Neural Networks; Backpropagation; Intro to Keras
Week 06	Molecular Featurization Techniques; Chemical Descriptors
Week 07	Fall Break
Week 08	Molecular Featurization Techniques; Chemical Descriptors
Week 09	Class Presentations; Class Presentations
Week 10	Unsupervised learning, Clustering; Dimensionality Reduction Techniques
Week 11	Explainable AI; Best Practices
Week 12	Gaussian Processes; Active Learning/Bayesian Optimization
Week 13	Debate or Special Topic Focus I ; Thanksgiving
Week 14	Special Topic Focus II ; Special Topic Focus III (overflow)
Week 15	End-of-term Presentations

Possible Focus Topics

- Advanced neural network architectures
- Generative Modeling
- Diffusion Models
- Transformers
- Large Language Models
- …

Course Logistics: Management/Discussion

Instructor: *Me* (A325)
co-Instructor: *Dr. Shengli (Bruce) Jiang* (A326)
Office hours: TBD



Format:

- In **lecture (15%)**, we will present material, perform *simple* examples/demonstrations with time for troubleshooting, engage in discussion, and occasionally have student-led presentations. I generally recommend that you go.
- For **assignments (40%)**, you will build upon, apply, and critically think about the things covered in lecture. Many problems will be presented in somewhat open-ended fashion.
- For a **midterm (15%),** we will cover an approved literature paper with baseline example
- In the **final project (30%)**, you get to delve into an approved project/problem more deeply

Platforms:

- We are planning all interactions in the near-term to be in-person, within University policy/guidelines (possible recording of lectures but don't count on it)
- We are using **Canvas** for general course management (check here for content, reading, assignments, etc.)
- We will also host a **slack channel**:
  For technical/small questions, I encourage Slack. More official correspondence should go through e-mail

CANVAS@PRINCETON

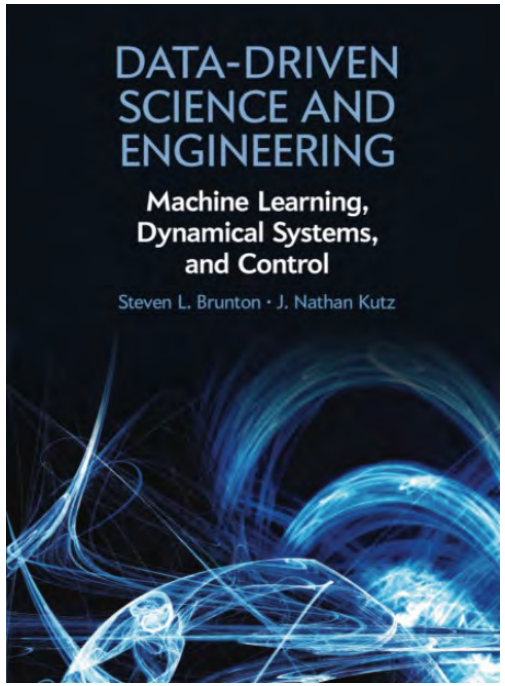
slack

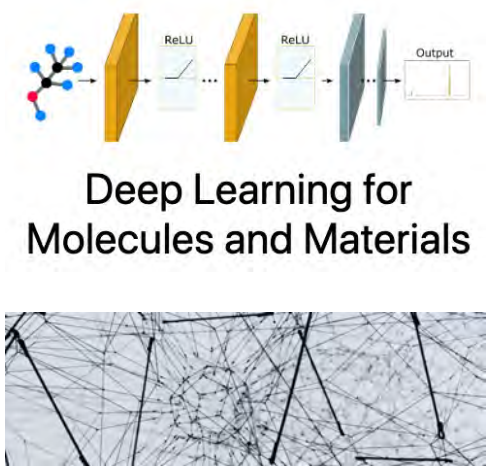
Course Logistics: Management/Discussion

Reading:

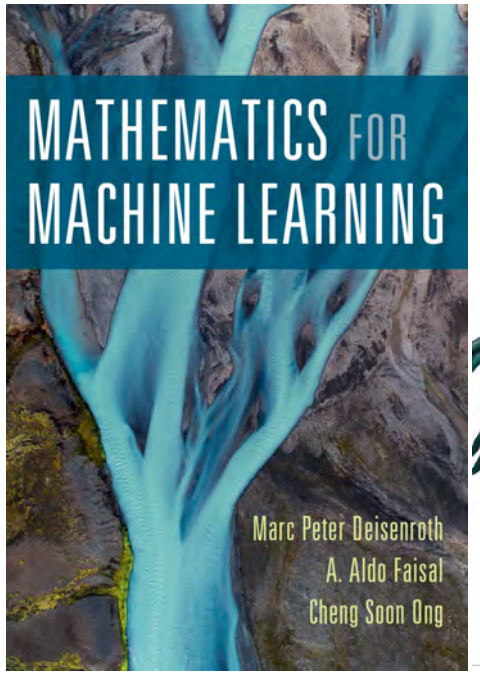
- We do not require a textbook! Selected suggested readings (resources) will be posted as will lecture notes/slides. You are encouraged to explore topics in greater detail as needed. There are myriad useful online resources, many of which are free! Doc pages, online tutorials, and Stack overflow are your friends. Nonetheless, I do think that there is value in dedicated reading of a more formal resource.

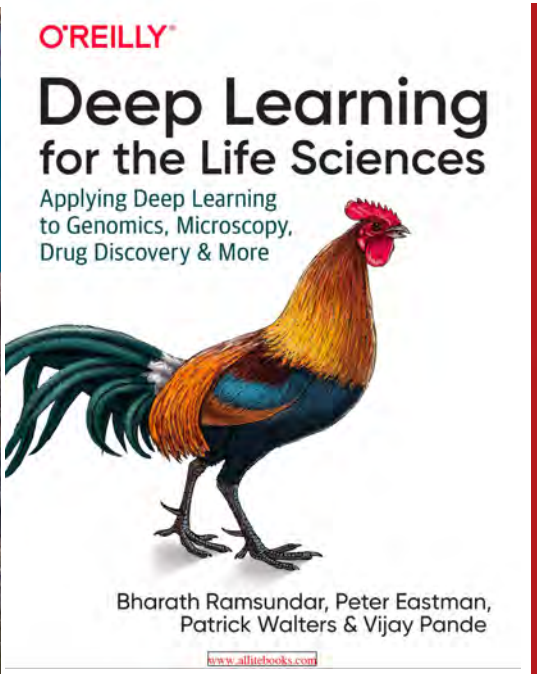
Here are some good references that we may make use of...

