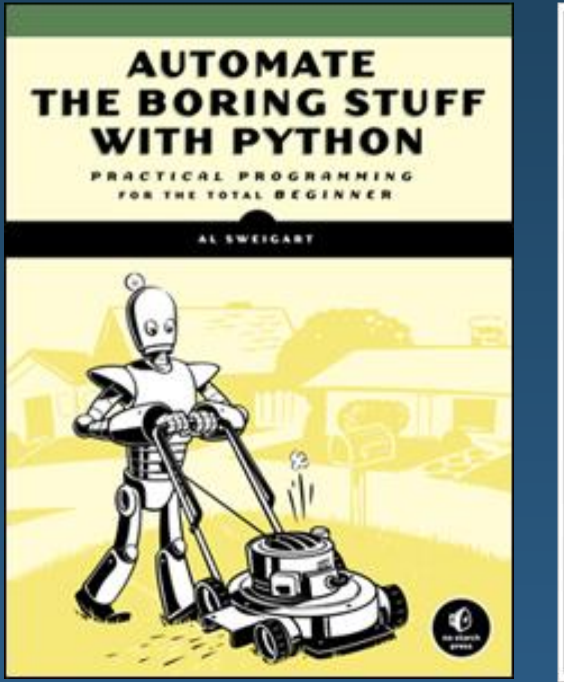
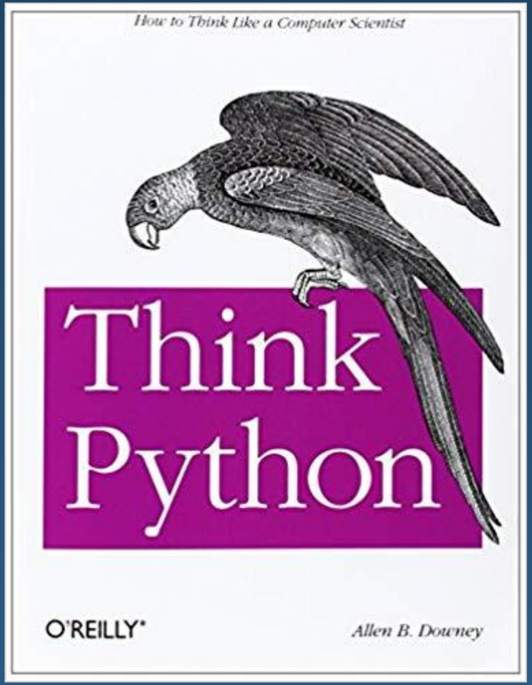
Python Basics

November 2018

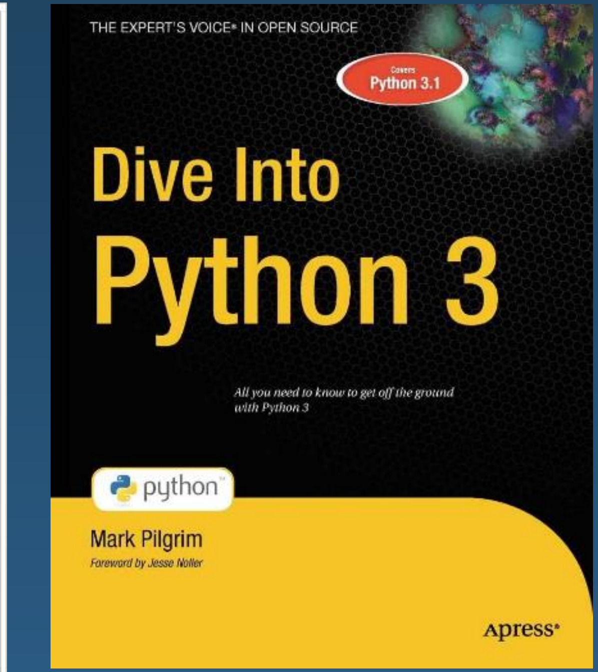
References and Recommended Materials


AUTOMATE THE BORING STUFF WITH PYTHON
PRACTICAL PROGRAMMING FOR THE TOTAL BEGINNER
AL SWEIGART

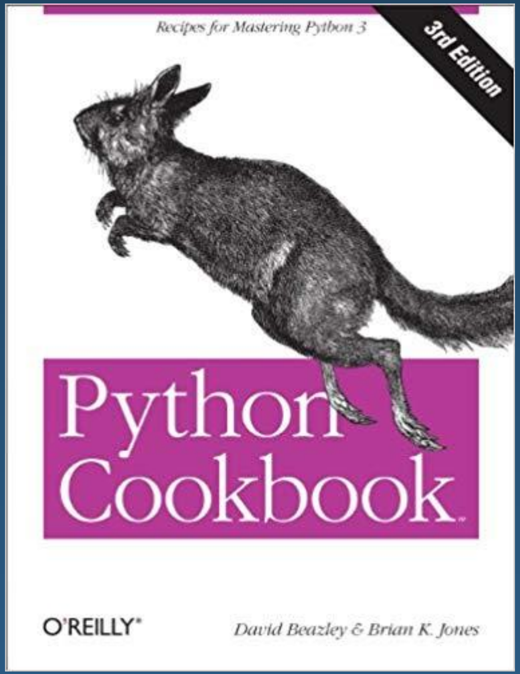
(online book)



(online book)


THE EXPERT'S VOICE® IN OPEN SOURCE
Covers Python 3.1
Dive Into Python 3
All you need to know to get off the ground with Python 3
python
Mark Pilgrim
Foreword by Jesse Noller
Apress

(中文版)


Recipes for Mastering Python 3
3rd Edition
Python Cookbook
O'REILLY
David Beazley & Brian K. Jones

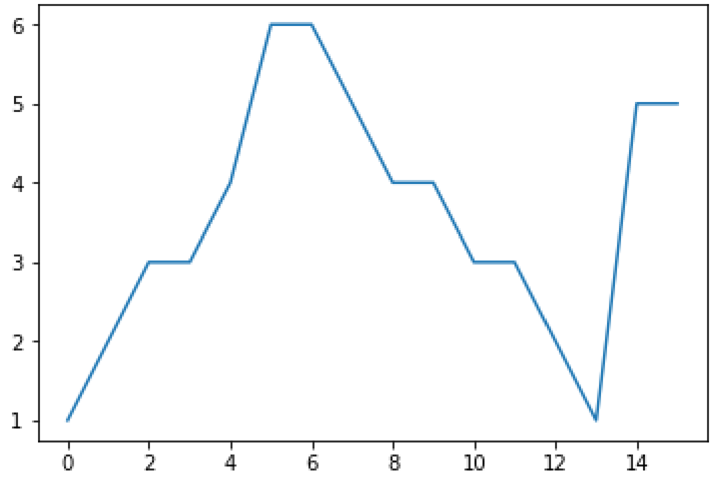
(中文版)

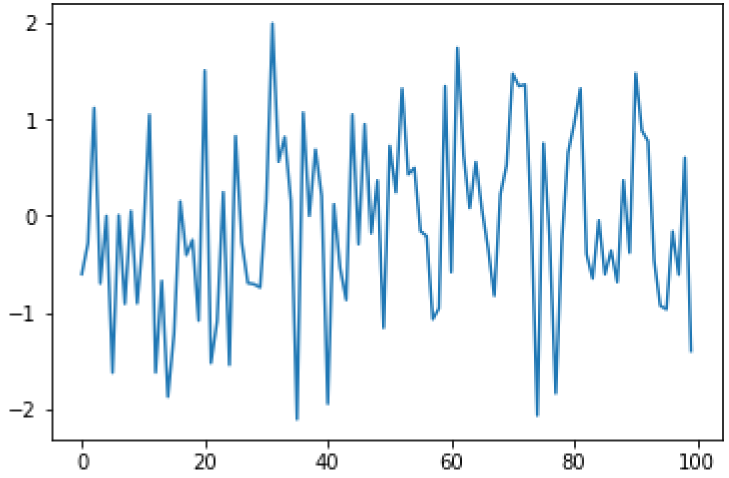
References and Recommended Materials

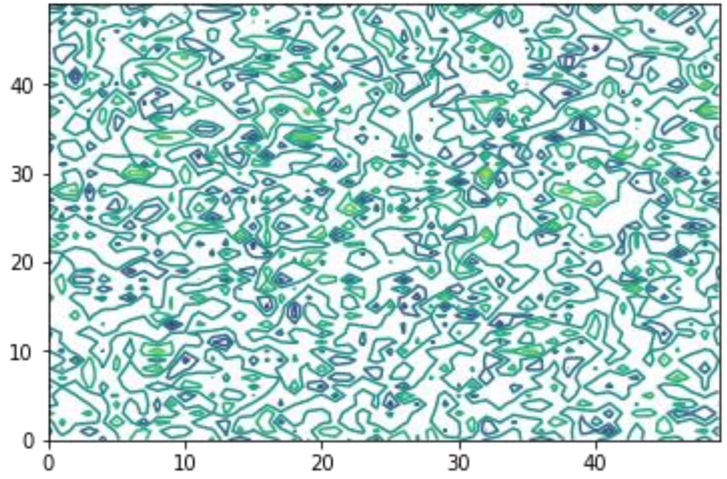
- Automate the Boring Stuff with Python (online book)
- Think Python: How to Think Like a Computer Scientist (online book)
- Dive into Python 3 (中文版)
- Python Cookbook 3rd Edition (中文版)

Start with an Example

- 000_the_first_example.ipynb



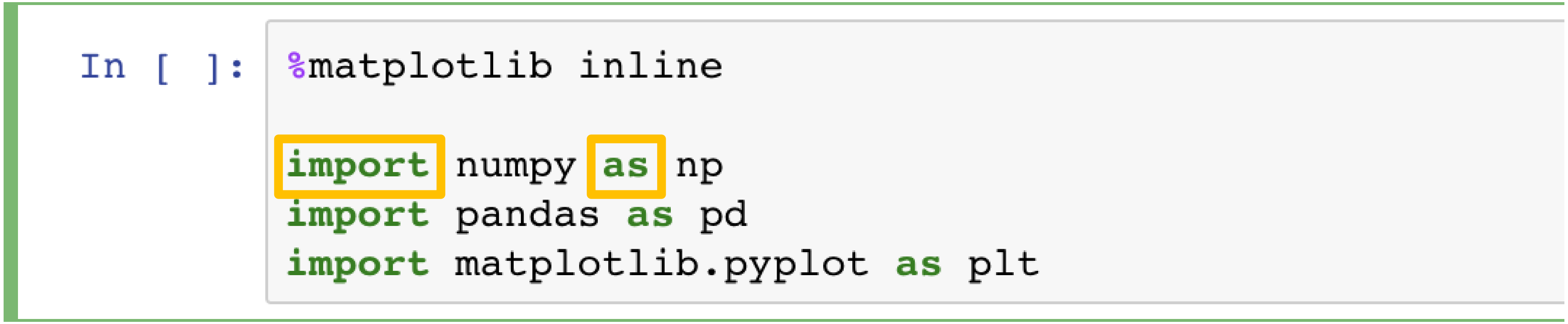




```
# symmetric difference: items appearing in only one set primes ^ odds # with an operator primes.symmetric_difference(odds) # equivalently with a method
```

Start with an Example

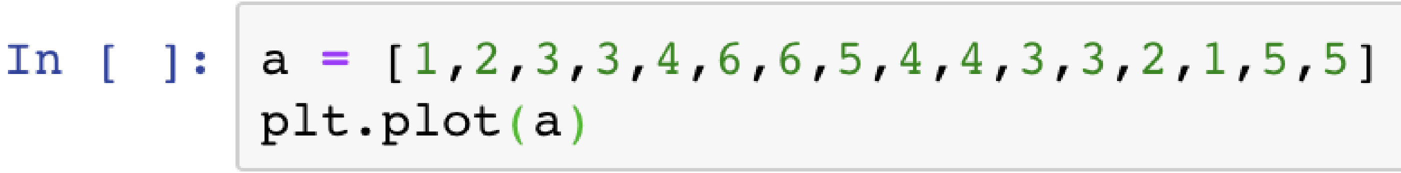
- Loading libraries



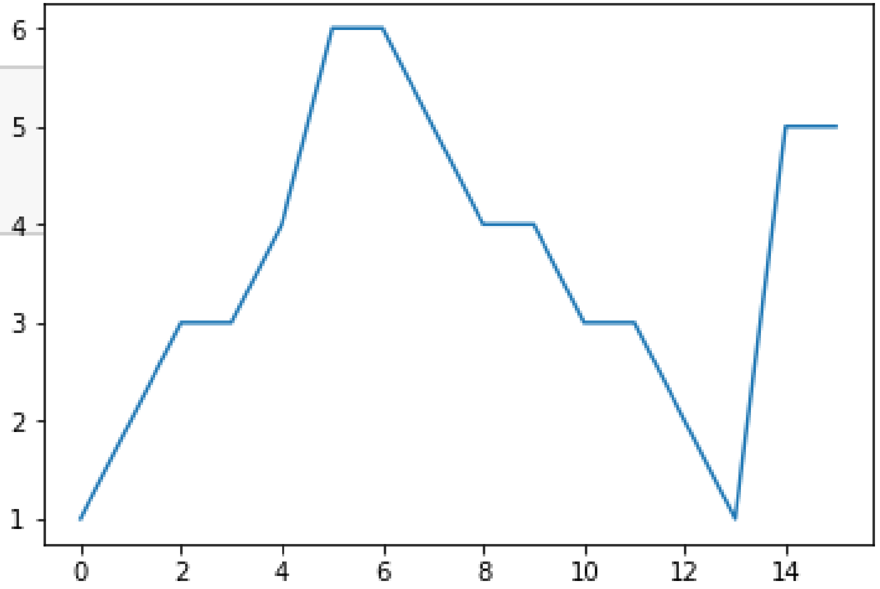
```
In [ ]: %matplotlib inline

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

Start with an Example

- Python built-in data structures: list
- **List** is a collection which is ordered and changeable. Allows duplicate members.
- In python, you can plot a vector with one line of code

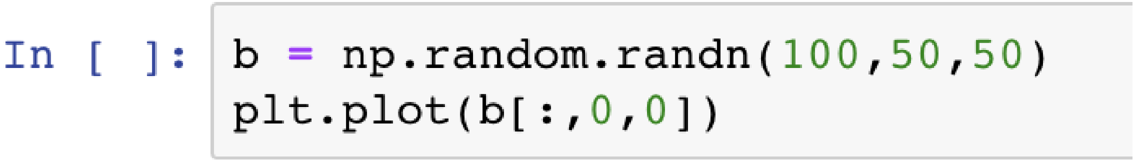


