**Introduction of Python**

1. Exponentiation: \*\*. This operator raises the number to its left to the power of the number to its right: for example 4\*\*2 will give 16.
2. Modulo: %. It returns the remainder of the division of the number to the left by the number on its right, for example 18 % 7 equals 4.
3. variable and type:

  int, or integer: a number without a fractional part.

  float: a number with integer and a fractional parts = real number

        height = 1.75

string

bool: Ture or False; z = True, type(z) = bool

1. check the type: type(variable\_name); float stands for real number
2. string: can use both “” or ‘’; "av" + “vs” = “avvs” just combine the string together
3. bool:
4. str(): convert a value into a string
5. int(); float(); bool(): convert a value into an integer/float/bool
6. list: new\_list = [1, 2, 3, 4, 5]; a collection of values – possible to contain values with differen types

list = [4, ‘hello’]

1. list of lists: fam = [ ["mom", 1.75], ["dad", 1.80], ["sister", 1.67] ]
2. subsetting a list: my\_list = [1, 3, 5, 7, 10]

index from 0 for the fist element: my\_list[3] = 7

index -1, -2, -3… from the last element; my\_list[-2] = 7

list slicing: my\_list[2:4] = [5, 7]; list [start (inclusive) : end (exclusive)]

my\_list[2: ] = [5, 7, 10] include all element from 2nd position

my\_list[:] : return all elements in the list

my\_list[:2]: return element with index 0, 1

my\_list[2:]: return element with index 1, 2, 3, …

list1 = [ ["mom", 1.75], ["dad", 1.80], ["sister", 1.67] ]

list1 [-1][1] : return 1.67

1. list manipulation: list1 = [ "mom", 1.75, "dad", 1.80, "sister", 1.67 ]

change list element: list1[5] = 1.65; list1[4:] = [“brother”, 1.89]

adding element: list2 = list1 + ["me", 1.76]

deleting element: del() del(list1[0])

1. clearify:

x = ["a", "b", "c"]

y = x

y[1] = "z"

y = ['a', 'z', 'c']

x = ['a', 'z', 'c']

# both x and y changes, when use = sign, paste the reference

x = ["a", "b", "c"]

y = x[:]

y[1] = "z"

y = ['a', 'z', 'c']

x = ['a', 'b', 'c']

1. function

max(); min(); len(): length

round(1.68, 1) = 1.7; round(1.68) : default to the nearest integer

help(round)

sorted(numbers, reverse = False): increasing order

sorted(numbers, reverse = True): decreasing order

1. method: functions that belong to different type of objects

str: capitalize(); replace()

float:bit\_length(); conjugate()

list: index(); count()

ex: list1 = [ "mom", 1.75, "dad", 1.80, "sister", 1.67 ]

list1.index("mom")

list1.count(1.67): count the number of 1.67 in the list

list1.append("me") = [ "mom", 1.75, "dad", 1.80, "sister", 1.67 , "me"] : add element to a list

list1.reverse() : reverse the order of the element in the list

ex: name = "liz"

name.capitalize() = 'Liz'; capitalize the first letter in the string

name.replace("z", "sa") = 'lisa'

name.index("z") = 2

name.upper() = LIZ

1. packages: directory of Python Scripts, each script = module and specify functions, methods, and types

install package: <http://pip.readthedocs.io/en/stable/installing/>

import package:

import numpy: works for array; numpy.array([1, 2, 3]) (this import method is the clearest)

import numpy as np: np.array([1, 2, 3])

from numpy import array: array([1, 2, 3]), can directly call for array (this import method only use array functions from the package)

ex. Import math package to get pi: math.pi

from math import pi: pi

1. convert an angle in degrees to an angle in radians: radians()
2. list1 = [1, 2,3, 4]

list2 = [2, 3, 4, 5]

can not do the calculation for result = list1/list2

array1 = numpy.array([1,2,3])

array2 = numpy.array([2,5,3])

can do the calculation for result = array1/array2

convert list to numpy array the do the calculation

attention: numpy array can contain only one type

array1 = numpy.array([1, 2, 3])

list1 = [1, 2, 3]

list1 + list1 = [1, 2, 3, 1, 2, 3]

array1 + array1 = [2, 4, 6]

numpy array is an object which has its own methods (functions)

