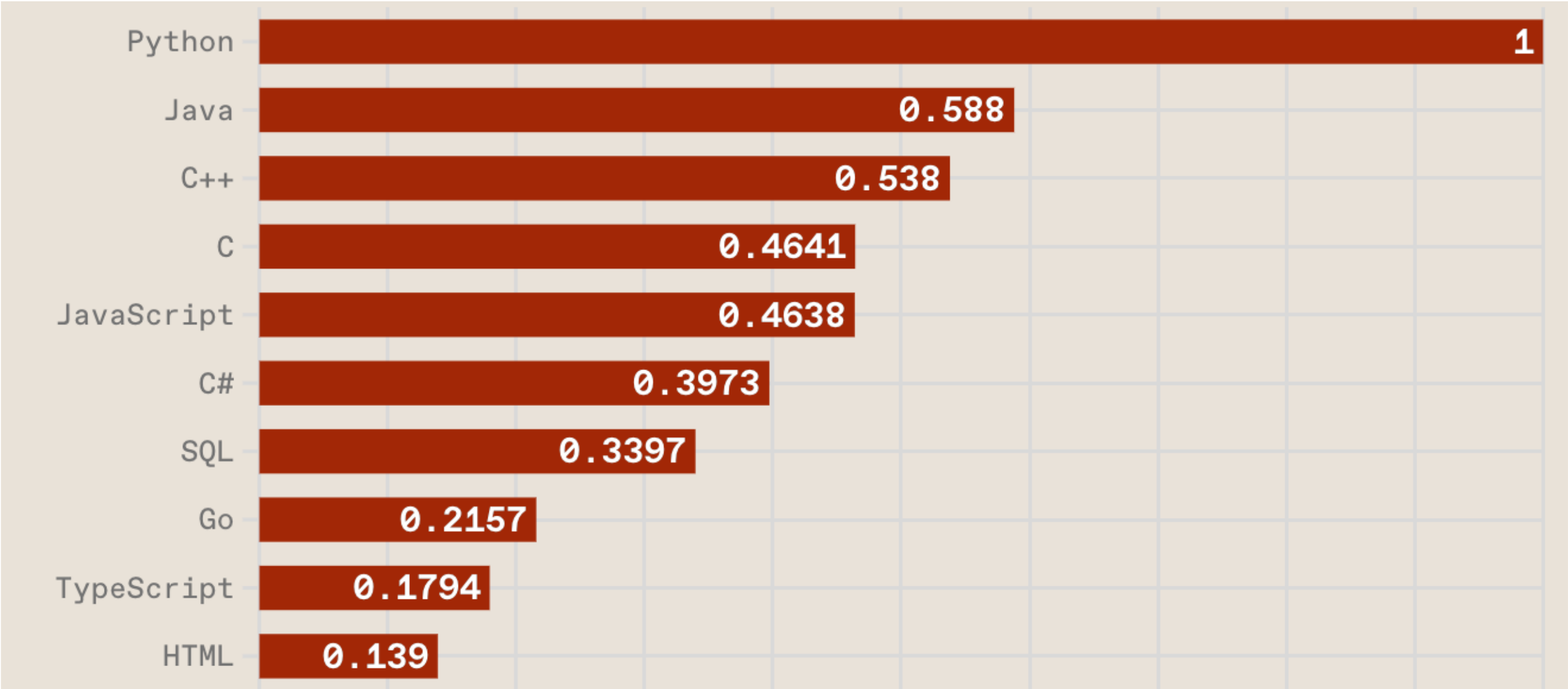
Phân tích dữ liệu

Python Programming

TS. Đỗ Như Tài
dntai@sgu.edu.vn

Top Programming Languages

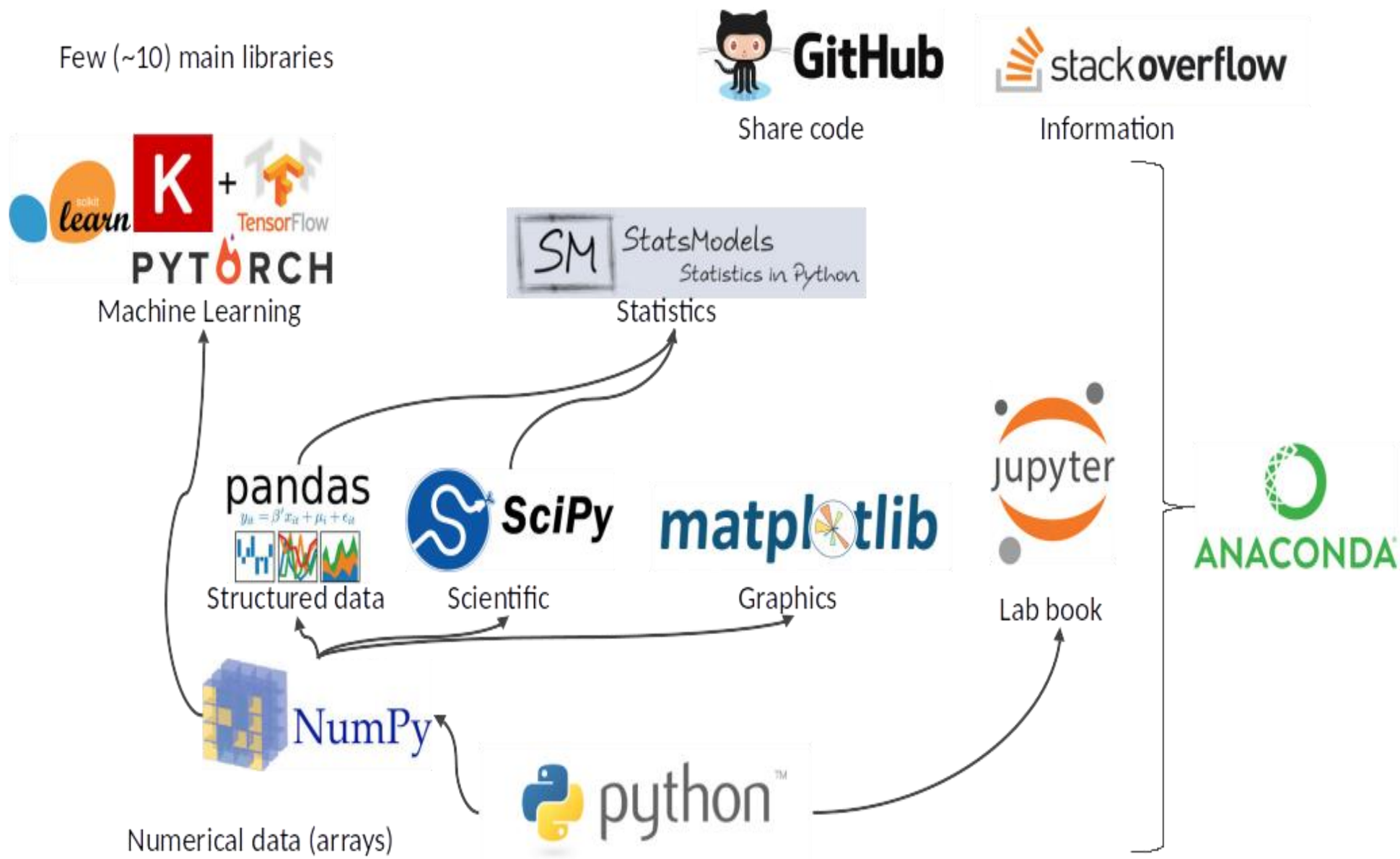


https://spectrum.ieee.org/the-top-programming-languages-2023

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

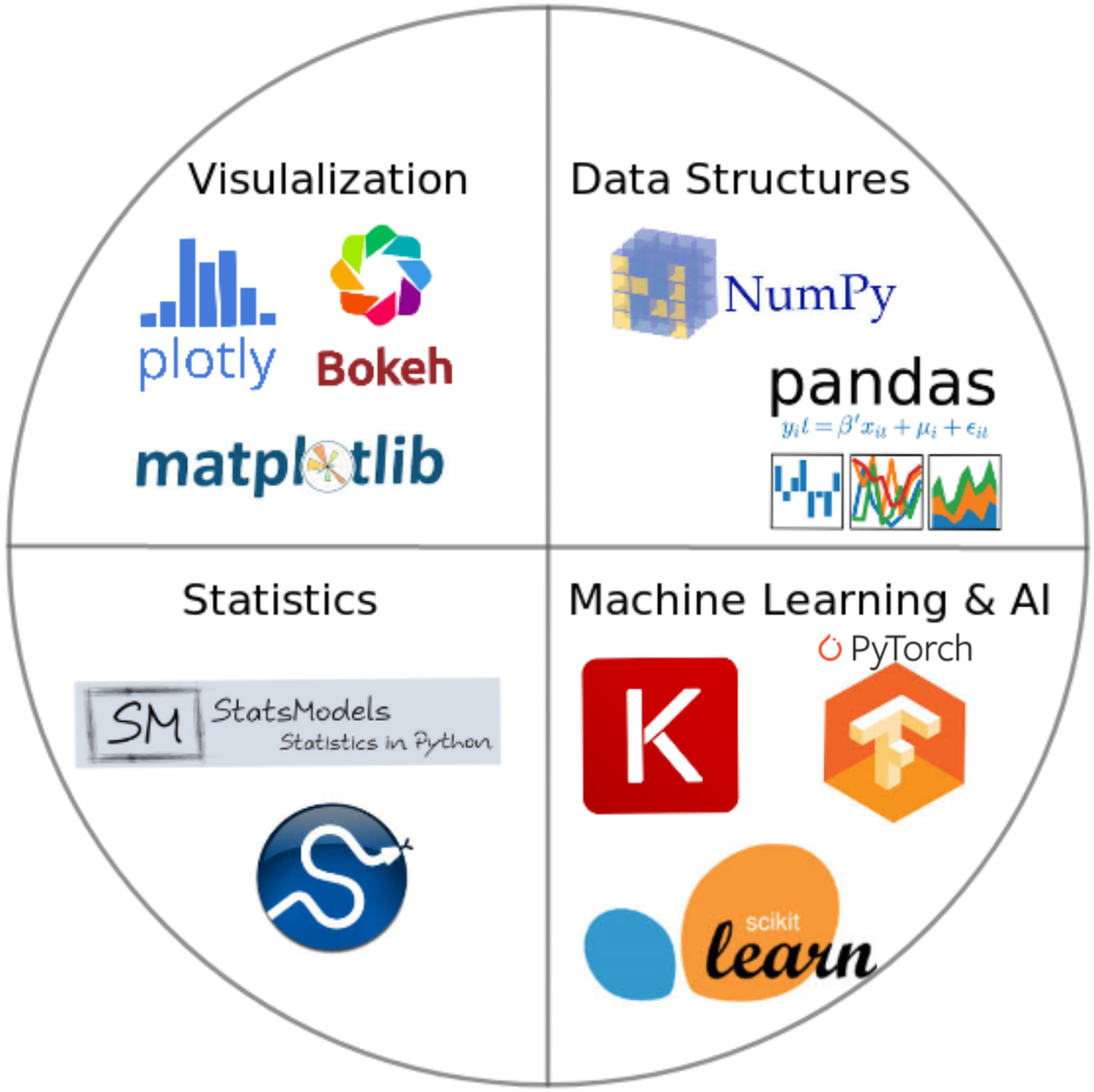
Source: https://www.python.org/doc/essays/blurb/

Python Ecosystem for Data Science



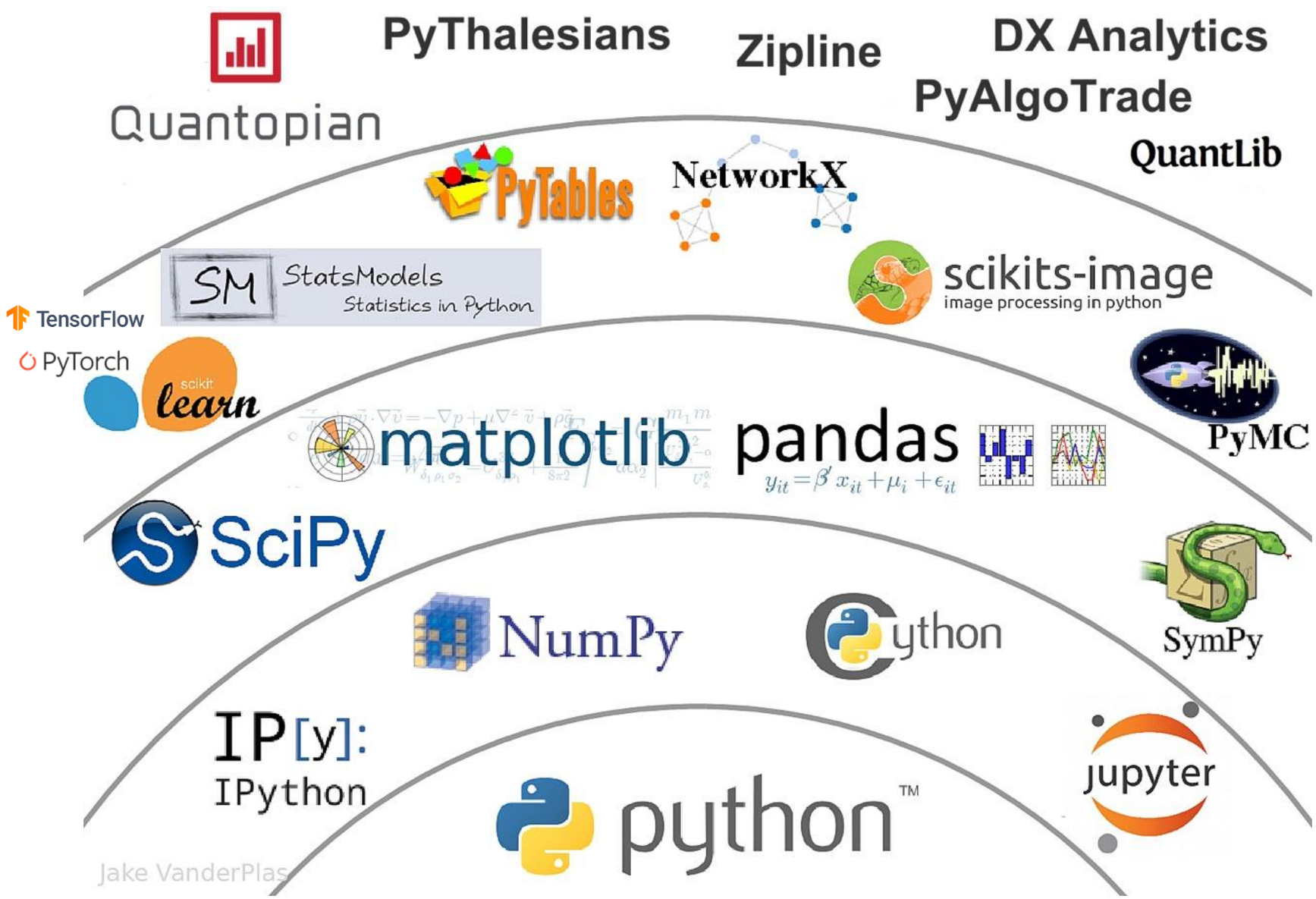
Source: https://medium.com/pyfinance/why-python-is-best-choice-for-financial-data-modeling-in-2019-c0d0d1858c45

Python Ecosystem for Data Science



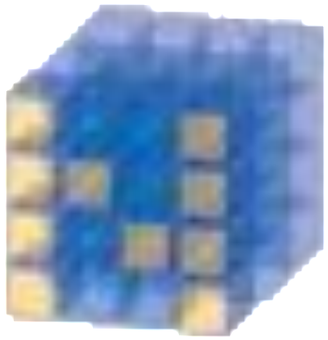
Source:https://duchesnay.github.io/pystatsml/introduction/python_ecosystem.html

The Quant Finance PyData Stack



Source: http://nbviewer.jupyter.org/format/slides/github/quantopian/pyfolio/blob/master/pyfolio/examples/overview_slides.ipynb#/5

