

Jupyter Notebooks and Python

Dr. Thomas Robey

Felina Rivera Calzadillas

Supercomputing Challenge Kickoff

September 30, 2023

Introduction

Jupyter Notebooks can contain computer code, equations, text and images. The computer code can be run right from the notebook. Over 40 computer languages are supported. You can try different things out and document what you are doing so that you can go back and find things later. You can run your model multiple times and put the results in your notebook. Writing a report or presentation is much easier by just going back to your notebook. Notebooks can be shared.

Tip: As you write and update your computer code get in the habit of cutting and pasting it into a notebook. Add the date and the reason for the change. Even if it is a code snippet that cannot be run on its own, having a record of your changes allow going back to previous versions if you run into problems. If you do this properly then it will give you much more freedom to try different things without worrying about breaking things.

Google Colab

Google Colab is Jupyter Notebooks hosted in the cloud. We will be using Google Colab for sessions in the experienced track since it makes it easy for students to participate remotely. It also allows students to keep all their work from the Kickoff to refer to later. Google Colab makes it easy to share code and other information with team members, mentors and others.

To use Google Colab you will need a Google account. Most students will already have a Google account but if you do not already have an account:

Creating a Google Account

Accessing Google Colab

There are a couple of ways to access Google Colab. The first is by going to:

<https://colab.research.google.com/notebooks/intro.ipynb>

Then click the blue button at the top right to sign into your Google account. Then you should be able to open notebook in a new tab by going to File -> New notebook.

or

Sign into your Google account and paste in the link: <http://colab.research.google.com/#create=true>

or

add Google Colab to your apps

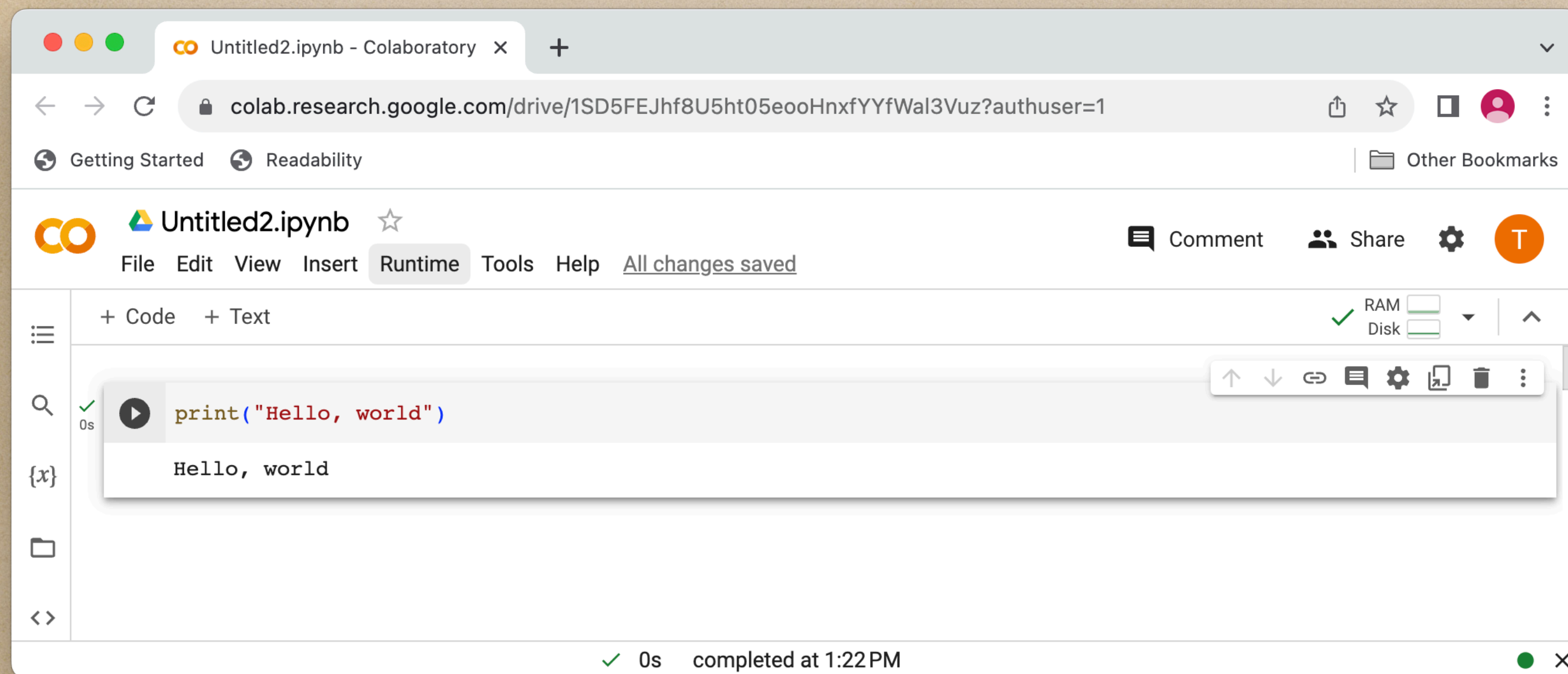
- Click on the Google apps icon on the upper right next to your account icon.
- Click on Google Drive
- On the upper left click on the "+ New" button
- At the bottom select More
- Connect more apps
- Type colab into the search box
- Select Colaboratory

Now clicking on "+ New" and More then you can select Google Colaboratory to create a new notebook.