```
In [ ]: a = [1,2,3,3,4,6,6,5,4,4,3,3,2,1,5,5]
        plt.plot(a)
```

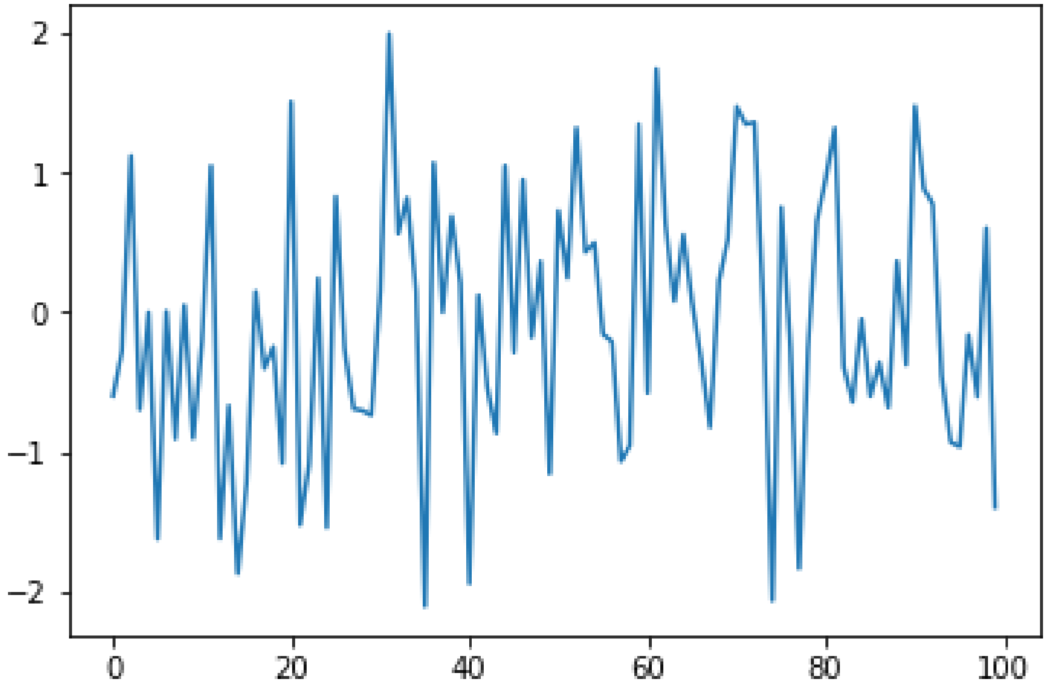


Start with an Example

- Want some random numbers? Simply specify how many you want.
- You can access n-dimensional numpy array easily.

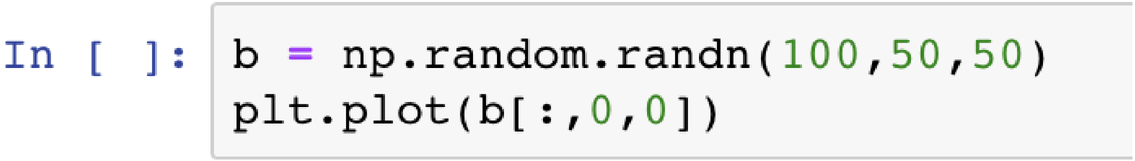


```
In [ ]: b = np.random.randn(100,50,50)
        plt.plot(b[:,0,0])
```

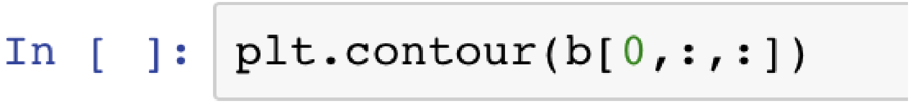


Start with an Example

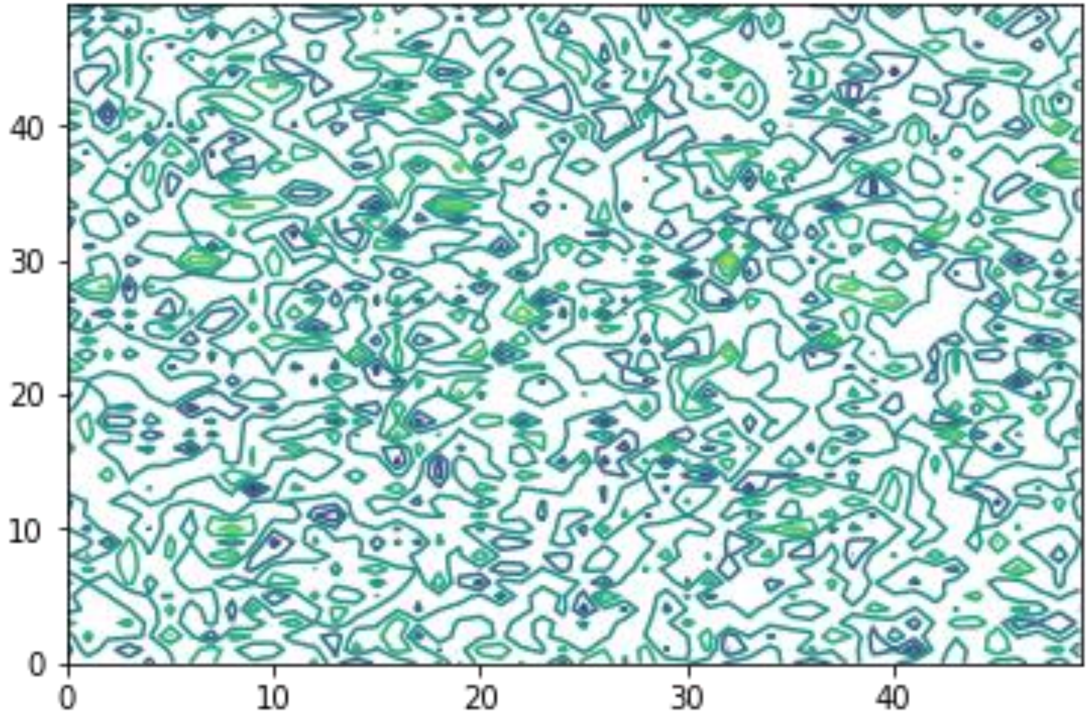
- Want some random numbers? Simply specify how many you want.
- You can access n-dimensional numpy array easily.



```
In [ ]: b = np.random.randn(100,50,50)
        plt.plot(b[:,0,0])
```



```
In [ ]: plt.contour(b[0,:,:])
```

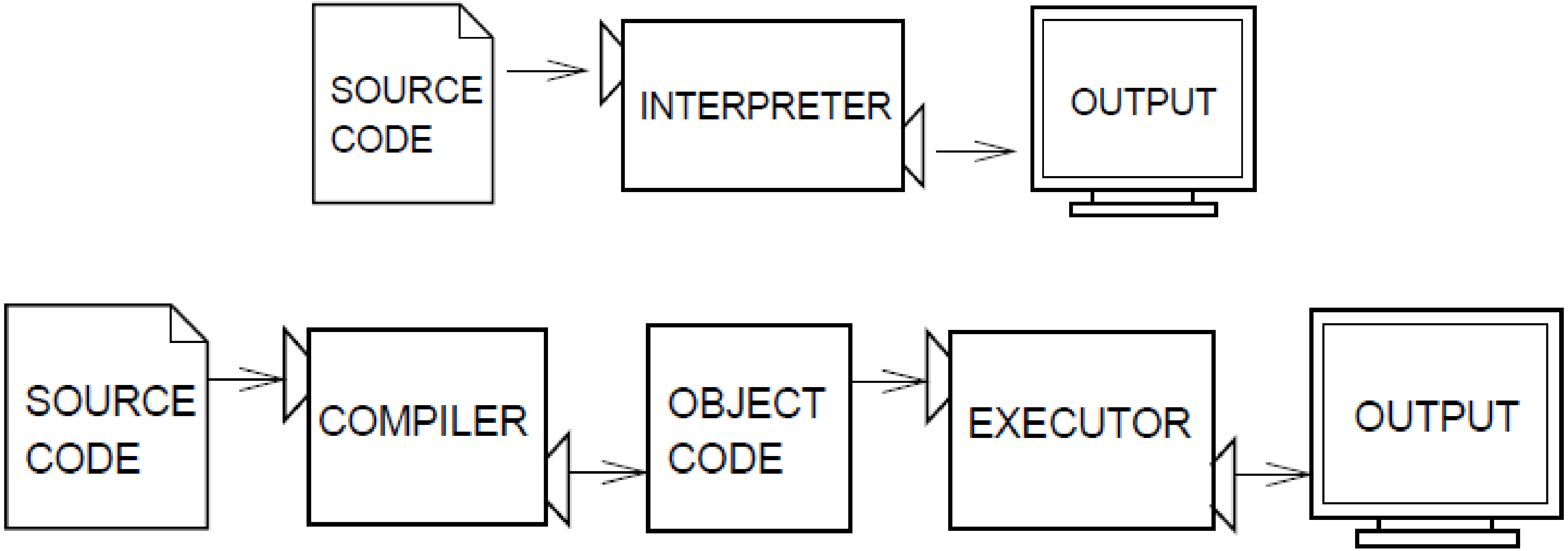


Outline of Python Basics

- Python Core
  - Variables and Controls
  - Function
  - Built-in Data Structures
  - Input / Output
  - Classes and Objects
- NumPy
- Pandas
- Matplotlib

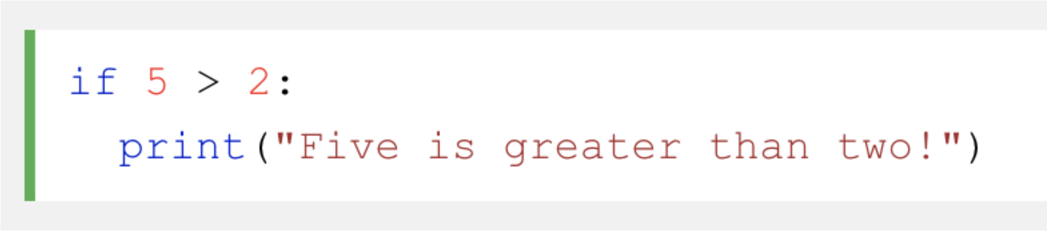
Before you code in Python

- Python is a interpreted language



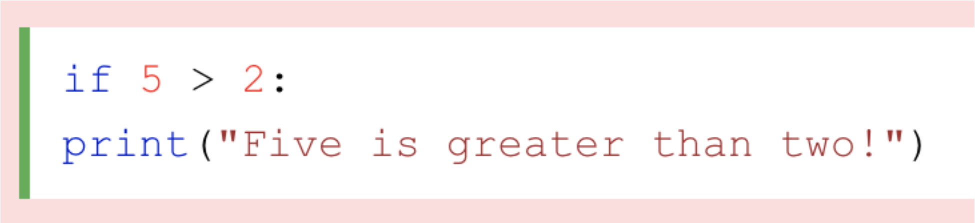
Before you code in Python

- Indentation is strictly enforced in python
  - Correct



```
if 5 > 2:
  print("Five is greater than two!")
```

  - Error



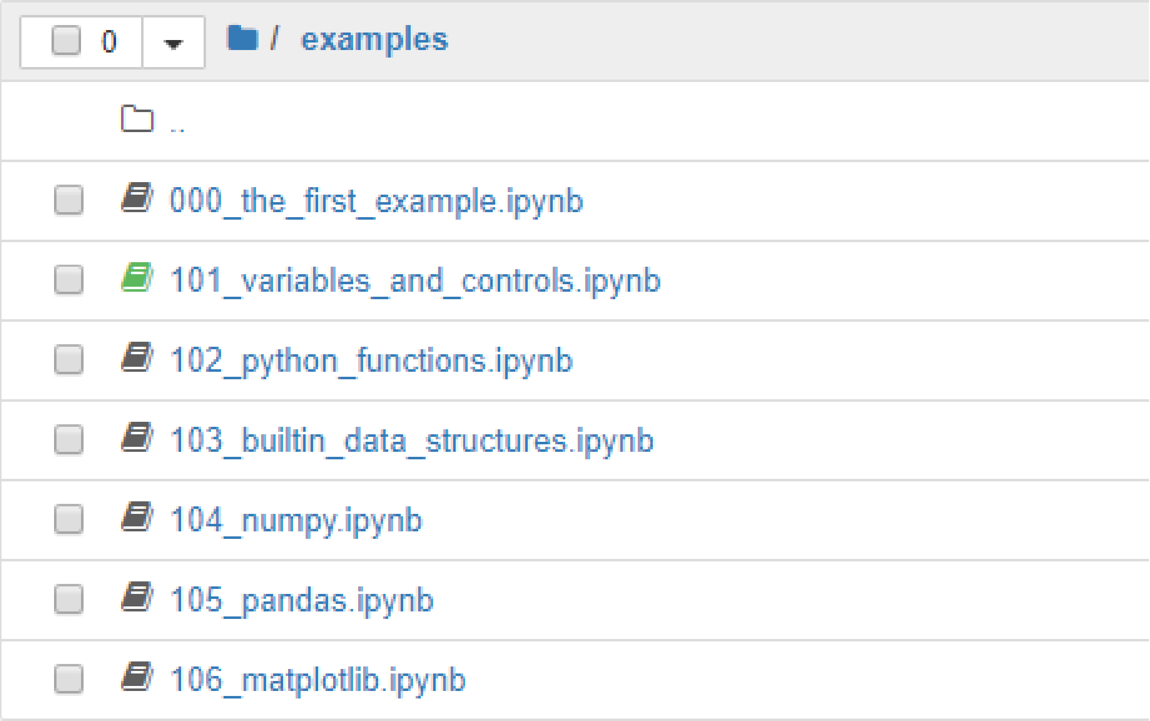
```
if 5 > 2:
print("Five is greater than two!")
```

- Tab is different from space.
- Setup your text editor carefully.

Python Variables and Controls

(1/12)

- Live demo: 101_variables_and_controls.ipynb
- Variables
- Casting
- Operators
- Controls:
  - `if…else…`
  - `while` loop
  - for loop



□ 0 ▾ / examples

..

□ 000_the_first_example.ipynb

□ 101_variables_and_controls.ipynb

□ 102_python_functions.ipynb

□ 103_builtin_data_structures.ipynb

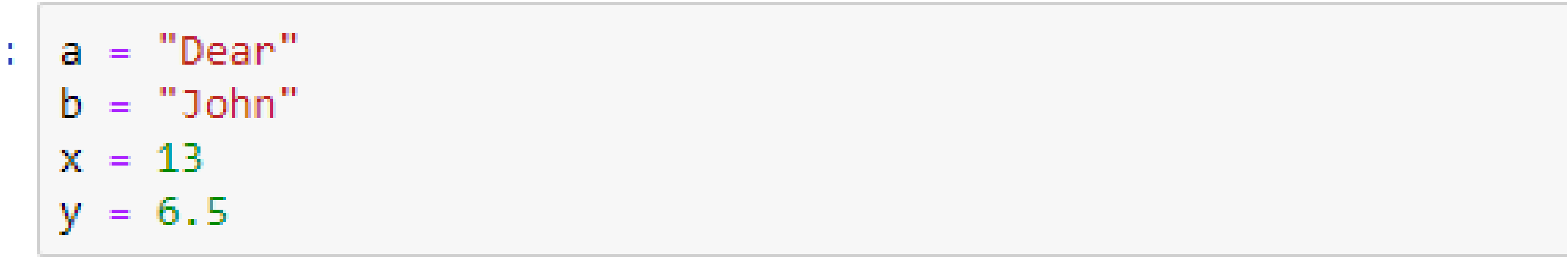
□ 104_numpy.ipynb

□ 105_pandas.ipynb

□ 106_matplotlib.ipynb

Python Variables and Controls

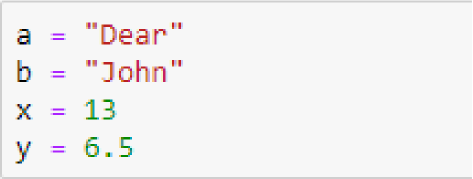
- Creating Variables
  - Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.



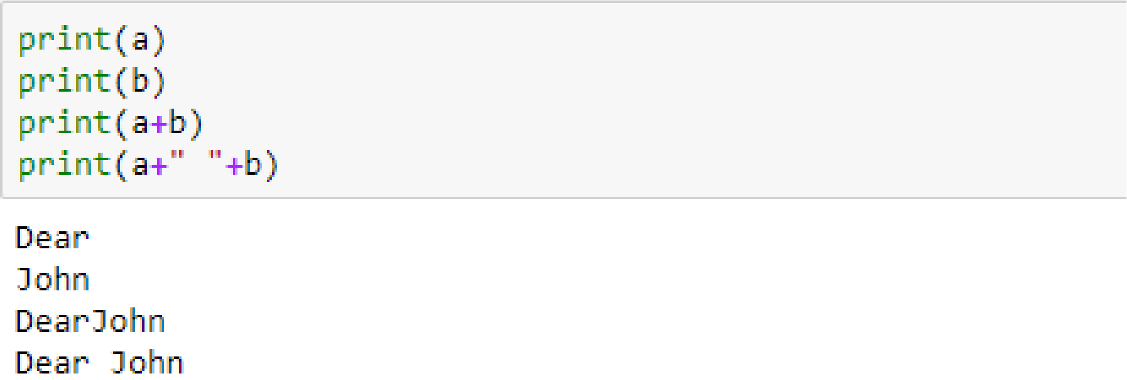
```
a = "Dear"
b = "John"
x = 13
y = 6.5
```

Python Variables and Controls

- Output Variables
  - Use `print()` statement to output variables.

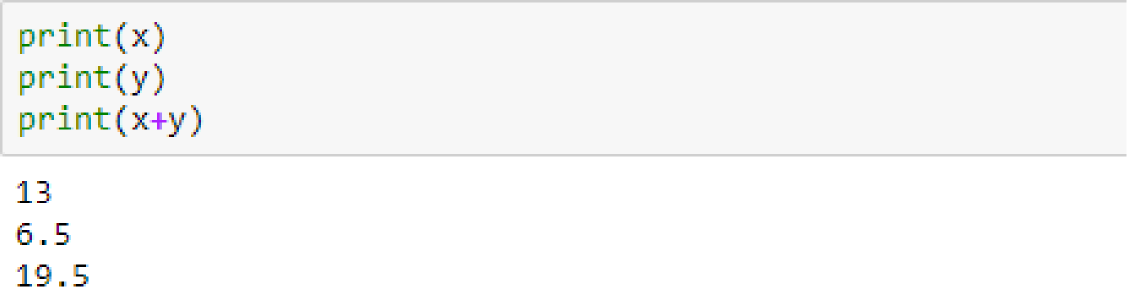


