LAB 1:
PYTHON FOUNDATION

University of Washington, Seattle

Fall 2024

OUTLINE

Part 1: Getting Started

- Python Environment Setup

Part 2: Python Basics

- Data types & variables
- Operators in Python
- Conditionals, Loops, Functions

Part 3: NumPy and Plotting

- Introduction to NumPy
- Plotting with Matplotlib

Part 4: Regression with Scikit-learn

- Introduction to Scikit-learn
- Ridge regression example

Lab Assignment

- Scaling Data with Standard Scaling
- Data Splitting
- Regression with scikit-learn

Supplementary: Basic Debugging in Python

PART 1:

GETTING STARTED

Python Environment Setup

Python Environment Setup

Python Environment Options

Option 1

miniCONDA®

Conda Package manager
+ Base Python
+ Base modules

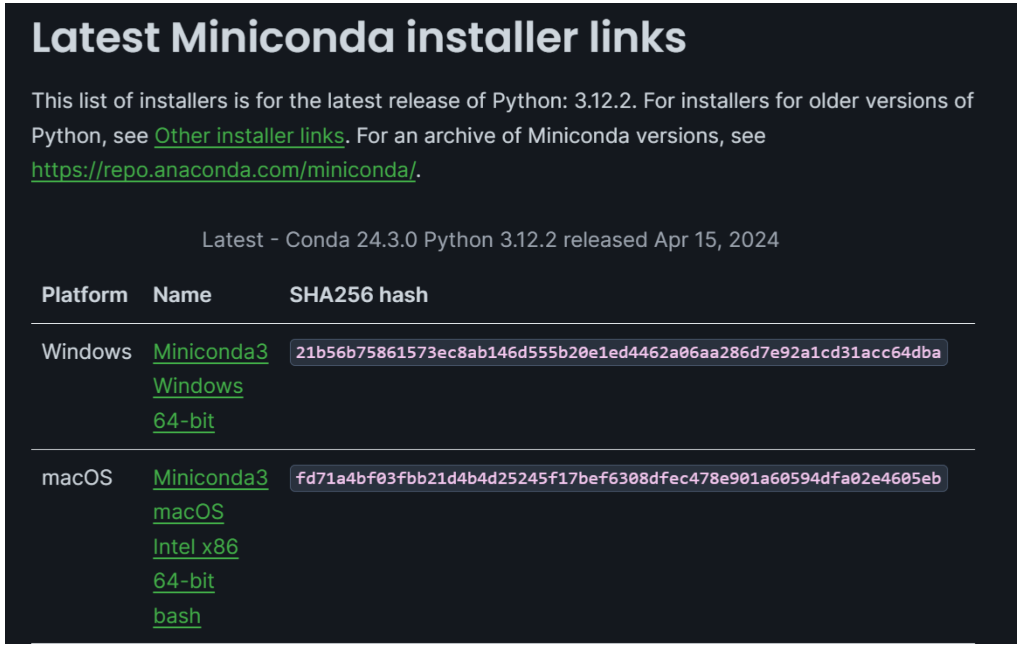
Option 2

ANACONDA®

Conda Package manager
+ Base Python
+ Base modules
+ 150 additional packages

Installing Python environment

Download pages for Miniconda and Anaconda installers

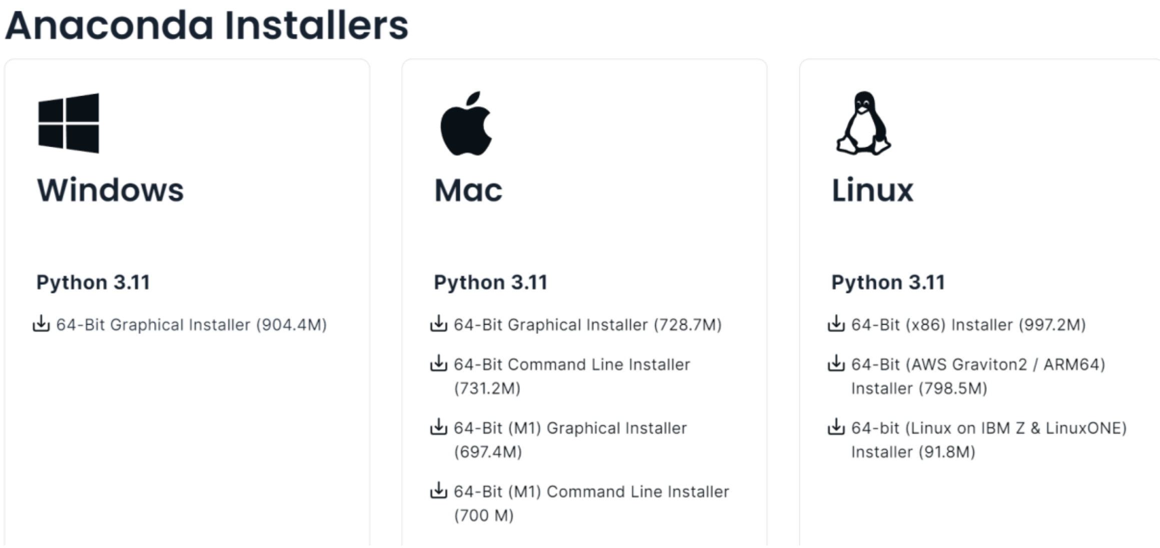


Latest Miniconda installer links

This list of installers is for the latest release of Python: 3.12.2. For installers for older versions of Python, see Other installer links. For an archive of Miniconda versions, see https://repo.anaconda.com/miniconda/.

Latest - Conda 24.3.0 Python 3.12.2 released Apr 15, 2024

Platform	Name	SHA256 hash
Windows	Miniconda3 Windows 64-bit	`21b56b75861573ec8ab146d555b20e1ed4462a06aa286d7e92a1cd31acc64dba`
macOS	Miniconda3 macOS Intel x86 64-bit bash	`fd71a4bf03fbb21d4b4d25245f17bef6308dfec478e901a60594dfa02e4605eb`



Anaconda Installers

Windows

Python 3.11

- 64-Bit Graphical Installer (904.4M)

Mac

Python 3.11

- 64-Bit Graphical Installer (728.7M)
- 64-Bit Command Line Installer (731.2M)
- 64-Bit (M1) Graphical Installer (697.4M)
- 64-Bit (M1) Command Line Installer (700 M)

Linux

Python 3.11

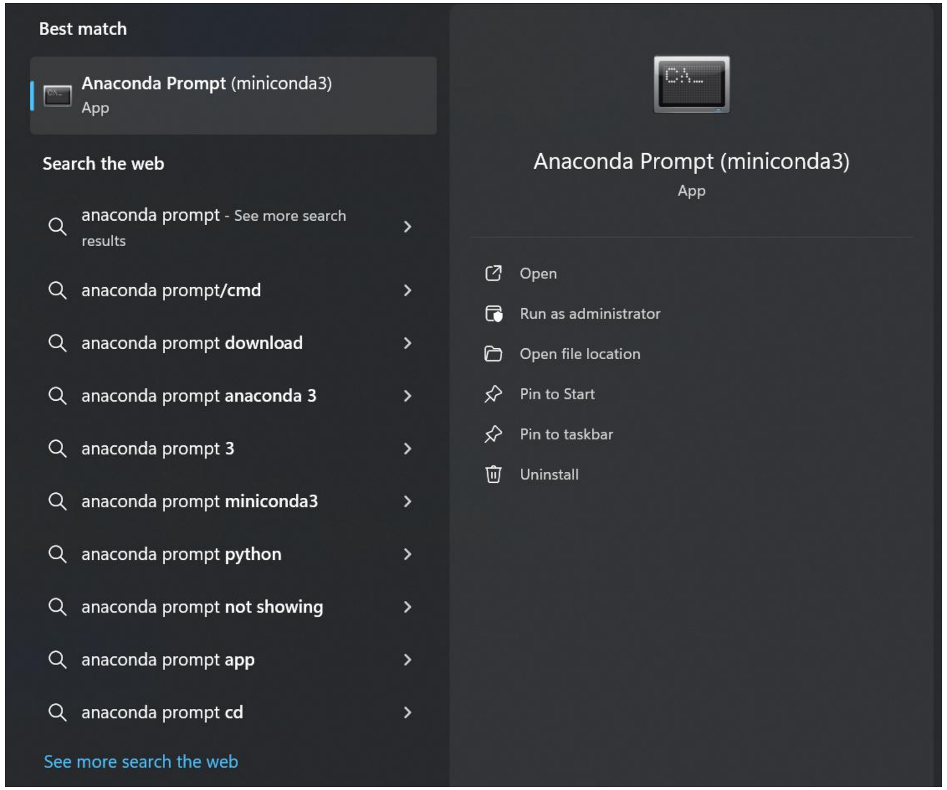
- 64-Bit (x86) Installer (997.2M)
- 64-Bit (AWS Graviton2 / ARM64) Installer (798.5M)
- 64-bit (Linux on IBM Z & LinuxONE) Installer (91.8M)

https://docs.anaconda.com/free/miniconda/index.html

https://www.anaconda.com/download

Starting Anaconda prompt

Windows



Best match

Anaconda Prompt (miniconda3)
App

Search the web

- anaconda prompt - See more search results
- anaconda prompt/cmd
- anaconda prompt download
- anaconda prompt anaconda 3
- anaconda prompt 3
- anaconda prompt miniconda3
- anaconda prompt python
- anaconda prompt not showing
- anaconda prompt app
- anaconda prompt cd

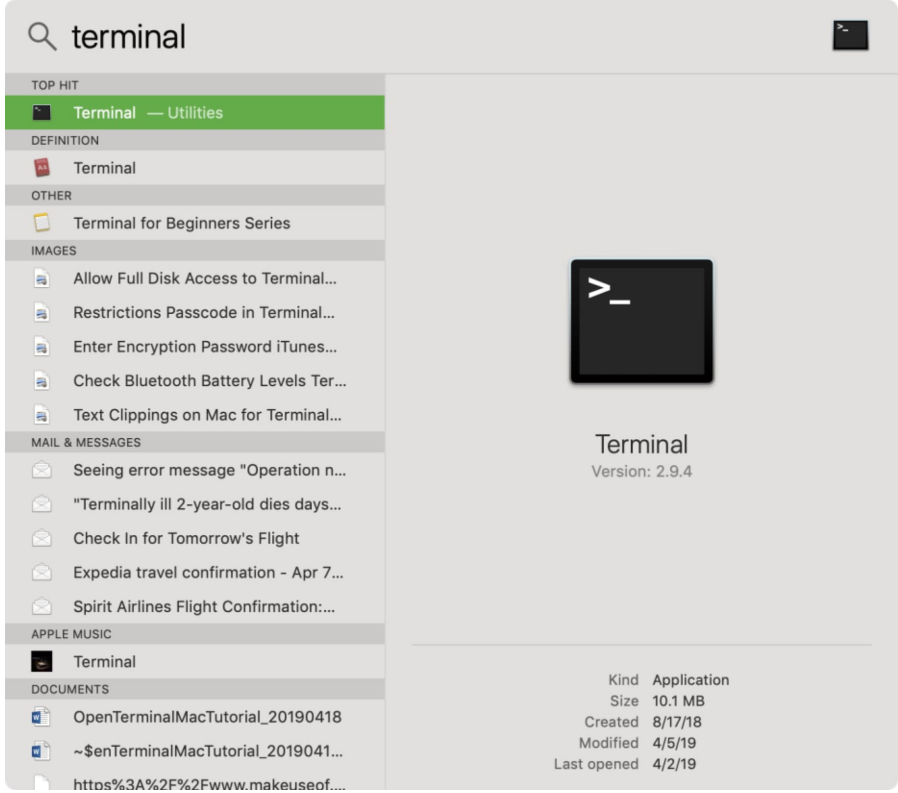
See more search the web

Anaconda Prompt (miniconda3)
App

- Open
- Run as administrator
- Open file location
- Pin to Start
- Pin to taskbar
- Uninstall

From start menu, enter
Anaconda Prompt

Mac/Linux



terminal

TOP HIT
Terminal — Utilities

DEFINITION
Terminal

OTHER
Terminal for Beginners Series

IMAGES
Allow Full Disk Access to Terminal...
Restrictions Passcode in Terminal...
Enter Encryption Password iTunes...
Check Bluetooth Battery Levels Ter...
Text Clippings on Mac for Terminal...

MAIL & MESSAGES
Seeing error message "Operation n...
"Terminally ill 2-year-old dies days...
Check In for Tomorrow's Flight
Expedia travel confirmation - Apr 7...
Spirit Airlines Flight Confirmation:...

APPLE MUSIC
Terminal

DOCUMENTS
OpenTerminalMacTutorial_20190418
~$enTerminalMacTutorial_2019041...
https%3A%2F%2Fwww.makeuseof....

Terminal
Version: 2.9.4

Kind	Application
Size	10.1 MB
Created	8/17/18
Modified	4/5/19
Last opened	4/2/19

Enter **Terminal**

Installing Python dependencies

Within **Anaconda Prompt** or **Terminal**

If using **Miniconda**:

```
> conda install scipy matplotlib ipython jupyter seaborn
```

Install Python dependencies

If using **Anaconda,** above packages should be pre-installed

Starting up Jupyter Notebook

Windows

Start Anaconda Prompt

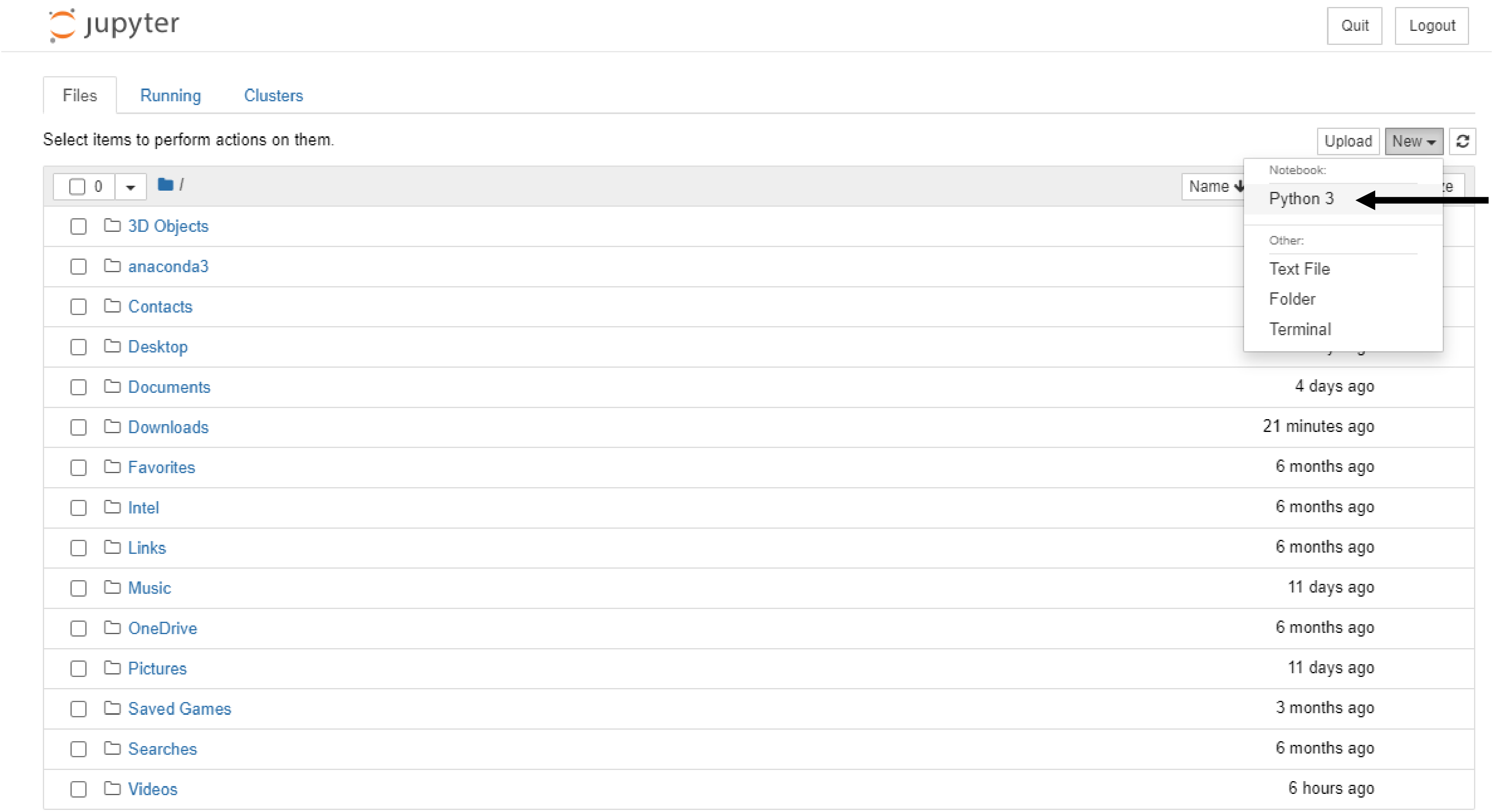
Type “jupyter notebook”

Mac/Linux

Start terminal

Type “jupyter notebook”

Starting up Jupyter Notebook



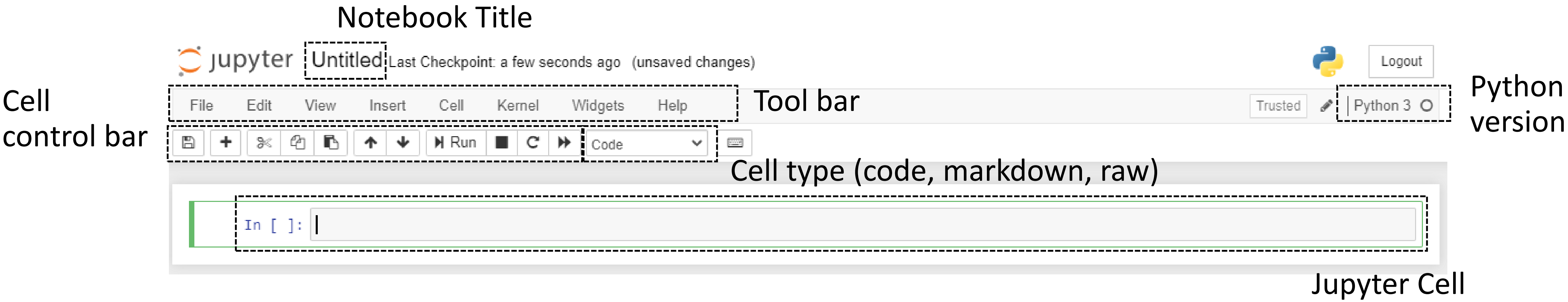
Select items to perform actions on them.

□ 0 /	Name ↓
□ 3D Objects	
□ anaconda3	
□ Contacts	
□ Desktop	
□ Documents	4 days ago
□ Downloads	21 minutes ago
□ Favorites	6 months ago
□ Intel	6 months ago
□ Links	6 months ago
□ Music	11 days ago
□ OneDrive	6 months ago
□ Pictures	11 days ago
□ Saved Games	3 months ago
□ Searches	6 months ago
□ Videos	6 hours ago

Create a new notebook

You can also use Jupyter Navigator to load .ipynb notebook files

Jupyter Notebook



See https://www.dataquest.io/blog/jupyter-notebook-tutorial to familiarize yourself with basic controls

Online option: Google Colaboratory

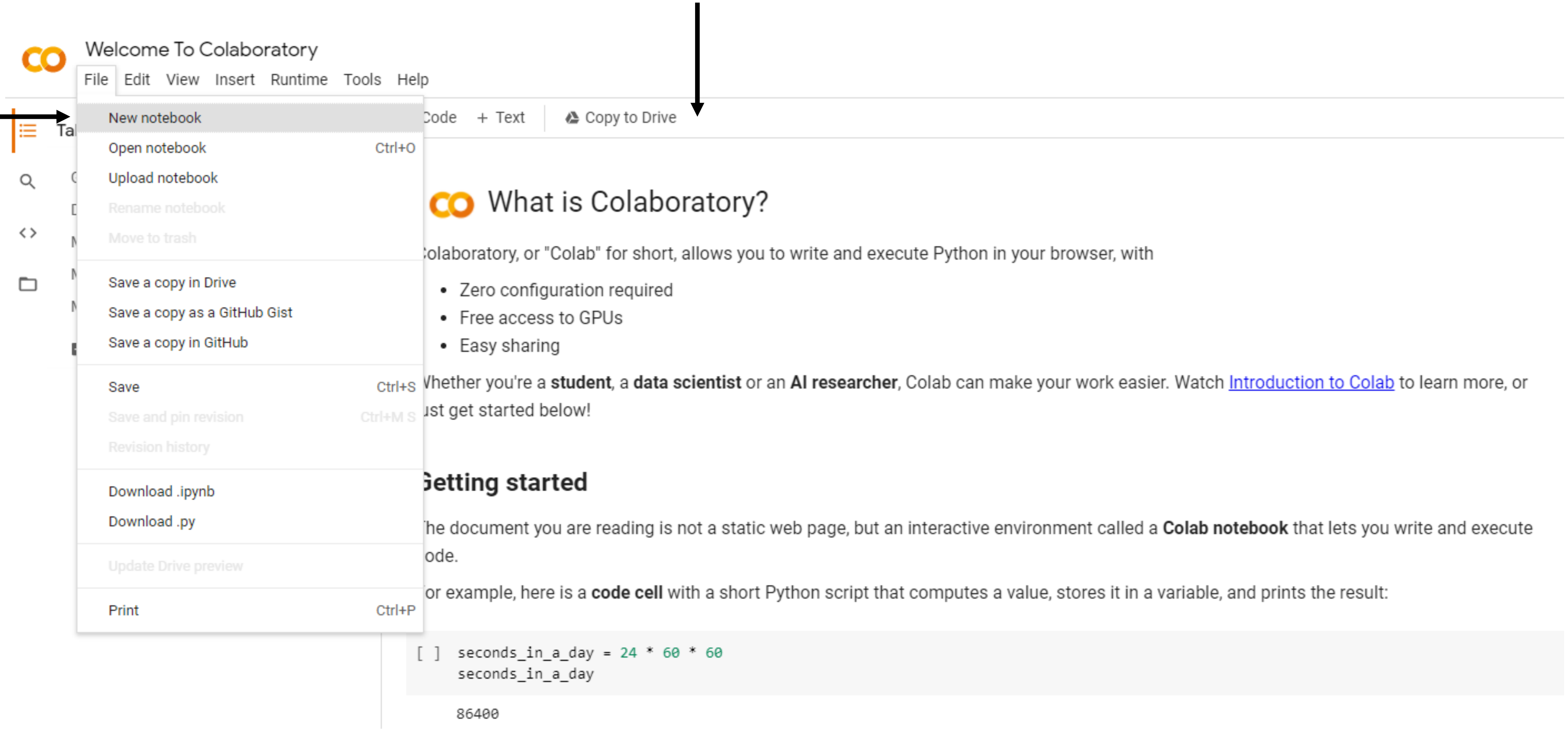
A free Jupyter notebook environment that runs in the cloud

- Saves in Google drive
- Github commit style code sharing with others
- Maximum runtime of 12hrs (Free version)
- Pre-equipped with latest scientific packages (Numpy, Scipy, etc)

Setting up Google Colaboratory

Tutorial to Colab https://colab.research.google.com/notebooks/intro.ipynb

Create new Notebook



What is Colaboratory?

olaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Vhether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or ust get started below!