Numpy



NumPy
Base
N-dimensional array
package

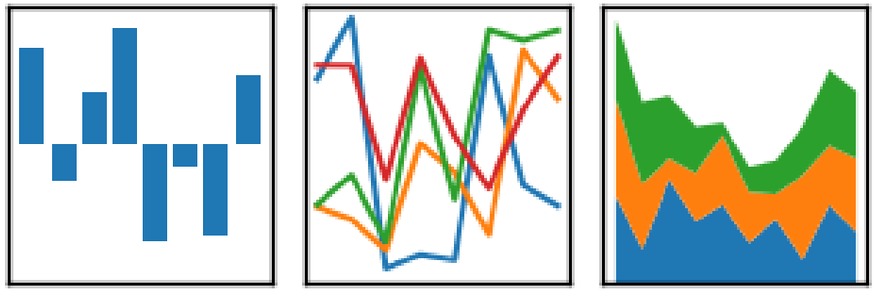
Python matplotlib

Source: https://matplotlib.org/

Python Pandas

pandas

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



http://pandas.pydata.org/

W3Schools Python

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT

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.

https://www.w3schools.com/python/

W3Schools Python

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT

Python Functions

❮ Previous Next ❯

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:

Example

Get your own Python Server

```
def my_function():
  print("Hello from a function")
```

https://www.w3schools.com/python/

W3Schools Python

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT

Python File Open

❮ Previous Next ❯

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

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:

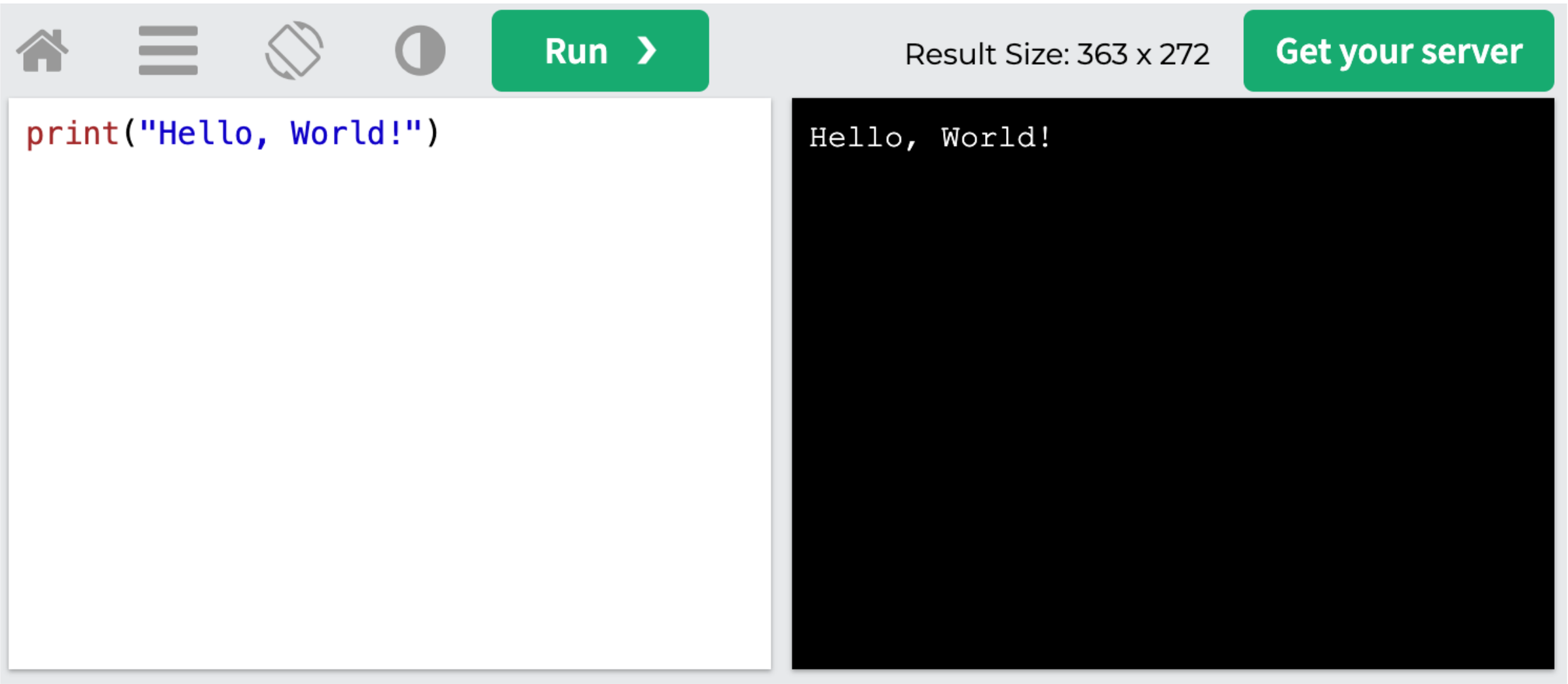
`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"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

https://www.w3schools.com/python/

W3Schools Python: Try Python



https://www.w3schools.com/python/trypython.asp?filename=demo_default

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

Google for Education > Python

Search

English

Filter

Home > Products > Google for Education > Python

Was this helpful?

Google's Python Class

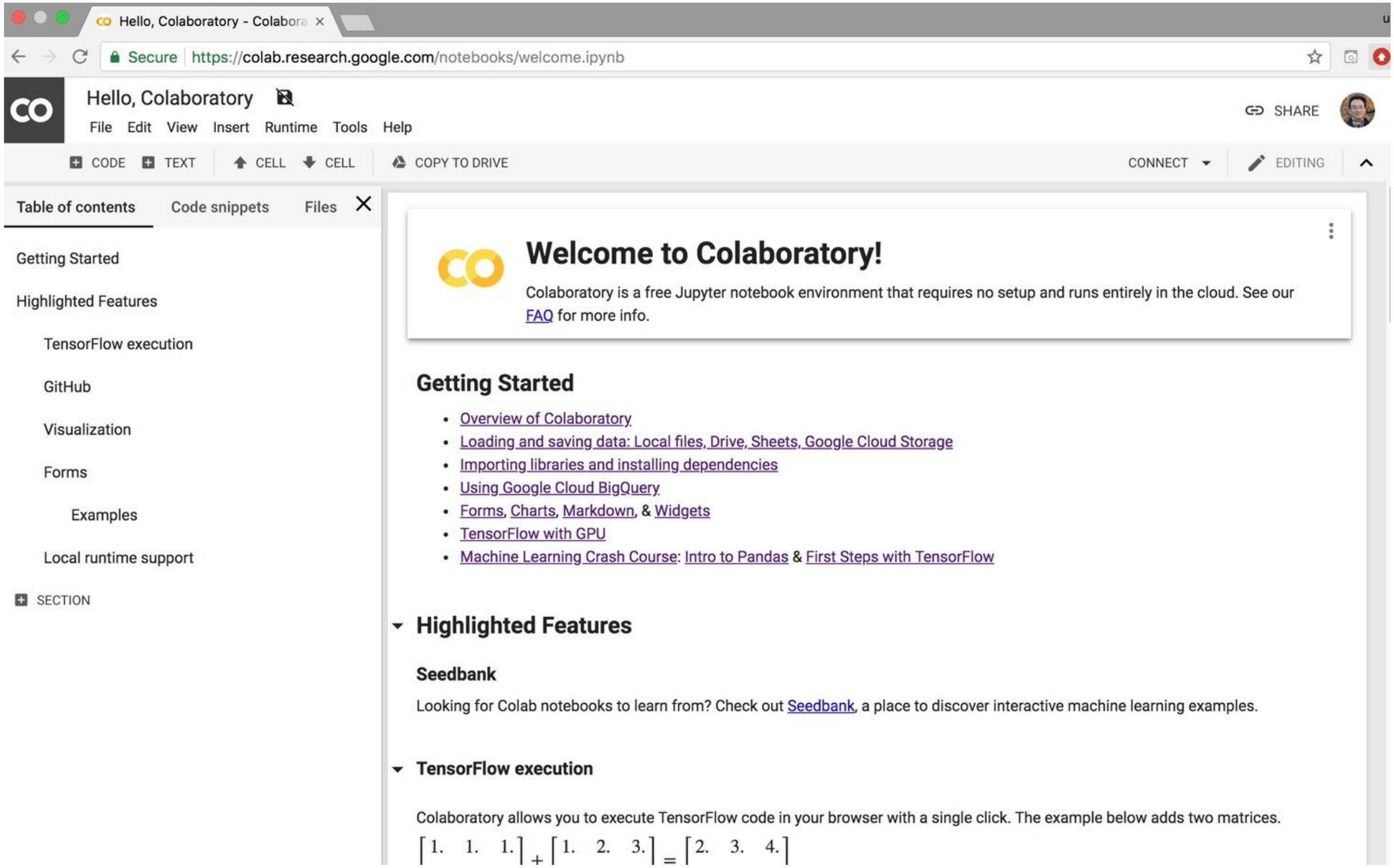
Welcome to Google's Python Class -- this is a free class for people with a little bit of programming experience who want to learn Python. The class includes written materials, lecture videos, and lots of code exercises to practice Python coding. These materials are used within Google to introduce Python 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!

