Zero-to-Python

July, 2025

Table of Contents

# About this Course

This course serves as first introduction to Python by introducing how to write scripts using a text editor and running them at the UNIX command line. By the end of the exercises you should have the tools you need to follow and full fledged Python tutorial.



### Learning Goals

* Practice a writing script, running it, updating script, running it, creating new script
* Parse text files line by line
* Appreciate how Unix commands are just files/programs/scripts

# 1 Introduction

Why Python? Python is a common introductory language in computer science departments due to its relative ease of learning, versatility for many applications, and overall popularity. It is great general purpose language and complements more specialized languages like R. In this course we won’t do a deep dive into Python (for that follow a full fledged Python tutorial like the one described in Ch 9. Next Steps) but rather we’ll cover:

* Practical skills for writing a Python script and running it at the UNIX command line
* Taste of how Python works and how it can be used to parse through text files line by line
* Understand that UNIX commands can, and often are, Python scripts

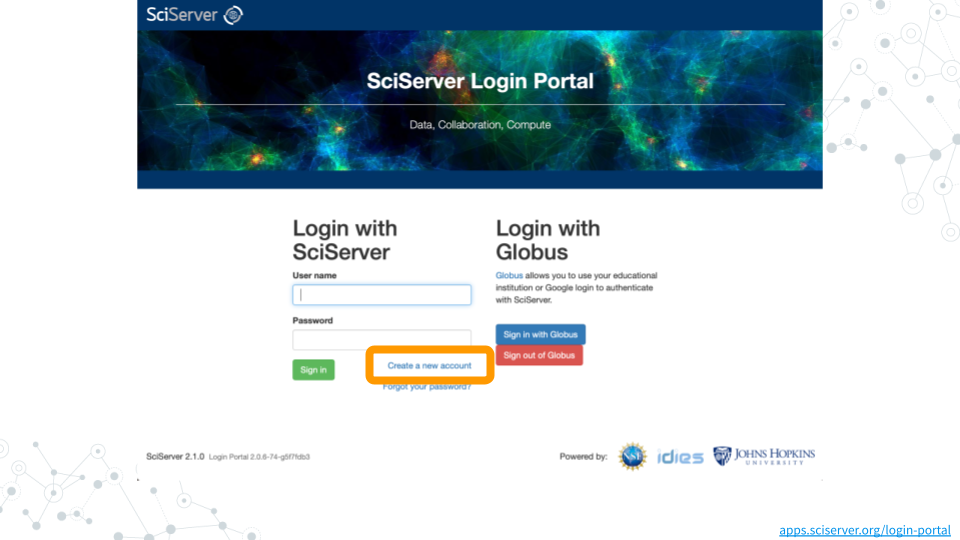
# 2 Set Up Compute

SciServer is an online platform for doing scientific data analysis. It is used by scientists studying astronomy, biology, oceanography, and more, and is free as long as you are using it for scientific research.  
Using SciServer means you do not need a fancy computer or need to install any special programs on your computer, you can just log in with your internet browser to start doing research.

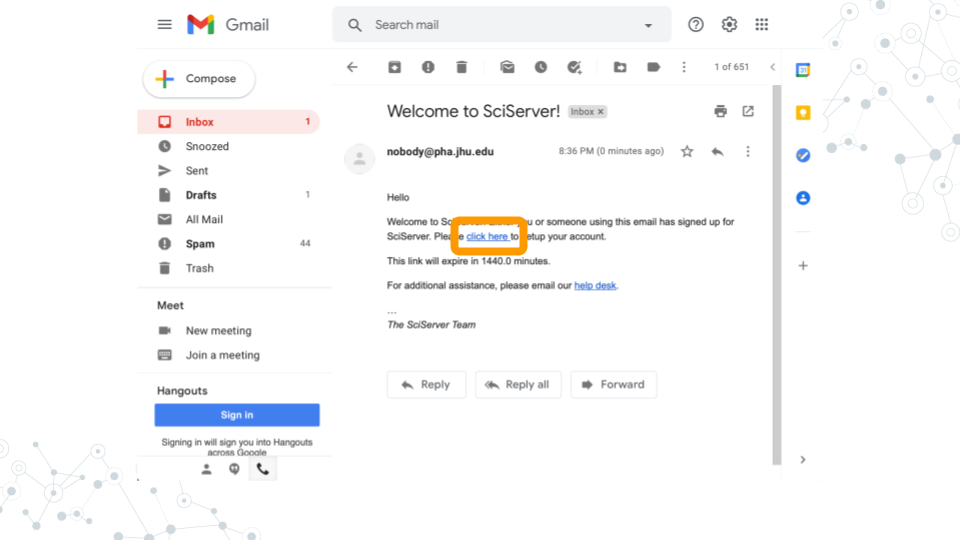
We will use the Jupyter development environment provided in the SciServer Essentials 2.0 image. This web-based environment provides all the tools we need for this course: a text editor and a UNIX environment to run Python scripts.