1. numpy array subsetting:

age = numpy.array([24, 35, 37, 59])

age[2] = 37

age > 36, the return is T or F

age[age>36], the return is [37, 59]: a way to subset an array

1. 2D numpy array: contain only single type

import numpy

this\_array = numpy.array([[1, 2, 3], [4, 5, 6]])

this\_array.shape: (2, 3): the array has 2 rows and 3 columns

this\_array[0]: select the first row

this\_array[0][1]: first row, 2nd element

another to subset: this\_array[0, 1]: first row, 2nd column

this\_array[: , 0:1]: all rows, first column; this\_array[: , 0:2]: all rows, first and 2nd columns

this\_array[1, :] : 2nd row, all columns

1. data analysis

np.mean(this\_array[: , 2]): mean for the 3rd column

np.median(this\_array[: , 2]): median for the 3rd column

np.std(this\_array[: , 2]): standard deviation for the 3rd column

np.corrcoef(this\_array[: , 2], this\_array[:, 1]): correlation coefficient between the 2nd column and 3rd column

1. generate data

import numpy as np

np.random.normal(1.75, 0.20, 1000) (mean, sd, sample size)

**Intermediate Python**

1. Basic plot: data visualization

Import matplotlib.pyplot as plt

plt.plot(x-var, y-var); plt.show() (line plot)

plt.scatter(x-var, y-var) (scatterplot)

change the scale of axis: plt.xscale('log'); plt.yscale('log')

histogram: import matplotlib.pyplot as plt: plt.hist(vectors, bins), the default: bins = 10

plt.clf(): cleans the plot up again can start afresh

1. Customization:

Label: plt.xlavel(“x-label-name”); plt.ylavel(“x-label-name”); plt.show()

Title: plt.title(“title-name); plt.show()

Range of y: plt.yticks([0, 2, 4, 6, 8, 10])

Label for each y coordinate point: plt.yticks([0, 2, 4, 6, 8], [‘0B’, ‘2B’, ‘4B’, ‘6B’, ‘8B’])

(the length of the two elements must be same

size: plt.scatter(x, y, s = var3), s = size, the size of the points vary by different var3 value

color: plt.scatter(x, y, s = var3, c = col)

opacity: alpha from 0 to 1; plt.scatter(x, y, s = var3, c = col, alpha = 0.8); 0: transparent;

1: not transparent

add text: plt.text(x-position, y-position, “text”)

add gridlines to the plot: plt.grid(True)

1. Dictionary

dictionary = {key1:value1, key2:value2, …}, dictionary[keyi], the return is value i

ex: world = {“Chin”a:200, “US”:170, …}

if want the number for China, use code:

world[“China”]

get all keys: dictionary. keys()

attention: the key of the dictionary must be unique: for repeat key, the value will be

recapped; keys have to be immutable objects, once created, cannot be changed

add more data to a dictionary: dictionary[“key-to-be-added] = value

check whether a key is in a dictionary: “keyi” in dictionary, i.e, “China” in world

(the return is T or F)

if want to update the value for a key that already exist, just add the key with a new value

again, since the key in a dictionary should be unique

delete a key from a dictionary: del(dictionary[“keyi”])

comparison between list and dictionary:

list: list.index() – select; update, del() – remove, indexed by range of numbers, access

values by index

dictionary: select, update, remove, indexed by unique keys, access values by keys

1. Tabular dataset: table: row: observation; column: variables

Attention: 2D numpy array can only contain single data type

pandas: high level data manipulation tool built on numpty package – dataframe

data frame can contain different data types

1. Data frame from dictionary:

ex. dic = {

“country”: [“China”, “Russia”, “Brazil”, “India”],

“capital”: [“Beijing”, “Moscow”, “Brasilia”, “New Delhi”],

“area”: [9.597, 1.221, 17.10, 8.516],

“population”: [200.5, 50.6, 800.5, 67,8] }

(keys are columns – variables; values are rows)

import pandas as pd

this\_dataframe = pd.DataFrame(dic)

this\_dataframe

add row name for a data frame: this\_dataframe.index = [“CH”, “RU”, “BR”, “IN”]