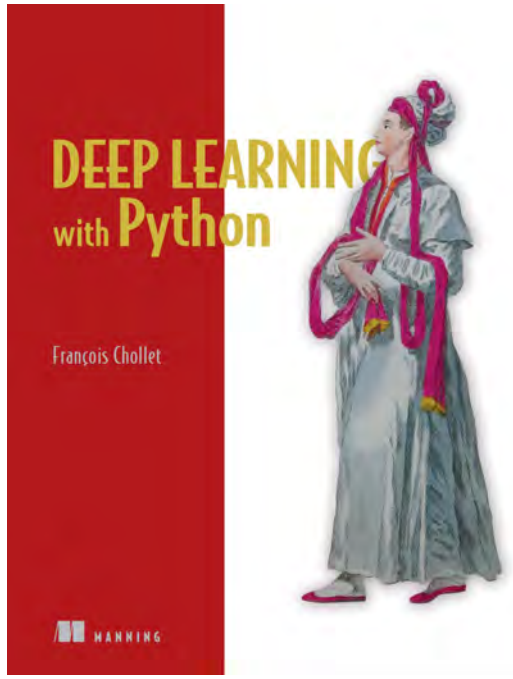




dmol.pub
Andrew D. White







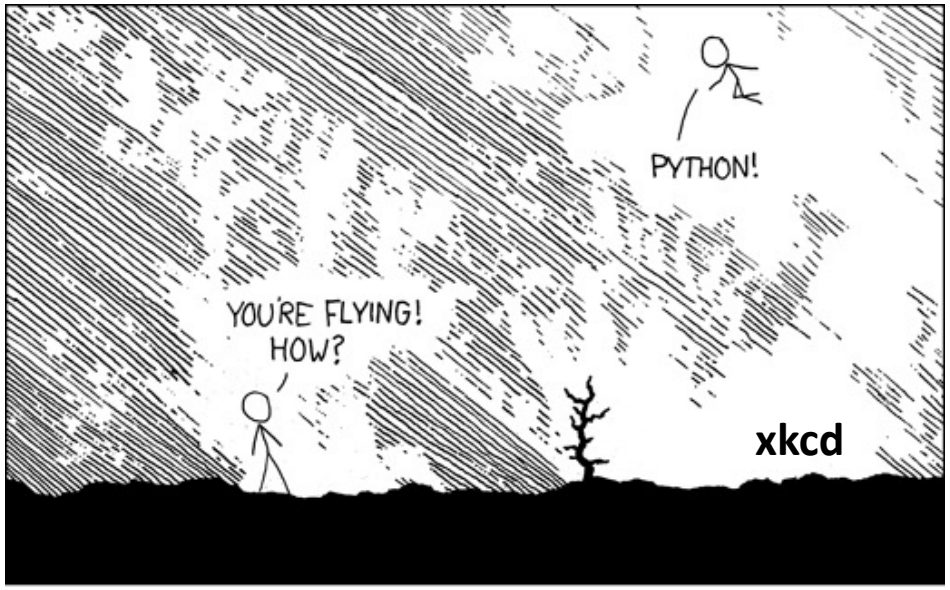
Course Logistics: Programming

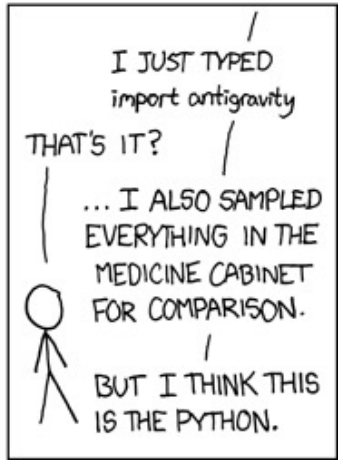
In this course, you will have to do programming to complete assignments and test what you've learned.

You can choose to work with whatever language or adopt whatever paradigm that you want, at your own risk.

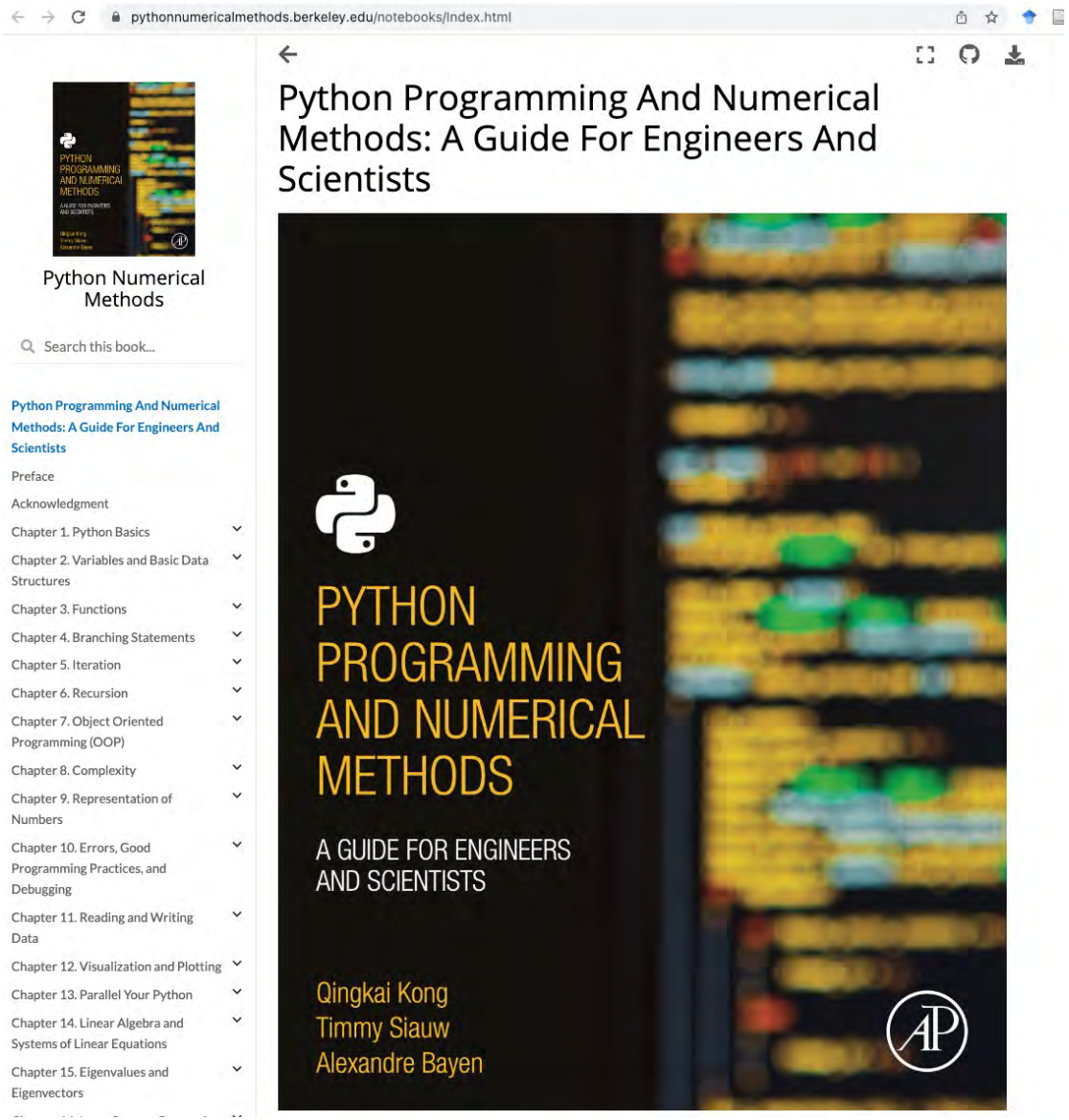
Course demonstrations and explanations will be done primarily in Python and associated libraries; I recommend that you use it

- Python has tons of functionality and support due to a large user base (we can't spend loads of time coding numerical recipes, but we can import a lot of useful functionality from, e.g., NumPy and SciPy modules
- Python is the go-to language in scientific computing, analysis, and machine learning (It is prevalent in some industrial sectors but not others)
- Perhaps relatedly, it is available anywhere you go!
- It is an interpretive/scripting language == no compilation required
- Downside is it can be a bit slow...





