신영

1.1 파이썬이란?

- **오픈 소스**라 무료로 자유롭게 이용 가능
- 자체의 뛰어난 성능에 넘파이(numpy)와 사이파이(SciPy)같은 수치 계산과 통계 처리를 다루는 탁월한 라이브러리가 더해져 **데이터 과학 분야**에서 널리 쓰임

1.2 파이썬의 기본 문법

1. 산술 연산

```
a = 3
b = 4
print(a + b)
```

```
print(a - b)
print(a * b)
print(a / b)
>>> 7
>>> -1
>>> 12
>>> 0.75

print(a**b) # 제곱근
>>> 81

print(7%3) # 모듈러(나머지) 연산
print(3%7)
>>> 1
>>> 3
```

2. 자료형

자료형 : 데이터의 성질을 나타내는 것

> 파이썬의 자료형 - 동적타입이므로 자료형에 따라서 변수가 달라짐
>
> - 숫자형
> - 정수형
> - 실수형
> - 지수형
> - 복소수
> - 8진수
> - 16진수
> - 문자열형

> - 리스트
> - 튜플
> - 딕션너리
> - bool

> 확장형 자료형
>
> - 집합
> - 넘파이

파이썬에서는 type() 함수로 특정 데이터의 자료형을 알아볼 수 있음.

```
# int type
a = 5
print(type(a))
>>> <class 'int'>

# string type
b = str(a)    # '5'
print(type(b))
>>> <class 'str'>

# string > ASCII
print(ord('5'))   #
>>> 53


# int > 16진수
print(hex(53))
>>> 0x35

# string > int
print(type('5'))
print(int('5'))
print(type(int('5')))
```

```
>>> <class 'str'>
>>> 5
>>> <class 'int'>

# int > string
print(type(53))
print(chr(53))
print(type(chr(53)))
>>> <class 'int'>
>>> 5
>>> <class 'str'>
```

문자열(str) 자료형

```
print("%10s" % "hi")
print('---------------')
print("%-10sjane." % 'hi')
>>>         hi
>>> ---------------
>>> hi        jane.
```

```
## lower
a = "HI"
print(a.lower())
>>> hi

## strip
a = "  hi  "
a.strip()
>>> 'hi'

## replace
a = "Life is too short"
print(a.replace("Life","Your leg"))
>>> Your leg is too short

## split
a = 'Life is too short'
print(a.split())
>>> ['Life', 'is', 'too', 'short']
```

```
b = "a:b:c:d"
print(b.split(':'))
>>> ['a', 'b', 'c', 'd']
```

리스트(list) 자료형

인덱스와 슬라이싱(slicing) 기법을 통해 원하는 원소에 접근할 수 있다.

```
## Slicing
a = "Life is too short, You need python"
print(a[0:4])

a = "20010331Rainy"
date = a[:8]
weather = a[8:]

print(date)
print(weather)

>>> Life
>>> 20010331
>>> Rainy
```

```
## append
a = [1,2,3]
print(a)
a.append(4)
print(a)

>>> [1, 2, 3]
>>> [1, 2, 3, 4]
```

```
## sort
a = [1,4,3,2,]
```

```
a.sort()
print(a)

>>> [1, 2, 3, 4]
```

```
##reverse
a = ['a','c','b']
a.reverse()
print(a)

>>> ['b', 'c', 'a']
```

```
## index
a = [1,2,3]
print(a)
print(a.index(3))
print(a.index(1))
>>> [1, 2, 3]
>>> 2
>>> 0
```

```
## insert
a = [1,2,3]
a.insert(0,4)
print(a) # [4, 1, 2, 3]


## remove
a = [1,2,3,1,2,3]
a.remove(3)
print(a)  # [1, 2, 1, 2, 3]


## pop
a = [1,2,3]
```

```
print(a.pop()) # 3
print(a) # [1, 2]




## count
a = [1,2,3,1]
print(a.count(1)) # 2
```

```
## extend
a = [1,2,3]
a.extend([4,5])
print(a)  # [1, 2, 3, 4, 5]


b = [6,7]
a.extend(b)
print(a) # [1, 2, 3, 4, 5, 6, 7]
```