1. Data size is very large: data fram from CSV file: comma-separated valurs

this\_dataframe = pd.read\_csv(file path)

if the first column in the file contains the row index, add argument index.col = 0

this\_dataframe = pd.read\_csv(file path, index.col = 0)

1. Index and select data from a data frame

Column access:

this\_dataframe[“country”] to select the single column of country name

type (this\_dataframe[“country”]) the return is series: 1D labelled array

double brackets: type(this\_dataframe[[“country”]]) the turn is dataframe

sub-dataframe: this\_dataframe[[“country”, capital]])

Row access: row is indexed from 0, this\_dataframe[1:3], 2nd and 3rd row

(first number is inclusive, second number is exclusive)

Use loc function to access rows:

this\_dataframe.loc [“index.name”] the return is a series

this\_dataframe.loc[[“index.name”]] the return is a sub-dataframe

row and column access: this\_dataframe.loc[[“CH”, “BR”], [“country”, “capital”]]

(first specify index label then the variable names)

all rows and selected columns: this\_dataframe.loc[: , [“country”, “capital”]]

summary:

square brackets:

column access: this\_dataframe[[“country”, “capital”]]

row access: through slicing: this\_dataframe[1:3]

loc (label-based)

row access: this\_dataframe.loc[[“CH”, “BR”]]

column access: this\_dataframe.loc[:, [“country”, “capital”]]

row and column access: this\_dataframe.loc[[“CH”, “BR”] , [“country”, “capital”]]

iloc (index-based)

row access: this\_dataframe.loc[[0, 2]]

row and column access: this\_dataframe.loc[[0, 2] , [0, 1]]

column access: this\_dataframe.loc[; [0, 1]]

1. Comparison operation

‘ca’ < ‘cd’ the return is T, according to alphabet order

Comparison works only for data with same type

Boolean operators: and, or: , not: not True : the return is False

Numpy arrary: np.logical\_and(); np.logical\_or(); np.logical\_not();

np.logical\_or(my\_house > 18.5, my\_house < 10)

np.logical\_and(my\_house < 11, your\_house < 11)

1. If, elif, and else
2. If-else

if condition:

expression

else:

expression

ex:

z = 4

if z % 2 == 0:

print(“z is even”)

else:

print(“z is odd”)

1. elif

if condition:

expression

elif condition:

expression

else condition:

expression

ex.

z = 6

if z % 2 == 0:

print(“z is divisible by 2”)

elif z % 3 == 0:

print(“z is divisible by 3”)

else:

print(“z is neither divisible by 2 nor by 3”)

attention: if the “if” condition meet, never reach to the “elif” condition

the return for the above example is z is divisible by 2

1. filtering Pandas dataframe

dic = {

“country”: [“China”, “Russia”, “Brazil”, “India”],

“capital”: [“Beijing”, “Moscow”, “Brasilia”, “New Delhi”],

“area”: [9.597, 1.221, 17.10, 8.516],

“population”: [200.5, 50.6, 800.5, 67,8] }

import pandas as pd

this\_dataframe = pd.DataFrame(dic)

goal: choose countries with area larger than 9

this\_dataframe["area"] select the area column as a series

index = this\_dataframe["area"] > 9 the return is T or F

this\_dataframe[index]

1. while loop: repeated if the statement is true until the statement is unsatisfied

**while condition:**

**expression**

ex.

error = 50

while error > 1 :

error = error/4

print(error)

1. for loop: for each variable in the sequence, execute the expression

**for var in seq:**

**expression**

ex.

fem = [1, 2, 3, 4]

for var in fem:

print (var)

adding index before each print use enumerate:

fem = [1, 2, 3, 4]

for index, var in enumerate(fem)

print(“index” + str(index) + “: ” + str(var))

ex.

for index, c in enumerate('family'):

print("index" + str(index) + ": " + c)

for index, c in enumerate('family'):

print(c.capitalize())

1. loop data structures:
2. dictionary: us dictionary.items()

dic = {'china':200, 'us':150, 'egp':320 }

for key, value in dic.items():

print(key + " -- " + str(value))

attention: for k, v in dic.items():, the name of the k and v are arbitrary but the first stands

for the key and the second stands for the value

1. numpy array

import numpy as np

np\_height = np.array([1, 2, 3, 4, 5])

np\_weight = np.array([5, 4, 3, 3, 1])

com = np.array([np\_height, np\_weight])

