Introduction to Python

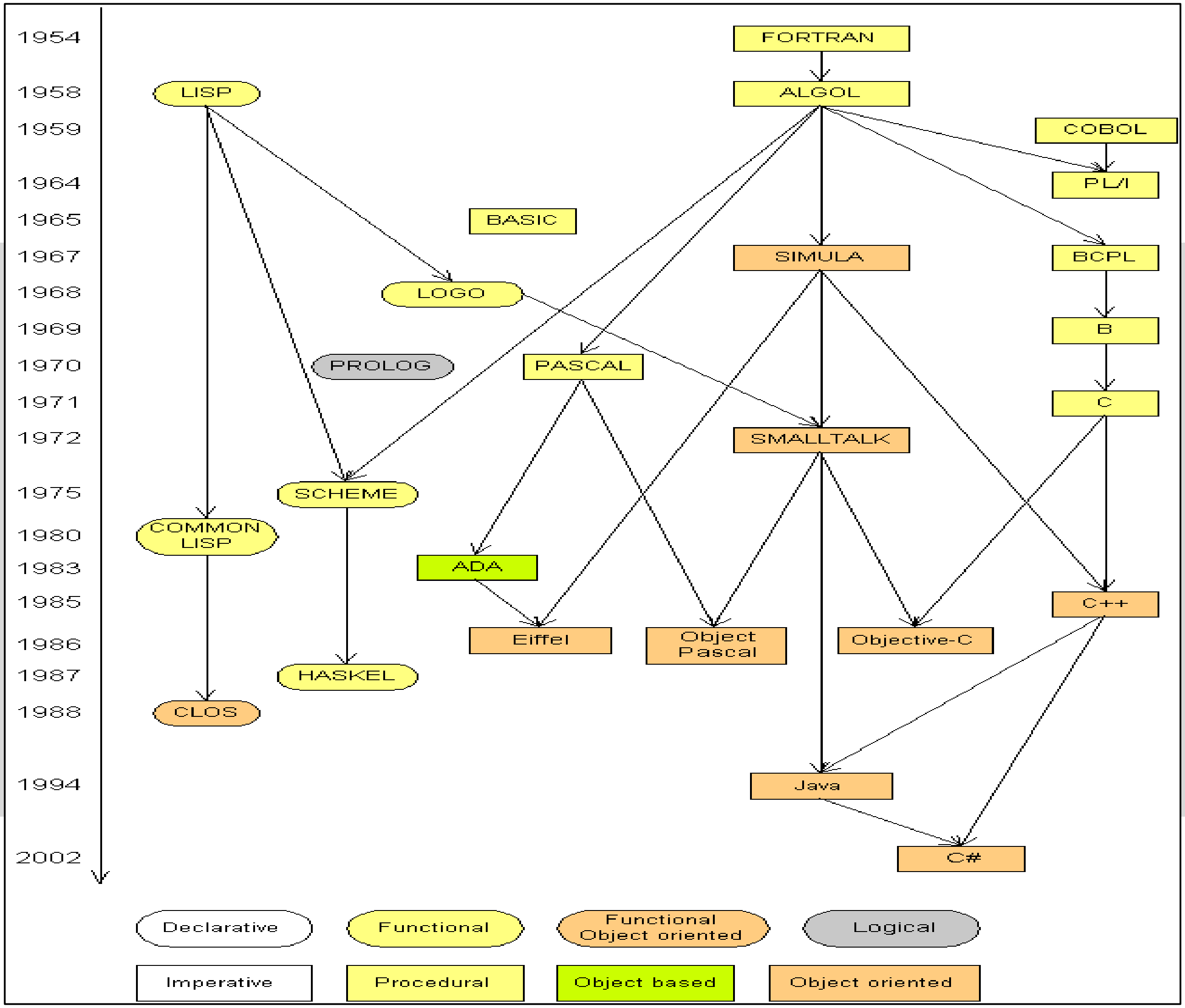
Agenda

In this session, you will learn about:

- History of Python
- Python Environments
- Python Coding
- Data Structures

Some Common Languages in use

Languages History



History of Python

Guido Van Rossum

Invented in December 1989

- First public release in 1991
- Open source from beginning
- Managed by Python Software Foundation

Python

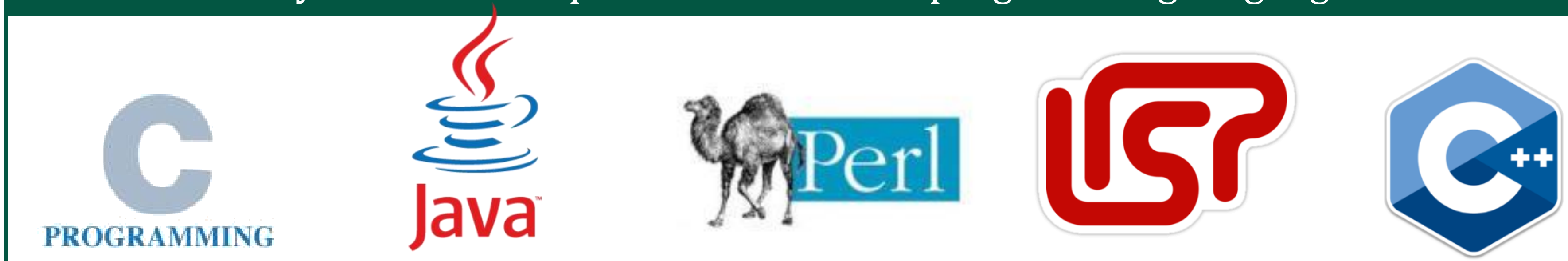
Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open.

Python is an interpreted language, do not need to be compiled to run.

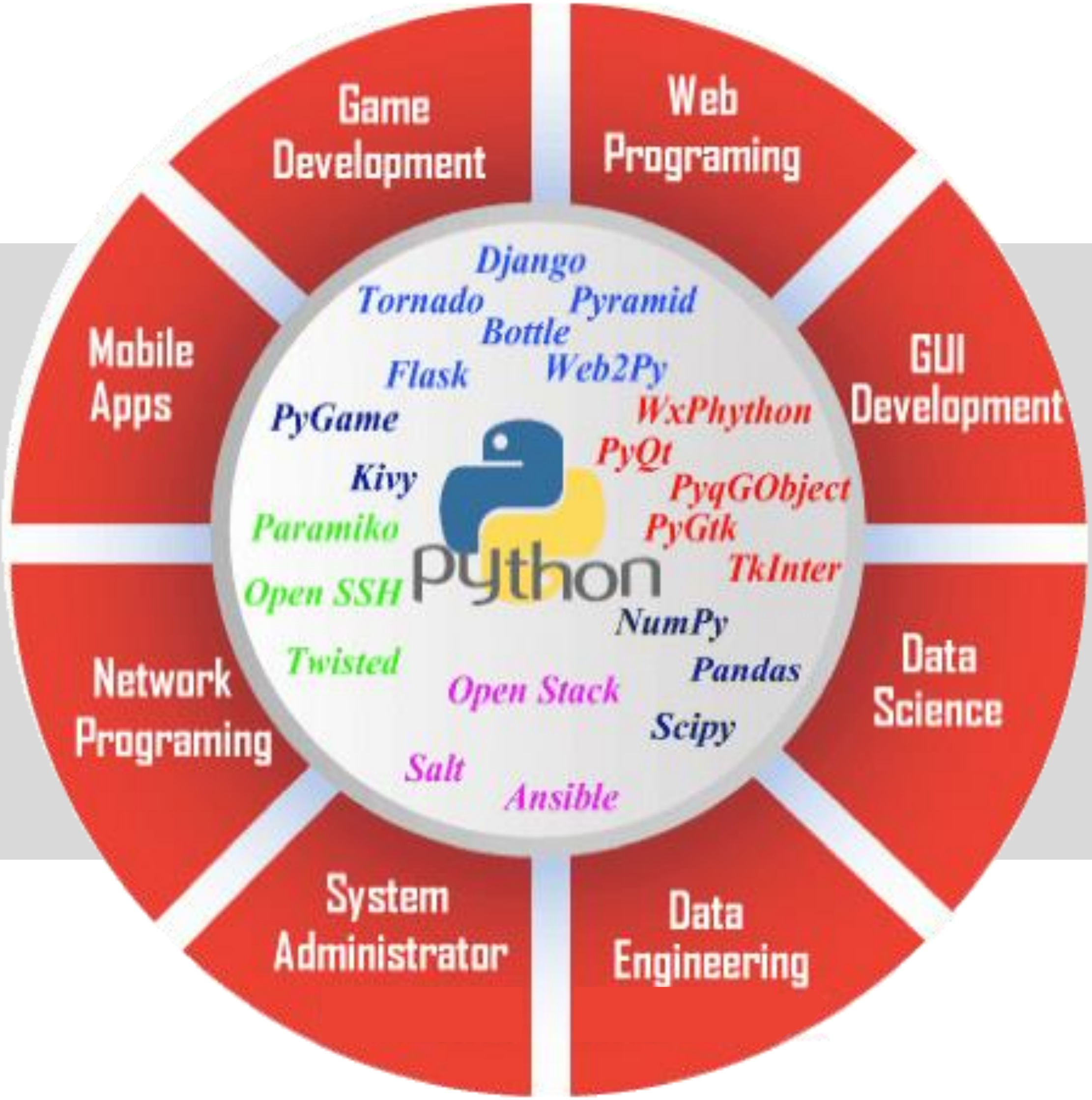
Python is a high-level language, which means a programmer can focus on what to do instead of how to do it.

Writing programs in Python takes less time than in another language.

Python drew inspiration from other programming languages:

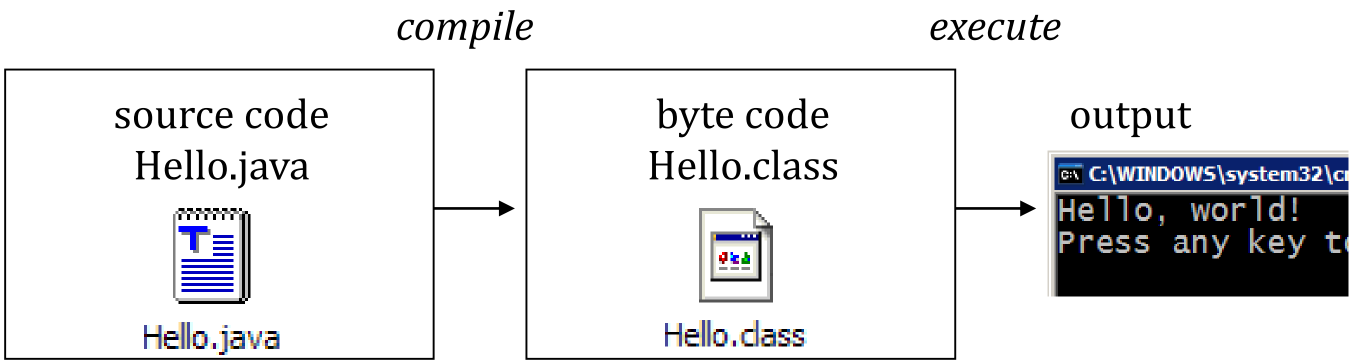


Python Used For



Compiling and Interpreting

Many languages require to compile the program into a form that the machine understands.



compile

execute

source code
Hello.java

Hello.java

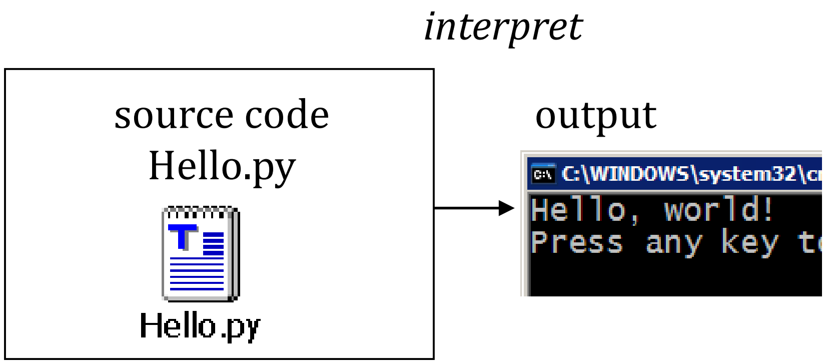
byte code
Hello.class

Hello.class

output

```
Hello, world!
Press any key t
```

Python is instead directly interpreted into machine instructions.



interpret

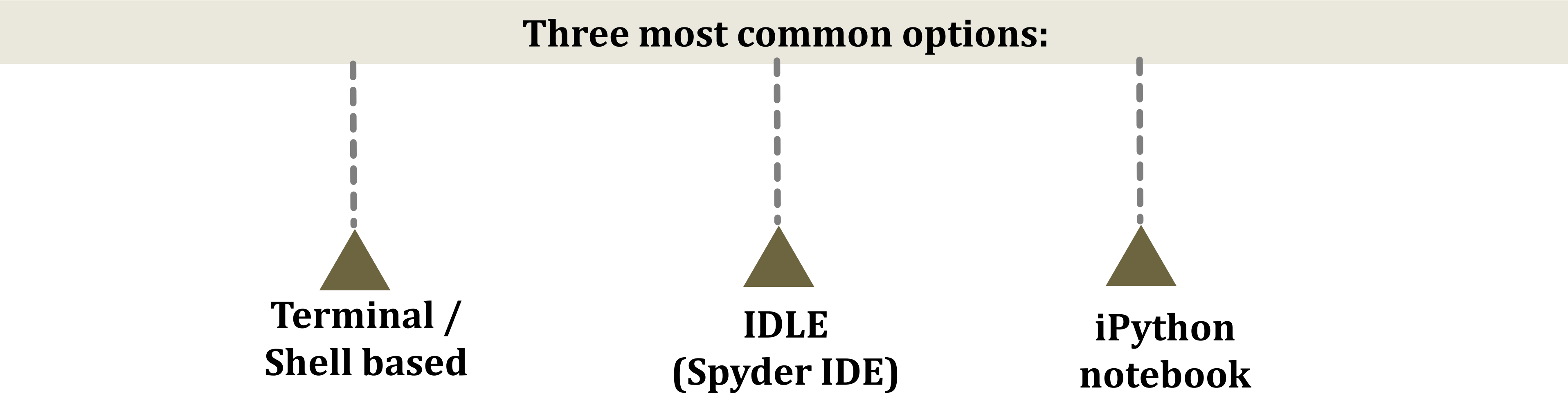
source code
Hello.py

Hello.py

output

```
Hello, world!
Press any key t
```

Development Environment

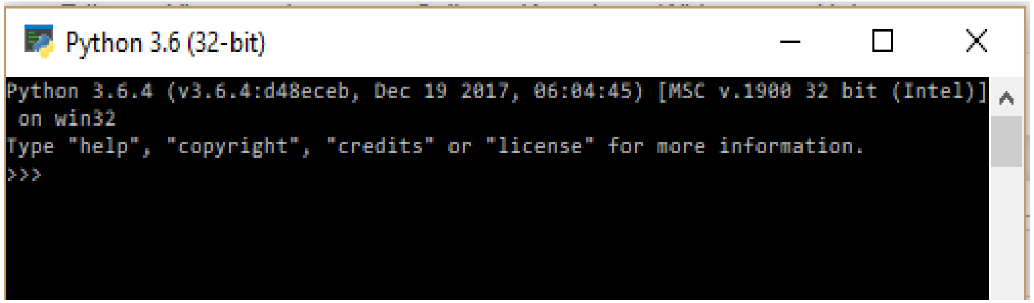


Three most common options:

