Bi195PythonIntro

October 6, 2022

1 Bi 195 - Python Introduction

Hi everyone! Here is a simple and quick starter to Python, Google Colab, and some of the libraries used for Bi 195.

We'll start by explaining how to use Google Colab (if you're not using Colab, we highly recommend you do). This here is a text cell. If you double click on the text, it will open into edit mode! You can write whatever you want in the box and see what will actually be visible on the right.

You can also add special characters to make titles and other fun things.

2 I am a big title - big titles make sections

2.1 I am a medium title

2.1.1 I am a small title... and so on

You can add ** to make things **bold** and just one * to make things *italicised*. There are a bunch of other cool things you do, and you can find more documentation here.

If you want to neatly separate your sections you can add a horizontal bar to any block of text by doing:

---

3 Basic Python

[1]:
```
# You can also make code boxes!  The code running in here is Python 3.7.
# To run a cell, press on the white play icon in the top left corner of the␣
 ↪cell.

# Let's walk through some REALLY basic Python programming

# This is a comment, as I have added a # sign at the front of the line

'''
```

```
This is a comment, too!
I have just added three single quotes to the beginning and end of the comment␣
 ↪block

And I can go over multiple lines
and add code that isn't evaluated
x = 23
'''

# You can comment and uncomment large chunks of code by highlighting them and␣
 ↪then pressing
# "CMD" + "/" on Mac
# "control" + "/" on Windows

# You set variables like this
variable = 2 # integer
v = [1,2,3] # array of integers

# And you can print your variables to the notebook in two ways:
# 1 - a print statement
print(variable, v)

# 2 - call them in the last line of a code cell
v = 'variable'
variable, v
```

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

[1]:
```
(2, 'variable')
```

[2]:
```
# For Bi195 we will commonly use the numpy (https://numpy.org/doc/stable/user/
 ↪index.html)
# and maptlotlib pyplot (https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.
 ↪pyplot.html) libraries.

# You access these libraries by adding these import statements to your file:
# Usually they go at the very top of your file.

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

[3]:
```
# Let's walk through some examples

# First we can look at the example code from the coin flip example.
# Here we are finding the height and angular position of a rotating object␣
 ↪thrown in the air
```

```python
# We begin by initializing two arrays to keep track of y and theta.
coiny = []
cointheta = []

# We can also intialize some variables for the initial velocity and angular␣
 ↪velocity
v0 = 50
w0 = .1

# We can then loop through 100 'seconds'
for t in range(100):
    # And for each iteration, we want to add the current height
    coiny.append(-t**2/2 + v0*t)
    # and velocity to our arrays
    cointheta.append(np.cos(w0*t))

print(f'coiny array: {coiny}') # This is a f-interpolated string, where we can␣
 ↪plug in variables using {}
print(f'cointheta array: {cointheta}\n')

# We can also do the same using list comprehension
coinylc = [-t**2/2 + v0 * t for t in range(100)]
cointhetalc = [np.cos(w0*t) for t in range(100)]

# We can check if these arrays are equal visually by looking at print␣
 ↪statements...
print(f'coiny array using list comprehension: {coinylc}')
print(f'cointheta array using list comprehension: {cointhetalc}')

# ...or by using assert statements and/or booleans
# In Python, we use == to check equality, and you can build logic statments by␣
 ↪simply writing 'and' / 'or' / 'not'/ etc.
assert(coiny == coinylc and cointheta == cointhetalc)
truth = coiny == coinylc and cointheta == cointhetalc
```

