

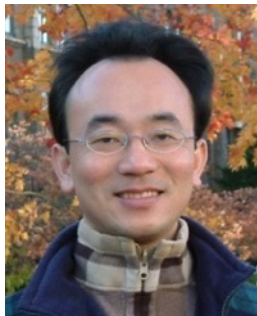
Python for Accounting Applications

Functions and Modules; Files and Exception Handling

1131PAA05

ACC2, NTPU (U2004) (Fall 2024)

Wed 6, 7, 8, (14:10-17:00) (9:10-12:00) (B3F10)



Min-Yuh Day, Ph.D,
Professor

Institute of Information Management, National Taipei University

<https://web.ntpu.edu.tw/~myday>



Syllabus

Week Date Subject/Topics

1 2024/09/11 Introduction to Python for Accounting Applications

2 2024/09/18 Python Programming and Data Science

3 2024/09/25 Foundations of Python Programming

4 2024/10/02 Data Structures

5 2024/10/09 Control Logic and Loops

6 2024/10/16 Functions and Modules; Files and Exception Handling

7 2024/10/23 Data Analytics and Visualization with Python

8 2024/10/30 Midterm Project Report

Syllabus

Week Date Subject/Topics

9 2024/11/06 Self-Learning

10 2024/11/13 Obtaining Data From the Web with Python

11 2024/11/20 Statistical Analysis with Python

12 2024/11/27 Machine Learning with Python

13 2024/12/04 Text Analytics with Generative AI and Python

14 2024/12/11 Applications of Accounting Data Analytics with Python

15 2024/12/18 Applications of ESG Data Analytics with Python

16 2024/12/25 Final Project Report

Outline

- Python Functions: **def** myfunction():
- Python Classes/Objects: **class** MyClass:
- Python Modules: **import** mymodule
- Python Files (File Handling): **open()**
- Python Try Except (Exception Handling):
 - **try: except: else: finally:**

Python Functions and Modules

- Python Functions
 - **def** myfunction():
- Python Classes/Objects
 - **class** MyClass:
- Python Modules
 - mymodule.py
 - **import** mymodule

Files and Exception Handling

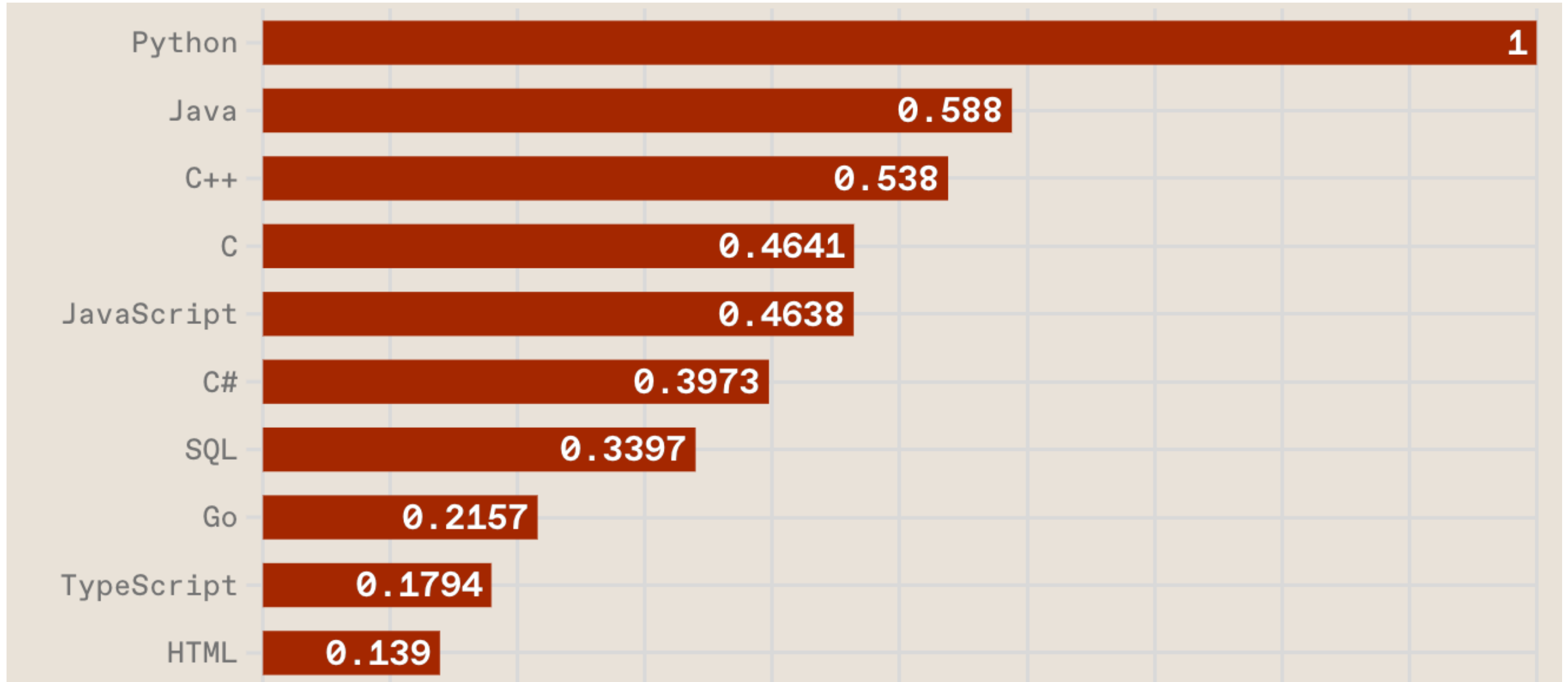
- **Python Files (File Handling)**
 - **open()**
 - **f = open("myfile.txt")**
- **Python Try Except (Exception Handling)**
 - **try:**
 - **except:**
 - **else:**
 - **finally:**