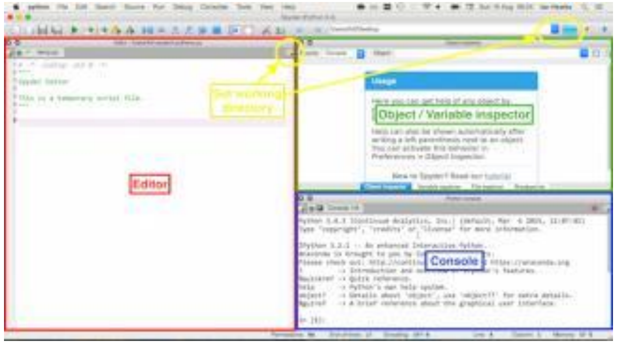
Terminal / Shell based

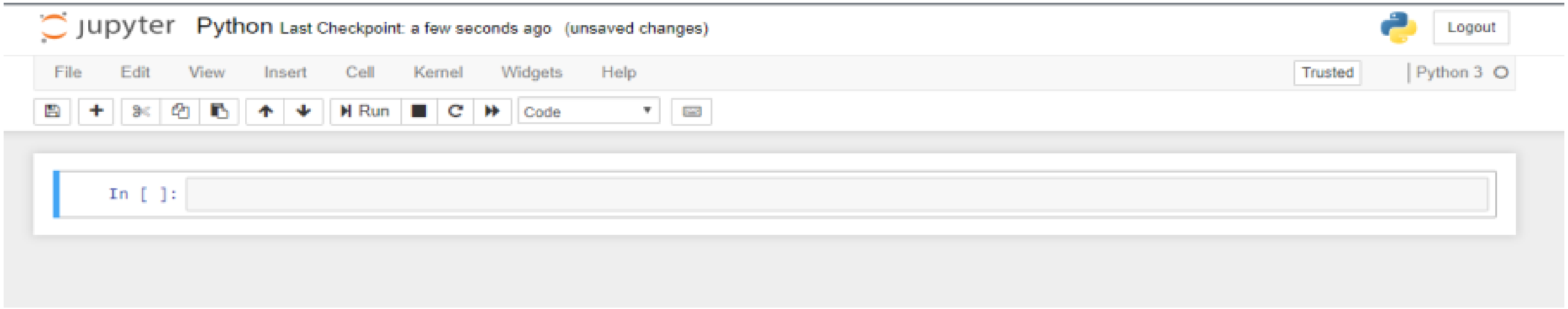
IDLE (Spyder IDE)

iPython notebook

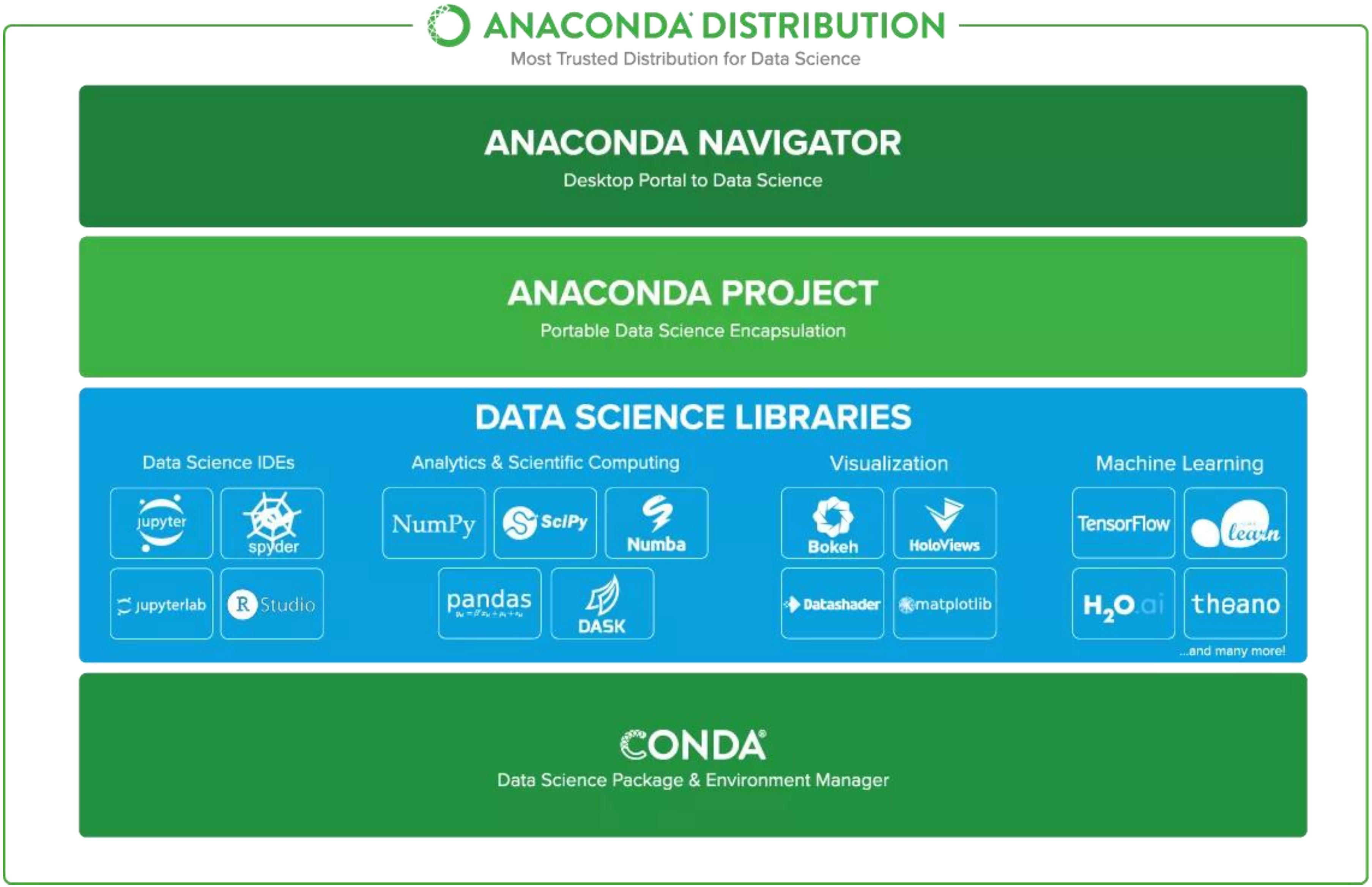


```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```





Anaconda



ANACONDA DISTRIBUTION

Most Trusted Distribution for Data Science

ANACONDA NAVIGATOR

Desktop Portal to Data Science

ANACONDA PROJECT

Portable Data Science Encapsulation

DATA SCIENCE LIBRARIES

Data Science IDEs

Analytics & Scientific Computing

Visualization

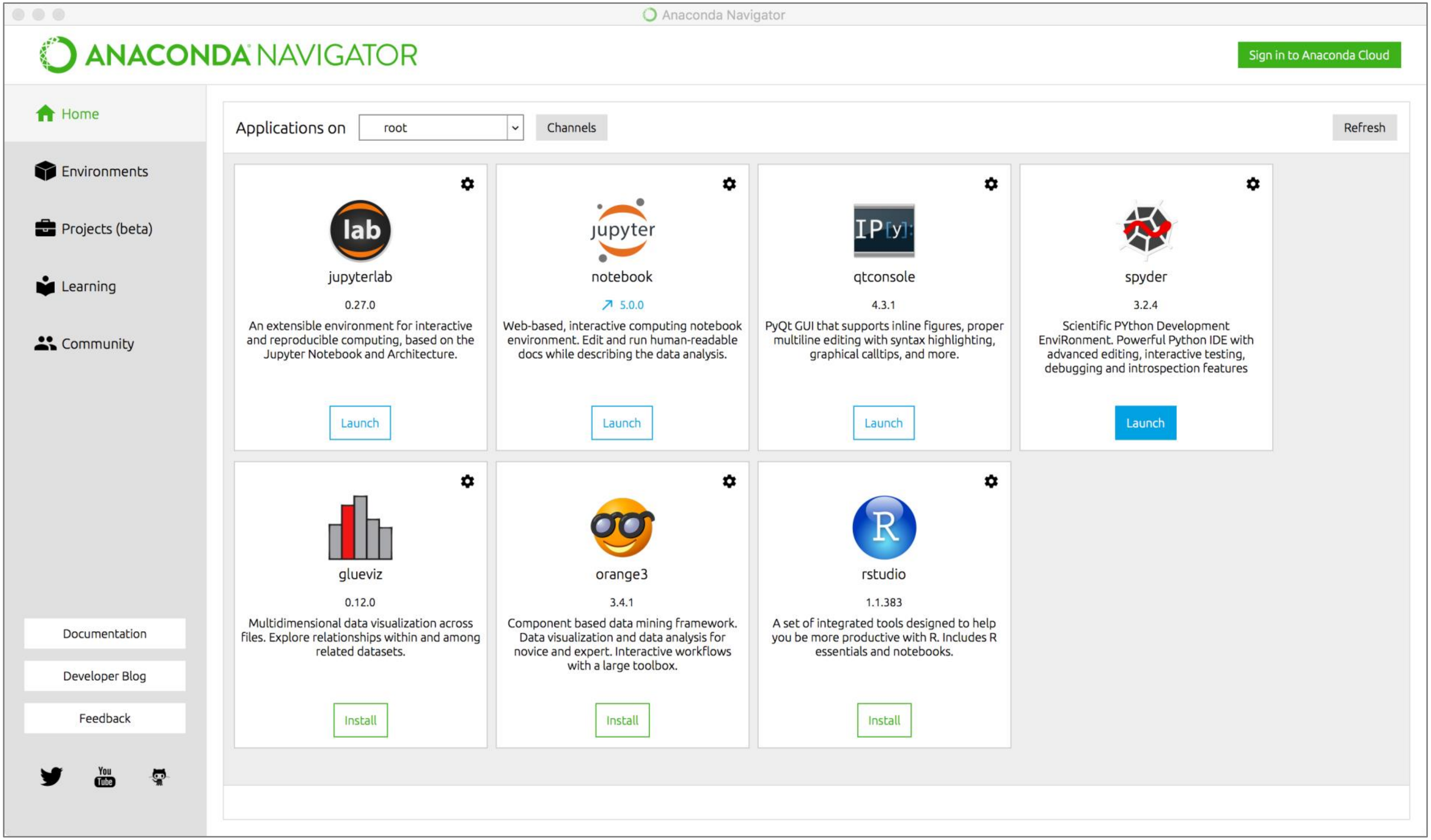
Machine Learning

...and many more!

CONDA

Data Science Package & Environment Manager

Anaconda Navigator



jupyterlab

0.27.0

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch

notebook

↗ 5.0.0

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

3.2.4

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch

glueviz

0.12.0

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Install

orange3

3.4.1

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

Install

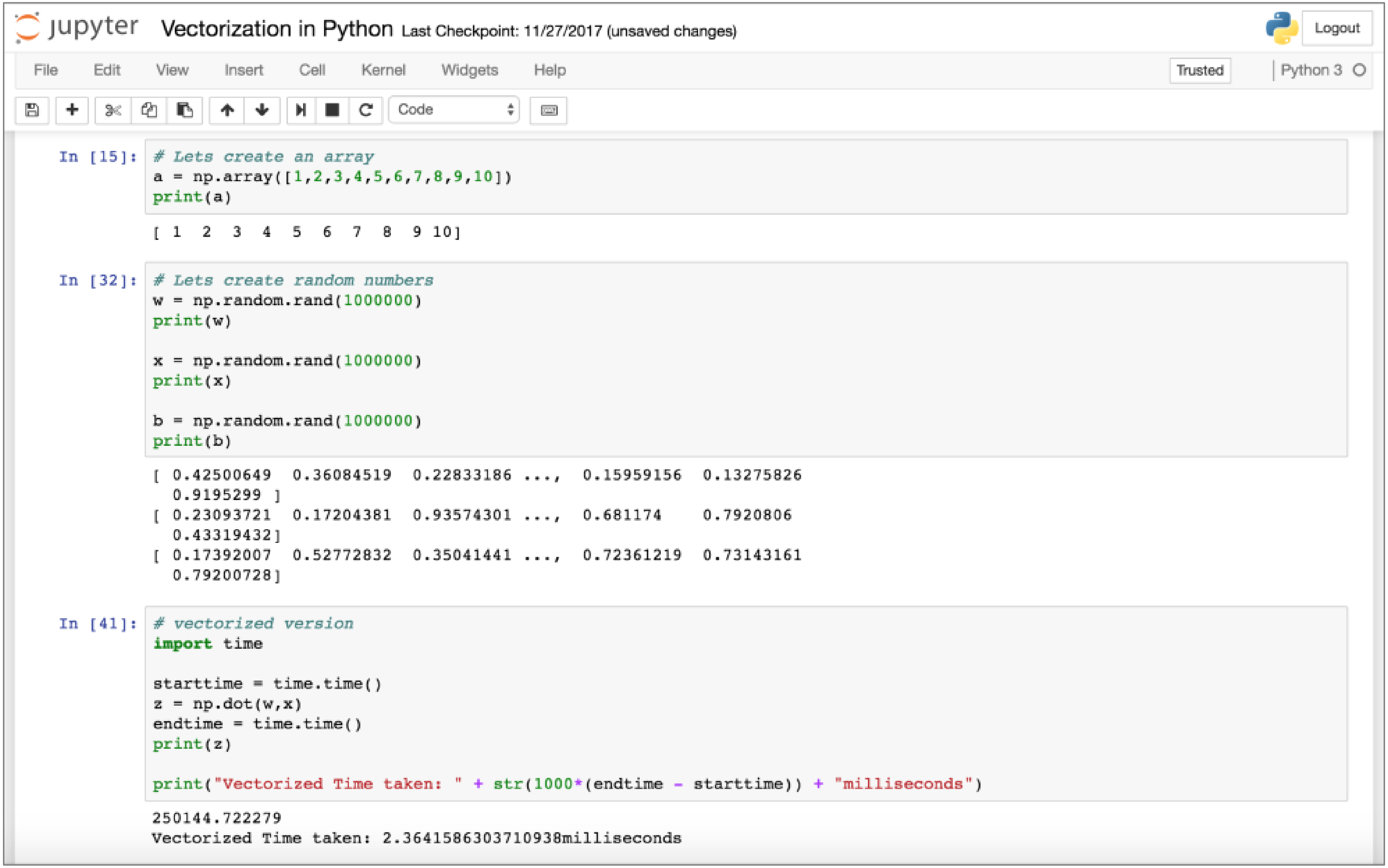
rstudio

1.1.383

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Install

Jupyter IDE



In [15]:
```
# Lets create an array
a = np.array([1,2,3,4,5,6,7,8,9,10])
print(a)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

