Funded by the National Science Foundation, Grant #2320373

CyberInfrastructure Training and Education for Synchrotron X-Ray Science (X-CITE)

Programming Essentials: An Introduction to Python and Jupyter

Anirban Mandal, Erik Scott, Sajith Sasidharan (RENCI, UNC Chapel Hill)
Ewa Deelman, Karan Vahi, Mats Rynge (ISI, USC)
Matthew Miller, Werner Sun, Peter Ko, Kelly Nygren, Keara Soloway, Rolf Verberg (CHESS, Cornell)
Brandon Sorge (IUPUI)

NSF OAC Award Numbers: 2320373, 2320375, 2320374

Funded by the National Science Foundation, Grant #2320373

Level Setting

We have one hour.

My goals:

1. Introduce you to working with the Jupyter environment
2. Highlight some of the basics of Python for people who have only used other languages (C, Java, R, Matlab…) or perhaps even no programming experience
3. Can you can program in Python at the level of writing loops and functions? You shouldn't be here, you should be writing papers and proposals. We'll see you at 9:55.

Funded by the National Science Foundation, Grant #2320373

On the Web

Our training materials are on CHESS's mass storage (/nfs/chess/user/x-cite/X-CITE) for your convenience.

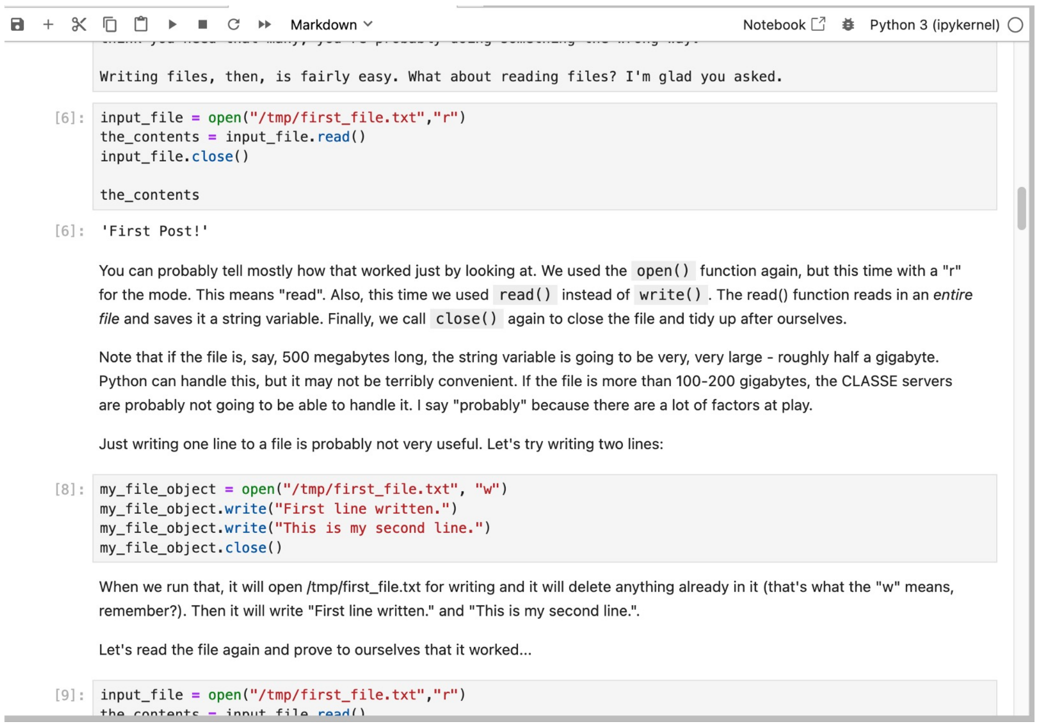
Their permanent home is https://xcitecourse.org/.

Funded by the National Science Foundation, Grant #2320373

Jupyter

Jupyter is a programming environment that lets you mix programming and free-form text in a web-based "notebook".

Strongly reminiscent of Mathematica, but the idea dates back to the 1970s (Don Knuth)



Writing files, then, is fairly easy. What about reading files? I'm glad you asked.

```
[6]: input_file = open("/tmp/first_file.txt","r")
     the_contents = input_file.read()
     input_file.close()

     the_contents
```

```
[6]: 'First Post!'
```

You can probably tell mostly how that worked just by looking at. We used the `open()` function again, but this time with a "r" for the mode. This means "read". Also, this time we used `read()` instead of `write()`. The read() function reads in an *entire file* and saves it a string variable. Finally, we call `close()` again to close the file and tidy up after ourselves.

Note that if the file is, say, 500 megabytes long, the string variable is going to be very, very large - roughly half a gigabyte. Python can handle this, but it may not be terribly convenient. If the file is more than 100-200 gigabytes, the CLASSE servers are probably not going to be able to handle it. I say "probably" because there are a lot of factors at play.

Just writing one line to a file is probably not very useful. Let's try writing two lines:

```
[8]: my_file_object = open("/tmp/first_file.txt", "w")
     my_file_object.write("First line written.")
     my_file_object.write("This is my second line.")
     my_file_object.close()
```

When we run that, it will open /tmp/first_file.txt for writing and it will delete anything already in it (that's what the "w" means, remember?). Then it will write "First line written." and "This is my second line.".

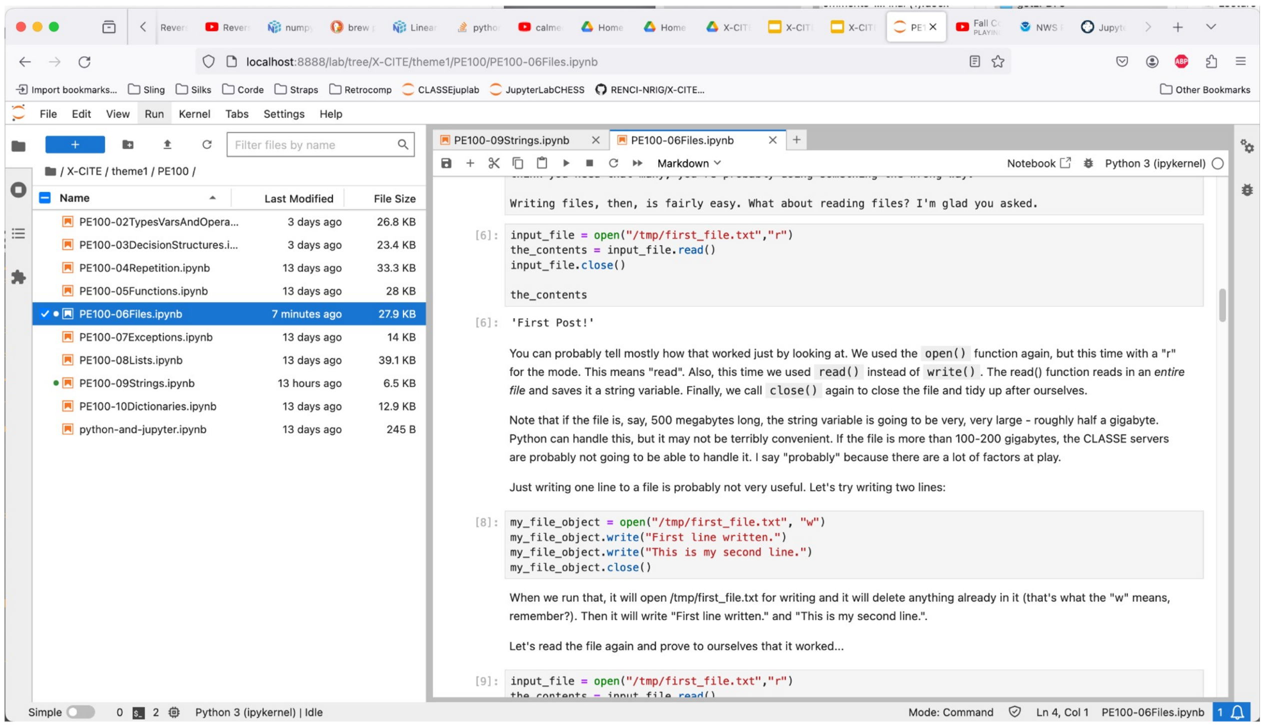
Let's read the file again and prove to ourselves that it worked...

```
[9]: input_file = open("/tmp/first_file.txt","r")
```

Funded by the National Science Foundation, Grant #2320373

Jupyter Lab (newer, better Jupyter)

Jupyter Lab adds a file browser, nicer interface, a built-in debugger, and much more. It's the direction Jupyter is moving in.


localhost:8888/lab/tree/X-CITE/theme1/PE100/PE100-06Files.ipynb
Import bookmarks... Sling Silks Corde Straps Retrocomp CLASSEjuplab JupyterLabCHESS RENCI-NRIG/X-CITE... Other Bookmarks
File Edit View Run Kernel Tabs Settings Help
Filter files by name
/ X-CITE / theme1 / PE100 /
PE100-09Strings.ipynb
PE100-06Files.ipynb
Markdown
Notebook
Python 3 (ipykernel)
Simple 0 2 Python 3 (ipykernel) | Idle
Mode: Command Ln 4, Col 1 PE100-06Files.ipynb 1

Name	Last Modified	File Size
PE100-02TypesVarsAndOpera...	3 days ago	26.8 KB
PE100-03DecisionStructures.i...	3 days ago	23.4 KB
PE100-04Repetition.ipynb	13 days ago	33.3 KB
PE100-05Functions.ipynb	13 days ago	28 KB
PE100-06Files.ipynb	7 minutes ago	27.9 KB
PE100-07Exceptions.ipynb	13 days ago	14 KB
PE100-08Lists.ipynb	13 days ago	39.1 KB
PE100-09Strings.ipynb	13 hours ago	6.5 KB
PE100-10Dictionaries.ipynb	13 days ago	12.9 KB
python-and-jupyter.ipynb	13 days ago	245 B

```
Writing files, then, is fairly easy. What about reading files? I'm glad you asked.
```

```
[6]: input_file = open("/tmp/first_file.txt","r")
     the_contents = input_file.read()
     input_file.close()

     the_contents

[6]: 'First Post!'
```

You can probably tell mostly how that worked just by looking at. We used the `open()` function again, but this time with a "r" for the mode. This means "read". Also, this time we used `read()` instead of `write()`. The read() function reads in an *entire file* and saves it a string variable. Finally, we call `close()` again to close the file and tidy up after ourselves.