```
a = "Dear"
b = "John"
x = 13
y = 6.5
```



```
print(a)
print(b)
print(a+b)
print(a+" "+b)
```

```
Dear
John
DearJohn
Dear John
```

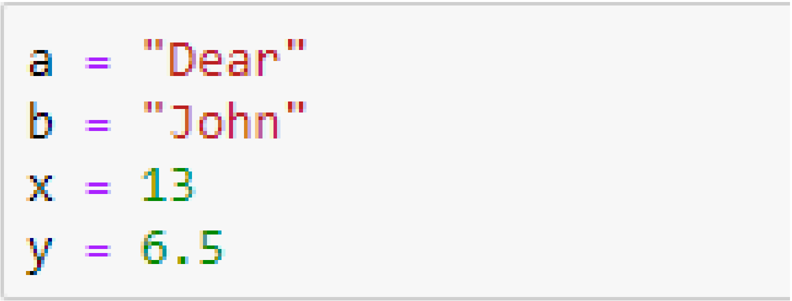


```
print(x)
print(y)
print(x+y)
```

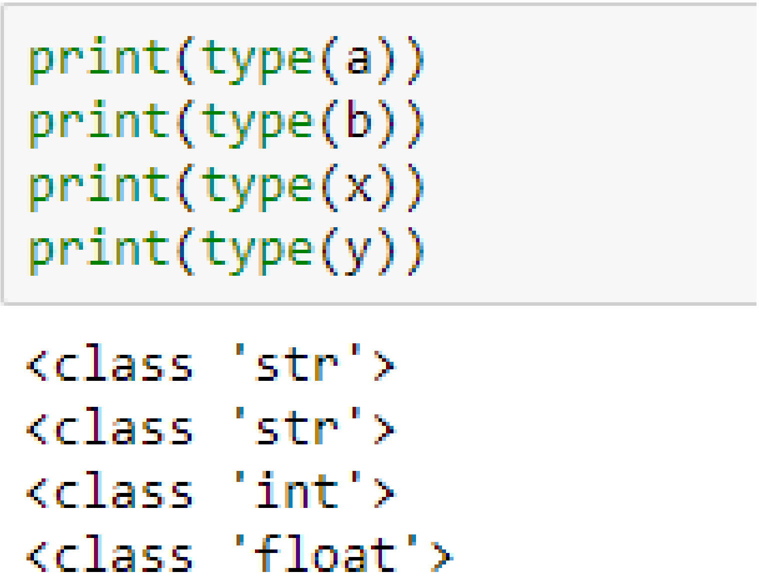
```
13
6.5
19.5
```

Python Variables and Controls (4/12)

- Variable Types
  - Most commonly used variable types: strings, integers, and float point numbers.



```
a = "Dear"
b = "John"
x = 13
y = 6.5
```



```
print(type(a))
print(type(b))
print(type(x))
print(type(y))
```

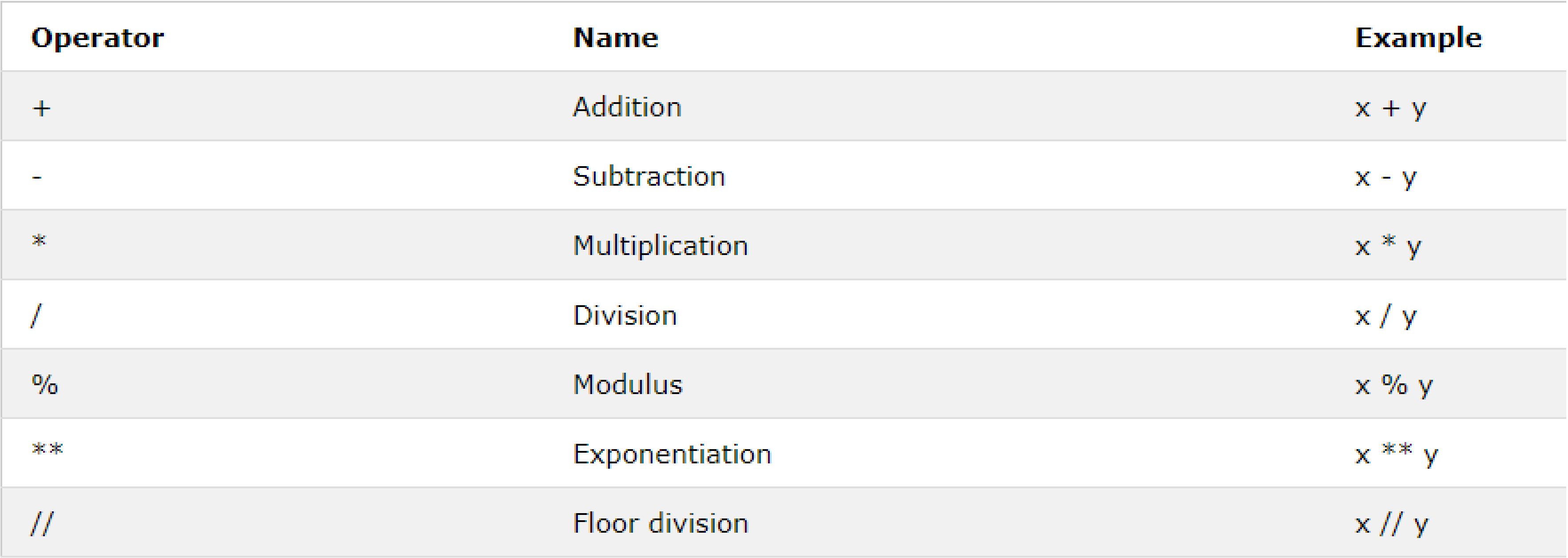
```
<class 'str'>
<class 'str'>
<class 'int'>
<class 'float'>
```

Python Variables and Controls

- Specify a Variable Type (Casting)
  - `int()`
    - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
  - `float()`
    - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
  - `str()`
    - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Python Variables and Controls (6/12)

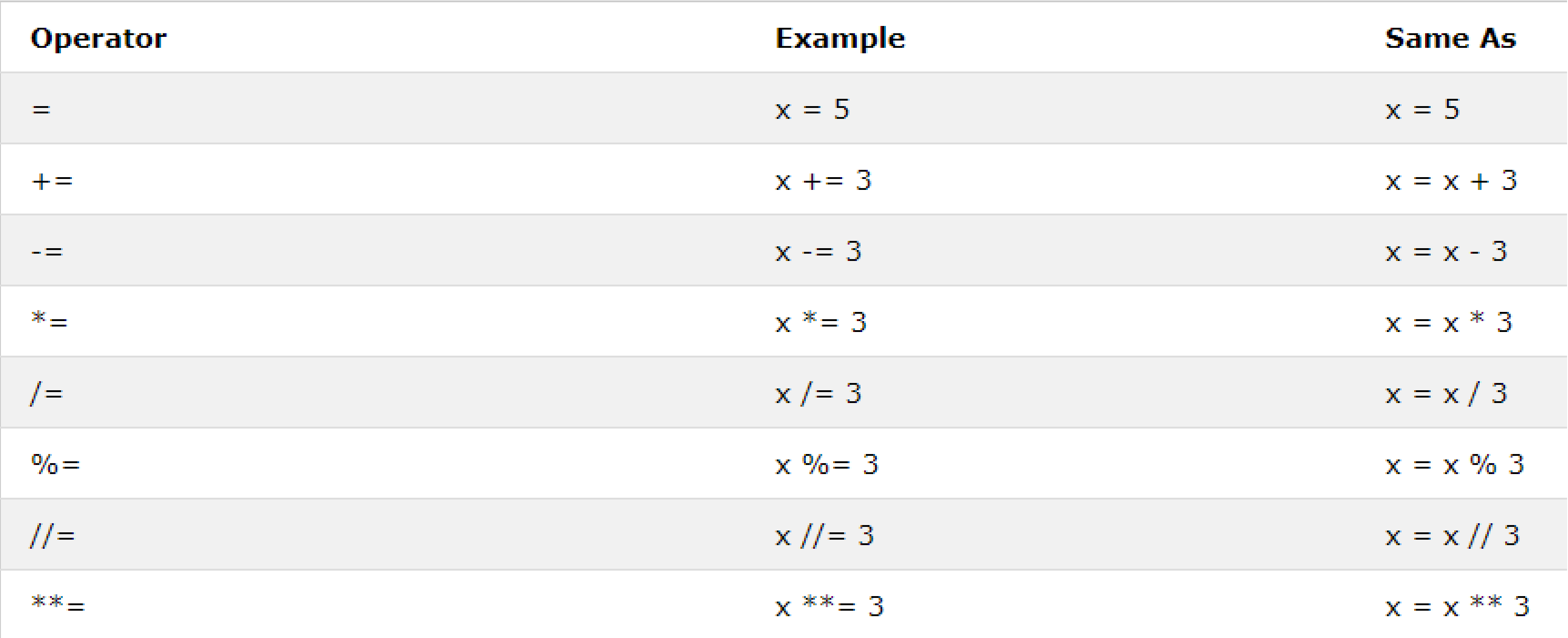
- Python Operators
  - Arithmetic Operators



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Variables and Controls (7/12)

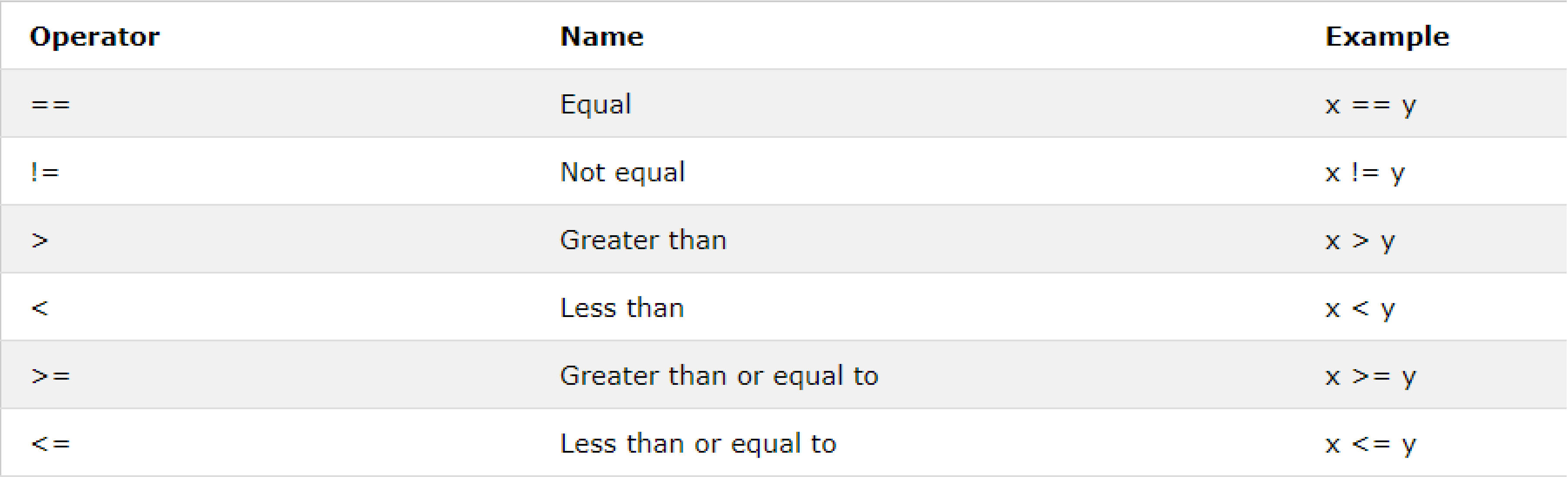
- Python Operators
  - Assignment Operators



Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Python Variables and Controls (8/12)

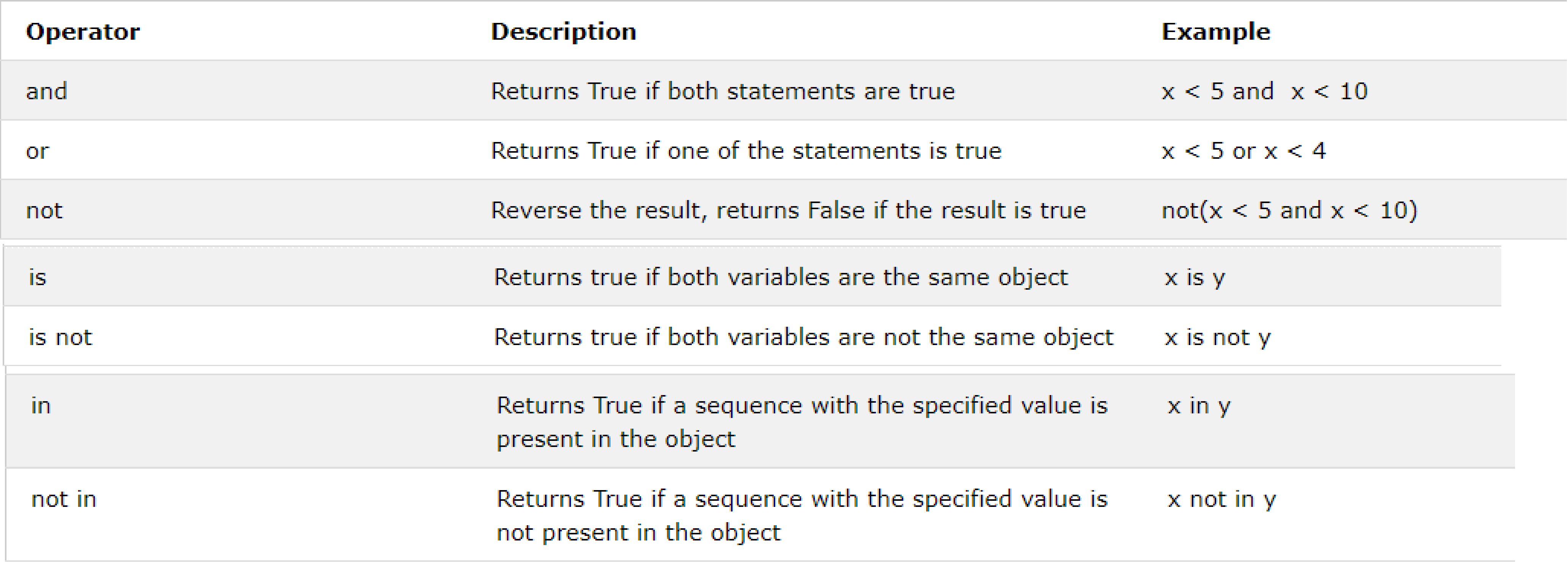
- Python Operators
    - Comparison Operators



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

Python Variables and Controls

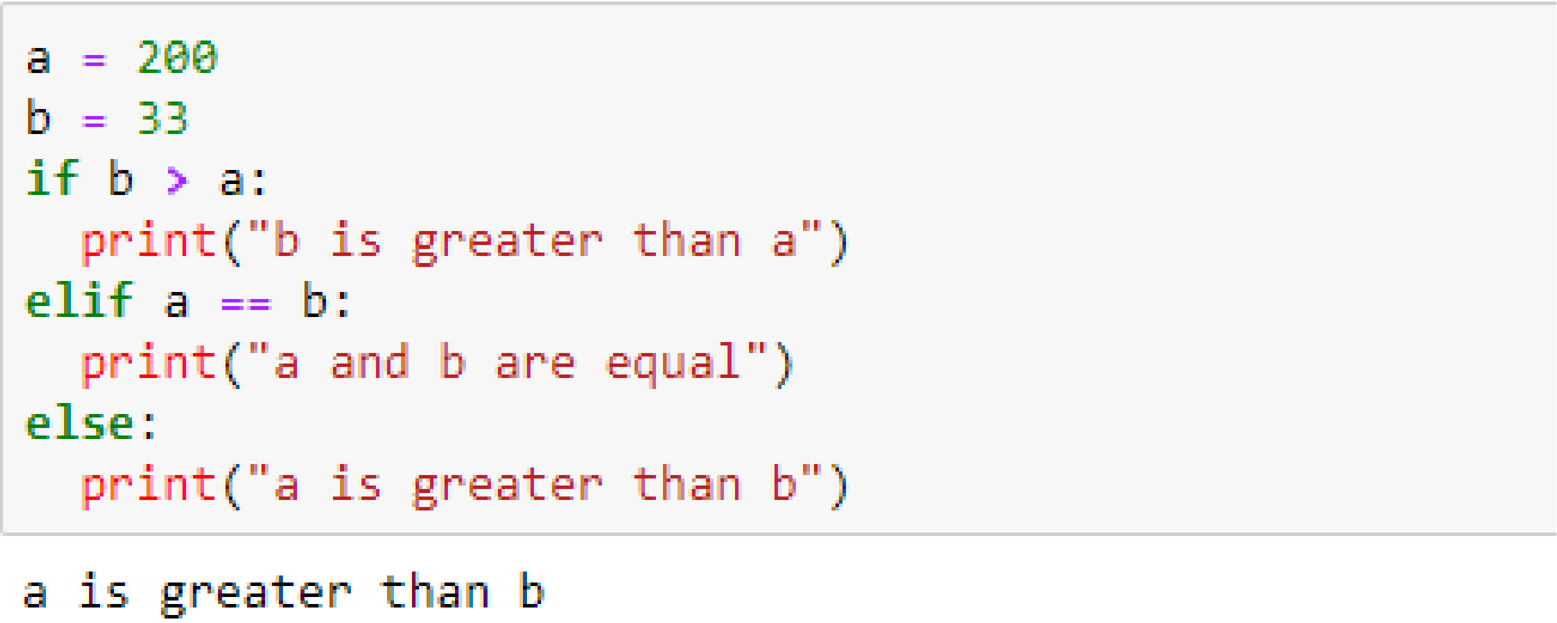
- Python Operators
  - Logical Operators



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)
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Variables and Controls

- Python Controls
  - If…else…
    - `if`
    - `elif`
    - `else`
    - Indentation!

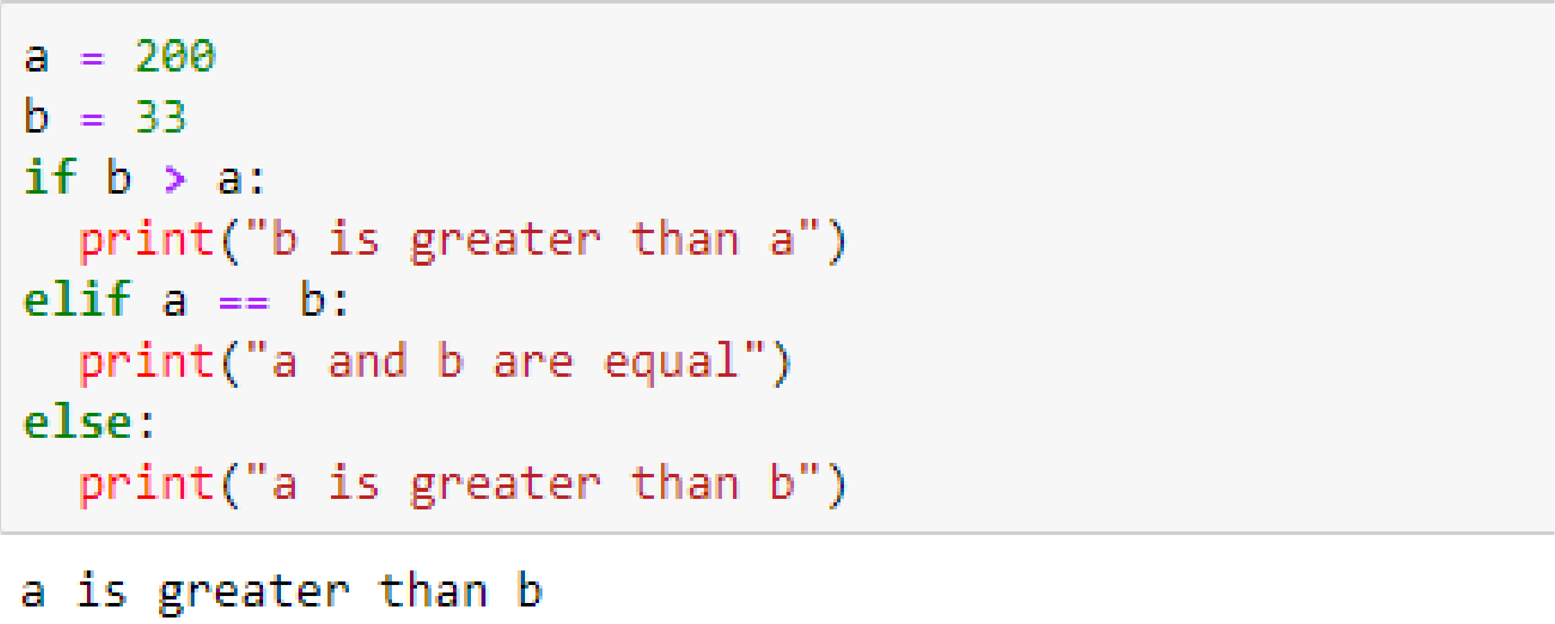