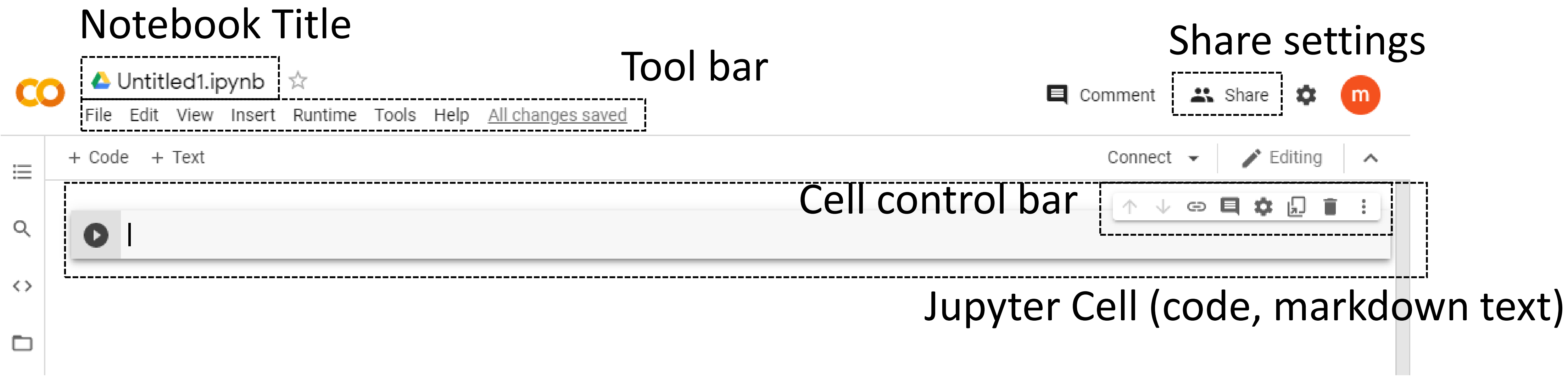
etting started

he document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute ode.

or example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ]  seconds_in_a_day = 24 * 60 * 60
     seconds_in_a_day

     86400
```

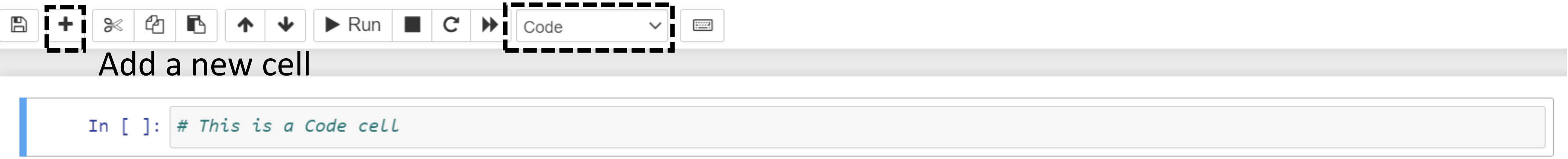
Setting up Google Colaboratory



See **Getting Started** part of https://colab.research.google.com/notebooks/intro.ipynb to familiarize yourself with basic controls

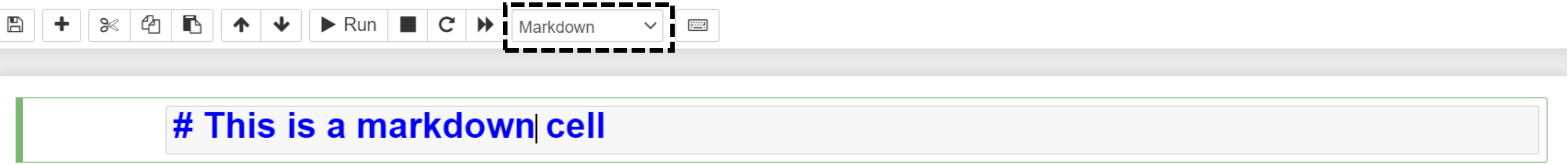
Code vs Markdown Cell

Jupyter Notebook



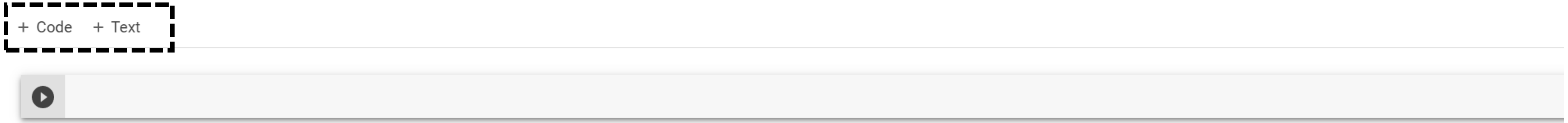
Add a new cell

```
In [ ]: # This is a Code cell
```



```
# This is a markdown cell
```

Google Colab



PART 2:

PYTHON BASICS

Python Data Types and Variables

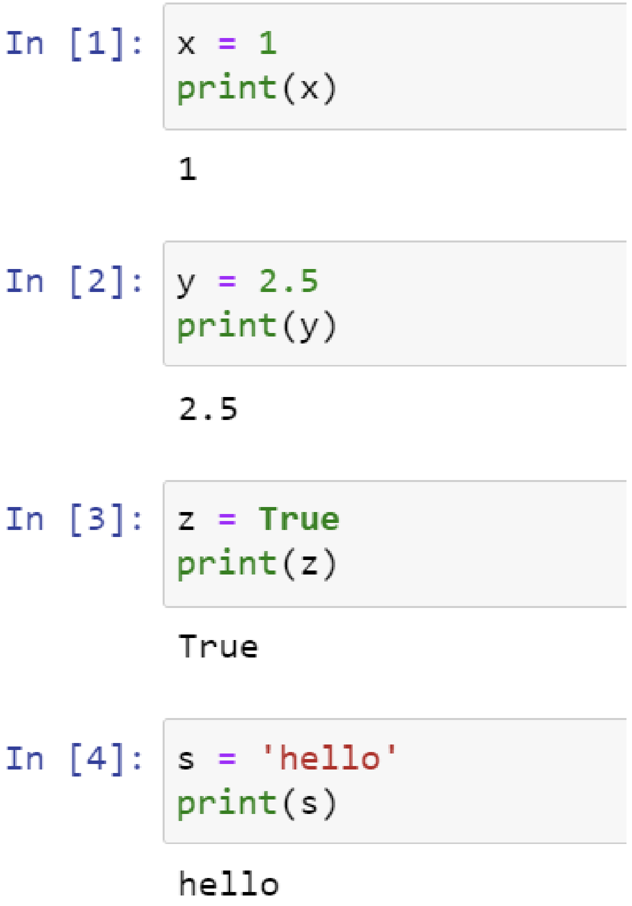
Operators in Python

Conditionals, Loops, Functions

Python Data Types and Variables

Python Data Types and Variables

Jupyter Notebook Code

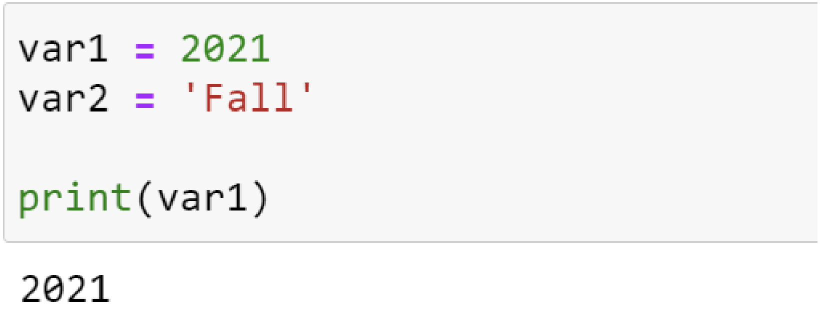


```
In [1]: x = 1
        print(x)

        1

In [2]: y = 2.5
        print(y)

        2.5

In [3]: z = True
        print(z)

        True

In [4]: s = 'hello'
        print(s)

        hello
```

Variable	Data Type	Value
x	int	1
y	float	2.5
z	bool	True
s	str	‘hello’

Printing Variables with ‘print’

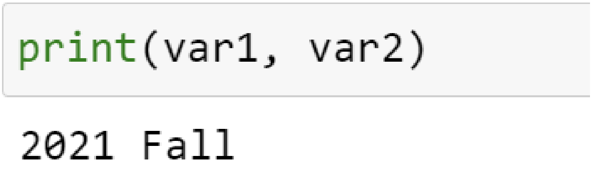
Print single variable



```
var1 = 2021
var2 = 'Fall'

print(var1)
```

```
2021
```

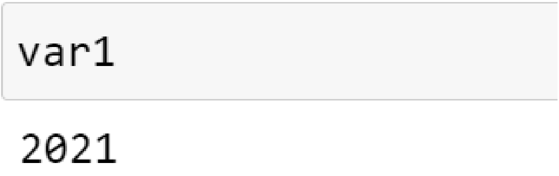
Print multiple variable



```
print(var1, var2)
```

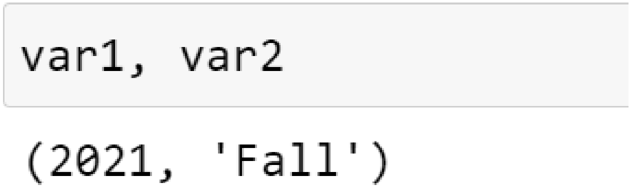
```
2021 Fall
```

Variables called in a cell can be displayed without print function, as ‘outputs’



```
var1
```

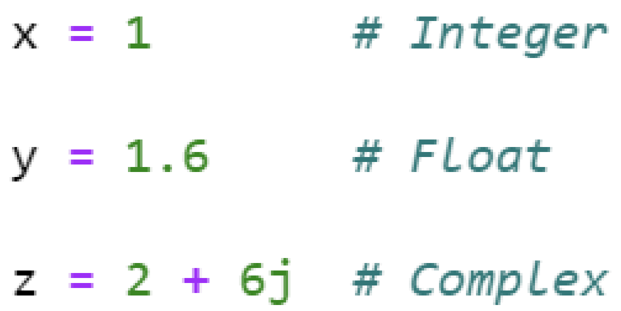
```
2021
```



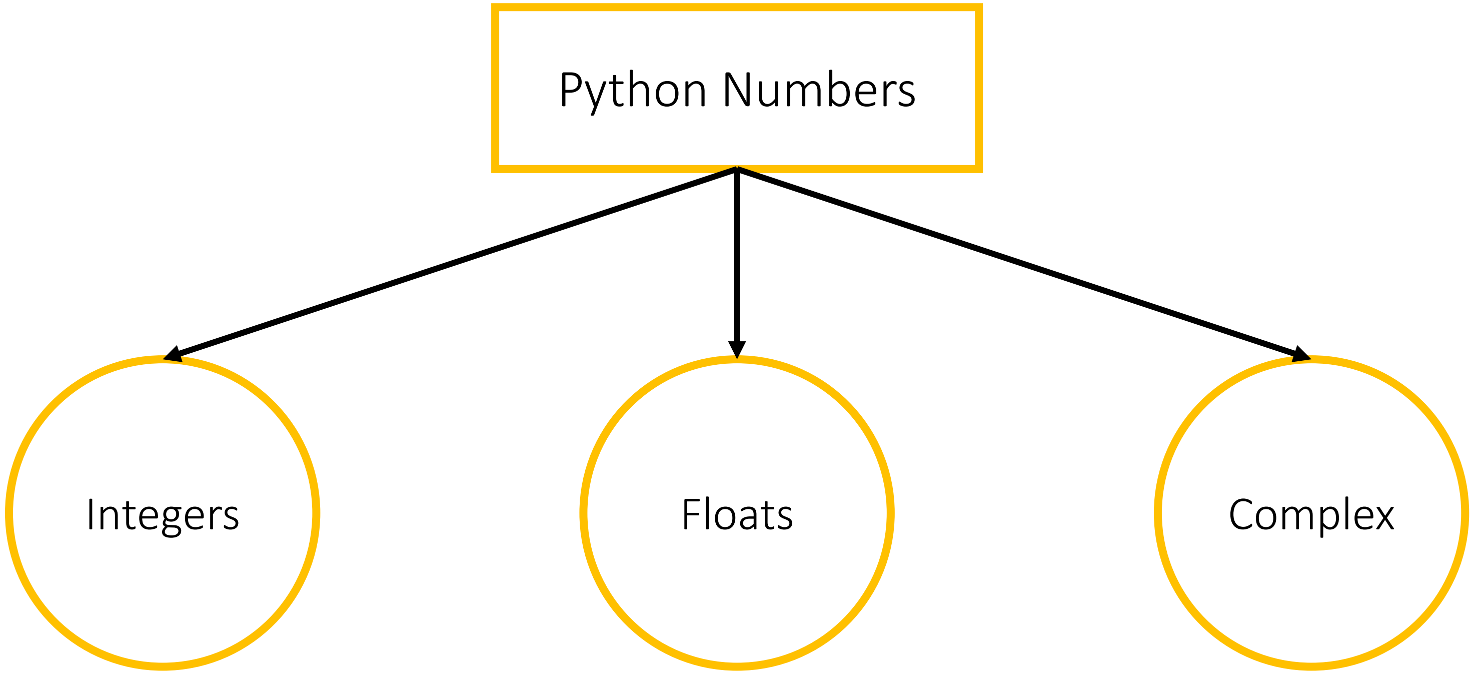
```
var1, var2
```

```
(2021, 'Fall')
```

Python Data Types: Numbers



```
x = 1        # Integer

y = 1.6      # Float

z = 2 + 6j   # Complex
```



Python Numbers

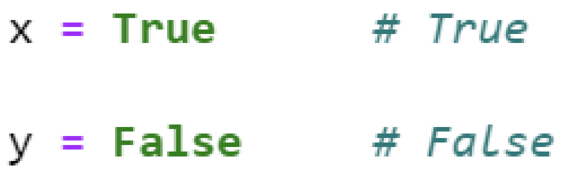
Integers

Floats

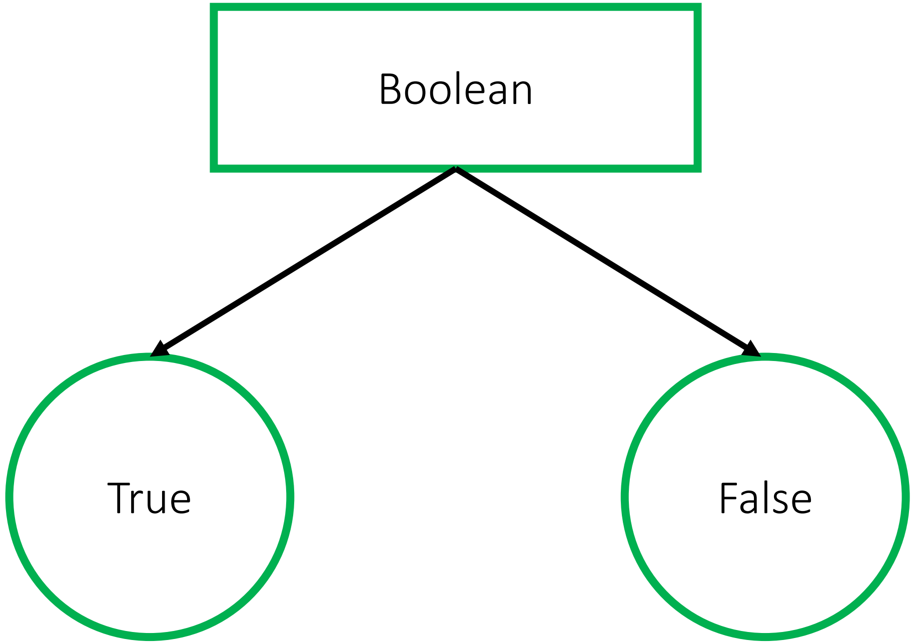
Complex

Examples   1, 300, -50   0.001, -15.7, 6.5, 1.0   6.4 – 3.6j, 8j, -10-2j

Python Data Types: Booleans

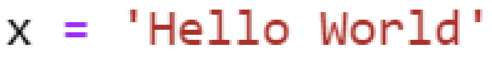


```
x = True        # True

y = False       # False
```



First letter should be capitalized

Python Data Type: Strings

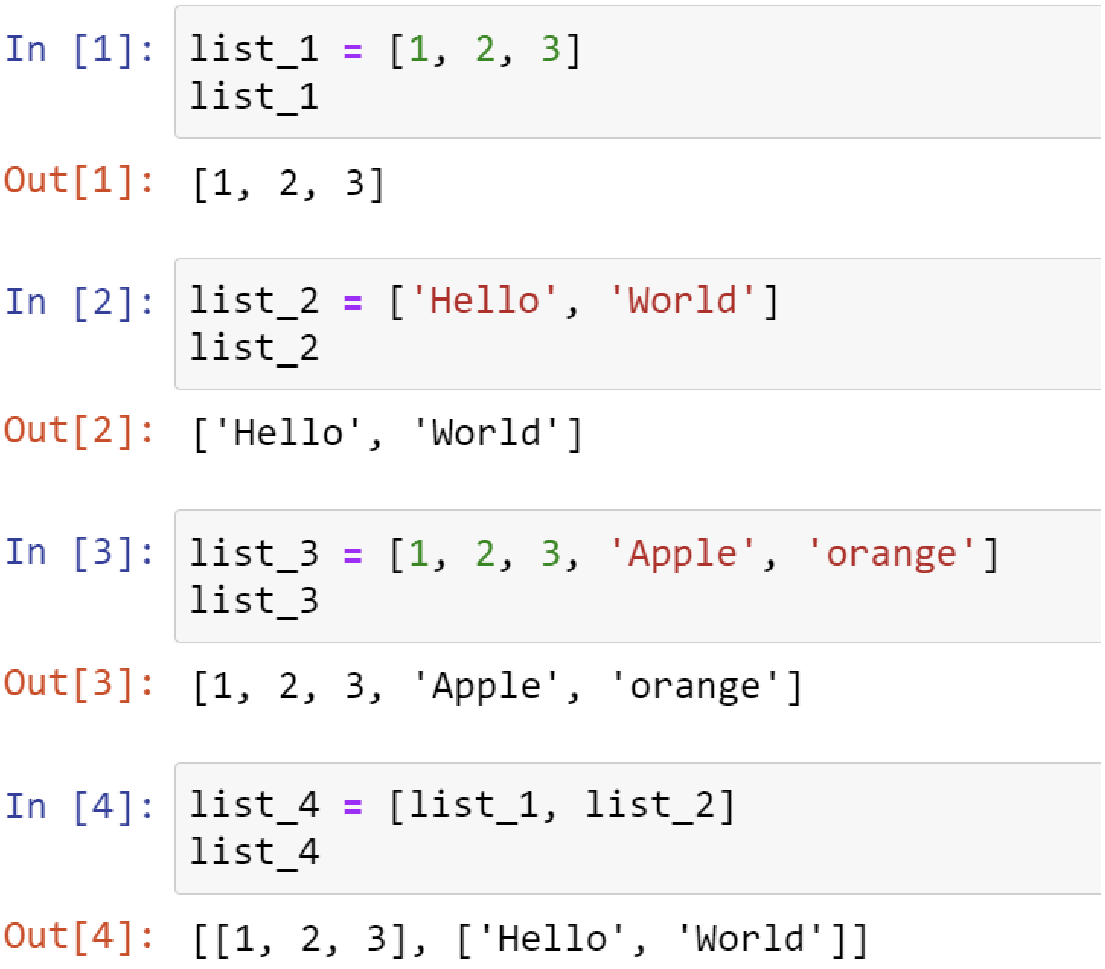


```
x = 'Hello World'
```

	H	e	l	l	o		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10

Length of string = 11

Grouping Data with Python Lists



```
In [1]:  list_1 = [1, 2, 3]
         list_1

Out[1]:  [1, 2, 3]

In [2]:  list_2 = ['Hello', 'World']
         list_2

Out[2]:  ['Hello', 'World']

In [3]:  list_3 = [1, 2, 3, 'Apple', 'orange']
         list_3

Out[3]:  [1, 2, 3, 'Apple', 'orange']

In [4]:  list_4 = [list_1, list_2]
         list_4

Out[4]:  [[1, 2, 3], ['Hello', 'World']]
```

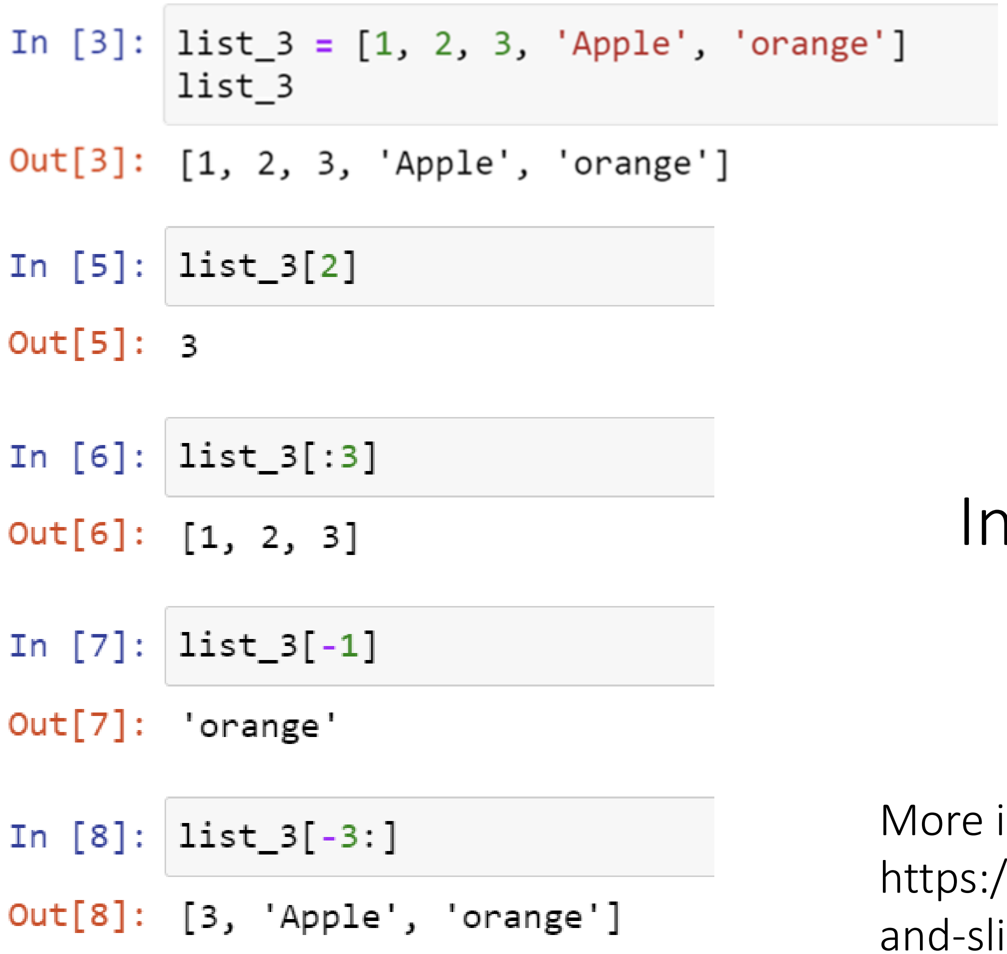
List of numbers

List of strings

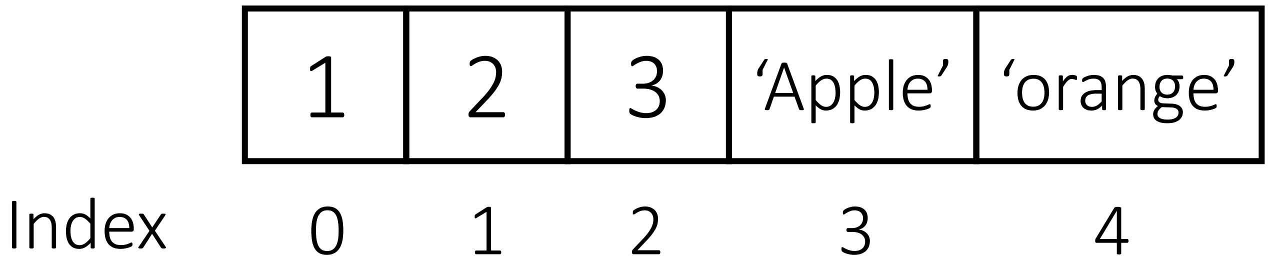
List of numbers + strings

List of lists

Indexing Lists



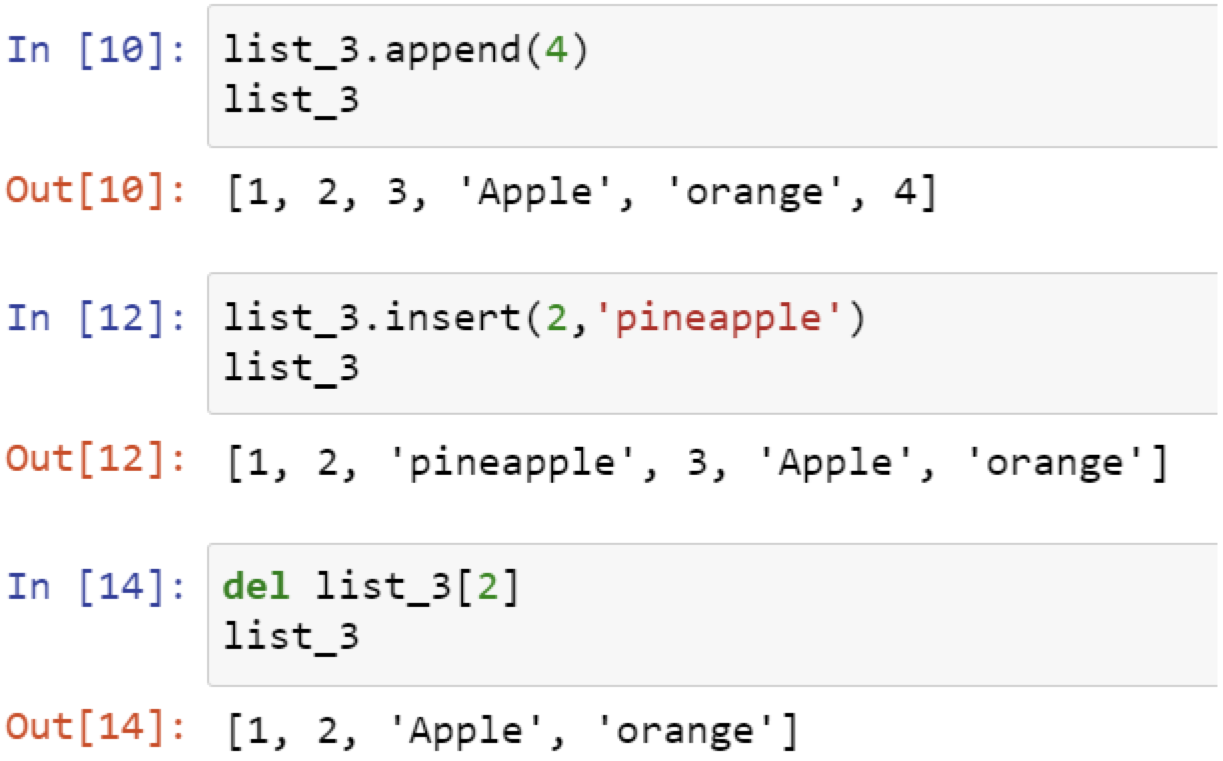
```
In [3]: list_3 = [1, 2, 3, 'Apple', 'orange']
        list_3