Python

Use your mouse to click in the box in your browser. Type in

- `print("Hello, world")`



The screenshot shows a web browser window with a single tab titled "Untitled2.ipynb - Colaboratory". The address bar displays the URL `colab.research.google.com/drive/1SD5FEJhf8U5ht05eooHnxfYYfWal3Vuz?authuser=1`. The notebook interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", along with a status indicator "All changes saved". A toolbar at the top right shows "Comment", "Share", and a user profile icon. The main workspace contains a code cell with the Python code `print("Hello, world")`. The cell's execution history shows a successful run with a green checkmark, a play button icon, and a duration of "0s". The output of the code is displayed as "Hello, world". A status bar at the bottom of the notebook indicates "0s completed at 1:22 PM".

Click on the run button to the left of the cell to run the code or go up to the menu under Runtime. Your Python code should execute and the output displayed. Congratulations, you have just run your code!

Jupyter Notebooks for learning Python 3: <https://github.com/jerry-git/learn-python3>

Comments begin with '#'.

```
#This is a comment
```

Try adding a comment to your code. Comments can start at the beginning of a line or at the end of a line.

Strings

The screenshot shows a web browser window with a single tab titled "Untitled2.ipynb - Colaboratory". The address bar shows the URL: `colab.research.google.com/drive/1SD5FEJhf8U5ht05eooHnxfYYfWal3Vuz?authuser=1#scrollTo=VVCzo...`. The notebook interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", and a "Saving..." status. On the right, there are icons for "Comment", "Share", and a user profile. The main workspace has a toolbar with "+ Code" and "+ Text" options, and a RAM/Disk usage indicator. A code cell is active, containing the following Python code:

```
string = "Hello, world"
print(string)
string = string.replace("Hello", "Goodbye")
print(string)
```

The output of the code cell is displayed below the code:

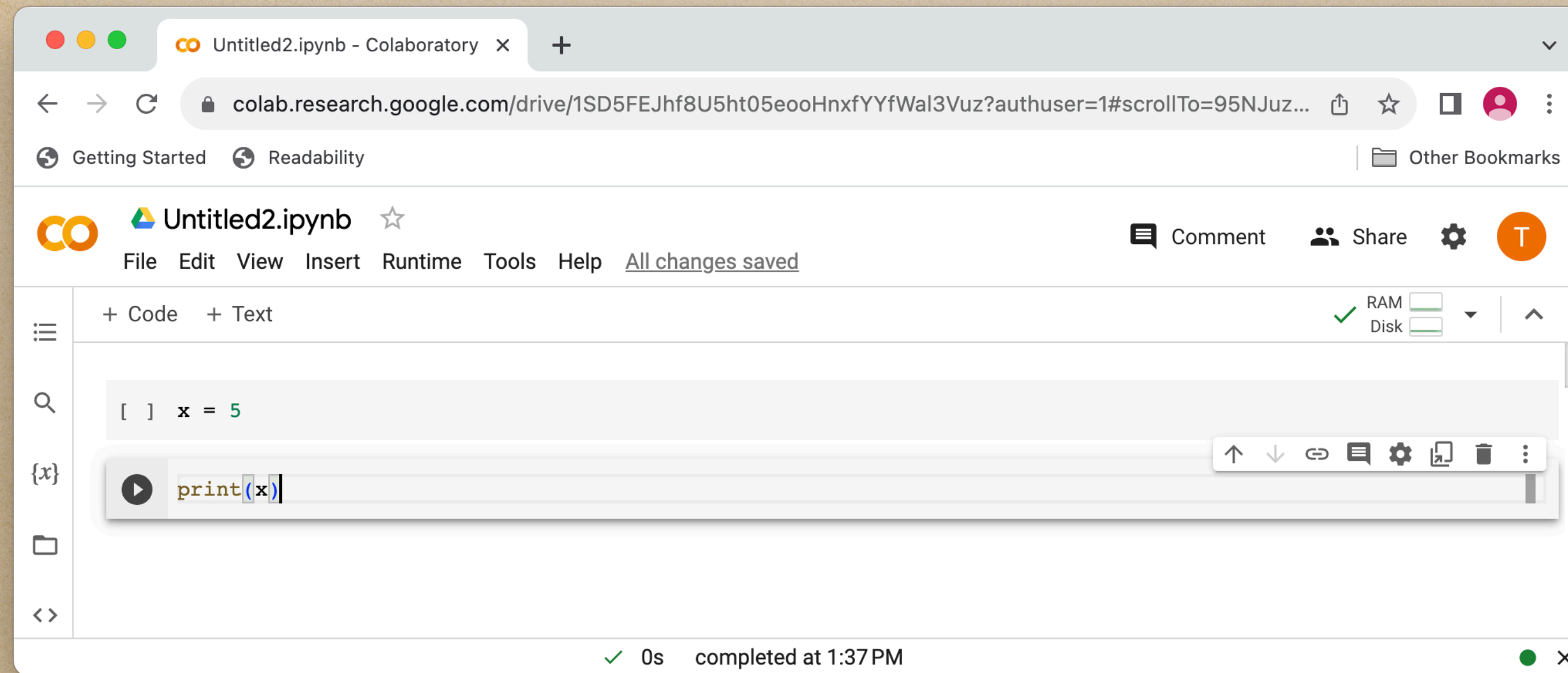
```
Hello, world
Goodbye, world
```

At the bottom of the code cell, a status bar indicates: `✓ 0s completed at 1:37 PM`. The notebook window has standard macOS-style window controls (red, yellow, green buttons) in the top-left corner and a close button (X) in the top-right corner.

Numbers

- bool
- int
- float
- decimal

Try this in your notebook.



