대학 수학 학습을 위한 파이썬 부트캠프

II: Python의 문법과 자료형

구성

Python 프로그래밍의 이해 (변수와 연산)		
9/25(토) 13:00 ~ 15:00	주제	· Python 소개 · 변수 및 기본 연산자
	실습 예제	· Python 개발 환경 세팅(Anaconda 설치, Jupyter notebook 활용) · Python 코딩 기초 실습
Python의 문법과 자료형 (벡터와 행렬 및 수치 미분)		
10/2(토) 13:00 ~ 15:00	주제	· 기초 문법 (Syntax, comment, indentation) · 기본자료형 · 복합자료형 (list, vector, matrix) · 함수의 정의
	실습 예제	· Python 자료형 활용 예제 – 미적분학 1: 행렬의 연산 · 함수 활용 예제 – 수치 미분
Python 제어문 활용 (테일러 급수와 근 찾기 알고리즘)		
10/9(토) 13:00 ~ 15:00	주제	· 조건문, 반복문 · 재귀함수
	실습 예제	· 반복문 실습 예제 – 미적분학 1: 테일러 급수 연산 · 재귀함수 실습 예제 – 미적분학 1: Newton's method
Python 라이브러리 활용 (다변수함수 시각화)		
10/16(토) 13:00 ~ 15:00	주제	· 라이브러리 활용(Matplotlib, numpy, math, sympy 등) · Python 응용
	실습 예제	· 미적분학 2: 다변수 함수 3D 시각화 (그래프, 등위면, 극점, 안장점 등)

※ Zoom 온라인 미팅으로 실시간 진행됩니다.

※ 주차별 강의는 Co-티칭으로 진행되며, 불참시 녹화 영상이 제공되지 않습니다.

변수

- 변수 (Variable)
  - 하나의 값을 저장할 수 있는 저장 공간
  - 변수는 *식별자*(Identifier)를 통해 *선언*(Declaration)을 해야 사용할 수 있다
  - 한 변수는 한가지 *자료형*(Type)을 가지며 저장된 값(Data)는 변경될 수 있다

```
# 식별자가 'x'인 변수를 선언(declaration)하고,
# 정수 자료형 값 3을 할당한다
x = 3

language = 'python'
version = 3.7
```

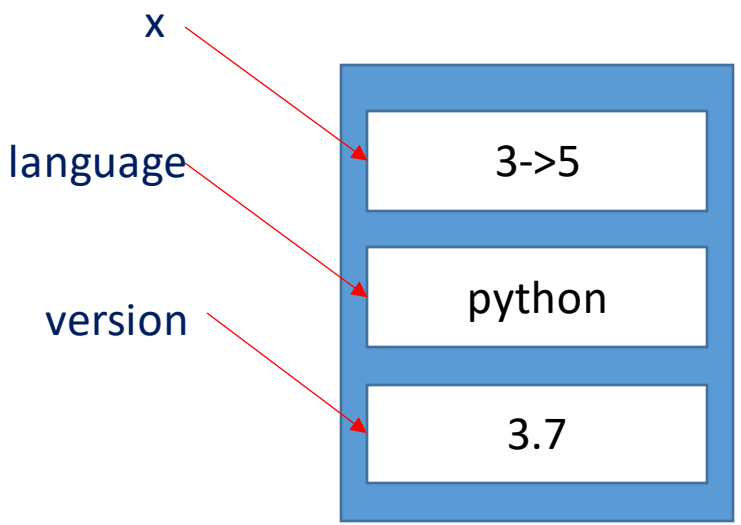
선언과 식별자

- 선언
  - 프로그램에서 메모리에 공간을 생성하고, 식별자를 지정해 공간에 접근가능하도록 하는 작업
  - 파이썬에서는 변수를 선언할 때 자료형과 값을 지정해야한다

```
# 식별자가 'x'인 변수를 선언(declaration)하고,
# 정수 자료형 값 3을 할당한다
x = 3
language = 'python'
version = 3.7

x = 5
```

- 식별자(변수명)
  - 객체를 식별하기위해 프로그래머가 붙이는 이름

- 식별자 권장사항 (가독성을 위해)
  - 의미 있는 단어 사용
  - 두 단어 이상의 조합은 camelCase 표기법 사용
    (변수 첫글자는 소문자 사용)
  - 단어와 단어 사이를 _로 구분



DRAM 메모리

자료형

- 파이썬의 모든 값은 데이터의 유형을 가지며 이를 자료형(Type)이라고 함
  - 자료형에 따라 데이터가 컴퓨터 메모리에 저장되는 방식이 결정됨
  - 자료형에 따라 수행 가능한 있는 연산의 종류가 다름
  - 파이썬은 변수에 값을 할당할 때 자료형을 결정한다

