

데이터과학을 위한 파이썬 프로그래밍



04. 조건문과 반복문

목차

1. 조건문
2. Lab: 어떤 종류의 학생인지 맞추기
3. 반복문
4. Lab: 구구단 계산기
5. 조건문과 반복문 실습
6. Lab: 숫자 찾기 게임
7. Lab: 연속적인 구구단 계산기
8. Lab: 평균 구하기
9. 코드의 오류를 처리하는 방법

01

조건문

01. 조건문

- 만약 다음처럼 성적이 나열되어 있을 때, 학점을 부여하는 프로그램은 어떻게 만들 수 있을까?

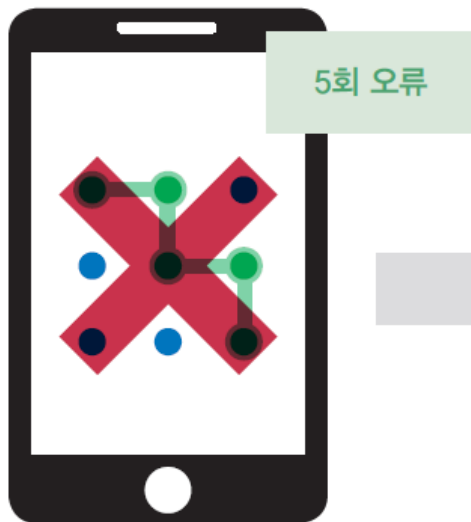
• 점수: 38, 65, 89, 16, 95, 71, 63, 48, 49, 66, 37

- ① 점수에 따른 학점의 기준을 정한다. ex) 95점 이상 'A+', 60점 미만 'F'
 - ② 기준을 바탕으로 첫 번째 점수를 판단한다. ex) 38점은 60점 미만이므로 'F'
 - ③ 다음 점수로 이동하면서 ②를 반복한다. ex) 37점은 60점 미만이므로 'F'
 - ④ 더 이상 판단할 점수가 없을 때 프로그램을 종료한다
- 어떤 기준으로 결정해야 하는가? → **조건의 설정**
 - 언제까지 해야 하는가? → **반복의 설정**

01. 조건문

■ 조건문의 개념

- **조건문(conditional statement):** 조건에 따라 특정 동작을 하도록 하는 프로그래밍 명령어이다.
- 파이썬에서는 조건문을 사용하기 위해 if, else, elif 등의 명령 키워드를 사용한다.
- 스마트폰 잠금 해제 패턴이 5회 틀리면, 20초 동안 대기 상태로 만들어라.



조건의 기준



실행할 명령

```
import time
time.sleep(5) # delay for 5 seconds

from time import sleep
sleep(0.1)    # delay for 0.1 seconds
```

[조건문의 예시: 스마트폰 잠금 해제 패턴]

01. 조건문

■ if -else문

- if-else문의 기본 문법은 다음과 같다.

if <조건>:	# if를 쓰고 조건 삽입 후 ':' 입력
→ <수행 명령 1-1>	# 들여쓰기 후, 수행 명령 입력
<수행 명령 1-2>	# 같은 조건에서 실행일 경우 들여쓰기 유지
else:	# 조건이 불일치할 경우 수행할 명령
<수행 명령 2-1>	# 조건 불일치 시 수행할 명령 입력
<수행 명령 2-2>	# 조건 불일치 시 수행할 명령 들여쓰기 유지

- ① if 뒤에는 참과 거짓을 판단할 수 있는 조건문이 들어가야 하고, 조건문이 끝나면 반드시 **콜론(:)**을 붙여야 한다.
- ② **들여쓰기(indentation)**를 사용하여 해당 조건이 참일 경우 수행할 명령을 작성한다.
- ③ if의 조건이 거짓일 경우 else문이 수행된다. **else문은 생략해도 상관없다.** 만약 조건에 해당하지 않는 경우에 따로 처리해야 한다면 else문을 넣으면 된다.

예제: if-else.py

↑ 줄리

■ if -else문 : [코드 4-1] 해석

- 1~2행 : 먼저 나이를 입력하는 메시지가 나타나면 사용자가 나이를 입력한다.
- 3~6행 : 만약, 나이가 30세 미만일 경우 클럽에 입장할 수 있다는 메시지를 출력하고, 30세 이상일 경우에는 클럽에 입장할 수 없다는 메시지를 화면에 출력한다. 여기서 핵심은 myage < 30이라는 조건이다. 사용자가 입력한 값은 myage 변수에 정수형으로 바뀌 할당 된다. 다음으로 그 값이 30보다 작다는 조건이 True일 때, 즉 myage < 30이 참일 경우 print("Welcome to the Club")이라는 구문이 동작하고, 해당 조건이 False일 때는 else로 묶인 구문이 동작한다.

01. 조건문

■ 조건의 판단 : 비교 연산자, True 혹은 False를 반환

- 비교 연산자(또는 조건 연산자): 어떤 것이 큰지 작은지 같은지를 비교하는 것으로, 그 비교 결과는 참(True)이나 거짓(False)이 된다.

비교 연산자	비교 상태	설명
$x < y$	~보다 작음	x가 y보다 작은지 검사
$x > y$	~보다 큼	x가 y보다 큰지 검사
$x == y$	같음	x와 y의 값이 같은지 검사
$x \text{ is } y$	같음(메모리 주소)	x와 y의 메모리 주소가 같은지 검사
$x != y$	같지 않음	x와 y의 값이 같지 않은지 검사
$x \text{ is not } y$	같지 않음(메모리 주소)	x와 y의 메모리 주소가 같지 않은지 검사
$x >= y$	크거나 같음	x가 y보다 크거나 같은지 검사
$x <= y$	작거나 같음	x가 y보다 작거나 같은지 검사

[비교 연산자]

01. 조건문

■ 조건의 판단 : 비교 연산자

- = 연산자는 배정(assignment)의 의미. ==연산자는 비교 결과를 반환한다.
- 다음 코드에서 두 값이 모두 같으니 결과는 True이다. 사실 조건문 코드를 볼 때, 언제나 이러한 코드가 True 또는 False로 치환된다고 생각하면 이해하기 쉽다.

```
>>> 7 == 7
```

01. 조건문



다음 페이지의 설명으로 같음

a==b.py

■ 조건의 판단 : 비교 연산자

- == 연산자는 두 변수의 값이 같음을 비교하지만, is 연산자는 두 변수의 메모리상 주소가 같은지를 비교한다.
- ~~파이썬은 처음 인터프리터를 시작할 때 5~256까지 변하지 않는 메모리(정적 메모리) 주소에 값을 할당한다. 그리고 해당 값을 다른 변수가 사용할 때, 같은 메모리 주소를 사용하여 참조한다.~~ → 현 단계에서는 일부 맞고 일부 틀림.
- 다음 코드에서 a와 b, 둘 다 100일 때는 is와 ==이 모두 같다고 나오지만, 둘 다 300일 경우에는 값만 같고 메모리 주소는 다르다고 나온다. : 실제로 수행해 보면 모두 True가 나옴.

```
>>> a = 100
>>> b = 100
>>> a is b
True
>>> a == b
True
>>> a = 300
>>> b = 300
>>> a == b
True
>>> a is b
False
```

a==b.py
실습 0

참고: 변수의 메모리 배열과 최적화 문제는 파이썬 버전, 상황에 따라 다른 듯 하니 현 단계에서는 크게 유념하지 않았으면 좋겠음.

마지막 줄의 a is b의 실제 수행 결과가 아래에 표기한 바와 같이 교과서는 다르게 True로 출력된다.