Out[3]: [1, 2, 3, 'Apple', 'orange']

In [5]: list_3[2]

Out[5]: 3

In [6]: list_3[:3]

Out[6]: [1, 2, 3]

In [7]: list_3[-1]

Out[7]: 'orange'

In [8]: list_3[-3:]

Out[8]: [3, 'Apple', 'orange']
```



	1	2	3	‘Apple’	‘orange’
Index	0	1	2	3	4

More information on indexing:
https://railsware.com/blog/python-for-machine-learning-indexing-and-slicing-for-lists-tuples-strings-and-other-sequential-types/

Append, Insert, Delete List Elements



```
In [10]: list_3.append(4)
         list_3

Out[10]: [1, 2, 3, 'Apple', 'orange', 4]

In [12]: list_3.insert(2,'pineapple')
         list_3

Out[12]: [1, 2, 'pineapple', 3, 'Apple', 'orange']

In [14]: del list_3[2]
         list_3

Out[14]: [1, 2, 'Apple', 'orange']
```

Appending a new value

Inserting a new value into an index

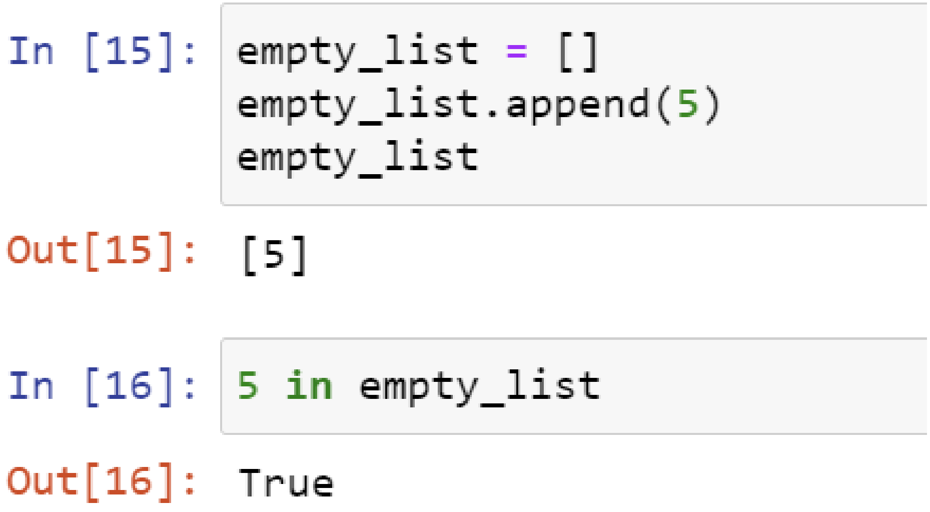
2: Index to insert,

‘pineapple’: Value to insert

Deleting an existing value

2: Index to delete

Empty List and Element Check



```
In [15]: empty_list = []
         empty_list.append(5)
         empty_list

Out[15]: [5]
```

Appending a value to an empty list []

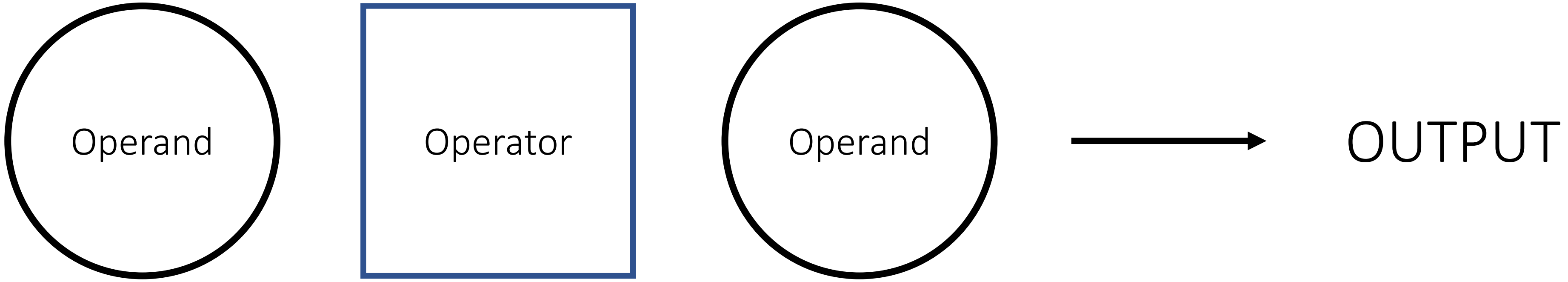
```
In [16]: 5 in empty_list

Out[16]: True
```

Checking if an element is in the list

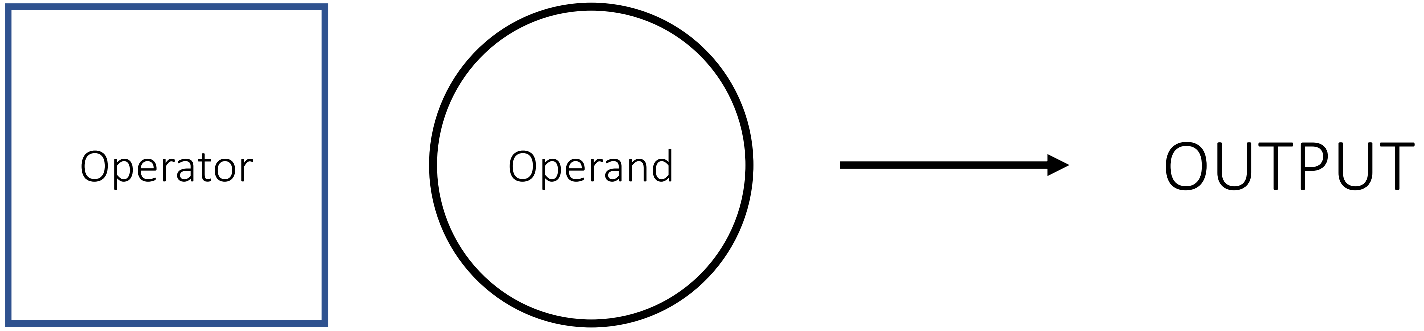
Operators in Python

Operators in Python

1. Binary Operator



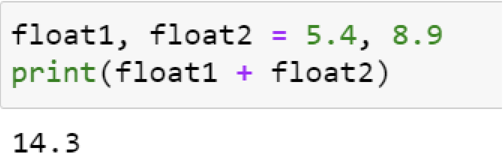
2. Unary Operator



Arithmetic Operators

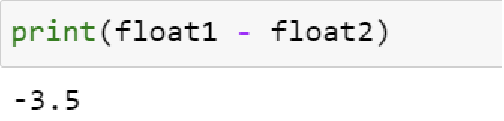
	Operator
Addition	+
Subtraction	−
Multiplication	*
Exponent	**
Division	/
Modulo	%

Example



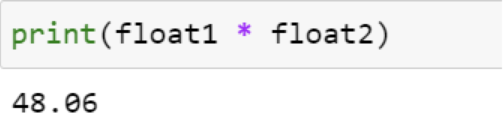
```
float1, float2 = 5.4, 8.9
print(float1 + float2)
```

```
14.3
```



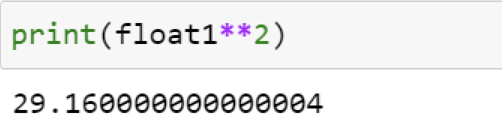
```
print(float1 - float2)
```

```
-3.5
```



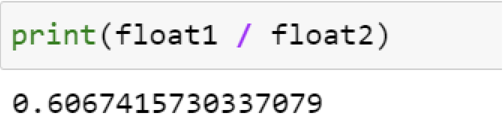
```
print(float1 * float2)
```

```
48.06
```



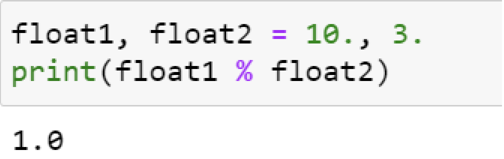
```
print(float1**2)
```

```
29.160000000000004
```



```
print(float1 / float2)
```

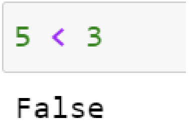
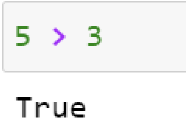
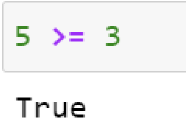
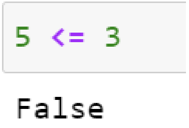
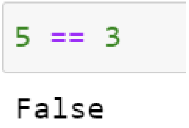
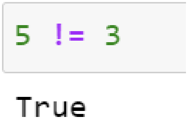
```
0.6067415730337079
```



```
float1, float2 = 10., 3.
print(float1 % float2)
```

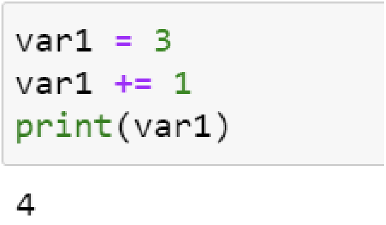
```
1.0
```

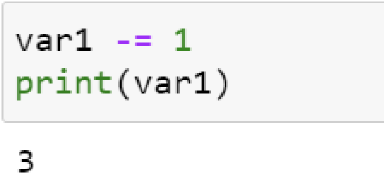
Comparison Operators

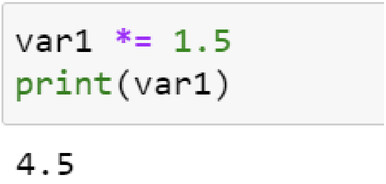
	Operator	Example
Greater Than	<	 `5 < 3` `False`
Less Than	>	 `5 > 3` `True`
Greater Than or Equal to	>=	 `5 >= 3` `True`
Less Than or Equal to	<=	 `5 <= 3` `False`
Equivalent to	==	 `5 == 3` `False`
Not Equivalent to	!=	 `5 != 3` `True`

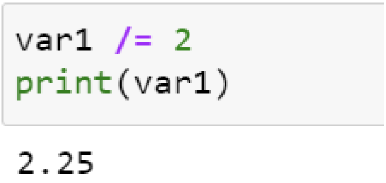
Assignment Operators

	Operator	Example

Add and Assign	+=	
		`var1 = 3`
		`var1 += 1`
		`print(var1)`
		`4`

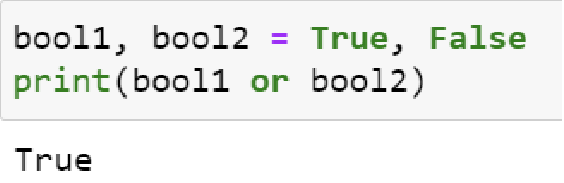
Subtract and Assign	-=	
		`var1 -= 1`
		`print(var1)`
		`3`

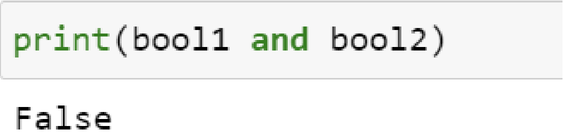
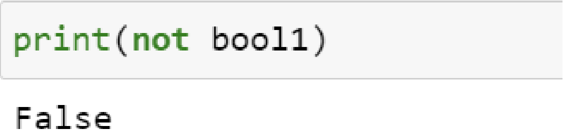
Multiply and Assign	*=	
		`var1 *= 1.5`
		`print(var1)`
		`4.5`

Divide and Assign	/=	
		`var1 /= 2`
		`print(var1)`
		`2.25`

Logical Operators

	Operator	Example

OR	or	 `bool1, bool2 = True, False`
		`print(bool1 or bool2)`
		`True`

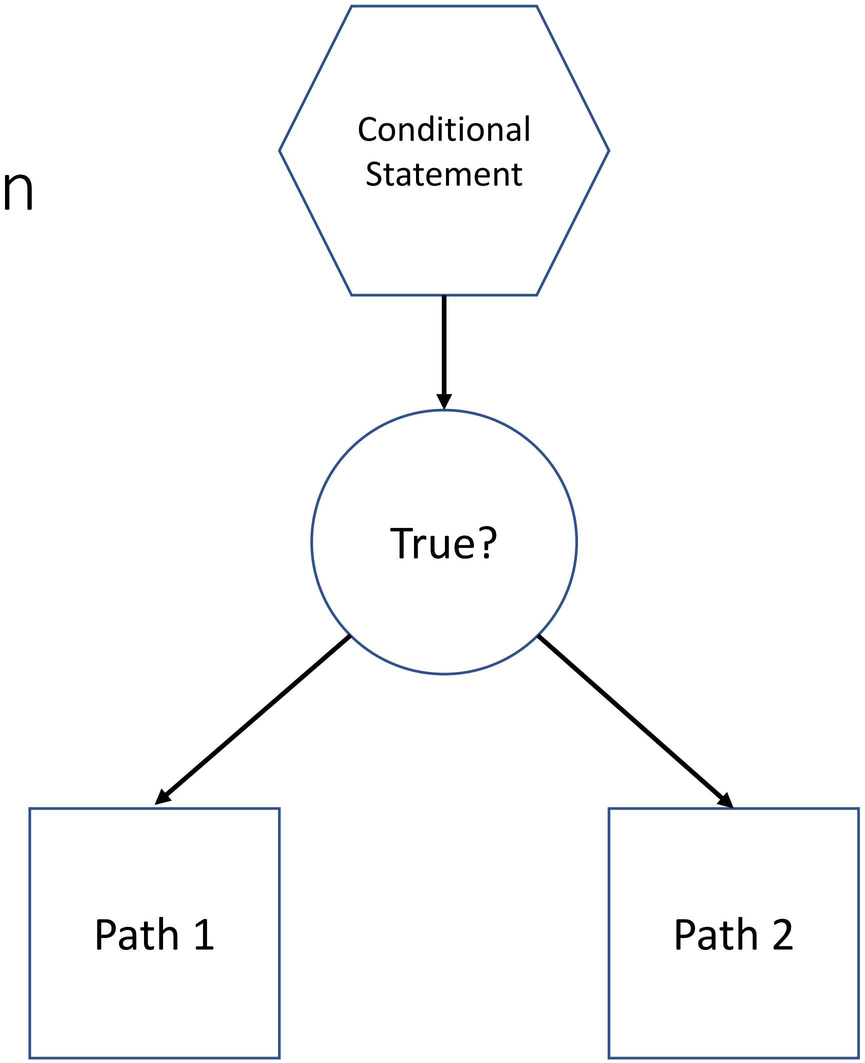
AND	and	 `print(bool1 and bool2)` `False`
NOT	not	 `print(not bool1)` `False`

Conditionals, Loops, Functions

Conditional Statements

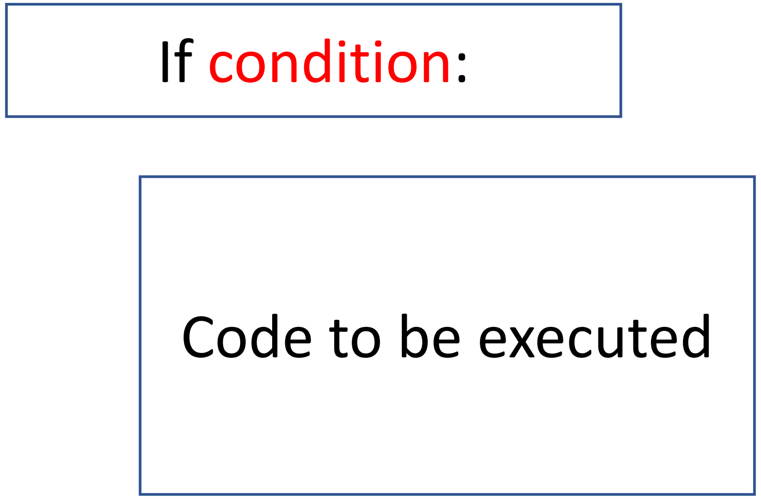
Types of conditional statements in Python

- If
- If-else
- If-elif-else



if statement

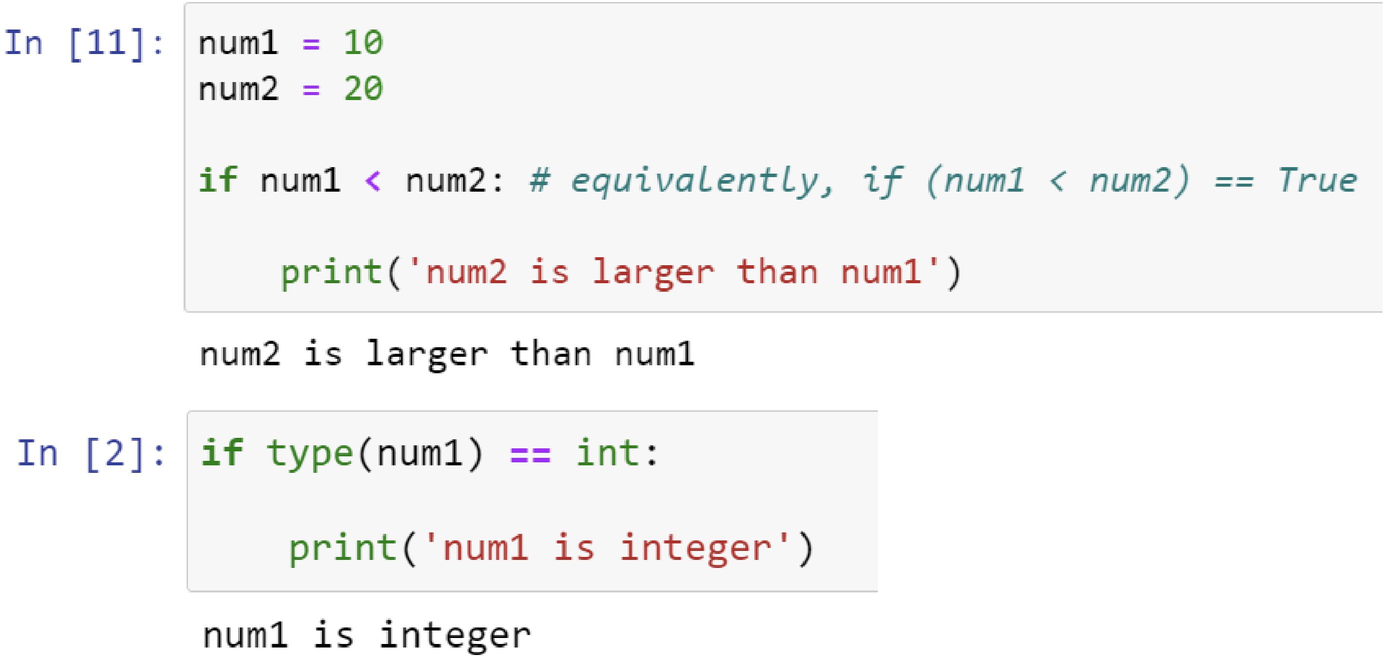
Implementation structure



If condition:

Code to be executed

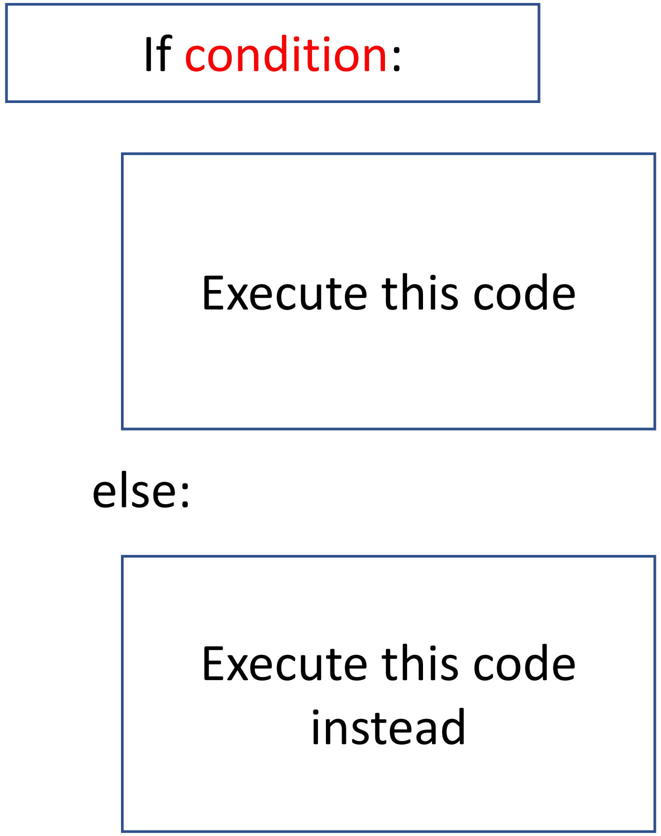
Code example



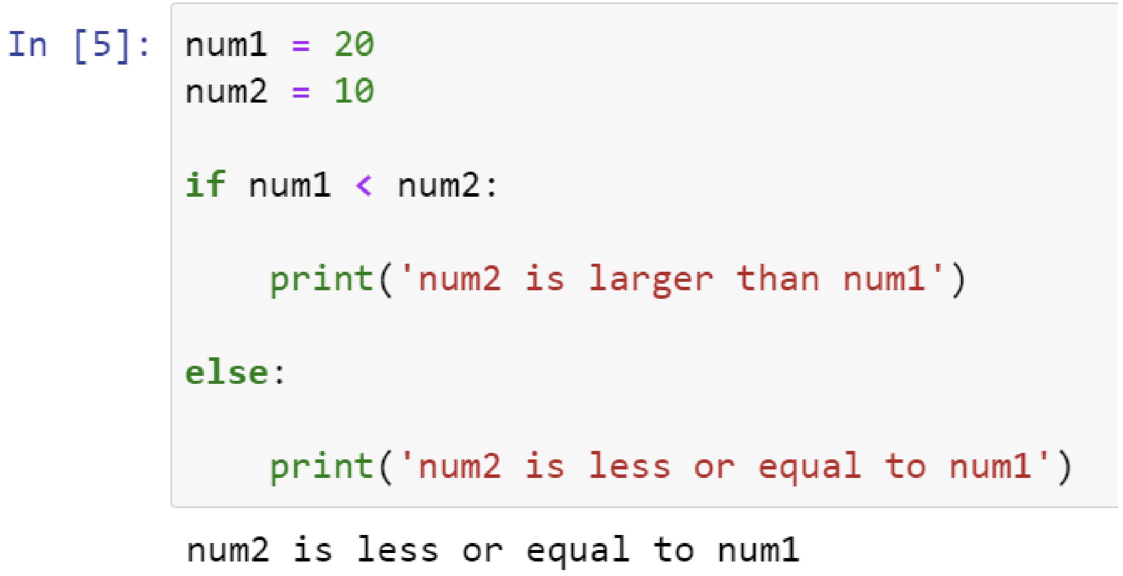
```
In [11]: num1 = 10
         num2 = 20

         if num1 < num2: # equivalently, if (num1 < num2) == True

             print('num2 is larger than num1')

         num2 is larger than num1

In [2]:  if type(num1) == int:

             print('num1 is integer')

         num1 is integer
```

If-else Statement

Implementation structure



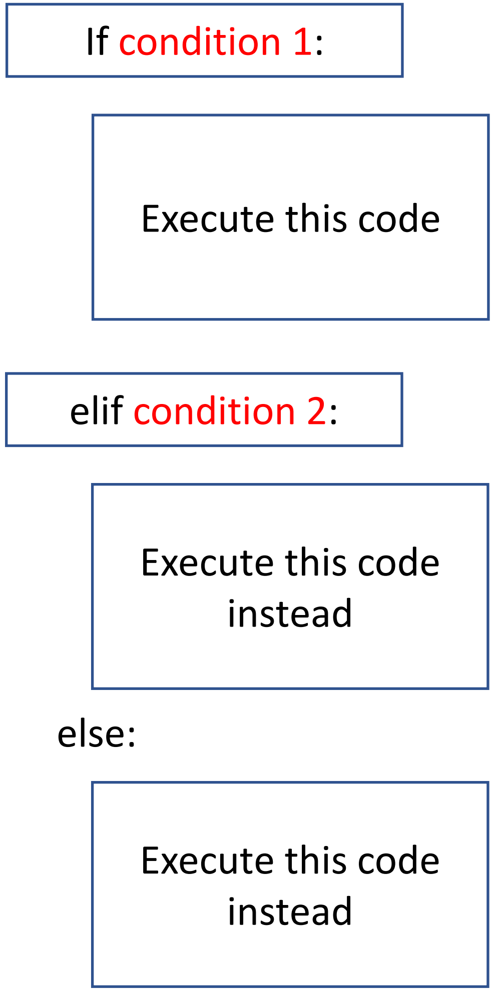
Code example



