Python and Libraries

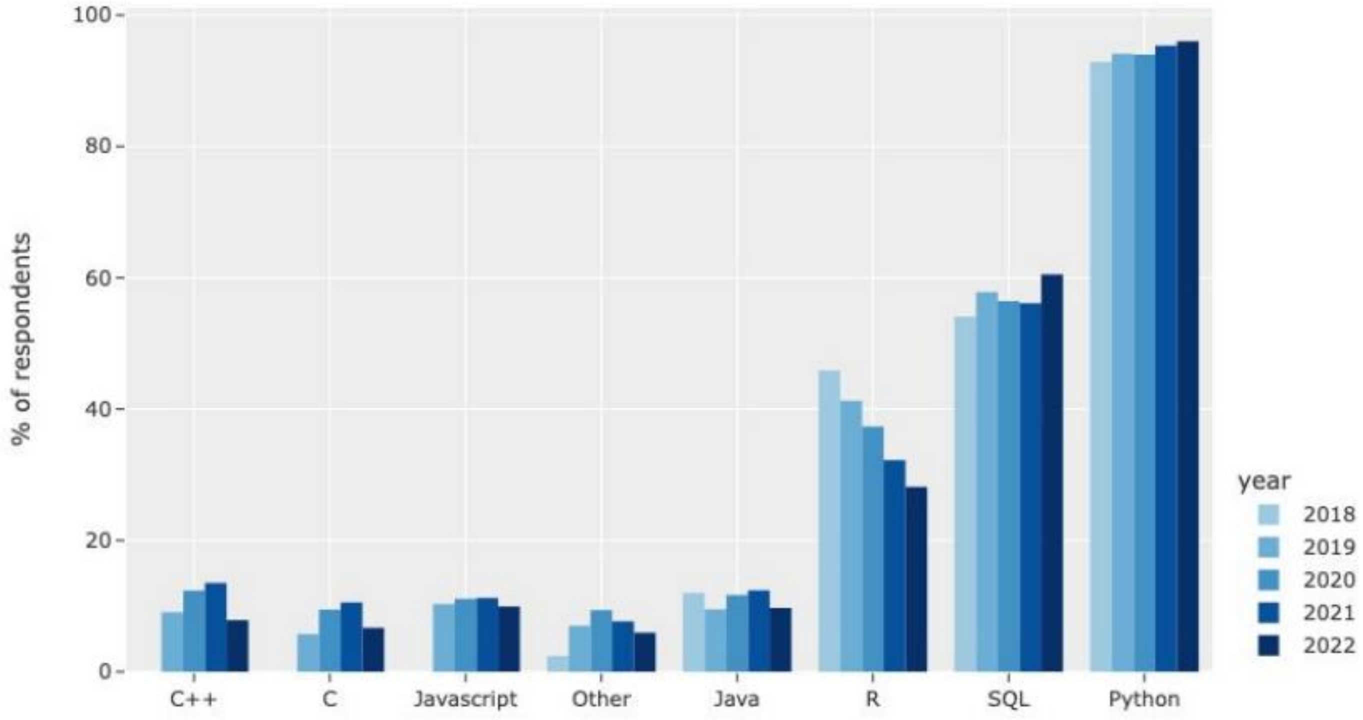
Juntae Kim

Department of Computer Science and Engineering

Dongguk University

Kaggle's DS & ML Survey 2022

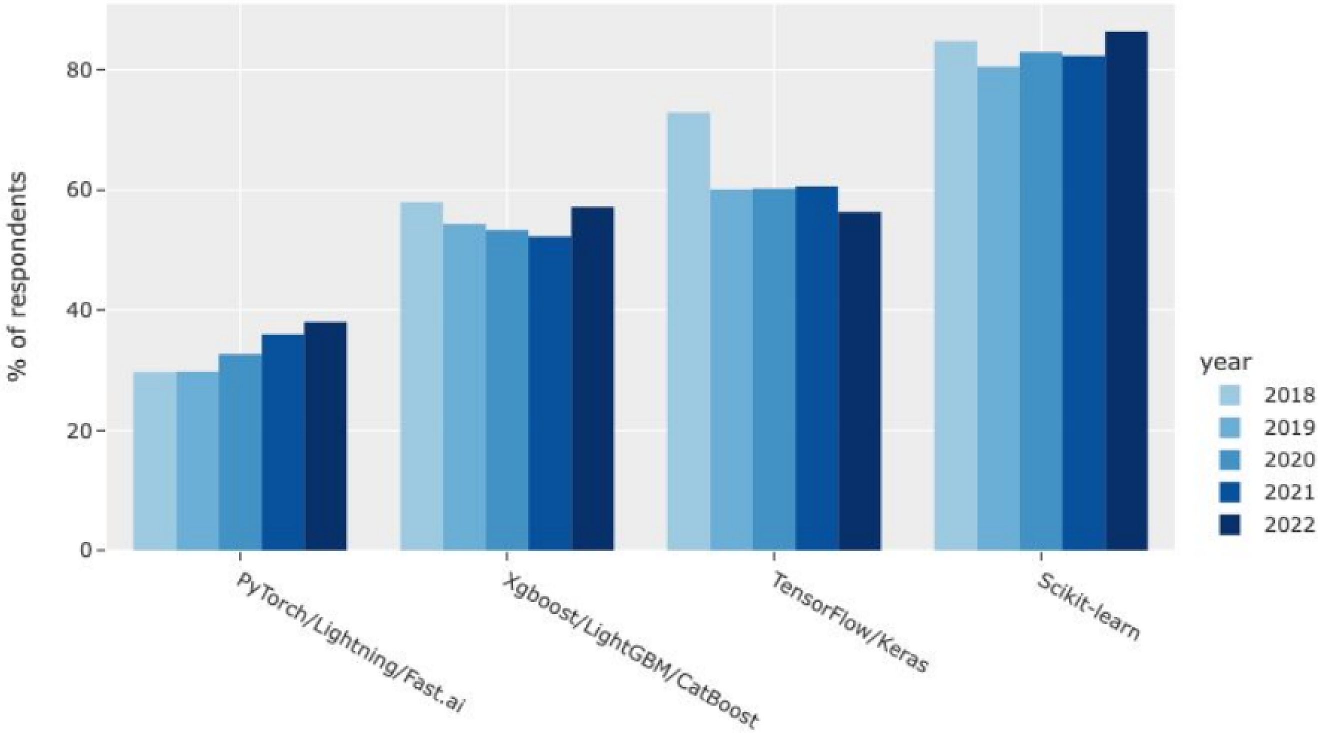
- Most common programming skills for data scientists



https://www.kaggle.com/kaggle-survey-2022

Kaggle’s DS & ML Survey 2022

- Most popular Machine Learning framework



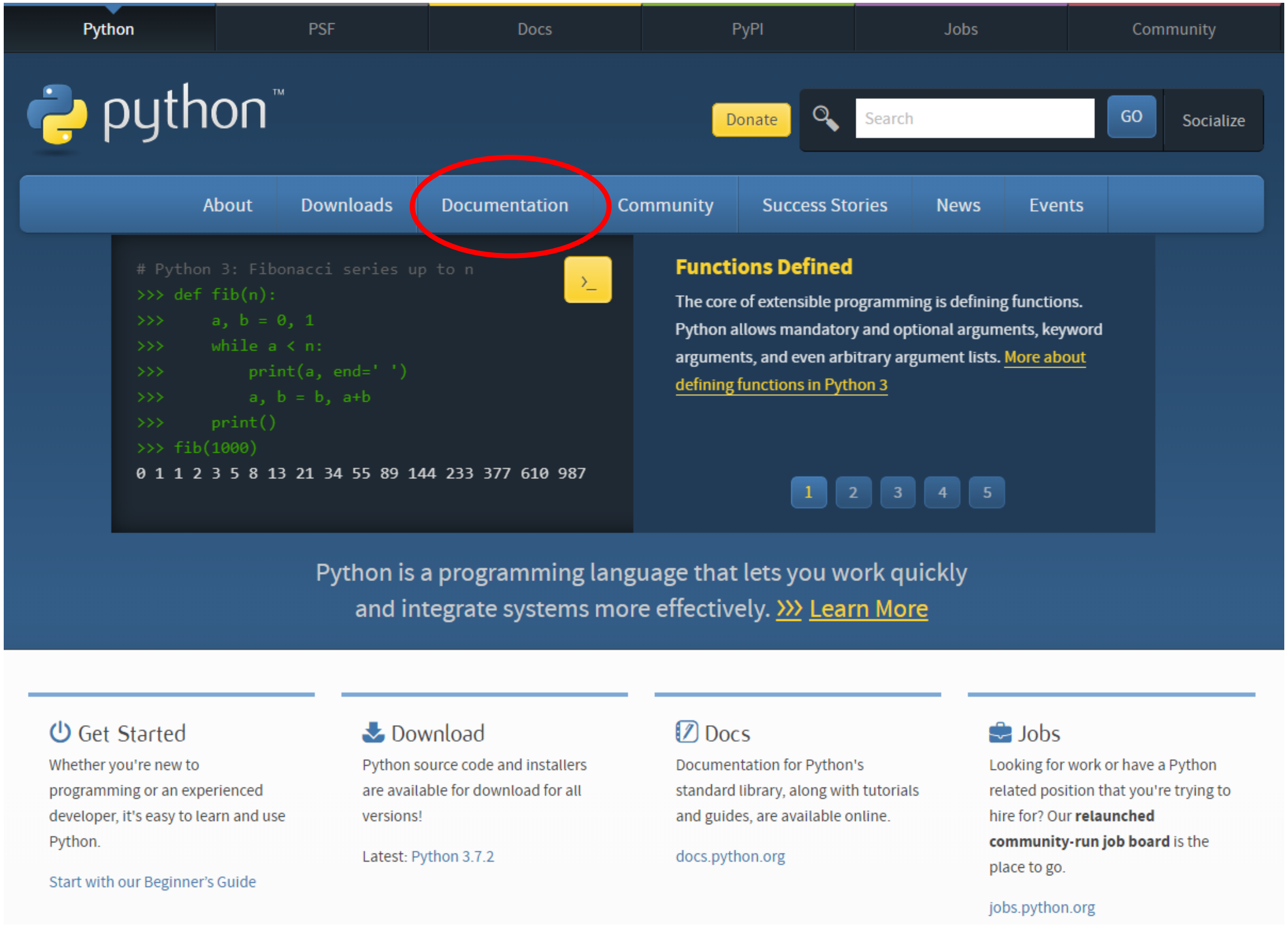
https://www.kaggle.com/kaggle-survey-2022

What is Python?

- Python
  - High-level programming language released in 1991, which was created by Guido van Rossum
  - Python is interpretive, object-oriented, dynamic typed(check the data type in run time) and interactive programming language
  - Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming
  - The Zen of Python
    - Beautiful is better than ugly
    - Explicit is better than implicit
    - Simple is better than complex
    - Flat is better than nested
    - Sparse is better than dense
    - Readability counts
    - …

https://www.python.org/dev/peps/pep-0020

www.python.org



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

Start with our Beginner's Guide

Download

Python source code and installers are available for download for all versions!

Latest: Python 3.7.2

Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

docs.python.org

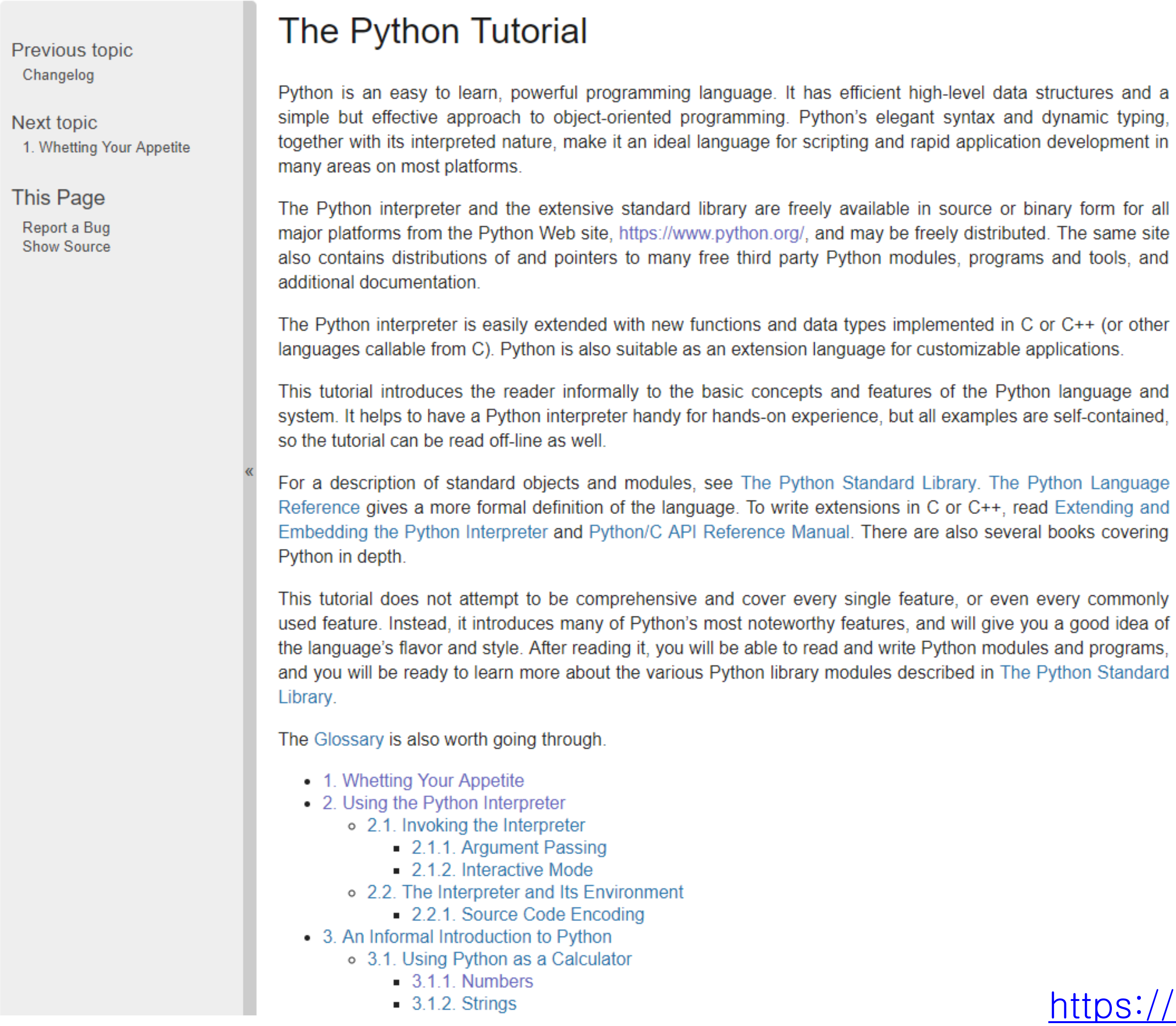
Jobs

Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.

jobs.python.org

https://www.python.org

Tutorial



Previous topic

Changelog

Next topic

1. Whetting Your Appetite

This Page

Report a Bug

Show Source

«

The Python Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in The Python Standard Library.

The Glossary is also worth going through.

Documentation

↓

Python Docs

↓

Tutorial

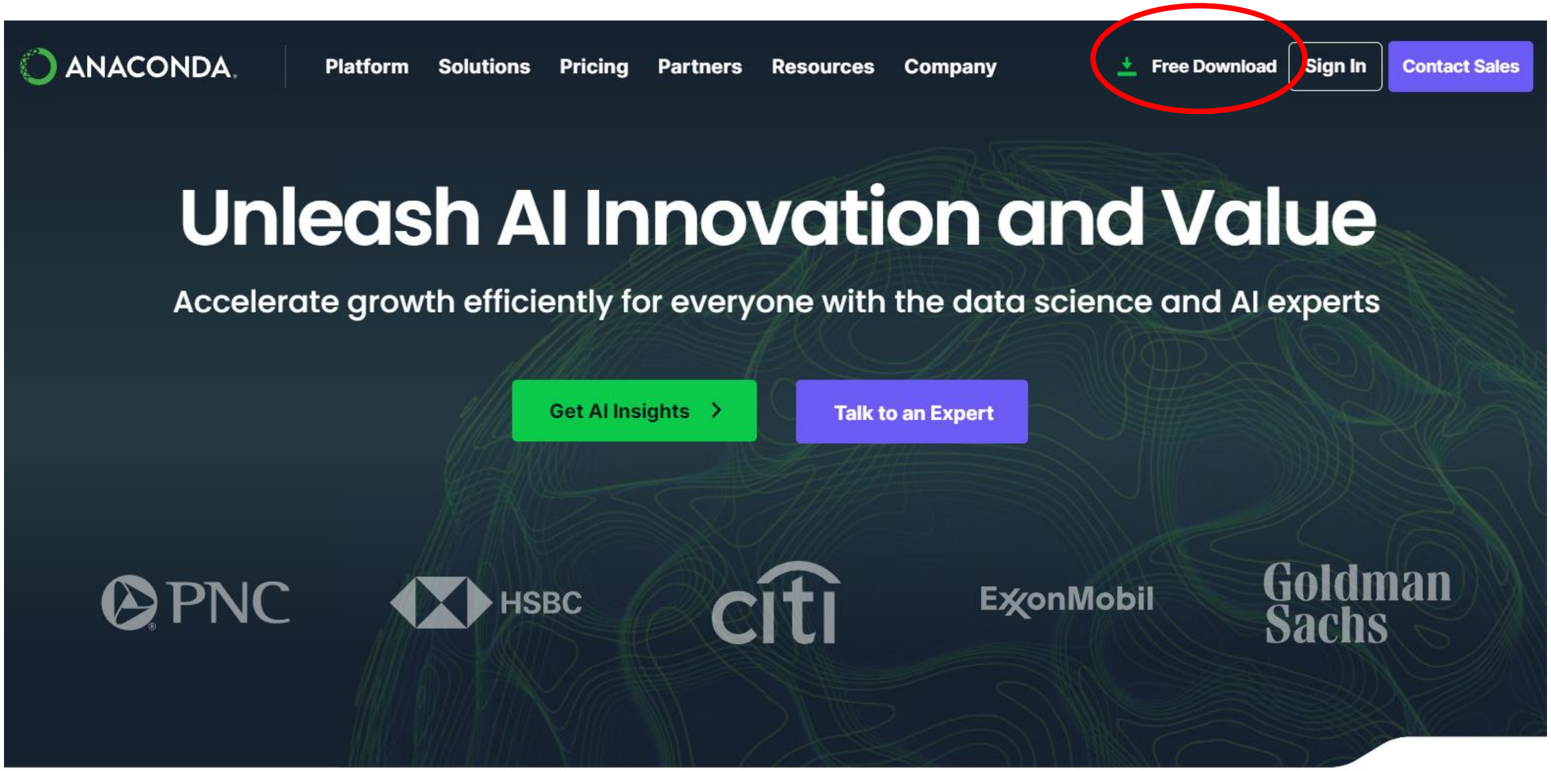
https://docs.python.org/3/tutorial/

Anaconda

- Anaconda
  - A free and open-source distribution of the Python programming languages for scientific computing
  - Python + Libraries + Tools
- Typical AI/ML-related libraries supported by Anaconda
  - Numpy
    - It provides multidimensional array object, vector operation and linear algebra
  - Pandas
    - It provides 'Dataframe' to address the type of table data
  - Matplotlib
    - It provides several tools of drawing graph, chart and visualization
  - Scikit-Learn
    - It provides packages of some machine Learning algorithms and various models of machine learning functions

