SAMSUNG

Together for Tomorrow!
Enabling People
Education for Future Generations

Samsung Innovation Campus

Artificial Intelligence Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 2.

Python Programming

AI Course

Samsung Innovation Campus

Chapter Description (1/2)

Duration: 24 Hours.

Purpose:

- Introduction to Python programming.
- Become familiarized with Jupyter notebook.
- Use Python to automatize simple tasks.
- Write functions, classes and algorithms for problem-solving.

Target Audience:

- This course is suitable for anyone who has previous experience in any modern programming language that implements class.
- This course is a prerequisite for every course in the AI curriculum.

Prerequisite:

- A student taking this course should be familiar with the concept of programming in any language.
- A student taking this course should also be familiar with the basic concepts of the information system.

Chapter Description (2/2)

Objectives:

- Use Jupyter notebook and Markdown.
- Understand basic data types and composite data types of Python.
- Differentiate mutable data types from immutable data types.
- Understand different control structures for decision making and iteration.
- Develop own user defined functions (UDFs).
- Organize the code in modules and classes.
- Understand and implement algorithms and simple data structures.
- Utilize libraries that control the operating system.
- Control the Excel, Word, PDF documents and retrieve data from them.

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 2.

Python Programming

UNIT 1.

Python I

Samsung Innovation Campus

UNIT 1.
Python I

What this unit is about:

- This unit is an introduction to programming with Python.
- You will get familiarized with Jupyter notebook and Markdown.

Expected outcome:

- Ability to edit Python code with Jupyter notebook.

How to check your progress:

- Coding Exercises.
- Quiz.

Chapter 2.

Python Programming

UNIT 1. Python I

1.1. About Python Programming Language.

1.2. A Quick Look at Python.

1.3. Jupyter Notebook.

Unit 2. Python II

2.1. Basic Data Types.

2.2. Composite Data Types.

2.3. Data Type Mutability.

What is Python?

Snake?

???

???

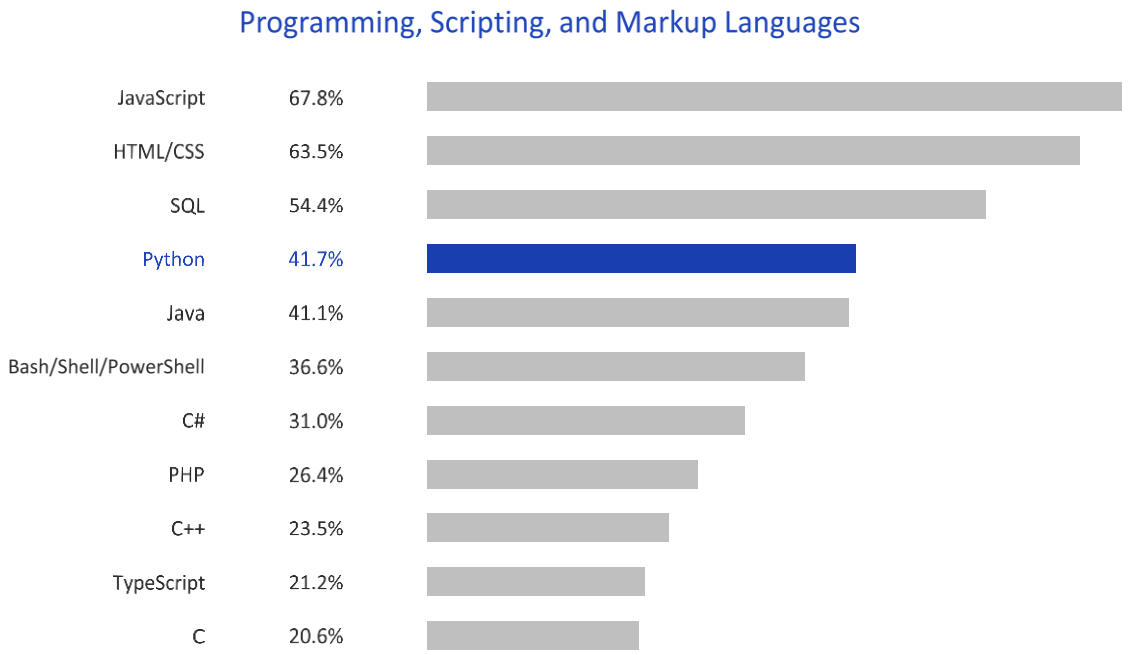
???

What is Python?

Python is a programming language!!

Popularity of Python

Python is ranked among the top programming languages.



(Source: https://insights.stackoverflow.com/survey/2019#technology-_-programming-scripting-and-markup-languages)

Popularity of Python

Python is ranked among the top programming languages.

Language	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. PHP		84.9
7. R		82.9
8. JavaScript		82.6
9. Go		76.4
10. Assembly		74.1

(Source: https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages)

Benefits of Using Python

- Python is easy to learn.
- Python is open-source and free, yet powerful.
- Python is widely used for artificial intelligence.

> “ Life is too short, You need Python! ”

Chapter 2.

Python Programming

UNIT 1. Python I

1.1. About Python Programming Language.

1.2. A Quick Look at Python.

1.3. Jupyter Notebook.

Unit 2. Python II

2.1. Basic Data Types.

2.2. Composite Data Types.

2.3. Data Type Mutability.

Python Quick Look : Arithmetic Operations

- You can use Python as a calculator.
- You can do arithmetic operations such as addition, subtraction, multiplication and division.
- Just use the corresponding arithmetic operators.

```
>>> 1 + 1
2
```

```
>>> 3.0 / 2.0
1.5
```

```
>>> 4 * 5
20
```

Python Quick Look : Variables (1/2)

- A variable is a container that stores data.
- In Python, a variable is created the first time you assign a value to it.

```
>>> x = 3
>>> y = 4
>>> x + y
7
```

Python Quick Look : Variables (2/2)

You may assign numbers as well as strings to the variable.

```
>>> x = 'Hello, World !'
>>> print(x)
Hello, World !
```

Python Quick Look : Control Structures (1/3)

Conditional structure if-else:

```
>>> x = 333
>>> if x > 100:
...         print('x is greater than 100.')
...     else:
...         print('x is smaller or equal than 100.')
...
x is greater than 100.
```

Python Quick Look : Control Structures (2/3)

- You can make iteration loops with while.
- The following is an example that prints out 1 through 5.

```
>>> x = 0
>>> while x < 5:
...             x = x + 1
...             print(x)
...
1
2
3
4
5
```

Python Quick Look : Control Structures (3/3)

- You can make iteration loops with for.
- The following is an example that prints out 1 through 5.

```
>>> for x in [1,2,3,4,5]:
...         print(x)
...
1
2
3
4
5
```

Python Quick Look : Functions

- There are several Python built-in function.
- Users can also define their own functions. They are called UDFs (User Defined Functions).

```
>>> def add2(x, y):
...                 return x + y
...
>>> print(add2(10,20))
30
```

Different Versions of Python : V2.7 vs V3.7

- As of July 2019, the latest version of Python is 3.7. This is the version you will use in this course.
- Python 3.7 is not backward compatible with Python 2.7.
- In the following slides, you will explore some of the major differences.