```
coiny array: [0.0, 49.5, 98.0, 145.5, 192.0, 237.5, 282.0, 325.5, 368.0, 409.5,
450.0, 489.5, 528.0, 565.5, 602.0, 637.5, 672.0, 705.5, 738.0, 769.5, 800.0,
829.5, 858.0, 885.5, 912.0, 937.5, 962.0, 985.5, 1008.0, 1029.5, 1050.0, 1069.5,
1088.0, 1105.5, 1122.0, 1137.5, 1152.0, 1165.5, 1178.0, 1189.5, 1200.0, 1209.5,
1218.0, 1225.5, 1232.0, 1237.5, 1242.0, 1245.5, 1248.0, 1249.5, 1250.0, 1249.5,
1248.0, 1245.5, 1242.0, 1237.5, 1232.0, 1225.5, 1218.0, 1209.5, 1200.0, 1189.5,
1178.0, 1165.5, 1152.0, 1137.5, 1122.0, 1105.5, 1088.0, 1069.5, 1050.0, 1029.5,
1008.0, 985.5, 962.0, 937.5, 912.0, 885.5, 858.0, 829.5, 800.0, 769.5, 738.0,
705.5, 672.0, 637.5, 602.0, 565.5, 528.0, 489.5, 450.0, 409.5, 368.0, 325.5,
282.0, 237.5, 192.0, 145.5, 98.0, 49.5]
cointheta array: [1.0, 0.9950041652780258, 0.9800665778412416,
0.955336489125606, 0.9210609940028851, 0.8775825618903728, 0.8253356149096782,
```

0.7648421872844884, 0.6967067093471654, 0.6216099682706644, 0.5403023058681398,
0.4535961214255773, 0.3623577544766734, 0.26749882862458735,
0.16996714290024081, 0.0707372016677029, -0.029199522301288815,
-0.12884449429552486, -0.2272020946930871, -0.3232895668635036,
-0.4161468365471424, -0.5048461045998576, -0.5885011172553458,
-0.6662760212798244, -0.7373937155412458, -0.8011436155469337,
-0.8568887533689473, -0.9040721420170612, -0.9422223406686583,
-0.9709581651495907, -0.9899924966004454, -0.9991351502732795,
-0.9982947757947531, -0.9874797699088649, -0.9667981925794609,
-0.9364566872907963, -0.896758416334147, -0.848100031710408,
-0.7909677119144165, -0.7259323042001399, -0.6536436208636119,
-0.5748239465332685, -0.4902608213406994, -0.40079917207997545,
-0.30733286997841935, -0.2107957994307797, -0.11215252693505398,
-0.01238866346289056, 0.08749898343944727, 0.18651236942257576,
0.28366218546322625, 0.37797774271298107, 0.4685166713003771,
0.5543743361791615, 0.6346928759426347, 0.70866977429126, 0.7755658785102502,
0.8347127848391598, 0.8855195169413194, 0.9274784307440359, 0.960170286650366,
0.9832684384425847, 0.9965420970232175, 0.9998586363834151, 0.9931849187581926,
0.9765876257280235, 0.9502325919585293, 0.9143831482353194, 0.8693974903498248,
0.8157251001253568, 0.7539022543433046, 0.6845466664428059, 0.6083513145322546,
0.5260775173811045, 0.43854732757439036, 0.3466353178350258,
0.25125984258225487, 0.15337386203786435, 0.05395542056264886,
-0.04600212563953695, -0.14550003380861354, -0.2435441537357911,
-0.33915486098383624, -0.4313768449706208, -0.5192886541166856,
-0.6020119026848236, -0.6787200473200125, -0.7486466455973999,
-0.811093014061656, -0.8654352092411123, -0.9111302618846769,
-0.9477216021311119, -0.974843621404164, -0.9922253254526034,
-0.9996930420352065, -0.9971721561963784, -0.9846878557941267,
-0.9623648798313097, -0.9304262721047533, -0.8891911526253609]

coiny array using list comprehension: [0.0, 49.5, 98.0, 145.5, 192.0, 237.5,
282.0, 325.5, 368.0, 409.5, 450.0, 489.5, 528.0, 565.5, 602.0, 637.5, 672.0,
705.5, 738.0, 769.5, 800.0, 829.5, 858.0, 885.5, 912.0, 937.5, 962.0, 985.5,
1008.0, 1029.5, 1050.0, 1069.5, 1088.0, 1105.5, 1122.0, 1137.5, 1152.0, 1165.5,
1178.0, 1189.5, 1200.0, 1209.5, 1218.0, 1225.5, 1232.0, 1237.5, 1242.0, 1245.5,
1248.0, 1249.5, 1250.0, 1249.5, 1248.0, 1245.5, 1242.0, 1237.5, 1232.0, 1225.5,
1218.0, 1209.5, 1200.0, 1189.5, 1178.0, 1165.5, 1152.0, 1137.5, 1122.0, 1105.5,
1088.0, 1069.5, 1050.0, 1029.5, 1008.0, 985.5, 962.0, 937.5, 912.0, 885.5,
858.0, 829.5, 800.0, 769.5, 738.0, 705.5, 672.0, 637.5, 602.0, 565.5, 528.0,
489.5, 450.0, 409.5, 368.0, 325.5, 282.0, 237.5, 192.0, 145.5, 98.0, 49.5]
cointheta array using list comprehension: [1.0, 0.9950041652780258,
0.9800665778412416, 0.955336489125606, 0.9210609940028851, 0.8775825618903728,
0.8253356149096782, 0.7648421872844884, 0.6967067093471654, 0.6216099682706644,
0.5403023058681398, 0.4535961214255773, 0.3623577544766734, 0.26749882862458735,
0.16996714290024081, 0.0707372016677029, -0.029199522301288815,
-0.12884449429552486, -0.2272020946930871, -0.3232895668635036,
-0.4161468365471424, -0.5048461045998576, -0.5885011172553458,
-0.6662760212798244, -0.7373937155412458, -0.8011436155469337,

