Course Introduction & Python Programming

Shunsuke Tsuda

BROWN

ECON 2020 Computing for Economists

Spring 2023

Prerequisite of ECON2020: Assignment 0

Assignment 0: Due at 9am on the 2nd Meeting Day

1. Gentzkow, Matthew and Jesse M. Shapiro 2014. "Code and Data for the Social Sciences: A Practitioner's Guide." (link) Read all.
2. Create GitHub (student discount version of Pro) account, install Git, and link Git to your GitHub account (Follow the guidance in the next slide).
3. After you have created your GitHub account, go to the invitation link sent via canvas and accept the assignment. Then, you will have your private repository for this assignment.
4. Follow the instructions inside there!

   After the instructor adds you to the "students" team, you will also have an access to all class materials.

   Note: We will use GitHub in the whole course for distributing and submitting your assignments. Git & GitHub are also very useful for managing your research project. We will come back to them in detail again in the software engineering part.

Setting up Git & GitHub

1. Create your GitHub account from here. Request the student free plan from here, which allows you to create a free private repository.
2. Install Git from here.
3. Open GitBash (in Windows) or the Terminal (in Mac).
4. When we use Git on a new computer for the first time, we need to configure a few settings, which will be used globally (i.e., for all projects).

   Link Git to your GitHub account: Set your username and email same as in your GitHub account.

```
$ git config --global user.name "stsuda"
$ git config --global user.email "shunsuke_tsuda@brown.edu"
```

Remark. This course uses a command line interface for operating Git.
There are also some graphical interfaces:
GitKraken/GitHub Desktop/RStudio (for R projects)

Two Objectives of This Course

Familialize yourself with basic concepts in

- **Software engineering**:
  Become familiar with the functions of a computer and learn how to use it wisely, write better code, and organize data nicely.
- **Scientific computation**:
  How to numerically solve problems that cannot be solved by hand.

Software Engineering: Motivation

- Most economists need computers for their research: design questionnaires; design experiments; scrape data; input raw data from field; clean data; transform data; merge data; execute statistical analyses; simulate models; format results; produce plots; write documentations; write up a draft; make a presentation

Software Engineering: Motivation

- Most economists need computers for their research: design questionnaires; design experiments; scrape data; input raw data from field; clean data; transform data; merge data; execute statistical analyses; simulate models; format results; produce plots; write documentations; write up a draft; make a presentation
- Most economists are *amateur* computer programmers without a formal training of computer science.

  That's fine. I'm an amateur, too.
- Most economists follow self-trained practices.

  That's where problems lie. (Pick a paper from one of top 5 economics journals and see its code.)
- Why not import some sophisticated ways of engineering softwares for improving our lives?

Principles of a Productive Research Practice

- **Portability**: Code should work in any machines without any changes.
  Techniques — Directories
- **Clarity & Maintainability**: Easy, direct, and straightforward to understand code. Easy to maintain and develop further any time.
  Techniques — Project management, Abstraction, Documentation
- **Accuracy**: Programs do what researchers intend. Design to easily detect if programs do what researchers did not intend.
  Techniques — Debugging, Unit test, Logging
- **Efficiency**: Write algorithms that conserve computing resources and save computing time.
  Techniques — Vectorization, Parallelization, High Performance Computing
- **Reproducibility**: Automate the whole research process. & Be able to reproduce any stage of research process.
  Techniques — Automation, Version Control

Scientific Computation: Motivation

- In the 1st-year coursework, you are trained to analytically solve a model to gain intuitive insights in a simple world
- Analytical tractability and simpleness are desirable features

Scientific Computation: Motivation

- In the 1st-year coursework, you are trained to analytically solve a model to gain intuitive insights in a simple world
- Analytical tractability and simpleness are desirable features
- At the same time, focusing only on such a tiny class of models decreases your research possibilitiies
- Many models do not have analytical solutions (or take infinite amount of time if computing by hand)
- Quantitative features are also important
    - Models should explain data not only qualitatively but also quantitatively
    - Need to simulate the model for this purpose

Scientific Computation: Motivation

- In the 1st-year coursework, you are trained to analytically solve a model to gain intuitive insights in a simple world
- Analytical tractability and simpleness are desirable features
- At the same time, focusing only on such a tiny class of models decreases your research possibilitiies
- Many models do not have analytical solutions (or take infinite amount of time if computing by hand)
- Quantitative features are also important
  - Models should explain data not only qualitatively but also quantitatively
  - Need to simulate the model for this purpose
- Cover numerical differentiation and integration, nonlinear equation-solving, and numerical optimization

Computers are fast, but not as wise as humans

- Numerical computations are not just the magic to implement things that human cannot do by hand.
- Always risky: we cannot directly check if a solution obtained by a computer is correct.
- Accuracy of results depends on human's understanding of limitations of computers.

Computers are fast, but not as wise as humans

- Numerical computations are not just the magic to implement things that human cannot do by hand.
- Always risky: we cannot directly check if a solution obtained by a computer is correct.
- Accuracy of results depends on human's understanding of limitations of computers.
- Choose and apply appropriate numerical methods to solve problems.
- Understand trade-offs between various numerical methods.
- Various applications from economics research will illustrate these points in class.

Worth Investing

- Recent development of computers and algorithms, and diversification of data sources are remarkable.
- Examples include, but do not limit to:
  Use satellite imageries to measure urban road congestion, environment damage in tropical forest, infrastructure damage by civil war, housing quality in a slum, etc;
  Elicit internal ideology of politicians from text and/or voice data;
  Use mobile phone metadata to infer social networks and migration;
  Develop smartphone app to collect GPS coords of commuter trips; Scrape messages of terrorist organizations from a dark web;
  ...
- Solid programming, software engineering, and computational skills are becoming more and more valuable for increasing your research possibilities!

Summary

This course familializes basic concepts in

- **Software engineering**:
  Use computer wisely; Write better code; Organize data nicely
  - Improve portability, clarity, maintainability, accuracy, efficiency, and reproducibility of economics research projects
- **Scientific computation**:
  How to solve problems that cannot be solved by hands.
  - Numerical differentiation and integration; Equation solving; Numerical optimization
  - Applications in economics research

Though we use Python, these concepts are **language-agnostic**.

Python Basics

Acknowledgement and References

I deeply thank Guixing Wei for sharing his lecture notes, on which this lecture is partly based.

This lecture also uses the following resources:

- QuantEcon
- Python for Economists by Alex Bell

Other references (can use them as dictionaries):

- Introducing Python
- Python for Data Analysis

Preliminary Remarks

- Do not spend too long time on sitting down with a textbook to study Python itself.
- Rather, find and implement tasks which directly connect to your research projects using Python.
- When doing your own research, you have to teach yourself a lot of things.
- Make a good habit of resolving technical questions by yourself, by googling, Python Documentation, Stack Overflow, etc.
- A bad habit is to ask a friend/TA/lecturer: "What does this error mearn? How can we resolve it?"
  $> 99\%$ of errors can be resolved by googling.
- We focus on Python 3.x, not 2.7. See, for example, this for the difference between Python 2 and 3.

Use Your Knowledge Wisely!

- Knowledge of one programming language helps to learn another programming language.
- Different programming languages can execute similar operations.
- If you have experiences in other languages and if you have in mind what you want to execute, then some cheatsheets of translations across languages help a lot:
  - Stata↔Python; Stata↔Pandas
  - Matlab/Julia↔Python; Matlab↔Numpy

Overview

- Why Python?
- Integrated Development Environment (IDE) & Spyder
- Data types & Basic Operations
- Storage: RAM vs Disk
- Flow Control: Conditional Statements & Loops
- Functions
- Modules & Packages
  - Develop and Import your Own Modules
  - NumPy & Pandas
  - Regular Expressions
  - Other Useful Packages (for Economists)
- Floating-Point Arithmetic
- Object-Oriented Programming

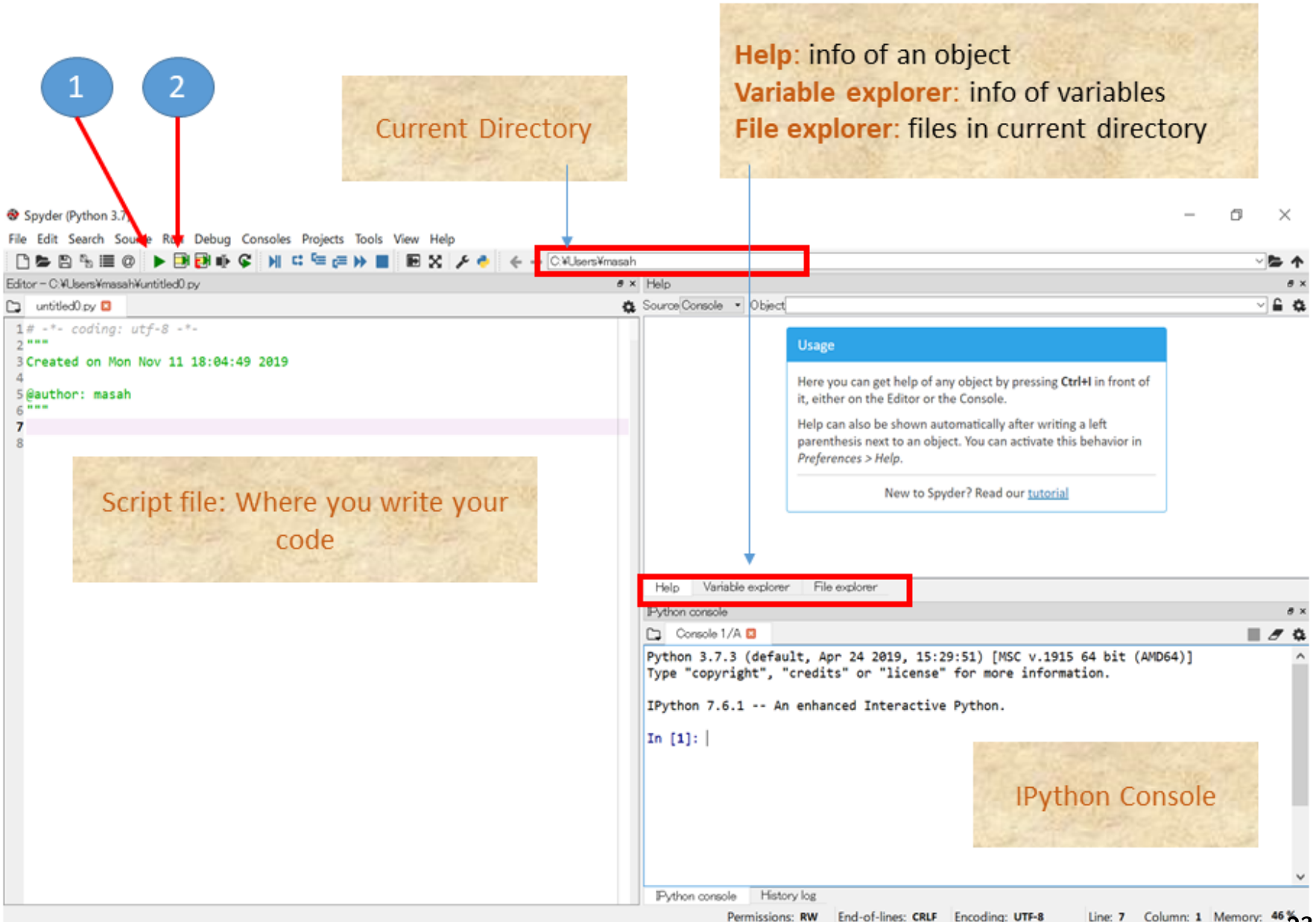
Why Python?