- 파이썬의 자료형은 기본자료형과 복합자료형으로 구분된다
  - 기본자료형: Intger, Float, Boolean, String
  - 복합자료형: List, Tuple, Dictionary, Set

기본 자료형 – 숫자(Numeric Data Type)

- 정수 (Integer Type)
  - 소수점이 없는 숫자의 조합으로 이루어지며 음수도 포함한다
  - 기본적으로 10진수를 사용하지만 2진수, 8진수, 16진수로도 표현할 수 있다
  - 파이썬에서는 int로 표현된다

- 실수 (Floating point Type)
  - 소수점이 있는 숫자의 조합으로 이루어지며 음수도 포함
  - 파이썬에서는 float으로 표현

기본 자료형 - Boolean

- 불리언 (Boolean Type)
  - 참(True)과 거짓(False) 두 가지 값 중 한 개만을 취할 수 있는 논리 자료형
  - 대소문자 구분: (true, false, TRUE, FALSE)는 올바른 값으로 인정하지 않는다
  - 파이썬에서는 bool으로 표현된다

```
x = True
print(x)

y = False
print(y)
```

```
True
False
```

기본 자료형 - String

- 문자열 (String Type)
  - 시작과 끝을 따옴표로 묶어서 만든 순서가 있는 글자의 배열이다
  - 작은 따옴표('...') 혹은 큰 따옴표 ("...") 사용가능

```
small_string = '작은 따옴표 문자열'

large_string = "큰 따옴표 문자열"

print(large_string)

print("큰 따옴표 문자열")
```

```
큰 따옴표 문자열
큰 따옴표 문자열
```

기초 문법 – 할당(assign)

- 변수 할당 (=)
  - 값을 변수의 메모리 영역에 저장하는 작업
  - '=' 연산자의 *오른쪽에 있는 값*을 *왼쪽의 변수*에 대입한다
  - 프로그래밍 언어의 할당(=) 기호는 수학적의 기호와 다른 의미를 가짐

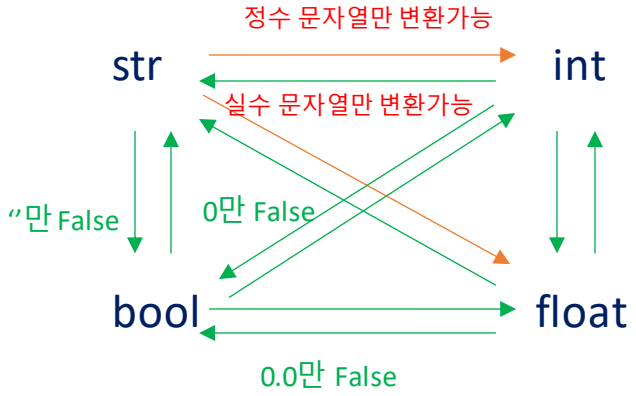
```
#변수 선언
# x는 3이다
x = 3

# x는 x+5다
x = x + 5
```

x는 x+5(와 같)다 (X)
x는 x+5(의 값을 가진)다 (O)

기초 문법 – 형변환

- 형변환 (Type Casting)
  - 어떤 자료형을 다른 자료형으로 변환시키는 과정
  - 서로 다른 자료형 변수를 연산하기 위해서 형변환이 필요한 경우가 있다
  - 형변환 방법: type(객체값)
  - 변환타입: str, int, float, bool

```
a = 3.14
b = '5'

c = a + b
d = a + int(b)
e = a + float(b)
```



복합 자료형

- List: 순서를 가지는 객체의 연속된 배열
- Tuple: 리스트와 유사한 구조이나 한번 값을 지정하면 값의 변경이 불가능
- Dictionary: 키(key)와 값(value)로 구성된 자료형
- Set: 집합에 관련된 작업을 쉽게 처리하기 위해 만들어졌으며, 중복 값이 없고, 순서가 없는 자료형

```
myList = [1,2,3,4,5,6,7,8,9,10]

myTuple = (1,2,3,4,5,6,7,8,9,10)

myDictionary = {"first":1, "second":2, "third": 3}

mySet1 = set([1,2,3,4,5,6,7,8,9,10])
mySet2 = set([10,9,8,7,6,5,4,3,2,1])
```