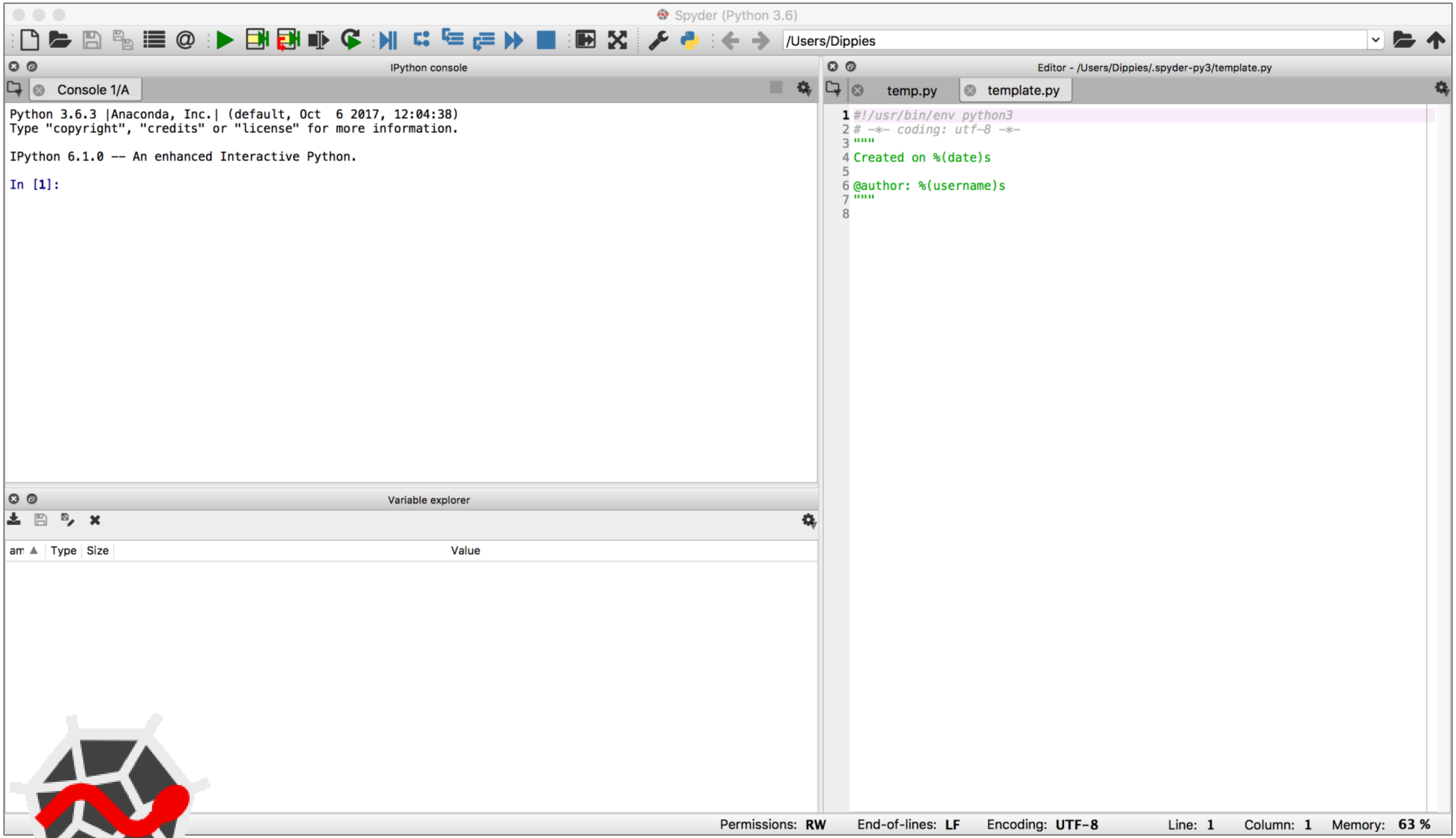
In [32]:
```
# Lets create random numbers
w = np.random.rand(1000000)
print(w)

x = np.random.rand(1000000)
print(x)

b = np.random.rand(1000000)
print(b)
```

```
[ 0.42500649  0.36084519  0.22833186 ...,  0.15959156  0.13275826
  0.9195299 ]
[ 0.23093721  0.17204381  0.93574301 ...,  0.681174    0.7920806
  0.43319432]
[ 0.17392007  0.52772832  0.35041441 ...,  0.72361219  0.73143161
  0.79200728]
```

In [41]:
```
# vectorized version
import time

starttime = time.time()
z = np.dot(w,x)
endtime = time.time()
print(z)

print("Vectorized Time taken: " + str(1000*(endtime - starttime)) + "milliseconds")
```

```
250144.722279
Vectorized Time taken: 2.3641586303710938milliseconds
```

Spyder IDE

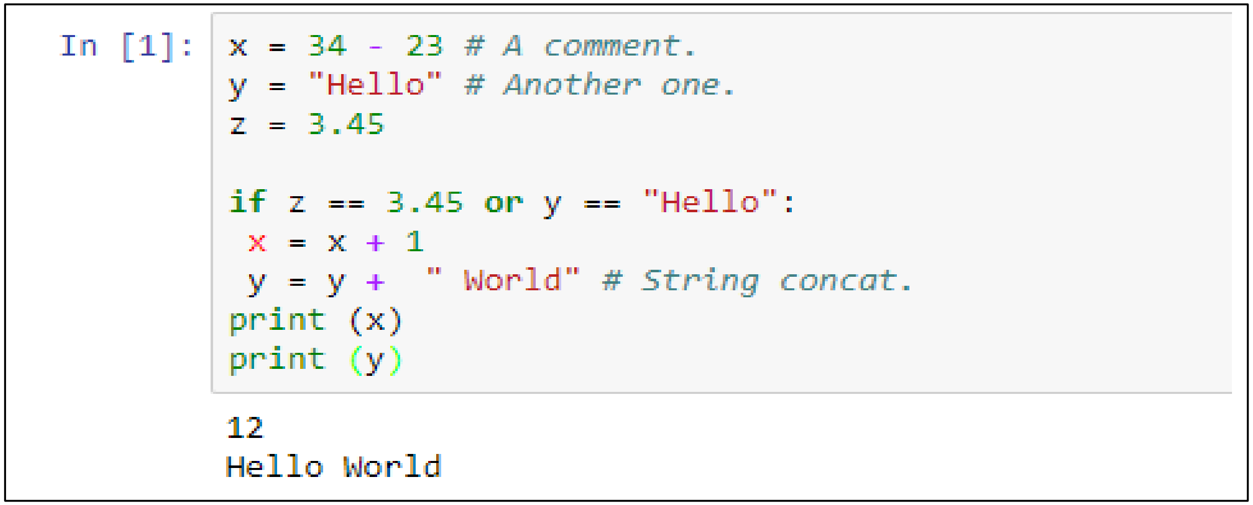


```
Python 3.6.3 |Anaconda, Inc.| (default, Oct  6 2017, 12:04:38)
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on %(date)s

@author: %(username)s
"""

```

Permissions: RW   End-of-lines: LF   Encoding: UTF-8   Line: 1   Column: 1   Memory: 63 %

A Python Code Sample



```
x = 34 - 23 # A comment.
y = "Hello" # Another one.
z = 3.45

if z == 3.45 or y == "Hello":
 x = x + 1
 y = y +  " World" # String concat.
print (x)
print (y)
```

```
12
Hello World
```

- Assignment uses **=** and comparison uses **==**
- For numbers **+-*/%** are as expected
- Logical operators are words (**and, or, not**)
- The basic printing command is **print**
- Start comments with **#:** the rest of line is ignored

Understanding Reference Semantics

Assignment manipulates references

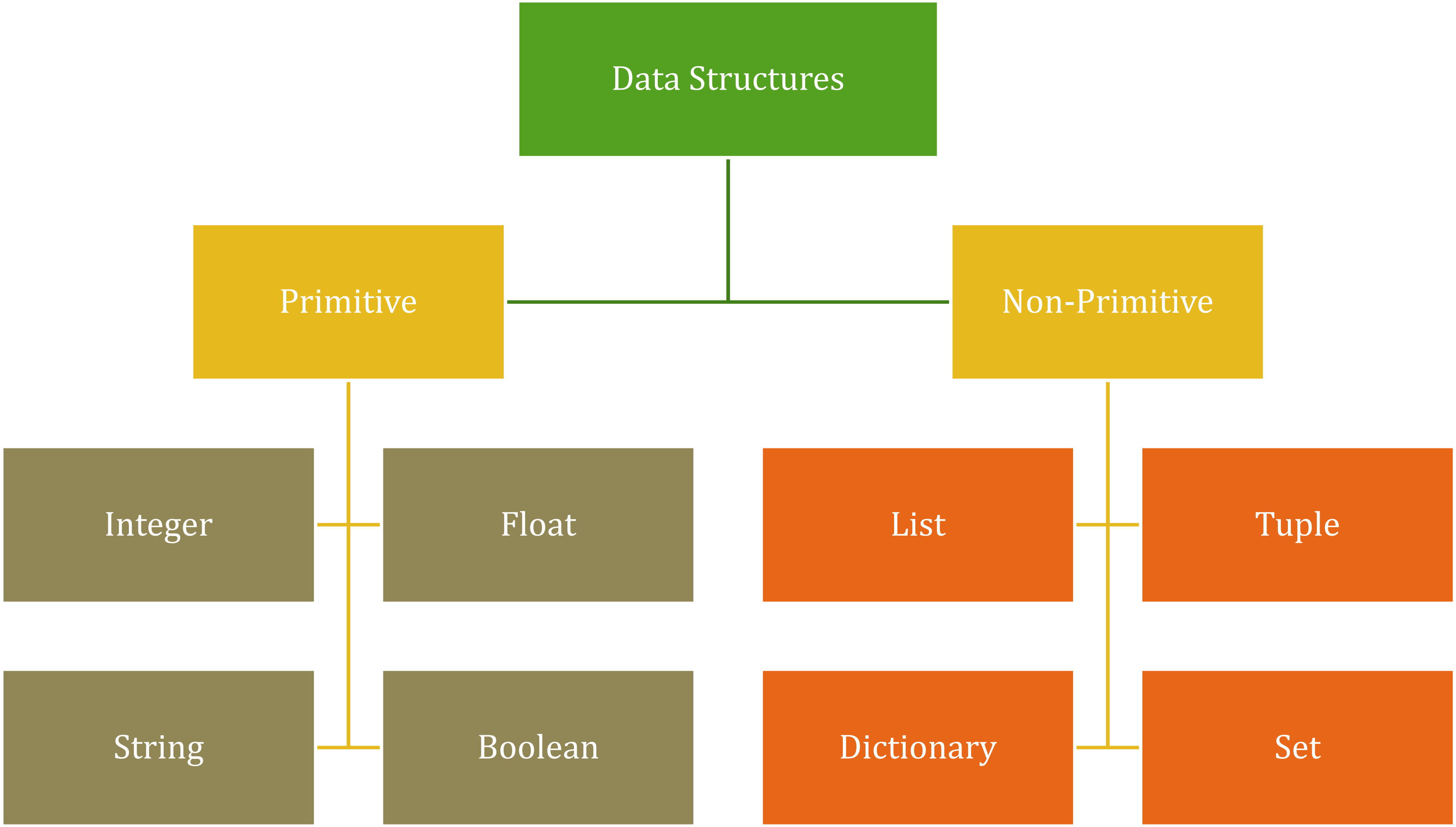
x = y does not make a copy of y

x = y makes x reference the object y references

Example

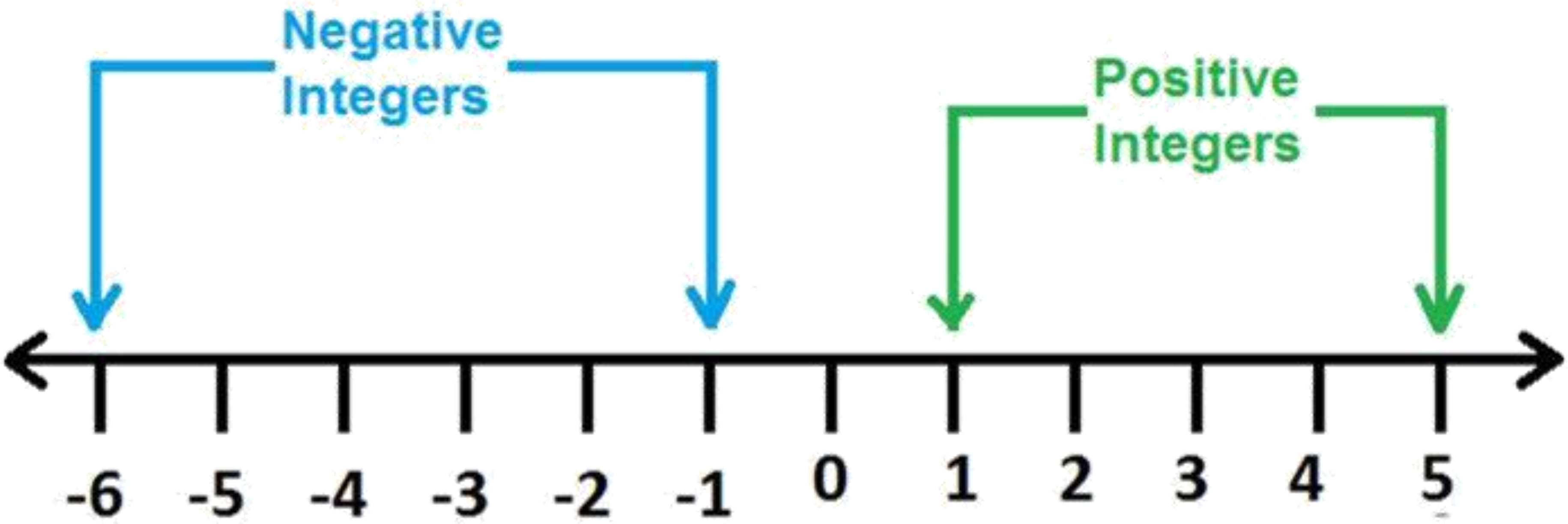
```
>>> a = [1, 2, 3]        # variable a now references the list [1, 2, 3]
>>> b = a                # variable b now references what variable a
                         references
>>> a.append(4)          # this changes the list variable a references
>>> print b              # if we print what variable b references, we see
[1, 2, 3, 4]             # variable b has changed…
```