https://www.anaconda.com/

www.anaconda.com



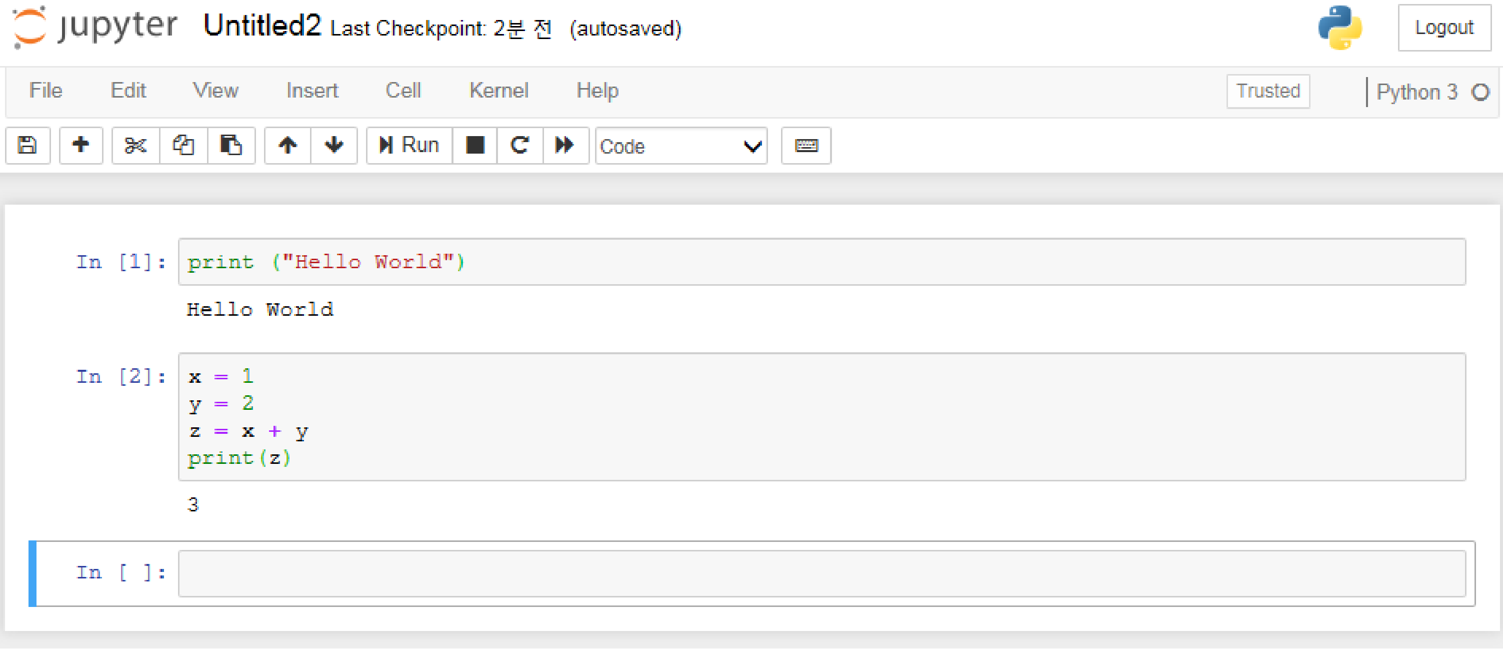
Unleash AI Innovation and Value

Accelerate growth efficiently for everyone with the data science and AI experts

https://www.anaconda.com/

Jupyter Notebook

- Jupyter Notebook
  - A tool for writing codes and executing the codes in the web browser
    1. Execute the Jupyter Notebook
    2. Go to your folder
    3. New → Python 3 (Create a script) → Write codes → Run



```
In [1]: print ("Hello World")

        Hello World

In [2]: x = 1
        y = 2
        z = x + y
        print(z)

        3

In [ ]:
```

Python Programming Basic

- Contents
  - Comments
  - Constants, Variable
  - Conditional Statements
  - Iteration Statements
  - Function
  - List, Dictionary, Tuple, Set
  - File
  - Class

https://docs.python.org/3/tutorial/

Comments

- Sentences to explain the codes

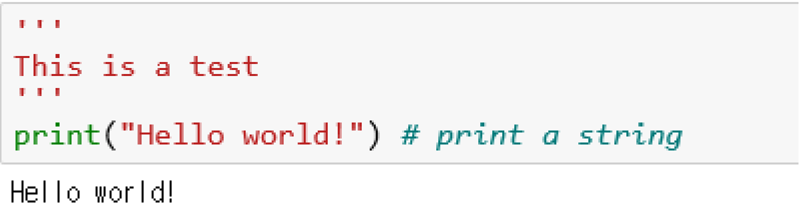
Comment

C, C++

// comment
/*
multi–line comments
*/

Python

# comment
’ ’ ’
multi–line comments
’ ’ ’

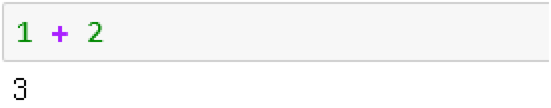


```
'''
This is a test
'''
print("Hello world!") # print a string
```

```
Hello world!
```

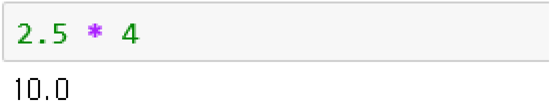
Constants

- Fixed values(unchangeable numbers, strings, etc.)
  - Numbers
    - Integer: -2, 0, 100
    - Real numbers: -2.5, 0.0, 99.9
  - String constant
    - Single/double quotation marks(‘ or “)
    - ‘hello’, “hello”
  - Logical/special constant
    - True, False
    - None



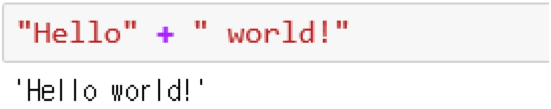
```
1 + 2
```

```
3
```



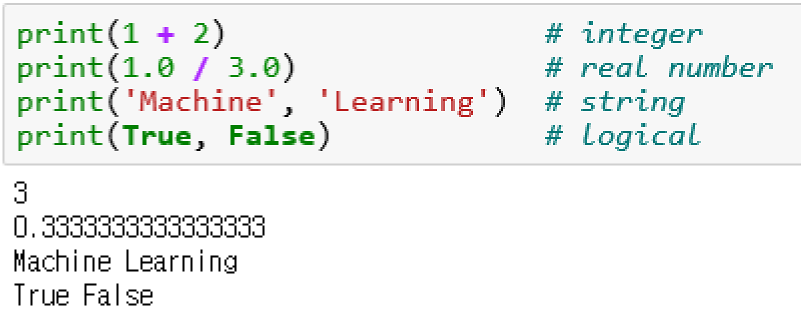
```
2.5 * 4
```

```
10.0
```



```
"Hello" + " world!"
```

```
'Hello world!'
```

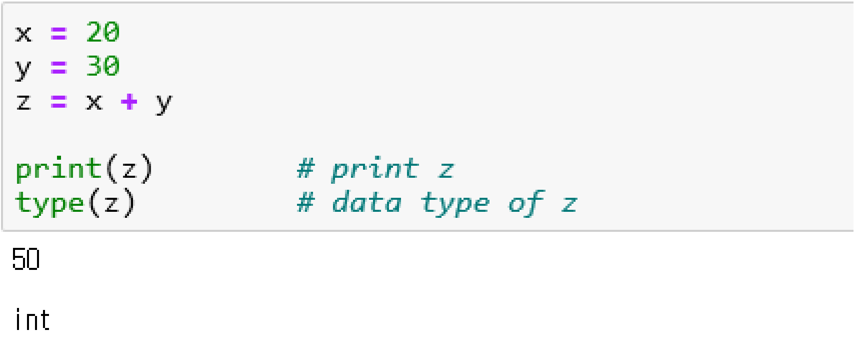


```
print(1 + 2)                      # integer
print(1.0 / 3.0)                  # real number
print('Machine', 'Learning')      # string
print(True, False)                # logical
```

```
3
0.3333333333333333
Machine Learning
True False
```

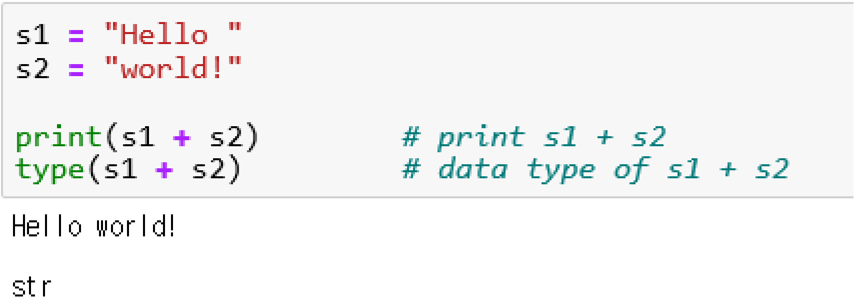
Variables

- Named memory to store values
  - Name consist of alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Name cannot start with a number
  - Data types can be checked by the function type()



```
x = 20
y = 30
z = x + y

print(z)        # print z
type(z)         # data type of z
```

```
50

int
```



```
s1 = "Hello "
s2 = "world!"

print(s1 + s2)        # print s1 + s2
type(s1 + s2)         # data type of s1 + s2
```

```
Hello world!

str
```

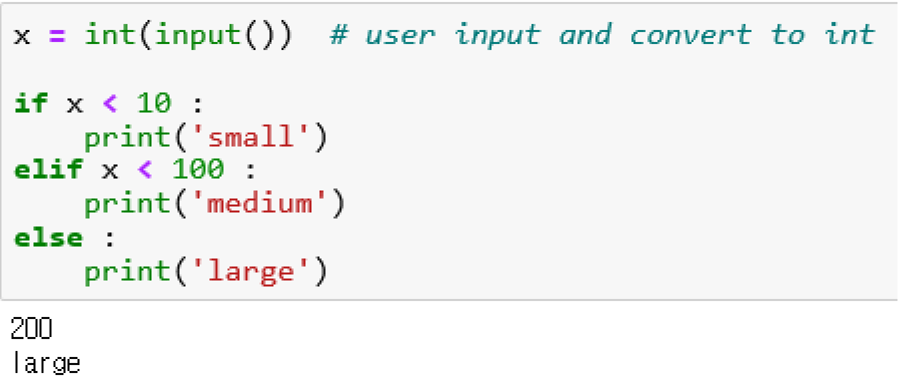
Operator	Operation
+	Plus
-	Minus
*	Multiply
/	Division
**	Power 2
%	Remainder

Operator priority

power > multiply > plus, minus

Conditional Statements

- If
  - If statement is control flow statements which helps us to run a particular code only when a certain condition is satisfied.



```
x = int(input())  # user input and convert to int

if x < 10 :
    print('small')
elif x < 100 :
    print('medium')
else :
    print('large')
```

```
200
large
```

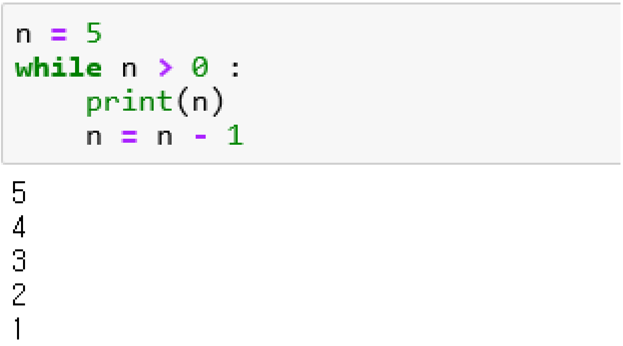
Operator	Meaning
<	Smaller
<=	Smaller or equal
==	Same
>=	Larger or equal
>	Larger
!=	Not equal

Note.

In Python, blocks are marked as indentations in functions, iteration statements and conditional statements, etc. It is strongly recommended that you do not use the tab

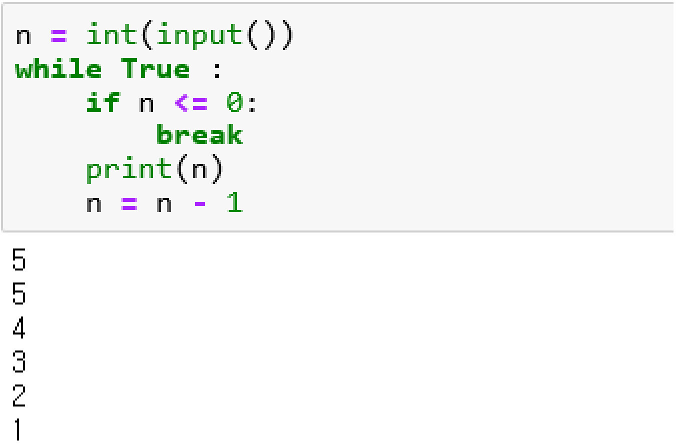
Iteration Statements

- while
  - The while statement enables the execution of a body of statements multiple times in a loop while a certain condition is true.



```
n = 5
while n > 0 :
    print(n)
    n = n - 1
```

```
5
4
3
2
1
```

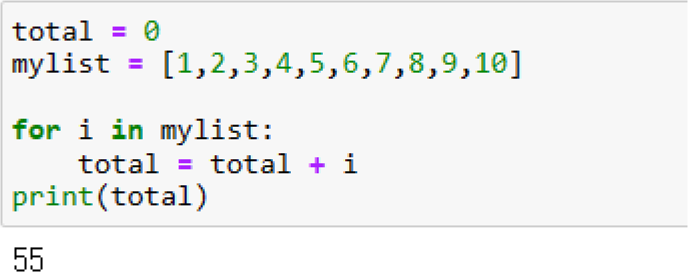


```
n = int(input())
while True :
    if n <= 0:
        break
    print(n)
    n = n - 1
```

```
5
5
4
3
2
1
```

Iteration Statements

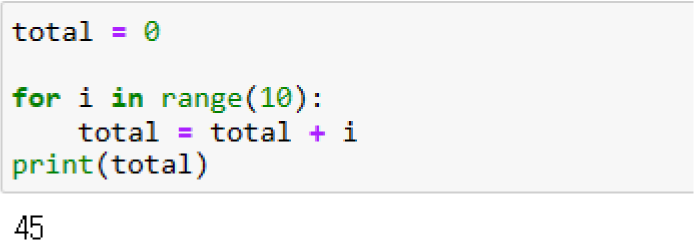
- for
  - for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.
  - list : a compound data types like an array in C (뒤에 설명)



```
total = 0
mylist = [1,2,3,4,5,6,7,8,9,10]

for i in mylist:
    total = total + i
print(total)
```

```
55
```

  - range(n) : it returns a list of numbers 0 to n-1



```
total = 0

for i in range(10):
    total = total + i
print(total)
```

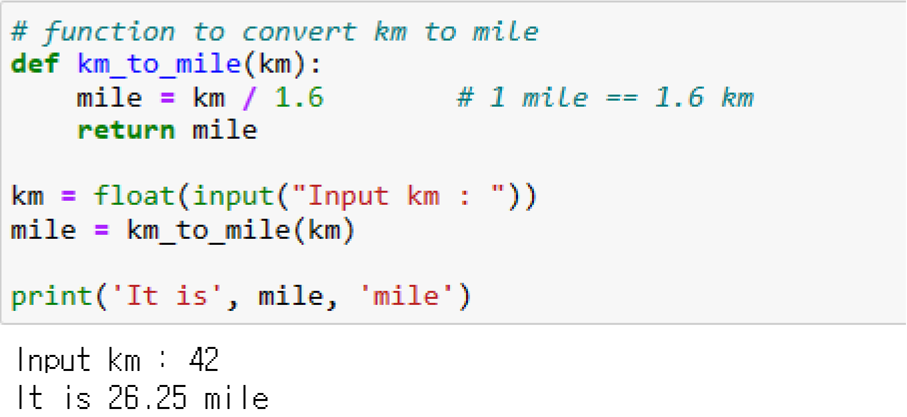
```
45
```

Function

- A named section of a program that performs a specific task. It is reusable code
  - Function definition
    - def function_name(arg_1, arg_2, …) :

      # statements

      return something

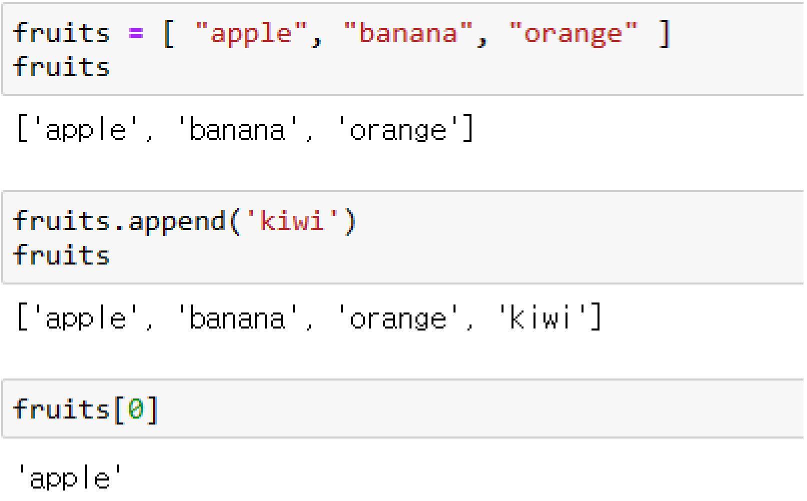


```
# function to convert km to mile
def km_to_mile(km):
    mile = km / 1.6         # 1 mile == 1.6 km
    return mile

km = float(input("Input km : "))
mile = km_to_mile(km)

print('It is', mile, 'mile')
```

```
Input km : 42
It is 26.25 mile
```

List

- A list is a data structure that stores ordered sequence of items
  - It can be initialized as [item1, item2, …]
  - It can be initialized by the function ‘list()’ and then append new item by the function ‘list.append(item)’



```
fruits = [ "apple", "banana", "orange" ]
fruits
```

```
['apple', 'banana', 'orange']
```

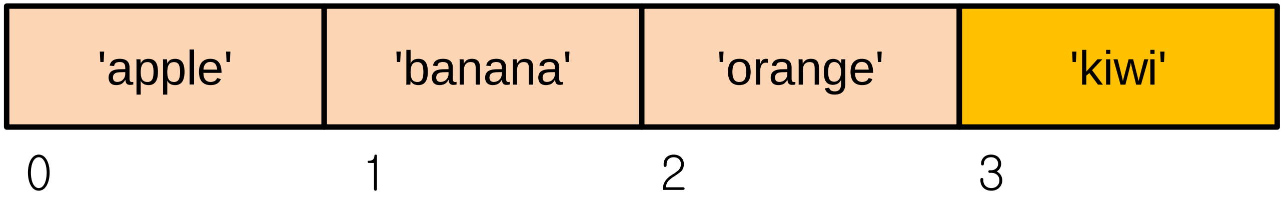
```
fruits.append('kiwi')
fruits
```

```
['apple', 'banana', 'orange', 'kiwi']
```

```
fruits[0]
```

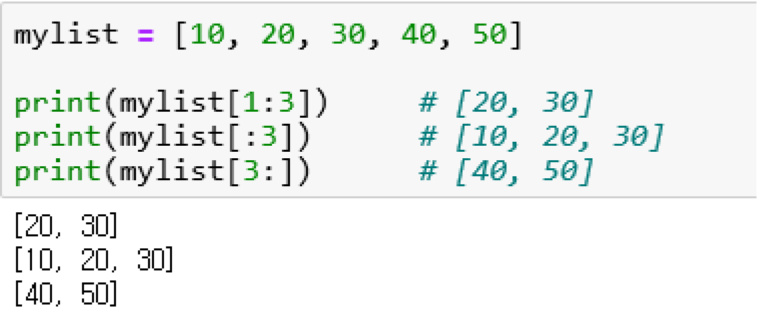
```
'apple'
```

list.append()



List

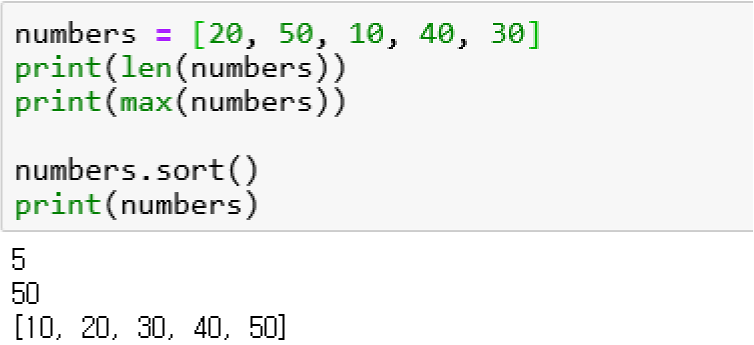
- Index of list
  - n:m - it means from ‘n’ to ‘m-1’



```
mylist = [10, 20, 30, 40, 50]

print(mylist[1:3])      # [20, 30]
print(mylist[:3])       # [10, 20, 30]
print(mylist[3:])       # [40, 50]
```

```
[20, 30]
[10, 20, 30]
[40, 50]
```