Versions of Python : Print

Python 2.7

```
>>> print('Hello')
Hello
>>> print 'Hello'
Hello
```

Python 3.7

```
>>> print('Hello')
Hello
>>> print 'Hello'                                                    # Parentheses should be used!
SyntaxError: Missing parentheses in call to 'print'. Did you mean print('Hello')?
```

- You can include comment after the pound sign #.

Versions of Python : Long Data Type

Python 2.7

```
>>> print(type(2**30))
<type 'int'>
>>> print(type(2**100))
<type 'long'>
```

Python 3.7

```
>>> print(type(2**30))
<class 'int'>
>>> print(type(2**100))
<class 'int'>
```

Versions of Python : Integer Division

Python 2.7

```
>>> 3/2                                  # Integer division.
1
>>> 3//2.0                               # Floor division.
1.0
```

Python 3.7

```
>>> 3/2                                  # Result is a real number
1.5
>>> 3//2                                 # Result is an integer.
1
>>> 3.0//2.0                             # Result is a real number 1.0.
1.0
```

Versions of Python : Range

Python 2.7

```
>>> x = range(10)
>>> print(x)                                    # Print the list object.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Python 3.7

```
>>> x = range(10)
>>> print(x)
range(0, 10)
>>> print(type(x))
<class 'range'>
```

Python Programming

UNIT 1. Python I

1.1. About Python Programming Language.
1.2. A Quick Look at Python.
1.3. Jupyter Notebook.

Unit 2. Python II

2.1. Basic Data Types.
2.2. Composite Data Types.
2.3. Data Type Mutability.

About Jupyter Notebook

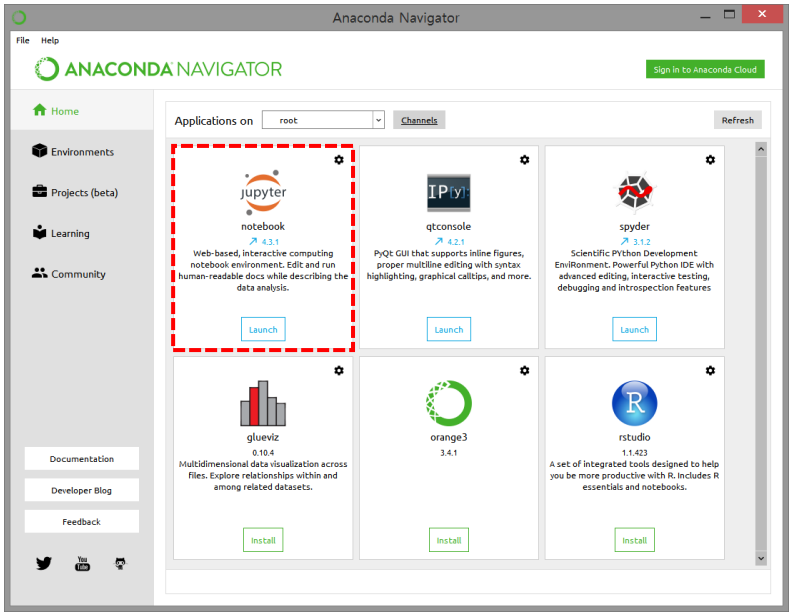
- Jupyter notebook evolved from IPython notebook.
- Jupyter notebook is an open-source interactive computing notebook for several different programming languages including Python.
  - Actually, Jupyter notebook also supports Julia, R, Haskell, Ruby, etc.
- It is a web-based application; runs on a browser.

Launching Jupyter Notebook

- From the command prompt:

