빅 데 이 터 혁 신 공 유 대 학

파이썬으로 배우는 기계학습

한동대학교 전산전자공학부

김영섭 교수

교육부 NRF 한국연구재단

Data Structures in Python
Chapter 1 - 1

- **Introduction - Review Python**
- Objects and References
- List Operations
- GitHub & Jupyter-Lab
- Markdown Tutorial

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전10:31)

Logistics - Piazza

- Enroll yourself for this course at **www.piazza.com**
  - You are required to have your Handong email address and know this course name.
- Use **Piazza** for Q&A and to submit your homework assignments.
  - 1st homework assignment is available at github.com/idebtor/DSpy/jupyter/**Ch1-2 Review(1) Overview.ipynb**
    - Section 2.2.2
      - "원격으로 웹 페이지 읽어 오기" (1) ~ (7)
    - Exercises:
      - 1. LIst Comprehension
      - 2 Palindrome
  - Due: 11:55 PM, One week from today (or the lecture day)
    - Upload the file itself at **pset1** folder in Piazza.
    - Late work is not accepted.

Table Of Contents

Chapter 1: Introduction to Data Structures and Python
- Python Review
- Object-Oriented Programming (OOP)

Chapter 2: Performance Analysis

Chapter 3: Linear Data Structures
- Stack and Queue

Chapter 4: Recursion

Chapter 5: Hashing, Sorting, and Searching

Chapter 6: Non-linear Data Structures
- Trees and Tree Algorithms

Chapter 7: Heap and Priority Queues

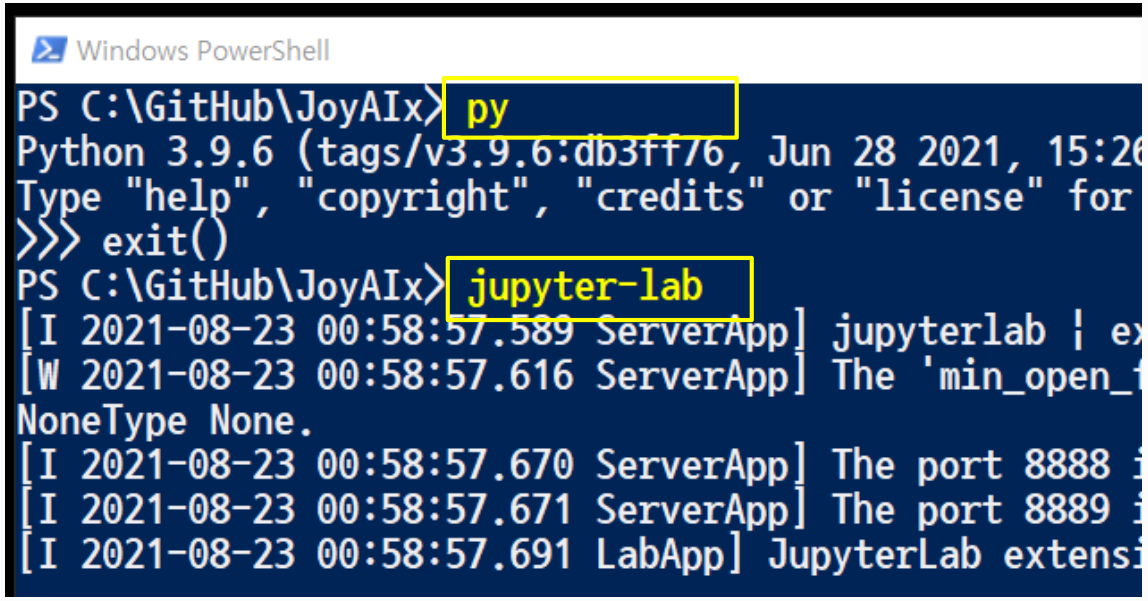
Chapter 8: (Optional) Graph and Graph Algorithms

