

Chapter 02

Python basic

Sejong Oh

Contents

- 01 Arithmetic operations
- 02 Variable
- 03 Data Type
- 04 print
- 05 list
- 06 tuple
- 07 if
- 08 for
- 09 Numpy vector
- 10 Array
- 11 Array operation
- 12 Slicing
- 13 조건을 만족하는 데이터의 수정
- 14 Help
- 15 Function
- 16 Store ndarray Object
- 17 Graph

00. Python

- **interpreted**, high-level, general-purpose programming language.
- Created by Guido van Rossum and first released in 1991
- Object-oriented programming and structured programming are fully supported
- Python 2.x and 3.x are different
- Known as proper language for machine learning & deep learning
- Well-known deep learning libraries such as tensorflow support python API



00. Python

Example program

```
import sys
import random
```

```
ans = True
```

define variable

```
while ans:
```

```
    question = raw_input("Insert a number: (press enter to quit) ")
```

```
    answers = random.randint(1,8)
```

```
    if question == "":
        sys.exit()
```

```
    elif answers == 1:
        print("You enter 1")
```

```
    elif answers == 2:
        print("You enter 2")
```

01. Arithmetic operations

- 1.1 Arithmetic operations의 이용



- Spyder 편집창에 산술 연산식 입력 후 <F9키>를 누르면 콘솔에 답이 출력됨
- Arithmetic operations은 다른 대부분의 프로그램 언어와 마찬가지로, +, -, *, /를 사용함.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Jul  4 18:47:50 2020
4
5  @author: DKU-MANGO
6  """
7
8
9  3+4
10 9+10
11
```

```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
```

```
IPython 7.12.0 -- An enhanced Interactive Python.
```

```
In [1]: 3+4
```

```
Out[1]: 7
```

```
In [2]: 9+10
```

```
Out[2]: 19
```

- 1.2 제곱

- 제곱은 **로 나타냄.

```
In 2 ** 8
```

```
Out 256
```

01. Arithmetic operations

Python Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then –

[[Show Example](#)]

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 30$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -10$
* Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/ Division	Divides left hand operand by right hand operand	$b / a = 2$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
** Exponent	Performs exponential (power) calculation on operators	$a ** b = 10 \text{ to the power } 20$

https://www.tutorialspoint.com/python/python_basic_operators.htm

02. Variable

- 2.1 Define variable

- 명시적 변수 선언이 없고 변수이름에 초기값을 저장하는 순간 변수 생성
- 저장하는 값의 자료형이 곧 변수의 자료형.

```
In      x = 1  
        y = 1 / 3  
        x + y
```

변수에 저장된 값을 확인하는
두가지 방법은?

```
Out     1.3333333333333333
```

- 2.2 Naming rule of variable

- Variable명은 Data_1, Data_2와 같이 여러 문자열로 나타낼 수 있음.
- Variable명에는 알파벳, 숫자, 밑줄(_)을 사용할 수 있음. 대문자와 소문자를 구별함.

```
In      Data_1 = 1 / 5  
        Data_2 = 3 / 5  
        Data_1 + Data_2
```

```
Out     0.8
```

03. Data Type

- 3.1 Data Type의 종류
 - Python에서 사용할 수 있는 데이터에는 정수와 실수, 문자열 등 다양한 종류가 있음.
 -
 - Python Data Type

자료형	사용 예	자료형의 의미
int 형	a=1	정수
float 형	a=1.5	실수
str 형	a='learning' 또는 b='abc'	문자열
bool 형	True 또는 False	참과 거짓
list 형	a=[1, 2, 3]	배열
tuple 형	a=(1, 2, 3) 또는 b=(2)	배열(수정 불가능)
ndarray 형	a=np.array([1, 2, 3])	행렬

03. Data Type

- 3.2 type으로 자료형 알아보기
 - Variable의 Data Type 확인 : type()

```
In type(100)

Out int

In type(100.1)

Out float
```

- 100은 int 형, 100.1은 float 형으로 취급하는 것을 알 수 있음.

```
In x = 100
   type(x)

Out int

In x = 100.1
   type(x)

Out float
```

- int 형 데이터를 Variable에 넣으면 그 Variable는 자동으로 int 형 Variable가 되고, float 형 데이터를 넣으면 float 형 Variable가 됨.

Python에서는 Variable의 Data Type을 명시적으로 선언하지 않음

03. Data Type

- 문자열
 - 문자열을 다루는 데에는 str 형이 사용됨.
 - 작은따옴표(') 또는 큰따옴표(")로 둘러싸면 문자열로 인식함.