Note that if the file is, say, 500 megabytes long, the string variable is going to be very, very large - roughly half a gigabyte. Python can handle this, but it may not be terribly convenient. If the file is more than 100-200 gigabytes, the CLASSE servers are probably not going to be able to handle it. I say "probably" because there are a lot of factors at play.

Just writing one line to a file is probably not very useful. Let's try writing two lines:

```
[8]: my_file_object = open("/tmp/first_file.txt", "w")
     my_file_object.write("First line written.")
     my_file_object.write("This is my second line.")
     my_file_object.close()
```

When we run that, it will open /tmp/first_file.txt for writing and it will delete anything already in it (that's what the "w" means, remember?). Then it will write "First line written." and "This is my second line.".

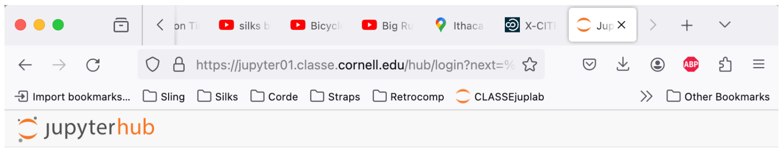
Let's read the file again and prove to ourselves that it worked...

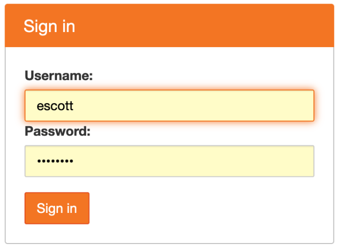
```
[9]: input_file = open("/tmp/first_file.txt","r")
```

Funded by the National Science Foundation, Grant #2320373

Connect to Jupyter Lab

Point your browser at https://jupyter01.classe.cornell.edu/

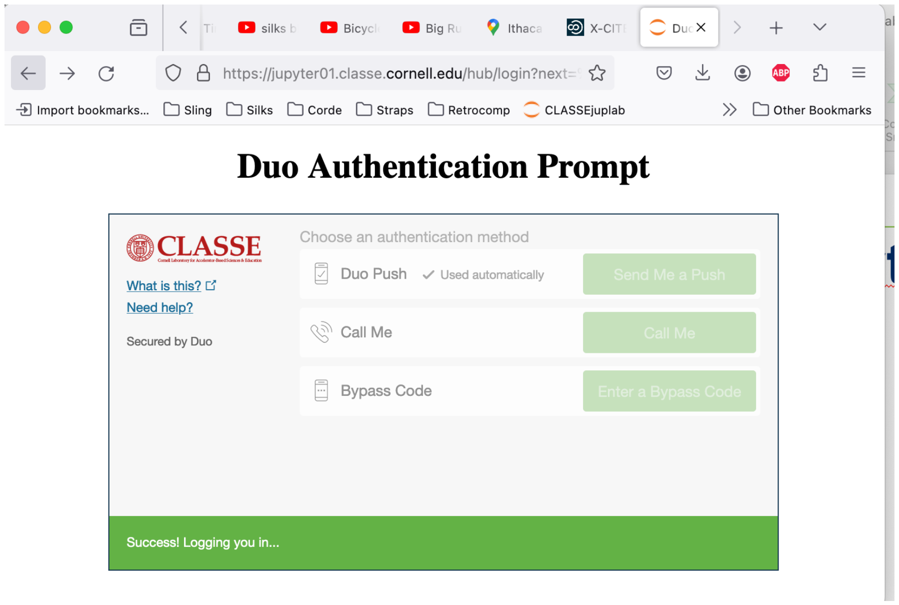




CLASSE username and password go here…

Funded by the National Science Foundation, Grant #2320373

Connect to Jupyter Lab



https://jupyter01.classe.cornell.edu/hub/login?next=

Duo Authentication Prompt

CLASSE

What is this?

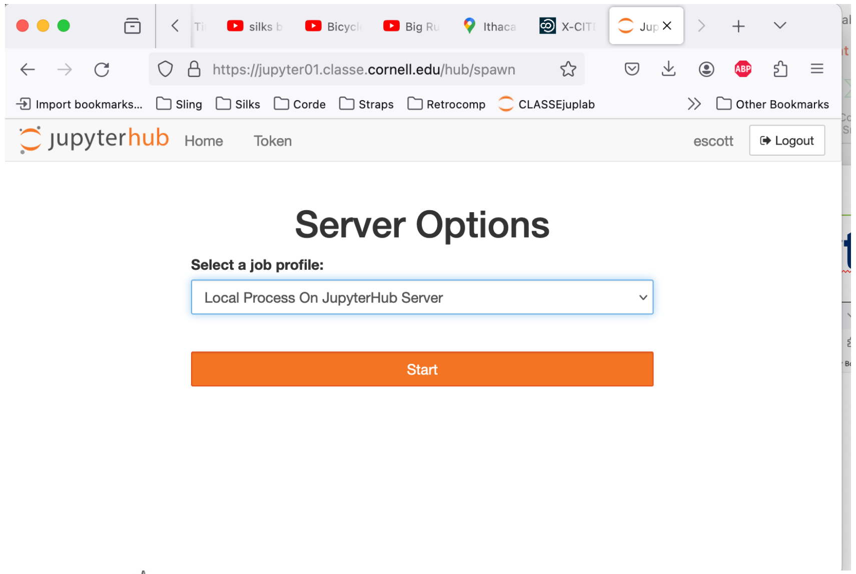
Need help?

Secured by Duo

Choose an authentication method

Duo Push ✓ Used automatically	Send Me a Push
Call Me	Call Me
Bypass Code	Enter a Bypass Code

Success! Logging you in...



https://jupyter01.classe.cornell.edu/hub/spawn

Server Options

Select a job profile: Local Process On JupyterHub Server

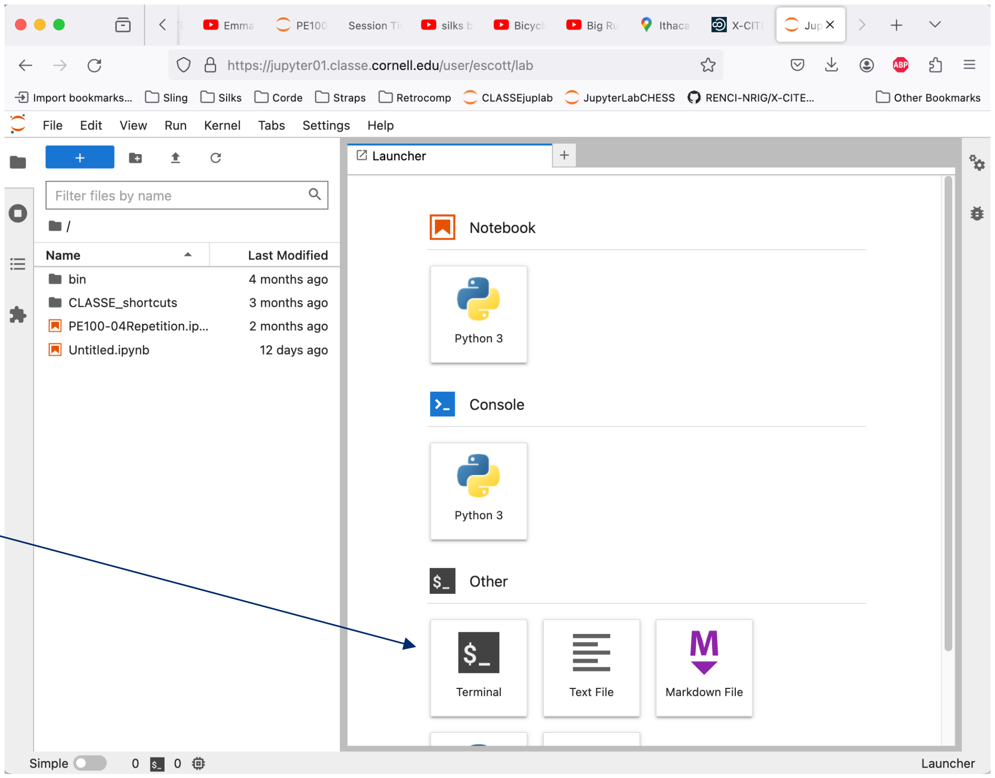
Connect to Jupyter Lab

Funded by the National Science Foundation, Grant #2320373

Copy tutorial materials

a. Launcher tab: Terminal
b. cd /nfs/chess/user/<username>
c. cp -R /nfs/chess/user/x-cite/X-CITE/ .
d. There was a space and a period at the end of that last command! ☺
e. Upper and lower case matter ☺
f. Replace <username> with your username (like "escott") ☺