→ True

Optimization in Python — the Interning Technique for Improved Performance

a==b.py
실습 1

- **Variable interning:** Interning: 구금하다. 변수를 cache화하여 그 값을 읽을 때는 미리 저장해 놓은 고속 메모리 공간에서 가져오게 한다는 개념. 고속화를 위한 전략. 실제로 어떤 방식으로 운영되는지는 현 단계 이상 수준의 학습이 필요해 보인다.
- -5~256까지의 수에 대해서는 값을 변수에 할당하지 않고 단지 그 값을 지시하는 주소만을 관리한다.
- 값을 읽어오지 않고도 주소만으로 그 값을 활용하는 방식으로 수행시간 단축을 도모하고자 하는 목적으로 활용되었던 것 같다.
- 그러나, 실험에 의하면 이러한 전략은 단지 값의 범위[-5, 256]만으로 결정되는 것은 아닌 것으로 보인다.
- 범위를 넘어서는 어떤 값에 대해서도 variable interning을 실행한다.
- 그러나 range() 함수를 사용할 때는 값의 범위에도 영향받을 때가 있다.
- 명시적으로 sys.intern()을 사용하여 interning을 행하면 실질적으로도 빠른 속도의 액세스가 가능한 것을 확인하였다.

01. 조건문

■ 조건의 판단 : True와 False의 치환

- 컴퓨터는 기본적으로 이진수만 처리할 수 있으며, True는 1로, False는 0으로 처리한다.
- 아래 코드를 실행하면 True가 출력된다. 그 이유는 앞서 설명한 것처럼 컴퓨터는 존재하면 True, 존재하지 않으면 False로 처리하기 때문이다.

```
>>> if 1: print("True")
... else: print("False")
```

```
In[2]: if 1: print("True")
...: else: print("False")
...:
True
In[3]: if 0: print("True")
...: else: print("False")
```

- 아래 코드를 실행하면 True가 출력된다. 먼저 $3 > 5$ 는 False이고 False는 결국 0으로 치환된다. 그래서 이것을 다시 치환하면 $(0) < 10$ 이 되고, 이 값은 참이므로 True가 반환된다.

```
>>> (3 > 5) < 10
```

```
In[2]: a=3<5
In[3]: a
Out[3]: True
In[4]: a < 10
Out[4]: True
```

파이썬에서는 bool의 값을 1과 0으로 표현하지 않고, True, False로 표현한다.
A=1 혹은 A=0을 수행하면 변수A는 정수형으로 만들어진다.



```
>>> A=True
>>> A
True
>>> type(A)
<class 'bool'>
>>> A=1
>>> A
1
>>> type(A)
<class 'int'>
```

01. 조건문

■ 조건의 판단 : 논리 연산자

- 논리 연산자는 and · or · not문을 사용해 조건을 확장할 수 있다.

연산자	설명	예시
and	두 값이 모두 참일 경우 True, 그렇지 않을 경우 False	(7 > 5) and (10 > 5)는 True (7 > 5) and (10 < 5)는 False
or	두 값 중 하나만 참일 경우 True, 두 값 모두 거짓일 경우 False	(7 < 5) or (10 > 5)는 True (7 < 5) or (10 < 5)는 False
not	값을 역으로 반환하여 판단	not (7 < 5)는 True not (7 > 5)는 False

[논리 연산자]

01. 조건문

■ 조건의 판단 : 논리 연산자

- and는 둘 다 참이어야 True, or는 둘 중 하나만 참이어도 True, not은 참이면 False이고 거짓이면 True를 출력한다.

```
>>> a = 8
>>> b = 5
>>> a == 8 and b == 4
False
>>> a > 7 or b > 7
True
>>> not (a > 7)
False
```

01. 조건문

■ if-elif-else문

- 중첩 if문을 간단히 표현하려면 if-elif-else문을 사용한다.
- 다음 같은 점수판이 있다고 가정하자.

점수(score)	학점(grade)
98	
37	
16	
86	
71	
63	

[점수판]

■ if-elif-else문

- 점수에 맞는 학점을 주기 위해 [코드 4-2]와 같이 코드를 입력하면, 어떤 학점으로 계산될까?

코드 4-2 grade.py

```
1 score = int(input("Enter your score: "))
2
3 if score >= 90:
4     grade = 'A'
5 if score >= 80:
6     grade = 'B'
7 if score >= 70:
8     grade = 'C'
9 if score >= 60:
10    grade = 'D'
11 if score < 60:
12    grade = 'F'
13
14 print(grade)
```

Enter your score: 98

D

← 사용자 점수 입력

← 잘못된 값 출력

만약 점수가 98점이라면,
3, 5, 7, 9번 줄을 모두 수행하게 될 것이다.

■ if-elif-else문

- 실제 [코드 4-2]를 실행하면 모든 값이 'D'나 'F'로 나온다.
- 이유는 바로 다음 그림과 같이 코드가 한 줄씩 차례대로 실행되기 때문이다.

```
score = 98
if score >= 90: True
    grade = 'A' grade = 'A'
if score >= 80: True
    grade = 'B' grade = 'A' → 'B'
if score >= 70: True
    grade = 'C' grade = 'B' → 'C'
if score >= 60: True
    grade = 'D' grade = 'C' → 'D'
if score < 60: False
    grade = 'F' grade = 'D' → 'D'
print(grade)
```

[if문만을 이용한 학점 계산기에 98을 할당했을 때 결과 산출 방식]

■ if-elif-else문

- [코드 4-2]의 문제를 해결하기 위해서는 여러 개의 조건을 하나의 if문에서 검토할 수 있도록 elif를 사용한 if-elif-else문으로 작성해야 한다. elif는 else if의 줄임 말로, if문과 같은 방법으로 조건문을 표현할 수 있다.

코드 4-3 if-elif-else.py

```
1 score = int(input("Enter your score: "))
2
3 if score >= 90: grade = 'A'           # 90 이상일 경우 A
4 elif score >= 80: grade = 'B'        # 80 이상일 경우 B
5 elif score >= 70: grade = 'C'        # 70 이상일 경우 C
6 elif score >= 60: grade = 'D'        # 60 이상일 경우 D
7 else: grade = 'F'                    # 모든 조건에 만족하지 못할 경우 F
8
9 print(grade)
```

Enter your score: 98

A

← 사용자 점수 입력

← 올바른 값 출력

02

Lab: 어떤 종류의 학생인지 맞추기

02. Lab: 어떤 종류의 학생인지 맞추기

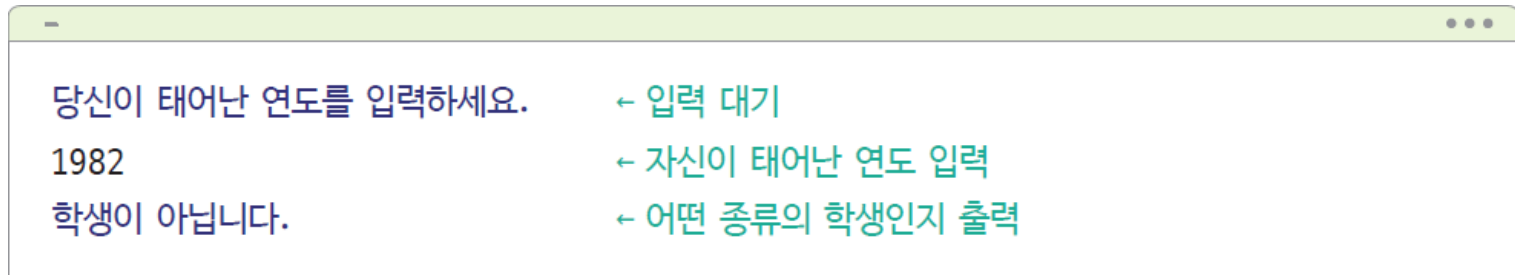
■ 실습 내용

- 조건문을 이용하여 '어떤 종류의 학생인지 맞추는 프로그램'을 만들어 보자.
- 이 프로그램을 작성하는 규칙은 다음과 같다.

