Python_02

제어문(Control statement)

- 순차, 선택, 반복구조
- 파이썬은 기본적으로 위에서부터 아래로 차례대로 명령을 수행
- 특정 상황에 따라 코드를 선택적으로 실행(분기/조건)하거나 계속하여 실행(반복)하는 제어가 필요함
- 제어문은 순서도(flowchart)로 표현이 가능

코드 스타일 가이드

PEP 8 - Style Guide for Python Code

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python. This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide[2].

https://peps.python.org/pep-0008/

styleguide

Python is the main dynamic language used at Google. This style guide is a list of dos and don'ts for Python programs. To help you format code correctly, we've created a settings file for Vim. For Emacs, the default settings should be fine. Many teams use the yapf auto-formatter to avoid arguing over formatting.

https://google.github.io/styleguide/pyguide.html

```
print('hello')
print("world")
# 혼용 불가

if True:
        print(True)
else:
            print(False)

a= 'apple'
```

잘못된 스타일 가이드

—>
—>
—>
—>
—>
—>

```
print('hello')
print('world')

if True:
    print(True)
else:
    print(False)

a = 'apple'
```

올바른 코드 스타일 가이드

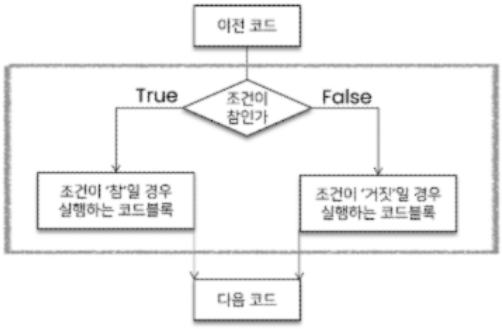
들여쓰기

- Space Sensitive
  - 문장을 구분할 때, 중괄호({,}) 대신 들여쓰기를 사용
  - 들여쓰기를 할때는 4칸(space*4) 혹은 1탭(1Tab)을 입력
    - 주의! 한 코드 안에서는 반드시 한 종류의 들여쓰기를 사용 → 혼용 금지
  - Tab으로 들여쓰면 계속 탭으로 틀여써야 함
  - 원칙적으로는 공백(빈칸, space)사용을 권장

조건문

조건문은 참/거짓을 판단할 수 있는 조건식과 함께 사용

순서도 == 생각을 가시화



조건문

실행문

조건문 기본

- 조건에는 참/거짓에 대한 조건식
  - 조건이 참인 경우 이후 들여쓰기 되어있는 코드 블록을 실행
  - 이외의 경우 else이후 들여쓰기 되어있는 코드 블록을 실행
    - else는 선택적으로 활용할 수 있음

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

```
a = 5
if a > 5:
    print('5 초과')
else:
    print('5 이하')
print(a)
```

```
num = int(input())

if num % 2:
    print('홀수')
else:
    print('짝수')
```

복수 조건문

복수의 조건식을 활용할 경우 elif를 활용하여 표현함

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

```
dust = int(input())

if dust > 150:
    print('매우 나쁨')
elif dust > 80:
    print('나쁨')
elif dust > 30:
```

```
    print('보통')
else:
    print('좋음')
print('미세먼지 확인 완료')
```

중첩 조건문

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

```
dust = int(input())

if dust < 0:
    print('값이 잘못되었습니다')
elif dust > 150:
    print('매우 나쁨')
    if dust >= 300:
        print('실외 활동을 자제하세요')
elif dust > 80:
    print('나쁨')
elif dust > 30:
    print('보통')
else:
    print('좋음')


print('미세먼지 확인 완료')
```

조건 표현식

- 조건 표현식(Conditional Expression)이란?
  - 조건 표현식을 일반적으로 조건에 따라 값을 정할 때 활용
  - 삼항 연산자(Ternary Operator)로 부르기도 함

```
True인 경우 값 if 조건 else False인 경우
value = num if num >= 0 else -num
```

```
num = 2
result = '홀수' if num % 2 else '짝수'
print(result)
```

```
num = -5
value= num if num >= 0 else 0
print(value)

#################################

num = -5
if num >= 0:
   value = num
else:
    value = 0
print(value)
```

> Falsy
>
> [ ], ( ),{ }, “”, None

반복문

- while 문
  - 종료 조건에 해당하는 코드를 통해 반복문을 종료시켜야 함
- for 문
  - 반복가능한 객체를 모두 순회하면 종료(별도의 종료 조건이 필요 없음)
- 반복 제어
  - break, continue, for-else