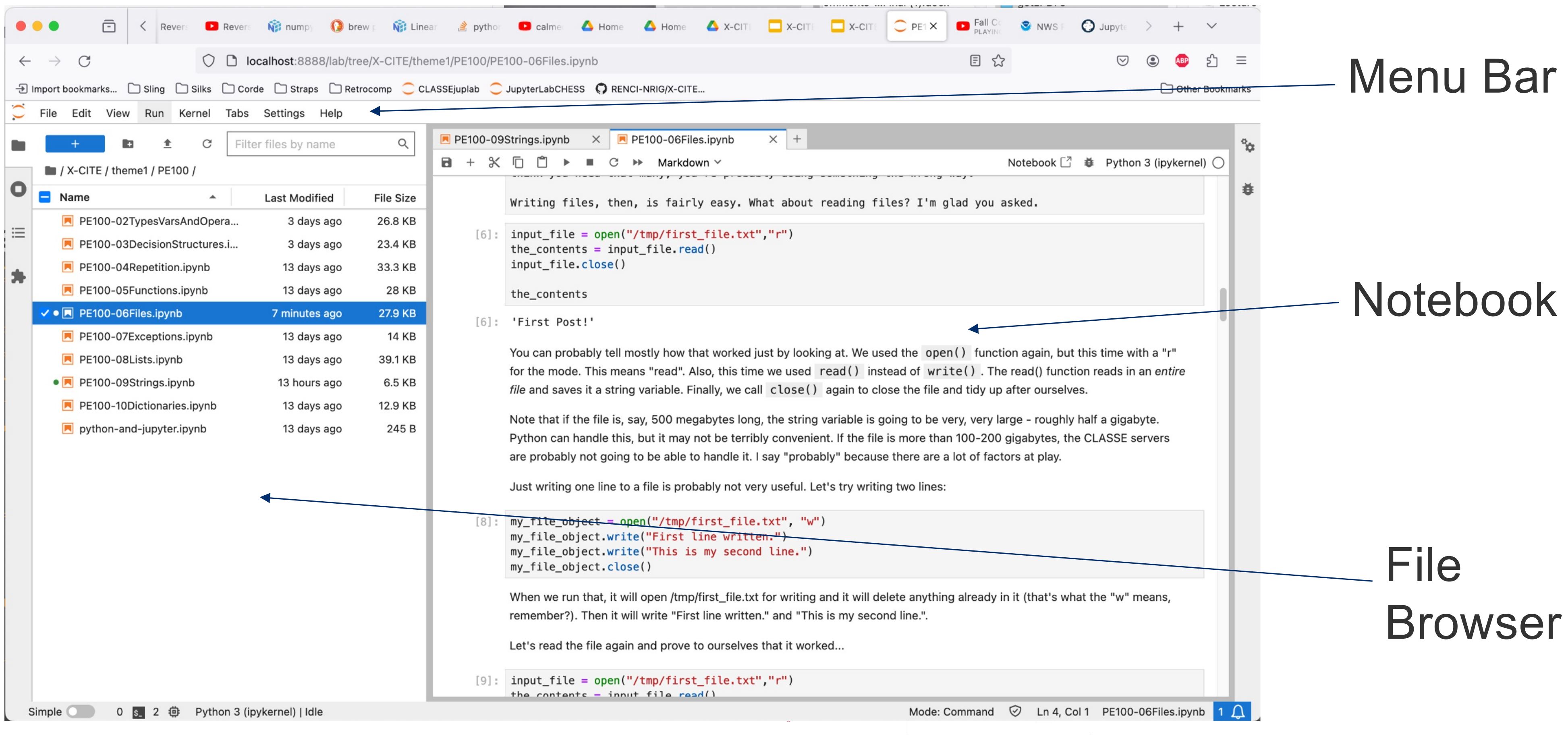
"Terminal" is right here:



Name	Last Modified
bin	4 months ago
CLASSE_shortcuts	3 months ago
PE100-04Repetition.ip...	2 months ago
Untitled.ipynb	12 days ago

Funded by the National Science Foundation, Grant #2320373

Navigating Jupyter


Menu Bar
Notebook
File Browser
File
Edit
View
Run
Kernel
Tabs
Settings
Help
localhost:8888/lab/tree/X-CITE/theme1/PE100/PE100-06Files.ipynb
Filter files by name
/ X-CITE / theme1 / PE100 /
PE100-09Strings.ipynb
PE100-06Files.ipynb
Markdown
Notebook
Python 3 (ipykernel)
Simple 0 2 Python 3 (ipykernel) | Idle
Mode: Command Ln 4, Col 1 PE100-06Files.ipynb

Name	Last Modified	File Size
PE100-02TypesVarsAndOpera...	3 days ago	26.8 KB
PE100-03DecisionStructures.i...	3 days ago	23.4 KB
PE100-04Repetition.ipynb	13 days ago	33.3 KB
PE100-05Functions.ipynb	13 days ago	28 KB
PE100-06Files.ipynb	7 minutes ago	27.9 KB
PE100-07Exceptions.ipynb	13 days ago	14 KB
PE100-08Lists.ipynb	13 days ago	39.1 KB
PE100-09Strings.ipynb	13 hours ago	6.5 KB
PE100-10Dictionaries.ipynb	13 days ago	12.9 KB
python-and-jupyter.ipynb	13 days ago	245 B

```
Writing files, then, is fairly easy. What about reading files? I'm glad you asked.
```

```
[6]: input_file = open("/tmp/first_file.txt","r")
     the_contents = input_file.read()
     input_file.close()

     the_contents
```

```
[6]: 'First Post!'
```

You can probably tell mostly how that worked just by looking at. We used the `open()` function again, but this time with a "r" for the mode. This means "read". Also, this time we used `read()` instead of `write()`. The read() function reads in an *entire file* and saves it a string variable. Finally, we call `close()` again to close the file and tidy up after ourselves.

Note that if the file is, say, 500 megabytes long, the string variable is going to be very, very large - roughly half a gigabyte. Python can handle this, but it may not be terribly convenient. If the file is more than 100-200 gigabytes, the CLASSE servers are probably not going to be able to handle it. I say "probably" because there are a lot of factors at play.

Just writing one line to a file is probably not very useful. Let's try writing two lines:

```
[8]: my_file_object = open("/tmp/first_file.txt", "w")
     my_file_object.write("First line written.")
     my_file_object.write("This is my second line.")
     my_file_object.close()
```

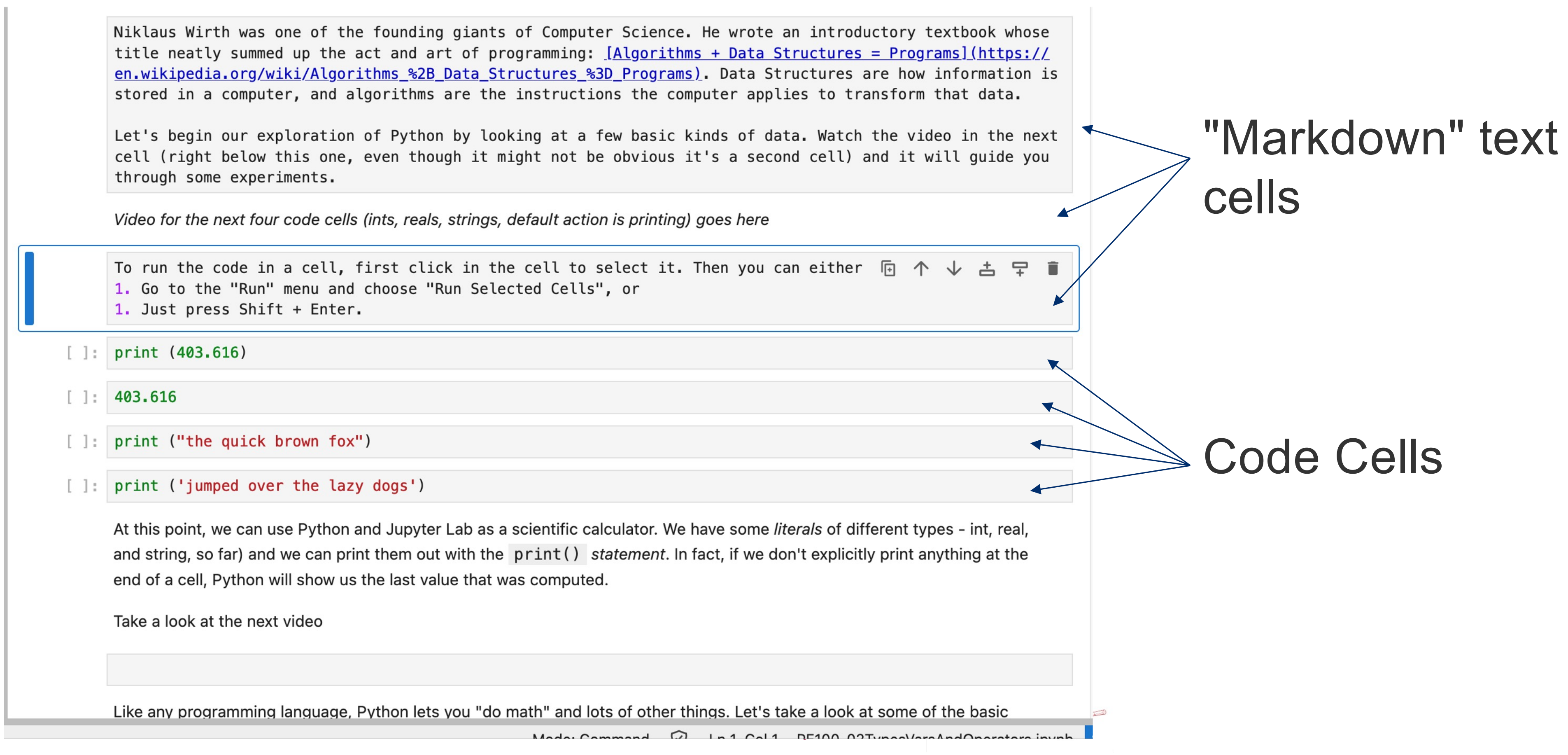
When we run that, it will open /tmp/first_file.txt for writing and it will delete anything already in it (that's what the "w" means, remember?). Then it will write "First line written." and "This is my second line.".

Let's read the file again and prove to ourselves that it worked...

```
[9]: input_file = open("/tmp/first_file.txt","r")
```

Funded by the National Science Foundation, Grant #2320373

The Notebook



```
Niklaus Wirth was one of the founding giants of Computer Science. He wrote an introductory textbook whose
title neatly summed up the act and art of programming: [Algorithms + Data Structures = Programs](https://
en.wikipedia.org/wiki/Algorithms_%2B_Data_Structures_%3D_Programs). Data Structures are how information is
stored in a computer, and algorithms are the instructions the computer applies to transform that data.

Let's begin our exploration of Python by looking at a few basic kinds of data. Watch the video in the next
cell (right below this one, even though it might not be obvious it's a second cell) and it will guide you
through some experiments.
```

Video for the next four code cells (ints, reals, strings, default action is printing) goes here

```
To run the code in a cell, first click in the cell to select it. Then you can either
1. Go to the "Run" menu and choose "Run Selected Cells", or
1. Just press Shift + Enter.
```

```
[ ]: print (403.616)
```

```
[ ]: 403.616
```

```
[ ]: print ("the quick brown fox")
```

```
[ ]: print ('jumped over the lazy dogs')
```

At this point, we can use Python and Jupyter Lab as a scientific calculator. We have some *literals* of different types - int, real, and string, so far) and we can print them out with the `print()` *statement*. In fact, if we don't explicitly print anything at the end of a cell, Python will show us the last value that was computed.