```
In      x = 'learning'  
        type(x)
```

```
Out     str
```

04. print()

- print 의 이용
 - Variable명을 입력해 실행하는 것만으로 그 내용이 표시됨
 - Spyder 편집창에서 Variable명 선택 후 <F9> 클릭도 Variable의 내용 출력
 - Print() 를 이용한 출력.

```
In      x = 1 / 3
        print(x)
        y = 2 / 3
        print(y)
```

```
Out     0.3333333333333333
        0.6666666666666666
```

04. print()

- 4.2 문자열과 수치를 함께 표시 1
 - 문자열과 함께 수치를 표시하고 싶은 경우에는 다음처럼 표현함.

```
In      print('x= ' + str(x))           # str() : 숫자를 문자열로 변환
```

```
Out      x = 0.3333333333333333
```

- 문자열과 수치를 조합하는 다른 방법으로 format을 사용하면 편리함.

```
In      print('weight = {0} kg'.format(x))
```

```
Out      weight = 0.3333333333333333 kg
```

04. print()

- 4.3 문자열과 수치를 함께 표시 2
 - 여러 Variable를 표시할 경우에는 문자열 내에 {0}, {1}, {2}를 지정
 - {수치:.nf}를 입력하면 소수점 이하 n자리까지 표시함.

```
In      x = 1 / 3
        y = 1 / 7
        z = 1 / 4
        print ('weight: {0} kg {1} kg {2} kg'.format(x, y, z))
```

```
Out      weight: 0.3333333333333333 kg, 0.14285714285714285 kg 0.25 kg
```

```
In      print('weight: {0:.2f} kg, y={1:.2f} kg, z={2:.2f} kg'.format(x,y,z))
```

```
Out      weight: 0.33 kg, y=0.14 kg, z=0.25 kg
```

05. list

- 5.1 list의 이용

- 여러 데이터를 하나의 단위로 취급하고 싶은 경우, list (리스트)형을 사용함.
- list는 **리스트명[]**을 사용하여 나타냄.

```
In      x = [1, 1, 2, 3, 5] # list 정의
        print(x) # list 표시
```

주석(코멘트)

```
Out      [1, 1, 2, 3, 5]
```

- Python에서 리스트(배열)의 요소 번호(인덱스)는 0부터 시작함.

```
In      x[0]
```

```
Out      1
```

```
In      x[2]
```

```
Out      2
```

05. list

- x는 list 형, x[0]는 int 형인 것을 알 수 있음.
- x는 int 형으로 구성된 list 형이라고 이해할 수 있음. list 형은 str 형으로도 만들 수 있음.

In

```
print(type(x))  
print(type(x[0]))
```

Out

```
<class 'list'>  
<class 'int'>
```

05. list

- 5.2 2차원 배열

- list 형은 2차원 배열 형태로도 만들 수 있음.
- 3차원 배열, 4차원 배열도 중첩 깊이를 늘려 만들 수 있음.

```
In      a=[[1, 2, 3], [4, 5, 6]]  
        print(a)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
Out      [[1, 2, 3], [4, 5, 6]]
```

- list 요소를 수정하려면 **리스트명[요소 번호] = 값**로 할 수 있음.

```
In      x=[1, 1, 2, 3, 5]  
        x[3] = 100  
        print(x)
```

```
Out      [1, 1, 2, 100, 5]
```


05. list

- 5.3 list의 길이
 - list의 길이는 len()을 사용해 확인할 수 있음.

```
In      x=[1, 1, 2, 3, 5]
        len(x)
```

```
Out      5
```

- 5.4 연속된 정수로 이루어진 range
 - 5에서 9까지와 같이 연속된 정수 데이터를 만들려면 range(시작숫자, 끝숫자+1)를 사용함.

```
In      y = range(5, 10)
        print(y[0], y[1], y[2], y[3], y[4])
```

```
Out      5 6 7 8 9
```

Range 의 요소값은 변경 불가

05. list

- range 형은 list Function를 사용하여 요소를 수정할 수 있는 list 형으로 변환할 수 있음.

```
In      z=list(range(5, 10))  
        print(z)
```

```
Out     [5, 6, 7, 8, 9]
```

- 시작숫자를 생략하고 range(끝숫자+1)을 입력하면 0부터 시작되는 수열이 만들어짐.