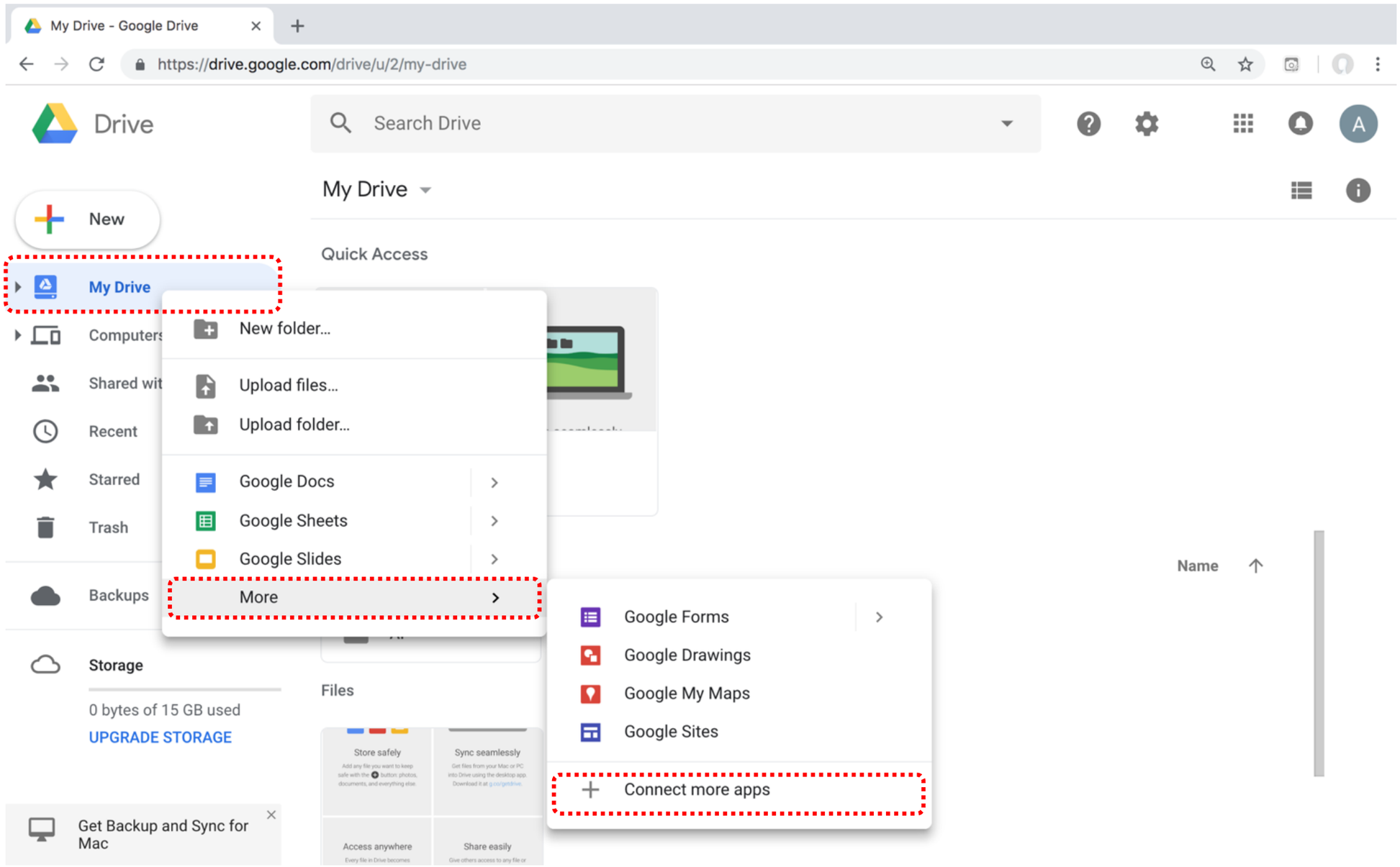
https://developers.google.com/edu/python