튜플(tuple) 자료형

```
## Tuple
t = (1,2,'a','b')
print(t) # (1, 2, 'a', 'b')
print(type(t)) # <class 'tuple'>


t1 = (1,2,'a','b')
# del t1[0] ## Type Error : Tuple object doesn't support item


t1 = (1,2,'a','b')
```

```
t1[0] = 'c'
## Type Error
```

```
## indexing
t1 = (1,2,'a','b')
print(t1[0]) # 1
print(t1[3]) # b

## slicing
t1 = (1,2,'a','b')
print(t1[1:]) # (2, 'a', 'b')

## add
t2 = (3,4)
print(t1 + t2) # (1, 2, 'a', 'b', 3, 4)

## mul
print(t2*3) # (3, 4, 3, 4, 3, 4)
```

딕셔너리(dict) 자료형

딕셔너리는 키(key)와 값(value)를 한쌍으로 저장한다.

```
me = ['height':180] # 딕셔너리 생성
me['height']    # 원소 접근 , 180

me['weight'] = 70   # 새 원소 추가
print(me) # {'weight': 70, 'height': 180}


a = { 'a': [1,2,3]}  # Value에 리스트도 넣을 수 있다.
```

```
# 딕셔너리 쌍 추가하기
a = {1:'a'}
print(a) # {1: 'a'}
```

```
a[2] = 'b'
print(a) # {1: 'a', 2: 'b'}

# 딕셔너리 원소 삭제
del a[1]
print(a) # {2: 'b'}



# 원소 출력
grade = {'pey':10,'julliet':99}
print(grade) # {'pey': 10, 'julliet': 99}


print(grade['pey']) # 10

print(grade['julliet']) # 99


a = {1:'a',1:'b'}
print(a) # {1: 'b'}
```

```
## keys and values
a = {'name':'pey','phone':'010999323','birth':'1118'}
print(a)
# {'name': 'pey', 'phone': '010999323', 'birth': '1118'}

print(a.keys())
# dict_keys(['name', 'phone', 'birth'])

print(a.values())
# dict_values(['pey', '010999323', '1118'])

print(a.items())
```

```
# dict_items([('name', 'pey'), ('phone', '010999323'), ('birt

a.clear()
print(a)
# {}
```

```
a = {'name':'pey','phone':'0102383924','birth':'0912'}
print(a) # {'name': 'pey', 'phone': '0102383924', 'birth': '0

print(a.get('name')) # pey

print(a.get('phone')) # 0102383924


print('name' in a) # True

print('email' in a) # False
```

집합(set) 자료형

```
# 집합 자료형은 어떻게 만들까?
#    집합(set)은 파이썬 2.3부터 지원
#    집합에 관련된 것들을 쉽게 처리하기 위해 만들어진 자료형
#    집합 자료형은 다음과 같이 set 키워드를 이용해 만듬.

s1 = set([1,2,3])
print(s1) # {1, 2, 3}
print(type(s1)) # # <class 'set'>

s2 = {1,2,3}
```

```
print(s2) # {1, 2, 3}
print(type(s2)) # <class 'set'>



# 자료형의 특징
#    중복을 허용하지 않는다.
#    순서가 없다(Unordered)
s2 = set("Hello")
print(s2) # {'o', 'H', 'l', 'e'}



## Union
s1 = set([1,2,3,4,5,6])
s2 = set([4,5,6,7,8,9])
print(s1 & s2) # {4, 5, 6}

print( s1.intersection(s2))# {4, 5, 6}
```

```
## Union
s1 = set([1,2,3,4,5,6])
s2 = set([4,5,6,7,8,9])
print(s1 | s2)

print( s1.union(s2))

# {1, 2, 3, 4, 5, 6, 7, 8, 9}
# {1, 2, 3, 4, 5, 6, 7, 8, 9}

## Difference
s1 = set([1,2,3,4,5,6])
s2 = set([4,5,6,7,8,9])

print(s1 - s2)
print(s2 - s1)
```

```
print(s1.difference(s2))
print(s2.difference(s1))
# {1, 2, 3}
# {8, 9, 7}
# {1, 2, 3}
# {8, 9, 7}
```

```
# 값 1개 추가하기(add)
s1 = set([1, 2, 3])
s1.add(4)
s1
# {1, 2, 3, 4}


# 값 여러 개 추가하기(update)
s1 = set([1, 2, 3])
s1.update([4, 5, 6])
s1
# {1, 2, 3, 4, 5, 6}


# 특정 값 제거하기(remove)
s1 = set([1, 2, 3])
s1.remove(2)
s1
# {1, 3}
```