```
In      list(range(10))
```

```
Out     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

06. tuple

- 6.1 tuple의 이용
 - 배열을 나타내는 형태로 list 형 이외에 tuple (튜플) 형이라는 배열 Variable 가 있음.
 - tuple은 list와는 달리 **요소를 수정할 수 없음.**
 - tuple 형은 (1, 2, 3)과 같이 괄호()를 사용하여 배열을 생성.

```
In      a=(1, 2, 3)
        print(a)
```

```
Out      (1, 2, 3)
```

06. tuple

- 6.2 요소의 참조
 - tuple 형에서 요소를 참조하려면 list 형과 같은 방식(대괄호[])을 사용함.
 -

```
In      a[1]
```

```
Out      2
```

- tuple 형도 type()으로 알 수 있음.

```
In      type(a)
```

```
Out      tuple
```

06. tuple

- 6.3 길이가 1인 tuple
 - 길이가 1인 tuple은 (1,)과 같이 쉼표(,)를 붙임.
 - (1, 2)는 tuple이지만, (1)은 tuple이 아님.

```
In      a = (1)
        type(a)
```

```
Out      int
```


```
In      a = (1,)
        type(a)
```

```
Out      tuple
```

07. if

- 7.1 if의 사용
 - 프로그램을 조건에 따라 나누어 실행시키려면 if을 사용함.
 - Python에서는 들여쓰기로 코드 블록을 나타냄

```
In      x = 11
        if x > 10:
            print('x is ') # ... (A1)
            print('      larger than 10.') # ... (A2)
        else:
            print('x is smaller than 11') # ... (B1)
```



```
if (x>10) {
    print(..)
    print(..)
} else {
    print(..)
}
```

```
Out      x is
          larger than 10.
```

- 첫 행에서 x 값으로 11을 대입하고 있으므로, if의 $x > 10$ 이라는 조건이 참(True).
4칸 오른쪽으로 들여쓰기 된 A1, A2행이 모두 실행.

07. if

- 7.2 비교 연산자

비교 연산자	내용
<code>a == b</code>	a와 b가 같다
<code>a > b</code>	a가 b보다 크다
<code>a >= b</code>	a가 b 이상이다
<code>a < b</code>	a가 b보다 작다
<code>a <= b</code>	a가 b 이하이다
<code>a != b</code>	a와 b는 다르다

- 여러 조건을 사용하려면, `and`(그리고)와 `or`(또는)를 사용함.

```
In      x = 15
        if 10 <= x and x <= 20:
            print('x is between 10 and 20.')
```

```
Out      x is between 10 and 20.
```

08. for

- 8.1 for의 이용

```
In      for i in [1, 2, 3]:  
        print(i)
```

```
Out      1  
        2  
        3
```

- 8.2 enumerate의 이용

```
In      num = [2, 4, 6, 8, 10]  
        for i, n in enumerate(num):  
            num[i] = n * 2  
        print(num)
```

```
Out      [4, 8, 12, 16, 20]
```

- 8.3 range()의 이용

```
for i in range(10):  
    print(i)
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```


09. Numpy vector (1차원배열)

- Numpy
 - NumPy is a python library used for working with arrays.
 - It also has functions for working in domain of linear algebra, fourier transform, and matrices.
 - NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
 - NumPy stands for Numerical Python.
- Why Use NumPy ?
 - In Python we have **lists** that serve the purpose of arrays, but they are slow to process.
 - NumPy aims to provide an array object that is up to **50x faster** than traditional Python lists.
 - The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with ndarray very easy.
 - Arrays are very frequently used in data science, where speed and resources are very important.

09. Numpy vector (1차원배열)

- 9.1 For using numpy ..
 - Python으로 vector나 Array을 나타내려면 넘파이(NumPy)라는 module (library)를 통해 기능을 확장.
 - import로 간단히 가져올 수 있음.

```
In import numpy as np
```

- 9.2 vector의 정의
 - vector(1차원 배열)은 np.array(list 형)으로 정의함.
 - type(x)를 입력하면 x가 numpy.ndarray 형인 것을 알 수 있음.

```
In x=np.array([1, 2, 3])  
x
```

```
Out array([1, 2, 3])
```

```
In type(x)
```

```
Out numpy.ndarray
```

09. vector (1차원배열)

- 9.3 요소의 참조
 - 하나의 요소를 참조하려면 list 형과 마찬가지로 대괄호 []를 사용함.

