

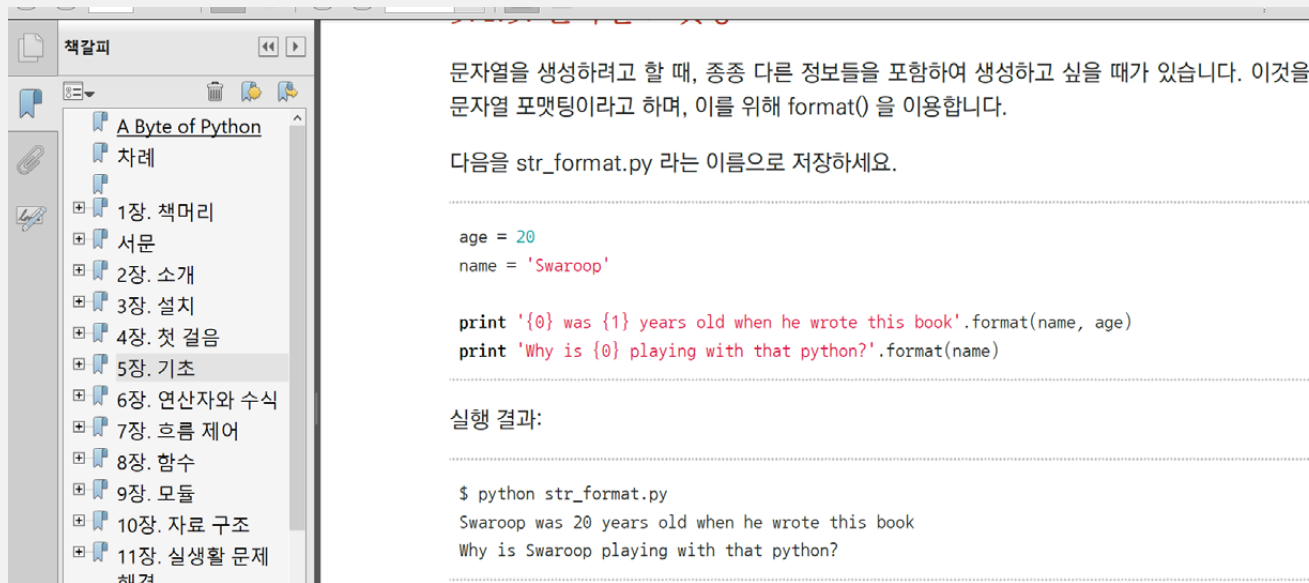
수치해석

(2019학년도 1학기)

[4주/1차시 학습내용]: 숙제내용, Numpy,
Arrays in Python

Homework #1

- Python Textbook 1 : byte_of_python
- Ch 5, ch 6, ch 7, ch 8, ch 9
 - 실행 결과 있는 예제 수행 한 후, 화면 결과와 소감을 소스 코드와 함께 문서작업하여 가상강의실에 제출 (1)
- Due Date: 4월 11일 수업 시작 전에 A4용지로 제출 (2)



print 다음에는 () 괄호가 필요

5.4.5. 문자열 포매팅

문자열을 생성하려고 할 때 종종 다른 저널의 표기법을 새겨놓고 쓰는 때가 있습니다. 이것은 문자열 포매팅이라고 하며,

다음은 str_format.py 라는

```
age = 20
```

```
name = 'Swaroop'
```

```
print '{0} was {1} years
```

```
print 'Why is {0} playing
```

실행 결과:

```
$ python str_format.py
```

```
Swaroop was 20 years old
```

```
Why is Swaroop playing w
```