Take a look at the next video

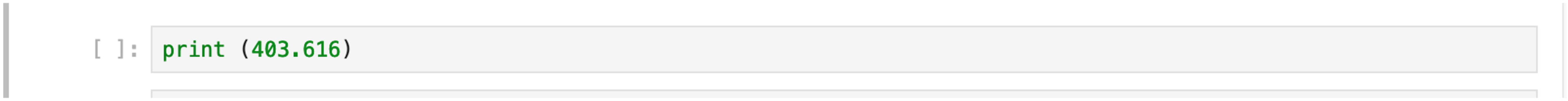
Like any programming language, Python lets you "do math" and lots of other things. Let's take a look at some of the basic

Mode: Command Ln 1, Col 1 PE100-02TypesVarsAndOperators.ipynb

Funded by the National Science Foundation, Grant #2320373

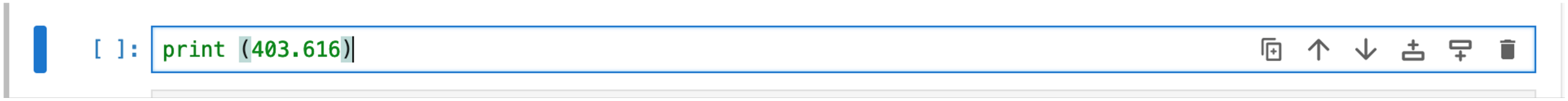
Running Code

A block of code (one line, in this case) ready to be run…



```
[ ]: print (403.616)
```

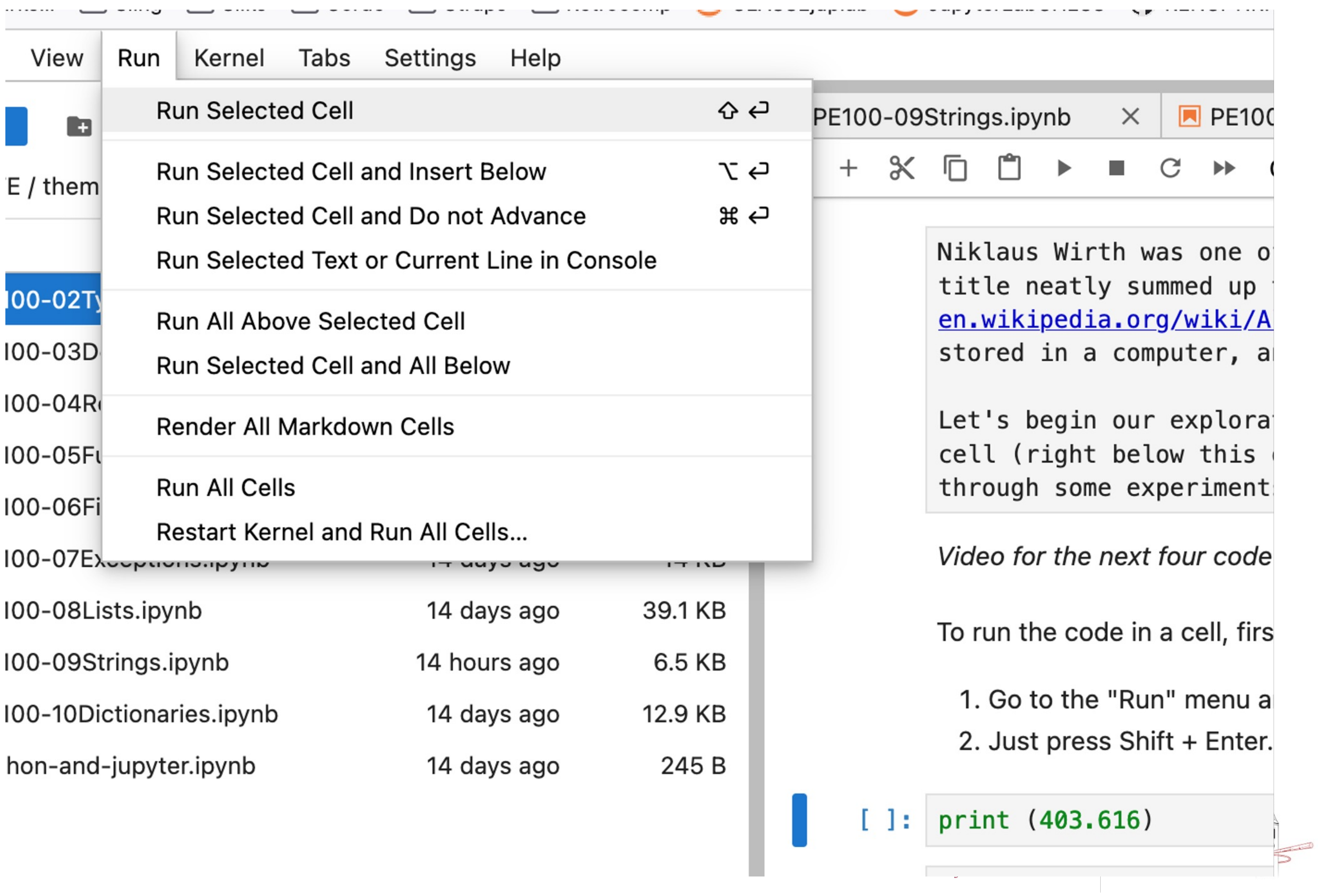
Click in it to select that cell (notice the blue bar to the left - marks the current cell.



```
[ ]: print (403.616)
```

Funded by the National Science Foundation, Grant #2320373

Running code


View
Run
Kernel
Tabs
Settings
Help
Run Selected Cell ⇧ ↩
Run Selected Cell and Insert Below ⌥ ↩
Run Selected Cell and Do not Advance ⌘ ↩
Run Selected Text or Current Line in Console
Run All Above Selected Cell
Run Selected Cell and All Below
Render All Markdown Cells
Run All Cells
Restart Kernel and Run All Cells...
PE100-09Strings.ipynb
PE100

E / them

100-02Ty		
100-03D		
100-04R		
100-05Fu		
100-06Fi		
100-07Exceptions.ipynb	14 days ago	14 KB
100-08Lists.ipynb	14 days ago	39.1 KB
100-09Strings.ipynb	14 hours ago	6.5 KB
100-10Dictionaries.ipynb	14 days ago	12.9 KB
hon-and-jupyter.ipynb	14 days ago	245 B

```
Niklaus Wirth was one o
title neatly summed up
en.wikipedia.org/wiki/A
stored in a computer, a

Let's begin our explora
cell (right below this
through some experiment
```

Video for the next four code

To run the code in a cell, firs

1. Go to the "Run" menu a
2. Just press Shift + Enter.

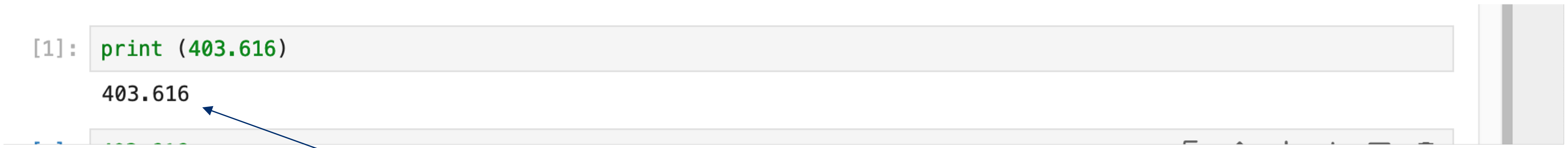
```
[ ]: print (403.616)
```

One way: "Run Selected Cell" command on the Run menu.

Alternative: notice the keyboard accelerator shown on the right edge of the menu command? Shift+Enter

Funded by the National Science Foundation, Grant #2320373

Output



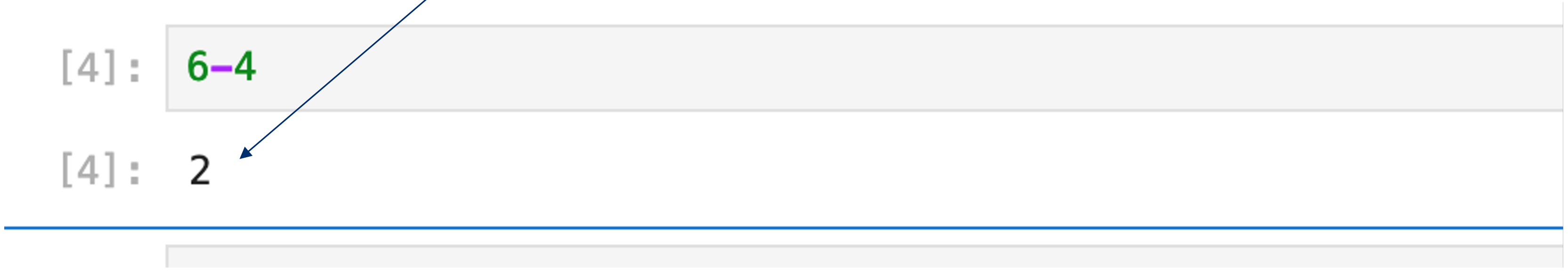
```
[1]: print (403.616)
403.616
```

Output is shown below the code we just ran.
Strictly speaking it's one cell with two parts…

Funded by the National Science Foundation, Grant #2320373

What gets displayed

1. Any print() statements where we explicitly say what to print out - precise control.
2. Any error messages are printed - by default with a red background!
3. If the last line of code (or they only line of code, if there's only one) computes a value, it will be printed.



```
[4]: 6-4

[4]: 2
```

Funded by the National Science Foundation, Grant #2320373

Moving on… to Python!

That's enough of the rudiments of Jupyter to get you started. Now let's start writing a little bit of Python.