불(bool) 자료형

bool이라는 자료형은 True(참)와 False(거짓)이라는 두 값 중 하나를 취한다.

and, not, or 연산자와 같이 사용 가능

```
hungry = True
sleepy = False
print(not hungry)  # False
print(hungry and sleepy)  # False
print(hungry or sleepy)  # True
```

```
# 값                참 or 거짓
# "python"      참
# ""            거짓
# [1, 2, 3]     참
# []            거짓
# ()            거짓
# {}            거짓
# 1             참
# 0             거짓
# None          거짓

# 비어 있으면       거짓
# 비어 있지않으면   참


# while 조건문:
#     수행할 문장

a = [1, 2, 3, 4]
while a:
    print(a.pop())
```

3. 변수

파이썬은 **동적 언어**

즉, 변수의 자료형을 상황에 맞게 자동으로 결정한다.

```
>>> x = 10      # 초기화
>>> print(x)    # x값 출력
10
>>> x = 100     # 변수에 값 대입
>>> print(X)
100
>>> y = 3.14
>>> x * y
314.0
>>> type(x * y)
<class 'float'>
```

4. if 문

조건에 따른 처리는 if/else문 사용

```
hungry = True
if hungry:
    print("I'm hungry")  # I'm hungry
else:
    print("I'm not hungry")
```

5. for 문

반복문

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

for i in range(0, 5):
    print(i)  # 0, 1, 2, 3, 4
```

```
# 리스트 내포(list comprehension 1)
a = [1, 2, 3, 4]
result = []
for num in a:
    result.append(num * 3)
print(result) # [3, 6, 9, 12]

# 리스트 내포(list comprehension 2)
result = [num * 3 for num in a if num % 2 == 0]
print(result) # [6, 12]
```

6. 함수

특정 기능을 수행하는 일련의 명령들을 묶어 하나의 함수로 정의

```
def hello():
    print("Hello World!")

hello()  # Hello World!
```

함수는 인자를 취할 수도 있다.

```
def hello(object):
    print("Hello " + object + "!")

hello("cat") # Hello cat!
```

```
# 입력값이 몇 개가 될지 모르는 함수
# 매개변수로 *args를 사용한다.

def sum_many(*args):
    print(args)

result = sum_many(1,2,3,4,5,6,7,8,9,10)
# (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
######################3

def sum_many(*args):
    sum = 0
    for i in args:
        sum = sum + i
    return sum

result = sum_many(1,2,3)
print(result)  #6
#input('press any key...')

result = sum_many(1,2,3,4,5,6,7,8,9,10)
print(result) # 55
```

```
# 인자가 여러 개인 경우

def func(**kwargs):
    print(kwargs)

func(a=1)
func(num = 'foo', age=3)

# {'a': 1}
# {'num': 'foo', 'age': 3}
```

```
# 함수 결과 값이 두개 이상인 경우

def sum_and_mul(a,b):
    return a+b, a*b

a = sum_and_mul(3,4)
print(a) # (7, 12)

#input('press any key...')
```

```
sum, mul = sum_and_mul(3, 4)
print(sum, mul) # 7 12
```

```
# return의 다른 쓰임새 - 특별한 상황에서 빠져 나가고 싶을 때

def say_nick(nick):
    if nick == "바보":
        return
    print("나의 별명은 %s 입니다." % nick)

say_nick('야호')

#input('press any key...')

say_nick('바보')

# 나의 별명은 야호 입니다.
```

```
# 입력인수에 초기 값을 미래 정해 둔 경우
# 초기값이 정해진 함수는 마지막에 값을 둔다.


