4th International Cassiopea Workshop

Tutorial 4: Mathematical modeling and Python

Tutorial presenters

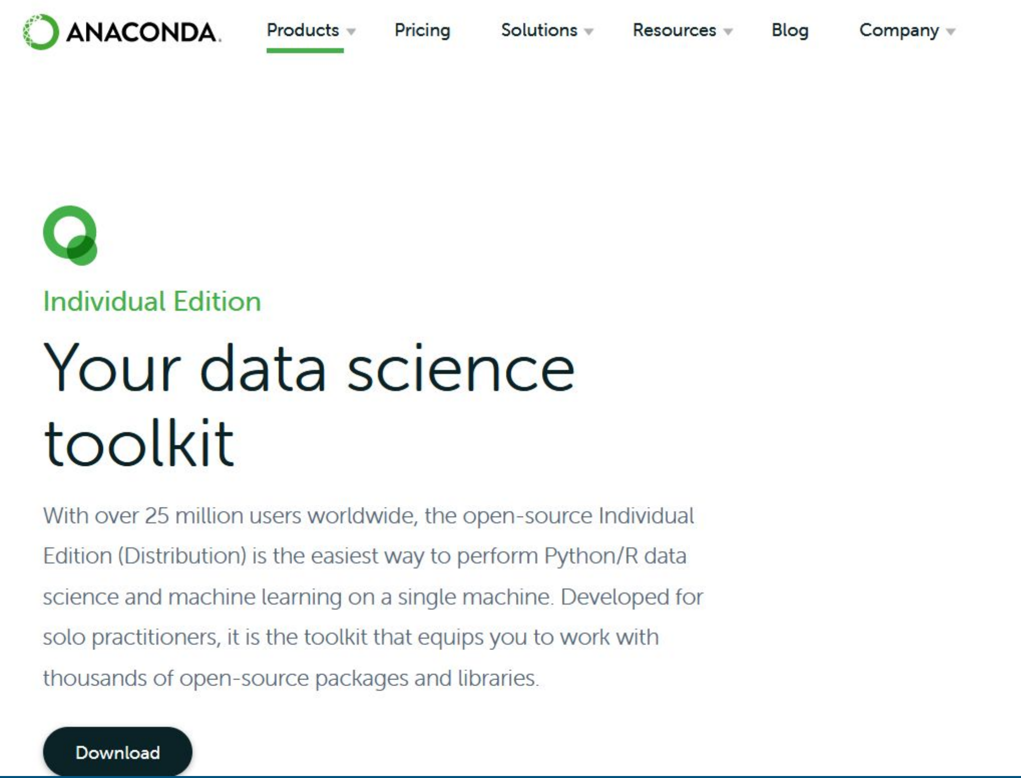
Christina Hamlet

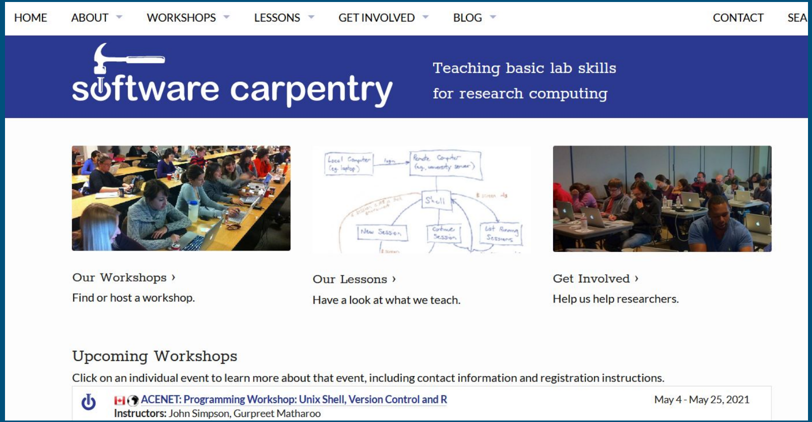
Bucknell University

Laura Miller

University of Arizona

Resources





https://software-carpentry.org/

Getting started

Install anaconda: https://www.anaconda.com/products/individual

Follow instructions for your operating system. Jupyter is auto-bundled.

If you are using a different Python distribution --

Mac instructions: go to terminal <pip install jupyterlab>

(pip should be installed with python. If not look here → https://pip.pypa.io/en/stable/installing/)

Getting started

In terminal, create a new directory and navigate to it.

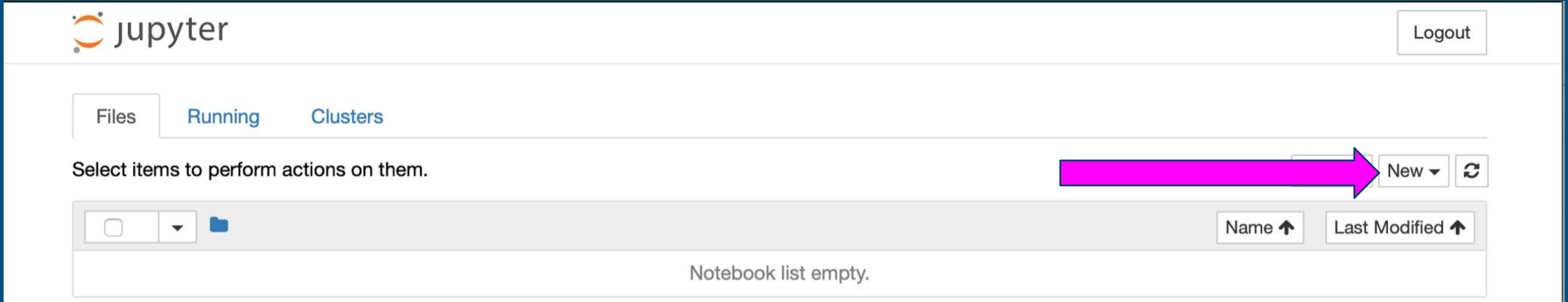
On a Mac:

<mkdir jupyterwork> is what I will be using

Type <jupyter notebook> to launch the interface

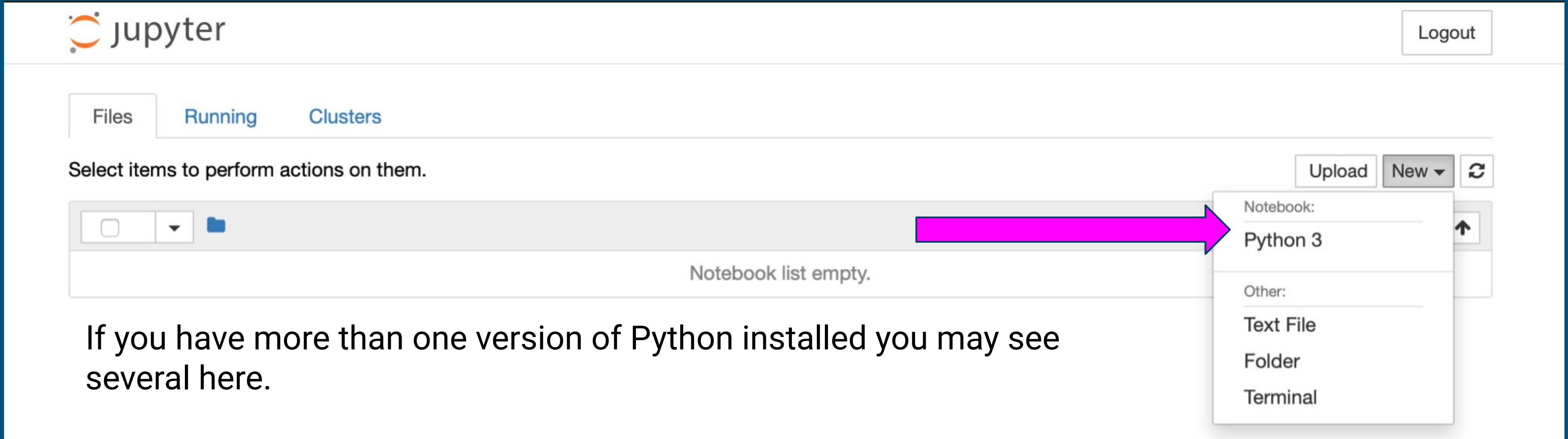
Set up kernel (first time)