```
-0.8568887533689473, -0.9040721420170612, -0.9422223406686583,
-0.9709581651495907, -0.9899924966004454, -0.9991351502732795,
-0.9982947757947531, -0.9874797699088649, -0.9667981925794609,
-0.9364566872907963, -0.896758416334147, -0.848100031710408,
-0.7909677119144165, -0.7259323042001399, -0.6536436208636119,
-0.5748239465332685, -0.4902608213406994, -0.40079917207997545,
-0.30733286997841935, -0.2107957994307797, -0.11215252693505398,
-0.01238866346289056, 0.08749898343944727, 0.18651236942257576,
0.28366218546322625, 0.37797774271298107, 0.4685166713003771,
0.5543743361791615, 0.6346928759426347, 0.70866977429126, 0.7755658785102502,
0.8347127848391598, 0.8855195169413194, 0.9274784307440359, 0.960170286650366,
0.9832684384425847, 0.9965420970232175, 0.9998586363834151, 0.9931849187581926,
0.9765876257280235, 0.9502325919585293, 0.9143831482353194, 0.8693974903498248,
0.8157251001253568, 0.7539022543433046, 0.6845466664428059, 0.6083513145322546,
0.5260775173811045, 0.43854732757439036, 0.3466353178350258,
0.25125984258225487, 0.15337386203786435, 0.05395542056264886,
-0.04600212563953695, -0.14550003380861354, -0.2435441537357911,
-0.33915486098383624, -0.4313768449706208, -0.5192886541166856,
-0.6020119026848236, -0.6787200473200125, -0.7486466455973999,
-0.811093014061656, -0.8654352092411123, -0.9111302618846769,
-0.9477216021311119, -0.974843621404164, -0.9922253254526034,
-0.9996930420352065, -0.9971721561963784, -0.9846878557941267,
-0.9623648798313097, -0.9304262721047533, -0.8891911526253609]
```

[4]:
```
# But, be careful because assert statements throw an error and stop your code
 ↪if they are not satisfied
# Try uncommenting the line below and running this cell!
# assert(coiny == cointheta)
```

[5]:
```
# If you ever need to use something that throws an error, we recommend using a
 ↪try/except statement:
try:
    assert(coiny == cointheta)
except:
    print(f'Uh oh! There was an error!')
```