# def say_myself(name, sex=1, old):
def say_myself(name, old, sex=1):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if sex:
        print("남자입니다.")
    else:
        print("여자입니다.")

say_myself("박응용", 27)
```

```
#input('press any key...')

say_myself("박응용", 27, 1)

#input('press any key...')

say_myself("박응선", 27, 0)


# 나의 이름은 박응용 입니다.
# 나이는 27살입니다.
# 남자입니다.
# 나의 이름은 박응용 입니다.
# 나이는 27살입니다.
# 남자입니다.
# 나의 이름은 박응선 입니다.
# 나이는 27살입니다.
# 여자입니다.
```

```
# 함수 안에서 선언된 변수의 효력 범위
aa = 1
def vartest(aa):
    aa = aa + 1

vartest(aa)
print(aa) # 1
```

```
# 전역 변수 aa와 함수내의 aa는 이름은 같지만 다른 변수임
# 전역변수 aa 인자로 전달이 전달이 됨 업데이트 되지않음

def vartest01(b):
    b = b + 1

vartest01(aa)
print(aa) # 1
```

```
# 에러 발생하는 경우
def vartest(a):
    a = a + 1
    print(a)

vartest(3)

print(a)   # a라는 변수가 없기 때문에 에러가 발생
           # a라는 변수는 함수안에서 사용하는 지역 변수 이다.
```

```
# 함수 안에서 함수 밖의 변수를 변경하는 방법
aa = 1                    # return 이용하기
def vartest02(aa):
    aa = aa +1
    return aa


aa = vartest02(aa)
print(aa) # 2

# global 명령을 이용하기
def vartest03():
    global aa
    aa = aa+1

vartest03()
print(aa)
```

```
# 함수를 딱 한 줄만으로 만듬
# Lisp 언어에서 물려받음


```

```
lambda 매개변수 : 표현식
```

```
#def add(a, b):
#    return a+b
add = lambda a, b: a+b          # lambda를 이용한 경우
result = add(3, 4)
print(result) # 7


(lambda a, b: a+b)(3, 4) # 7


def add(a, b):                  # 함수를 이용한 경우
    return a+b

result = add(3, 4)
print(result) # 7
```

7. 클래스

개발자가 직접 클래스를 정의하면 독자적인 자료형을 만들 수 있다.

클래스에는 그 클래스만의 전용 함수(메서드)와 속성을 정의할 수 있음.

```
class 클래스 명:
    def __init__(self, 인수, ...):     # 생성자
        ...
    def 메서드 명 1(self, 인수, ...):      # 메서드 1
        ...
    def 메서드 명 2(self, 인수, ...):      # 메서드 2
        ...
```

_ _init_ _이라는 메서드는 **클래스를 초기화**하는 방법을 정의한다.

이 메서드를 생성자(constructor)라고 하며, 클래스의 인스턴스가 만들어질 때 한 번만 불림.

파이썬에서는 메서드의 첫 번째 인수로 자신(자신의 인스턴스)를 나타내는 **self를 명시적으로 쓰는 것**이 특징이다.

예시)

```
class Man:
    def __init__(self, name):
        self.name = name
        print("Initialized!")

    def hello(self):
        print("Hello " + self.name + "!")

    def goodbye(self):
        print("Good-bye " + self.name + "!")

m = Man("David") # Initialized!
m.hello()   # Hello David!
m.goodbye() # Good-bye David!
```

8. Numpy

Numpy는 고성능의 수치 계산을 위한 파이썬 라이브러리로, 다차원 배열 객체와 다양한 수학 함수 제공.

위와 같이 import 문을 통해 라이브러리를 가져와야 함.

```
import numpy as np
```

이 때, np라는 이름으로 가져옴으로써 앞으로 넘파이 메소드들을 np를 통해 참조할 수 있도록 함.

- 넘파이 배열 생성

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> print(X)
```

```
[1. 2. 3.]
>>> type(X)
<class 'numpy.ndarray'>
```