- A growing number of people use Python!
  - The Incredible Growth of Python
  - Becoming the world's most popular coding language
- High readability: the code looks very close to how humans think.
- Free, flexible, and a lot of packages for a wide range of purposes are being developed.
- Python can do most things that Stata/Matlab can do.
- Suitable for handling with large-scale datasets.
- High speed of numerical computation:
  Comparison across numerical computing language in GARCH
- But not necessarily the best for many tasks!

IDE & Spyder

- Integrated Development Environment (IDE): a program which integrates several tools, such as text editor, debugger, search for libraries, etc.
- Spyder: a free IDE which comes with Anaconda.
- Some features of Spyder:
  - Variable explorer
  - Offers built-in integration with many popular scientific packages
  - IPython console
- There are also many other popular IDEs (e.g., PyCharm; IDLE, etc).
- Jupyer Notebook, a popular web-based application of organizing codes, will also be used in later sections.

What Spyder looks like

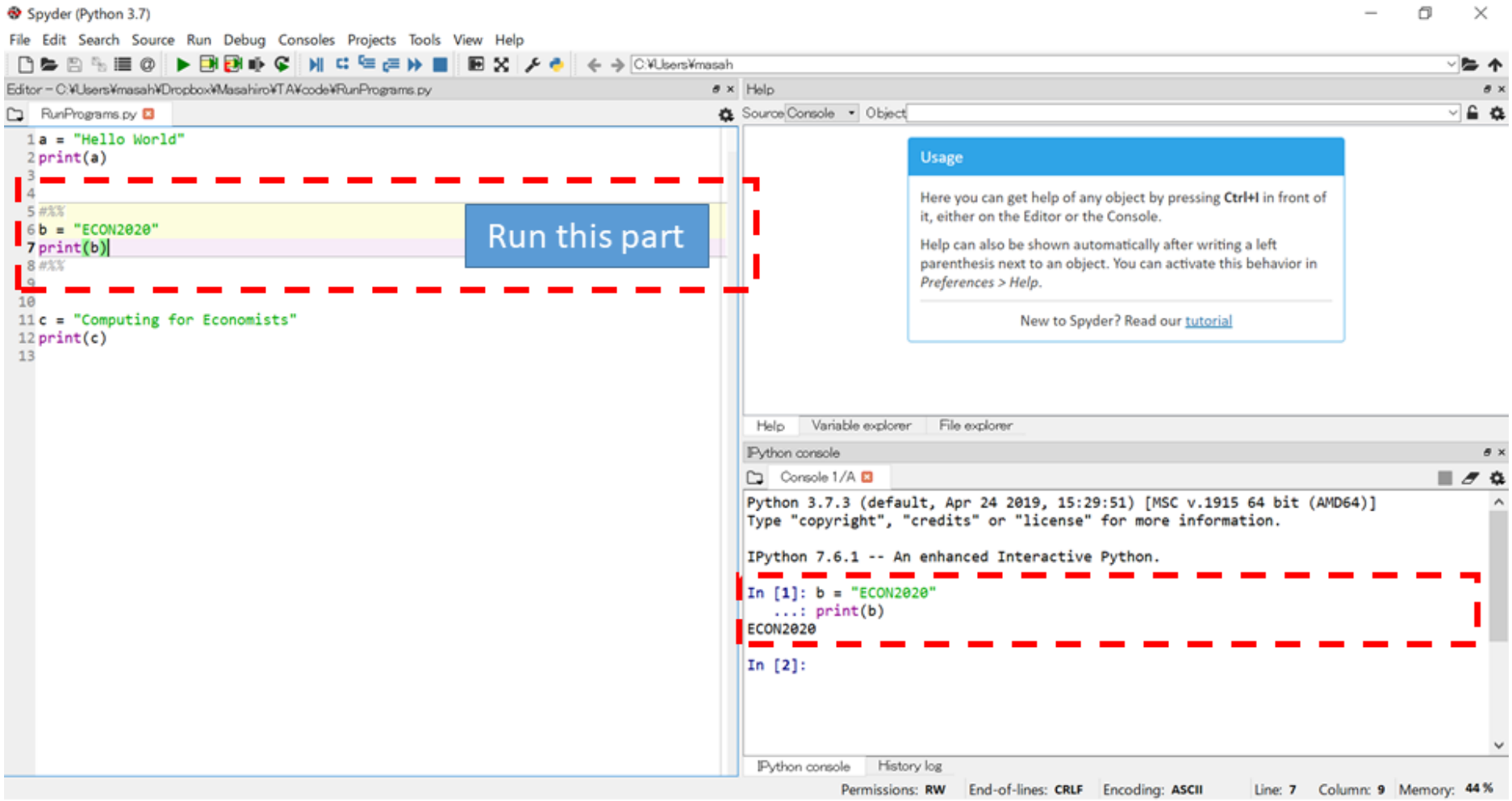


Programs and Run

- In Assignment 0, you have already written your first program in a script file.

- How to run your code in Spyder:
  - Run a whole script file: click ① in the previous slide.
  - Run a part of your code: enclose a part with "#%%" and click ② in the previous slide. In the next slide, we will show how this looks like.

Run the Part of Code



IPython Console in Spyder

Alternatively, you can type inputs directly and see outputs in IPython console.

```
# We use Python 3.X.
In [9]: 2**3
Out[9]: 8
In [10]: 2-4
Out[10]: -2
In [11]: 8/5
Out[11]: 1.6
In [12]: print("Hello")
Hello
In [13]:cd  #current directory
C:\Users\stsuda
```

Data Types

Python has four basic data types:

- **Boolean**: True, False
- **Number**: Integer, Floating, Complex
- **String**: Sequence of letters
- **Collection**: List, Tuple, Dictionary, Set, and Array (NumPy)

Boolean, Number, and String

```
# Boolean
In : e = 1 == 0
...: e
Out: False
In : type(e)
Out: bool

# Integer
In : c = - 3
In : type(c)
Out: int

# Float
In : o = 123.456
In : type(o)
Out: float

# String
In : n = 'econ'
In : type(n)
Out: str
```

Basic Operations – Basic Arithmetic

```
# a = 10, b = 3, c = False, d = True
In : a,b,c,d = 10,3,1==0,1>0

In : a+b #addition
Out: 13
In : a-b #subtraction
Out: 7
In : a*b #multiplication
Out: 30
In : a/b #division
Out: 3.3333333333333335
In : a**b #exponentiation
Out: 1000
```

Basic Operations – Boolean, Numbers, and Strings

```
# Boolean
In : bool() # Give the boolean value "False" without any values
Out: False
In : bool("")
Out: False
In : bool(a) # Give the boolean value "True" for any values
Out: True
In : bool("econ")
Out: True

# Boolean + Numbers
In : a + c # (False = 0 or 0.0)
Out: 10
In : a + d # (True = 1 or 1.0)
Out: 11

# Strings
In : str(a)
Out: '10'
In : 'Brown-' + 'econ'
Out: 'Brown-econ'

# Numbers + Strings (ERROR!)
In : a + 'Brown'
Traceback (most recent call last):

  File "<ipython-input-2-49615062b033>", line 1, in <module>
    a + 'Brown'

TypeError: unsupported operand type(s) for +: 'int' and 'str'

In : str(a) + "Brown"
Out: '10Brown'
```

Collection Types – List & Tuple

- **List**: comma-separated elements enclosed by the square brackets (**mutable**: can be modified)

```
# Empty list
In : a = []
In : type(a)
Out: list

# Nonempty list
In : a = [1, 2, 3, 4]
...: a
Out: [1, 2, 3, 4]
```

- **Tuple**: comma-separated elements enclosed by the parentheses (**immutable**: cannot be modified)

```
# Empty tuple
In : b = ()
In : type(b)
Out: tuple

# Nonempty tuple
In : b = (1, 2, 3, 4)
...: b
Out: (1, 2, 3, 4)
```

Collection Types – Dictionary & Set

- **Dictionary**: consists of key-value pairs enclosed by curly brackets (**mutable**)

```
# Empty dictionary
In :c = {}
In :type(c)
Out:dict

# Nonempty dictionary
In : c = {1: "Apple", 2: "Orange"} #keys: 1, 2 & their values: "Apple", "Orange".
...: c
Out: {1: 'Apple', 2: 'Orange'}
```

- **Set**: an unordered collection of distinct elements (**mutable**)

```
# Empty set
In :d = set()
In :type(d)
Out:set

# Nonempty set
In : d = set([1,2,3,3])
...: d
Out: {1, 2, 3}
```

Operations of List – Indexing/Slicing/Editing

```
In : e = [1,1,2,3,5,8,13,21,34]
...: len(e)  # Get the length of the list
Out: 9
# Extract elements or a part of the list
In : e[0]  # Indexing starts at 0, NOT 1!
Out: 1
In : e[-1] # Indexing starts at -1 backwards!
Out: 34
In : e[3:6] # Get a list with the index from 3 to 5
Out: [3, 5, 8]
In : e[:] # Get a whole list
Out: [1, 1, 2, 3, 5, 8, 13, 21, 34]

# A dot calls a method (function) bundled into an object.
# In the case below, the method is a function "index()" and an object is a list "e".
# We will cover this point in the Objected Oriented Programming (OOP) section later.
In : e.index(21) # Return the index of the value 21
Out: 7
In : e.index(1) # Return the first index if we have duplicate values.
Out: 0

# Edit elements inside the list
In : e[2] = 4 # Change the element with index 2
...: e
Out: [1, 1, 4, 3, 5, 8, 13, 21, 34]
In : e[1:3] = ['brown', 'econ'] # Change the elements with the index from 1 to 2.
...: e
Out: [1, 'brown', 'econ', 3, 5, 8, 13, 21, 34]
In : e[1:3] = [1, 2, 2]  # Change the elements with the index from 1 to 2.
...: e
Out: [1, 1, 2, 2, 3, 5, 8, 13, 21, 34]
In : del e[3]  # Delete the element with the index 3.
...: e
Out: [1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Operations of List – Concatenation & Repetition – List of Lists

```
# Concatenation
In : a = [1,2,3]
...: b = [4,5,6,7,8]
In : a + b
Out: [1, 2, 3, 4, 5, 6, 7, 8]
In : b + a
Out: [4, 5, 6, 7, 8, 1, 2, 3]