```
Uh oh! There was an error!
```

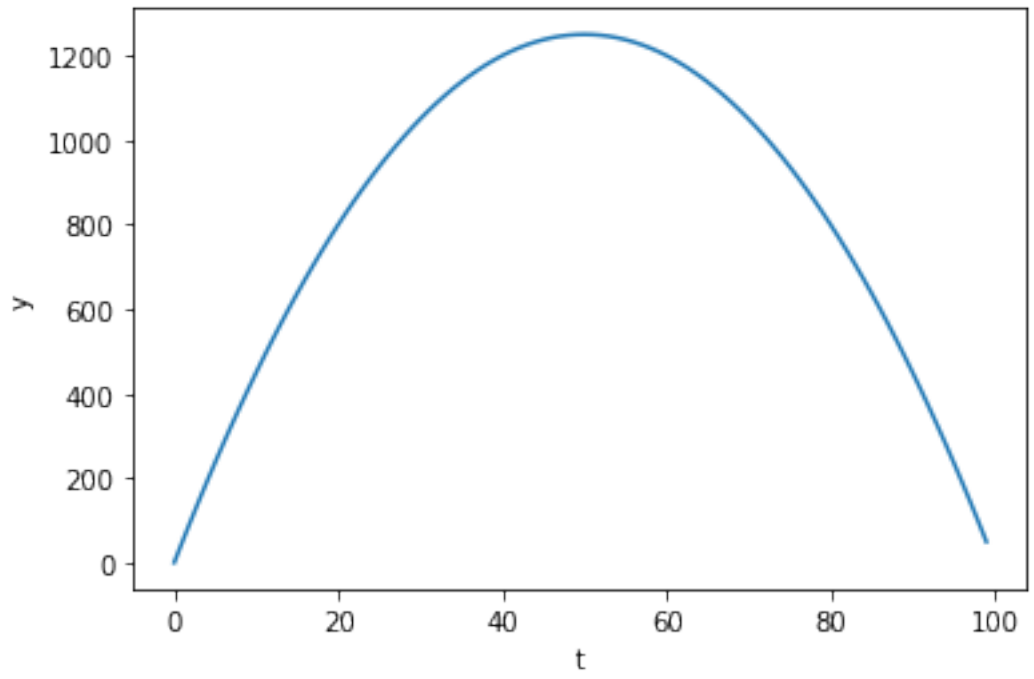
[6]:
```
# A very brief intro on function writing
# You define a function like this:
def myFunction(arg1, arg2, arg3):
    mean = (arg1 + arg2 + arg3) / 3
    return mean

# And you can call your function like this:
arg1 = 1
arg2 = 2
```

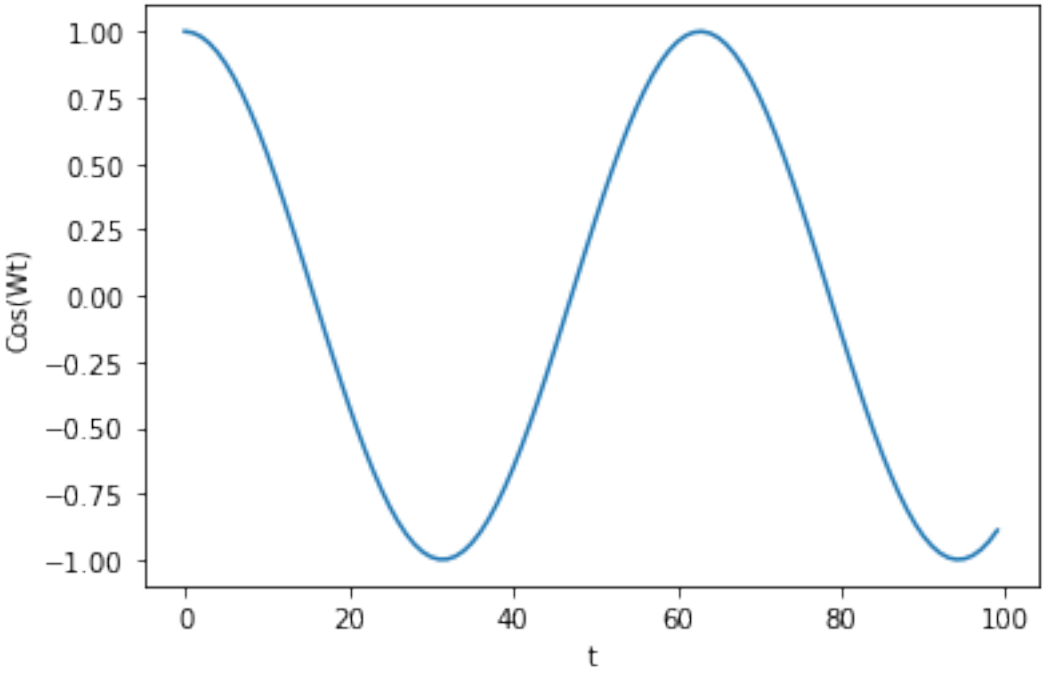
```
arg3 = 3
mean1 = myFunction(arg1, arg2, arg3)
mean2 = myFunction(4, 5, 6)
print(f'mean1 = {mean1}, mean2 = {mean2}')
```

