



# 4th International Cassiopea Workshop



Tutorial 4: Mathematical modeling  
and Python



# Tutorial presenters

Christina Hamlet

Bucknell University

Laura Miller

University of Arizona

---

# Resources

[Products ▾](#)[Pricing](#)[Solutions ▾](#)[Resources ▾](#)[Blog](#)[Company ▾](#)


Individual Edition


## Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.


[Download](#)

[HOME](#) [ABOUT ▾](#) [WORKSHOPS ▾](#) [LESSONS ▾](#) [GET INVOLVED ▾](#) [BLOG ▾](#) [CONTACT](#) [SEARCH](#)


 **software carpentry** Teaching basic lab skills  
for research computing



[Our Workshops ▸](#)  
Find or host a workshop.




[Our Lessons ▸](#)  
Have a look at what we teach.



[Get Involved ▸](#)  
Help us help researchers.

### Upcoming Workshops

Click on an individual event to learn more about that event, including contact information and registration instructions.

 **ACENET: Programming Workshop: Unix Shell, Version Control and R**  
Instructors: John Simpson, Gurpreet Matharoo

May 4 - May 25, 2021

<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:

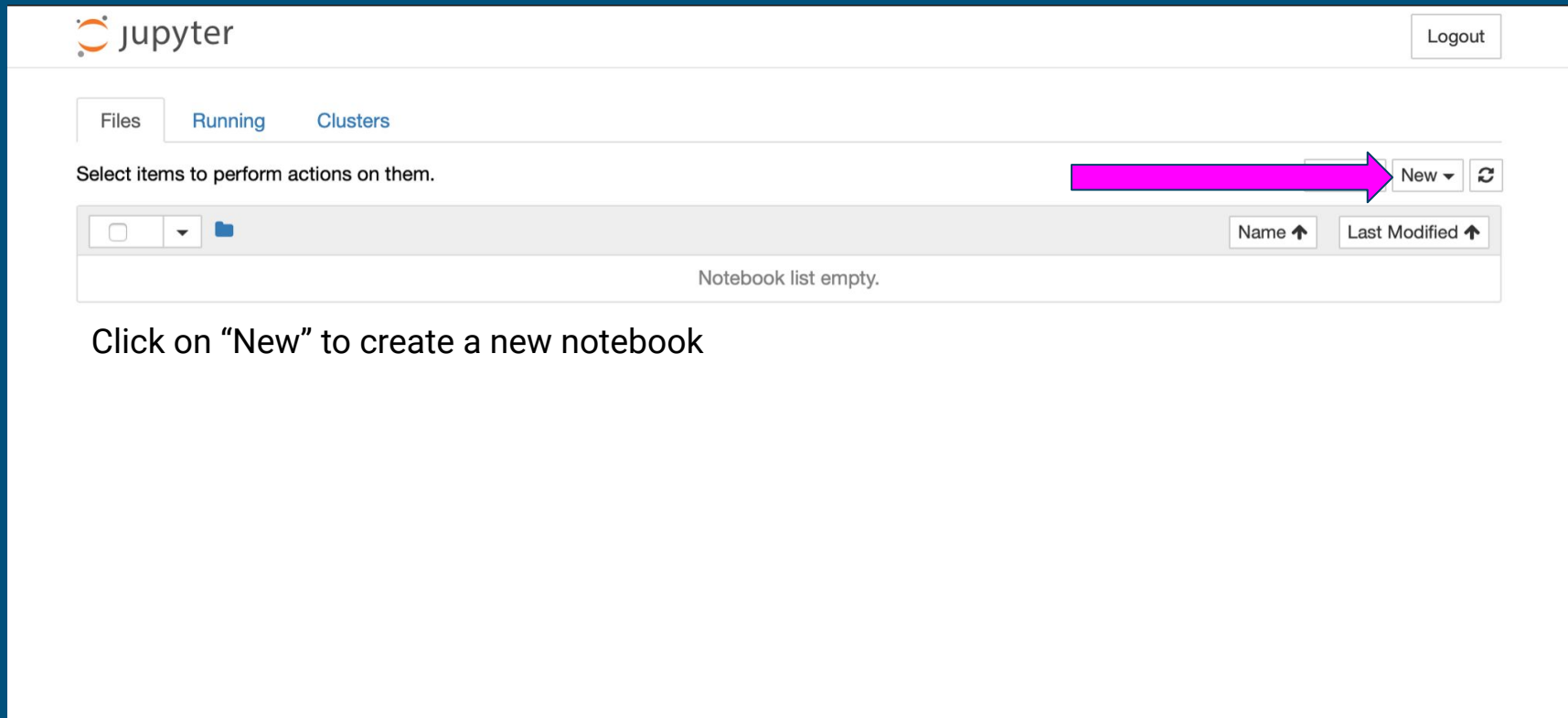
`<mkdir jupyterwork>` is what I will be using

Type `<jupyter notebook>` to launch the interface

Set up kernel (first time)



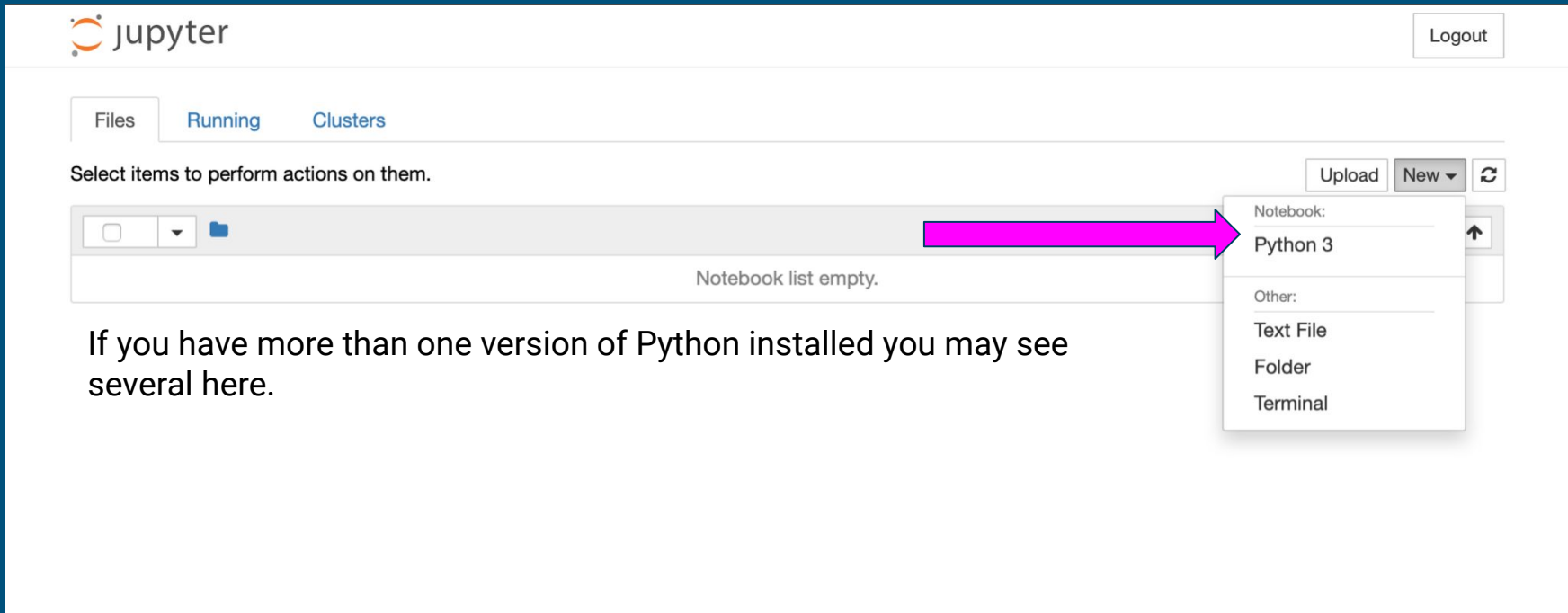
# Setting



The screenshot displays the JupyterLab web interface. At the top left is the Jupyter logo and the word "jupyter". At the top right is a "Logout" button. Below the header are three tabs: "Files", "Running", and "Clusters". The "Files" tab is active. Below the tabs, the text "Select items to perform actions on them." is followed by a red arrow pointing to a "New" button with a dropdown arrow and a refresh icon. Below this is a file browser area with a search bar, a dropdown menu, and a folder icon. To the right of the file browser are two buttons: "Name ↑" and "Last Modified ↑". The text "Notebook list empty." is centered below the file browser.

Click on "New" to create a new notebook

# Setting



The image shows the JupyterLab web interface. At the top left is the Jupyter logo and the word "jupyter". At the top right is a "Logout" button. Below the header are three tabs: "Files", "Running", and "Clusters". The "Files" tab is active. Below the tabs is a message: "Select items to perform actions on them." Below this is a file browser area with a search bar, a dropdown menu, and a folder icon. A pink arrow points from the file browser area to the "New" button in the top right. The "New" button has a dropdown menu open, showing options: "Notebook:", "Python 3", "Other:", "Text File", "Folder", and "Terminal".

jupyter

Logout

Files Running Clusters

Select items to perform actions on them.