While문

- while문은 조건식이 참인 경우 반복적으로 코드를 실행
  - 조건이 참인 경우 들여쓰기 되어있는 코드 블록이 실행됨
  - 코드 블록이 모두 실행되고, 다시 조건식을 검사하며 반복적으로 실행됨
  - while문은 무한 루프를 하지 않도록 종료 조건이 반드시 필요

```
while 조건:
    # Code block
```

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

> 복합 연산자(In-Place Operator)
>
> - 복합 연산자는 연산과 할당을 합쳐 놓은 것
>
> ex) 반복문을 통해서 개수를 카운트하는 경우

for문

- for 문은 시퀀스(string, tuple, list, range)를 포함한 순회 가능한 객체(iterable)의 요소를 모두 순회
- 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')
```

1. for문을 이용한 문자열(String) 순회

```
chars = input()
for i in chars:
    print(i)
```

```
chars = input()
for i in range(len(chars)):
    print(chars[i])
```

2. 딕셔너리(Dictionary) 순회

- 딕셔너리는 기본적으로 Key를 순회하며, Key를 통해 값을 활용

```
grades = {'john' : 80, 'eric': 90}
for student in grades:
    print(student, grades[student])
```

3. 추가 메서드를 활용한 딕셔너리(Dictionary) 순회

- 추가 메서드를 활용하여 순회할 수 있음
  - keys() : key로 구성된 결과
  - values() : value로 구성된 결과
  - items(): (key, value)의 튜플로 구성된 결과

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

print(grades.keys()) # dict_keys(['john', 'eric'])
print(grades.values()) # dict_values([80, 90])
print(grades.items()) # dict_items([('john', 80), ('eric', 90)])

for student, grade in grades.items():
    print(student, grade)
'''
john 80
eric 90
'''
```

4. 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
'''
print(enumerate(members)) # <enumerate object at 0x00000248F7476080>
print(list(enumerate(members))) # [(0, '민수'), (1, '영희'), (2, '철수')]
print(list(enumerate(members, start=1))) # [(1, '민수'), (2, '영희'), (3, '철수')]
```

5. 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 세제곱의 결과가 담긴 리스트

```
cubic_list = []
for number in range(1,4):
    cubic_list.append(number ** 3)
print(cubic_list) # [1, 8, 27]

cubic_list = [number ** 3 for number in range(1,4)]
print(cubic_list) # [1, 8, 27]
```

6. Dictionary Comprehension

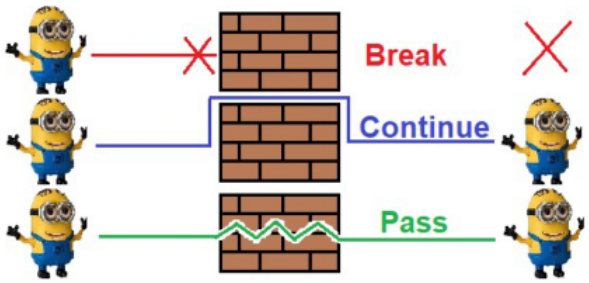
- 표현식과 제어문을 통해 특정한 값을 가진 딕셔너리를 간결하게 생성하는 방법

```
{key : value for 변수 in iterable}
{key : value for 변수 in iterable if 조건식}
```

```
cubic_dict = {}
for number in range(1,4):
    cubic_dict[number] = number ** 3
print(cubic_dict) # {1: 1, 2: 8, 3: 27}

cubic_dict = {number : number ** 3 for number in range(1,4)}
print(cubic_dict) # {1: 1, 2: 8, 3: 27}
```

반복문 제어



- break
  - 반복문을 종료

```
n = 0
while True:
    if n == 3:
        break
    print(n)
    n += 1
```

- continue
  - continue 이후의 코드 블록은 수행하지 않고, 다음 반복을 수행

```
for i in range(6):
    if i % 2 == 0:
        continue
    print(i)
```

- for-else
  - 끝까지 반복문을 실행한 이후에 else문 실행
  - break를 통해 중간에 종료되는 경우 else 문은 실행되지 않음

```
for char in 'apple':
    if char == 'b':
        print('b!')
        break
else:
    print('b가 없습니다.')
# b가 없습니다.
```

```
for char in 'banana':
    if char == 'b':
        print('b!')
        break
else:
    print('b가 없습니다.')
# b!
```

- pass
  - 아무것도 하지 않음(문법적으로 필요하지만, 할 일이 없을 때 사용)

```
for i in range(4):
    if i == 2:
        pass
    print(i)
'''
0
1
2
3
'''
```

```
for i in range(4):
    if i == 2:
        continue
    print(i)
'''
0
1
3
'''
```