```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

Python Variables and Controls

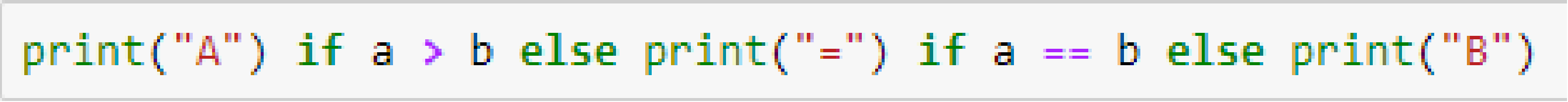
- Python Controls
  - If…else…
    - `if`
    - `elif`
    - `else`
    - Indentation!



```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

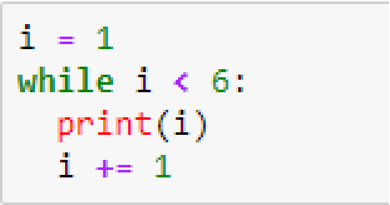
- Short version:



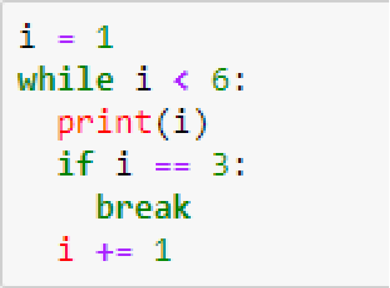
```
print("A") if a > b else print("=") if a == b else print("B")
```

Python Variables and Controls (12/12)

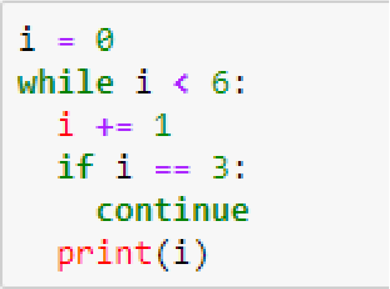
- Python Controls
  - Loops
    - The `while` loop
    - The `for` loop
    - Stop the loop: `break`
    - Skip some loop: `continue`



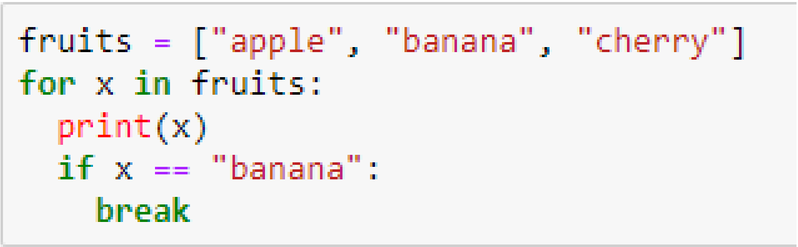
```
i = 1
while i < 6:
  print(i)
  i += 1
```



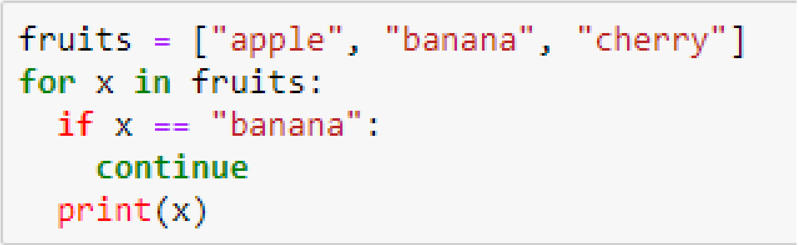
```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```



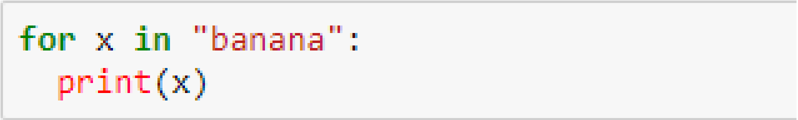
```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```



```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```



```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```



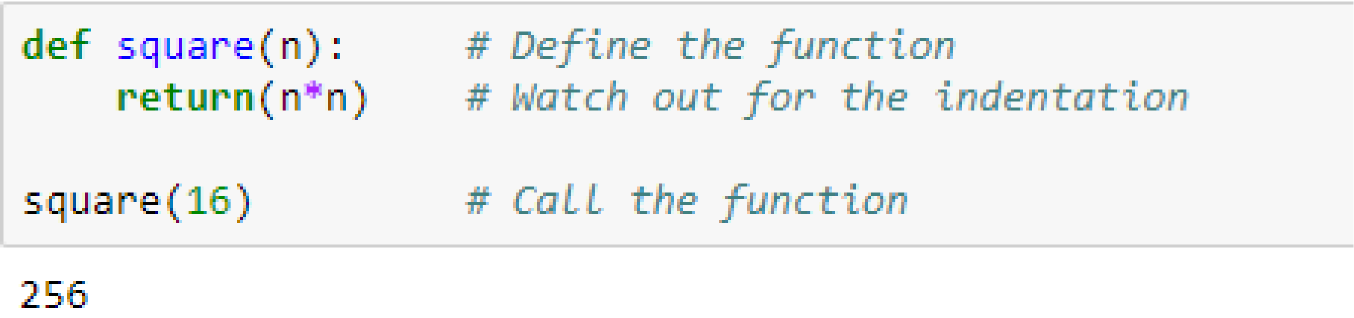
```
for x in "banana":
  print(x)
```

Python Functions

Python Functions

(1/8)

- What is a function?
  - A function is **a block of code** which only runs when it is called.
  - You can pass data, known as **parameters / arguments**, into a function.
  - A function can **return** data as a result.
- A simple example



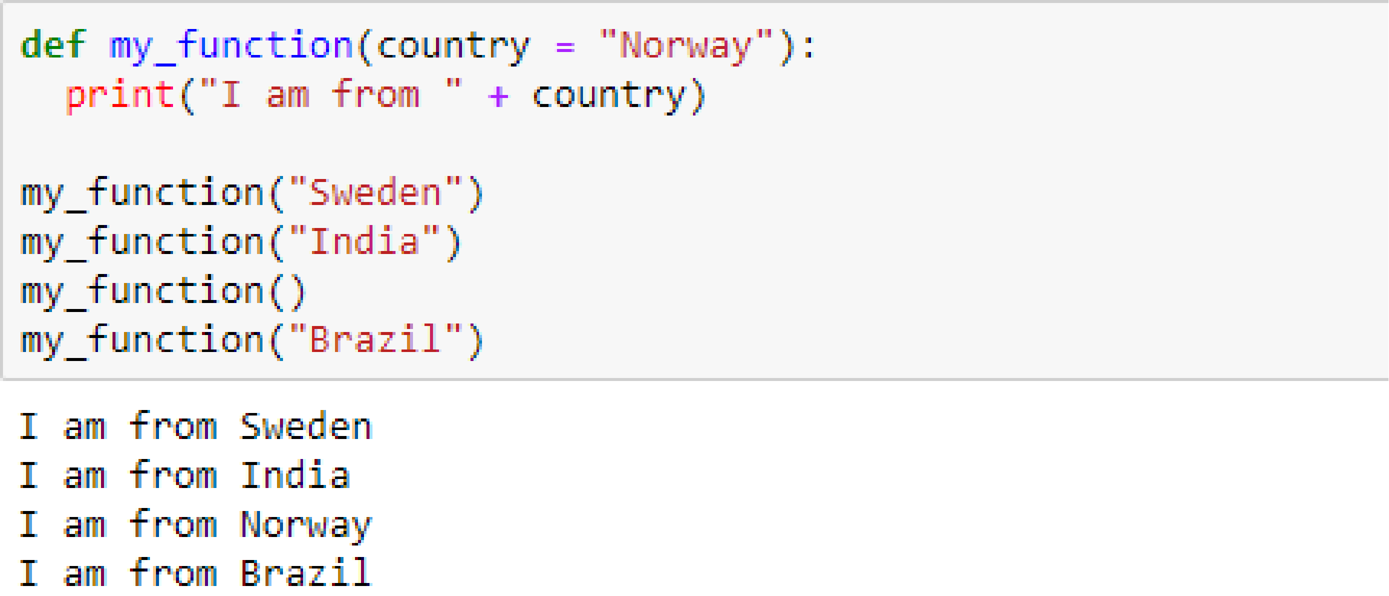
```
def square(n):      # Define the function
    return(n*n)     # Watch out for the indentation

square(16)          # Call the function
```

```
256
```

Python Functions

(2/8)

- Default Parameter Value
  - A set of pre-defined values can be assigned to the parameters, so that the function can be called without specified parameters.