(for 1D numpy array)

for var in np.nditer(np\_height):

print(var)

(for 2D numpy array, use np.ndier(array\_name) to print each element)

for var in np.nditer(com):

print(var)

1. data frame

dic = {

'country': ['China', 'Russia', 'Brazil', 'India'],

'capital': ['Beijing', 'Moscow', 'Brasilia', 'New Delhi'],

'area': [9.597, 1.221, 17.10, 8.516],

'population': [200.5, 50.6, 800.5, 67.8] }

dic\_df = pd.DataFrame(dic)

dic\_df.index = ['CH', 'RU', 'BR', 'IN']

(print each row with respect to the label)

for label, row in dic\_df.iterrows():

print(label)

print(row)

(selective print: only want print the capital for each row label)

for label, row in dic\_df.iterrows():

print(label + ": " + row["capital"])

(add column in a data frame)

(for all index label, add the column length)

for label, row in dic\_df.iterrows():

dic\_df.loc[label, "name\_length"] = len(row["country"])

dic\_df

(alternative way: add a column in the dataframe directly)

dic\_df["name\_length"] = dic\_df["country"].apply(len)

ex.

dic\_df ["COUNTRY"] = dic\_df ['country'].apply(str.upper)

(only string has the method upper())

1. random generators: need to import numpy and there is random package

np.random.rand(): return a number between 0 and 1

(this is pseudo-random numbers – depend on seed

np.random.seed(123)

coin = np.random.randint(0, 2): generate number 0 or 1 (2 is exclusive)

if coin == 0:

print(“head”)

else:

print(“tail”)

1. random walk: many random draws that ith outcome depends on i-1th outcome

ex. roll a coin, 0 for head and 1 for tail (not random walk since independence)

import numpy as np

np.random.seed(123)

outcomes = []

for x in range(10):

coin = np.random.randint(0, 2)

if coin == 0:

outcomes.append("head")

else:

outcomes.append("tail")

print(outcomes)

ex. (this is a random walk)

import numpy as np

np.random.seed(123)

tails = [0]

for x in range(10):

coin = np.random.randint(0, 2)

tails.append(tails[x] + coin)

print(tails)

1. max() function: return the larger one: max(10, 4) = 10
2. A. distribution:

import numpy as np

np.random.seed(123)

tails = [0]

for x in range(10):

coin = np.random.randint(0, 2)

tails.append(tails[x] + coin)

print(tails)

(if there are 100 runs- simulate the process for 100 times)

import numpy as np

np.random.seed(123)

final\_tails = []

for x in range(100):

tails = [0]

for x in range(10):

coin = np.random.randint(0, 2)

tails.append(tails[x] + coin)

final\_tails.append(tails[-1])

print(final\_tails)

(final\_tails store the number of tails for each of the 100 runs, each run has 10 random draws from 0 and 1)

1. visualize the distribution: histogram

plt.hist(final\_tails)

plt.show()

1. transpose an array：
2. np\_aw\_t = np.transpose(np\_aw)

**Importing data into Python**

1. read in plain text file (only read, cannot write)

# set the working dictionary first

cd '/Users/apple/Desktop'

filename = 'HonorCode.txt'

file = open(filename, mode = 'r') # 'r' is to read

text = file.read()

file.close()

print(text)

简化：

file = open('HonorCode.txt', mode = 'r') # 'r' is to read

print(file.read())

**or**

cd '/Users/apple/Desktop'

with open('HonorCode.txt', 'r') as file:

print(file.read())

**(the first way must close the file before print, the second way do not need to close the**

**file)**

1. write in plain text file

filename = 'HonorCode.txt'

file = open(filename, mode = 'w') # 'w' is to write

file.close()

1. command:

!: gives complete system shell access

! ls: display the contents of your current dictionary

1. check whether a file is close

print(file.closed)