# Repetition
In : econ = ["Brown", "Econ", "Dept"]
...: rept = econ*3
...: rept
Out: ['Brown', 'Econ', 'Dept', 'Brown', 'Econ', 'Dept', 'Brown', 'Econ', 'Dept']

# Creating a list containing [major, student] pair lists
In : major_student_pair = []
...: major    = ["Political Economy", "Development Economics", "Game Theory"]
...: student  = ["An", "Ken", "Shun"]
...: major_student_pair = [[major[0], student[0]], [major[1], student[1]], \
...:                       [major[2], student[2]]  # (Line continuation by \)
...: major_student_pair
Out:
[['Political Economy', 'An'], ['Development Economics', 'Ken'], ['Game Theory', 'Shun']]

In : major_student_pair[1]
Out: ['Development Economics', 'Ken']

In : major_student_pair[1][0]
Out: 'Development Economics'
```

Operations of List – Some Methods for Lists

```
In : e = [1,1,2,3,5,8,13,21,34]
In : e.append(55) # Append 55 to the list
...: e
Out: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

In : a = [3, 2, 3]
...: b = [101, 100, 99]
In : a.count(3) # Count the number of 3
Out: 2

In : a.sort() # Sort a
...: a
Out: [2, 3, 3]
In : x = sorted(b) # x is sorted b
...: x
Out: [99, 100, 101]

In : a.extend(b) # Append elements of b to a
...: a   # See the difference from a + b. Here, a itself has changed!
Out: [2, 3, 3, 101, 100, 99]

In : a.append(b)  # Append b to the above
...: a # Now, list b comes inside list a
Out: [2, 3, 3, 101, 100, 99, [101, 100, 99]]
```

Operations of Dictionary – Keys, Values, and Items – Membership – Remove & Update

```
# Create a dictionary from a list
In : isocnt  = [['SEN', 'Senegal'], ['NGA', 'Nigeria'], ['MAR', 'Morocco']]
...: dictiso = dict(isocnt)
...: dictiso
Out: {'SEN': 'Senegal', 'NGA': 'Nigeria', 'MAR': 'Morocco'}
In : len(dictiso)   # Get Length of dictionary
Out: 3
In : dictiso['SEN']   # Indexing
Out: 'Senegal'
In : dictiso.keys()   # Get all the keys
Out: dict_keys(['SEN', 'NGA', 'MAR'])
In : dictiso.values()   # Get all the values
Out: dict_values(['Senegal', 'Nigeria', 'Morocco'])
In : dictiso.items()   # Get the all the items
Out: dict_items([('SEN', 'Senegal'), ('NGA', 'Nigeria'), ('MAR', 'Morocco')])

In : 'NGA' in dictiso.keys()   # Whether a key is in a dictionary
Out: True
In : 'Nigeria' in dictiso.values()   # Whether a value is in a dictionary
Out: True
In : ('NGA', 'Nigeria') in dictiso.items()   # Whether an item is in a dictionary
Out: True

In : dictiso.clear()   # Remove all all the elments of a dict object
...: dictiso
Out: {}
In : dictiso1  = {1: 'Senegal', 2: 'Nigeria', 3: 'Morocco'}
...: dictiso2  = {2: 'Senegal', 4: 'China', 5: 'Japan'}
...: dictiso1.update(dictiso2)   # Update a dict object using another dict object
...: dictiso1
Out: {1: 'Senegal', 2: 'Senegal', 3: 'Morocco', 4: 'China', 5: 'Japan'}
```

Operations of Set

```
In : e = [1,1,2,3,5,8,13,21,34]
...: f = set(e)  # Create set objects from a list
...: f
Out: {1, 2, 3, 5, 8, 13, 21, 34}
In : g = {1,1,2,3,5,8,13,21,34}  #Create set objects
...: g
Out: {1, 2, 3, 5, 8, 13, 21, 34}
In : len(g)  # Length of a set object
Out: 8
In : 8 in g  # Whether 8 is in g or not.
Out: True

In : a = {1,2,3,4,5}
...: b = {5,6,7,8}
...: c = a | b  # Union of two sets
...: c
Out: {1, 2, 3, 4, 5, 6, 7, 8}
In : c = a & b  # Intersection of two sets
...: c
Out: {5}
In : c = a ^ b  # Get elements in either set but not in both
...: c
Out: {1, 2, 3, 4, 6, 7, 8}
In : c = a - b  # Get the elements in set a but not in set b
...: c
Out: {1, 2, 3, 4}
In : c.add(8)  # Add & discard an element from a set object
...: c
Out: {1, 2, 3, 4, 8}
In : c.discard(8)
...: c
Out: {1, 2, 3, 4}
In : c.clear()  # Clear all elements
...: c
Out: set()
```

Operations of Tuple – Packing & Unpacking

```
# A comma defines a tuple.
In : a = 1,2,3
...: a
Out: (1, 2, 3)

# A tuple with a single element.
# Remind that a comma defines a tuple.
In : b = 1,
...: b
Out: (1,)

# What happen?
In : c = (1)
...: c # integer
Out: 1

# Packing
In : d = (3,4,5) # same as d = 3,4,5
...: d
Out: (3, 4, 5)

# Unpacking
In : one, two, three = d
...: two
Out: 4
```

Operations of Tuple – Indexing & Slicing

```
# Indexing and Slicing
# Same as list operations.
In : d = (3,4,5)
...: d[0]
Out: 3

In : d[-1]
Out: 5

In : d[1:2]
Out: (4,)

In : d[:2]
Out: (3, 4)

In : d[:]
Out: (3, 4, 5)

In : d[-2:]
Out: (4, 5)

In : d.index(3)
Out: 0

# Try to modify an element
In : d[0] = 9 # ERROR because tuple is immutable!
TypeError: 'tuple' object does not support item assignment

# Clear tuple
In : del d
```

When We Need Tuple

- List and tuple look similar, but tuple is more useful than list in terms of
  - immutability: when you would not like to change any element by accident
  - memory-saving
  - dictionary keys

```
In : import sys   # System-specific parameters and functions

In : List  = [1]*1000
...: Tuple = (1,)*1000

In : sys.getsizeof(List)  # Memory size of list
Out: 8064
In : sys.getsizeof(Tuple) # Memory size of tuple
Out: 8048

# As dictionary keys
In : dictiso = dict([[(1,), 'Senegal'], [(2,3,4), 'Nigeria'], [5, 'Morocco']])
...: dictiso # Tuple can become a dictionary key.
Out: {(1,): 'Senegal', (2, 3, 4): 'Nigeria', 5: 'Morocco'}

In : dictiso = dict([[[1], 'Senegal'], [[2,3,4], 'Nigeria'], [5, 'Morocco']])
...: dictiso # List cannot!
TypeError: unhashable type: 'list'
```

RAM vs. Disk

- RAM:
  - Temporary storage. Once you close Python, all variables and data disappear.
  - Faster processing.
- Disk:
  - Permanent storage. Even if you close Python, all variables and data do not disappear.
  - Slower processing.

Writing & Reading

```
# Writing to RAM (This is what we have done so far.)
In : a = 'Line1: This is test A.'
...: b = 'Line2: This is test B.'

# Writing to Disk
# "w" claims writing.
# If "testfile.txt" already exists in a current directory (cd), the file is overwritten.
# If not, the file "testfile.txt" is generated.
In : file = open(r"testfile.txt","w")  # open the file
...: file.write('Line1: This is test A.\n') # \n claims starting a newline.
...: file.write('Line2: This is test B.')
...: file.close() # Must close the file # Check the file in cd.

# Alternative way: close the file automatically.
In : with open(r"testfile.txt","w") as f: # open the file
...:     f.write('Line1: This is test A.\n')
...:     f.write('Line2: This is test B.')


# Reading vars in RAM (This is what we have done so far.)
In : a
Out: 'Line1: This is test A.'

# Readig a file in Disk
# "r" claims reading.
# "testfile.txt" must exist in the cd.
In : file = open(r"testfile.txt","r")  # open the file
...: file.read()
...: file.close() # Must close the file

# Alternative way: close the file automatically.
In : with open(r"testfile.txt","r") as f: # open the file
...:     f.read()
```

Counting 100 thousands: RAM vs. Disk

```
import time
# RAM
In : start = time.time()
...: a = 0
...: for i in range(1,100001):
...:     a += 1
...: end = time.time() - start
...: print ("elapsed time:{0}".format(end) + "[sec]")
elapsed time:0.003101825714111328[sec]

# Writing a file and store a counter var
In : counter = r"counter.txt"
...: with open(counter,"w") as f: # open the file
...:     f.write(str(0))

# Disk
In : start = time.time()
...: for i in range(1,100001):
...:
...:     with open(counter,"r") as f:
...:         a = int(f.read())
...:
...:     with open(counter,"w") as f:
...:         f.write(str(a + 1))
...:
...: print("Count to {0}!".format(a + 1))
...: end = time.time() - start
...: print ("elapsed time:{0}".format(end) + "[sec]")
elapsed time:52.164246559143066[sec]
```

Flow Control

- You may want to write code block based on conditions/criteria.
- You may want to apply the same operation element by element.

- Conditional statement: If statement

- Loops
  - For-loops
  - While-loops

If-statement

```
# Python needs indentation!
In : a = 2
...: b = 4
...: if a*2 > b:
...: print("a*2 > b")
...: elif a*2 < b:
...: print("a*2 < b")
...: else:
...: print("a*2 = b")
  File "<ipython-input-56-21036570a311>", line 5
    print("a*2 > b")
        ^
IndentationError: expected an indented block

# This is the correct way.
In : a = 2
...: b = 4
...: if a*2 > b:
...:     print("a*2 > b")
...: elif a*2 < b:
...:     print("a*2 < b")
...: else:
...:     print("a*2 = b")
a*2 = b
```

For-Loops

```
# Simple loop
In :for j in (1,2,3):
...:     print(j)
1
2
3
# range (m) is from 0 to m-1
In :for i in range(3):
...:     print(i)
0
1
2
# range (n,m) is from n to m-1
In :for i in range(2,4):
...:     print(i)
2
3
# range(n,m,s) is from n to m-1 with s steps
In :for i in range(1,6,2):
...:     print(i)
1
3
5

# List containing strings
In :cnt = ["US", "UK", "France"]
...:for j in cnt:
...:     print(j)
US
UK
France
```

While-Loops

```
# Plus 1 until "econ" reaches to 10
In :econ = 7
...:while econ < 10:
...:    print(econ)
...:    econ += 1
...:print("Break!")
7
8
9
Break!
```

List Comprehension

```
# Create a list by loop within list.
In : a = [1,2,3]
...: b = [i - 1 for i in a]
...: b
Out: [0, 1, 2]

# Create a list by loop within list with if-statement.
In : c = [i for i in a if i > 1]
...: c
Out: [2, 3]
```

Ranks of Loops and If-statement

```
In : for i in range(1,2):
...:     for j in range(1,5): # Needs indentation for inside loop
...:         print(i + j)
...:
...:print("End of Loop") # Good to indicate the end of the loop
2
3
4
5
End of Loop

In : for i in range(1,3):
...:     if i < 2: # Needs indentation for inside if-statemnt
...:         print(i)
...:
...:print("End of Loop") #End of loop
1
End of Loop
```

Break and Continue

```
In :nameslist =["Tom"]*100 + ["Ken"]
# Break
In :for name in nameslist:
...:     if name == "Ken":
...:         print("I found Ken!")
...:         break
I found Ken!