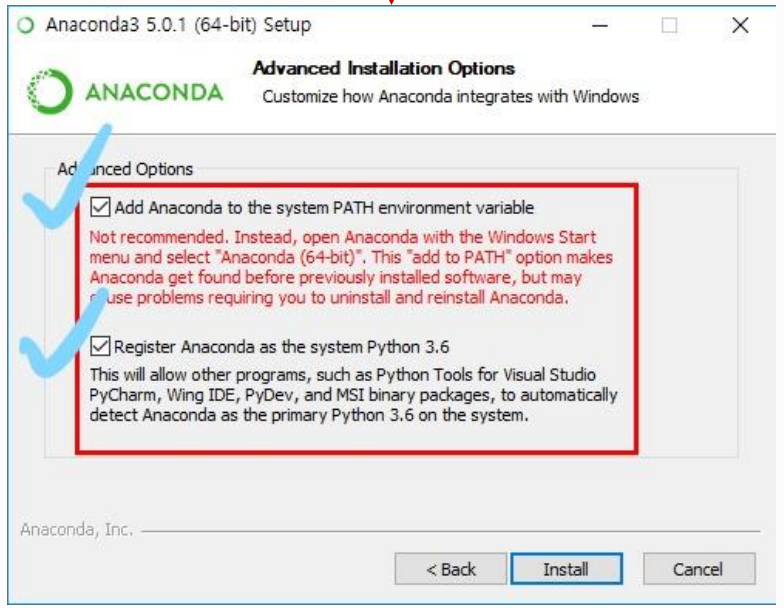
Learning outcomes

- A student who successfully completes this course will be able to:
  - Define a **class** to model and represent an **object. (OOP)**
  - **Write programs that store and manipulate data in standard linear data. structures (arrays, linked lists, stacks, queues) and non-linear data structures (hash tables, trees).**
  - Write code which handles important **exception** types.
  - Compare the efficiency of algorithms using standard **big-O notation.**
  - Implement **recursive data structures and solutions** to simple problems.
  - Explain the basic algorithm for any of the studied **sorting** methods.
  - Get familiar with **regular expressions** to extract data from a body of text.
- Tools to use
  - Use **Jupyter-lab** (or Jupyter notebook) for coding and reading the textbook.
  - Use **GitHub** to save and get source code files and resources for this course.

Git, GitHub & GitHub Desktop, Anaconda, and Jupyter-lab

- Install **Git** and **GitHub Desktop**
- Get an account at www.github.com
  - Read **GettingStarted.md** in **www.github.com/idebtor/DSpy/**
  - Clone **www.github.com/idebtor/DSpy** repository in your local computer.
- Install **Andaconda** package:
  - Make sure that you check the both options.
- Now you can start **Jupyter-lab** in console.





Agenda

- Topics:
  - Python Review
    - list
    - list comprehension
- References:
  - www.github.com/idebtor/DSpy:
    - Ch1-1: Introduction
    - Ch1-2: Review(1) ~ (7)
  - Problem Solving with Algorithms and Data Structures using Python
    - Chapter 1: Introduction

Python

- It is a programming language designed to be easy to read but **powerful**
  - **Readability**
  - **Simplicity**
  - **Extensibility**
- Ways of running a program
  - Interactive execution — great for learning
  - Creating a module (file) and executing the module
  - Use **Jupyter-lab** (or **Jupyter Notebook)** for interactive execution and documentation
- Install **Anaconda** Package that includes Python and Jupyter-lab (or Jupyter Notebook) and a very large library of standard modules