Upload New ↻

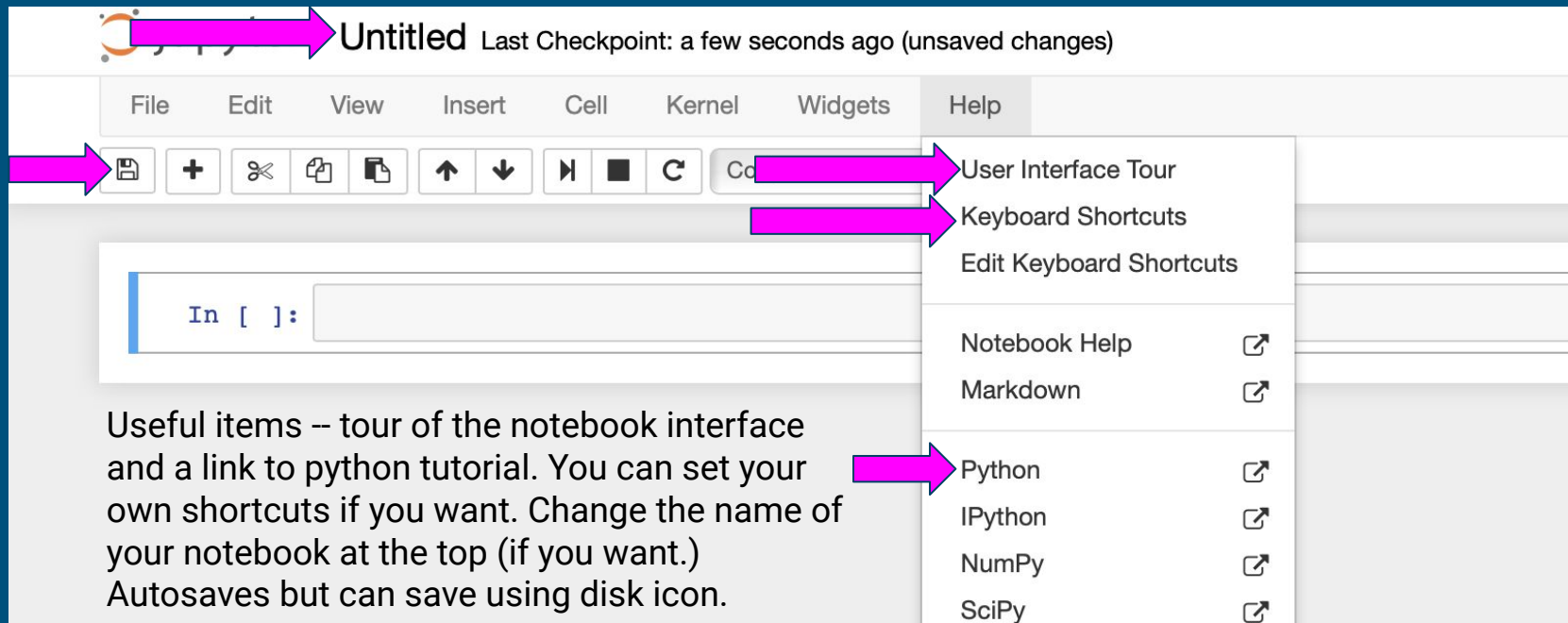
Notebook:  
Python 3

Other:  
Text File  
Folder  
Terminal

Notebook list empty.

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

# Setting

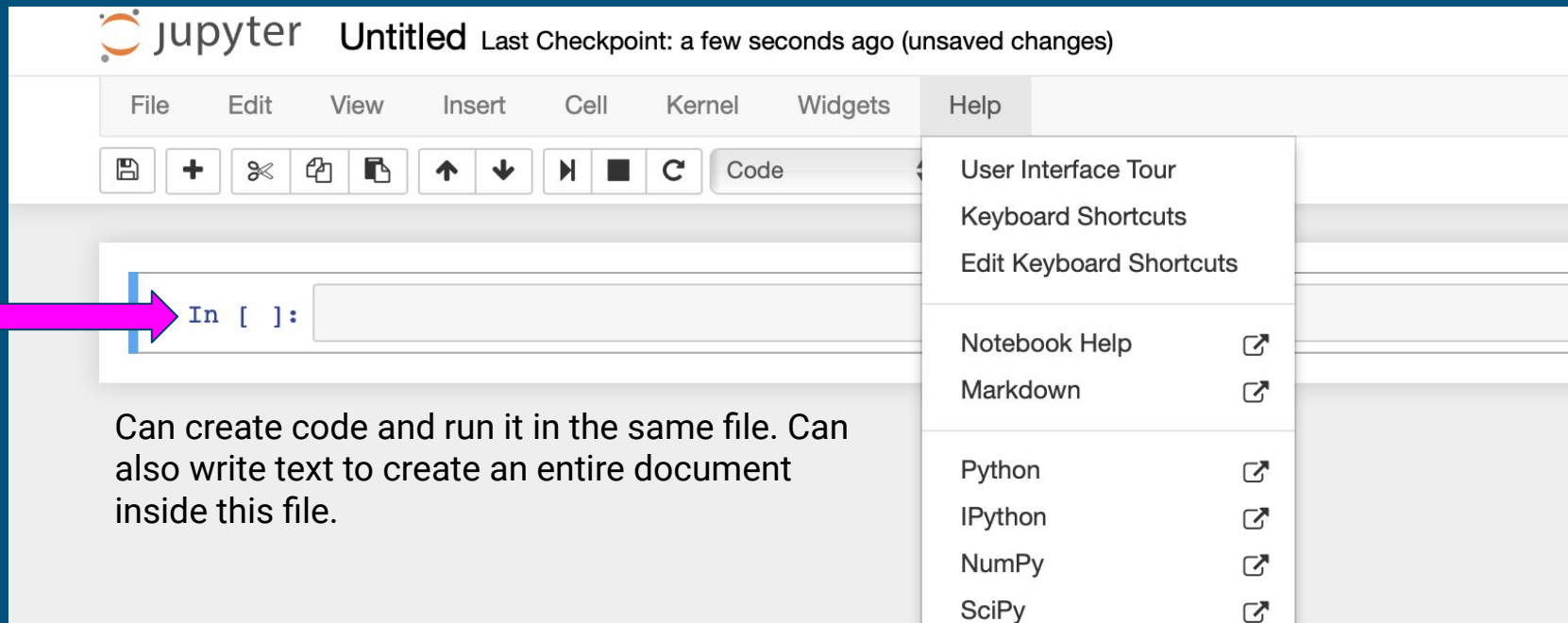


The screenshot shows the Jupyter Notebook interface. At the top, the title bar says "Untitled" followed by "Last Checkpoint: a few seconds ago (unsaved changes)". Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". The "Help" menu is open, showing options: "User Interface Tour", "Keyboard Shortcuts", "Edit Keyboard Shortcuts", "Notebook Help", "Markdown", "Python", "IPython", "NumPy", and "SciPy". A pink arrow points to the "File" menu, and another pink arrow points to the "Save" icon (a floppy disk) in the toolbar. A third pink arrow points to the "Python" option in the "Help" menu. Below the toolbar is a code input area with "In [ ]:" followed by a text box. Below the code input area is a text box containing the following text:

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.