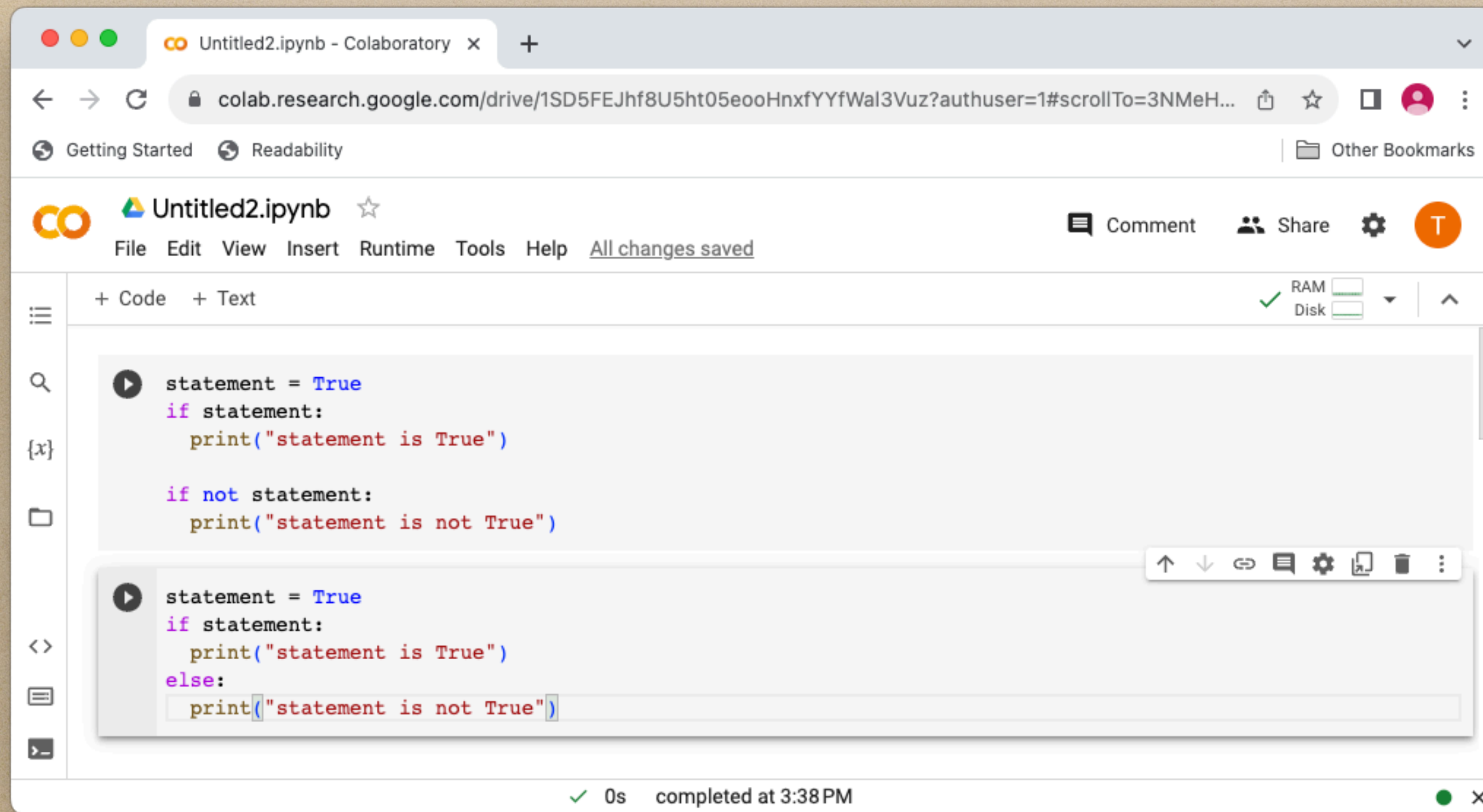
Click on the run button for the second cell. What happens? Try running the first cell and then running the second cell. Why is the result different?

Data Structures

- Lists - ordered collection of data. Items can be various data types.
- Tuples - like a list but immutable (cannot be changed).
- Sets - unordered collection not containing duplicates.
- Dictionaries - a collection of key value pairs.
- Arrays

Conditionals

Conditionals allow statements to be executed only if a condition is true (or false).



The screenshot shows a Google Colaboratory notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1SD5FEJhf8U5ht05eooHnxfYYfWal3Vuz?authuser=1#scrollTo=3NMeH...`. The notebook title is "Untitled2.ipynb". The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", with a status indicator "All changes saved".

The notebook contains two code cells. The first cell contains the following Python code:

```
statement = True
if statement:
    print("statement is True")

if not statement:
    print("statement is not True")
```

The second cell contains the following Python code:

```
statement = True
if statement:
    print("statement is True")
else:
    print("statement is not True")
```

At the bottom of the notebook, a status bar indicates that the execution completed successfully in 0 seconds at 3:38 PM.