```
mean1 = 2.0, mean2 = 5.0
```

4 Matplotlib

[7]:
```
# We now want to see what the data looks like!  Let's plot it
# We create a figure and get its axes by calling on the subplots function of␣
 ↪the plt library
fig1, ax1 = plt.subplots()

# We can now set the labels of the axes and plot the height
ax1.set_xlabel('t')
ax1.set_ylabel('y')
ax1.plot(coiny)

# And then ceeate a second figure to do the same for the theta
fig2, ax2 = plt.subplots()
plt.plot(cointheta)
ax2.set_xlabel('t')
ax2.set_ylabel('Cos(Wt)')

plt.show() # This line isn't necessary. Try commenting it out and seeing what␣
 ↪happens.
```


1200
1000
800
600
400
200
0
y
0
20
40
60
80
100
t


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Cos(Wt)
0
20
40
60
80
100
t

5 Numpy

[8]:
```
# Now, let's talk about Numpy (np). It makes quantitative calculation a lot␣
↪easier

# Say we want to make an array of values between 0 and 10 increasing by 0.5:
arr = np.arange(0, 10, 0.5)
print(f'Array: {arr}')

# And then we want to find its mean and variance:
mean = np.mean(arr)
var = np.var(arr)
print(f'Mean: {mean}\nVar: {var}')

# We can access individual elements in an array by indexing
first_item = arr[0] # Remember Python uses 0-indexing
fourth_item = arr[3]
last_item = arr[-1]
print(f'1st item: {mean}\n4th item: {var}\nLast item: {last_item}')
```

```
Array: [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
Mean: 4.75
Var: 8.3125
1st item: 4.75
4th item: 8.3125
Last item: 9.5
```

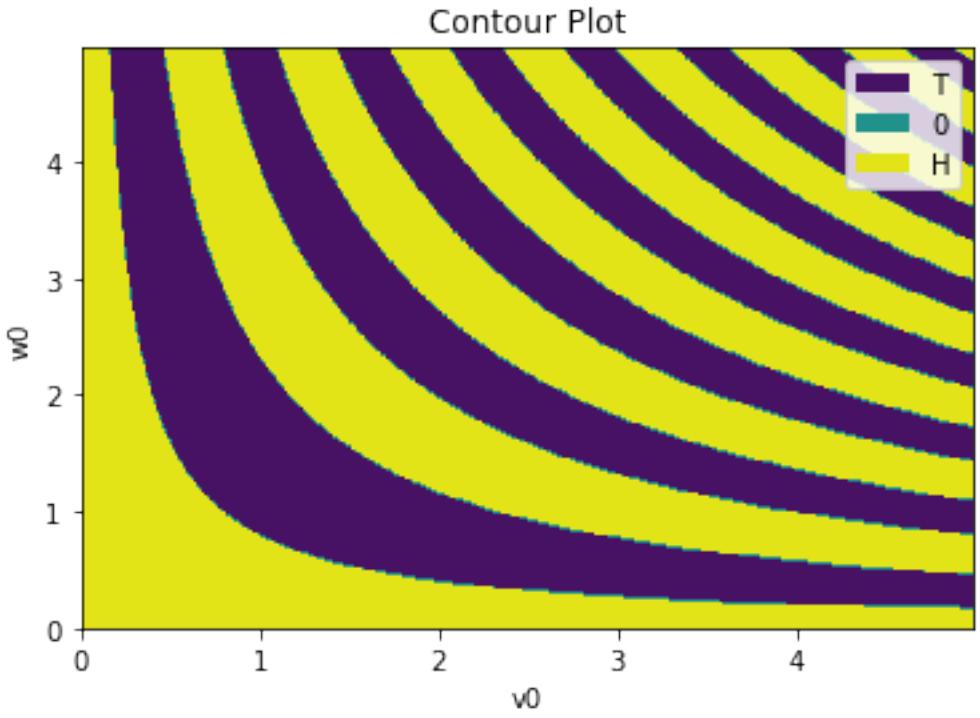
6 Step-by-step contour plot

[9]:
```
# For the last bit of code walkthrough, we'll more closely examine the coin␣
↪flip example's contour plot