Google Colab

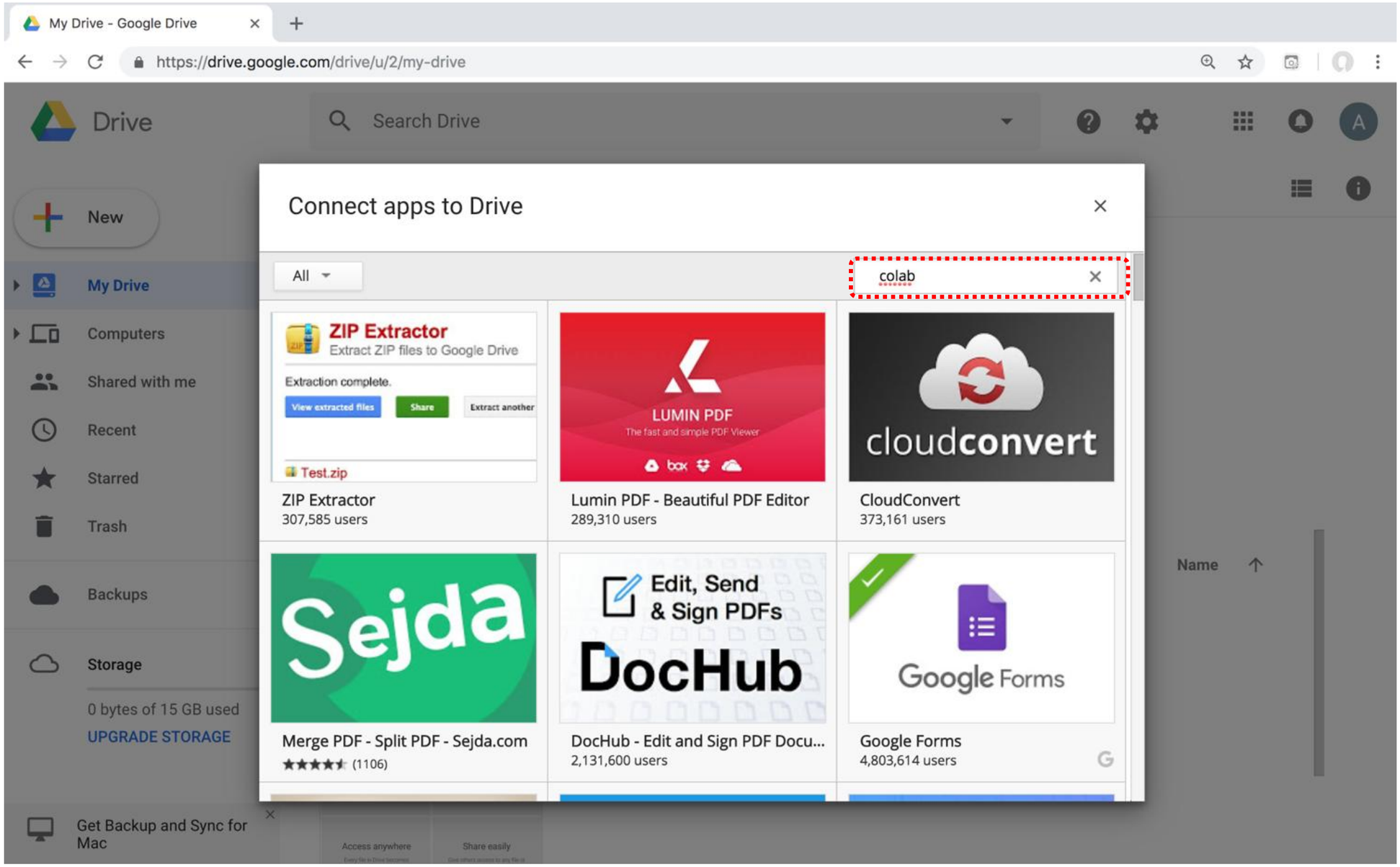


https://colab.research.google.com/notebooks/welcome.ipynb

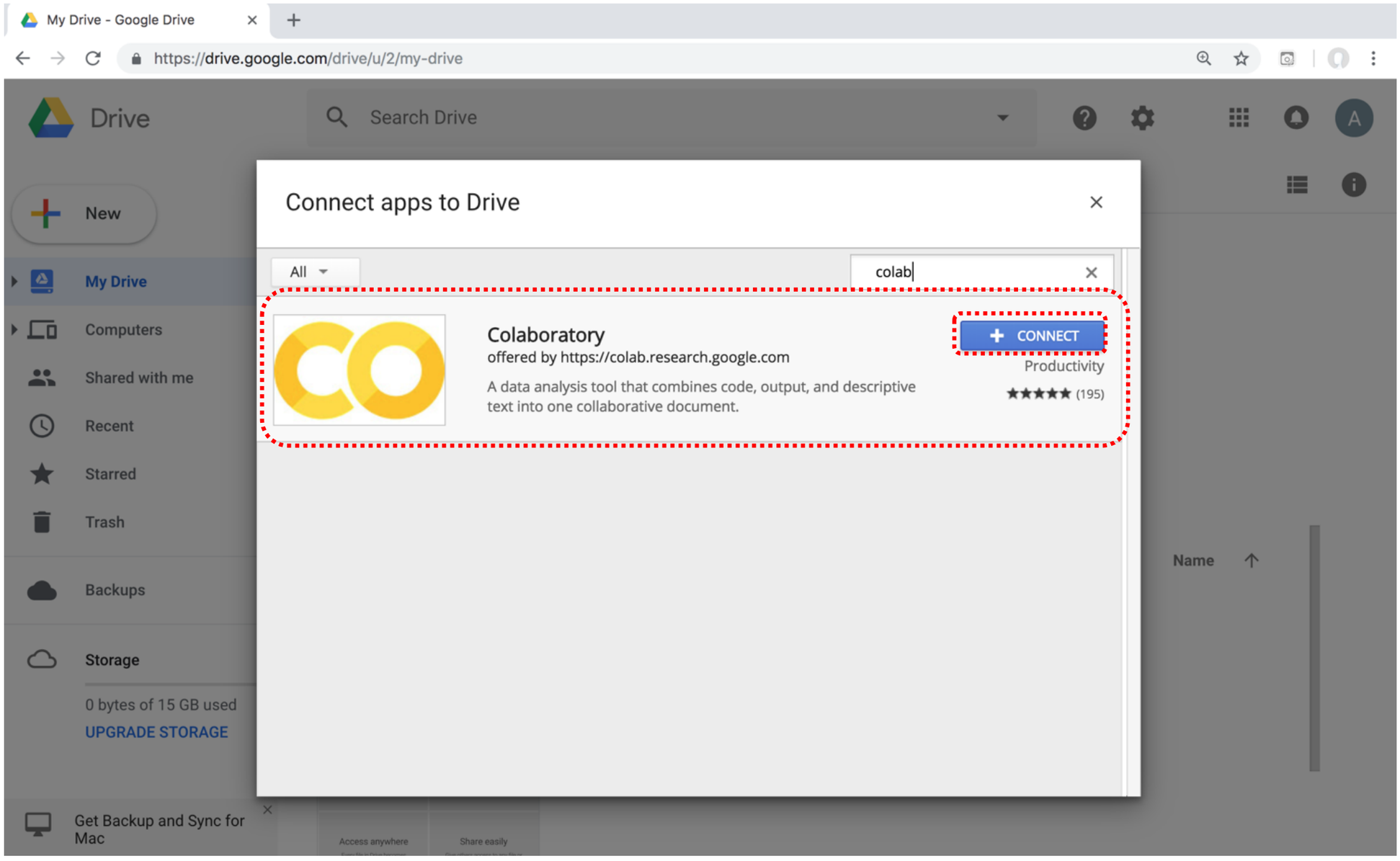
Connect Google Colab in Google Drive



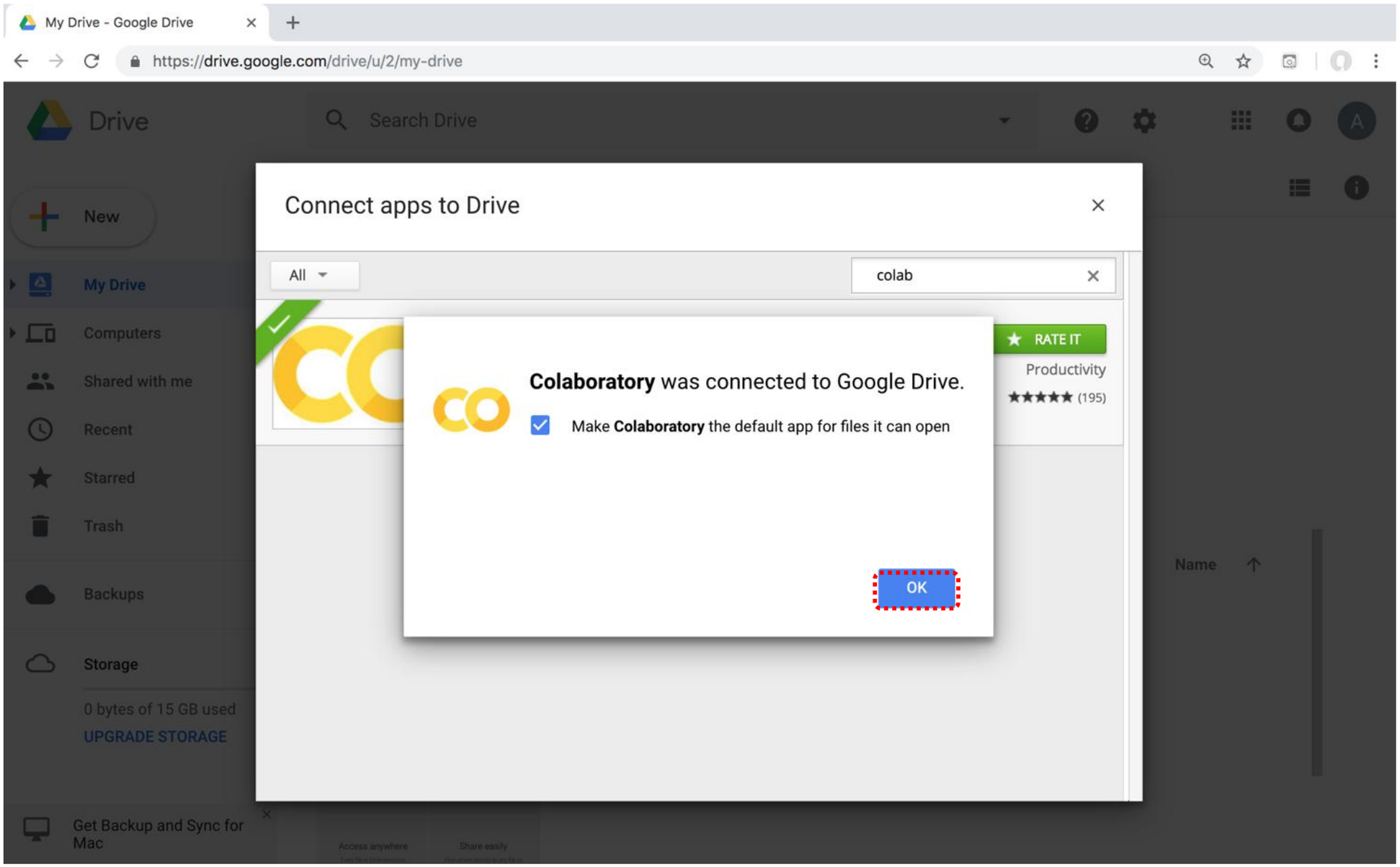
Google Colab



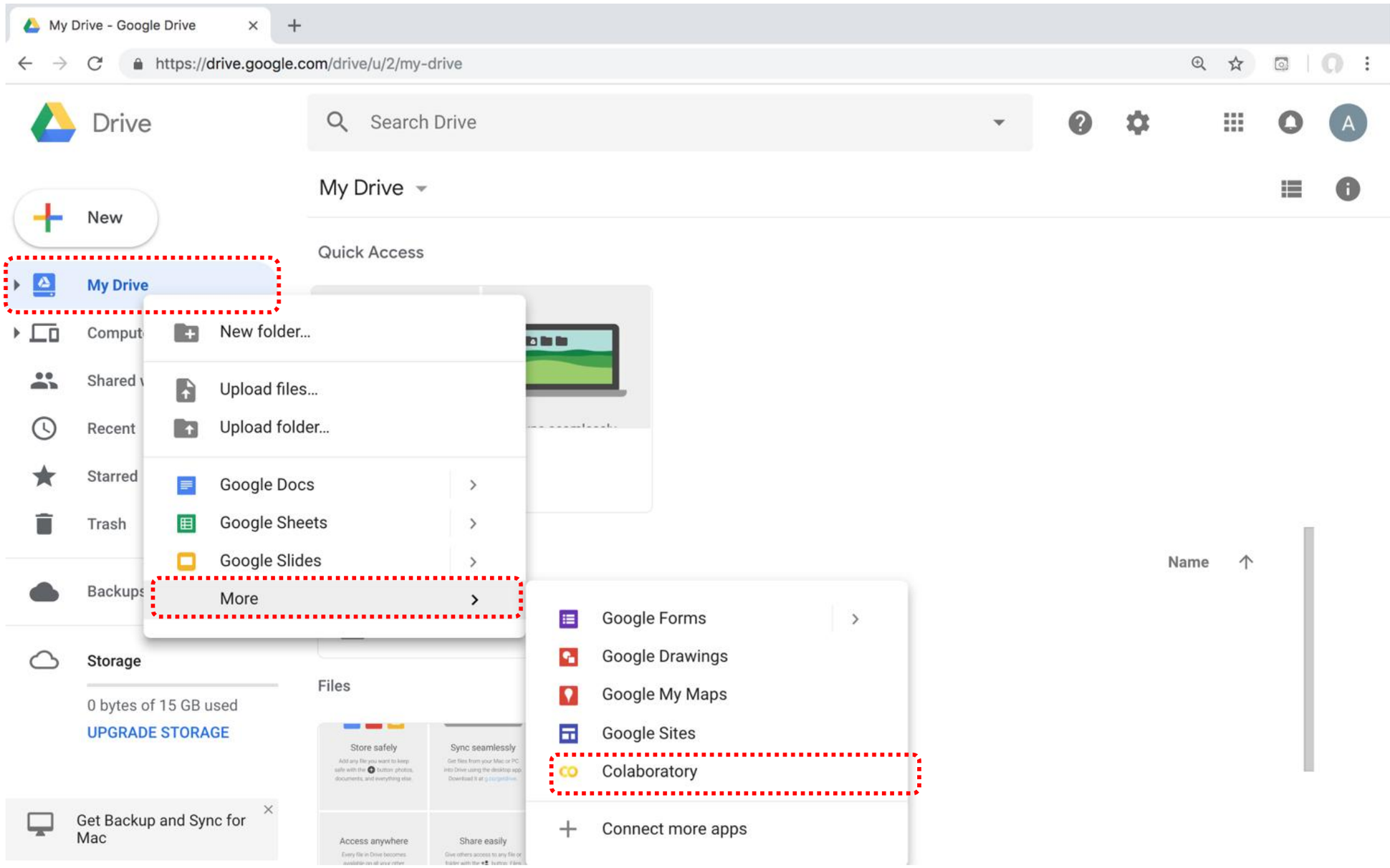
Google Colab



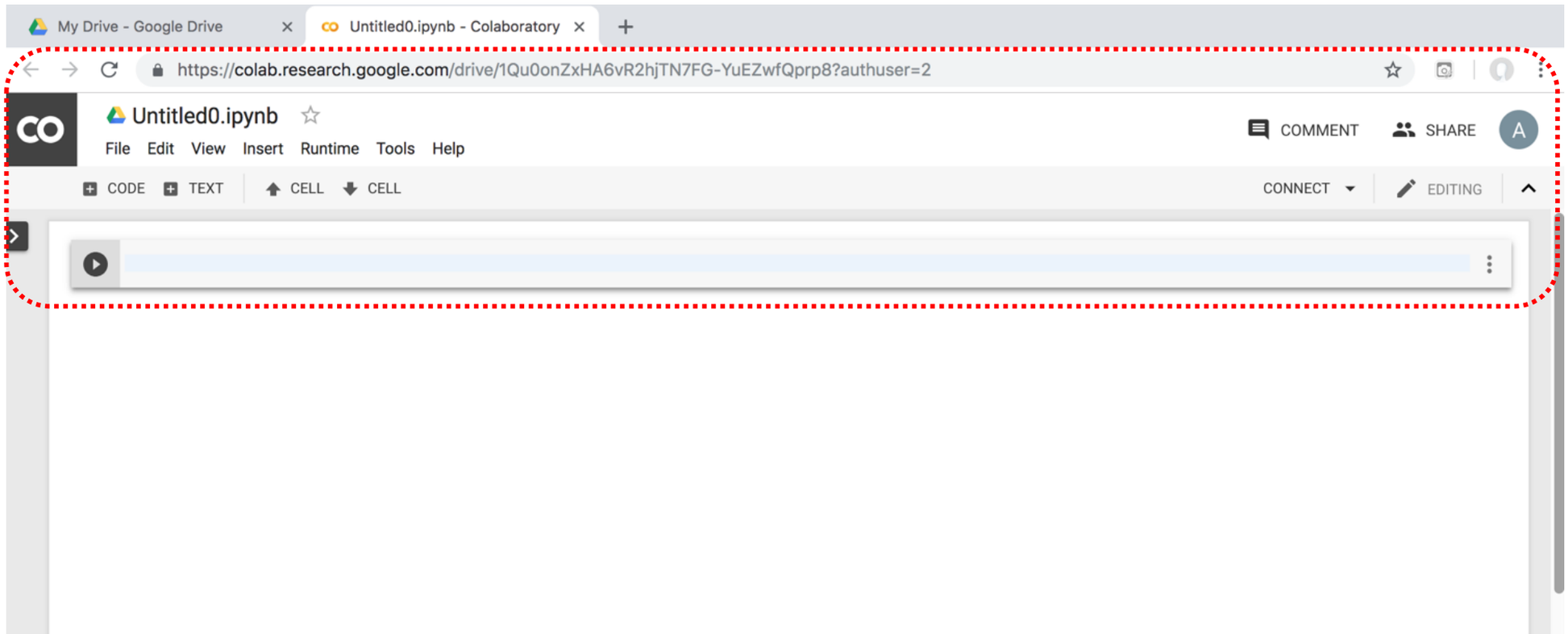
Connect Colaboratory to Google Drive



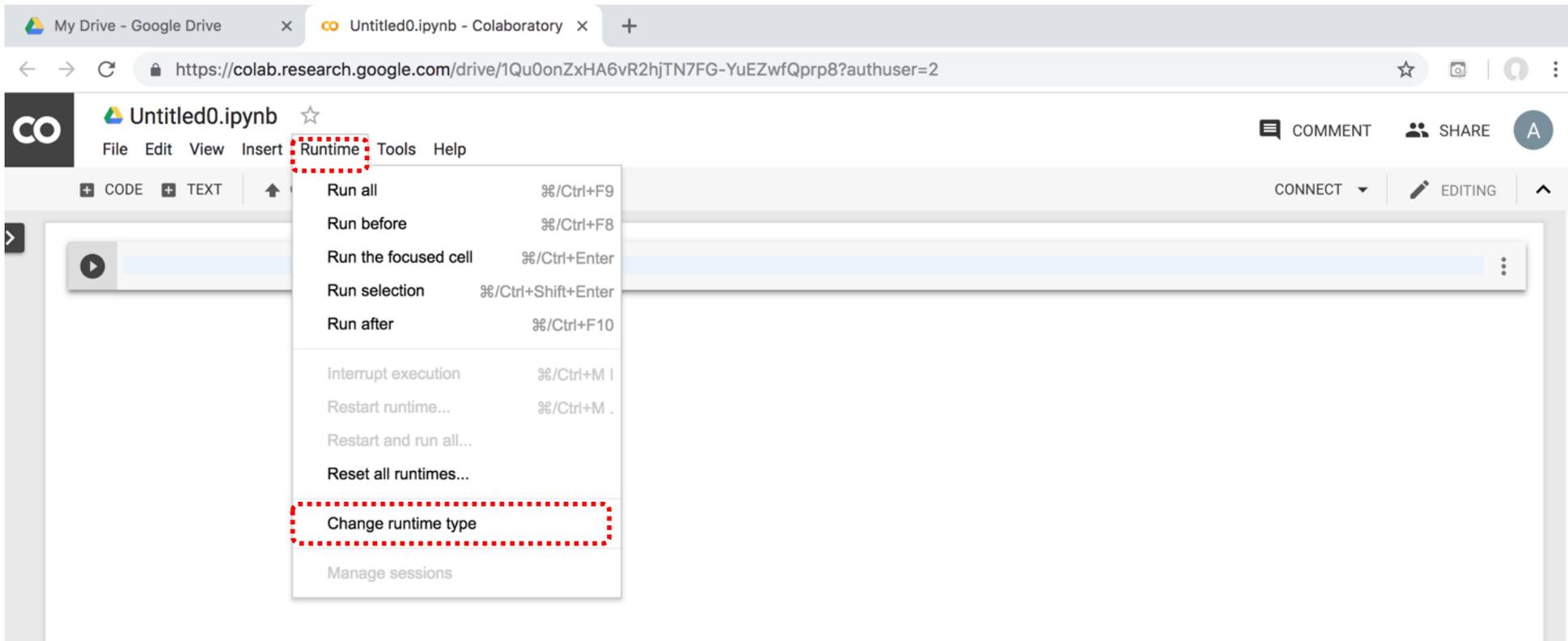
Google Colab



Google Colab

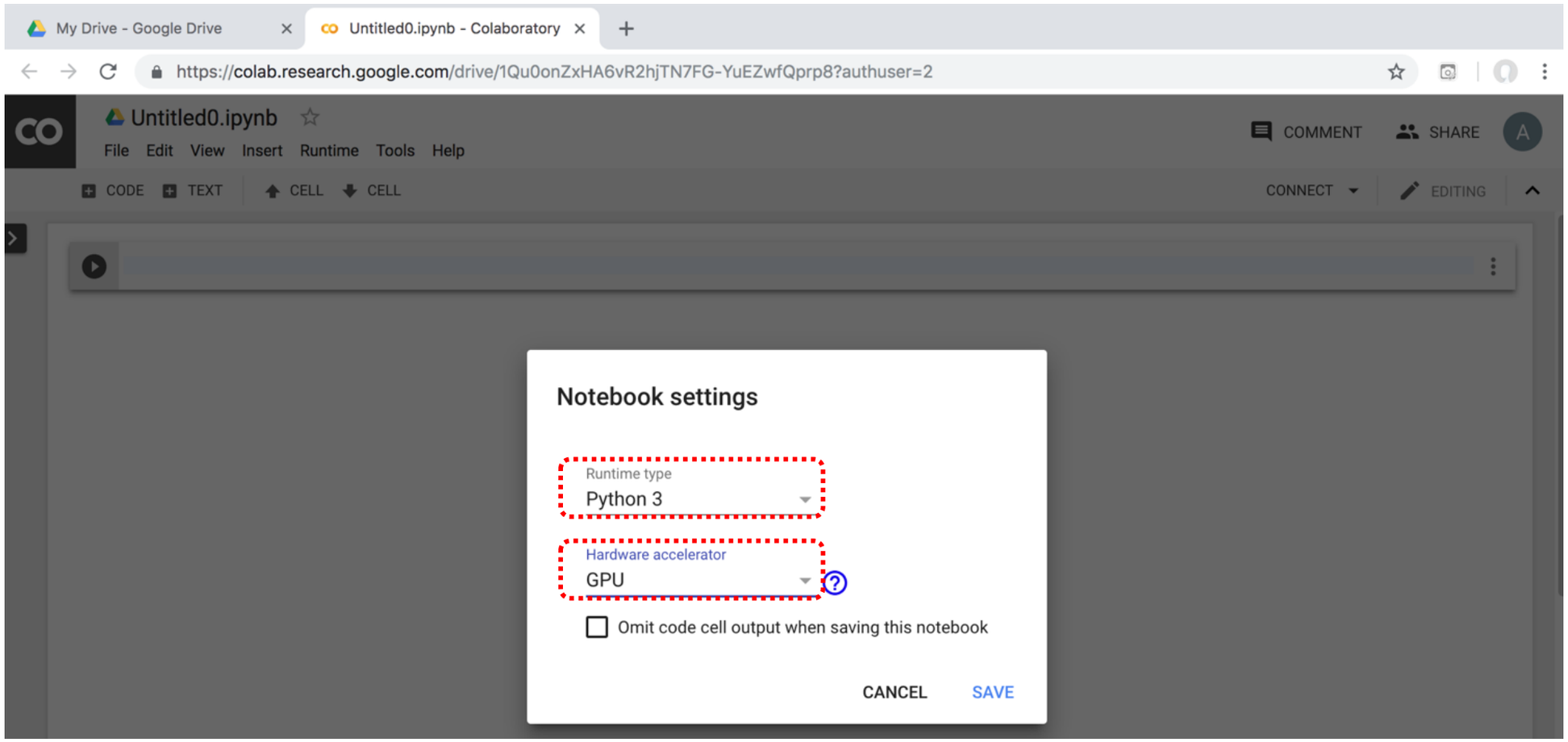


Google Colab



Run Jupyter Notebook

Python3 GPU
Google Colab



Google Colab Python Hello World

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

Python in Google Colab

python101.ipynb - Colaboratory

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT?authuser=2#scrollTo=wsh36fLxDKC3

python101.ipynb

File Edit View Insert Runtime Tools Help

COMMENT SHARE A

CODE TEXT CELL CELL CONNECTED EDITING

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

```
194.87
```

[11]

```
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
```

[12]

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

```
194.87
```

[13]

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

```
Pass
```

Python Programming

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

Source: https://www.w3resource.com/python-exercises/python-basic-exercise-39.php