# Continue
In :for name in nameslist:
...:     if name == "Ken":
...:         continue  #Skip next line and move to a next loop.
...:         print("I found Ken!")
```

Functions

- Functions help us to:
  - Reuse your code in different occasions.
  - Avoid copying and pasting for repetitive tasks.
  - Logically divide your project into a set of different sub-tasks, which is easier to manage.

E.g.) Functions

```
# Function without returns
In : def g(e,c,o,n):
...:     d = e*c*o*n
...:     print(d)
In : g(1,2,3,4)    # Call function g()
24

# Function with a single return
In : def g_return(e,c,o,n):
...:     return e*c*o*n
In : g_return(1,2,3,4)    # Call function g()
Out: 24

# Functions with multiple returns
In : def h(e,c,o,n):
...:     return e,c,o,n
In : h(1,2,3,4)    # Call function h()
Out: (1, 2, 3, 4) # tuple
In : def q(x,y):
...:     return [x+3, y+4]
In : q(2,3)    # Call function q()
Out: [5, 7] # list
```

E.g.) Anonymous Functions

```
# A function returns a squared value.
# lambda defines an anonymous function after it.
In : square = lambda x: x**2
...: square(8)
Out: 64

# A lambda can take more than one arguments.
In : expo = lambda x,y: x**y
...: expo(3,2)
Out: 9

# Note: A lamda can generate only a single return.
# You cannot do like below:
In : twosums = lambda x,y: x+3,y+4
Traceback (most recent call last):

  File "<ipython-input-150-ecfddaf1e31d>", line 1, in <module>
    twosums = lambda x,y: x+3,y+4

NameError: name 'y' is not defined
```

Modules & Packages

- We saw some conveniences of functions. However, once you shut down the interpreter, the definitions you have made (functions and variable) are lost.
- As the scale of your research project gets larger, your program gets longer.
- Shorten your program & maintain your codes easily by:
  - Splitting your program into several files
  - Storing a specific function that you use repetitively
- **Module**: a single python code file containing Python definitions and statements that can be imported into other programs.
- **Package**: a collection of Python modules.

  E.g.) NumPy, Pandas, SciPy, Scikit-learn, NLTK, etc.

Import Modules & Packages

There are many publicly-available modules and packages.

```
## E.g. 1) "math" module
# Import "math" module and set its name as "ma"
In : import math as ma
# Call the function inside ma which calculates square root
In : ma.sqrt(2)
Out: 1.4142135623730951
# Need to specify the imported module ma before sqrt.
In : sqrt(2)
Traceback (most recent call last):

  File "<ipython-input-3-66e338417901>", line 1, in <module>
    sqrt(2)

NameError: name 'sqrt' is not defined

# Or, import only the function "sqrt" from the module "math"
In : from math import sqrt
In : sqrt(2)
Out: 1.4142135623730951


## E.g. 2) The module of miscellaneous operating system interfaces
In : import os
...: os.getcwd()    # Current directory
Out: 'C:\\Users\\masah\\Dropbox\\Masahiro'
# Change directory
In : os.chdir(r"C:\Users\masah\Dropbox\Masahiro")
In : os.getcwd()    # Verify the change
Out: 'C:\\Users\\masah\\Dropbox\\Masahiro'
```

Import Your Own Module

```
# Create the module "fourelements.py", imported in the main program
# Recall: Module name = File name
```

```
def g(e,c,o,n):
    """Multiplications:
       Multiply 4 inputs
    """
    d = e*c*o*n
    return d
def f(e,c,o,n):
    """Subtractions"""
    d = e-c-o-n
    return d
```

```
# The main program, containing the same name function:
```

```
import fourelements
def g(e,c,o,n):
        d = e+c+o+n
        return d
```

```
In : g(1,2,3,4)
Out: 10
In : fourelements.g(1,2,3,4)
Out: 24

In : help(fourelements.g)
Help on function g in module fourelements:

g(e, c, o, n)
    Multiplications:
    Multiply 4 inputs
```

Install/Update/Remove Packages

- **MacOS:** Open up the Terminal
  Windows: Open up Anaconda Powershell or Command Prompt
  Type each command.

- Install packages
  - A single package: `conda install package-name`
  - A package with a specific version:
    `conda install ‘‘package-name=version’’`

- Update packages
  - A single package: `conda update package-name`
  - Multiple packages:
    `conda update package-name-1 package-name-2`
  - All packages: `conda update --all`

- Remove packages
  - A single package: `conda remove package-name`
  - Multiple packages:
    `conda remove package-name-1 package-name-2`
  - All packages: `conda remove --all`

Note. `pip install package-name` instead of `conda` can also work

NumPy

- Excellent package for scientific computation:
  - Fast N-dimensional array (ndarray) processing.
  - Sophisticated mathematical functions.
- Many scientific packages are also built on NumPy.
- Widely used in academia and industries.
- NumPy official documentation

  Helpful cheatsheets for Matlab or Julia users: Matlab/Julia↔Python; Matlab↔Numpy
- Import NumPy first:

```
In : import numpy as np
```

NumPy Arrays – Types

```
# Create an array with four zeros
In : zeros = np.zeros(4)
...: zeros
Out: array([0., 0., 0., 0.])

In : type(zeros[0])
Out: numpy.float64

# Create an array with four ones (integers)
In : ones   = np.ones(4, dtype=int)
...: ones
Out: array([1, 1, 1, 1])

In : type(ones[0])
Out: numpy.int32

# Create an array with two ones (boolean)
In : boolones = np.ones(2, dtype=bool)
...: boolones
Out: array([ True,  True])

In : type(boolones[0])
Out: numpy.bool_
```

NumPy Arrays vs. Lists

- Similarities:
  - Storing data.
  - Mutability.
  - Can be indexed and sliced.
- Differences:
  - Can implement arithmetic easily with ndarrays.

```
# Easy to transform between lists and ndarrays
In : list_eg = [1,2,3,4]
In : ndarray_eg = np.array(list_eg)
In : ndarray_eg
Out: array([1, 2, 3, 4])
In : ndarray_eg.tolist()
Out: [1, 2, 3, 4]

In : ndarray_eg/2
Out: array([0.5, 1. , 1.5, 2. ])
In : list_eg/2   # Cannot implement arithmetics with lists
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

NumPy Arrays – Shape & Dimension

```
In : zeros = np.zeros(4) # a flat array with no dimension
..: zeros.shape
Out: (4,)

In : zeros.shape = (2,2) # convert into a two-dimensional array
...: zeros
Out:
array([[0., 0.],
       [0., 0.]])
In : zeros.shape
Out: (2, 2)

In : zeros = np.zeros((4,1)) # 4 x 1 array
...: zeros
Out:
array([[0.],
       [0.],
       [0.],
       [0.]])
```

NumPy Arrays – Creating Arrays

```
In : empty = np.empty(5)   # Create an empty array
...: empty
Out: array([0.  , 0.25, 0.5 , 0.75, 1.  ])
# Garbage values: Helps when we do not need explicit initialization of arrays
# (Fast to create garbage values than np.zeros or np.ones)

# Create a grid of evenly spaced numbers
In : grid = np.linspace(2, 9, 3)  # an array from 2 to 9 with 3 elements
...: grid
Out: array([2. , 5.5, 9. ])

# Create a grid of evenly spaced values with a specific interval size.
In : grid = np.arange(1,2,0.1) #[1,2) with 0.1 steps.
...: grid
Out: array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9])

# Create an identity array
In : identity = np.identity(3)
...: identity
Out:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])

# Create an array with integers
In : intarray = np.array([[10,2], [7,4]], dtype = int)
...: intarray
Out:
array([[10,  2],
       [ 7,  4]])
```

NumPy Arrays – Indexing

```
In : a = np.array([1,3,4])
...: a[0] # index starts at zero!
Out: 1
In : a[-1] # index starts at -1 backwards.
Out: 4
In : a[0:2] # extract an array from index 0 to 1.
Out: array([1, 3])
In : b = np.array([[1,3],[4,5]])
...: b[0,0] # extract an array with position (0,0).
Out: 1
In : b[:,1] # extract all columns with index 1.
Out: array([3, 5])
In : b[1,:] # extract all rows with index 1
Out: array([4, 5])
In : b[:] # extract all rows and columns
Out:
array([[1, 3],
       [4, 5]])

In : c = np.linspace(5,25,5)
...: c[a] # extract elements with index [1,3,4].
Out: array([10., 20., 25.])
In : d = np.array([1, 1, 0, 0, 1], dtype = bool)
...: c[d] # extract elements by a boolean array.
Out: array([ 5., 10., 25.])

In : e = np.array([4,5,3,0])
...: e.nonzero() # returns indexes of non-zero values
Out: (array([0, 1, 2], dtype=int64),)
```

NumPy Arrays – Methods

```
In : econ = np.array([10, 4, 2, 1])
...: econ.sort()
...: econ
Out: array([ 1,  2,  4, 10])
In : econ.sum()    # Sum over the elements
Out: 17
In : econ.max()    # The maximum element
Out: 10
In : econ.argmax()    # The index of the maximum element (10)
Out: 3
In : econ.mean()    # Mean over elements
Out: 4.25
In : econ.cumsum()    # Cumulative sums
Out: array([ 1,  3,  7, 17], dtype=int32)
In : econ.cumprod()    # Cumulative products
Out: array([ 1,  2,  8, 80], dtype=int32)
In : econ.var()    # Variance
Out: 12.1875

# Methods handling missing data
In : nanecon = np.array([10, 4, 2, 1, np.nan])
...: nanecon.sum()
Out: nan
In : np.nansum(nanecon)
Out: 17.0
In : np.nanmax(nanecon)
Out: 10.0
```

Basic Operations on Arrays – Basic Math

```
In : a = np.array([4,3,2,1])
...: b = np.array([1,2,3,4])
...: a + b   # Add element by element
Out: array([5, 5, 5, 5])

In : a * b   # Product element by element
Out: array([4, 6, 6, 4])

In : a + 2   # Add 2 to each element
Out: array([6, 5, 4, 3])

In : a * 2   # Multiply each element by 2
Out: array([8, 6, 4, 2])

# Two-dimensional arrays
In : c = np.array([[1,2],[2,1]])
...: d = np.array([[3,4],[4,3]])
...: c + d
Out:
array([[4, 6],
       [6, 4]])

In : c + 2
Out:
array([[3, 4],
       [4, 3]])

In : c / d   # Element-wise division
Out:
array([[0.33333333, 0.5       ],
       [0.5       , 0.33333333]])

# Multiplication is more complicated! -> next slide
```