```
C:\ Users\ JohnDoe\ Python Scripts>jupyter notebook
```

- Launch from Anaconda Navigator:
  - Please, remember to download the version 3.7.
  - To find out more, go to https://www.anaconda.com/distribution/



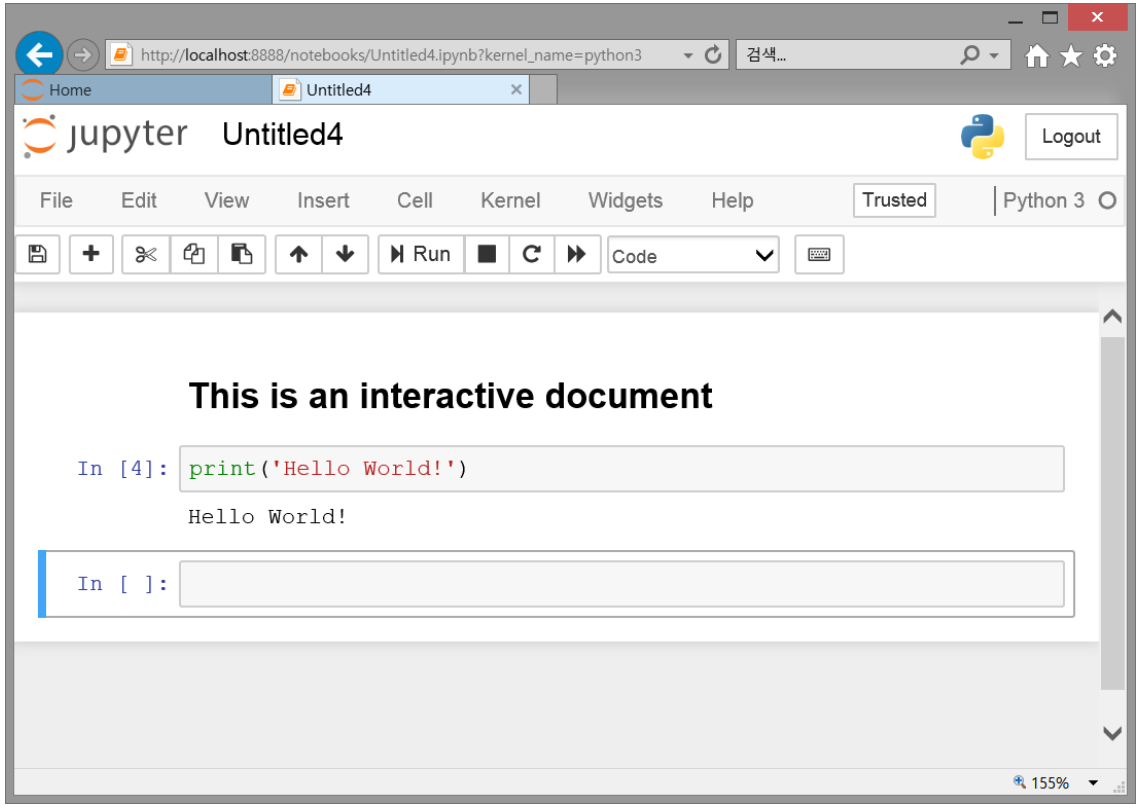
Features of Jupyter Notebook

Some of note-worthy features of Jupyter notebook are:

- Function auto-complete.
- Codes are organized in cell units.
- Supports Markdown, LaTex, etc. for formatting text.

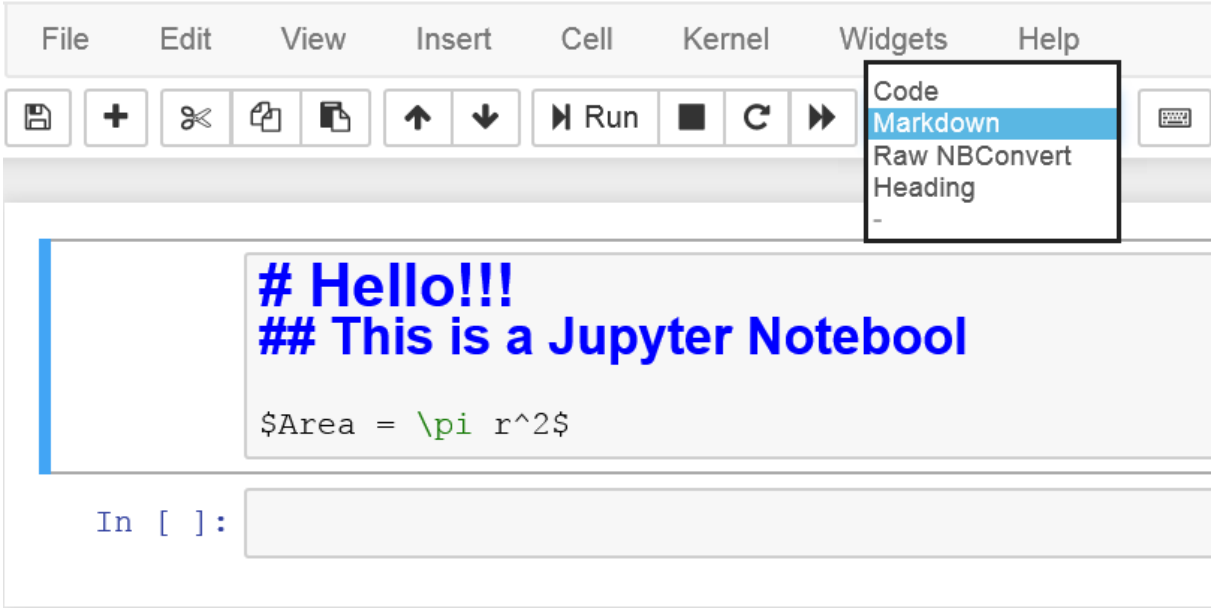
Jupyter Notebook Quick Introduction (1/10)

- A snapshot of Jupyter notebook in use:



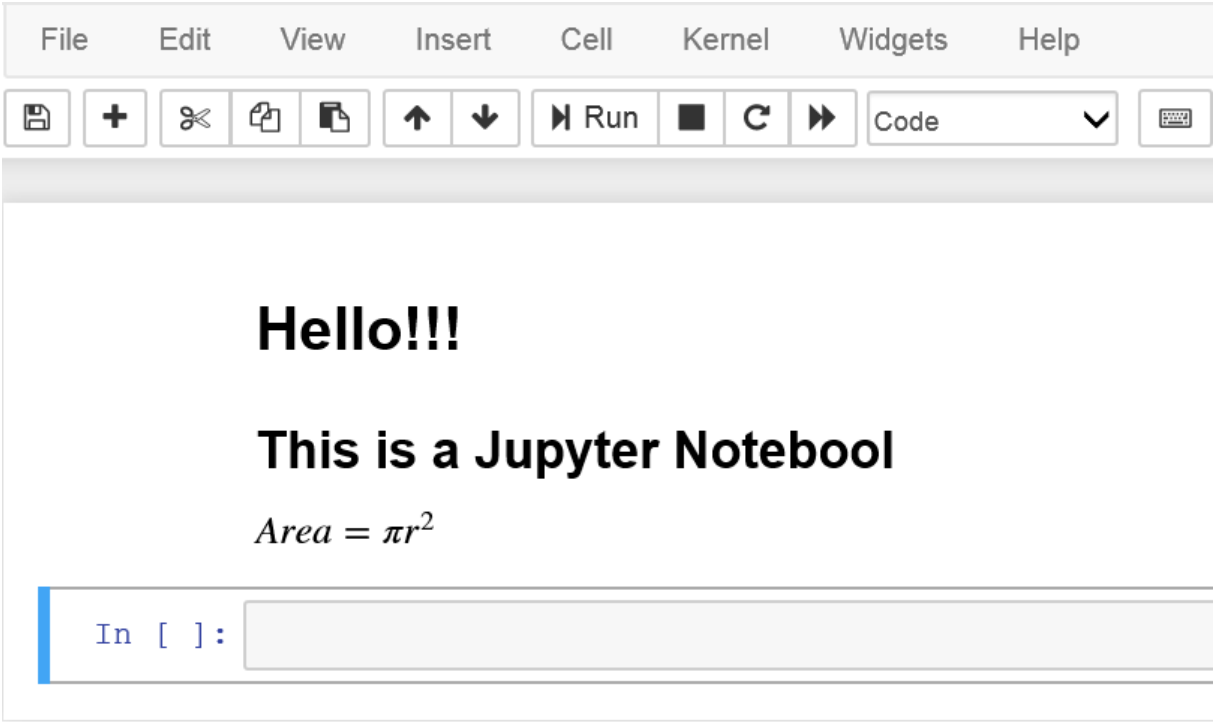
Jupyter Notebook Quick Introduction (2/10)

- You can switch between cell types: Code, Markdown, etc.



Jupyter Notebook Quick Introduction (3/10)

An example of compiled Markdown and LaTex formula.



Jupyter Notebook Quick Introduction (4/10)

- When you click on a cell, its border turns green. This is the edit mode.

- Some useful shortcuts while in the edit mode are:

Key Stroke	Action
CTRL + a	Select the whole line.
CTRL + d	Delete the whole line.
CTRL + z	Undo the last change.
CTRL + s	Save and checkpoint.

- From the command mode you can switch to the edit mode by pressing the [ENTER] key.
- In both the edit and the command modes, press [SHIFT] + [ENTER] to run or compile the selected cell.

Jupyter Notebook Quick Introduction (5/10)

You can enter into the command mode by selecting a cell and then pressing the [ESC] key.

- The cell border turns blue.

Some useful shortcuts while in the command mode are:

Key Stroke	Action
a	Insert a cell above.
b	Insert a cell below.
d + d (twice)	Delete the current cell.
m	Change the cell type to Markdown.
y	Change the cell type to code.

Jupyter Notebook Quick Introduction (6/10)

Some of the useful Markdown tags:

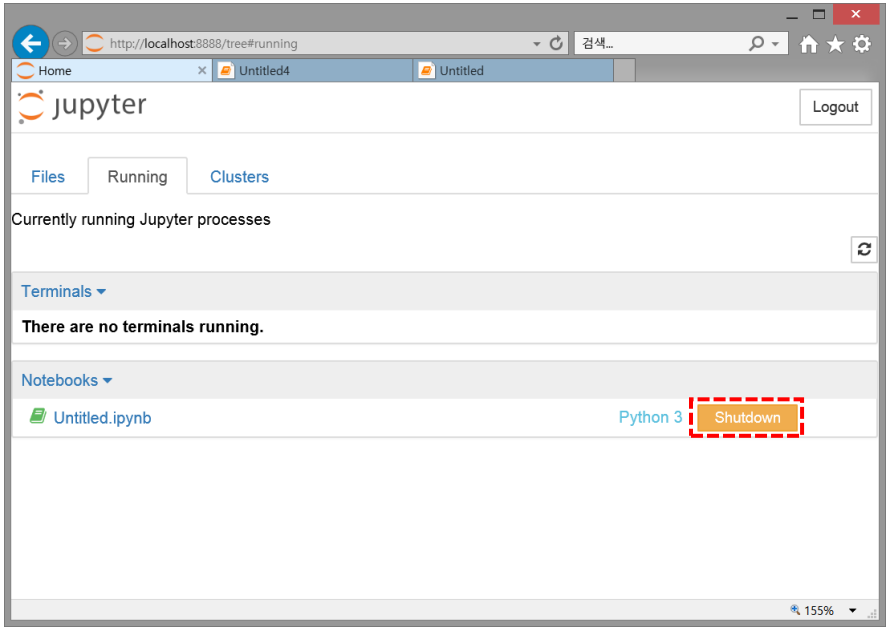
- You can find more information at https://daringfireball.net/projects/markdown/

Tag	Action
#	Title largest.
##	Title next largest.
- , *, or number	List items.