Setting



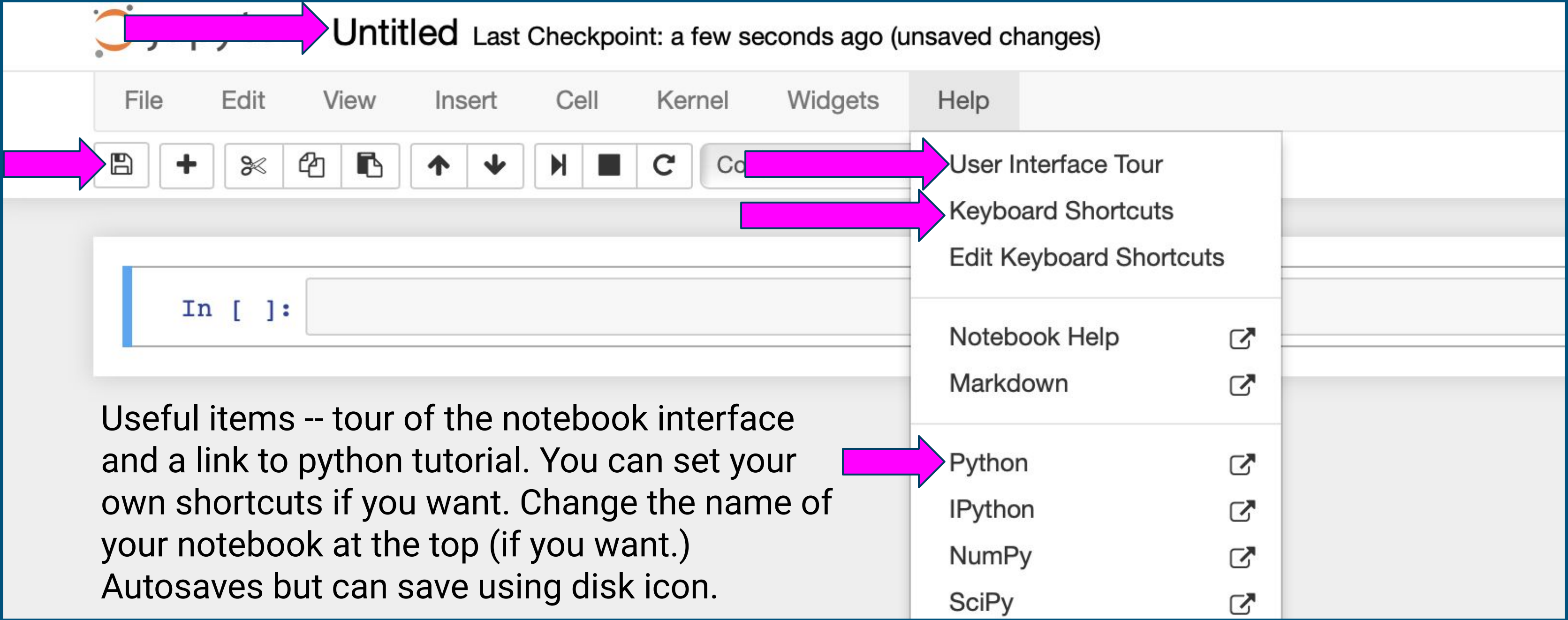
Click on “New” to create a new notebook

Setting



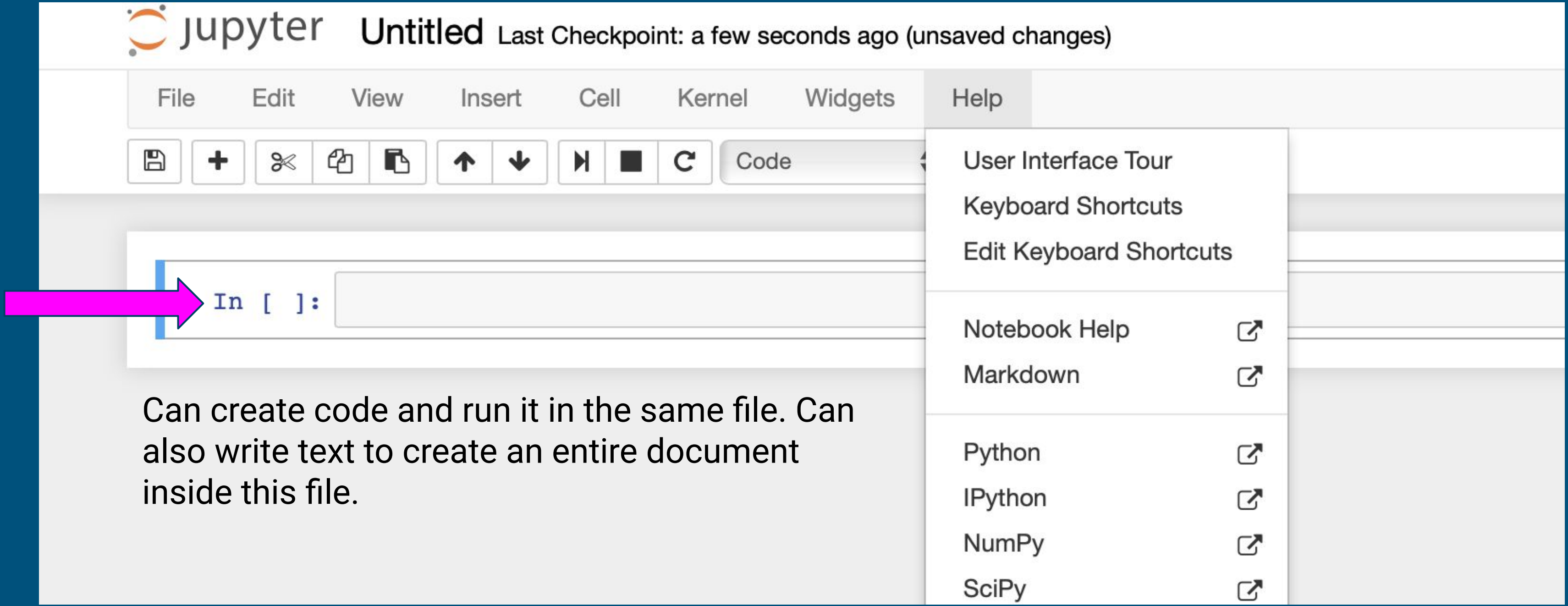
If you have more than one version of Python installed you may see several here.

Setting



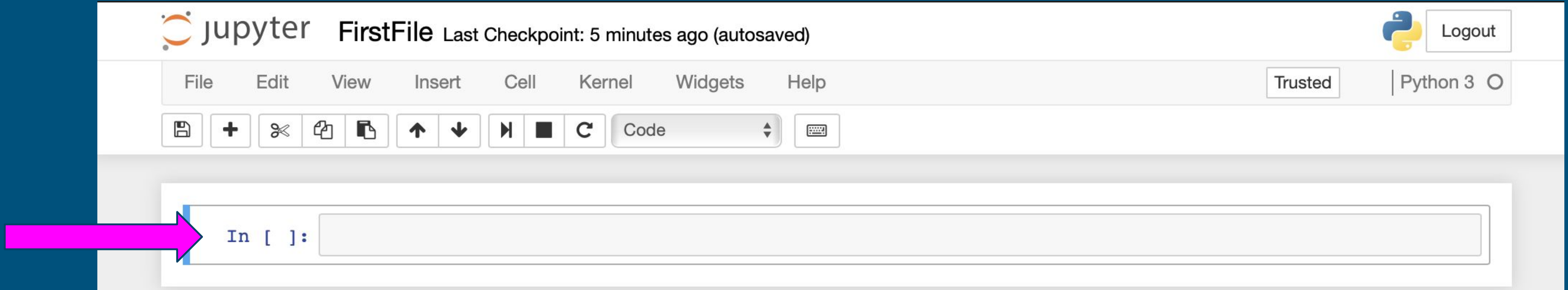
Useful items -- tour of the notebook interface and a link to python tutorial. You can set your own shortcuts if you want. Change the name of your notebook at the top (if you want.) Autosaves but can save using disk icon.