Basic Operations on Arrays – Matrix Multiplication

```
In : a = np.array([[1, 2], [3, 4]])
...: b = np.array([[1, 2], [1, 2]])
In : a @ b   # Matrix multiplication (a*b in Matlab)
Out:
array([[ 3,  6],
       [ 7, 14]])
In : np.dot(a,b) # Another way of matrix multiplication (a*b in Matlab)
Out:
array([[ 3,  6],
       [ 7, 14]])
In : a * b   # Element-wise multiplication (a.*b in Matlab)
Out:         # np.multiply(a, b) also gives the same
array([[1, 4],
       [3, 8]])
In : c = np.array([[0.6], [0.8]])
...: d = np.array([1,2])
...: e = np.array([[1],[2]])
In : c * d   # Matrix multiplication (c*d in Matlab). NOT c @ d here!!!
Out:
array([[0.6, 1.2],
       [0.8, 1.6]])
In : c * e  #: Again, element-wise multiplication (c.*e in Matlab)
Out:
array([[0.6],
       [1.6]])
# 3 ways to generate a column vector: c=[0.6; 0.8] in Matlab
In : c_1, c_2 = 0.6, 0.8
...: c_mat1 = np.array([c_1, c_2]).reshape(2,1)
...: c_mat2 = np.array([c_1, c_2]).reshape(-1,1)
...: c_mat3 = np.array([[c_1], [c_2]]) # check if all these are the same.
# Tuple is also treated as an column vector.
In : a @ (0,1)
Out: array([2, 4])
```

Basic Operations on Arrays – Mutability

```
In : j = np.array([1,2,3,4])
...: j[0] = 10 # Change the element
...: j
Out: array([10,  2,  3,  4])


In : j = np.array([1,2,3,4])
...: k = j
...: k[0] = 10
...: j
Out: array([10,  2,  3,  4])
# Changing the element in k = changing the same element in j !!!

# How to prevent j from reflecting a change in k above?
In : j = np.array([1,2,3,4])
...: k = np.copy(j)
...: k[0] = 10
...: j
Out: array([1, 2, 3, 4])
# No change in j
```

Pandas

- Excellent packages for data analysis and manipulation.
- Pandas can handle many data file format including stata, csv, excel, sql, etc.
- Pandas is built on NumPy and designed for manipulation, missing data, queries, splitting and so on.
- Come back later again with more detail in the data management & visualization section.
- Pandas official documentation

  Helpful cheatsheets for Stata users:
  Stata↔Python; Stata↔Pandas
- Import Pandas first:

```
In : import pandas as pd
```

Series & Dataframe

- Two data structures in Pandas:
  - **Series**: a one-dimensional labeled array that is able to hold any data type.
  - **Dataframe**: a two-dimensional data structure.

Basic Operations: Series – Indexing

```
# E.g.) Series
In : import pandas as pd
...: series = pd.Series([1,3,10],index=['row1','row2','row3'])
...: series
Out:
row1     1
row2     3
row3    10
dtype: int64

# Extract elements by indexes
In : series.index
Out: Index(['row1', 'row2', 'row3'], dtype='object')

In : series['row2'] # get the element with index name 'row2'
Out: 3
In : series.loc['row2'] # get the element with index name 'row2'
Out: 3

In : series[0] # get the element with index 0
Out: 1
In : series.iloc[1] # get the element with index 1
Out: 3
```

Basic Operations: Series – Slicing

```
## How to slice series by indexes
# The end label is included when you use .loc().
In : series.loc[['row1','row2']]
Out:
row1    1
row2    3
dtype: int64

In : series.loc['row1':'row3']
Out:
row1     1
row2     3
row3    10
dtype: int64

# The end label is not included when you use .iloc().
In : series.iloc[0:2]
Out:
row1    1
row2    3
dtype: int64
```

Basic Operations: Series – Querying & Boolean Filtering

```
## Slice series with specified conditions.
# Return the row(s) satisfied with a bracketed condition.
In : series[series>3]
Out:
row3    10
dtype: int64

# Return the boolean series indicating\
# if a condition following an equal is satisfied.
In : bseries = series >3
...: bseries
Out:
row1    False
row2    False
row3     True
dtype: bool

# Return the row(s) satisfied with True value.
In : series[bseries]
Out:
row3    10
dtype: int64
```

Basic Operations: Series – Conversion – Series.items()

```
# Find out the data type
In : series.dtype
Out: dtype('int64')

# Convert data type into strings
In : series_str = series.astype('str')
...: series_str.dtype
Out: dtype('O')

# See the conversion from int to string.
In : series_str.iloc[1]
Out: '3'

## Iteration: series.items()
# Series.item() generates a tuple (index, value)
# whenever a series is iterated.
In : for index, value in series.items():
...:     print("{}:{} + 10 = {}".format(index, value, value + 10))
row1:1 + 10 = 11
row2:3 + 10 = 13
row3:10 + 10 = 20
```

Basic Operations: Series – Apply Functions

```
# Apply functions
In : def f(x):
...:     return x+5
...: series.apply(f)
Out:
row1     6
row2     8
row3    15
dtype: int64
In : series.apply(lambda y:y - 5)
Out:
row1   -4
row2   -2
row3    5
dtype: int64
# The index label is gone when you use the map function!
In : result = pd.Series(map(lambda y:y - 5, series))
...: result
Out:
0   -4
1   -2
2    5
dtype: int64
```

Basic Operations: Series – Concatenation

```
# Suppose that we'd like to add the below series.
In : add = pd.Series([10,3,10],index=['row1','row2','row3'])
...: add
Out:
row1    10
row2     3
row3    10
dtype: int64

# Append "add" to "series".
In : combine = series.append(add)
...: combine
Out:
row1     1
row2     3
row3    10
row1    10
row2     3
row3    10
dtype: int64

# Return the series with index label 'row1'.
In : combine.loc['row1']
Out:
row1     1
row1    10
dtype: int64

In : combine[0]
Out: 1
```

Basic Operations: Series – Updating by Values and Indexes

```
# Updating by values
In : series.replace([1,10],[5,9])
Out:
row1    5
row2    3
row3    9
dtype: int64

# Updating by indexes
In : series.update(add)
...: series
Out:
row1    10
row2     3
row3    10
dtype: int64
```

Basic Operations: DataFrame – Indexing

```
# E.g.) DataFrame
In : df = pd.DataFrame([[1,2,3],[3,9,6],[10,2,-6]],
...:                   columns= ['col1', 'col2', 'col3'],
...:                   index=['row1','row2','row3'])
...: df # Look at the difference between Series (1-dim) and DataFrame (2-dim)!
Out:
      col1  col2  col3
row1     1     2     3
row2     3     9     6
row3    10     2    -6

# Extract elements by indexes
In : df.col1 # Extract the column "col1".
Out:
row1     1
row2     3
row3    10
Name: col1, dtype: int64

In : df.iloc[0] # Extract the row with the index 0.
Out:
col1    1
col2    2
col3    3
Name: row1, dtype: int64

In : df.iloc[[1,2]] # Extract the rows with the indexes 1 and 2.
Out:
      col1  col2  col3
row2     3     9     6
row3    10     2    -6
```

(Cont'd) Basic Operations: DataFrame – Indexing

```
# Extract the element (1,2).
In : df.iloc[1,2]
Out: 6
# Extract the row with the index 0.
In : df.iloc[0:1]
Out:
      col1  col2  col3
row1     1     2     3
# Extract the row and the column with the index 0.
In : df.iloc[0:1,0:1]
Out:
      col1
row1     1
# Extract the rows from the index "row2" to "row3".
In : df.loc['row2':'row3']
Out:
      col1  col2  col3
row2     3     9     6
row3    10     2    -6
# Extract the element ("row3","col3").
In : df.col3.row3
Out: -6
In : df['col3']['row3']
Out: -6
```

Basic Operations: DataFrame – Querying & Boolean Filtering

```
## How to slice dataframe with specified conditions.
# Return the dataframe with boolean indicating\
# if each element is satisfied with a condition.
In : df.col1 >= 3
Out:
row1    False
row2     True
row3     True
Name: col1, dtype: bool

# Return the dataframe with elements\
# satisfied with a bracketed condition.
In : df[df.col1 >= 3]
Out:
      col1  col2  col3
row2     3     9     6
row3    10     2    -6

# Multiple conditions
In : df[(df.col1 >=3) & (df.col2 >= 3)]
Out:
      col1  col2  col3
row2     3     9     6
```

Basic Operations: DataFrame – Iteration

```
# iteritems(): iterate over columns and each column is returned as a series.
In : for col, val in df.iteritems():
...:     if col == 'col1':
...:         print('The column is {} and the value is \n{}'.format(col, val))
The column is col1 and the value is
row1     1
row2     3
row3    10
Name: col1, dtype: int64

# iterrows(): iterate over rows and each row is returned as a (index, series) pair.
In : for row, val in df.iterrows():
...:     if row == 'row2'
...:         print('The row is {} and the value is \n{}'.format(row, val))
The row is row2 and the value is
col1    3
col2    9
col3    6
Name: row2, dtype: int64

# itertuples(): iterate over rows and each row is returned as a named tuple.
In : for tup in df.itertuples():
...:     print(tup)
Pandas(Index='row1', col1=1, col2=2, col3=3)
Pandas(Index='row2', col1=3, col2=9, col3=6)
Pandas(Index='row3', col1=10, col2=2, col3=-6)
```

Basic Operations: DataFrame – Apply function

```
# Apply sum over rows (axis = 0)
In : df.apply(sum, axis = 0)
Out:
col1    14
col2    13
col3     3
dtype: int64

# Apply sum over columns (axis = 1)
In : df.apply(sum, axis = 1)
Out:
row1     6
row2    18
row3     6
dtype: int64

# Apply an anonymous function to each element.
In : df.applymap(lambda x:x*2)
Out:
      col1  col2  col3
row1     2     4     6
row2     6    18    12
row3    20     4   -12
```

Basic Operations: DataFrame – Updating by Methods

```
In : df.applymap(lambda x:x*2)
Out:
      col1  col2  col3
row1     2     4     6
row2     6    18    12
row3    20     4   -12
In : df    # The original df is not updated by the previous method
Out:
      col1  col2  col3
row1     1     2     3
row2     3     9     6
row3    10     2    -6
In : df = df.apply(lambda x:x*2)    # Update df by this
...: df
Out:
      col1  col2  col3
row1     2     4     6
row2     6    18    12
row3    20     4   -12
In : df['col1'] = df.col1.apply(lambda x:x*2)
...: df    # Update only column1
Out:
      col1  col2  col3
row1     4     4     6
row2    12    18    12
row3    40     4   -12
```

Basic Operations: DataFrame – Missing Data

```
# Dataframe with missing values
df = pd.DataFrame([[1,2,],[3, 6],[10,2,-6]],
                  columns= ['col1', 'col2', 'col3'],
                  index=['row1','row2','row3'])

# If a cell is nan, this returns True.
In : df.isna()   # or, df.isnull()
Out:
       col1   col2   col3
row1  False  False   True
row2  False  False   True
row3  False  False  False

# If a cell is not nan, this returns True.
In : df.notna()
Out:
      col1  col2   col3
row1  True  True  False
row2  True  True  False
row3  True  True   True
```