>	Quotes.
[text](URL)	Hyperlink.
* *Italic* *	Italic font.
** Bold **	Bold font.
$ ~~ $, $$ ~~ $$	LaTex expression.
---------	Horizontal line.
 	Line break.

Jupyter Notebook Quick Introduction (7/10)

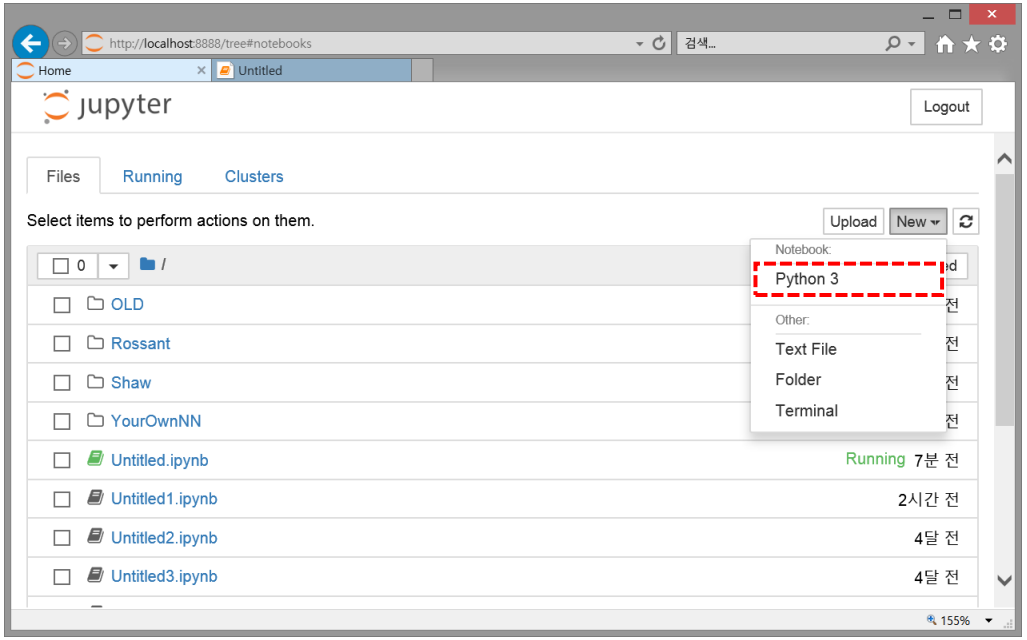
From the Home page click on the Running tab to view all the running notebooks.

- You can click on the Shutdown button to terminate a notebook.



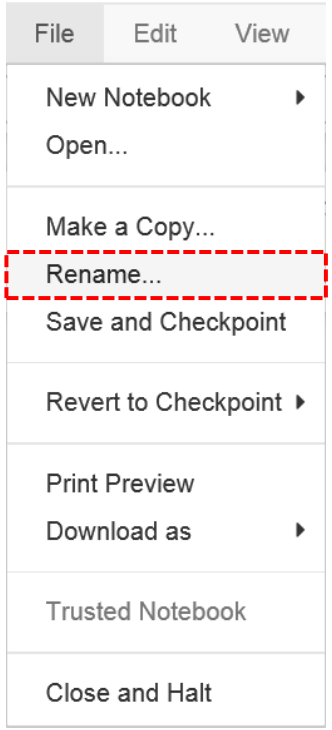
Jupyter Notebook Quick Introduction (8/10)

Press the New dropdown and then select Python 3 to start a new notebook.



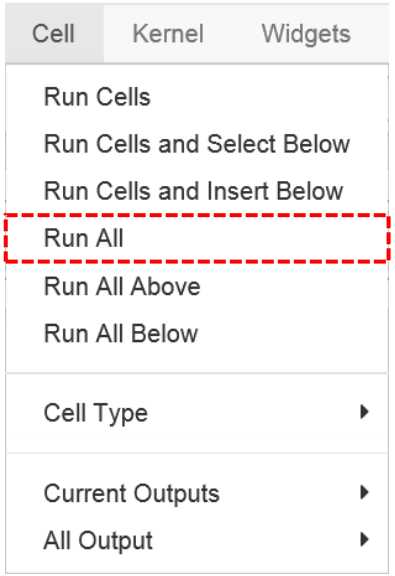
Jupyter Notebook Quick Introduction (9/10)

Enter into a notebook, then select Files → Rename in order to change the default name.



Jupyter Notebook Quick Introduction (10/10)

Enter into a notebook, then select Cell → Run All to execute all the cells from top to bottom in a row.



Coding Exercise #0101

Follow practice steps on 'ex_0101.ipynb'

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 2.

Python Programming

UNIT 2.

Python II

Samsung Innovation Campus

UNIT 2.
Python II

What this unit is about:

- In this unit you will learn about the different data types, both basic and composite.
- You will also learn how Python stores values in the memory and how to safely replicate objects.

Expected outcome:

- Ability to store, retrieve and modify data in Python objects.

How to check your progress:

- Coding Exercises.
- Quiz.

Python Programming

UNIT 1. Python I

1.1. About Python Programming Language.

1.2. A Quick Look at Python.

1.3. Jupyter Notebook.

Unit 2. Python II

2.1. Basic Data Types.

2.2. Composite Data Types.

2.3. Data Type Mutability.

Basic Data Types : Number (1/4)

Numbers:

- Integers (int): 123, 33, 0, -123, etc.
- Floating point (float): 123.456, 3.141592, 9.12e15, etc.
- Octal (int): 0o123, 0o456, etc.
- Hexadecimal (int): 0xA0, 0xFFAA, etc.

Basic Data Types : Number (2/4)

Arithmetic operations with numbers:

```
In[1] : 1.5 + 1
Out[1]: 2.5
```

```
In[2] : 3.0 / 1.5
Out[2]: 2.0
```

```
In[3] : 4 * 5
Out[3]: 20
```

Basic Data Types : Number (3/4)

Other operations with numbers:

```
In[1] : 2 ** 3                                    # Exponentiation.
Out[1]: 8
```

```
In[2] : 8 % 5                                     # Modulus.
Out[2]: 3
```

```
In[3] : 8.0 // 5.0                                # Floor division.
Out[3]: 1.0
```

Basic Data Types : Number (4/4)

You can assign numbers to variables.

```
In[1] : x = 2.1
In[2] : y = 3.5
In[3] : x + y
Out[3]: 5.6
```

Basic Data Types : String (1/11)

Use single or double quotation marks.

```
"Hello World"
 'Hello World'
```

~~"He said, "Python is very easy" "~~
~~'He said, 'Python is very easy' '~~

```
"He said, 'Python is very easy' "
'He said, "Python is very easy" '
```

Basic Data Types : String (2/11)

Multi-line strings:

```
In[1]  : my_string = """Hello !!!
   ...: my
   ...: name
   ...: is
   ...: Python"""
In[2]  : print(my_string)
Hello !!!
my
name
is
Python
```

Basic Data Types : String (3/11)

Escape characters start with a backslash.

- They have special meanings within a string sentence.

Escape Character	Meaning
\n	Line change.
\t	Tab.
\\	Backslash character.
\'	Single quotation.
\"	Double quotation.
\r	Carriage return.
\a	Bell.
\b	Back space.

Basic Data Types : String (4/11)

Use of escape characters (example).

```
"He said,\" Python is very easy \" "
'He said,\' Python is very easy \' '
```

Basic Data Types : String (5/11)

Concatenation (+) and Repetition (*):

```
In[1] : str1 = 'First string.'
In[2] : str2 = 'Second string.'
In[3] : str1 + str2
Out[3]: 'First string. Second string.'
```

```
In[1] : str = 'Python.'
In[2] : str * 3
Out[2]: 'Python. Python. Python.'
```

Basic Data Types : String (6/11)

Indexing and Slicing:

```
In[1] : str = 'Life is too short, You need Python!'
In[2] : str[0]
Out[2]: 'L'
```

```
In[3] : str[3]
Out[3]: 'e'
```

```
In[4] : str[-1]                                    # The first from the end.
Out[4]: '!'
```

- In Python, indices start from 0. So, the first character corresponds to the index 0!
- Negative index means position from the end; index -1 corresponds to the last character.

Basic Data Types : String (7/11)

Indexing and Slicing:

```
In[5] : str[0:4]                    # Index 0~3.
Out[5]: 'Life'
```

```
In[6] : str[:17]                    # Index 0~16.
Out[6]: 'Life is too short'
```

```
In[7] : str[19:]                    # Index 19~end.
Out[7]: 'You need Python!'
```

- An index range a:b includes a and excludes b.

Basic Data Types : String (8/11)

Python string methods:

```
In[1] : x = 'Python'
In[2] : len(x)                    # String length. This is a Python built-in function.
Out[2]: 6
```

```
In[3] : x.count('h')              # Counts the 'h' character.
Out[3]: 1
```

```
In[4] : x.upper()                 # Convert to the uppercase.
Out[4]: 'PYTHON'
```

```
In[5] : x.lower()                 # Convert to the lowercase.
Out[5]: 'python'
```

Basic Data Types : String (9/11)

Python string methods:

```
In[1] : x = 'Python'
In[2] : x.find('o')                                  # Position of the 'o'.
Out[2]: 4

In[3] : x.find('w')                                  # Look for the position of the 'w'. No match.
Out[3]: -1

In[4] : x.index('o')                                 # Look for the position of the 'o'.
Out[4]: 4

In[5] : x.index('w')                                 # Look for the position of the 'w'. No match.
ValueErrorTraceback (most recent call last)
<ipython-input-46-47c892f4e96d> in <module>()
----> 1 x.index('w')
ValueError: substring not found
```

Basic Data Types : String (10/11)

Python string methods:

```
In[1] : x = 'Life is too short, You need Python!'
In[2] : x.split(' ')                                    # The output is a list (*).
Out[2]: ['Life', 'is', 'too', 'short,', 'You', 'need', 'Python!']
```

```
In[3] : y = x.split(' ')
In[4] : a = ' '
In[5] : a.join(y)
Out[5]: 'Life is too short, You need Python!'
```

(*) We will learn more about it later.

Basic Data Types : String (11/11)

Some of the useful Python string methods:

- For a more extensive coverage please go to https://www.python.org/

Function	Explanation
x.lstrip()	Trim left white spaces.
x.rstrip()	Trim right white spaces.
x.strip()	Trim white spaces from both sides.
x.replace(str1, str2)	Replace string from str1 to str2.
x.count(str)	Count the string pattern.
x.find(str)	Position of the first match. -1 if no match is found.
x.index(str)	Position of the first match. Error if no match is found.
x.join(str_list)	Join the string list items using x as connector.
x.split(y)	Split a string by y.
x.upper()	Convert to the upper case.
x.lower()	Convert to the lower case.
len(x)	String length. This is not a string method but a built-in Python function.

Basic Data Types : Boolean (1/6)

Boolean data type:

- Can have only two values: True or False.
- Notice that True/False has the capitalized first character.
- Boolean types are often used to construct conditional expressions.

Example: `2 == 3` False

`1 < 2` True

Basic Data Types : Boolean (2/6)

Boolean and, or, not operators:

```
In[1] : True and False                    # and.
Out[1]: False
In[2] : True and True
Out[2]: True
In[3] : False or False
Out[3]: False                    # or.
In[4] : False or True
Out[4]: True
In[5] : not False                         # not.
Out[5]: True
```

Basic Data Types : Boolean (3/6)

Assume that X and Y are Boolean statements, then the truth table of the and operator is:

X	Y	X and Y
False	False	False
False	True	False
True	False	False
True	True	True

Basic Data Types : Boolean (4/6)

Assume that X and Y are Boolean statements, then the truth table of the or operator is:

X	Y	X or Y
False	False	False
False	True	True
True	False	True
True	True	True

Basic Data Types : Boolean (5/6)

Assume that X is a Boolean statement, then the truth table of the not operator is:

X	not X
False	True
True	False

Basic Data Types : Boolean (6/6)

Boolean values are implicitly assumed in the following cases:

Example	True/False	Explanation
"Python"	True	Not empty string.
""	False	Empty string.
[ ]	False	Empty list (*).
( )	False	Empty tuple (*).
{ }	False	Empty dictionary (*).
[1, 2, 3]	True	Not empty list (*).
0	False	-
1	True	-
None	False	-

(*) Later we will learn more about the list, tuple and dictionary.

Coding Exercise #0102

Follow practice steps on 'ex_0102.ipynb'

Python Programming

UNIT 1. Python I

1.1. About Python Programming Language.

1.2. A Quick Look at Python.

1.3. Jupyter Notebook.

Unit 2. Python II

2.1. Basic Data Types.

2.2. Composite Data Types.

2.3. Data Type Mutability.

Composite Data Types in Python

There are 4 composite data types in Python:

- List: stores multiple values that may or may not be of the same type.
- Tuple: similar to the list with some restrictions. Tuples are immutable objects (*).
- Dictionary: associative array.
- Set: unordered collection of values without repetition.

(*) More about this later.

Composite Data Types : List (1/10)

A list contains comma separated values within square brackets.