- 넘파이의 산술 연산

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> y = np.array([2.0, 4.0, 6.0])
>>> x + y
array([ 3., 6., 9.])
>>> x - y
array([ -1., -2., -3.])
>>> x * y
array([2., 8., 18.])
>>> x / y
array([ 0.5, 0.5, 0.5])
```

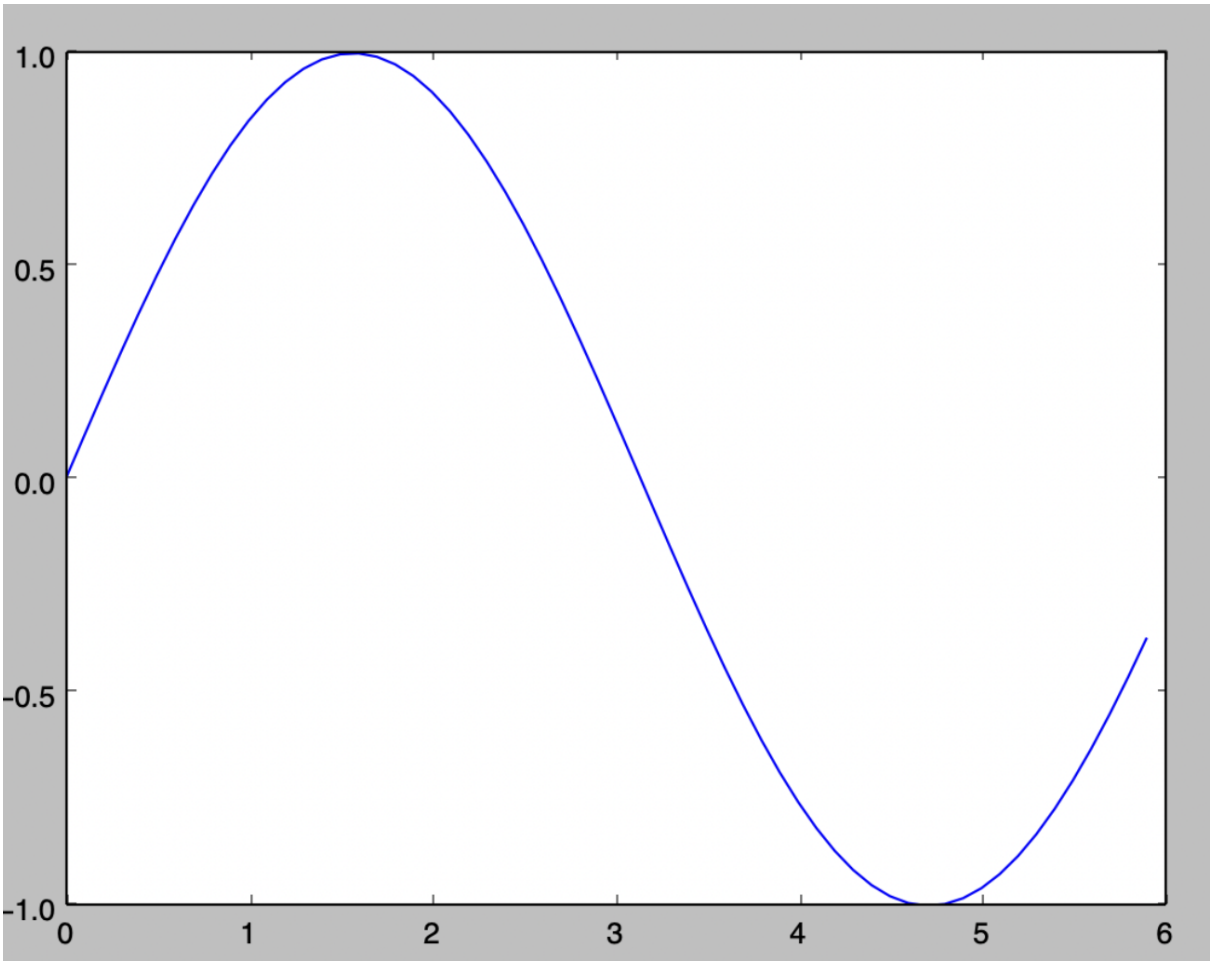
9. Matplotlib

Matplotlib은 데이터 시각화를 위한 파이썬 라이브러리로, 다양한 그래프와 차트를 그릴 수 있는 기능 제공.

```
import numpy as np
import matplotlib.pyplot as plt

# 데이터 준비
x = np.arange(0, 6, 0.1)    # 0에서 6까지 0.1 간격으로 생성
y = np.sin(x)

# 그래프 그리기
plt.plot(x, y)
plt.show()
```



- 추가적인 pyplot의 기능

```
import numpy as np
import matplotlib.pyplot as plt

# 데이터 준비
x = np.arange(0, 6, 0.1)
y1 = np.sin(x)
y2 = np.cos(x)

# 그래프 그리기
plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle="--", label="cos") # cos 함수는 점선으
plt.xlabel("x") # x축 이름
plt.ylabel("y") # y축 이름
plt.title('sin & cos')  # 제목
```

```
plt.legend()    # 범례
plt.show()
```