- Functions with the list as a parameter and methods of the list

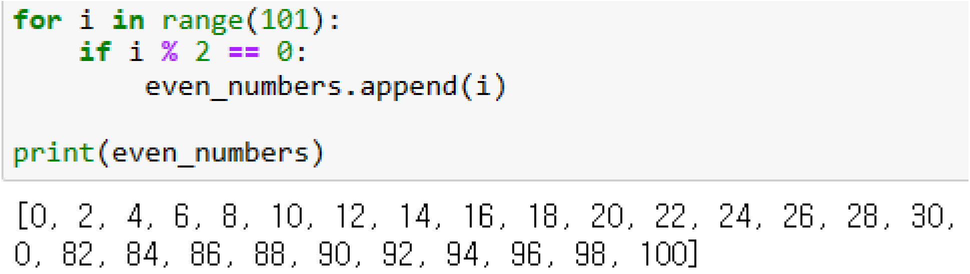


```
numbers = [20, 50, 10, 40, 30]
print(len(numbers))
print(max(numbers))

numbers.sort()
print(numbers)
```

```
5
50
[10, 20, 30, 40, 50]
```

List

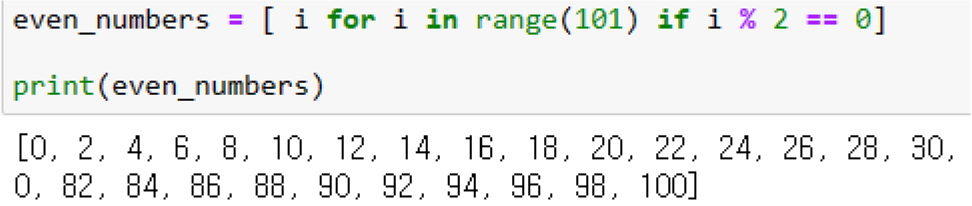
- List comprehension
  - make a list using for



```
for i in range(101):
    if i % 2 == 0:
        even_numbers.append(i)

print(even_numbers)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
0, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]
```

  - Shorter syntax for creating a new list based on the values of an existing list

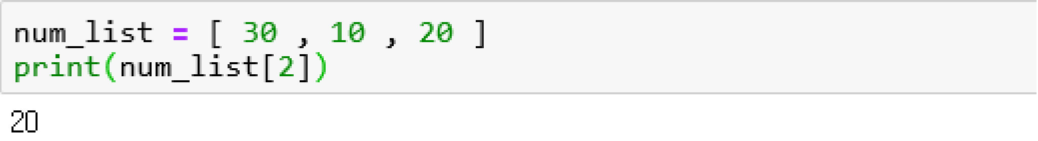


```
even_numbers = [ i for i in range(101) if i % 2 == 0]

print(even_numbers)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
0, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]
```

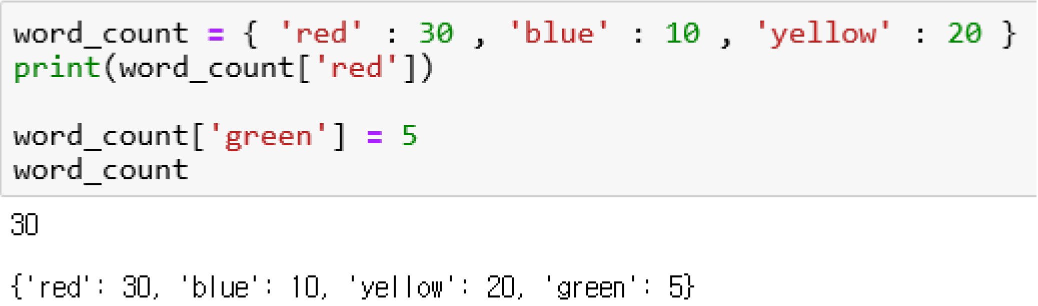
Dictionary

- A dictionary is a data structure that stores unordered <key : value> pairs
  - It can be initialized as { key : value , key : value , … } or
  - Make empty dictionary using 'dict()', and then insert the new (key, value) pair
  - List : items are indexed by location(integer)
  - Dictionary : values are indexed by keys



```
num_list = [ 30 , 10 , 20 ]
print(num_list[2])
```

```
20
```

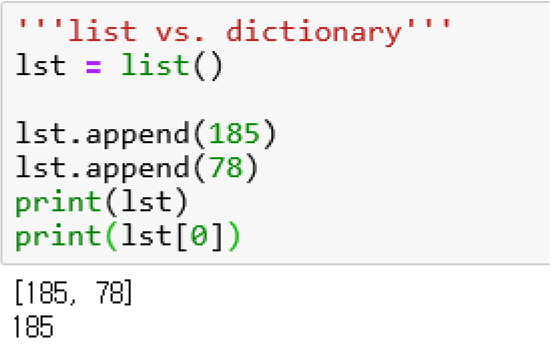


```
word_count = { 'red' : 30 , 'blue' : 10 , 'yellow' : 20 }
print(word_count['red'])

word_count['green'] = 5
word_count
```

```
30

{'red': 30, 'blue': 10, 'yellow': 20, 'green': 5}
```

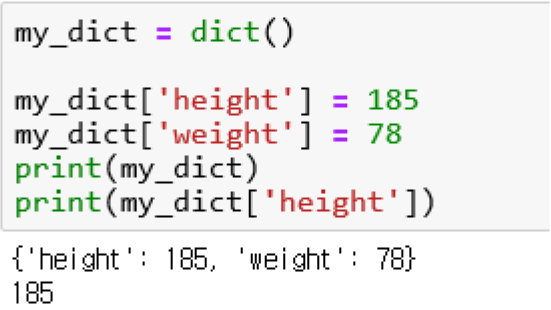
Dictionary

- list vs. dictionary



```
'''list vs. dictionary'''
lst = list()

lst.append(185)
lst.append(78)
print(lst)
print(lst[0])
```

```
[185, 78]
185
```



```
my_dict = dict()

my_dict['height'] = 185
my_dict['weight'] = 78
print(my_dict)
print(my_dict['height'])
```

```
{'height': 185, 'weight': 78}
185
```

List

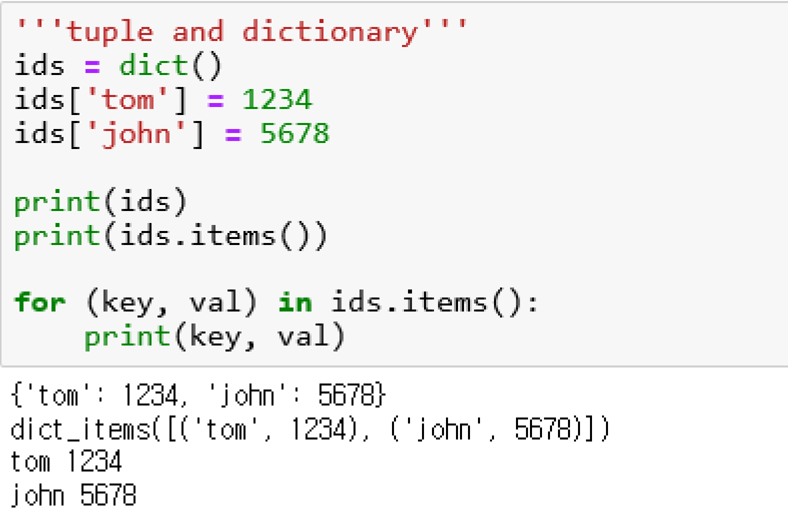
index	item
0	185
1	78

Dictionary

key	value
‘height’	185
‘weight’	78

Tuple

- A tuple is a collection of items which is ordered and **unchangeable**.
  - It is more simple and efficient because the tuple is saved with the condition not to be changed in the future
  - 'items()' method of dictionary returns list of (key, value) tuples

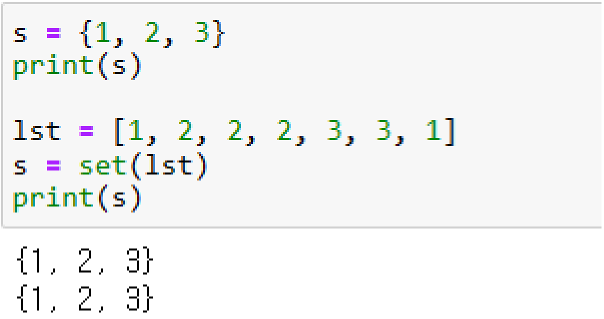


```python
'''tuple and dictionary'''
ids = dict()
ids['tom'] = 1234
ids['john'] = 5678

print(ids)
print(ids.items())

for (key, val) in ids.items():
    print(key, val)
```

```
{'tom': 1234, 'john': 5678}
dict_items([('tom', 1234), ('john', 5678)])
tom 1234
john 5678
```

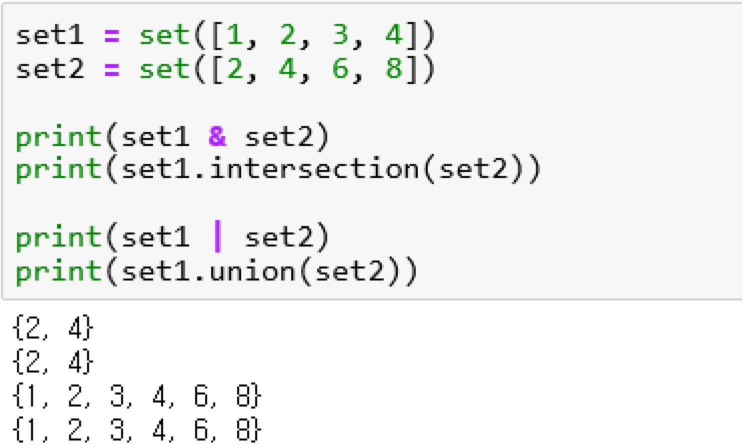
Set

- A set is an unordered collection of items with no duplicates
  - It can be initialized as { item1, item2, … } or
  - It can be defined by the function ‘set()’
  - Set operations : &, intersection(), |, union(), -, difference()



```
s = {1, 2, 3}
print(s)

lst = [1, 2, 2, 2, 3, 3, 1]
s = set(lst)
print(s)
```

```
{1, 2, 3}
{1, 2, 3}
```



```
set1 = set([1, 2, 3, 4])
set2 = set([2, 4, 6, 8])

print(set1 & set2)
print(set1.intersection(set2))

print(set1 | set2)
print(set1.union(set2))
```

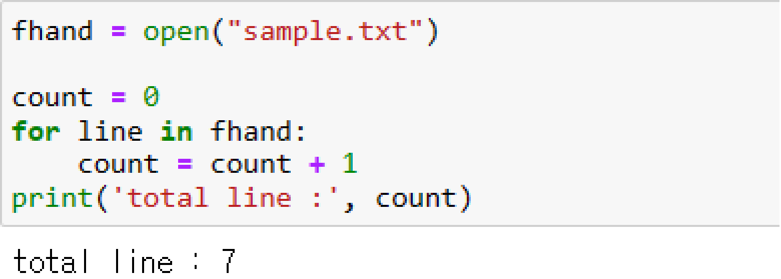
```
{2, 4}
{2, 4}
{1, 2, 3, 4, 6, 8}
{1, 2, 3, 4, 6, 8}
```

Container

- Container
  - Data structure for storing variables
- The four previously discussed data structures - lists, dict, tuple, and set - are called containers
  - list [,]
  - dict { , }
  - tuple (,)
  - set { , }
- When programming in python, it is important to understand the characteristics of the four containers and write the appropriate data structures for your situation

File processing

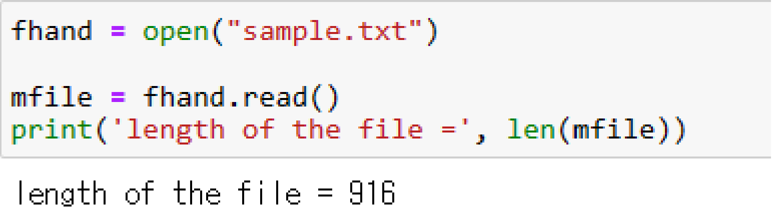
- Opening the file
  - open() – it returns the handle used to manipulate the file
  - The file handle is a sequence of each line of the file



```
fhand = open("sample.txt")

count = 0
for line in fhand:
    count = count + 1
print('total line :', count)
```

```
total line : 7
```

- Read all contents of the file
  - read() – it reads all contents in the file and returns it as a sequence of a string

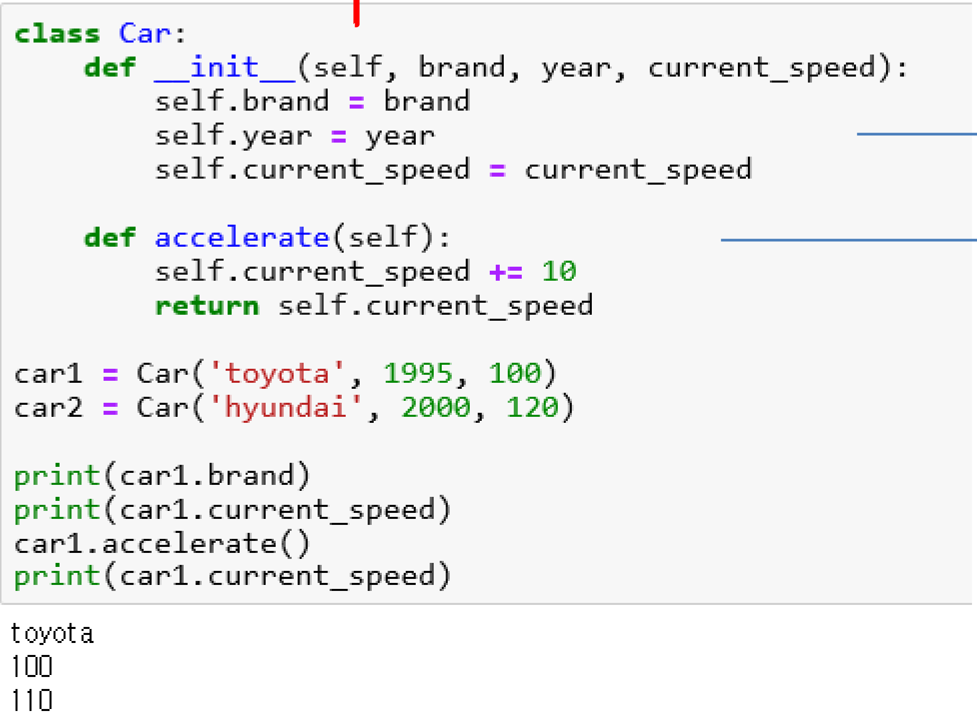


```
fhand = open("sample.txt")

mfile = fhand.read()
print('length of the file =', len(mfile))
```

```
length of the file = 916
```

Class

- Object = Attribute + Method
  - Classes provide a means of bundling data and functionality together
  - Creating a new class creates a new *type* of object, allowing new *instances* of that type to be made



```python
class Car:
    def __init__(self, brand, year, current_speed):
        self.brand = brand
        self.year = year
        self.current_speed = current_speed

    def accelerate(self):
        self.current_speed += 10
        return self.current_speed

car1 = Car('toyota', 1995, 100)
car2 = Car('hyundai', 2000, 120)

print(car1.brand)
print(car1.current_speed)
car1.accelerate()
print(car1.current_speed)
```

```
toyota
100
110
```

"self" represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class

brand, year, current_speed are attributes

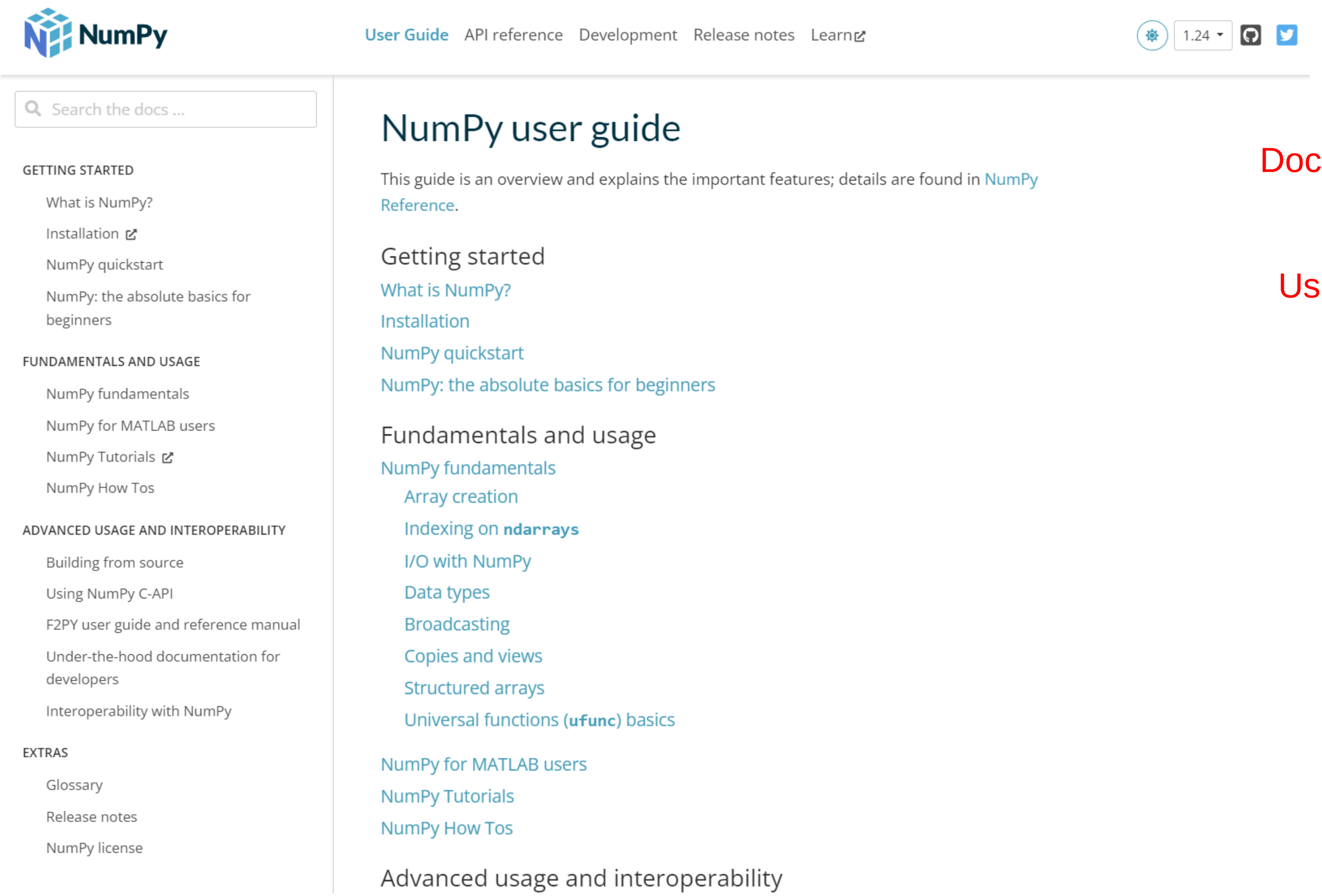
accelerate is a method

What is Numpy?

- Numpy
  - A library for scientific computing with Python
  - It provides easy and efficient operation of the multi-dimensional arrays
  - It provides useful linear algebra, Fourier transform, and random number capabilities
  - It is used with other python libraries to implement various statistical and numerical analysis techniques

https://numpy.org/

User Guide



NumPy

User Guide API reference Development Release notes Learn

1.24

Search the docs ...

GETTING STARTED

- What is NumPy?
- Installation
- NumPy quickstart
- NumPy: the absolute basics for beginners

FUNDAMENTALS AND USAGE

- NumPy fundamentals
- NumPy for MATLAB users
- NumPy Tutorials
- NumPy How Tos

ADVANCED USAGE AND INTEROPERABILITY

- Building from source
- Using NumPy C-API
- F2PY user guide and reference manual
- Under-the-hood documentation for developers
- Interoperability with NumPy

EXTRAS

- Glossary
- Release notes
- NumPy license

NumPy user guide

This guide is an overview and explains the important features; details are found in NumPy Reference.