```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

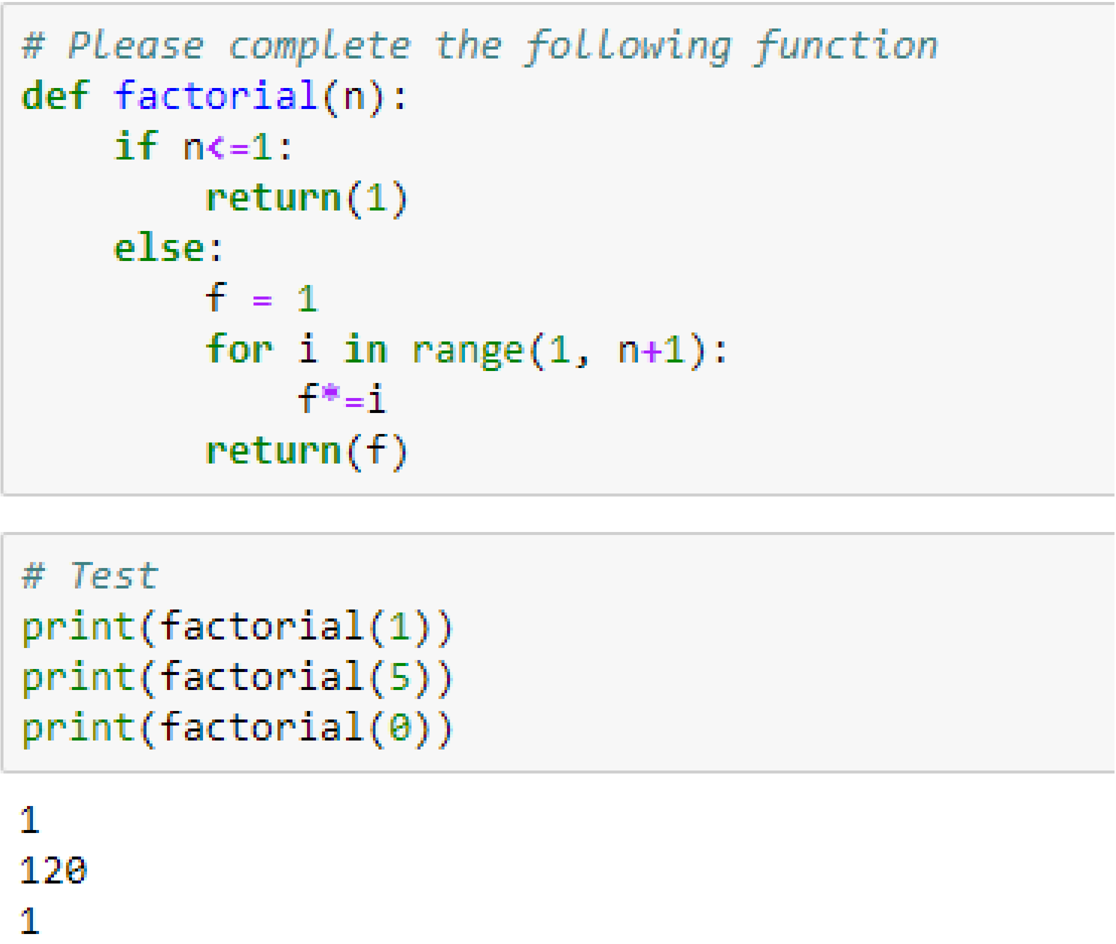
Python Functions

- **Exercise 1 Factorial**
  - In mathematics, the factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example:
  - `5! = 5 * 4 * 3 * 2 * 1 = 120`
  - By definition, `0! = 1`.
  - Please compose a function named `factorial`, which takes in one integer parameter and return the factorial of the integer.

Python Functions

(4/8)

- Exercise 1



```
# Please complete the following function
def factorial(n):
    if n<=1:
        return(1)
    else:
        f = 1
        for i in range(1, n+1):
            f*=i
        return(f)
```

```
# Test
print(factorial(1))
print(factorial(5))
print(factorial(0))
```

```
1
120
1
```

Python Functions

- **Exercise 2 Combination**
  - In mathematics, a combination is a selection of items from a collection, such that the order of selection does not matter.
  - More formally, a **k-combination of a set S** is a subset of k distinct elements of S. If the set has n elements, the number of k-combinations is equal to the binomial coefficient.

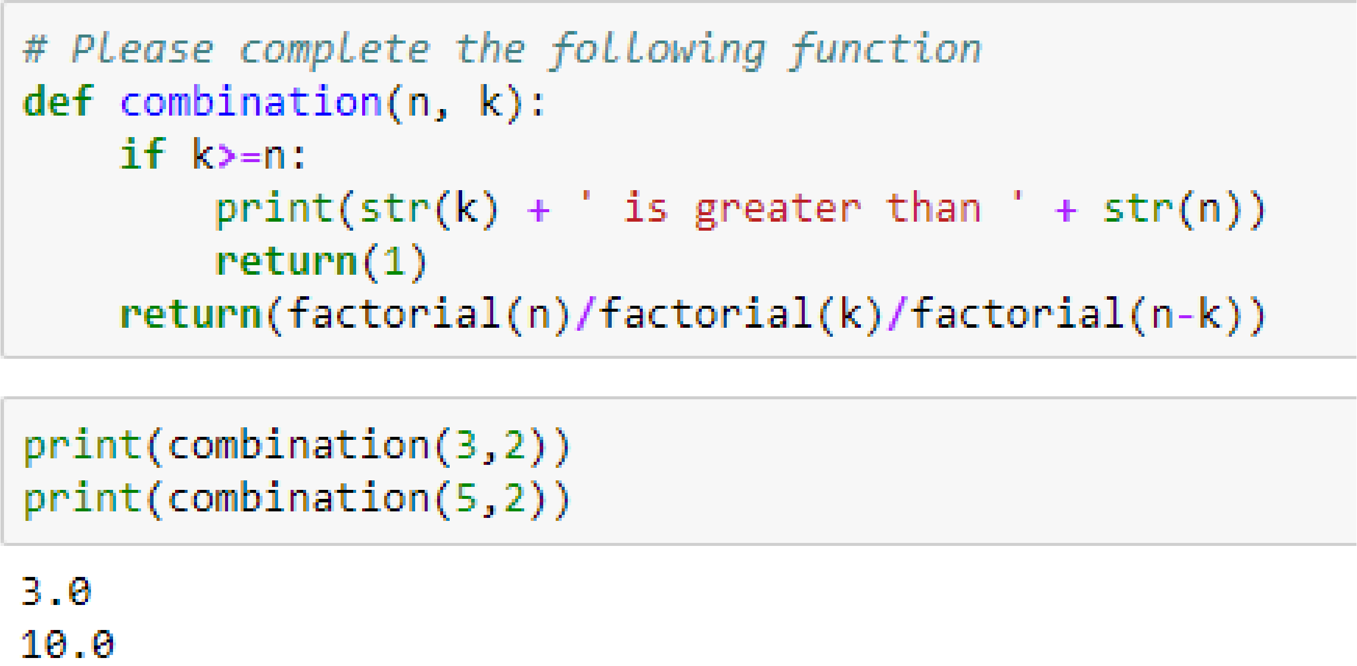
$$combination(n,k) = \frac{n(n-1)\ldots(n-k+1)}{k(k-1)\ldots1} = \frac{n!}{k!(n-k)!}$$

  - Please compose a function named `combination` which takes two integer parameters and return the number of combinations. Please use the `factorial` function you just completed.

Python Functions

(6/8)

- Exercise 2 Combination



```
# Please complete the following function
def combination(n, k):
    if k>=n:
        print(str(k) + ' is greater than ' + str(n))
        return(1)
    return(factorial(n)/factorial(k)/factorial(n-k))
```

```
print(combination(3,2))
print(combination(5,2))
```

```
3.0
10.0
```

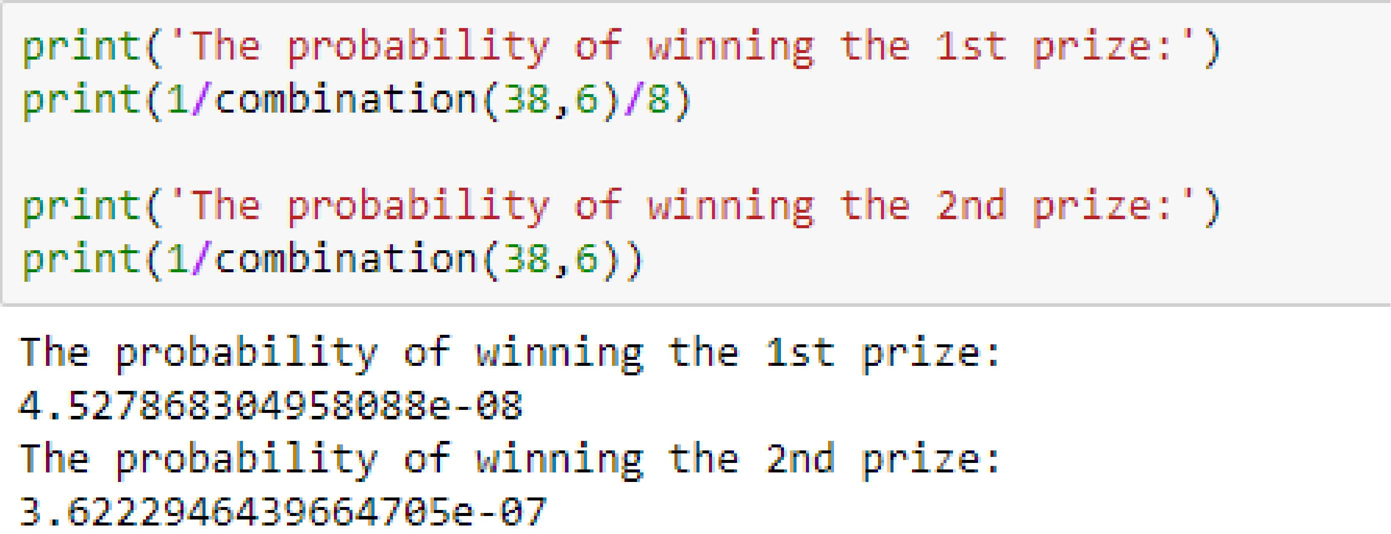
Python Functions

- **Exercise 3 Lottery Time!**
  - 威力彩是一種樂透型遊戲，其選號分為兩區，您必須從第1個選號區中的01~38 的號碼中任選6個號碼，並從第2個選號區中的01~08的號碼中任選1個號碼進行投注，這六個+一個號碼即為您的投注號碼。
    - **頭獎**：第1區六個獎號全中，且第2區亦對中獎號
    - **貳獎**：第1區六個獎號全中，但第2區未對中
  - Please use the `combination` function you completed earlier, to estimate the probability of winning the 1st and 2nd prize.

Python Functions

(8/8)

- **Exercise 3 Lottery Time!**



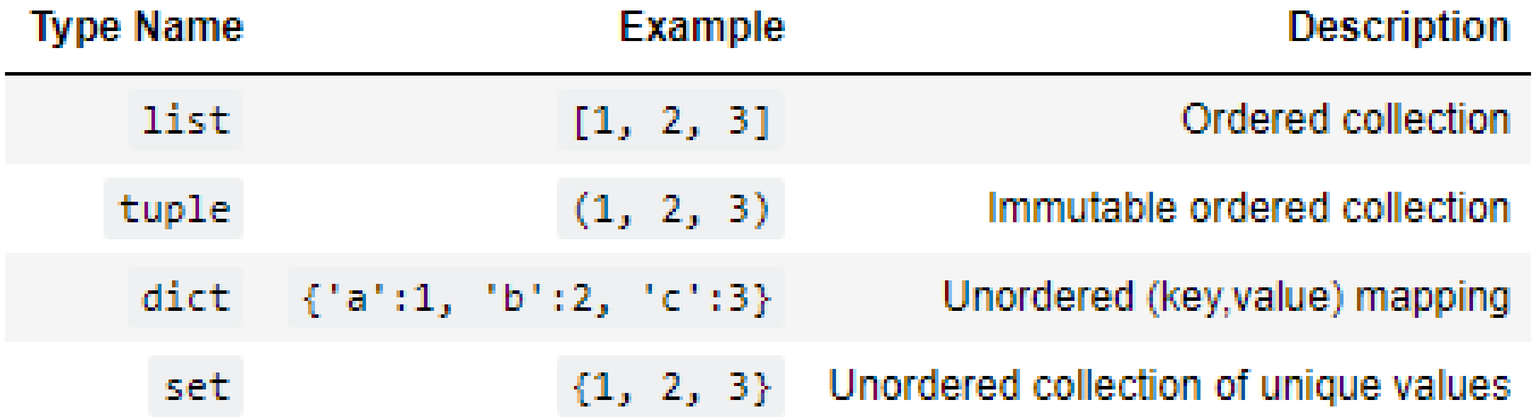
```
print('The probability of winning the 1st prize:')
print(1/combination(38,6)/8)

print('The probability of winning the 2nd prize:')
print(1/combination(38,6))
```

```
The probability of winning the 1st prize:
4.527868304958088e-08
The probability of winning the 2nd prize:
3.6222946439664705e-07
```

Python Built-in Data Structures

Python Built-in Data Structures

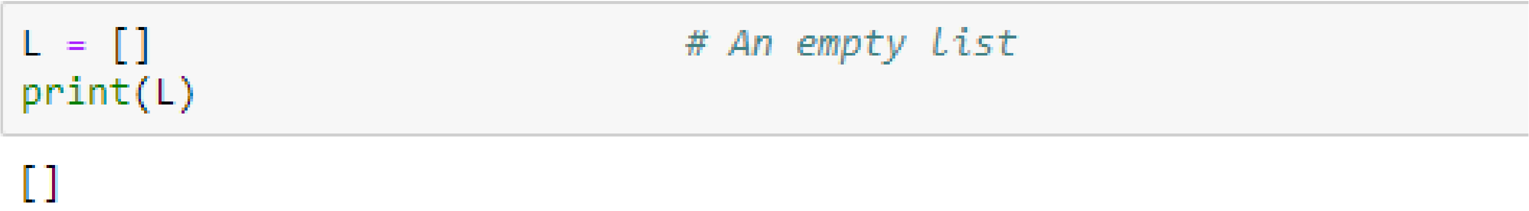
- Earlier we introduced some basic data types in python, such as `int`, `float`, `str`. Python also has several built-in compound types, which act as containers for other types.



Type Name	Example	Description
`list`	`[1, 2, 3]`	Ordered collection
`tuple`	`(1, 2, 3)`	Immutable ordered collection
`dict`	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
`set`	`{1, 2, 3}`	Unordered collection of unique values

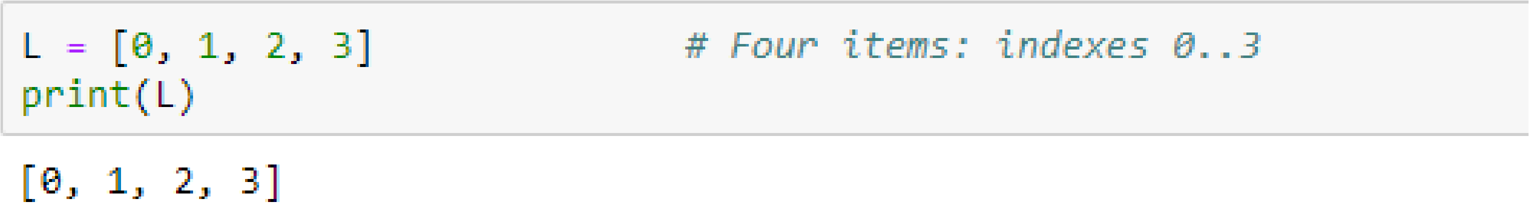
Python Built-in Data Structures

- Lists
  - The lists can contain any sort of object: numbers, strings, and even other lists.
  - Lists may be changed in-place by assignment to offsets and slices, list method calls, deletion statements.



```
L = []                              # An empty list
print(L)
```

```
[]
```

