

Sept 28, 2022

Programming basics
with Bash, Python, and R

Suzy Strickler
srs57@cornell.edu

- <https://www.rstudio.com/products/rstudio/download/#download>
- sudo dpkg -i rstudio-2021.09.0-351-amd64.deb

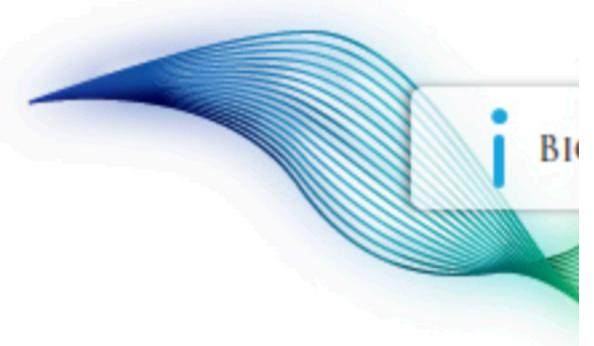
CORNELL UNIVERSITY
INSTITUTE OF BIOTECHNOLOGY

Search BioHPC

Home BioHPC Cloud User Guide Contact Us Login

institute of biotechnology >> brc >> bioinformatics >> internal >> workshops

Workshops

Please join [our mailing list](#) if you would like to be notified about new workshops.
The links to all recorded workshop sessions are [available here](#).

Virtual Workshops (Pre-recorded)

You can participate in a virtual workshop by reading workshop materials, watching video presentations and carrying out exercises on an assigned BioHPC Cloud server

- [Introduction to BioHPC Cloud](#)
- [Linux for Biologists](#)
- [Software installation and Conda](#)
- [Using Docker in BioHPC Cloud](#)
- [Parallel processing, scheduling and load balancing](#)
- [Sequence alignment, clustering, and phylogenetic analysis](#)
- [Genomic Sequence Data Analysis](#)
- [RNA-seq Data Analysis](#)
- [Genome feature and epigenomics data analysis for ChIP-Seq and ATAC-Seq experiments](#)

Current Live Virtual Workshops

These live workshops are held online via Zoom, and consist of presentations, Q&A, and hands-on exercises to be carried out on an assigned BioHPC Cloud server

- [Python and Jupyter Notebook \(October 18 2021\)](#)
- [R and Rstudio \(October 25 2021\)](#)
- [Docker and Singularity \(November 1 2021\)](#)

Office hour 12:30 - 1:30 pm today in G22 PS
See Canvas for link and schedule

Homework Solutions

Please log on to the william server using the credentials that will be supplied to you individually via email. The address for William is: william.sgn.cornell.edu

Please answer the below questions and store them in a text file called "class1_assignment.txt". You can either create the text file on your local computer and transfer it to william, or you can use one of the available text editors on william to create your answer file.

- 1) What is the hard disk size of William? du -lh
- 2) How much of the hard disk is currently free? df -lh
- 3) How much RAM does william have? free -m
- 4) Which users are currently running processes on William? ps -aux |cut -f1 -d " " |sort -u
- 5) How many CPUs is the process using the most memory using? top -> shift + m
- 6) How many people have an account on William? who
- 7) Which user is using the most hard disk space? du -sh *
- 8) What is the path to the file: canu-smartdenovo_pass5.fasta locate
- 9) Which version(s) of Python are installed on the system? python
- 10) Which version of R is on the system? R

What are scripts?

- A text document that contains instructions to be executed by a program
- Usually combine existing components
- Usually use an interpreter instead of compiler to translate to machine code
 - **Bash, Python, Perl, JavaScript, PHP, Ruby**

Why use scripts?

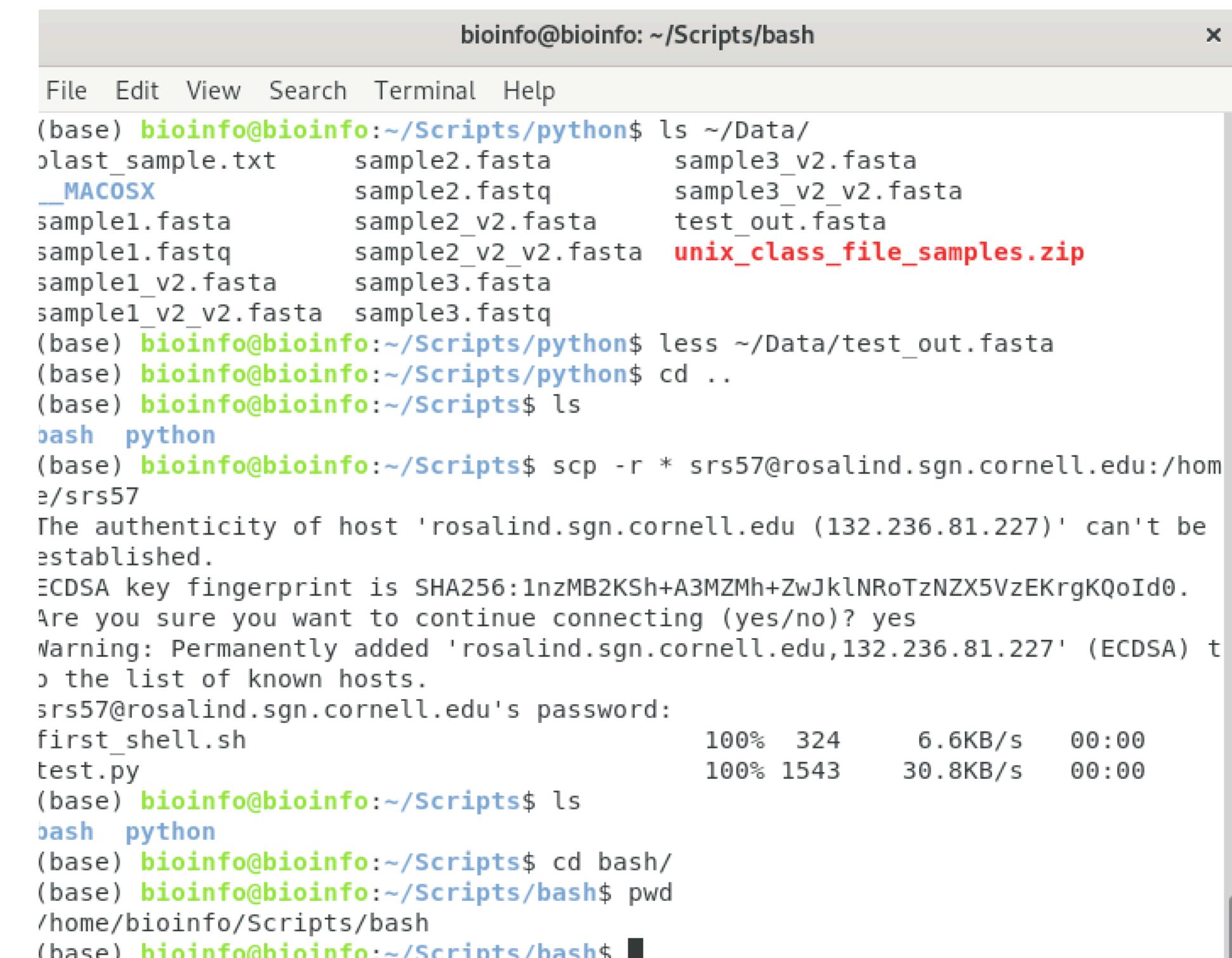
- Automation of step-by-step processes
- Reproducibility
- Share with others

Text Editors for Programming

- Don't write code in Word!
- Vi, Vim, Emacs
- **Atom**
- Sublime
- Visual Studio

Simple Scripting with Bash

- Bash is a Unix shell, both a command language and scripting language
- Configuration file, bash.rc, is written in bash
- A bash shell script contains a list of commands
 - Since we are using the bash shell in our VM, these are the same commands we went over last time

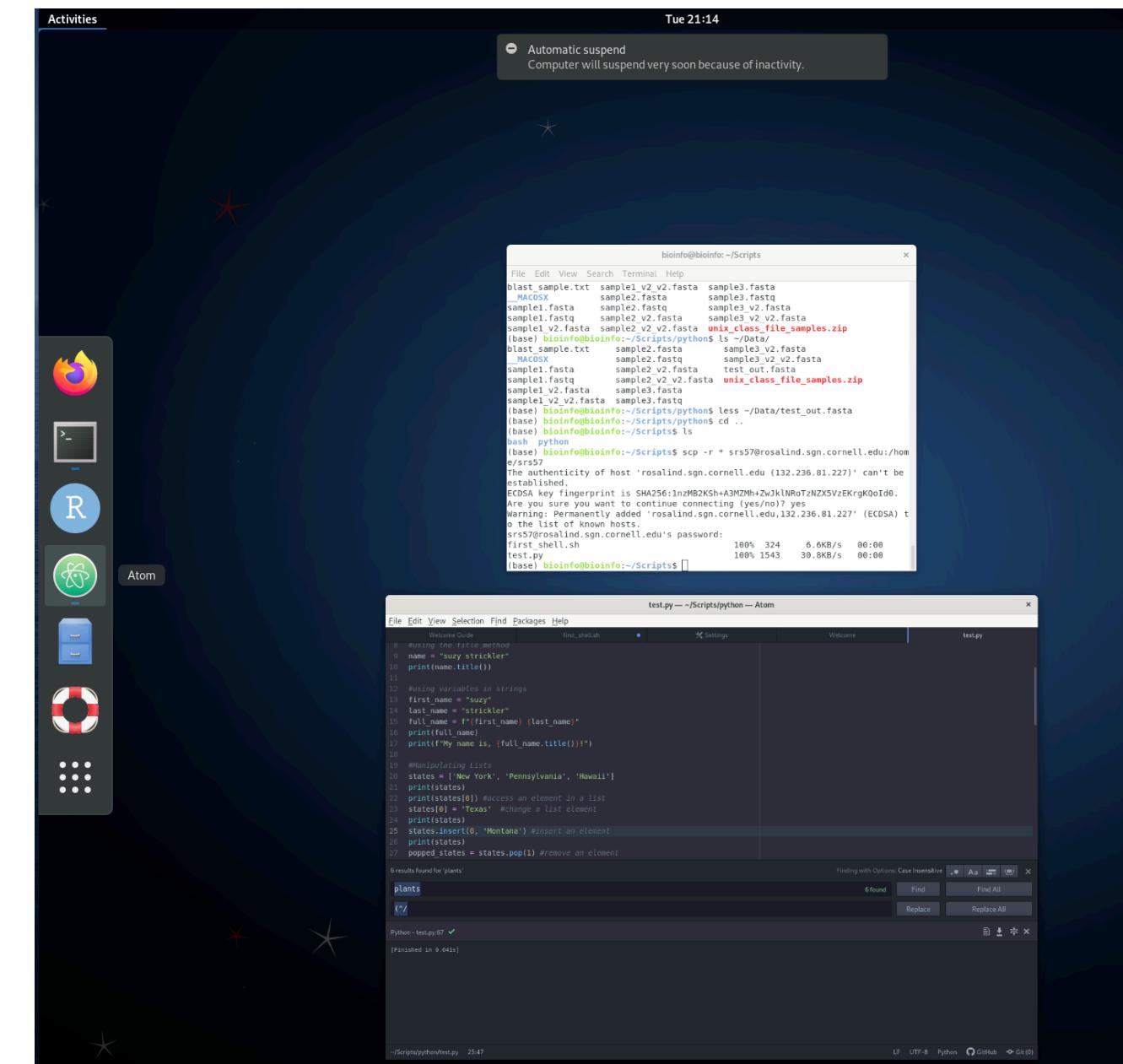


A screenshot of a terminal window titled "bioinfo@bioinfo: ~/Scripts/bash". The window shows a series of bash commands and their outputs. The commands include ls, less, cd, scp, python, and pwd. The output includes file lists, the contents of a FASTA file, a warning about host authenticity, a password prompt, and the current working directory.

```
bioinfo@bioinfo:~/Scripts/bash
File Edit View Search Terminal Help
(base) bioinfo@bioinfo:~/Scripts/python$ ls ~/Data/
blast_sample.txt      sample2.fasta      sample3_v2.fasta
__MACOSX              sample2.fastq      sample3_v2_v2.fasta
sample1.fasta          sample2_v2.fasta   test_out.fasta
sample1.fastq          sample2_v2_v2.fasta unix_class_file_samples.zip
sample1_v2.fasta       sample3.fasta
sample1_v2_v2.fasta    sample3.fastq
(base) bioinfo@bioinfo:~/Scripts/python$ less ~/Data/test_out.fasta
(base) bioinfo@bioinfo:~/Scripts/python$ cd ..
(base) bioinfo@bioinfo:~/Scripts$ ls
bash python
(base) bioinfo@bioinfo:~/Scripts$ scp -r * srs57@rosalind.sgn.cornell.edu:/home/srs57
The authenticity of host 'rosalind.sgn.cornell.edu (132.236.81.227)' can't be established.
ECDSA key fingerprint is SHA256:1nzMB2KSh+A3MZMh+ZwJKlNRoTzNZX5VzEKrgKQoId0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'rosalind.sgn.cornell.edu,132.236.81.227' (ECDSA) to the list of known hosts.
srs57@rosalind.sgn.cornell.edu's password:
first_shell.sh          100% 324      6.6KB/s  00:00
test.py                 100% 1543     30.8KB/s 00:00
(base) bioinfo@bioinfo:~/Scripts$ ls
bash python
(base) bioinfo@bioinfo:~/Scripts$ cd bash/
(base) bioinfo@bioinfo:~/Scripts/bash$ pwd
/home/bioinfo/Scripts/bash
(base) bioinfo@bioinfo:~/Scripts/bash$
```