```
In      x[0]
```

```
Out      1
```

- 9.4 요소의 수정
 - 요소를 수정하려면 `x[수정할 요소 번호] = 수치`를 사용함.

```
In      x[0] = 100  
        print(x)
```

```
Out      [100 2 3]
```

09. vector (1차원배열)

- 9.5 연속된 정수 vector의 생성
 - np.arange(n)으로 요소의 값이 1씩 증가하는 vector 배열을 만들 수 있음.

```
In      print(np.arange(10))
```

```
Out      [0 1 2 3 4 5 6 7 8 9]
```

- 9.6 ndarray 형의 주의점
 - ndarray 형의 내용을 복사하려면 일반Variable처럼 $b = a$ 를 사용하는 것이 아니라 $b = a.copy()$ 를 사용해야 됨.

```
In      a = np.array([1, 1])
         b = a.copy()
         print('a =' + str(a))
         print('b =' + str(b))
         b[0] = 100
         print('b =' + str(b))
         print('a =' + str(a))
```

```
Out      a =[1 1]
         b =[1 1]
         b =[100 1]
         a =[1 1]
```

09. vector (1차원배열)

- 9.7 vector operation

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([5,6,7,8])

print(a+b)
print(2*a)
print(b-5)
print(6*b - 2*a)
```

```
In [32]: print(a+b)
[ 6  8 10 12]
```

```
In [33]: print(2*a)
[2 4 6 8]
```

```
In [34]: print(b-5)
[0 1 2 3]
```

```
In [35]: print(6*b - 2*a)
[28 32 36 40]
```

10. Array

- 10.1 Array의 정의

- ndarray의 2차원 배열로 다음과 같이 Array을 정의할 수 있음.

```
In x = np.array([[1, 2, 3], [4, 5, 6]])  
print(x)
```

```
Out [[1 2 3]  
     [4 5 6]]
```

- 10.2 Array의 크기

- Array(배열)의 크기는 ndarrayVariable명.shape 명령으로 알 수 있음.

```
In x = np.array([[1, 2, 3], [4, 5, 6]])  
x.shape
```

x.dtype # 원소의 자료형 확인

```
Out (2, 3)
```

```
In [45]: x.dtype  
Out[45]: dtype('int32')
```

10. Array

- 10.3 요소의 참조
 - 요소를 참조하려면 다음과 같이 차원마다 ','로 구분하여 나타냄.

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

```
Out 6
```

- 10.4 요소의 수정
 - 행과 열의 인덱스가 0에서 시작. 다음과 같이 기술하여 요소를 수정함.

```
In x = np.array([[1, 2, 3], [4, 5, 6]])  
x[1, 2] = 100  
print(x)
```

```
Out [[ 1  2  3]  
     [ 4  5 100]]
```

10. Array

- 10.5 요소가 0과 1인 ndarray 만들기
 - 모든 요소가 0인 ndarray는 np.zeros(size)로 만들 수 있음.

```
In print(np.zeros(10))
```

```
Out [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

- size=(2, 10)을 사용하면 모든 요소가 0인 2 × 10 크기의 Array이 생성됨.

```
In print(np.zeros((2, 10)))
```

```
Out [[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
     [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

- 요소를 0이 아니라 1로 하려면 np.ones(size)를 사용함.

```
In print(np.ones((2, 10)))
```

```
Out [[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]  
     [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]]
```


10. Array

- 10.6 요소가 랜덤인 Array 생성
 - 요소가 랜덤(임의)인 Array을 생성하는 경우에는 `np.random.rand(size)`를 사용
 - 각 요소는 0에서 1사이의 균일한 분포를 보이며 Array 형태로 생성함.
 - `np.random.randn(size)`를 사용하면 평균 0 분산 1의 가우스 분포로 난수를

```
In      np.random.rand(2, 3)
```

```
Out     array([[ 0.61172168,  0.20792486,  0.95905162],  
              [ 0.86475323,  0.18373685,  0.55318816]])
```

10. Array

- 10.7 Array의 크기 변경
 - Array의 크기를 변경하는 경우 Variable명.reshape(n, m)를 사용함.

```
x = np.array([[1,2,3],[4,5,6]])
print(x)
x.shape
x = x.reshape(3,2)
print(x)
x.shape
```

```
In [51]: x = np.array([[1,2,3],[4,5,6]])
```

```
In [52]: print(x)
[[1 2 3]
 [4 5 6]]
```

```
In [53]: x.shape
Out[53]: (2, 3)
```

```
In [54]: x = x.reshape(3,2)
```

```
In [55]: print(x)
[[1 2]
 [3 4]
 [5 6]]
```

```
In [56]: x.shape
Out[56]: (3, 2)
```