- 나이는 (2020 - 태어난 연도 + 1)로 계산
- 26세 이하 20세 이상이면 '대학생'
- 20세 미만 17세 이상이면 '고등학생'
- 17세 미만 14세 이상이면 '중학생'
- 14세 미만 8세 이상이면 '초등학생'
- 그 외의 경우는 '학생이 아닙니다.' 출력

02. Lab: 어떤 종류의 학생인지 맞추기

■ 실행 결과



당신이 태어난 연도를 입력하세요.	← 입력 대기
1982	← 자신이 태어난 연도 입력
학생이 아닙니다.	← 어떤 종류의 학생인지 출력

02. Lab: 어떤 종류의 학생인지 맞추기

예제: student.py

■ 문제 해결

- 어떤 종류의 학생인지 맞추는 프로그램의 결과 코드는 [코드 4-4]와 같다.

코드 4-4 student.py

```
1 print("당신이 태어난 연도를 입력하세요.")
2 birth_year = input()
3 age = 2020 - int(birth_year) + 1
4
5 if age <= 26 and age >= 20:
6     print("대학생")
7 elif age < 20 and age >= 17:
8     print("고등학생")
9 elif age < 17 and age >= 14:
10    print("중학생")
11 elif age < 14 and age >= 8:
12    print("초등학생")
13 else:
14    print("학생이 아닙니다.")
```

02. Lab: 어떤 종류의 학생인지 맞추기

예제: student.py

■ 문제 해결 : [코드 4-4] 해석

- 1행 : 처음 프로그램을 실행하면 '당신이 태어난 연도를 입력하세요.'를 출력하도록 코딩한 것이다.
- 2행 : 사용자의 입력을 birth_year 변수에 할당하는데, 사용자의 입력은 문자형이므로 계산을 위해 3행에서 int() 함수를 사용하여 정수형으로 변경한다.
- 3행 : 나이를 계산한 후, 이 값을 age에 할당한다. 사용자가 '1982'를 입력하면 (2020- 1982 + 1)로 계산하여 결과값 '39'를 age 변수에 할당한다. 지금부터 age 변수값이 39라고 가정하고 설명하겠다.
- 5~12행 : age의 값으로 각각 if문과 elif문을 수행한다. 첫 번째 조건 age <= 26 and age >= 20은 True일까, False일까? 두 가지 조건을 모두 만족해야 하는데 39 <= 26과 39 >= 20은 두 번째 조건만 True이므로 and 구문에 의해 전체는 False로 변환된다. 첫 번째조건이 False이므로 다음 조건을 비교한다.

02. Lab: 어떤 종류의 학생인지 맞추기

여기서  잠깐! 논리 연산자 없이 비교 연산자를 사용할 경우

```
>>> 1 <= 2 < 10
True
>>> 1 <= 100 < 10
False
```

```
>>> 1<2<10
True
>>> 1<100<10
False
>>> (1<100) < 10
True
>>> 1<2<=1
False
>>> (1<2)<=1
True
>>> 100<101<102
True
>>> 100<101<101
False
```

파이썬에서는 직관적인 표현을 허용하고 있다.
A<B<C 표현은 직관적인 표현 그대로 A는 B보다 작고, B는 C보다 작으면 True를 반환한다.
그러나 만약 괄호가 있으면 다르다.
(A<B) < C는 일단 A<B보다 작으면 True이고 이 값(사실상 1)이 C보다 작으면 True를 반환한다.

03

반복문

03. 반복문

■ 반복문의 개념

- **반복문(loop)** : 말 그대로 문장을 반복해 만드는 것으로, 정해진 동작을 반복적으로 수행할 때 내리는 명령어이다.
- 반복문은 모든 프로그램에서 핵심적으로 사용된다. 반복문은 반복 시작 조건, 종료 조건, 수행 명령으로 구성되어 있으며, 들여쓰기와 블록(block)으로 구분한다. 파이썬의 반복문은 for와 while 등의 명령 키워드를 사용한다.

03. 반복문

■ for~in 문

- **for~in 문:** 기본적인 반복문으로, 반복 범위를 지정하여 반복을 수행한다.
- for문으로 반복문을 만들 때는 먼저 for를 입력하고 반복되는 범위를 지정해야 한다.
- 범위를 지정하는 방법에는 두 가지가 있다. 첫 번째 방법은 [코드 4-5]와 같이 리스트를 사용하는 것이다. 두 번째 방법은 [코드 4-6]과 같이 변수 자체를 출력하는 방법이다.

문장 형식: *for target_list in expression_list*

expression_list는 sequence(list, tuple, string)와 같은 iterable object가 사용된다. 이 문장에 의해 생성된 iterator가 expression_list의 item을 하나씩 연속적으로 바꾸어 가면서 target_list에 assign한다.

--- 나중에 추가로 공부할 내용 ---

1) 참고: Python iterable과 iterator의 의미

2) 예제, iter_next.py 참조: iter() 함수와 next() 함수로 for~in loop를 분석

3) 예제, range_vs_list.py 참조: iterable_objec로 range()와 list 자료를 사용 할 때의 수행 속도 비교.

■ for문

- ① 리스트를 사용해서 반복되는 범위를 지정하는 방법

코드 4-5 for1.py

```
1 for loopier in [1, 2, 3, 4, 5]:  
2     print("hello")
```

```
hello  
hello  
hello  
hello  
hello
```

■ for문 : [코드 4-5] 해석

① 리스트를 사용해서 반복되는 범위를 지정하는 방법

- [코드 4-5]에서는 [1, 2, 3, 4, 5]라는 리스트를 사용하였다.
- 이 리스트에 있는 각각의 값을 하나씩 가져와 `looper`라는 변수에 할당하는데, 한 번 할당할 때마다 그 아래쪽에는 들여쓰기 한 명령문 구문 `print("hello")`를 실행한다.
- 최종적으로 [1, 2, 3, 4, 5]에서 값을 모두 한 번씩 수행하므로, 총 다섯 번의 반복이 일어나 'hello'가 다섯 번 출력된다.

■ for문

- ② 변수 자체를 출력하여 반복되는 범위를 지정하는 방법

코드 4-6 for2.py

```
1 for loop in [1, 2, 3, 4, 5]:  
2     print(loop)
```

```
1  
2  
3  
4  
5
```

■ for문 : [코드 4-6] 해석

- ② 변수 자체를 출력하여 반복되는 범위를 지정하는 방법
- 리스트 [1, 2, 3, 4, 5]의 각각의 값이 한 번 반복문을 돌 때마다 loop 변수에 할당되어 그 값들이 화면에 출력된다.

■ for문

- 만약 100번 반복해야 한다면 코드를 어떻게 작성해야 할까? 리스트를 가지고 1부터 100까지 모든 값을 적기에는 너무 오래 걸린다. 이럴 때는 '**range**'라는 키워드를 사용한다.

순차적인 값을 원소로 하는 list 자료를 만들어 사용하는 것 보다는 range() 함수를 사용하는 것이 수행 속도 관점에서 유리하다.