## 2.1 Join SciServer

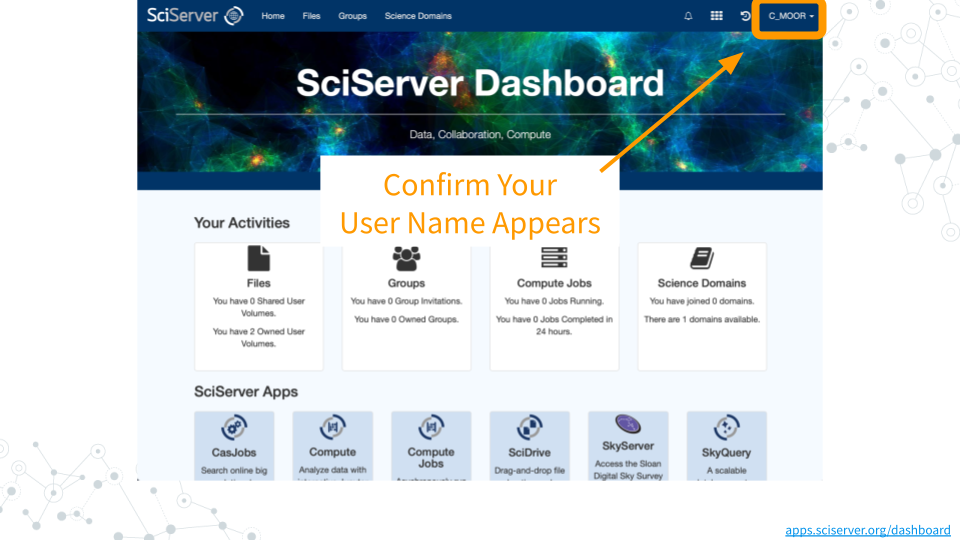
The first step is to create an account on SciServer. To do this, open <https://sciserver.org> in a web browser and click “Login to SciServer”. After clicking on “Create a new account”, fill out the form with a username, email, and password. Note that you cannot change your username later.



Check your email and click on the verification link. This will log you into SciServer.

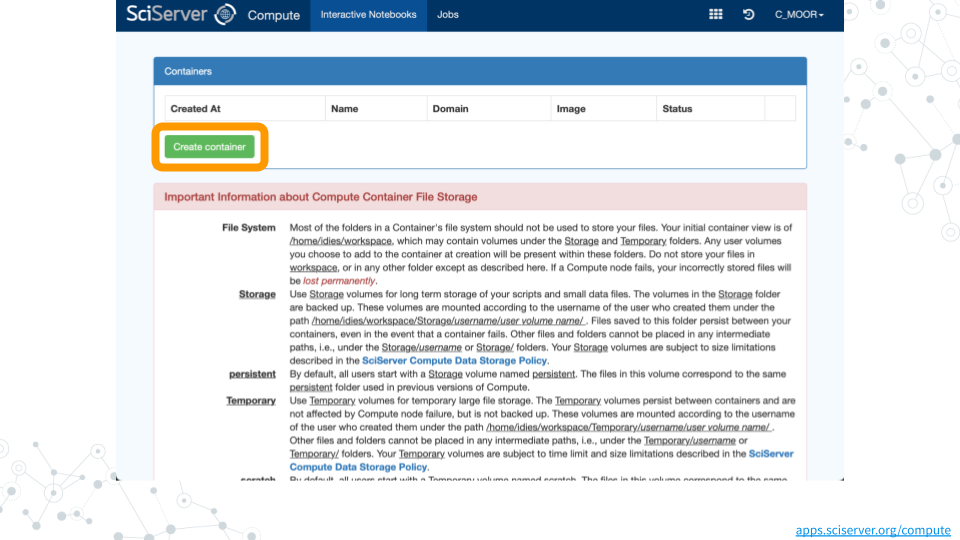


Confirm that your username appears in the upper right hand corner.