Funded by the National Science Foundation, Grant #2320373

Data Types

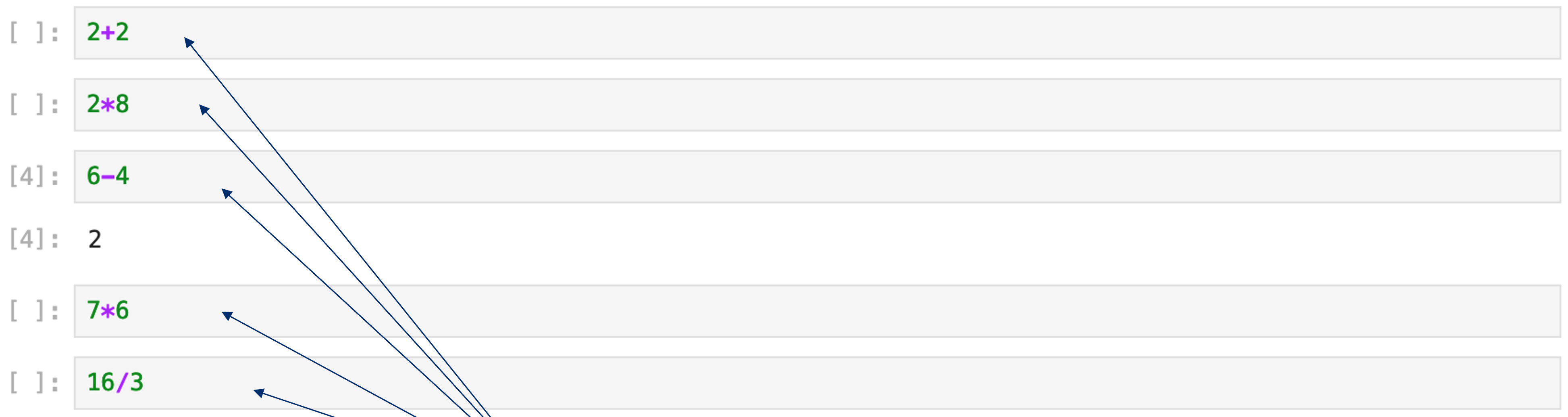
Python can only handle a few kinds of data, but they can combine to represent anything.

Integers (`int`) - arbitrarily huge, positive or negative
Real Numbers (`float`) - "double precision"
Strings (`string`)

There are also lists, sets, and objects in the full tutorial.

Basic Arithmetic

Funded by the National Science Foundation, Grant #2320373



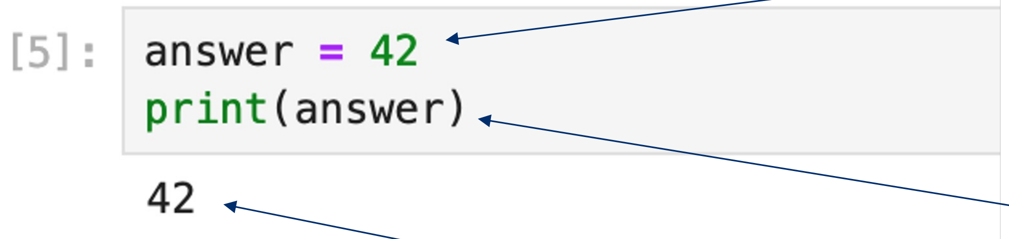
```
[ ]: 2+2

[ ]: 2*8

[4]: 6-4

[4]: 2

[ ]: 7*6

[ ]: 16/3
```

Click in each code cell, shift+enter run run it.

Funded by the National Science Foundation, Grant #2320373

Storing information

Data is stored in "variables", which are just places to put arbitrary data and that have names.



```
[5]: answer = 42
     print(answer)

     42
```

Create a variable with the name "answer". Set it equal to 42.

Print the contents of "answer".

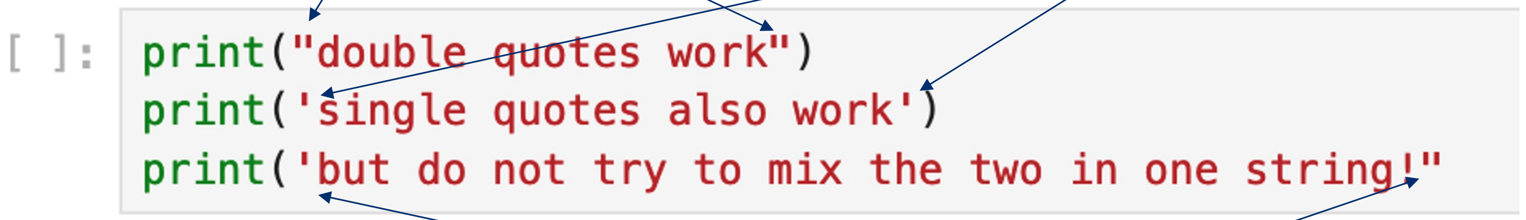
The output.

Funded by the National Science Foundation, Grant #2320373

Strings

Surround strings with double quotes…

…or single quotes…



```
[ ]: print("double quotes work")
     print('single quotes also work')
     print('but do not try to mix the two in one string!")
```

But when you start with one, you have to end with that same one.

Funded by the National Science Foundation, Grant #2320373

Decision and Control Flow Structures

Like the languages you're used to, Python offers:

- **if** statements
- **while** loops
- **for** loops - unusually powerful in Python's case

Unlike what you're probably familiar with, the syntax takes some getting used to.

Funded by the National Science Foundation, Grant #2320373

Python and Indentation

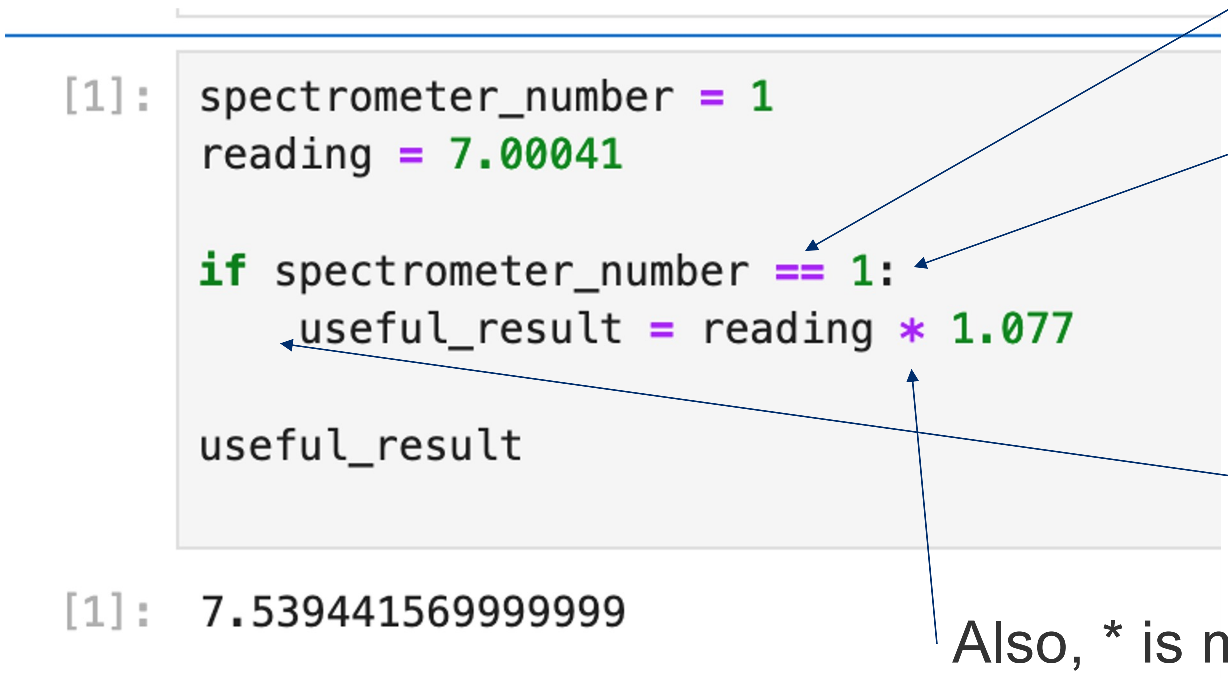
Have you ever sat down and made an outline for a document? A thesis or dissertation, perhaps?

Remember how "more indentation" meant "finer detail in a more local scope, less globally important?"

Good. Keep that in mind.

Funded by the National Science Foundation, Grant #2320373

if statements



```
[1]: spectrometer_number = 1
     reading = 7.00041

     if spectrometer_number == 1:
         useful_result = reading * 1.077

     useful_result

[1]: 7.539441569999999
```

Test for equals

The colon!

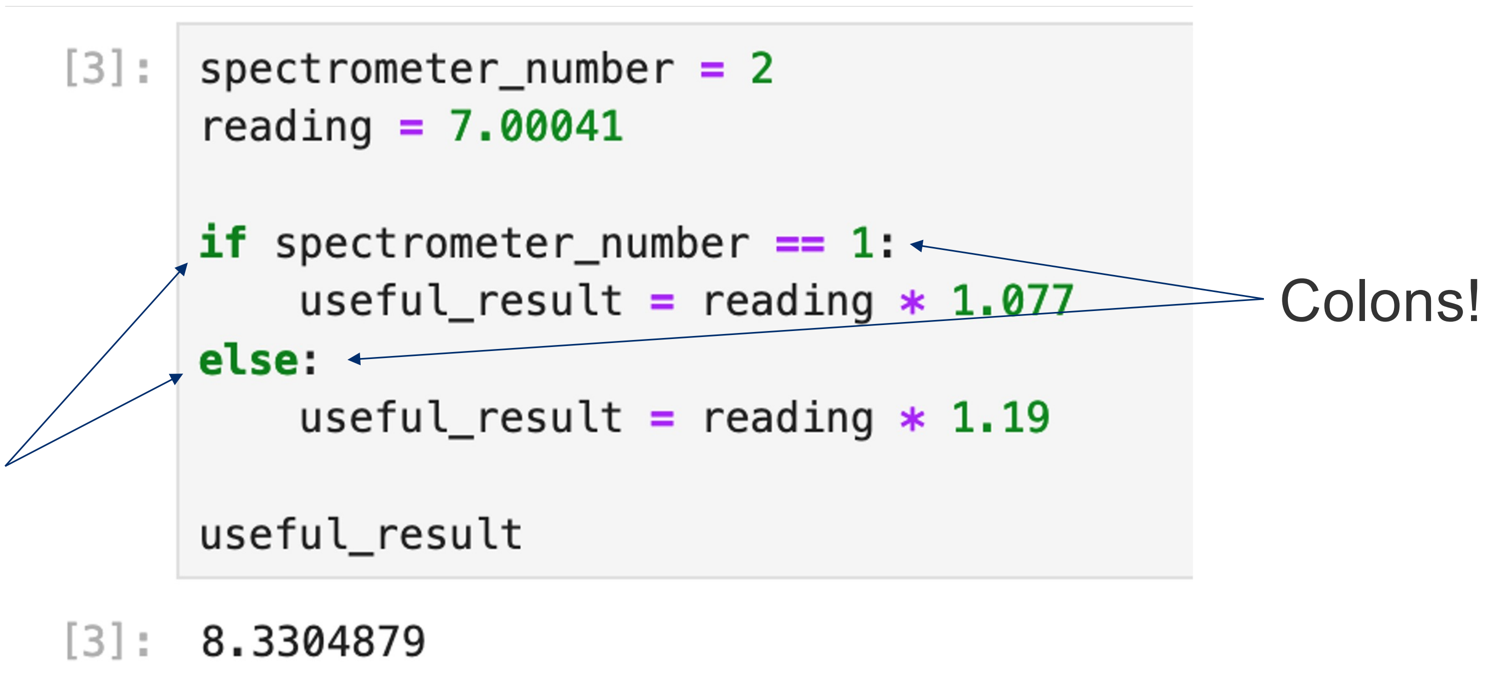
Indentation has meaning

Also, * is multiplication.