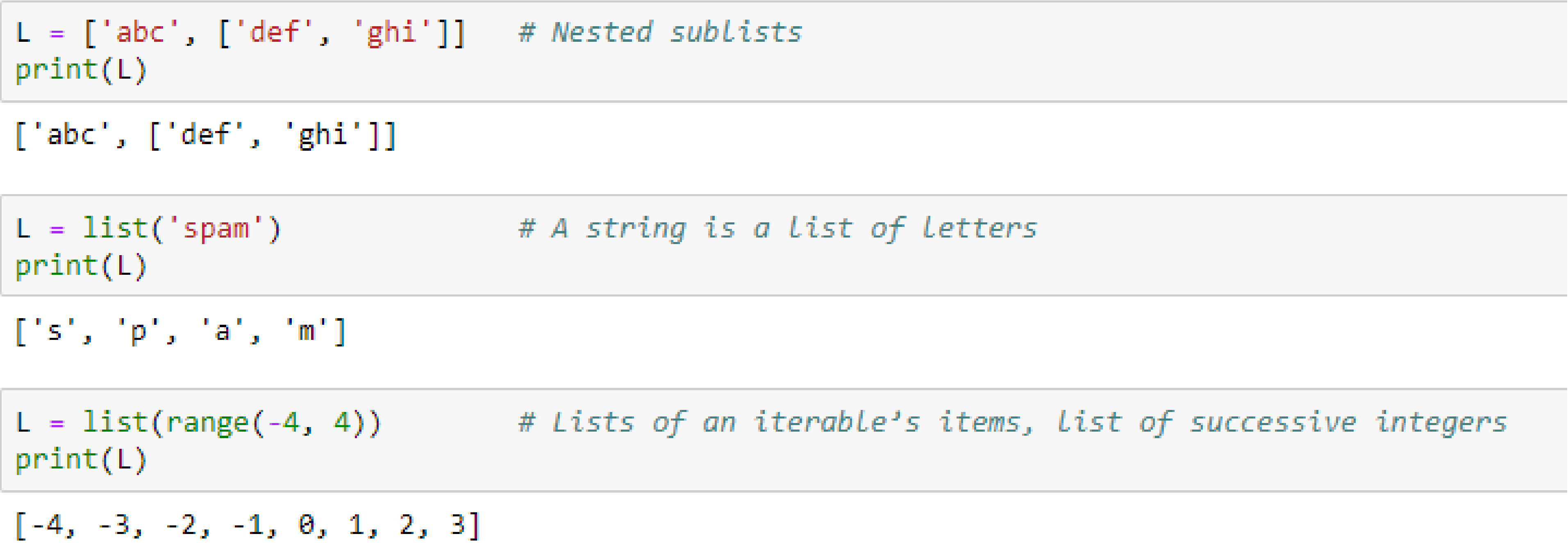


```
L = [0, 1, 2, 3]                    # Four items: indexes 0..3
print(L)
```

```
[0, 1, 2, 3]
```

Python Built-in Data Structures

- Lists



```
L = ['abc', ['def', 'ghi']]    # Nested sublists
print(L)
```

```
['abc', ['def', 'ghi']]
```

```
L = list('spam')               # A string is a list of letters
print(L)
```

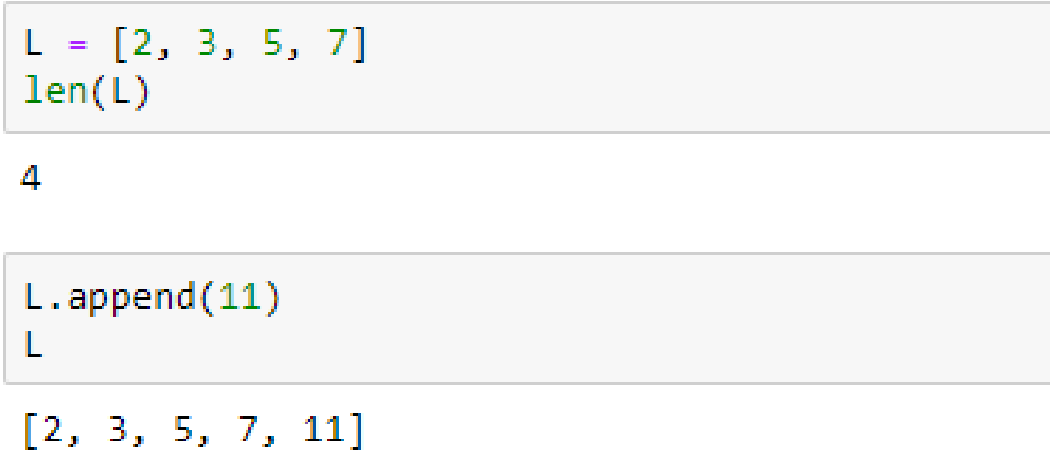
```
['s', 'p', 'a', 'm']
```

```
L = list(range(-4, 4))         # Lists of an iterable’s items, list of successive integers
print(L)
```

```
[-4, -3, -2, -1, 0, 1, 2, 3]
```

Python Built-in Data Structures

- Basic operation of lists
  - `len(L)`: the length of a list
  - `list.append(e)`: append `e` to the end of the list



```
L = [2, 3, 5, 7]
len(L)
```

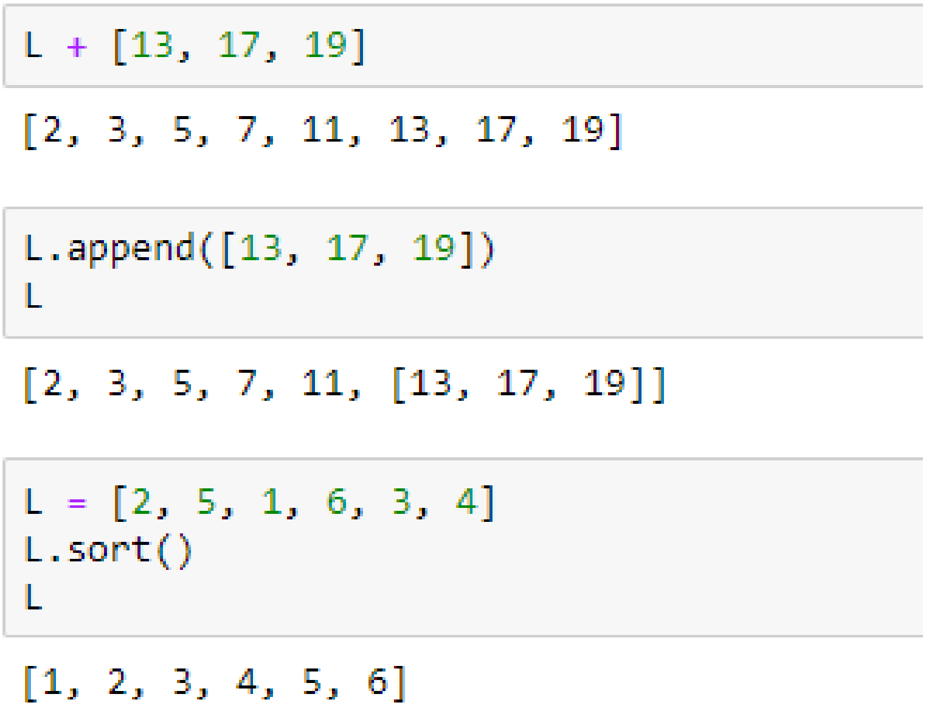
```
4
```

```
L.append(11)
L
```

```
[2, 3, 5, 7, 11]
```

Python Built-in Data Structures

- Basic operation of lists
  - `list.append(e)`: append `e` to the end of the list
  - `+`: concatenate two lists
  - `list.sort()`: in-place sorting of a list



```
L + [13, 17, 19]
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

```
L.append([13, 17, 19])
L
```

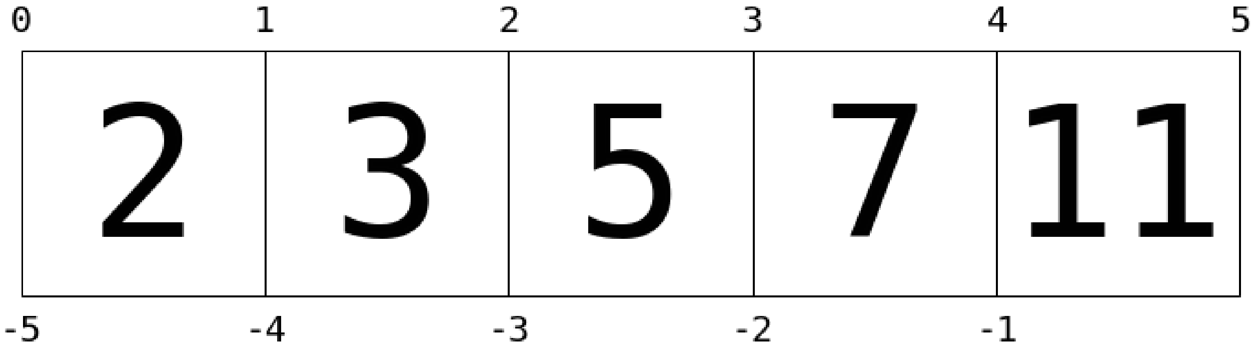
```
[2, 3, 5, 7, 11, [13, 17, 19]]
```

```
L = [2, 5, 1, 6, 3, 4]
L.sort()
L
```

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

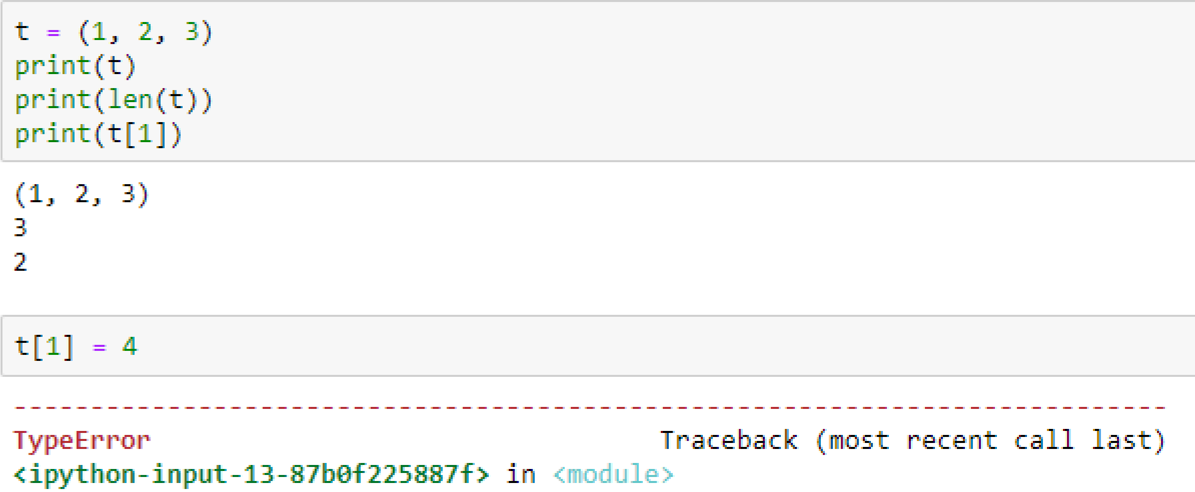
Python Built-in Data Structures

- Indexing the members in a list
  - `L[i]`: the *i*-th member
  - `L[i][j]`: index a member in a 2D list
  - `L[i:j]`: range indexing
  - `L[-1]`: the last element of the inde



Python Built-in Data Structures

- Tuples are in many ways similar to lists, but
  - they are defined with () rather than [].
  - tuples have a length, and can be indexed like a list.
  - Touples are **immutable**:
    - once they are created, their size and contents **cannot be changed**.



```
t = (1, 2, 3)
print(t)
print(len(t))
print(t[1])
```

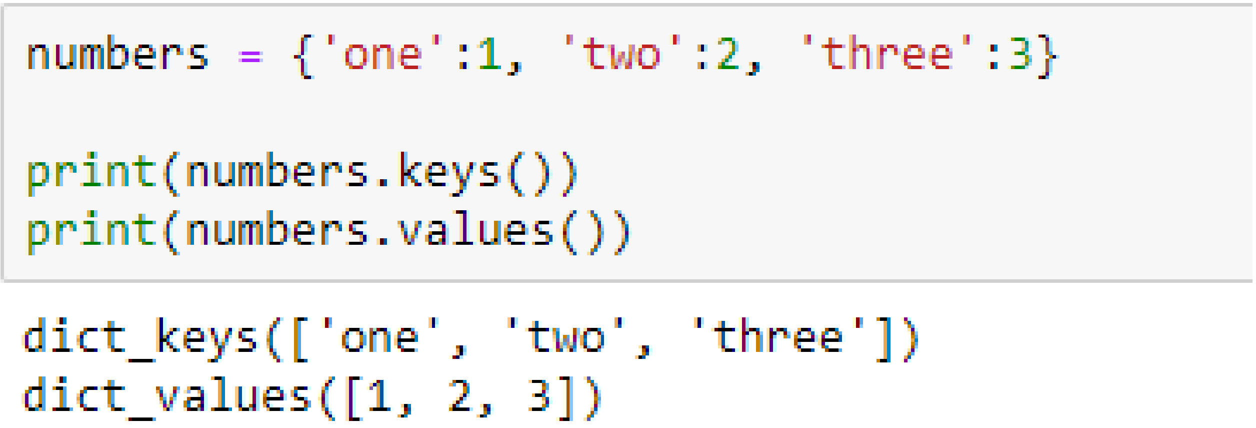
```
(1, 2, 3)
3
2
```

```
t[1] = 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-87b0f225887f> in <module>
```

Python Built-in Data Structures

- Dictionary
  - Dictionaries are extremely flexible mappings of ***keys to values***
  - The basis of much of Python's internal implementation

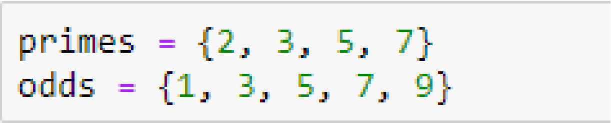


```
numbers = {'one':1, 'two':2, 'three':3}

print(numbers.keys())
print(numbers.values())
```

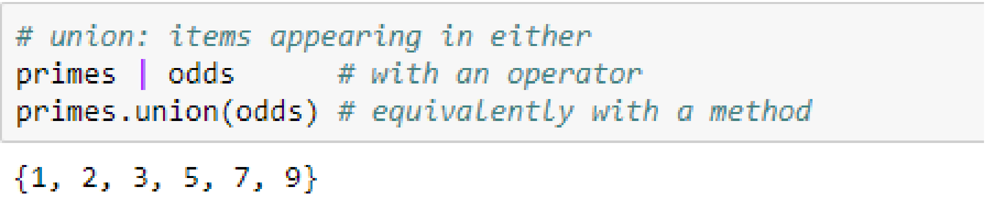
```
dict_keys(['one', 'two', 'three'])
dict_values([1, 2, 3])
```

Python Built-in Data Structures

- Set
  - The set contains unordered collections of unique items.
  - They are defined with the curly brackets.

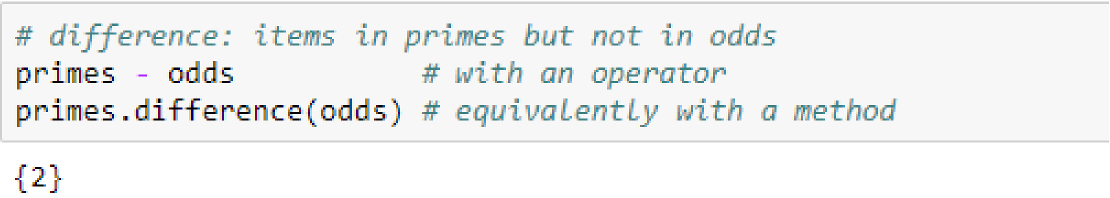


```
primes = {2, 3, 5, 7}
odds = {1, 3, 5, 7, 9}
```



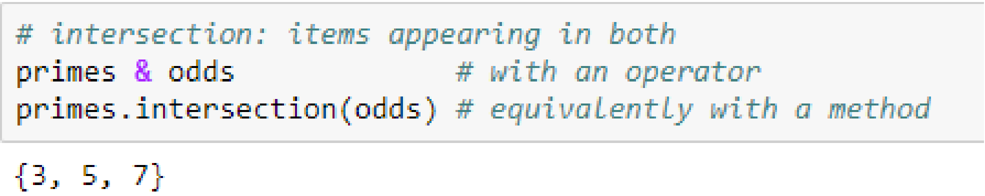
```
# union: items appearing in either
primes | odds      # with an operator
primes.union(odds) # equivalently with a method
```