IPython

```
Type "copyright", "credits" or "license" for more information.
```

```
IPython 4.2.0 -- An enhanced Interactive Python.
```

```
--> Introduction and overview of IPython's features.
```

```
%quickref --> Quick reference.
```

```
help --> Python's own help system.
```

```
object? --> Details about 'object', use 'object??' for extra details.
```

```
In [1]: age=20
```

```
In [2]: name='Swaroop'
```

```
In [3]: print '{0} was {1} years old when he wrote this book'.format(name, age)
```

```
File "<ipython-input-3-46312128de35>", line 1
```

```
print '{0} was {1} years old when he wrote this book'.format(name, age)
```

```
SyntaxError: invalid syntax
```

```
In [4]: print "{0} was {1} years old when he wrote this book".format(name, age)
```

```
File "<ipython-input-4-ab09675c07b0>", line 1
```

```
print "{0} was {1} years old when he wrote this book".format(name, age)
```

```
SyntaxError: invalid syntax
```

```
In [5]: print ('{0} was {1} years old when he wrote this book'.format(name, age))
```

```
Swaroop was 20 years old when he wrote this book
```

```
In [6]:
```

print 다음에는 () 괄호가 필요

```
In [6]: print(name + ' is ' + str(age) + ' years old')
Swaroop is 20 years old
```

```
.....
# 소수점 이하 셋째 자리까지 부동 소숫점 숫자 표기 (0.333)
print '{0:.3f}'.format(1.0/3)
# 밑줄(_)로 11칸을 채우고 가운데 정렬(^)하기 (__hello__)
print '{0:_^11}'.format('hello')
# 사용자 지정 키워드를 이용해 (Swaroop wrote A Byte of Python) 표기
print '{name} wrote {book}'.format(name='Swaroop',
                                   book='A Byte of Python')
.....
```

실행 결과:

```
.....
0.333
__hello__
Swaroop wrote A Byte of Python
.....
```

Homework #2

- Python Textbook 2: numpy
- Page 12 까지
 - 실행 결과 있는 예제 수행 한 후, 화면 결과와 소감을 소스 코드와 함께 문서작업하여 가상강의실에 제출 (1)
- Due Date: 4월 18일 수업 시작 전에 A4용지로 제출 (2)

다음은 str_format.py 라는 이름으로 저장하세요.

```
age = 20
name = 'Swaroop'

print '{0} was {1} years old when he wrote this book'.format(name, age)
print 'Why is {0} playing with that python?'.format(name)
```

실행 결과:

```
$ python str_format.py
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

Homework #3

- Python Homework Textbook 2: numpy
- Page 24 까지
 - 실행 결과 있는 예제 수행 한 후, 화면 결과와 소감을 소스 코드와 함께 문서작업하여 가상강의실에 제출 (1)
- Due Date: 5월 2일 수업 시작 전에 A4용지로 제출 (2)

다음은 str_format.py 라는 이름으로 저장하세요.

```
age = 20
name = 'Swaroop'

print '{0}| was {1} years old when he wrote this book'.format(name, age)
print 'Why is {0} playing with that python?'.format(name)
```

실행 결과:

```
$ python str_format.py
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

Homework #4

- Python Homework Textbook 1 :
- byte_of_python
 - Ch 10, ch 11, ch 12, ch 13, ch 14
 - 실행 결과 있는 예제 수행 한 후, 화면 결과와 소감을 소스 코드와 함께 문서작업하여 가상강의실에 제출 (1)
- Due Date: 5월 9일 수업 시작 전에 A4용지로 제출 (2), 연장 없음
- 한 번에 #1, #2, #3, #4를 제출하는 학생은, 숫자로 숙제 번호를 반드시 구별할 것
 - 구별하지 않을 시 한 번 제출한 것으로 채점됨

예제 (ds_using_list.py 로 저장하세요):

```
# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print 'I have', len(shoplist), 'items to purchase.'

print 'These items are:',
for item in shoplist:
    print item,

print '\nI also have to buy rice.'
shoplist.append('rice')
print 'My shopping list is now', shoplist

print 'I will sort my list now'
shoplist.sort()
print 'Sorted shopping list is', shoplist

print 'The first item I will buy is', shoplist[0]
olditem = shoplist[0]
del shoplist[0]
print 'I bought the', olditem
print 'My shopping list is now', shoplist
```

실행 결과:

```
$ python ds_using_list.py
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
```

들여쓰기(Indentation)

- 파이썬에서 공백은 중요한 역할을 합니다.
 - 사실, 한 행의 앞에 붙어있는 공백이 정말 중요합니다.
- 이것을 _들여쓰기_라 부릅니다.
- 한 논리적 명령행의 앞에 붙어있는 공백 (빈 칸 혹은 탭)은 논리적 명령행의 들여쓰기 단계를 의미하며, 이것은 한 명령의 범위를 구분하는 데 사용됩니다.