Getting started

- What is NumPy?
- Installation
- NumPy quickstart
- NumPy: the absolute basics for beginners

Fundamentals and usage

- NumPy fundamentals
  - Array creation
  - Indexing on `ndarrays`
  - I/O with NumPy
  - Data types
  - Broadcasting
  - Copies and views
  - Structured arrays
  - Universal functions (`ufunc`) basics
- NumPy for MATLAB users
- NumPy Tutorials
- NumPy How Tos

Advanced usage and interoperability

Documentation

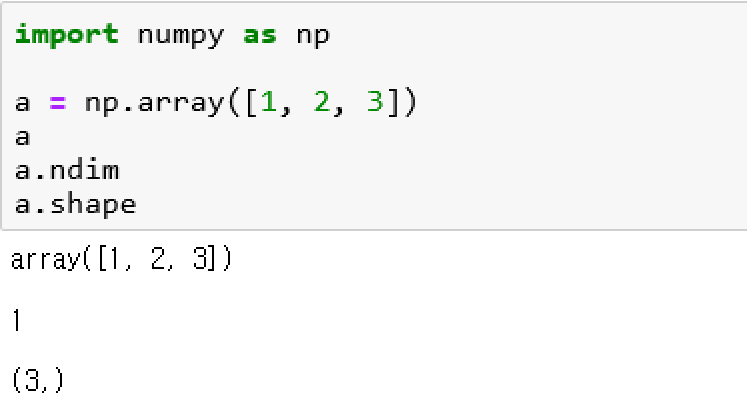
⬇

User Guide

https://numpy.org/doc/stable/user/

Array Creation

- array( ), ndim, shape



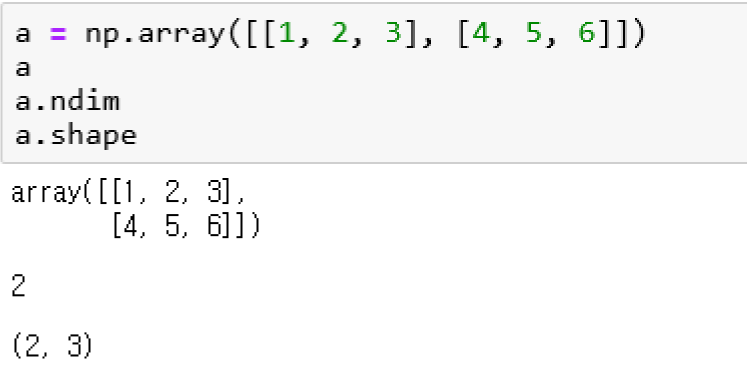
```
import numpy as np

a = np.array([1, 2, 3])
a
a.ndim
a.shape
```

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

1

(3,)
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

1-dimensional array (1 axes)

Shape is 3 (tuple indicating the size in each dimension)



```
a = np.array([[1, 2, 3], [4, 5, 6]])
a
a.ndim
a.shape
```

```
array([[1, 2, 3],
       [4, 5, 6]])

2

(2, 3)
```

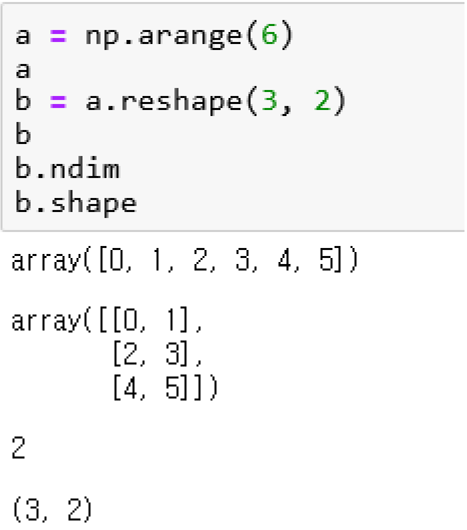
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

2-dimensional array

Shape is 2 x 3

Array Creation

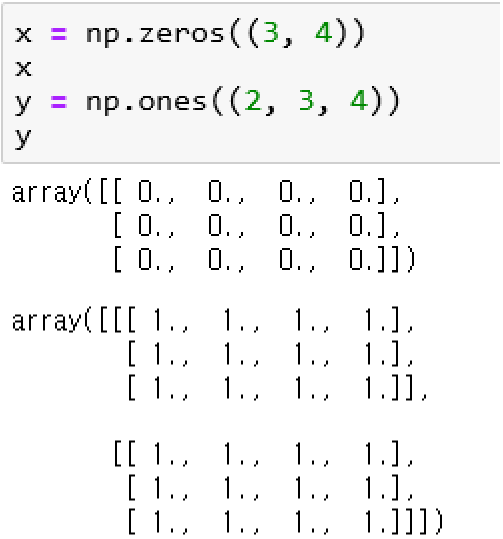
- arange( ), reshape( ), zeros( ), ones( )



```
a = np.arange(6)
a
b = a.reshape(3, 2)
b
b.ndim
b.shape
```

```
array([0, 1, 2, 3, 4, 5])

array([[0, 1],
       [2, 3],
       [4, 5]])

2

(3, 2)
```

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}$$

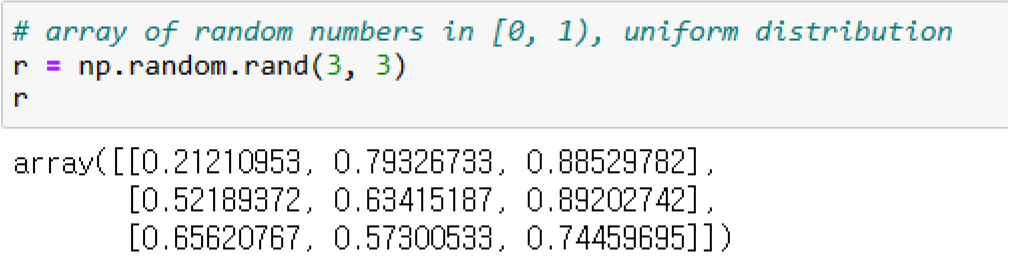


```
x = np.zeros((3, 4))
x
y = np.ones((2, 3, 4))
y
```

```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])

array([[[ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.]],

       [[ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.]]])
```

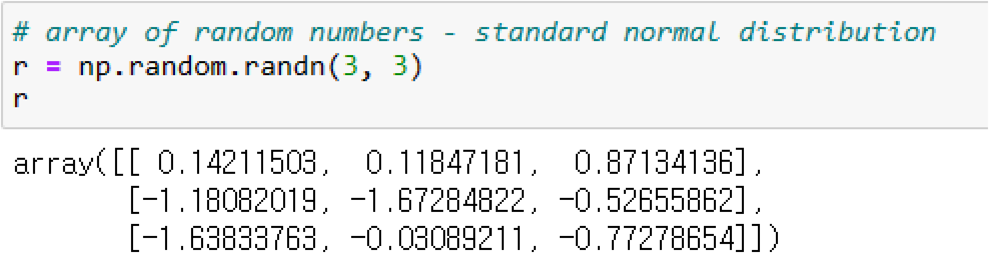
Array Creation

- random.rand( ), random.randn ()



```
# array of random numbers in [0, 1), uniform distribution
r = np.random.rand(3, 3)
r
```

```
array([[0.21210953, 0.79326733, 0.88529782],
       [0.52189372, 0.63415187, 0.89202742],
       [0.65620767, 0.57300533, 0.74459695]])
```

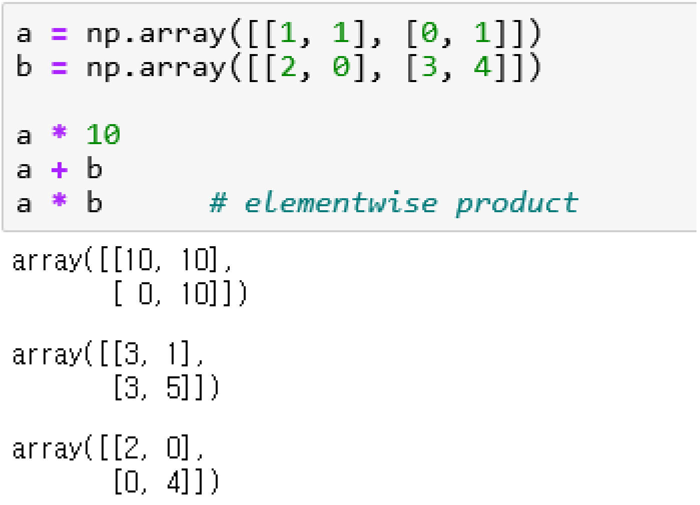


```
# array of random numbers - standard normal distribution
r = np.random.randn(3, 3)
r
```

```
array([[ 0.14211503,  0.11847181,  0.87134136],
       [-1.18082019, -1.67284822, -0.52655862],
       [-1.63833763, -0.03089211, -0.77278654]])
```

Array Operations

- +, -, *, dot(matrix multiplication), T(matrix transpose)



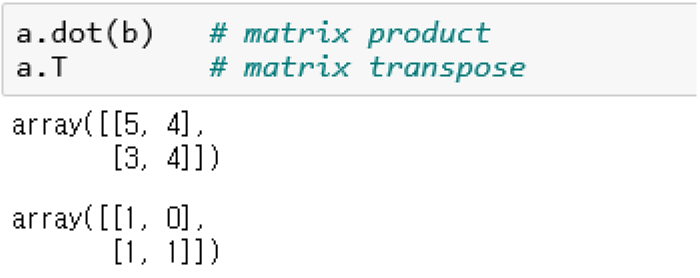
```
a = np.array([[1, 1], [0, 1]])
b = np.array([[2, 0], [3, 4]])

a * 10
a + b
a * b         # elementwise product
```

```
array([[10, 10],
       [ 0, 10]])

array([[3, 1],
       [3, 5]])

array([[2, 0],
       [0, 4]])
```

$$a = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 2 & 0 \\ 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 0 \\ 3 & 4 \end{bmatrix}$$



```
a.dot(b)      # matrix product
a.T           # matrix transpose
```

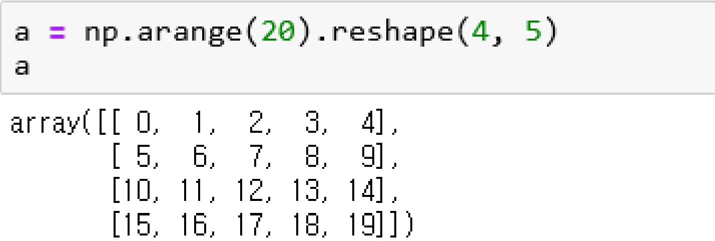
```
array([[5, 4],
       [3, 4]])

array([[1, 0],
       [1, 1]])
```

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{T}$$

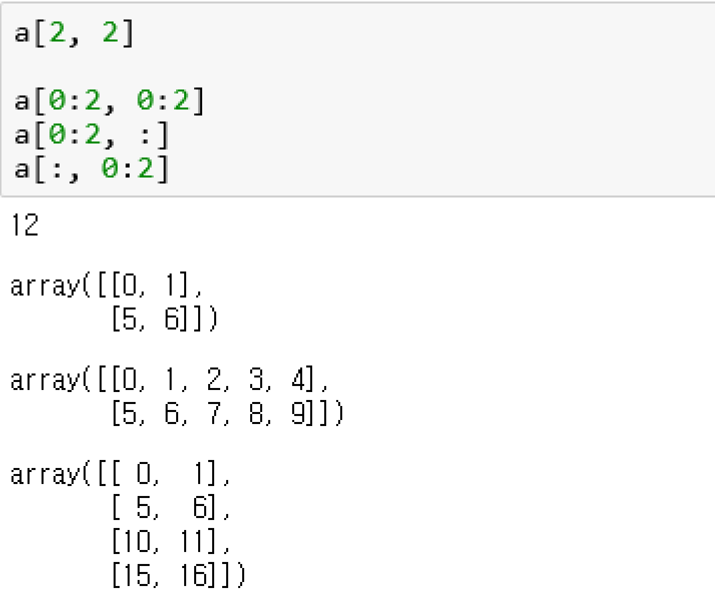
Indexing

- Indexing, slicing



```
a = np.arange(20).reshape(4, 5)
a
```

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

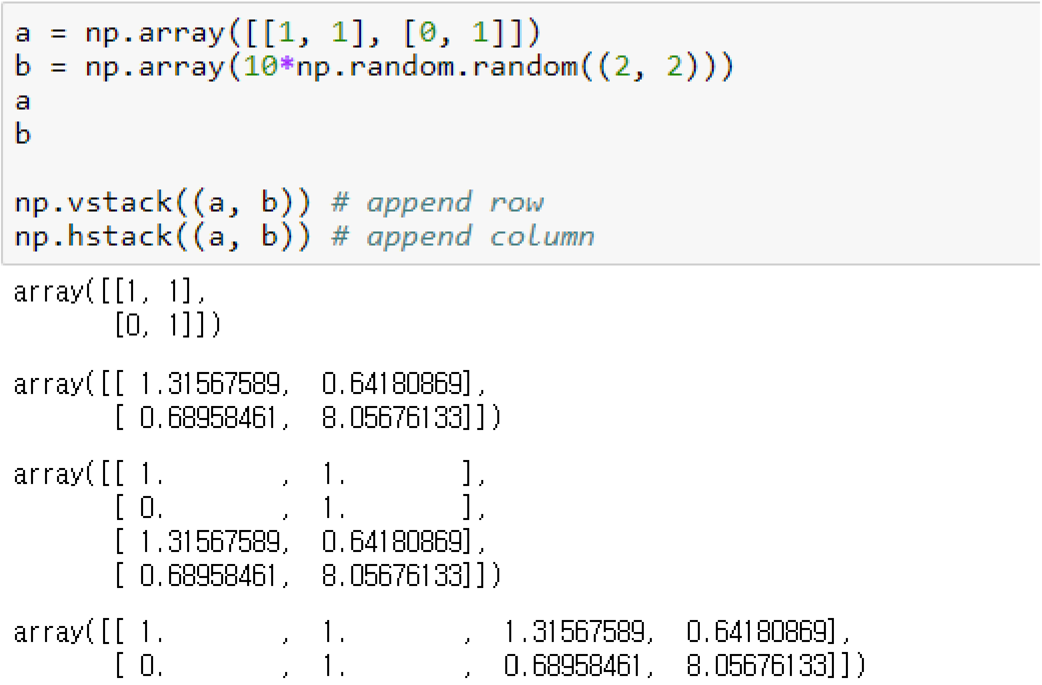


```
a[2, 2]

a[0:2, 0:2]
a[0:2, :]
a[:, 0:2]
```

```
12

array([[0, 1],
       [5, 6]])

array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])

array([[ 0,  1],
       [ 5,  6],
       [10, 11],
       [15, 16]])
```

Stacking

- vstack(), hstack()

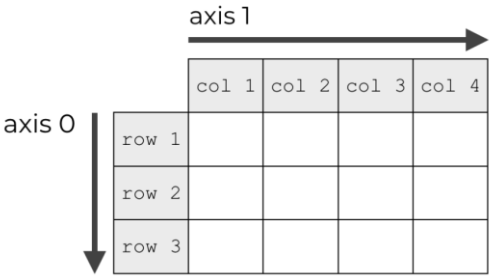


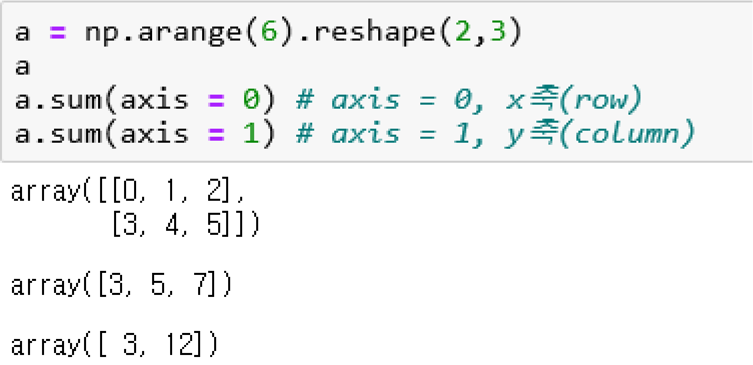
```
a = np.array([[1, 1], [0, 1]])
b = np.array(10*np.random.random((2, 2)))
a
b

np.vstack((a, b)) # append row
np.hstack((a, b)) # append column
```

```
array([[1, 1],
       [0, 1]])

array([[ 1.31567589,  0.64180869],
       [ 0.68958461,  8.05676133]])

array([[ 1.        ,  1.        ],
       [ 0.        ,  1.        ],
       [ 1.31567589,  0.64180869],
       [ 0.68958461,  8.05676133]])

array([[ 1.        ,  1.        ,  1.31567589,  0.64180869],
       [ 0.        ,  1.        ,  0.68958461,  8.05676133]])
```

Axis

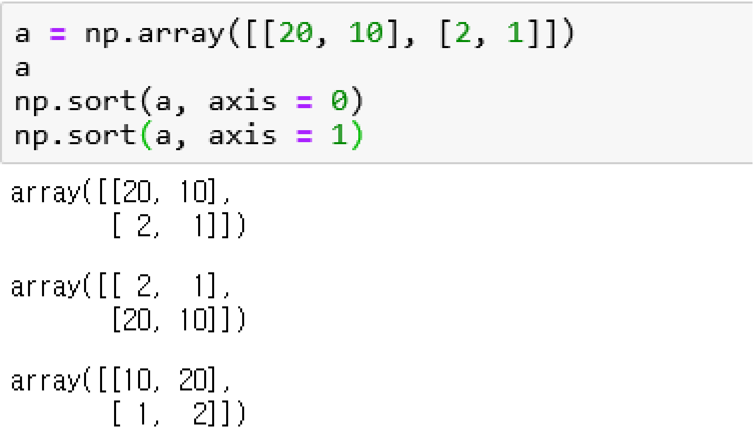
- For array of shape M x N,
  - axis = 0 : M rows
  - axis = 1 : N columns





```
a = np.arange(6).reshape(2,3)
a
a.sum(axis = 0) # axis = 0, x축(row)
a.sum(axis = 1) # axis = 1, y축(column)
```

```
array([[0, 1, 2],
       [3, 4, 5]])

array([3, 5, 7])

array([ 3, 12])
```

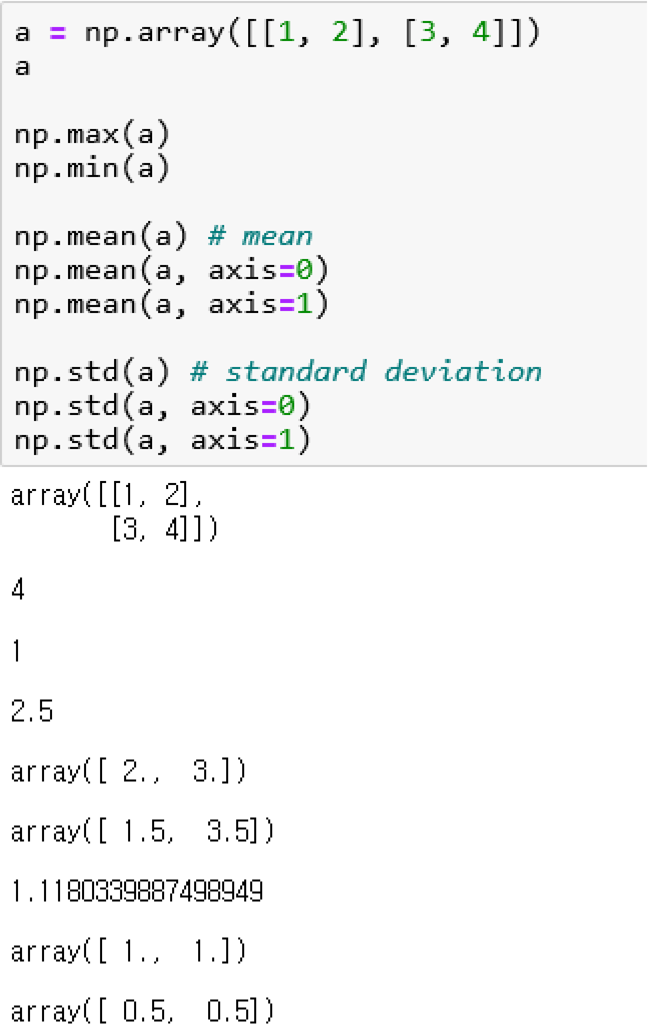


```
a = np.array([[20, 10], [2, 1]])
a
np.sort(a, axis = 0)
np.sort(a, axis = 1)
```

```
array([[20, 10],
       [ 2,  1]])

array([[ 2,  1],
       [20, 10]])

array([[10, 20],
       [ 1,  2]])
```