Data Structures



Primitive Data Structures

Integers



- Integer represent positive or negative whole number with no decimal point, Integer in Python 3 are of unlimited size
- They are immutable data types
- Changing the value of a number data type results in a newly allocated object

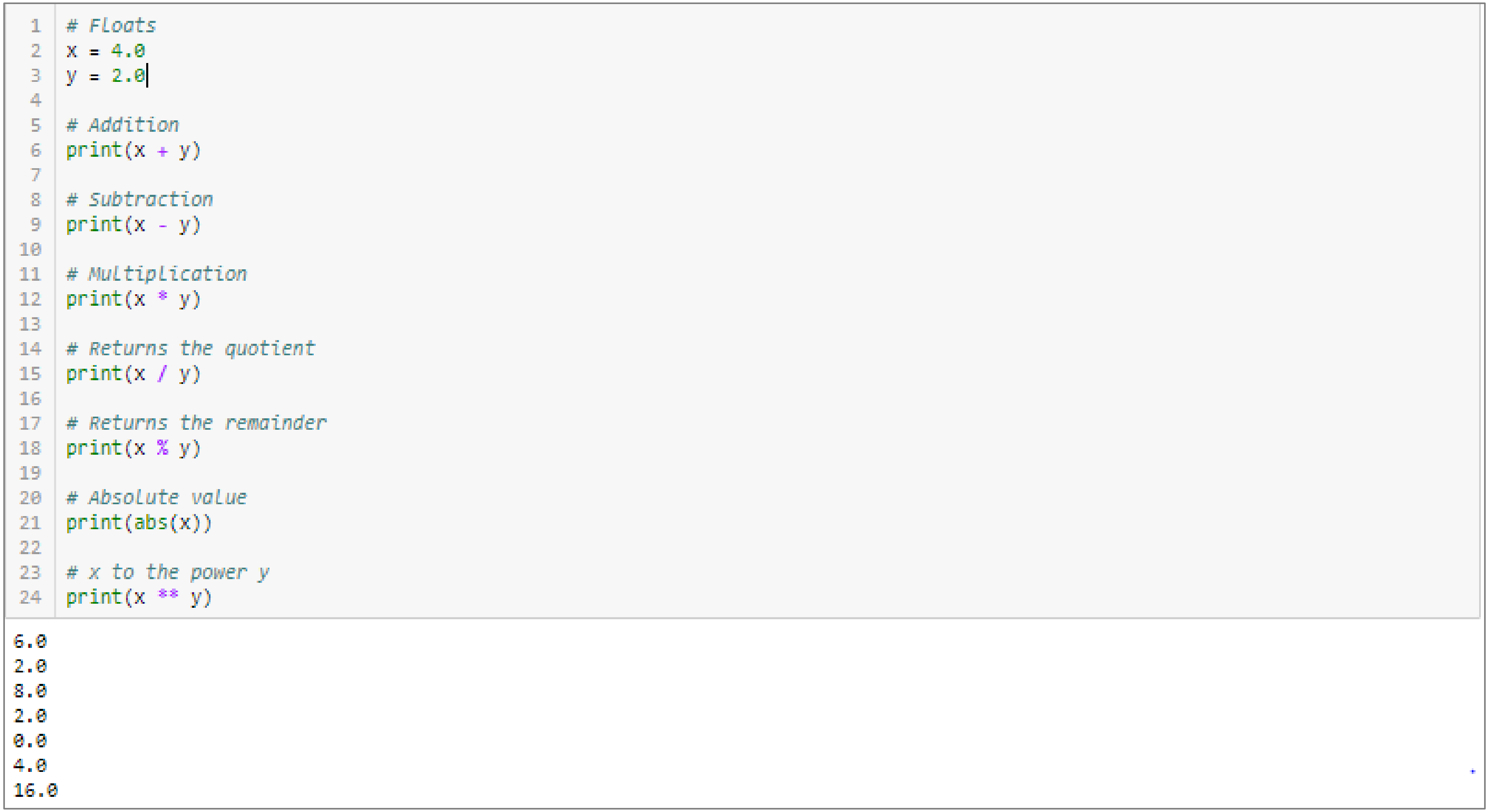
Float

“Float” stands for ‘floating point number’

Float represent real numbers and are written with a decimal point dividing the integer and the fractional parts.

Float may also be in scientific notation, with ’E’ or ‘e’ indicating the power of 10 (3.7e2 = 3.7 * 7.389 = 27.3393)

Example: Float



```python
# Floats
x = 4.0
y = 2.0

# Addition
print(x + y)

# Subtraction
print(x - y)

# Multiplication
print(x * y)

# Returns the quotient
print(x / y)

# Returns the remainder
print(x % y)

# Absolute value
print(abs(x))

# x to the power y
print(x ** y)
```

```
6.0
2.0
8.0
2.0
0.0
4.0
16.0
```

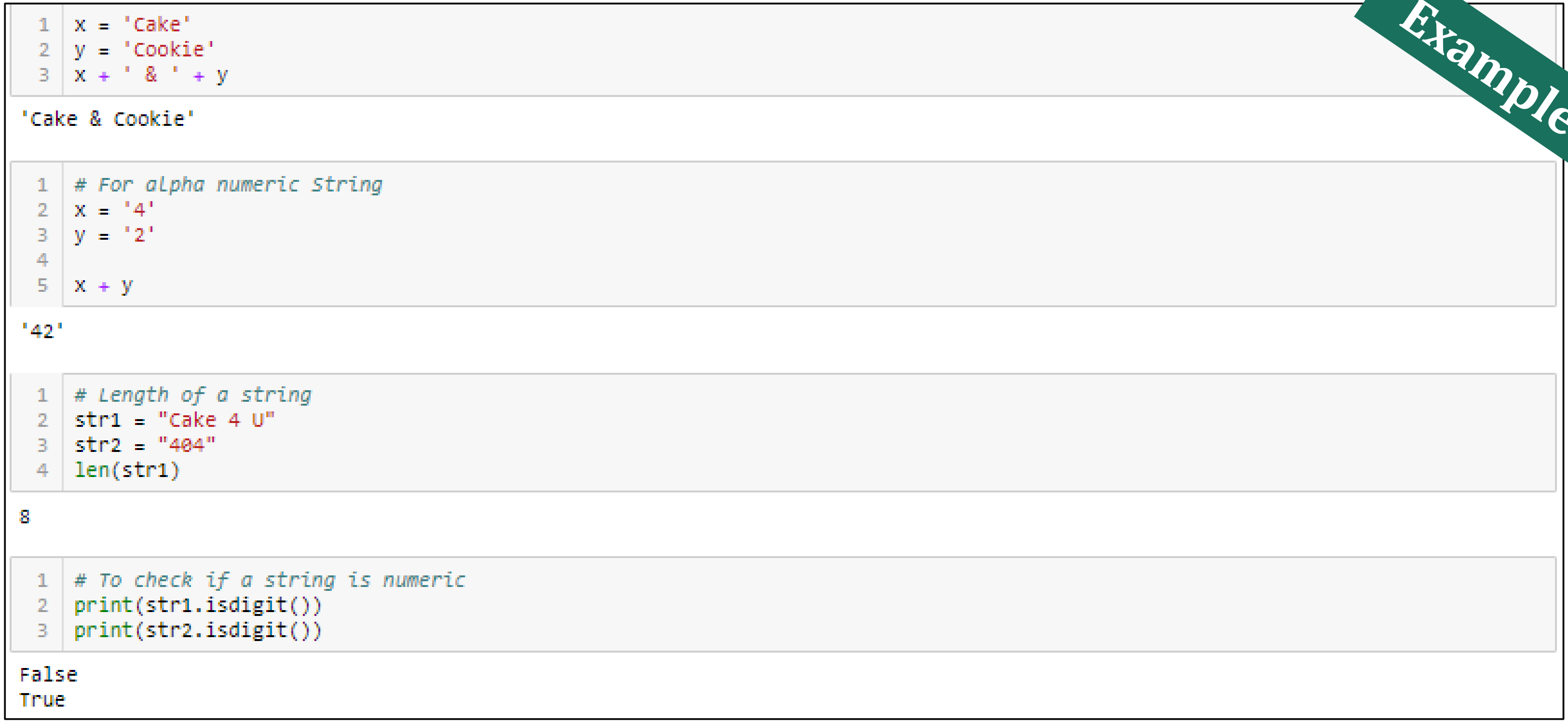
String

Strings are collections of alphabets, words or other characters

Strings are immutable sequence of characters.

Single and double quotes are special characters used to define strings

Example



```
x = 'Cake'
y = 'Cookie'
x + ' & ' + y
```

```
'Cake & Cookie'
```

```
# For alpha numeric String
x = '4'
y = '2'

x + y
```

```
'42'
```

```
# Length of a string
str1 = "Cake 4 U"
str2 = "404"
len(str1)
```

```
8
```

```
# To check if a string is numeric
print(str1.isdigit())
print(str2.isdigit())
```

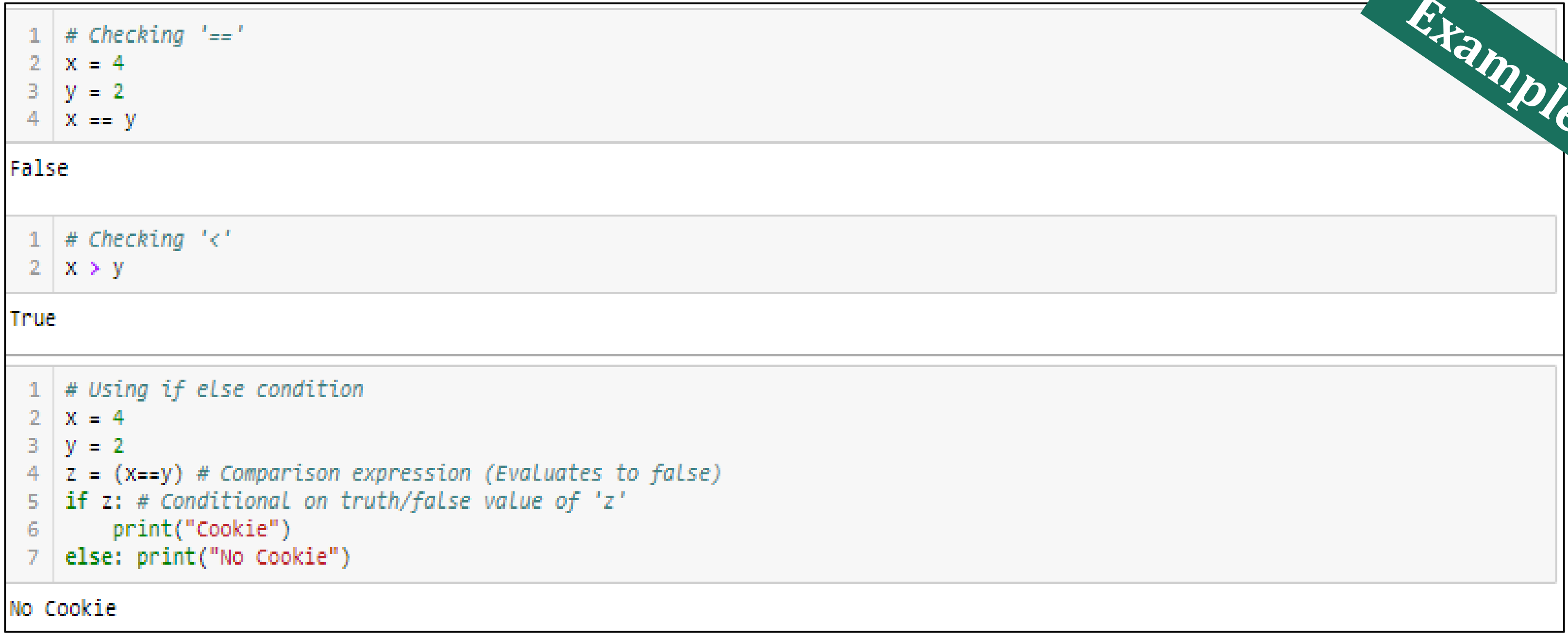
```
False
True
```

Boolean

Boolean can take up the values: **True** and **False**, which often makes them interchangeable with the integers 1 and 0.

Booleans are useful in conditional and comparison expressions.

Example



```
# Checking '=='
x = 4
y = 2
x == y
```

```
False
```

```
# Checking '<'
x > y
```

```
True
```

```
# Using if else condition
x = 4
y = 2
z = (x==y) # Comparison expression (Evaluates to false)
if z: # Conditional on truth/false value of 'z'
    print("Cookie")
else: print("No Cookie")
```

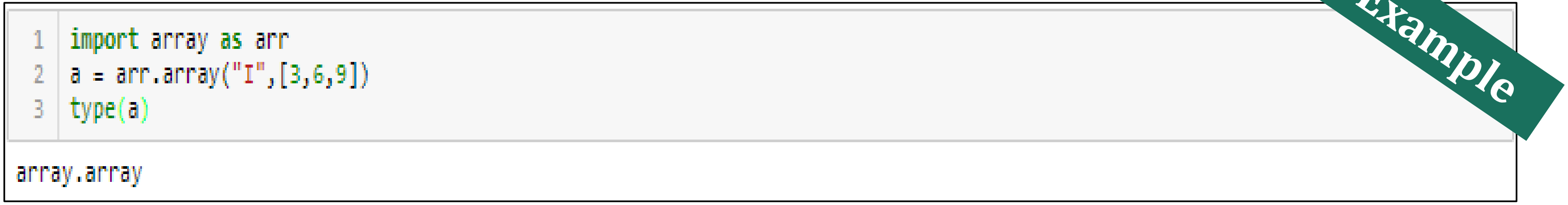
```
No Cookie
```

Non – Primitive Data Structures

Array

- Arrays are a compact way of collecting basic data types
- In general, arrays in Python, actually refer to lists
- This type of list has elements of the same data type
- Arrays are supported by the array module

Example



```
import array as arr
a = arr.array("I",[3,6,9])
type(a)
```

```
array.array
```

List

A list is a container object.

It is homogenous and can have duplicate values.