# We first create arrays of values between 0 and 5 increasing by 0.01 or 0.02
# These represent a distribution of initial angular and linear velocities
v0 = np.arange(0, 5, .01)
w0 = np.arange(0, 5, .02)

# Let's take a look at the dimensions of these arrays:
dimv0 = v0.shape
dimw0 = w0.shape
print(f'Dimensions of v0 array: {dimv0}\nDimensions of w0 array: {dimw0}')
print('A shape tuple (r, c) means we have r rows and c columns in our array')
print('If there is no value for c, then it means we have a 1-dimensional array␣
↪with r entries\n')
```

```
# We then create a 2-D grid of features by using np.meshgrid
# This gives us two arrays of dimension dimw0 x dimv0 (or 250 x 500) where
# X has 250 rows where each row consists of the 500 entries from v0 and
# Y has 500 columns where each column consists of the 250 entries from w0
[X, Y] = np.meshgrid(v0, w0)
dimx = X.shape
dimy = Y.shape
print(f'Dimensions of X array: {dimx}\nDimensions of Y array: {dimy}')

# This is just initializing our figure to only have 1 subplot
fig, ax = plt.subplots(1, 1)

# We then create Z, which will determine the contour height at a specific␣
 ↪coordinate (x, y) for all X, Y
# Notice then, that Z is an array of dimension (250,500), for every (x, y)␣
 ↪coordinate
Z = np.sign(np.cos(X*Y*2))
dimz = Z.shape
print(f'Dimensions of Z array: {dimz}')

# The line below plots contour the lines defined by Z.
# The levels keyword argument allows you to manually define the cutoffs between␣
 ↪contours
# that are colored differently.  Here we create cutoffs so that values less␣
 ↪than -1.1 are
# defined as the first contour, values in between -1.1 and -.9 define the␣
 ↪second contour, and so on
cs = ax.contourf(X, Y, Z,levels = [-1.1,-.9, .9,1.1])

# We then set our titles and axes labels
ax.set_title('Contour Plot')
ax.set_xlabel('v0')
ax.set_ylabel('w0')

# This final bit of code allows us to create a legend
# This line is simply building a list using list comprehension (see above)␣
 ↪which creates a
# rectangle of each unique color in the contour plot.  We then add these␣
 ↪rectangles to the
# legend, with a manual labeling of each rectangle.  Note that there are three␣
 ↪colors since
# there are three contours, the "0" contour is for the case when the coin lands␣
 ↪perfectly
# on its side
proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
```

```
    for pc in cs.collections];
plt.legend(proxy, ["T", "0", "H"])
plt.show()
```

```
Dimensions of v0 array: (500,)
Dimensions of w0 array: (250,)
A shape tuple (r, c) means we have r rows and c columns in our array
If there is no value for c, then it means we have a 1-dimensional array with r
entries