Lists

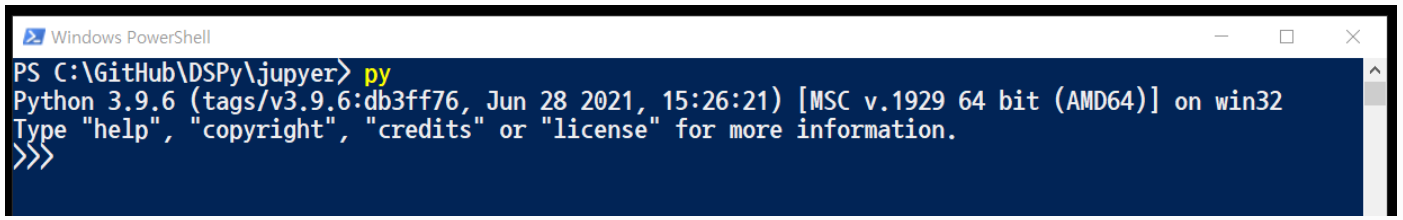
- Lists are a built-in type in Python
  - Use **square brackets** to signify a list
  - Lists can contain any type of data, or any mixture of data
  - Examples:

```
my_list1 = [1, 2, 3]

my_list2 = ['Hello', 'Is', 'there', 'anybody', 'out', 'there?']

my_list3 = [1, 5.899, 'Hello']

my_list4 = [4, 2, 6, 9, 3]
```

List functions

- Numerous list functions are supported
  - Use **help(list)** to find out the functions
  - Examples:

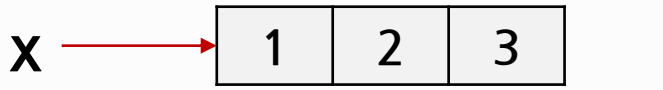
```
>>> x = [1, 2, 3]
>>> len(x)
3
>>> x + [4]
[1, 2, 3, 4]
>>> x += [5]
[1, 2, 3, 5]
>>> 3 in x
True
>>> x[0]
1
>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
```

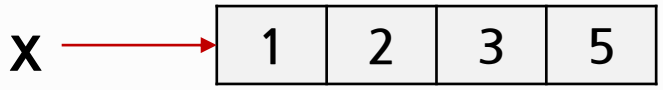


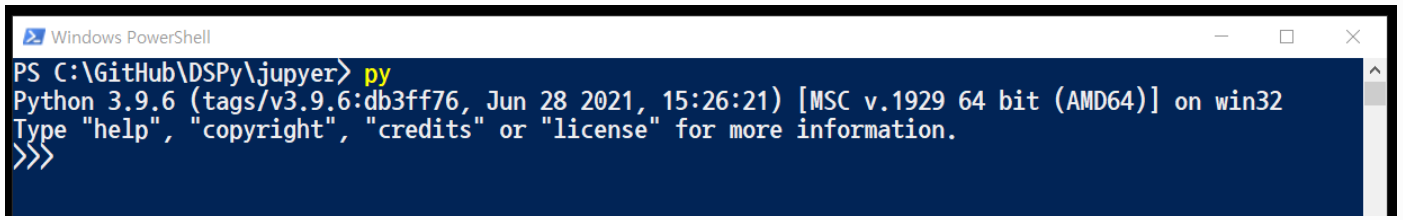
List functions

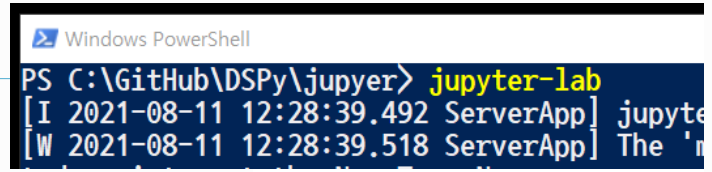
- Numerous list functions are supported
  - Use **help(list)** to find out the functions
  - Examples:

```
>>> x = [1, 2, 3]
>>> len(x)
3
>>> x + [4]
[1, 2, 3, 4]
>>> x += [5]
[1, 2, 3, 5]
>>> 3 in x
True
>>> x[0]
1
>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
```










```
[1]: x = [1, 2, 3]
     len(x)

[1]: 3

[2]: x + [4]

[2]: [1, 2, 3, 4]

[3]: x += [5]
     x

[3]: [1, 2, 3, 5]

[4]: 3 in x

[4]: True

[5]: x[0]

[5]: 1

[6]: [1, 2, 3] * 2

[6]: [1, 2, 3, 1, 2, 3]
```