Python

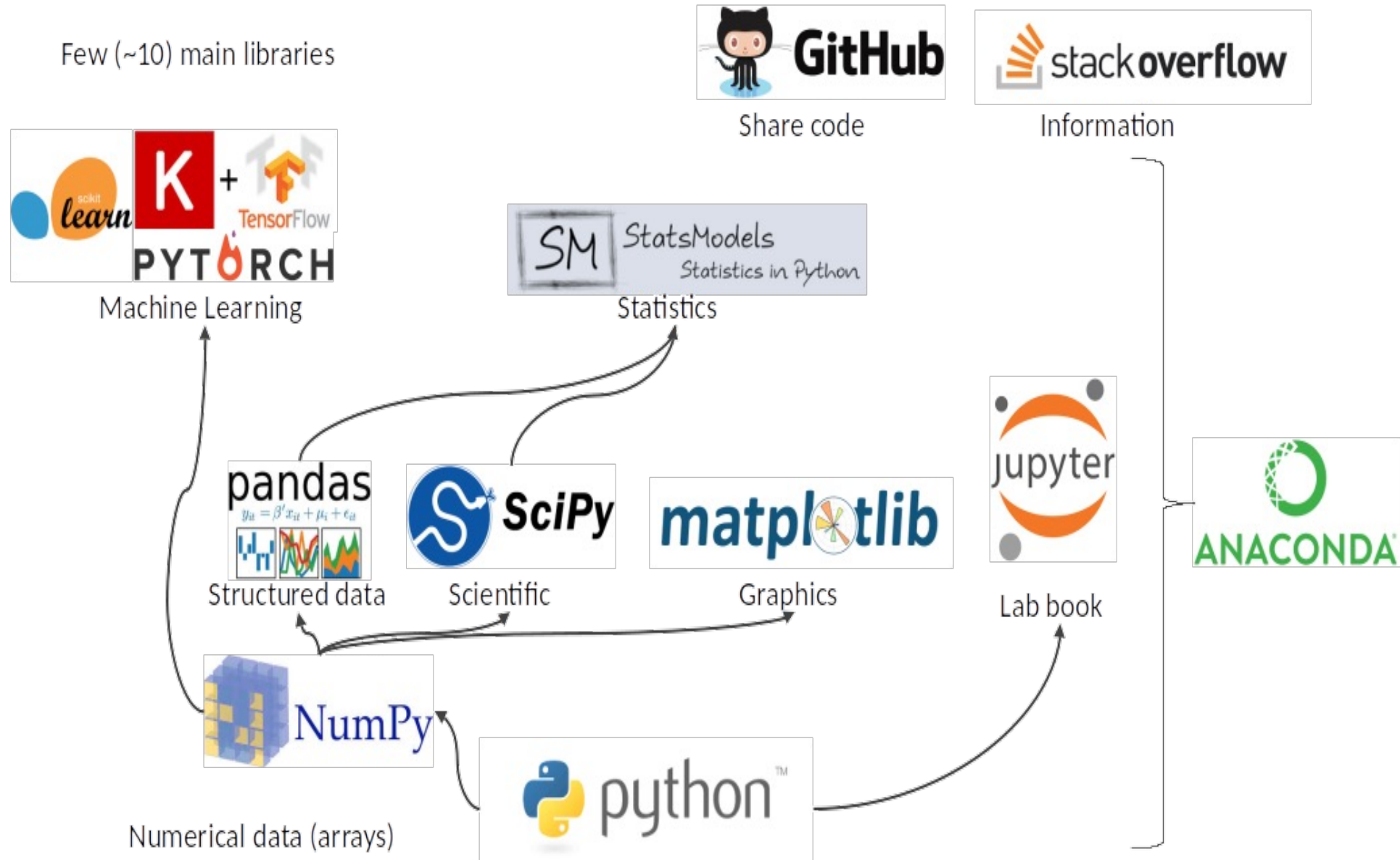
Programming

Top Programming Languages

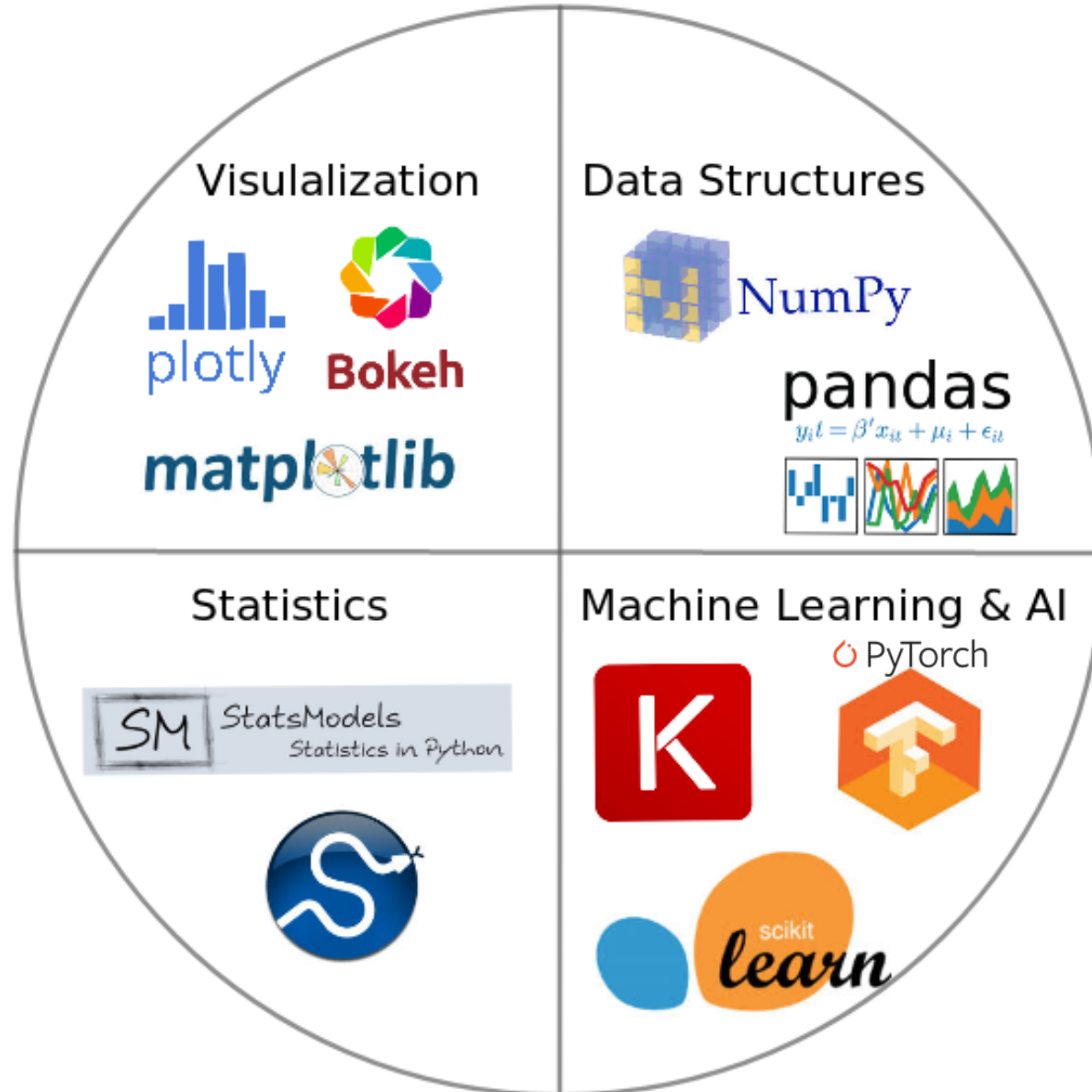


Python is an
interpreted,
object-oriented,
high-level
programming language
with
dynamic semantics.

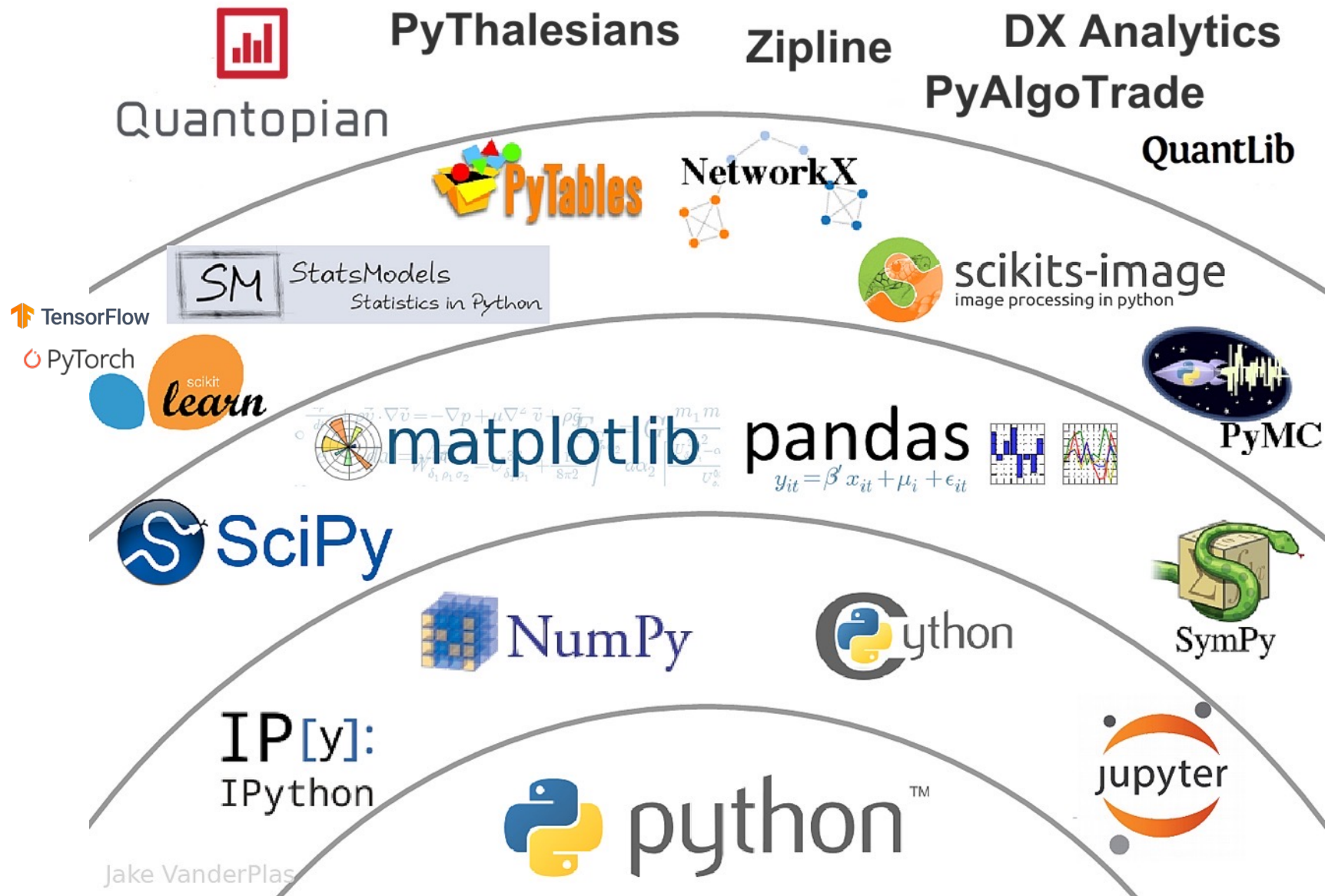
Python Ecosystem for Data Science



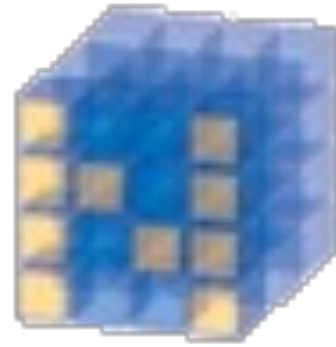
Python Ecosystem for Data Science



The Quant Finance PyData Stack



NumPy



NumPy

Base

N-dimensional array
package

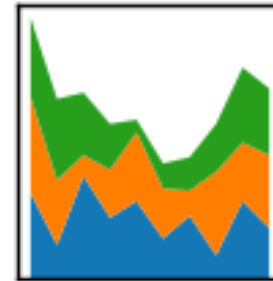
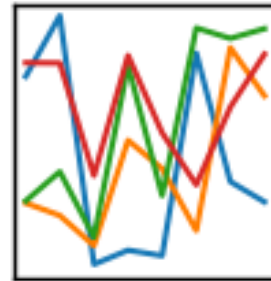
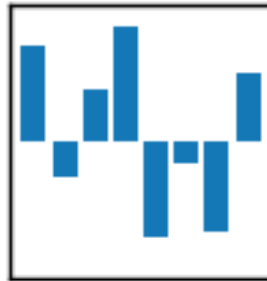
Python
matplotlib
matplotlib

Python

Pandas

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



- Python Tutorial
- Python HOME**
- Python Intro
- Python Get Started
- Python Syntax
- Python Comments
- Python Variables
- Python Data Types
- Python Numbers
- Python Casting
- Python Strings
- Python Booleans
- Python Operators
- Python Lists
- Python Tuples
- Python Sets
- Python Dictionaries
- Python If...Else
- Python While Loops
- Python For Loops
- Python Functions

Python Tutorial

[← Home](#)

[Next >](#)

Learn Python

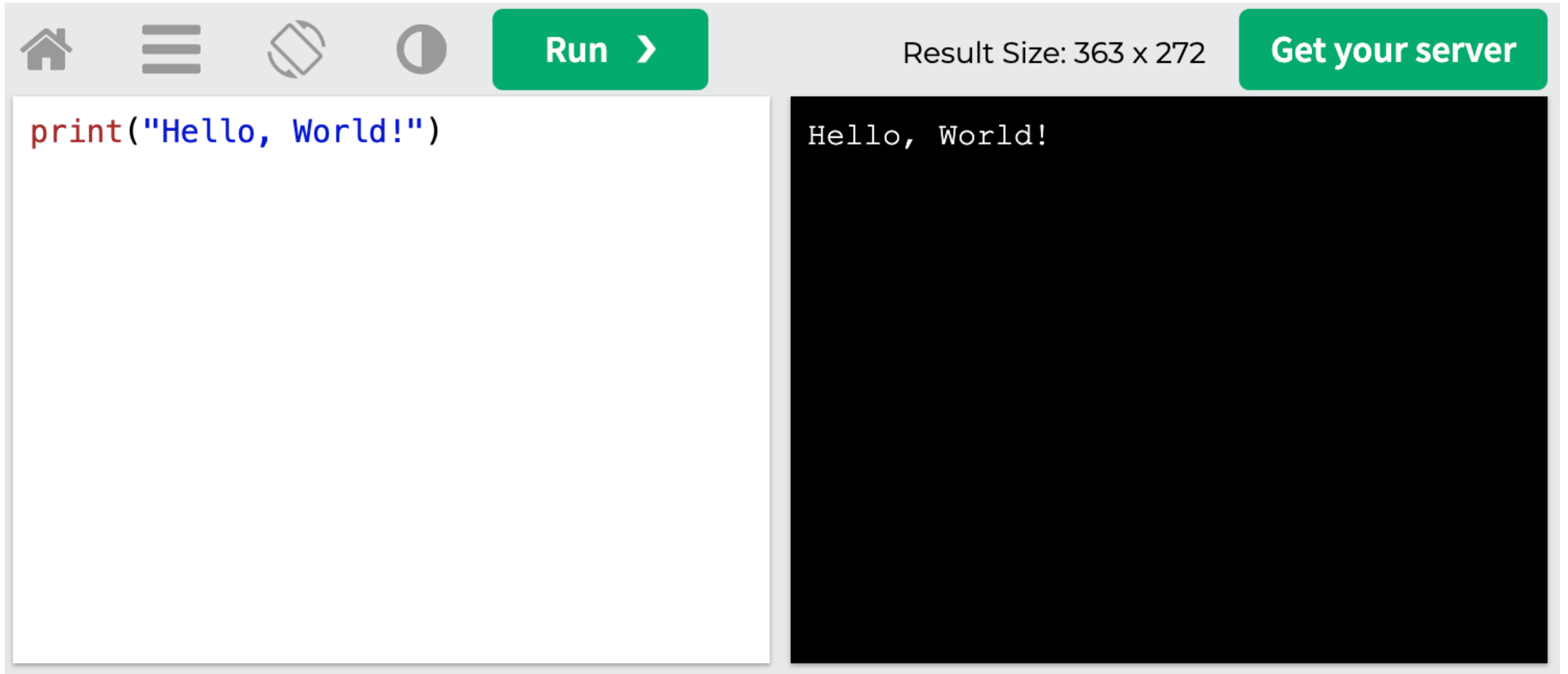
Python is a popular programming language.
Python can be used on a server to create web applications.

