

대학 수학 학습을 위한 파이썬 부트캠프

III: Python 제어문의 활용

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.

프로그래밍 공통 문법

1. 입출력
2. 변수 & 자료형
3. 조건문
4. 반복문
5. 함수

Remind: 들여쓰기

- 들여쓰기 (Indentation)

- 파이썬은 프로그램은 코드를 **한 줄 씩(line by line)** 차례대로 **실행**
- 프로그래밍 언어는 코드를 블록화하여 처리 순서, 방식을 제어한다
- 파이썬은 블록의 구분을 위하여 들여쓰기를 사용
 - 콜론(:) 기호를 통해 블록의 시작을 알림
 - 일반적으로 Tab(space 4번)
 - 원칙적으로는 space를 사용 권장

함수의 블록(처리 방식 제어)
= 함수의 정의, 실행

메인 코드의 블록
= 실행

```
# 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)
```

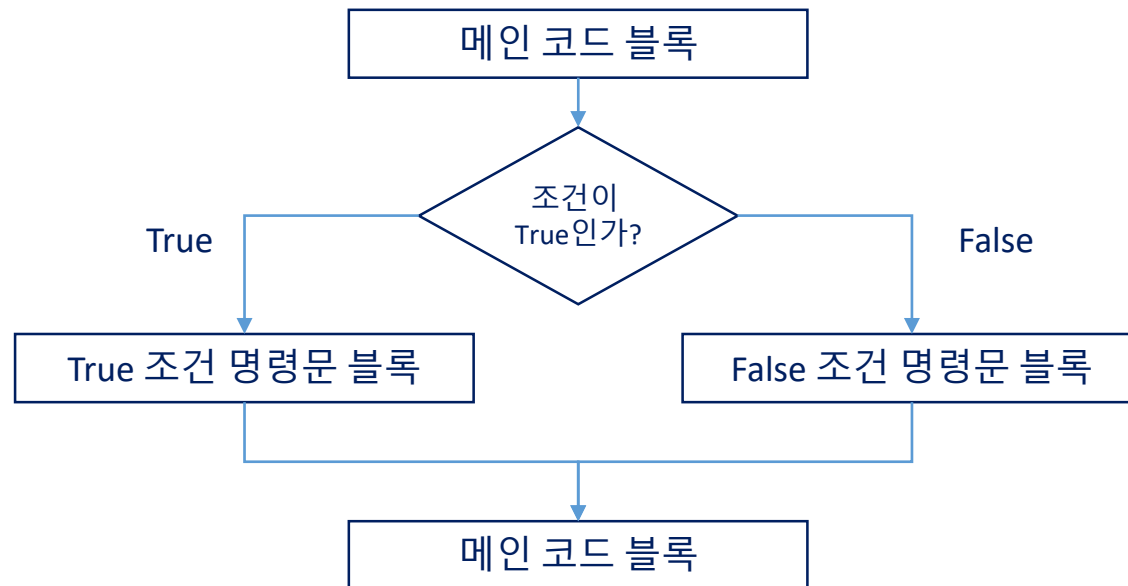
student_1_score
...
average !!!