List supports the following operations:

Append new elements

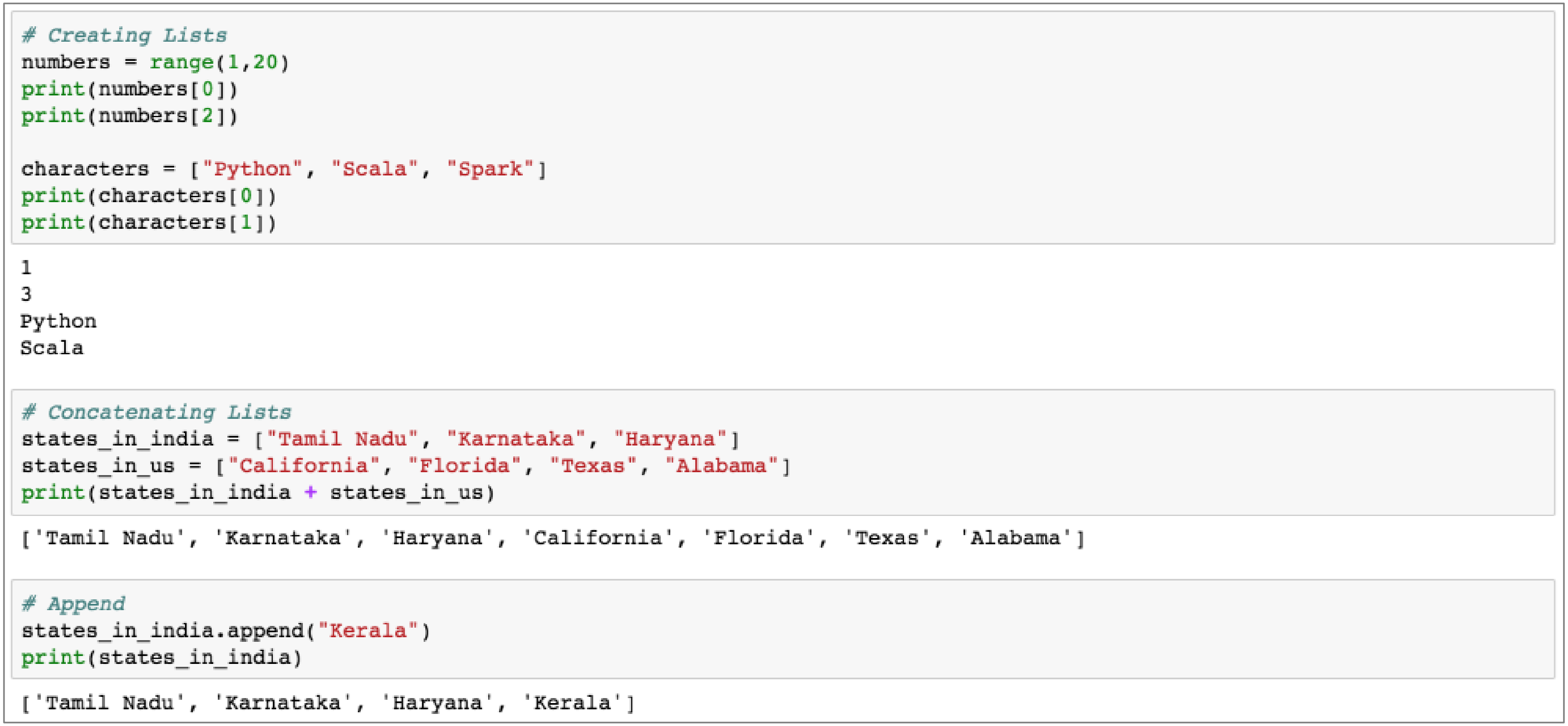
Access using indexes

min() & max()

Concatenate

in & not in

Example: List



```python
# Creating Lists
numbers = range(1,20)
print(numbers[0])
print(numbers[2])

characters = ["Python", "Scala", "Spark"]
print(characters[0])
print(characters[1])
```

```
1
3
Python
Scala
```

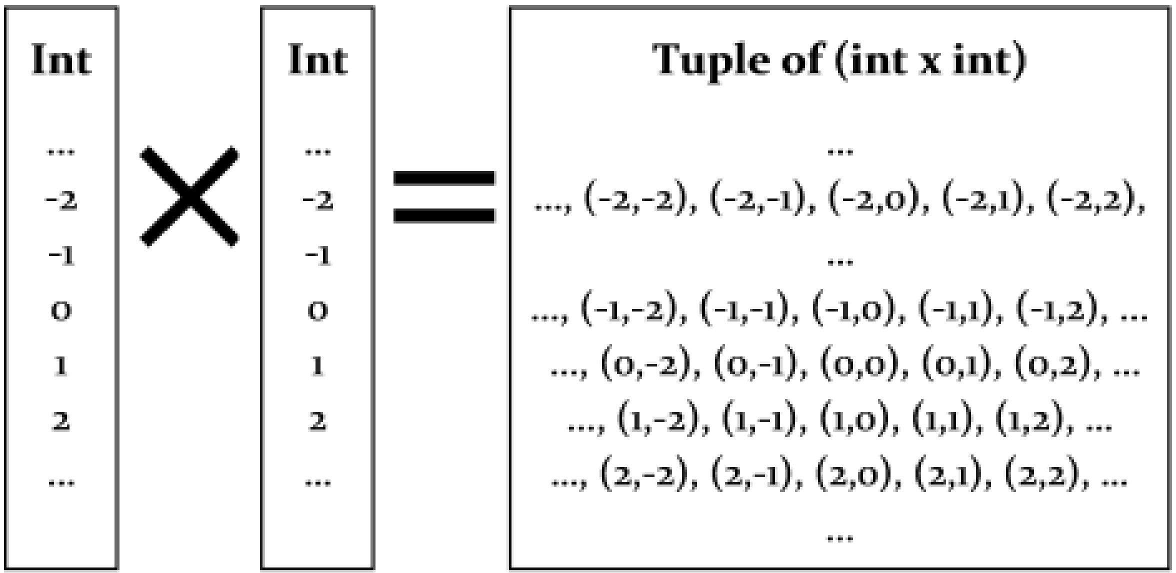
```python
# Concatenating Lists
states_in_india = ["Tamil Nadu", "Karnataka", "Haryana"]
states_in_us = ["California", "Florida", "Texas", "Alabama"]
print(states_in_india + states_in_us)
```

```
['Tamil Nadu', 'Karnataka', 'Haryana', 'California', 'Florida', 'Texas', 'Alabama']
```

```python
# Append
states_in_india.append("Kerala")
print(states_in_india)
```

```
['Tamil Nadu', 'Karnataka', 'Haryana', 'Kerala']
```

Tuples



Int

...
-2
-1
0
1
2
...

Int

...
-2
-1
0
1
2
...

Tuple of (int x int)

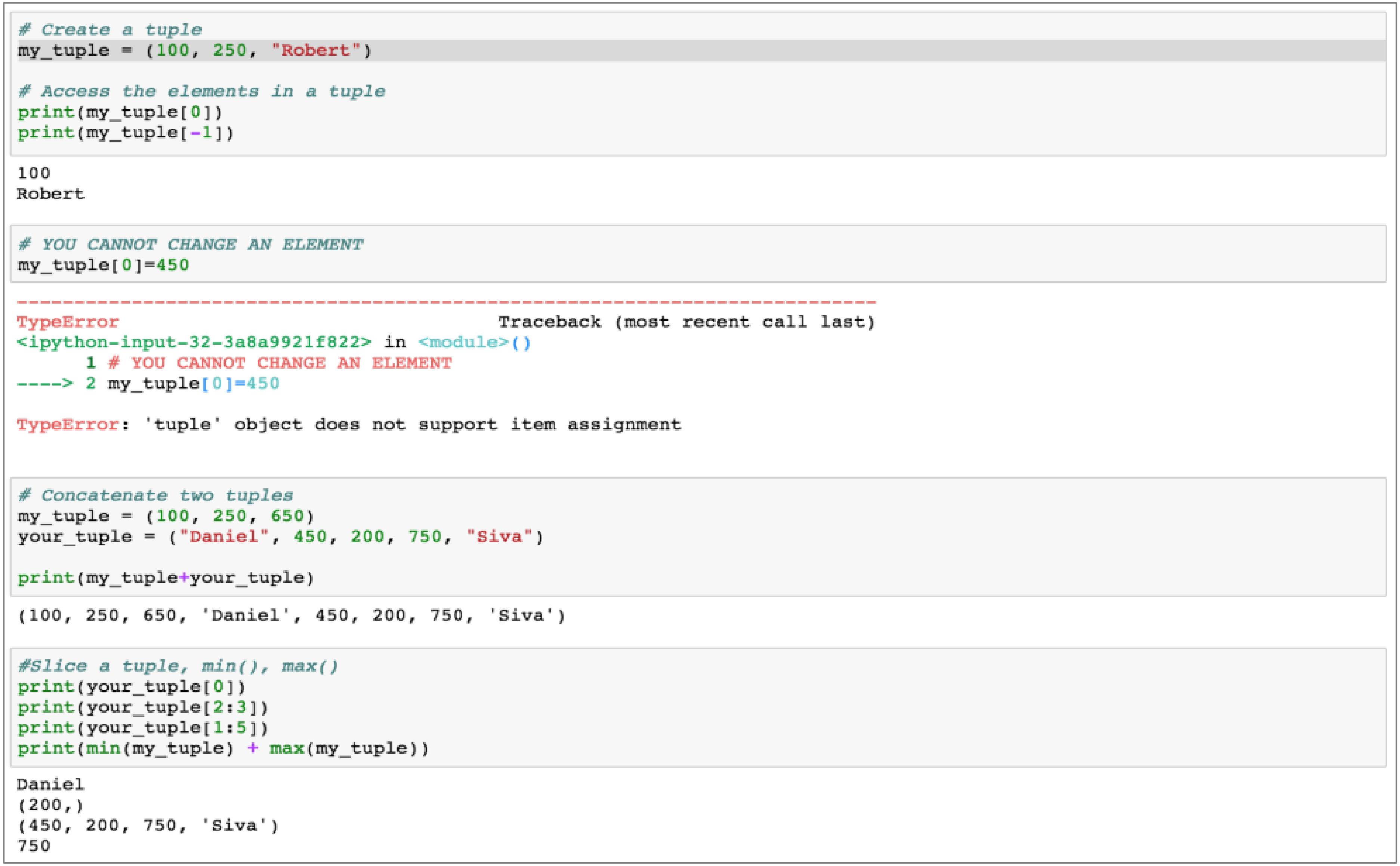
...
..., (-2,-2), (-2,-1), (-2,0), (-2,1), (-2,2),
...
..., (-1,-2), (-1,-1), (-1,0), (-1,1), (-1,2), ...
..., (0,-2), (0,-1), (0,0), (0,1), (0,2), ...
..., (1,-2), (1,-1), (1,0), (1,1), (1,2), ...
..., (2,-2), (2,-1), (2,0), (2,1), (2,2), ...
...

Tuple is a collection of objects separated by commas (,)

The difference between tuples and list is that tuples are immutable, which means once defined you cannot delete, add or edit any values inside it

Like sting indices, tuple indices start at 0 and they can be sliced and concatenated

Example: Tuples



```
# Create a tuple
my_tuple = (100, 250, "Robert")

# Access the elements in a tuple
print(my_tuple[0])
print(my_tuple[-1])
```

```
100
Robert
```

```
# YOU CANNOT CHANGE AN ELEMENT
my_tuple[0]=450
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-3a8a9921f822> in <module>()
      1 # YOU CANNOT CHANGE AN ELEMENT
----> 2 my_tuple[0]=450

TypeError: 'tuple' object does not support item assignment
```

```
# Concatenate two tuples
my_tuple = (100, 250, 650)
your_tuple = ("Daniel", 450, 200, 750, "Siva")

print(my_tuple+your_tuple)
```

```
(100, 250, 650, 'Daniel', 450, 200, 750, 'Siva')
```

```
#Slice a tuple, min(), max()
print(your_tuple[0])
print(your_tuple[2:3])
print(your_tuple[1:5])
print(min(my_tuple) + max(my_tuple))
```

```
Daniel
(200,)
(450, 200, 750, 'Siva')
750
```

Dictionary

Dictionary is an unordered collection of key-value pairs

Use { } curly brackets to construct the dictionary

[ ] square brackets to index it

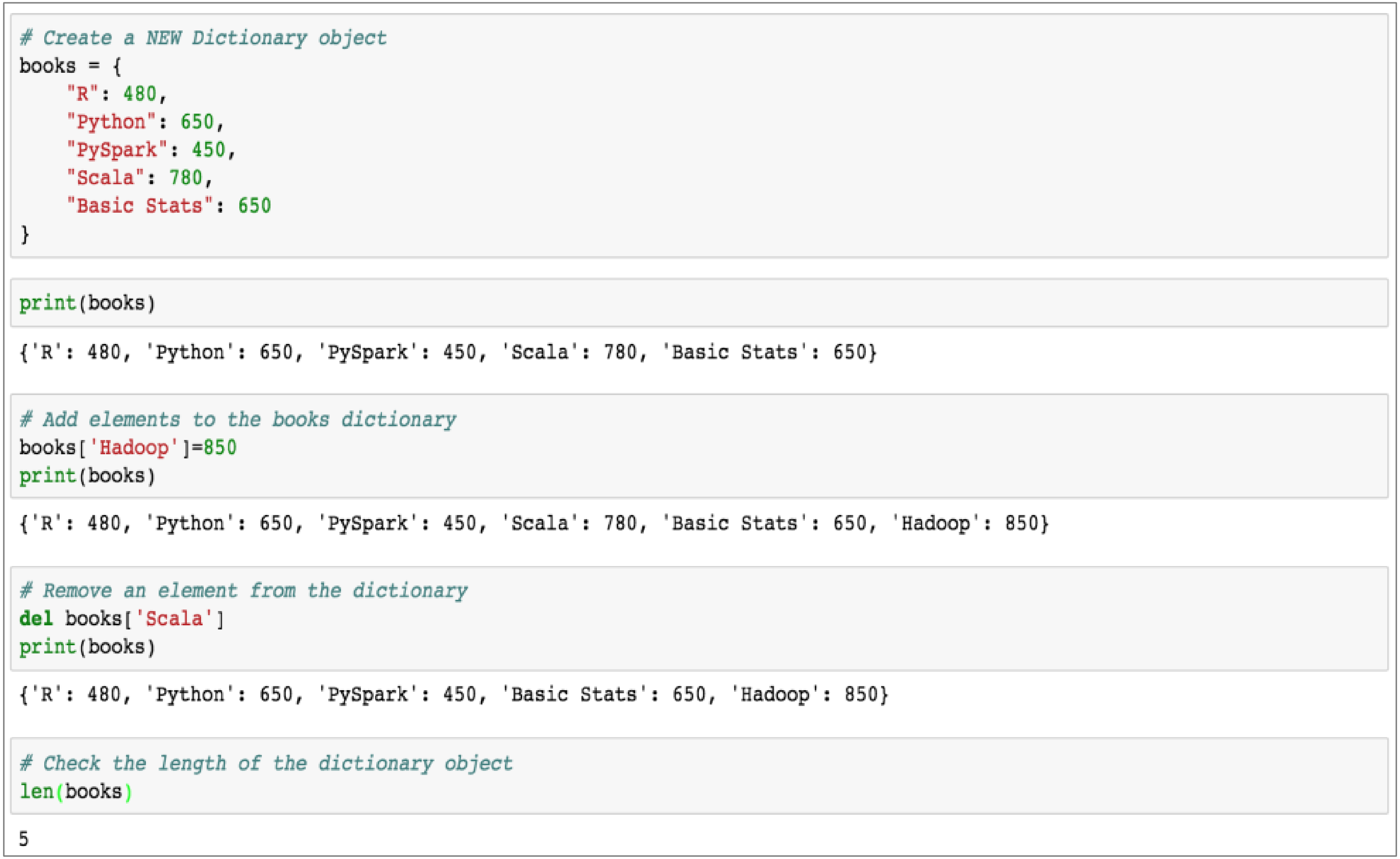
You can:

Access values in a dictionary

Update dictionary

Delete dictionary elements

Example: Dictionary Objects



```
# Create a NEW Dictionary object
books = {
     "R": 480,
     "Python": 650,
     "PySpark": 450,
     "Scala": 780,
     "Basic Stats": 650
}
```

```
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Scala': 780, 'Basic Stats': 650}
```

```
# Add elements to the books dictionary
books['Hadoop']=850
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Scala': 780, 'Basic Stats': 650, 'Hadoop': 850}
```

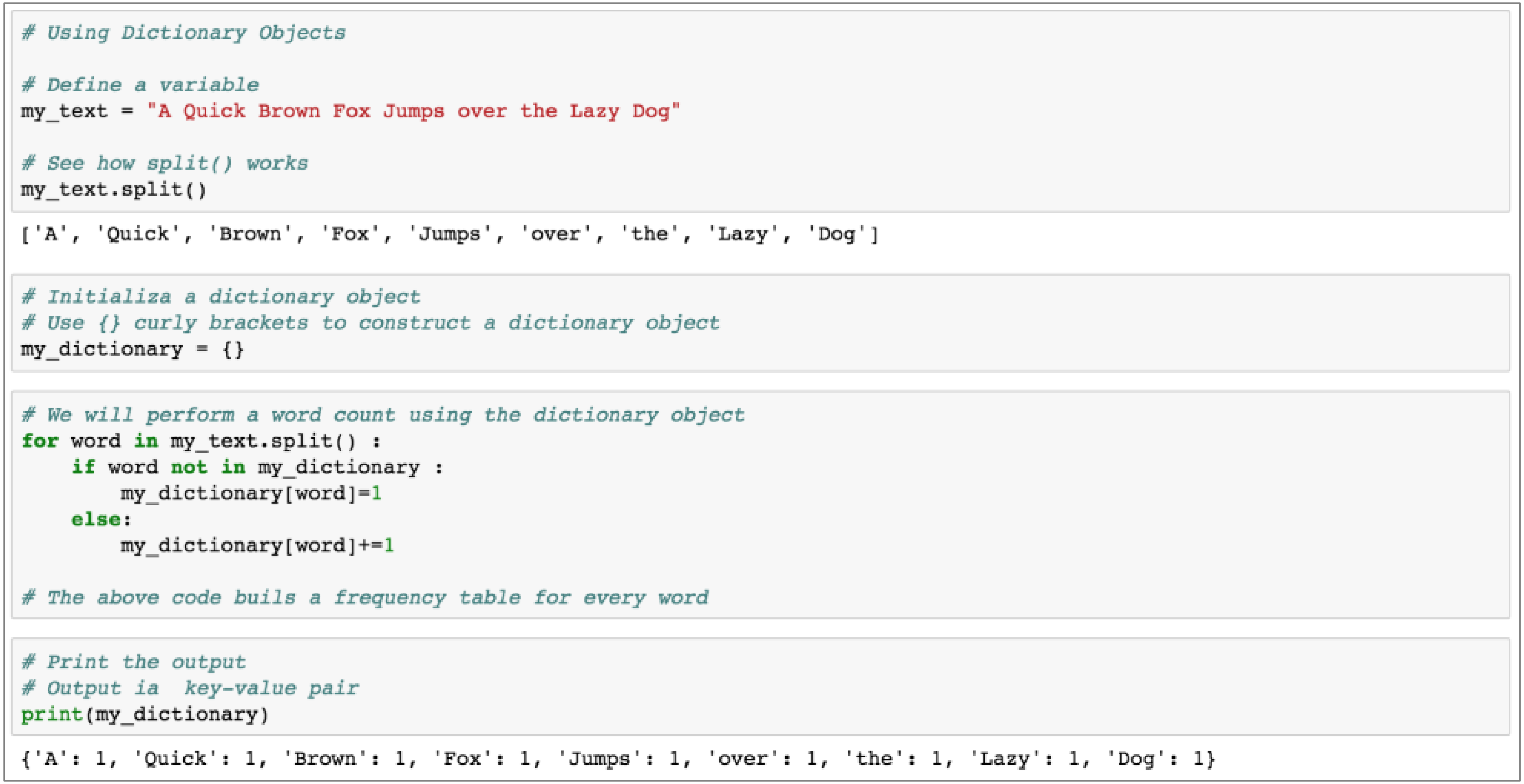
```
# Remove an element from the dictionary
del books['Scala']
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Basic Stats': 650, 'Hadoop': 850}
```

```
# Check the length of the dictionary object
len(books)
```

```
5
```

Example: Dictionary Objects (contd)



```
# Using Dictionary Objects

# Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()
```

```
['A', 'Quick', 'Brown', 'Fox', 'Jumps', 'over', 'the', 'Lazy', 'Dog']
```

```
# Initializa a dictionary object
# Use {} curly brackets to construct a dictionary object
my_dictionary = {}
```

```
# We will perform a word count using the dictionary object
for word in my_text.split() :
    if word not in my_dictionary :
        my_dictionary[word]=1
    else:
        my_dictionary[word]+=1

# The above code buils a frequency table for every word
```

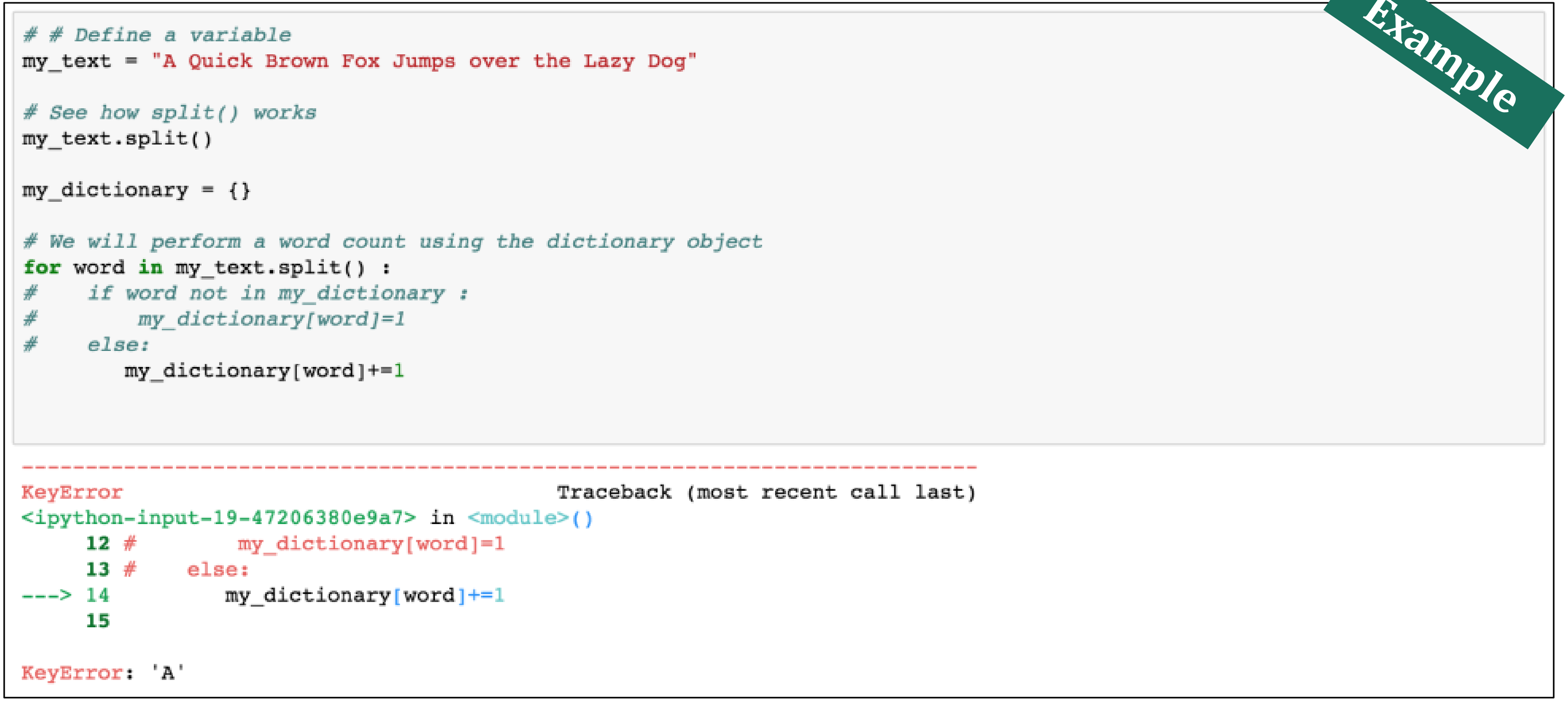
```
# Print the output
# Output ia  key-value pair
print(my_dictionary)
```

```
{'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Dog': 1}
```

Default Dictionary

- A class named default dictionary, its in the collection module
- This takes care of the Key Error

Example



```
# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = {}

# We will perform a word count using the dictionary object
for word in my_text.split() :
#     if word not in my_dictionary :
#         my_dictionary[word]=1
#     else:
        my_dictionary[word]+=1
```

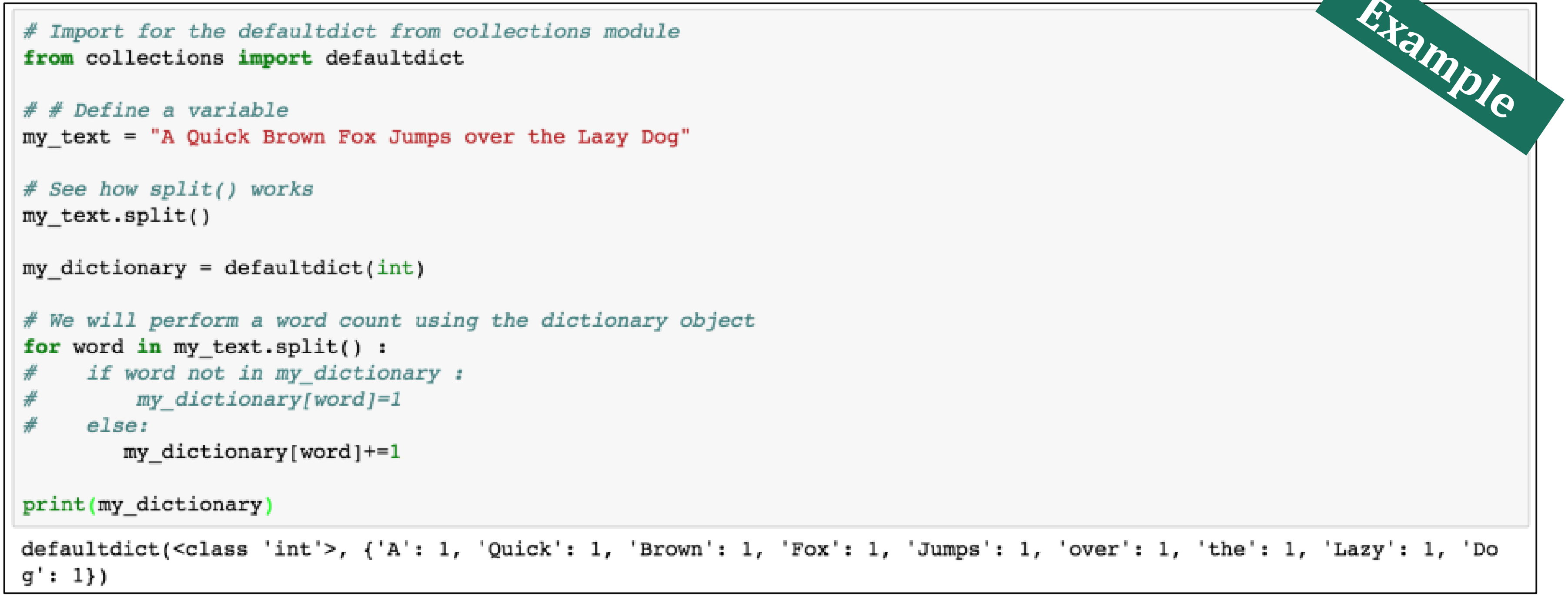
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-47206380e9a7> in <module>()
     12 #         my_dictionary[word]=1
     13 #     else:
---> 14         my_dictionary[word]+=1
     15