[Start learning Python now »](#)

Learning by Examples

With our "Try it Yourself" editor, you can edit Python code and view the result.

W3Schools Python: Try Python

A screenshot of the W3Schools Python Try Python interface. The interface has a light gray header with navigation icons (home, menu, refresh, moon) on the left, a green 'Run >' button in the center, and 'Result Size: 363 x 272' and a green 'Get your server' button on the right. Below the header, there is a white code editor on the left containing the Python code `print("Hello, World!")` and a black terminal window on the right displaying the output 'Hello, World!'.

LearnPython.org



learnpython.org

[Home](#)

[About](#)

[Certify](#)

[More Languages](#) ▾

[Python](#)

[Java](#)

[HTML](#)

[Go](#)

[C](#)

[C++](#)

[JavaScript](#)

[PHP](#)

[Shell](#)

[C#](#)

[Perl](#)

[Ruby](#)

[Scala](#)

[SQL](#)

Get started learning Python with [DataCamp's](#) free [Intro to Python tutorial](#). Learn Data Science by completing interactive coding challenges and watching videos by expert instructors. [Start Now!](#)

Ready to take the test? Head onto [LearnX](#) and get your Python Certification!

This site is generously supported by [DataCamp](#). DataCamp offers online interactive [Python Tutorials](#) for Data Science. Join **11 millions** other learners and get started learning Python for data science today!

Good news! You can save 25% off your Datacamp annual subscription with the code [LEARNPYTHON23ALE25](#) - [Click here to redeem your discount!](#)

Welcome

Welcome to the LearnPython.org interactive Python tutorial.

Whether you are an experienced programmer or not, this website is intended for everyone who wishes to learn the Python programming language.

You are welcome to join our group on [Facebook](#) for questions, discussions and updates.

After you complete the tutorials, you can get certified at [LearnX](#) and add your certification to your LinkedIn profile.

Just click on the chapter you wish to begin from, and follow the instructions. Good luck!

<https://www.learnpython.org/>

Google's Python Class



Filter

Overview

Python Set Up

Python Intro

Strings

Lists

Sorting

Dicts and Files

Regular Expressions

Utilities

Lecture Videos

1.1 Introduction, strings

1.2 Lists and sorting

1.3 Dicts and files

2.1 Regular expr

2.2 Utilities

2.3 Utilities urllib

2.4 Conclusions

Python Exercises



Google's Python Class

Welcome to Google's Python Class -- this is a free class for people with a little bit of programming experience who want to learn Python. The class includes written materials, lecture videos, and lots of code exercises to practice Python coding. These materials are used within Google to introduce Python to people who have just a little programming experience. The first exercises work on basic Python concepts like strings and lists, building up to the later exercises which are full programs dealing with text files, processes, and http connections. The class is geared for people who have a little bit of programming experience in some language, enough to know what a "variable" or "if statement" is. Beyond that, you do not need to be an expert programmer to use this material.

To get started, the Python sections are linked at the left -- [Python Set Up](#) to get Python installed on your machine, [Python Introduction](#) for an introduction to the language, and then [Python Strings](#) starts the coding material, leading to the first exercise. The end of each written section includes a link to the code exercise for that section's material. The lecture videos parallel the written materials, introducing Python, then strings, then first exercises, and so on. At Google, all this material makes up an intensive 2-day class, so the videos are organized as the day-1 and day-2 sections.

This material was created by [Nick Parlante](#) working in the engEDU group at Google. Special thanks for the help from my Google colleagues John Cox, Steve Glassman, Piotr Kaminski, and Antoine Picard. And finally thanks to Google and my director Maggie Johnson for the enlightened generosity to put these materials out on the internet for free under the [Creative Commons Attribution 2.5](#) license -- share and enjoy!

<https://developers.google.com/edu/python>

Google Colab

Table of contents

- Getting Started
- Highlighted Features
 - TensorFlow execution
- GitHub
- Visualization
- Forms
- Examples
- Local runtime support

SECTION

Welcome to Colaboratory!

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. See our [FAQ](#) for more info.

Getting Started

- [Overview of Colaboratory](#)
- [Loading and saving data: Local files, Drive, Sheets, Google Cloud Storage](#)
- [Importing libraries and installing dependencies](#)
- [Using Google Cloud BigQuery](#)
- [Forms, Charts, Markdown, & Widgets](#)
- [TensorFlow with GPU](#)
- [Machine Learning Crash Course: Intro to Pandas & First Steps with TensorFlow](#)

Highlighted Features

Seedbank

Looking for Colab notebooks to learn from? Check out [Seedbank](#), a place to discover interactive machine learning examples.

TensorFlow execution

Colaboratory allows you to execute TensorFlow code in your browser with a single click. The example below adds two matrices.

$$\begin{bmatrix} 1. & 1. & 1. \end{bmatrix} + \begin{bmatrix} 1. & 2. & 3. \end{bmatrix} = \begin{bmatrix} 2. & 3. & 4. \end{bmatrix}$$

Connect Google Colab in Google Drive

The image shows a browser window with the Google Drive interface. The address bar displays 'https://drive.google.com/drive/u/2/my-drive'. The main content area shows the 'My Drive' section with a 'New' button. A red dashed box highlights the 'New' button and the 'More' option in the dropdown menu. Another red dashed box highlights the 'Connect more apps' option in the 'More' submenu. The interface includes a search bar, navigation icons, and a storage usage indicator showing '0 bytes of 15 GB used'.

My Drive - Google Drive

Search Drive

My Drive

Quick Access

New

My Drive

Computers

Shared with me

Recent

Starred

Trash

Backups

Storage

0 bytes of 15 GB used

UPGRADE STORAGE

Get Backup and Sync for Mac

Files

Store safely

Sync seamlessly

Access anywhere

Share easily

New folder...

Upload files...

Upload folder...

Google Docs

Google Sheets

Google Slides

More

Google Forms

Google Drawings

Google My Maps

Google Sites

Connect more apps

Google Colab

The screenshot shows the Google Drive interface with a 'Connect apps to Drive' dialog box open. The dialog box has a search bar at the top with 'colab' entered and highlighted by a red dashed border. Below the search bar, there is a grid of app cards. Each card displays the app's logo, name, and user count. The apps listed are:

- ZIP Extractor**: Extract ZIP files to Google Drive. Extraction complete. 307,585 users.
- Lumin PDF**: Beautiful PDF Editor. The fast and simple PDF Viewer. 289,310 users.
- CloudConvert**: 373,161 users.
- Sejda**: Merge PDF - Split PDF - Sejda.com. 1106 reviews.
- DocHub**: Edit, Send & Sign PDFs. 2,131,600 users.
- Google Forms**: 4,803,614 users.

The background shows the Google Drive interface with a sidebar on the left containing navigation options like 'My Drive', 'Computers', 'Shared with me', 'Recent', 'Starred', 'Trash', 'Backups', and 'Storage'. The top of the browser shows the address bar with the URL 'https://drive.google.com/drive/u/2/my-drive'.

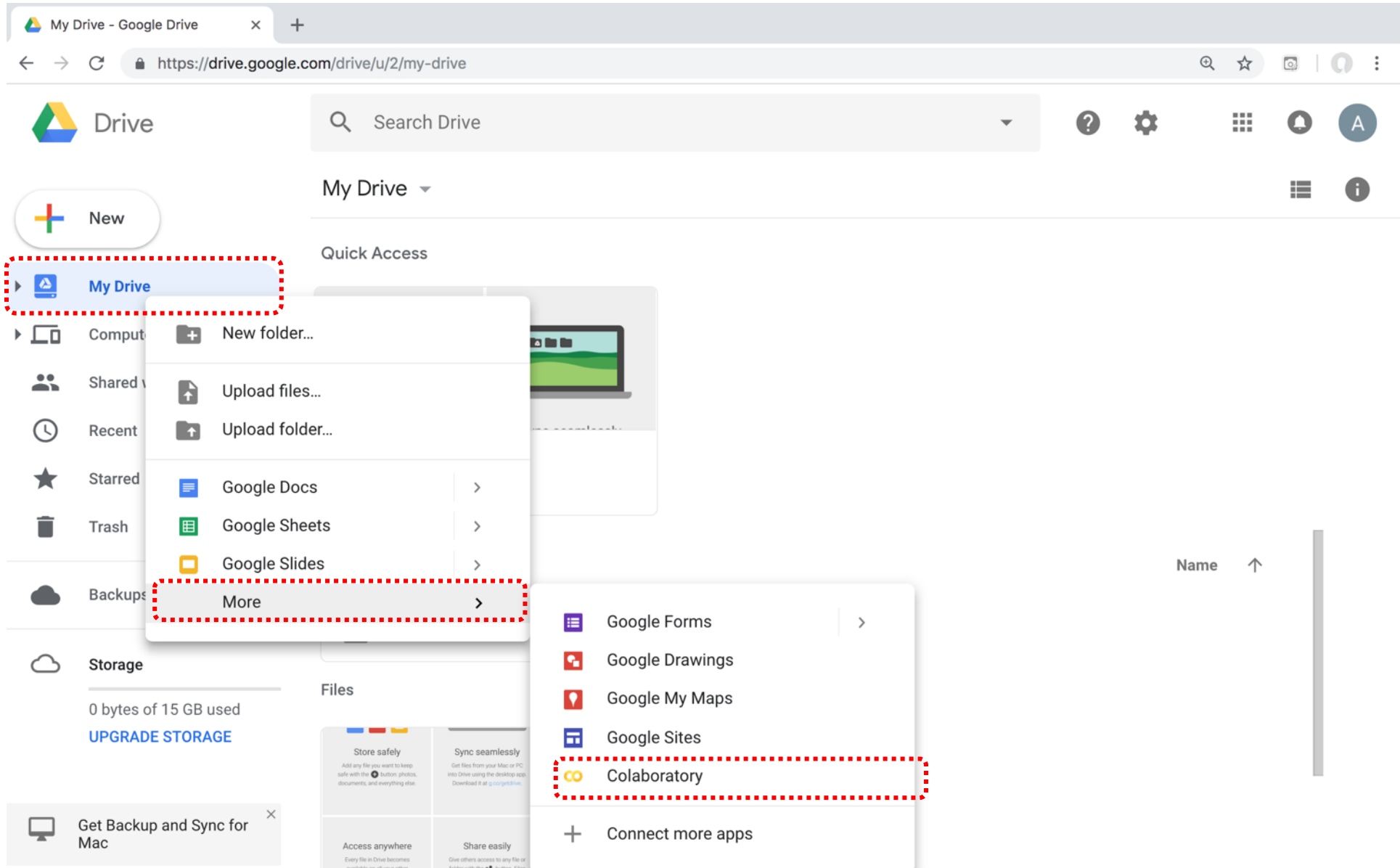
Google Colab

The image shows a browser window with the Google Drive interface. A modal dialog titled "Connect apps to Drive" is open in the center. The dialog has a search bar at the top with the text "colab" entered. Below the search bar, a list of apps is displayed. The first app, "Colaboratory", is highlighted with a red dashed border. The app's details include a yellow "CO" logo, the name "Colaboratory", the URL "https://colab.research.google.com", a description: "A data analysis tool that combines code, output, and descriptive text into one collaborative document.", and a rating of five stars with "(195)" reviews. A blue button with a white plus sign and the text "+ CONNECT" is positioned to the right of the app details. The background shows the Google Drive sidebar with options like "New", "My Drive", "Computers", "Shared with me", "Recent", "Starred", "Trash", "Backups", and "Storage". The top navigation bar includes the Drive logo, a search bar, and various utility icons.

