제어문

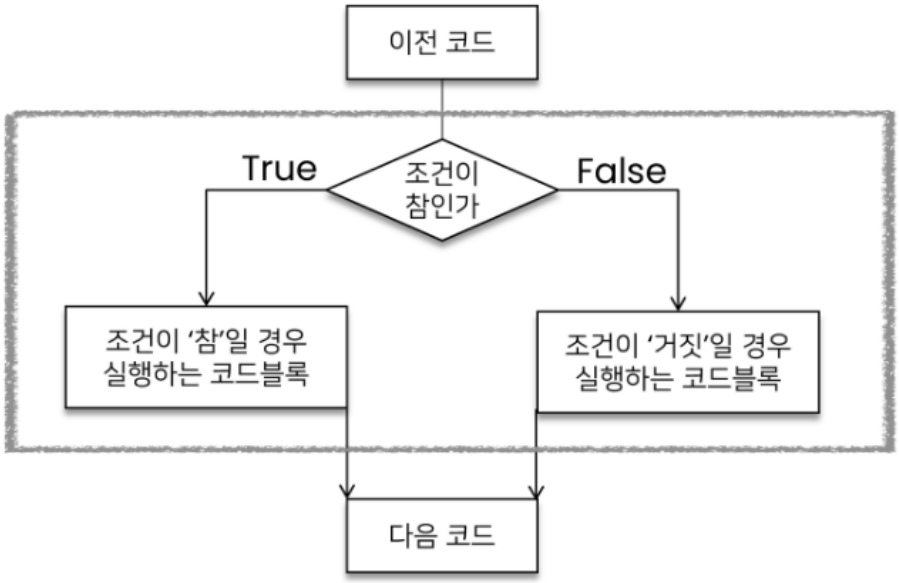
작성일시	@2022년 7월 20일 오전 9:03
내용	문법 제어문 파이썬
주차	1주차
자료	

제어문(Control Statement)

- 파이썬: 위에서 아래로 명령 수행
- 특정 상황에 따라 코드를 **선택적으로 실행(분기/조건)**하거나 **계속하여 실행(반복)**
- **순서도(Flowchart)**로 표현 가능
- 조건문
- 반복문

조건문(Conditional Statement)

- 참/거짓을 판단할 수 있는 조건식과 함께 사용



기본 형식

- 참/거짓에 대한 조건식

```
if 조건 == True:
    # Run this Code
else:
    # Run this Code
```

복수 조건문

- elif를 활용하여 표현

```
if 조건:
    # Code block
elif 조건:
    # Code block
elif 조건:
    # Code Block
else:
    # Code Block
```

조건 표현식(Conditional Expression)

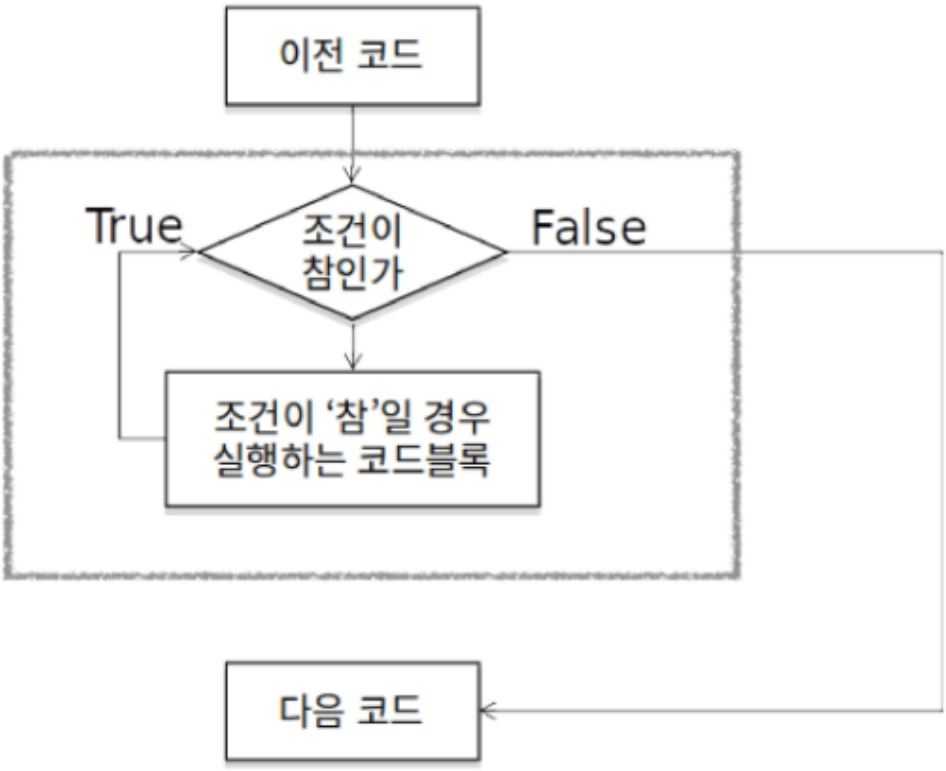
- 일반적으로 조건에 따라 값을 정할 때 활용
- **삼항 연산자(Ternary Operateor)라** 부르기도 함

```
True인 경우 값 if 조건 else false인 경우 값

# ex1) 절댓값 저장하기
num = int(input())
num = -num if num >= 0 else - num
# ex2) 홀수/짝수 판별하기
num = int(input())
result = '홀수입니다' if num % 2 else '짝수입니다'
print(result)
```