Setting



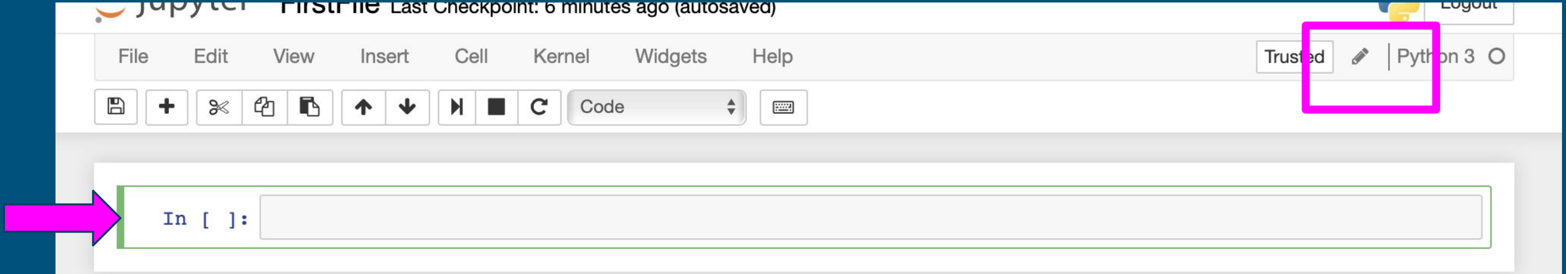
Can create code and run it in the same file. Can also write text to create an entire document inside this file.

Using command mode



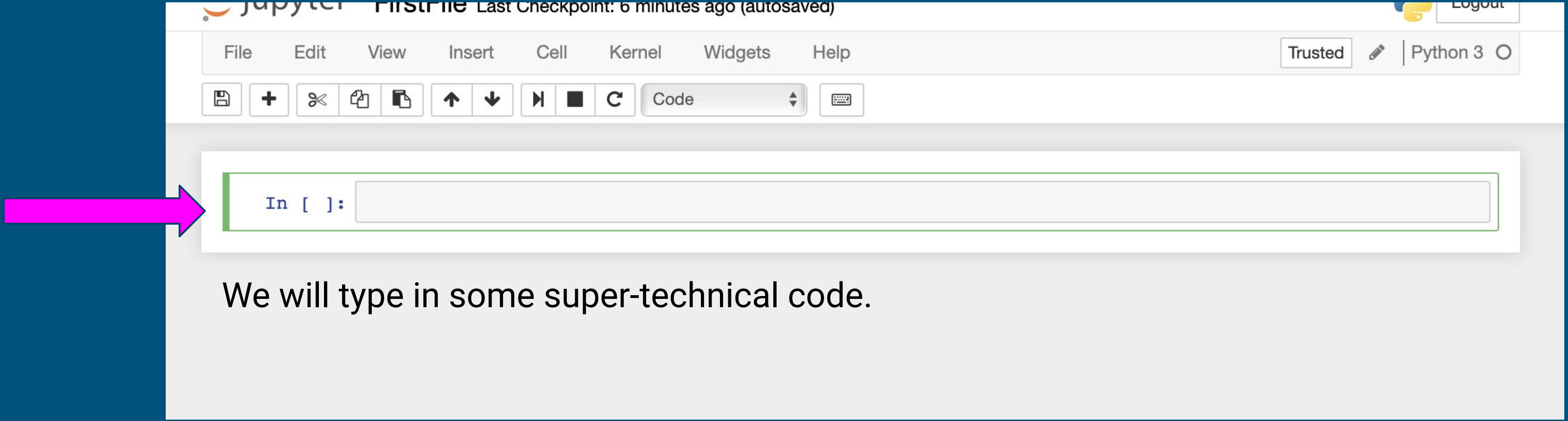
This is a code interface. When it’s not selected it is blue. Select it and it should turn green.

Using command mode

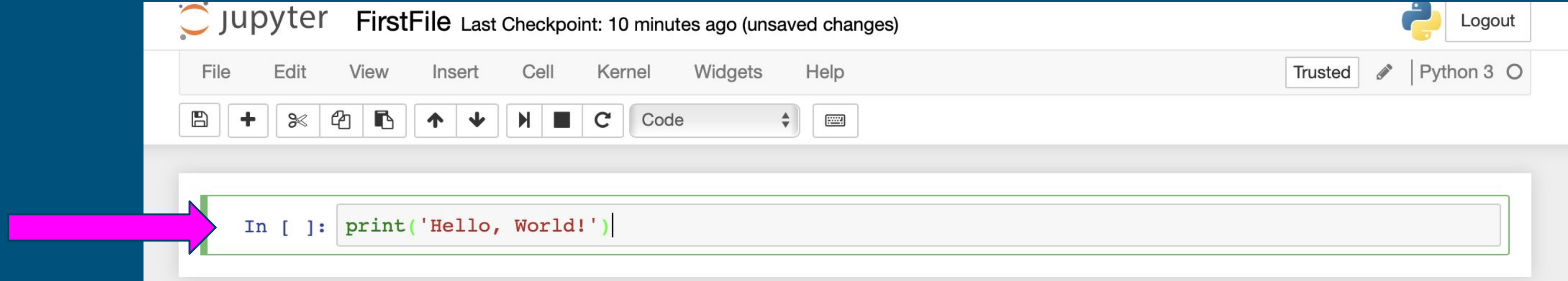


Click on window, it will turn green. (Edit mode) You will also see pencil in the top right corner