코드 4-7 for3.py

```
1 for loopier in range(100):  
2     print("hello")
```



```
hello          ← 100번 반복  
:  
:  
hello
```

■ for문 : [코드 4-7] 해석

- range 문법의 기본 구조

```
for 변수 in range(시작 번호, 마지막 번호, 증가값)
```

- ~~range는 마지막 번호의 마지막 숫자 바로 앞까지 리스트를 만든다. 즉, range(1, 5)라고 하면 [1, 2, 3, 4]의 리스트를 만들고, range(0, 5)라고 하면 [0, 1, 2, 3, 4]의 리스트를 만든다.~~
- => 초보자용 해석: 당분간 그렇게 이해해도 좋지만, 진실은 아니다!!
- 앞의 시작 번호와 증가값은 생략할 수 있으며, 생략했을 경우 초기값으로 시작 번호는 0을, 증가값은 1을 입력한다.

내부 원소가 많은 큰 규모의 리스트를 생성하는 것은 불합리하다.

예를 들어 100만회의 loop를 위해 100만개의 원소를 생성하는 것은 메모리 낭비이고 수행 시간 최적화 측면에서 불리하다.

range() 함수에 의해 생성된 값은 그 자체가 range형 객체라서 아직 iterable object로 바뀐 sequence 데이터는 아니다. 이 객체는 for 문과 협조하여 지정된 회수의 loop를 수행하게 한다.

* range 자료형이 따로 있다.

예제: for3.py

range() 함수는 자체의 자료형으로 취급된다.

```
a = range(5)
```

```
print(f'a = range(5): type(a)={type(a)}, a={a}')
```

a = range(5): type(a)=<class 'range'>, a=range(0, 5)

range 자료형은 list나 tuple로 바꿀 수 있다.

sequence 자료형: <class 'range'>, <class 'list'>, <class 'tuple'>, <class 'string'>

sequence 자료형은 모두 iterable_object이다.

따라서 "for k in iterable_object" 구문에 사용될 수 있다.

```
l = list(a)
```

```
print(f'l = list(a): type(l)={type(l)}, len(l)={len(l)}, l={l}')
```

l = list(a): type(l)=<class 'list'>, len(l)=5, l=[0, 1, 2, 3, 4]

```
t = tuple(a)
```

```
print(f't = list(a): type(t)={type(t)}, len(t)={len(t)}, t={t}')
```

t = list(a): type(t)=<class 'tuple'>, len(t)=5, t=(0, 1, 2, 3, 4)

03. 반복문

여기서 잠깐! 반복문에서 알아두면 좋은 상식

1. 반복문의 변수는 대부분 i, j, k 로 지정한다. 이것은 수학에서 변수를 x, y, z 로 정하는 것과 비슷한 프로그래밍 관례이다.
2. 반복문은 대부분 0부터 반복을 시작한다. 이것도 일종의 관례이다. 하지만 비주얼 베이직(Visual Basic)처럼 1부터 시작하는 언어도 있다. 프로그래밍 언어는 아주 오래전부터 발전했으며, 초기의 컴퓨터들은 메모리가 매우 작아 하나라도 작은 수부터 저장하는 것이 용이하였다. 그래서 0부터 시작하는 이진수의 특징 때문에 대부분 언어가 0부터 인덱스를 시작한다.
3. 반복문을 잘못 작성하면 무한 루프라고 하는 오류가 발생할 수 있다. 무한 루프는 반복 명령이 끝나지 않는 프로그램 오류로, CPU와 메모리 등 컴퓨터의 리소스를 과다하게 점유하여 다른 프로그램에도 영향을 미친다.

03. 반복문

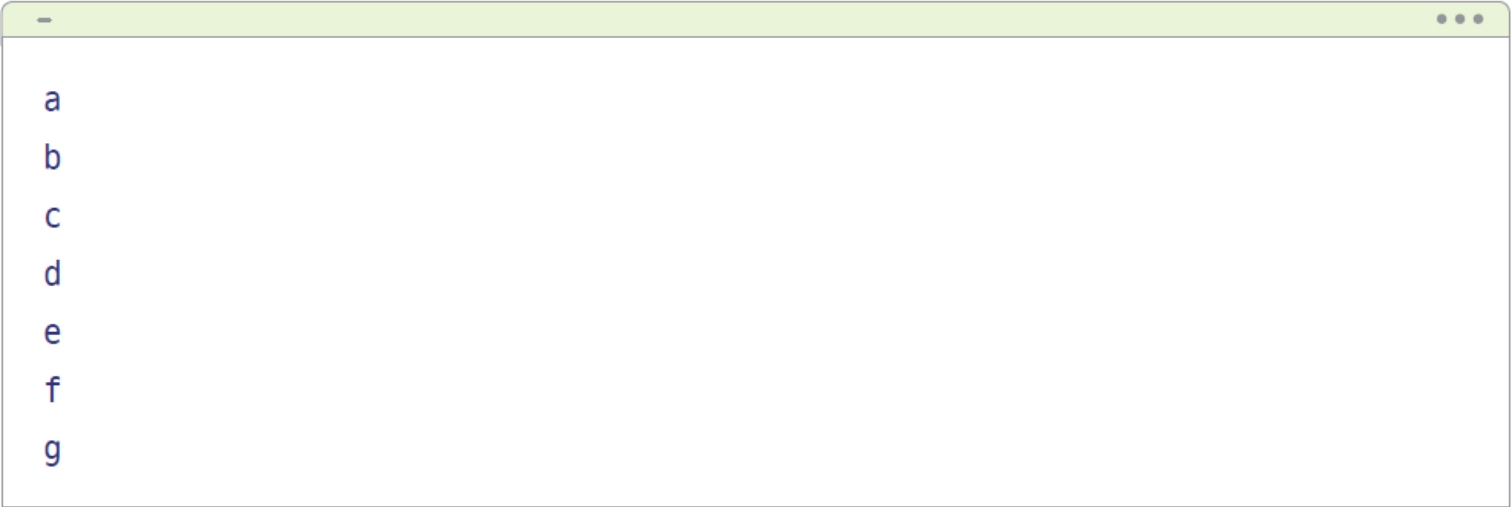
예제: for4.py

■ for문 *for ~ in 문에 사용할 수 있는 iterable object 사례: string*

- 문자열도 리스트와 같은 연속적인 데이터를 표현하므로 각 문자를 변수 i에 할당하여 화면에 출력한다.

코드 4-8 for4.py

```
1 for i in 'abcdefg':  
2     print(i)
```



```
a  
b  
c  
d  
e  
f  
g
```

■ for문 *for ~ in 문에 사용할 수 있는 iterable object 사례: list*

- 숫자를 화면에 출력하듯이 문자열로 이루어진 리스트의 값들도 사용할 수 있다.

코드 4-9 for5.py

```
1 for i in ['americano', 'latte', 'frappuccino']:
2     print(i)
```

```
americano
latte
frappuccino
```

03. 반복문

예제: for6.py

- **for문** *for ~ in 문에 사용할 수 있는 iterable object 사례: range()
range()는 range 타입 객체를 반환한다.*

- range 구문을 이용하여 1부터 9까지 2씩 증가시키는 for문을 [코드 4-10]에서 확인해 보자.

코드 4-10 for6.py

```
1 for i in range(1, 10, 2):      # 1부터 9까지 2씩 증가시키면서 반복문 수행
2     print(i)
```

```
1      For loop 안에서 range type object는 루프를 돌 때 마다 새로운 값을
3      반환한다.
5      본 사례의 경우는 i에게 반환한다.
7      미리 모든 값을 list 자료 등으로 만들어 두고 수행하는 것이 아니다.
9      이렇게 하는 것이 수행 속도가 빠르다.
```

■ for문

- range 구문을 사용하여 10부터 2까지 1씩 감소시키는 반복문은 [코드 4-11]과 같다.