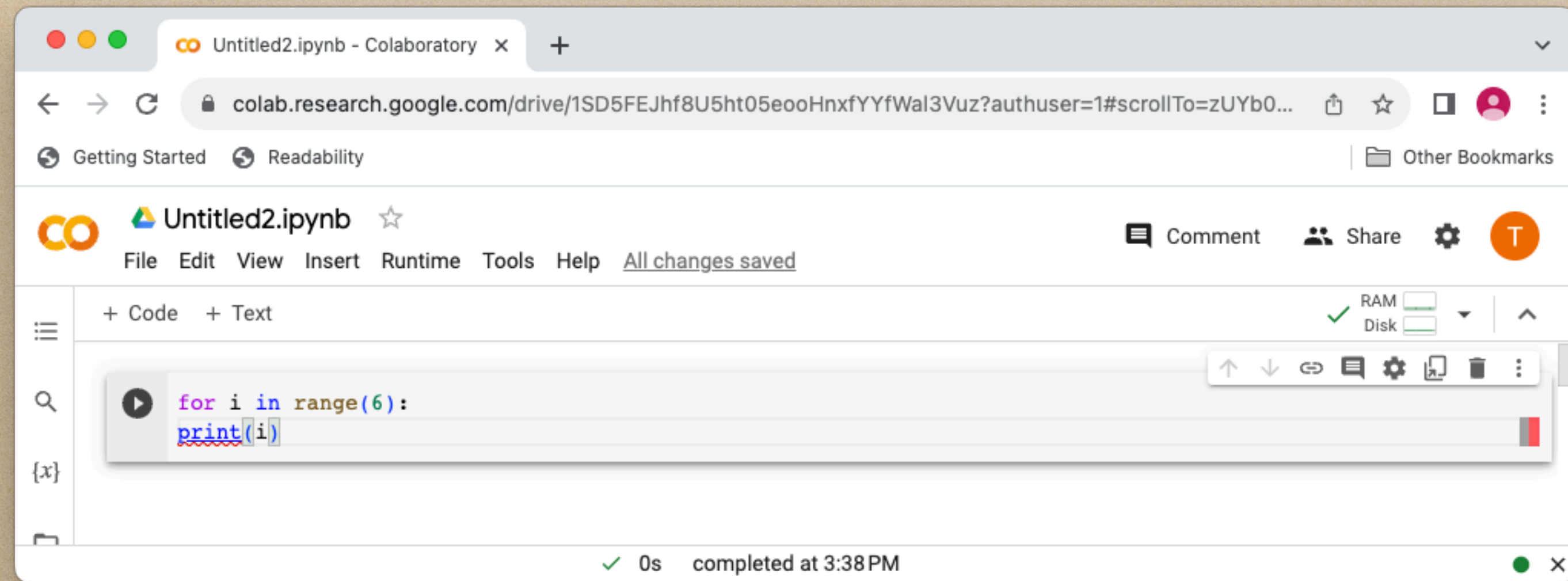
Loops

- while - condition stated at start
- for - execute over a list

There are a couple of special statements used in loops

- break - exit the loop
- continue - skip the rest of the loop and start again from the top

Try this



The screenshot shows a web browser window with a single tab titled "Untitled2.ipynb - Colaboratory". The address bar contains the URL "colab.research.google.com/drive/1SD5FEJhf8U5ht05eooHnxfYYfWal3Vuz?authuser=1#scrollTo=zUYb0...". The notebook interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", along with a "Comment" button and a "Share" button. The main workspace contains a code cell with the following Python code:

```
for i in range(6):  
    print(i)
```

The code cell is currently selected, and the status bar at the bottom indicates "0s completed at 3:38 PM".

Indent the second statement and try running the code again? What does the indentation do? What about `range(6)`?