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
```

Summary

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

Python
Data Structures

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

https://www.w3schools.com/python/python_lists_methods.asp

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])
print(x[1])
print(x[2])
print(x[-1])
```

```
10
20
30
50
```

Source: http://pythonprogramminglanguage.com/tuples/

Sets {}

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

```
True
False
True
3
3
2
```

Source: http://cs231n.github.io/python-numpy-tutorial/

Dictionary {key : value}

```
Python Dictionary
Key  → Value
'EN' → 'English'
'FR' → 'French'
```

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

```
English
```

Source: http://pythonprogramminglanguage.com/dictionary/

Python Data Structures

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

Python for Finance Applications

```
# Python Lists
expenses = [72.50, 80.75, 50.00, 90.25]
total_expenses = sum(expenses)
print("Total expenses:", total_expenses)
```

```
Total expenses: 293.5
```

Python for Finance Applications

```
# Python Tuples
accounts = (("Cash", 1001), ("Accounts Receivable", 1002),
("Inventory", 1003))
for account in accounts:
    print("Account name:", account[0], "Account number:", account[1])
```

```
Account name: Cash Account number: 1001
Account name: Accounts Receivable Account number: 1002
Account name: Inventory Account number: 1003
```

Python for Finance Applications

```
# Python Sets
account_numbers = {1001, 1002, 1003}
new_account_number = 1004
if new_account_number not in account_numbers:
    print("Account number", new_account_number, "is not in use.")
```

```
Account number 1004 is not in use.
```

Python for Finance Applications

```python
# Python Dictionaries
accounts = {"1001": {"name": "Cash", "balance": 500.00, "type": "Asset"},
"1002": {"name": "Accounts Receivable", "balance": 1000.00, "type": "Asset"},
"2001": {"name": "Accounts Payable", "balance": 750.00, "type": "Liability"}}
for account_number, account_info in accounts.items():
print("Account number:", account_number)
print("Account name:", account_info["name"])
print("Account balance:", account_info["balance"])
print("Account type:", account_info["type"])
```

```
Account number: 1001
Account name: Cash
Account balance: 500.0
Account type: Asset
Account number: 1002
Account name: Accounts Receivable
Account balance: 1000.0
Account type: Asset
```

```
Account number: 2001
Account name: Accounts Payable
Account balance: 750.0
Account type: Liability
```

Summary

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

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
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Source: https://www.w3schools.com/python/python_operators.asp