코드 4-11 for7.py

```
1 for i in range(10, 1, -1):    # 10부터 2까지 1씩 감소시키면서 반복문 수행
2     print(i)
```



```
10
9
8
7
6
5
4
3
2
```


■ while문

- **while문** : 제시한 조건이 True이면 명령 블록을 수행하는 일을 반복한다. 해당 조건이 False 일 경우 반복 명령 블록을 더 이상 수행하지 않고 다음 명령 라인으로 이동한다.

코드 4-12 while.py

```
1 i = 1           # i 변수에 1 할당
2 while i < 10:    # i가 10 미만인지 판단
3     print(i)     # 조건을 만족할 때 i 출력
4     i += 1       # i에 1을 더하는 것을 반복하다가 i가 10이 되면 반복 종료
```



```
1
2
3
4
5
6
7
8
9
```

03. 반복문

여기서 잠깐! for문과 while문 상호 변환 가능

- for문과 while문은 기본적으로 유사하며, 서로 변환이 가능하다. 하지만 두 구문의 쓰임에는 차이가 있다. For문은 일반적으로 반복 횟수를 정확하게 알고 있고, 반복 횟수가 변하지 않을 때 사용한다. 반면, while문은 반복 실행 횟수가 명확하지 않고 어떤 조건을 만족하면 프로그램을 종료하고자 할 때 사용한다.
- 예를 들어, 학생들의 성적을 채점하는 프로그램을 작성한다고 하자. 이미 학생이 총 몇 명인지 명확하게 알고 있으므로 for문을 사용하는 것이 좋다. 하지만 가위바위보를 한다고 가정했을 때 '이기면 종료하라'라는 조건을 주면 언제 이길지 모르므로 while문을 사용하는 것이 낫다.

```
for i in range(0, 5)  
    print(i)
```

(a) 반복 실행 횟수를 명확히 알 때

```
i = 0  
while i < 5:  
    print(i)  
    i = i + 1
```

(b) 반복 실행 횟수가 명확하지 않을 때

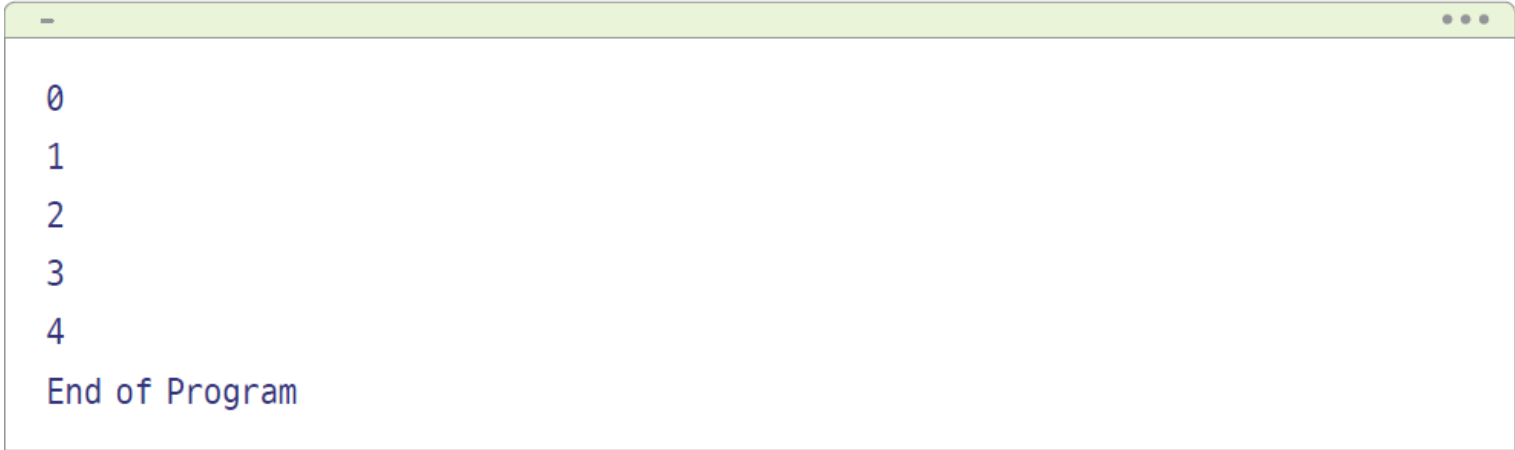
[for문과 while문 상호 변환]

■ 반복문의 제어 : break문

- **break문** : 반복문에서 반복을 종료하여 loop 문을 벗어난다.

코드 4-13 break.py

```
1 for i in range(10):  
2     if i == 5: break          # i가 5가 되면 반복 종료  
3     print(i)  
4 print("End of Program")      # 반복 종료 후 'End of Program' 출력
```



```
0  
1  
2  
3  
4  
End of Program
```

Break 문

예제: break.py 실습 1

break 문은 루프 문을 강제로 빠져나올 때, 즉, 아직 루프 조건이 `False`가 되지 않았거나 iterable object 자료의 끝까지 루프가 도달하지 않았을 경우에 루프 문의 실행을 강제로 정지시키고 싶을 때 사용된다.

break 문을 써서 for 루프나 while 루프를 빠져 나왔을 경우에는 그 루프에 딸린 else 블록은 실행되지 않는다.

while True:

s = input('Enter something : ')

if s == 'quit':

break *# 'quit'를 입력하여 이곳에 오면 while문을 빠져 나간다.*

else:

print('Hey. I am here.')

print('Length of the string is', len(s))

print('Done')

Enter something : abc

Hey. I am here.

Length of the string is 3

Enter something : 123 456

Hey. I am here.

Length of the string is 7

Enter something : quit

Done

■ 반복문의 제어 : continue문

- **continue문** : 특정 조건에서 남은 명령을 건너뛰고 다음 반복문을 수행한다.

코드 4-14 continue.py

```
1 for i in range(10):
2     if i == 5: continue      # i가 5가 되면 i를 출력하지 않음
3     print(i)
4 print("End of Program")     # 반복 종료 후 'End of Program' 출력
```

0

1

2

3

4

6 ←

7

8

9

End of Program

*continue 때문에 i=6인 다음 반복문을 수행한다.
이 때문에 5가 출력되지 않음.*

■ 반복문의 제어 : else문

- **else문** : for~in, while 블록문의 수행을 마치고 한 번 더 실행해 주는 역할을 한다.

코드 4-15 else.py

```
1 for i in range(10):
2     print(i)
3 else:
4     print("End of Program")
```

```
0
1
2
3
4
5
6
7
8
9
End of Program
```

```
while i < 43:
    i += 1
else:
    print(i)
```

else는 if 문 뿐만 아니라 for ~ in, while과도 함께 쓰이는 듯 하다.
if 문과 함께 쓰인다면 if 혹은 else 블록문이 둘 중의 하나만 수행되는 반면
for ~ in, while과 함께 수행되면 해당 블록이 수행되고 난 후
else 블록이 추가로 한 번 더 수행된다.
위 사례의 블록은 다음과 같이 코딩하는 것과 동작이 같다.
while i < 43:
i += 1
print(i)

03. 반복문

여기서 잠깐! 반복문을 제어하는 구문의 사용

- 개인적으로 위 제어 구문들은 되도록 사용하지 않는 것을 권한다. 특히 긴 코드를 작성할 때, 중간에 break문이나 continue문이 있다면 의도치 않게 코드가 오작동할 수도 있다. 특히 많은 사람과 함께 코딩할 때는 이러한 코드들로 인해 예측하지 못한 작동을 할 수도 있으니 주의해야 한다.