Using command mode

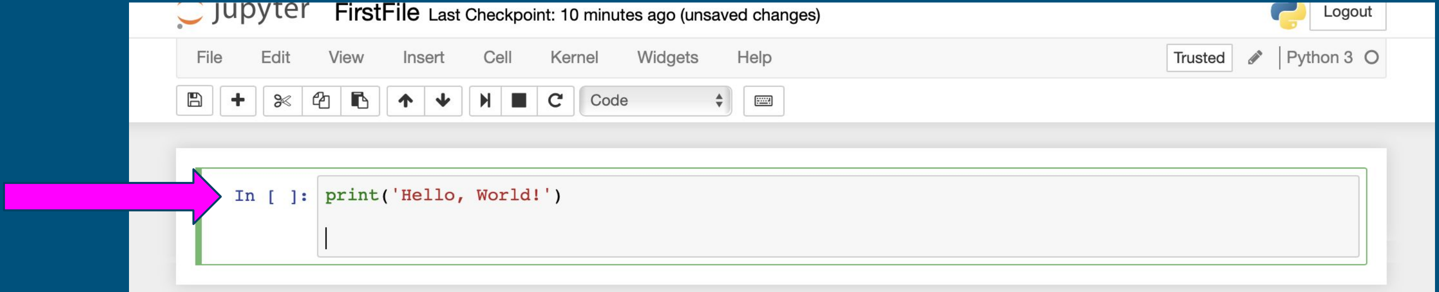


Using command mode



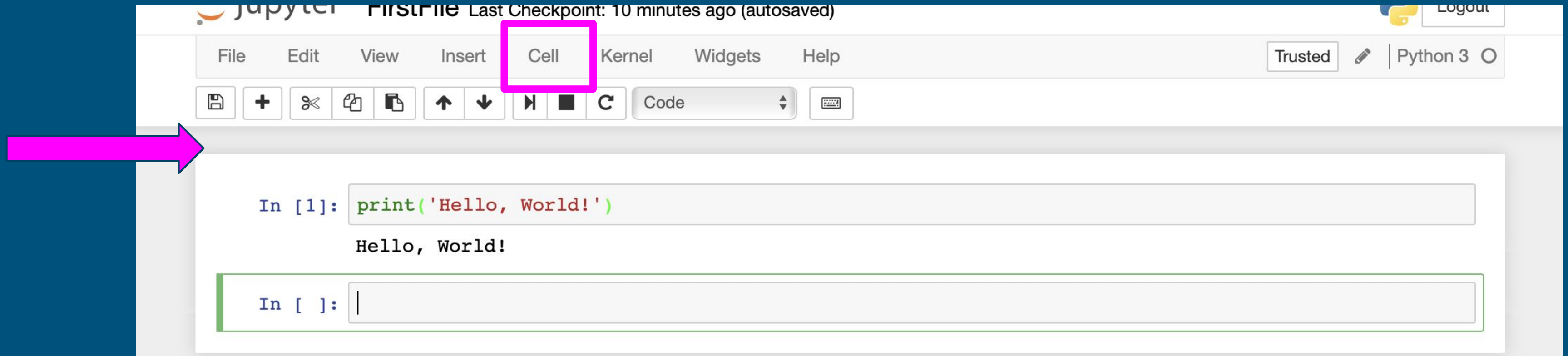
Type **print('Hello, World!')** into the box

Using command mode



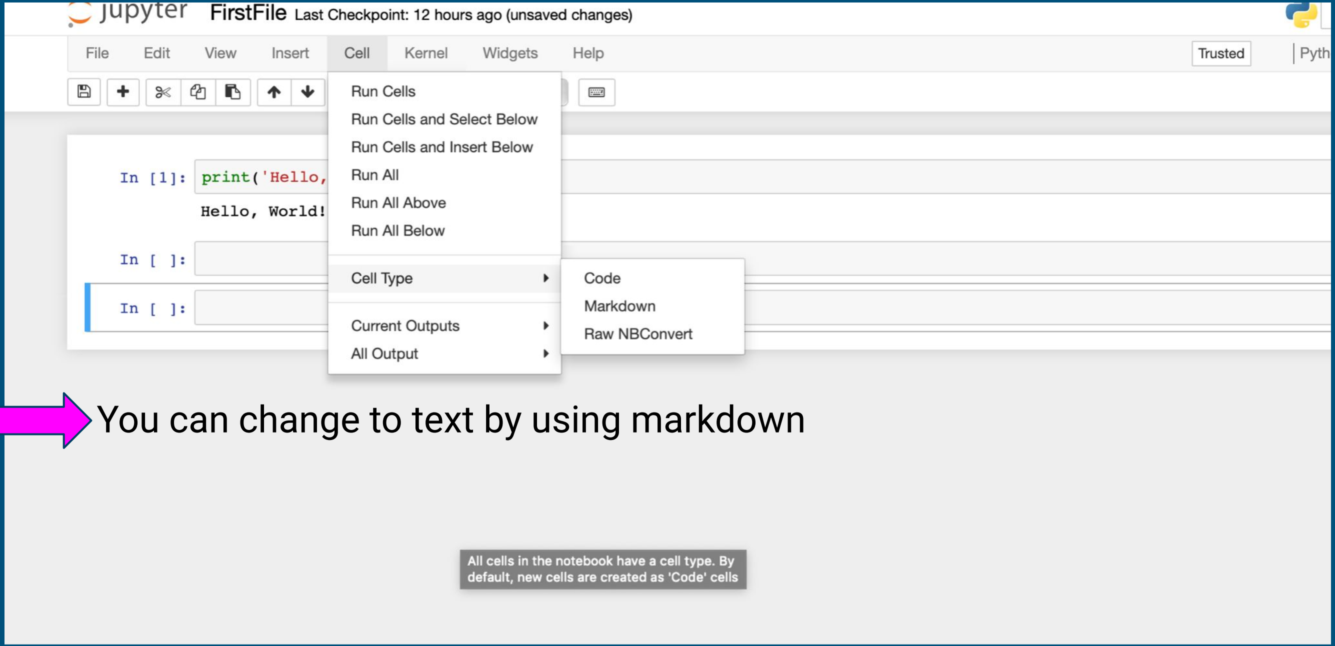
Pressing Enter/Return will just move to the next line..

Using command mode



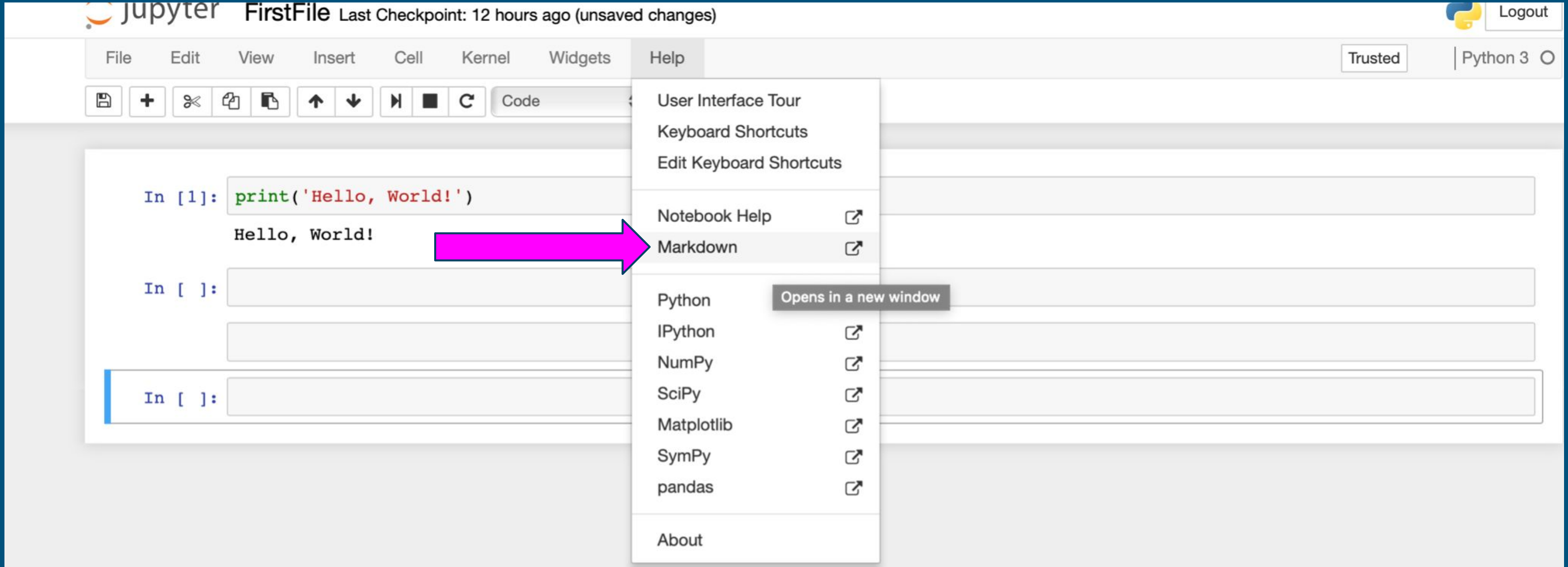
Pressing Ctrl/Command+Enter/Return will execute the commands inside the cell. And create a new cell below it. This can also be done from the “cell” menu.