1. print only few lines from a file: readline()

with open('HonorCode.txt', 'r') as file:

print(file.readline()) # print the first line

print(file.readline()) # print the second line

print(file.readline()) # print the third line

1. flat file: table data: row: record; column: variable/feature/attribute

header: the first row: recording the name of each variable: it is important to know whether the file has a header

file extension: .csv-comma separated by values

.txt-text file

1. import flat file:
2. numpy package: import as numpty array (require: all data are numerical)
3. if there is no header

cd '/Users/apple/Desktop'

import numpy as np

filename = 'numerical.txt'

data = np.loadtxt(filename, delimiter = ',')

data

1. if the file has a header and need to skip the first row

cd '/Users/apple/Desktop'

import numpy as np

filename = 'numerical.txt'

data = np.loadtxt(filename, delimiter = ',', skiprows = 1)

data

1. if want only the first and third columns in the data

cd '/Users/apple/Desktop'

import numpy as np

filename = 'numerical.txt'

data = np.loadtxt(filename, delimiter = ',', skiprows = 1, usecols = [0, 2])

data

1. customizing the type of numpy array imported

cd '/Users/apple/Desktop'

import numpy as np

filename = 'numerical.txt'

data = np.loadtxt(filename, delimiter = ',', skiprows = 1, dtype = str)

data

(import the array as strings)

1. “,” comma-delimited; “\t”: tab-delimited
2. dtype = float; dtype = str
3. import for mixed data type

import numpy as np

data = np.genfromtxt('titanic.csv', delimiter = ',', names = True, dtype = None)

(names = T: the file has a header; dtype = None: keep the original data type for each

variable)

print(data)

np.shape(data): the type of data is called structured array, Python treats it as a 1D

array, where each element of the array is a row of the file. Check the number of rows

by np.shape(data)

data[i]: index from 0 return the row

data[“column\_name”]: return the entire column

1. data = np.recfromcsv(file, delimiter = ',', names = True), this is similar for np.genfromtxt() but the default dtype is None

1. pandas: import as data frame – 2-dimensional labeled data structures
2. general import

cd '/Users/apple/Desktop'

import pandas as pd

filename = 'diamonds.csv'

data = pd.read\_csv(filename)

data.head() # check the first 5 rows for the data frame

# convert the dataframe to an array

data.array = data.values

data.array[0:3]

1. import without header and import selected rows

data = pd.read\_csv(file, nrows = 5, header = None)

(header = None: there is no header in the file)

1. more arguments

data = pd.read\_csv(file, sep='\t', comment='#', na\_values = 'Nothing')

(sep is same as the dilimitor; comment = “#”: treat everything after # as comment and

do not report it in the data frame; na\_values = “Nothing”, for all NA or NaN values in

the dataframe, display “Nothing”)

1. import other file types
2. pickled files: not human readable-serialize the data – convert the object into a sequence of bytes or byte stream; native file type in python

import pickle

with open('pickled\_fruit.pkl', 'rb') as file:

data = pickle.load(file)

print(data)

1. import from excel spreadsheet （use pandas: create dataframe natively）

cd '/Users/apple/Desktop'

import pandas as pd

file = 'this.xlsx'

data = pd.ExcelFile(file)

# an excel file may contains multiple sheets

# check the names of all sheets

print(data.sheet\_names)

# extract a certain sheet with function data.parse(sheet\_name as str or the index)

df1 = data.parse('ss')

df2 = data.parse(0)

(df1 = df2)

select the 1st sheet and customize it

df1 = data.parse(0, skiprows=1, names=['Country', 'AAM due to War (2002)'])

(skip the first row and rename the columns – should be a list type)

select the 2nd sheet first column, skip column name and rename it for “Country”

df2 = data.parse(1, parse\_cols=0, skiprows=1, names=['Country'])

1. explore the current working dictionary
2. ! ls
3. import os

wd = os.getcwd()

os.listdir(wd)

(os is a library)

1. import SAS and Stata files

# SAS file

import pandas as pd

from sas7bdat import SAS7BDAT

with SAS7BDAT('file\_name.sas7bdat') as file:

df\_sas = file.to\_data\_frame()

# connect the file to the SAS file

# Stata file

import pandas as pd

data = pd.read\_stata('file\_name,dta')

1. import HDF5 file: hierarchical data format version 5: storing for large quantities of numerical data

import h5py

filename = 'file\_name.hdf5'

data = h5py.File(filename, 'r')

print(type(data))

# the result should be hdf5 type

# check the structure of HDF5 files

for key in data.keys():

print(key)