반복문

- 특정 조건을 만족할 때까지 같은 동작을 계속 반복하고 싶을 때 사용



While 문

- While문은 조건식이 참인 경우 반복적으로 코드를 실행
  - 조건이 참인 경우 들여쓰기 되어있는 코드 블럭이 실행됨
  - 코드 블록이 모두 실행되고, 다시 조건식을 검사하여 반복적으로 실행됨

- While문은 무한루프를 하지 않도록 **종료조건**이 반드시 필요

```
a = 0
while a < 5:    #종료조건
    print(a)
    a += 1
print('끝')
```

- 복합 연산자(In - Place Operator)
  - 연산과 할당을 합쳐 놓은 것

```
a = 0
while a < 5:
    print(a)
    a += 1   # 복합 연산자
print('끝')
```

for 문

- 시퀀스(str, tuple, list, range)를 포함한 순회 가능한 객체(iterable)의 요소를 모두 순회
  - 처음부터 끝까지 모두 순회하므로 별도의 종료 조건이 필요하지 않음
- Iterable
  - 순회할 수 있는 자료형(string, list, dict, tuple, range, set 등)
  - 순회형 함수(range, enumerate)

```
for 변수명 in iterable:
        # Code block

for fruit in ['apple', 'mango', 'banana']:
    print(fruit)
print('끝')

'''
apple
mango
banana
Rmx
'''
```

- for문 을 이용한 문자열(string)순회

```
chars = input()

#happy

#1 바로 하나씩 꺼내오기
for char in chars:
        print(char)

'''
h
a
p
p
y
'''

#2 인덱스 슬라이싱 활용하기
for idx in range(len(chars)):
        print(chars[idx])

'''
h
a
p
p
y
'''
```

- 딕셔너리(Dictionary) 순회
  - key를 순회하며 key를 통해 값을 활용

```
grades = {'john' : 80, 'eric' : 90}

for student in grades:
    print(student, grades[student])

'''
john 80
eric 90
'''
```

  - 추가 메서드 활용
    - keys(): key로 구성된 결과
    - values(): value로 구성된 결과
    - items(): (key, value)의 튜플로 구성된 결과

```
# 추가 메서드
grades = {'john' : 80, 'eric' : 90}
print(grades.keys())
print(grades.values())
print(grades.items())

'''
dict_keys {['john', 'eric']}
dict_values {[80, 90]}
dict_items {[{'john', 80), ('eric', 90)]}
'''

# for문에서 활용
grades = {'john' : 80, 'eric' : 90}

for student, grade in grades.items():
    print(student, grade)

'''
john 80
eric 90
'''
```

- enumerate 순회
  - enumerate()
    - 인덱스와 객체를 쌍으로 담은 열거형(enumerate) 객체 반환
    - (index, value)형태의 tuple로 구성된 열거 객체를 반환

```
members = ['민수', '영희', '철수' ]

for idx, number in enumerate(members):
        print(idx, number)

'''
0 민수
1 영희
2 철수
'''
```

List Comprehension

- 표현식과 제어문을 통해 특정한 값을 가진 리스트를 간결하게 생성하는 방법

```
[code for 변수 in iterable]
[code for 변수 in iterable if 조건식]

# 1 ~ 3의 세제곱 리스트 만들기
```

```
cubic_list = []
for number in range(i, 4):
    cubic_list.append(number ** 3)
print(cubic_list)

# [1, 8, 27]

# List comprehension 활용하기
cubic_list = [number ** 3 for number in range(1, 4)]
print(cubic_list)

# [1, 8, 27]
```

Dictionary Comprehension

```
{code for 변수 in iterable]}
{code for 변수 in iterable if 조건식}

cubic_dict = {number: number ** 3 for number in range(1, 4)}
print(cubic_dict)

# [1, 8, 27]
```

반복문 제어

- break
  - 반복문을 종료
- continue
  - continue 이후의 코드 블록은 수행하지 않고, 다음 반복을 수행
- for-else
  - 끝까지 반복문을 실행한 이후에 else문 실행
    - break을 통해 중간에 종료되는 경우 else문은 실행되지 않음
- pass
  - 아무것도 하지 않음(문법적으로 필요하지만, 할 일이 없을 때 사용)