Let's create a bash shell script

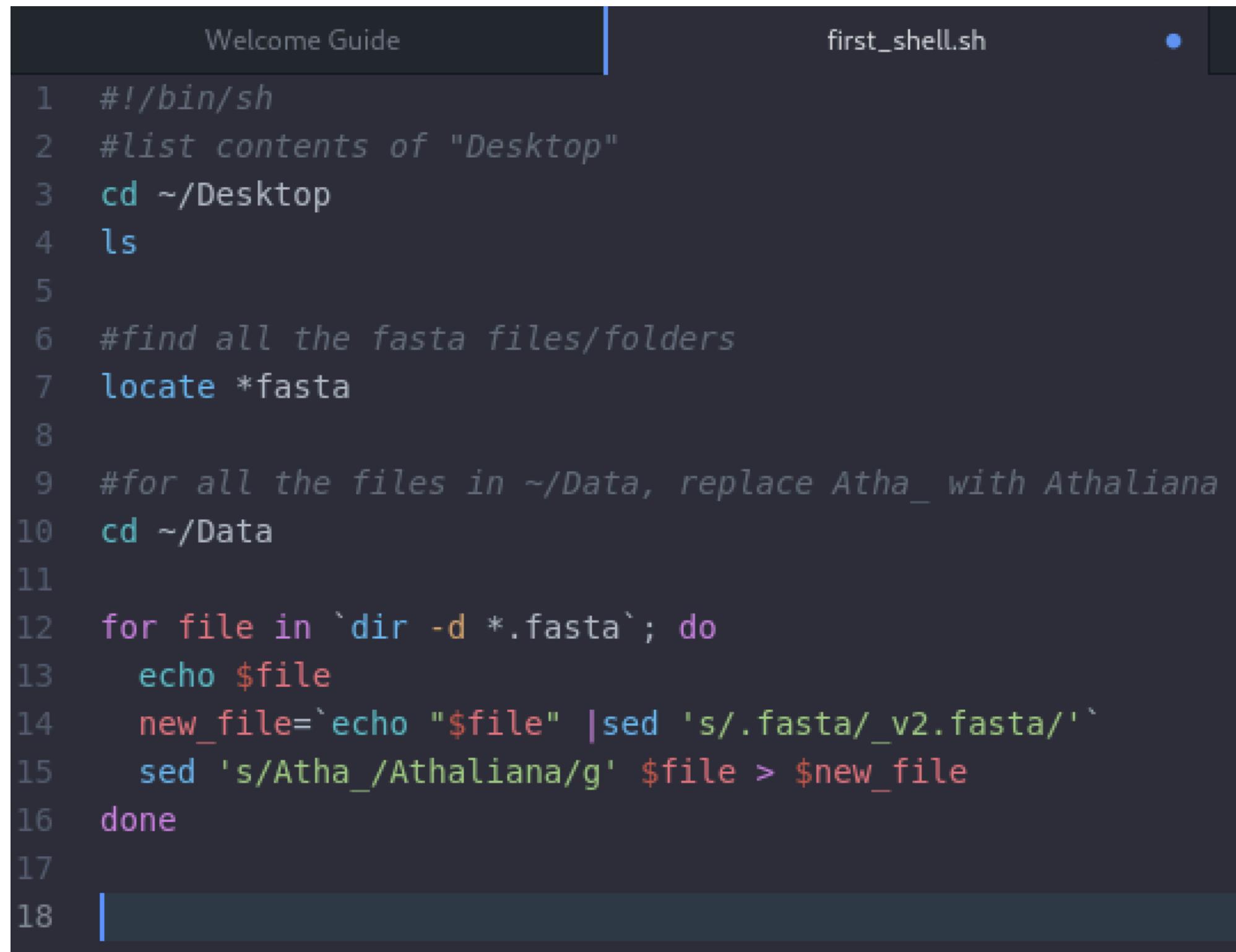
- Open Atom on your VM
- File -> Add Project Folder
 - Select /home/bioinfo/Scripts
 - New folder: “bash”
 - Select “ok”
- File -> New file



Let's create a bash shell script

https://github.com/bcbc-group/PLSCI7202/blob/master/first_shell.sh

We'll call it `first_shell.sh`



```
1 #!/bin/sh
2 #list contents of "Desktop"
3 cd ~/Desktop
4 ls
5
6 #find all the fasta files/folders
7 locate *fasta
8
9 #for all the files in ~/Data, replace Atha_ with Athaliana
10 cd ~/Data
11
12 for file in `dir -d *.fasta`; do
13   echo $file
14   new_file=`echo "$file" |sed 's/.fasta/_v2.fasta/'`
15   sed 's/Atha_/Athaliana/g' $file > $new_file
16 done
17
18 |
```

We can run the script from within Atom
Shift + Ctrl + b

We can also run it from the command line
`bash ~/Scripts/bash/first_shell.sh`

Why use bash?

- It is useful for chaining other programs together.
- This allows one to automate tasks and build pipelines.

```
1 #!/bin/sh
2
3 #move to annotation directory
4 cd /opt/annotation
5
6 #get some files
7 cp /scratch/Botany2020NMGWorkshop/annotation/2transfer/list.txt .
8 cp /scratch/Botany2020NMGWorkshop/annotation/2transfer/uniprot_sprot_plants.fasta .
9
10 #run Portcullis: https://bioconda.github.io/recipes/portcullis/README.html
11 #docker pull maplesond/portcullis:stable #already done with Adrian
12 docker run --rm -v /scratch/annotation_output:/data maplesond/portcullis:stable portcullis full -t 7 -v /data/contig_15.fasta /data/SRR5046448_contig15.sort.bam
13
14 #run Mikado pipeline: https://bioconda.github.io/recipes/mikado/README.html
15 #docker pull cyverseuk/mikado #already done with Adrian
16 docker run --rm -v /scratch/annotation_output:/data cyverseuk/mikado mikado configure --list list.txt --reference data/contig_15.fasta --mode permissive --scoring-method jaccard
17 docker run --rm -v /scratch/annotation_output:/data cyverseuk/mikado mikado prepare --json-conf /data/configuration.yaml
18
19 makeblastdb -in uniprot_sprot_plants.fasta -dbtype prot
20 blastx -max_target_seqs 5 -num_threads 7 -query mikado_prepared.fasta -outfmt 5 -db uniprot_sprot_plants.fasta -evalue 0.000001 2> blast.log | sed '/^$/d' | gzip
21
22 screen -L /opt/TransDecoder-TransDecoder-v5.5.0/TransDecoder.LongOrfs -t mikado_prepared.fasta
23
24 screen -L hmmscan --cpu 7 --domtblout pfam.domtblout Pfam-A.hmm mikado_prepared.fasta.transdecoder_dir/longest_orfs.pep
25
26 blastp -query mikado_prepared.fasta.transdecoder_dir/longest_orfs.pep -db uniprot_sprot_plants.fasta -max_target_seqs 1 -outfmt 6 -evalue 1e-5 -num_threads 7 >
27
28 /opt/TransDecoder-TransDecoder-v5.5.0/TransDecoder.Predict -t mikado_prepared.fasta --retain_blastp_hits blastp.outfmt6 --cpu 7 --retain_pfam_hits pfam.domtblout
29
30 docker run --rm -v /scratch/annotation_output:/data cyverseuk/mikado mikado serialise --json-conf /data/configuration.yaml --xml /data/mikado.blast.xml.gz --orf
```

The Python Language

- First released in 1991
- Efficient and elegant - fewer lines of code
- Object-oriented
- Runs on many operating systems
- Monty Python's Flying Circus
- Python 2 (discontinued 2020) vs Python 3 - not completely backward compatible

Setting up Python

- Linux - usually already installed but may need updating
- OSX
- Windows
- What version is on our VM?

Code Libraries

- Modularizing code
 - Breaking apart problem into individual units
 - Easier to maintain
 - Reusable
- Modules, Packages - grouping of modules, Library - a collection of packages
- Libraries we will use:
 - BioPython, NumPy, Pandas in later classes
- Package Repos
 - Python Package Index (PyPi) and Anaconda

Comments

- Allows you to add human readable text to your code
- In Python comment lines start with: #
- Allows you and others to understand your code

Jupyter Notebooks

- Starting the notebook server

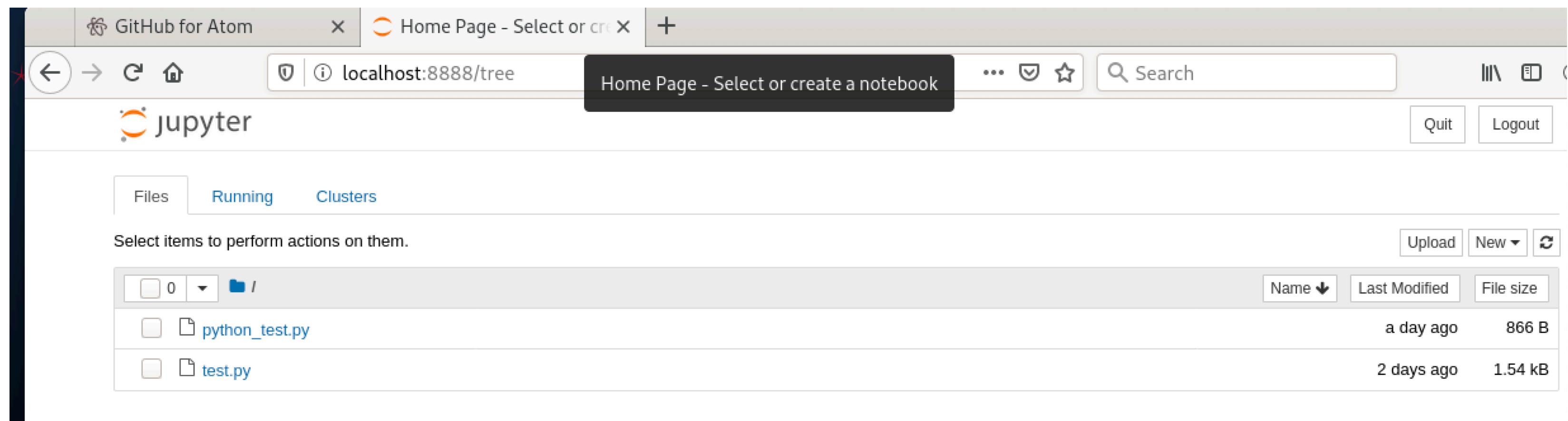
mkdir ~/Scripts/python

cd ~/Scripts/python/

jupyter notebook

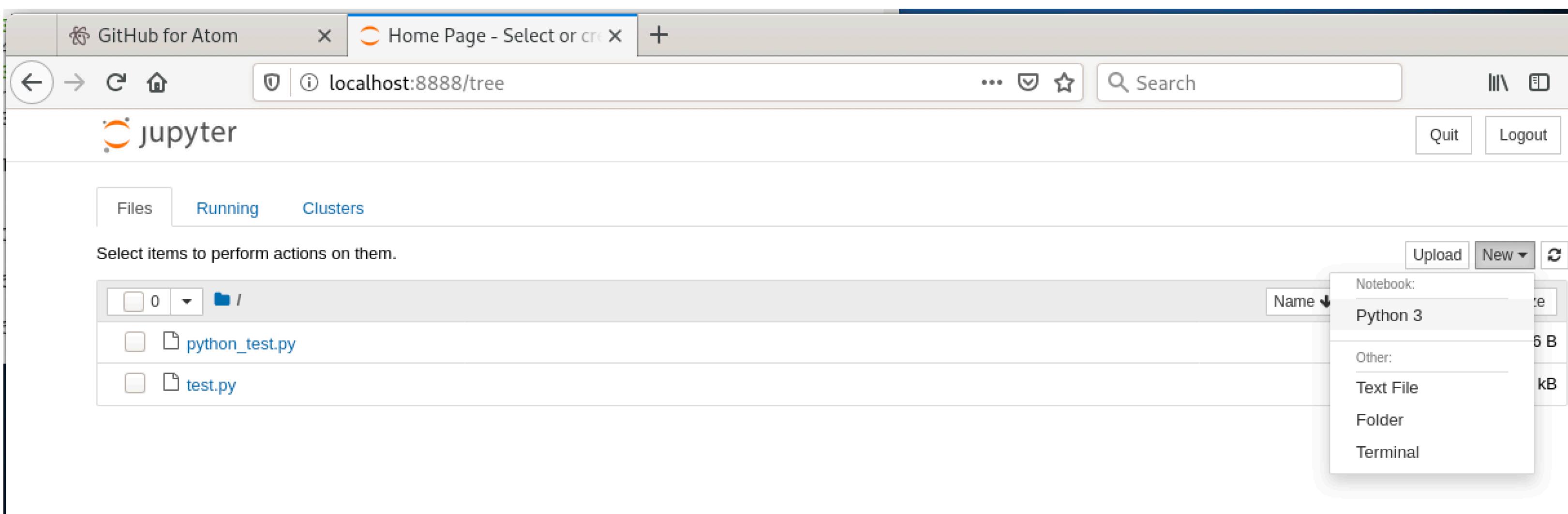
```
bioinfo@bioinfo:~/Scripts/python
File Edit View Search Terminal Help
(base) bioinfo@bioinfo:~$ cd Scripts/python/
(base) bioinfo@bioinfo:~/Scripts/python$ jupyter
usage: jupyter [-h] [--version] [--config-dir] [--data-dir] [--runtime-dir]
                [--paths] [--json]
                [subcommand]
jupyter: error: one of the arguments --version subcommand --config-dir --data-di
r --runtime-dir --paths is required
(base) bioinfo@bioinfo:~/Scripts/python$ jupyter notebook
[I 20:05:44.744 NotebookApp] Writing notebook server cookie secret to /home/bioi
nfo/.local/share/jupyter/runtime/notebook_cookie_secret
[T 20:05:45.292 NotebookApp] JupyterLab extension loaded from /home/bioinfo/min
```