KeyError: 'A'
```

Default Dictionary

- Import the defaultdict from collections module
- Initialize the dictionary object with defaultdict( )
- When the dictionary object encounters a key that was not seen before, it initializes the key with a value returned by int( ), in this case 0 (zero)

Note: We passed int( ) to the defaultdict( );

Example



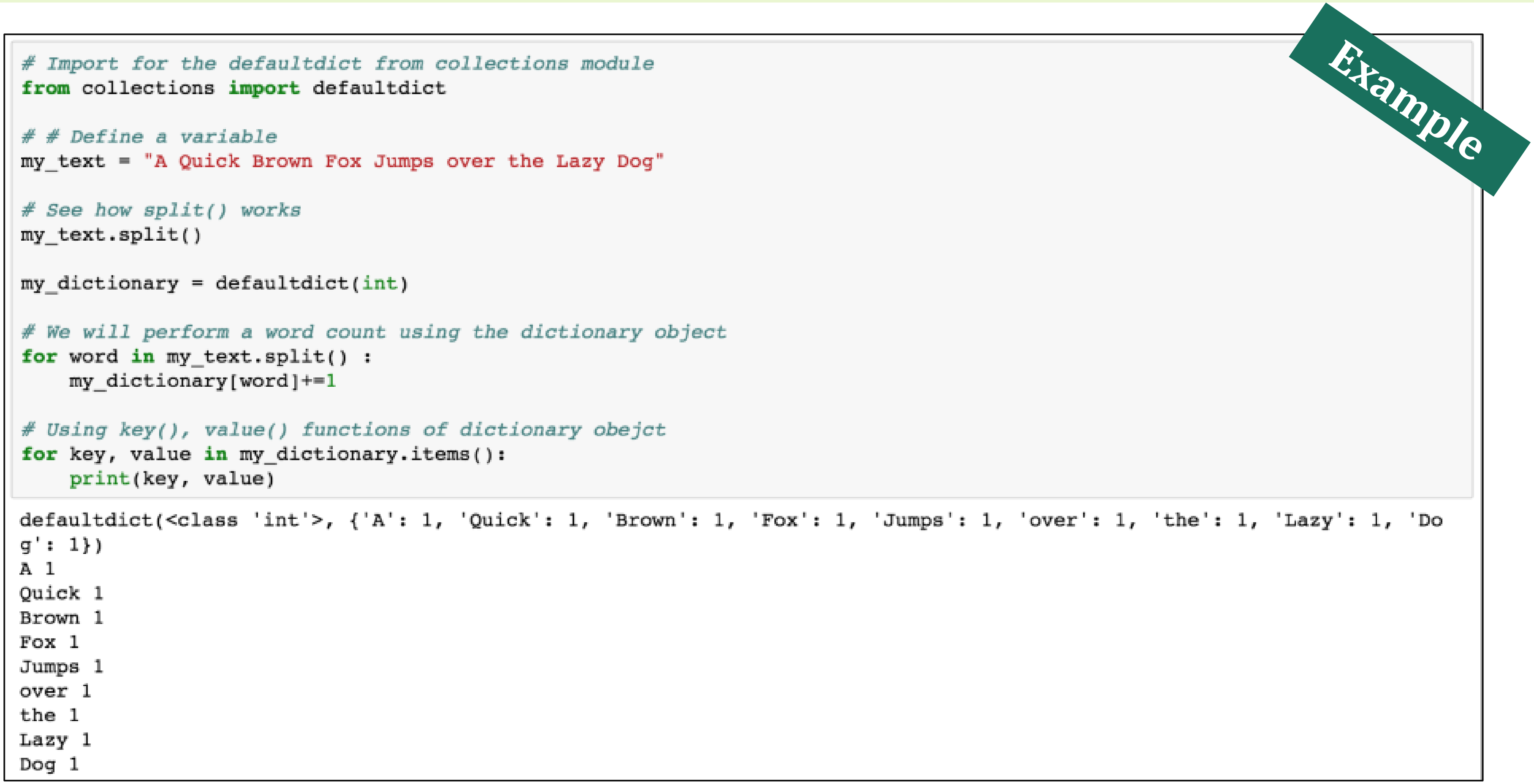
```
# Import for the defaultdict from collections module
from collections import defaultdict

# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = defaultdict(int)

# We will perform a word count using the dictionary object
for word in my_text.split() :
#    if word not in my_dictionary :
#        my_dictionary[word]=1
#    else:
        my_dictionary[word]+=1

print(my_dictionary)
```

```
defaultdict(<class 'int'>, {'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Do
g': 1})
```

Loop Through the Dictionary

- Use keys( ) to loop through the keys in the dictionary object
- Use values( ) to loop through the values in the dictionary object

Example



```python
# Import for the defaultdict from collections module
from collections import defaultdict

# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = defaultdict(int)

# We will perform a word count using the dictionary object
for word in my_text.split() :
    my_dictionary[word]+=1

# Using key(), value() functions of dictionary obejct
for key, value in my_dictionary.items():
    print(key, value)
```

```
defaultdict(<class 'int'>, {'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Do
g': 1})
A 1
Quick 1
Brown 1
Fox 1
Jumps 1
over 1
the 1
Lazy 1
Dog 1
```

Sets

Set is very similar to list data structure.
Set is unordered collection of homogeneous elements.

Do not allow duplicates

Set is generally used to remove duplicate elements from a list

Set supports the following operations:

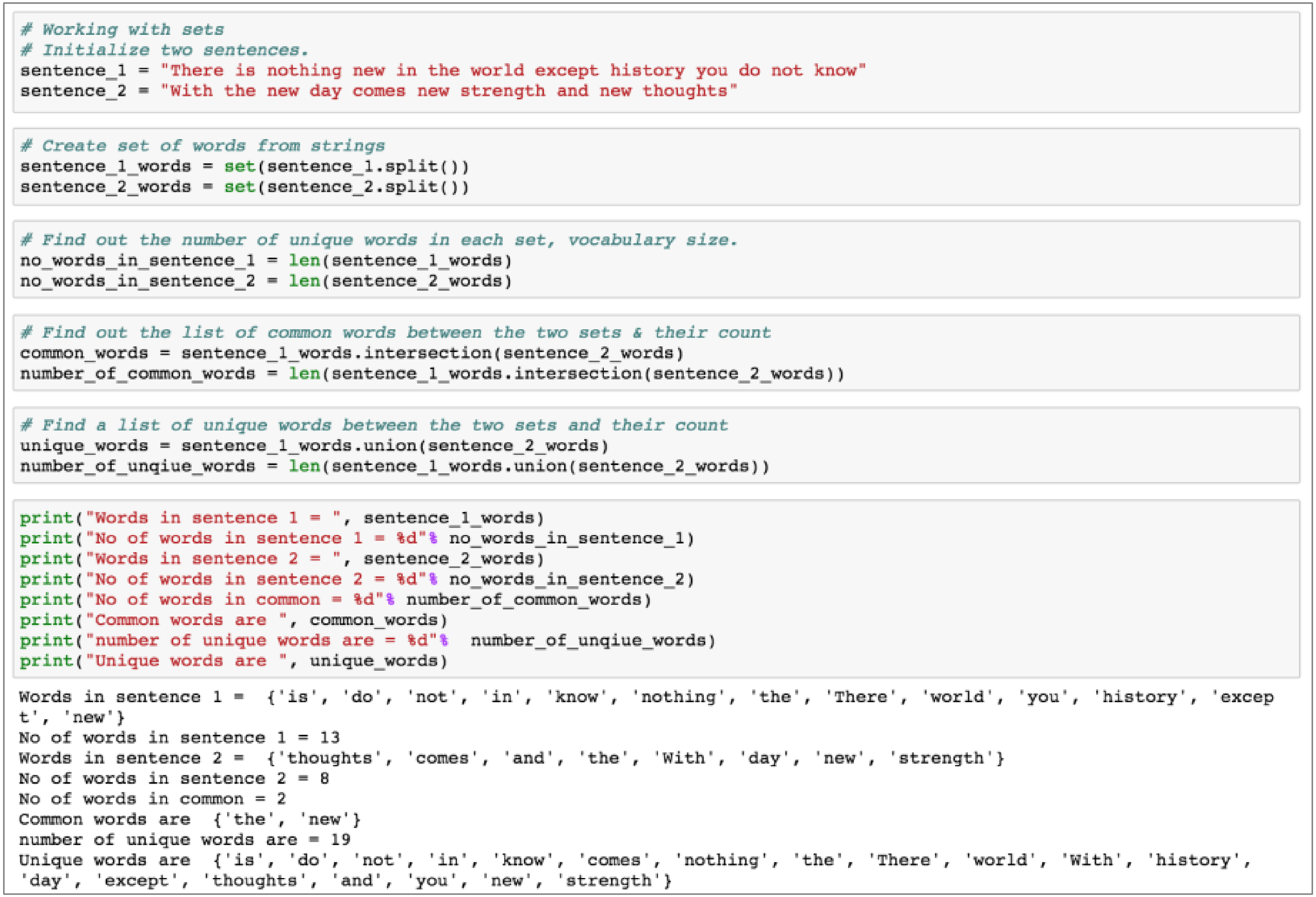
Intersection

Union

Difference

Symmetric Difference

Example: Sets



```
# Working with sets
# Initialize two sentences.
sentence_1 = "There is nothing new in the world except history you do not know"
sentence_2 = "With the new day comes new strength and new thoughts"
```

```
# Create set of words from strings
sentence_1_words = set(sentence_1.split())
sentence_2_words = set(sentence_2.split())
```

```
# Find out the number of unique words in each set, vocabulary size.
no_words_in_sentence_1 = len(sentence_1_words)
no_words_in_sentence_2 = len(sentence_2_words)
```

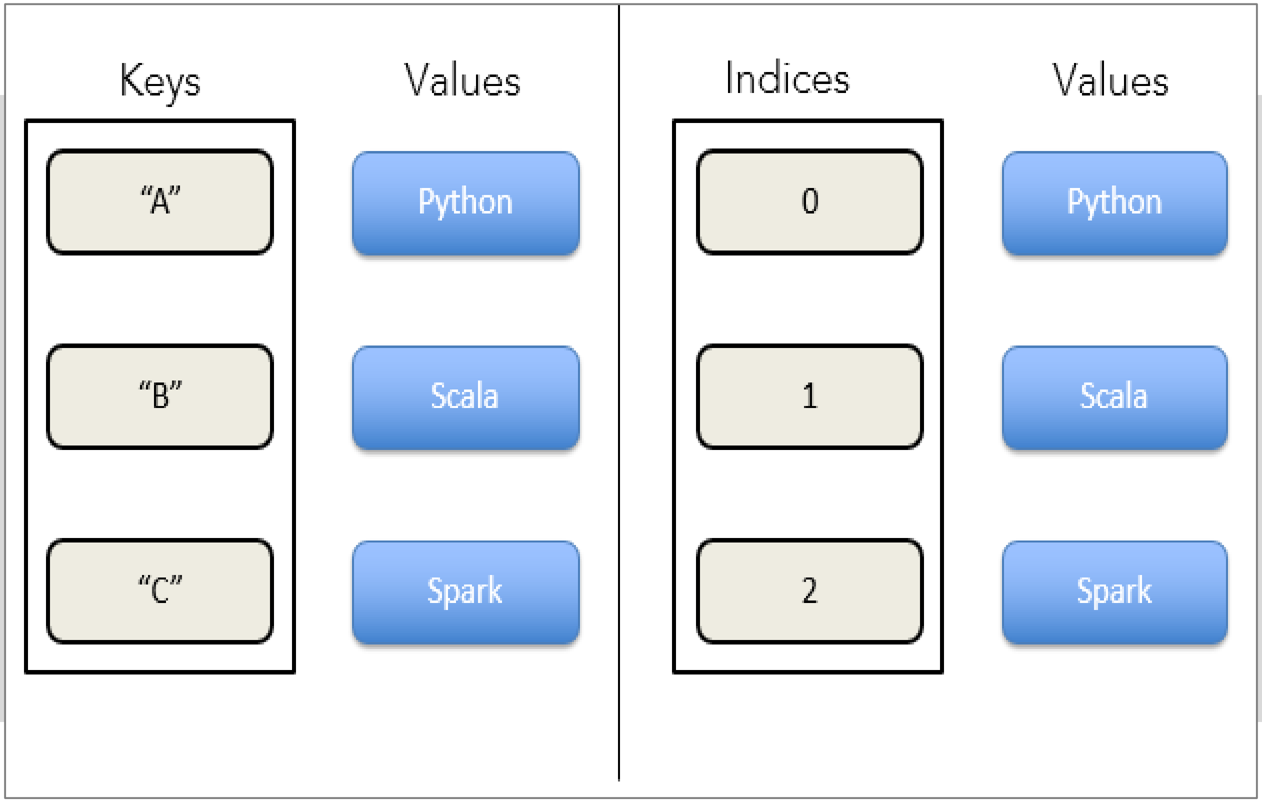
```
# Find out the list of common words between the two sets & their count
common_words = sentence_1_words.intersection(sentence_2_words)
number_of_common_words = len(sentence_1_words.intersection(sentence_2_words))
```

```
# Find a list of unique words between the two sets and their count
unique_words = sentence_1_words.union(sentence_2_words)
number_of_unqiue_words = len(sentence_1_words.union(sentence_2_words))
```

```
print("Words in sentence 1 = ", sentence_1_words)
print("No of words in sentence 1 = %d"% no_words_in_sentence_1)
print("Words in sentence 2 = ", sentence_2_words)
print("No of words in sentence 2 = %d"% no_words_in_sentence_2)
print("No of words in common = %d"% number_of_common_words)
print("Common words are ", common_words)
print("number of unique words are = %d"%  number_of_unqiue_words)
print("Unique words are ", unique_words)
```

```
Words in sentence 1 =  {'is', 'do', 'not', 'in', 'know', 'nothing', 'the', 'There', 'world', 'you', 'history', 'excep
t', 'new'}
No of words in sentence 1 = 13
Words in sentence 2 =  {'thoughts', 'comes', 'and', 'the', 'With', 'day', 'new', 'strength'}
No of words in sentence 2 = 8
No of words in common = 2
Common words are  {'the', 'new'}
number of unique words are = 19
Unique words are  {'is', 'do', 'not', 'in', 'know', 'comes', 'nothing', 'the', 'There', 'world', 'With', 'history',
'day', 'except', 'thoughts', 'and', 'you', 'new', 'strength'}
```

List Vs. Dictionary



Tuple Vs. List

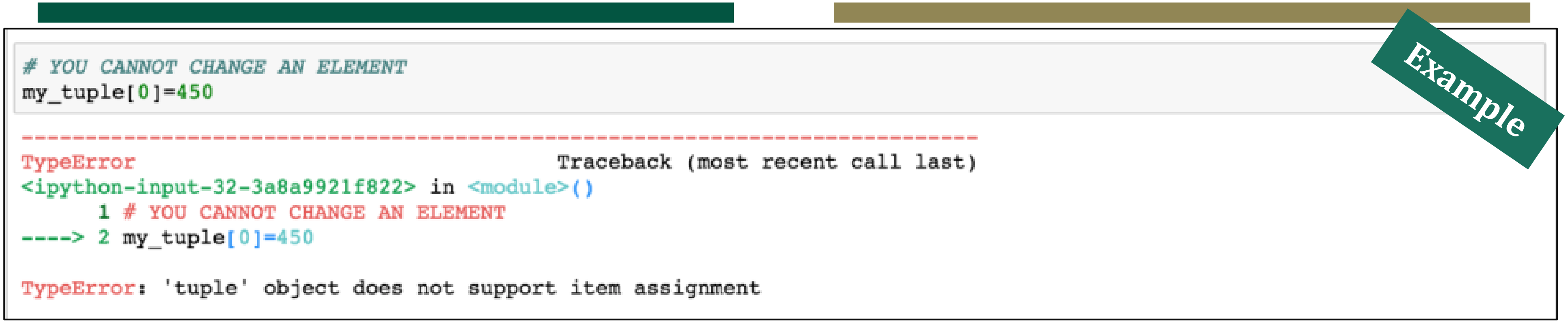
Tuple

- ( )
- Immutable
- Sequences of different kinds
- Faster than List
- Convert list to tuple
- Tu = tuple(li)

List

- [ ]
- Mutable
- Sequences of same kind
- Slower than Tuple
- Convert tuple to list
- Li = list(tu)

Example