```
{1, 2, 3, 5, 7, 9}
```



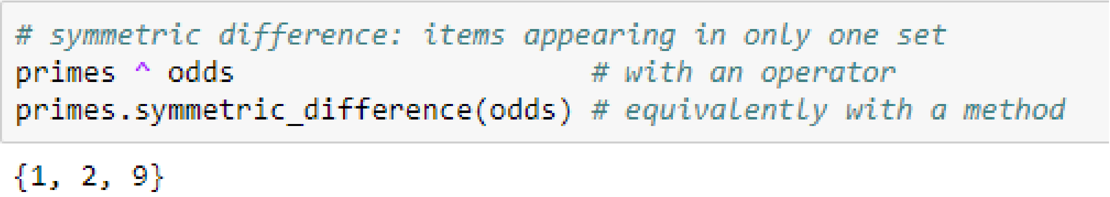
```
# difference: items in primes but not in odds
primes - odds           # with an operator
primes.difference(odds) # equivalently with a method
```

```
{2}
```



```
# intersection: items appearing in both
primes & odds             # with an operator
primes.intersection(odds) # equivalently with a method
```

```
{3, 5, 7}
```



```
# symmetric difference: items appearing in only one set
primes ^ odds                     # with an operator
primes.symmetric_difference(odds) # equivalently with a method
```

```
{1, 2, 9}
```

More Data Structures

- The lists, tuples, dictionaries, and sets are very useful.
- But for data analysis, we need some advanced data structures with functions to speed up the processing.

NumPy

SciPy.org

scikit learn

matplotlib

pandas

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

Python Libraries for Data Science

Python Libraries for Data Science

NumPy:

- Introduces new objects such as **multi-dimensional arrays** and matrices
- Introduces functions for advanced **mathematical** and **statistical** operations
- Provides **vectorization** of mathematical operations (significant performance gain)

Link: http://www.numpy.org/

Python Libraries for Data Science

SciPy:

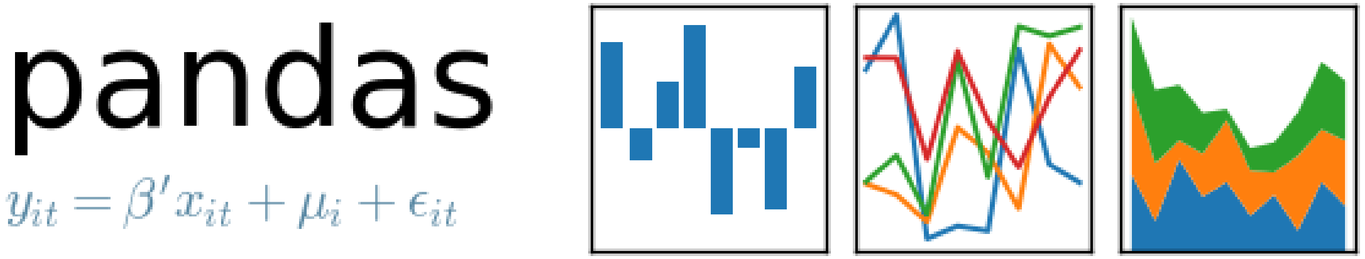
- Provides algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more.

Link: https://www.scipy.org/scipylib/

Python Libraries for Data Science

Pandas:

- Adds **DataFrame** designed to work with table-like data
- Provides **data manipulation** tools such merging, sorting, slicing, and aggregation
- Provides tools for **missing data** processing



Link: http://pandas.pydata.org/

Python Libraries for Data Science

SciKit-Learn:

- Provides implementations of classical **machine learning** algorithms such as classification, regression, clustering, and model validation

Link: http://scikit-learn.org/

Python Libraries for Data Science

matplotlib:

- A python **2D plotting library** which produces publication quality figures in a variety of hardcopy formats

Link: https://matplotlib.org/

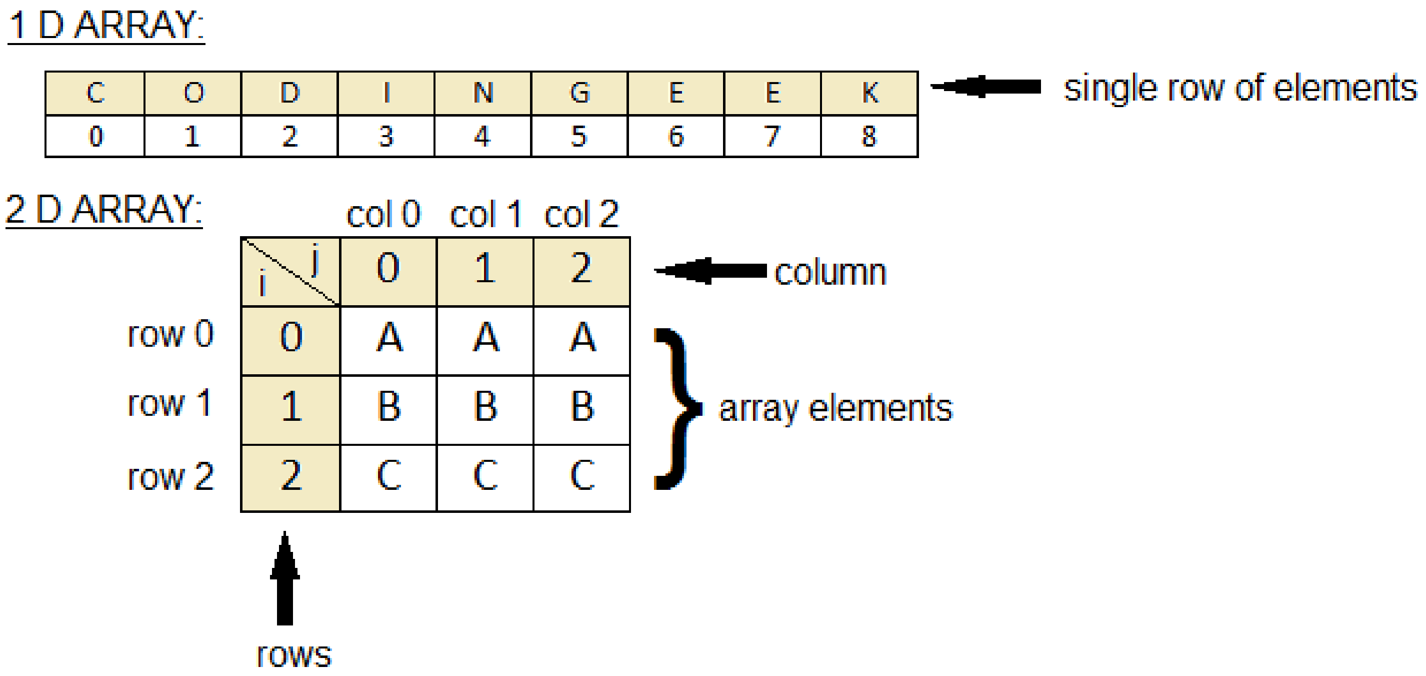
NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful **N-dimensional array object**
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful **linear algebra**, **Fourier transform**, and **random number** capabilities

NumPy Array Structures

- Key attributes
  - dtype
  - shape
  - ndim
  - strides
  - data



1 D ARRAY:

C	O	D	I	N	G	E	E	K
0	1	2	3	4	5	6	7	8

single row of elements

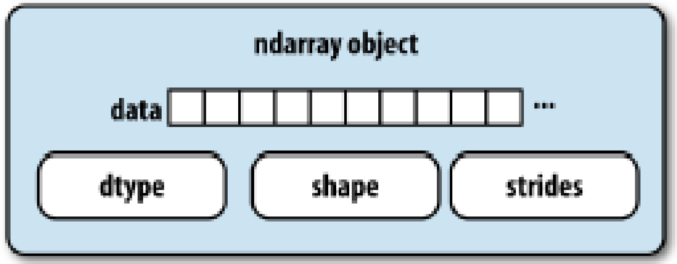
2 D ARRAY:

		col 0	col 1	col 2
	i \ j	0	1	2
row 0	0	A	A	A
row 1	1	B	B	B
row 2	2	C	C	C

column

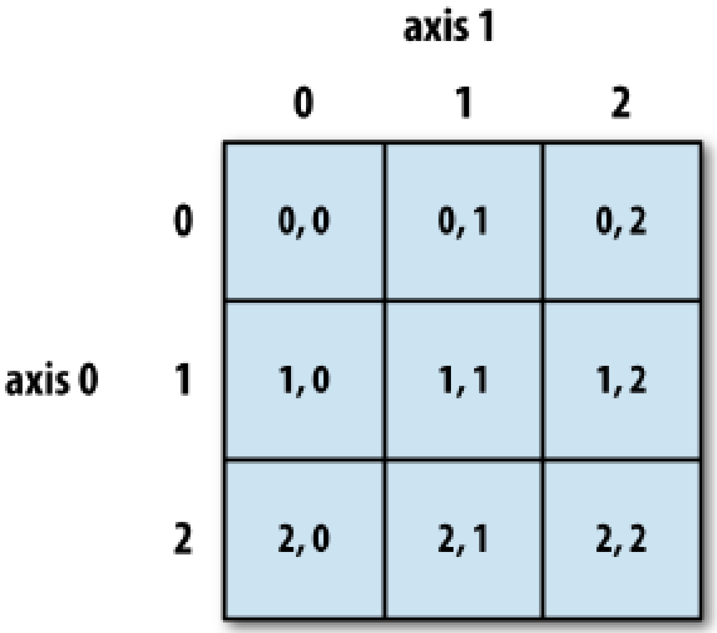
array elements

rows

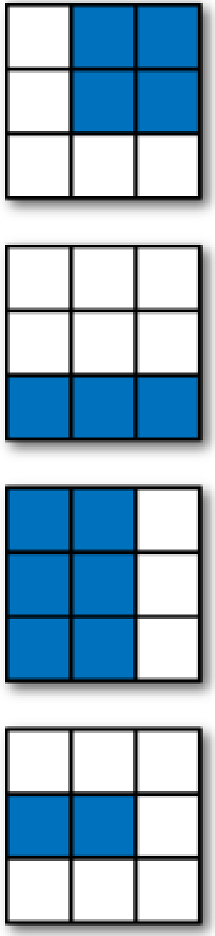


NumPy Array Examples

- Array indexing and shape



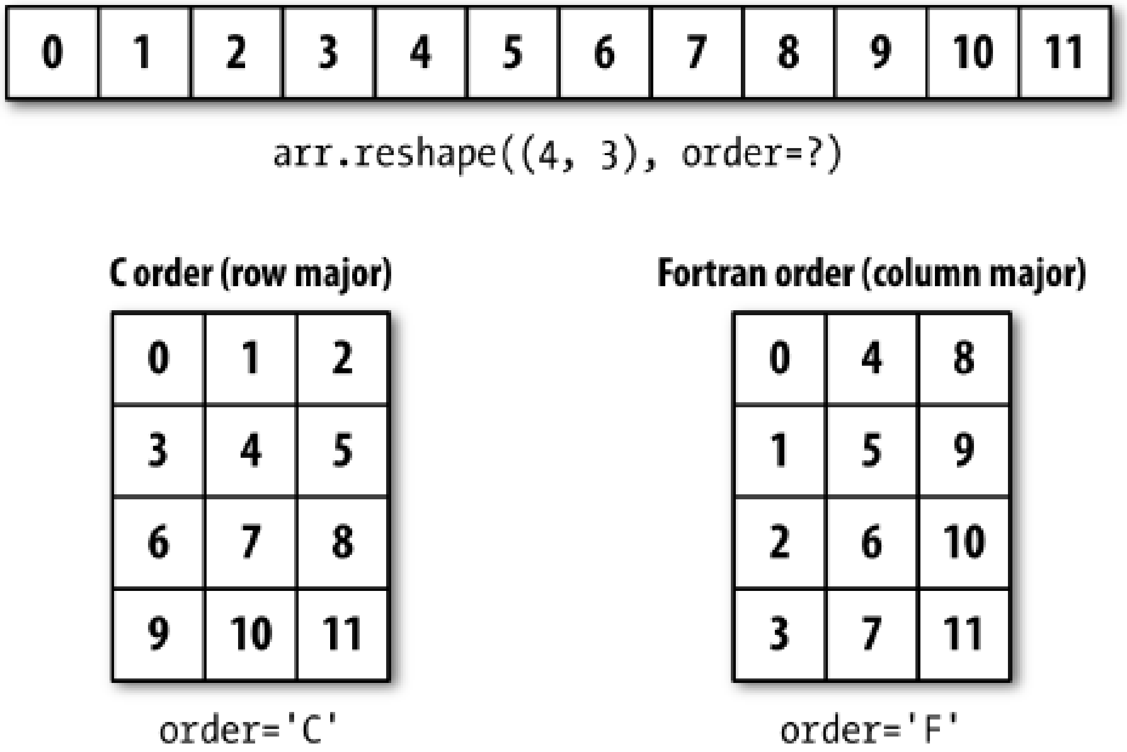
		axis 1		
		0	1	2
	0	0,0	0,1	0,2
axis 0	1	1,0	1,1	1,2
	2	2,0	2,1	2,2



Expression	Shape
arr[:2, 1:]	(2, 2)
arr[2] arr[2, :] arr[2:, :]	(3,) (3,) (1, 3)
arr[:, :2]	(3, 2)
arr[1, :2] arr[1:2, :2]	(2,) (1, 2)

NumPy Array Operations

- `.reshape`: change the dimension of arrays



0	1	2	3	4	5	6	7	8	9	10	11