들여쓰기(Indentation)

- 이것은 같은 들여쓰기 단계에 있는 명령들은 반드시 같은 들여쓰기를 사용해야 함을 의미합니다.
- 이러한 같은 들여쓰기를 사용하고 있는 명령들의 집합을 **블록 (block)** 이라고 부릅니다.

들여쓰기(Indentation)

```
i = 5
# 다음 행에서 오류가 발생합니다! 행 앞에 잘못된 공백이 한 칸 있습니다.
print 'Value is ', i
print 'I repeat, the value is ', i
```

위 예제를 실행하면 다음과 같이 오류가 발생합니다.

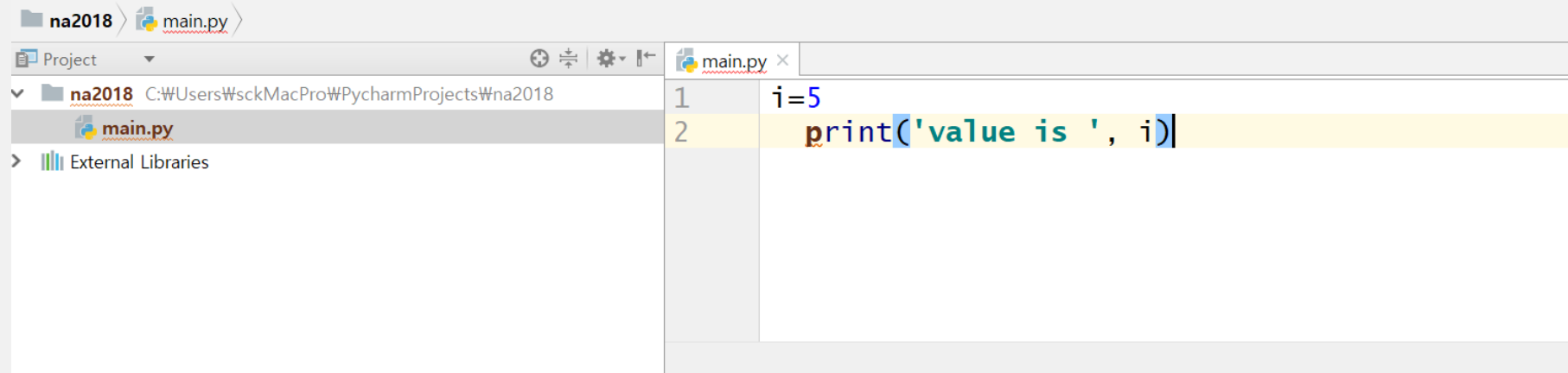
```
File "whitespace.py", line 5
    print 'Value is ', i
    ^
```

IndentationError: unexpected indent

두 번째 행 앞에 공백이 한 칸 있다는 점을 확인
들여쓰기 하는 법: 들여쓰기를 할 때에는 공백 4개를 이용

들여쓰기(Indentation)

File Edit View Navigate Code Refactor Run Tools VCS Window Help



Run main

```
C:\Anaconda3\python.exe C:/Users/sckMacPro/PycharmProjects/na2018/main.py
File "C:/Users/sckMacPro/PycharmProjects/na2018/main.py", line 2
    print('value is ', i)
    ^
IndentationError: unexpected indent

Process finished with exit code 1
```