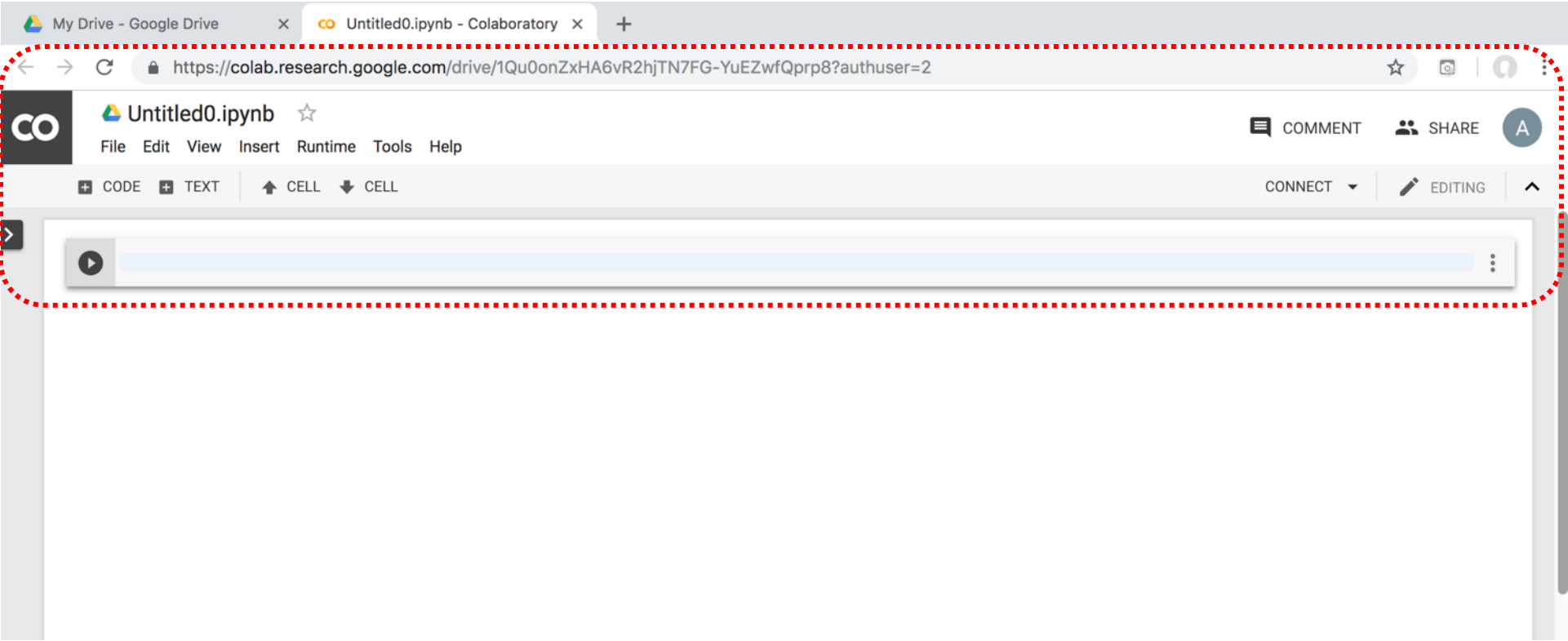
Connect Colaboratory to Google Drive

The screenshot shows the Google Drive web interface. At the top, the browser tab is labeled 'My Drive - Google Drive' and the address bar shows 'https://drive.google.com/drive/u/2/my-drive'. The Drive sidebar on the left includes options like 'New', 'My Drive', 'Computers', 'Shared with me', 'Recent', 'Starred', 'Trash', 'Backups', and 'Storage'. The main area displays a 'Connect apps to Drive' dialog box with a search filter 'colab'. A confirmation message from Colaboratory is centered, stating 'Colaboratory was connected to Google Drive.' and 'Make Colaboratory the default app for files it can open' with a checked checkbox. An 'OK' button is highlighted with a red dashed border. To the right of the message is a 'RATE IT' section for 'Productivity' with a 5-star rating and 195 reviews.