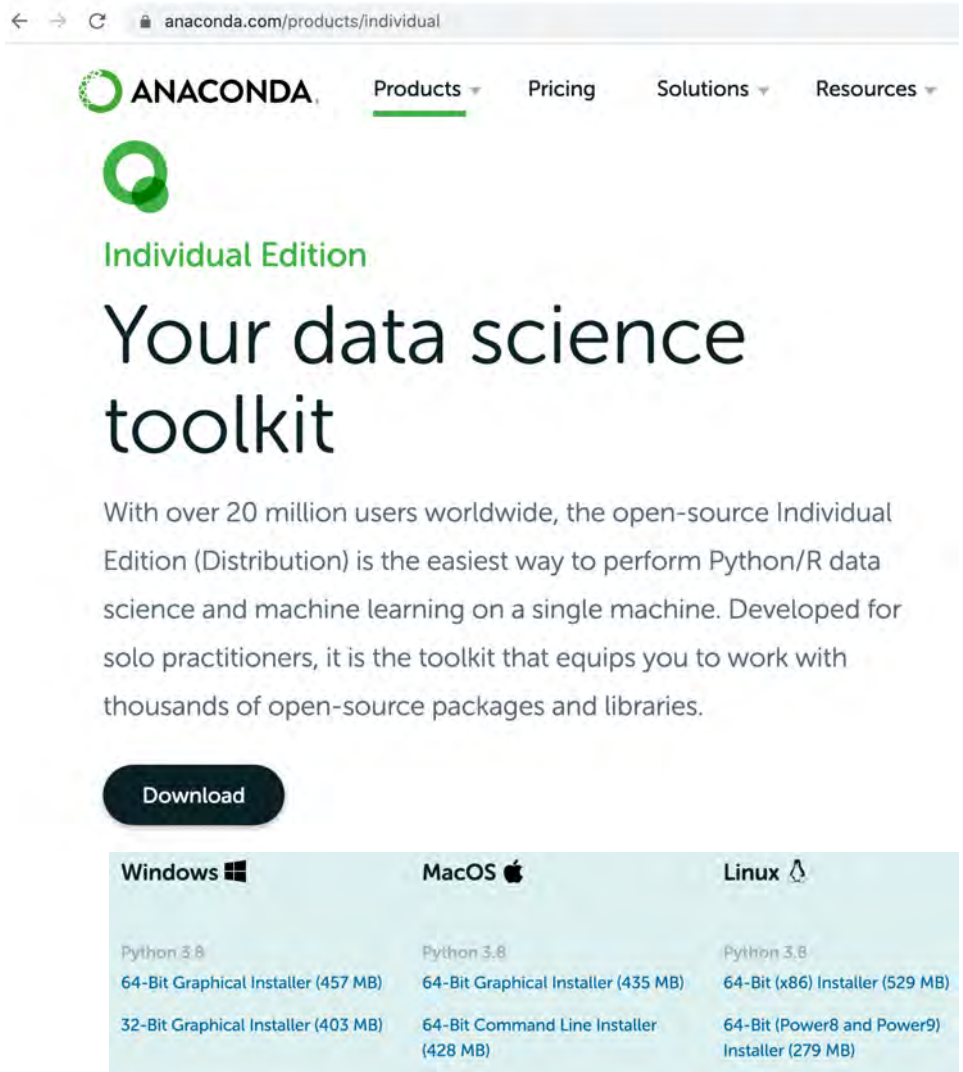
Course Logistics: Programming



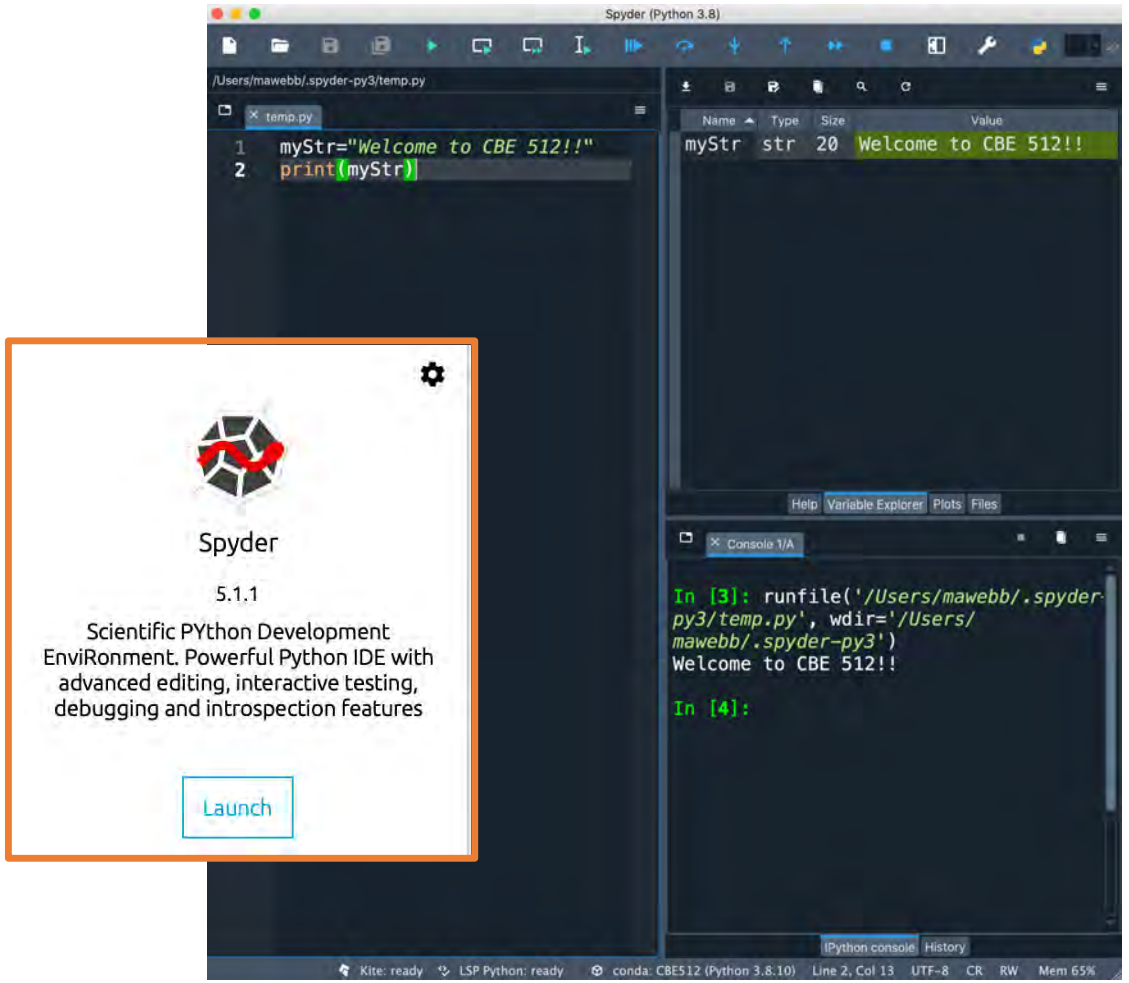
- You have to program in this class, but this is a useful skill to have! Don't be afraid and be engaged.
- There are many resources available online and some that will be provided through the course. These can help you with syntax via mimicry. Understanding may require deeper investigation.
- Use Google, StackOverflow, and **ChatGPT** liberally as needed.
- If you want a companion book to peruse, then this one is freely available online https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html
- Python changes. Don't be surprised if things that once worked no longer do. We have to be adaptive.

Course Logistics: Programming

To make things *simple,* I would pick up a python distribution from **Anaconda** to get base functionality, and then we will add things to it! (*will do this today*)

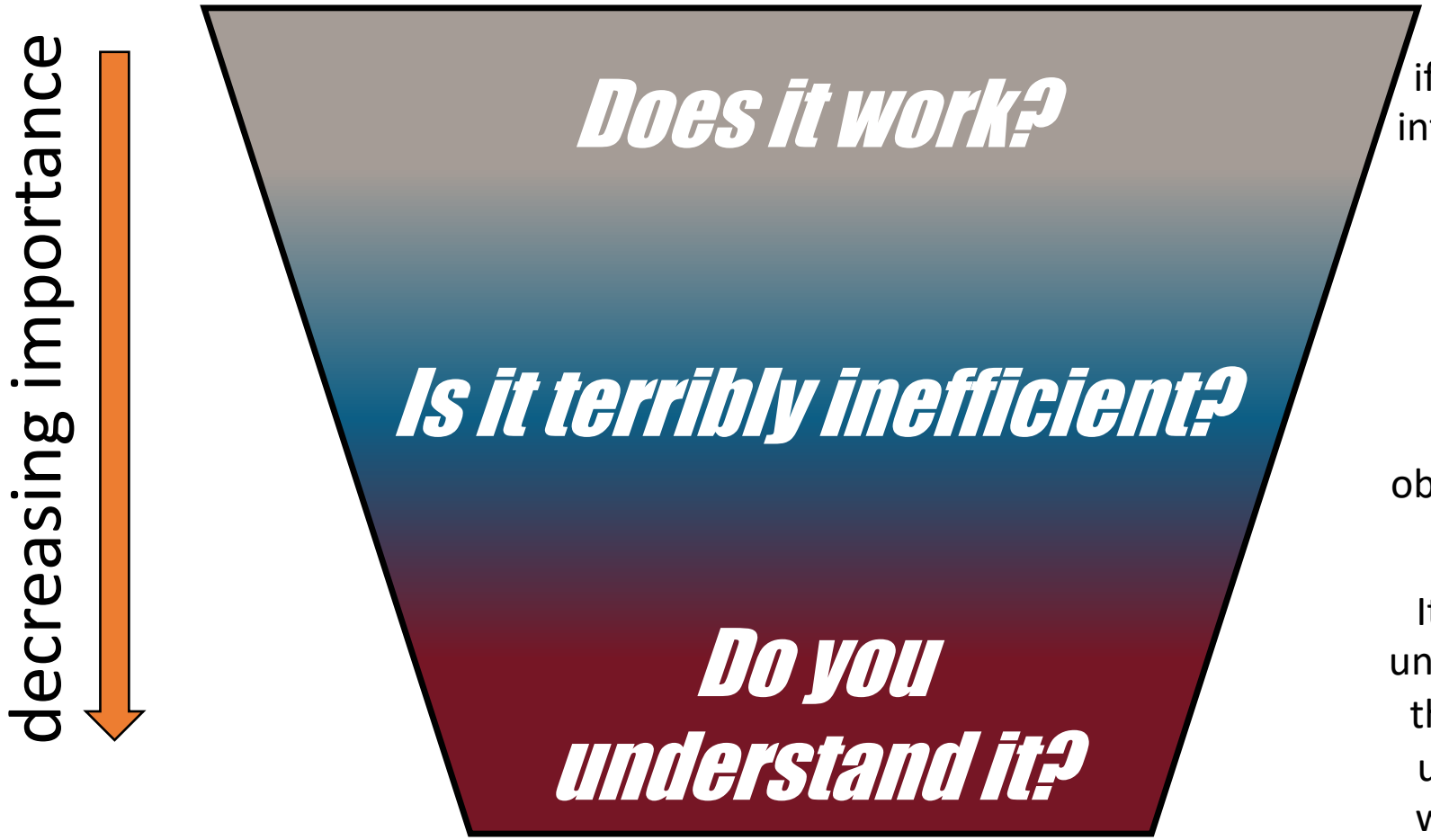


I usually code/manage everything in a terminal using vim, but **Anaconda** comes with a number of GUI editors that are pretty reasonable and will make it feel like MATLAB with immediate feedback



Course Logistics: Programming

Prof. Webb's Surprising Hierarchy of Programming Importance



if the code does not work as intended, then it is not useful.

if the code works but is so inefficient that it cannot practically accomplish the objective , then it is not useful.