흐름 제어와 들여 쓰기

```
number = 23
g = input('Give me a number: ')
guess=int(g)

print("Entered Value is ",guess)

if guess == number:
    # New block starts here
    print('Congratulations, you guessed it.')
    print('but you do not win any prizes!')
    # New block ends here

elif guess < number:
    # Another block
    print('No, it is a little higher than that')

else:
    # Another block
    print('No, it is a little lower than that')

print('Done')
# This last statement is always executed,
# after the if statement is executed.
```

Give me a number: 34
Entered Value is 34
No, it is a little lower than that
Done

Process finished with exit code 0

Function (함수)

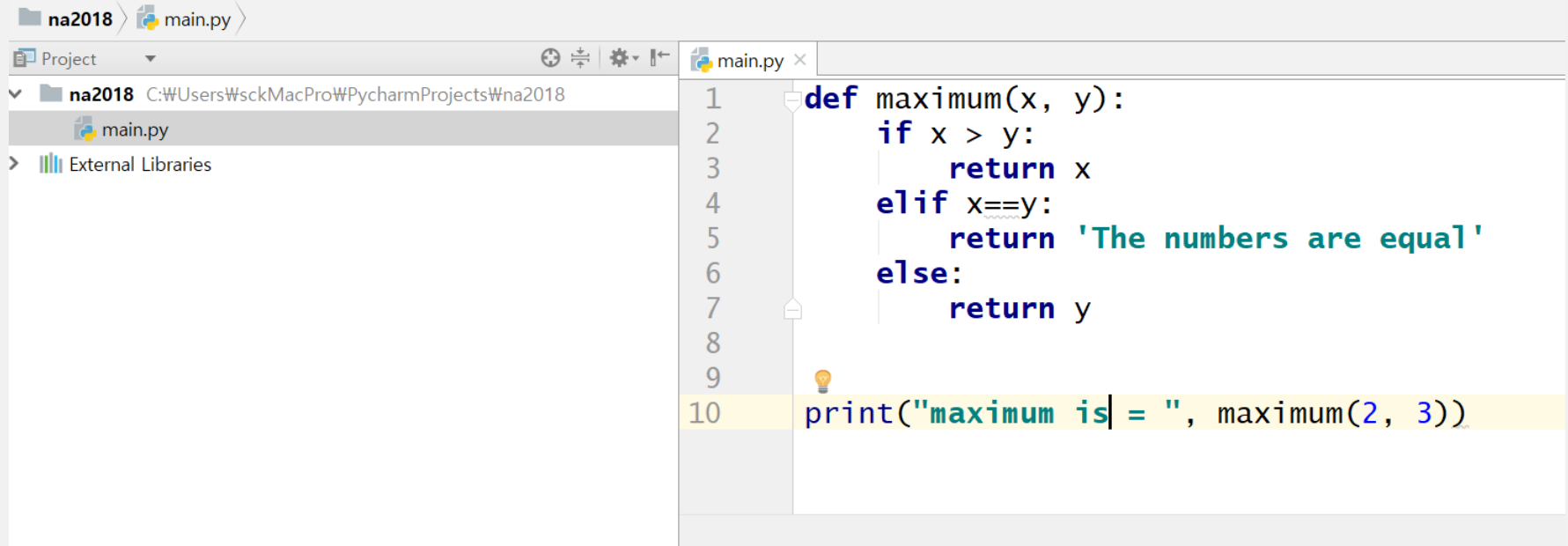
- 재사용 가능한 프로그램의 조각
 - Reusable unit of program
- 특정 블록의 명령어 덩어리를 묶어 이름을 짓고, 그 이름을 프로그램 어디에서건 사용함
- 그 블록에 포함된 명령어들을 몇 번이고 다시 실행할 수 있게 하는 것
- 이를 보고 함수를 ‘호출한다’라고 함
 - 사실 우리는 이미 앞에서 `size`이나 `range` 와 같은 많은 내장 함수들을 사용해 왔음.