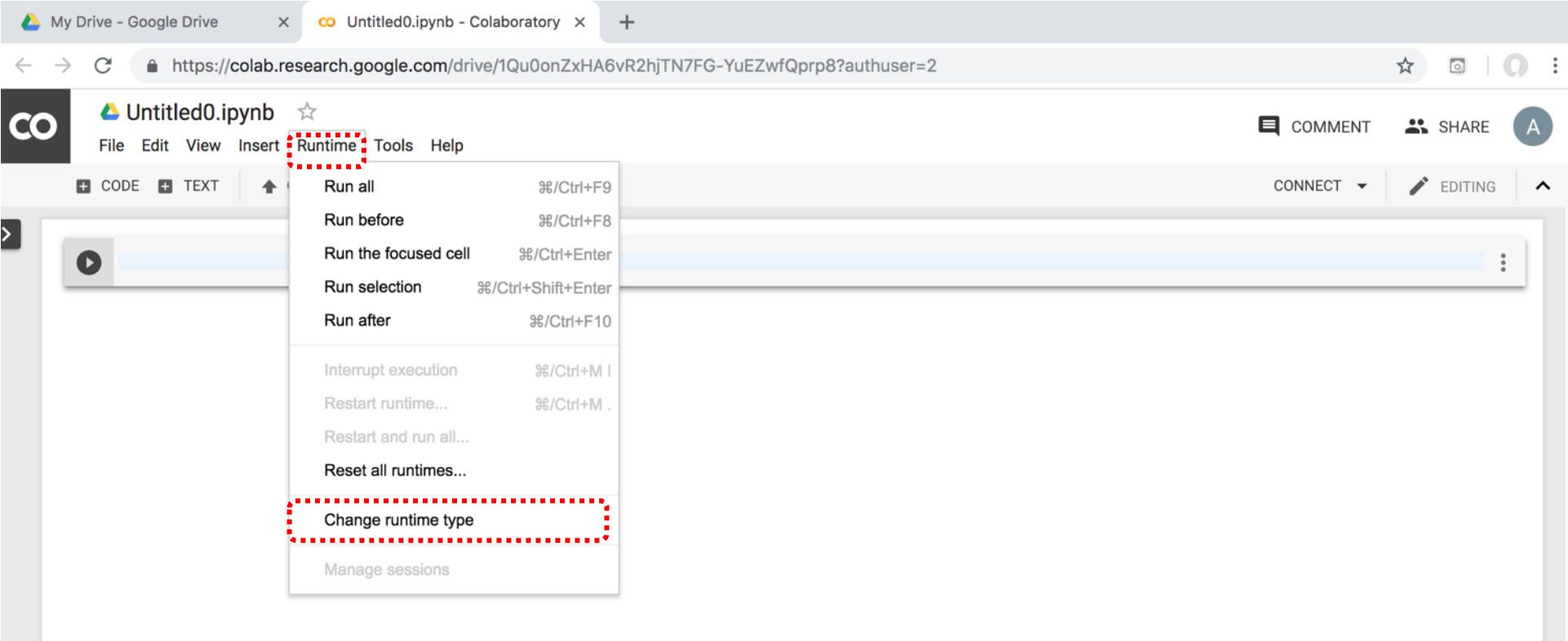
Google Colab



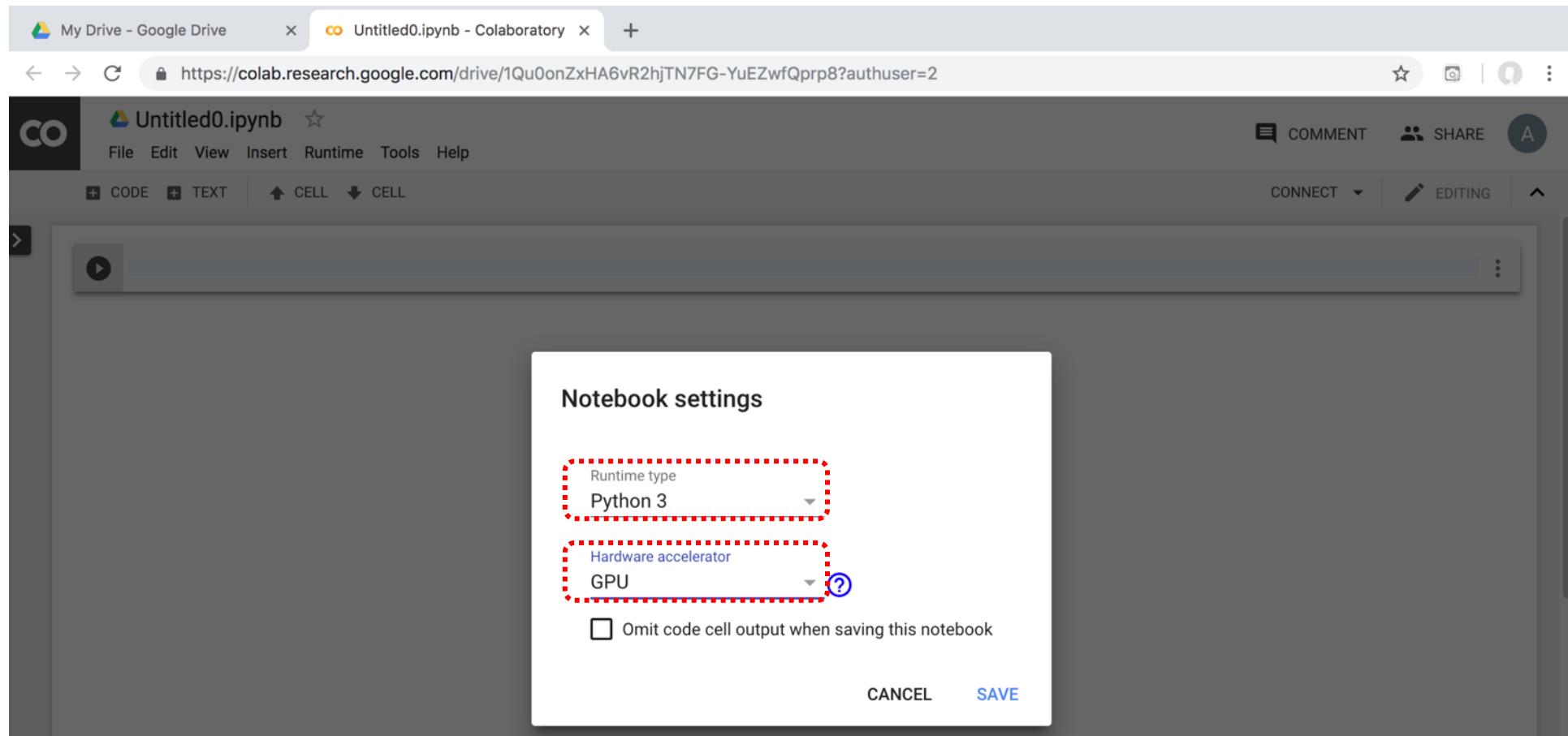
Google Colab



Google Colab



Run Jupyter Notebook Python3 GPU Google Colab



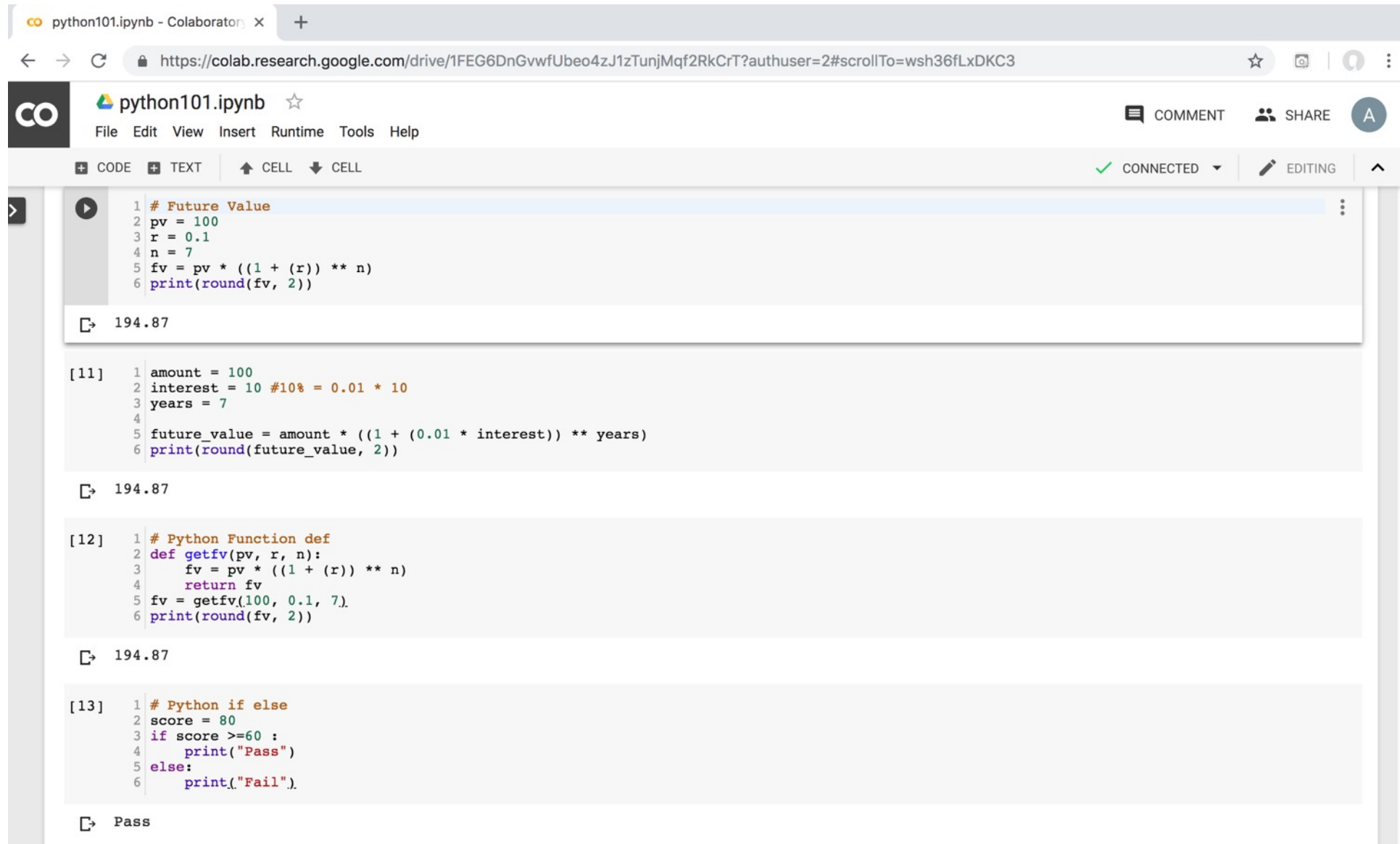
Google Colab Python Hello World

```
print('Hello World')
```



Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>



The screenshot shows a Google Colab notebook titled "python101.ipynb". The interface includes a browser address bar, a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), and a toolbar with options for CODE, TEXT, CELL, and a status indicator showing "CONNECTED" and "EDITING".

The notebook contains four code cells:

- Cell 1:** A code cell with the following Python code:

```
1 # Future Value
2 pv = 100
3 r = 0.1
4 n = 7
5 fv = pv * ((1 + (r)) ** n)
6 print(round(fv, 2))
```

The output is "194.87".
- Cell [11]:** A code cell with the following Python code:

```
1 amount = 100
2 interest = 10 #10% = 0.01 * 10
3 years = 7
4
5 future_value = amount * ((1 + (0.01 * interest)) ** years)
6 print(round(future_value, 2))
```

The output is "194.87".
- Cell [12]:** A code cell with the following Python code:

```
1 # Python Function def
2 def getfv(pv, r, n):
3     fv = pv * ((1 + (r)) ** n)
4     return fv
5 fv = getfv(100, 0.1, 7)
6 print(round(fv, 2))
```

The output is "194.87".
- Cell [13]:** A code cell with the following Python code:

```
1 # Python if else
2 score = 80
3 if score >=60 :
4     print("Pass")
5 else:
6     print("Fail").
```

The output is "Pass".

<https://tinyurl.com/aintpupython101>



Anaconda
The Most Popular
Python
Data Science Platform

Download Anaconda



Products ▾

Pricing

Solutions ▾

Resources ▾

Partners ▾

Blog

Company ▾

Contact Sales

Data science technology for a better world.

Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. Start working with thousands of open-source packages and libraries today.

Download 

Get Additional Installers



<https://www.anaconda.com/download>

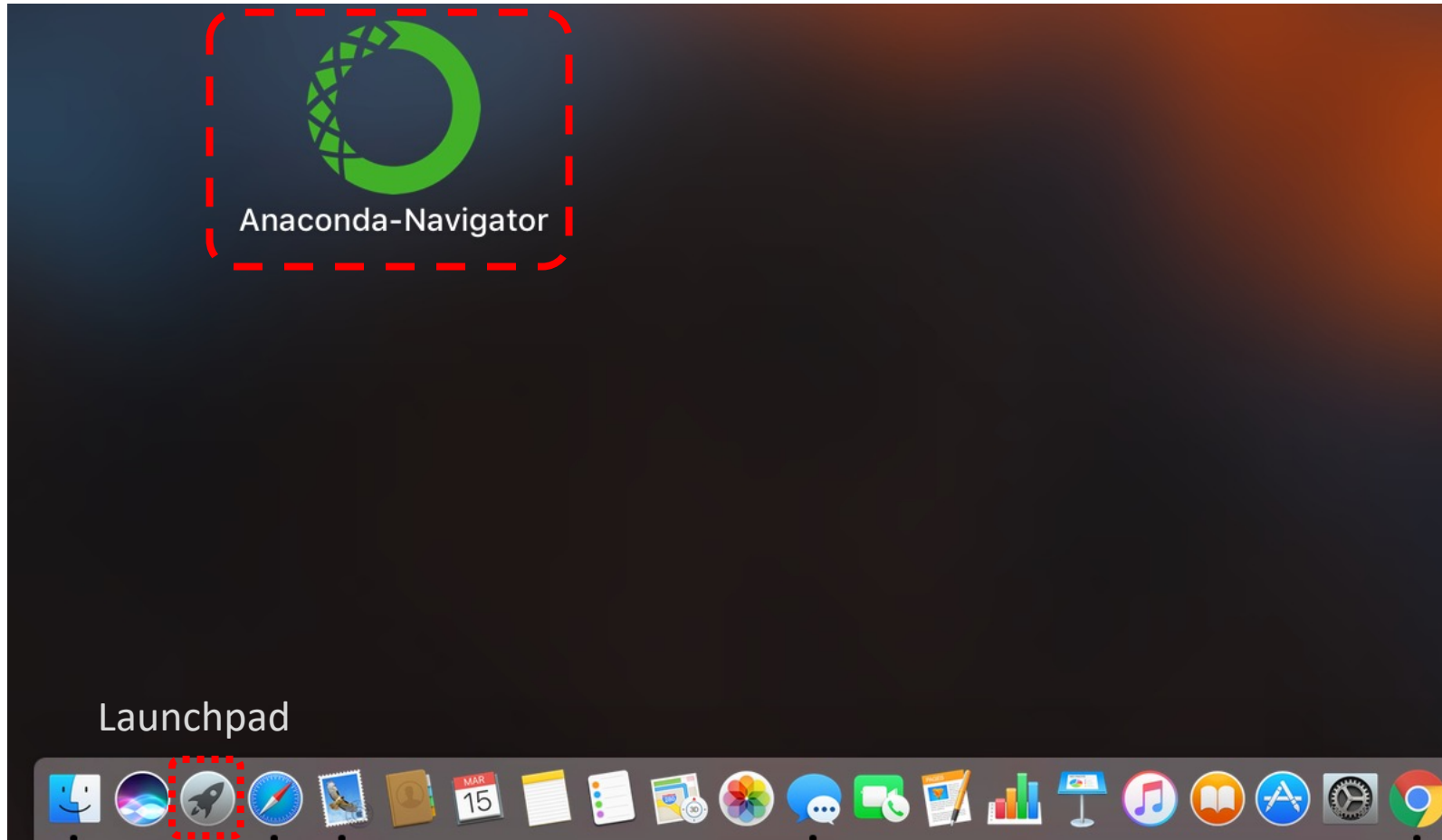




Python

HelloWorld

Anaconda-Navigator



Anaconda Navigator

The screenshot displays the Anaconda Navigator desktop application. At the top, the title bar reads "Anaconda Navigator". Below it, the application header features the "ANACONDA NAVIGATOR" logo on the left and a "Sign in to Anaconda Cloud" button on the right. A left-hand sidebar contains navigation options: "Home", "Environments", "Learning", and "Community". At the bottom of the sidebar are links for "Documentation", "Developer Blog", and "Feedback", along with social media icons for Twitter, YouTube, and GitHub.