04

Lab: 구구단 계산기

04. Lab: 구구단 계산기

■ 실습 내용

- 이번 Lab에서는 앞에서 배운 반복문을 이용하여 구구단 계산기를 만들어 보자.
- 구구단 계산기의 규칙은 다음과 같다.

- 프로그램이 시작되면 '구구단 몇 단을 계산할까?'가 출력된다.
- 사용자는 계산하고 싶은 구구단 숫자를 입력한다.
- 프로그램은 '구구단 n단을 계산한다.'라는 메시지와 함께 구구단의 결과를 출력한다.

04. Lab: 구구단 계산기

■ 실행 결과

구구단 몇 단을 계산할까?

5

구구단 5단을 계산한다.

$5 \times 1 = 5$

$5 \times 2 = 10$

⋮

$5 \times 8 = 40$

$5 \times 9 = 45$

04. Lab: 구구단 계산기

예제: calculator1.py

■ 문제 해결

코드 4-16 calculator1.py

```
1 print("구구단 몇 단을 계산할까?")
2 user_input = input()
3 print("구구단", user_input, "단을 계산한다.")
4 int_input = int(user_input)
5 for i in range(1, 10):
6     result = int_input * i
7     print(user_input, "x", i, "=", result)
```

```
구구단 몇 단을 계산할까?
5
구구단 5 단을 계산한다.
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
```

■ 문제 해결 : [코드 4-16] 해석

- 1행 : 처음 프로그램을 실행하면 `print()` 함수를 사용하여 '구구단 몇 단을 계산할까?'를 출력할 수 있도록 코딩한다.
- 2행 : 사용자가 입력한 값을 `user_input` 변수에 할당한다.
- 3행 : `user_input` 변수를 이용하여 화면에 구구단을 계산한다는 메시지를 출력한다.
- 4행 : 아직 `user_input` 변수가 문자열 변수이므로 `int()` 함수를 사용하여 정수형으로 변환한다.
- 5~7행 : 입력된 숫자와 1부터 9까지 숫자의 곱을 반복문으로 표현한다. 따라서 `range(1, 10)`을 이용하여 1부터 9까지의 숫자를 `i`에 할당한다. 이 값들은 모두 사용자가 입력한 값의 정수형 변환값인 `int_input`과 곱하여 `result` 변수에 할당된다. 마지막으로 `print()` 함수에 의해 값이 출력된다.

05

조건문과 반복문 실습

05. 조건문과 반복문 실습

예제(수정본): reverse_sentence.py

■ 문자열 역순 출력

- `reverse_sentence` : 입력된 문자열을 역순으로 출력하는 변수이다.

```
1 sentence = "I love you."  
2 reverse_sentence = ''  
3 for char in sentence:  
4     reverse_sentence = char + reverse_sentence  
5     print(char+'|'+reverse_sentence)  
6  
7 print()  
8 print(reverse_sentence)  
9 print(len(sentence), len(reverse_sentence))
```

현재의 문자에 읽어 두었던 문자(열)를 더한다.

수행결과 ➡

```
I|I  
 | I  
l|l I  
o|ol I  
v|vol I  
e|evol I  
 | evol I  
y|y evol I  
o|oy evol I  
u|uoy evol I  
.|.uoy evol I  
  
.uoy evol I  
11 11
```

05. 조건문과 반복문 실습

■ 문자열 역순 출력

- **reverse_sentence** : 입력된 문자열을 역순으로 출력하는 변수이다.

코드 4-17 reverse_sentence.py

```
1 sentence = "I love you"
2 reverse_sentence = ' '
3 for char in sentence:
4     reverse_sentence = char + reverse_sentence
5 print(reverse_sentence)
```



uoy evol I

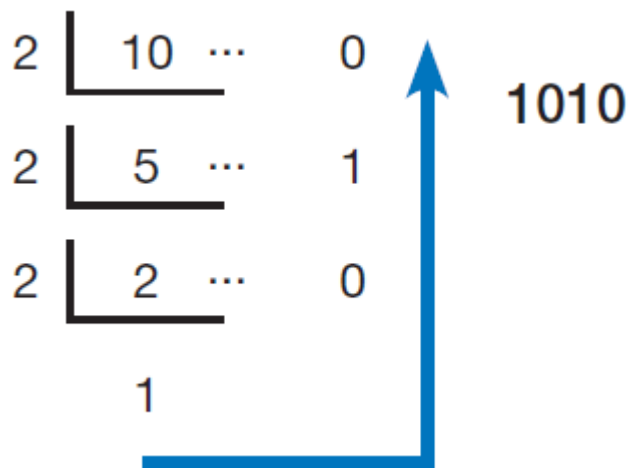
스트링 자료형은 immutable이다.

내부 원소 값을 바꿀 수는 없지만, 자료에 추가로 붙이는 것은 가능하다.

05. 조건문과 반복문 실습

■ 십진수를 이진수로 변환

- 십진수 숫자를 2로 계속 나눈 후, 그 나머지를 역순으로 취하면 이진수가 된다.



[십진수를 이진수로 변환하는 방법]

05. 조건문과 반복문 실습

예제: decimal.py

■ 십진수를 이진수로 변환

- 십진수를 이진수로 변환하는 코드

코드 4-18 decimal.py

```
1 decimal = 10
2 result = ''
3 while (decimal > 0):
4     remainder = decimal % 2    %는 나누고 난 후 나머지를 취하는 연산자
5     decimal = decimal // 2     //는 나누고 난 후 몫을 취하는 연산자.
6     result = str(remainder) + result
7 print(result)
```

나머지를 역순으로 수집하기 위해 현재의 나머지 스트링에 기존의 나머지 스트링을 더한다.

1010

05. 조건문과 반복문 실습

예제: decimal.py

■ 십진수를 이진수로 변환 : [코드 4-18] 해석

- 4행 : `remainder = decimal % 2` 코드는 나머지를 구해 `remainder` 변수에 저장하는 역할을 한다.
- 5행 : `decimal = decimal // 2` 코드는 현재의 십진수를 2로 나눈 몫을 다시 `decimal` 변수에 저장한다.
- 6행 : 값의 역순을 `result` 변수에 저장하는 `result = str(remainder) + result` 코드를 사용한다. 반복문이 진행될 때마다 진행되는 변수의 변화는 아래와 같다.