Function (함수)

- 함수는 def 키워드를 통해 정의됨
 - Use 'def' keyword
- def 뒤에는 함수의 식별자 이름을 입력하고, 괄호로 감싸여진 함수에서 사용될 인자(arguments)의 목록을 입력
- 마지막으로 콜론을 입력하면 함수의 정의가 끝남.
 - `def function_name(arguments):`
- 새로운 블록이 시작되는 다음 줄부터는 이 함수에서 사용될 명령어들을 입력함.

함수와 Return

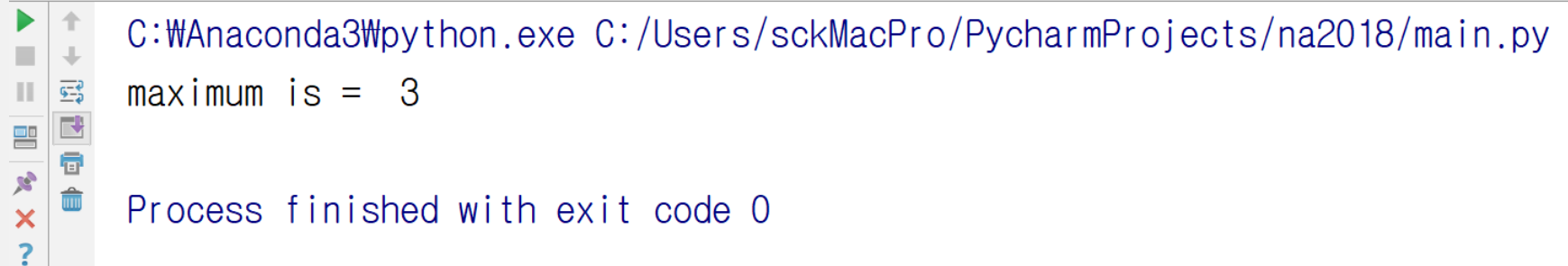
File Edit View Navigate Code Refactor Run Tools VCS Window Help



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The left sidebar shows a project named 'na2018' with a file 'main.py' selected. The main editor window displays the following Python code:

```
1 def maximum(x, y):
2     if x > y:
3         return x
4     elif x==y:
5         return 'The numbers are equal'
6     else:
7         return y
8
9
10 print("maximum is | = ", maximum(2, 3))
```

Run main



The Run console shows the execution of the script. The command executed is `C:\WAnaconda3\python.exe C:/Users/sckMacPro/PycharmProjects/na2018/main.py`. The output is `maximum is = 3`. The process finished with exit code 0.

Module

- 함수를 통해 여러분의 프로그램 안에서 코드를 재사용하는 방법에 대해서 배워 보았습니다.
- 그러면 여러 함수들을 한꺼번에 불러들여 재사용하는 방법은 없을까요? 네, 이럴 때 모듈을 이용합니다.
- 모듈을 작성하는 데에는 여러 가지 방법이 있습니다만, 가장 간단한 방법은 .py 확장자를 가진 파일을 하나 만들고 그 안에 함수들과 변수들을 정의해 두는 것입니다.

Import Module

- 모든 파이썬 프로그램은 곧 모듈이기 때문입니다.
- 즉 .py 확장자를 가진 이름으로 저장되기만 하면 됩니다.
- 다른 프로그램에서 import 명령을 통해 모듈을 불러와 사용할 수 있습니다.