```
In [5]: num1 = 20
        num2 = 10

        if num1 < num2:

            print('num2 is larger than num1')

        else:

            print('num2 is less or equal to num1')

        num2 is less or equal to num1
```

If-elif-else Statement

Implementation structure



If condition 1:

Execute this code

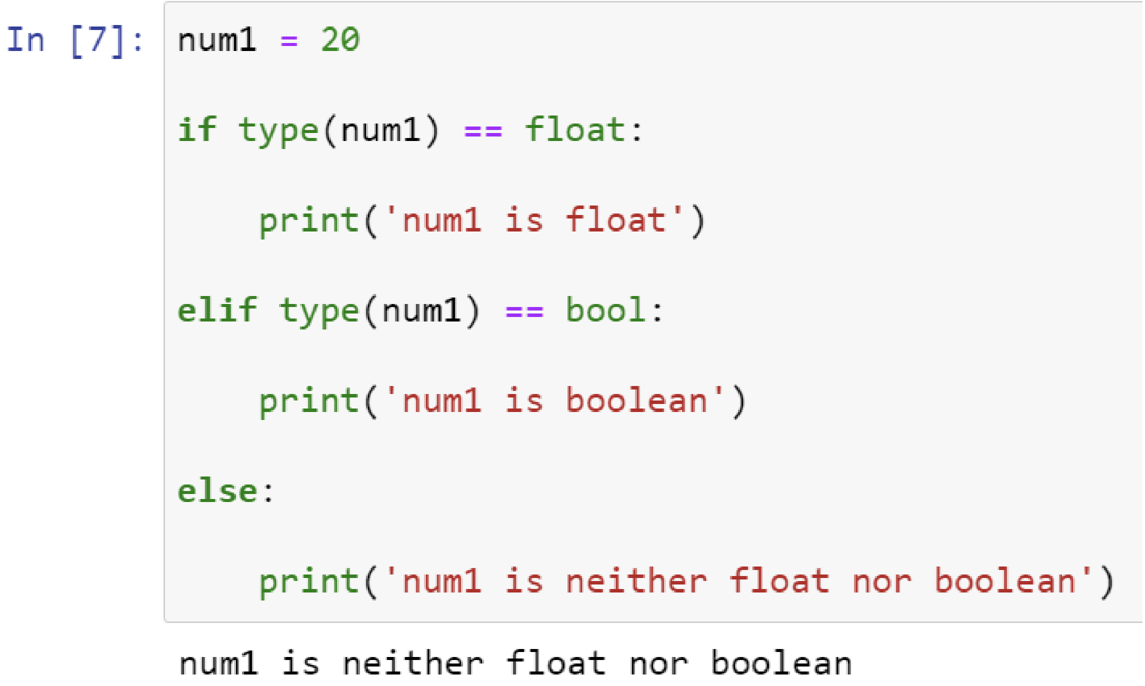
elif condition 2:

Execute this code instead

else:

Execute this code instead

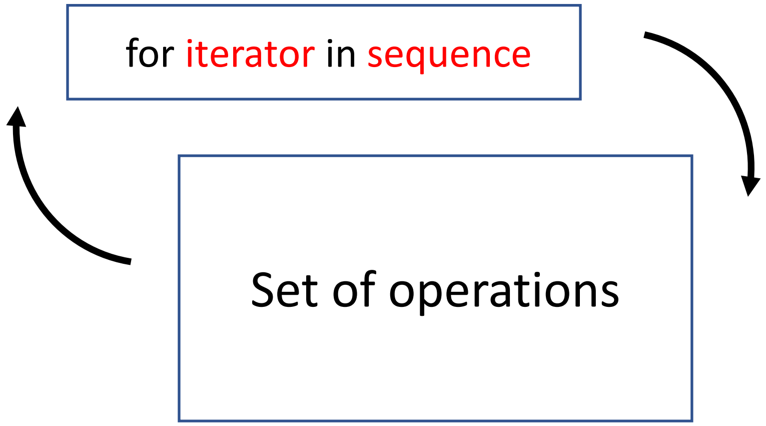
Code example

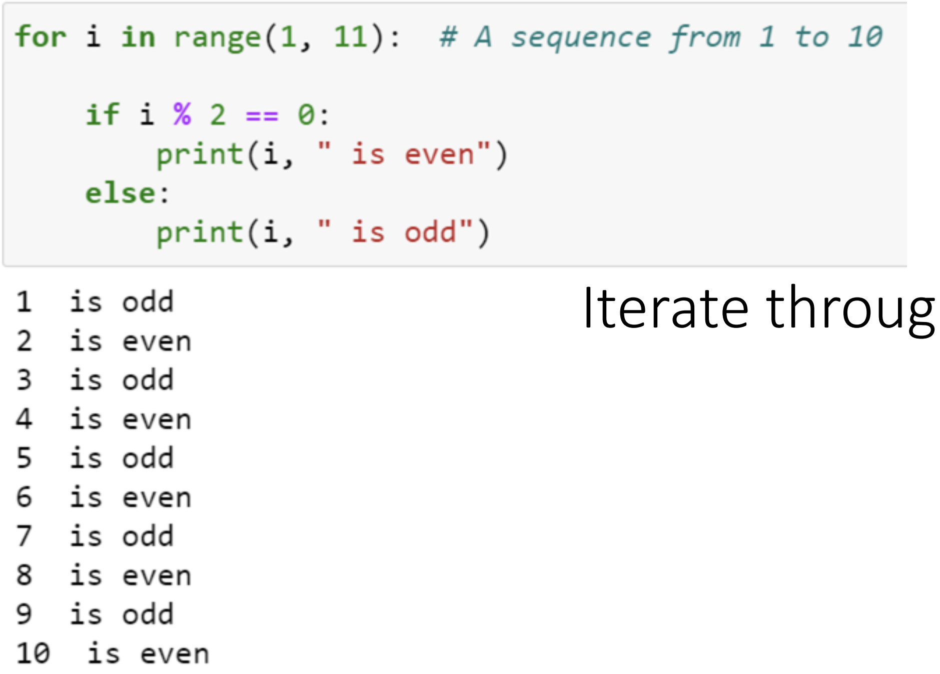


```
In [7]: num1 = 20

        if type(num1) == float:

            print('num1 is float')

        elif type(num1) == bool:

            print('num1 is boolean')

        else:

            print('num1 is neither float nor boolean')

num1 is neither float nor boolean
```

Note: You can have multiple elif conditions between if and else

for Loop

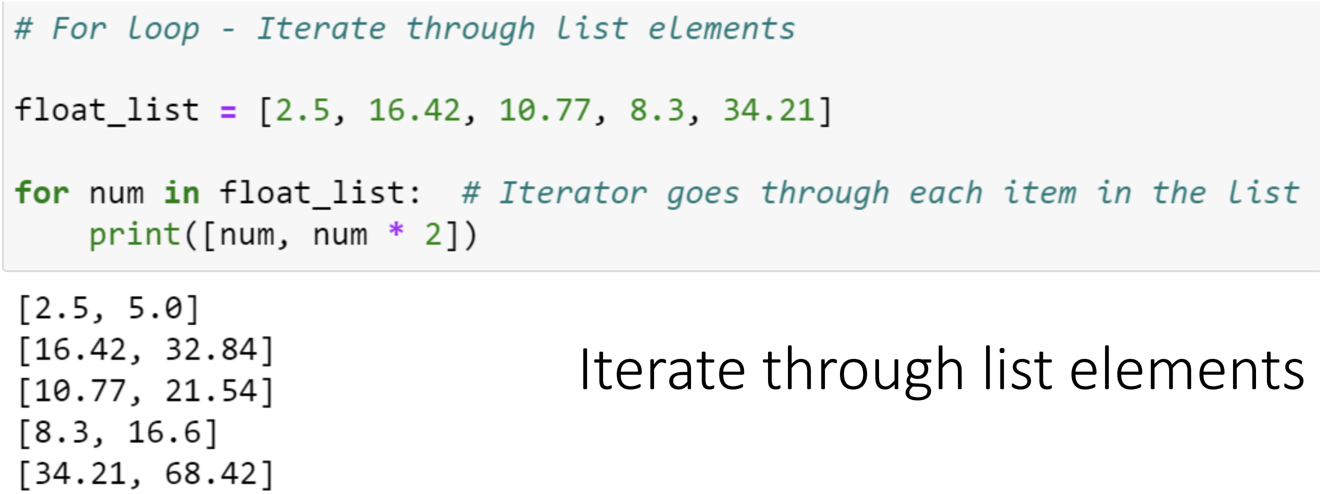




```
for i in range(1, 11):  # A sequence from 1 to 10

    if i % 2 == 0:
        print(i, " is even")
    else:
        print(i, " is odd")
```

```
1  is odd
2  is even
3  is odd
4  is even
5  is odd
6  is even
7  is odd
8  is even
9  is odd
10  is even
```

Iterate through sequence

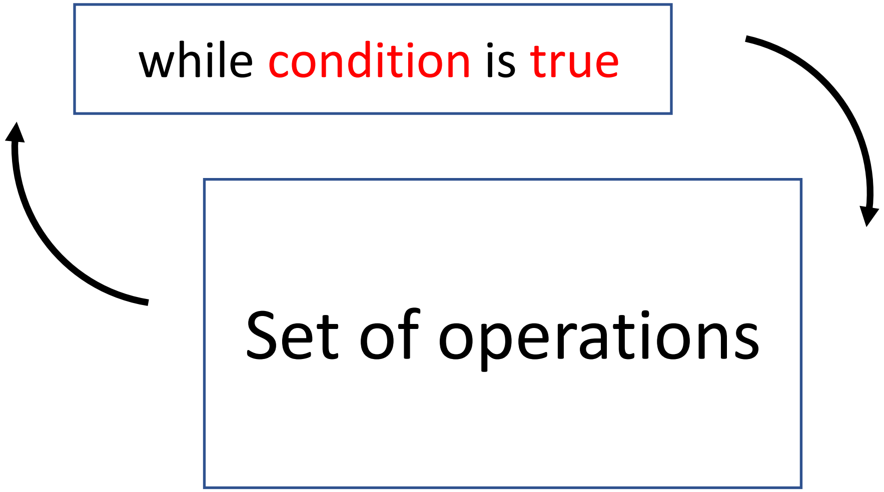


```
# For loop - Iterate through list elements

float_list = [2.5, 16.42, 10.77, 8.3, 34.21]

for num in float_list:  # Iterator goes through each item in the list
    print([num, num * 2])
```

```
[2.5, 5.0]
[16.42, 32.84]
[10.77, 21.54]
[8.3, 16.6]
[34.21, 68.42]
```

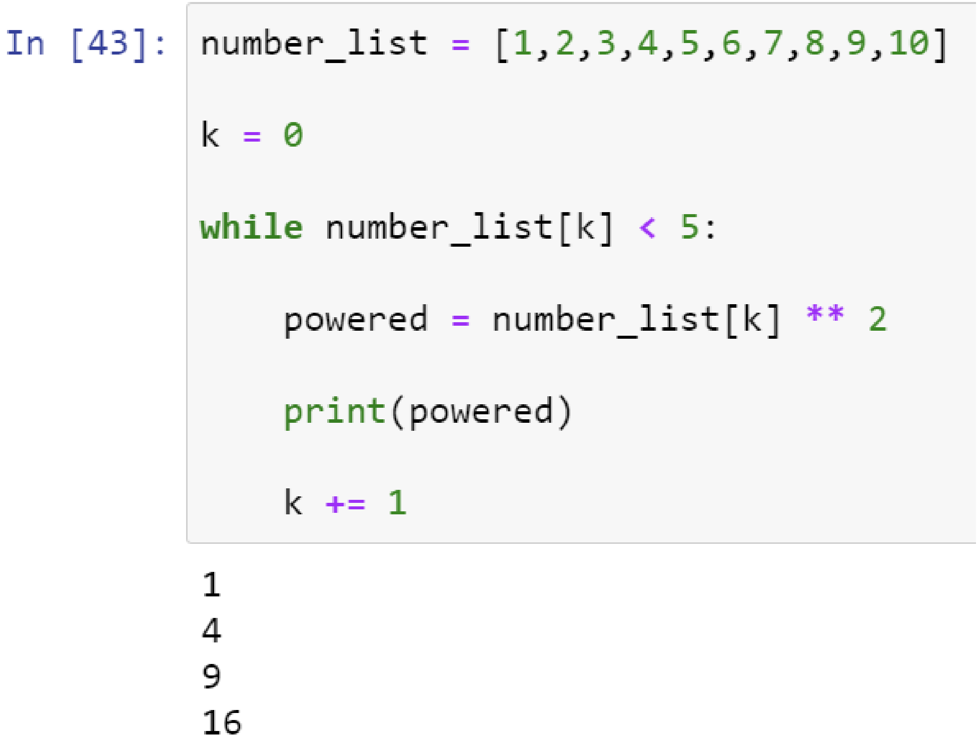
Iterate through list elements

while Loop

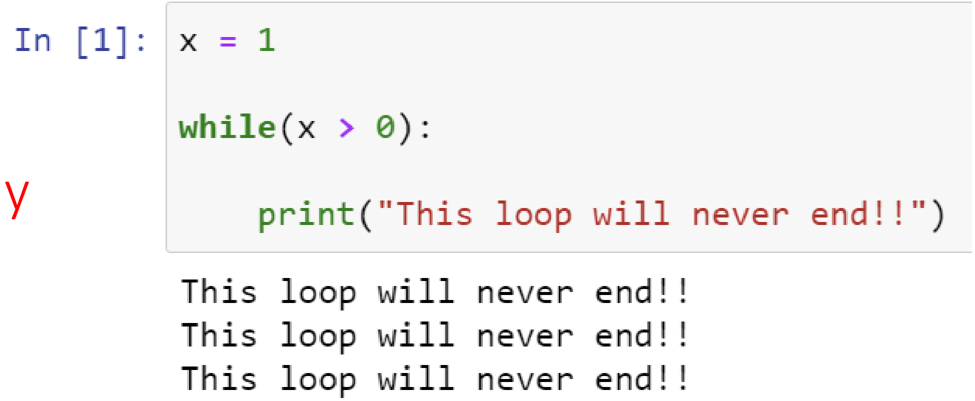


while condition is true

Set of operations



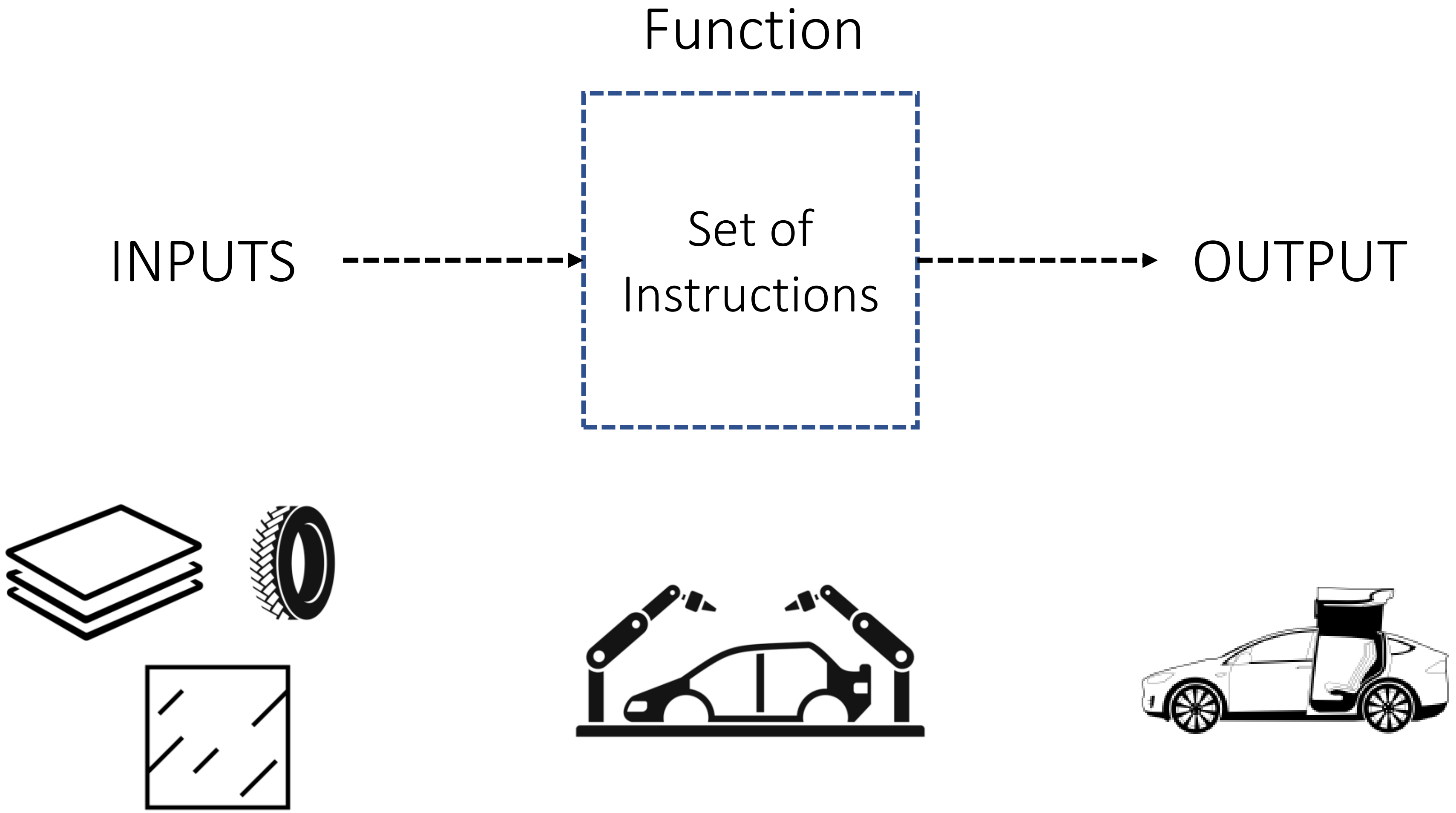
```
In [43]: number_list = [1,2,3,4,5,6,7,8,9,10]

         k = 0

         while number_list[k] < 5:

             powered = number_list[k] ** 2

             print(powered)

             k += 1

1
4
9
16
```



```
In [1]: x = 1

        while(x > 0):

            print("This loop will never end!!")