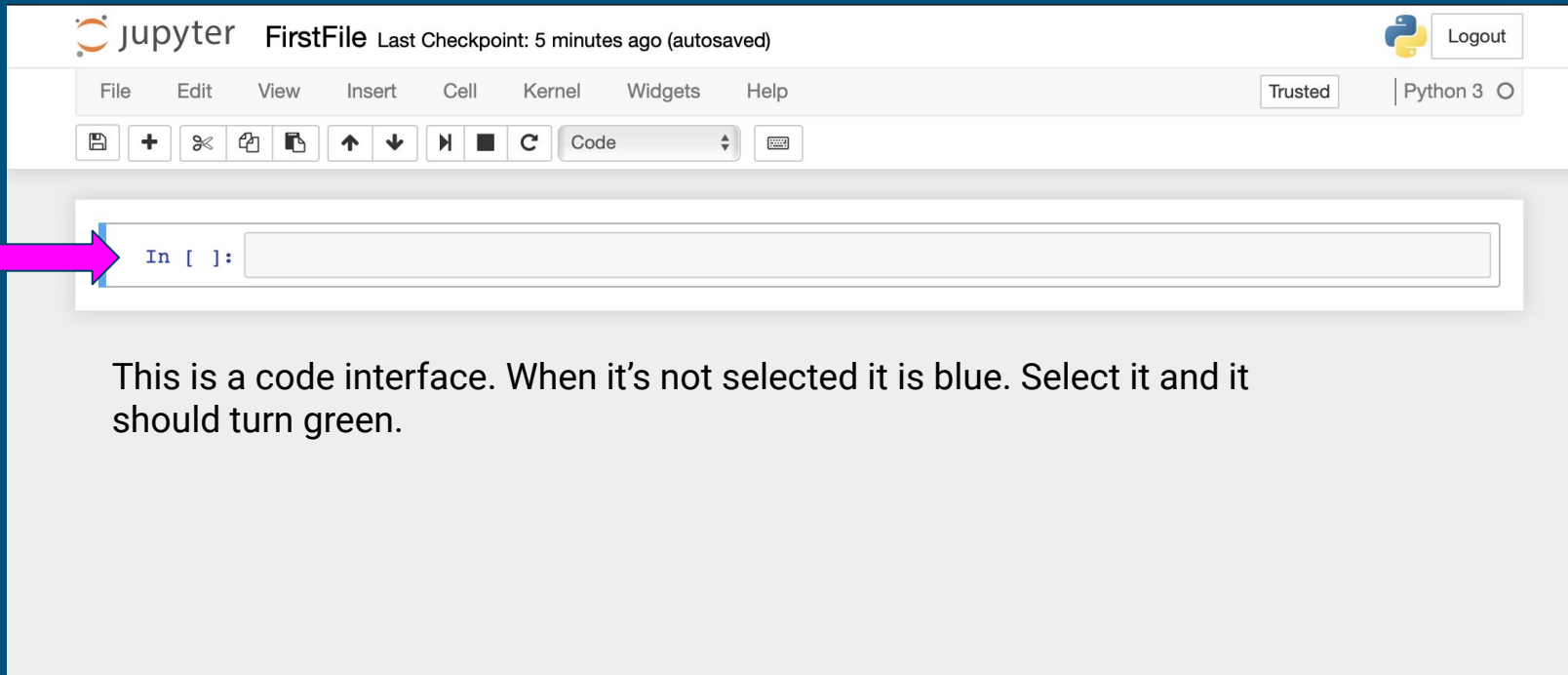
# Setting



The screenshot shows the Jupyter Notebook interface. At the top, the title bar reads "jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)". Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". The "Help" menu is open, displaying a list of options: "User Interface Tour", "Keyboard Shortcuts", "Edit Keyboard Shortcuts", "Notebook Help" (with an external link icon), "Markdown" (with an external link icon), "Python" (with an external link icon), "IPython" (with an external link icon), "NumPy" (with an external link icon), and "SciPy" (with an external link icon). In the main workspace, a code cell is visible with the prompt "In [ ]:". A pink arrow points to this prompt. Below the code cell, there is a text box containing the following text:

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

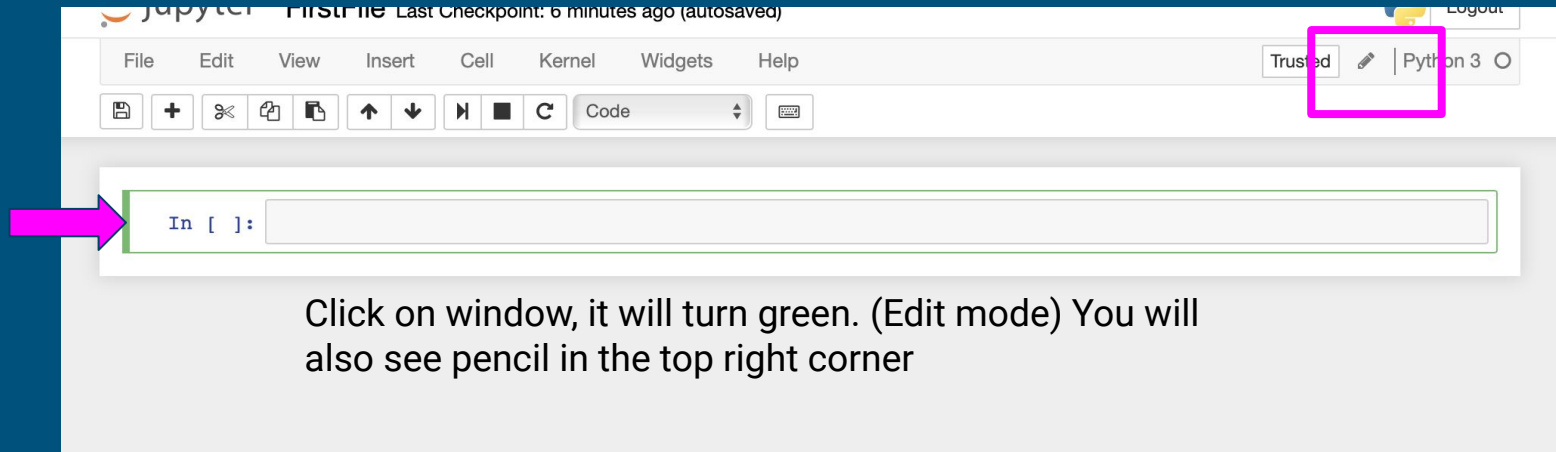
# Using command mode



The screenshot displays the Jupyter web interface. At the top, the header shows the Jupyter logo, the filename 'FirstFile', and a status message 'Last Checkpoint: 5 minutes ago (autosaved)'. On the right, there is a 'Logout' button and a Python 3 environment selector. Below the header is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A 'Trusted' status indicator is also present. Under the menu bar is a toolbar with icons for saving, adding, undo, redo, copy, paste, and navigation. The main area contains a single code cell. The cell's left margin is highlighted with a blue vertical bar, and a pink arrow points to it from the left. The cell's content area is empty and has a light gray background. Below the cell, there is a text explanation.