```
arr.reshape((4, 3), order=?)
```

C order (row major)

0	1	2
3	4	5
6	7	8
9	10	11

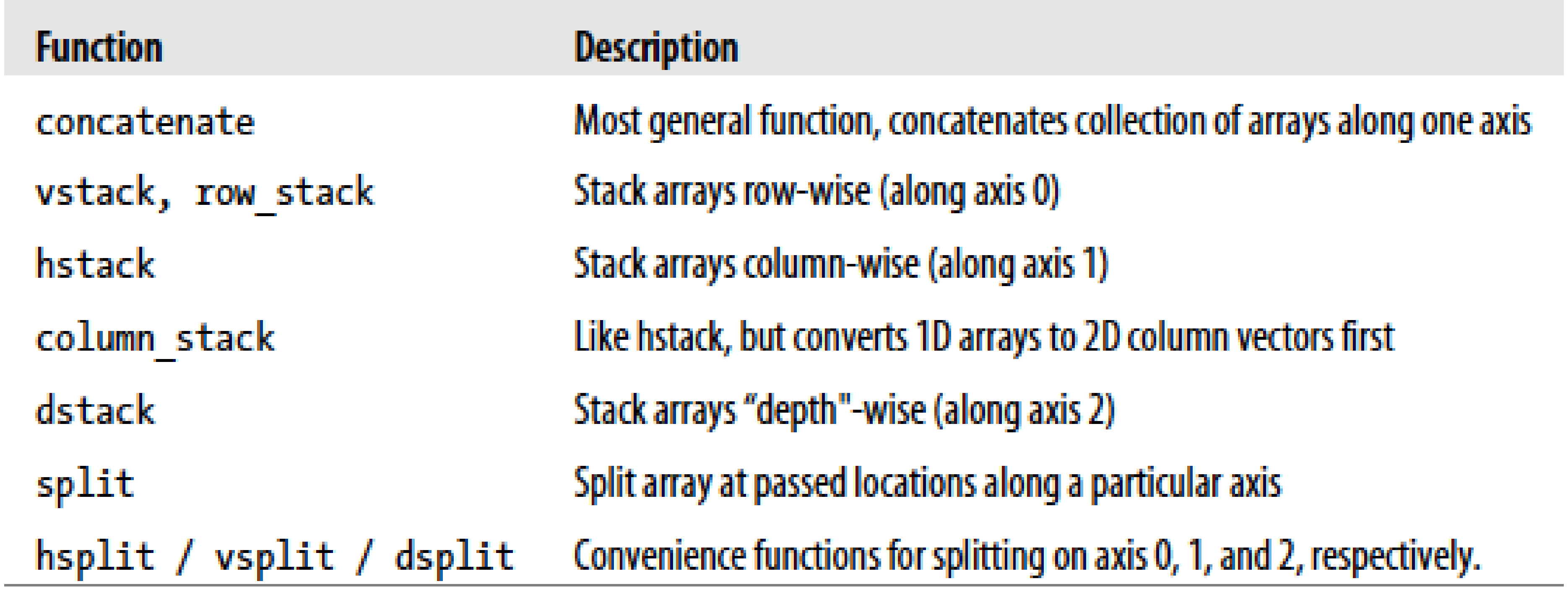
order='C'

Fortran order (column major)

0	4	8
1	5	9
2	6	10
3	7	11

order='F'

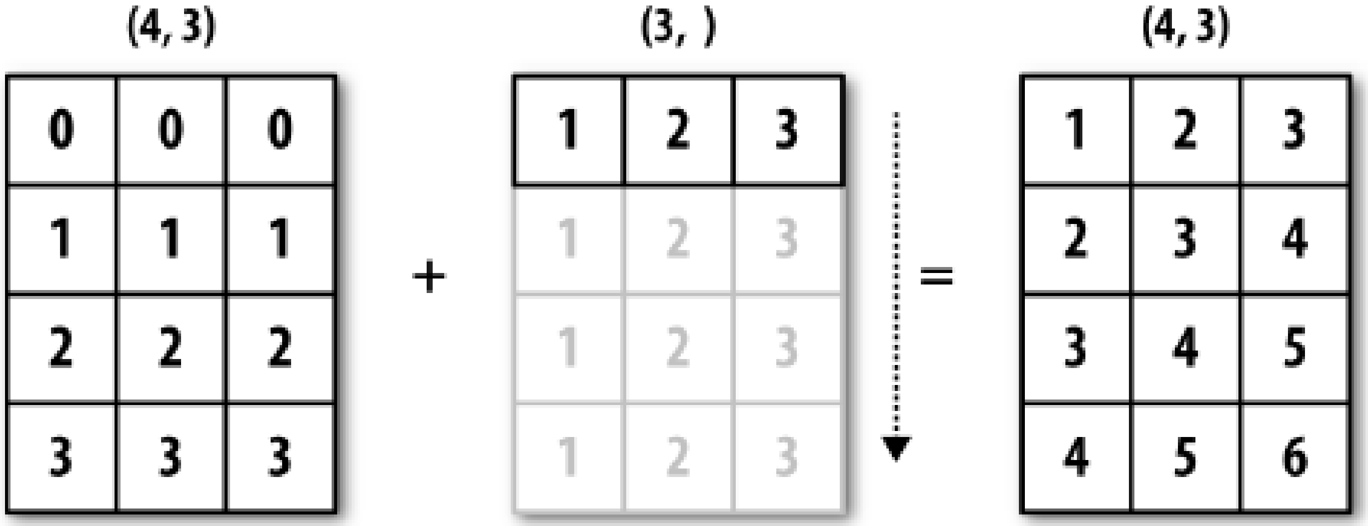
NumPy Array Operations



Function	Description
`concatenate`	Most general function, concatenates collection of arrays along one axis
`vstack, row_stack`	Stack arrays row-wise (along axis 0)
`hstack`	Stack arrays column-wise (along axis 1)
`column_stack`	Like hstack, but converts 1D arrays to 2D column vectors first
`dstack`	Stack arrays "depth"-wise (along axis 2)
`split`	Split array at passed locations along a particular axis
`hsplit / vsplit / dsplit`	Convenience functions for splitting on axis 0, 1, and 2, respectively.

NumPy Array Broadcasting

- Broadcasting describes how arithmetic works **between arrays of different shapes**.
- It is a very powerful feature, but one that can be easily misunderstood, even by experienced users.



(4, 3)

0	0	0
1	1	1
2	2	2
3	3	3

+

(3,)

1	2	3
1	2	3
1	2	3
1	2	3

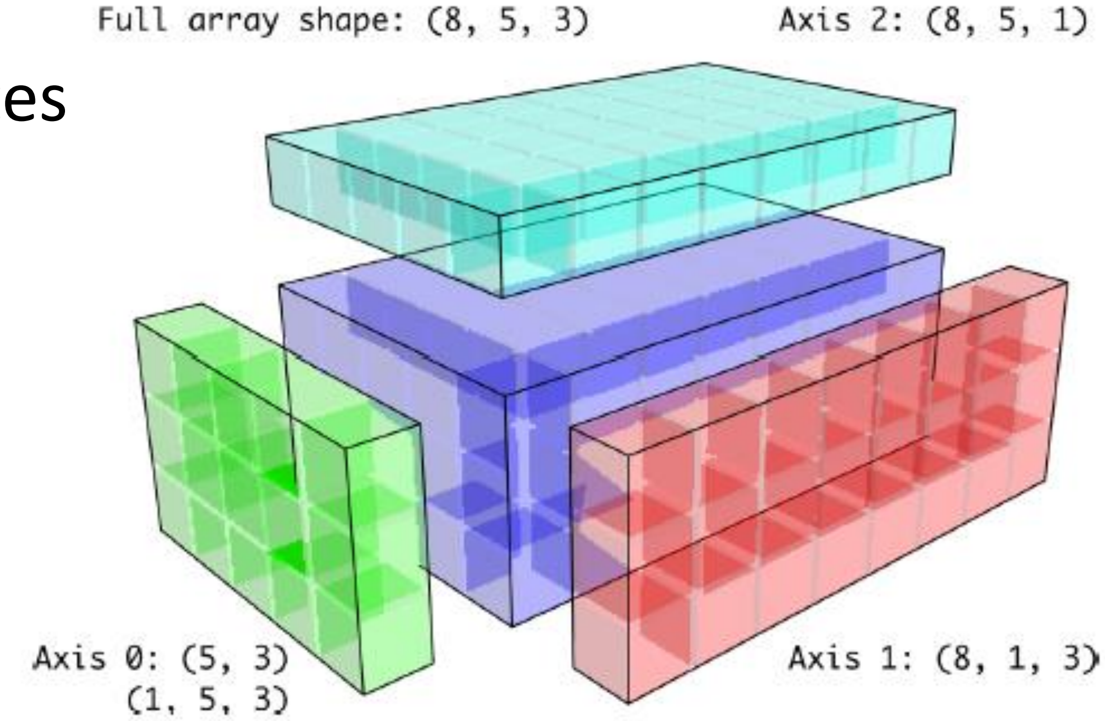
=

(4, 3)

1	2	3
2	3	4
3	4	5
4	5	6

NumPy Array Broadcasting

- Broadcasting can be applied to 3D arrays as well.
- We can easily calculate composites and anomalies of maps.

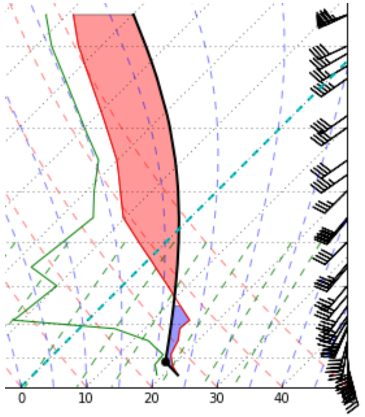


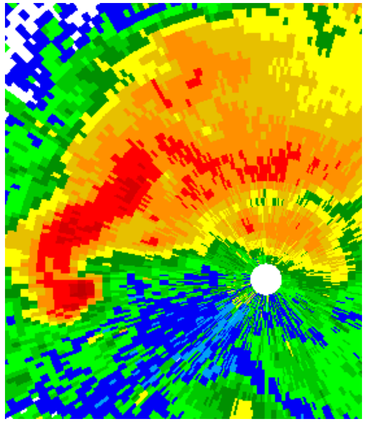
Pandas

- Pros:
  - Pandas provides powerful tools for processing **table-like data**.
  - Pandas adds **indexes and labels** to 1d and 2d NumPy arrays.
  - Easy handling of **missing data**.
  - Pandas **DataFrame** object can now be directly used by R.
- Cons:
  - However, most meteorological data is not in table-form.
- Bottom line:
  - We will use Pandas for its rich **data input/output** capability.
  - Pandas can serve as a good data cleaning tool.

Matplotlib

- Visualization is an important part of data analytics.
- MetPy
  - meteorology-focused plotting (e.g. skew-T, hodograph, station plot)
  - a growing list of meteorological calculations (e.g. advection, dewpoint, mixing_ratio, etc.)
  - reading common meteorological file formats (e.g. GINI satellite images, NEXRAD Level 2 and 3)
  - gridding and interpoloation tools
  - color table manipulation

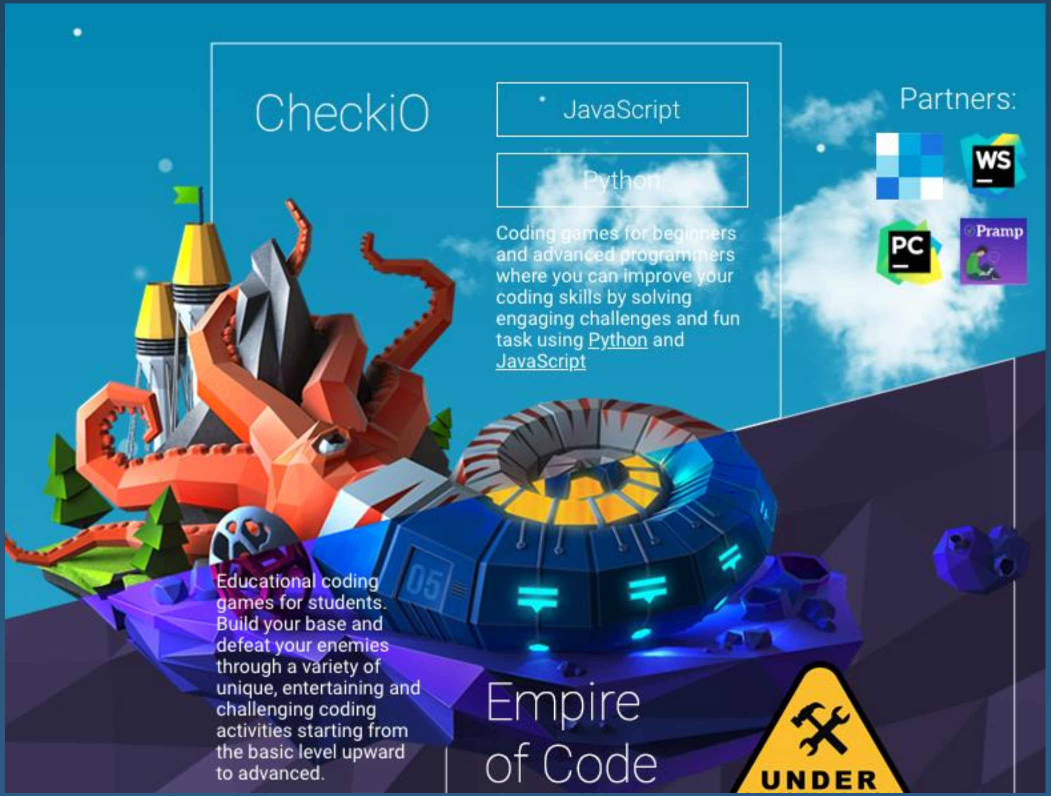




Exercises

- 練習範例
    - 107_week1_exercise.ipynb
- 習題
    - https://goo.gl/forms/JMu5PdM3DO3fxqtK2

Recommended Resources



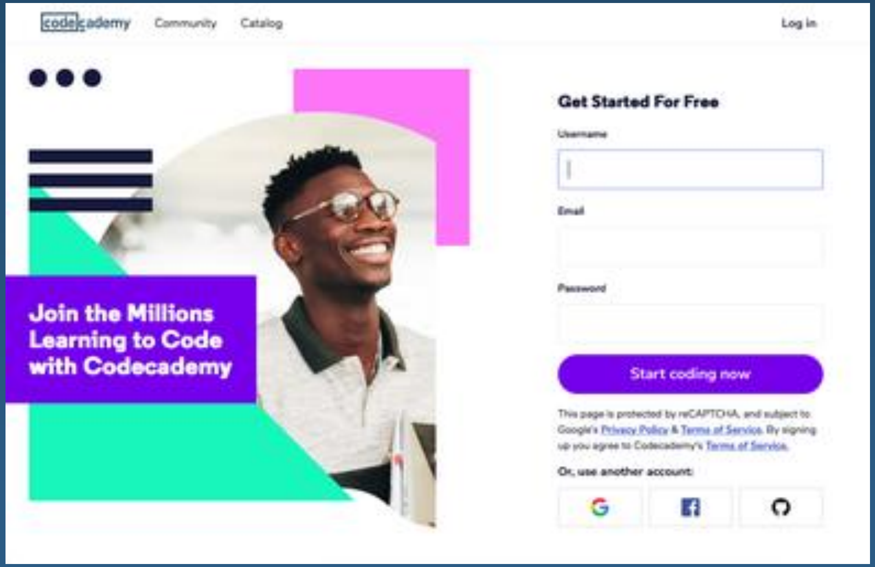
CheckiO

Coding games for beginners and advanced programmers where you can improve your coding skills by solving engaging challenges and fun task using Python and JavaScript

Educational coding games for students. Build your base and defeat your enemies through a variety of unique, entertaining and challenging coding activities starting from the basic level upward to advanced.

https://checkio.org/

codecademy



Join the Millions Learning to Code with Codecademy

Get Started For Free

Username

Email

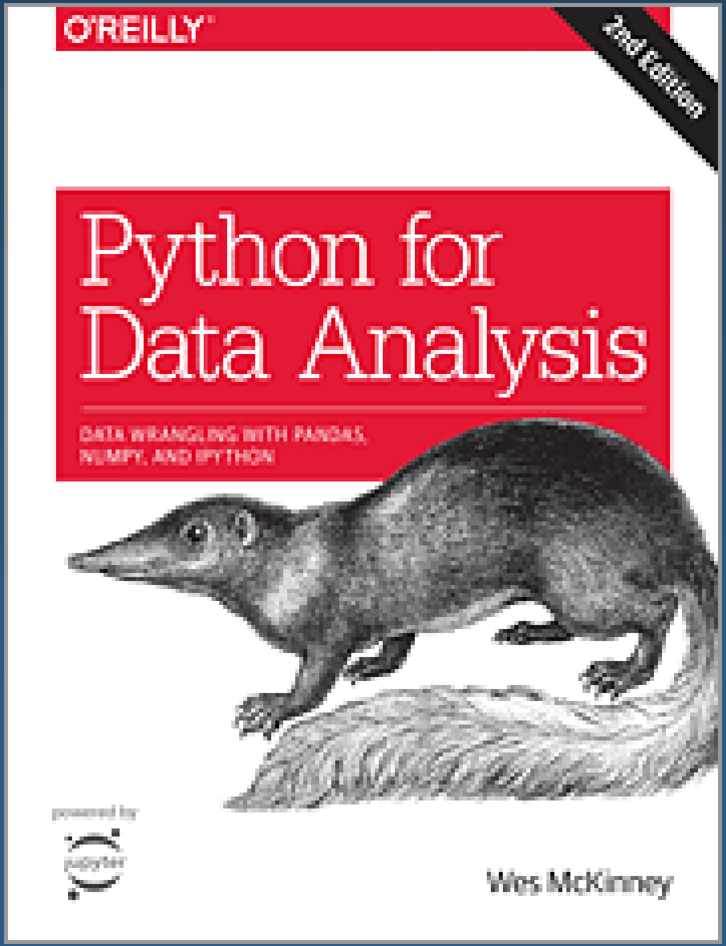
Password

This page is protected by reCAPTCHA, and subject to Google's Privacy Policy & Terms of Service. By signing up you agree to Codecademy's Terms of Service.

Or, use another account:

https://www.codecademy.com/

Recommended Resources



Python for Data Analysis, 2nd Edition

Data Wrangling with Pandas, NumPy, and IPython

By William McKinney

Publisher: O'Reilly Media
Release Date: October 2017
Pages: 550

Get complete instructions for manipulating, processing, cleaning, and crunching datasets in Python. Updated for Python 3.6, the second edition of this hands-on guide is packed with practical case studies that show you how to solve a broad set of data analysis problems effectively. You'll learn the latest versions of pandas, NumPy, IPython, and Jupyter in the process.

Next ▷

Python

Data Analysis

Atmospheric Sciences

Next ▷

Data Analytics