Funded by the National Science Foundation, Grant #2320373

if-else

if and **else** at same level of indentation as the rest of the block it's in.

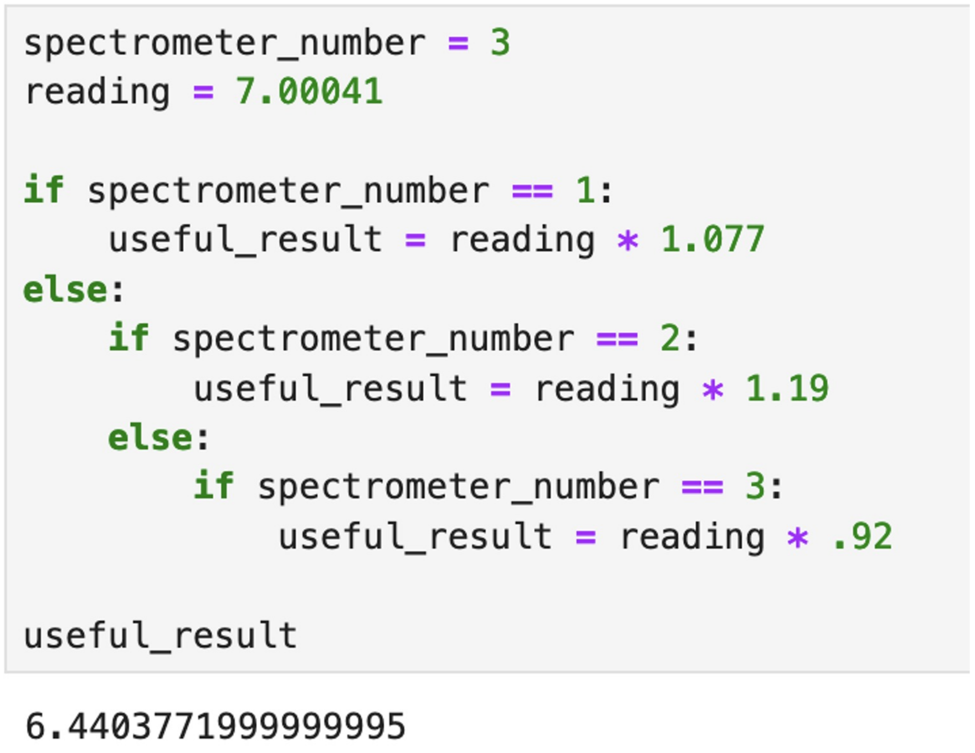


```
[3]: spectrometer_number = 2
     reading = 7.00041

     if spectrometer_number == 1:
         useful_result = reading * 1.077
     else:
         useful_result = reading * 1.19

     useful_result

[3]: 8.3304879
```

Colons!

Funded by the National Science Foundation, Grant #2320373

Several Choices - Brute Force Soln.?



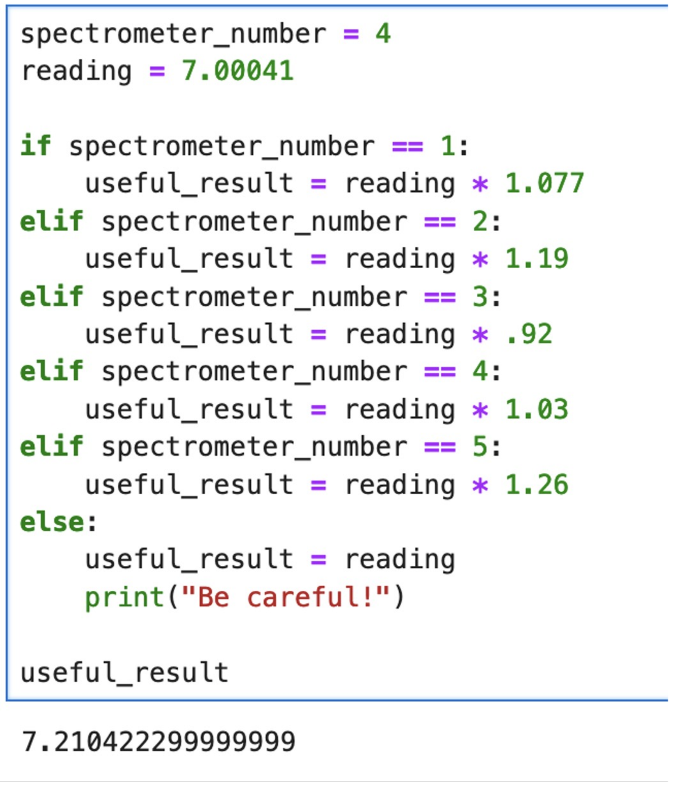
```
spectrometer_number = 3
reading = 7.00041

if spectrometer_number == 1:
    useful_result = reading * 1.077
else:
    if spectrometer_number == 2:
        useful_result = reading * 1.19
    else:
        if spectrometer_number == 3:
            useful_result = reading * .92

useful_result
```

```
6.4403771999999995
```

One way to handle multiple options: keep nesting more if-else statements as deep as needed.

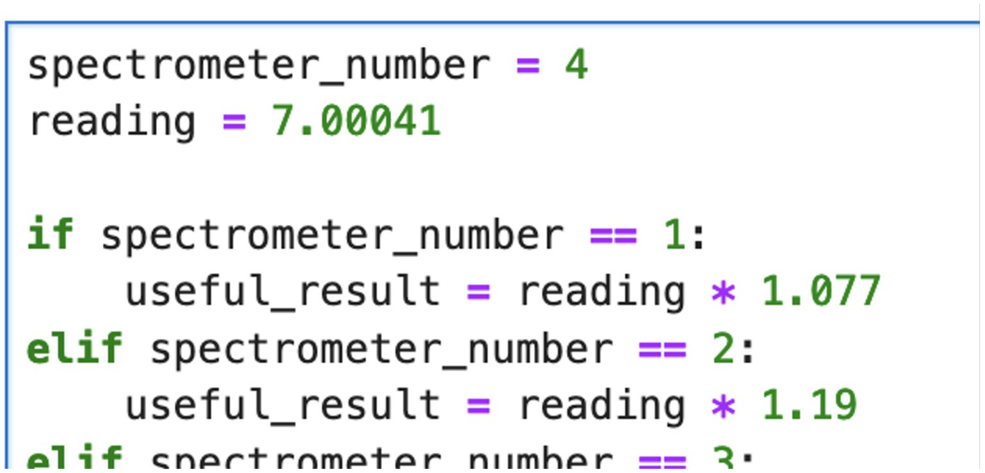
Funded by the National Science Foundation, Grant #2320373

elif - much more readable!



```
spectrometer_number = 4
reading = 7.00041

if spectrometer_number == 1:
    useful_result = reading * 1.077
elif spectrometer_number == 2:
    useful_result = reading * 1.19
elif spectrometer_number == 3:
    useful_result = reading * .92
elif spectrometer_number == 4:
    useful_result = reading * 1.03
elif spectrometer_number == 5:
    useful_result = reading * 1.26
else:
    useful_result = reading
    print("Be careful!")

useful_result
```

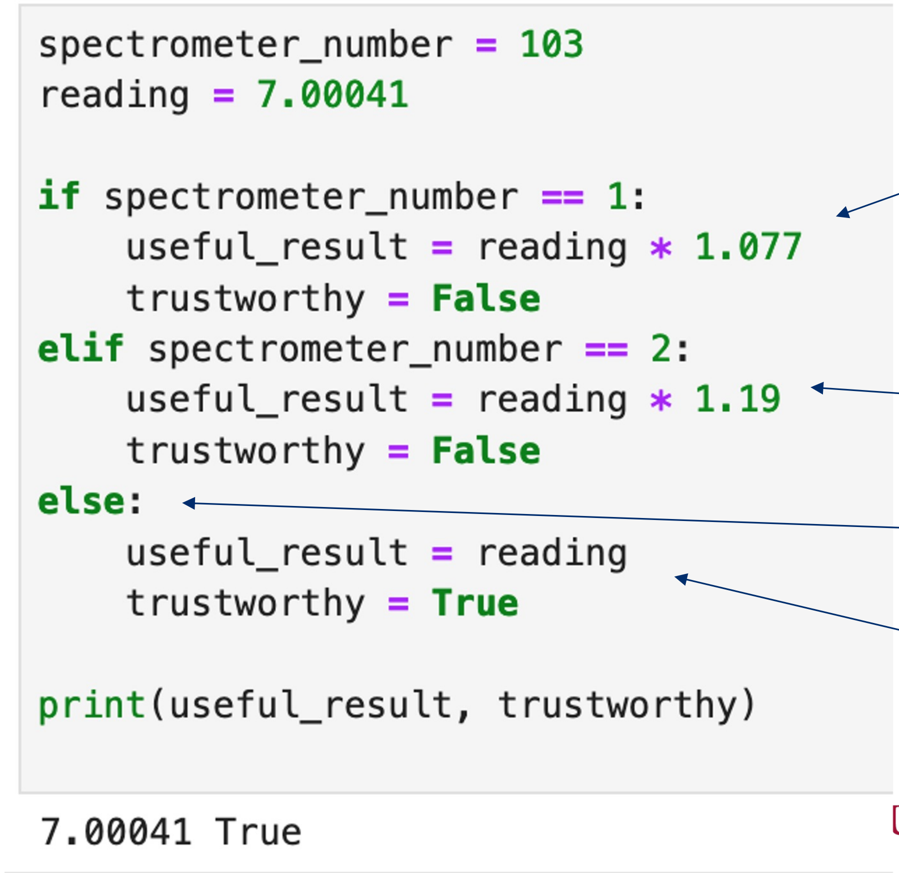
```
7.210422299999999
```



```
spectrometer_number = 4
reading = 7.00041

if spectrometer_number == 1:
    useful_result = reading * 1.077
elif spectrometer_number == 2:
    useful_result = reading * 1.19
elif spectrometer_number == 3:
```