The main content area is titled "Applications on" and shows a dropdown menu set to "base (root)" and a "Channels" button. A "Refresh" button is located in the top right of this section. The applications are arranged in a grid:

- jupyterlab** (0.31.5): An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. [Launch]
- jupyter notebook** (5.4.0): Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. [Launch]
- qtconsole** (4.3.1): PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. [Launch]
- spyder** (3.2.6): Scientific PYTHON Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. [Launch]
- vscode** (1.22.2): Streamlined code editor with support for development operations like debugging, task running and version control. [Launch]
- glueviz** (0.12.4): Multidimensional data visualization across files. Explore relationships within and among related datasets. [Install]

The "jupyter notebook" application card is highlighted with a red dashed border, and its "Launch" button is enclosed in a solid red box.

Jupyter Notebook

The screenshot shows a web browser window with the Jupyter Notebook interface. The address bar shows the URL `localhost:8888/tree/Documents/Data/BDA`. The page title is "jupyter" and there is a "Logout" button in the top right. Below the navigation tabs ("Files", "Running", "Clusters"), there is a message "Select items to perform actions on them." and buttons for "Upload", "New", and a refresh icon. A file browser table is displayed, showing the current directory path as `/ Documents / Data / BDA`. The table has columns for "Name" and "Last Modified". A folder named `..` is listed with a "seconds ago" timestamp. Below the table, a message states "The notebook list is empty." A red dashed box highlights the file browser table and the empty message.

Home x

localhost:8888/tree/Documents/Data/BDA

jupyter Logout

Files Running Clusters

Select items to perform actions on them. Upload New ↕ ↻

<input type="checkbox"/> 0	<code>/ Documents / Data / BDA</code>	Name ↓	Last Modified
<input type="checkbox"/>	<code>..</code>		seconds ago

The notebook list is empty.

Jupyter Notebook

New Python 3

The screenshot shows a web browser window with the Jupyter Notebook interface. The browser's address bar displays `localhost:8888/tree/Documents/Data/BDA`. The Jupyter logo and a 'Logout' button are visible at the top. Below the navigation tabs ('Files', 'Running', 'Clusters'), there is a prompt: 'Select items to perform actions on them.' To the right of this prompt are 'Upload', 'New', and a refresh icon. The 'New' dropdown menu is open, showing options under 'Notebook:' (with 'Python 3' highlighted) and 'Other:' (with 'Text File', 'Folder', and 'Terminal'). The file browser below shows the path `/ Documents / Data / BDA` and a message: 'The notebook list is empty.'

print("hello, world")

The screenshot shows a web browser window with two tabs: 'Home' and 'HelloWorld'. The address bar shows the URL 'localhost:8888/notebooks/Documents/Data/BDA/HelloWorld.ipynb'. The Jupyter Notebook interface is displayed, with the title 'jupyter HelloWorld (autosaved)' and a 'Logout' button. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The 'Trusted' status and 'Python 3' kernel are also visible. The toolbar contains various icons, with the 'Run' button (a play icon) highlighted with a red box. Below the toolbar, a code cell is shown with the input 'In [1]: print("hello, world")' and the output 'hello, world'. The code cell is also highlighted with a red box. Below the code cell, there is an empty input field labeled 'In []:'.





Python

Programming

Foundations of Python Programming

- **Python Syntax**
 - **Python Comments**
- **Python Variables**
- **Python Data Types**
 - **Python Numbers**
 - **Python Casting**
 - **Python Strings**
- **Python Operators**
- **Python Booleans**

Python Hello World

```
print("Hello World")
```

```
print("Hello World")
```

Python Syntax

comment

```
# comment
```

Python Syntax

Indentation

the spaces at the beginning of a code line
4 spaces

```
score = 80
if score >= 60 :
    print("Pass")
```

Python Variables

```
# Python Variables  
x = 2  
price = 2.5  
word = 'Hello'  
  
word = 'Hello'  
word = "Hello"  
word = '''Hello'''
```

Python Variables

```
x = 2
```

```
y = x + 1
```

python_version()

```
# comment  
from platform import python_version  
print("Python Version:", python_version())
```

Python Version: 3.10.12

Python Data Types

```
x = "Hello World"    #str
x = 2                #int
x = 2.5              #float
x = 7j                #complex
```


Python Data Types

```
x = ["apple", "banana", "cherry"] #list
x = ("apple", "banana", "cherry") #tuple
x = range(6) #range
x = {"name" : "Tom", "age" : 20} #dict
x = {"apple", "banana", "cherry"} #set
x = frozenset({"apple", "banana", "cherry"})
#frozenset
```

Python Data Types

```
x = True #bool
x = b"Hello" #bytes
x = bytearray(5) #bytearray
x = memoryview(bytes(5)) #memoryview
x = None #NoneType
```

Python Casting

```
x = str(3) # x will be '3'  
y = int(3) # y will be 3  
z = float(3) # z will be 3.0  
print(x, type(x))  
print(y, type(y))  
print(z, type(z))
```

```
3 <class 'str'>  
3 <class 'int'>  
3.0 <class 'float'>
```

Python Numbers

```
x = 2 # int
y = 3.4 # float
z = 7j #complex
print(x, type(x))
print(y, type(y))
print(z, type(z))
```

```
2 <class 'int'>
3.4 <class 'float'>
7j <class 'complex'>
```

Python Arithmetic Operators

Operator	Name	Example
+	Addition	$7 + 2 = 9$
-	Subtraction	$7 - 2 = 5$
*	Multiplication	$7 * 2 = 14$
/	Division	$7 / 2 = 3.5$
//	Floor division	$7 // 2 = 3$ (Quotient)
%	Modulus	$7 \% 2 = 1$ (Remainder)
**	Exponentiation	$7 ** 2 = 49$

Python Basic Operators

```
print('7 + 2 =', 7 + 2)
print('7 - 2 =', 7 - 2)
print('7 * 2 =', 7 * 2)
print('7 / 2 =', 7 / 2)
print('7 // 2 =', 7 // 2)
print('7 % 2 =', 7 % 2)
print('7 ** 2 =', 7 ** 2)
```

7 + 2 = 9
7 - 2 = 5
7 * 2 = 14
7 / 2 = 3.5
7 // 2 = 3
7 % 2 = 1
7 ** 2 = 49

Python Booleans: True or False

```
# Python Booleans: True or False  
print(3 > 2)  
print(3 == 2)  
print(3 < 2)
```

Python BMI Calculator

```
# BMI Calculator in Python
height_cm = 170
weight_kg = 60
height_m = height_cm/100
BMI = (weight_kg/(height_m**2))

print("Your BMI is: " + str(round(BMI,1)))
```

Your BMI is: 20.8

Future value
of a specified
principal amount,
rate of interest, and
a number of years

How much is your \$100 worth after 7 years?

```
# How much is your $100 worth after 7 years?  
fv = 100 * 1.1 ** 7  
print('fv = ', round(fv, 2))  
# output = 194.87
```

```
fv = 194.87
```

Future Value

```
# Future Value
pv = 100
r = 0.1
n = 7

fv = pv * ((1 + (r)) ** n)
print(round(fv, 2))
```

194.87

Future Value

```
# Future Value
amount = 100
interest = 10 #10% = 0.01 * 10
years = 7

future_value = amount * ((1 + (0.01 * interest)) ** years)
print(round(future_value, 2))
```

194.87

Python

Data Structures

Python Data Structures

- Python Lists []
- Python Tuples ()
- Python Sets {}
- Python Dictionaries {k:v}

Python Data Structures

```
fruits = ["apple", "banana", "cherry"] #lists []
colors = ("red", "green", "blue") #tuples ()
animals = {'cat', 'dog'} #sets {}
person = {"name" : "Tom", "age" : 20} #dictionaries {}
```

Python Data Types

```
x = ["apple", "banana", "cherry"] #list
x = ("apple", "banana", "cherry") #tuple
x = {"name" : "Tom", "age" : 20} #dict
x = {"apple", "banana", "cherry"} #set
```


Python Collections

- **There are four collection data types in the Python programming language**
- **List []**
 - **a collection which is ordered and changeable. Allows duplicate members.**
- **Tuple ()**
 - **a collection which is ordered and unchangeable. Allows duplicate members.**
- **Set {}**
 - **a collection which is unordered, unchangeable, and unindexed. No duplicate members.**
- **Dictionary {k:v}**
 - **a collection which is ordered and changeable. No duplicate members.**

Python Dictionaries {k:v}

- **As of Python version 3.7, dictionaries are ordered.**
- **In Python 3.6 and earlier, dictionaries are unordered.**

Lists []

```
x = [60, 70, 80, 90]
print(len(x))
print(x[0])
print(x[1])
print(x[-1])
```

4
60
70
90

Lists []

- **len():** how many items
- **type():** data type
- **list() constructor:** creating a new list

Python List Methods

• Method	Description
• <code>append()</code>	Adds an element at the end of the list
• <code>clear()</code>	Removes all the elements from the list
• <code>copy()</code>	Returns a copy of the list
• <code>count()</code>	Returns the number of elements with the specified value
• <code>extend()</code>	Add the elements of a list (or any iterable), to the end of the current list
• <code>index()</code>	Returns the index of the first element with the specified value
• <code>insert()</code>	Adds an element at the specified position
• <code>pop()</code>	Removes the element at the specified position
• <code>remove()</code>	Removes the item with the specified value
• <code>reverse()</code>	Reverses the order of the list
• <code>sort()</code>	Sorts the list

Tuples ()

A **tuple** in Python is a collection that **cannot be modified**.
A tuple is defined using **parenthesis**.

```
x = (10, 20, 30, 40, 50)
print(x[0])           10
print(x[1])           20
print(x[2])           30
print(x[-1])          50
```

Sets {}

```
animals = {'cat', 'dog'}
print('cat' in animals)      True
print('fish' in animals)    False
animals.add('fish')
print('fish' in animals)    True
print(len(animals))         3
animals.add('cat')
print(len(animals))         3
animals.remove('cat')
print(len(animals))         2
```

Dictionary {key : value}

Python Dictionary

Key → Value

'EN' → 'English'

'FR' → 'French'

```
k = { 'EN': 'English', 'FR': 'French' }  
print(k['EN'])
```

English

Python Data Structures

```
fruits = ["apple", "banana", "cherry"] #lists []
colors = ("red", "green", "blue") #tuples ()
animals = {'cat', 'dog'} #sets {}
person = {"name" : "Tom", "age" : 20} #dictionaries {}
```

Python Control Logic and Loops

Python Control Logic and Loops

- Python **if else**
 - **if elif else**
 - Booleans: True, False
 - Operators: ==, !=, >, <, >=, <=, and, or, not
- Python **for** Loops
 - **for**
- Python **while** Loops
 - **While**
 - break
 - continue