This loop will never end!!
This loop will never end!!
This loop will never end!!
```

Note: while loop has a potential to run infinitely if not set correctly

Functions



Function

INPUTS

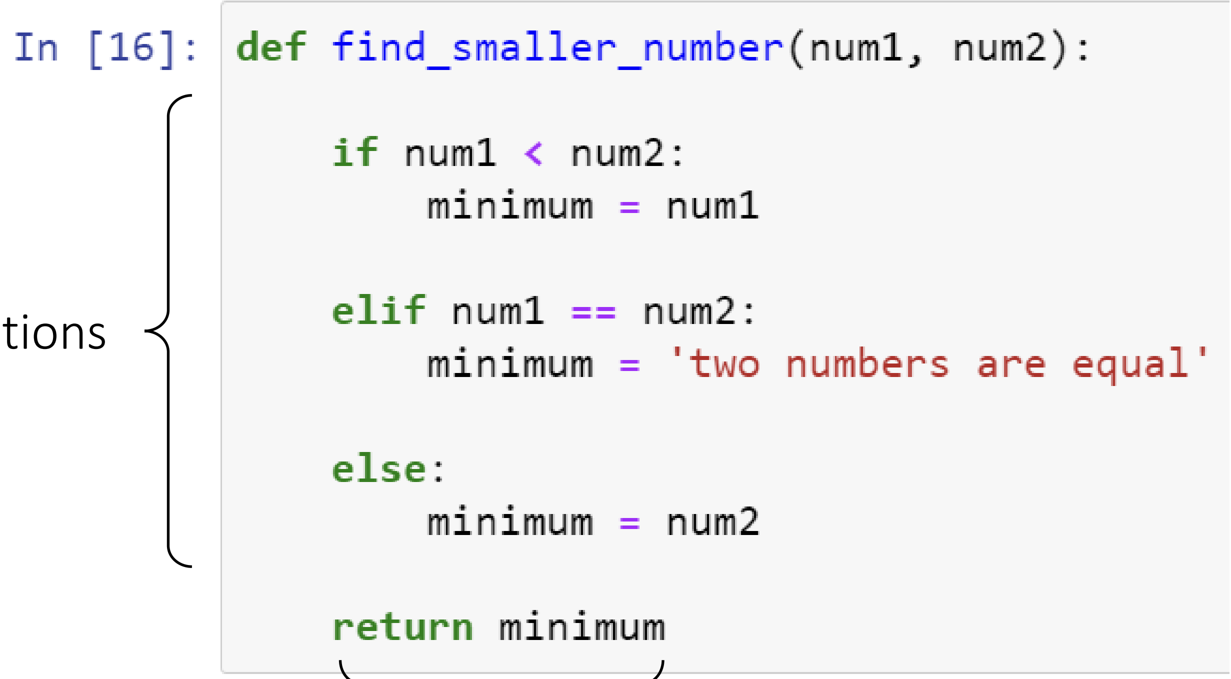
Set of Instructions

OUTPUT

Defining Functions

Define function name

Input parameters



```
In [16]: def find_smaller_number(num1, num2):

             if num1 < num2:
                 minimum = num1

             elif num1 == num2:
                 minimum = 'two numbers are equal'

             else:
                 minimum = num2

             return minimum
```

Set of instructions

Return output

Note: ‘return’ is NOT required for defining a function

PART 3:

NUMPY AND PLOTTING

Introduction to NumPy

Plotting with Matplotlib

Introduction to NumPy

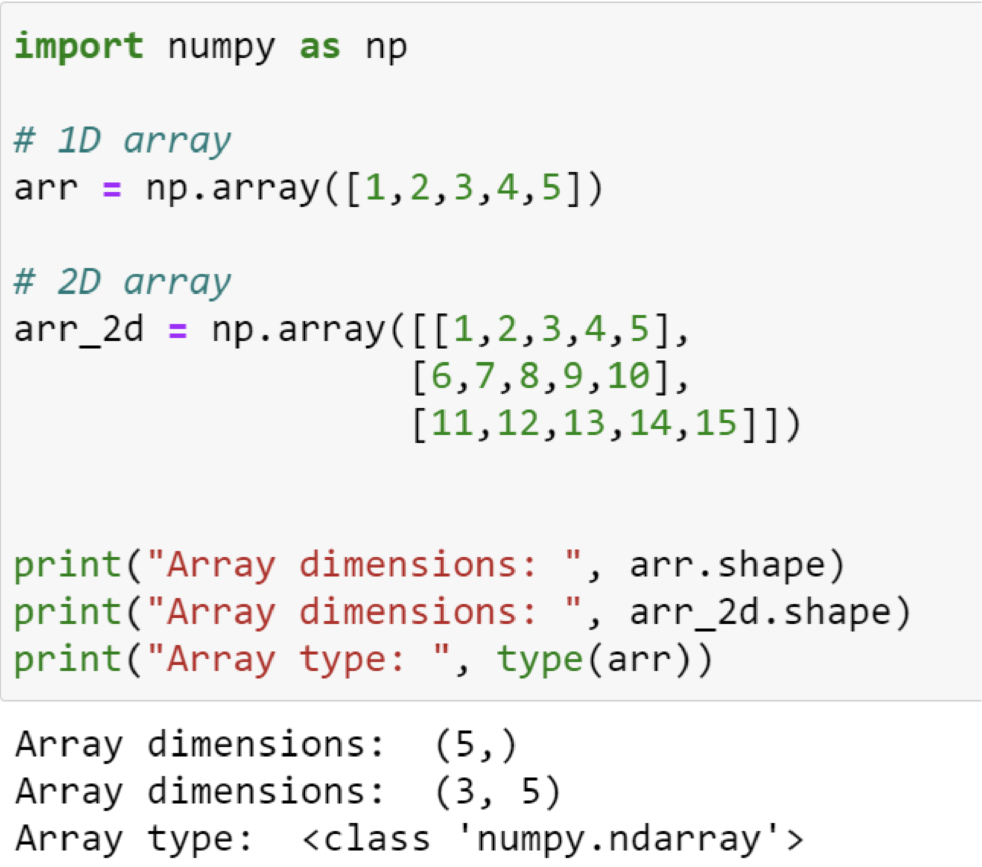
What is NumPy?

Fundamental package for scientific computing in Python

- Supports multi-dimensional array object
- Provides assortment of mathematical routines for arrays
- Fast array operations through pre-compiled C
- Support array-wide broadcasting for operations
- Included in Anaconda 3

Constructing NumPy Arrays

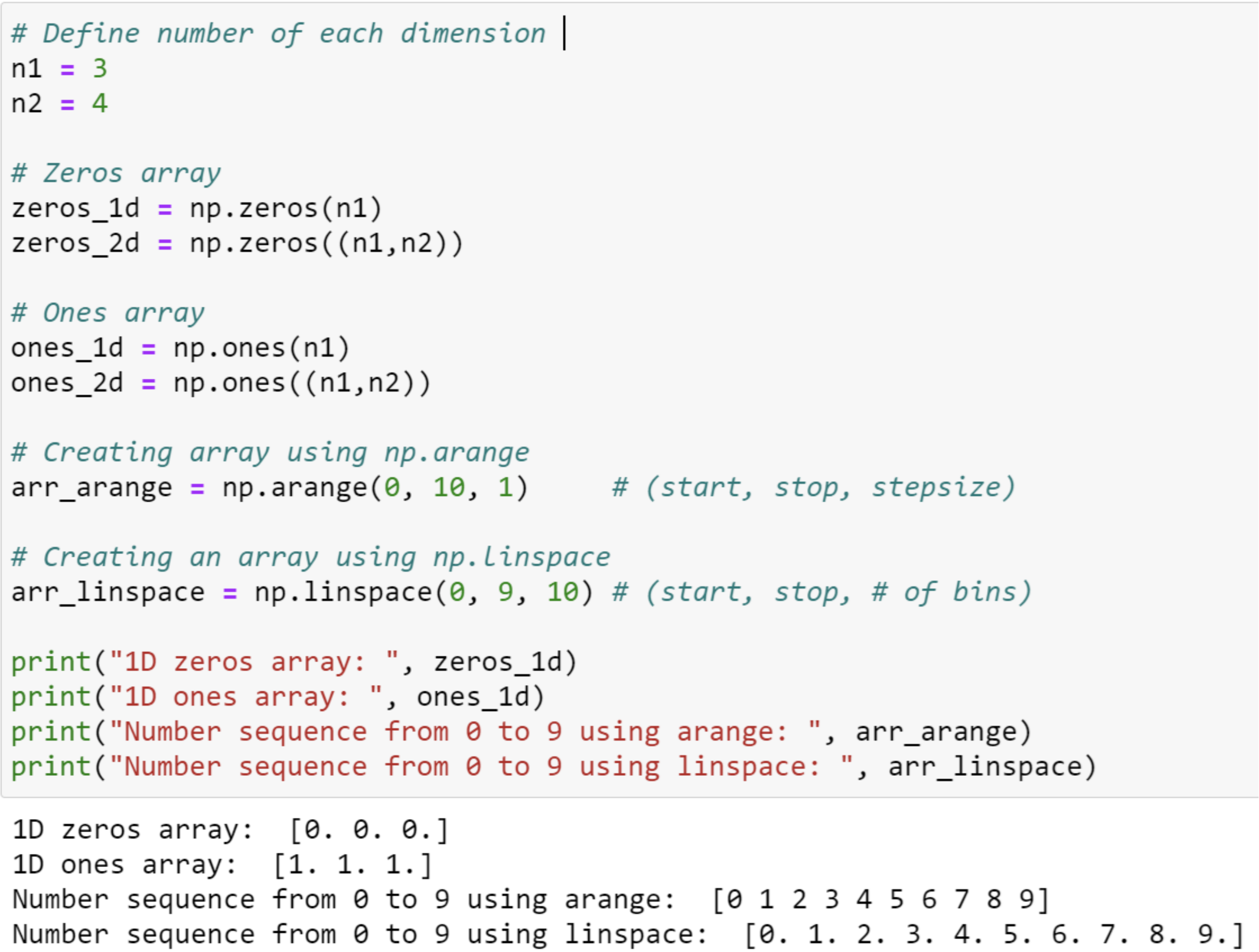
From Python lists



```
import numpy as np

# 1D array
arr = np.array([1,2,3,4,5])

# 2D array
arr_2d = np.array([[1,2,3,4,5],
                   [6,7,8,9,10],
                   [11,12,13,14,15]])


print("Array dimensions: ", arr.shape)
print("Array dimensions: ", arr_2d.shape)
print("Array type: ", type(arr))
```

```
Array dimensions:  (5,)
Array dimensions:  (3, 5)
Array type:  <class 'numpy.ndarray'>
```

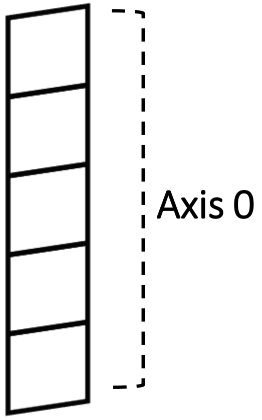
From Numpy commands

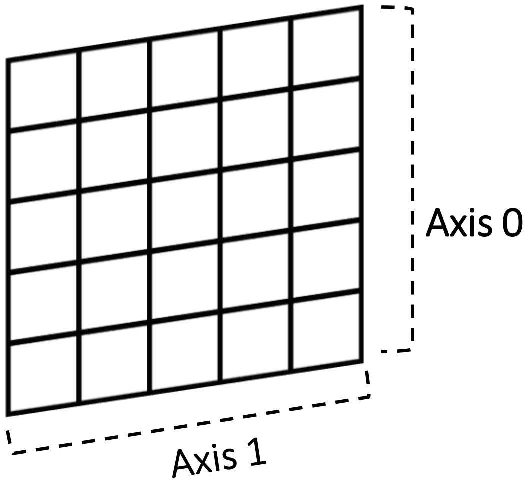


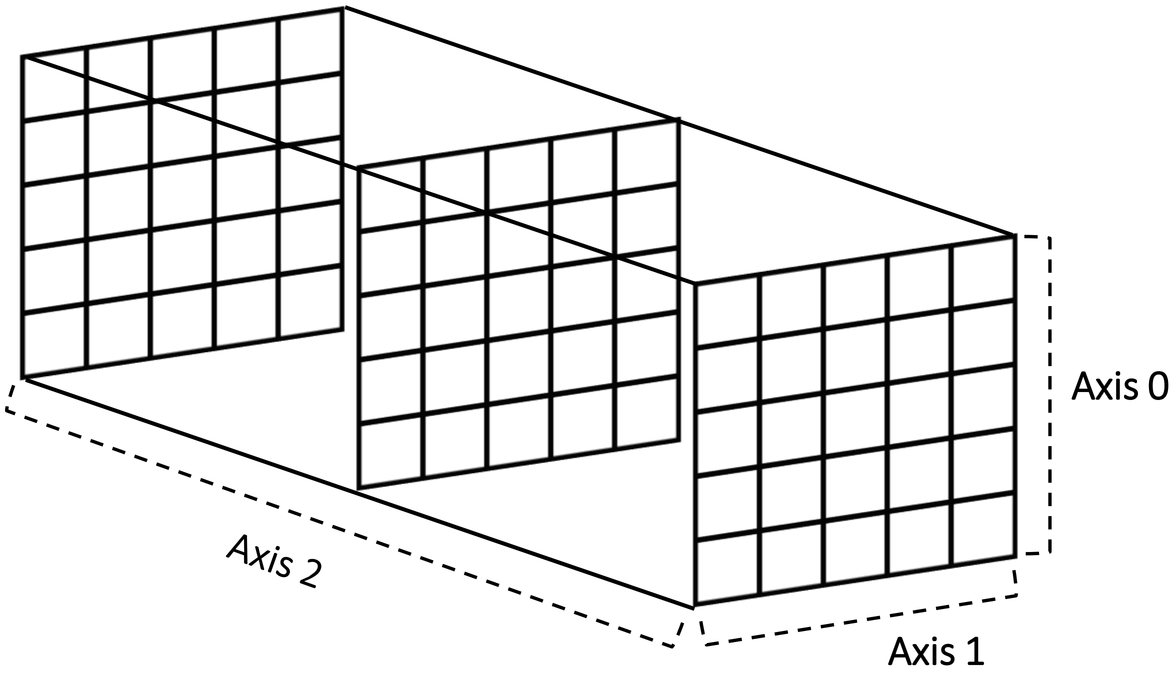
```
# Define number of each dimension
n1 = 3
n2 = 4

# Zeros array
zeros_1d = np.zeros(n1)
zeros_2d = np.zeros((n1,n2))

# Ones array
ones_1d = np.ones(n1)
ones_2d = np.ones((n1,n2))

# Creating array using np.arange
arr_arange = np.arange(0, 10, 1)     # (start, stop, stepsize)

# Creating an array using np.linspace
arr_linspace = np.linspace(0, 9, 10) # (start, stop, # of bins)

print("1D zeros array: ", zeros_1d)
print("1D ones array: ", ones_1d)
print("Number sequence from 0 to 9 using arange: ", arr_arange)
print("Number sequence from 0 to 9 using linspace: ", arr_linspace)
```

```
1D zeros array:  [0. 0. 0.]
1D ones array:  [1. 1. 1.]
Number sequence from 0 to 9 using arange:  [0 1 2 3 4 5 6 7 8 9]
Number sequence from 0 to 9 using linspace:  [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

Data Structures as Numpy Arrays







1-D

Shape = (i,)

e.g. time series data

2-D

Shape = (i,j)

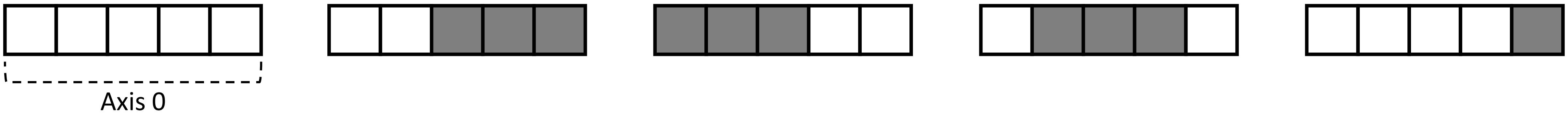
e.g. data frame, table, greyscale image

3-D

Shape = (i,j,k)

e.g. RGB color image

Slicing Arrays (1D)



arr

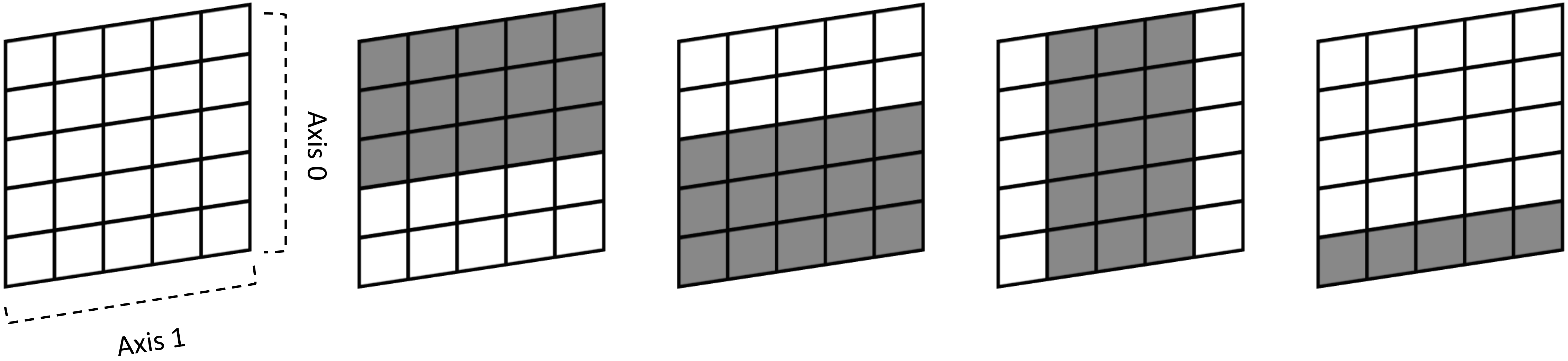
arr[2:]
=
arr[-3:]

arr[:3]
=
arr[:-2]

arr[1:4]
=
arr[-4:-1]

arr[4]
=
arr[-1]

Slicing Arrays (2D)



arr

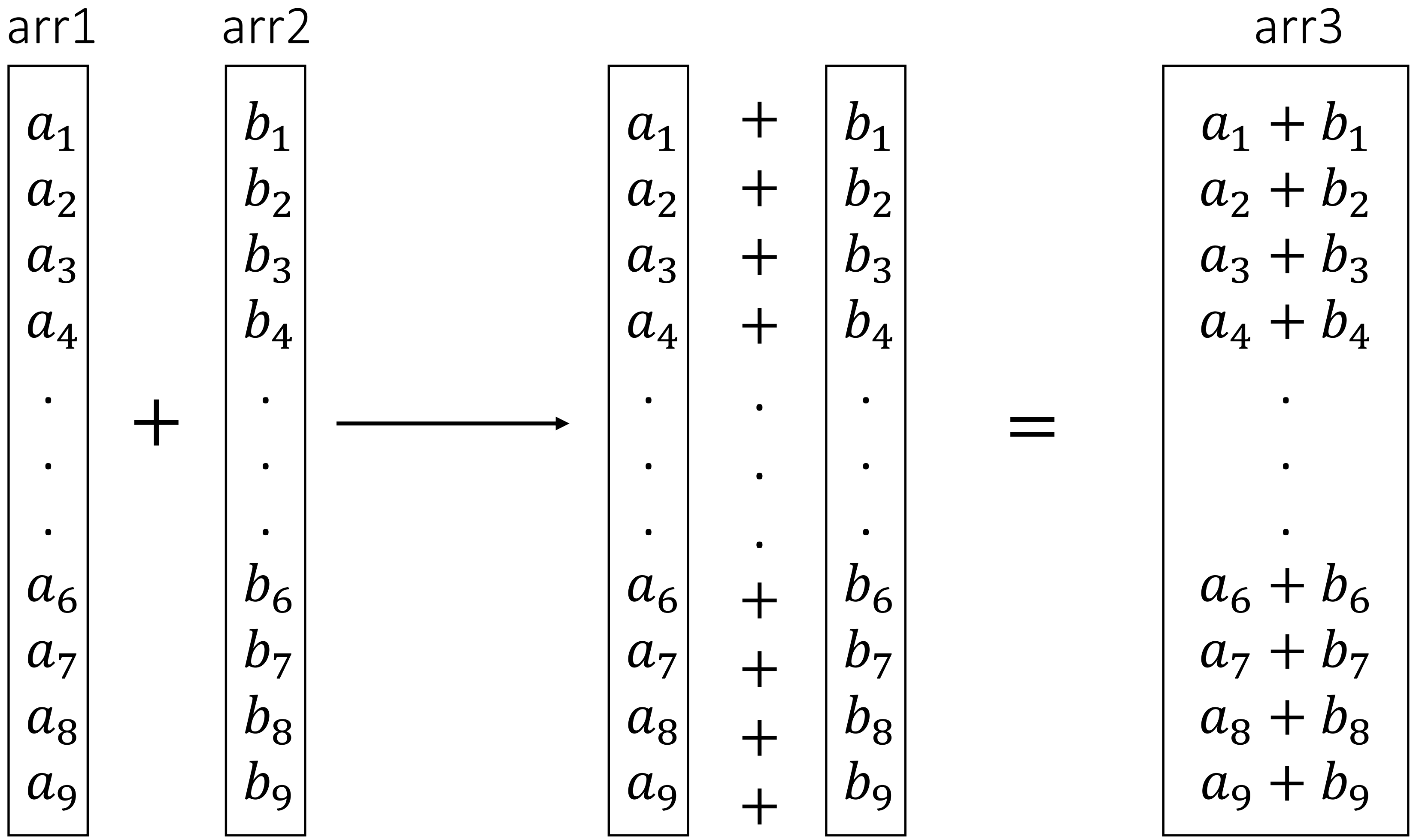
arr[:3]
||
arr[:3, :]
||
arr[:-2]
||
arr[:-2, :]

arr[2:]
||
arr[2:, :]
||
arr[-3:]
||
arr[-3:, :]

arr[:, 1:4]
||
arr[:, -4:-1]

arr[4]
||
arr[4, :]
||
arr[-1]
||
arr[-1, :]

Array-wide Operations in NumPy



$$\begin{array}{c} \text{arr1} \\ \boxed{\begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ . \\ . \\ . \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{matrix}} \end{array} + \begin{array}{c} \text{arr2} \\ \boxed{\begin{matrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ . \\ . \\ . \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{matrix}} \end{array} \longrightarrow \boxed{\begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ . \\ . \\ . \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{matrix}} \begin{matrix} + \\ + \\ + \\ + \\ . \\ . \\ . \\ + \\ + \\ + \\ + \end{matrix} \boxed{\begin{matrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ . \\ . \\ . \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{matrix}} = \begin{array}{c} \text{arr3} \\ \boxed{\begin{matrix} a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \\ a_4 + b_4 \\ . \\ . \\ . \\ a_6 + b_6 \\ a_7 + b_7 \\ a_8 + b_8 \\ a_9 + b_9 \end{matrix}} \end{array}$$

numpy.add(arr1, arr2)

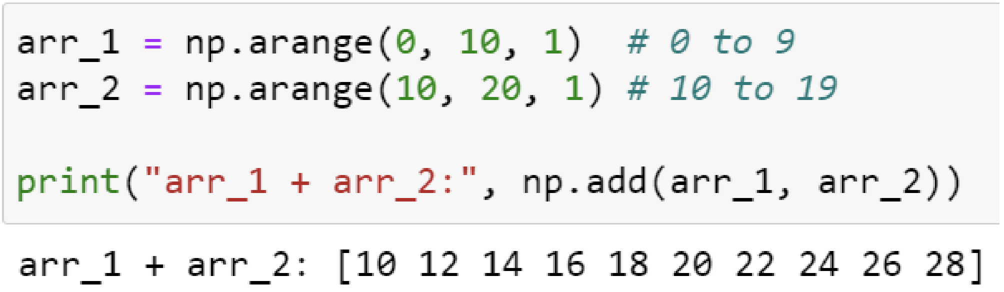
NumPy Arithmetic Operators

Operator

Example

Addition

np.add()

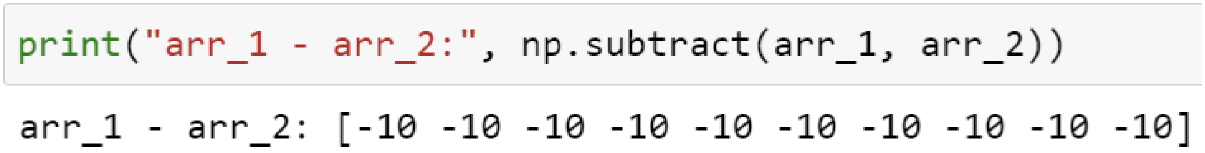


```
arr_1 = np.arange(0, 10, 1)  # 0 to 9
arr_2 = np.arange(10, 20, 1) # 10 to 19

print("arr_1 + arr_2:", np.add(arr_1, arr_2))
```

```
arr_1 + arr_2: [10 12 14 16 18 20 22 24 26 28]
```

Subtraction

np.subtract()

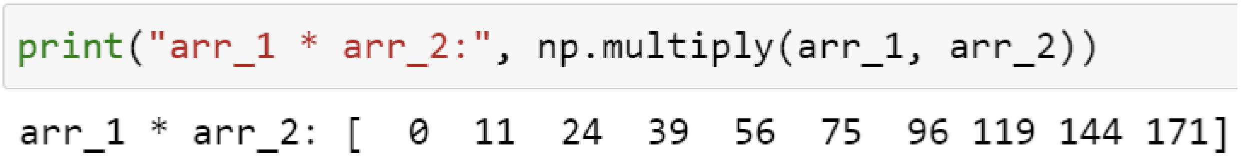


```
print("arr_1 - arr_2:", np.subtract(arr_1, arr_2))
```

```
arr_1 - arr_2: [-10 -10 -10 -10 -10 -10 -10 -10 -10 -10]
```

Multiplication

np.multiply()



```
print("arr_1 * arr_2:", np.multiply(arr_1, arr_2))
```

```
arr_1 * arr_2: [  0  11  24  39  56  75  96 119 144 171]
```

Note: The syntax assumes “import numpy as np”