It may not be worthwhile to understand every line/function that you use. You do need to understand enough to work with it and verify its validity.

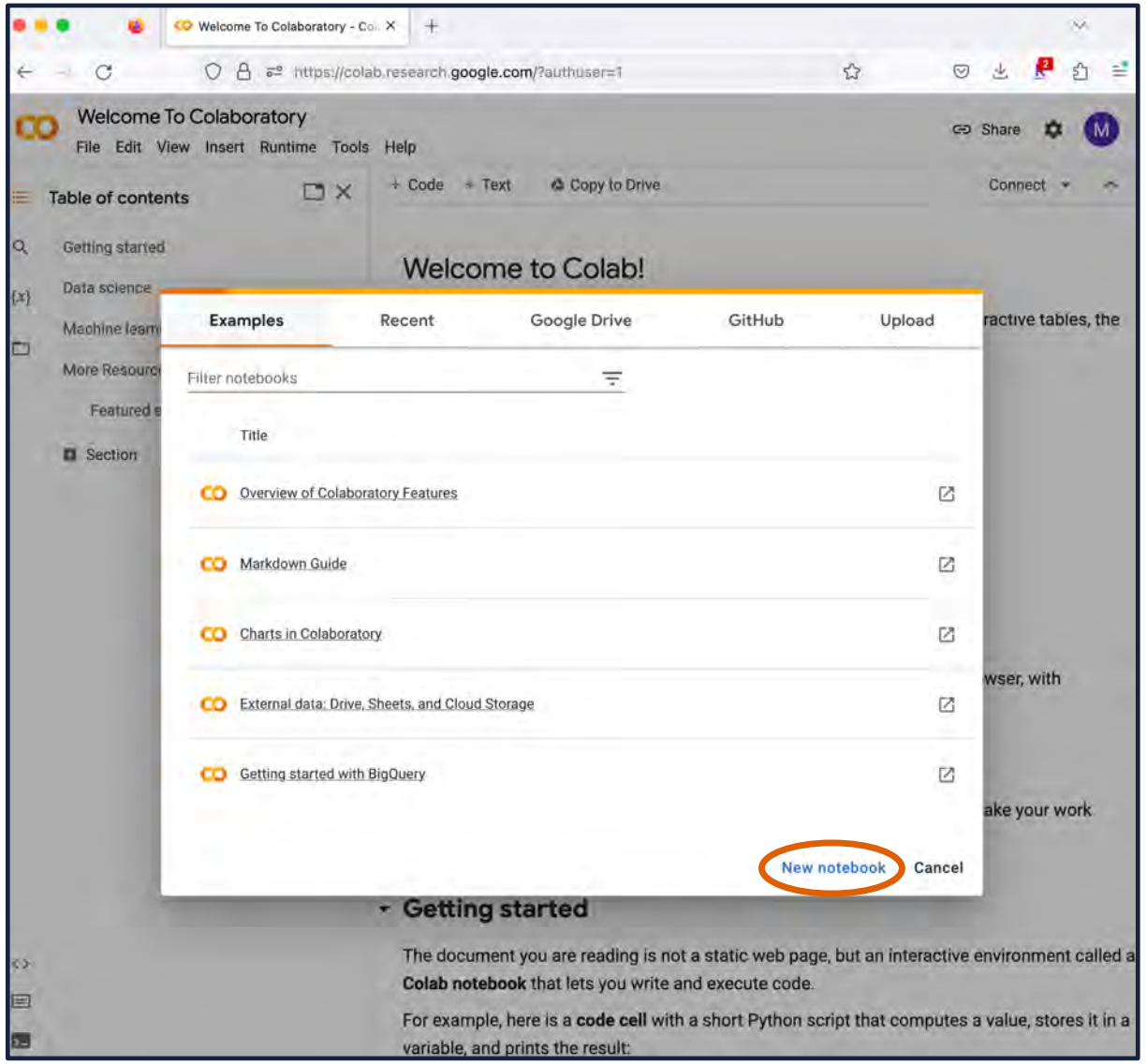
Interactive Python Time

Using Google Colab

Using Google Colab, we can obtain working python environments that functionally equip us with everything we need for the course (pretty much… we may need to tweak things here and there)

- Type https://colab.research.google.com into your URL.

- Click "New notebook"



Setting up a Python environment on your machine

If you want to set up python on your personal machine, there are a few ways to go about this. This may be preferable than google colab for certain problem sets (and if you need to work offline)

Step 1. Go to https://www.anaconda.com/products/individual to get the Anaconda Individual Edition

we will use Python 3.8 (latest version for Anaconda)

Step 2. Install it.

if you have a terminal…

Step 3. Setup a new environment (or add to your existing environment)

if you have a terminal…

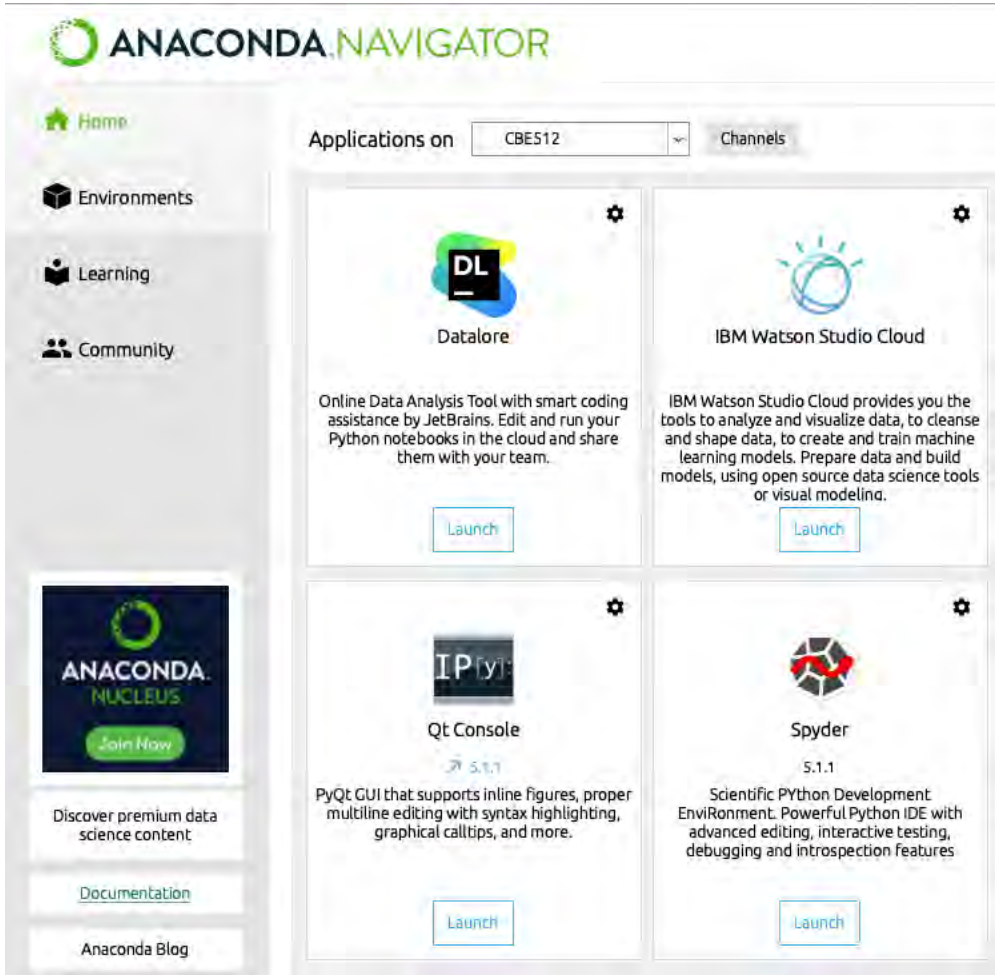
```
conda create -n CBE512 tensorflow numpy jupyter matplotlib scikit-learn pandas
openbabel rdkit hyperopt scipy seaborn pytorch torchvision -c conda-forge pytorch
```

if you do not have a terminal…

Using Anaconda Navigator

if you do not have a terminal…

Step 3a. Open the Anaconda Navigator



Step 3b. Click on **Environments** and make a new environment for the course (mine is called `CBE512')