Funded by the National Science Foundation, Grant #2320373

if - elif - else



```
spectrometer_number = 103
reading = 7.00041

if spectrometer_number == 1:
    useful_result = reading * 1.077
    trustworthy = False
elif spectrometer_number == 2:
    useful_result = reading * 1.19
    trustworthy = False
else:
    useful_result = reading
    trustworthy = True

print(useful_result, trustworthy)
```

```
7.00041 True
```

If ***and only if*** it's 1

If ***and only if*** it's 2

"else" ***never*** has a conditional test

Otherwise, any other number

Repetition

Funded by the National Science Foundation, Grant #2320373

Sometimes just making a straight-through control flow decision isn't good enough.

We need to run a piece of code repeatedly.

Sometimes we know well in advance how many times, sometimes we don't.

Funded by the National Science Foundation, Grant #2320373

"while" loops

When we don't know in advance how many times we need to run a piece of code, we use a "while" loop.

By "not knowing in advance", we mean "we don't know how many times to run it until we start the loop - we'll get on into the loop and decide at the last possible moment".

Funded by the National Science Foundation, Grant #2320373

What while loops are…

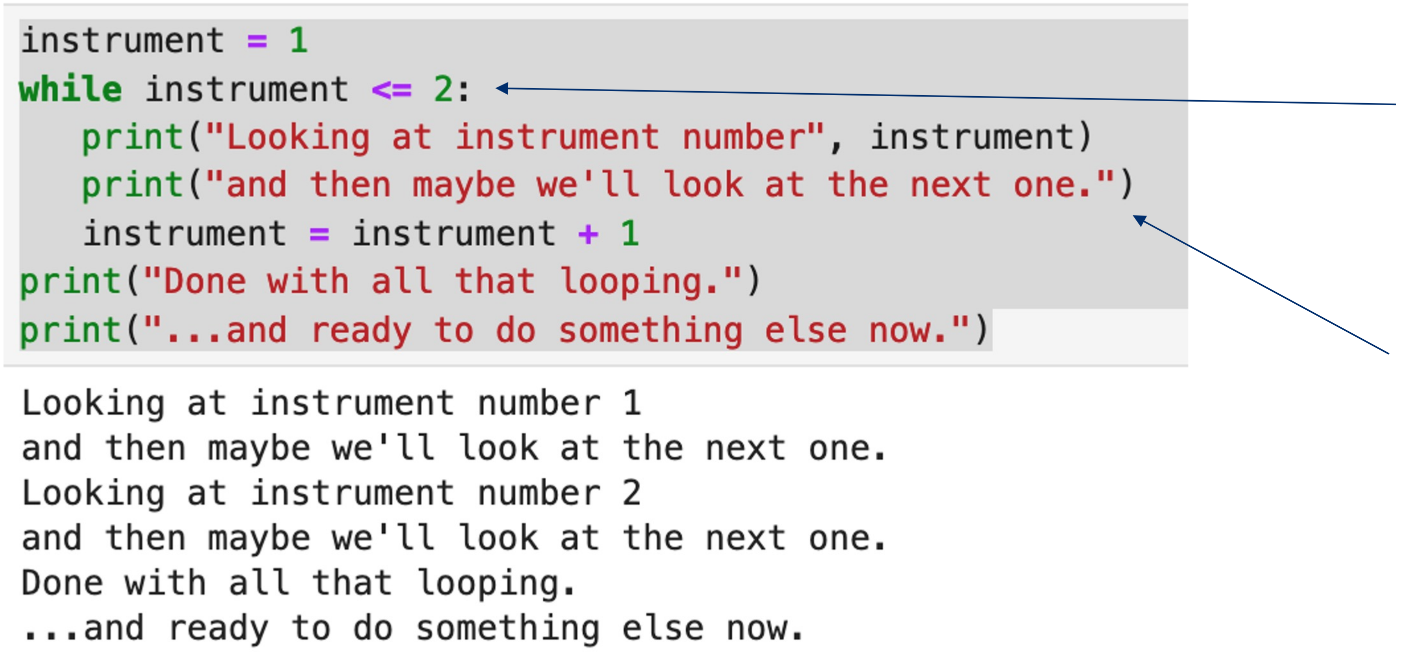
A while loop first tests the conditional statement.

If it's false, the while loop is finished and execution moves on to the next line after the entire loop. None of the while loop's code block is executed.

If the test was true, on the other hand, the indented code block is executed. When that block is done running, control goes back up to testing the conditional statement again and it all repeats.

Funded by the National Science Foundation, Grant #2320373

A while loop



```python
instrument = 1
while instrument <= 2:
   print("Looking at instrument number", instrument)
   print("and then maybe we'll look at the next one.")
   instrument = instrument + 1
print("Done with all that looping.")
print("...and ready to do something else now.")
```

```
Looking at instrument number 1
and then maybe we'll look at the next one.
Looking at instrument number 2
and then maybe we'll look at the next one.
Done with all that looping.
...and ready to do something else now.
```

While loop's conditional statement

Body of the loop

Funded by the National Science Foundation, Grant #2320373

Input/Output (I/O) and while loops

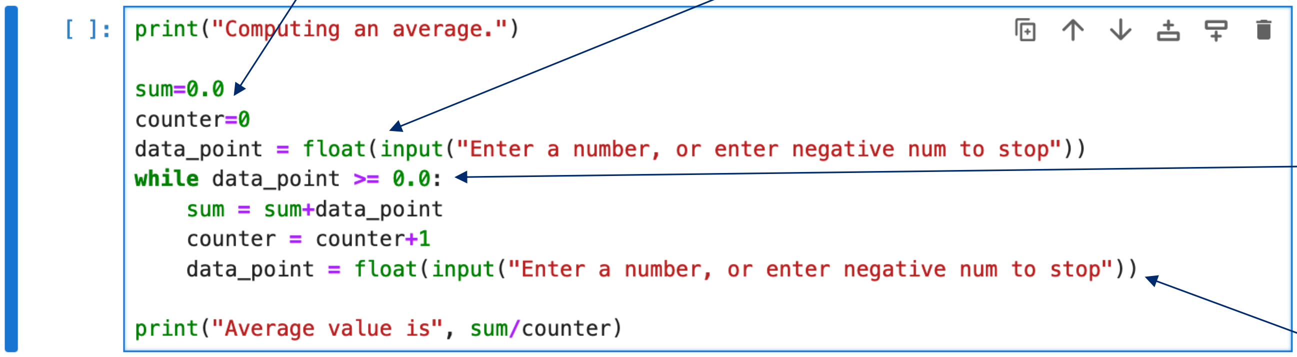
Average an unknown number of readings, reading numbers until one of them is negative…

get ready.

input() always returns a string, never a number, so use typecasting (type coercion)

stop when we finally see a negative

get the next data point – otherwise it will loop until the heat death of the universe

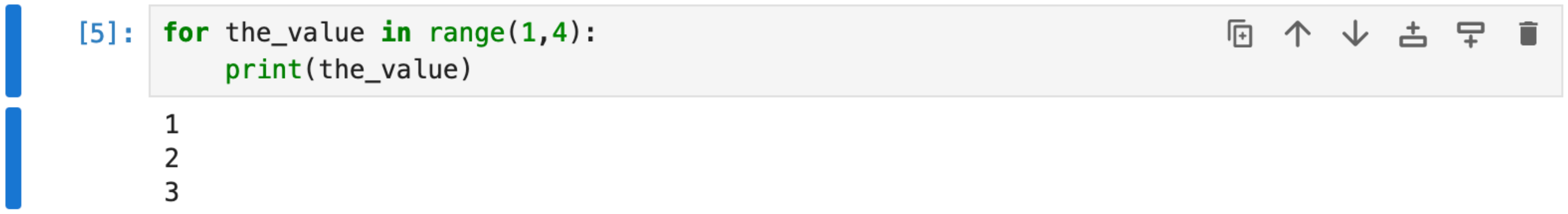


```
[ ]: print("Computing an average.")

     sum=0.0
     counter=0
     data_point = float(input("Enter a number, or enter negative num to stop"))
     while data_point >= 0.0:
         sum = sum+data_point
         counter = counter+1
         data_point = float(input("Enter a number, or enter negative num to stop"))

     print("Average value is", sum/counter)
```

Funded by the National Science Foundation, Grant #2320373

"for" loops

When you know how many times your loop will run, before it starts running, then you can use a for loop.



```
[5]: for the_value in range(1,4):
         print(the_value)

     1
     2
     3
```

Funded by the National Science Foundation, Grant #2320373

How for loops work

Unlike most languages, the for loop in python is quite versatile. It doesn't just take starting and ending numbers and blast through them.

The syntax is a giveaway:

`for` *variable* **`in`** *something-like-a-set*:

Note: boldface, huge font… This is the important part!

Funded by the National Science Foundation, Grant #2320373

What's up with this range() stuff?

Now the odd bit with range() makes sense.[citation needed]

range() creates a set of integers beginning at the starting value and continuing until it detects it was going to make a number that would go beyond the ending value.