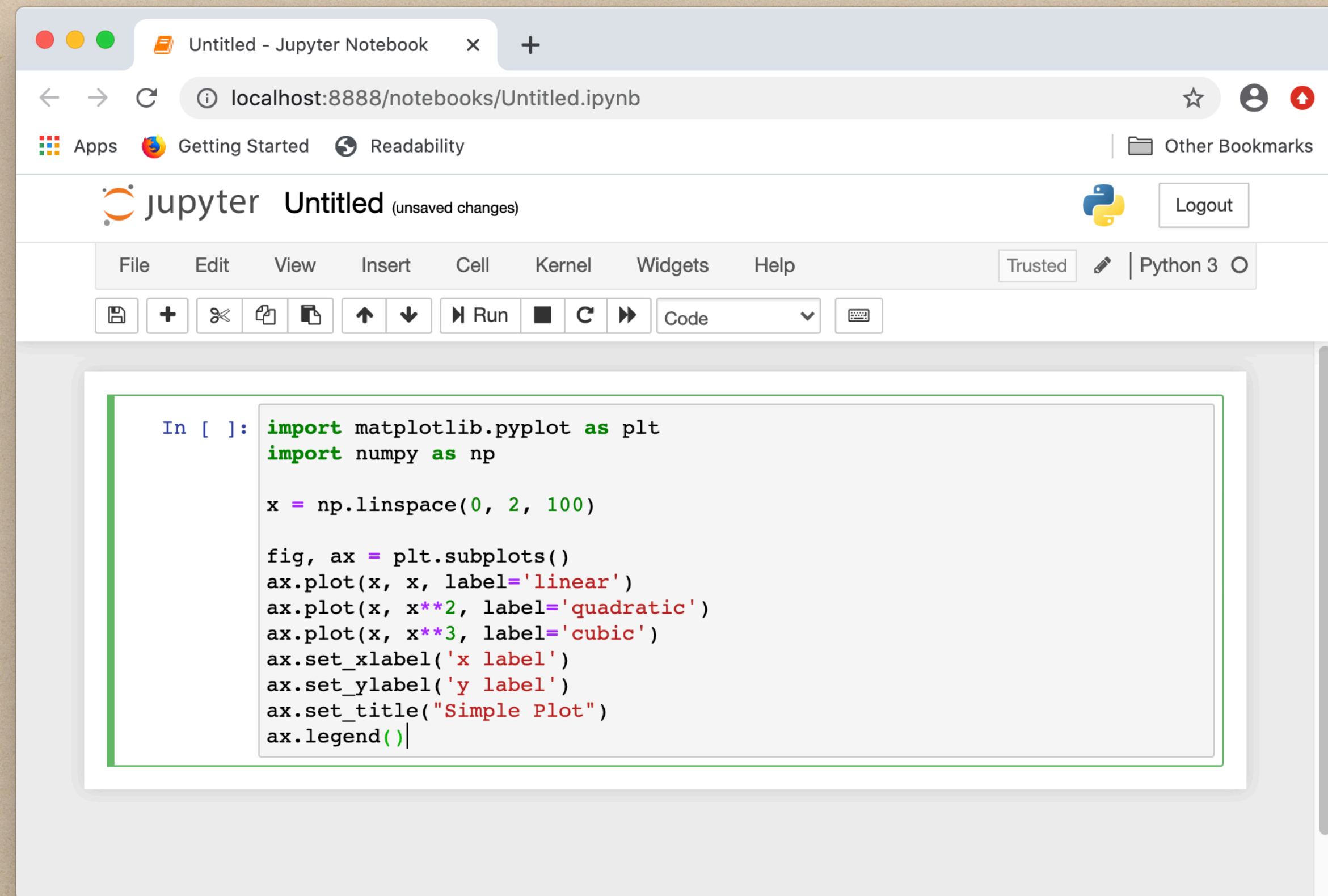
Charts and Plots

Python uses matplotlib to create graphs. In a code cell:

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

```
x = np.linspace(0, 2, 100)
```

```
fig, ax = plt.subplots()  
ax.plot(x, x, label='linear')  
ax.plot(x, x**2, label='quadratic')  
ax.plot(x, x**3, label='cubic')  
ax.set_xlabel('x label')  
ax.set_ylabel('y label')  
ax.set_title("Simple Plot")  
ax.legend()
```



The screenshot shows a Jupyter Notebook interface in a browser window. The browser address bar shows 'localhost:8888/notebooks/Untitled.ipynb'. The notebook title is 'Untitled (unsaved changes)'. The code cell contains the following Python code:

```
In [ ]: import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.linspace(0, 2, 100)  
  
fig, ax = plt.subplots()  
ax.plot(x, x, label='linear')  
ax.plot(x, x**2, label='quadratic')  
ax.plot(x, x**3, label='cubic')  
ax.set_xlabel('x label')  
ax.set_ylabel('y label')  
ax.set_title("Simple Plot")  
ax.legend()
```

Untitled - Jupyter Notebook x +

localhost:8888/notebooks/Untitled.ipynb

Apps Getting Started Readability Other Bookmarks

Jupyter Untitled (unsaved changes) Python 3 Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [4]: import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

fig, ax = plt.subplots()
ax.plot(x, x, label='linear')
ax.plot(x, x**2, label='quadratic')
ax.plot(x, x**3, label='cubic')
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")
ax.legend()
```

Out[4]: <matplotlib.legend.Legend at 0x1078373d0>

x label	linear	quadratic	cubic
0.00	0.00	0.00	0.00
0.25	0.25	0.06	0.01
0.50	0.50	0.25	0.12
0.75	0.75	0.56	0.42
1.00	1.00	1.00	1.00
1.25	1.25	1.56	1.97
1.50	1.50	2.25	3.38
1.75	1.75	3.06	5.51
2.00	2.00	4.00	8.00

In []: |

More information: <https://matplotlib.org/index.html>

Remote and Local Files

```
import requests
```

```
url = 'https://people.sc.fsu.edu/~jburkardt/data/csv/trees.csv'
```

```
filename = 'file.csv'
```

```
response = requests.get(url)
```

```
with open(filename, 'wb') as f:
```

```
    f.write(response.content)
```

```
with open(filename, 'r') as f:
```

```
    content = f.read()
```

```
    print(content)
```

Matplotlib Exercise

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data = pd.read_csv(filename).values
x = [row[1] for row in data]
y = [row[2] for row in data]
fig, ax = plt.subplots()
ax.plot(x, y, label='')
ax.set_xlabel('Girth')
ax.set_ylabel('Height')
ax.set_title('Trees')
```

colab.ipynb - Colaboratory

colab.research.google.com/drive/1R7h_t9mlAtisFqxTIZGDIPHNCN-BeQJFZ#scrollTo=0860xZsjTmn_

Getting Started Readability

colab.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data = pd.read_csv(filename).values
x = [row[1] for row in data]
y = [row[2] for row in data]
fig, ax = plt.subplots()
ax.plot(x, y, label='')
ax.set_xlabel('Girth')
ax.set_ylabel('Height')
ax.set_title('Trees')
```

Text(0.5, 1.0, 'Trees')

Girth	Height
8.5	70
9.5	64
10.5	72
11.5	83
12.5	75
13.5	85
14.5	80
15.5	72
16.5	77
17.5	82
18.5	80
19.5	84
20.5	87

0s completed at 7:29 PM

Programming Paradigms

For large computer programs often written by multiple people there can be problems with duplicate variable names that inadvertently overlap and documentation becomes more important. For variables the idea of scope becomes important. For documentation and maintainability the idea is to keep code local in some way so understanding the code requires only understanding a limited part of the code. Back when FORTRAN was written computer programs were not very long but as programs became more complex problems were discovered. Example: In FORTRAN it was possible to call a function and pass in the number 3. The function would call this by a variable name. If you set this variable to 2 then throughout the program every time you tried to print a 3 it would instead print a 2. Another example: FORTRAN used COMMON blocks where the variable scope could be shared across code. This meant a variable could be changed anywhere using that COMMON block. For longer and more complex code there are a couple of programming paradigms:

- Functional programming
- Object oriented programming

Functional Programming

- The problem in FORTRAN is that variables are passed by reference. This means that the storage location of the variable is passed. Most modern programming languages by default pass by value; they copy what is in that storage location to another storage location for the function to use. For large arrays or other variables this can be inefficient so they still can be passed by reference.
- Functions are first class objects that can be saved to a variable and passed around. This allows code to be stored as a variable.

The idea is that calling a function will not change the state of the code calling the function except by what is explicitly returned in an assignment. So the calling code does not have to understand the details of what is happening in the function. Similarly the function does not have to understand anything about the calling code since everything it needs is contained in the function: local variables and limited code.

Object Oriented Programming

The basic idea for object oriented code is that all variables and code pertaining to that part of the computer program are stored in an object. Thus the variable scope and the code that needs to be understood is limited to that object.

- Class - the code that is used to create an object.
- Object - a state created by a class that contains:
 - Properties which are variables
 - Methods which are code similar to a function

Object oriented programming can be very inefficient since it can require copying large data structures. Often abstraction and inheritance are used which can make it difficult to understand since code is no longer local. Also documentation for object oriented code is huge and almost always incomplete.

Application Programming Interface (APIs)

Whatever the programming paradigm or other ways of organizing your code the most important aspect is deciding on interfaces between parts of your code and making sure that the interfaces are well-defined and documented. The interface should also be relatively stable since any changes can cause other people a lot of work changing their code.

Functions