Step 3c. Switch to your new environment. Find and install the **Spyder** application. (if you want an interactive GUI, then you can do this even if you already used the terminal to set up the environment)

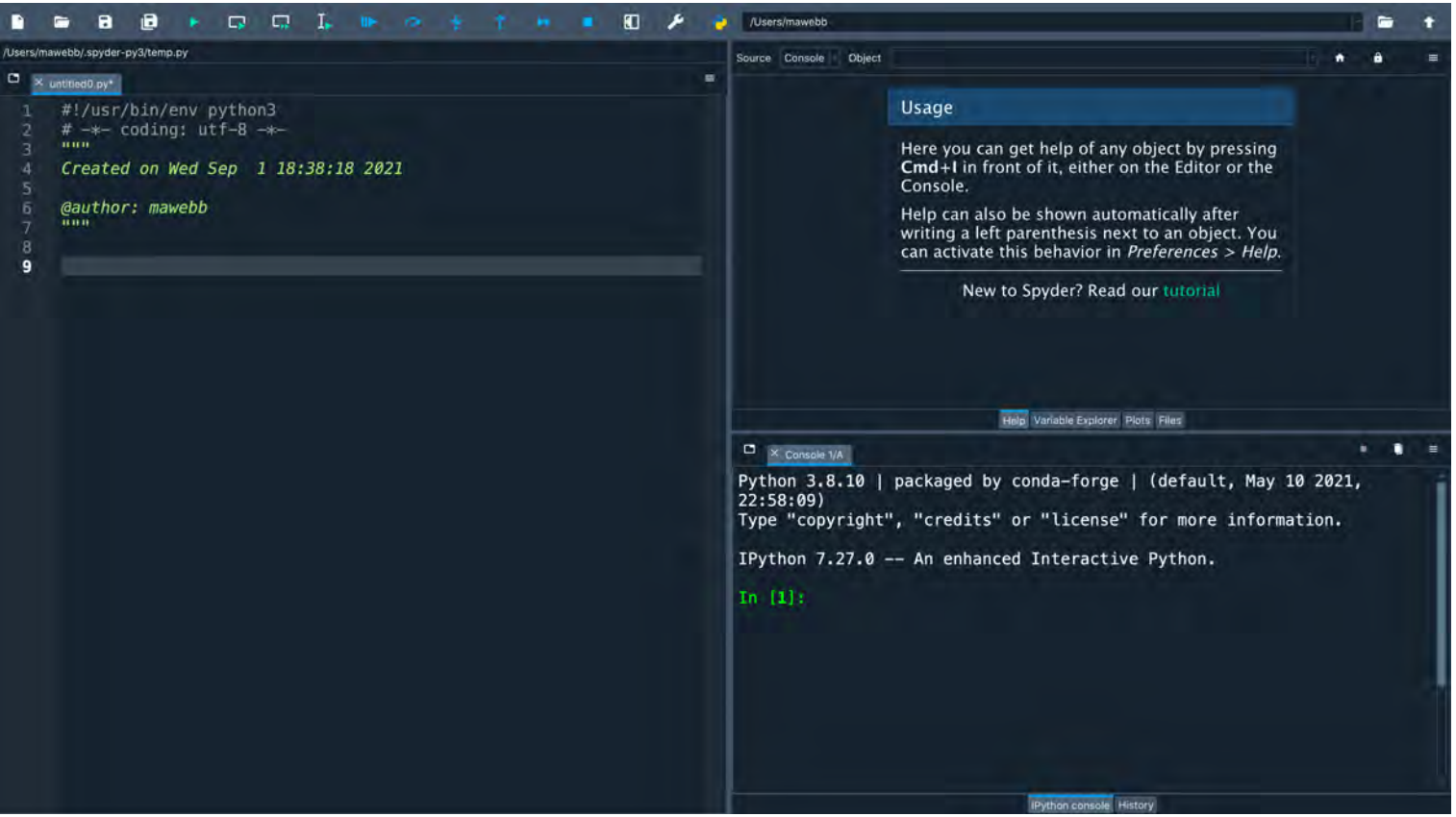
Step 3d. Launch Spyder

Using Anaconda Navigator

if you do not have a terminal…

Step 3e. Click on channels and add conda-forge (if it does not already appear) and “pytorch” if you want to have that handy.

Step 3f. From within Spyder, we can use the pseudo terminal/Console and issue the following:




```
conda install tensorflow numpy jupyter matplotlib scikit-learn pandas openbabel rdkit hyperopt scipy seaborn pytorch torchvision -c pytorch
```

Step 3g. Check to see that everything was installed with

```
conda list
```

The Philosophy of Python

```
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 as a Calculator

We say an instruction or operation is **executed** when it is resolved by the computer. An instruction is executed at the command prompt by typing it where you see the >>> symbol in the Python shell (or `In [1]:` sign in Ipython) and then pressing `Enter`. Or type it in the code cell in Jupyter notebook and pressing `Shift + Enter`. Since we will use Jupyter notebook for the rest of the book, here we will show all the examples in Ipython shell (Python shell is similar), so that you can familiar with different options.

TRY IT! Compute the sum of 1 and 2.

```
In [1]: 1 + 2
Out[1]: 3
```

An **order of operations** is a standard order of precedence that different operations have in relationship to one another. Python utilizes the same order of operations that you learned in grade school. Powers are executed before multiplication and division, which are executed before addition and subtraction. Parentheses, (), can also be used in Python to supersede the standard order of operations.

TRY IT! Compute $\frac{3*4}{(2^2+4/2)}$.

```
In [2]: (3*4)/(2**2 + 4/2)
Out[2]: 2.0
```

Python has many basic arithmetic functions like `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `exp`, `log`, `log10` and `sqrt` stored in a module (We will explain it later in this chapter) called **math**. We can import this module first to get access to these functions.

```
In [6]: import math
```

The way we use these mathematical functions is `module.function`, the inputs to them are always placed inside of parentheses that are connected to the function name. For trigonometric functions, it is useful to have the value of π available. You can call this value at any time by typing `math.pi` in the code cell. Note that the value of π is stored in Python to 16 digits.

TRY IT! Find the square root of 4.

```
In [7]: math.sqrt(4)
Out[7]: 2.0
```

TRY IT! Compute the $sin(\frac{\pi}{2})$.

```
In [8]: math.sin(math.pi/2)
Out[8]: 1.0
```

** There are many duplicate functions in other namespaces (for example, in numpy)**

Python as a Calculator

TRY IT! Compute e^{log10}.

```
In [9]: math.exp(math.log(10))
Out[9]: 10.000000000000002
```

Note that the `log` function in Python is log_e, or the natural logarithm. It is not log_{10}. If you want to use log_{10}, you need to use `math.log10`.

TIP! You can see the result above should be 10, but it is showing 10.000000000000002, this is due to Python's number approximation, which we will learn more in chapter 9.

TRY IT! Compute sum 2 + 5i

```
In [16]: 2 + 5j
Out[16]: (2+5j)
```

Note that, in Python imaginary part is using j instead of i to represent.

Another way to represent complex number in Python is to use the complex function.

```
In [17]: complex(2,5)
Out[17]: (2+5j)
```

We just learned to use Python as a calculator to deal with different data values. In Python, there are a few data types we need to know, for numerical values, int, float, and complex are the types associated with the values.

- **int**: Integers, such as 1, 2, 3, ...
- **float**: Floating-point numbers, such as 3.2, 6.4, ...
- **complex**: Complex numbers, such as 2 + 5j, 3 + 2j, ...

TIP! Using the UP ARROW in the command prompt recalls previous commands that were executed. If you accidentally type a command incorrectly, you can use the UP ARROW to recall it, and then edit it instead of retyping the entire line.

Logical Expressions and Operators

Logical expressions allow you to manage conditional code execution

In Python, a logical expression that is true will compute to the value "True". A false expression will compute to the value "False". This is a new data type we come across - **boolean**, which has the built-in values `True` and `False`. For the purpose of this book, "True" is equivalent to 1, and "False" is equivalent to 0. Distinguishing between the numbers 1 and 0 and the logical values "True" and "False" is beyond the scope of this book, but it is covered in more advanced books on computing. Logical expressions are used to pose questions to Python. For example, "3 < 4" is equivalent to, "Is 3 less than 4?" Since this statement is true, Python will compute it as 1. However, 3 > 4 is false, therefore Python will compute it as 0.

Comparison operators compare the value of two numbers, and they are used to build logical expressions. Python reserves the symbols >, >=, <, <=, ! =, ==, to denote "greater than", "greater than or equal", "less than", "less than or equal", "not equal", and "equal", respectively. Let's start with an example a = 4, b = 2, and see the following table:

TRY IT! Compute the logical expression for "Is 5 equal to 4?" and "Is 2 smaller than 3?"