조건문

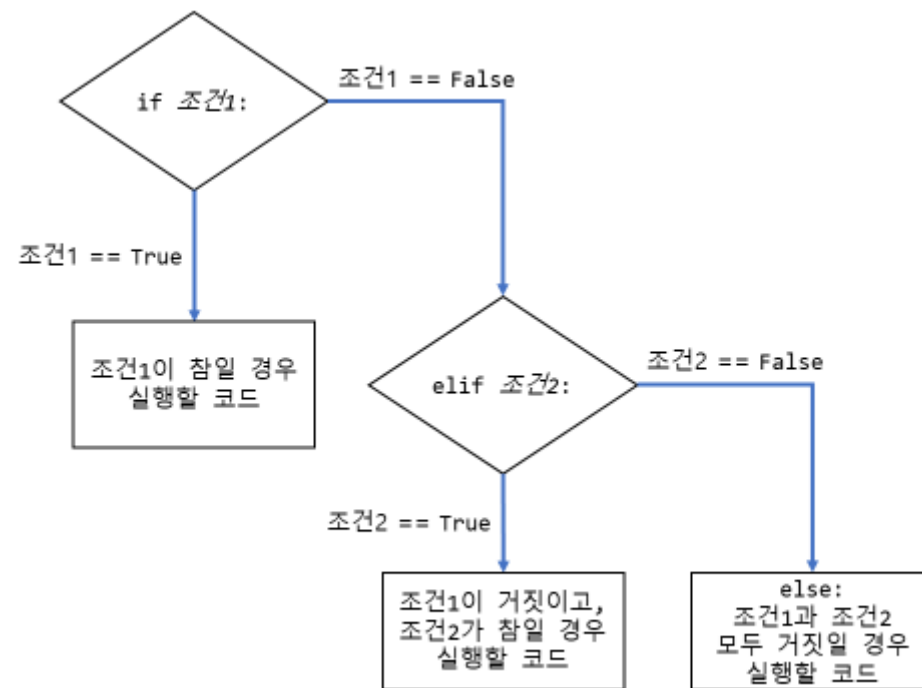
- 조건문

- 어떤 조건에 따라 프로그램의 논리가 특정한 방향으로 진행되는 경우에 활용
- 다수의 선택지가 주어지고 조건문의 결과에 따라서 해당하는 코드 블록을 실행한다



if 조건문

- 조건은 참(True)과 거짓(False)로 구분되는 값을 가져야 한다
- if문은 반드시 한 개 있어야 한다
 - elif문과 else문은 생략 가능하다
 - else문은 한 개만 올 수 있고, 항상 마지막에 쓸 수 있다
- if, elif, else 문의 마지막에는 콜론(:)을 붙인다
 - 조건을 참(True)으로 만족하는 경우 해당 조건문 블록의 명령문을 실행한다
 - 모든 조건문이 거짓이면 else 블록의 명령문을 실행한다



조건

- 조건은 결과값이 항상 참(True) 또는 거짓(False)인 명제만 가능하다

== : 같다

!= : 다르다

>= : 크거나 같다

<= : 작거나 같다

> : 크다

< : 작다

in : 포함된다

not in: 포함되지 않는다

```
x = 10
y = 20
z = 30
myList = [10,25,30]

x == y           #False
x != y           #True
x >= y           #False
x <= y           #True
x > y            #False
x < y            #True
x in myList      #True
z not in myList  #False
x != 20 and x in myList #True and True == True
y == (x+z)/2 or z > 40 #True or Flase == True
```

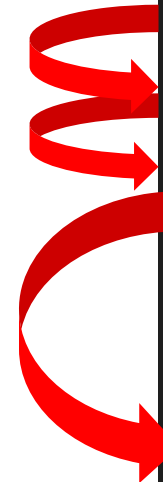
- 프로그래밍 언어는 and와 or를 통해 연쇄적으로 조건을 확인할 수 있다
 - and: 좌우 조건이 모두 참일 참
 - or: 좌우 조건중 하나라도 참이면 참