```
# YOU CANNOT CHANGE AN ELEMENT
my_tuple[0]=450
```

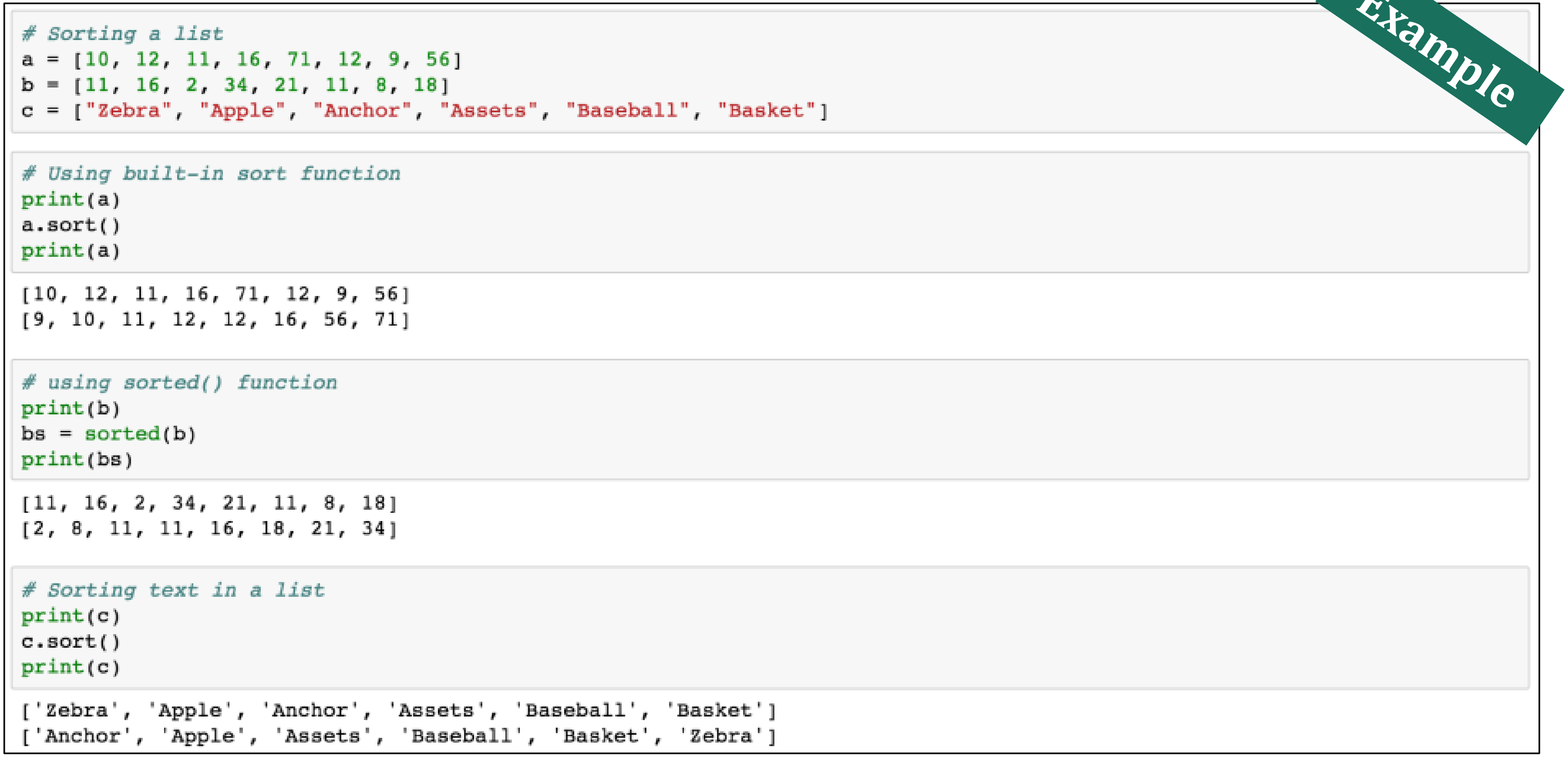
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-3a8a9921f822> in <module>()
      1 # YOU CANNOT CHANGE AN ELEMENT
----> 2 my_tuple[0]=450

TypeError: 'tuple' object does not support item assignment
```

In the above, changing an element in a tuple does not work. Try changing an element in a list and it will work.

Sorting a List

- Use built-in sort function in the list
- Use sorted ( ) function

Example



```
# Sorting a list
a = [10, 12, 11, 16, 71, 12, 9, 56]
b = [11, 16, 2, 34, 21, 11, 8, 18]
c = ["Zebra", "Apple", "Anchor", "Assets", "Baseball", "Basket"]
```

```
# Using built-in sort function
print(a)
a.sort()
print(a)
```

```
[10, 12, 11, 16, 71, 12, 9, 56]
[9, 10, 11, 12, 12, 16, 56, 71]
```

```
# using sorted() function
print(b)
bs = sorted(b)
print(bs)
```

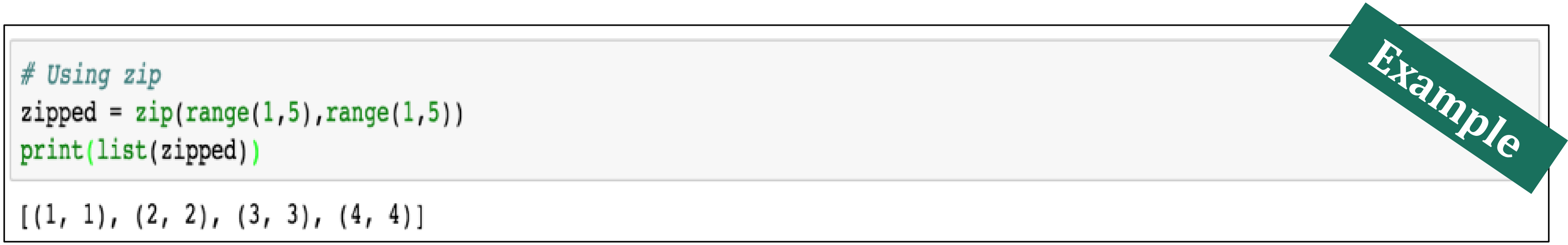
```
[11, 16, 2, 34, 21, 11, 8, 18]
[2, 8, 11, 11, 16, 18, 21, 34]
```

```
# Sorting text in a list
print(c)
c.sort()
print(c)
```

```
['Zebra', 'Apple', 'Anchor', 'Assets', 'Baseball', 'Basket']
['Anchor', 'Apple', 'Assets', 'Baseball', 'Basket', 'Zebra']
```

Zip

- Zip - a built-in function
- Zip takes 2 equal length collections and merges them in pairs



Example

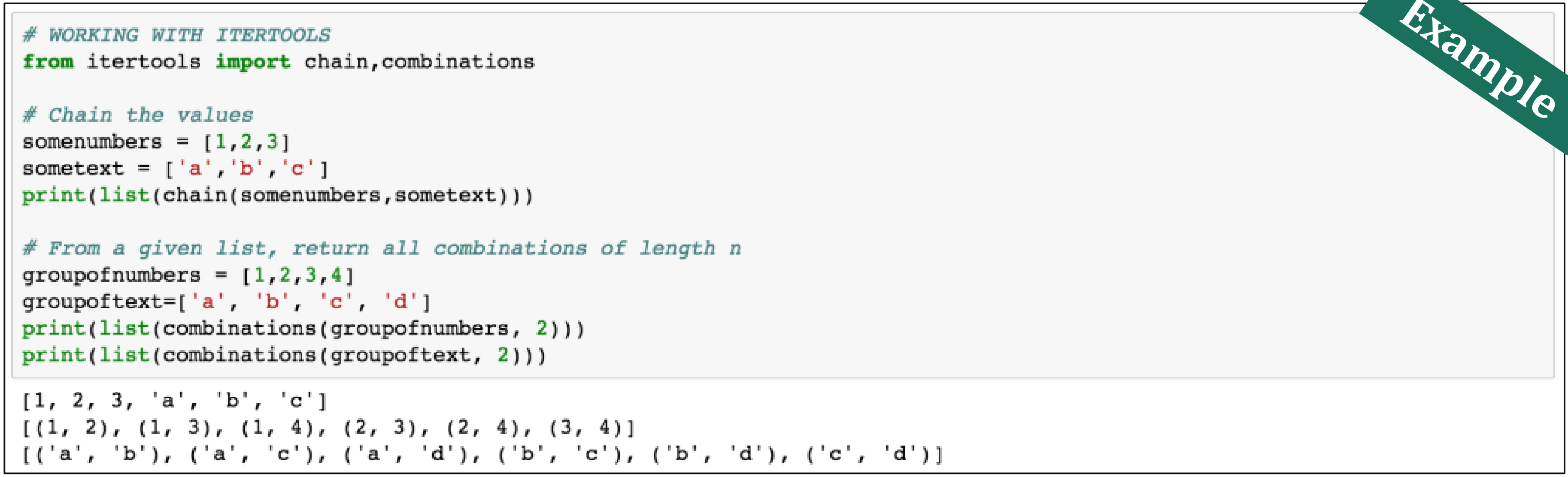
```
# Using zip
zipped = zip(range(1,5),range(1,5))
print(list(zipped))
```

```
[(1, 1), (2, 2), (3, 3), (4, 4)]
```

Itertools

- Itertools includes functions to work with iterables
- Memory-efficient & fast

Example



```
# WORKING WITH ITERTOOLS
from itertools import chain,combinations

# Chain the values
somenumbers = [1,2,3]
sometext = ['a','b','c']
print(list(chain(somenumbers,sometext)))

# From a given list, return all combinations of length n
groupofnumbers = [1,2,3,4]
groupoftext=['a', 'b', 'c', 'd']
print(list(combinations(groupofnumbers, 2)))
print(list(combinations(groupoftext, 2)))
```

```
[1, 2, 3, 'a', 'b', 'c']
[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
[('a', 'b'), ('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd'), ('c', 'd')]
```

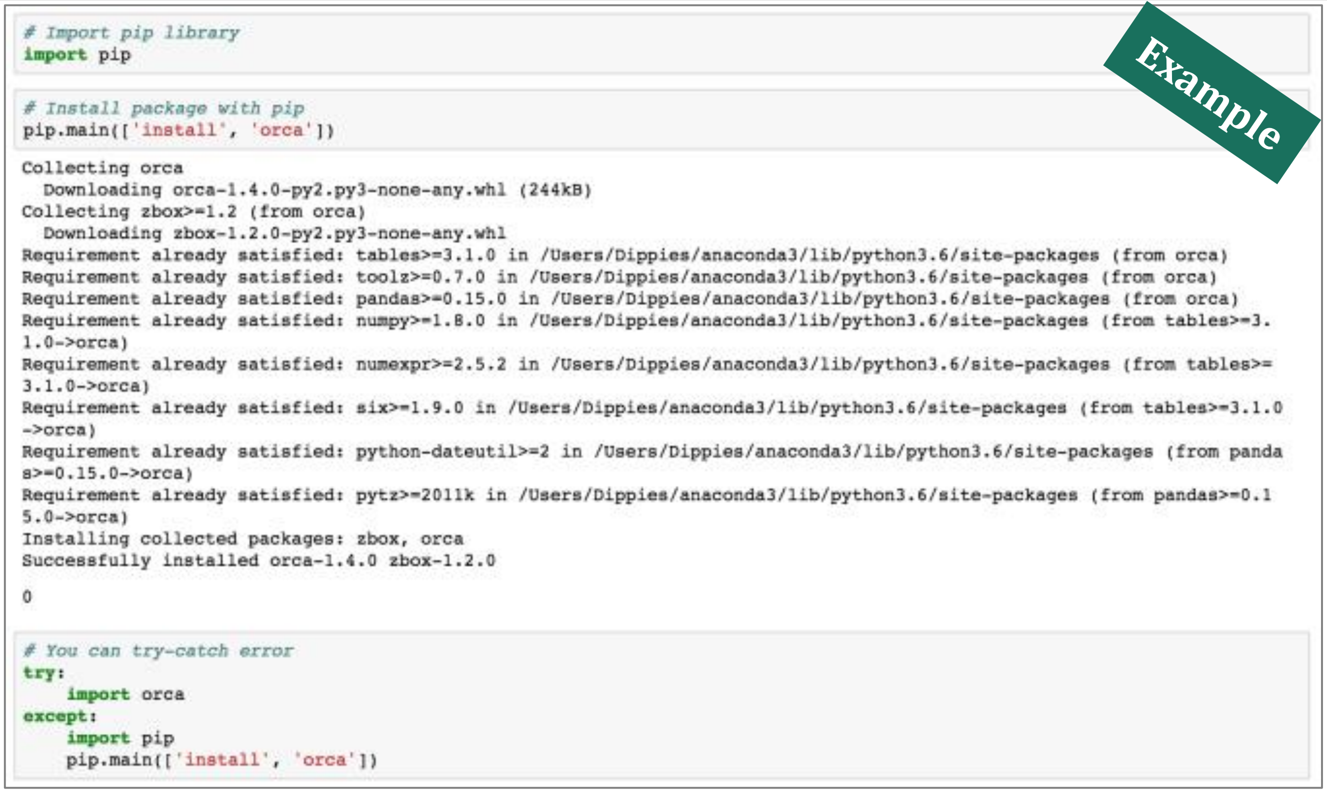
Install Packages in Python

pip command

```
pip install <package name>
pip install <package name> -- upgrade
pip list
```

conda command

```
conda install <package name>
conda install <package name> -- upgrade
conda list
```



```
# Import pip library
import pip
```

```
# Install package with pip
pip.main(['install', 'orca'])
```

```
Collecting orca
  Downloading orca-1.4.0-py2.py3-none-any.whl (244kB)
Collecting zbox>=1.2 (from orca)
  Downloading zbox-1.2.0-py2.py3-none-any.whl
Requirement already satisfied: tables>=3.1.0 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from orca)
Requirement already satisfied: toolz>=0.7.0 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from orca)
Requirement already satisfied: pandas>=0.15.0 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from orca)
Requirement already satisfied: numpy>=1.8.0 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from tables>=3.
1.0->orca)
Requirement already satisfied: numexpr>=2.5.2 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from tables>=
3.1.0->orca)
Requirement already satisfied: six>=1.9.0 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from tables>=3.1.0
->orca)
Requirement already satisfied: python-dateutil>=2 in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from panda
s>=0.15.0->orca)
Requirement already satisfied: pytz>=2011k in /Users/Dippies/anaconda3/lib/python3.6/site-packages (from pandas>=0.1
5.0->orca)
Installing collected packages: zbox, orca
Successfully installed orca-1.4.0 zbox-1.2.0

0
```

```
# You can try-catch error
try:
    import orca
except:
    import pip
    pip.main(['install', 'orca'])
```