List

- 순서를 객체의 연속된 배열
  - matrix, vector, 배열 등 복수의 값을 표현하는데 특화된 자료형
  - 유연성이 높으며 가장 보편적으로 사용
- 대괄호[]으로 데이터를 감싸 표현하며, 감싸인 데이터를 요소(element)라고 한다
- 리스트는 어떠한 자료형도 담을 수 있다
- 생성 후 내용의 변경이 다양하게 가능하다
- List 변수선언
  - 대괄호 []를 사용한 리스트 생성
  - List()함수를 사용한 리스트 생성

```
myList1 = []
myList2 = list()

myList3 = [1,2,3,4,5]
```

```
# Complex List 구조
# 첫번째 element: 0                (int)
# 두번째 element: 3.14             (float)
# 세번째 element: 'example_string'  (str)
# 네번째 element: True             (bool)
# 다섯번째 element: [1,2,3,4,5]    (list)
complex_list = [0, 3.14, 'example_string', True, [1,2,3,4,5]]
```

리스트 인덱스

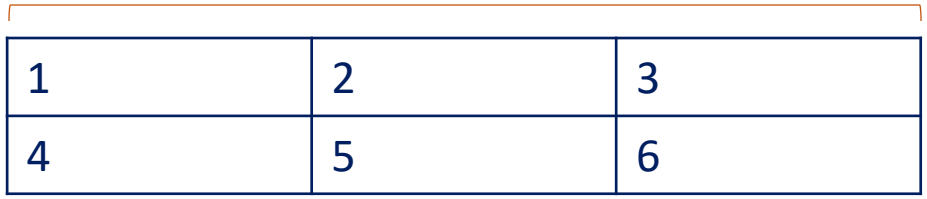
- 인덱스(Index): 각 요소마다 정해진 위치
  - 특정 인덱스에 위치하는 요소 접근법: *identifier*[*인덱스*]
  - 첫 요소의 인덱스: [0]
  - 요소가 n개인 리스트에서 마지막 요소의 인덱스: [n-1] 또는 [-1] (List의 길이를 모르더라도 [-1]을 써서 항상 마지막 요소 값을 얻을 수 있다)
  - 리스트의 일부 범위의 요소들에 접근법: *Identifier*[*시작인덱스*:*끝인덱스*]
  - 시작 인덱스의 값을 포함하여 가져오며, 끝 인덱스의 값은 포함하지 않는다
  - 시작 인덱스가 0일 때와 끝 인덱스가 n일 때는 각각 생략이 가능하다

```
#Identifier = myList, list의 길이 = 5
myList = [100, 50, 70, 90, 240]
#index     0    1   2   3    4
#index    -5   -4  -3  -2   -1

print(myList[0])
print(myList[-5])

subList1 = myList[1:3]
# subList1 = [50, 70]
subList2 = myList[:3]
# subList2 = [100, 50, 70]
```

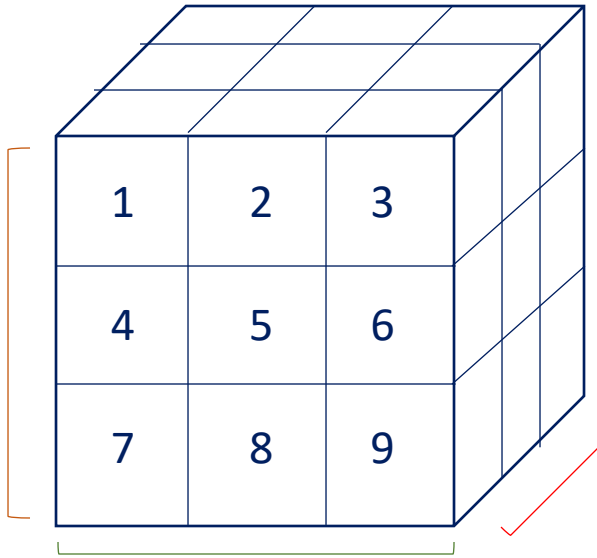
중첩 리스트

- 다차원 행렬은 중첩된 리스트의 모습으로 표현 가능하다
- 각 요소 값은 다중 인덱스 사용으로 접근한다
  - myVector[3], myMatrix[1][0], myCube[0][1][0]

2차 인덱스



1차 인덱스



2차 인덱스

1차 인덱스

3차 인덱스

```
myVector = [1,2,3,4]
myMatrix = [
            [1,2,3],
            [4,5,6]
        ]

myCube = [
            [
                [1,2,3],
                [4,5,6],
                [7,8,9]
            ],
            [
                [10,11,12],
                [13,14,15],
                [16,17,18]
            ],
            [
                [19,20,21],
                [22,23,24],
                [25,2627]
            ]
        ]
```