```
5 == 4
```

```
False
```

```
2 < 3
```

```
True
```

Operator	Description	Example	Results
>	greather than	a > b	True
>=	greater than or equal	a >= b	True
<	less than	a < b	False
<=	"less than or equal	a <= b	False
!=	not equal	a != b	True
==	equal	a == b	False

Logical Expressions and Operators

Logical expressions allow you to manage conditional code execution

Logical operators are operations between two logical expressions that, for the sake of discussion, we call P and Q. The fundamental logical operators we will use herein are **and**, **or**, and **not**.

Operator	Description	Example	Results
and	greather than	P and Q	True if both P and Q are True False otherwise
or	greater than or equal	P or Q	True if either P or Q is True False otherwise
not	less than	not P	True if P is False False if P is True

TRY IT! Assuming P is true, use Python to determine if the expression $(P\ AND\ NOT\ (Q))\ OR\ (P\ AND\ Q)$ is always true regardless of whether or not Q is true. Logically, can you see why this is the case? First assume Q is true:

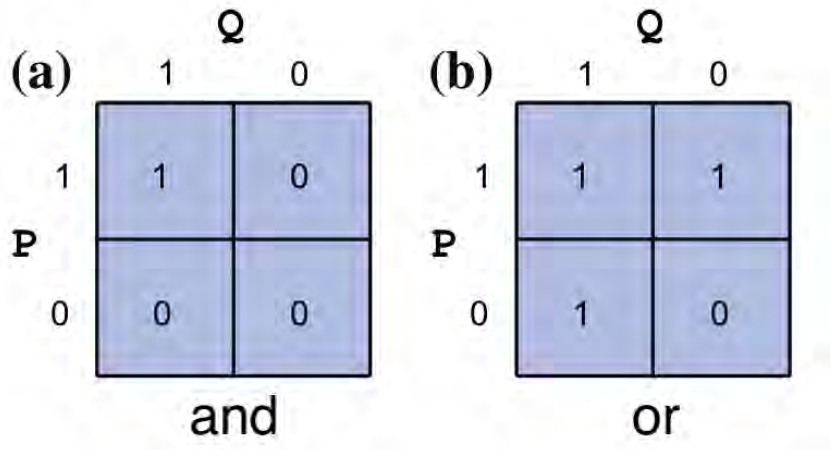
```
(1 and not 1) or (1 and 1)
```

```
1
```

Now assume Q is false

```
(1 and not 0) or (1 and 0)
```

```
True
```



Logical Expressions and Operators

Logical expressions allow you to manage conditional code execution

TRY IT! Compute (1 + 3)>(2 + 5)

```
1 + 3 > 2 + 5
```

```
False
```

TIP! Even when the order of operations is known, it is usually helpful for you and those reading your code to use parentheses to make your intentions clearer. In the preceding example (1 + 3) > (2 + 5) is clearer.

WARNING! In Python's implementation of logic, 1 is used to denote true and 0 for false. However, 1 and 0 are still numbers. Therefore, Python will allow abuses such as: (3 > 2) + (5 > 4), which will resolve to 2.

```
(3 > 2) + (5 > 4)
```

```
2
```

Storing information and variables

When programming, it is useful to be able to store information in variables. A **variable** is a string of characters and numbers associated with a piece of information. The **assignment operator**, denoted by the "=" symbol, is the operator that is used to assign values to variables in Python. The line `x=1` takes the known value, 1, and **assigns** that value to the variable with name "x". After executing this line, this number will be stored into this variable. Until the value is changed or the variable deleted, the character x behaves like the value 1.

```
x = 1
x
```

```
1
```

TRY IT! Assign the value 2 to the variable y. Multiply y by 3 to show that it behaves like the value 2.

```
y = 2
y
```

```
2
```

```
y*3
```

```
6
```

- If you have coded in other languages, you might be disturbed that you don't have to declare types in your variables!
- Python intuits the types which can be a great timesaver or end in disaster if you are not careful.
- I generally recommend that you initialize data structures anyway if they will be of known size. This tends to be more efficient than allocating on-the-fly

Storing information and variables

Note that the equal sign in programming is *not* the same as a truth statement in mathematics. In math, the statement *x = 2* declares the universal truth within the given framework, *x is 2*. In programming, the statement x=2 means a known value is being associated with a variable name, *store* 2 in x. Although it is perfectly valid to say *1 = x* in mathematics, assignments in Python always go *left*: meaning the value to the right of the equal sign is assigned to the variable on the left of the equal sign. Therefore, *1=x* will generate an error in Python. The assignment operator is always last in the order of operations relative to mathematical, logical, and comparison operators.

There are some restrictions on the names variables can take. Variables can only contain alphanumeric characters (letters and numbers) as well as underscores. However, the first character of a variable name must be a letter or underscores. Spaces within a variable name are not permitted, and the variable names are case-sensitive (e.g., *x* and *X* will be considered different variables).

Strings in Python

A string is a sequence of characters, such as "Hello World" we saw in chapter 1. Strings are surrounded by either single or double quotation marks. We could use *print* function to output the strings to the screen.

TRY IT! Print "I love Python!" to the screen.

```
print("I love Python!")
```

TRY IT! Assign the character "S" to the variable with name s. Assign the string "Hello World" to the variable w. Verify that s and w have the type string using the *type* function.

```
s = "S"
w = "Hello World"
```

A string is an array of characters, therfore it has length to indicate the size of the string. For example, we could check the size of the string by using the built-in function *len*.

```
len(w)
```

```
11
```

Note that a blank space, " ", between "Hello" and "World" is also a type *str*. Any symbol can be a char, even the ones that have been reserved for operators. Note that as a *str*, they do not perform the same function. Although they look the same, Python interprets them completely differently.

Indexing, Slicing, and Manipulating Strings

Strings also have indexes to indicate the location of each character, so that we could easily find out some character. The index of the position start with 0, as shown in the following picture.

Character	H	e	l	l	o		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10

We could get access to any character by using a bracket and the index of the position. For example, if we want to get the character 'W', then we need to do:

```
w[6]
```

```
'W'
```

We could also select a sequence as well using string slicing. For example, if we want to get the "World", we could do the following command.

```
w[6:11]
```

```
'World'
```

[6:11] means the start position is from index 6 and the end position is index 10. For Python string slicing range, the upper-bound is exclusive, which means that [6:11] is actually to slice the characters from 6 -> 10. The syntax for slicing in Python is [start:end:step], the 3rd one - step is optional.

You can ignore the end position if you want to slice to the end of the string. For example, the following command is the same as the above one:

```
w[6:]
```

```
'World'
```

TRY IT! Slice the "Wor" within the word "World".

```
w[6:-2]
```

```
'Wor'
```

TRY IT! Retrieve every other character in the variable w

```
w[::2]
```

```
'HloWrd'
```

In Python, string as an object that has various methods that could be used to manipulate it (we will talk more about object-oriented programming later). They way to get access to the various methods is to use this patter "string.method_name".

TRY IT! Turn the variable w to upper case.

```
w.upper()
```

```
'HELLO WORLD'
```

TRY IT! Count the number of occurrence for letter "l" in w.

```
w.count("l")
```

```
3
```

TRY IT! Replace the "World" in variable w to "Berkeley".

```
w.replace("World", "Berkeley")
```

```
'Hello Berkeley'
```

Indexing, Slicing, and Manipulating Strings

There are lots of ways to provide formatted print outputs; learn some!