Writing text



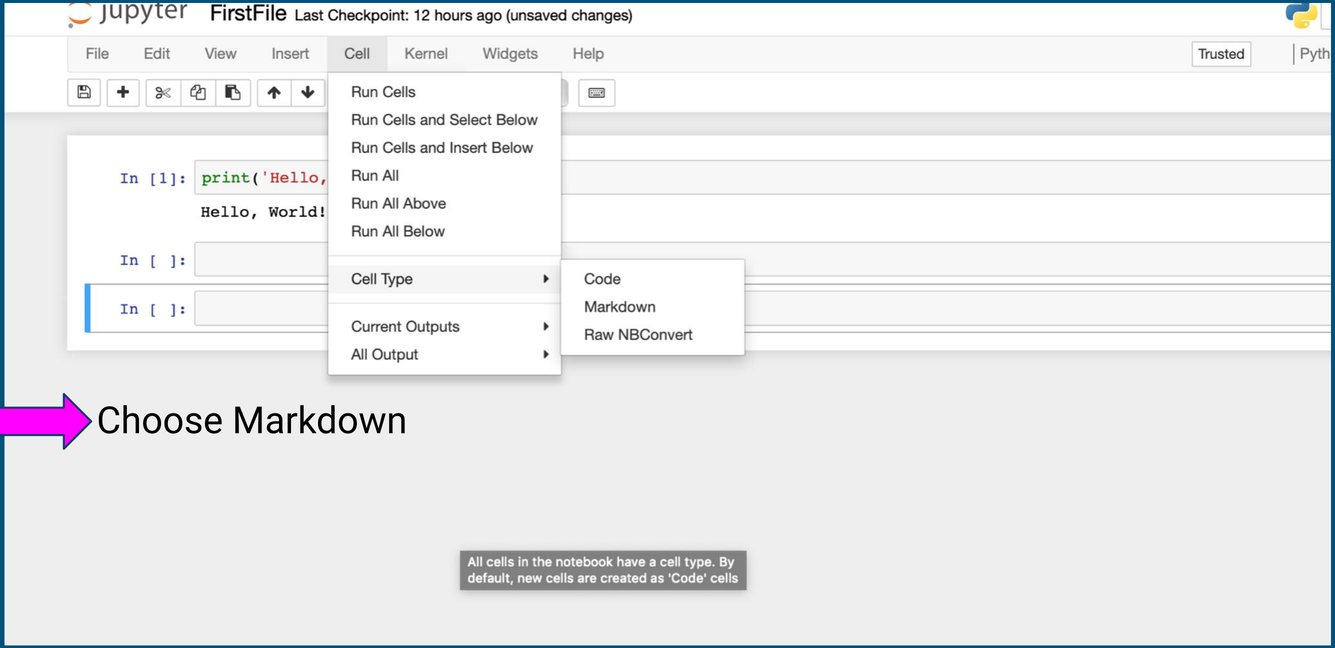
You can change to text by using markdown

Writing text



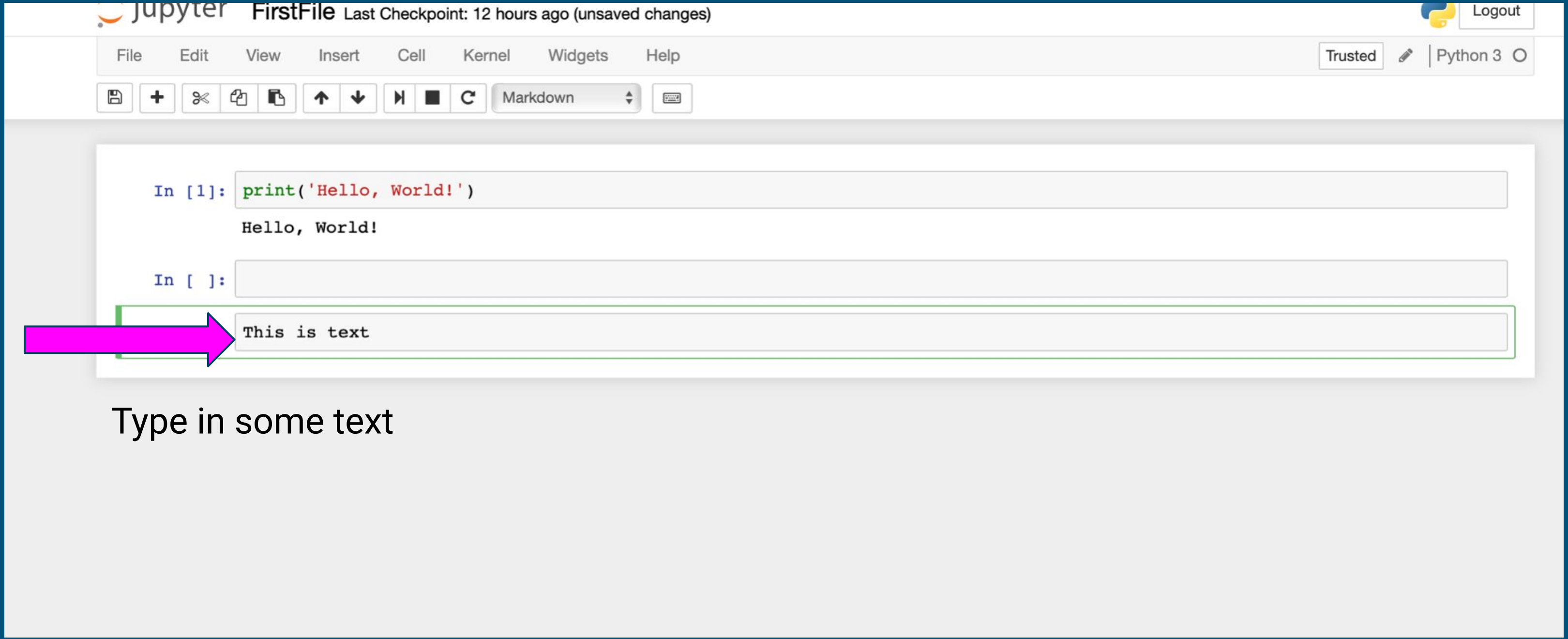
More information about Markdown is here. You can do fancy formatting.