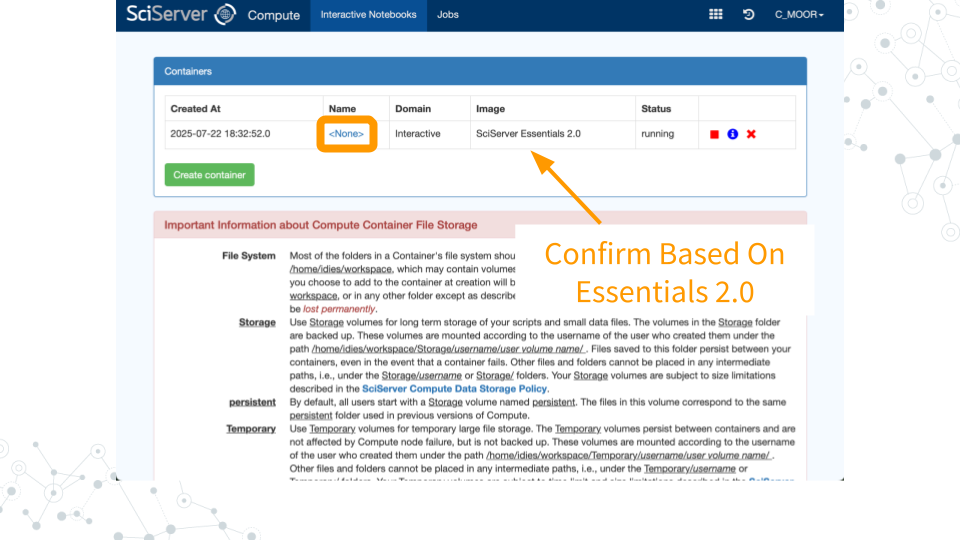


## 2.2 Start Jupyter

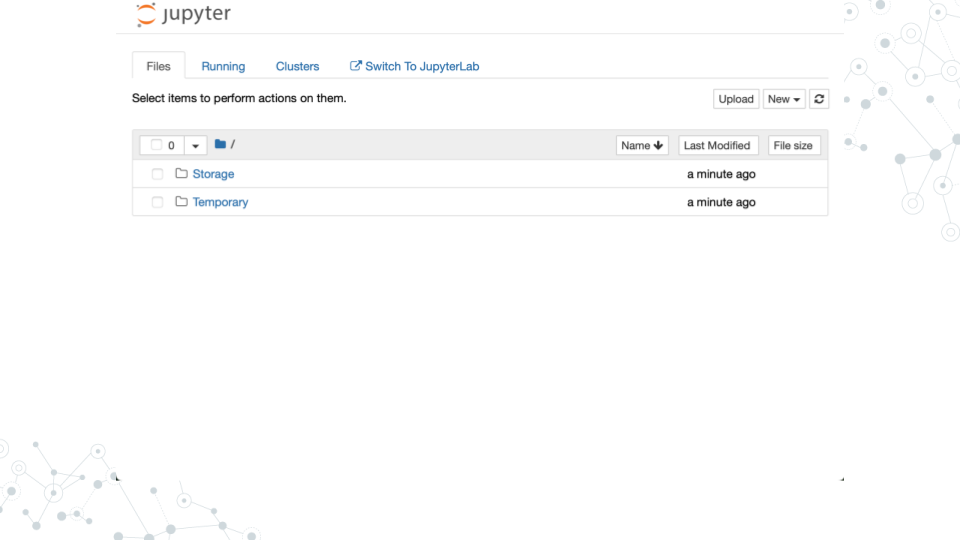
To create a container running Jupyter, click “Compute” in the SciServer Dashboard (<https://apps.sciserver.org/dashboard>). Click on “Create container” to see what possible compute configurations are available and select SciServer Essentials 2.0.