```
greeting = "hello"
dept   = "CBE"
number = 512
print("%s %s %d"%(greeting,dept,number))
print("{:>10s} {:s} {:>5d}".format(greeting,dept,number))
print(f'{greeting} {dept} {number}')
print(greeting + " " + dept + " " + str(number))
print(greeting,dept,number)
```

```
hello CBE 512
     hello CBE   512
hello CBE 512
hello CBE 512
hello CBE 512
```

Python Lists

We saw strings in the previous section that could hold a sequence of characters. Now, let's see a more versatile sequential data structure in Python - Lists. The way to define it is to use a pair of brackets [ ], and the elements within it are separated by commas. A list could hold any type of data: numerical, or strings or other types. For example:

```
list_1 = [1, 2, 3]
list_1
```

```
[1, 2, 3]
```

```
list_2 = ['Hello', 'World']
list_2
```

```
['Hello', 'World']
```

We could put mixed types in the list as well

```
list_3 = [1, 2, 3, 'Apple', 'orange']
list_3
```

```
[1, 2, 3, 'Apple', 'orange']
```

We can also nest the lists, for example:

```
list_4 = [list_1, list_2]
list_4
```

```
[[1, 2, 3], ['Hello', 'World']]
```

List	1	2	3	Apple	Orange
Index	0	1	2	3	4

TRY IT! Get the 3rd element in list_3

```
list_3[2]
```

```
3
```

TRY IT! Get the first 3 elements in list_3

```
list_3[:3]
```

```
[1, 2, 3]
```

TRY IT! Get the last element in list_3

```
list_3[-1]
```

```
'orange'
```

Similarly, we could get the length of the list by using the *len* function.

```
len(list_3)
```

```
5
```

We can also concatenate two lists by simply using a + sign.

Python Lists

We can also concatenate two lists by simply using a + sign.

TRY IT! Add list_1 and list_2 to one list.

```
list_1 + list_2
```

```
[1, 2, 3, 'Hello', 'World']
```

New items could be added to an existing list by using the *append* method from the list.

```
list_1.append(4)
list_1
```

```
[1, 2, 3, 4]
```

Note! The *append* function operate on the list itself as shown in the above example, 4 is added to the list. But in the list_1 + list_2 example, list_1 and list_2 won't change. You can check list_2 to verify this.

We could also insert or remove element from the list by using the methods *insert* and *remove*, but they are also operating on the list directly.

```
list_1.insert(2,'center')
list_1
```

```
[1, 2, 'center', 3, 4]
```

Note! Using the *remove* method will only remove the first occurrence of the item (read the documentation of the method). There is another way to delete an item by using its index - function *del*.

```
del list_1[2]
list_1
```

```
[1, 2, 3, 4]
```

You can also predefine an empty list and append to it, but be wary!

Python Tuples

Let's learn one more different sequence data structure in Python - Tuples. It is usually defined by using a pair of parentheses ( ) , and its elements are separated by commas. For example:

```
tuple_1 = (1, 2, 3, 2)
tuple_1
```

```
(1, 2, 3, 2)
```

You may ask, what's the difference between lists and tuples? If they are similar to each other, why do we need another sequence data structure?

Well, tuples are created for a reason. From the Python documentation:

> Though tuples may seem similar to lists, they are often used in different situations and for different purposes. Tuples are **immutable**, and usually contain a **heterogeneous** sequence of elements that are accessed via unpacking (see later in this section) or indexing (or even by attribute in the case of named tuples). Lists are **mutable**, and their elements are usually **homogeneous** and are accessed by iterating over the list.

TRY IT! Get the length of tuple_1.

```
len(tuple_1)
```

```
4
```

TRY IT! Get the elements from index 1 to 3 for tuple_1.

```
tuple_1[1:4]
```

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

TRY IT! Count the occurrence for number 2 in tuple_1.

```
tuple_1.count(2)
```

```
2
```

What does it mean by immutable? It means the elements in the tuple, once defined, they can not be changed. But elements in a list can be changed without any problem. For example:

```
list_1 = [1, 2, 3]
list_1[2] = 1
list_1
```

```
[1, 2, 1]
```

```
tuple_1[2] = 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-76fb6b169c14> in <module>()
----> 1 tuple_1[2] = 1

TypeError: 'tuple' object does not support item assignment
```

Python Dictionaries

We introduced several sequential data type in the previous sections. Now we will introduce you a new and useful type - Dictionaries. It is a mapping type, which makes it a totally different type than the ones we talked before. Instead of using a sequence of numbers to index the elements (such as lists or tuples), dictionaries are indexed by keys, which could be a string, number or even tuple (but not list). A dictionary is a key-value pairs, and each key maps to a corresponding value. It is defined by using a pair of braces { }, while the elements are a list of comma separated key:value pairs (note the key:value pair is separated by the colon, with key at front and value at the end).

```
dict_1 = {'apple':3, 'oragne':4, 'pear':2}
dict_1
```

```
{'apple': 3, 'oragne': 4, 'pear': 2}
```

Within a dictionary, elements are stored without order, therefore, you can not access a dictionary based on a sequence of index numbers. To get access to a dictionary, we need to use the key of the element - dictionary[key].

TRY IT! Get the element 'apple' from dict_1.

```
dict_1['apple']
```

```
3
```

TRY IT! Define an empty dictionary named *school_dict* and add value "UC Berkeley":"USA".

```
school_dict = {}
school_dict['UC Berkeley'] = 'USA'
school_dict
```

```
{'UC Berkeley': 'USA'}
```

TRY IT! Add another element "Oxford":"UK" to *school_dict*.

```
school_dict['Oxford'] = 'UK'
school_dict
```

```
{'UC Berkeley': 'USA', 'Oxford': 'UK'}
```

We could get all the keys in a dictionary by using the *keys* method, or all the values by using the method *values*.

TRY IT Get all the keys and values from dict_1.

```
dict_1.keys()
```

```
dict_keys(['apple', 'oragne', 'pear'])
```

```
dict_1.values()
```

```
dict_values([3, 4, 2])
```

We could also get the size of a dictionary by using the *len* function.

```
len(dict_1)
```

```
3
```

We could also check if an element belong to a dictionary using the operator *in*.

TRY IT! Determine if "UC Berkeley" is in *school_dict*.

```
"UC Berkeley" in school_dict
```

```
True
```

TRY IT! Determine whether "Harvard" is not in *school_dict*.

```
"Harvard" not in school_dict
```

```
True
```

Python through numpy: access to arrays

NumPy is important in scientific computing, it is coded both in Python and C (for speed). On its website, a few important features for Numpy is listed:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

In order to use Numpy module, we need to import it first. A conventional way to import it is to use "np" as a shortened name.

```
import numpy as np
```

WARNING! Of course, you could call it any name, but conventionally, "np" is accepted by the whole community and it is a good practice to use it for obvious purposes.

Writing functions in Python

Programming often requires repeating a set of tasks over and over again. For example, the *math.sin* function in Python is a set of tasks (i.e., mathematical operations) that computes an approximation for *sin (x)*. Rather than having to retype or copy these instructions every time you want to use the *sin* function, it is useful to store this sequence of instruction as a function that you can call over and over again.

Writing your own functions is the focus of this chapter, and it is the single most powerful use of computer programming. By the end of this chapter, you should be able to declare, write, store, and call your own functions.

We can define our own functions. A function can be specified in several ways. Here we will introduce the most common way to define a function which can be specified using the keyword *def*, as showing in the following:

```
def function_name(argument_1, argument_2, ...):
    '''
    Descriptive String
    '''
    # comments about the statements
    function_statements

    return output_parameters (optional)
```

Writing functions in Python

TRY IT! Define a function named *my_adder* to take in 3 numbers and sum them.

```
def my_adder(a, b, c):
    """
    function to sum the 3 numbers
    Input: 3 numbers a, b, c
    Output: the sum of a, b, and c
    author:
    date:
    """

    # this is the summation
    out = a + b + c

    return out
```

WARNING! If you don't indent you code for defining function, you will get an **IndentationError**.

TRY IT! Use your function my_adder to compute the sum of a few numbers. Verify that the result is correct. Try calling the help function on my_adder.

```
d = my_adder(1, 2, 3)
d
```

TRY IT! Use the string '1' as one of the input arguments to *my_adder*. Also use a list as one of the input arguments to *my_adder*.

```
d = my_adder('1', 2, 3)
```

EXAMPLE: Poor representation of my_adder.