```
In[1] : a = [ ]                                  # An empty list.
In[2] : b = [1, 2, 3]                            # A list with numeric values.
In[3] : c = ['Life', 'is', 'too', 'short']       # A list with strings.
In[4] : d = [1, 2, 'Life', 'is']                 # A list with mixed data type values.
In[5] : e = [1, 2, ['Life', 'is']]               # A nested list.
```

```
In[6] : e[1]                                     # Indexing of a list.
Out[6]: 2
In[7] : e[2]
Out[7]: ['Life', 'is']
In[8] : e[2][1]
Out[8]: 'is'
```

- In Python, indices start from 0. Lists are no exception.

Composite Data Types : List (2/10)

Slicing a list:

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : a[:2]                    # From the index 0 to 1 (inclusive).
Out[2]: [1, 2]
In[3] : a[2:]                    # From the index 2 to the end.
Out[3]: [3, 4, 5]
In[4] : a[:]                     # The whole list!
Out[4]: [1, 2, 3, 4, 5]
```

```
In[1] : a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
In[2] : a[2:5]
Out[2]: [3, ['a', 'b', 'c'], 4]
In[3] : a[3][:2]
Out[3]: ['a', 'b']
```

Composite Data Types : List (3/10)

Concatenation (+) and Repetition (*):

```
In[1] : a = [1, 2, 3]
In[2] : b = [4, 5, 6]
In[3] : a + b
Out[3]: [1, 2, 3, 4, 5, 6]
```

```
In[1] : a = [1, 2, 3]
In[2] : a * 3
Out[2]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Composite Data Types : List (4/10)

Modifying a list: change one or multiple entries.

```
In[1] : a = [1, 2, 3]
In[2] : a[1] = -1
In[3] : a
Out[3]: [1, -1, 3]
```

```
In[4] : a[1:3]
Out[4]: [-1,3]
In[5] : a[1:3] = [3, 5, 7, 9, 11]
In[6] : a
Out[6]: [1, 3, 5, 7, 9, 11]
```

Composite Data Types : List (5/10)

Modifying a list: careful with the following differences.

```
In[1] : a = [1, 2, 3]
In[2] : a[1:2] = [-1, -2, -3]                    # Partial change of the list values.
In[3] : a
Out[3]: [1, -1 , -2, -3, 3]
```

```
In[1] : a = [1, 2, 3]
In[2] : a[1] = [-1, -2, -3]                      # An entry is replaced by another list.
In[3] : a
Out[3]: [1, [-1 , -2, -3], 3]
```

Composite Data Types : List (6/10)

Modifying a list: partial removal.

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : a[1:3] = []                    # Partial removal.
In[3] : a
Out[3]: [1, 4, 5]
```

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : del a[2]                       # Removes a specific entry.
In[3] : a
Out[3]: [1, 2, 4, 5]
```

Composite Data Types : List (7/10)

Python list methods:

```
In[1] : a = [3, 1, 5, 2, 4]
In[2] : a.sort()                    # Sorting of a list object (lasting effect).
In[3] : a
Out[3]: [1, 2, 3, 4, 5]
```

```
In[1] : a = [1, 2, 3]
In[2] : a.append(4)                 # A value is added as an entry.
In[3] : a.append([5,6])             # A list is added as an entry.
In[4] : a
Out[4]: [1, 2, 3, 4, [5, 6]]
```

Composite Data Types : List (8/10)

Python list methods:

```
In[1] : a = [3, 1, 5, 2, 4]
In[2] : a.sort(reverse=True)                    # Sorting in reverse order.
In[3] : a
Out[3]: [5, 4, 3, 2, 1]
```

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : a.reverse()                             # Reverse the order. Different from sorting.
In[3] : a
Out[3]: [5, 4, 3, 2, 1]
```

Composite Data Types : List (9/10)

Python list methods:

```
In[1] : a = [3, 1, 5, 2, 4]
In[2] : sorted(a)                    # Displays a sorted view.
Out[2]: [1, 2, 3, 4, 5]
In[3] : a
Out[3]: [3, 1, 5, 2, 4]              # No permanent changes.
```

Composite Data Types : List (10/10)

Some of the useful Python list methods:

- For a more extensive coverage please go to https://www.python.org/

Function	Explanation
x.insert(pos, val)	Insert a value at a specified position.
x.remove(val)	Remove a specific value from a list.
x.pop()	Remove and return the last value of a list.
x.count(val)	Count the appearances of a value in a list.
x.extend(y)	Extend the list x with another list y. Means x = x + y.
x.append(val)	Appends a value as an entry.
x.sort()	Sorting of a list.
x.reverse()	Reversal of the entry ordering.
x.index(val)	Looks for the position of a specific value. Error if no match is found.
len(x)	Total number of entries in a list. A built-in Python function.

Composite Data Types : Tuple (1/5)

- Tuples are similar to the lists with some restrictions.
- A tuple contains comma separated values within parentheses.
- However, a tuple is an immutable object: you cannot change its content once it is created.

Composite Data Types : Tuple (2/5)

A tuple contains comma separated values within parentheses.

```
In[1] : a = ( )                                  # An empty tuple.
In[2] : b = (1, 2, 3)                            # A tuple with numeric values.
In[3] : c = ('Life', 'is', 'too', 'short')       # A tuple with strings.
In[4] : d = (1, 2, 'Life', 'is')                 # A tuple with mixed data type values.
In[5] : e = (1, 2, ('Life', 'is'))               # A nested tuple.
```

```
In[6] : e[1]                                     # Indexing of a tuple.
Out[6]: 2
In[7] : e[2]
Out[7]: ('Life', 'is')
In[8] : e[2][1]
Out[8]: 'is'
```

- In Python, indices start from 0. Tuples are no exception.

Composite Data Types : Tuple (3/5)

Slicing a tuple:

```
In[1] : a = (1, 2, 3, 4, 5)
In[2] : a[:2]                    # From the index 0 to 1 (inclusive).
Out[2]: (1, 2)
In[3] : a[2:]                    # From the index 2 to the end.
Out[3]: (3, 4, 5)
In[4] : a[:]                     # The whole tuple!
Out[4]: (1, 2, 3, 4, 5)
```

```
In[1] : a = (1, 2, 3, ('a', 'b', 'c'), 4, 5)
In[2] : a[2:5]
Out[2]: (3, ('a', 'b', 'c'), 4)
In[3] : a[3][:2]
Out[3]: ('a', 'b')
```

Composite Data Types : Tuple (4/5)

Concatenation (+) and Repetition (*):

```
In[1] : a = (1, 2, 3)
In[2] : b = (4, 5, 6)
In[3] : a + b
Out[3]: (1, 2, 3, 4, 5, 6)
```

```
In[1] : a = (1, 2, 3)
In[2] : a * 3
Out[2]: (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Composite Data Types : Tuple (5/5)

Errors occur in the following cases:

```
In[1] : a = (1, 2, 3)
In[2] : a[1] = -1                                   # Will result in an error.

TypeErrorTraceback (most recent call last)
<ipython-input-134-c684cd0a8714> in <module>()
----> 1 a[1] = -1
TypeError: 'tuple' object does not support item assignment
```

```
In[1] : a = (1, 2, 3)
In[2] : del a[1]                                    # Will result in an error.

TypeErrorTraceback (most recent call last)
<ipython-input-135-dbf82171d8ac> in <module>()
----> 1 del a[1]
TypeError: 'tuple' object doesn't support item deletion

In[3] : del a                                       # Delete the object. OK!
```

Composite Data Types : Dictionary (1/6)

- Python dictionary is an implementation of associative array.
- Dictionary is a collection of key and value pairs.

```
{key1:value1, key2:value2, key3:value3,…}
```

- In a dictionary values are accessed by their associated keys.
- Dictionaries cannot be concatenated using + nor can be repeated using *.
  - By the way, strings, lists, tuples can be concatenated and repeated.

Composite Data Types : Dictionary (2/6)

Dictionaries are comma separated key:value pairs contained within curly braces {}.

- A colon ':' separates the key from its associated value.