- Notebook Dashboard

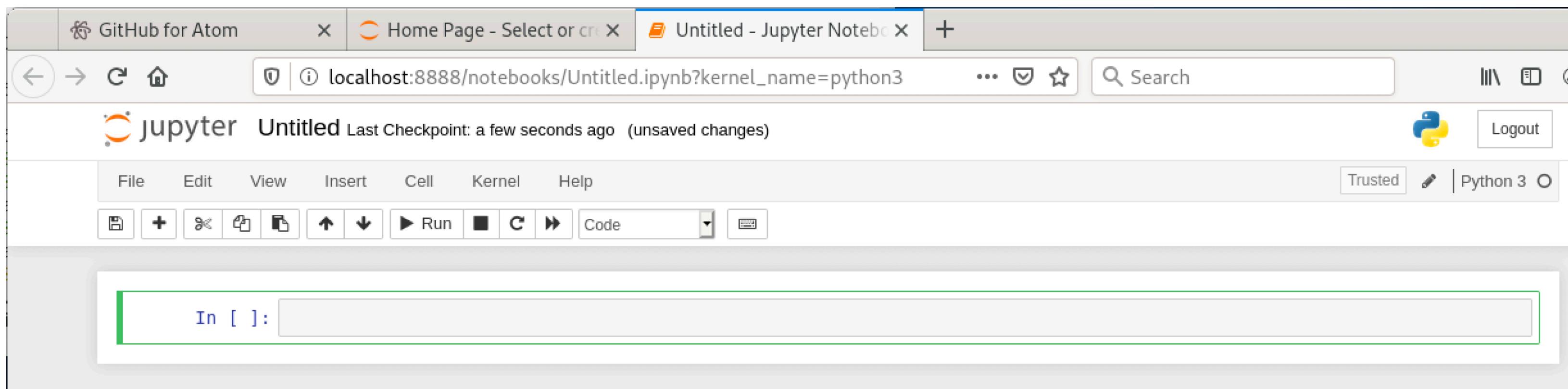


Jupyter Notebooks

- Create a new notebook

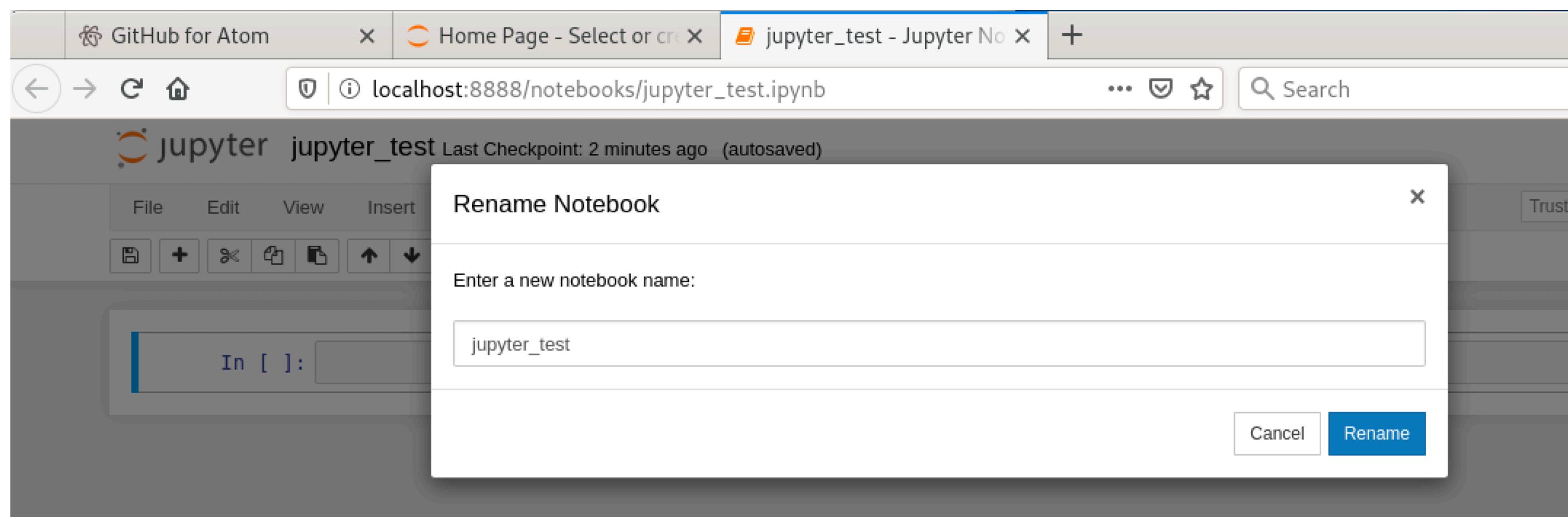
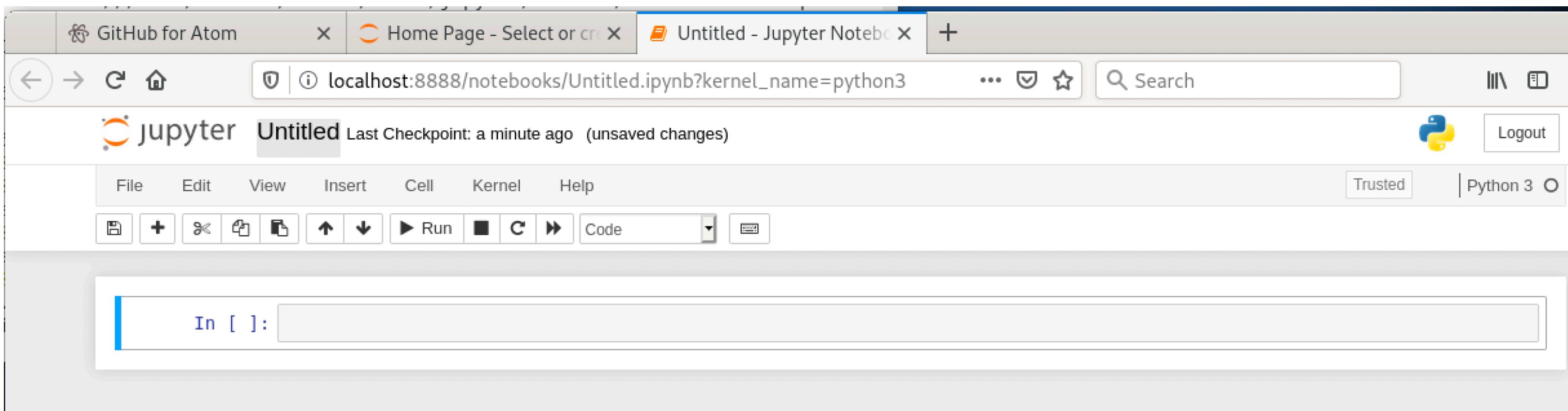


Jupyter Notebooks



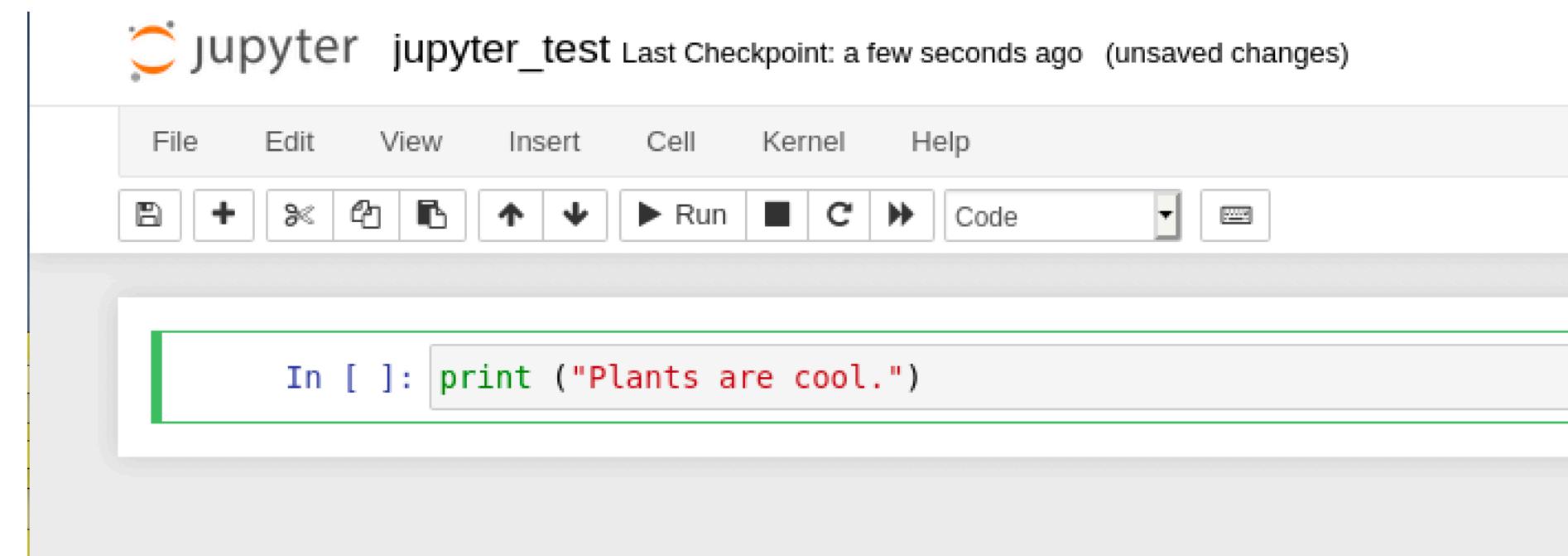
Jupyter Notebooks

- Name the notebook



Jupyter Notebooks

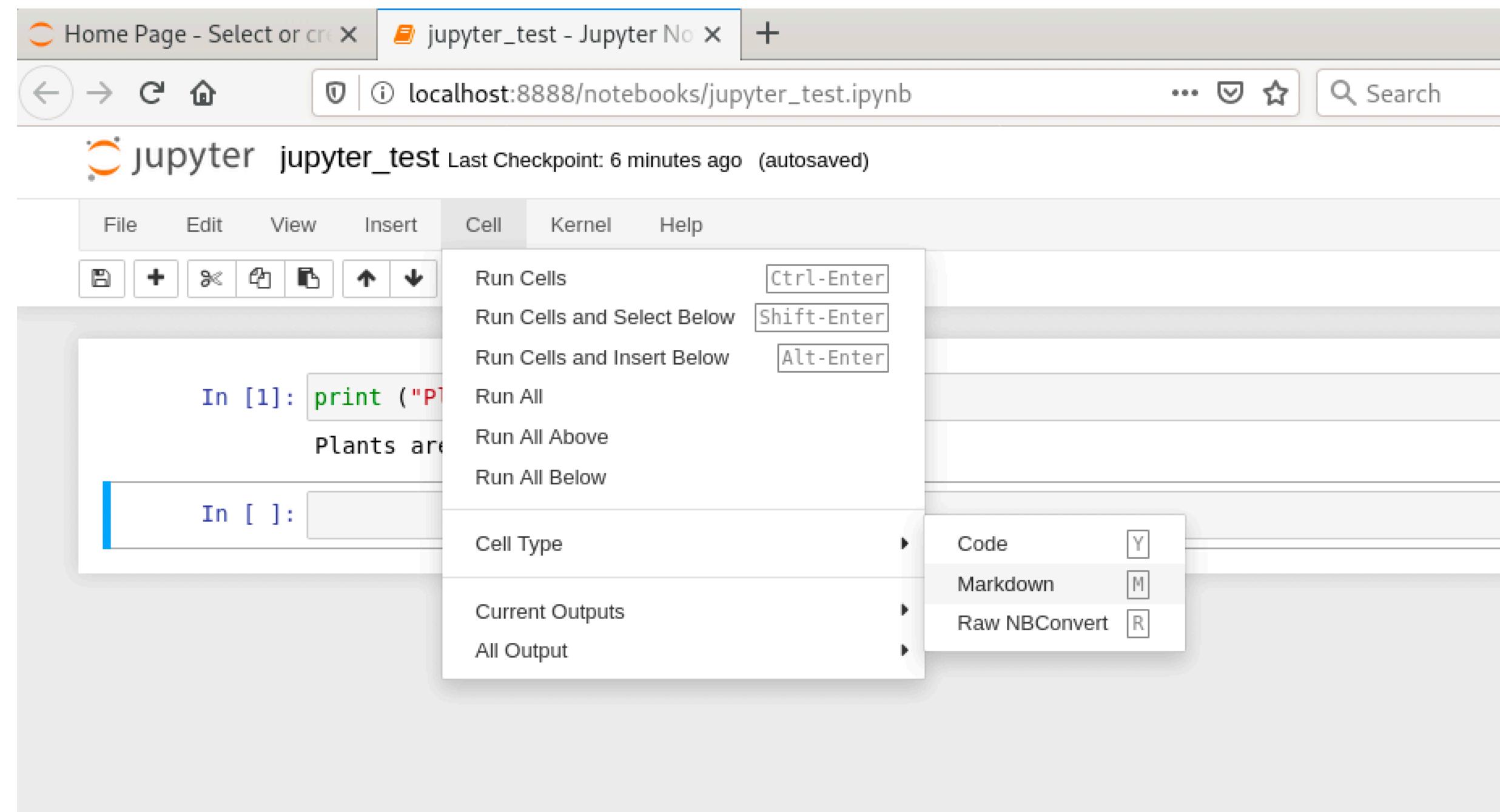
- Writing code and executing cells



Shift + enter

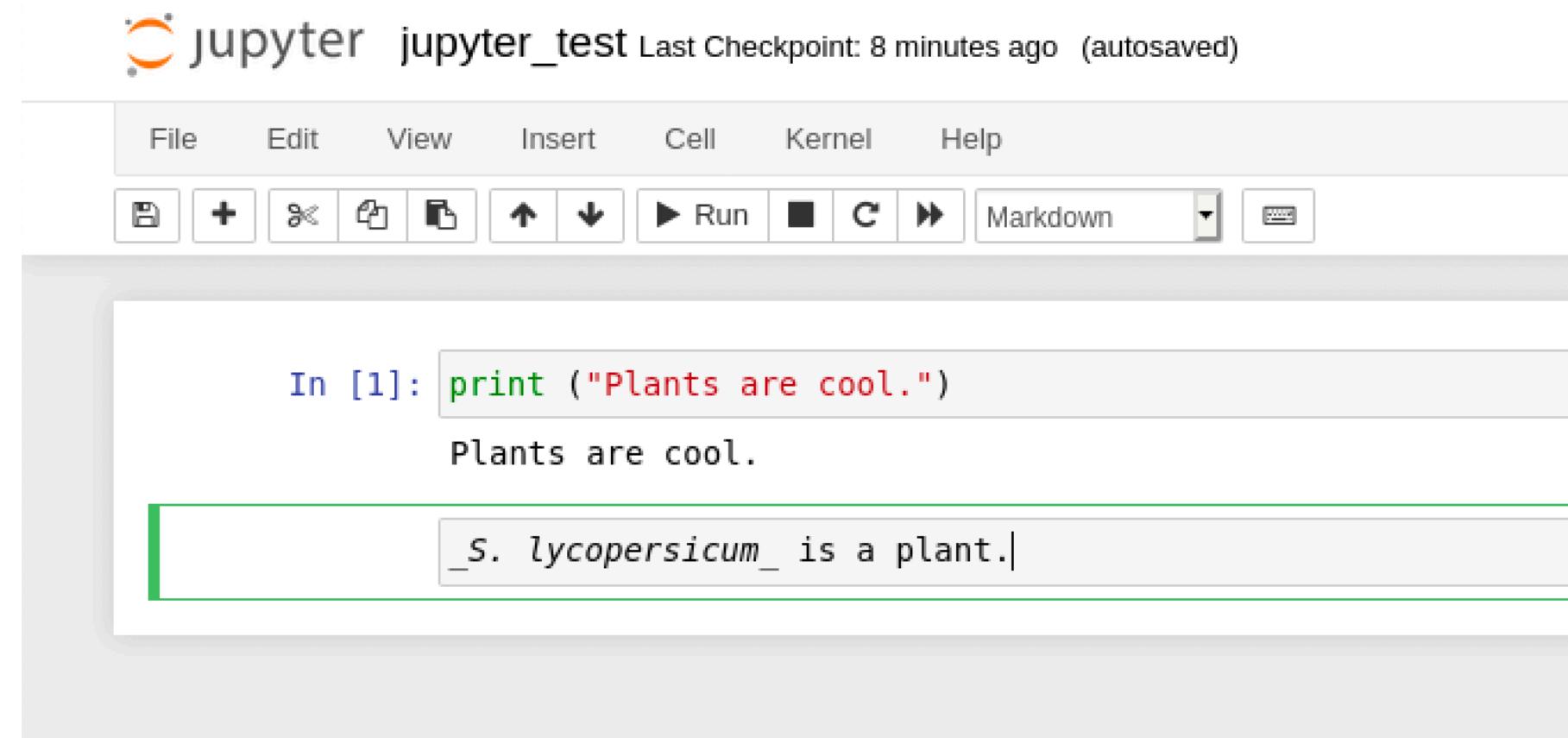
Jupyter Notebooks

- There are four cell types: Code, Markdown, Raw NBConvert, and Heading. Code and Markdown are the main types used.