Then the for loop just runs the code block for every value in the set.

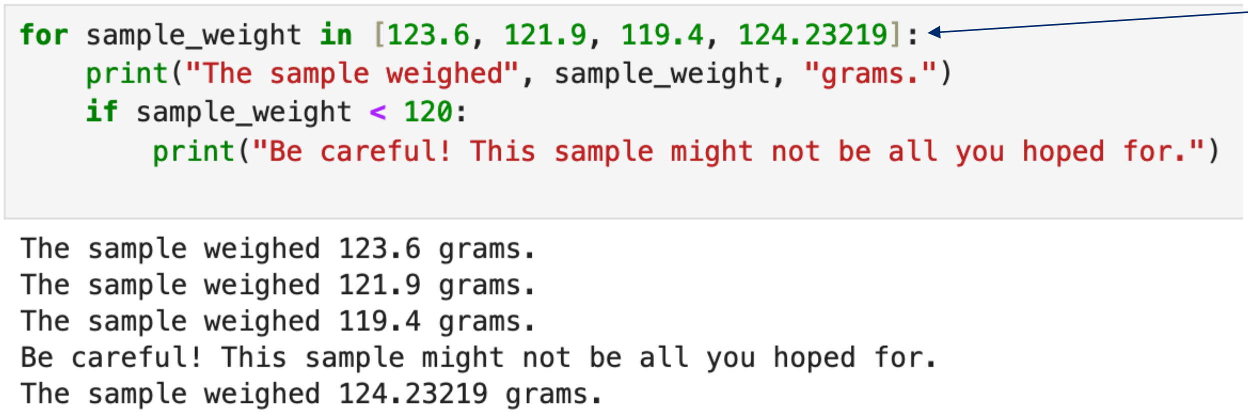
Funded by the National Science Foundation, Grant #2320373

OK, yes, little lies

1. Don't worry, range() doesn't really create the whole set at once. It dribbles the numbers out one at a time as they're used. It's reasonably efficient.
2. They aren't really sets, they're iterables. Sets don't allow duplicates, iterables are just things that have a first thing and a way to either read the next thing or tell you there isn't a next thing.

Funded by the National Science Foundation, Grant #2320373

Iterables: make "for" loops versatile again



```python
for sample_weight in [123.6, 121.9, 119.4, 124.23219]:
    print("The sample weighed", sample_weight, "grams.")
    if sample_weight < 120:
        print("Be careful! This sample might not be all you hoped for.")
```

```
The sample weighed 123.6 grams.
The sample weighed 121.9 grams.
The sample weighed 119.4 grams.
Be careful! This sample might not be all you hoped for.
The sample weighed 124.23219 grams.
```

For instance, "lists" are iterables.

The lines in a file are iterables, too!

Funded by the National Science Foundation, Grant #2320373

Cool range() tricks

One argument? Start at **0**, the argument is the ending value.

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

Two args? Start and End.

```
for i in range(7,10):
    print(i)
```

```
7
8
9
```

3? Start, end, and step size

```
for i in range (12,20,3):
    print(i)
```

```
12
15
18
```

Step size can be negative:

```
for i in range(6, -3, -2):
    print(i)
```

```
6
4
2
0
-2
```

Protip: Always check your starting and ending values. It's very easy to be "off by one".

Trust me… this will eventually get you.

Funded by the National Science Foundation, Grant #2320373

Nested Loops

Like "if" statements, "for" and "while" loops can be nested.
Traversing multidimensional data? Complex workflow?

```python
for x in range(3):
    for y in range(2):
        print("x=",x," y=", y)
```

```
x= 0  y= 0
x= 0  y= 1
x= 1  y= 0
x= 1  y= 1
x= 2  y= 0
x= 2  y= 1
```

Funded by the National Science Foundation, Grant #2320373

If, while, and for: very closely related

If, while, and for statements all have:

1. A test that evaluates to True or False
2. A code block that executes if the test returns True

If statement: run code block once if true
While: keep frantically running that code block and checking the conditional test as long as the test is True
For: the test is "can I get a remaining item from that iterable?"

Funded by the National Science Foundation, Grant #2320373

Functions()

What is a function?

Pre-calculus flashback: it's a "black box" that takes one or more inputs and produces precisely one output.

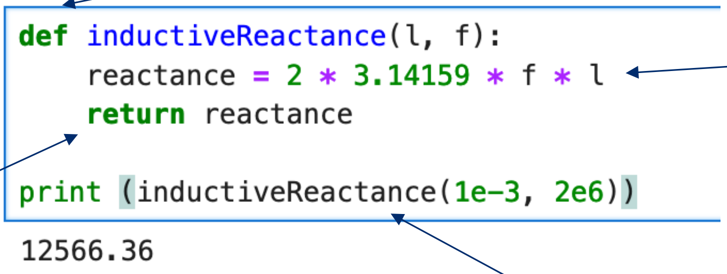
- sin(x)
- bessel(v,z)

Computer Science: a lump of code that takes one or more inputs and produces ("returns") precisely zero or one output.

Funded by the National Science Foundation, Grant #2320373

Create your own functions

"def" means "define a function"



```
def inductiveReactance(l, f):
    reactance = 2 * 3.14159 * f * l
    return reactance

print (inductiveReactance(1e-3, 2e6))
```

```
12566.36
```

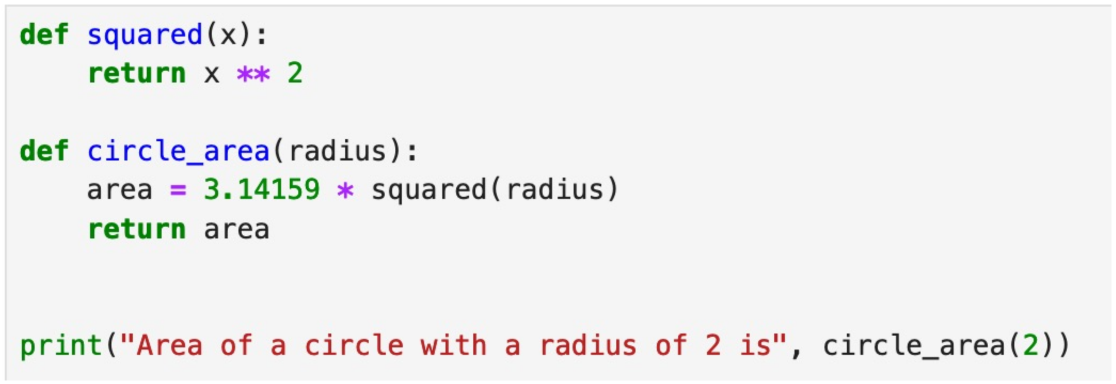
indented code block

the function computed a value, so we can exit the function and produce ("return") this value

function values can be the argument to other functions. Remember f(g(x)) ("chain rule").

Funded by the National Science Foundation, Grant #2320373

Functions calling functions calling…



```
def squared(x):
    return x ** 2

def circle_area(radius):
    area = 3.14159 * squared(radius)
    return area


print("Area of a circle with a radius of 2 is", circle_area(2))
```

Funded by the National Science Foundation, Grant #2320373

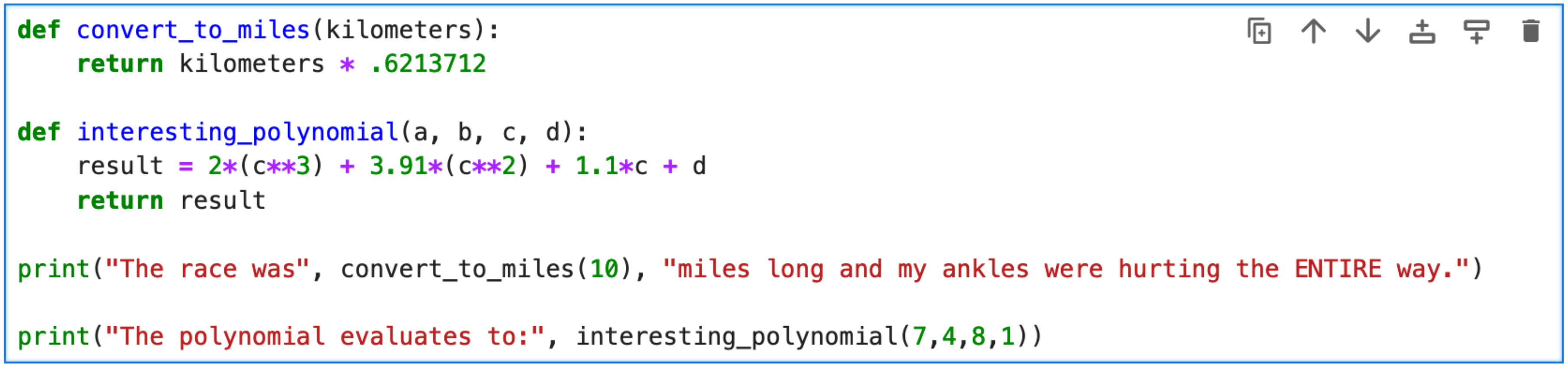
Function and Variable naming

- No *keywords* (e.g., False is invalid)
- No spaces (e.g., my function is invalid)
- The first character must be:
  - a-z, A-Z, or _ (the underscore character)
  - No numbers (e.g., 1st_function is invalid)
- After the first character, the following are allowed:
  - a-z, A-Z, _, and 0-9
  - No other symbols (e.g., get_voltage&current is invalid)

renci USC Viterbi School of Engineering Information Sciences Institute CHESS CORNELL HIGH ENERGY SYNCHROTRON SOURCE

Return

Funded by the National Science Foundation, Grant #2320373



```
def convert_to_miles(kilometers):
    return kilometers * .6213712

def interesting_polynomial(a, b, c, d):
    result = 2*(c**3) + 3.91*(c**2) + 1.1*c + d
    return result

print("The race was", convert_to_miles(10), "miles long and my ankles were hurting the ENTIRE way.")

print("The polynomial evaluates to:", interesting_polynomial(7,4,8,1))
```

Funded by the National Science Foundation, Grant #2320373

Brain full yet?

Thank You!

Funded by the National Science Foundation, Grant #2320373