List comprehensions

- A powerful feature of the Python language.
  - A list can be created using instructions that appear within the square bracket
  - Generate a new list by applying a function to every member of an original list

- The syntax of a “list comprehension” is tricky.
  - If you’re not careful, you might think it is a for-loop, an ‘in’ operation, or an ‘if’ statement since all three of these keywords (‘for’, ‘in’, and ‘if’) can also be used in the syntax of a list comprehension.
  - It’s something special all its own

```
my_list = [ x for x in range(9) ]
```

List comprehensions: Syntax 1

- The general format is as follows:

```
[expression for variable in sequence]
```

  - Where expression is some calculation or operation acting upon the variable.
    - For each member of the sequence, calculate a new value using expression, and then we collect these new values into a new list which becomes the return value of the list comprehension.

- Examples:

```
spells = [ ch for ch in 'Ann' ]
```

```
['A', 'n', 'n']
```

```
c_degs = [ -10, 0, 10, 100 ]
f_degs =
```

```
[14.0, 32.0, 50.0, 212.0]
```

```
basket = ['   banana', ' kiwi   ']
fruits =
```

```
['banana', 'kiwi']
```

```
square =
```

```
[1, 4, 9, 16]
```

List comprehensions: Syntax 2

- If the original list contains a variety of different types of values, then the calculations contained in the expression should be able to operate correctly on all of the types of list members.

```
items = [ 'hello', [1, 2], (a, b, c) ]
length =
```

```
[5, 2, 3]
```

- If the members of list are other containers, then the name can consist of a container of names that match the **type** and "**shape**" of the list members.

```
store = [ ('apple', 1), ('kiwi, 2), ('pear', 3) ]
order =
```

```
[2, 4, 6]
```

```
store = [ ('apple', 1), ('kiwi, 2), ('pear', 3) ]
order =
```

```
{'apple': 2, 'kiwi': 4, 'pear': 6}
```

List comprehensions: Syntax 3

- The expression of a list comprehension could also contain user-defined functions.

```
def c2f(c):
    return c * 1.8 + 32
```

```
c_degs = [ -10, 0, 100 ]
cflist =
```

```
[(-10, 14.0), (0, 32.0), (100, 212.0)]
```

- We can also create a list of tuples, and convert it a dictionary:

```
vector = [ 2, 4, 6 ]
square =
```

```
{2: 4, 4: 16, 6: 36}
```

List comprehensions that uses conditions (Filtered list)

- We can extend the syntax for a list comprehension to include a condition:

```
evens =                                                    [0, 2, 4, 6, 8]
```

- The general format is as follows:

```
[expression for variable in sequence if condition]
```

  - Similar to regular list comprehensions, except now we might not perform the expression on every member of the list.
  - We first check each member of the list to see if it satisfies a filter condition. Those list members that return False for the filter condition will be omitted from the list before the list comprehension is evaluated.

List comprehensions: Filtered List

- Examples:

```
vector = [ 2, 4, 6 ]
my_vec = [ 3 * x for x in vector if x > 3 ]
```

```
[12, 18]
```

```
vector = [ 2, 4, 6 ]
my_vec = [ 3 * x for x in vector if x < 2 ]
```

```
[]
```

Summary: Features of lists

- Information in a list is stored contiguously in memory
  - location of the information can be calculated
  - location = start of the list + index * size of each element
- Efficiency issues
  - It takes the same time to access any of the elements
  - Slow to move elements around (i.e. add and delete elements from within the list)

Data Structures in Python
Chapter 1 - 1

- **Introduction - Review Python**
- Objects and References
- List Operations
- GitHub & Jupyter-Lab
- Markdown Tutorial

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University