<u>Loop</u>	<u>decimal¹</u>	<u>remainder</u>	<u>decimal²</u>	<u>result¹</u>	<u>result²</u>
0	10	0	5		0
1	5	1	2	0	10
2	2	0	1	10	010
3	1	1	0	010	1010

[반복문이 진행될 때마다 진행되는 변수의 변화]

06

Lab: 숫자 찾기 게임

■ 실습 내용

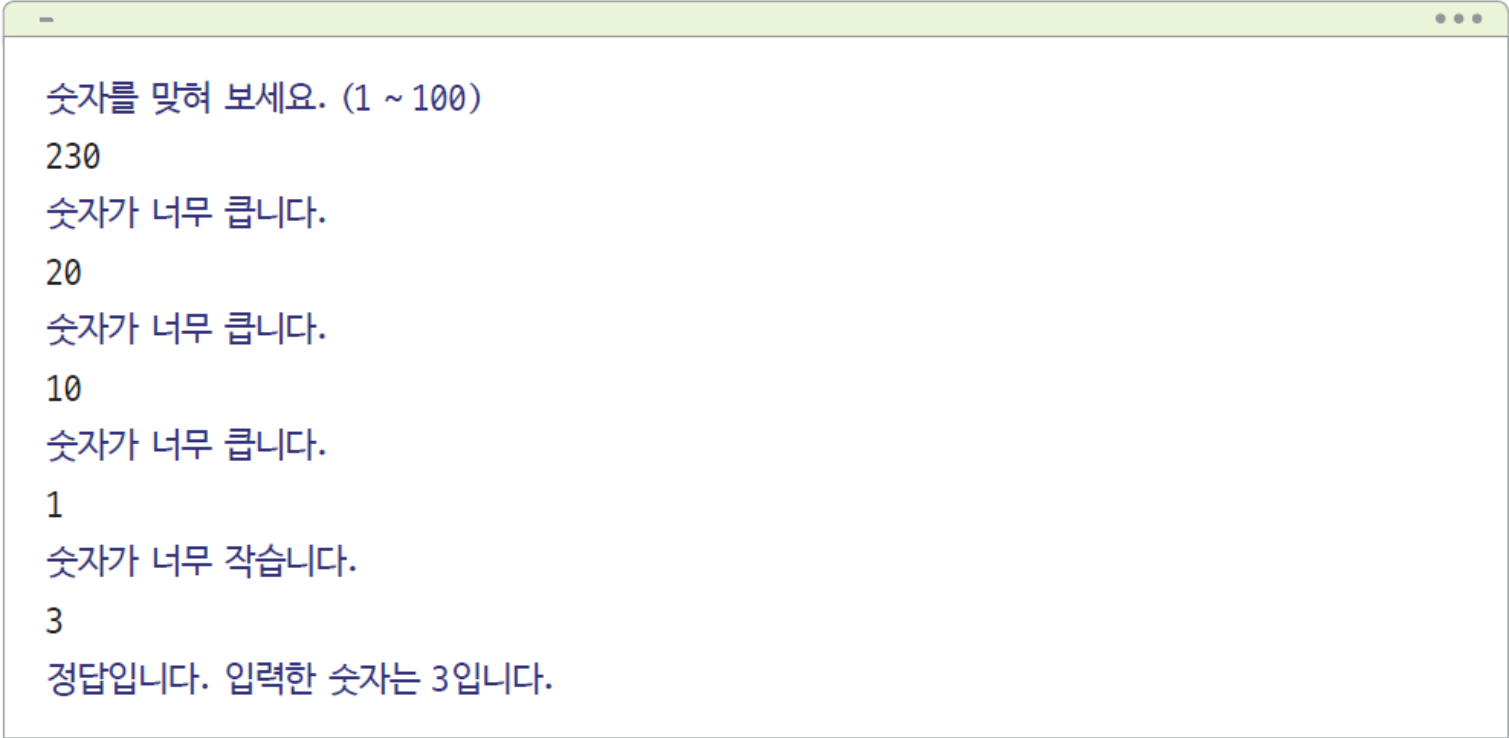
- 이번 Lab에서는 지금까지 배운 반복문과 조건문을 토대로 숫자 찾기 게임 프로그램을 만들어 보자.
- 이 프로그램의 규칙은 다음과 같다.

- 먼저 컴퓨터가 1에서 100까지 중 임의의 숫자를 생성한다.
- 다음으로 사용자가 추측하는 숫자를 입력하면 컴퓨터가 생성한 임의의 숫자보다 큰지, 작은지를 계속 비교해 준다.
- 정답을 맞힐 때까지 계속하다가 맞히면 '정답입니다. 입력한 숫자는 n입니다.'를 출력한다.

06. Lab: 숫자 찾기 게임

예제: guess_number.py

■ 실행 결과



```
숫자를 맞춰 보세요. (1 ~ 100)
230
숫자가 너무 큼니다.
20
숫자가 너무 큼니다.
10
숫자가 너무 큼니다.
1
숫자가 너무 작습니다.
3
정답입니다. 입력한 숫자는 3입니다.
```

06. Lab: 숫자 찾기 게임

예제: guess_number.py

■ 문제 해결

코드 4-19 guess_number.py

```
1 import random                                # 난수 발생 함수 호출
2 guess_number = random.randint(1, 100)        # 1~100 사이 정수 난수 발생
3 print("숫자를 맞춰 보세요. (1 ~ 100)")
4 users_input = int(input())                   # 사용자 입력을 받음
5 while (users_input is not guess_number):     # 사용자 입력과 난수가 같은지 판단
6     if users_input > guess_number:           # 사용자 입력이 클 경우
7         print("숫자가 너무 큼니다.")
8     else:                                    # 사용자 입력이 작을 경우
9         print("숫자가 너무 작습니다.")
10    users_input = int(input())                # 다시 사용자 입력을 받음
11 else:
12    print("정답입니다.", "입력한 숫자는", users_input, "입니다.") # 종료 조건
```

■ 문제 해결 : [코드 4-19] 해석

- 1행 : 난수를 생성하기 위해 `random` 모듈을 사용한다.
- 2행 : `random` 모듈 안에는 `randint()`라는 함수가 있다. `randint(1, 100)`은 1에서 100까지의 임의의 `random` 숫자를 생성하라는 뜻이다. 이 구문을 사용해 임의의 숫자를 `guess_number` 변수에 할당한다.
- 3행 : 사용자가 숫자를 입력할 수 있도록 `print()` 함수를 사용하였다.
- 4행 : 입력된 숫자가 정수형으로 변환되어 `user_input` 변수에 저장된다.
- 5~10행 : `user`가 입력한 `user_input`값과 컴퓨터가 만든 `guess_number`가 같지 않을 동안, 즉 `users_input is not guess_number` 조건이 만족하는 동안 `while`문은 계속 작동한다. `while`문은 `if`문을 사용해 숫자가 크면 '숫자가 너무 큼니다.', 숫자가 작으면 '숫자가 너무 작 습니다.'를 출력한다. 한 번의 반복문이 진행된 후 계속 반복되면서 사용자에게 다시 입력 을 받아 계속 비교한다.
- 11~12행 : `user_input`과 `guess_number`가 같아지면 반복문을 빠져나와 '정답입니다.'를 출 력한다.

07

Lab: 연속적인 구구단 계산기

07. Lab: 연속적인 구구단 계산기

예제: calculator2.py

■ 실습 내용

- 지금까지 배운 반복문과 조건문을 토대로 연속적인 구구단 계산기 프로그램을 만들어 보자.
- 이 프로그램의 규칙은 다음과 같다.

- 프로그램이 시작되면 '구구단 몇 단을 계산할까요(1~9)?'가 출력된다.
- 사용자는 계산하고 싶은 구구단 숫자를 입력한다.
- 프로그램은 '구구단 n단을 계산합니다.'라는 메시지와 함께 구구단의 결과를 출력한다.
- 기존 문제와 달리, 이번에는 프로그램이 계속 실행되다가 종료 조건에 해당하는 숫자(여기에서는 0)를 입력하면 종료된다.

07. Lab: 연속적인 구구단 계산기

예제: calculator2.py

■ 실행 결과

구구단 몇 단을 계산할까요(1~9)?

3

구구단 3단을 계산합니다.

$3 \times 1 = 3$

$3 \times 2 = 6$

$3 \times 3 = 9$

$3 \times 4 = 12$

$3 \times 5 = 15$

$3 \times 6 = 18$

$3 \times 7 = 21$

$3 \times 8 = 24$

$3 \times 9 = 27$

구구단 몇 단을 계산할까요(1~9)?