11. Array operation

- 11.1 Array의 Arithmetic operations

- Arithmetic operations $+$, $-$, $*$, $/$ 는 대응되는 요소들의 연산으로 실행됨.

```
In x = np.array([[4, 4, 4], [8, 8, 8]])  
y = np.array([[1, 1, 1], [2, 2, 2]])  
print(x + y)
```

$$\begin{pmatrix} 4 & 4 & 4 \\ 8 & 8 & 8 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 5 & 5 \\ 10 & 10 & 10 \end{pmatrix}$$

```
Out [[ 5  5  5]  
     [10 10 10]]
```

- 11.2 스칼라 \times Array

- 스칼라를 Array에 곱하면 다음과 같이 모든 요소에 적용됨.

```
In x = np.array([[4, 4, 4], [8, 8, 8]])  
print(10 * x)
```

```
Out [[40 40 40]  
     [80 80 80]]
```

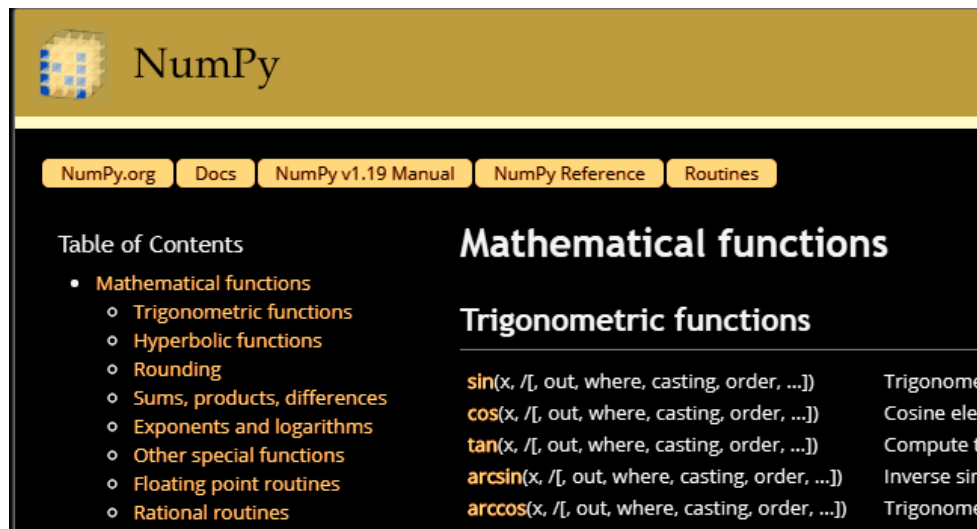
11. Array operation

- 11.3 산술 Function
 - 넘파이에는 다양한 수학 Function가 준비됨. $\log(x)$ Function는 다음과 같이 계산됨.

```
In x = np.array([[4,4,4],[8,8,8]])  
np.log(x)
```

```
Out array([[1.38629436, 1.38629436, 1.38629436],  
          [2.07944154, 2.07944154, 2.07944154]])
```

- Function list : <https://numpy.org/doc/stable/reference/routines.math.html>



The screenshot shows the NumPy website interface. At the top, there's a header with the NumPy logo and name. Below it, a navigation bar contains links: NumPy.org, Docs, NumPy v1.19 Manual, NumPy Reference, and Routines. The main content area is divided into two columns. The left column, titled 'Table of Contents', lists various function categories: Mathematical functions, Trigonometric functions, Hyperbolic functions, Rounding, Sums, products, differences, Exponents and logarithms, Other special functions, Floating point routines, and Rational routines. The right column, titled 'Mathematical functions', further categorizes them under 'Trigonometric functions', listing $\sin(x)$, $\cos(x)$, $\tan(x)$, $\arcsin(x)$, and $\arccos(x)$ with their respective descriptions.