Basic Operations: DataFrame – Joining Data

```
# Random dataframes
df1 = pd.DataFrame([[1,2,3],[3,9,6],[10,2,-6]],
                   columns= ['col1', 'col2', 'col3'],
                   index=['row1','row2','row3'])

df2 = pd.DataFrame([[3,9,6],[1,2,3],[10,2,-6]],
                   columns= ['col1', 'col2', 'col3'],
                   index=['row1','row2','row3'])

# Join two dataframes using index
In : df1.join(df2, rsuffix = '_2') # the names of columns in df2 are appended to '_2'.
Out:
      col1  col2  col3  col1_2  col2_2  col3_2
row1     1     2     3       3       9       6
row2     3     9     6       1       2       3
row3    10     2    -6      10       2      -6

# Join two dataframes using index
In : df1.join(df2, lsuffix = '_1') # the names of columns in df1 are appended to '_1'.
Out:
      col1_1  col2_1  col3_1  col1  col2  col3
row1       1       2       3     3     9     6
row2       3       9       6     1     2     3
row3      10       2      -6    10     2    -6
```

Basic Operations: DataFrame – Merging Data

```
# Random dataframes
df3 = pd.DataFrame([[1,2,3],[3,9,6],[10,2,-6]], columns= ['col1_1', 'col2', 'col3_1'],
                   index=['row1','row2','row3'])
df4 = pd.DataFrame([[3,9,6],[1,2,3],[10,2,-6]], columns= ['col1_2', 'col2', 'col3_2'],
                   index=['row1','row2','row3'])

# Merge two dataframes using index
In : df3.merge(df4, left_index=True, right_index=True)
Out:
      col1_1  col2_x  col3_1  col1_2  col2_y  col3_2
row1       1       2       3       3       9       6
row2       3       9       6       1       2       3
row3      10       2      -6      10       2      -6

# Merge two dataframes on a common column ('col2')
In : df3.merge(df4, on = 'col2') # Note: merge many to many by default
Out:
   col1_1  col2  col3_1  col1_2  col3_2
0       1     2       3       1       3
1       1     2       3      10      -6
2      10     2      -6       1       3
3      10     2      -6      10      -6
4       3     9       6       3       6

# Merge two dataframes on different columns
In : df3.merge(df4, left_on = 'col1_1', right_on = 'col1_2')
Out:
   col1_1  col2_x  col3_1  col1_2  col2_y  col3_2
0       1       2       3       1       2       3
1       3       9       6       3       9       6
2      10       2      -6      10       2      -6
```

Basic Operations: DataFrame – "Validate" option

```
# Use "validate" so that you can find if you have duplicate IDs!
In : df3.merge(df4, on = 'col2', validate = "one_to_many")
MergeError: Merge keys are not unique in left dataset; not a one-to-many merge
```

pandas.DataFrame.merge official documentation

We will come back to merging with more detail again in the data management & visualization section.

Regular Expressions

- We learnt that Python can handle string data.
- You may want to...
  - find all email addresses in a document.
  - extract latitude and longitude values from (latitude, longitude).
  - find if a specific string exists in another string.
- The Python module "re" enables you to work with regular expressions.
- We cover basic operations here. See Regular Expression Operations for more detail.
- Also, we will come back to this topic again in the text-mining section for research applications later.

Module: re – Search & Match

```
# Importing re module
import re

# Search
# span = (n, m): the position of a matched character
In : re.search('@', 'masahiro_kubo@brown.edu')
Out: <re.Match object; span=(13, 14), match='@'>

# Match
# cannot find if a character is in between two letters.
In : re.match('@', 'masahiro_kubo@brown.edu') #returns nothing

In : re.match('@', '@brown.edu')
Out: <re.Match object; span=(0, 1), match='@'>

# Using special characters to extract information you need.
# . : any character
# * : any number of characters
In : re.match('.*@', 'masahiro_kubo@brown.edu')
Out: <re.Match object; span=(0, 14), match='masahiro_kubo@'>
```

Module: re – Findall, Split, and Sub

```
# Find all
In : re.findall('s', 'Brown University Economics')
Out: ['s', 's']

In : re.findall('s.', 'Brown University Economics')
Out: ['si']

# Extract characters starting with 's'
# . : any character
# ? : match zero or one occurences
In : re.findall('s.?', 'Brown University Economics')
Out: ['si', 's']

# Split a string by 'University'
In : re.split('University', 'Brown University Economics')
Out: ['Brown ', ' Economics']

# Replace 'n' by '?'
re.sub('n', '?', 'Brown University Economics')
Out: 'Brow? U?iversity Eco?omics'
```

E.g.) Extracting Latitude and Longitude

```
# Random latitude and longitude information list
lat_lon = ["lat:35.689722, lon:139.692222",\
           "lat:51.507222, lon:-0.1275",\
           "lat:41.823611, lon:-71.422222"]

# Create a list containing latitude information
# \d : a number
# \. : the decimal point
In : lat = [re.findall('\d*\.\d*', i)[0] for i in lat_lon]
...: lat
Out: ['35.689722', '51.507222', '41.823611']
```

Exercises:

Q. Extract longitudes.

Q. Transform the obtained list into the list consisting of numbers, not strings (so that ArcGIS can understand them as geographic coordinates).

Other Useful Pakcages for Economists

- **SciPy**: an open-source library for mathematics, science, and engineering
  - Used often in sections for numerical methods later
  - E.g.) integration, equation-solving, optimization
- **Scikit-learn**: an open-source library for regressions and machine learning algorithms
  - Built on NumPy, Pandas, and SciPy
  - Partly covered in sections for research applications
- **Natural Language Toolkit (NLTK)**: a leading platform for working with human language data
  - Much more functions than the regular expression module that we saw
  - Used intensively in the text-mining section later
- Check out many other available packages depending on your various purposes!

Floating Point Arithmetic

Floating Point Arithmetic

- Most computing languages (not specific to Python) use floating points to represent wide range of values with their finite capacities.
- **Floating-point numbers are not necessarily equal to their true values!**

Floating Point Arithmetic

- Most computing languages (not specific to Python) use floating points to represent wide range of values with their finite capacities.
- **Floating-point numbers are not necessarily equal to their true values!**

Recall your assignment 0:

```
# 1.0 x 3 = 3?
In : 1.0*3 == 3
Out: True
# 0.1 x 3 = 0.3?
In : .1*3 == .3
Out: False
In :  .1*3
Out: 0.30000000000000004

# What is a floating-point number displayed as 0.1?
# Display "0.1" with 18 significant digits
In : format(0.1, '.18g')
Out: '0.10000000000000006'
```

Floating-Point Numbers ≠ True Decimal Values

- E.g.) 0.125 as the decimal fraction (base 10) and 0.001 as the binary fraction (base 2) have the identical values:
  - $0.125 = \frac{1}{10} + \frac{2}{100} + \frac{5}{1000}$: Decimal fraction
  - $0.001 = \frac{0}{2} + \frac{0}{4} + \frac{1}{8}$: Binary fraction.

Floating-Point Numbers ≠ True Decimal Values

- E.g.) 0.125 as the decimal fraction (base 10) and 0.001 as the binary fraction (base 2) have the identical values:
  - $0.125 = \frac{1}{10} + \frac{2}{100} + \frac{5}{1000}$: Decimal fraction
  - $0.001 = \frac{0}{2} + \frac{0}{4} + \frac{1}{8}$: Binary fraction.
- Decimal fractions you enter are only approximated by their corresponding binary fractions stored in a machine.
  - The decimal value 0.1 cannot be represented exactly as a base 2 fraction.
  - In base 2, 0.1 is the infinitely repeating fraction:
    $0.0001100110011001100110011...$
    $(= 0/2^1 + 0/2^2 + 0/2^3 + 1/2^4 + 1/2^5 + ...)$

Floating-Point Numbers ≠ True Decimal Values

- E.g.) 0.125 as the decimal fraction (base 10) and 0.001 as the binary fraction (base 2) have the identical values:
  - $0.125 = \frac{1}{10} + \frac{2}{100} + \frac{5}{1000}$: Decimal fraction
  - $0.001 = \frac{0}{2} + \frac{0}{4} + \frac{1}{8}$: Binary fraction.
- Decimal fractions you enter are only approximated by their corresponding binary fractions stored in a machine.
  - The decimal value 0.1 cannot be represented exactly as a base 2 fraction.
  - In base 2, 0.1 is the infinitely repeating fraction:
    $0.0001100110011001100110011...$
    $(= 0/2^1 + 0/2^2 + 0/2^3 + 1/2^4 + 1/2^5 + ...)$
- On the other hand, for integers, Python does not distinguish, for example, between 3.00 and 3.

More in Details

- Most machines use IEEE-754 floating point arithmetic, and almost all platforms map Python floats to IEEE-754 “double precision”.

- Most floating numbers are normalized in this way:
  $x = \pm(1 + f) \cdot 2^N$.

- A double-precision float consists of a total of 64 bits:
  - 1 bit for the sign of the number $(\pm)$,
  - 11 bits for the exponent (N), and
  - 52 bits for the fraction/mantissa (f).

Limitations

- Of course, floating point cannot represent all numbers.
  - f must satisfy $0 \leq f < 1$
  - $2^{52} \cdot f$ must be an integer in the interval: $0 \leq 2^{52} \cdot f < 2^{53}$.
  - N must be an integer in the interval: $-1022 \leq N \leq 1023$.
- In an absolute sense, there are the upper bound and lower bound of the values which floating point can represent.
  - The larger values than the upper bound are said to be inf or $infinity$.
  - The smaller values than the lower bound are 0.

E.g.) What is the floating number 0.1?

Example: $x = 0.1$.

$$0.1 = 2^{52} \cdot (1 + f) \cdot 2^{J}$$
$$\Rightarrow 2^{52} \cdot (1 + f) = 2^{-J}/10$$

Recalling $2^{52} \cdot f$ is an integer containing 52 bits (53 bits including plus 1) and $2^{52} \leq 2^{56}/10 < 2^{53}$, then $J = -56$. Hence,

$$\begin{aligned} 2^{52} \cdot (1 + f) &= 7205759403792794. \\ \Rightarrow 2^{52} \cdot (1 + f) \cdot 2^{J} (= 0.1) &= 7205759403792794 \cdot 2^{-56}. \end{aligned}$$

```
# Let's verify above.
In  : 0.1 * (2 ** 56)
Out : 7205759403792794.0 # 2**52*(1 + f)
```

- If more interested in why the errors arise in detail, see:
  - Floating Point Arithmetic: Issues and Limitations.
  - Moler, Cleve. "Floating points." eps 2.5 (1996): 52.