Python Logical Operators

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

Source: https://www.w3schools.com/python/python_operators.asp

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
```

Summary

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

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

Source: https://www.w3resource.com/python-exercises/python-basic-exercise-39.php

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)
```

https://www.w3schools.com/python/python_classes.asp

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 Obects

```
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
```

https://www.learnpython.org/en/Classes_and_Objects

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()
```

Summary

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

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

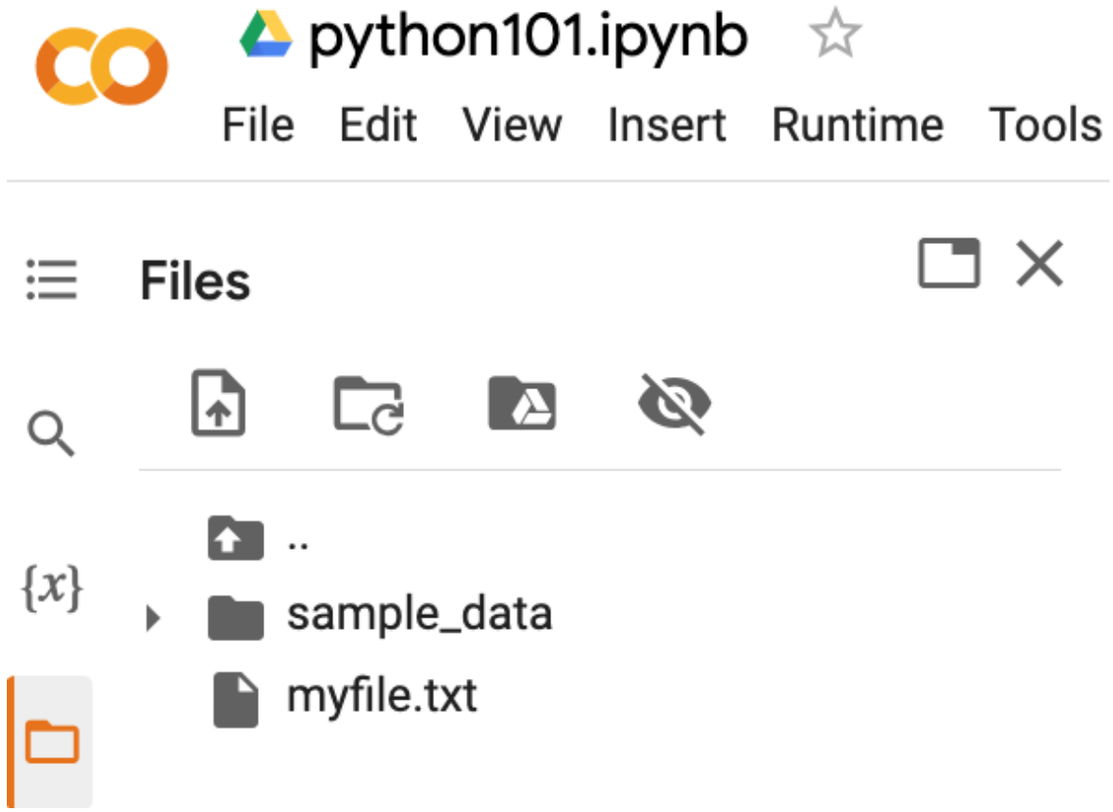
```
# Python File Input / Output
with open('myfile.txt', 'a+') as file:
    file.write('\n' + 'New line')


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
```



Python OS, IO, files, and Google Drive

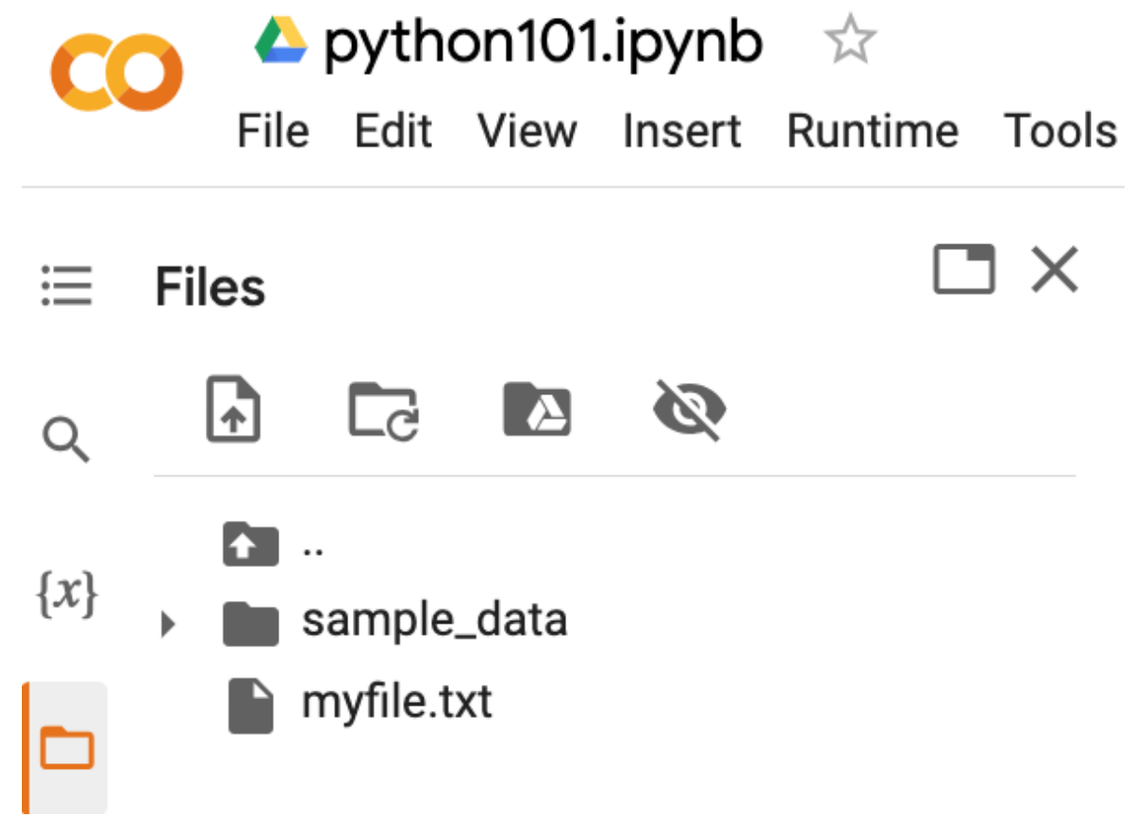
```
import os

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

```
/content
```

os.listdir()

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

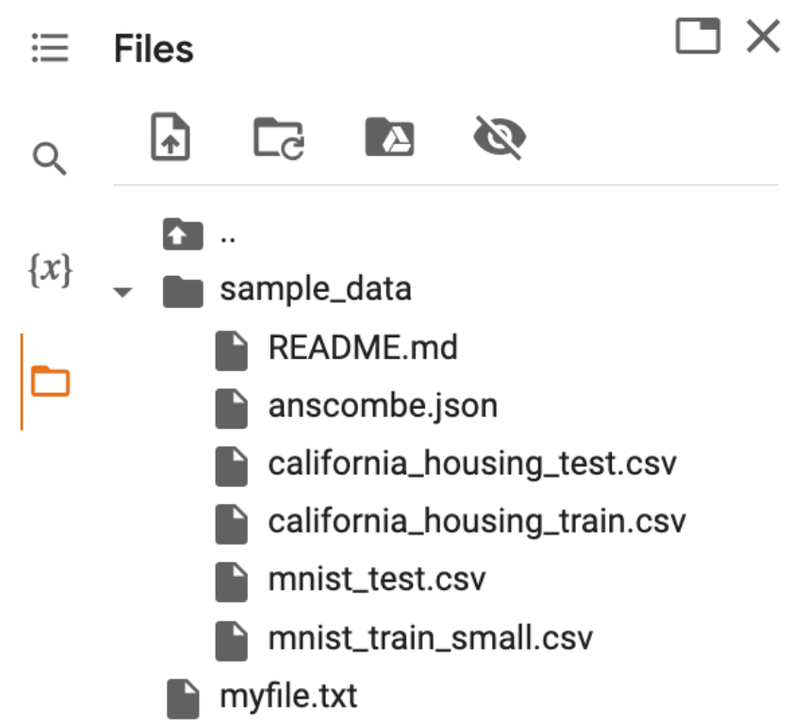
```
['.config',
'myfile.txt',
'sample_data']
```



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

THANKS FOR YOUR ATTENTION!