After clicking “Create”, you should now see a new entry in your list of containers. Confirm that it is using the SciServer Essentials 2.0 image. Finally, click on the name of your container and a new tab will open.



Voila! You should now see the Jupyter interface.



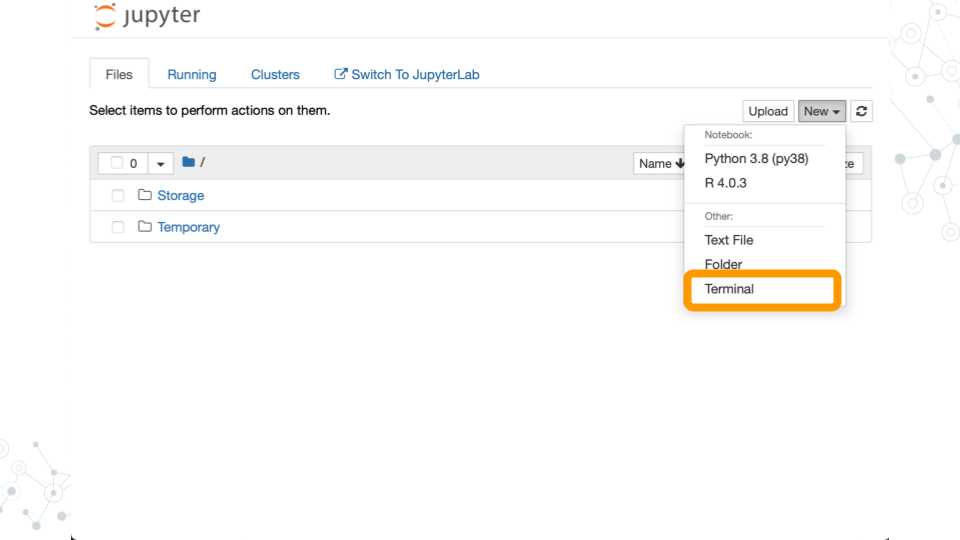
# 3 Running Programs

## 3.1 Learning objectives

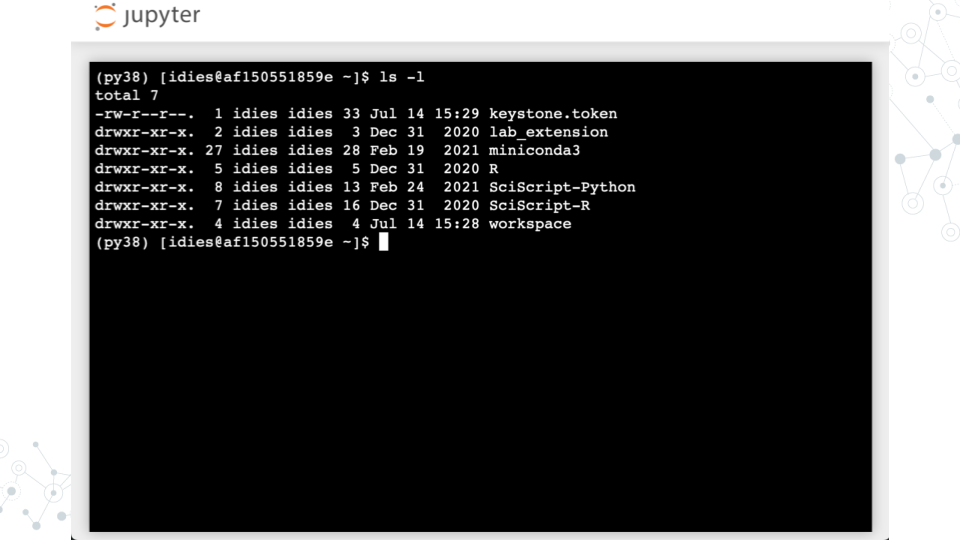
* Run UNIX commands
* Wrap UNIX commands in a Bash script
* Make a Bash script executable
* Run a Bash script