Basic Statistics

- max( ), min( )
- mean( ), std( ), var( ), …



```
a = np.array([[1, 2], [3, 4]])
a

np.max(a)
np.min(a)

np.mean(a) # mean
np.mean(a, axis=0)
np.mean(a, axis=1)

np.std(a) # standard deviation
np.std(a, axis=0)
np.std(a, axis=1)
```

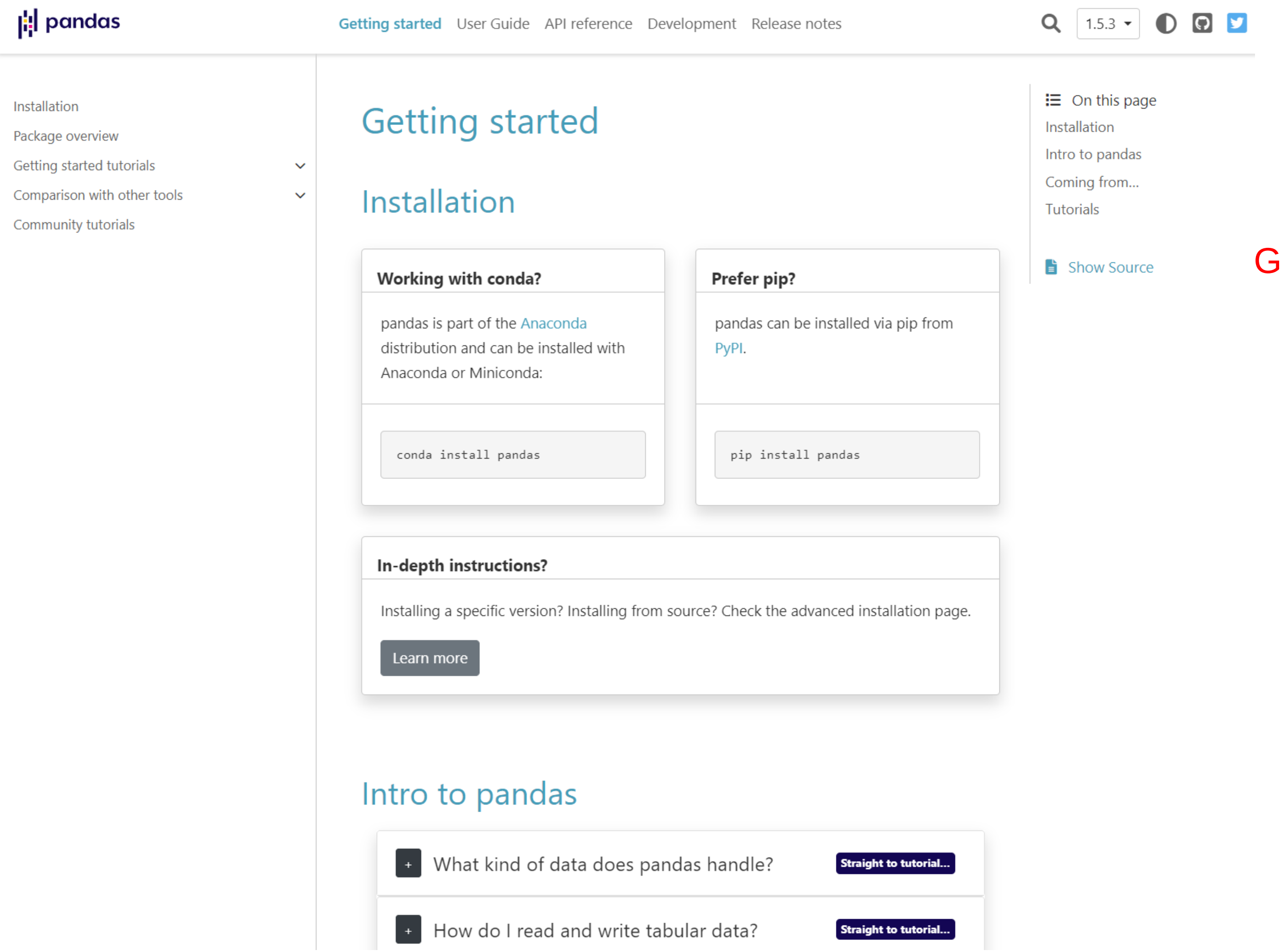
```
array([[1, 2],
       [3, 4]])

4

1

2.5

array([ 2.,  3.])

array([ 1.5,  3.5])

1.118033988749895

array([ 1.,  1.])

array([ 0.5,  0.5])
```

What is Pandas?

- Pandas
  - A Python package providing fast, flexible, and expressive data structures designed to make working with relational or labeled data easy and intuitive
  - Pandas is well suited for many different kinds of data:
    - Tabular data as in an SQL table or Excel spreadsheet
    - Time series data, matrix data, any other form of statistical data sets
  - The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases
  - It provides handling missing data, inserting and deleting data, data alignment, merging and joining data sets, etc.

https://pandas.pydata.org

Getting Started



Installation

Package overview

Getting started tutorials

Comparison with other tools

Community tutorials

Getting started

Installation

Working with conda?

pandas is part of the Anaconda distribution and can be installed with Anaconda or Miniconda:

```
conda install pandas
```

Prefer pip?

pandas can be installed via pip from PyPI.

```
pip install pandas
```

In-depth instructions?

Installing a specific version? Installing from source? Check the advanced installation page.

Intro to pandas

+ What kind of data does pandas handle? Straight to tutorial...

+ How do I read and write tabular data? Straight to tutorial...

On this page

Installation

Intro to pandas

Coming from...

Tutorials

Show Source

Getting started

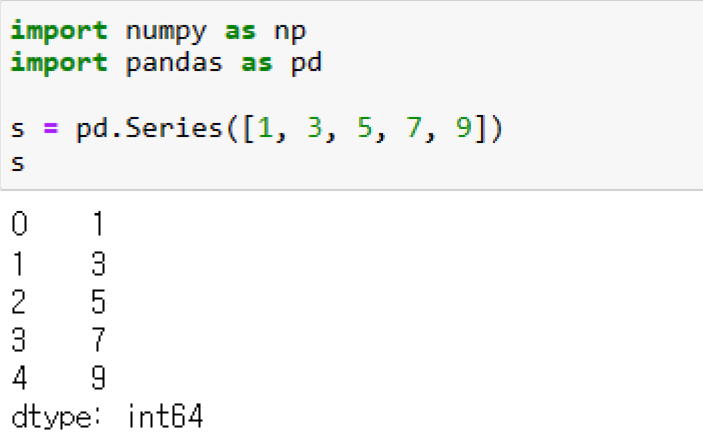
↓

Getting started tutorials

https://pandas.pydata.org/docs/getting_started

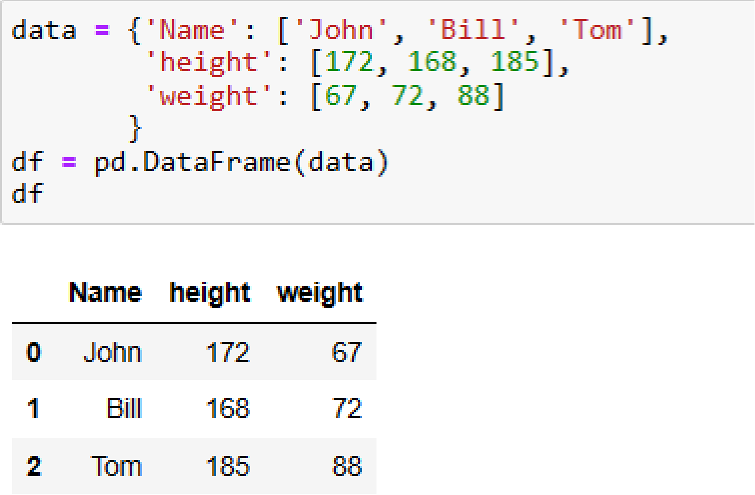
Object Creation

- Series, DataFrame



```
import numpy as np
import pandas as pd

s = pd.Series([1, 3, 5, 7, 9])
s
```

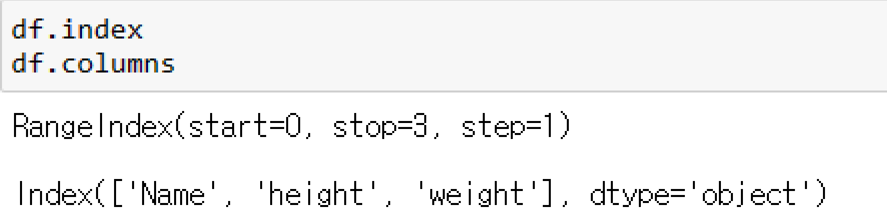
```
0    1
1    3
2    5
3    7
4    9
dtype: int64
```



```
data = {'Name': ['John', 'Bill', 'Tom'],
        'height': [172, 168, 185],
        'weight': [67, 72, 88]
       }
df = pd.DataFrame(data)
df
```

	Name	height	weight
0	John	172	67
1	Bill	168	72
2	Tom	185	88

- Index & column

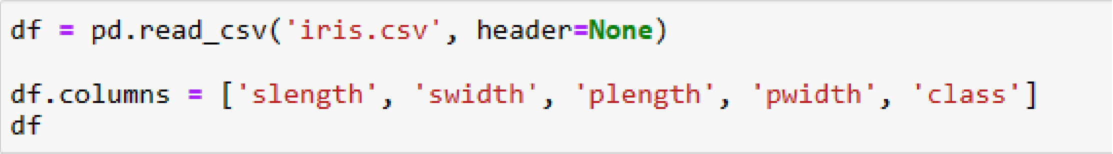


```
df.index
df.columns
```

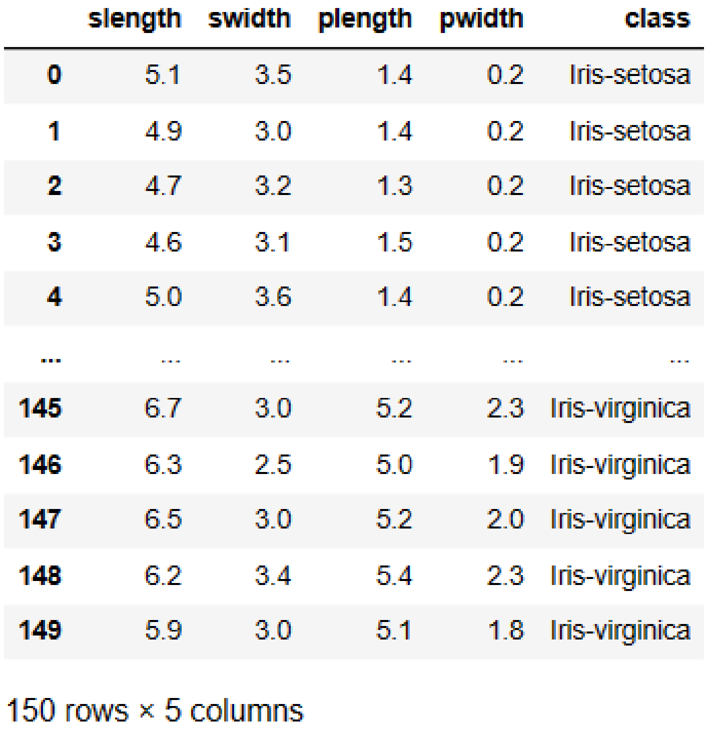
```
RangeIndex(start=0, stop=3, step=1)

Index(['Name', 'height', 'weight'], dtype='object')
```

Getting Data In

- read_csv()



```
df = pd.read_csv('iris.csv', header=None)

df.columns = ['slength', 'swidth', 'plength', 'pwidth', 'class']
df
```

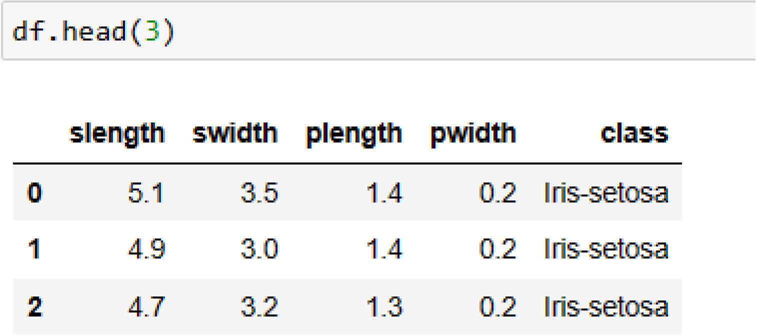


	slength	swidth	plength	pwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

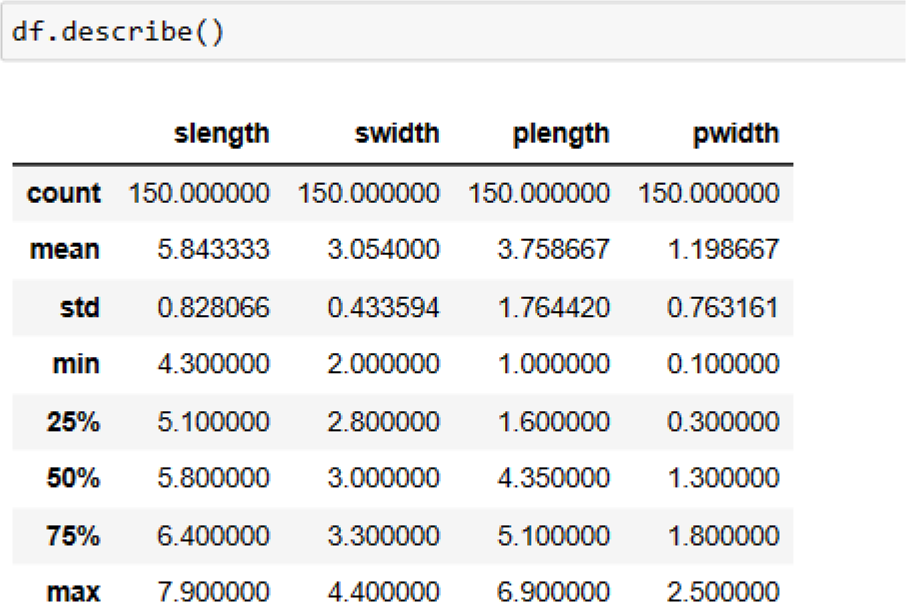
Viewing Data

- head(), describe()



```
df.head(3)
```

	slength	swidth	plength	pwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

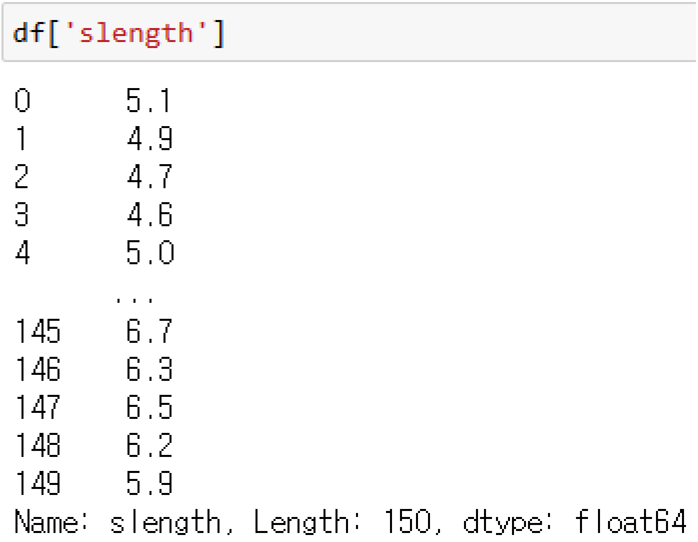


```
df.describe()
```

	slength	swidth	plength	pwidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

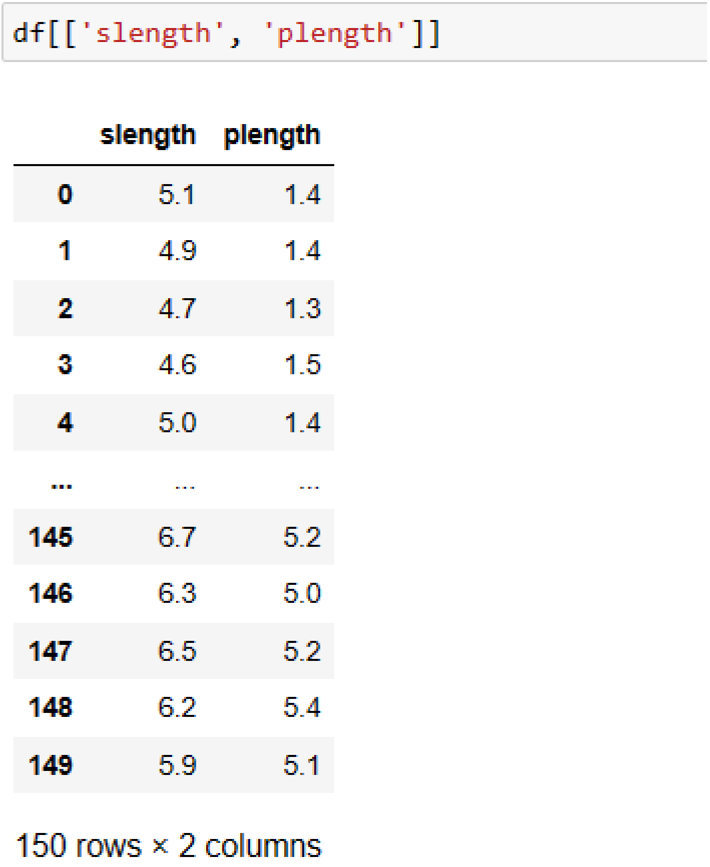
Selection

- Indexing by column names (labels)



```
df['slength']
```

```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: slength, Length: 150, dtype: float64
```



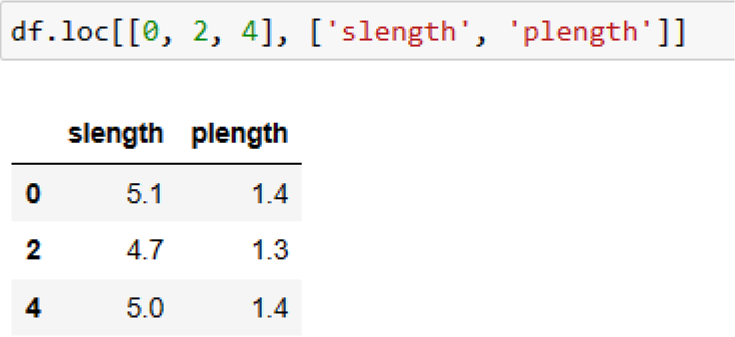
```
df[['slength', 'plength']]
```

	slength	plength
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4
...	...	...
145	6.7	5.2
146	6.3	5.0
147	6.5	5.2
148	6.2	5.4
149	5.9	5.1

150 rows × 2 columns

Selection

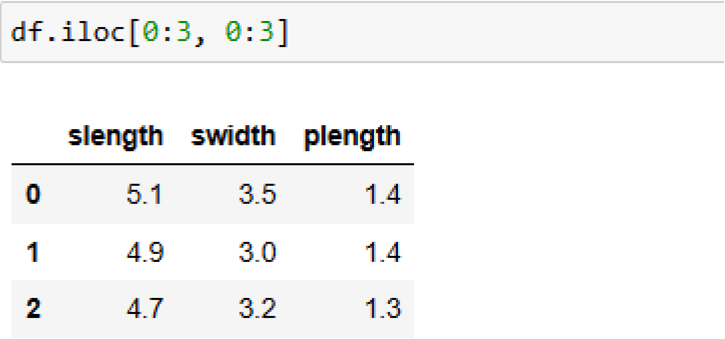
- Selection by labels - loc()



```
df.loc[[0, 2, 4], ['slength', 'plength']]
```

	slength	plength
0	5.1	1.4
2	4.7	1.3
4	5.0	1.4

- Selection by position - iloc()