Writing text

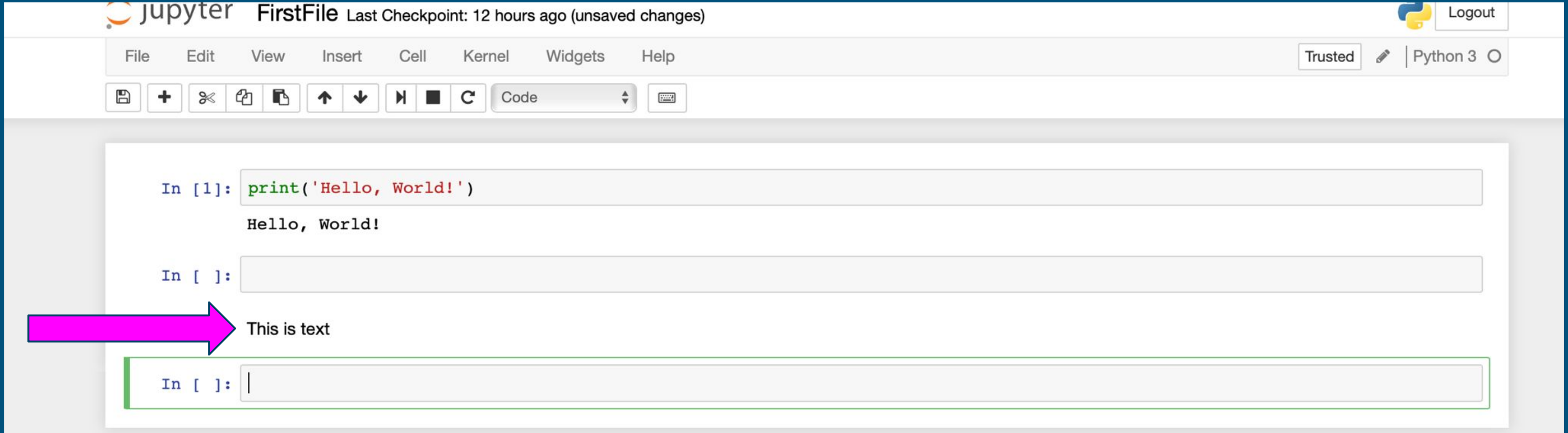


Choose Markdown

Writing text



Writing text



Press control/command+enter to turn it to text

For-loops

This is text

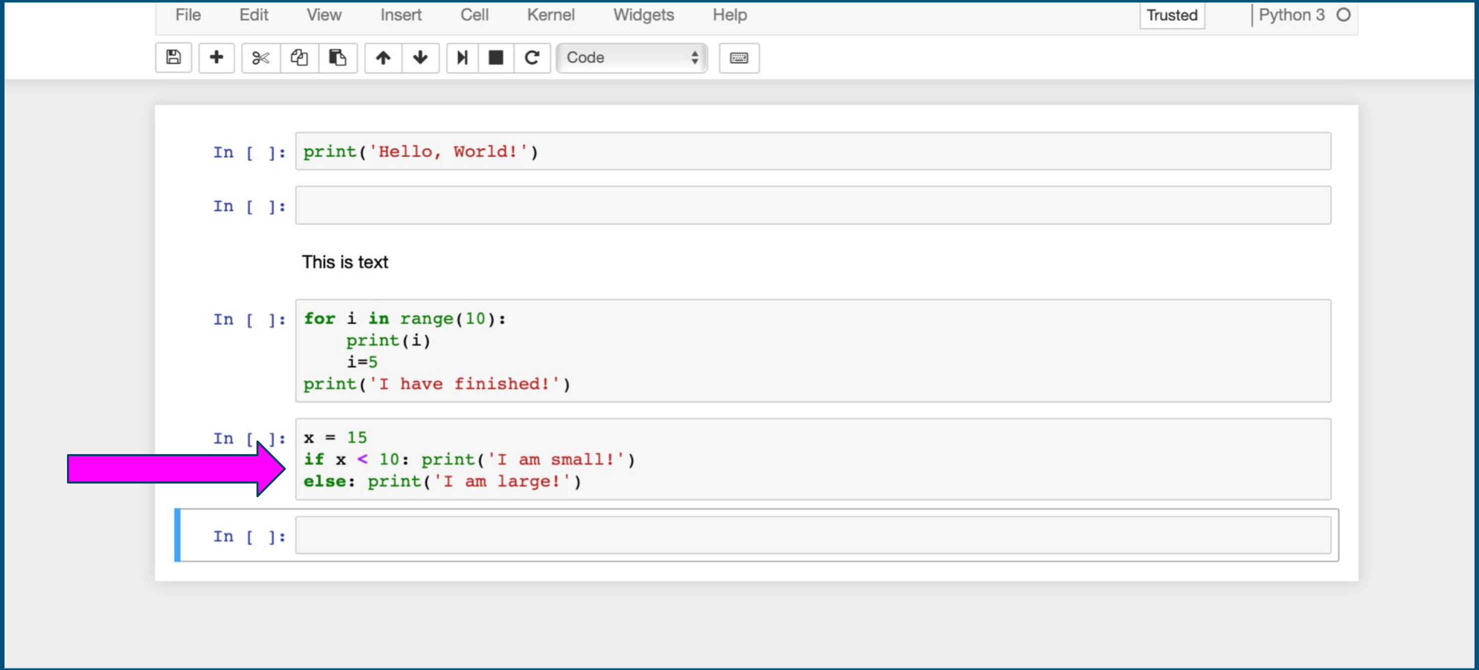
In [17]:
```python
for i in range(10):
    print(i)
    i=5
print('I left the loop!')
print(i)
print('I have finished!')
```

```
0
1
2
3
4
5
6
7
8
9
I left the loop!
5
I have finished!
```

Syntax of for-loop

Check: Why did the 5 print between the text lines?

If statement syntax



If Statement output

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

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

```
In [ ]:
```

This is text

```
In [ ]: for i in range(10):
            print(i)
            i=5
        print('I have finished!')
```

```
In [6]: x = 15
        if x < 10: print('I am small!')
        else: print('I am large!')
```

```
I am large!
```

```
In [ ]:
```

Test it: change x to 4 and re-execute the code. Then try other numbers

ODE example

I am large!

Now let's solve x'=2x on [0,1] with initial value of x(0) = 1

```
import numpy as np
import math
h = 0.1 #step size
total_time = 1.0
t = 0  #initial time
x = 1  #initial value
N_steps = math.ceil((1.0-0.0)/(h)) #determine number of steps
print("%0.3f" % t,'\t', "%0.3f" % x)
for i in range(N_steps):
    t=t+h
    x=x + h*(2*x)
    print("%0.3f" % t,'\t', "%0.3f" % x)
print("finished!")
```

```
0.000    1.000
0.100    1.200
0.200    1.440
0.300    1.728
0.400    2.074
0.500    2.488
0.600    2.986
0.700    3.583
0.800    4.300
0.900    5.160
1.000    6.192
finished!
```

Import needed packages/modules

```
In [ ]:
```

ODE example

I am large!

Now let's solve x'=2x on [0,1] with initial value of x(0) = 1

In [35]:
```
import numpy as np
import math
h = 0.1 #step size
total_time = 1.0
t = 0  #initial time
x = 1  #initial value
N_steps = math.ceil((1.0-0.0)/(h)) #determine number of steps
print("%0.3f" % t,'\t', "%0.3f" % x)
for i in range(N_steps):
    t=t+h
    x=x + h*(2*x)
    print("%0.3f" % t,'\t', "%0.3f" % x)
print("finished!")
```

```
0.000    1.000
0.100    1.200
0.200    1.440
0.300    1.728
0.400    2.074
0.500    2.488
0.600    2.986
0.700    3.583
0.800    4.300
0.900    5.160
1.000    6.192
finished!
```

Set step size and final time

In [ ]:

ODE example

I am large!

Now let's solve x'=2x on [0,1] with initial value of x(0) = 1

In [35]:

```
import numpy as np
import math
h = 0.1 #step size
total_time = 1.0
t = 0  #initial time
x = 1  #initial value
N_steps = math.ceil((1.0-0.0)/(h)) #determine number of steps
print("%0.3f" % t,'\t', "%0.3f" % x)
for i in range(N_steps):
    t=t+h
    x=x + h*(2*x)
    print("%0.3f" % t,'\t', "%0.3f" % x)
print("finished!")
```

```
0.000    1.000
0.100    1.200
0.200    1.440
0.300    1.728
0.400    2.074
0.500    2.488
0.600    2.986
0.700    3.583
0.800    4.300
0.900    5.160
1.000    6.192
finished!
```

Initial values

In [ ]:

ODE example

I am large!

Now let's solve x'=2x on [0,1] with initial value of x(0) = 1

In [35]:
```
import numpy as np
import math
h = 0.1 #step size
total_time = 1.0
t = 0  #initial time
x = 1  #initial value
N_steps = math.ceil((1.0-0.0)/(h)) #determine number of steps
print("%0.3f" % t,'\t', "%0.3f" % x)
for i in range(N_steps):
    t=t+h
    x=x + h*(2*x)
    print("%0.3f" % t,'\t', "%0.3f" % x)
print("finished!")
```

```
0.000     1.000
0.100     1.200
0.200     1.440
0.300     1.728
0.400     2.074
0.500     2.488
0.600     2.986
0.700     3.583
0.800     4.300
0.900     5.160
1.000     6.192
finished!
```

Calculate number of steps needed

In [ ]:

ODE example

I am large!