Data Representation

- 내가 풀고자 하는 문제에 따라 데이터를 어떤 자료형으로 표현할지 설계하는 것이 프로그래밍에 요구된다
  ex) 학생 5명의 수학, 과학, 영어 점수를 표현하라

```
student_1_score = [100,50,70]
student_2_score = [55,60,55]
student_3_score = [40,30,30]
student_4_score = [20,10,15]
student_5_score = [80,70,20]
```

```
math = [100,55,40,20,80]
science = [50,60,30,10,70]
english = [75,55,30,15,20]
```

```
score = [
        [100,50,70],
        [55,60,55],
        [40,30,30],
        [20,10,15],
        [80,70,20]
    ]
```

Q1. 과목별 평균을 구하는 프로그램
Q2. 학생마다 얻은 평균점수를 구하는 프로그램
Q3. 과학 점수를 기준으로 정렬해서 과목별 점수들을 출력하는 프로그램

리스트 연산자

- 요소 값 변경
  - Identifier[index] = new_value
- 요소 값 삭제
  - del: 인덱스로 삭제
  - pop(index): 인덱스로 삭제
  - remove(value): 값으로 삭제
- 요소 값 추가
  - append(value): 맨 마지막에 추가
  - insert(index, value): index위치에 추가
  - + : 두 list를 결합
- 멤버십 연산자
  - List 안에 값이 존재하는지 여부를 True, False로 알려줌
  - in
  - not in

```
myList1 = [1,2,3]
myList2 = [4,5,6]
myList3 = [7,8,9]

#[1,2,1]
myList1[2] = 1

# [4,5]
del myList2[2]

# [4]
myList2.pop(1)

# []
myList2.remove(4)

#[7,8,9,10]
myList3.append(10)

#[7,8,9,5,10]
myList3.insert(3,5)

# [1,2,1,7,8,9,5,10]
newList = myList1 + myList3

#True
1 in myList1

#True
2 not in myList3
```

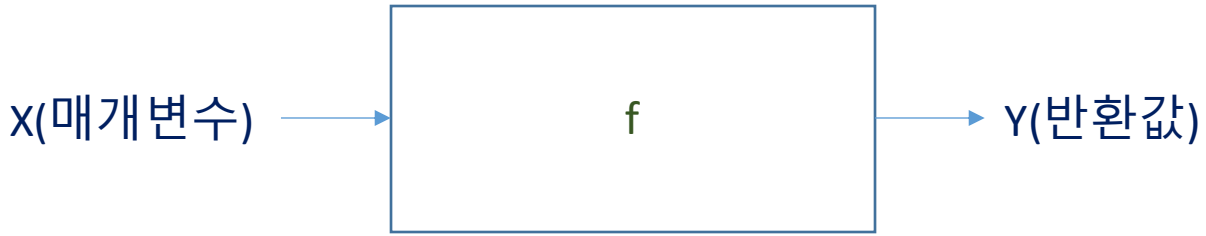
함수

- 함수(Function)
  - 어떠한 작업을 하는 코드를 모아 묶어 이름을 붙인것 (=코드 여러줄)
  - 함수의 이름과 명령문의 묶음으로 구성되어 있다
  - 프로시저(procedure), 메서드(method) 등으로 불림

- 파이썬 함수의 정의 구조

```
def 함수명(매개 변수):
    명령문1
    명령문2
    ...
    명령문n
    return 반환값
```

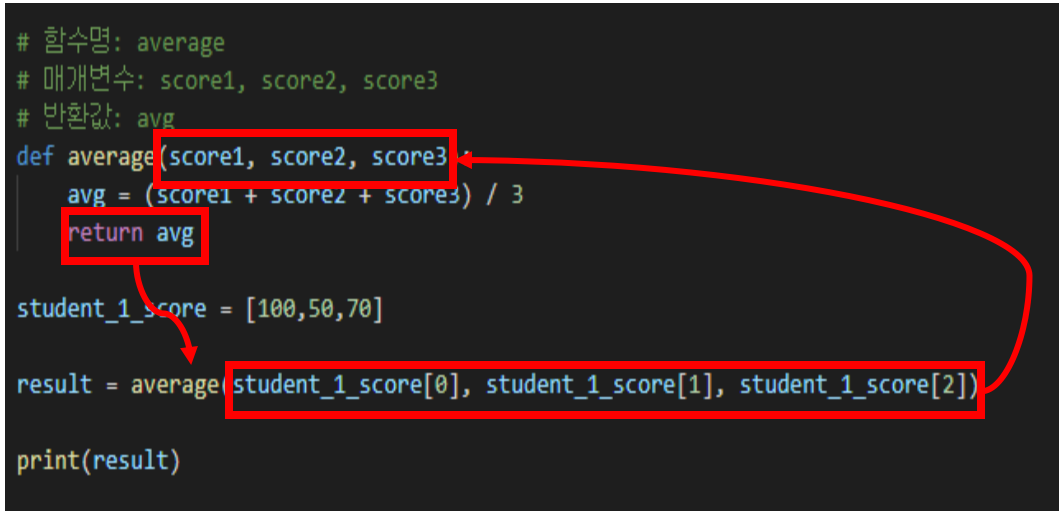
  - 매개 변수가 없는 함수도 가능하다
  - 반환값이 없는 함수도 가능하다




```python
# 함수명: average
# 매개변수: score1, score2, score3
# 반환값: avg
def average(score1, score2, score3):
    avg = (score1 + score2 + score3) / 3
    return avg
```