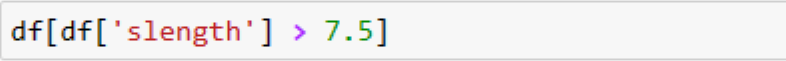


```
df.iloc[0:3, 0:3]
```

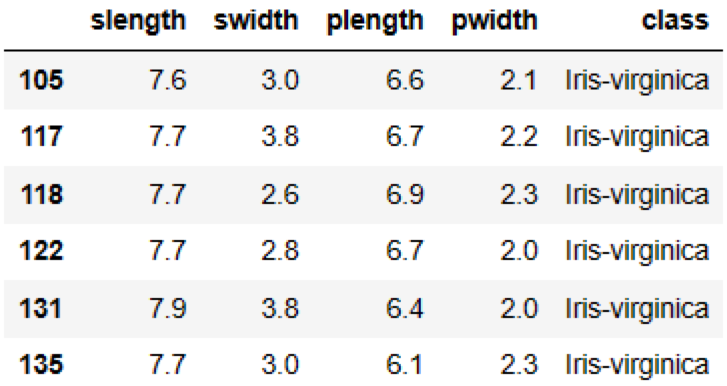
	slength	swidth	plength
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7	3.2	1.3

Selection

- Boolean indexing

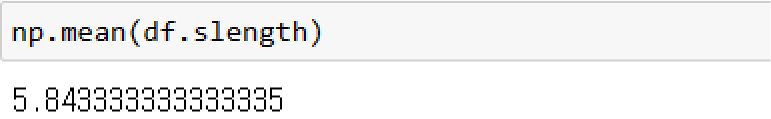


```
df[df['slength'] > 7.5]
```



	slength	swidth	plength	pwidth	class
105	7.6	3.0	6.6	2.1	Iris-virginica
117	7.7	3.8	6.7	2.2	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica

- Accessing columns as an attribute

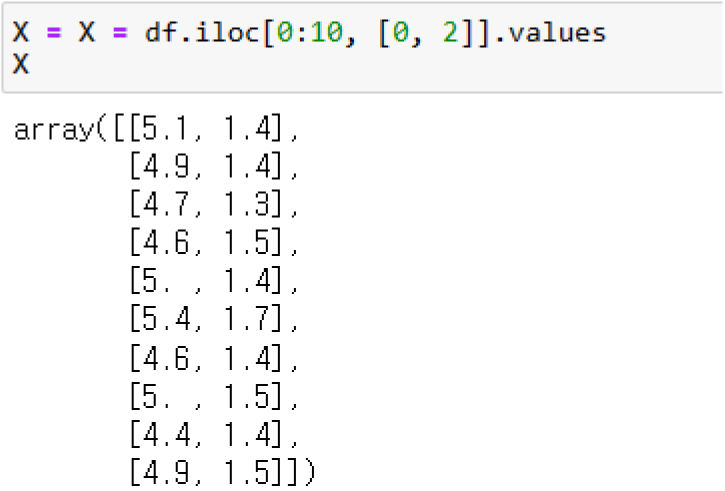


```
np.mean(df.slength)
```

```
5.843333333333335
```

Get Numpy Array

- values



```
X = X = df.iloc[0:10, [0, 2]].values
X
```

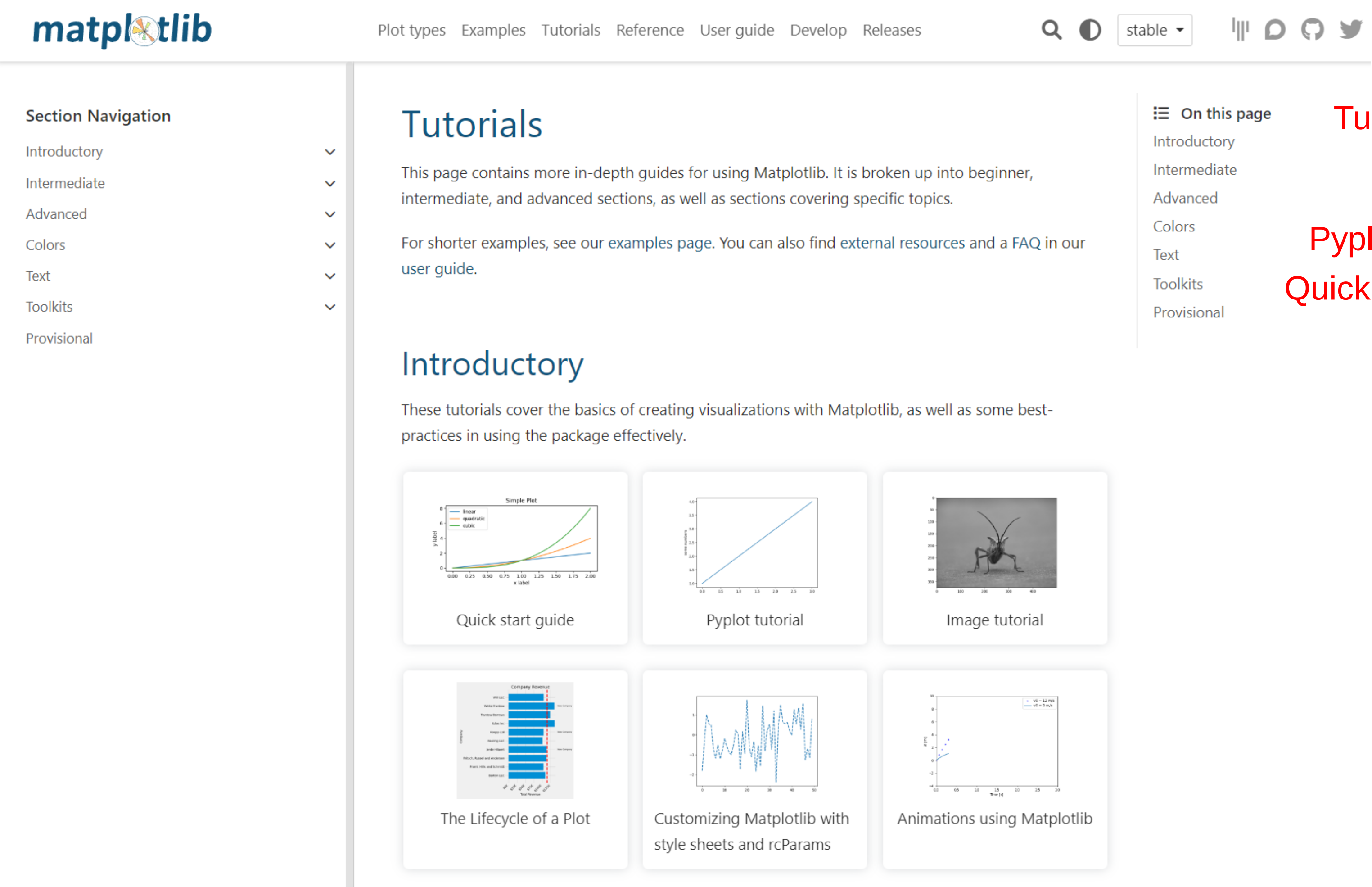
```
array([[5.1, 1.4],
       [4.9, 1.4],
       [4.7, 1.3],
       [4.6, 1.5],
       [5. , 1.4],
       [5.4, 1.7],
       [4.6, 1.4],
       [5. , 1.5],
       [4.4, 1.4],
       [4.9, 1.5]])
```

What is Matplotlib?

- Matplotlib
  - One of the python library to visualize the data
  - It provides object-oriented API that can embed various types of data
  - Usually it is used with Scipy, Numpy and Pandas to visualize the results of training in Scikit-learn and Tensorflow

https://matplotlib.org/

Tutorials



matplotlib
Plot types Examples Tutorials Reference User guide Develop Releases
stable

Section Navigation

- Introductory
- Intermediate
- Advanced
- Colors
- Text
- Toolkits
- Provisional

Tutorials

This page contains more in-depth guides for using Matplotlib. It is broken up into beginner, intermediate, and advanced sections, as well as sections covering specific topics.

For shorter examples, see our examples page. You can also find external resources and a FAQ in our user guide.

Introductory

These tutorials cover the basics of creating visualizations with Matplotlib, as well as some best-practices in using the package effectively.

- Quick start guide
- Pyplot tutorial
- Image tutorial
- The Lifecycle of a Plot
- Customizing Matplotlib with style sheets and rcParams
- Animations using Matplotlib

On this page

- Introductory
- Intermediate
- Advanced
- Colors
- Text
- Toolkits
- Provisional

Tutorials

↓

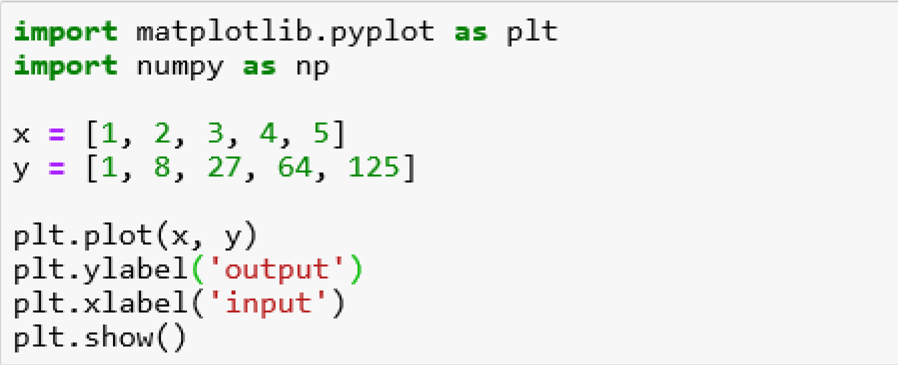
Pyplot tutorial

Quick start guide

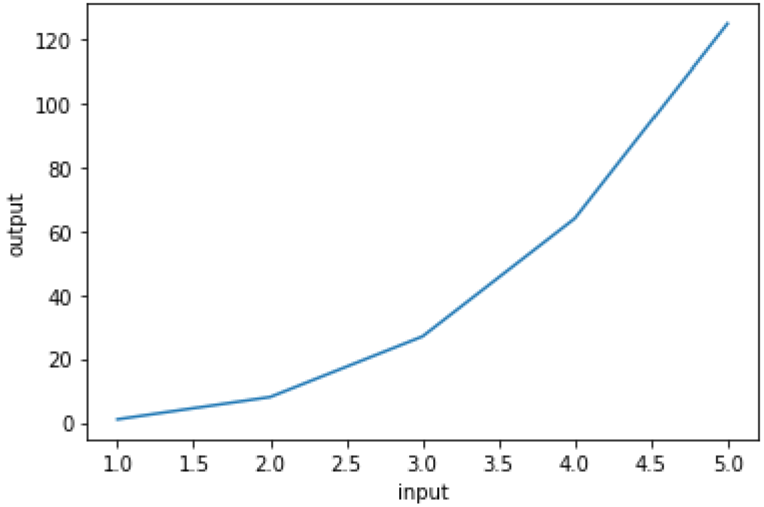
https://matplotlib.org/tutorials/

Plot

- plt.plot(x, y)

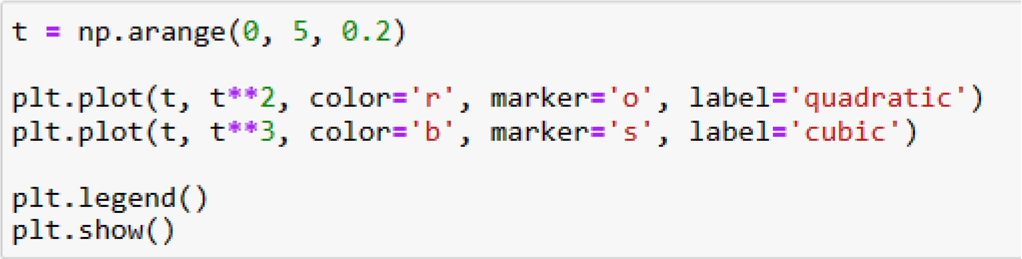


```
import matplotlib.pyplot as plt
import numpy as np

x = [1, 2, 3, 4, 5]
y = [1, 8, 27, 64, 125]

plt.plot(x, y)
plt.ylabel('output')
plt.xlabel('input')
plt.show()
```

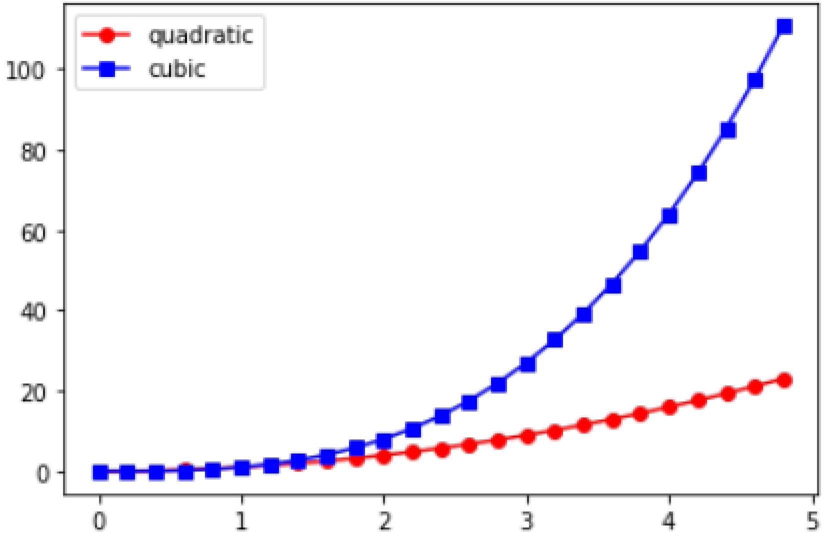


Plot

- Colors, shapes, labels
  - Colors(color=)
    - Red : r
    - Blue : b
    - Green : g
  - Marker Shapes(marker=)
    - Circle : o
    - Square : s
    - Triangle : ^
  - Labels(label=)
- plt.legend()

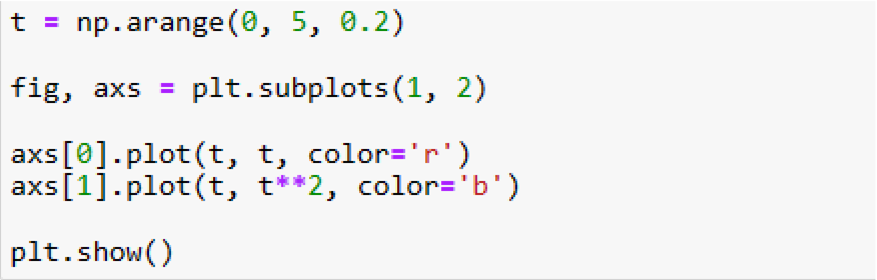


```
t = np.arange(0, 5, 0.2)

plt.plot(t, t**2, color='r', marker='o', label='quadratic')
plt.plot(t, t**3, color='b', marker='s', label='cubic')

plt.legend()
plt.show()
```

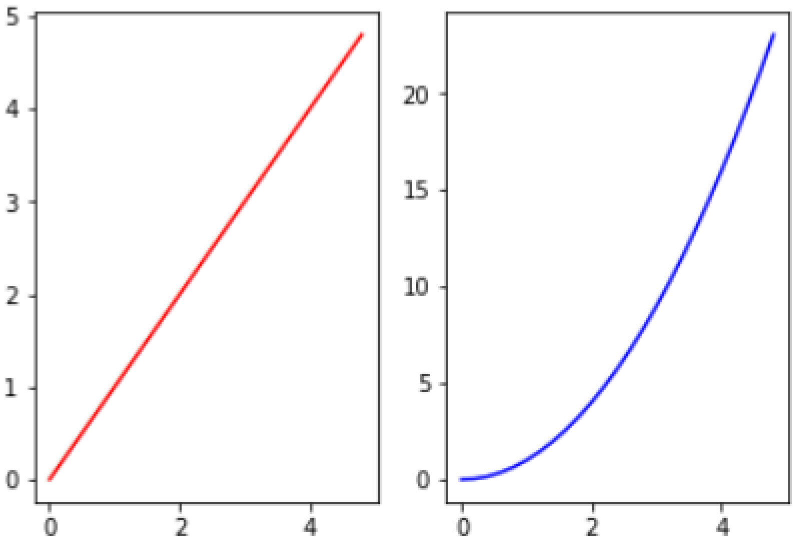


Subplots

- Explicit interface
  - Figure
    - Matplotlib graphs data on Figures
  - Axes
    - Figure contain one or more Axes
    - Axes is an area where points can be specified in terms of
      x-y coordinates

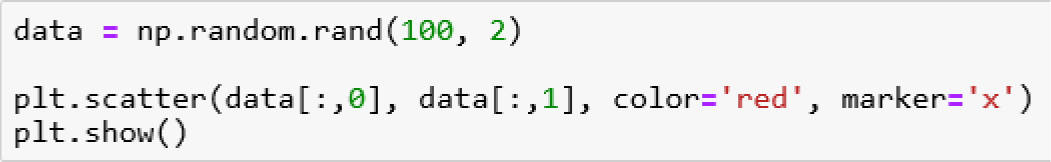


```
t = np.arange(0, 5, 0.2)

fig, axs = plt.subplots(1, 2)

axs[0].plot(t, t, color='r')
axs[1].plot(t, t**2, color='b')

plt.show()
```

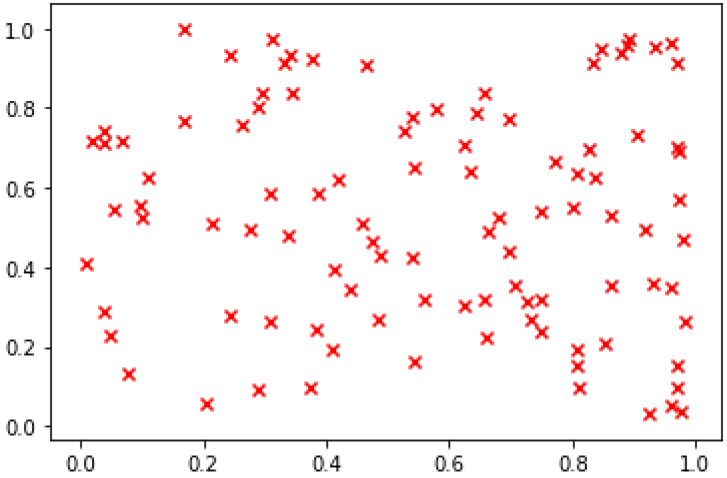


Scatter plot

- plt.scatter(x, y)

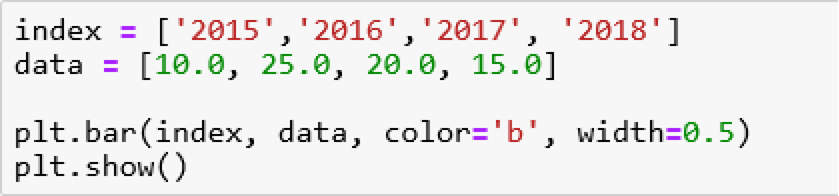


```
data = np.random.rand(100, 2)

plt.scatter(data[:,0], data[:,1], color='red', marker='x')
plt.show()
```

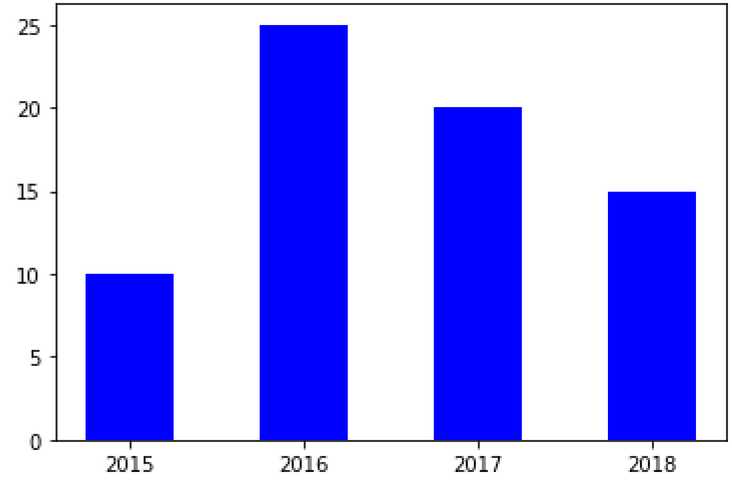


Bar plot

- plt.bar(x, y)