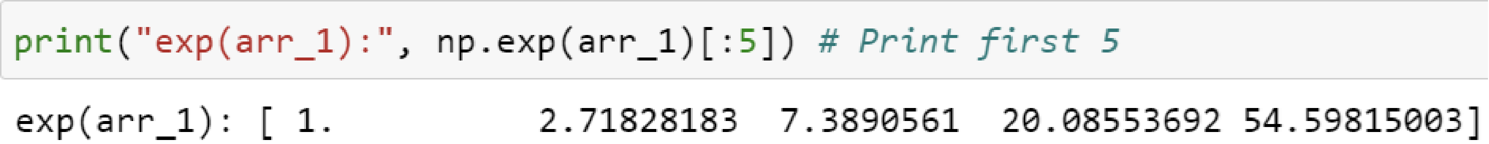
NumPy Arithmetic Operators

Operator

Example

Exponent

np.exp()

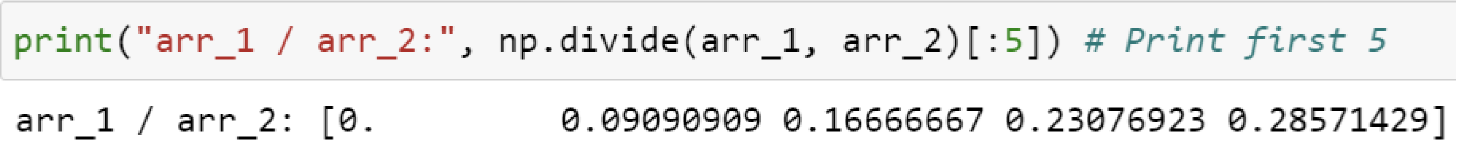


```
print("exp(arr_1):", np.exp(arr_1)[:5]) # Print first 5
```

```
exp(arr_1): [ 1.          2.71828183  7.3890561  20.08553692 54.59815003]
```

Division

np.divide()

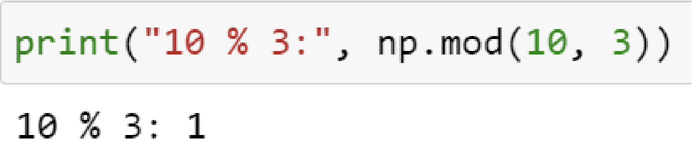


```
print("arr_1 / arr_2:", np.divide(arr_1, arr_2)[:5]) # Print first 5
```

```
arr_1 / arr_2: [0.         0.09090909 0.16666667 0.23076923 0.28571429]
```

Modulo

np.mod()



```
print("10 % 3:", np.mod(10, 3))
```

```
10 % 3: 1
```

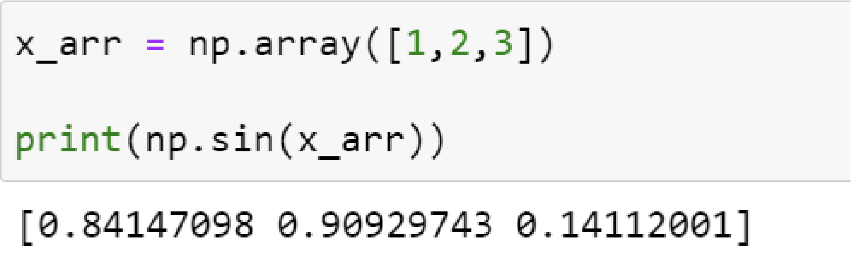
Math Operators

Operator

Example

Sine

np.sin(x)

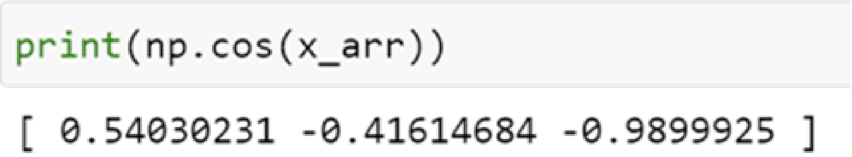


```
x_arr = np.array([1,2,3])

print(np.sin(x_arr))
```

```
[0.84147098 0.90929743 0.14112001]
```

Cosine

np.cos(x)

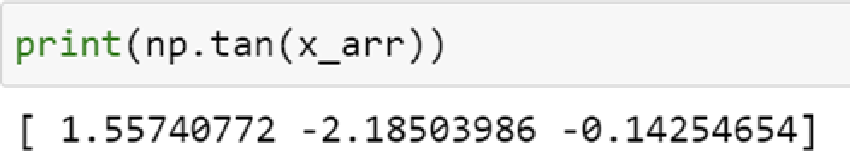


```
print(np.cos(x_arr))
```

```
[ 0.54030231 -0.41614684 -0.9899925 ]
```

Tangent

np.tan(x)

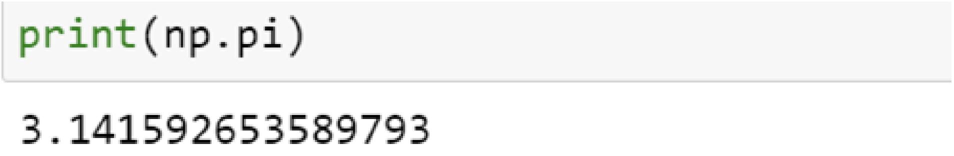
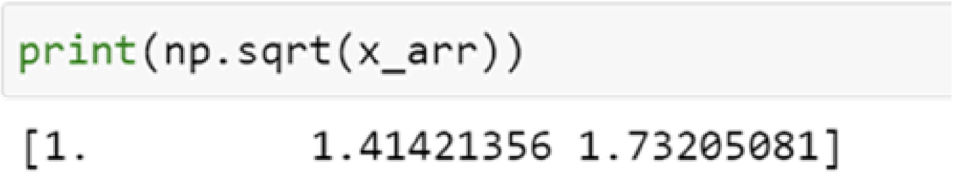


```
print(np.tan(x_arr))
```

```
[ 1.55740772 -2.18503986 -0.14254654]
```

Note: Trigonometric functions require radians as inputs

Math Operators

	Operator	Example
Pi	np.pi	
Square Root	np.sqrt(x)	

```
print(np.pi)
```

```
3.141592653589793
```

```
print(np.sqrt(x_arr))
```

```
[1.         1.41421356 1.73205081]
```

More functions: https://numpy.org/doc/stable/reference/routines.math.html

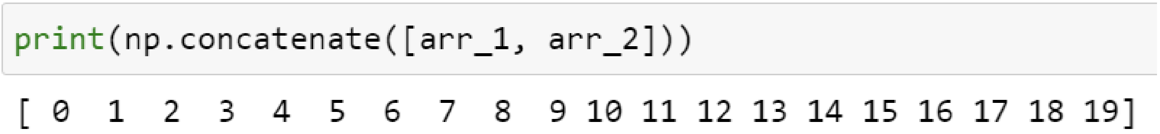
Combining Arrays

Operator

Example

Concatenation

np.concatenate()

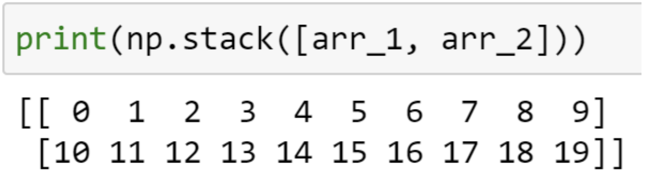


```
print(np.concatenate([arr_1, arr_2]))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Stack Dimensions

np.stack()

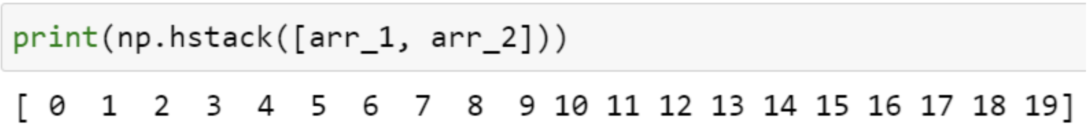


```
print(np.stack([arr_1, arr_2]))
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]]
```

Horizontal Stack

np.hstack()

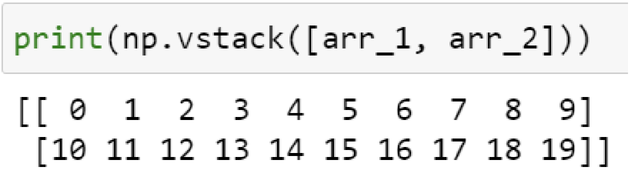


```
print(np.hstack([arr_1, arr_2]))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Vertical Stack

np.vstack()



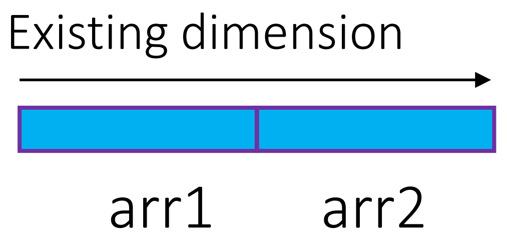
```
print(np.vstack([arr_1, arr_2]))
```

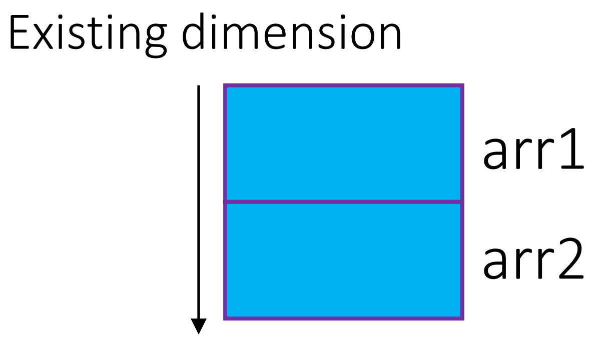
```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]]
```

Combining Arrays

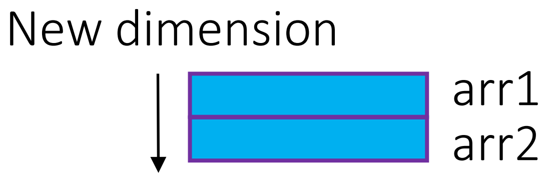
Operator | 1D | 2D

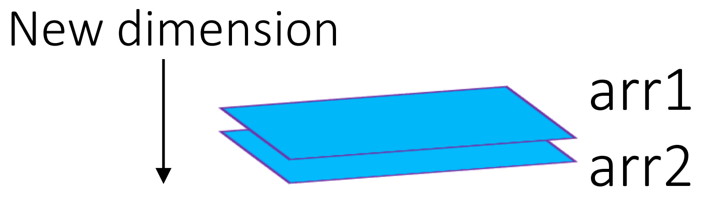
np.concatenate()





np.stack()

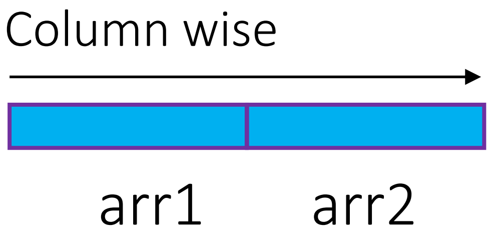


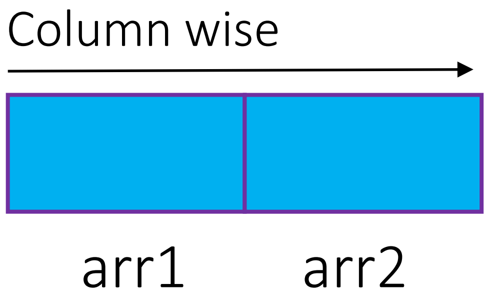


Combining Arrays

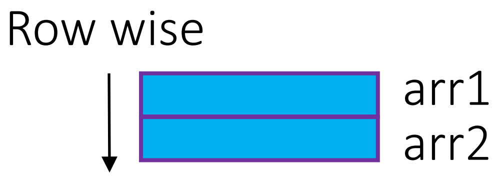
Operator | 1D | 2D

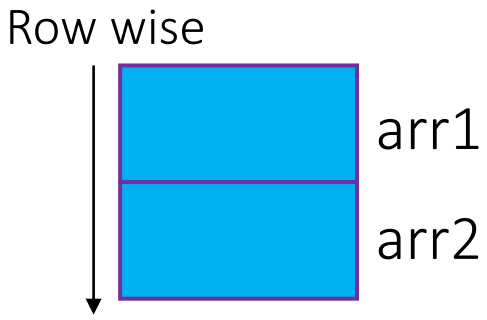
np.hstack()





np.vstack()





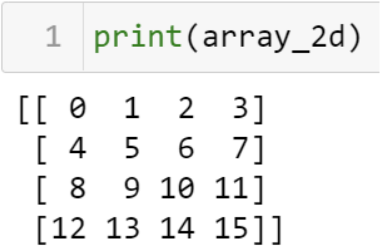
Array Splitting

Operator

Example

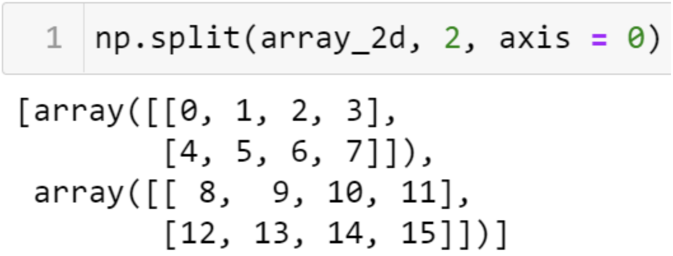
Split the array into sub-arrays (axis defines direction)

np.split()



```
print(array_2d)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

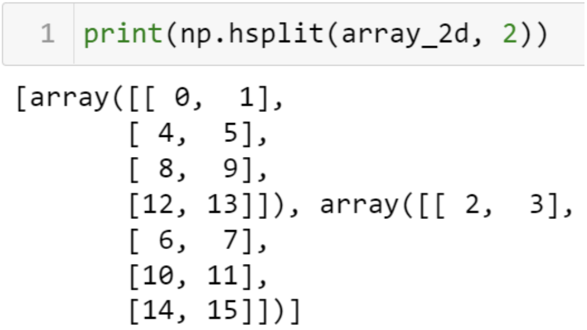


```
np.split(array_2d, 2, axis = 0)
```

```
[array([[0, 1, 2, 3],
        [4, 5, 6, 7]]),
 array([[ 8,  9, 10, 11],
        [12, 13, 14, 15]])]
```

Split the array column-wise

np.hsplit()

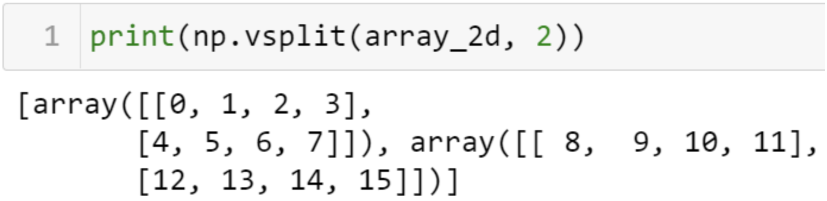


```
print(np.hsplit(array_2d, 2))
```

```
[array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]]), array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15]])]
```

Split the array row-wise

np.vsplit()



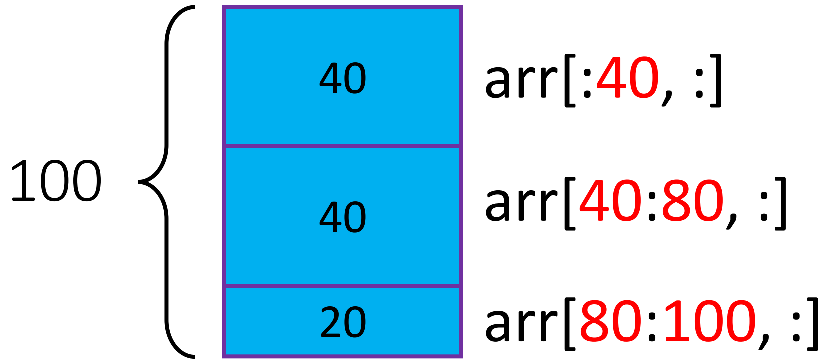
```
print(np.vsplit(array_2d, 2))
```

```
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])]
```

Array Splitting

Cumulative split indices

```
np.split(arr, [40, 80, 100], axis = 0)
```



100

40

40

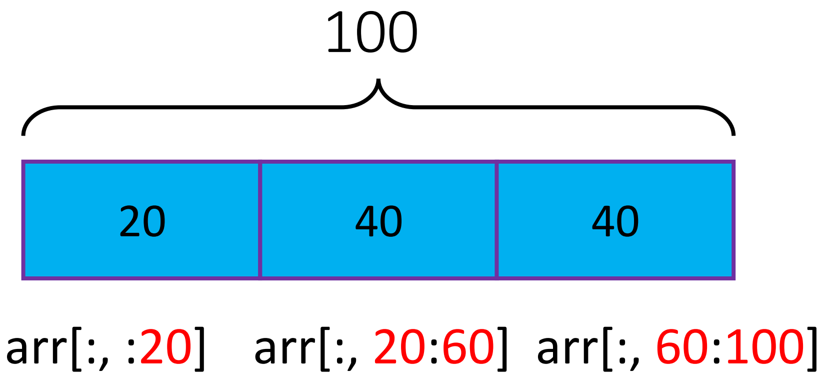
20

arr[:40, :]

arr[40:80, :]

arr[80:100, :]

```
np.split(arr, [20, 60, 100], axis = 1)
```



100

20

40

40

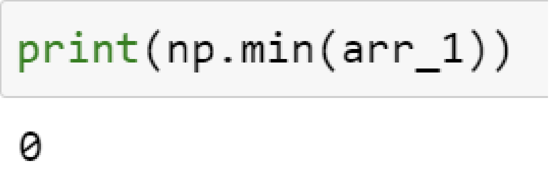
arr[:, :20]

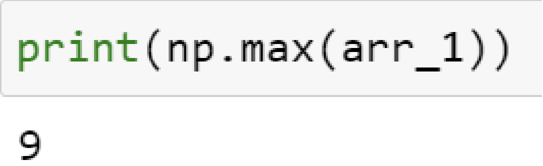
arr[:, 20:60]

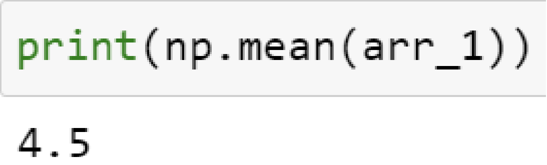
arr[:, 60:100]

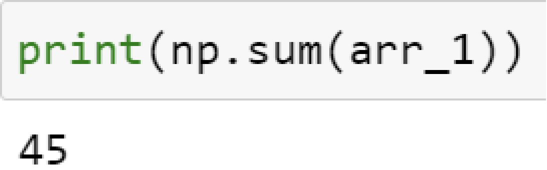
Characteristic Values of Arrays

	Operator	Example

Minimum Value	np.min()	
		`print(np.min(arr_1))`
		`0`

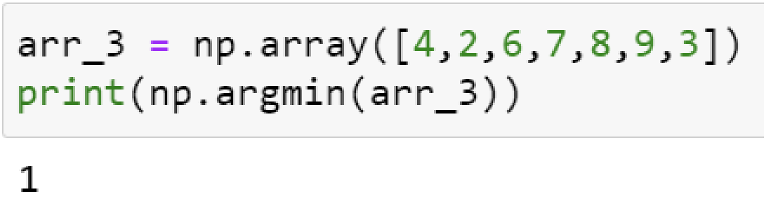
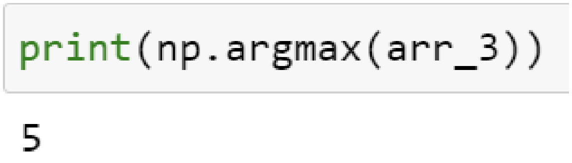
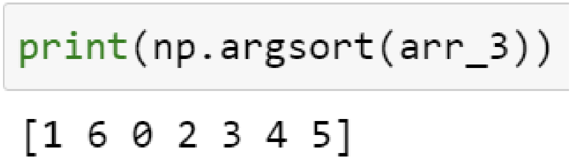
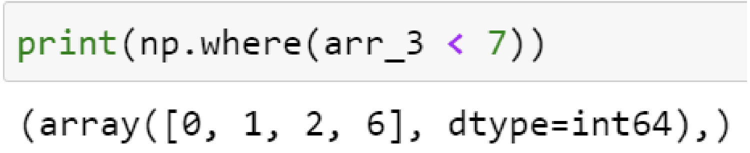
Maximum Value	np.max()	
		`print(np.max(arr_1))`
		`9`

Mean Value	np.mean()	
		`print(np.mean(arr_1))`
		`4.5`

Summed Value	np.sum()	
		`print(np.sum(arr_1))`
		`45`

Note: axis parameter allows you to compute characteristic value alongside specific axis - e.g. np.sum(arr_1, axis =0): summation along row axis.

Indexing Arrays

	Operator	Example
Minimum Value Index	np.argmin()	
Maximum Value Index	np.argmax()	
Sort Indices (low to high)	np.argsort()	
Find Indices satisfying a Condition	np.where()	

```
arr_3 = np.array([4,2,6,7,8,9,3])
print(np.argmin(arr_3))
```

```
1
```

```
print(np.argmax(arr_3))
```

```
5
```

```
print(np.argsort(arr_3))
```

```
[1 6 0 2 3 4 5]
```

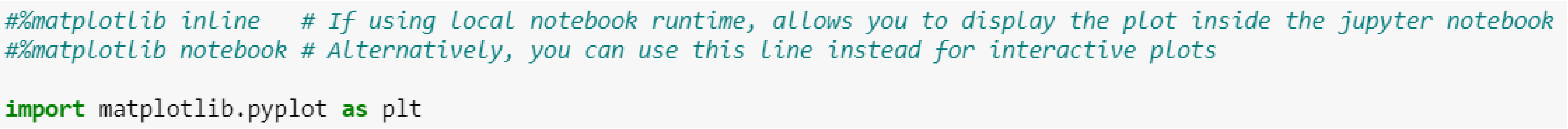
```
print(np.where(arr_3 < 7))
```