## 3.2 Run UNIX commands

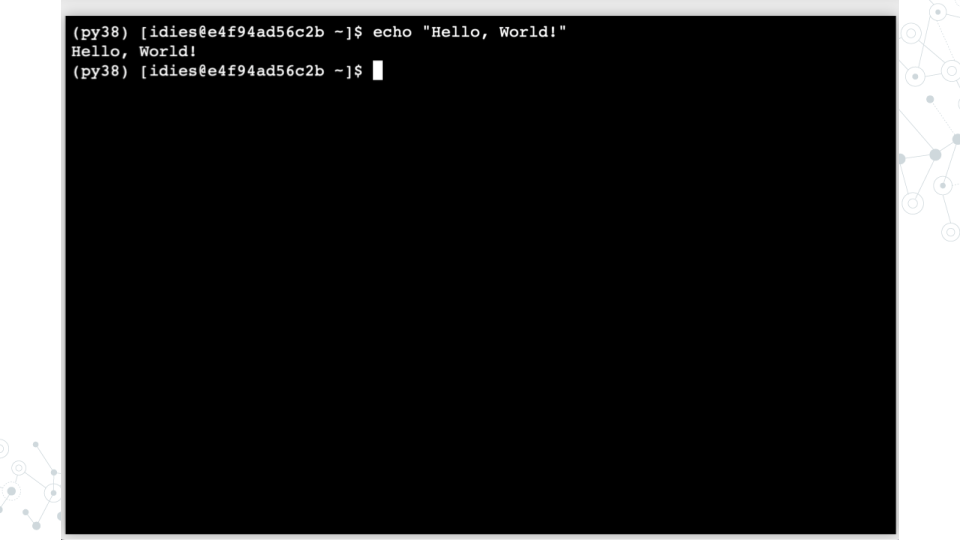
* Start a terminal



* Run the ls -l command to list files (the -l is a command line argument that instructs the ls program to modify its operation so that longer details are provided about each file)

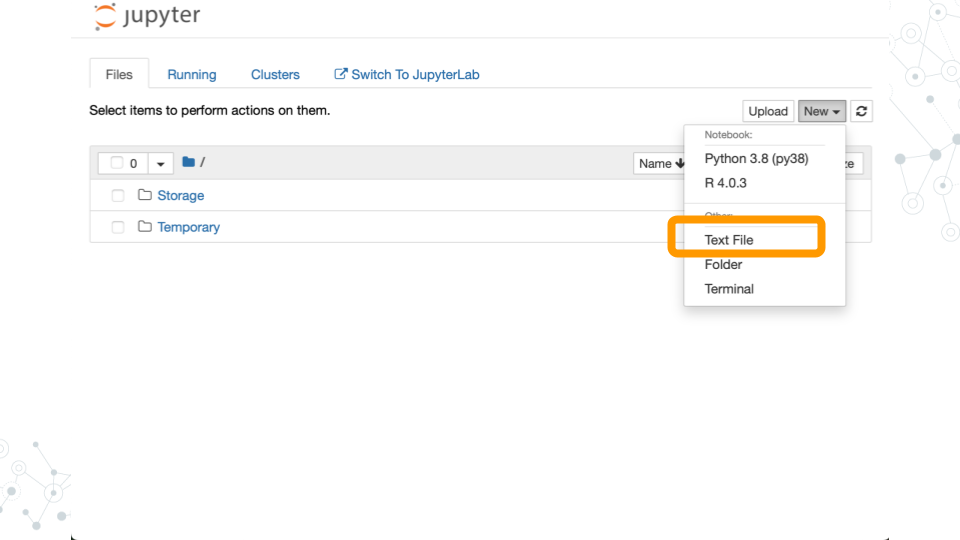


* Run echo "Hello, World!" to print text to the terminal



## 3.3 Wrap commands in a Bash script

* Create Text File



* Write your first Bash script
  + Add the following and save the file as 00-hello.sh



## 3.4 Run a Bash script

Now go back to the terminal:

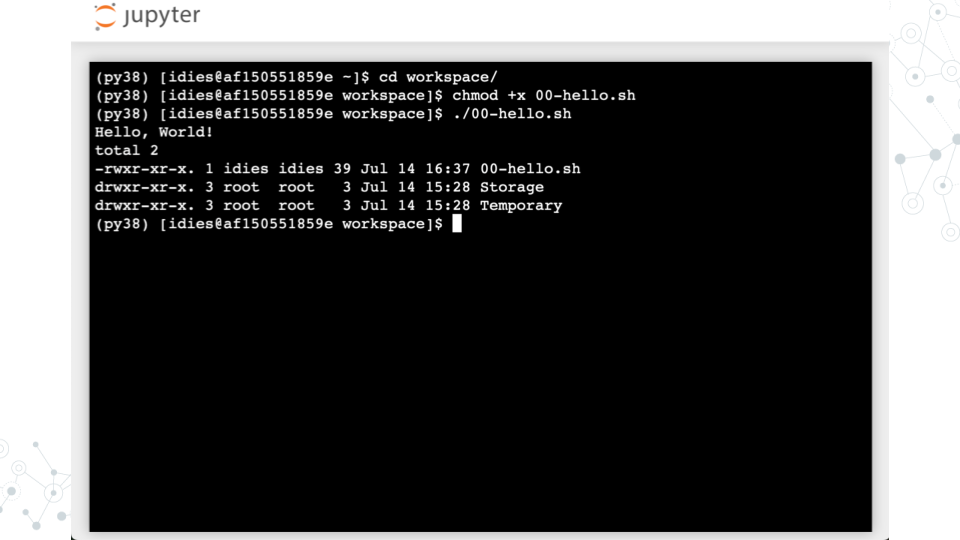
* Change to the workspace directory where you saved the file using the cd command

cd workspace/

* Make the script executable using the chmod command. This command changes the file permissions to allow execution (+x) of the script as a program

chmod +x 00-hello.sh

* Run the script by typing ./00-hello.sh in the terminal:



## 3.5 Summary

Congratulations! You have just:

* Created a Bash script that prints “Hello, World!” to the terminal

# 4 Hello, World!

## 4.1 Learning objectives

* Understand how to run Python scripts from the command line

## 4.2 Python “Hello, World!” example

* Create a new file named 01-helloworld.py using the text editor and write the following code:
* #!/usr/bin/env python3  
  print("Hello, World!")
  + The beginning of the first line, #!, it is called a “hashbang” or “shebang”. That indicates which interpreted should process (Python in this case) the file.
  + We use /usr/bin/env to find the Python interpreter in the user’s environment, which is more flexible than specifying a path like /usr/bin/python3. This way, it works regardless of where Python is installed on the system.
  + python3 is the interpreter that will run the script. It ensures that the script is executed with Python 3, which is important because Python 2 and Python 3 have different syntax and features.
  + Lastly print("Hello, World!") is the actual Python code that prints “Hello, World!” to the terminal.

## 4.3 Run a Python script

* Save the file and make it executable:
* chmod +x 01-helloworld.py
* In the terminal, run the script by typing:
* ./01-helloworld.py
* You should see the output:
* Hello, World!

## 4.4 Summary

Congratulations! You have just:

* Created a Python script that prints “Hello, World!” to the terminal
* Made the script executable
* Ran the script from the command line

# 5 Command line arguments

## 5.1 Learning objectives

* Understand how to accept command line arguments in Python scripts
* Learn how to access command line arguments using the sys module
* Understand how to handle command line arguments

## 5.2 Printing All Command Line Arguments

Let’s make our Python scripts interactive by accepting input from the command line. Create a new file called 02-arguments.py and type:

```python  
#!/usr/bin/env python3  
  
import sys  
  
print(sys.argv)  
```

* Save the file and make it executable:
* chmod +x 02-arguments.py
* Now run it directly with some arguments:
* ./02-arguments.py hello world 123
* You’ll see output like ['./02-arguments.py', 'hello', 'world', '123']

The sys module provide access to the system-specific parameter. The variable sys.argv contains all command line arguments passed to your script, including the script name itself as the first element.

## 5.3 Understanding Lists

Lists in Python are ordered collections of items enclosed in square brackets, like [1, 2, 3] or ["apple", "banana"]. Lists can contain different types of data and are accessed by the position (index), starting from 0. For example, my\_list[0] gets the first item, my\_list[1] gets the second item, and so in.