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

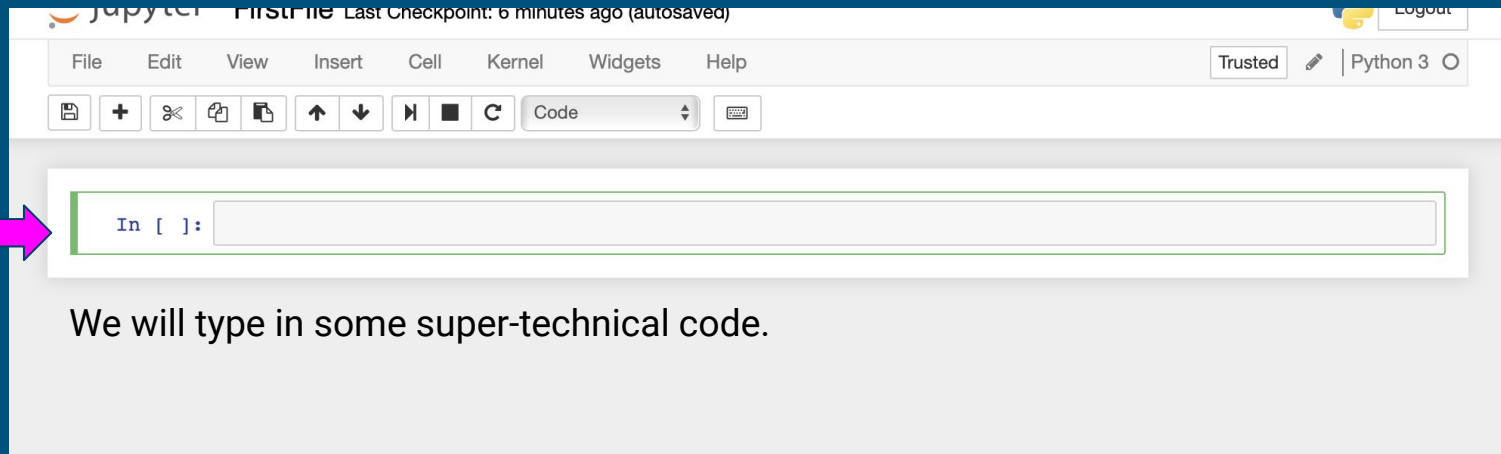
# Using command mode



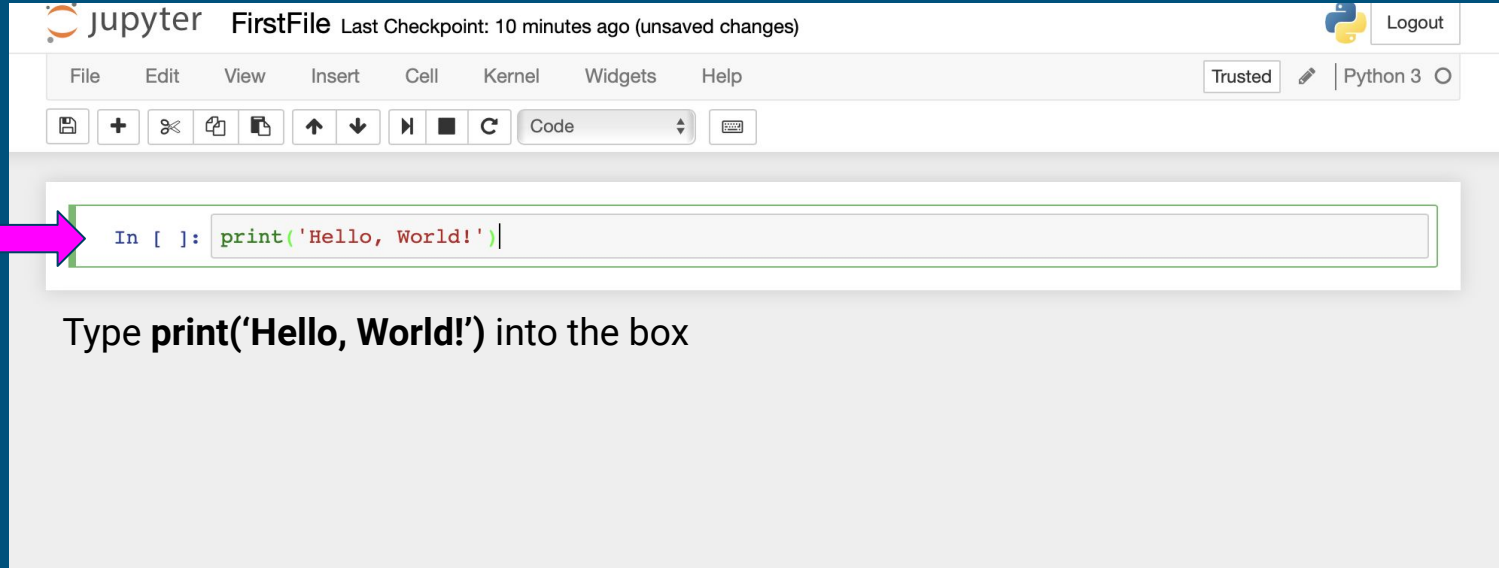
The screenshot displays the JupyterLab web interface. At the top, the header shows the Jupyter logo, the file name 'FirstFile', and the status 'Last Checkpoint: 6 minutes ago (autosaved)'. Below this is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar, there is a 'Trusted' button, a pencil icon, and the text 'Python 3'. A pink box highlights this top right area. Below the menu bar is a toolbar with various icons for file operations and cell execution. The main area contains a code cell with the prompt 'In [ ]:'. A pink arrow points to the left margin of this cell. Below the code cell, there is a text box with the instruction: 'Click on window, it will turn green. (Edit mode) You will also see pencil in the top right corner'.

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

# Using command mode



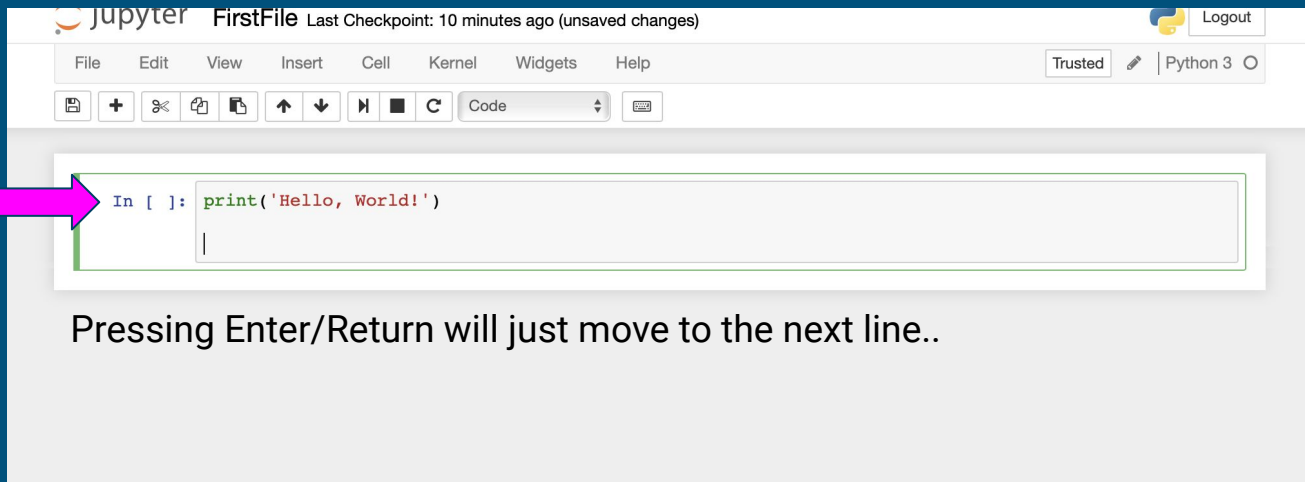
# Using command mode



The image shows a Jupyter Notebook interface. At the top, the title bar reads "Jupyter FirstFile" and "Last Checkpoint: 10 minutes ago (unsaved changes)". Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for "Trusted", a pencil icon, and "Python 3". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and a dropdown menu currently set to "Code". The main area of the notebook contains a single code cell. A pink arrow points to the left side of the code cell, indicating it is in command mode. The code cell contains the text "In [ ]: print('Hello, World!')".

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

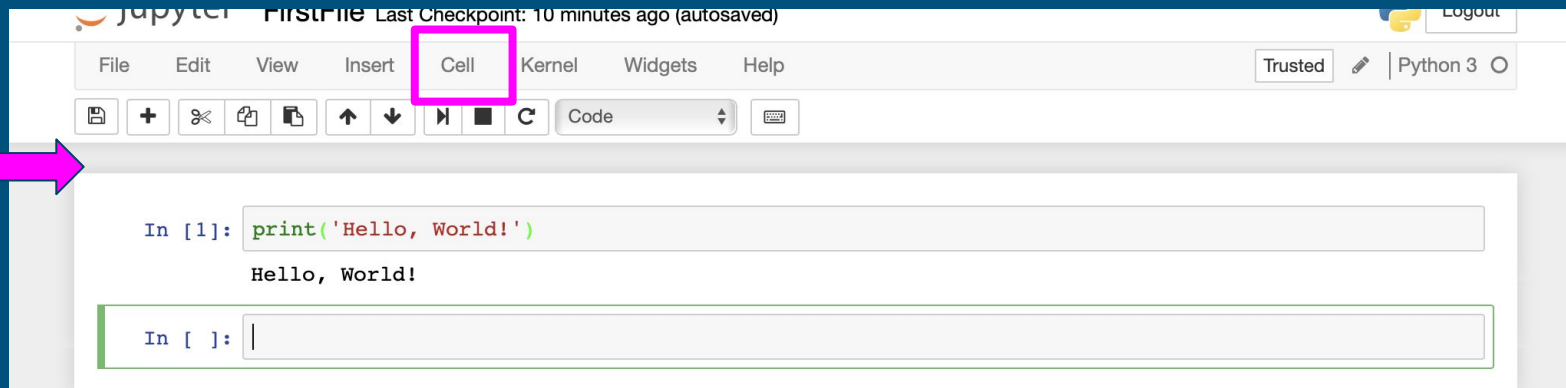
# Using command mode



The screenshot shows the Jupyter Notebook interface. At the top, the title bar reads "Jupyter FirstFile Last Checkpoint: 10 minutes ago (unsaved changes)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". To the right of the menu bar are buttons for "Trusted", a pencil icon, and "Python 3". Below the menu bar is a toolbar with icons for saving, adding, deleting, and other actions. The main area shows a code cell with the prompt "In [ ]:" and the code `print('Hello, World!')`. A pink arrow points to the prompt "In [ ]:". Below the code cell, the text "Pressing Enter/Return will just move to the next line.." is displayed.