```
(array([0, 1, 2, 6], dtype=int64),)
```

Plotting with Matplotlib

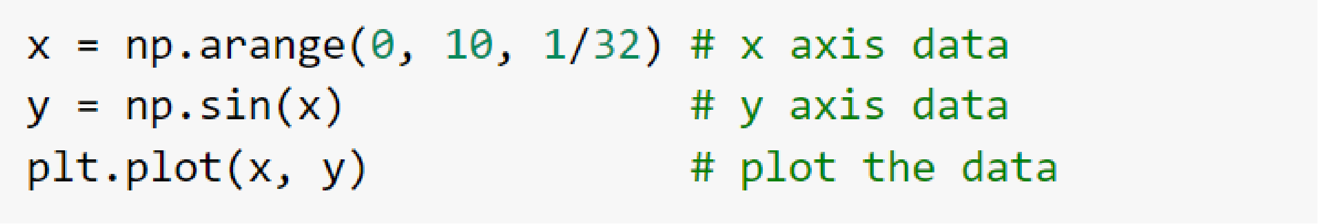
Basic Plotting with Matplotlib

Import Matplotlib



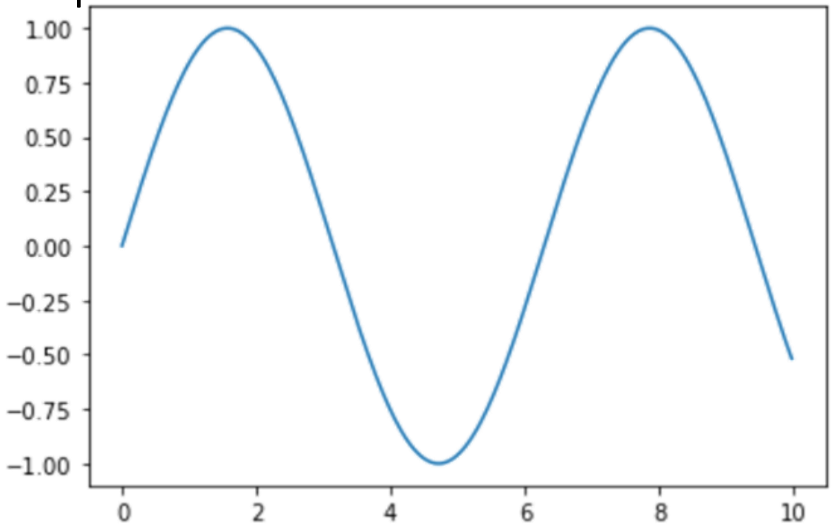
```
#%matplotlib inline   # If using local notebook runtime, allows you to display the plot inside the jupyter notebook
#%matplotlib notebook # Alternatively, you can use this line instead for interactive plots

import matplotlib.pyplot as plt
```

Code



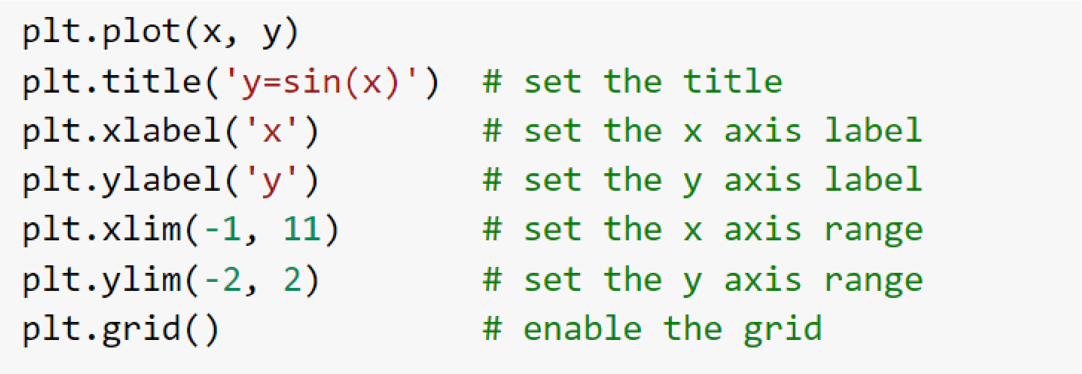
```
x = np.arange(0, 10, 1/32) # x axis data
y = np.sin(x)              # y axis data
plt.plot(x, y)             # plot the data
```

Output



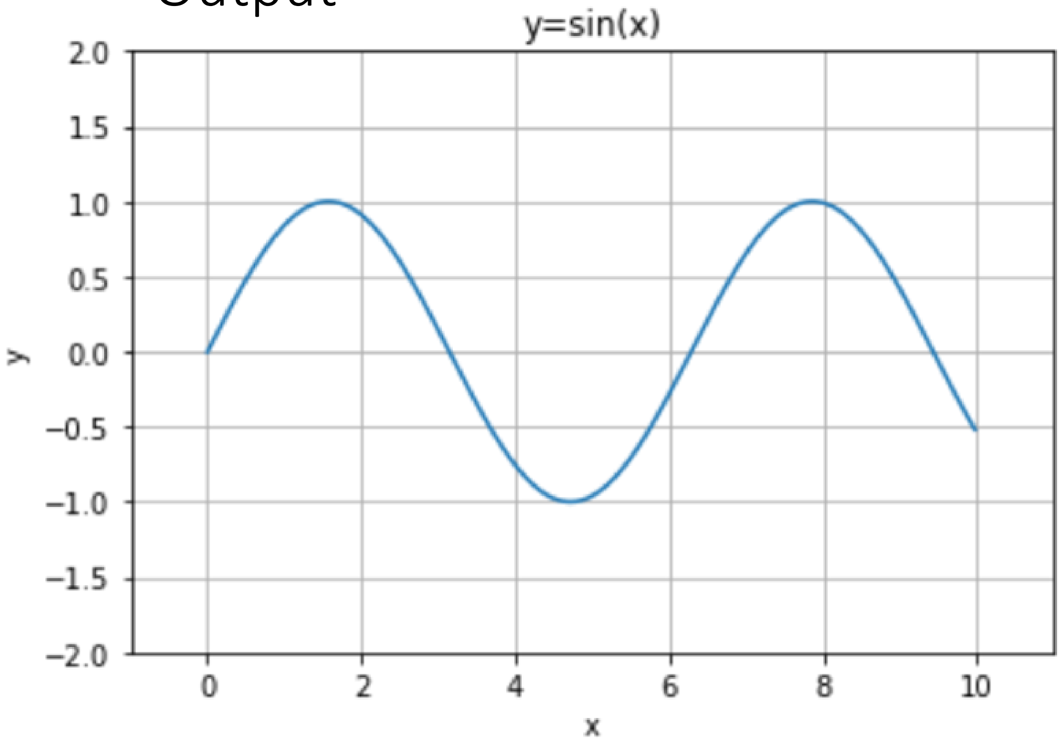
Labeling Your Plots

Code



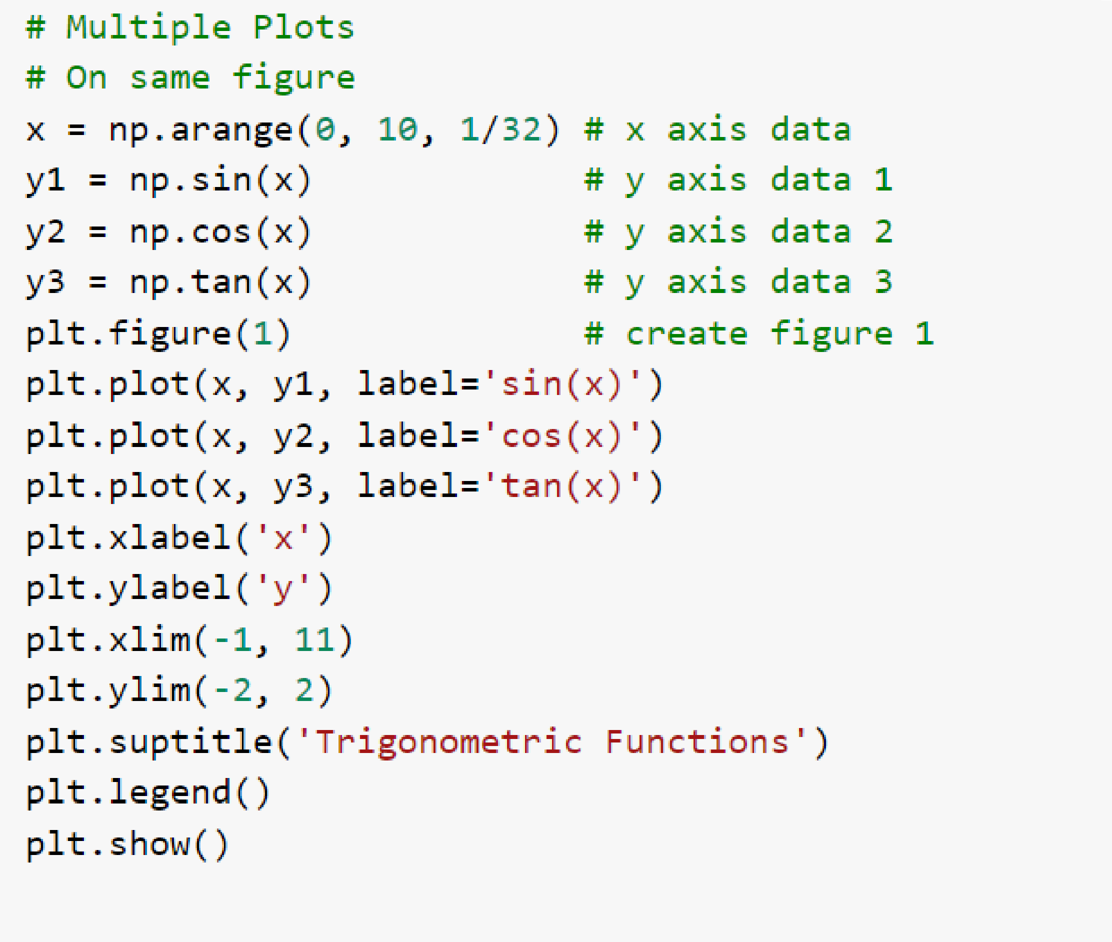
```
plt.plot(x, y)
plt.title('y=sin(x)')  # set the title
plt.xlabel('x')        # set the x axis label
plt.ylabel('y')        # set the y axis label
plt.xlim(-1, 11)       # set the x axis range
plt.ylim(-2, 2)        # set the y axis range
plt.grid()             # enable the grid
```

Output



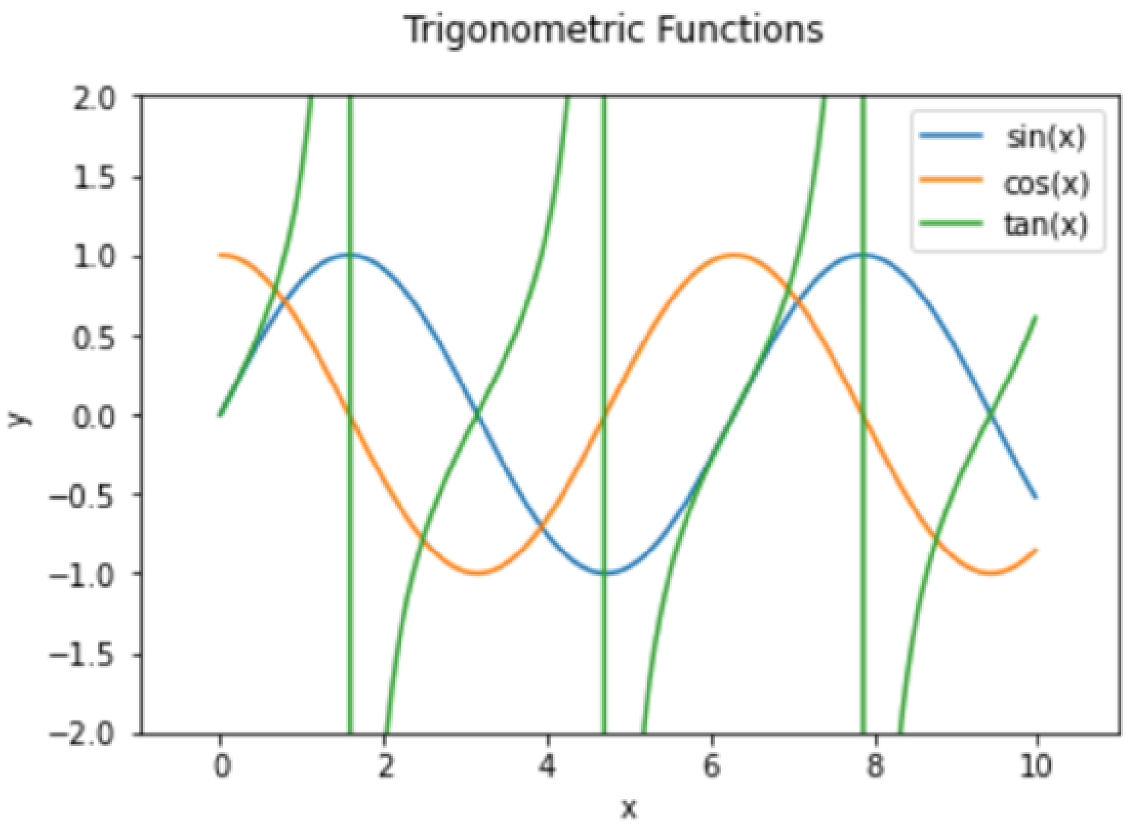
Multiple Plots

Code



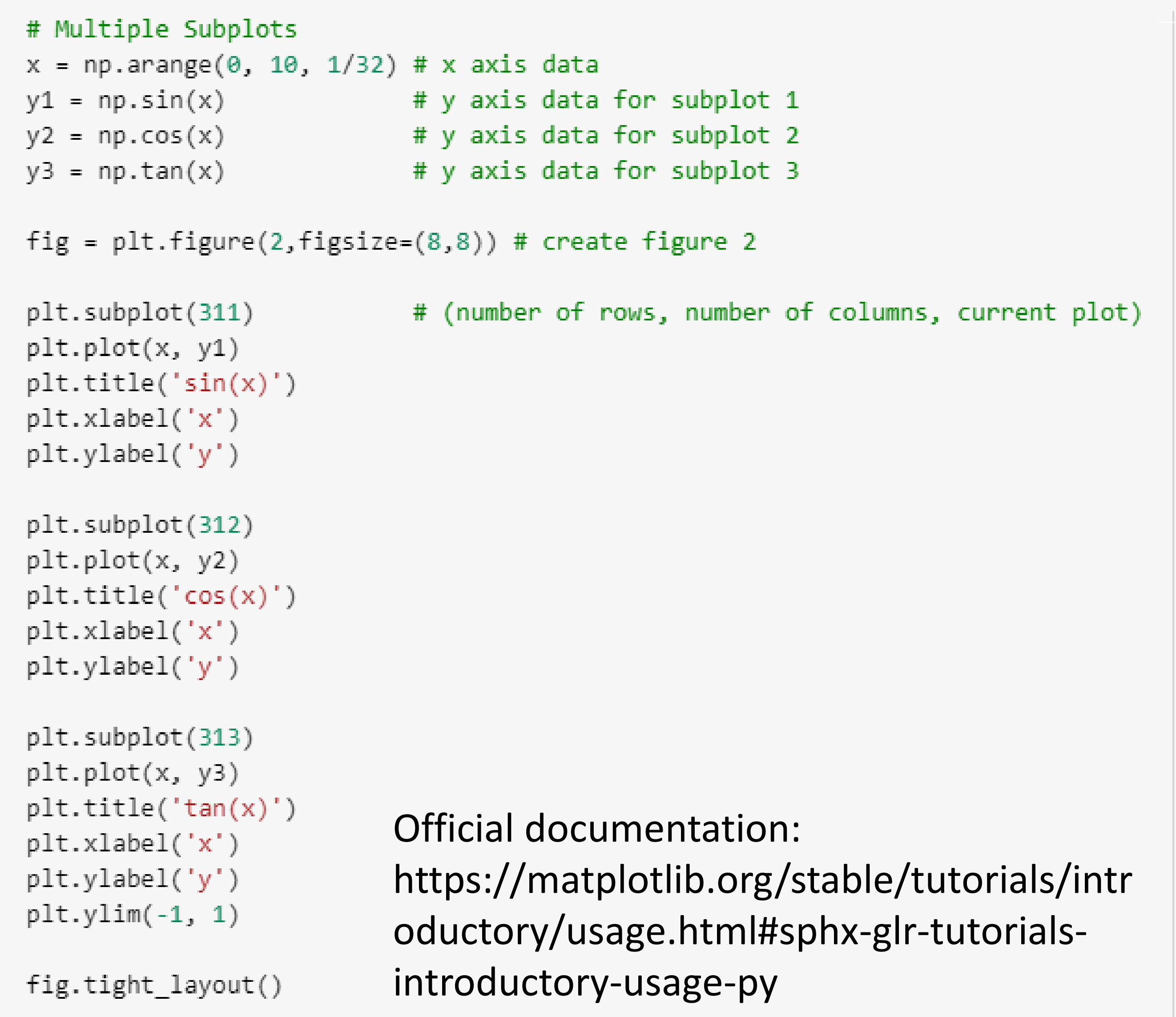
```
# Multiple Plots
# On same figure
x = np.arange(0, 10, 1/32) # x axis data
y1 = np.sin(x)             # y axis data 1
y2 = np.cos(x)             # y axis data 2
y3 = np.tan(x)             # y axis data 3
plt.figure(1)              # create figure 1
plt.plot(x, y1, label='sin(x)')
plt.plot(x, y2, label='cos(x)')
plt.plot(x, y3, label='tan(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-1, 11)
plt.ylim(-2, 2)
plt.suptitle('Trigonometric Functions')
plt.legend()
plt.show()
```

Output



Creating Subplots

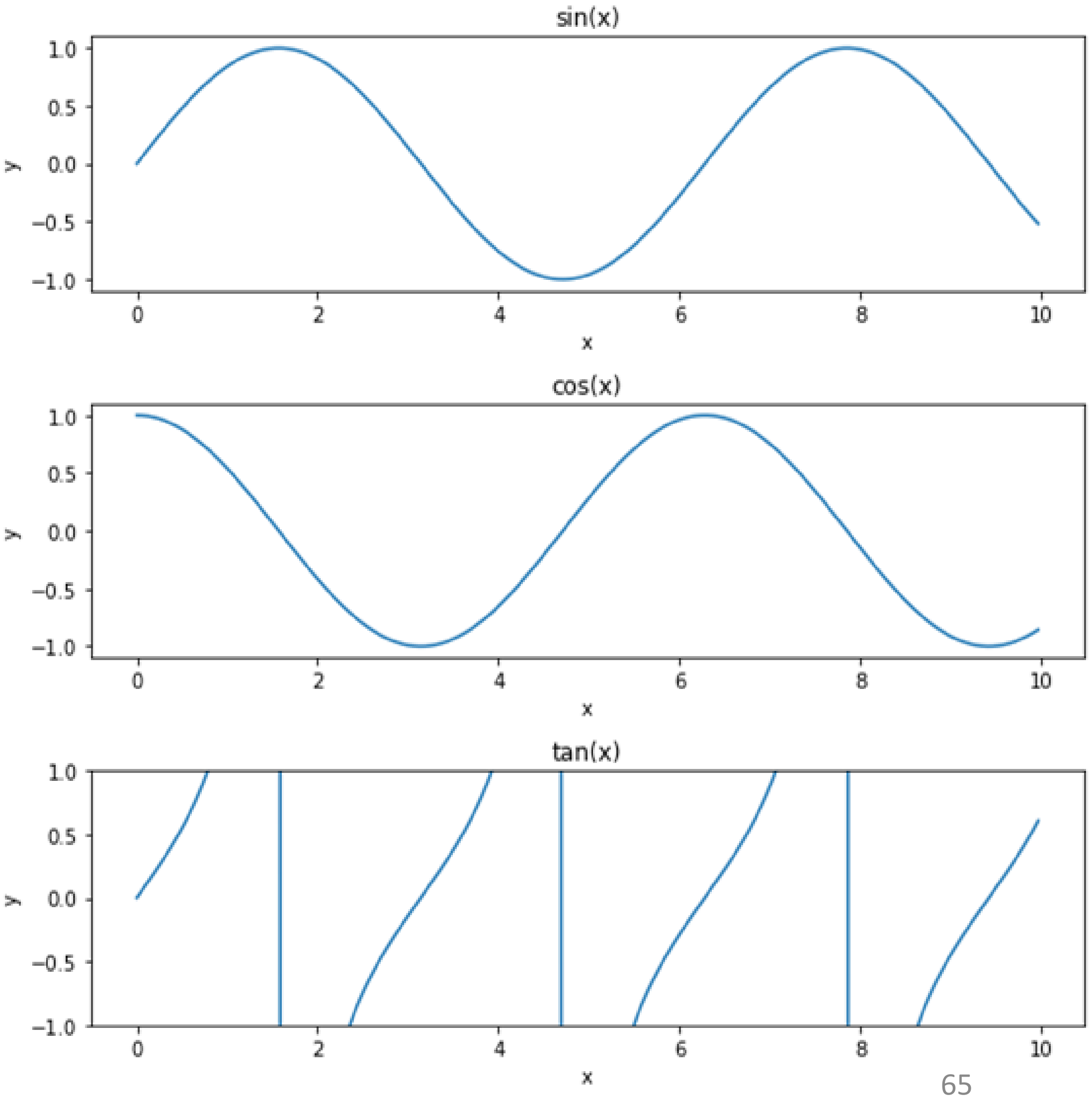
Code



```
# Multiple Subplots
x = np.arange(0, 10, 1/32) # x axis data
y1 = np.sin(x)             # y axis data for subplot 1
y2 = np.cos(x)             # y axis data for subplot 2
y3 = np.tan(x)             # y axis data for subplot 3

fig = plt.figure(2,figsize=(8,8)) # create figure 2

plt.subplot(311)           # (number of rows, number of columns, current plot)
plt.plot(x, y1)
plt.title('sin(x)')
plt.xlabel('x')
plt.ylabel('y')

plt.subplot(312)
plt.plot(x, y2)
plt.title('cos(x)')
plt.xlabel('x')
plt.ylabel('y')

plt.subplot(313)
plt.plot(x, y3)
plt.title('tan(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-1, 1)