Practical Issues

- Some practical issues for floating point math based on Moler, Cleve. "Floating points." eps 2.5 (1996): 52.
    - Round-off error
    - Matrix manipulation
    - Cancellation

Round-off Error

```
In : x = np.arange(0.988, 1.012, .0001)
...: y = -(x-1)**2 + 4
...: x_max = x[y.argmax()]  # Should be 1 mathematically.
...: x_max
Out: 0.9999999999999987

In : min(abs(x-1)) # Never hit 1 in x!
Out: 1.3322676295501878e-15
```

This round-off error will matter a lot in scientific computations, especially in numerical differentiation!

Matrix Manipulation

Singular equations:

$$\begin{cases} 0.1x + 0.3y = 1 \\ x + 3y = 10 \end{cases}$$

```
# Matrix A below does not have an inverse, but...
In : A = np.array(([0.1, 0.3],[1 ,3]))
...: np.linalg.inv(A) # Inverse of matrix A
Out :
array([[ 1.08086391e+17, -1.08086391e+16],
       [-3.60287970e+16,  3.60287970e+15]])
```

This is because floating point numbers 0.1 and 0.3 are not exactly equal to 0.1 and 0.3, respectively.

Cancellation

- Repeated addition and subtraction for tiny numbers can cause severe errors.

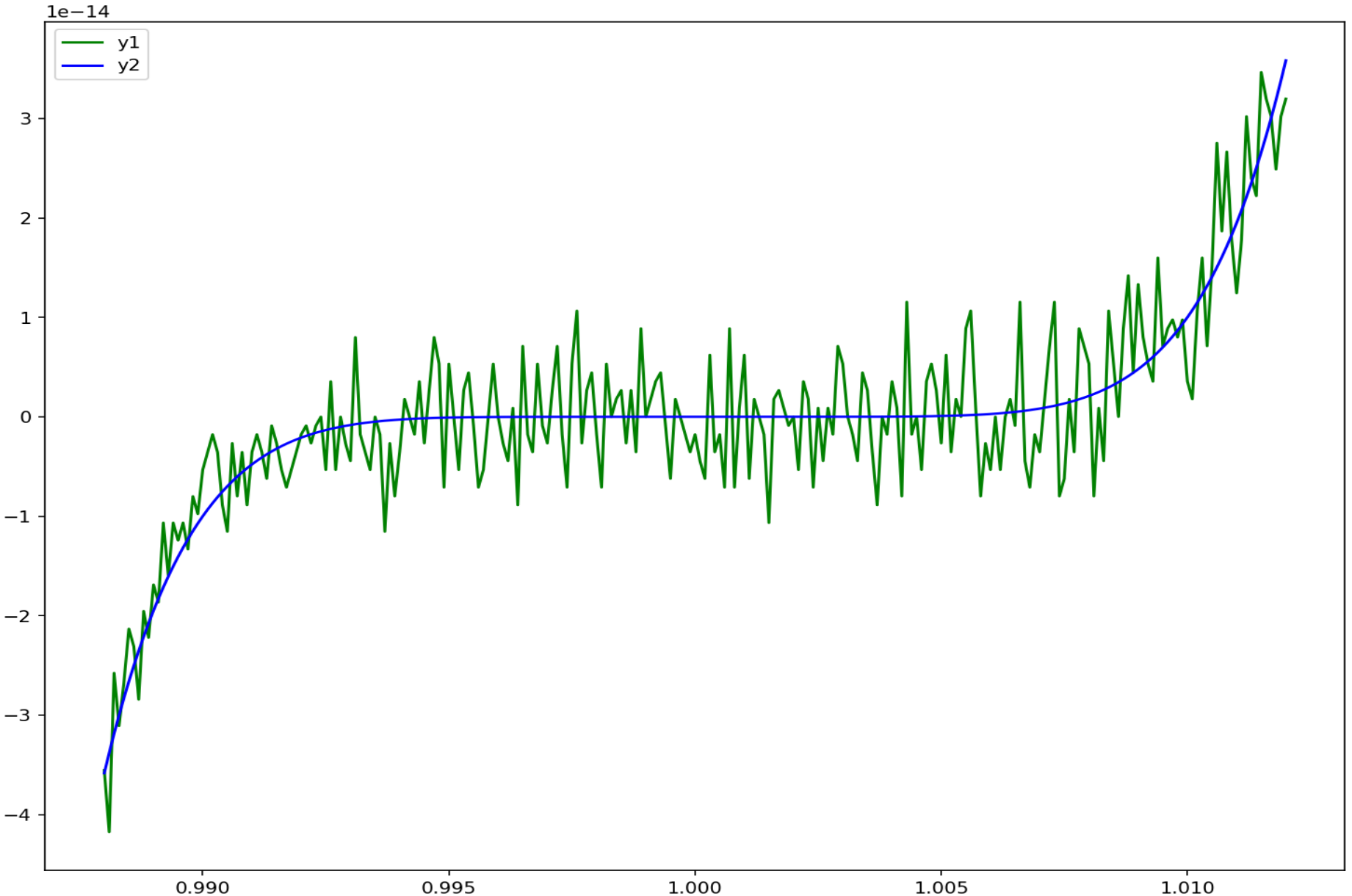
```
# y1 and y2 are the same function.
In : import numpy as np
...: import matplotlib.pyplot as plt
...: x  = np.arange(0.988, 1.012, .0001)
...: y1 = x**7- 7*x**6+ 21*x**5- 35* x**4+ 35*x**3- 21*x**2+ 7*x- 1
...: y2 = (x - 1)**7

# What is the value of x with y nearly equal to 1? - Should be nearly 1.
In : zero1 = x[abs(y1).argmin()]
...: zero1
Out: 0.9923999999999995 #A substantial discrepancy from 1!

In : zero2 = x[abs(y2).argmin()]
...: zero2
Out: 0.9999999999999987
```

Plotting these two reveals what happens.

(Cont'd) Cancellation



How to Judge If Two Values/Arrays are Sufficiently Close?

```
In : import numpy as np
...: import math
...: d = .1*3

# Without any functions (1e-09 = 1*10**(-9))
In : if 0.3 - d < 1e-09:
...:     print("0.3")
0.3

# Use math.isclose(a, b,  rel_tol, abs_tol)
# abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)
In : math.isclose(d, 0.3, rel_tol=1e-09, abs_tol=0.0)
Out: True

# Use np.isclose(a, b,  rtol, atol, equal_nan)
# absolute(a - b) <= (atol + rtol * absolute(b))
# equal_nan: Whether to compare NaN's as equal.
In : np.isclose([d, 0.32], [0.3, 0.3], rtol=1e-05, atol=1e-08, equal_nan=False)
Out: array([ True, False])

# Use np.allclose(a, b,  rtol, atol, equal_nan)
# Returns True if two arrays are element-wise equal within a tolerance.
# absolute(a - b) <= (atol + rtol * absolute(b))
In : np.allclose([d, 0.32], [0.3, 0.3], rtol=1e-05, atol=1e-08, equal_nan=False)
Out: False
```

Intro to Object Oriented Programming (OOP)

Programming Paradigms

Procedural: The program moves through instructions linearly

- Simple to write and easy to do line-by-line trial and error

Programming Paradigms

Procedural: The program moves through instructions linearly

- Simple to write and easy to do line-by-line trial and error

Functional: The program moves from function to function

- Separating tasks into sub-functions make the code readable, scannable, and maintainable
- By looking at the "main" function, easy to understand the entire strucuture of the program
- Easy to run a subset of the entire program

Programming Paradigms

Procedural: The program moves through instructions linearly

- Simple to write and easy to do line-by-line trial and error

Functional: The program moves from function to function

- Separating tasks into sub-functions make the code readable, scannable, and maintainable
- By looking at the "main" function, easy to understand the entire strucuture of the program
- Easy to run a subset of the entire program

Object Oriented Programming (OOP): a programming paradigm which provides a means of structuring programs so that data and its behaviors are bundled into an individual object (???)

Object Oriented Programming (OOP): Overview

Object Oriented Programming (OOP): a programming paradigm which provides a means of structuring programs so that data and its behaviors are bundled into an individual object

- Variables, functions, or both are grouped together in such a way that all the included objects "see" each other and can interact intimately
- Recognized as the almost unique successful paradigm for complex software
- Python is an object-oriented language (Documentation)
- See Software Carpentry lectures for more detail

Advantages:

- Object types can be generalized by using **classes**, simplifying our program
- One class can be used to create many objects, including variables and methods (functions), all from the same flexible piece of code
- Maintainability: A class can **inherit** attributes and behaviors from another class, allowing us to define common behaviors once in one place and override just those definitions that want to be changed

$\Rightarrow$ Particularly beneficial for larger scale programs

Objects

- In Python, everything in memory is an object.
- Everything includes lists, functions, modules, etc.
- An object consists of four elements:
  - Type: string, list, number, boolean, etc
  - Identity
  - Data and Attributes
  - Methods

Identity

A unique identifier is assigned to each object, so that Python can keep track of each object.

```
# All ([], True, and a) are objects.
In: id([])
Out: 2809772609352

In: id(True)
Out: 140736572856656

In: a = 4
...: id(a)
Out: 140736573379488
```

Data and Attributes

- Data is contained in some type of an object (e.g. int, boolean, float).
- An object contains attributes.
- An attribute name is obtained by typing a dot after an object name.

```
In : econ = 1 # econ is an object.
...: econ.imag # An attribute by which you can get imaginary part.
Out: 0

In : univ = "Brown"
...: univ.__class__ # An attribute by which you can know what type is.
Out: str
```

Methods

- Methods are functions and callable attributes that are bundled with objects.
    - Using callable() enables you to identify if something is callable or not.

```
In : univ = "Brown"
...: callable(univ.count) # This is a callable attribute.
Out: True
# "count" method enables you to count how many times "B" appears in the object.
In : univ.count("B")
Out: 1
```

- Like the example above, if you define another object with string type, it has the same bundled attributes and methods (functions)!
- Again, we can know an outcome/data behavior through an object, which makes programs easier to be understood.

Classes

- OK, but how can we create multiple methods (functions) and attributes that you want bundled into an object?
- Classes are user-defined molds in which you can define multiple methods and attributes.

E.g.) Classes

```
class Person: # define a class

    def __init__(self, name, age): #define an initial method
        self.name = name  # add name attribute
        self.age = age # add age attribute

    def cohort(self, eventyear):  #define "birth cohort" method
        self.birth = eventyear - self.age


# Instantiate object
ken = Person("Ken", 32)

In : ken.name
Out : Ken

In : ken.age
Out: 32

# Use "birth cohort" method and pass "eventyear" instance variable
In : ken.cohort(2019)
...: ken.birth
Out: 1987

# Each instance stores data in dictionary.
In : ken.__dict__
Out: {'name': 'Ken', 'age': 32, 'birth': 1987}
```

E.g.) Class Method

- So far, we use methods (e.g. "birth cohort" method) which require an instance (e.g. "eventyear") in order to call it (**instance method**).
- Below we use a class method that belongs to the class as a whole (**class method**)

```
class US:
    def __init__(self, state): # instance method
        self.state = state

    def country(cls):  #class method
        return "US"

In : RI = US("Rhode Island") # instance: "Rhode Island"
...: RI.state
Out: Rhode Island

In : RI.country()
Out: US
```

Inheritance

- You may want to define a new class by modifying methods in a class that you have already defined.
- A new class can inherit methods from a class that you have already defined.
- Newly formed classes are called *child classes*, and the classes that *child classes* are derived from are called *parent classes*.