11. Array operation

- 11.4 Array 곱의 계산 (수학적 의미의 행렬 곱셈)
 - 요소별로 계산하지 않는 Array 곱은 **Variable명1.dot(Variable명2)**로 계산할 수 있음.

```
In      v = np.array([[1, 2, 3], [4, 5, 6]])  
        w = np.array([[1, 1], [2, 2], [3, 3]])  
        print(v.dot(w))
```

```
Out      [[14 14]  
          [32 32]]
```

$$\begin{matrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & \times & \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} & = & \begin{bmatrix} 14 & 14 \\ 32 & 32 \end{bmatrix} \\ 2 \times 3 & & 3 \times 2 & & 2 \times 2 \end{matrix}$$

12. Slicing

- 12.1 Slicing의 이용
 - list와 ndarray에서 여러 요소를 한 번에 추출할 때 슬라이스(slice)라는 편리한 방법을 사용할 수 있음.
 - 슬라이스는 ':'을 사용하여 나타냄.

```
In      x = np.arange(10)
        print(x)
        print(x[:5])
```

```
Out      [0 1 2 3 4 5 6 7 8 9]
         [0 1 2 3 4]
```

```
In [76]: x[2:5]
Out[76]: array([2, 3, 4])
```

- Variable명[n:] 을 입력하면 n에서 마지막 요소까지 참조됨.

```
In      print(x[5:])
```

```
Out      [5 6 7 8 9]
```

12. Slicing

- Array slicing

```
c = np.array([[1,2,3,4],[5,6,7,8]])  
c  
c[:3,:2]
```

```
c[:,1]
```

```
c[0,:]
```

```
In [79]: c  
Out[79]:  
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
In [80]: c[:3,:2]  
Out[80]:  
array([[1, 2],  
       [5, 6]])
```

```
In [84]: c[:,1]  
Out[84]: array([2, 6])
```

```
In [85]: c[0,:]  
Out[85]: array([1, 2, 3, 4])
```

13. 조건을 만족하는 데이터의 수정

- 13.1 bool 배열 사용

- 넘파이를 통해 Array 데이터에서 특정 조건을 만족하는 것을 추출하여 쉽게 수정할 수 있음.

```
In      x = np.array([1, 1, 2, 3, 5, 8, 13])  
        x > 3
```

```
Out      array([False, False, False, False,  True,  True,  True])
```

[x를 정의하고 $x > 3$ 을 입력하면 결과가 True 또는 False로 bool 형식의 배열이 반환 됨]

```
In      x[x > 3]
```

```
Out      array([ 5, 8, 13])
```

[bool 배열의 요소를 참조하면, True 요소만 출력됨]

```
In      x[x > 3] = 999  
        print(x)
```

```
Out      [ 1 1 2 3 999 999 999]
```

[$x > 3$ 을 충족하는 요소만 999로 바꾸려면 위와 같이 입력함]

14. Help

- 14.1 Help 사용
 - Function의 설명을 확인하는 **help(Function명)**
 - Function의 다양한 사용법을 help 명령으로 확인할 수 있음.

In

```
help(np.random.randint)
```

Out

```
Help on built-in function randint:  
randint(...) method of mtrand.RandomState instance  
    randint(low, high=None, size=None, dtype='l')  
        Return random integers from 'low' (inclusive) to 'high' (exclusive).  
        ( ...중략... )
```

15. Function

- 15.1 Function의 사용
 - 프로그램의 일부를 Function로 정리할 수 있음.
 - 자주 사용하는 반복된 코드는 Function로 만드는 것이 좋음.
 - Function는 **def Function명():** 으로 시작하여, Function의 내용은 들여쓰기로 정의 함.

```
In      def my_func1():  
        print('Hi!')  
        #함수 my_func1()의 정의는 여기까지  
        my_func1() # 함수를 실행
```

```
Hi!
```

```
In      def my_func2(a, b):  
        c = a + b  
        return c
```

```
my_func2(1, 2)
```

```
Out     3
```

15. Function

- 15.2 인수와 반환값
 - Function에 전달할 Variable를 인수(parameter)라고 함.
 - Function의 출력은 반환값(return value)이라고 함.

```
In def my_func3(D):  
    m = np.mean(D)  
    s = np.std(D)  
    return m, s
```

- 여러 반환값을 받는 방법은 `data_mean, data_std = my_func3(data)`와 같이 ','로 구분 기술함.

```
In data=np.random.randn(100)  
data_mean, data_std = my_func3(data)  
print('mean:{0:3.2f}, std:{1:3.2f}'.format(data_mean, data_std))
```

```
Out mean:0.10, std:1.04
```