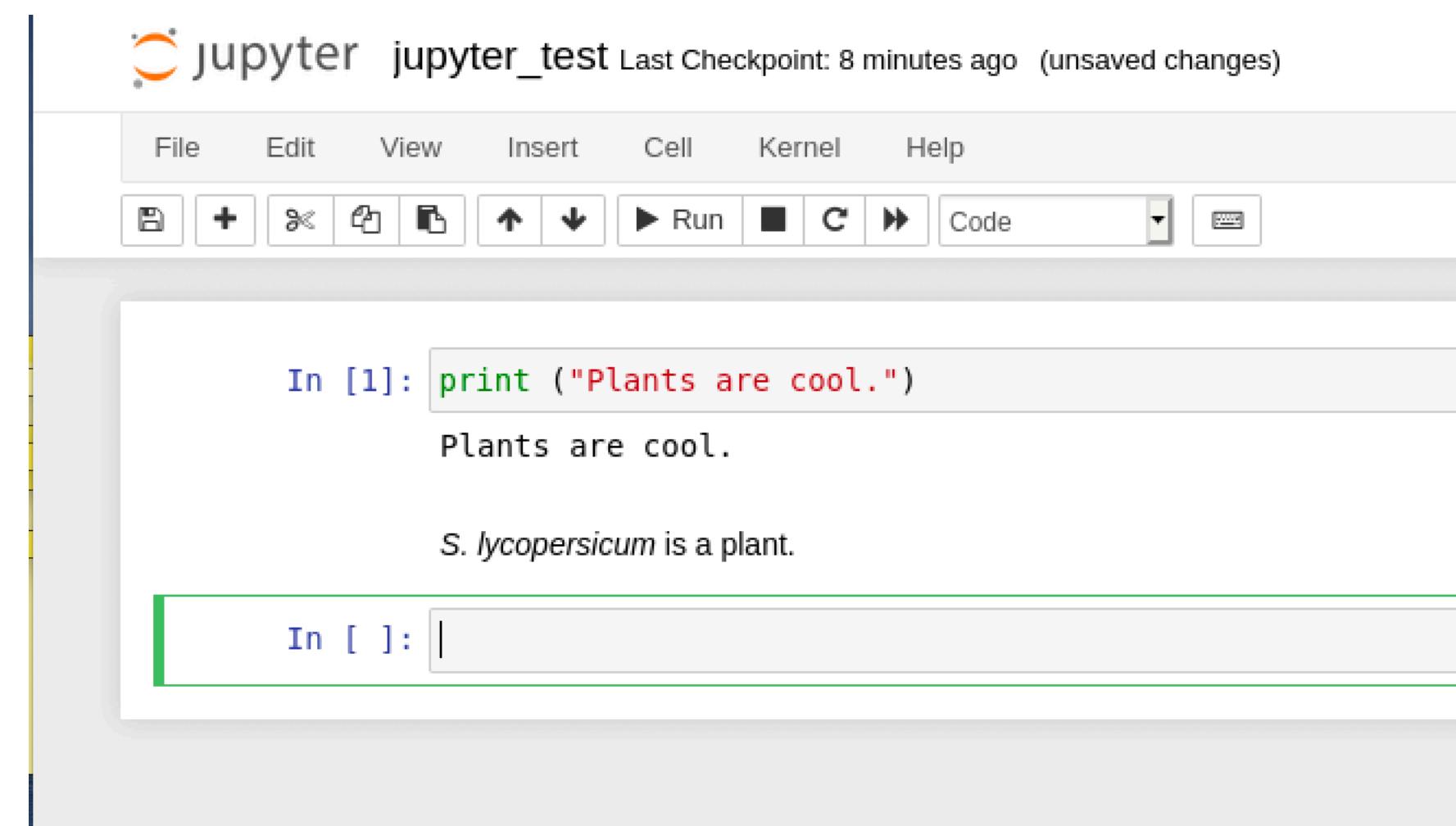
Jupyter Notebooks

- Markdown
- Esc + m



A screenshot of a Jupyter Notebook interface. The title bar says "jupyter jupyter_test Last Checkpoint: 8 minutes ago (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and New, followed by Run, Cell, and Help. A dropdown menu shows "Markdown". The main area shows a code cell with the command `In [1]: print ("Plants are cool.")`. The output below it is `Plants are cool.`. Below that is a text cell containing the text `_S. lycopersicum_ is a plant.`, which is highlighted with a green border, indicating it is selected or being edited.

Shift + enter



A screenshot of a Jupyter Notebook interface, identical to the one above but with a vertical blue selection bar on the left side of the code cell. The title bar says "jupyter jupyter_test Last Checkpoint: 8 minutes ago (unsaved changes)". The menu bar and toolbar are the same. The main area shows the same code cell and output as the first screenshot. The text cell below it now contains the text `S. lycopersicum is a plant.`. The input cell below is empty and labeled "In []:".

Jupyter Notebooks

- Jupyter supports Markdown which is based on html

A screenshot of a Jupyter Notebook interface. The top bar shows the URL `localhost:8888/notebooks/jupyter_test.ipynb`. The toolbar includes standard browser controls and Jupyter-specific buttons for file operations, cell creation, and kernel management. The main area contains two code cells. The first cell, labeled "In [1]", contains the Python code `print ("Plants are cool.")`, which outputs "Plants are cool." The second cell contains the Markdown code `# Header 1
Header 2
Header 3`, which displays the corresponding hierarchical headers.

Shift + enter

A screenshot of a Jupyter Notebook interface. The top bar shows the URL `localhost:8888/notebooks/jupyter_test.ipynb`. The toolbar is identical to the first screenshot. The main area shows two code cells. Both cells contain the same code as the first screenshot: "In [1]: print ("Plants are cool.")" followed by the output "Plants are cool.". Below these, three header cells are shown: "Header 1", "Header 2", and "Header 3". A green border highlights the first cell, indicating it is currently active or selected. The text "In []:" is visible at the bottom of the active cell.

Jupyter Notebooks

- Markdown

The screenshot shows a Jupyter Notebook interface with the following elements:

- Header:** jupyter jupyter_test Last Checkpoint: 14 minutes ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Help, and various icons for file operations.
- Cell 1 (Markdown):** In [1]: `print ("Plants are cool.")`
Output: Plants are cool.
- Text Cells:** S. lycopersicum is a plant.
- Section Headers:** Header 1, Header 2, Header 3.
- List Cell:** A code cell containing a list:
 - * plant1
 - * plant2
 - * plant3
 - * flower
- Cell 2 (Text):** In []: (empty)
- Cell 3 (Text):** In [1]: `print ("Plants are cool.")`
Output: Plants are cool.
S. lycopersicum is a plant.
- Cell 4 (List):** In []:
 - plant1
 - plant2
 - plant3
 - flower

Shift + enter

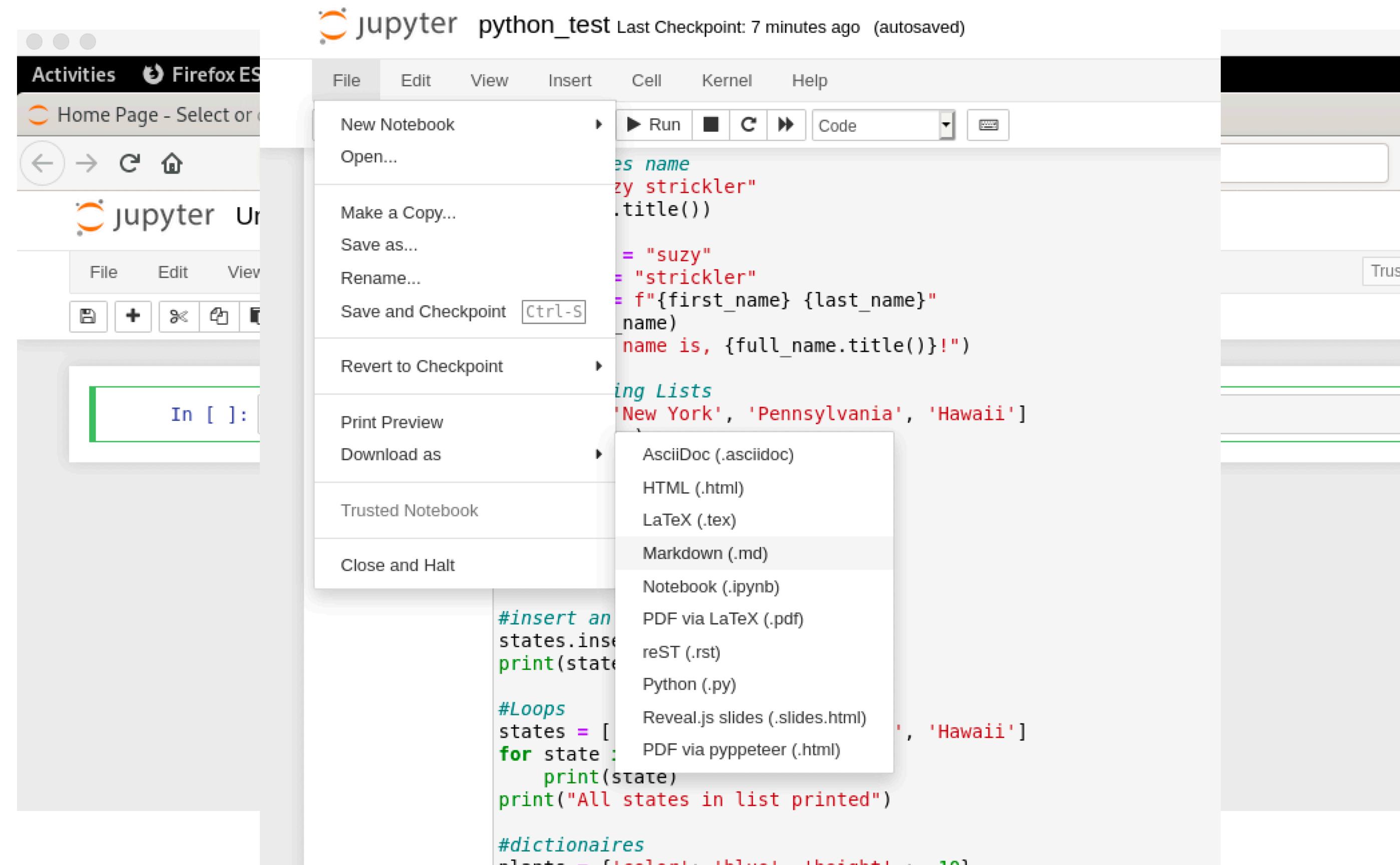
Jupyter Notebooks

- nbconvert can convert notebooks to pdf, HTML, LaTeX, and other formats

```
Shell
$ jupyter nbconvert py_examples.ipynb --to pdf
```

- Can also use menu to download as various file types

Jupyter Notebooks



Jupyter Notebooks

- Extensions add functionality to a notebook

Shell

```
$ jupyter nbextension install EXTENSION_NAME
```

This only installs the extension but does not make it active. You will need to enable an extension after installing it by running the following:

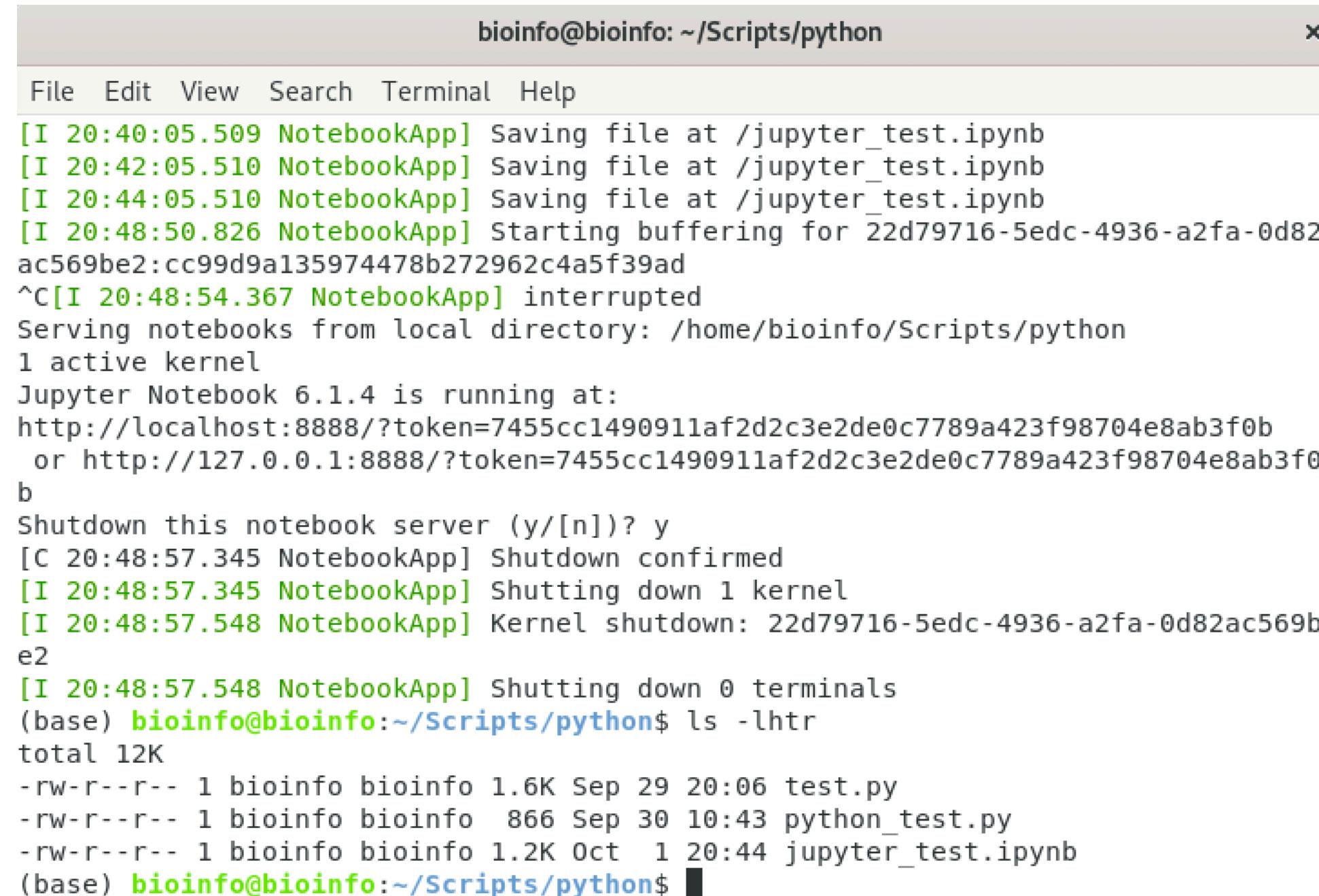
Shell

```
$ jupyter nbextension enable EXTENSION_NAME
```

- Examples: Table of Contents, Variable Inspector, ExecuteTime
 - <https://towardsdatascience.com/jupyter-notebook-extensions-517fa69d2231>

Jupyter Notebooks

- Notebook is stored with extension .ipynb (Json format)



A screenshot of a terminal window titled "bioinfo@bioinfo: ~/Scripts/python". The window shows the following text:

```
[I 20:40:05.509 NotebookApp] Saving file at /jupyter_test.ipynb
[I 20:42:05.510 NotebookApp] Saving file at /jupyter_test.ipynb
[I 20:44:05.510 NotebookApp] Saving file at /jupyter_test.ipynb
[I 20:48:50.826 NotebookApp] Starting buffering for 22d79716-5edc-4936-a2fa-0d82
ac569be2:cc99d9a135974478b272962c4a5f39ad
^C[I 20:48:54.367 NotebookApp] interrupted
Serving notebooks from local directory: /home/bioinfo/Scripts/python
1 active kernel
Jupyter Notebook 6.1.4 is running at:
http://localhost:8888/?token=7455cc1490911af2d2c3e2de0c7789a423f98704e8ab3f0b
or http://127.0.0.1:8888/?token=7455cc1490911af2d2c3e2de0c7789a423f98704e8ab3f0
b
Shutdown this notebook server (y/[n])? y
[C 20:48:57.345 NotebookApp] Shutdown confirmed
[I 20:48:57.345 NotebookApp] Shutting down 1 kernel
[I 20:48:57.548 NotebookApp] Kernel shutdown: 22d79716-5edc-4936-a2fa-0d82ac569b
e2
[I 20:48:57.548 NotebookApp] Shutting down 0 terminals
(base) bioinfo@bioinfo:~/Scripts/python$ ls -lhtr
total 12K
-rw-r--r-- 1 bioinfo bioinfo 1.6K Sep 29 20:06 test.py
-rw-r--r-- 1 bioinfo bioinfo 866 Sep 30 10:43 python_test.py
-rw-r--r-- 1 bioinfo bioinfo 1.2K Oct 1 20:44 jupyter_test.ipynb
(base) bioinfo@bioinfo:~/Scripts/python$ █
```