```
In[1] : a = { }                                          # An empty dictionary.
In[2] : b = {'NAME': 'JAMES', 'GENDER': 'MALE', 'AGE': 35}   # Numbers and strings as values.
In[3] : c = {1: 'Life', 2: 'is', 3: 'short' }
In[4] : d = {'x':[1,2,3], 'y':[4,5,6]}                   # Lists are used as values.
In[5] : e = {1:b, 2:c}                                   # Dictionaries are used as values.
```

```
In[6] : b['NAME']                                        # Dictionary indexing with a key.
Out[6]: 'JAMES'
In[7] : e[2]
Out[7]: {1: 'Life', 2: 'is', 3: 'short' }
In[8] : e[2][3]
Out[8]: 'short'
```

Composite Data Types : Dictionary (3/6)

Adding, deleting and changing dictionary pairs:

```
In[1] : a = {}                                                  # An empty dictionary.
In[2] : a['NAME'] = 'JAMES'                                     # Add a pair.
In[3] : a['GENDER'] = 'MALE'                                    # Add a pair.
In[4] : a['AGE'] = 35                                           # Add a pair.
In[5] : a
Out[5]: {'NAME': 'JAMES', 'GENDER': 'MALE', 'AGE': 35}
```

```
In[6] : del a['AGE']                                            # Delete a pair.
In[7] : a
Out[7]: {'NAME': 'JAMES', 'GENDER': 'MALE'}
In[8] : a['NAME'] = 'JOHN'                                      # Change the value of a pair.
In[9] : a
Out[9]: {'NAME': 'JOHN', 'GENDER': 'MALE'}
```

Composite Data Types : Dictionary (4/6)

Python dictionary methods:

```
In[1] : a = {'NAME': 'JAMES', 'GENDER': 'MALE', 'AGE': 35}
In[2] : a.keys()                                          # Display the keys.
Out[2]: dict_keys(['NAME', 'GENDER', 'AGE'])
```

```
In[3] : a.values()                                        # Display the values.
Out[3]: dict_values(['JAMES', 'MALE', 35])
```

```
In[4] : a.items()                                         # Show the content of a dictionary.
Out[4]: dict_items([('NAME', 'JAMES'), ('GENDER': 'MALE'), ('AGE': 35)])
```

```
In[5] : a.clear()                                         # Clear all the pairs.
In[6] : a
Out[6]: {}
```

Composite Data Types : Dictionary (5/6)

Python dictionary methods:

```
In[1] : a = {'name': 'JONE', 'gender': 'MALE', 'age': 35}
In[2] : a['wage']                                          # Error occurs.
KeyErrorTraceback (most recent call last)
<ipython-input-189-127109090150> in <module>()
----> 1 a['wage']

KeyError: 'wage'
```

```
In[3] : a.get('name')                                      # Similar to a['name'].
Out[3]: 'JONE'
In[4] : a.get('wage')                                      # Similar to a['wage'].
                                                           # No error occurs. Instead None is returned.
```

```
In[5] : a.get('wage', 0)                                   # 0 is set as the default value.
Out[5]: 0                                                  # The default value when no matching key is found.
```

Composite Data Types : Dictionary (6/6)

Check whether a given key exists in the dictionary: in operator.

```
In[1] : a = {'name': 'JOHN', 'gender': 'MALE', 'age': 35}
In[2] : 'name' in a                                          # Is 'name' among the keys?
Out[2]: True
In[3] : 'wage' in a                                          # Is 'wage' among the keys?
Out[3]: False
```

Composite Data Types : Set (1/6)

Characteristics of a set:

- Elements are not repeated.
- There is no concept of ordering (sorting) nor indexing.
- Regarding a set, the meaningful question to ask is whether a value belongs to it or not.

Composite Data Types : Set (2/6)

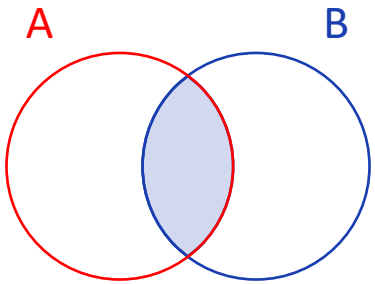
- A set is displayed as a series of comma separated values within the curly braces {}.
- However, use set() function to create a new set object.

```
In[1]  : a = set()                          # An empty set.
In[2]  : b = set([1,2,3,3,3,3,4,5])         # A set is created from a list.
In[3]  : b
Out[3]:  {1,2,3,4,5}                        # No repeated elements!
In[4]  : type(b)
Out[3]:  set
```

Composite Data Types : Set (3/6)

Set operations: intersection, union and difference.

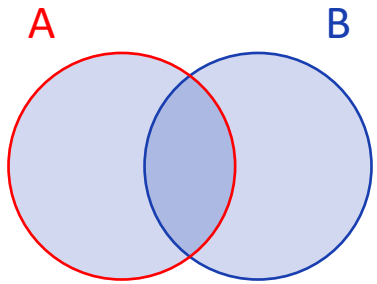
```
In[1] : s1 = set([1, 2, 3, 4, 5])
In[2] : s2 = set([4, 5, 6, 7, 8])
In[3] : s1 & s2                                  # Intersection of s1 and s2.
Out[3]: {4, 5}
In[4] : s1.intersection(s2)                      # The same as above.
Out[4]: {4, 5}
```



Composite Data Types : Set (4/6)

Set operations: intersection, union and difference.

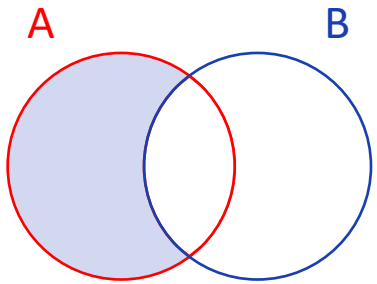
```
In[5] : s1 | s2                    # Union of s1 and s2.
Out[5]: {1, 2, 3, 4, 5, 6, 7, 8}
In[6] : s1.union(s2)               # The same as above.
Out[7]: {1, 2, 3, 4, 5, 6, 7, 8}
```

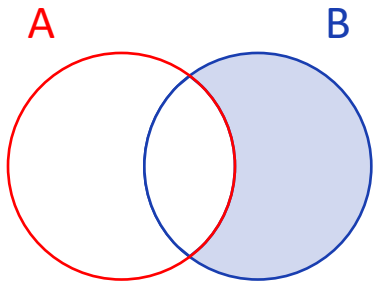


Composite Data Types : Set (5/6)

Set operations: intersection, union and difference.

```
In[8] : s1 - s2                    # Set difference.
Out[8]: {1, 2, 3}
In[9] : s1.difference(s2)          # The same as above.
Out[9]: {1, 2, 3}
In[10] : s2 – s1
Out[10]: {6, 7, 8}                 # For a difference, the order matters.
```





Composite Data Types : Set (6/6)

Python set methods.