fig.tight_layout()
```

Official documentation: https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py

Output



PART 4:

Regression with Scikit-learn

Introduction to Scikit-learn

Ridge regression example

Scikit-learn Python library

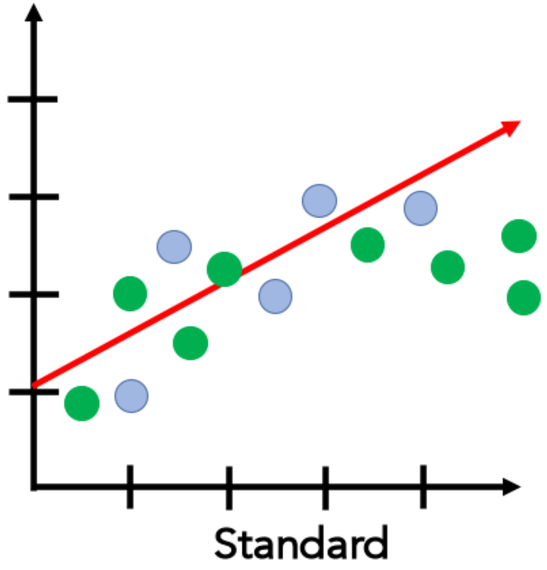
Machine learning library for Python

Features various algorithms including:

- Classification
- Regression
- Clustering

Designed to work with SciPy and NumPy

Ridge regression using Scikit-learn



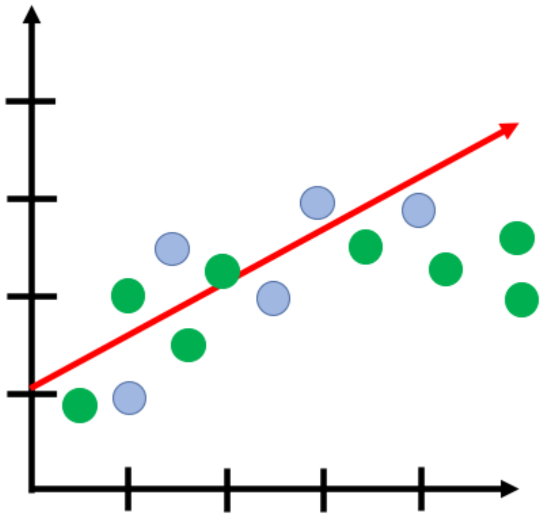
Minimizes $(y - Xb)^T(y - Xb)$

Solution $b = (X^T X)^{-1} X^T y$

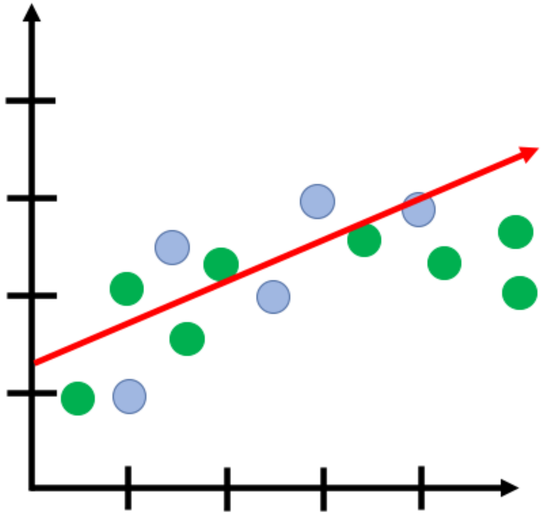
Unbiased

High variance

Ridge regression using Scikit-learn



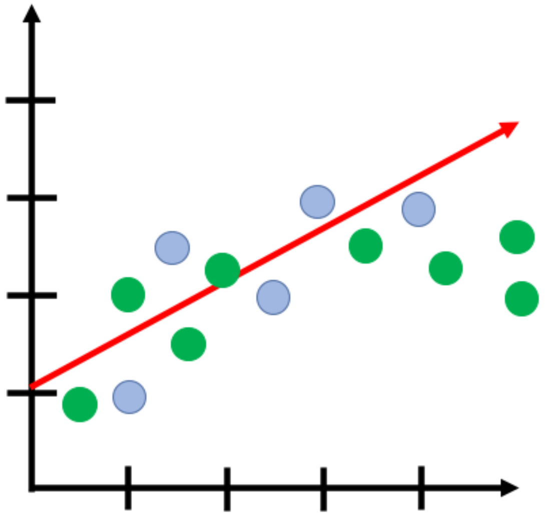
Standard



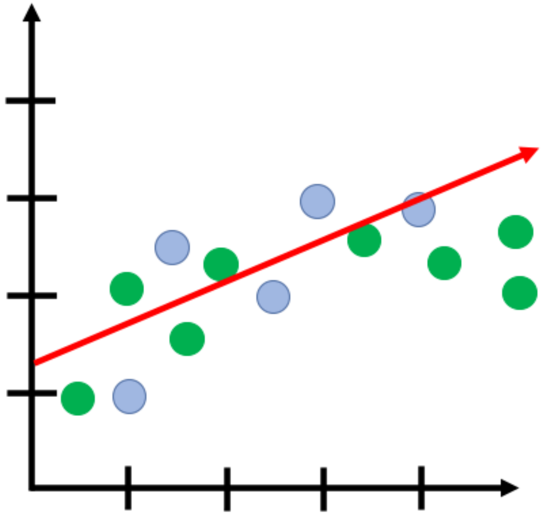
Ridge

Minimizes	$(y - Xb)^T(y - Xb)$	$(y - Xb)^T(y - Xb) + \lambda \lvert b \rvert^2$
Solution	$b = (X^TX)^{-1}X^Ty$	$b = (X^TX + \lambda I)^{-1}X^Ty$
	Unbiased	*Biased*
	High variance	*Low variance*

Ridge regression using Scikit-learn



Standard

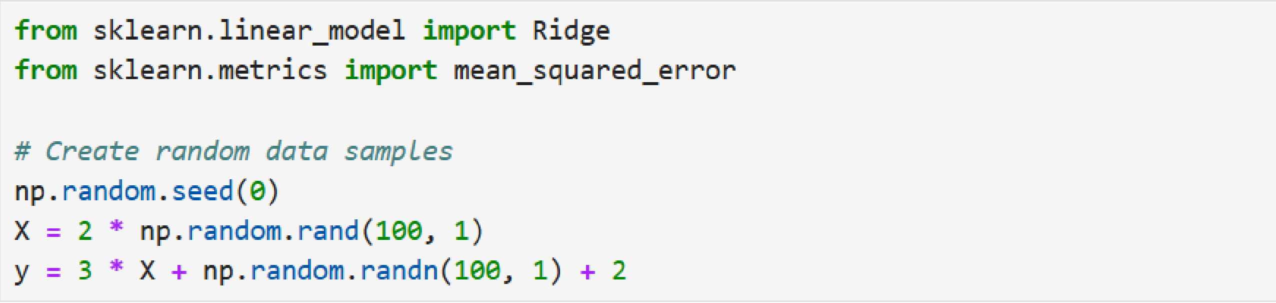


Ridge

Minimizes	$(y - Xb)^T(y - Xb)$	$(y - Xb)^T(y - Xb) + \lambda\|b\|^2$
Solution	$b = (X^TX)^{-1}X^Ty$	$b = (X^TX + \lambda I)^{-1}X^Ty$
	Unbiased	*Biased*
	High variance	*Low variance*

Ridge regression provides better regularization via penalizing b

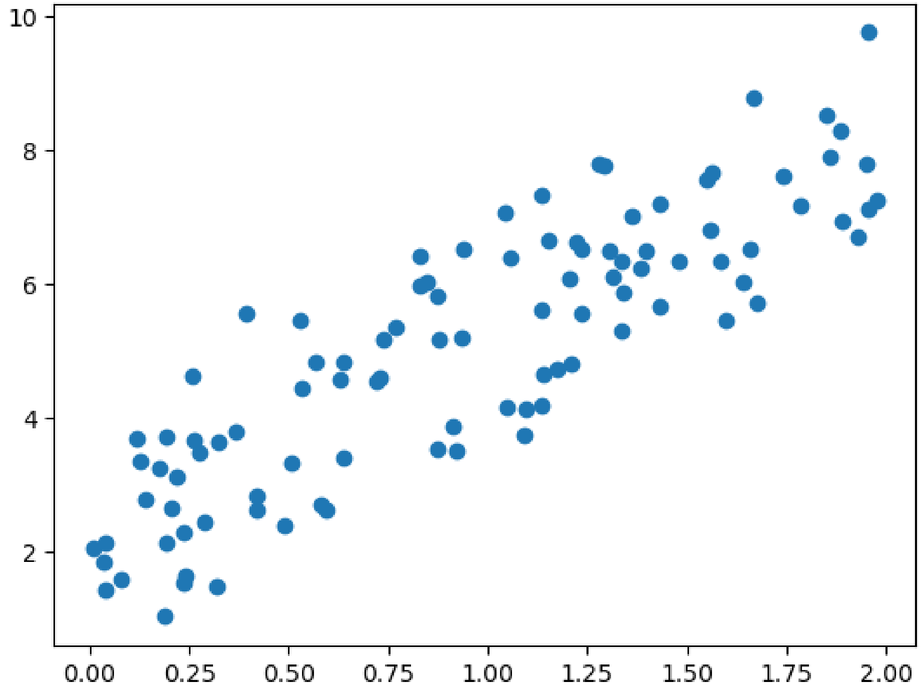
Ridge regression Example



```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Create random data samples
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 3 * X + np.random.randn(100, 1) + 2
```

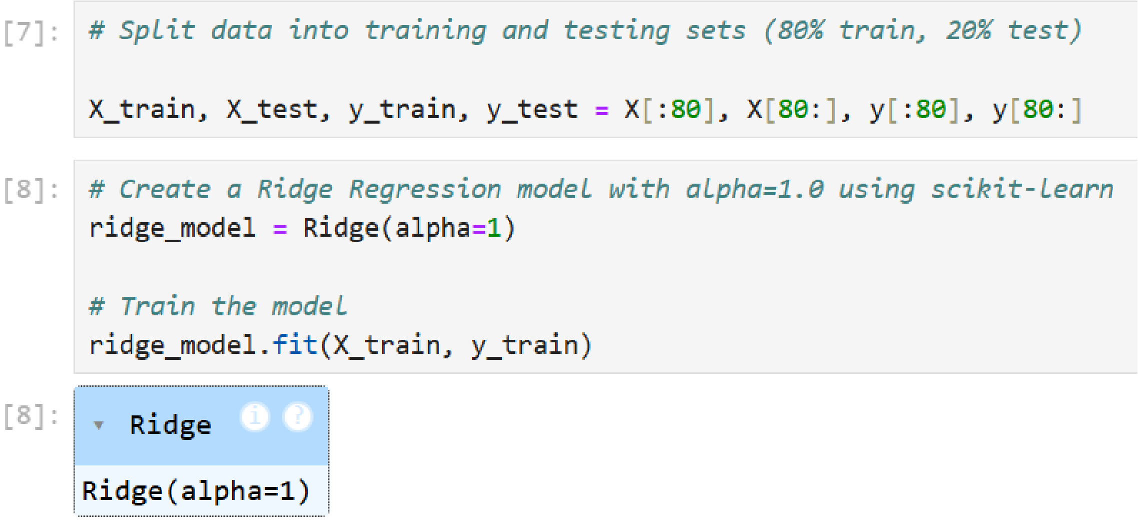
Import ridge regression algorithm from scikit-learn

Create random data samples with some gaussian noise



Plot the dataset using matplotlib plt.scatter()

Ridge regression Example



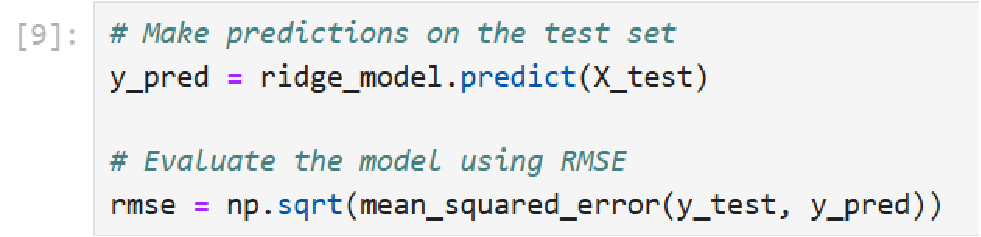
```
[7]: # Split data into training and testing sets (80% train, 20% test)

     X_train, X_test, y_train, y_test = X[:80], X[80:], y[:80], y[80:]

[8]: # Create a Ridge Regression model with alpha=1.0 using scikit-learn
     ridge_model = Ridge(alpha=1)

     # Train the model
     ridge_model.fit(X_train, y_train)

[8]: ▾ Ridge

     Ridge(alpha=1)
```

Split the data into 80% training and 20% testing

Define the ridge regression model with ridge parameter ($\lambda = 1$)

Fit the model using training data

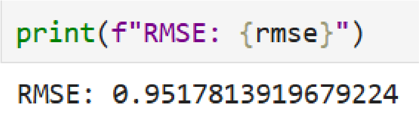
Ridge regression Example



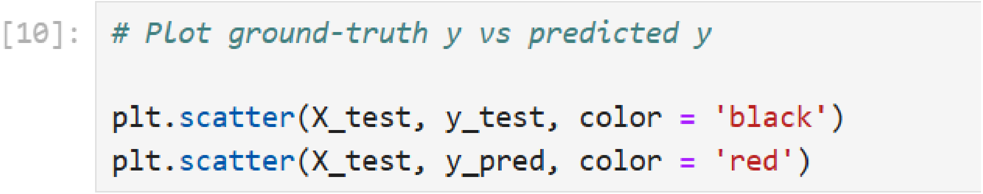
```
[9]: # Make predictions on the test set
     y_pred = ridge_model.predict(X_test)

     # Evaluate the model using RMSE
     rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

Use .predict() to test the fitted model with respect to test features

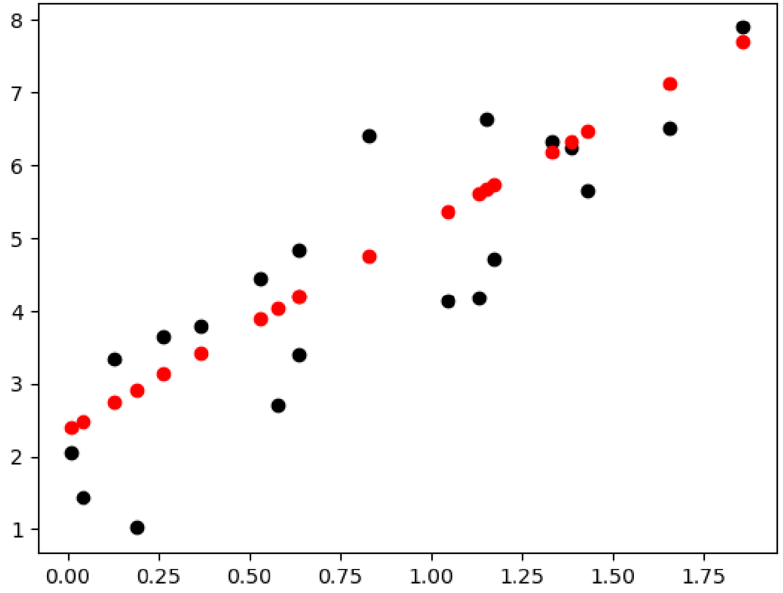
Compute the RMSE between the ground-truth vs predicted y



```
print(f"RMSE: {rmse}")
RMSE: 0.9517813919679224
```



```
[10]: # Plot ground-truth y vs predicted y

      plt.scatter(X_test, y_test, color = 'black')
      plt.scatter(X_test, y_pred, color = 'red')
```



Plot ground-truths vs predicted y using scatterplot

LAB 1 ASSIGNMENT:

Data Preparation Techniques for Machine Learning

Download ipynb template in Canvas page:

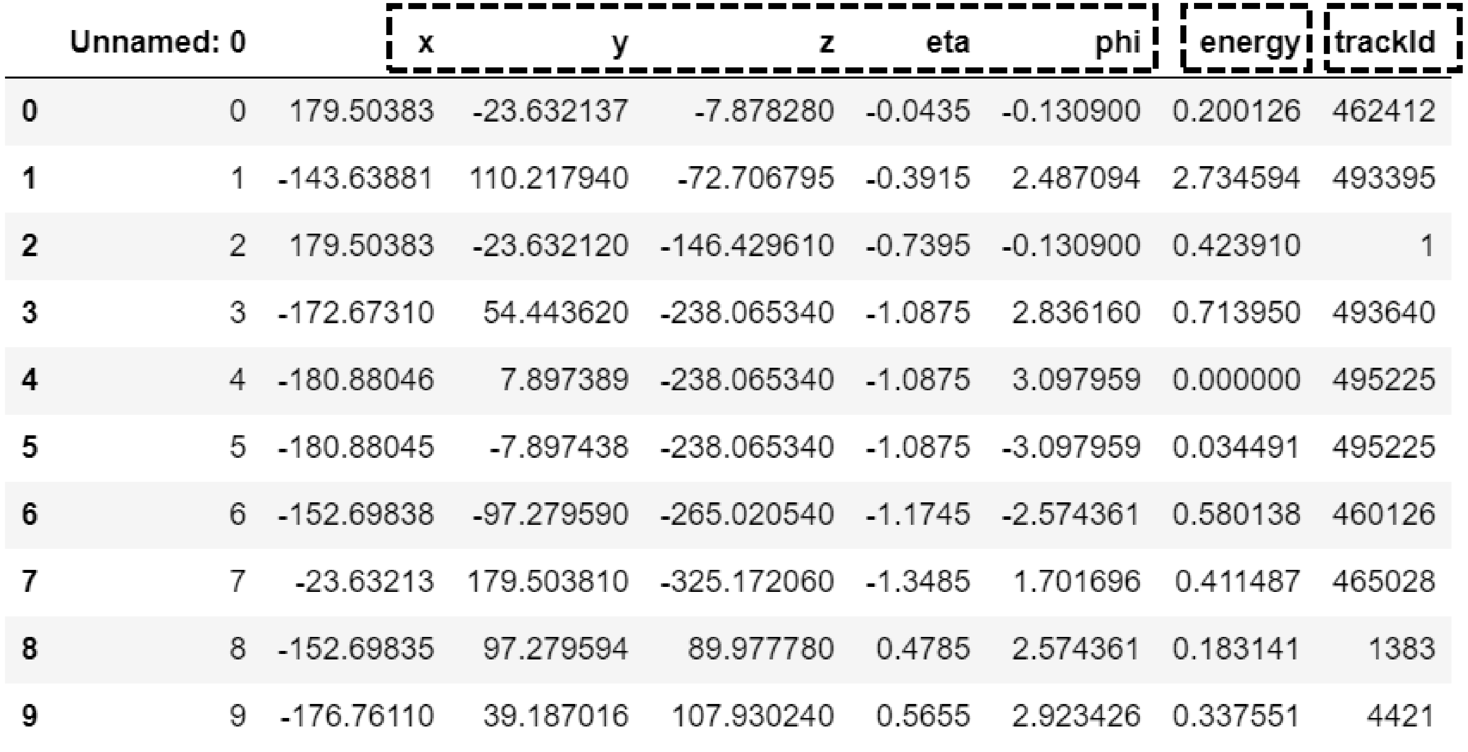
Assignments/Lab 1 report → click “Lab 1 Report Template”

Particle Collison Data from CERN

Positional coordinates of the cell

Energy deposited

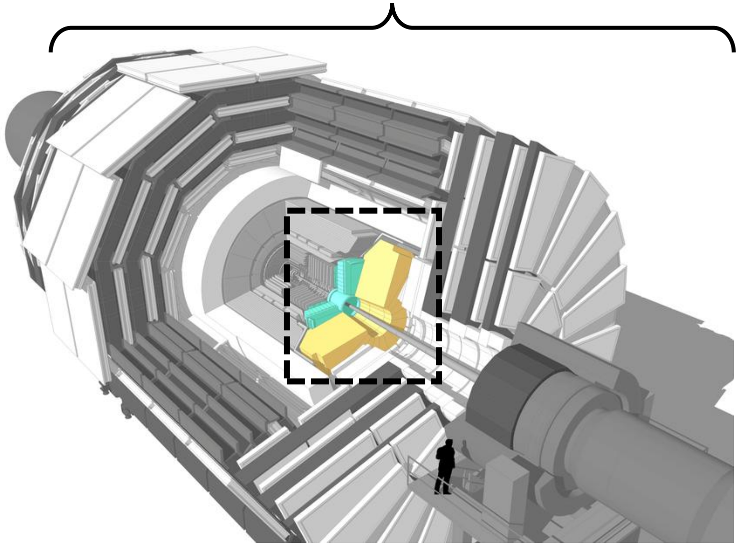
Particle ID



	Unnamed: 0	x	y	z	eta	phi	energy	trackId
0	0	179.50383	-23.632137	-7.878280	-0.0435	-0.130900	0.200126	462412
1	1	-143.63881	110.217940	-72.706795	-0.3915	2.487094	2.734594	493395
2	2	179.50383	-23.632120	-146.429610	-0.7395	-0.130900	0.423910	1
3	3	-172.67310	54.443620	-238.065340	-1.0875	2.836160	0.713950	493640
4	4	-180.88046	7.897389	-238.065340	-1.0875	3.097959	0.000000	495225
5	5	-180.88045	-7.897438	-238.065340	-1.0875	-3.097959	0.034491	495225
6	6	-152.69838	-97.279590	-265.020540	-1.1745	-2.574361	0.580138	460126
7	7	-23.63213	179.503810	-325.172060	-1.3485	1.701696	0.411487	465028
8	8	-152.69835	97.279594	89.977780	0.4785	2.574361	0.183141	1383
9	9	-176.76110	39.187016	107.930240	0.5655	2.923426	0.337551	4421

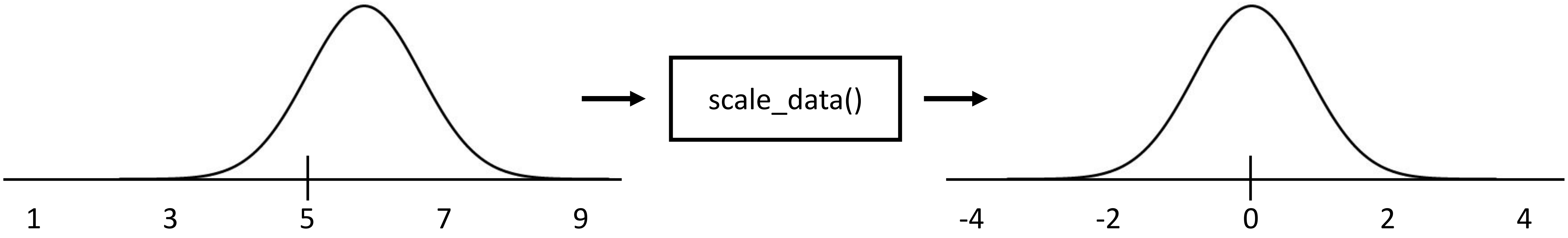
420 rows (data points), 7 columns (features)

Compact Muon Solenoid (CMS) @ LHC



High Granularity Calorimeter

Exercise 1: Scaling Data with Standard Scaling



- In Machine Learning, the dataset is usually scaled ahead of time so that it is easier for the computer to **learn** and **understand** the problem.

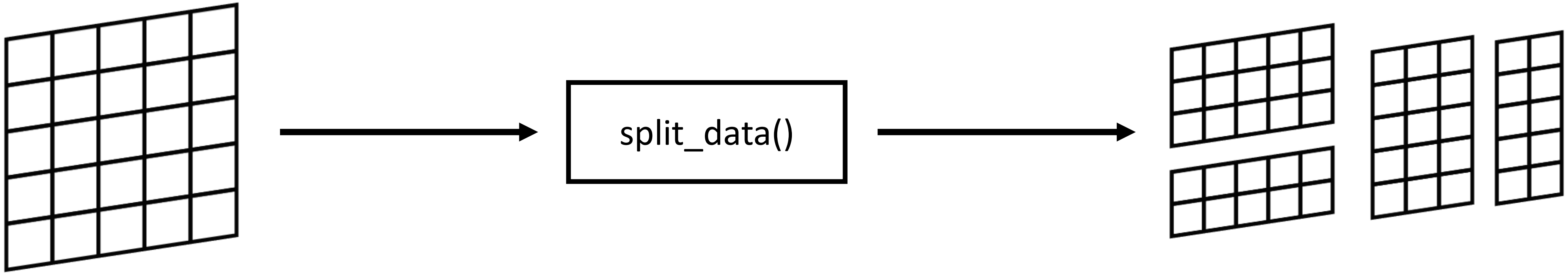
- One of the most frequently used method is ‘**standard scaling**’, where the data is scaled by $z = (x - \mu)/\sigma$. (x = original datapoint, μ = mean of the data, σ = standard deviation)

- Write a function “**scale_data()**” which takes 2D NumPy array as an input and perform standard scaling on its columns. The function should output a new 2D array containing scaled column data.

- Test your function with selected columns in CMS calorimeter dataset (**hgcal.csv**).

- Plot the scaled dataset for the selected columns by using the provided matplotlib histogram function.

Exercise 2: Data Splitting



- In this exercise you will write a function called **split_data**() which given a NumPy array, it splits the array into sub-arrays.

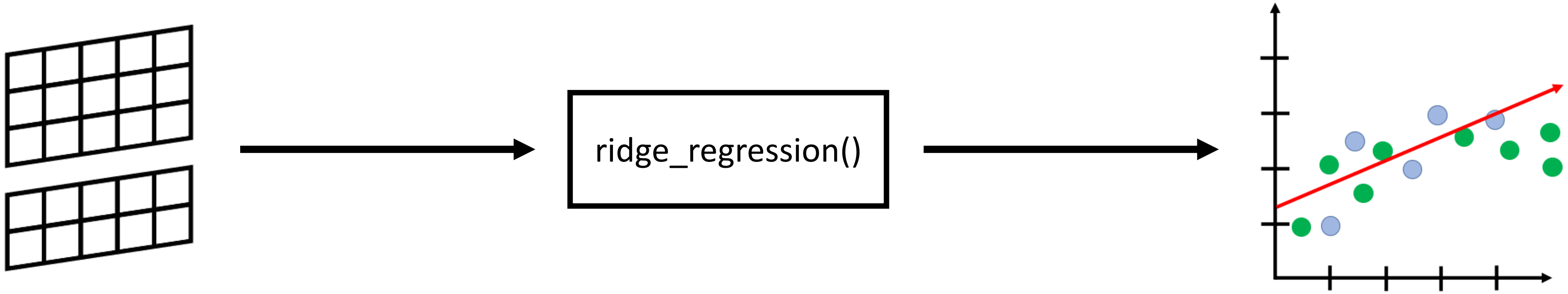
- Data splitting is used to divide the dataset into training, validation and testing sets, which we will describe in later lab.

- The function should take following parameters
  - arr – 2D NumPy array representing a dataset
  - split_proportions – a list containing split ratios, e.g., [0.2, 0.3, 0.5]
  - axis – a direction to be splitted (0 = row-wise, 1 = column-wise)

- Test your function on the scaled dataset from exercise 1 with given parameters in the lab template.

- Confirm that your sub arrays have correct dimensions by printing their shape

Exercise 3: Prediction with Ridge Regression



ridge_regression()

- In this exercise you will write a function called **ridge_regression**() which given training and test data, outputs predictions fitted by Ridge regression.
- Use the split_data() function in exercise 2 to split the data into train (80%) and test (20%) with respect to rows.
- You can use the algorithm provided by Scikit-learn to implement the function. Use **ridge parameter = 1** for alpha value.

- The function should take following parameters
  - X_train, y_train – Features (positional coordinates) and energy values used for training (80%)
  - X_test – Features used for testing (20%)
  - alpha – ridge parameter defining the penalty

- After prediction, scale back the predicted energy values to original scales and report the RMSE with respect to ground-truth.

- Plot the predicted vs ground-truth energy values (original scale) using matplotlib