# each of the key is a HDF5 file

# meta quality strain: three HDF5 files

print(type(data['meta']))

# the result should be a group

# for each key in a HDF5 file

for key in data['meta'].keys():

print(key)

# this print the key in the meta HDF5 file

# if the output keys are introduction detector duration

# access to values

print(data['meta']['introduction'].value, data['meta']['duration'].value)

1. import matlab files: .mat

use library: scipy

scipy.io.loadmat() – read .mat file

scipy.io.savemat() – write .mat file

import scipy.io

filename = 'file\_name.mat'

mat = scipy.io.loadmat(filename)

print(type(mat))

# the result is dictionary

# key: matlab variable names

# values: objects assigned to variables

print(type(may['variable\_name']))

# the result is numpty array

补充：

# print the keys of a dictionary: print(mat.keys())

# access the value of a key in a dictionary: print(type(mat['CYratioCyt'])) – should be numpy array

# get the shape of an array: np.shape(mat['CYratioCyt'])

1. rational database

based on relational model of data;

contains multiple tables but they are linked

ex. Northwind database: orders table; customers table; employees tables

orders table has columns: order ID, customer ID and employee ID – linked

SQL: Structured Query Language: fast and simple

SQLite database: file\_name.sqlite

1. creating a database engine in Python – connect to a database (ex SQL)

package: squlalchemy

from sqlalchemy import create\_engine

engine = create\_engine('sqlite:///Northwind.sqlite')

table\_names = engine.table\_names()

print(table\_names)

# this will print a list of table names in the database

# 'sqlite:///Northwind.sqlite' is called the connection string to the SQLite database Northwind.sqlite

1. SQL query: getting data from the database

1).

# SELECT \* FROM Table\_name

# return all columns of all rows of the table

SELECT \* FROM Orders

# the star after SELECT stands for all columns

1. Basic process:

Import packages and functions – create the database engine – connect to the engine – query the database – save query results to a data frame – close the connection

1. Coding process:

cd '/Users/apple/Desktop/'

# import packages

from sqlalchemy import create\_engine

import pandas as pd

# create engine

engine = create\_engine('sqlite:///Northwind.sqlite')

# connect to the database

con = engine.connect()

# query the database

rs = con.execute('SELECT \* FROM Orders')

# save the result in a data frame

df = pd.DataFrame(rs.fetchall())

# fix the column names: keep the column names as in the table

df.columns = rs.keys()

# close the connection

con.close()

# print the head of the saved dataframe

print(df.head())

alternative coding:

from sqlalchemy import create\_engine

import pandas as pd

# create engine

engine = create\_engine('sqlite:///Northwind.sqlite')

# connect to the database

with engine.connect() as con:

rs = con.execute('SELECT OrderID, OrderDate, ShipName FROM Orders')

# select specific columns

df = pd.DataFrame(rs.fetchmany(size = 5))

# fetchmany(size = 5) only import 5 rows instead of all rows

df.columns = rs.keys()

1. Adding WHERE

rs = con.execute('SELECT \* FROM Employee WHERE EmployeeId >= 6')

(Employee is the table name, only import the rows with variable EmployeeId >= 6)

1. Adding order

rs = con.execute('SELECT \* FROM Employee ORDER BY BirthDate)

(import all rows from table- Employee and order them in increasing order by BirthDate)

1. Alternative query method by using pandas function

from sqlalchemy import create\_engine

import pandas as pd

df = pd.read\_sql\_query('SELECT \* FROM Orders', engine)

# use the pd function read\_sql\_query()

# the 2nd argument in the function is the engine that you want to connect

1. Ex. connect WHERE and ORDER BY

df = pd.read\_sql\_query('SELECT \* FROM Employee WHERE EmployeeId >= 6 ORDER BY BirthDate’, engine)

1. Advanced querying: exploiting table relationships:
2. Joining tables: want one variable in table 1 and another variable in table 2

from sqlalchemy import create\_engine

import pandas as pd

engine = create\_engine('sqlite:///Northwind.sqlite')

df = pd.read\_sql\_query('SELECT variable\_1, variable\_2 FROM table\_1 INNER JOIN table\_2 on table\_1.commen\_variable = table\_2.common\_variable', engine)

print(df.head())

# two tables have the common variable and match it to get the selected variable 1 and 2