```
In[1] : a = set([1, 2, 3, 4, 5])
In[2] : a.add(6)                          # Adds a value to the set.
In[3] : a
Out[3]: {1, 2, 3, 4, 5, 6}
```

```
In[4] : a.update([7, 8, 9])               # Adds several values at once.
In[5] : a
Out[5]: {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
In[6] : a.remove(9)                       # Removes the specified value.
In[7] : a
Out[7]: {1, 2, 3, 4, 5, 6, 7, 8}
```

Coding Exercise #0103

Follow practice steps on 'ex_0103.ipynb'

Python Programming

UNIT 1. Python I

1.1. About Python Programming Language.

1.2. A Quick Look at Python.

1.3. Jupyter Notebook.

Unit 2. Python II

2.1. Basic Data Types.

2.2. Composite Data Types.

2.3. Data Type Mutability.

Data Type Mutability (1/13)

Immutable data types:

- Number (int, float), Boolean (bool), string (str), tuple and Unicode are immutable.
- Variables of a mutable data type cannot be changed after being created.
- When you assign a new value to a immutable variable, a new object is created (with a new address).

Data Type Mutability (2/13)

Immutable data types:

```
In[1]  : a = 999                    # Number (int).
In[2]  : b = a
In[3]  : id(a)                      # ID.
Out[3]: 60560760L
In[4]  : id(b)                      # Shared ID between a and b.
Out[4]: 60560760L
```

```
In[5]  : b = 0                      # A new value assigned to b.
In[6]  : b
Out[6]: 0
In[7]  : a                          # Value of a remains the same as before.
Out[7]: 999
In[8]  : id(b)                      # A new ID for b. Thus, it's a new object.
Out[8]: 32333952L
```

Data Type Mutability (3/13)

Immutable data types:

```
In[1]  : a = 'abcd'                    # String (str).
In[2]  : b = a
In[3]  : id(a)                         # ID.
Out[3]: 87535088L
In[4]  : id(b)                         # Shared ID between a and b.
Out[4]: 87535088L
```

```
In[5]  : b = b + 'e'                   # This will change the ID. Thus, a new object.
In[6]  : b
Out[6]: 'abcde'
In[7]  : a                             # Value of a remains the same as before.
Out[7]: 'abcd'
In[8]  : id(b)
Out[8]: 87535368L                      # Verify the ID change for b.
```

Data Type Mutability (4/13)

Mutable data types:

- List, dictionary, set, etc. are mutable.
- Also custom classes are generally mutable.
- Variables of a mutable data type can be changed after being created.
- When you assign a new value to a mutable variable, the object remains with the same address (ID).

Data Type Mutability (5/13)

Mutable data types:

```
In[1]  : a = [1, 2, 3, 4, 5]                    # A list object.
In[2]  : b = a
In[3]  : id(a)                                  # ID.
Out[3]: 87729352L
In[4]  : id(b)                                  # Shared ID between a and b.
Out[4]: 87729352L
```

```
In[5]  : b[0] = -999                            # Change a value from b.
In[6]  : b
Out[6]: [-999, 2, 3, 4, 5]
In[7]  : a                                      # The same change shows up in a!
Out[7]: [-999, 2, 3, 4, 5]
In[8]  : id(b)
Out[8]: 87729352L                               # The ID of b is maintained!
```

Data Type Mutability (6/13)

Mutable objects within another immutable object:

```
In[1] : a = (1, 2, [3, 4] )          # A list (mutable) within a tuple (immutable).
In[2] : b = a
In[3] : id(a)                        # ID.
Out[3]: 86385864L
In[4] : id(b)                        # Shared ID between a and b.
Out[4]: 86385864L
```

```
In[5] : b[2][0] = -999               # Change a value within the mutable list!
In[6] : b
Out[6]: (1, 2, [-999,4])
In[7] : a
Out[7]: (1, 2, [-999,4])             # Mutable lists are actually the same!
In[8] : id(b)
Out[8]: 86385864L                    # Immutable tuple ID remains the same.
```

Data Type Mutability (7/13)

Mutable objects within another immutable object:

```
In[9]  : b[2] = [4, 5]                                  # Immutable elements of a tuple cannot be changed.

TypeErrorTraceback (most recent call last)
<ipython-input-359-b52acb363bbe> in <module>()
---> 1 b[2] = [4,5]
TypeError: 'tuple' object does not support item assignment
```

Data Type Mutability (8/13)

Variables:

```
In[1] : a = 'abcd'                    # A string object.
In[2] : b = 3
In[3] : type(a)                       # Show the data type.
Out[3]: str
In[4] : id(a)                         # ID.
Out[4]: 87535088L
In[5] : id(b)                         # A different ID for b. No surprise here.
Out[5]: 60535011L
In[6] : c = b
In[7] : b is c                        # Will return True.
Out[7]: True
In[8] : a is b                        # Will return False.
Out[8]: False
In[9] : del a                         # Delete a.
```

Data Type Mutability (9/13)

Variables:

```
In[1] : a, b, c = (111, True, 'aaa')          # Define multiple variables at once.
In[2] : a, b, c = [111, True, 'aaa']          # Define multiple variables at once.
In[3] : a = b = c = 777                       # Define multiple variables with the same value.
In[4] : x, y = 666, 777
In[5] : x, y = y, x                           # Swapping.
In[6] : x, y
Out[6]: (777, 666)
```

Data Type Mutability (10/13)

How to copy an object (variable):

- Simple assignment: the names are different but share the same ID (memory address).
- Shallow copy: creates a new composite data type object while the elements remain the same objects as before.
- Deep copy: creates a new composite data type object with newly created objects as elements.

Data Type Mutability (11/13)

How to copy an object (variable): Simple assignment.

```
In[1] : a = {'name': 'JOHN', 'gender': 'MALE', 'age': 35}          # A dictionary object.
In[2] : b = a                                                      # Simple assignment.
In[3] : a['name'] = 'JACK'
In[4] : a
Out[4]: {'age': 35, 'gender': 'MALE', 'name': 'JACK'}
In[5] : b
Out[5]: {'age': 35, 'gender': 'MALE', 'name': 'JACK'}
In[6] : id(a)
Out[5]: 86349208L
In[7] : id(b)                                                      # Same object under different names.
Out[5]: 86349208L
```

Data Type Mutability (12/13)

How to copy an object (variable): Shallow copy.

```
In[1] : a = [1, 2, [3, 4, 5]]          # A nested list.
In[2] : b = a[:]                       # Shallow copy.
In[3] : a[0] = 0                       # An immutable element is changed.
In[4] : a[2][0] = -999                 # An element from the mutable list is changed.
In[5] : a                              # Immutable element splits into different objects.
Out[5]: [0, 2, [-999, 4, 5]]
In[6] : b                              # Mutable element is still shared.
Out[6]: [1, 2, [-999, 4, 5]]
```

Data Type Mutability (13/13)

How to copy an object (variable): Deep copy.

```
In[1] : import copy                          # Import the copy module.
In[2] : a = [1, 2, [3, 4, 5]]                # A nested list.
In[3] : b = copy.deepcopy(a)                 # Deep copy.
In[4] : a[2][0] = 0
In[5] : a
Out[5]: [1, 2, [0, 4, 5]]
In[6] : b
Out[6]: [1, 2, [3, 4, 5]]
In[7] : id(a[2])
Out[7]: 87731848L
In[8] : id(b[2])                             # Mutable elements but different objects.
Out[8]: 86205960L
```

Coding Exercise #0104

Follow practice steps on 'ex_0104.ipynb'

SAMSUNG

Together for Tomorrow!

Enabling People

Education for Future Generations