Now let's solve x'=2x on [0,1] with initial value of x(0) = 1

In [35]:
```
import numpy as np
import math
h = 0.1 #step size
total_time = 1.0
t = 0  #initial time
x = 1  #initial value
N_steps = math.ceil((1.0-0.0)/(h)) #determine number of steps
print("%0.3f" % t,'\t', "%0.3f" % x)
for i in range(N_steps):
    t=t+h
    x=x + h*(2*x)
    print("%0.3f" % t,'\t', "%0.3f" % x)
print("finished!")
```

```
0.000	1.000
0.100	1.200
0.200	1.440
0.300	1.728
0.400	2.074
0.500	2.488
0.600	2.986
0.700	3.583
0.800	4.300
0.900	5.160
1.000	6.192
finished!
```

For-loop calculates values at each step and outputs them

In [ ]:

ODE example

```
I am large!
```

Now let's solve x'=2x on [0,1] with initial value of x(0) = 1

In [35]:
```python
import numpy as np
import math
h = 0.1 #step size
total_time = 1.0
t = 0  #initial time
x = 1  #initial value
N_steps = math.ceil((1.0-0.0)/(h)) #determine number of steps
print("%0.3f" % t,'\t', "%0.3f" % x)
for i in range(N_steps):
    t=t+h
    x=x + h*(2*x)
    print("%0.3f" % t,'\t', "%0.3f" % x)
print("finished!")
```

```
0.000    1.000
0.100    1.200
0.200    1.440
0.300    1.728
0.400    2.074
0.500    2.488
0.600    2.986
0.700    3.583
0.800    4.300
0.900    5.160
1.000    6.192
finished!
```

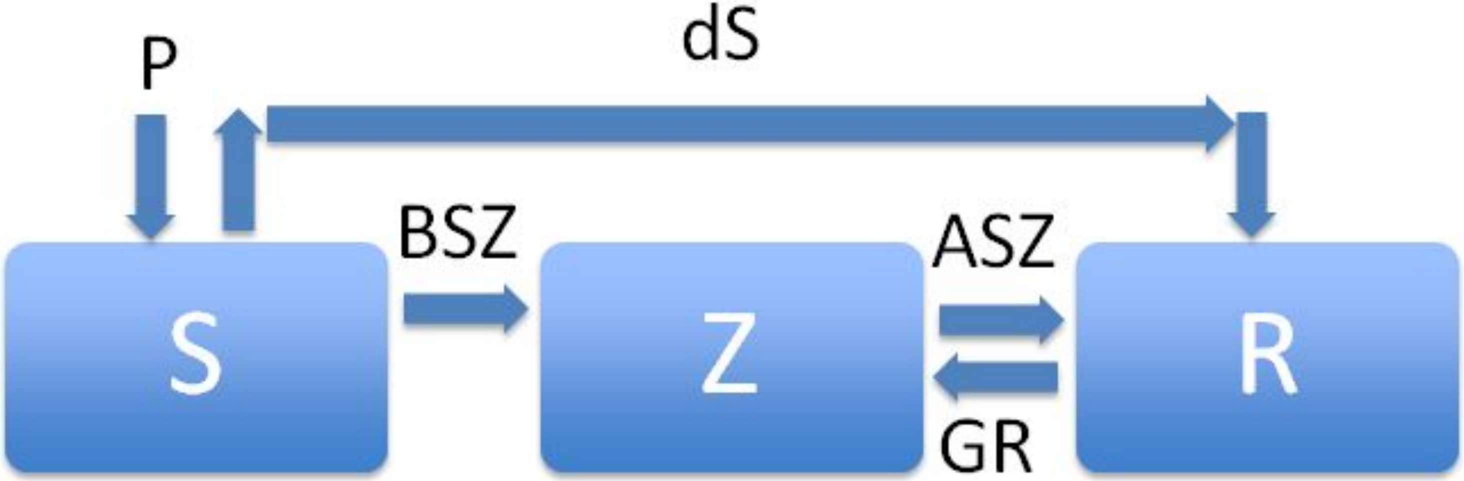
Prints output

Activity up next will show you syntax for working with arrays and plotting.

In [ ]:

Zombie Invasion!

A system of ODEs can be used to model a "zombie invasion", using the equations specified in Munz et al. 2009. This is essentially equivalent to an SIR model used to simulate the spread of disease. Such models have been applied to understand the spread of disease in corals and many other organisms (including humans!)



S: the number of susceptible victims

Z: the number of zombies

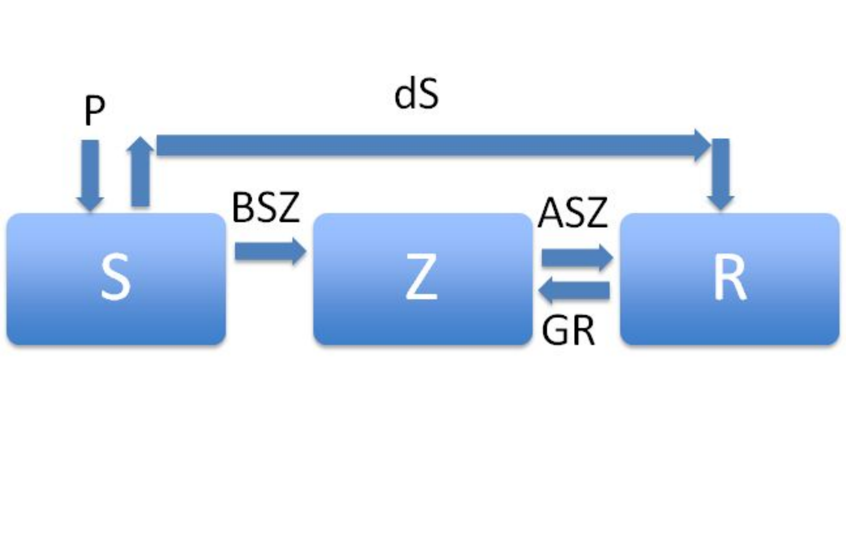
R: the number of people "killed"

Zombie model

dS/dt = P - B*S*Z - d*S

dZ/dt = B*S*Z + G*R - A*S*Z

dR/dt = d*S + A*S*Z - G*R



S: the number of susceptible victims

Z: the number of zombies

R: the number of people "killed"

P: the population birth rate

d: the chance of a natural death

B: the chance the "zombie disease" is transmitted (an alive person becomes a zombie)

G: the chance a dead person is resurrected into a zombie

A: the chance a zombie is totally destroyed

Import packages

```
# zombie apocalypse modeling
# This code has been modified from
http://www.scipy.org/Cookbook/Zombie_Apocalypse_ODEINT

#Below we import packages needed to solve and plot ODE's
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from pylab import savefig

# turn on interactive mode for matplotlib
plt.ion()
```

Set parameter values

```
# set the parameter values
P = 0       # birth rate
d = 0.0001  # natural death percent (per day)
B = 0.0095  # transmission percent  (per day)
G = 0.0001  # resurect percent (per day)
A = 0.0001  # destroy percent  (per day)
N = 1000         # Number of time steps
Tf = 10          # final time (days)
```

Set initial conditions

```
# initial conditions
S0 = 499                # initial population
Z0 = 1                  # initial zombie population
R0 = 0                  # initial death population
y0 = [S0, Z0, R0]       # initial condition vector
t  = np.linspace(0, Tf, N)   # time grid
```