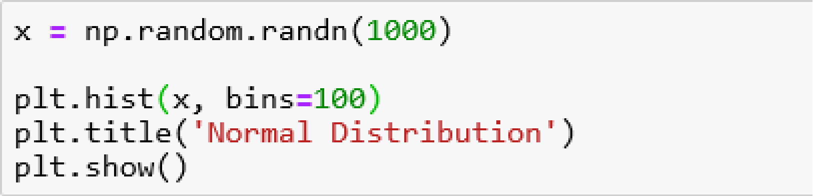


```
index = ['2015','2016','2017', '2018']
data = [10.0, 25.0, 20.0, 15.0]

plt.bar(index, data, color='b', width=0.5)
plt.show()
```

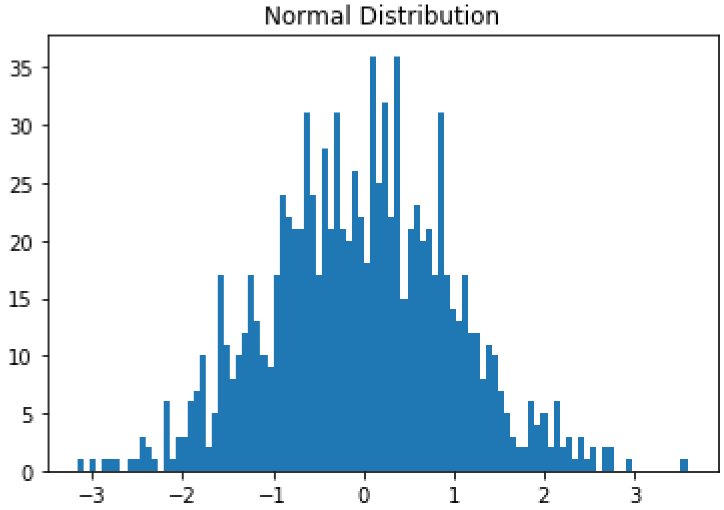


Histogram

- plt.hist(x)



```
x = np.random.randn(1000)

plt.hist(x, bins=100)
plt.title('Normal Distribution')
plt.show()
```

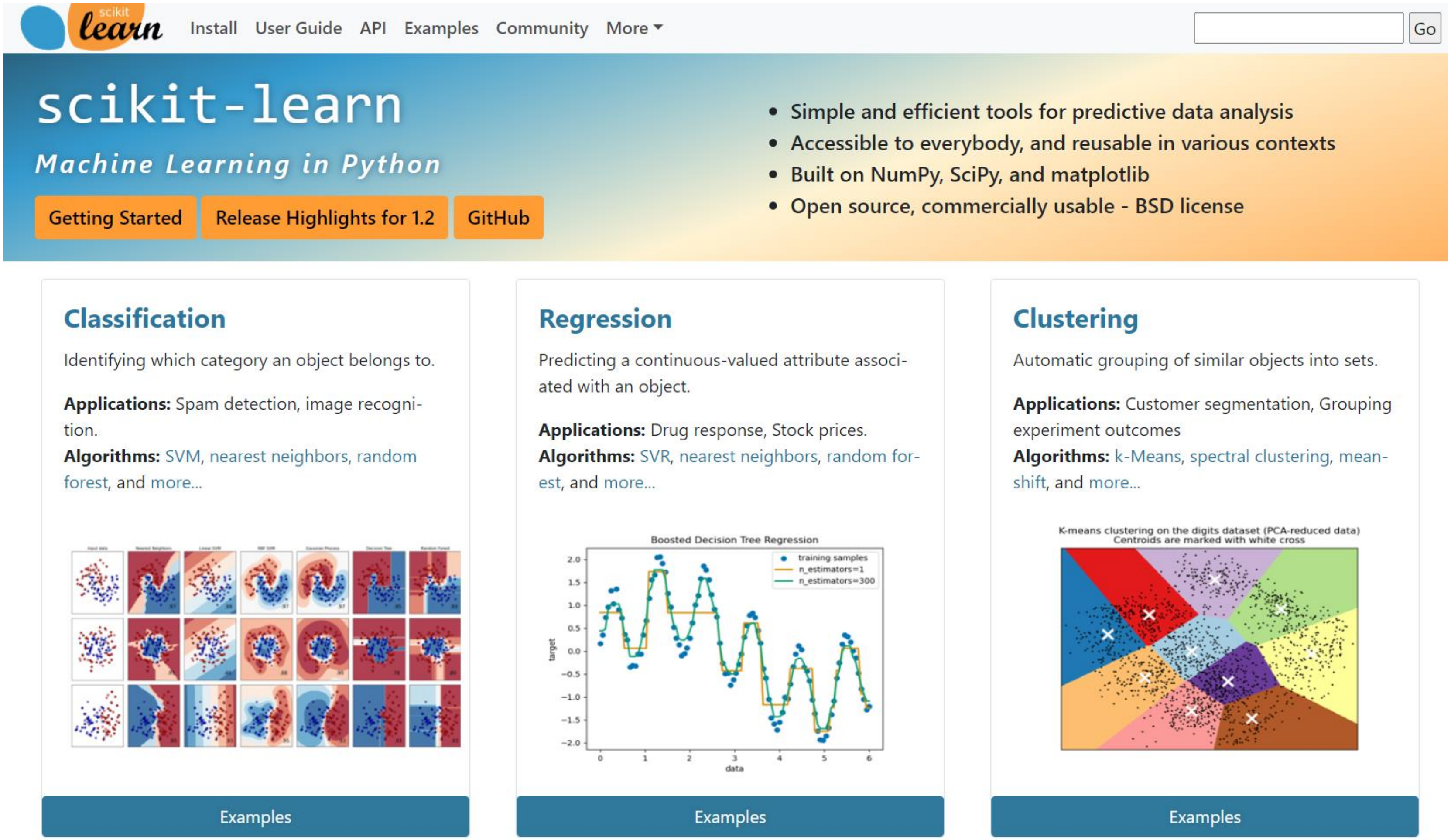


What is Scikit-learn?

- Scikit-learn
  - Open source machine learning library for the Python that supports supervised and unsupervised learning
  - It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries
  - It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

https://scikit-learn.org/stable/

scikit-learn.org



Install User Guide API Examples Community More ▾ Go

scikit-learn

Machine Learning in Python

Getting Started | Release Highlights for 1.2 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Examples

Clustering

Automatic grouping of similar objects into sets.

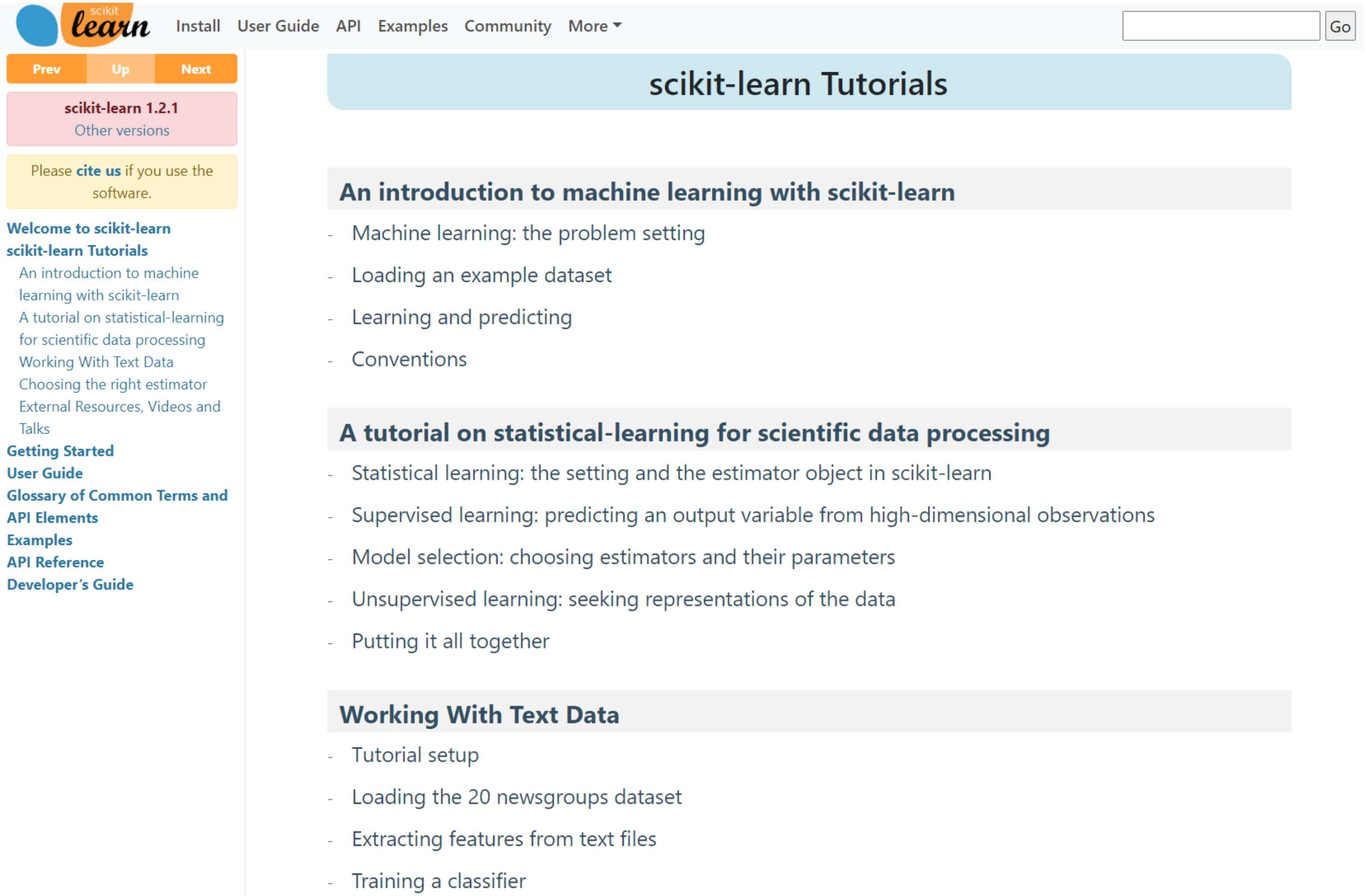
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

Examples

scikit
learn
Boosted Decision Tree Regression
training samples
n_estimators=1
n_estimators=300
target
data
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
1
2
3
4
5
6
K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross

https://scikit-learn.org/stable/

Tutorial



scikit-learn 1.2.1

Other versions

Please **cite us** if you use the software.

Welcome to scikit-learn

scikit-learn Tutorials

- An introduction to machine learning with scikit-learn
- A tutorial on statistical-learning for scientific data processing
- Working With Text Data
- Choosing the right estimator
- External Resources, Videos and Talks

Getting Started

User Guide

Glossary of Common Terms and

API Elements

Examples

API Reference

Developer's Guide

scikit-learn Tutorials

An introduction to machine learning with scikit-learn

- Machine learning: the problem setting
- Loading an example dataset
- Learning and predicting
- Conventions

A tutorial on statistical-learning for scientific data processing

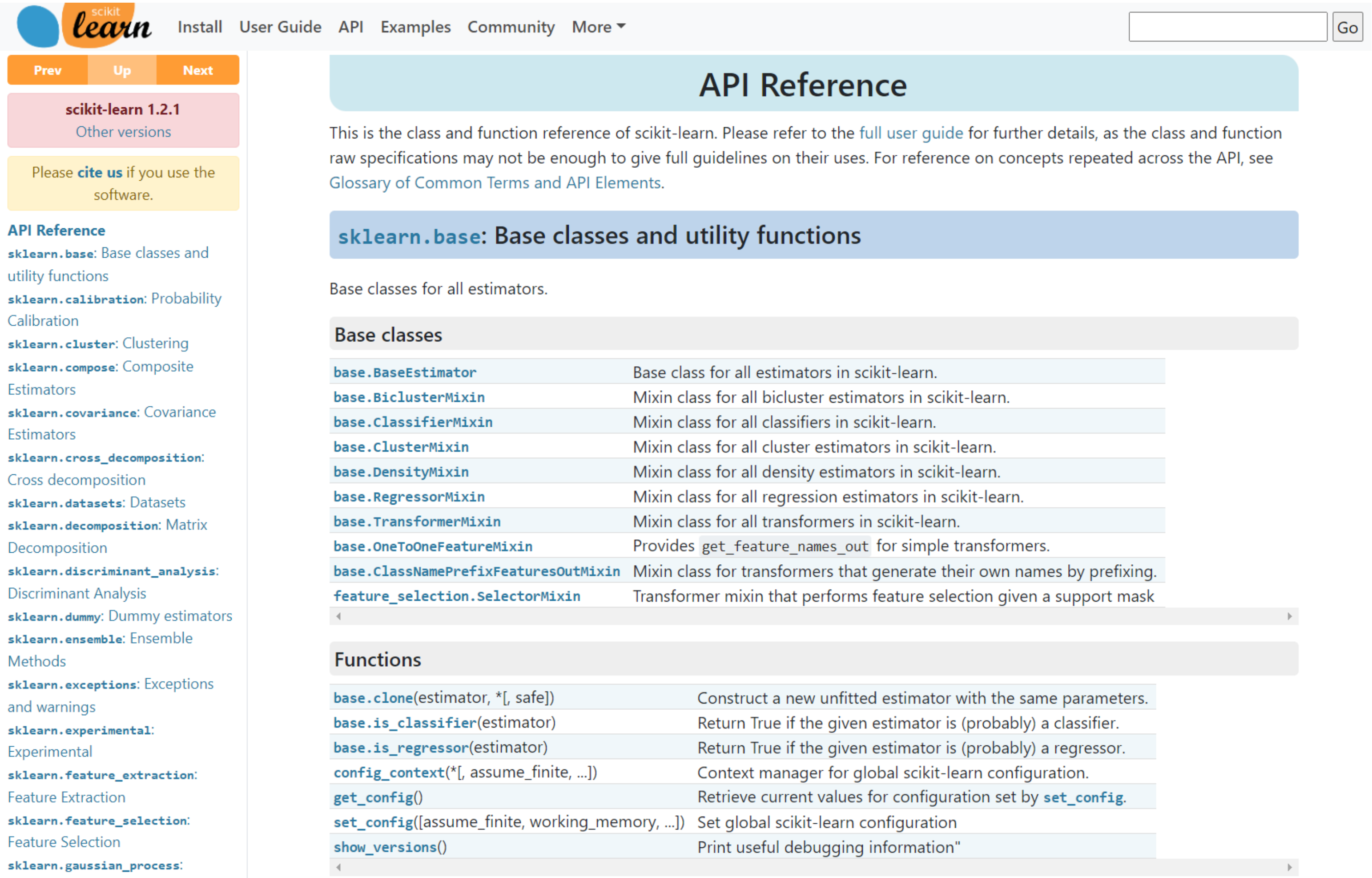
- Statistical learning: the setting and the estimator object in scikit-learn
- Supervised learning: predicting an output variable from high-dimensional observations
- Model selection: choosing estimators and their parameters
- Unsupervised learning: seeking representations of the data
- Putting it all together

Working With Text Data

- Tutorial setup
- Loading the 20 newsgroups dataset
- Extracting features from text files
- Training a classifier

https://scikit-learn.org/stable/tutorial/

API



scikit-learn 1.2.1
Other versions

Please **cite us** if you use the software.

API Reference
- `sklearn.base`: Base classes and utility functions
- `sklearn.calibration`: Probability Calibration
- `sklearn.cluster`: Clustering
- `sklearn.compose`: Composite Estimators
- `sklearn.covariance`: Covariance Estimators
- `sklearn.cross_decomposition`: Cross decomposition
- `sklearn.datasets`: Datasets
- `sklearn.decomposition`: Matrix Decomposition
- `sklearn.discriminant_analysis`: Discriminant Analysis
- `sklearn.dummy`: Dummy estimators
- `sklearn.ensemble`: Ensemble Methods
- `sklearn.exceptions`: Exceptions and warnings
- `sklearn.experimental`: Experimental
- `sklearn.feature_extraction`: Feature Extraction
- `sklearn.feature_selection`: Feature Selection
- `sklearn.gaussian_process`:

API Reference

This is the class and function reference of scikit-learn. Please refer to the full user guide for further details, as the class and function raw specifications may not be enough to give full guidelines on their uses. For reference on concepts repeated across the API, see Glossary of Common Terms and API Elements.

`sklearn.base`: Base classes and utility functions

Base classes for all estimators.

Base classes

`base.BaseEstimator`	Base class for all estimators in scikit-learn.
`base.BiclusterMixin`	Mixin class for all bicluster estimators in scikit-learn.
`base.ClassifierMixin`	Mixin class for all classifiers in scikit-learn.
`base.ClusterMixin`	Mixin class for all cluster estimators in scikit-learn.
`base.DensityMixin`	Mixin class for all density estimators in scikit-learn.
`base.RegressorMixin`	Mixin class for all regression estimators in scikit-learn.
`base.TransformerMixin`	Mixin class for all transformers in scikit-learn.
`base.OneToOneFeatureMixin`	Provides `get_feature_names_out` for simple transformers.
`base.ClassNamePrefixFeaturesOutMixin`	Mixin class for transformers that generate their own names by prefixing.
`feature_selection.SelectorMixin`	Transformer mixin that performs feature selection given a support mask

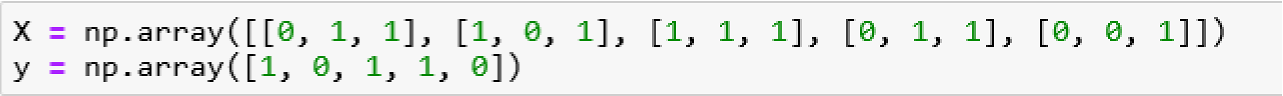
Functions

`base.clone`(estimator, *[, safe])	Construct a new unfitted estimator with the same parameters.
`base.is_classifier`(estimator)	Return True if the given estimator is (probably) a classifier.
`base.is_regressor`(estimator)	Return True if the given estimator is (probably) a regressor.
`config_context`(*[, assume_finite, ...])	Context manager for global scikit-learn configuration.
`get_config`()	Retrieve current values for configuration set by `set_config`.
`set_config`([assume_finite, working_memory, ...])	Set global scikit-learn configuration
`show_versions`()	Print useful debugging information"

https://scikit-learn.org/stable/modules/classes.html

Training a Model

- Task
  - Classify x = (x1, x2, x3) to y = 1 or 0
- Training dataset
  - 5 instances (example data) with known labels



```
X = np.array([[0, 1, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]])
y = np.array([1, 0, 1, 1, 0])
```

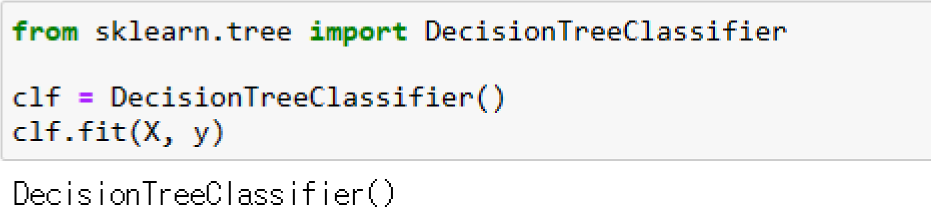
features label

5 instances

X			y
0,	1,	1	1
1,	0,	1	**0**
1,	1,	1	**1**
0,	1,	1	**1**
0,	0,	1	**0**

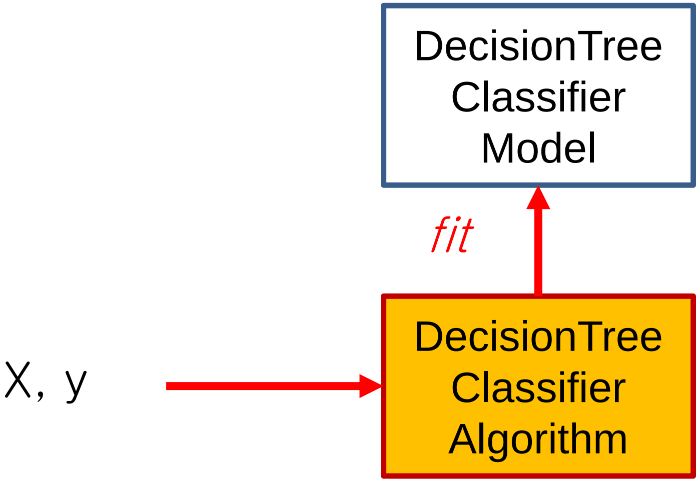
Training a Model

- Training (learning) - **fit**
  - Learning Decision Tree Classifier model with the training dataset