```
def hello():  
    print('Hello, world!')
```

```
hello()
```


Classes

```
class HelloClass:  
    def __init__(self, name):  
        self.name = name  
  
    def greet(self):  
        print(f'Hello {self.name}!')  
  
world = HelloClass(name='world')  
world.greet()
```

The method `__init__()` is called when an object is created. `greet()` is a method and `name (self.name)` is a property.

Untitled - Colaboratory

colab.research.google.com/drive/1Editl8hNSO64nNXiixesBBpOn4fYEAn2?au... Other Bookmarks

Getting Started Readability

Untitled

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text RAM Disk

```
class HelloClass:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print(f'Hello {self.name}!')

world = HelloClass(name='world')
world.greet()
```

Hello world!

0s completed at 4:58 PM

There is some additional code that makes it easier to debug a model. It helps when an agent is printed out that all their properties are printed. So a function is added to the class to convert the object to a string in the way we want.

```
def __str__(self):  
    return str(self.__class__) + ": " + str(self.__dict__)
```

Try adding this to the HelloClass, creating an object and then calling this function.

Advanced Example using GPU/TPU

This example uses artificial intelligence to perform optical character recognition. The model has already been built and this just applies the model.

```
!pip install keras_ocr
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import keras_ocr
import cv2
import math
import numpy as np
```

```
def midpoint(x1, y1, x2, y2):  
    x_mid = int((x1 + x2) / 2)  
    y_mid = int((y1 + y2) / 2)  
    return (x_mid, y_mid)
```

The code is going to create a mask over each word and will create annotations showing the words it finds.
Then it is going to fill in the mask using inpaint