Dimensions of X array: (250, 500)
Dimensions of Y array: (250, 500)
Dimensions of Z array: (250, 500)
```



7 Exporting your work

For this course, we ask that you turn in all of your work as a .pdf file. There are a few ways to do this using Google Colab.

7.1 Method 1

The first way to do this is manually by following these instructions:

1. Go to File > Print
2. Select Save as PDF

This method saves a printout of all of your code as well as your text boxes, with their outputs displayed. However, this isn't perfect as you can see by 'exportMethod1.pdf' in this directory. Some of the plots overlap with some of the text. In order to fix this, you can manually save the images and display them instead of just using console output. However, this is difficult and time consuming and we suggest you just use Method 2.

7.2 Method 2

Another way to do this is by adding the following cells of code to your notebook. However, it is extremely **IMPORTANT** that you have deleted your runtime and have run all of your cells **PRIOR** to running this cell! It is recommended to place this at the end of your notebook. For the sake of ease, you can comment out the entire code block as well.

Here are the step-by-step instructions on how to use Method 2 to convert your colab file to a pdf:

1. Go to Runtime > Disconnect and delete runtime.
2. Making sure that the cell below is commented out, go to Runtime > Run all.
3. Uncomment the below cell and run. This will spawn a box on your screen asking you to give permission to access your Google Drive. Make sure to log in and allow access. The tab should automatically close when verified.
4. Wait up to ~2 minutes, download the pdf file once it pops up in the left file pane (note that sometimes there is a delay of longer than 2 min, try rerunning the cell a few times if you have been waiting for more than 2 min).
5. Make sure to check that the contents are as desired.

[ ]:
```
# This first line hides all of the output to your notebook,
# you can remove it if you are curious as to what goes on behind the scenes
%%capture

# This next line clones the nbconvert repositiory from git, which
# was created to help make converting colab notebooks into pdfs and other␣
 ↪formats
# easier.  For more documentation, click on the link below
!git clone https://github.com/jupyter/nbconvert.git

# This navigates to the nbconvert directory of the repository
!cd nbconvert

# This line installs all of the dependencies for the repository to your VM
!pip install -e .

# This line installs the pandoc library (https://pandoc.org/)
!apt-get install pandoc

# And all of these lines install the inkscape library (https://inkscape.org/)
```

```
!apt-get update
!apt-get install inkscape
!add-apt-repository universe
!add-apt-repository ppa:inkscape.dev/stable - b
!apt-get update
!apt install inkscape

# This line installs the xelatex library and recommended packages (https://www.
 ↪overleaf.com/learn/latex/XeLaTeX)
!apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-generic-recommended

# When running this code it asks for permissions.  You must login to your␣
 ↪google account and
# allow colab to access your drive.
from google.colab import drive
drive.mount("/content/drive")
!cp "./drive/MyDrive/Colab Notebooks/Bi195PythonIntro.ipynb" ./

# This final line converts the file to a pdf
# Note that the format of this line is !jupyter nbconvert --to <format>␣
 ↪'NotebookName.ipynb'
!jupyter nbconvert --to pdf 'Bi195PythonIntro.ipynb'

# Once this cell has completed running, the pdf file should be ready to␣
 ↪download in the left file pane
# It can sometimes take a few minutes until the pdf file shows up
```

We highly recommend deleting the files that are created in the last cell, in order to more easily keep track of version. They will be automatically deleted when you disconnect from the runtime. These are "Bi195PythonIntro.ipynb" and "Bi195PythonIntro.pdf" for this case.

7.3 Method 3

This method also exports your work as a pdf, and uses much less code. This one has proven less consistent for me than Method 2, but for others this method is more consistent. Just choose whichever makes things easiest (and works) for you!

[11]:
```
# For more details refer to https://github.com/brpy/colab-pdf
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Bi195PythonIntro.ipynb')
```

7.4 Sharing

You can also share your work by clicking on the Share button in the top right corner, and adding the invitee's email.