Jupyter Notebooks

jupyter python_test Last Checkpoint: 7 minutes ago (autosaved)

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter python_test Last Checkpoint: 7 minutes ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Help, Run, Stop, Cell, Kernel, Help, Code, Cell Type, Cell Kernel.
- File Menu:** New Notebook, Open..., Make a Copy..., Save as..., Rename..., Save and Checkpoint (Ctrl-S), Revert to Checkpoint, Print Preview, Download as, Trusted Notebook, Close and Halt.
- Download as Submenu:** AsciiDoc (.asciidoc), HTML (.html), LaTeX (.tex), Markdown (.md) (highlighted), Notebook (.ipynb), PDF via LaTeX (.pdf), reST (.rst), Python (.py), Reveal.js slides (.slides.html), PDF via puppeteer (.html).
- Code Cell Content:**#insert an state
states.insert(0, "Hawaii")
print(state)

#Loops
states = ["Alabama", "Alaska", "Arizona", "Arkansas", "California", "Colorado", "Connecticut", "Delaware", "Florida", "Georgia", "Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota", "Mississippi", "Missouri", "Montana", "Nebraska", "Nevada", "New Hampshire", "New Jersey", "New Mexico", "New York", "North Carolina", "North Dakota", "Ohio", "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina", "South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Washington", "West Virginia", "Wisconsin", "Wyoming"]
for state in states:
 print(state)
print("All states in list printed")

#dictionaires
plants = [{"color": "blue", "height": 100}, {"color": "red", "height": 150}, {"color": "green", "height": 200}, {"color": "yellow", "height": 250}, {"color": "purple", "height": 300}, {"color": "orange", "height": 350}, {"color": "pink", "height": 400}, {"color": "brown", "height": 450}, {"color": "grey", "height": 500}, {"color": "black", "height": 550}, {"color": "white", "height": 600}, {"color": "lightblue", "height": 650}, {"color": "lightred", "height": 700}, {"color": "lightgreen", "height": 750}, {"color": "lightyellow", "height": 800}, {"color": "lightpurple", "height": 850}, {"color": "lightorange", "height": 900}, {"color": "lightpink", "height": 950}, {"color": "lightgrey", "height": 1000}, {"color": "lightblack", "height": 1050}, {"color": "lightwhite", "height": 1100}],
for plant in plants:
 print(plant["color"], plant["height"])
- Terminal Output:**[I 21:00:42.236 NotebookApp] Shutting down 0 terminals
(base) bioinfo@bioinfo:~/Scripts/python\$ ls -lhtr
total 16K
-rw-r--r-- 1 bioinfo bioinfo 1.6K Sep 29 20:06 test.py
-rw-r--r-- 1 bioinfo bioinfo 866 Sep 30 10:43 python_test.py
-rw-r--r-- 1 bioinfo bioinfo 1.2K Oct 1 20:44 jupyter_test.ipynb
-rw-r--r-- 1 bioinfo bioinfo 2.6K Oct 1 20:57 python_test.ipynb
(base) bioinfo@bioinfo:~/Scripts/python\$ ls ~/Downloads/
python_test.md
(base) bioinfo@bioinfo:~/Scripts/python\$

Jupyter Notebooks

The image displays two side-by-side screenshots of a Jupyter Notebook interface. Both screenshots show a toolbar at the top with various icons for file operations, cell creation, and kernel management. Below the toolbar, there are two code cells.

Left Screenshot: The title bar says "jupyter jupyter_test Last Checkpoint: 40 minutes ago (unsaved changes)". The first cell contains the code "In [1]: print ("Plants are cool.")" and its output "Plants are cool.". The second cell contains the code "S. lycopersicum is a plant." and its output "S. lycopersicum is a plant." A black callout box labeled "open the command palette" points to the "Cell" menu icon in the toolbar. The main area below the cells shows three header sections: "Header 1", "Header 2", and "Header 3". Under "Header 3", there is a bulleted list: "• plant1", "• plant2", "• plant3", and "▪ flower".

Right Screenshot: The title bar says "jupyter jupyter_test Last Checkpoint: 41 minutes ago (autosaved)". The first cell contains the same code and output as the left screenshot. A command palette is open, titled "jupyter-notebook command group". It lists various commands with their descriptions and keyboard shortcuts. Some commands are grouped under "Header 1", "Header 2", and "Header 3". The commands include:

- Header 1: automatically indent selection (command mode) Y
- Header 2: change cell to code (command mode) 1
- Header 2: change cell to heading 1 (command mode) 2
- Header 2: change cell to heading 2 (command mode) 3
- Header 2: change cell to heading 3 (command mode) 4
- Header 2: change cell to heading 4 (command mode) 5
- Header 2: change cell to heading 5 (command mode) 6
- Header 2: change cell to heading 6 (command mode) M
- Header 2: change cell to markdown (command mode) R
- Header 3: change cell to raw (command mode) Esc
- Header 3: clear all cells output (command mode) 0, 0
- Header 3: clear cell output (command mode) 0
- Header 3: close the pager (command mode) Esc
- Header 3: C confirm restart kernel (command mode) 0, 0
- Header 3: confirm restart kernel and clear output (command mode) 0, 0
- Header 3: D confirm restart kernel and run all cells (command mode) 0, 0
- Header 3: C confirm shutdown kernel (command mode) 0, 0

Jupyter Resources

- https://jupyter.readthedocs.io/en/latest/use/use-cases/data_science.html

Jupyter for Data Science

The purpose of this page is to highlight kernels and other projects that are central to the usage of Jupyter in data science. This page is not meant to be comprehensive or unbiased, but rather to provide an opinionated view of the usage of Jupyter in data science based on our interactions with users.

The following Jupyter kernels are widely used in data science:

- **python**
 - IPython ([GitHub Repo](#))
- **R**
 - IRkernel ([Documentation](#), [GitHub Repo](#))
 - IRdisplay ([GitHub Repo](#))
 - repr ([GitHub Repo](#))
- **Julia**
 - IJulia Kernel ([GitHub Repo](#))
 - Interactive Widgets ([GitHub Repo](#))
- **Bash** ([GitHub Repo](#))

Jupyter Notebooks

- Opening an existing notebook

jupyter notebook notebook.ipynb

https://github.com/bcbc-group/PLSCi7202/blob/master/first_python.py

Now let's look at some example Python code

- Can run programs in Atom (ctl+shift + b)

```
``` python
(base) bioinfo@bioinfo:~/Scripts/python$ python
Python 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

---

- Can run Python in a terminal session

- Can run as a Python script from the command line

- **print("Python is fun!")**

```
1 #testing out python
2 print("Python is fun!")
```

```
base) bioinfo@bioinfo:~/Scripts/python$ python test.py
```

# Variables

- Labels that are assigned to values
- Names contain only letters, numbers, and underscores. They cannot start with a number and cannot contain spaces
- Do not use function names, for example print
- Keep names short but descriptive
- Example:

```
message = "Python is fun!"
print(message)
```

```
4 #using variables
5 message = "Python is fun!"
6 print(message)
-
```

# Strings

- A series of characters
- Are surrounded by single or double quotes
- Example:

**“This is a string”**

**‘This is also a string’**

# Methods

- An action Python can perform on a piece of data
- Example:

**name = “suzy strickler”**

**print(name.title())**

```
8 #using the title method
9 name = "suzy strickler"
10 print(name.title())
```

# Variables in Strings

- Example:

**first\_name = “suzy”**

**last\_name = “strickler”**

**full\_name = f”{first\_name} {last\_name}”**

**print(full\_name)**

**print(f”My name is, {full\_name.title()}!”)**

```
12 #using variables in strings
13 first_name = "suzy"
14 last_name = "strickler"
15 full_name = f"{first_name} {last_name}"
16 print(full_name)
17 print(f"My name is, {full_name.title()}!")
```

# Lists

- A collection of items in a particular order
- Example:

```
states = ['New York', 'Pennsylvania', 'Hawaii']
```

```
print(states)
```

- Accessing elements of a list:

```
print(states[0])
```

- Changing an element

```
states[0] = 'Texas'
```

```
print(states)
```

```
19 #Manipulating Lists
20 states = ['New York', 'Pennsylvania', 'Hawaii']
21 print(states)
22 print(states[0]) #access an element in a list
23 states[0] = 'Texas' #change a list element
24 print(states)
25 states.insert(0, 'Montana') #insert an element
26 print(states)
27 popped_states = states.pop(1) #remove an element
28 print(popped_states)
29 print(states)
30
```

# Lists cont'd

- Adding elements

```
states.insert(0, 'Montana')
```

```
print(states)
```

- Removing elements

```
popped_states = states.pop(1)
```

```
print(popped_states)
```

```
19 #Manipulating Lists
20 states = ['New York', 'Pennsylvania', 'Hawaii']
21 print(states)
22 print(states[0]) #access an element in a list
23 states[0] = 'Texas' #change a list element
24 print(states)
25 states.insert(0, 'Montana') #insert an element
26 print(states)
27 popped_states = states.pop(1) #remove an element
28 print(popped_states)
29 print(states)
30
```

# Loops

- Repeat an action with every item in a list
- Indentation
- Example

```
31 #Loops
32 states = ['New York', 'Pennsylvania', 'Hawaii']
for state in states:
 print(state)
35 print("All states in list printed")
36
```

**print("All states in list printed")**

# If statements

**for state in states:**

**if state == ‘Hawaii’:**

**print(state.upper())**

**elif state == “New York”:**

**print(state.lower())**

**else:**

**print(state.title())**

```
37 #Loops - if statements
38 states = ['New York', 'Pennsylvania', 'Hawaii']
39 for state in states:
40 if state == 'Hawaii':
41 print(state.upper())
42 elif state == 'Montana':
43 print(state.lower())
44 else: print(state.title())
```

# Dictionaries

- A collection of key-value pairs
- Example:

```
plants = {'color': 'blue', 'height': 10}
```

```
print(plants['color'])
```

```
#add a new key-value pair
```

```
plants['sepal'] = 3
```

```
print(plants)
```

```
#remove a key-value pair
```

```
del plants['color']
```

```
46 #Dictionaries
47 plants = {'color': 'blue', 'height': 10}
48 print(plants['color'])
49 plants['sepal'] = 3 #add a new key-value pair
50 print(plants)
51 del plants['color'] #remove a key-value pair
52 print(plants)
53
```

# Functions

- Blocks of code that do a specific job
- docstring = describes what the function does

- Example

```
def weather():
 """Find out the weather"""
 answer = input("How is the weather?")
 print(answer)
```

```
54 #example of a function
55 def weather():
56 """Find out the weather"""
57 answer = input("How is the weather?")
58 print(answer)
59 weather()
60
```

**answer = input("How is the weather?")**

**print(answer)**

- Module - storing function in a separate file and then import it (**import weather**)

# Reading from a file

- In many cases, you will want your script to operate on some input file
- Example:

```
fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")
```

```
if fasta_file.mode == "r":
```

```
 contents = fasta_file.read()
```

```
 print(contents)
```

```
 fasta_file.close()
```

# Writing output to a file

- Example:

```
fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")
```

```
if fasta_file.mode == "r":
```

```
 contents = fasta_file.read()
```

```
 outF = open("/home/bioinfo/Data/test_out.fasta", "w")
```

```
 outF.write(contents)
```

```
 print(contents)
```

```
 fasta_file.close()
```

```
61 #reading in a file
62 fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")
63 if fasta_file.mode == "r":
64 contents = fasta_file.read()
65 outF = open("/home/bioinfo/Data/test_out.fasta", "w")
66 outF.write(contents)
67 fasta_file.close()
68 outF.close()
```

# General Introduction - R and RStudio

- What is R?
  - A programming, ‘scripting’ language
- Why use R?
  - Community
  - Free, open-source
  - Useful for statistics and data visualization
- How to use R?
  - We will use RStudio, a user-friendly interface



# Install R Studio

- <https://www.rstudio.com/products/rstudio/download/#download>
- sudo dpkg -i rstudio-2021.09.0-351-amd64.deb

# R Markdown

The screenshot shows the RStudio interface with two R Notebook windows open. The left pane is titled "Untitled1\*" and the right pane is titled "Untitled2\*". Both panes have the "R Notebook" tab selected in the top menu bar.

The code in both panes is identical, demonstrating the basic features of R Markdown:

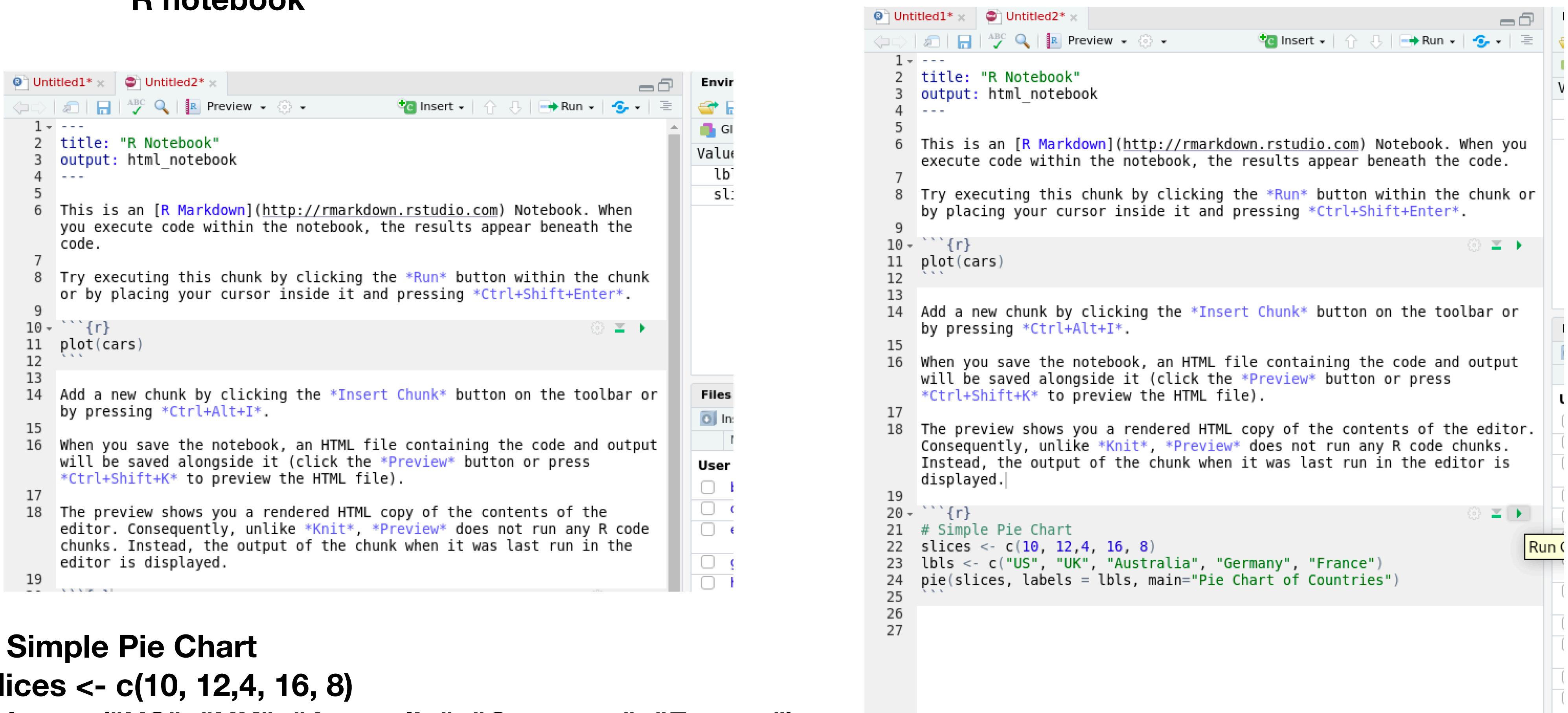
```
1 ---
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you
execute code within the notebook, the results appear beneath the code.
7
8 Try executing this chunk by clicking the *Run* button within the chunk or
by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
9
10```{r}
11 plot(cars)
12```
13
14 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or
by pressing *Ctrl+Alt+I*.
15
16 When you save the notebook, an HTML file containing the code and output
will be saved alongside it (click the *Preview* button or press
Ctrl+Shift+K to preview the HTML file).
17
18 The preview shows you a rendered HTML copy of the contents of the editor.
Consequently, unlike *Knit*, *Preview* does not run any R code chunks.
Instead, the output of the chunk when it was last run in the editor is
displayed.
19
20```{r}
21 # Simple Pie Chart
22 slices <- c(10, 12, 4, 16, 8)
23 lbls <- c("US", "UK", "Australia", "Germany", "France")
24 pie(slices, labels = lbls, main="Pie Chart of Countries")
25
26
27
28```{r}
29 # Simple Pie Chart
30 slices <- c(10, 12, 4, 16, 8)
31 lbls <- c("US", "UK", "Australia", "Germany", "France")
32 pie(slices, labels = lbls, main="Pie Chart of Countries")
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
```

The right pane also displays a "Run" dropdown menu with several options:

- Run
- Package for Dynamic R
- Editor for R
- With the C Library
- ME Types
- ipes and Core R and
- for R
- stringi Character String Processing Facilities
- stringr Simple, Consistent Wrappers for Common String Operations
- tinytex Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents
- xfun Miscellaneous Functions by 'Yihui Xie'

# R Markdown

## R notebook



The image shows two RStudio windows side-by-side, illustrating the R Notebook interface.