```
pipeline = keras_ocr.pipeline.Pipeline()
# Find a meme to read in.
img = keras_ocr.tools.read('https://gaiaes.com/763836.png')
prediction_groups = pipeline.recognize([img])
keras_ocr.tools.drawAnnotations(image=img, predictions=prediction_groups[0])
mask = np.zeros(img.shape[:2], dtype='uint8')
for box in prediction_groups[0]:
    x0, y0 = box[1][0]
    x1, y1 = box[1][1]
    x2, y2 = box[1][2]
    x3, y3 = box[1][3]
    x_mid0, y_mid0 = midpoint(x1, y1, x2, y2)
    x_mid1, y_mid1 = midpoint(x0, y0, x3, y3)
    thickness = int(math.sqrt((x2 - x1)**2 + (y2 - y1)**2))
    cv2.line(mask, (x_mid0, y_mid0), (x_mid1, y_mid1), 255, thickness)
img = cv2.inpaint(img, mask, 7, cv2.INPAINT_NS)
imgplot = plt.imshow(img)
plt.imsave('meme.png', img)
```

Running Code on GPUs

Go to Edit -> Notebook settings. Select a GPU or TPU. Then try running the code again.

The `timeit` package can be used to compute speedups using GPUs and TPUs. For an example:

<https://colab.research.google.com/notebooks/gpu.ipynb>

Implement timing and compute the speed up using different procession unit options.

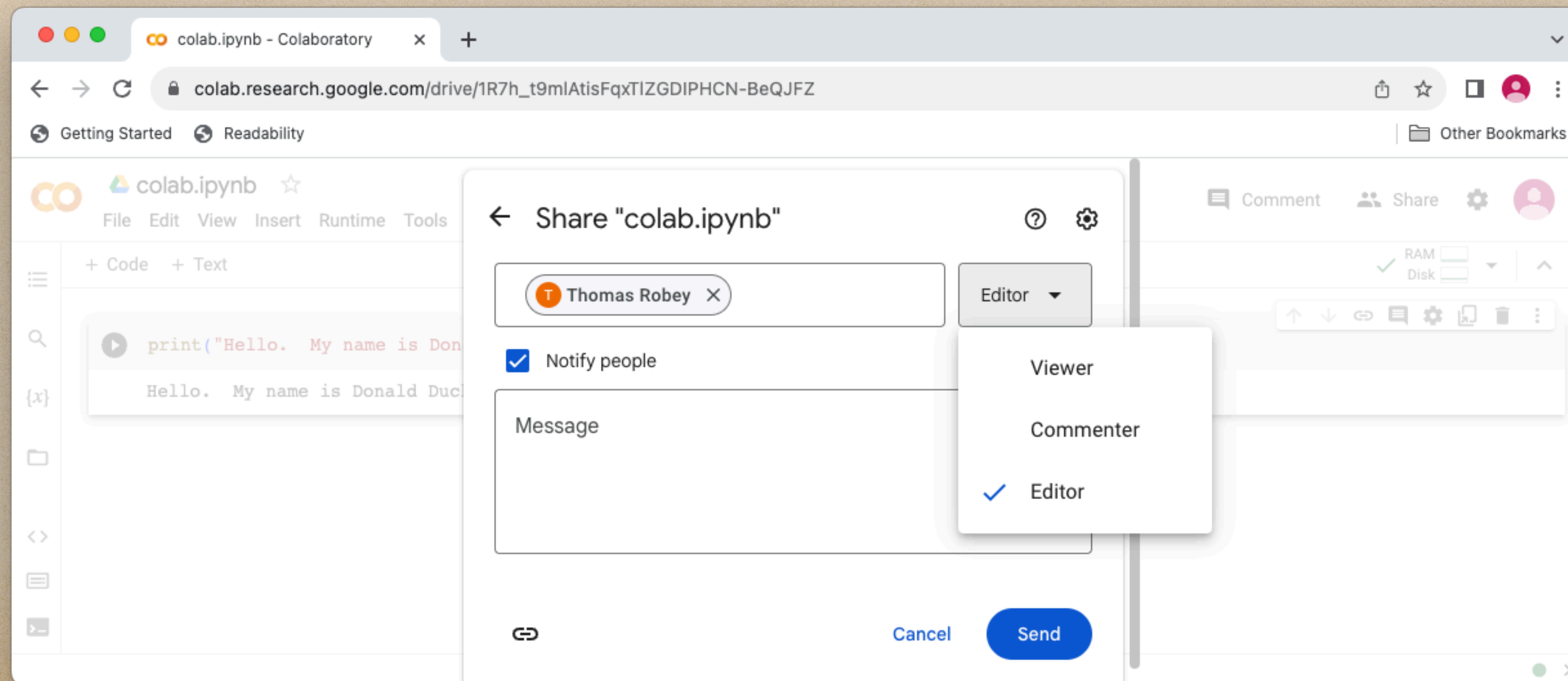
Saving a Notebook

Now let us name our notebook. Click on the name and enter colab

Then go to File -> Save a copy in Drive to save your notebook to Google Drive.

Sharing Your Notebook

A Jupyter notebook saved to a Github account can either be public or private. Similarly a file in your Google Drive can be shared. But your notebook can also be shared right from the notebook. Click on the Share link on the upper right of your notebook. There is a General access section where you can make your notebook restricted or make it available to anyone with the link. At the top you can grant access to individuals. There is also a setting for the type of access. This can view, comment or edit access. Try sharing this notebook with a team member.



Markdown Text

Jupyter Notebooks uses markdown text. Click on + Text to add a new text cell. Then with your new cell selected change Code to Markdown. Markdown uses # for headers. Repeat to get smaller headers. Make sure to include a space before your header text.

Surround text with single asterisks or underscores to get italics. Use two to make the text bold. Double click your markdown cell and try it! Use a dash, plus sign, or asterisk followed by a space to get a list.

You can enter code in your text (which cannot be run) by surrounding it with backticks. A block of code can be entered with triple backticks (and the computer language to get code highlighting).

Markdown cheat sheet: <https://www.markdownguide.org/cheat-sheet/>

The image shows a browser window with a single tab titled "Untitled - Colaboratory". The address bar contains the URL `colab.research.google.com/drive/1Editl8hNSO64nNXiixesBBpOn4fYEAn2?au...`. Below the address bar, there are navigation icons for "Getting Started" and "Readability", and a "Connect" button. The main header area includes the Colaboratory logo, the text "Untitled", and a star icon. A menu bar contains "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". A status indicator shows "All changes saved". On the right side of the header, there are icons for "Comment", "Share", and a terminal icon. The main workspace is split into two panes. The left pane is a code editor with a toolbar containing icons for text formatting (bold, italic, code), link, image, list, and other functions. The code in the editor is:

```
# Header 1
## Header 2
- item 1
- item 2
```python
print(x)
```The right pane shows the rendered output of the code. It features a vertical dashed line separating the code from the rendered content. The rendered content includes a large heading "Header 1", a smaller heading "Header 2", and a bulleted list with two items: "item 1" and "item 2". Below the list, there is a light gray box containing the text print(x). The browser window has standard macOS window controls (red, yellow, green buttons) in the top-left corner and a close button (X) in the top-right corner.
```

Mathematical Expressions

Jupyter Notebook has LaTeX support built-in which provides the ability to enter mathematical expressions. LaTeX or TeX can provide typeset quality mathematical expressions. If you submit an abstract for a conference they will often require it in LaTeX because they can give to a printer and the typesetting machines can understand it. This reduces the amount of work for the printer and ensures your abstract appears the way you intended it to appear. For inline equations enter the equation between single \$. For a block with an equation use double \$\$ instead.

LaTeX math cheat sheet: <http://tug.ctan.org/info/undergradmath/undergradmath.pdf>

Try an example from the cheat sheet.

Untitled - Colaboratory

colab.research.google.com/drive/1Editl8hNSO64nNXiixesBBpOn4fYEAn2?au...
Getting Started Readability Other Bookmarks

Untitled
File Edit View Insert Runtime Tools Help All changes saved
Comment Share

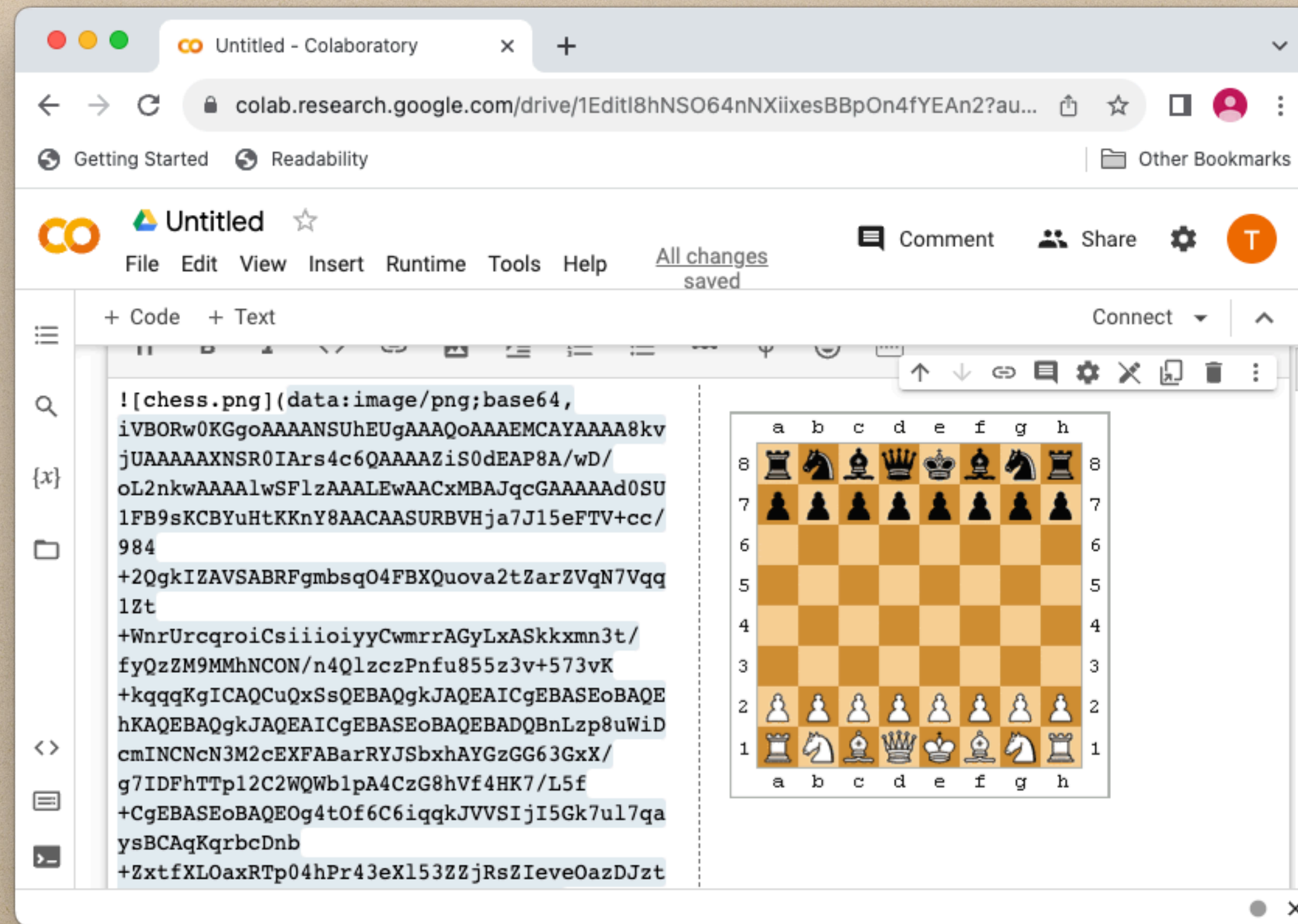
+ Code + Text Connect

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Images

Jupyter Notebooks has support for drag and drop for images since version 5.0.0. Just drag the file into a markdown cell.



Magic commands are beyond the scope of this presentation but you can get more information on the web or using the following magic commands

`%run` runs a program

`%magic` prints a complete guide to magic commands

`%quickref` prints a brief guide to magic commands

`%lsmagic` lists all the magic commands

If you want more information about a magic command follow it with a question mark

`%run?`

Appendix 1: Installing Python 3

It is easy to find instructions by searching on the web. Some are more complicated than necessary but if you get stuck just look around for instructions. Usually typing `python` gets the default version and `python3` (or `pip3`) runs just the Python 3 version in case Python 2 is also installed. Python 2 is no longer supported. Note that the long dash `—` is typed as two short dashes.

Windows

- Go to <https://www.python.org/downloads/windows>. Find the 32 bit or 64 bit executable installer for Python 3 and download it. There is a bug in Python 3.8 for connecting to browsers so choose 3.7.
- Run the installer. Make sure to check to add Python to the path. You may get a message to disable the path length limit. Select this if it appears.
- Type `python —version` to verify install.
- Launch from a terminal by typing `python`.

PIP is included with Python 3.4+. Check by typing `pip —version`.

Linux

Python comes already installed on Linux. Although Python 2 is no longer supported check by typing

- `python --version`
- `python3 --version`

If the default version is Python 2 then you may need to use `python3` and `pip3` instead of just `python` and `pip`.

Mac

Python is installed on a Mac but this is for the system to use. Leave this alone to avoid breaking things. Homebrew will be used. It is a package manager for Macs. First, Xcode will need to be installed. It can be found in the app store. It is a large download. Once Xcode is installed, then open up a terminal

- `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

Then install Python

- `brew install python3`

Check to verify installation

- `python --version`

Appendix II: Installing Jupyter Notebooks

Start by installing Python 3 and PIP 3 (see Appendix I).

For Windows and Mac in a terminal run

- `python3 -m pip install --upgrade pip`
- `python3 -m pip install jupyter`

For Linux (Debian derivatives) instead use

- `sudo apt install jupyter-notebook`

Getting Started

Start by installing Jupyter notebooks. There are instructions in the Appendices. Then from a terminal start Jupyter notebooks.

- jupyter notebook

For Linux run

- jupyter-notebook

This will open a browser. In the upper right hand corner click on New and select Python 3. Click on File -> Save as.. and select a name for your notebook.