if문 활용 예제

- 학생의 성적에 따른 학점을 주는 프로그램

90점 이상	A 학점
80점 이상 90점 미만	B 학점
70점 이상 80점 미만	C 학점
60점 이상 70점 미만	D 학점
60점 미만	F 학점

배점 규칙



```
student_score = 76
grade = ''

if student_score >= 90:
    grade = 'A'
elif student_score >= 80:
    grade = 'B'
elif student_score >= 70:
    grade = 'C'
elif student_score >= 60:
    grade = 'D'
else:
    grade = 'F'

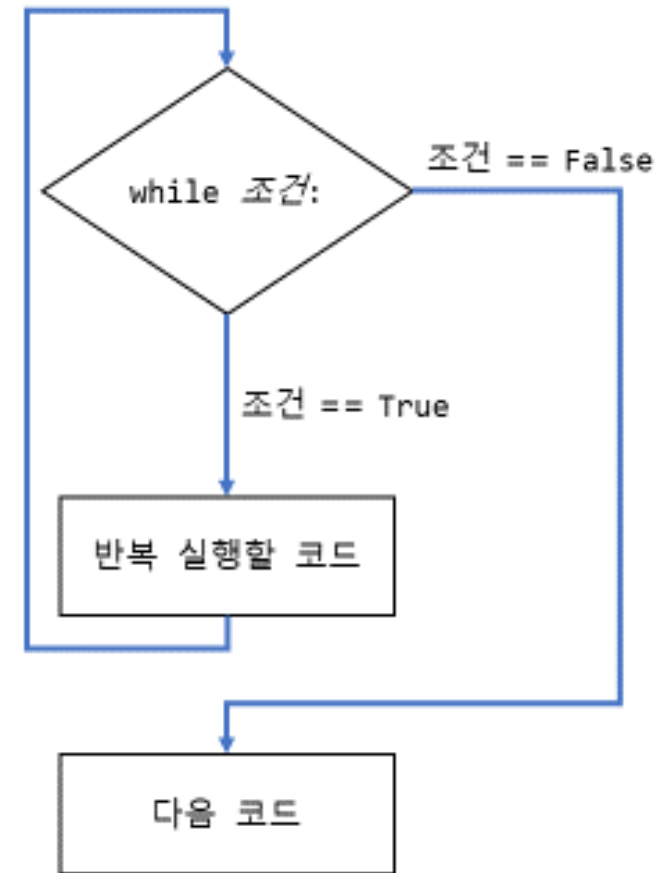
print(grade)
```

반복문

- 반복문(loop)
 - 특정 작업을 반복하여 실행하는 명령문
- 반복문에서 정해놓은 조건을 만족하는 동안 해당 코드 블록을 실행한다
 - 코드 블록 실행 이후 정해놓은 조건을 만족하는지 다시 검사
 - 조건을 만족하면 코드 블록 명령 반복
- 파이썬에서는 while, for 두가지 반복문을 지원한다

while 반복문

- if문과 유사한 동작
 - while문에는 조건이 들어온다
- while문의 마지막에는 콜론(:)을 붙인다
 - while문에 있는 조건이 참이면 while명령문 코드 블록을 실행한다
 - while문에 있는 조건이 거짓이면 메인 코드 블록을 실행한다



while문 활용 예제

- 학생 '들'의 성적에 따른 학점을 주는 프로그램

90점 이상	A 학점
80점 이상 90점 미만	B 학점
70점 이상 80점 미만	C 학점
60점 이상 70점 미만	D 학점
60점 미만	F 학점

배점 규칙

while 명령문 코드 블록

```
# index: 0, 1, 2, 3, 4
student_score = [76, 40, 60, 52, 99]
index = 0
grade = ''

while index < 5:
    if student_score[index] >= 90:
        grade = 'A'
    elif student_score[index] >= 80:
        grade = 'B'
    elif student_score[index] >= 70:
        grade = 'C'
    elif student_score[index] >= 60:
        grade = 'D'
    else:
        grade = 'F'

    print(grade)
    index = index + 1

print('여기는 메인 코드 블록입니다.')
```

실행 결과

```
C
F
D
F
A
여기는 메인 코드 블록입니다.
```

for 반복문

- while문에 비해 복합자료형에 특화된 동작

for 변수 in 복합자료형변수(순회형):
for-명령문

```
myList = [1,2,3,4,5,6,7,8,9]
for x in myList:
    #for - 명령문
    #...
```

- 다수의 값을 가지고 있는 복합자료형(List, Tuple, Dictionary, Set)변수의 값을 차례로 변수에 할당한 후 for명령문을 실행한다
 1. 변수는 복합자료형 변수의 첫번째 값을 할당 받는다
 2. for 명령문 실행
 3. 변수는 복합자료형 변수의 두번째 값을 할당 받는다
 4. for 명령문 실행
 5. ...
 6. 마지막 값을 할당받고 for 명령문 실행

for문 활용 예제

- 학생 '들'의 성적에 따른 학점을 주는 프로그램

90점 이상	A 학점
80점 이상 90점 미만	B 학점
70점 이상 80점 미만	C 학점
60점 이상 70점 미만	D 학점
60점 미만	F 학점

배점 규칙

for 명령문 코드 블록

```
#      index:    0,  1,  2,  3,  4
student_score = [76, 40, 60, 52, 99]
grade = ''

for score in student_score:
    if score >= 90:
        grade = 'A'
    elif score >= 80:
        grade = 'B'
    elif score >= 70:
        grade = 'C'
    elif score >= 60:
        grade = 'D'
    else:
        grade = 'F'

    print(grade)

print('여기는 메인 코드 블록입니다.')
```