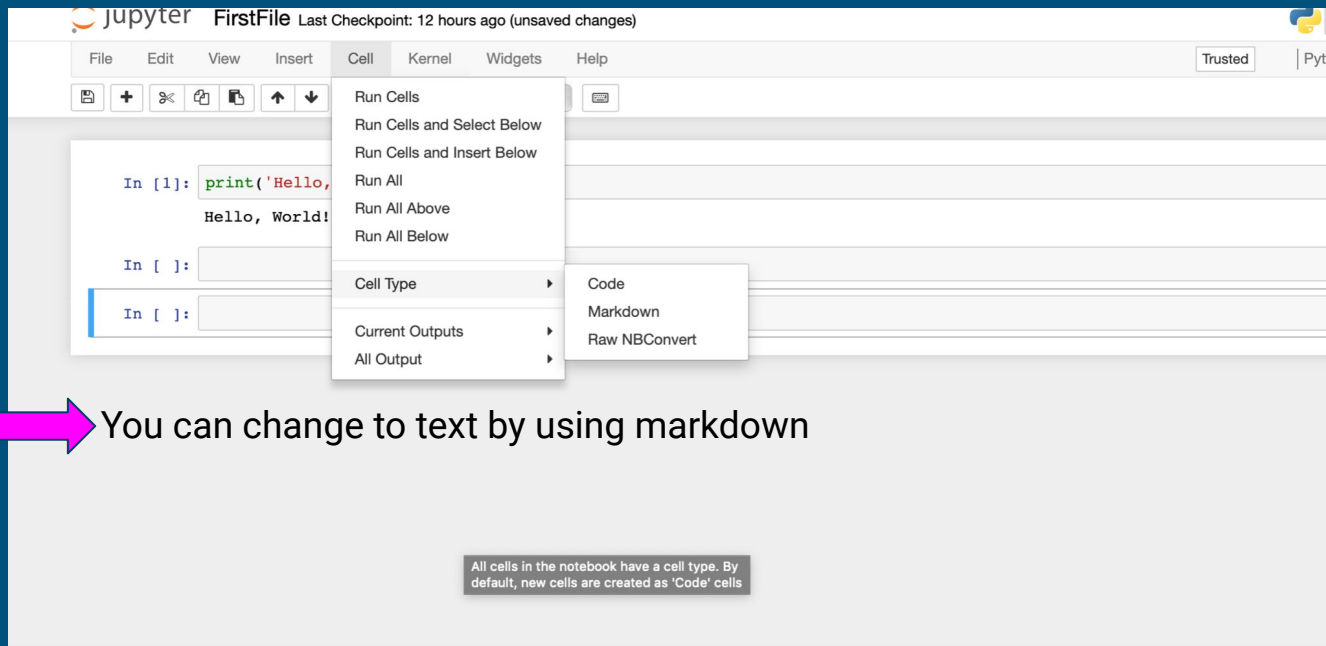
Pressing Enter/Return will just move to the next line..

# Using command mode



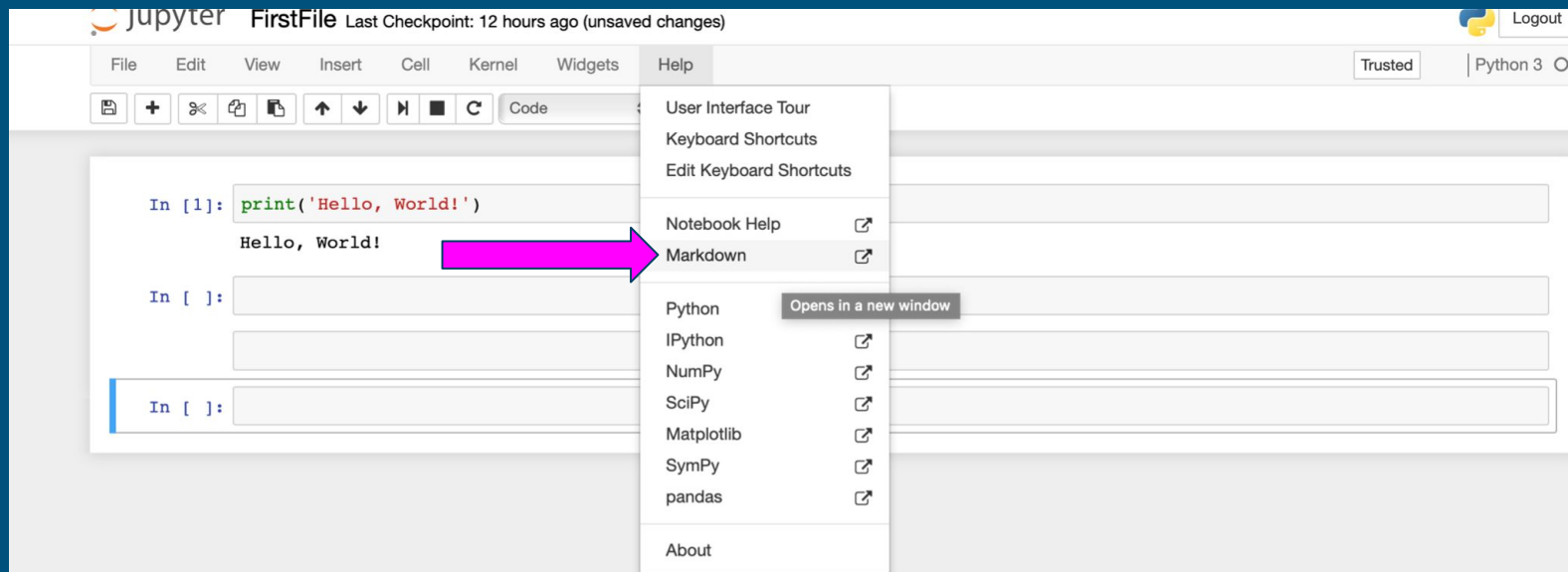
Pressing Ctrl/Command+Enter/Return will execute the commands inside the cell. And create a new cell below it. This can also be done from the “cell” menu.