Define system of differential equations

```
# solve the system dy/dt = f(y, t)
def f(y, t):
	Si = y[0]
	Zi = y[1]
	Ri = y[2]
	# the model equations (see Munz et al. 2009)
	f0 = P - B*Si*Zi - d*Si
	f1 = B*Si*Zi + G*Ri - A*Si*Zi
	f2 = d*Si + A*Si*Zi - G*Ri
	return [f0, f1, f2]
```

Solve the differential equations

```
# solve the DEs
soln = odeint(f, y0, t)
S = soln[:, 0]
Z = soln[:, 1]
R = soln[:, 2]
```

Plot the results

```
# plot results
plt.figure()
plt.plot(t, S, label='Living')
plt.plot(t, Z, label='Zombies')
plt.xlabel('Days from outbreak')
plt.ylabel('Population')
plt.title('Zombie Apocalypse - No Init. Dead Pop.; No New
Births.')
plt.legend(loc=0)
savefig('R0=0-P=0.png', dpi=100)
```

Try changing things

Now change the initial conditions so that some of the initial population is dead at the beginning. To do this, change the code as follows:

R0 = 0.01*S0 # 1% of initial pop is dead

How did increasing this initial condition affect the time at which there are more zombies than people?

Now change the initial conditions so that there are 10 new births daily.

P = 10 # 10 new births daily

New zombie dynamics

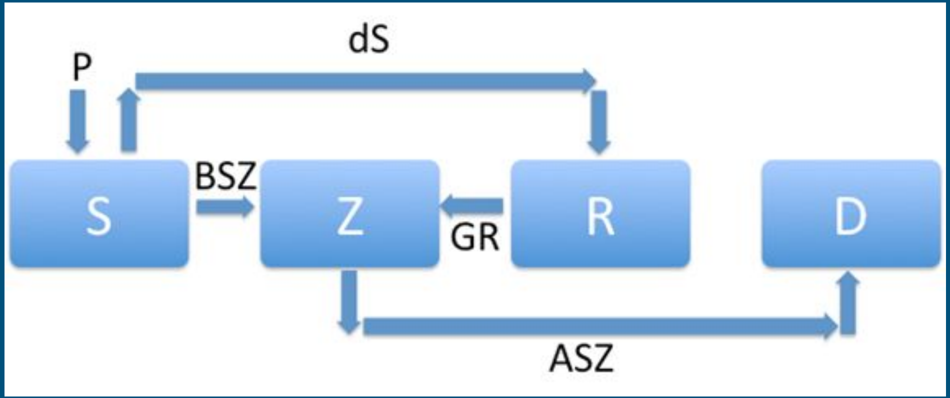
Save the zombie.py file as zombie.py. Set the parameters back to where we started (P=0, R0=0). Let’s change the dynamics so that when zombies are killed, they cannot be resurrected again. We will need to create a new compartment for this group, and let D be the number of completely removed (cannot be resurrected). The system of equations can be rewritten as follows:

dS/dt = P - B*S*Z - d*S

dZ/dt = B*S*Z + G*R - A*S*Z

dR/dt = d*S - G*R

dD/dt = A*S*Z



Changes in the code

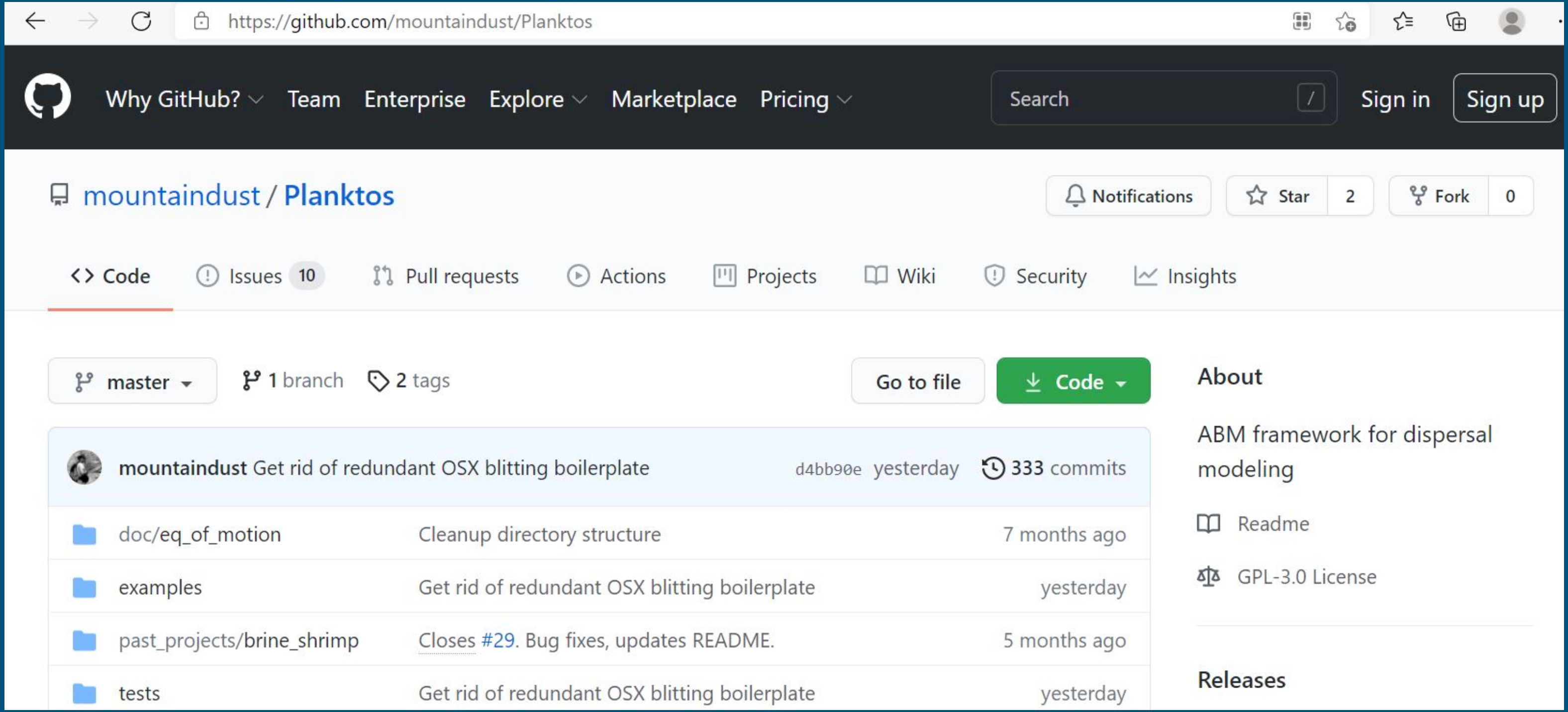
We need to change add dD/dt and its initial condition in several places in the code. Here is an overview of the changes you need to make:

1. In the initial conditions, add D0=0.
2. In the vector that holds the initial conditions, y0, add D0.
3. In def f(y,t), add Di=y[3] to hold the values of the totally removed people.
4. In def f(y,t), add f3 = A*Si*Zi, and change the equation for f2 to f2 = f2 = d*Si - G*Ri.
5. The vector that gets returned should now be [f0, f1, f2, f3]
6. Set D = soln[:, 3].
7. Plot the removed and totally removed by adding the lines

```
plt.plot(t, R, label='Removed')
plt.plot(t, D, label='Totally Removed')
```

Planktos

[GitHub - mountaindust/Planktos: ABM framework for dispersal modeling](https://github.com/mountaindust/Planktos)



ib2d

[GitHub - nickabattista/IB2d: An easy to use immersed boundary method in 2D, with full implementations in MATLAB and Python that contains over 60 built-in examples, including multiple options for fiber-structure models and advection-diffusion, Boussinesq approximations, and/or artificial forcing.]