실행 결과

```
C
F
D
F
A
여기는 메인 코드 블록입니다.
```

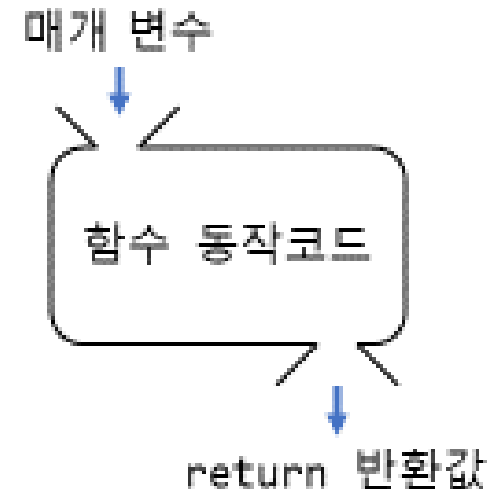
함수

- 함수(Function)

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

- 파이썬 함수의 정의 & 구조

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



재귀 함수

- 재귀 함수(Recursion Function)
 - 자기 자신을 호출하는 함수
- 함수가 마치 반복문과 같은 동작을 하게 만드는 방법
 - 계속 호출할 경우 무한 루프에 빠진다
 - 종료 조건을 주어서 함수 호출을 중단하고 빠져나올 방법을 주어야 한다

- 피보나치 수열

$$F_2 = F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \geq 3)$$

```
def fibonacci_1(n):  
    if n == 1:  
        return 1  
    elif n == 2:  
        return 1  
    else:  
        return fibonacci_1(n-1) + fibonacci_1(n-2)
```

재귀 함수의 동작

$$F_4 = 3$$

```
def fibonacci_1(4):  
    if n == 1:  
        return 1  
    elif n == 2:  
        return 1  
    else:  
        return fibonacci_1(4-1) + fibonacci_1(4-2)
```

⑩ return 3

⑦ return 2 ⑨ return 1

①

```
def fibonacci_1(3):  
    if n == 1:  
        return 1  
    elif n == 2:  
        return 1  
    else:  
        return fibonacci_1(3-1) + fibonacci_1(3-2)
```

④ return 1 ⑥ return 1

②

```
def fibonacci_1(2):  
    if n == 1:  
        return 1  
    elif n == 2:  
        return 1  
    else:  
        return fibonacci_1(2-1) + fibonacci_1(2-2)
```

⑧

```
def fibonacci_1(2):  
    if n == 1:  
        return 1  
    elif n == 2:  
        return 1  
    else:  
        return fibonacci_1(2-1) + fibonacci_1(2-2)
```

③

```
def fibonacci_1(1):  
    if n == 1:  
        return 1  
    elif n == 2:  
        return 1  
    else:  
        return fibonacci_1(1-1) + fibonacci_1(1-2)
```

⑤

재귀 함수와 반복문

- 모든 재귀 함수는 반복문으로 표현 가능하다
 - 재귀 함수: 짧은 코드로 표현 가능해 가독성이 높다
 - 반복문: 컴퓨터에 부담이 적으며 빠르다
 - 하드웨어의 성능이 열악한 시대가 아니기 때문에 가독성과 성능 중 목적에 맞는 프로그래밍이 필요

```
def fibonacci_2(n):  
    fn = 1  
    fn_1 = 1  
    fn_2 = 0  
  
    i = 1  
    while(i < n):  
        fn = fn_1 + fn_2  
  
        i = i + 1  
        fn_1 , fn_2 = fn, fn_1  
  
    return fn
```