Module

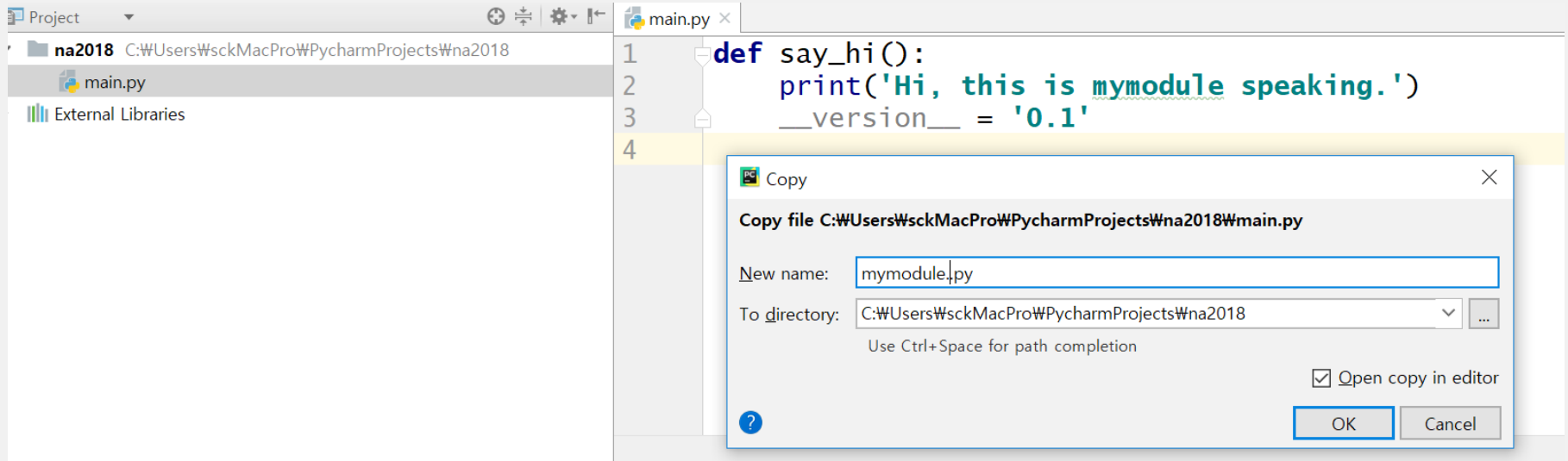
```
def say_hi():  
    print('Hi, this is mymodule speaking.')
```

```
__version__ = '0.1'
```

C:\Users\SCK_RetinaMAC\Anaconda3\python.exe C

Process finished with exit code 0

- Save as 'mymodule.py'
 - `print()`: 들여쓰기
 - `__version__`: 들여쓰기 하지 않음
- Run 'mymodule.py'



Run 'mymodule.py'

File Edit View Navigate Code Refactor Run Tools VCS Window Help

na2018 > mymodule..py

Project

- na2018 C:\Users\sckMacPro\PycharmProjects\na2018
 - main.py
 - mymodule..py
 - mymoduledemo.py
- External Libraries

main.py x mymodule..py x mymoduledemo.py x

```
1 def say_hi():  
2     print('Hi, this is mymodule speaking.')  
3     __version__ = '0.1'  
4
```

Run mymodule..

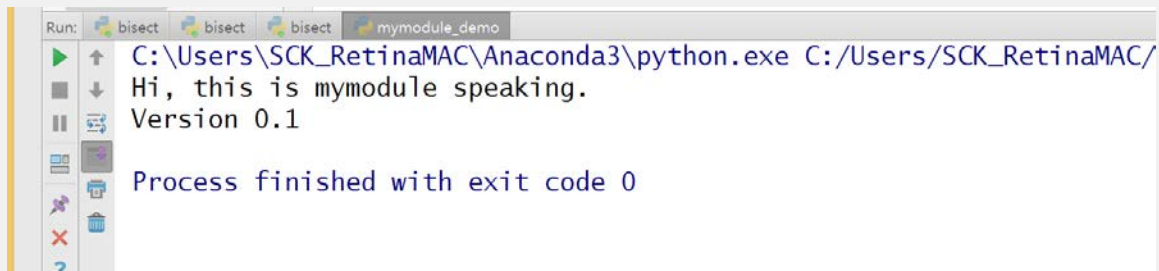
C:\Anaconda3\python.exe C:/Users/sckMacPro/PycharmProjects/na2018/mymodule..py

Process finished with exit code 0

Use Module

```
import mymodule  
  
mymodule.say_hi()  
print('Version', mymodule.__version__)
```

- Save as
‘mymoduledemo.py’
- Run
‘mymoduledemo.py’



Ch 2. Python Fundamental (Python Textbook 2: numpy, Arrays in Python)

Prof. Sang-Chul Kim

What is Numerical Python (Numpy)?