# Writing text



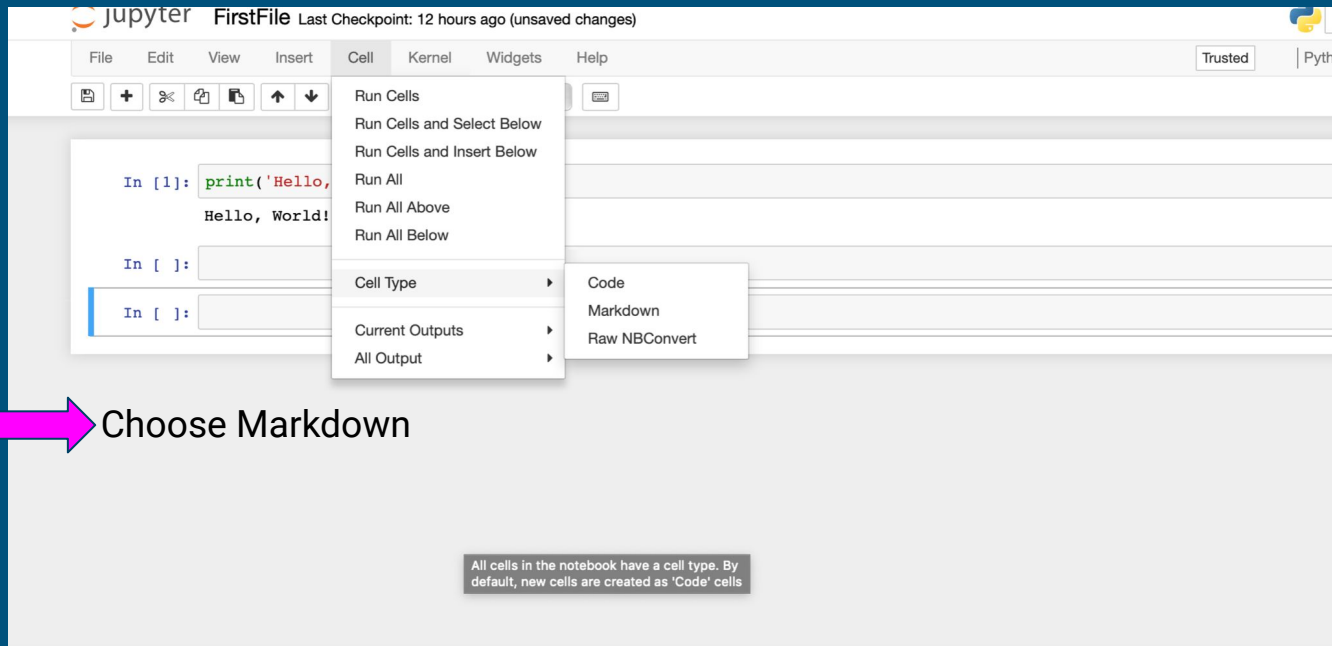


# Writing text

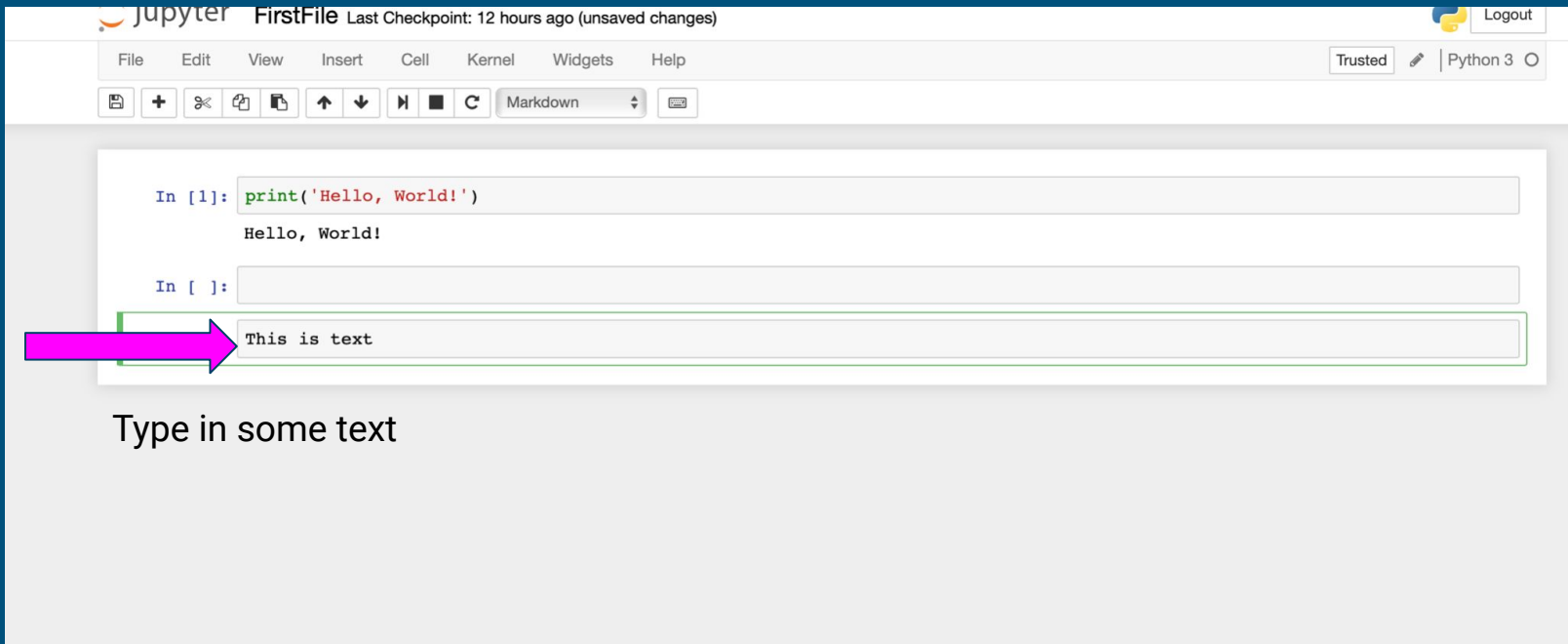


More information about Markdown is here. You can do fancy formatting.

# Writing text



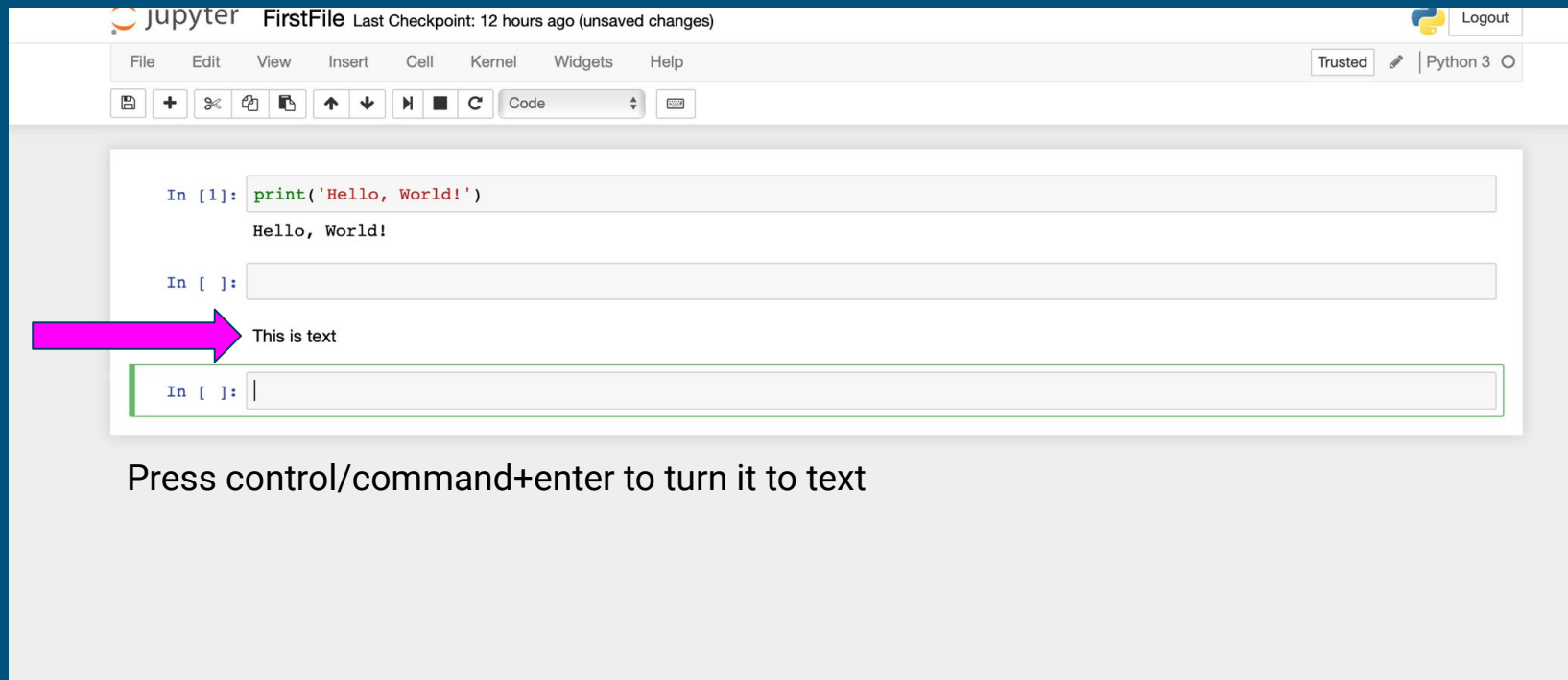
# Writing text



The screenshot shows a Jupyter Notebook interface. At the top, the title bar reads "Jupyter FirstFile Last Checkpoint: 12 hours ago (unsaved changes)". Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". On the right side of the menu bar, there is a "Trusted" status indicator, a pencil icon, and "Python 3" with a dropdown arrow. Below the menu bar is a toolbar with icons for saving, adding, deleting, and other actions, along with a "Markdown" dropdown menu. The main area of the notebook contains three cells. The first cell is a code cell with the text `In [1]: print('Hello, World!')` and the output `Hello, World!`. The second cell is a code cell with the text `In [ ]:`. The third cell is a text cell with the text `This is text`. A pink arrow points to the text cell.

Type in some text

# Writing text

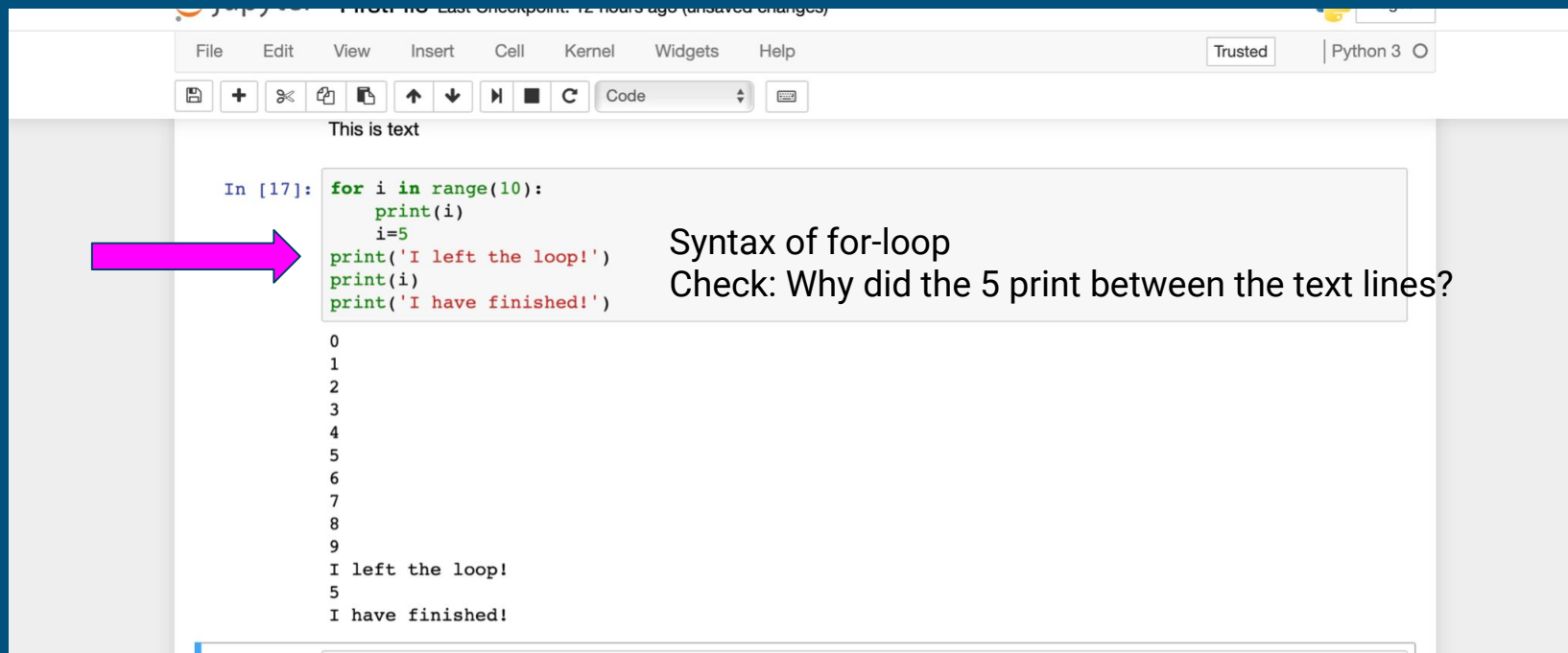


The screenshot shows a Jupyter Notebook interface. At the top, the header includes the Jupyter logo, the filename "FirstFile", and a status message "Last Checkpoint: 12 hours ago (unsaved changes)". On the right, there is a "Logout" button. Below the header is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for "Trusted", a pencil icon, and "Python 3". Below the menu bar is a toolbar with icons for saving, adding, deleting, and other actions, along with a dropdown menu currently set to "Code".