16. Store ndarray Object

- 16.1 ndarray 형을 저장
 - 하나의 ndarray 형을 파일에 저장하려면 np.save ('파일명.npy', Variable명)을 사용함.
 - 파일의 확장자는 .npy 임.(난수이므로 실행할 때마다 다른 결과가 저장됨).
 - 데이터를 로드하려면(읽으려면) np.load('파일명.npy')를 사용함.

```
In data = np.random.randn(5)
    print(data)
    np.save('datafile.npy', data) # 세이브
    data = []                     # 데이터 삭제
    print(data)
    data = np.load('datafile.npy') # 로드
    print(data)
```

```
Out [ 0.04283863 0.11556549 -0.12882679 1.03572699 1.2465202 ]
     []
     [ 0.04283863 0.11556549 -0.12882679 1.03572699 1.2465202 ]
```

16. Store ndarray Object

- 16.2 여러 ndarray 형을 저장
 - 여러 ndarray 형을 저장하려면 `np.savez('파일명.npz', Variable명1 = Variable명1, Variable명2 =Variable명2, ...)`을 사용함.

```
In data1 = np.array([1, 2, 3])
data2 = np.array([10, 20, 30])
np.savez('datafile2.npz', data1=data1, data2=data2) # 세이브
data1 = [] # 데이터 삭제
data2 = []
outfile = np.load('datafile2.npz') # 로드
print(outfile.files) # 저장된 데이터 표시
data1 = outfile['data1'] # data1 꺼내기
data2 = outfile['data2'] # data2 꺼내기
print(data1)
print(data2)
```

```
Out ['data1', 'data2']
[1 2 3]
[10 20 30]
```

17. Graph

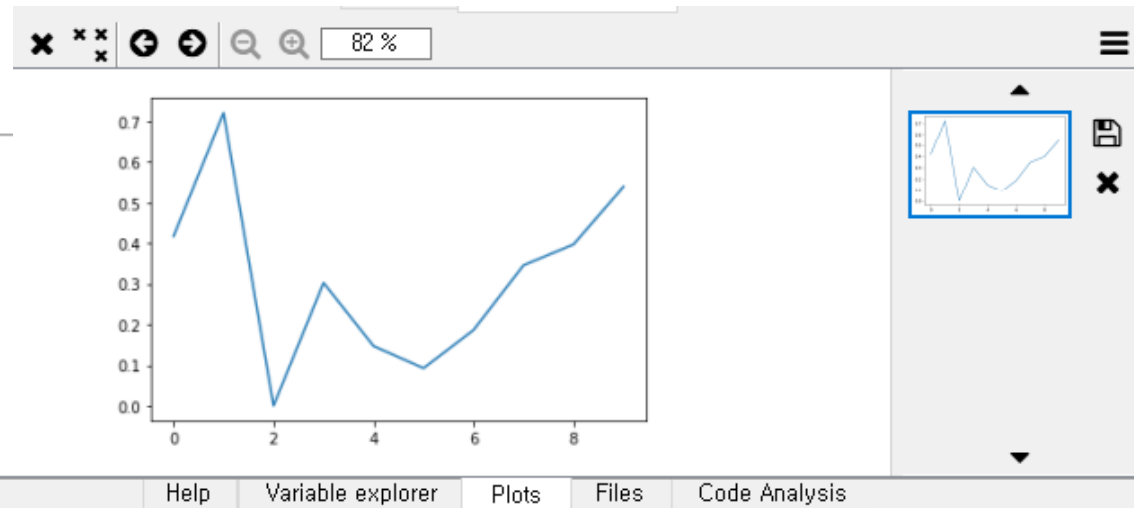
- 1.1 임의의 그래프 그리기
 - matplotlib의 pyplot 라이브러리를 import하고, 이를 plt라는 별칭을 만들어 사용함.
 - ipython console 에 그래프 표시 : %matplotlib inline 명령을 추가
 - 별도의 윈도우에 그래프 표시 : % matplotlib qt 명령을 추가 (default)

```
In      # 리스트 1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# data 작성
np.random.seed(1) # 난수를 고정
x = np.arange(10)
y = np.random.rand(10)

# 그래프 표시
plt.plot(x, y) # 짙은선 그래프를 등록
plt.show() # 그래프 그리기
```