- The central feature of NumPy is the array object class.
- Arrays are similar to lists in Python,
 - except that every element of an array must be of the same type,
 - typically a numeric type like float or int.
- Arrays make operations with large amounts of numeric data very fast and are generally much more efficient than lists.

np.array([1,4,5,8], float)

- Here, the function array takes two arguments: the list to be converted into the array and the type of each member of the list.

```
import numpy as np
a=np.array([1,4,5,8], float)

type(a)
```

```
a
Out[10]: array([ 1.,  4.,  5.,  8.])

type(a)
Out[11]: numpy.ndarray
```

np.array([1,4,5,8], float)

```
a[:2]
array([ 1.,  4.])
a
array([ 1.,  4.,  5.,  8.])
a[3]
8.0
a[0]
1.0
a[0]=5.
a
Out[18]: array([ 5.,  4.,  5.,  8.])
```


Two Dimensional Array

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
```

```
a
```

```
array([[ 1.,  2.,  3.],  
       [ 4.,  5.,  6.]])
```

```
a[0,0]
```

```
1.0
```

```
a[0,1]
```

```
2.0
```

Two Dimensional Array

- When standard mathematical operations are used with arrays, they are applied on an element-by-element basis.

```
a = np.array([1,2,3], float)
a
array([ 1.,  2.,  3.])
b = np.array([5,2,6], float)
b
array([ 5.,  2.,  6.])
a+b
array([ 6.,  4.,  9.])

a-b
array([-4.,  0., -3.])
a*b
array([ 5.,  4., 18.])
b/a
array([ 5.,  1.,  2.])
a%b
array([ 1.,  0.,  3.])
b**a
array([ 5.,  4., 216.])
```

Two Dimensional Array

- For two-dimensional arrays, multiplication remains elementwise and does *not* correspond to matrix multiplication.

```
a = np.array([[1,2], [3,4]], float)
```

```
a
```

```
array([[ 1.,  2.],  
       [ 3.,  4.]])
```

```
b = np.array([[2,0], [1,3]], float)
```

```
b
```

```
array([[ 2.,  0.],  
       [ 1.,  3.]])
```

```
a*b
```

```
array([[ 2.,  0.],  
       [ 3., 12.]])
```

Array iteration

- Array iteration

```
array([1, 4, 5])  
for x in a:  
    print(x)
```

```
1  
4  
5
```

Minimum and maximum values

- find the minimum and maximum element values

```
a = np.array([2, 1, 9], float)
a.min()
1.0
a.max()
9.0
```

Minimum and maximum values

- The `argmin` and `argmax` functions return the array indices of the minimum and maximum values:

```
a = np.array([2, 1, 9], float)
a.argmin()
1
a.argmax()
2
```

Unique elements

- Unique elements can be extracted from an array:

```
a = np.array([1, 1, 4, 5, 5, 5, 7], float)
np.unique(a)
array([ 1., 4., 5., 7.])
```

Comparison

- Arrays can be compared to single values using broadcasting:

```
a = np.array([1, 3, 0], float)
a>2
array([False,  True, False], dtype=bool)
```


Nonzero()

- Nonzero()

```
a = np.array([[0, 1], [3, 0]], float)
a
array([[ 0.,  1.],
       [ 3.,  0.]])
In[572]: a.nonzero()
(array([0, 1], dtype=int64), array([1, 0], dtype=int64))
```

NaN ("not a number") or finite

- NaN ("not a number") or finite

```
a = np.array([1, np.NaN, np.Inf], float)

array([ 1., nan, inf])
np.isnan(a)
array([False,  True, False], dtype=bool)
np.isfinite(a)
array([ True, False, False], dtype=bool)
```