The main area of the notebook contains three input cells. The first cell is a code cell with the text `In [1]: print('Hello, World!')` and its output `Hello, World!` displayed below it. The second cell is an empty code cell with the prompt `In [ ]:`. The third cell is a text cell, indicated by a pink arrow pointing to it from the left. It contains the text "This is text" and has a green border. Below the text cell is another empty code cell with the prompt `In [ ]:`.

Press control/command+enter to turn it to text

# For-loops



The screenshot shows a Jupyter Notebook interface. At the top, there's a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu bar is a toolbar with icons for saving, adding, deleting, and running code. The main area displays a code cell with the following Python code:

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

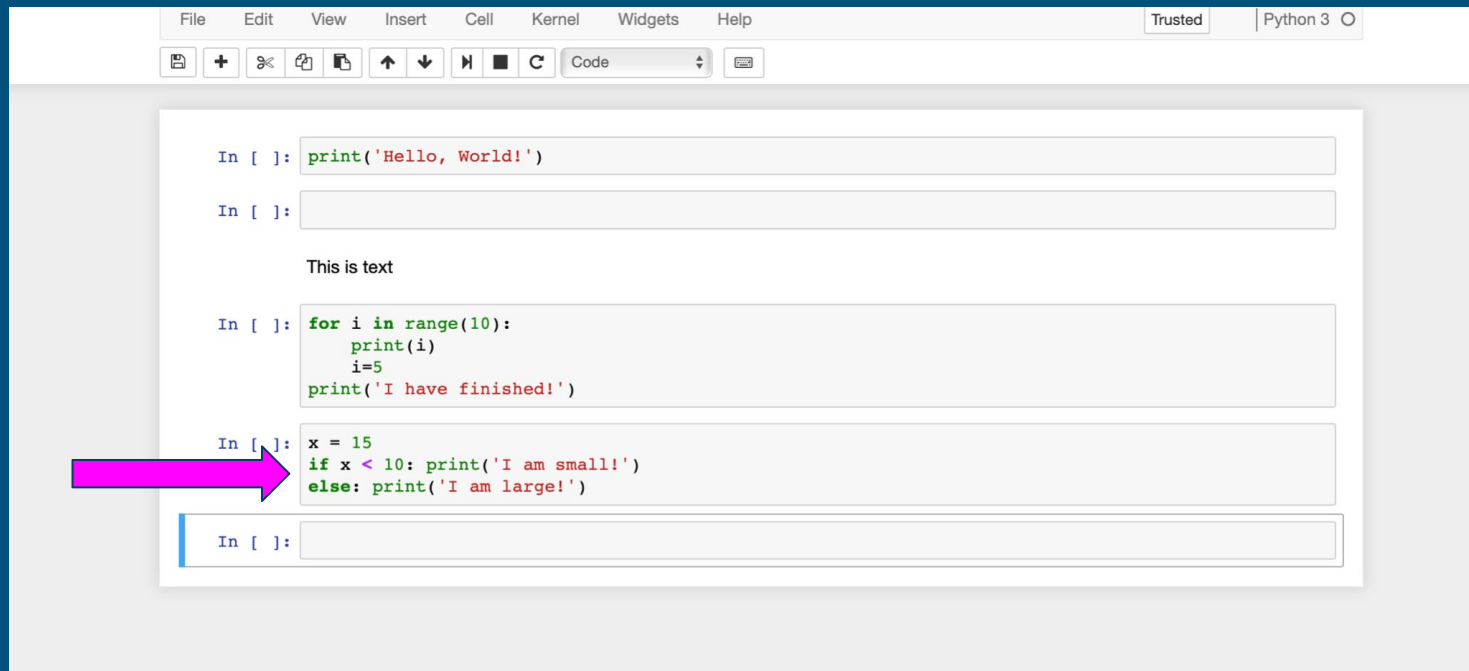
A pink arrow points to the first `print(i)` statement in the code. To the right of the code, there is a text box containing the text:

Syntax of for-loop  
Check: Why did the 5 print between the text lines?

Below the code cell, the output of the code is displayed:

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

# If statement syntax



The screenshot displays a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. The notebook contains several input cells:

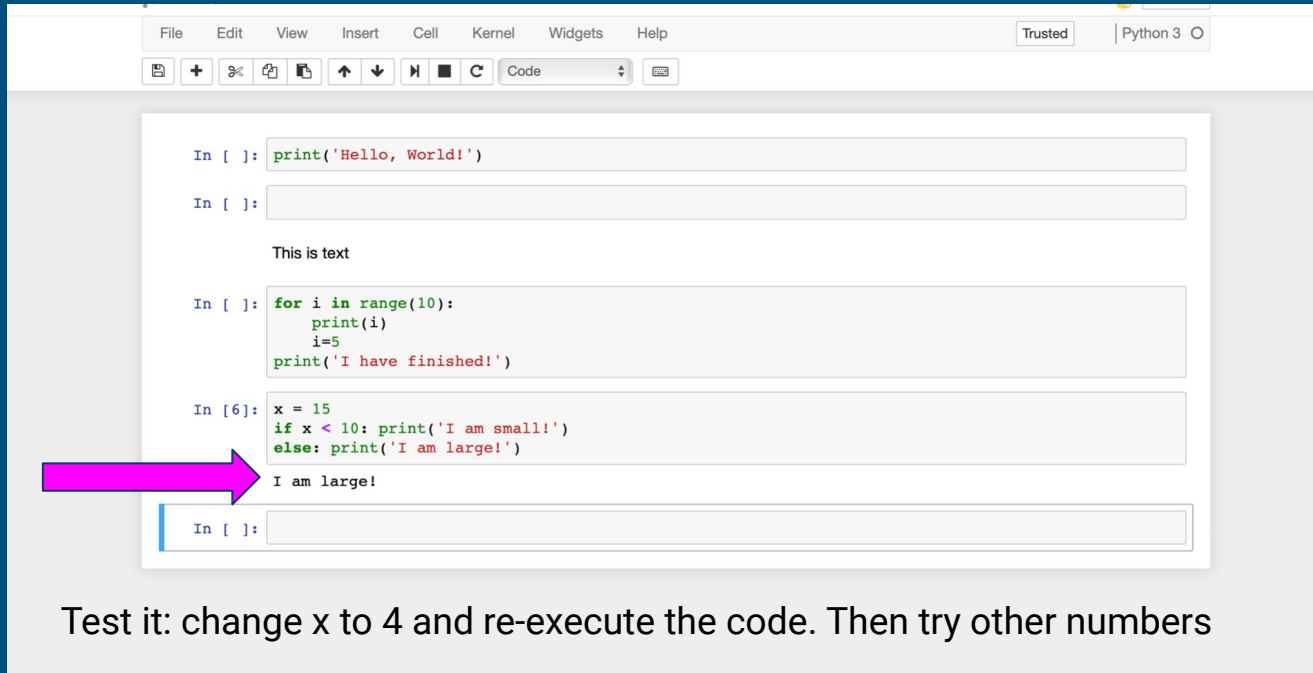
- Cell 1: `print('Hello, World!')`
- Cell 2: Empty
- Cell 3: Text output: "This is text"
- Cell 4: 

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

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

A pink arrow points to the fourth cell, highlighting the `if-else` statement syntax.