함수의 사용

- 함수의 호출(function call)
  - *함수의 이름(전달 인자)*
  - 전달 인자: 함수를 호출할때 매개변수에 넣어주는 값
  - 매개 변수: 함수 실행시 내부에서만 사용되는 식별자
    - 전달하는 인자와 반환하는 값이 어떤 타입인지 알아야 한다
  - 함수를 호출하는 것은 실제 함수의 명령문들을 그대로 대입하는것과 같은 효과를 보임



=

```
student_1_score = [100,50,70]

score1 = student_1_score[0]
score2 = student_1_score[1]
score3 = student_1_score[2]

avg = (score1 + score2 + score3) / 3
result = avg

print(result)
```

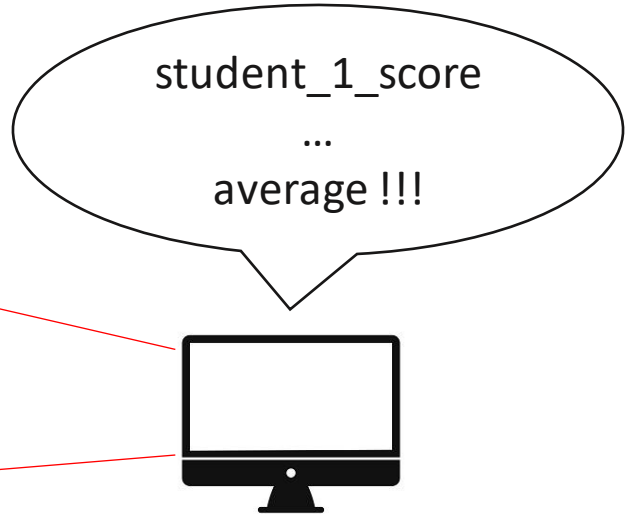
들여쓰기

- 들여쓰기 (Indentation)
  - 프로그래밍 언어는 코드를 블록화하여 처리 순서, 방식을 제어한다
  - 파이썬은 블록의 구분을 위하여 들여쓰기를 사용
    - 일반적으로 Tab(space 4번)
    - 원칙적으로는 space를 사용 권장

함수의 블록

코드의 블록

```
# 함수명: average
# 매개변수: score1, score2, score3
# 반환값: avg
def average(score1, score2, score3):
    avg = (score1 + score2 + score3) / 3
    return avg

student_1_score = [100,50,70]

result = average(student_1_score[0], student_1_score[1], student_1_score[2])

print(result)
```



함수 사용의 이유

- 재사용성
  - 코드의 불필요한 반복을 피할 수 있음

```python
# 함수명: average
# 매개변수: score1, score2, score3
# 반환값: avg
def average(score1, score2, score3):
    avg = (score1 + score2 + score3) / 3
    return avg

student_1_score = [100,50,70]
student_2_score = [55,60,55]
student_3_score = [40,30,30]
student_4_score = [20,10,15]
student_5_score = [80,70,20]

result1 = average(student_1_score[0], student_1_score[1], student_1_score[2])
result2 = average(student_2_score[0], student_2_score[1], student_2_score[2])
result3 = average(student_3_score[0], student_3_score[1], student_3_score[2])
result4 = average(student_4_score[0], student_4_score[1], student_4_score[2])
result5 = average(student_5_score[0], student_5_score[1], student_5_score[2])
```

함수 사용의 이유

- 가독성(안정성)
  - 프로그램을 체계적이고 간결하게 작성 가능함
  - 내부 동작에 대해 알 필요가 없음

  Ex) print() 함수

```
student_1_score = [100,50,70]

result = average(student_1_score[0], student_1_score[1], student_1_score[2])

print(result)
```

```
student_1_score = [100,50,70]

score1 = student_1_score[0]
score2 = student_1_score[1]
score3 = student_1_score[2]

avg = (score1 + score2 + score3) / 3
result = avg

print(result)
```