```
def abc(a, s2, d):
    z = a + s2
    z = z + d
    x = z
    return x
```

Don't write functions with useless variable names that make it impossible for other people (and you, two weeks from now) to understand your code!

TIP! Type 4 white spaces is one level of indentation, you can have deeper level indentation when you have nested function or if-statement (you will see this in next chapter). Also, sometimes you need to indent or un-indent a block of code. You can do this by first select all the lines in the code block, and press *Tab* and *Shift+Tab* to increase or decrease one level of indentation.

Python uses indentation prominently – it is kind of like Python's version of the semicolon in C/C++.

Writing functions in Python: multiple outputs

Python functions can have multiple output parameters. When calling a function with multiple output parameters, you can place the multiple variables you want assigned separated by commas. The function essentially will return the multiple result parameters in a tuple, therefore, you could unpack the returned tuple. Consider the following function (note that it has multiple output parameters):

```
def my_trig_sum(a, b):
    """
    function to demo return multiple
    author
    date
    """
    out1 = np.sin(a) + np.cos(b)
    out2 = np.sin(b) + np.cos(a)
    return out1, out2, [out1, out2]
```

TRY IT! Compute the function *my_trig_sum* for *a=2* and *b=3*. Assign the first output parameter to the variable c, the second output parameter to the variable *d*, and the third parameter to the variable *e*.

```
c, d, e = my_trig_sum(2, 3)
print(f"c ={c}, d={d}, e={e}")
```

```
c =-0.0806950697747637, d=-0.2750268284872752, e=[-0.0806950697747637, -0.2750268284872752]
```

You can also pack things into a dictionary and also understand some elements of function scope; this may help improve tidiness, efficiency, and extensibility of code

Writing loops in Python

A **for-loop** is a set of instructions that is repeated, or iterated, for every value in a sequence. Sometimes for-loops are referred to as **definite loops** because they have a predefined begin and end as bounded by the sequence.

The general syntax of a for-loop block is as follows.

CONSTRUCTION: For-loop

```
for looping variable in sequence:
    code block
```

TRY IT! What is the sum of every integer from 1 to 3?

```
n = 0
for i in range(1, 4):
    n = n + i

print(n)
```

```
6
```

EXAMPLE: Given a list of integers, *a*, add all the elements of *a*.

```
s = 0
a = [2, 3, 1, 3, 3]
for i in a:
    s += i # note this is equivalent to s = s + i

print(s)
```

```
12
```

WHAT IS HAPPENING?

1. First, the function *range(1, 4)* is generating a list of numbers beginning at 1 and ending at 3. Check the description of the function *range* and get familiar with how to use it. In a very simple form, it is *range(start, stop, step)*, and the *step* is optional with 1 as the default.
2. The variable *n* is assigned the value 0.
3. The variable *i* is assigned the value 1.
4. The variable *n* is assigned the value *n* + *i* (0 + 1 = 1).
5. The variable *i* is assigned the value 2.
6. The variable *n* is assigned the value *n* + *i* (1 + 2 = 3).
7. The variable *i* is assigned the value 3.
8. The variable *n* is assigned the value *n* + *i* (3 + 3 = 6).
9. With no more values to assign in the list, the for-loop is terminated with *n* = 6.

Note that, we could assign two different looping variables at the same time. There are other cases that we could do things similarly. For example, if we have two lists with same length, and we want to loop through them, we could do as the following example using the *zip* function:

```
a = ["One", "Two", "Three"]
b = [1, 2, 3]

for i, j in zip(a, b):
    print(i, j)
```

```
One 1
Two 2
Three 3
```

I/O in Python

To work with text files, we need to use *open* function which returns a *file object*. It is commonly used with two arguments:

```
f = open(filename, mode)
```

f is the returned file object. The filename is a string where the location of the file you want to open, and the *mode* is another string containing a few characters describing the way in which the file will be used, the common modes are:

- 'r', this is the default mode, which opens a file for reading
- 'w', this mode opens a file for writing, if the file does not exist, it creates a new file.
- 'a', open a file in append mode, append data to end of file. If the file does not exist, it creates a new file.
- 'b', open a file in binary mode.
- 'r+', open a file (do not create) for reading and writing.
- 'w+', open or create a file for writing and reading, discard existing contents.
- 'a+', open or create file for reading and writing, and append data to end of file.

TRY IT! Create a text file called *test.txt* and write a couple lines in it.

```
f = open('test.txt', 'w')
for i in range(5):
    f.write(f"This is line {i}\n")

f.close()
```

We could see the code above that we first opened a file object *f* with the file name 'test.txt'. We used "w+" for the mode, that indicates write We then write 5 lines (note the newline '\n' at the end of the string), and then we close the file object. We could see the content of the file in the following figure.

We could read a file from disk and store all the contents to a variable. Let's read in the *test.txt* file we created above and store all the contents in the file to a variable *content*.

```
f = open('./test.txt', 'r')
content = f.read()
f.close()
print(content)
```

```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is another line
```

Plotting with Python/Matplotlib

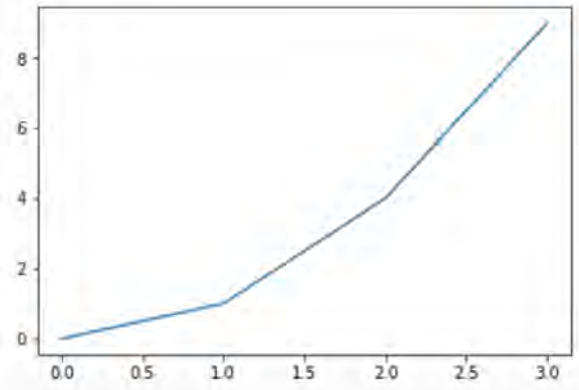
Python is very useful for making plots. Below is just some simple stuff, but it is really powerful!

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib notebook
```

The basic plotting function is *plot(x,y)*. The *plot* function takes in two lists/arrays, x and y, and produces a visual display of the respective points in x and y.

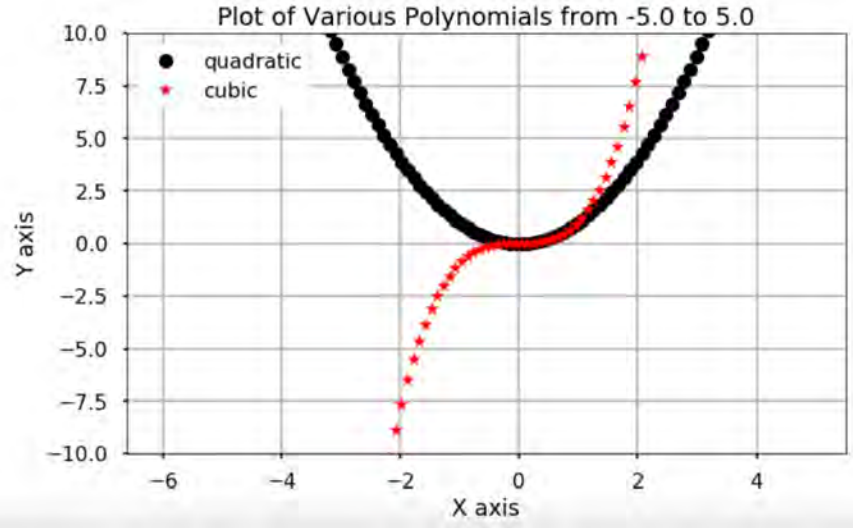
TRY IT! Given the lists x = [0, 1, 2, 3] and y = [0, 1, 4, 9], use the plot function to produce a plot of x versus y.

```
x = [0, 1, 2, 3]
y = [0, 1, 4, 9]
plt.plot(x, y)
plt.show()
```




```
plt.figure(figsize = (10,6))

x = np.linspace(-5,5,100)
plt.plot(x, x**2, 'ko', label = 'quadratic')
plt.plot(x, x**3, 'r*', label = 'cubic')
plt.title(f'Plot of Various Polynomials from {x[0]} to {x[-1]}')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend(loc = 2)
plt.xlim(-6,6)
plt.ylim(-10,10)
plt.grid()
plt.show()
```



Conditional statements in Python

CONSTRUCTION: Extended If-Else Statement Syntax

```
if logical expression P:
    code block 1
elif logical expression Q:
    code block 2
elif logical expression R:
    code block 3
else:
    code block 4
```

TRY IT! Write a function *my_thermo_stat(temp, desired_temp)*. The return value of the function should be the string *'Heat'* if temp is less than desired_temp minus 5 degrees, *'AC'* if temp is more than the desired_temp plus 5, and *'off'* otherwise.

```
def my_thermo_stat(temp, desired_temp):
    """
    Changes the status of the thermostat based on
    temperature and desired temperature
    author
    date
    :type temp: Int
    :type desiredTemp: Int
    :rtype: String
    """
    if temp < desired_temp - 5:
        status = 'Heat'
    elif temp > desired_temp + 5:
        status = 'AC'
    else:
        status = 'off'
    return status
```

```
status = my_thermo_stat(65,75)
print(status)
```

```
Heat
```

```
status = my_thermo_stat(75,65)
print(status)
```

```
AC
```

```
status = my_thermo_stat(65,63)
print(status)
```

```
off
```