# If Statement output



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. The notebook contains several input cells. The first cell has the code `print('Hello, World!')`. The second cell is empty. The third cell has the text "This is text". The fourth cell has the code `for i in range(10):`, `print(i)`, `i=5`, and `print('I have finished!')`. The fifth cell has the code `x = 15`, `if x < 10: print('I am small!')`, and `else: print('I am large!')`. The output of the fifth cell is "I am large!". A pink arrow points to the output "I am large!". The sixth cell is empty.

```
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

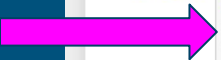
In [ ]:



# 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!
```

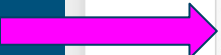
Set step size and final time

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!
```

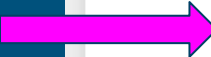
Initial values

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]: 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!
```

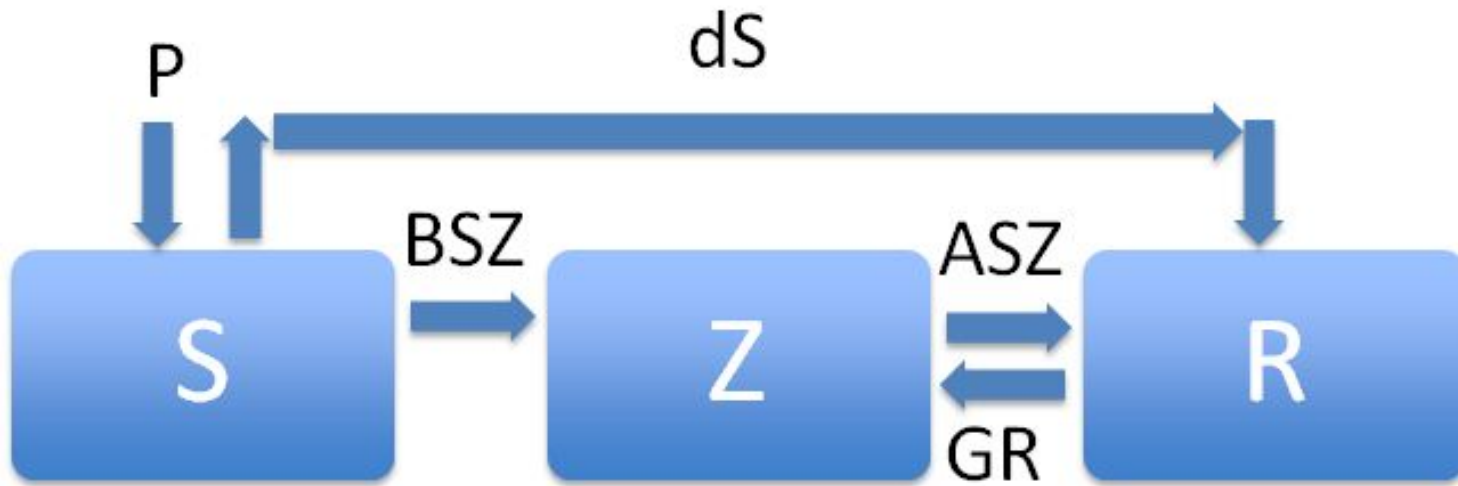
Prints output

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"

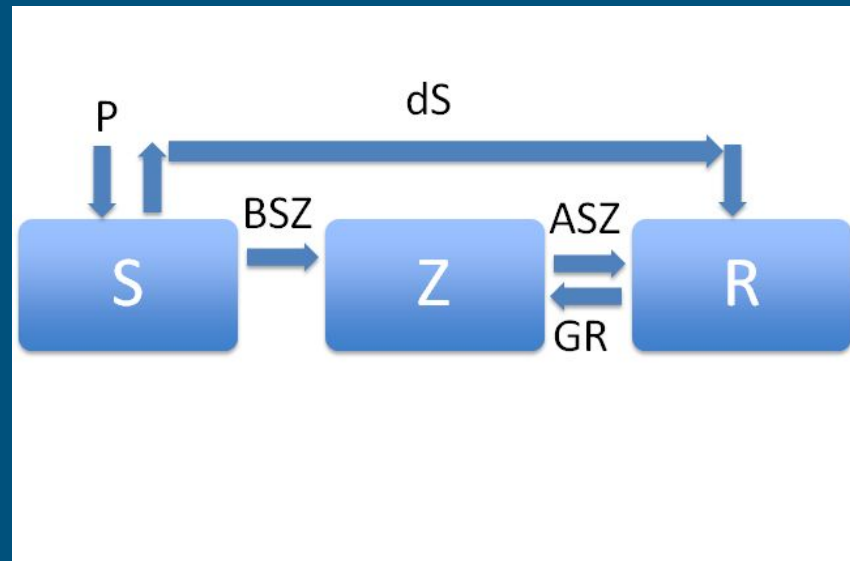
P: the population birth rate

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:

$R_0 = 0.01 * S_0$  # 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

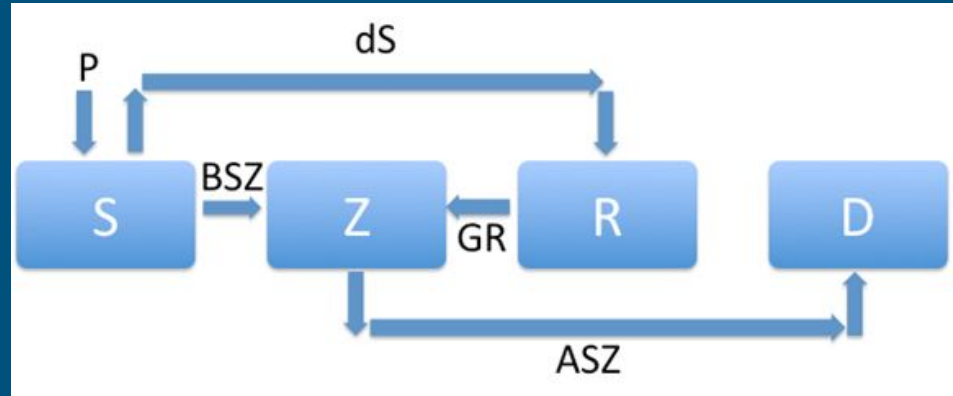
Save the zombie.py file as zombie.py. Set the parameters back to where we started ( $P=0$ ,  $R_0=0$ ). Let's change the dynamics so that when zombies are killed, they cannot be resurrected again. We will need to create a new compartment for this group, and let  $D$  be the number of completely removed (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  $D_0=0$ .
2. In the vector that holds the initial conditions,  $y_0$ , add  $D_0$ .
3. In `def f(y,t)`, add  $D_i=y[3]$  to hold the values of the totally removed people.
4. In `def f(y,t)`, add  $f_3 = A*Si*Zi$ , and change the equation for  $f_2$  to  $f_2 = f_2 = d*Si - G*Ri$ .
5. The vector that gets returned should now be  $[f_0, f_1, f_2, f_3]$
6. Set  $D = \text{soln[:, 3]}$ .
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

The screenshot shows the GitHub repository page for `mountaindust/Planktos`. The page layout includes a dark header with navigation links, a search bar, and sign-in/sign-up buttons. Below the header, the repository name is displayed with icons for notifications (2), stars (2), and forks (0). A secondary navigation bar contains links for Code, Issues (10), Pull requests, Actions, Projects, Wiki, Security, and Insights. The main content area shows the `master` branch selected, with a 'Go to file' button and a green 'Code' button. A commit history table lists recent changes, and the right sidebar provides an 'About' section with a description, README link, and license information, as well as a 'Releases' section.

https://github.com/mountaindust/Planktos

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search Sign in Sign up

mountaindust / Planktos

Notifications 2 Stars 2 Forks 0

<> Code Issues 10 Pull requests Actions Projects Wiki Security Insights

master 1 branch 2 tags

Go to file Code

mountaindust	Get rid of redundant OSX blitting boilerplate	d4bb90e yesterday	333 commits
doc/eq_of_motion	Cleanup directory structure	7 months ago	
examples	Get rid of redundant OSX blitting boilerplate	yesterday	
past_projects/brine_shrimp	Closes #29. Bug fixes, updates README.	5 months ago	
tests	Get rid of redundant OSX blitting boilerplate	yesterday	

About

ABM framework for dispersal modeling

Readme

GPL-3.0 License

Releases

# 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.

The screenshot shows the GitHub repository page for `nickabattista/IB2d`. The browser address bar displays `https://github.com/nickabattista/IB2d`. The repository header includes the GitHub logo, navigation links (Why GitHub?, Team, Enterprise, Explore, Marketplace, Pricing), a search bar, and buttons for Sign in and Sign up. Below the header, the repository name `nickabattista / IB2d` is shown, along with icons for Notifications, Star (72), and Fork (52). The main navigation bar includes links for Code, Issues (4), Pull requests, Actions, Projects, Wiki, Security, and Insights. The `Code` tab is selected. Below the navigation bar, there are buttons for `master` (with a dropdown), 5 branches, and 0 tags. A `Go to file` button and a green `Code` button with a download icon are also present. The repository content area shows a commit by `nickabattista` titled "updated IBM\_Driver for restart functionality with background co..." with commit hash `ddf56b6` and date "25 days ago". It also shows 896 commits. Below the commit list, there are folders for `data_analysis` and `IBM_Driver`. The `About` section on the right describes the repository as "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".