  (*child* class = *subclass* = *derived* class)

E.g.) Inheritance

```
# Parent class
class country:
    def __init__(self, name):
        self.name = name

# Child class
class admin1(country):  # inheritance
    pass # empty class besides the above method (inheritance)

US = country("US")
RI = admin1("Rhode Island")

In : US.name
Out: US

In : RI.name
Out: Rhode Island
```

Import a Class as Your Own Module

```
# Create the module "class_imported.py", imported in the main program
# Recall: Module name = File name
```

```
class Person:
    def __init__(self, name, age): #define an initial method
        self.name = name  # add name attribute
        self.age = age # add age attribute

    def cohort(self, eventyear):  #define "birth cohort" method
        self.birth = eventyear - self.age
```

```
# The main program: "class_call.py"
```

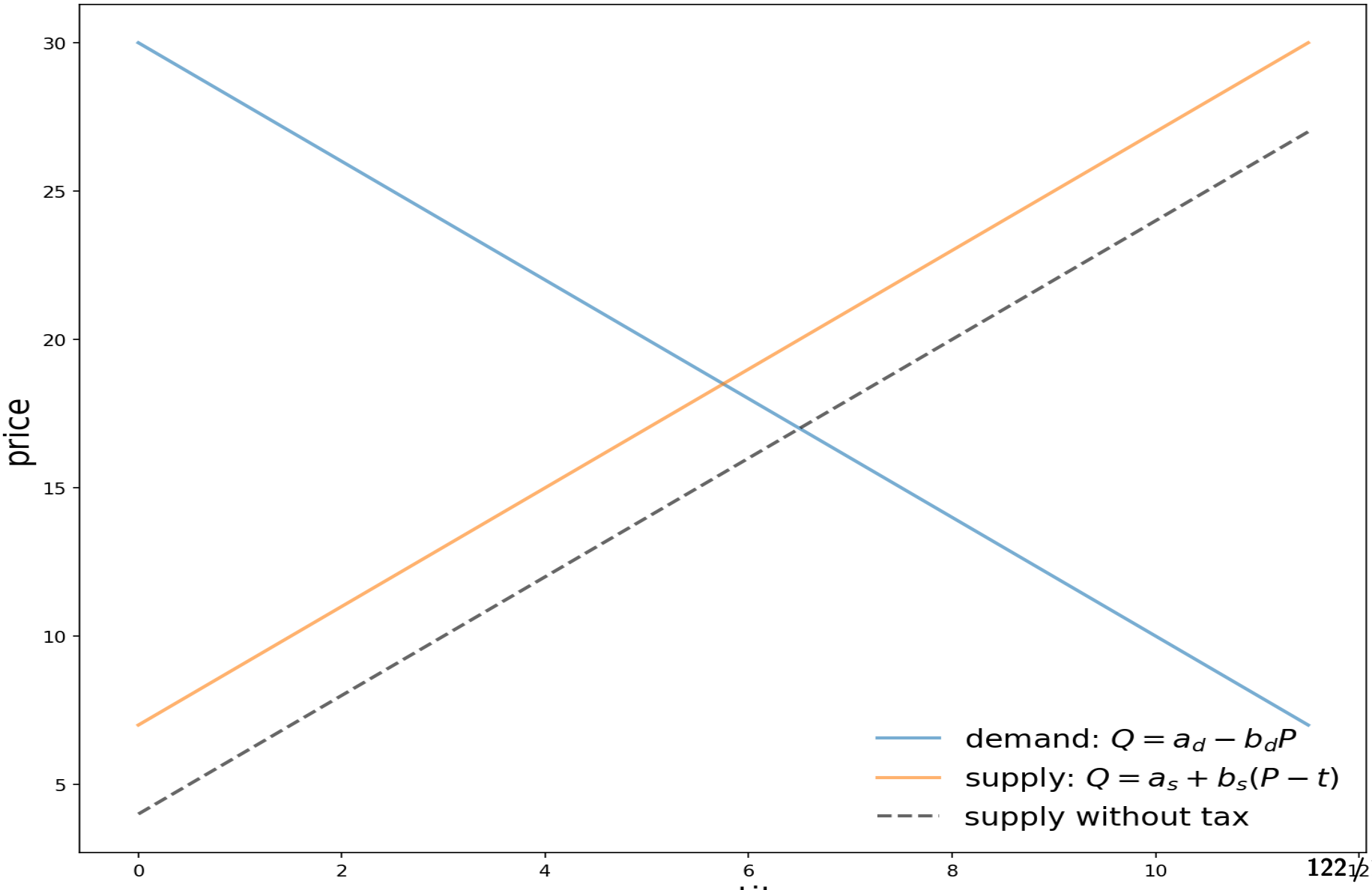
```
import class_imported #Import class_imported.py

#instantiate object in class that we have imported
ken = class_imported.Person("Ken", 32)
ken.cohort(2019)

# Inheritance from "Person" class in class_imported.py
class Person_inherited(class_imported.Person):
    pass
masa = Person_inherited("Masa", 29)
masa.cohort(2019)
```

```
In : ken.age
Out: 32
In : masa.birth
Out: 1990
```

E.g.) Market Supply & Demand



E.g.) Market Supply & Demand

```
class Market:
    def __init__(self, a_d, b_d, a_s, b_s, tax):
        """
        Set up market parameters. All parameters are scalars
        """
        self.a_d, self.b_d, self.a_s, self.b_s, self.tax = a_d, b_d, a_s, b_s, tax
        if a_d < a_s:
            raise ValueError('Insufficient demand')
    def price(self):
        "Compute equibrium price"
        return (self.a_d - self.a_s + self.b_s*self.tax) / (self.b_d + self.b_s)
    def quantity(self):
        "Compute equibrium quantity"
        return self.a_d - self.b_d*self.price()
    def cs(self):
        "Compute consumer surplus"
        integrand = lambda x: (self.a_d/self.b_d) - (x/self.b_d)
        area, error = quad(integrand, 0, self.quantity())
        return area - self.price()*self.quantity()
    def ps(self):
        "Compute producer surplus"
        integrand = lambda x: -(self.a_s/self.b_s) + (x/self.b_s)
        area, error = quad(integrand, 0, self.quantity())
        return (self.price() - self.tax)*self.quantity() - area
    def taxrev(self):
        "Compute tax revenue"
        return self.tax*self.quantity()
    def inv_demand(self,x):
        return (self.a_d/self.b_d) - (x/self.b_d)
    def inv_supply(self,x):
        return -(self.a_s/self.b_s) + (x/self.b_s) + self.tax
```

In-Class Exercise
for Object-Oriented Programming

In-Class Exercise (OOP)

Task. Define the class for the Malthusian model that you learnt.

- Population dynamics:

$$L_{t+1} = \frac{\gamma}{\rho} (AX)^{\alpha} L_t^{1-\alpha}$$

- Income dynamics:

$$y_{t+1} = \left[\frac{\rho}{\gamma}\right]^{\alpha} y_t^{1-\alpha}$$

where

- L_t : labor employed in period t.
- X : land.
- A : technological level.
- AX : effective resources.
- y_t : income per worker produced at time t $(= \frac{Y_t}{L_t})$.
- α : effective resources' share of output.
- γ : share of expenditure on children to income per worker.
- ρ : cost of raising a child.

(Cont'd) In-Class Exercise (OOP)

1. Define the method to derive dynamics of population and income per worker.
2. Define the method to derive population and income per worker at the steady state.
3. Define the method to derive when economy reaches to the steady state.
4. Derive the steady state values and the period when economy reaches to the steady state with above methods and the following values.
    - Initial conditions: $L_0 = 1$, $y_0 = \sqrt{2}$ and $AX = 2$
    - Parameter values: $\alpha = 0.5$, $\gamma = 0.3$, and $\rho = 0.6$.

- A helpful example with the Solow model from (an older version of) QuantEcon will be distributed.

Assignment 1

[1] Python Basics

1. `A = [1 ,2, 3]` & `B = ("EC", "CS", "AM")`.

   Define a new list `C` like below by using `A`, `B` and list comprehension:

   `C = ["1-EC", "2-CS", "3-AM"]`

2. `D = {1:"New York", 2:"Los Angeles", 3:"Chicago"}`

   Update `D` by adding `E = {4:"Houston", 5:"Phoenix"}`

3. `pop = [8398748, 3990456, 2705994, 2325502, 1660272]`

   By using a loop, generate a dictionary

   `F = {1: ("New York", 8398748), 2: ("Los Angeles", 3990456), 3: ("Chicago", 2705994), 4: ("Houston", 2325502), 5: ("Phoenix",1660272)}`

[1] Python Basics (Cont'd)

4. Let's find $\lim_{x \to 0} \frac{e^x - 1}{x}$.

- Generate a grid with the interval [-0.1,0.1) with 0.01 steps.
- Define the anonymous function in two ways.
  1. `numpy.exp(x)`
  2. `numpy.expm1(x)`
- Compare $\lim_{x \to 0} \frac{e^x - 1}{x}$ between 1 & 2. Which one is better?

5. Find $\lim_{n \to \infty} a_n$ in two ways: (1) a loop and (2) a function.

- $a_1 = 0$, $a_2 = 1$, and

$$a_{n+1} = \frac{1}{2}(a_n + a_{n-1}) \text{ if } n \geq 2.$$

- Generate a random array (100 x 100) using `np.random.rand(100,100)`.
- Find the nearest and the furthest value from the limit above and their positions in the array.

[2] Matrix Manipulation

1. Write a function file separately to calculate an inverse of 2 x 2 matrix, which addresses the issue arising from floating-point numbers that we have seen in the lecture.

2. By importing your function file, (try to) solve the two-by-two set of linear equations: (1) and (2).

$$\begin{cases} 0.9x + 4y = 10 \\ x + 3y = 6 \end{cases} \tag{1}$$

$$\begin{cases} 0.3x + 1.5y = 0.05 \\ 5.4x + 27y = 0.9 \end{cases} \tag{2}$$

[3] Regular Expressions

- First, just write down the below information in your script:

```
MessyList = ["New York:(latitude:40.6635,
longitude:-73.9387)", "Los Angeles:(lat:34.0194,
longi:-118.4108)"", "Chicago:(latitude:41.8376,
longitude:-87.6818)", "Houston:(latitude:29.7866,
longitude:-95.3909)"]
```

- Extract latitude and longitude information from the messy list and create a list like below.

```
list = [(city1, latitude1, longitude1), (city2, latitude2,
longitude2)]
```

- Extract latitude and longitude information from the messy list and create a dictionary in which each key can call both latitude and longitude information like below:

```
dict = { city1:  (latitude1, longitude1), city2:
(latitude2, longitude2) }
```