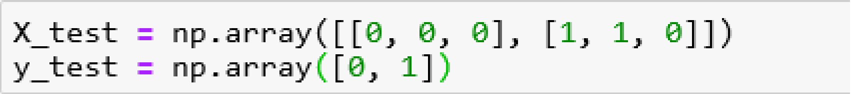
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X, y)
```

```
DecisionTreeClassifier()
```


DecisionTree Classifier Model
fit
X, y
DecisionTree Classifier Algorithm

Predicting using the Model

- Test dataset
  - 2 new instances



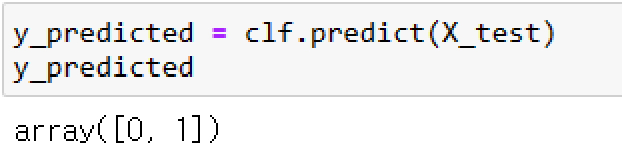
```
X_test = np.array([[0, 0, 0], [1, 1, 0]])
y_test = np.array([0, 1])
```

new instances

X			y
0,	0,	0	
1,	1,	0	

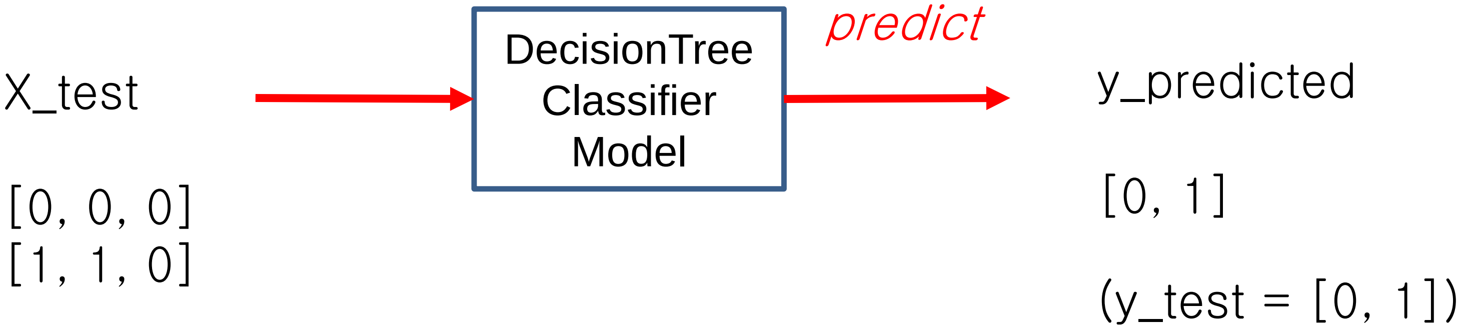
Predicting using the Model

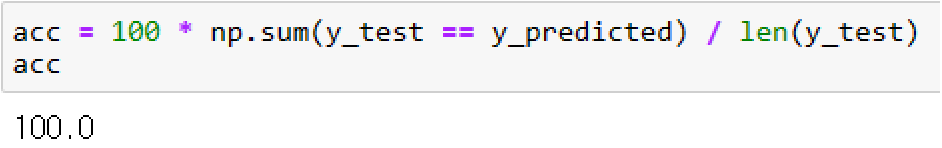
- Predicting labels – **predict**
  - Predict the label of new x using the learned model



```
y_predicted = clf.predict(X_test)
y_predicted
```

```
array([0, 1])
```





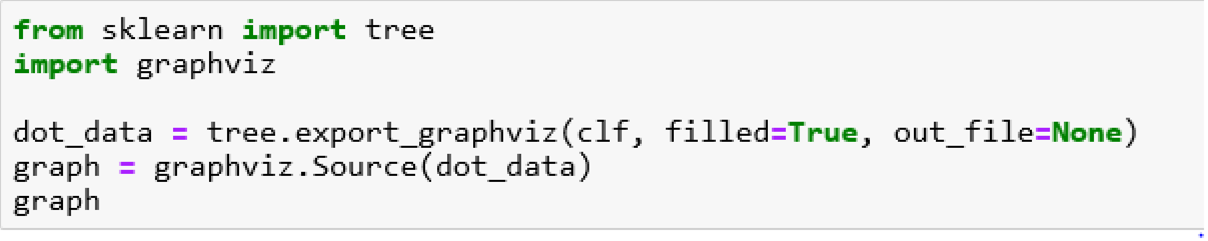
```
acc = 100 * np.sum(y_test == y_predicted) / len(y_test)
acc
```

```
100.0
```

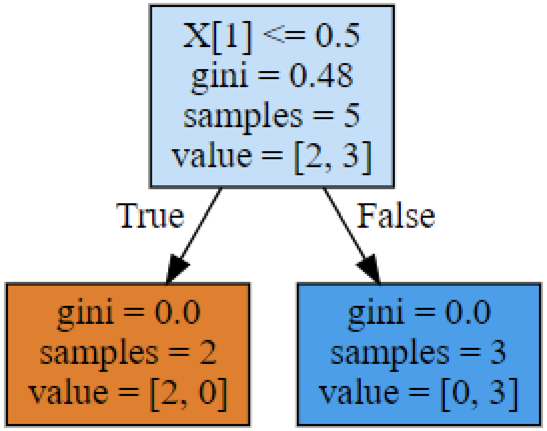
100 % accuracy

Visualize the Model

- Visualizing Decision Tree model using graphviz



```
from sklearn import tree
import graphviz

dot_data = tree.export_graphviz(clf, filled=True, out_file=None)
graph = graphviz.Source(dot_data)
graph
```

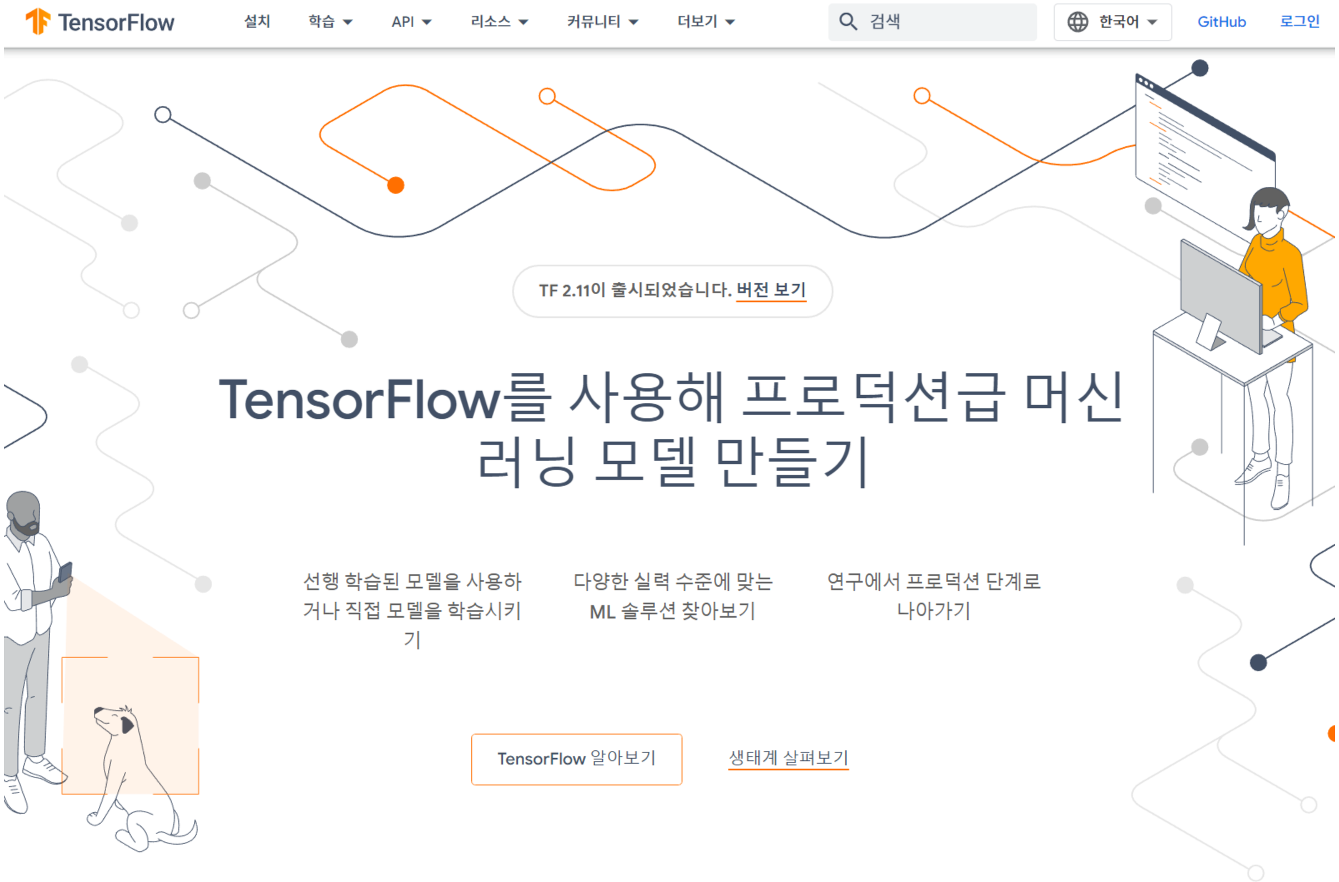


What is TensorFlow?

- TensorFlow
  - TensorFlow is an open source platform for machine learning
    - Originally developed by Google Brain team to conduct machine learning and deep neural networks research
  - It has a comprehensive ecosystem of tools, libraries and resources
  - TensorFlow computations are expressed as dataflow graphs on tensors
  - It can run on multiple CPUs and GPUs
- TensorFlow 2.0
  - Introduced a number of simplifications
  - Improvements to the performance on GPU

https://www.tensorflow.org/

tensorflow.org



TensorFlow 설치 학습 ▾ API ▾ 리소스 ▾ 커뮤니티 ▾ 더보기 ▾ 검색 한국어 ▾ GitHub 로그인

TF 2.11이 출시되었습니다. 버전 보기

TensorFlow를 사용해 프로덕션급 머신 러닝 모델 만들기

선행 학습된 모델을 사용하거나 직접 모델을 학습시키기

다양한 실력 수준에 맞는 ML 솔루션 찾아보기

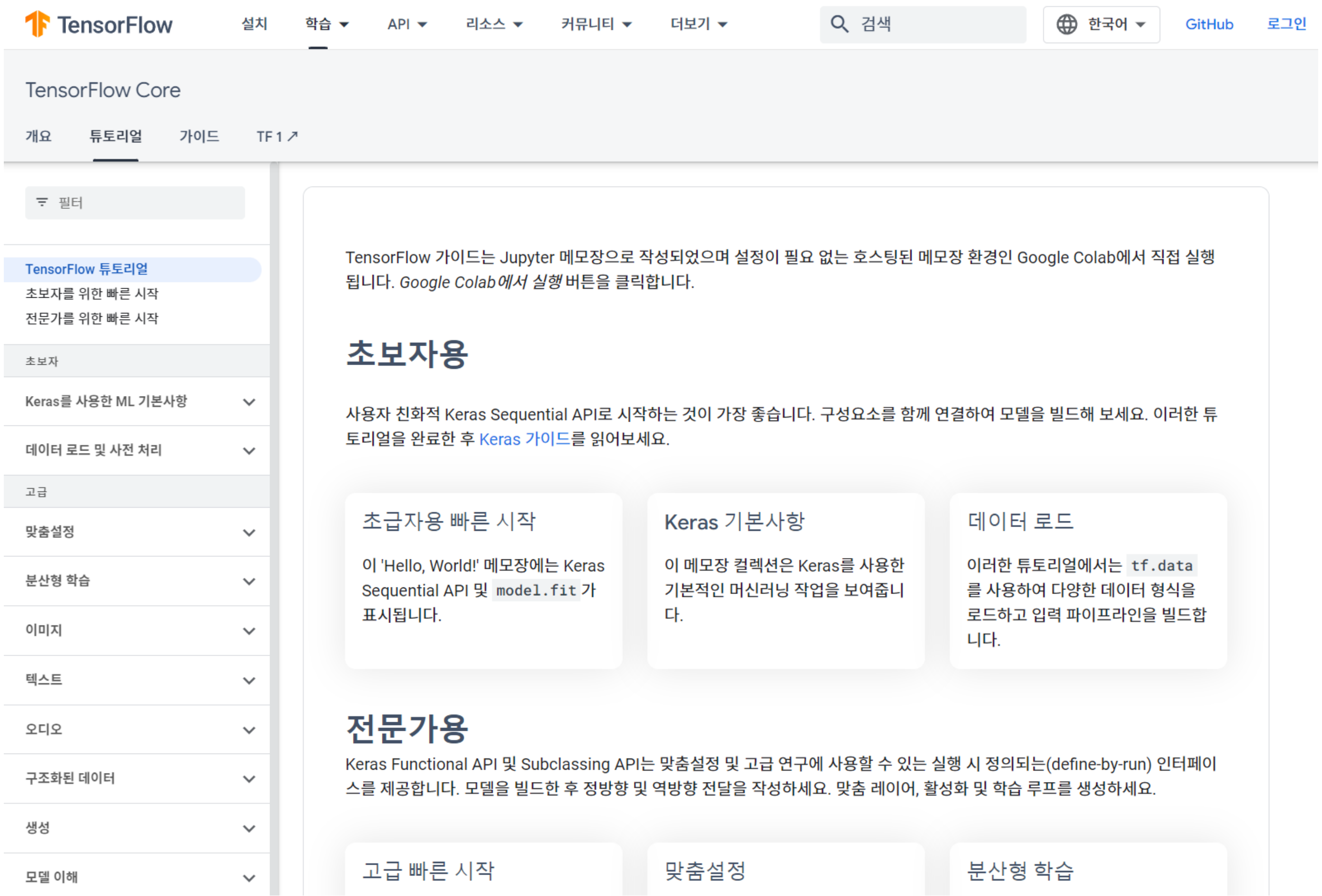
연구에서 프로덕션 단계로 나아가기

TensorFlow 알아보기 생태계 살펴보기

엔드 투 엔드 머신러닝 플랫폼

https://www.tensorflow.org/

Tutorial



TensorFlow 가이드는 Jupyter 메모장으로 작성되었으며 설정이 필요 없는 호스팅된 메모장 환경인 Google Colab에서 직접 실행됩니다. *Google Colab에서 실행* 버튼을 클릭합니다.

초보자용

사용자 친화적 Keras Sequential API로 시작하는 것이 가장 좋습니다. 구성요소를 함께 연결하여 모델을 빌드해 보세요. 이러한 튜토리얼을 완료한 후 Keras 가이드를 읽어보세요.

초급자용 빠른 시작

이 'Hello, World!' 메모장에는 Keras Sequential API 및 `model.fit` 가 표시됩니다.

Keras 기본사항

이 메모장 컬렉션은 Keras를 사용한 기본적인 머신러닝 작업을 보여줍니다.

데이터 로드

이러한 튜토리얼에서는 `tf.data` 를 사용하여 다양한 데이터 형식을 로드하고 입력 파이프라인을 빌드합니다.

전문가용

Keras Functional API 및 Subclassing API는 맞춤설정 및 고급 연구에 사용할 수 있는 실행 시 정의되는(define-by-run) 인터페이스를 제공합니다. 모델을 빌드한 후 정방향 및 역방향 전달을 작성하세요. 맞춤 레이어, 활성화 및 학습 루프를 생성하세요.

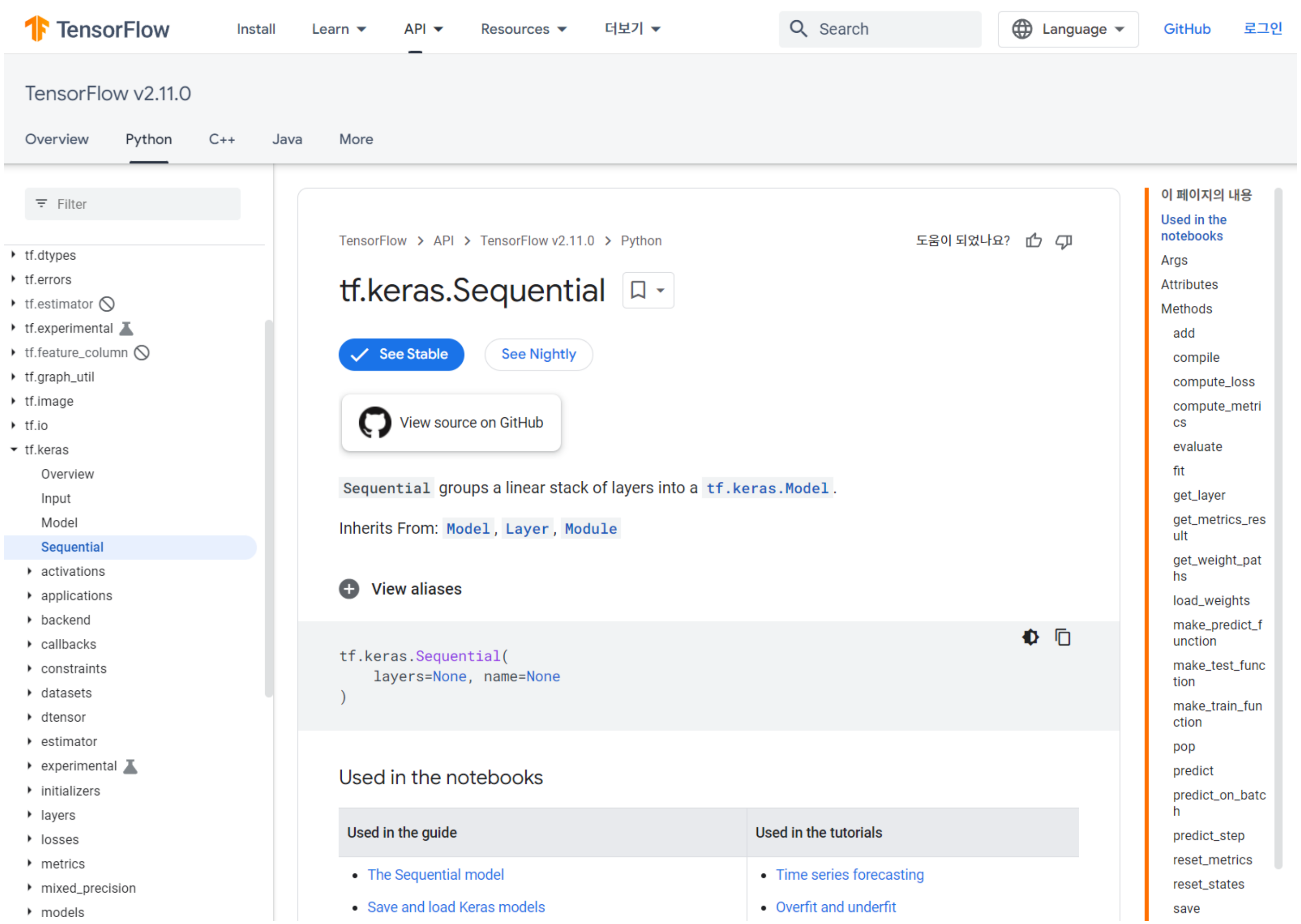
고급 빠른 시작

맞춤설정

분산형 학습

https://www.tensorflow.org/tutorials

API



tf.keras.Sequential

`Sequential` groups a linear stack of layers into a `tf.keras.Model`.

Inherits From: `Model`, `Layer`, `Module`

View aliases

```
tf.keras.Sequential(
    layers=None, name=None
)
```

Used in the notebooks

Used in the guide	Used in the tutorials
• The Sequential model	• Time series forecasting
• Save and load Keras models	• Overfit and underfit

https://www.tensorflow.org/api_docs/python/tf