Python **if...else**

- **Python if...else**
 - **if elif else**
 - **Booleans: True, False**
 - **Operators: ==, !=, >, <, >=, <=, and, or, not**

Python Conditions and **If** statements

- Python supports the usual **logical conditions** from mathematics:
 - Equals: $a == b$
 - Not Equals: $a != b$
 - Less than: $a < b$
 - Less than or equal to: $a <= b$
 - Greater than: $a > b$
 - Greater than or equal to: $a >= b$

Python Comparison Operators

Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>></code>	Greater than	<code>x > y</code>
<code><</code>	Less than	<code>x < y</code>
<code>>=</code>	Greater than or equal to	<code>x >= y</code>
<code><=</code>	Less than or equal to	<code>x <= y</code>

Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Python if

```
# Python if
score = 80
if score >= 60 :
    print ("Pass")
```


Python if else

```
# Python if else
score = 80
if score >= 60 :
    print("Pass")
else:
    print("Fail")
```

Python if elif else

```
score = 95
if score >= 90 :
    print("A")
elif score >=60 :
    print("Pass")
else:
    print("Fail")
```

Python if elif else

```
# Python if elif else
score = 90
grade = ""
if score >=90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "E"
print(grade)
```

Python **for** Loops

```
for i in range(1, 6):  
    print(i)
```

1
2
3
4
5

Python **for** loops

```
# for loops
for i in range(1,10):
    for j in range(1,10):
        print(i, ' * ', j, ' = ', i*j)
```

Python **while** Loops

- **while**
 - **break**
 - **continue**

Python **while** loops

```
# while loops
age = 10
while age < 20:
    print(age)
    age = age + 1
```

Python Functions and Modules

Python Functions and Modules

- Python Functions
 - **def** myfunction():
- Python Classes/Objects
 - **class** MyClass:
- Python Modules
 - mymodule.py
 - **import** mymodule

Python Functions

Python Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- Creating a Function
 - In Python a function is defined using the **def** keyword:

Python Function def

```
# Python Function def
# indentation for blocks. four spaces
def getfv(pv, r, n):
    fv = pv * ((1 + (r)) ** n)
    return fv
fv = getfv(100, 0.1, 7)
print(round(fv, 2))
```

194.87

Future value
of a specified
principal amount,
rate of interest, and
a number of years

How much is your \$100 worth after 7 years?

```
# How much is your $100 worth after 7 years?  
fv = 100 * 1.1 ** 7  
print('fv = ', round(fv, 2))  
# output = 194.87
```

```
fv = 194.87
```

Future Value

```
# Future Value
pv = 100
r = 0.1
n = 7

fv = pv * ((1 + (r)) ** n)
print(round(fv, 2))
```

194.87

Future Value

```
# Future Value
amount = 100
interest = 10 #10% = 0.01 * 10
years = 7

future_value = amount * ((1 + (0.01 * interest)) ** years)
print(round(future_value, 2))
```

194.87

Python Function

`def` `getfv()` **define get future value function**

```
# Python Function def
# indentation for blocks. four spaces
def getfv(pv, r, n):
    fv = pv * ((1 + (r)) ** n)
    return fv
fv = getfv(100, 0.1, 7)
print(round(fv, 2))
```

194.87

Python
Classes/Objects
class MyClass :

Python Classes/Objects

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- Create a Class:
 - To create a class, use the keyword **class**:

Python Classes/Objects

class MyClass:

```
# Python class
class MyClass:
    x = 5

c1 = MyClass()
print(c1.x)
```

Python Classes/Objects

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

```
p1 = Person("Alan", 20)
```

```
print(p1.name)
```

```
print(p1.age)
```

Alan

20

Python Classes/Objects

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("Alan", 20)
p1.myfunc()
```

Python Classes/Objects

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)
```

```
p1 = Person("Alan", 20)
p1.myfunc()
print(p1.name)
print(p1.age)
```

```
Hello my name is Alan
Alan
20
```

Python Classes and Objects

```
class Vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.00
    def description(self):
        desc_str = "%s is a %s %s worth $%.2f." %
(self.name, self.color, self.kind, self.value)
        return desc_str
```


Python Classes and Objects

```
car1 = Vehicle()
car1.name = "Fer"
car1.color = "red"
car1.kind = "convertible"
car1.value = 60000.00

car2 = Vehicle()
car2.name = "Jump"
car2.color = "blue"
car2.kind = "van"
car2.value = 10000.00

print(car1.description())
print(car1.name)
print(car2.description())
print(car2.name)
```

```
class Vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.00
    def description(self):
        desc_str = "%s is a %s %s
worth $%.2f." % (self.name, self.color,
self.kind, self.value)
        return desc_str
```

```
Fer is a red convertible worth $60000.00.
Fer
Jump is a blue van worth $10000.00.
Jump
```

Python Modules

Python Modules

- Consider a **module** to be the same as a **code library**.
- A file containing a set of functions you want to include in your application.
- Create a Module
 - To create a **module** just save the code you want in a **file** with the file extension **.py**:
- Use a Module
 - **import** module

Python Modules

```
# mymodule.py
def greeting(name):
    print("Hello, " + name)
```

```
import mymodule
mymodule.greeting("Alan")
```

```
mymodule.py
def greeting(name):
    print("Hello, " + name)
```

Python File Input / Output

```
# Python File Input / Output
with open('myfile.txt', 'w') as file:
    file.write('Hello World\nThis is Python File Input Output')

with open('myfile.txt', 'r') as file:
    text = file.read()
    print(text)
```

Hello World This is Python File Input Output

Python File Input / Output

```
# Python File Input / Output
filename = 'mymodule.py'
with open(filename, 'w') as file:
    text = '''def greeting(name):
print("Hello, " + name)
'''
    file.write(text)

with open(filename, 'r') as file:
    text = file.read()
print(filename)
print(text)
```

```
mymodule.py
def greeting(name):
    print("Hello, " + name)
```

Python Modules

```
import mymodule
```

```
# mymodule.py  
def greeting(name):  
    print("Hello, " + name)
```

```
import mymodule  
mymodule.greeting("Alan")
```

Hello, Alan

Python `main()` function

```
#Python main() function
def main():
    print("Hello World!")

if __name__ == "__main__":
    main()
```


Files and Exception Handling

Files and Exception Handling

- **Python Files (File Handling)**
 - **open()**
 - **f = open("myfile.txt")**
- **Python Try Except (Exception Handling)**
 - **try:**
 - **except:**
 - **else:**
 - **finally:**

File Handling

- The key function for working with files in Python is the **open()** function.
- The **open()** function takes two parameters; **filename**, and **mode**.
- There are four different methods (modes) for opening a file:
 - **"r" - Read** - Default value. Opens a file for reading, error if the file does not exist
 - **"a" - Append** - Opens a file for appending, creates the file if it does not exist
 - **"w" - Write** - Opens a file for writing, creates the file if it does not exist
 - **"x" - Create** - Creates the specified file, returns an error if the file exists

Python Files (File Handling)

```
f = open("myfile.txt", "w")  
f.write("Hello World")  
f.close()
```

```
f = open("myfile.txt", "r")  
text = f.read()  
print(text)  
f.close()
```

Hello World

Python Files (File Handling)

```
# Python File Input / Output
with open('myfile.txt', 'w') as file:
    file.write('Hello World')

with open('myfile.txt', 'r') as file:
    text = file.read()
print(text)
```

Hello World

Python Files

```
# Python File Input / Output
with open('myfile.txt', 'w') as file:
    file.write('Hello World\nPython File IO')

with open('myfile.txt', 'r') as file:
    text = file.read()
print(text)
```

Hello World

Python File IO

Python Files

```
# Python File Input / Output
with open('myfile.txt', 'a+') as file:
    file.write('\n' + 'New line')

with open('myfile.txt', 'r') as file:
    text = file.read()
print(text)
```

Hello World

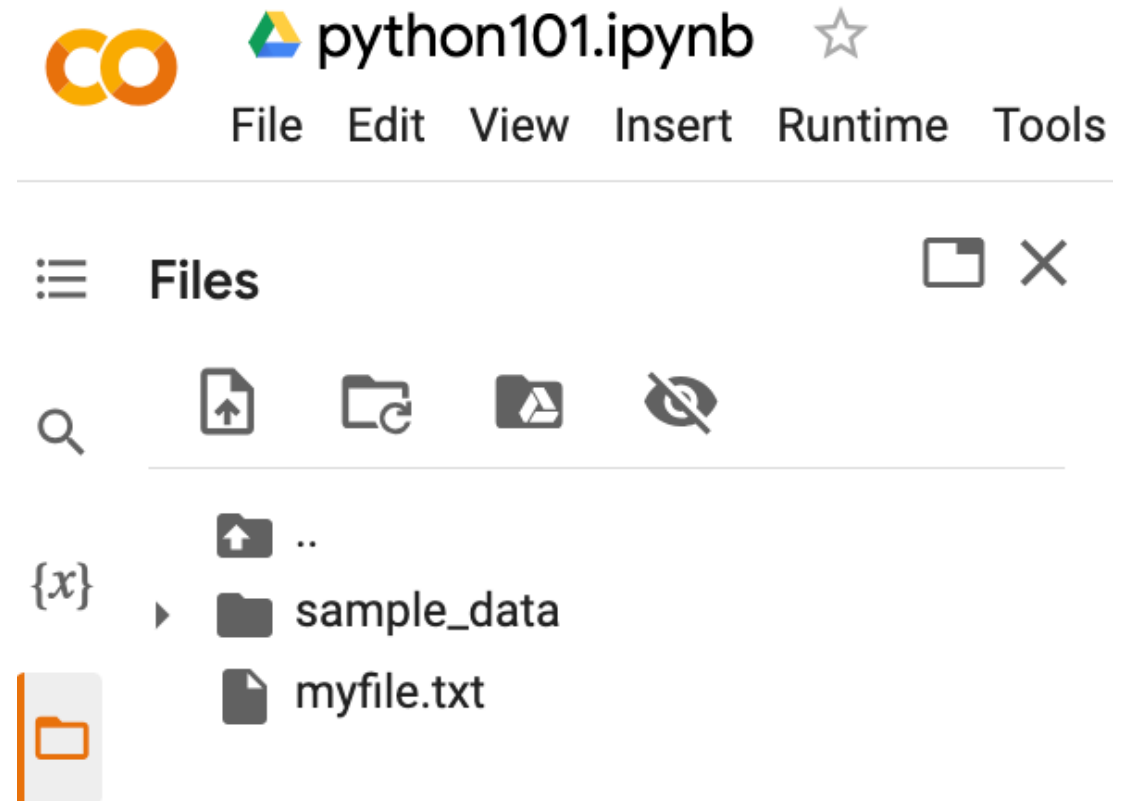
Python File IO

New line

Python Files

```
# !ls list files  
!ls
```

```
myfile.txt sample_data
```



The screenshot shows a Jupyter Notebook interface for a file named 'python101.ipynb'. The top navigation bar includes the 'CO' logo, the notebook name, a star icon, and a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', and 'Tools'. Below the navigation bar is a 'Files' sidebar with a search icon, a refresh icon, a folder icon, and a close icon. The file list shows a folder named 'sample_data' and a file named 'myfile.txt'. A folder icon is highlighted in the sidebar.