**Left Window (Untitled1):**

- Toolbar: Insert, Run, Preview, etc.
- Code pane:

```
1 ---
2 title: "R Notebook"
3 output: html_notebook
4 ---
5 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
6 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
7
8 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
9
10 ``{r}
11 plot(cars)
12
13 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
14 When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
15
16 The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
17
18
19
```
- Environment pane: Envir, Value, Label, Selection.
- Files pane: Shows files: In progress, User, etc.

**Right Window (Untitled2):**

- Toolbar: Insert, Run, Preview, etc.
- Code pane:

```
1 ---
2 title: "R Notebook"
3 output: html_notebook
4 ---
5 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
6 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
7
8 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
9
10 ``{r}
11 plot(cars)
12
13 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
14 When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
15
16 The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
17
18
19 ``{r}
20 # Simple Pie Chart
21 slices <- c(10, 12, 4, 16, 8)
22 lbls <- c("US", "UK", "Australia", "Germany", "France")
23 pie(slices, labels = lbls, main="Pie Chart of Countries")
24
25
26
27
```

```
Simple Pie Chart
slices <- c(10, 12, 4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```

# R Markdown Cheat Sheet

<https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf>

## R Markdown Cheat Sheet

learn more at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

rstudio 0.25.0 Updated: 8/14



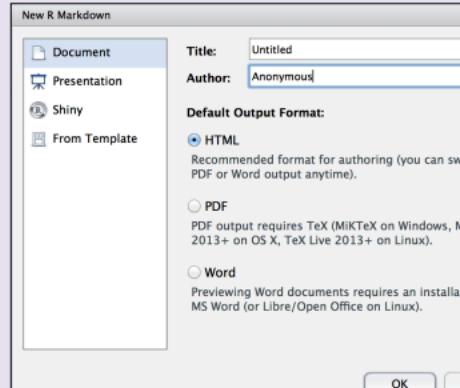
**1. Workflow** R Markdown is a format for writing reproducible, dynamic reports with R. Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more. To make a report:

i. Open - Open a file that uses the .Rmd extension. ii. Write - Write content with the easy to use R Markdown syntax. iii. Embed - Embed R code that creates output to include in the report. iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.



**2. Open File** Start by saving a text file with the extension .Rmd, or open an RStudio Rmd template

- In the menu bar, click **File > New File > R Markdown...**
- A window will open. Select the class of output you would like to make with your .Rmd file
- Select the specific type of output to make with the radio buttons (you can change this later)
- Click OK



**3. Markdown** Next, write your report in plain text. Use markdown syntax to describe how to format text in the final report.

syntax	becomes
Plain text End a line with two spaces to start a new paragraph. *italics* and _italics_ **bold** and __bold__ superscript <sup>2</sup> ~~strikethrough~~ [link](www.rstudio.com)	Plain text End a line with two spaces to start a new paragraph. italics and italics bold and bold superscript <sup>2</sup> strikethrough~~ [link]
# Header 1	<b>Header 1</b>
## Header 2	<b>Header 2</b>
### Header 3	<b>Header 3</b>
#### Header 4	<b>Header 4</b>
##### Header 5	<b>Header 5</b>
###### Header 6	<b>Header 6</b>
endash: --	endash: –
emdash: ---	emdash: —
ellipsis: ...	ellipsis: ...
inline equation: \$A = \pi r^2\$	inline equation: $A = \pi r^2$
image:	image: 
horizontal rule (or slide break):	horizontal rule (or slide break):
***	
> block quote	block quote
* unordered list	• unordered list
* item 2	• item 2