## 5.4 Accessing Specific Arguments

* Lets modify our script to print just the second command line argument:
* #!/usr/bin/env python3  
    
  import sys  
    
  print("Script name:", sys.argv[0])  
  print("First argument:", sys.argv[1])
* Run it with: ./02-arguments.py hello
* This prints:
* Script name: ./02-arguments.py  
  First argument: hello
* Notice how sys.argv[0] is always the script name, so the first actual argument is at index 1
* ***Warning***: If you don’t provide enough arguments, Python will crash with an “IndexError”. We’ll learn to handle this with if statements later

## 5.5 Arguments Are Strings

Command line arguments are always strings, even if they look like numbers.

* Update 02-arguments.py
* #!/usr/bin/env python3  
    
  import sys  
    
  # This won't work as expected  
  result = sys.argv[1] + sys.argv[2]  
  print("Without conversion:", result)  
    
  # Convert strings to integers first  
  num1 = int(sys.argv[1])  
  num2 = int(sys.argv[2])  
  print("With conversion:", num1 + num2)
* Run it with: ./02-arguments.py 5 3
* Output:
* Without conversion: 53  
  With conversion: 8

Without conversion, Python concatenates the strings “5” and “3” into “53”. The int() function converts string representations of numbers into actual integers that can be used in mathematical operations.

## 5.6 Summary

Congratulations! You have just:

* Created a Python script that accepts command line arguments
* Accessed specific arguments using sys.argv

# 6 Parsing files line by line

## 6.1 Learning objectives

* Understand how to read files in Python
* Learn how to iterate through lines in a file using a for loop

## 6.2 Create a Text File

* Let’s create a simple text file called sample.txt using the Jupyter text editor
* The file should contain the following lines:
* apple  
  banana  
  cherry  
  date

## 6.3 Open A File Stream

A file stream is like a pipeline that lets you read data from a file one piece at a time. The most common way to open a file is using the open() function.

* Create a new Python script called 03-parse-text-file.py
* #!/usr/bin/env python3  
    
  import sys  
    
  my\_file = open( sys.argv[1] )  
  print(my\_file)
* Save it and make it executable
* chmod +x 03-parse-text-file.py
* Run the script with the file name as an argument
* ./03-parse-text-file.py sample.txt
* This will print something like
* <\_io.TextIOWrapper name='sample1.txt' mode='r' encoding='UTF-8'>

As you can see, the print() function can’t print the file content directly. This output just indicates the file sample1.txt is opened in read mode ('r') with UTF-8 encoding.

## 6.4 Add a for loop

To read the file and print each line, we can use a for loop.

* Update 03-parse-text-file.py:
* #!/usr/bin/env python3  
    
  import sys  
    
  my\_file = open(sys.argv[1])  
    
  # for iterates through files  
  for my\_line in my\_file:  
   # objects have methods  
   my\_line = my\_line.rstrip("\n")  
   print( my\_line )  
    
  my\_file.close()
  + We use my\_file.close() to close the file after we’re done reading it. This is important to free up system resources.
* Save it and run the script again
* ./03-parse-text-file.py sample1.txt
* This will print each line of the file without extra spaces or newlines
* apple  
  banana  
  cherry  
  date

## 6.5 Summary

Congratulations! You have just:

* Created a Python script that reads a file
* Used a for loop to iterate through each line in the file
* Printed each line without extra spaces or newlines

# 7 Replicating the head command

## 7.1 Learning objectives

* How to replicate the head bash command in Python
* Understand how to read a file and print the first few lines
* Learn how to use a for loop to limit the number of lines printed

## 7.2 Create a longer file

* Let’s create a simple text file called sample2.txt using the Jupyter text editor
* The file should contain the following lines:
* apple  
  banana  
  cherry  
  date  
  elderberry  
  fig  
  grape  
  peach  
  kiwi  
  lemon

## 7.3 Practice the head command

This exercise should be done in the Jupyter terminal.

The head command in Bash prints the first few lines of a file. By default, it shows the first 10 lines, but you can specify a different number with the -n option.

* In the terminal, run the command:
* head sample2.txt
  + This will print all the 10 lines of sample2.txt
  + apple  
    banana  
    cherry  
    date  
    elderberry  
    fig  
    grape  
    peach  
    kiwi  
    lemon
* Now try running it with the -n option to print only the first 5 lines
* head -n 5 sample2.txt
  + You should see the following output
  + apple  
    banana  
    cherry  
    date  
    elderberry

## 7.4 Designing the head algorithm

We can replicate the head command by adding some logic to the for loop we introduced in Ch 6. Parsing text files.