Spyder [plots]에 표시됨



17. Graph

- 1.2 그래프 장식하기

In

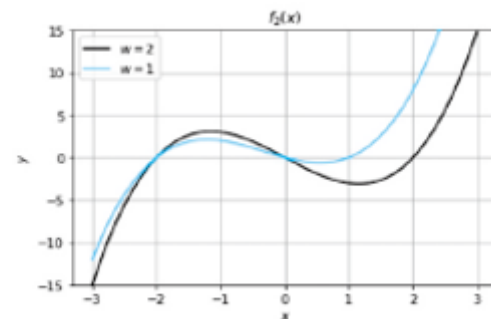
```
# 리스트 2-(7)
# 함수를 정의

def f2(x, w):
    return (x - w) * x * (x + 2) # (A) 함수 정의

# x를 정의
x = np.linspace(-3, 3, 100) # (B) x를 100 분할하기

# 차트 묘사
plt.plot(x, f2(x, 2), color='black', label='$w=2$') # (C)
plt.plot(x, f2(x, 1), color='cornflowerblue',
         label='$w=1$') # (D)
plt.legend(loc="upper left") # (E) 범례 표시
plt.ylim(-15, 15) # (F) y 축의 범위
plt.title('$f_2(x)$') # (G) 제목
plt.xlabel('$x$') # (H) x 라벨
plt.ylabel('$y$') # (I) y 라벨
plt.grid(True) # (J) 그리드
plt.show()
```

Out



17. Graph

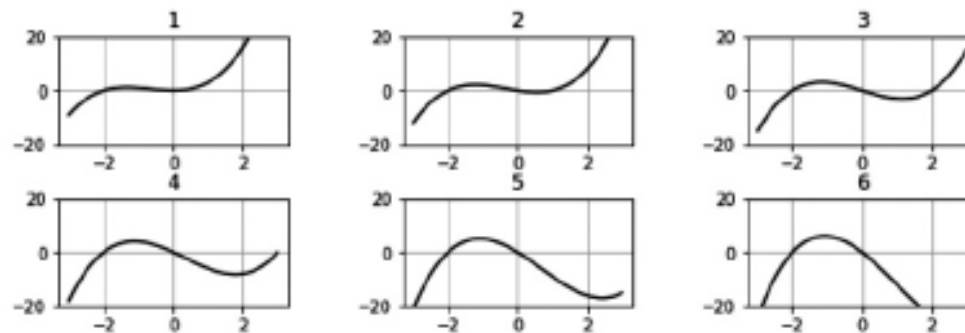
- 1.3 그래프를 여러 개 보여주기

- 여러 그래프를 나란히 표시하려면 `plt.subplot(n1, n2, n)`를 사용하여 전체를 세로 `n1`, 가로 `n2`로 나눈 `n`번째에 그래프가 그려짐.

In

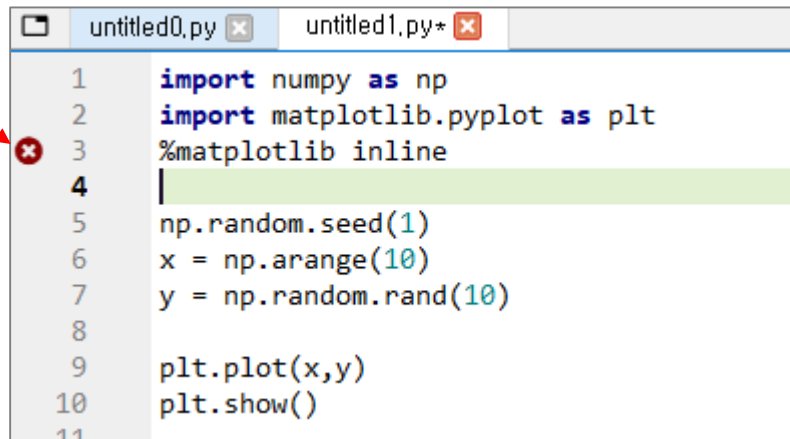
```
# 리스트 2-(9)
plt.figure(figsize=(10, 3)) # (A) figure 지정
plt.subplots_adjust(wspace=0.5, hspace=0.5) # (B) 그래프의 간격을 지정
for i in range(6):
    plt.subplot(2, 3, i + 1) # (C) 그래프 묘사의 위치를 지정
    plt.title(i + 1)
    plt.plot(x, f2(x, i), 'k')
    plt.ylim(-20, 20)
    plt.grid(True)
plt.show()
```

Out



Note. %matplotlib inline 에러

이렇게 syntax error 가 발생
하는 것 처럼 보이는데 무시함



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4 |
5 np.random.seed(1)
6 x = np.arange(10)
7 y = np.random.rand(10)
8
9 plt.plot(x,y)
10 plt.show()
11
```

- %matplotlib inline 를 실행해도 그래프가 console 에 표시되지 않고 plot 창에 표시되는 이유