+ sub-item 1	◦ sub-item 1
+ sub-item 2	◦ sub-item 2
1. ordered list	1. ordered list
2. item 2	2. item 2
+ sub-item 1	◦ sub-item 1
+ sub-item 2	◦ sub-item 2
Table Header   Second Header	Table Header      Second Header
----- -----	-----
Table Cell   Cell 2	Table Cell      Cell 2
Cell 3   Cell 4	Cell 3      Cell 4

**4. Choose Output** Write a YAML header that explains what type of document to build from your R Markdown file.

**YAML**  
A YAML header is a set of key: value pairs at the start of your file. Begin and end the header with a line of three dashes (---).  
  
This is the start of my report. The above is metadata saved in a YAML header.

The output value determines which type of file R will build from your .Rmd file (in Step 6)

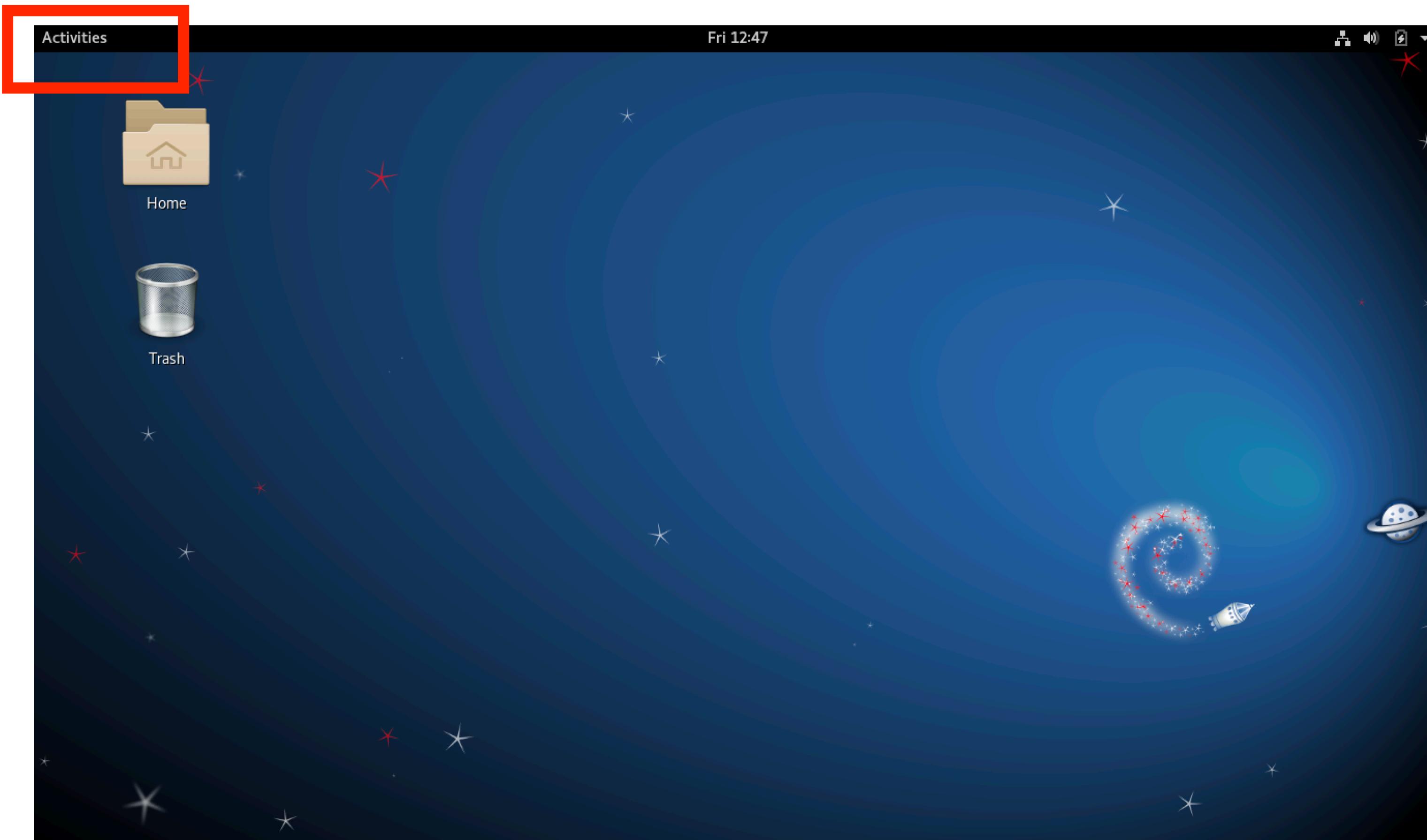
output: html_document	.....	html file (web page)
output: pdf_document	.....	pdf document
output: word_document	.....	Microsoft Word .docx
output: beamer_presentation	.....	beamer slideshow (pdf)
output: ioslides_presentation	.....	ioslides slideshow (html)

The RStudio template writes the YAML header for you

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

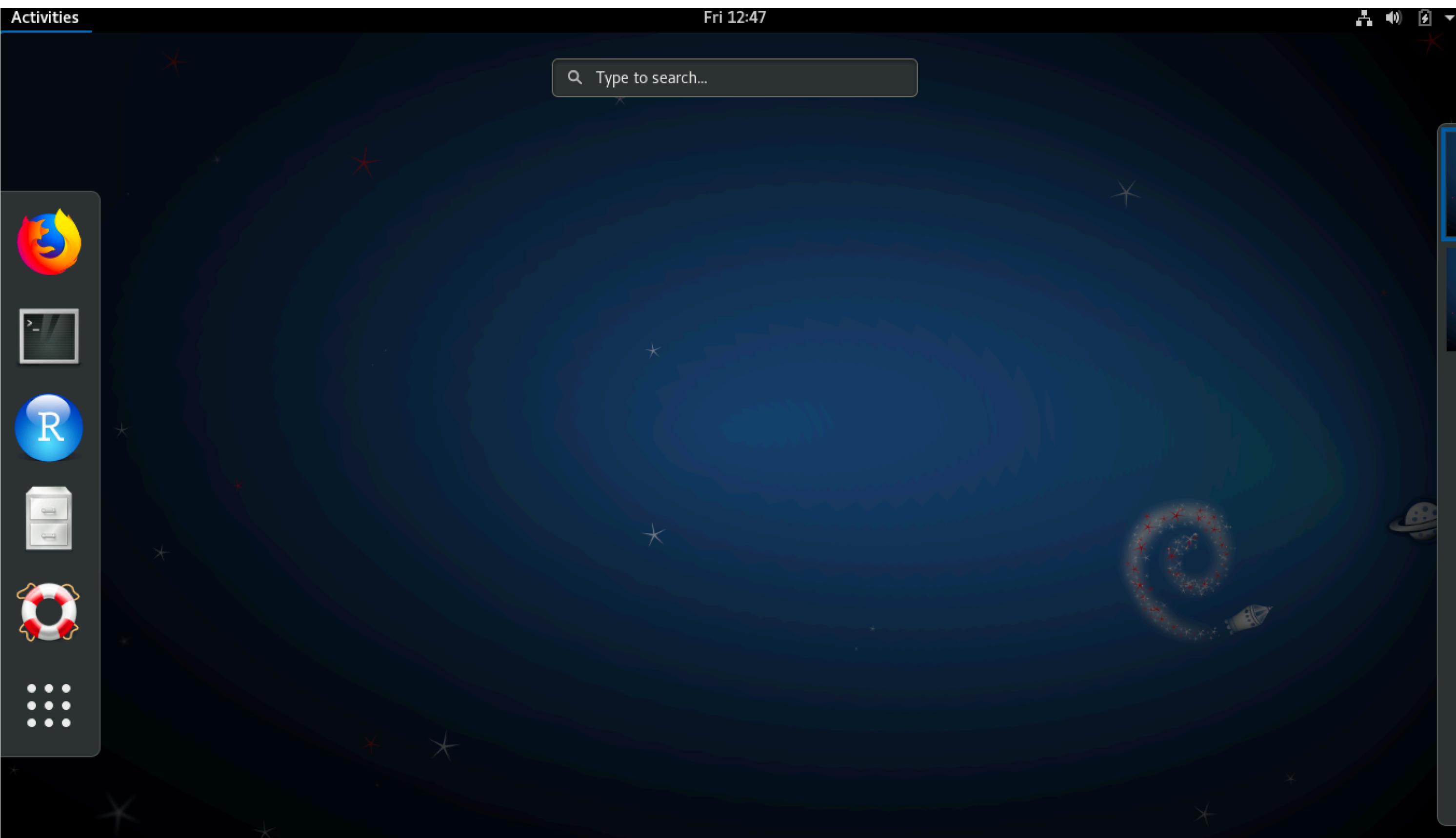
# General Introduction - R and RStudio

- Let's open RStudio



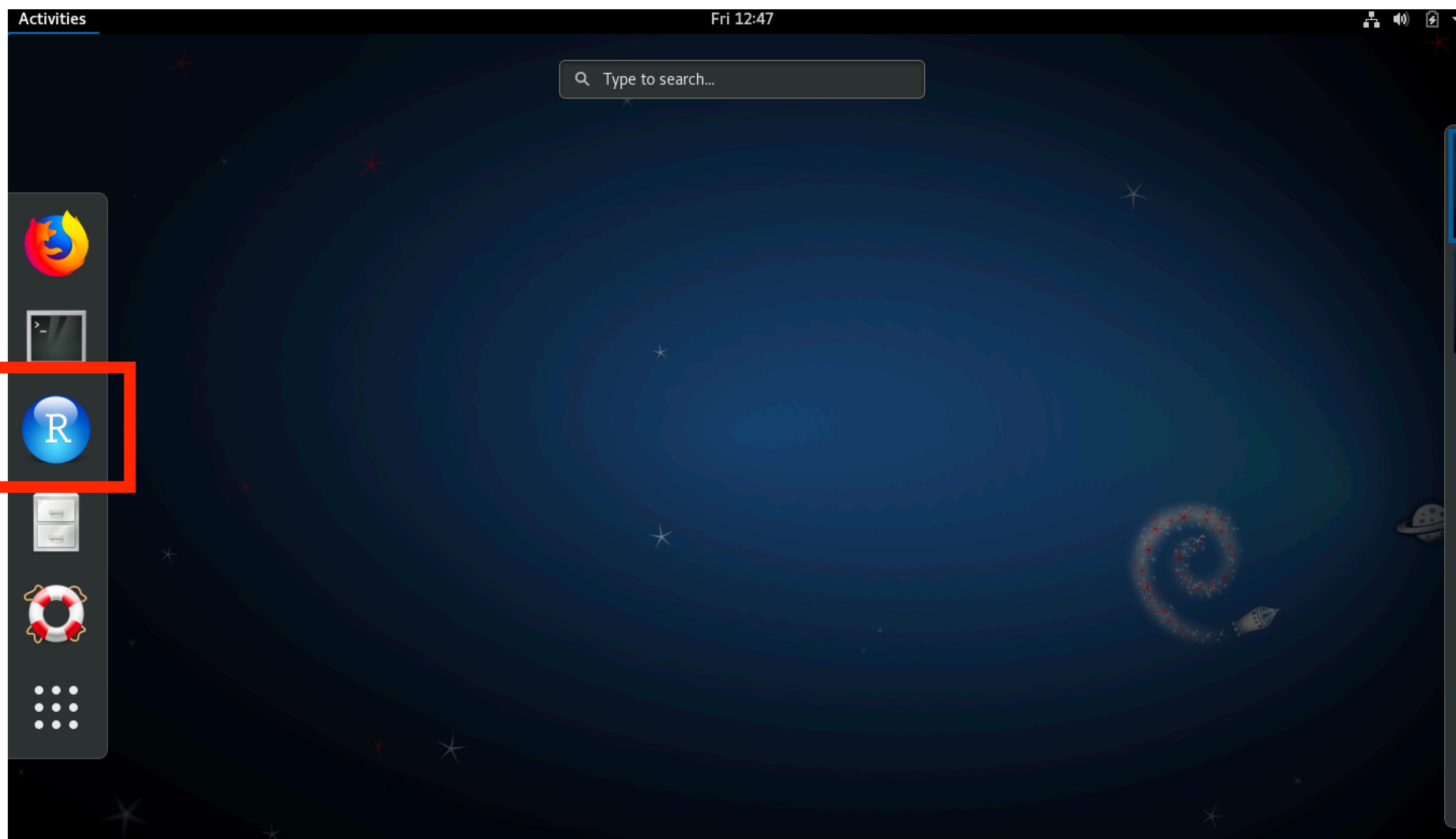
# General Introduction - R and RStudio

- Let's open RStudio



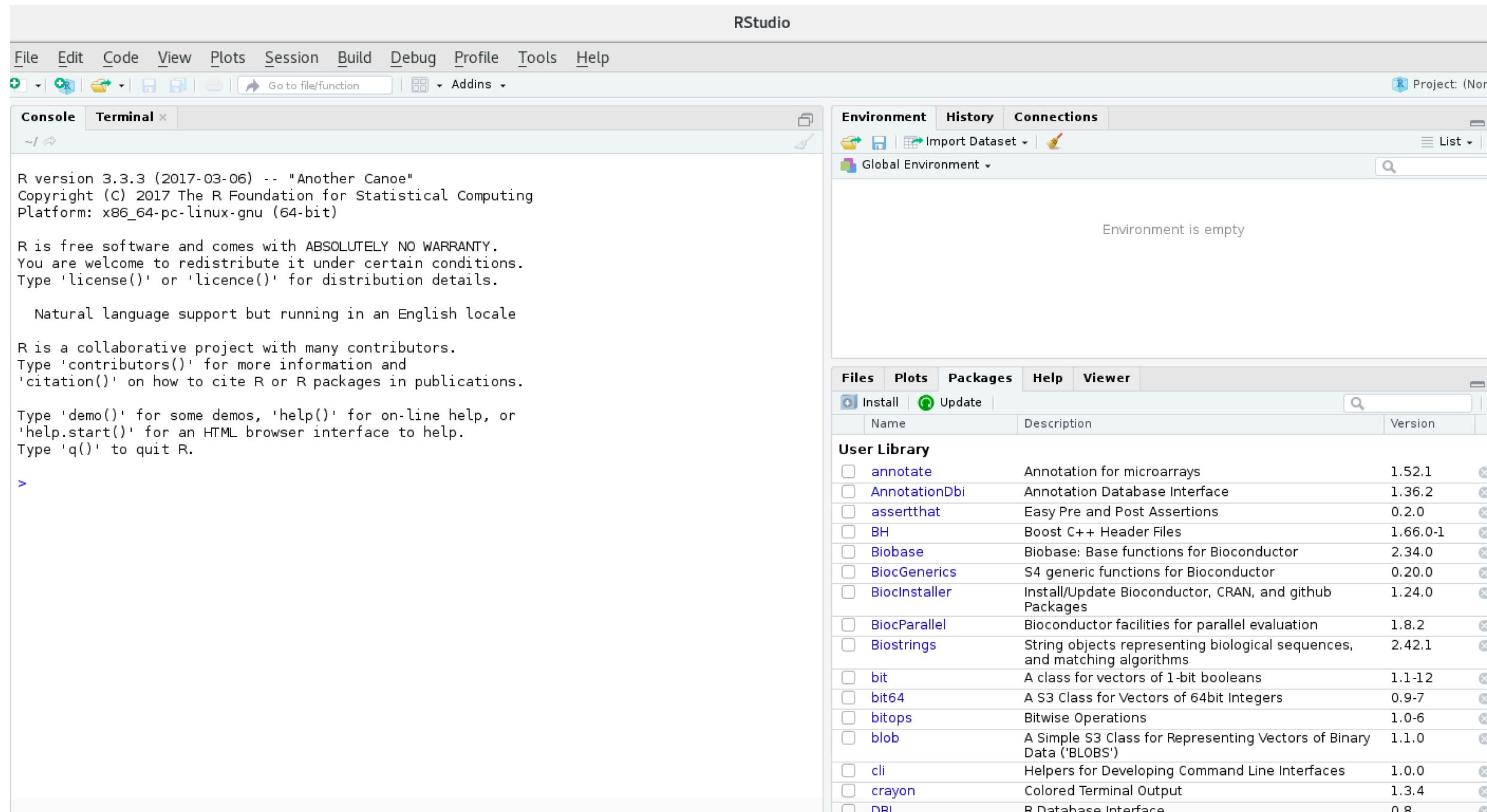
# General Introduction - R and RStudio

- Let's open RStudio



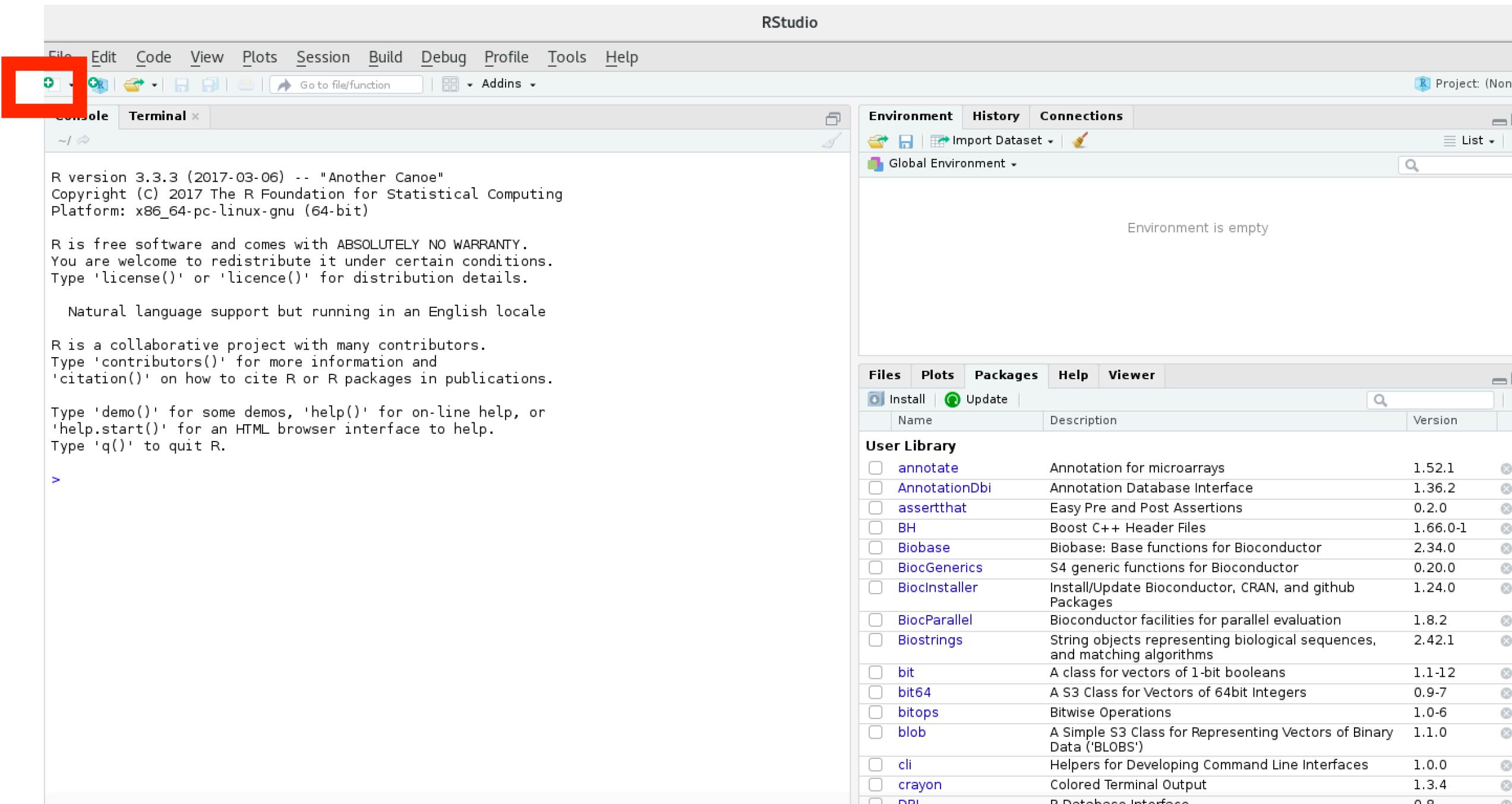
# General Introduction - R and RStudio

- Getting oriented in RStudio



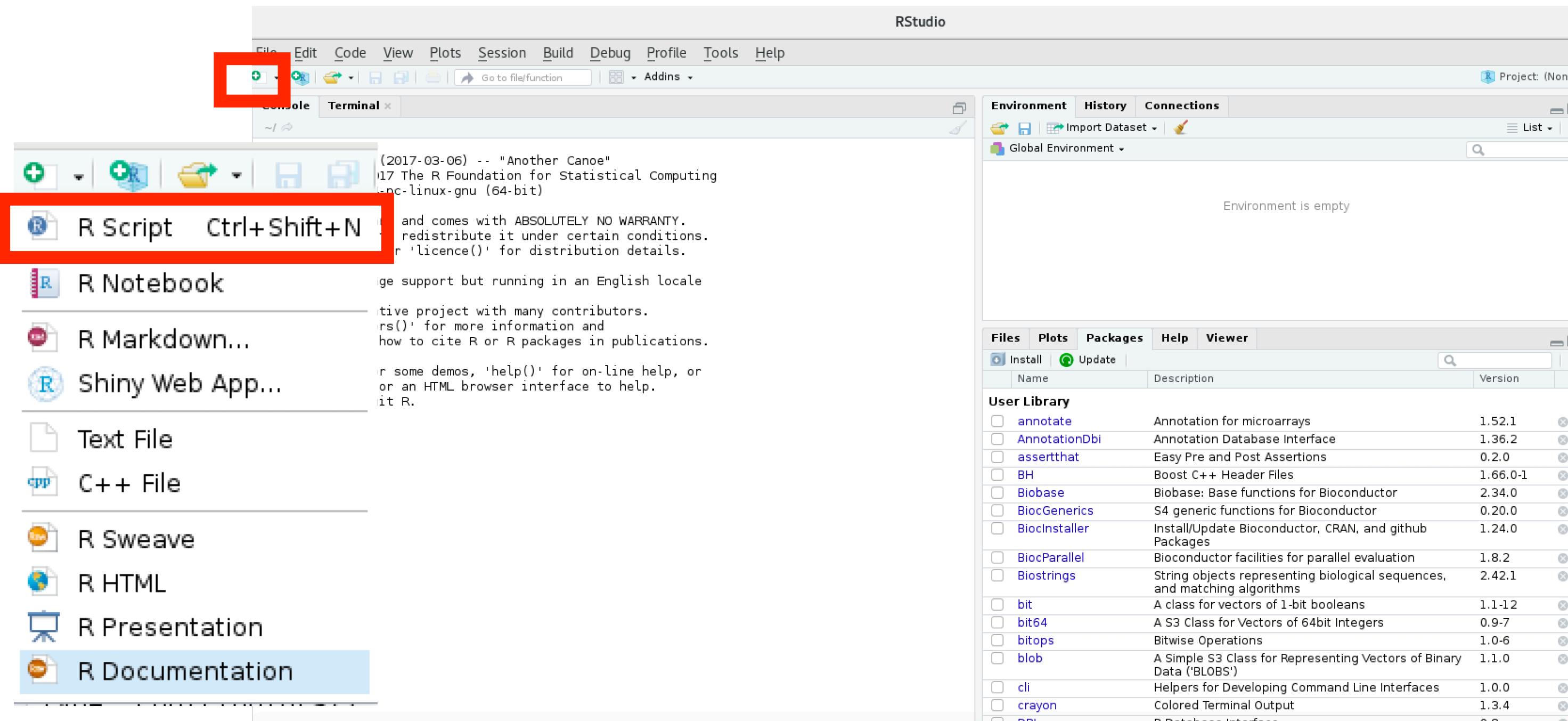
# General Introduction - R and RStudio

- Getting oriented in RStudio



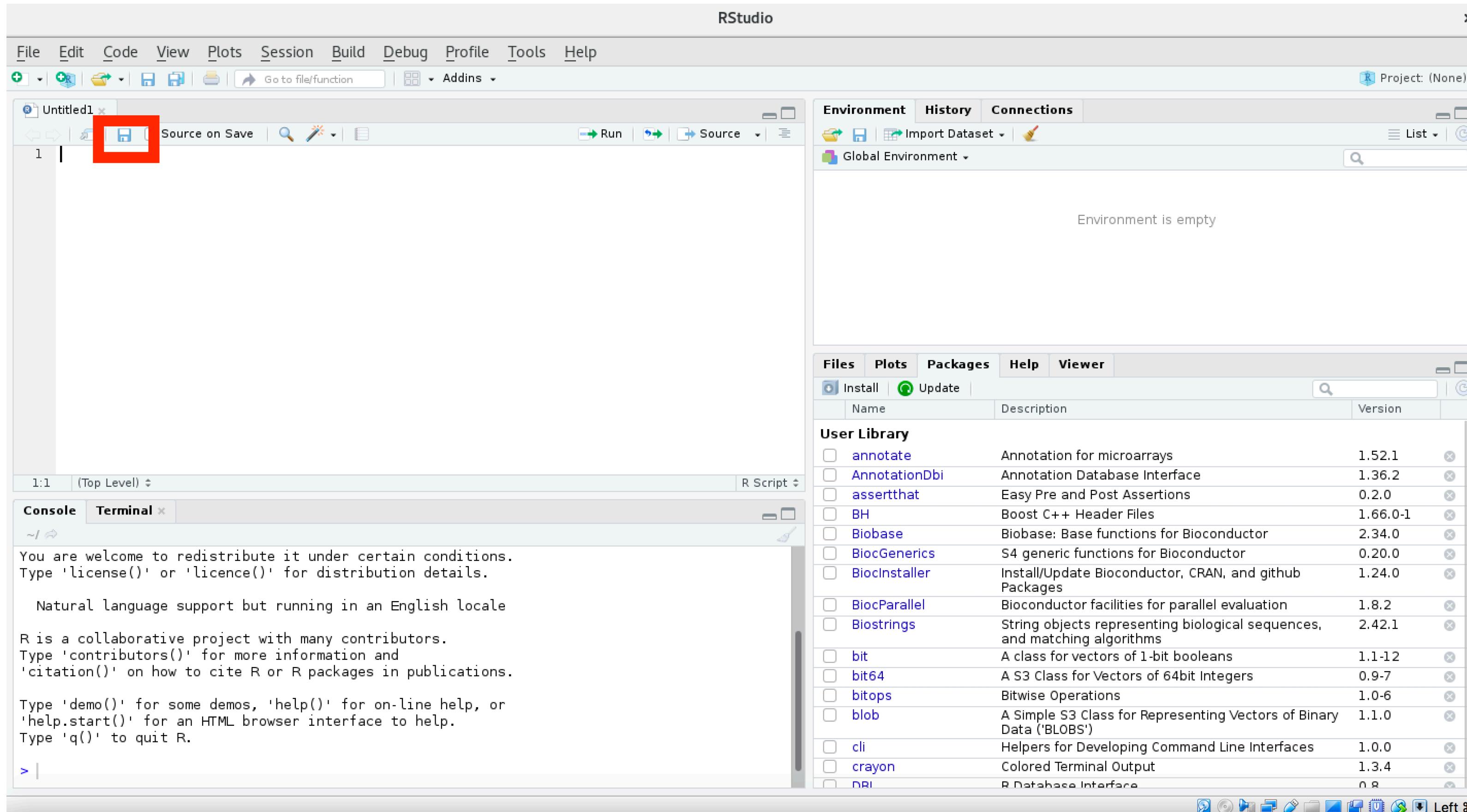
# General Introduction - R and RStudio

- Getting oriented in RStudio



# General Introduction - R and RStudio

- Getting oriented in RStudio



# General Introduction - R and RStudio

- Getting oriented in RStudio

The screenshot shows the RStudio IDE interface with the following components:

- Source:** A code editor window titled "Untitled1" where you can write your R code.
- Console:** A terminal window displaying the R startup message and basic help information.
- Environment/History:** A panel showing the Global Environment and a history of operations.
- Files/Plots/Packages/Help:** A package manager window showing the User Library with various R packages listed by name, description, and version.

**1. Source: where you write your code.**

**2. Console: where your code is evaluated.**

**3. Environment/History**

**4. Files/Plots/Packages/Help**

Name	Description	Version
assertthat	Easy file and post assertions	0.2.0
BH	Boost C++ Header Files	1.66.0-1
Biobase	Biobase: Base functions for Bioconductor	2.34.0
BiocGenerics	S4 generic functions for Bioconductor	0.20.0
BiocInstaller	Install/Update Bioconductor, CRAN, and github Packages	1.24.0
BiocParallel	Bioconductor facilities for parallel evaluation	1.8.2
Biostrings	String objects representing biological sequences, and matching algorithms	2.42.1
bit	A class for vectors of 1-bit booleans	1.1-12
bit64	A S3 Class for Vectors of 64bit Integers	0.9-7
bitops	Bitwise Operations	1.0-6
blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.1.0
cli	Helpers for Developing Command Line Interfaces	1.0.0
crayon	Colored Terminal Output	1.3.4
RDI	R Database Interface	0.8

# Location & Working Directory

To see your working directory (similar to pwd in Terminal):