Starting with a for loop is great for iterating through items in a collection, like lines in a file. We can add a counter variable and a conditional if statement to limit how many lines we print.

i = 0  
for my\_line in my\_file:  
 if i >= max\_lines:  
 break  
 my\_line = my\_line.rstrip("\n")  
 print( my\_line )  
 i = i + 1

Key points:

* We can use a counter variable i to keep track of how many lines we’ve printed. i = i + 1 increments the counter by 1 each time we print a line.
* The if statement checks if the number of lines printed exceeds the maximum specified by the user
* The max\_lines variable is set to 10 by default, but can be changed by providing a second command line argument

## 7.5 Coding step by step

* Create a new python script called 04-head.py and add what we introduced in Ch 6. Parsing text files
* #!/usr/bin/env python3  
    
  import sys  
    
  my\_file = open( sys.argv[1] )  
    
  for my\_line in my\_file:  
   my\_line = my\_line.rstrip("\n")  
   print( my\_line )  
    
  my\_file.close()
* Now add code after the open() to set a maximum number of lines to print or defaulting to 10 if not specified
* max\_lines = 10  
  if len(sys.argv) > 2:  
   max\_lines = int(sys.argv[2])
* Finally, add the for loop to limit the number of lines printed
* i = 0  
  for my\_line in my\_file:  
   if i >= max\_lines:  
   break  
   my\_line = my\_line.rstrip("\n")  
   print( my\_line )  
   i = i + 1
* The complete script should look like this:
* #!/usr/bin/env python3  
    
  import sys  
    
  my\_file = open( sys.argv[1] )  
    
  max\_lines = 10  
  if len(sys.argv) > 2:  
   max\_lines = int(sys.argv[2])  
    
  i = 0  
  for my\_line in my\_file:  
   if i >= max\_lines:  
   break  
   my\_line = my\_line.rstrip("\n")  
   print( my\_line )  
   i = i + 1  
    
  my\_file.close()
* Save the file and make it executable
* chmod +x 04-head.py
* Run the script with the file name and number of lines as arguments
* ./04-head.py sample2.txt 5
* This will print the first 5 lines of sample2.txt
* apple  
  banana  
  cherry  
  date  
  elderberry

## 7.6 Summary

Congratulations! You have just:

* Created a Python script that replicates the head command
* Used command line arguments to specify the file and number of lines to print

# 8 Knowledge Check

Coming soon!

# 9 Next Steps

Congratulations! You’ve successfully completed all the exercises and practiced creating multiple Python scripts and running them at the UNIX command line. In addition to these practical skills, you got a taste of how Python works, and how it can be used to parse through text files line by line. We hope this helps demystify ever so slightly how UNIX commands can, and often are, Python scripts. As a reminder, here’s what we covered in each chapter:

* Set up Compute: Join SciServer, start Jupyter
* Running Programs: Run UNIX commands, wrap UNIX commands in a Bash script, run a Bash script
* Hello, World!: Python “Hello, World!” example and run a Python script
* Command line arguments: Print all command line arguments, understanding lists, accessing specific arguments, and arguments are strings
* Parsing files line by line: Create a text file, open a file stream, add a for loop
* Replicating the head command: Practice the head command, designing the head algorithm, coding step by step

You should now have all the tools you need to follow a full fledged Python tutorial. Here is one recommendation from Software Carpentry:

* [Programming with Python](https://swcarpentry.github.io/python-novice-inflammation/)

As you continue your journey with Python, try replicating these two additional UNIX commands that are commonly used to filter and subset text files:

* grep.py - Print only lines that match a given string (hint: if pattern in line:)
* cut.py - For tab delimited files, print a specific column (hint: fields = line.split())

# About the Authors

These credits are based on our [course contributors table guidelines](https://www.ottrproject.org/more_features.html#giving-credits-to-contributors).

| Credits | Names |
| --- | --- |
| Lead Content Instructor(s) | Javier Carpinteyro-Ponce, Frederick Tan |
| Acknowledgements | Rajiv McCoy, Mike Sauria, and countless TAs |
| Dedication | [James Taylor](https://jxtxfoundation.org) |