Python OS, IO, files, and Google Drive

```
import os
```

```
cwd = os.getcwd()
```

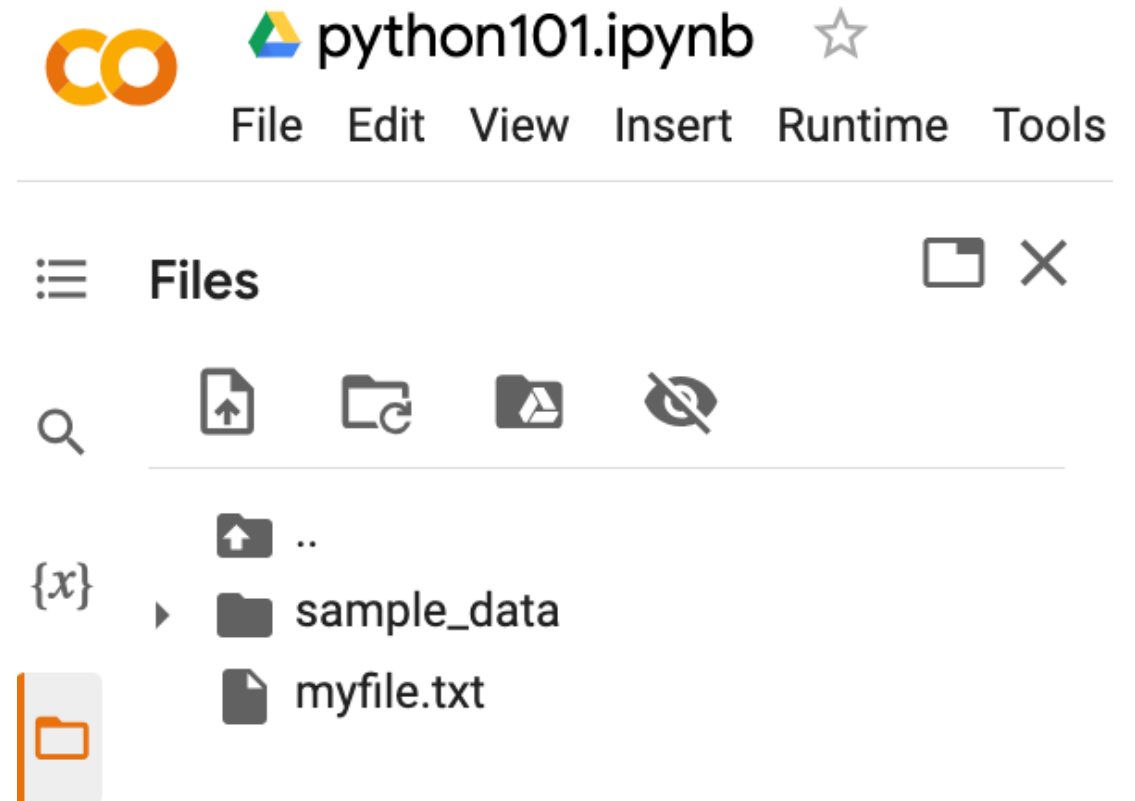
```
print(cwd)
```

/content

os.listdir()

```
os.listdir(cwd)
```

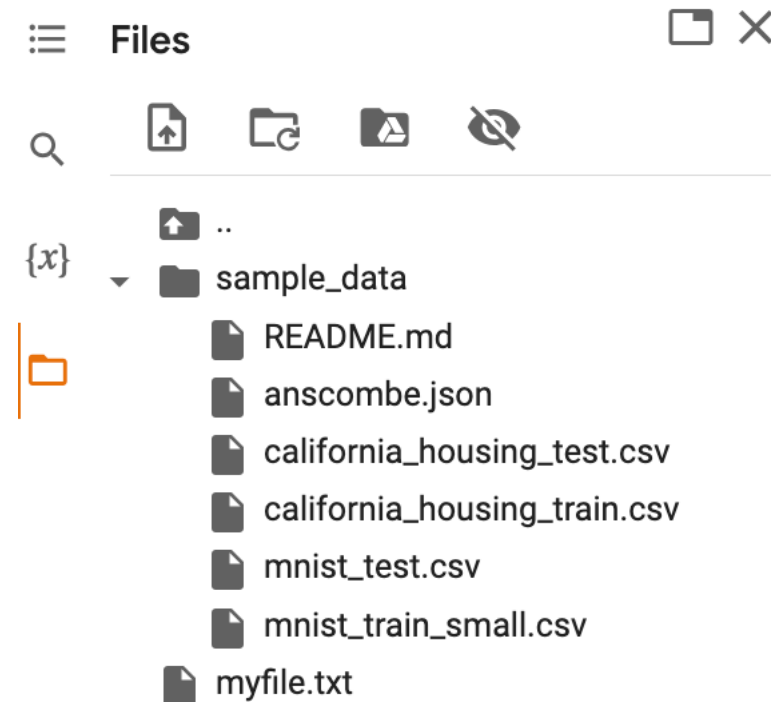
```
['.config',  
'myfile.txt',  
'sample_data']
```



os.path.join()

```
path = os.path.join(cwd, 'sample_data')
print(path)
os.listdir(path)
```

```
/content/sample_data
['README.md', 'anscombe.json',
'mnist_train_small.csv',
'mnist_test.csv',
'california_housing_train.csv',
'california_housing_test.csv']
```



from google.colab import files

```
from google.colab import files

with open('io_file_myday.txt', 'w') as f:
    f.write('Google Colab File Write Text some content Myday')

import time
time.sleep(1) # time sleep 1 second

files.download('io_file_myday.txt')
print('downloaded')
```

downloaded

Python Files

```
from google.colab import files
uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}"
with length {length} bytes'.format(
name=fn, length=len(uploaded[fn])))
```

User uploaded file "io_file_myday2.txt" with length 47 bytes

os.remove()

```
import os
if os.path.exists("myfile.txt"):
    os.remove("myfile.txt")
    print("myfile.txt removed")
else:
    print("The file does not exist")
```

myfile.txt removed

```
os.mkdir("myfolder1")  
os.rmdir("myfolder1")
```

```
import os  
os.listdir()  
os.mkdir("myfolder1")  
os.listdir()  
os.rmdir("myfolder1")  
os.listdir()
```

Python Try Except

- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **else** block lets you execute code when there is no error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

Python Try Except (Exception Handling)

try: **except:**

```
#Python try except
try:
    print(x)
except:
    print("Exception Error")
```

Python try: except: finally:

```
#Python try except finally
try:
    print("Hello")
except:
    print("Exception Error")
finally:
    print("Finally process")
```

```
Hello
Finally process
```

Python try: except: else:

```
#Python try except else
try:
    print("Hello")
except:
    print("Exception Error")
else:
    print("No exception")
```

Hello

No exception

Python try: except: else: finally:

```
try:  
    print("Hello")  
except:  
    print("Exception Error")  
else:  
    print("No exception")  
finally:  
    print("Finally process")
```

Hello

No exception

Finally process

Python try: except: else: finally:

```
try:
    price = float(input("Enter the price of the stock (e.g. 10):"))
    shares = int(input("Enter the number of shares (e.g. 2):"))
    total = price * shares
except Exception as e:
    print("Exception error:", str(e))
else:
    print("The total value of the shares is:", total)
finally:
    print("Thank you.")
```

```
Enter the price of the stock (e.g. 10):10
Enter the number of shares (e.g. 2):2
The total value of the shares is: 20.0
Thank you.
```

Python try: except: else: finally:

```
try:  
    file = open("myfile.txt")  
    file.write("Python write file")  
    print("file saved")  
except:  
    print("Exception file Error")
```

Exception file Error

Python try: except: else: finally:

```
try:  
    file = open("myfile.txt")  
    file.write("Python write file")  
    print("file saved")  
except:  
    print("Exception file Error")  
finally:  
    file.close()  
    print("Finally process")
```

```
Exception file Error  
Finally process
```

Python try: except: else: finally:

```
try:  
    file = open("myfile.txt", 'w')  
    file.write("Python write file")  
    print("file saved")  
except:  
    print("Exception file Error")  
finally:  
    file.close()  
    print("Finally process")
```

```
file saved  
Finally process
```


Summary

- Python Functions: **def** myfunction():
- Python Classes/Objects: **class** MyClass:
- Python Modules: **import** mymodule
- Python Files (File Handling): **open()**
- Python Try Except (Exception Handling):
 - **try: except: else: finally:**

References

- Wes McKinney (2022), "Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter", 3rd Edition, O'Reilly Media.
- Aurélien Géron (2023), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 3rd Edition, O'Reilly Media.
- Steven D'Ascoli (2022), Artificial Intelligence and Deep Learning with Python: Every Line of Code Explained For Readers New to AI and New to Python, Independently published.
- Stuart Russell and Peter Norvig (2020), Artificial Intelligence: A Modern Approach, 4th Edition, Pearson.
- Denis Rothman (2024), Transformers for Natural Language Processing and Computer Vision - Third Edition: Explore Generative AI and Large Language Models with Hugging Face, ChatGPT, GPT-4V, and DALL-E 3, 3rd ed. Edition, Packt Publishing
- Ben Auffarth (2023), Generative AI with LangChain: Build large language model (LLM) apps with Python, ChatGPT and other LLMs, Packt Publishing.
- Varun Grover, Roger HL Chiang, Ting-Peng Liang, and Dongsong Zhang (2018), "Creating Strategic Business Value from Big Data Analytics: A Research Framework", Journal of Management Information Systems, 35, no. 2, pp. 388-423.
- Junliang Wang, Chuqiao Xu, Jie Zhang, and Ray Zhong (2022). "Big data analytics for intelligent manufacturing systems: A review." Journal of Manufacturing Systems 62 (2022): 738-752.
- Ramesh Sharda, Dursun Delen, and Efraim Turban (2017), Business Intelligence, Analytics, and Data Science: A Managerial Perspective, 4th Edition, Pearson
- Python Programming, <https://pythonprogramming.net/>
- Python, <https://www.python.org/>
- Python Programming Language, <http://pythonprogramminglanguage.com/>
- Numpy, <http://www.numpy.org/>
- Pandas, <http://pandas.pydata.org/>
- Skikit-learn, <http://scikit-learn.org/>
- W3Schools Python, <https://www.w3schools.com/python/>
- Learn Python, <https://www.learnpython.org/>
- Google's Python Class, <https://developers.google.com/edu/python>
- Harvard University (2024), CS50x 2024 - Lecture 6 - Python, <http://www.youtube.com/watch?v=EHiORDZ31VA>
- Min-Yuh Day (2024), Python 101, <https://tinyurl.com/aintpupython101>