```
getwd()
```

To set your working directory (similar to cd in Terminal):

```
setwd("./")
```

To list what is in the directory (similar to ls in Terminal):

```
dir()
```



# Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

`56+29`

Multiplication:

`3*12`

Division:

`40/8`

Exponents:

`2^12`

Try to calculate the following:

`( (8 / 4) - 4.8 + (12 / (2+1) ) )^3`



# Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

`56+29`

Multiplication:

`3*12`

Division:

`40/8`

Exponents:

`2^12`

Try to calculate the following:

`( (8 / 4) - 4.8 + (12 / (2+1) ) )^3`

1.728



# Scalar Variables & Data Types

Variables can be used to “store” values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75
```

```
number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```



# Scalar Variables & Data Types

Variables can be used to “store” values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```



# Scalar Variables & Data Types

Variables can be used to “store” values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75
```

```
number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?



# Scalar Variables & Data Types

Variables can be used to “store” values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75
```

```
number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

```
number.of.round.progeny.peas <- number.of.total.progeny.peas * (1 - ratio.wrinkled)
number.of.round.progeny.peas
```



# Scalar Variables & Data Types

- A few tips
  - Choose meaningful variable names
  - Separate words using dots or underscores (no spaces allowed)
  - Variable names are case-sensitive



# Scalar Variables & Data Types

We can use different data types in our variables. To check the type of a variable, use `class()`:

Numeric

```
my_lucky_number <- 7

class(my_lucky_number)

class(number.of.round.progeny.peas)
```

Character

```
cool_fruit <- "tomato"

class(cool_fruit)
```



# Scalar Variables & Data Types

## Logical

```
my_name_is_Adrian <- TRUE
```

```
class(my_name_is_Adrian)
```



# Scalar Variables & Data Types

## Logical

```
my_name_is_Adrian <- TRUE
```

```
class(my_name_is_Adrian)
```

What would `class()` tell you for the following?

```
tomato_is_the_best_fruit <- "TRUE"
```

```
class(tomato_is_the_best_fruit)
```



# Functions

- A function is basically just a set of instructions
  - Takes some input
  - Does some things
  - Gives an output
- We can access functions in a few ways...



# Functions

- We can define our own custom functions

Let's define and use a simple function:

```
my-awesome-function <- function(input1){
 output <- input1^12 * 5 + 3
 return(output)
}

my-awesome-function(3)

my-awesome-function(5)

my-awesome-new-variable <- my-awesome-function(7)

my-awesome-new-variable
```



# Functions

- R also has 'pre-made' functions... can you think of any examples that we've used?



# Functions

- R also has 'pre-made' functions... can you think of any examples that we've used?

To access R documentation for a function, etc.:

```
?class
```

```
?plot
```



# Functions

- We can also install packages and load libraries to access sets of additional functions...



# Packages

- Let's try installing/loading a package called ggplot2:

```
install.packages("ggplot2")
```

```
library(ggplot2)
```



# Data Structures

- Vectors
- Matrices
- Data frames
- Lists



# Data Structures: Vectors

```
vector1 <- c(1,2,5.3,6,-2,4)
vector1
[1] 1.0 2.0 5.3 6.0 -2.0 4.0

vector2 <- c("one","two","three")
vector2
[1] "one" "two" "three"

vector3 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
vector3
[1] TRUE TRUE TRUE FALSE TRUE FALSE

vector4 <- rep(1:3, each=2)
vector4
[1] 1 1 2 2 3 3
```



# Data Structures: Vectors

```
vector5 <- rep(c(1, 0, 7), times = 5) #repeats c(1,0,7) vector 5 times
vector5

[1] 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7

vector6 <- rep(c(0, 4), times = c(5,3)) # you could tell R how many times to repeat each value.
vector6

[1] 0 0 0 0 0 4 4 4

vector7 <- rep(1:4,length.out=6) # you could also repeat the vector to the specified length even if the
vector7

[1] 1 2 3 4 1 2

vector8 <- seq(from = 7.5, to = 2.5, by = -0.5) # vector with numbers 7.5 to 2.5 in steps of 0.5.
vector8

[1] 7.5 7.0 6.5 6.0 5.5 5.0 4.5 4.0 3.5 3.0 2.5
```



# Data Structures: Vectors

We can transpose a vector:

```
t(vector1)
```

```
[,1] [,2] [,3] [,4] [,5] [,6]
[1,] 1 2 5.3 6 -2 4
```

We can access elements in a vector using the element's vector position:

```
vector1[c(2,4)]
```

```
[1] 2 6
```

We can add, subtract, multiply, divide, square root, and exponentiate vectors and scalars of numeric class. These can be done using operations:

```
vector1
```

```
[1] 1.0 2.0 5.3 6.0 -2.0 4.0
```

```
vector7
```

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

```
vector1 + vector7
```

```
[1] 2.0 4.0 8.3 10.0 -1.0 6.0
```



# Data Structures: Vectors

We can apply functions on vectors:

```
mean(vector1)
```

```
[1] 2.716667
```

```
median(vector1)
```

```
[1] 3
```



# Data Structures: Matrices

We can create matrix using `matrix(c(), byrow=, ncol=)`. Matrices can mix element class types.

```
matrix1 <- matrix(c(1,3,2,2,8,9), ncol=2)
matrix1

[,1] [,2]
[1,] 1 2
[2,] 3 8
[3,] 2 9

matrix2 <- matrix(c(1,3,2,2,8,9), ncol=2, byrow = T)
matrix2

[,1] [,2]
[1,] 1 3
[2,] 2 2
[3,] 8 9
```



# Data Structures: Matrices

We can access elements in matrix using element position:

```
c(matrix1[1,2], matrix1[2,2],matrix1[3,1])
```

```
[1] 2 8 2
```

Matrix operations (matrix multiplication, etc.) can also be performed.

Transpose of matrices (It is basically to interchange of rows and columns:

```
t(matrix1)
```

```
[,1] [,2] [,3]
[1,] 1 3 2
[2,] 2 8 9
```

We can also create vectors from a matrix:

```
matrix1[2,]
```

```
[1] 3 8
```

```
matrix1[,2]
```

```
[1] 2 8 9
```



# Data Structures: Data Frames

Data frames are used for storing data tables. It is usually a list of equal vectors of the same length.

```
c <- c(2, 3, 5)
d <- c("aa", "bb", "cc")
e <- c(TRUE, FALSE, TRUE)
df <- data.frame(c, d, e)
colnames(df) <- c("numeric", "character", "logical") # Add column names
df

numeric character logical
1 2 aa TRUE
2 3 bb FALSE
3 5 cc TRUE
```



# Reading Data into R

- Use functions such as:
  - `read.table()`
  - `read.csv()`



# R Package Repos

- CRAN

```
> install.packages("fortunes")
```

- Bioconductor

```
source("https://bioconductor.org/biocLite.R")
biocLite("S4Vectors")
```

- GitHub

```
library(githubinstall)
githubinstall("AnomalyDetection")
```

# When would you use bash vs R vs Python?

- Designing a data analysis pipeline that runs several commands and programs
- Writing a more sophisticated program to deploy a machine learning approach on a large data set
- Performing statistical analysis on a large data set

# When would you use bash vs R vs Python?

- Bash
  - Simple, great performance wise, good for piping output from one command into another command
  - Gets complicated with larger projects, not very elegant
- Python
  - Is more general purpose and can be used beyond data analysis, elegant
  - Not as many libraries as R
- R
  - Great for visualization (plotting) and for statistical analysis
  - Can be slow, code not as elegant, can be more difficult to learn
- Also depends on what you are comfortable with, some people prefer R, some prefer Python.

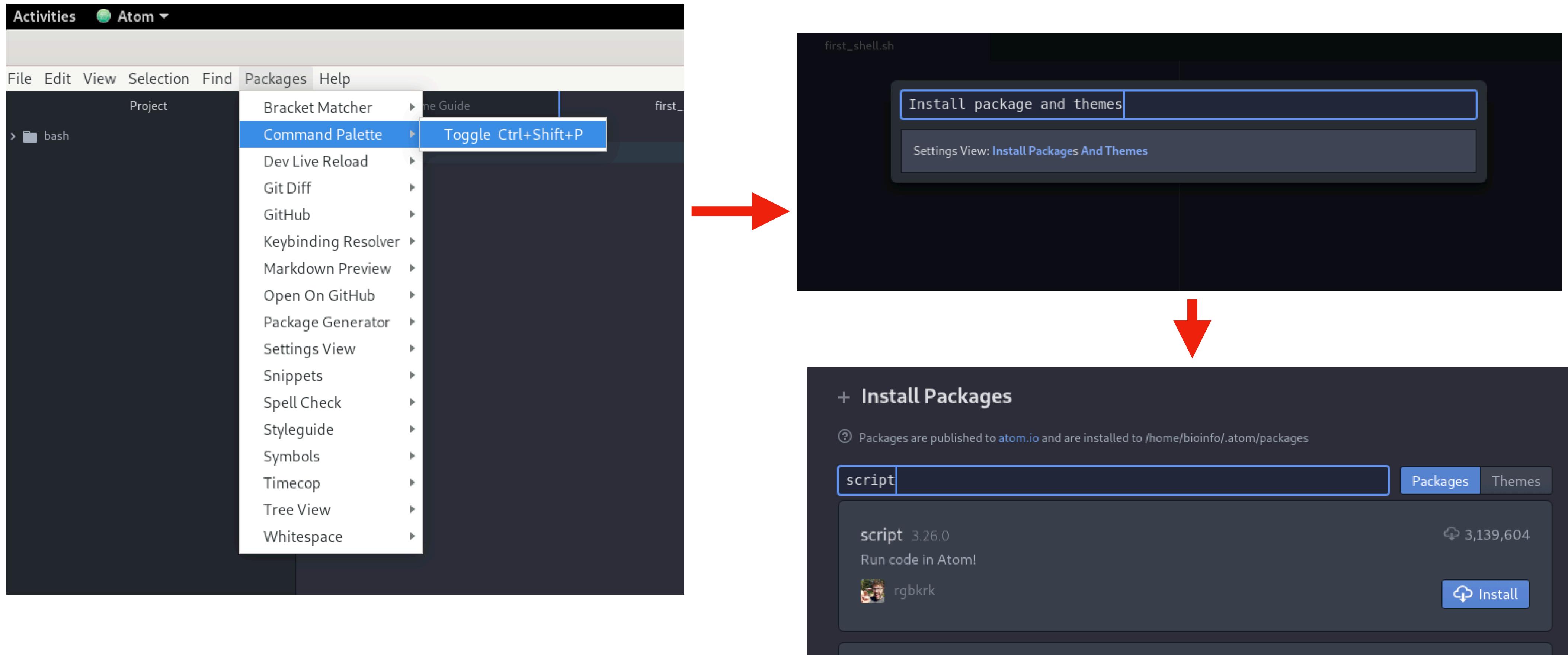
- All scripts from today will be added to GitHub here (in addition to on Canvas):

<https://github.com/bcbc-group/PLSCI7202>

# Assignment for next time

- Write an R or Python script that performs a task that will be useful for your research. If relevant, you can automate it with a bash shell script. Feel free to discuss with others!
- Please complete your script(s) before next class. We will use these in our next class to demonstrate best practices.

# Install script Package in Atom



- Ctl + shift + b