5

구구단 5단을 계산합니다.

구구단 몇 단을 계산할까요(1~9)?

11

잘못 입력했습니다 1부터 9 사이 숫자를 입력하세요.

9

구구단 9단을 계산합니다.

$9 \times 1 = 9$

$9 \times 2 = 18$

$9 \times 3 = 27$

$9 \times 4 = 36$

$9 \times 5 = 45$

$9 \times 6 = 54$

$9 \times 7 = 63$

$9 \times 8 = 72$

$9 \times 9 = 81$

구구단 몇 단을 계산할까요(1~9)?

07. Lab: 연속적인 구구단 계산기

예제: calculator2.py

■ 실행 결과

$5 \times 1 = 5$

$5 \times 2 = 10$

$5 \times 3 = 15$

$5 \times 4 = 20$

$5 \times 5 = 25$

$5 \times 6 = 30$

$5 \times 7 = 35$

$5 \times 8 = 40$

$5 \times 9 = 45$

구구단 몇 단을 계산할까요(1~9)?

0

구구단 게임을 종료합니다.

07. Lab: 연속적인 구구단 계산기

예제: calculator2.py

■ 문제 해결

코드 4-20 calculator2.py

```
1 print("구구단 몇 단을 계산할까요(1~9)?")
2 x = 1
3 while (x is not 0):
4     x = int(input())
5     if x == 0: break
6     if not(1 <= x <= 9):
7         print("잘못 입력했습니다", "1부터 9 사이 숫자를 입력하세요.")
8         continue
9     else:
10        print("구구단 " + str(x) + "단을 계산합니다.")
11        for i in range(1,10):
12            print(str(x) + " x " + str(i) + " = " + str(x*i))
13        print("구구단 몇 단을 계산할까요(1~9)?")
14 print("구구단 게임을 종료합니다.")
```

07. Lab: 연속적인 구구단 계산기

예제: calculator2.py

■ 문제 해결 : [코드 4-20] 해석

- 1행 : 화면에 사용자가 숫자를 입력할 수 있도록 안내 메시지를 출력하는 코딩을 하였다.
- 2행 : while문을 시작하기 전에 $x = 1$ 이라는 코드를 넣고 시작하는데, 이것은 while문을 시작할 때 초기화하는 과정으로 이해하면 된다. while문이 무조건 한 번은 실행되어야 하므로 while문에 들어가는 조건 ($x \text{ is not } 0$)을 만족할 수 있도록 x 에 1을 할당하는 것이다.
- 3행에서부터 while 반복문을 실행한다. 사용자에게 입력값을 받아 그 값을 x 에 할당하면서 프로그램이 시작된다. x 의 입력값에 따라 3가지 조건의 프로그램이 실행된다.
- 첫 번째 조건인 x 에 0이 할당됐을 때, 프로그램은 종료된다. 5행의 코드에서 보듯이 break문을 호출하여 while문을 빠져나가 14행의 `print()` 함수를 통해 "구구단 게임을 종료합니다." 를 화면에 출력한다.
- 6~8행 : 두 번째 조건인 x 가 1에서 9 사이의 숫자가 아닌 경우이다. 조건문으로는 `if not(1 <= x <= 9):`으로 표현한다. 해당 조건문에 만족하면 화면에 입력이 잘못됐다고 출력된 후 `continue`문이 while문의 처음으로 이동시킨다.
- 9~13행 : 마지막으로 위 두 조건을 모두 만족하지 않았을 때, 즉 1에서 9 사이의 숫자가 제대로 입력되었을 때 `for`문을 이용하여 화면에 입력된 숫자의 구구단 결과가 출력된다.
- 이렇게 반복문 안에 반복문이 있는 구조를 중첩 반복문(nested loop)이라고 한다.

08

Lab: 평균 구하기

(학생 별 3개 과목 평균 구하기)

08. Lab: 평균 구하기

예제: average.py

■ 실습 내용 A~E 학생에 대한 3개 과목의 학생 별 평균 점수 구하기

- 이번 Lab에서는 지금까지 배운 반복문과 조건문을 토대로 연속적인 평균 구하기 프로그램을 만들어 보자.

학생	A	B	C	D	E
국어 점수	49	80	20	100	80
수학 점수	43	60	85	30	90
영어 점수	49	82	48	50	100

- 이 프로그램의 규칙은 다음과 같다.

- 이차원 리스트이므로 각 행을 호출하고 각 열에 있는 값을 더해 학생별 평균을 구한다.
- for문 2개를 사용한다.

08. Lab: 평균 구하기

예제: average.py

■ 문제 해결

[47.0, 74.0, 51.0, 60.0, 90.0]

■ 실행 결과

코드 4-21 average.py

```
1 kor_score = [49, 80, 20, 100, 80]
2 math_score = [43, 60, 85, 30, 90]
3 eng_score = [49, 82, 48, 50, 100]
4 midterm_score = [kor_score, math_score, eng_score]
5
6 student_score = [0, 0, 0, 0, 0]
7 i = 0
8 for subject in midterm_score:
9     for score in subject:
10         student_score[i] += score        # 학생마다 개별로 교과 점수를 저장
11         i += 1                          # 학생 인덱스 구분
12     i = 0                                # 과목이 바뀔 때 학생 인덱스 초기화
13 else:
14     a, b, c, d, e = student_score        # 학생별 점수를 언패킹
15     student_average = [a/3, b/3, c/3, d/3, e/3]
16     print(student_average)
```


■ 문제 해결

- 1~3행 : 과목별 점수의 리스트를 생성한다. 국어 점수는 `kor_score`, 수학 점수는 `math_score`, 영어 점수는 `eng_score`에 리스트화한다.
- 4행 : `midterm_score`에 과목별 점수인 `kor_score`, `math_score`, `eng_score`의 리스트를 생성한다.
- 6행 : 학생별로 평균을 구해야 하므로 각 값을 저장할 수 있는 공간인 `student_score` 리스트를 생성하고, 0으로 초기화한다.
- 7행~12행 : 2개의 `for`문을 돌려 `midterm_score`에 있는 과목별 점수를 추출한다. 여기서 `i`는 `i`번째 학생을 의미하는 변수로, 10행의 `student_score[i] += score` 코드를 통해 해당과목 점수를 기존 `student_score[i]`에 계속 추가한다.
- 13행~16행 : `else`문으로 반복문이 모두 끝나면 3으로 나누어 개인별 성적을 출력한다. 이때 `student_average = [a/3, b/3, c/3, d/3, e/3]`은 개인별 총합을 3으로 나누고 학생별 평균을 `student_average` 변수에 저장한다. 마지막으로 16행에서 학생별 평균을 출력할 수 있도록 코딩한다.

08. Lab: 평균 구하기

예제: average.py

■ 문제 해결

	A	B	C	D	E
국어 점수	49	80	20	100	80
수학 점수	43	60	85	30	90
영어 점수	49	82	48	50	100

student_score[0]

student_score[1]

student_score[2]

student_score[3]

student_score[4]

[student_score[i]]

■ 미션

- 학생별 평균 점수 외에 추가로 과목별 평균을 구하시오.
- 즉, 3개 과목에 대한 5명 학생의 평균을 구하시오.
- 조건- 현재의 for loop 구조를 유지하는 것으로 한다.
- 출력 사례:
 - average of each student= [47.0, 74.0, 51.0, 60.0, 90.0]
 - average of each subject= [65.8, 61.6, 65.8]

09

코드의 오류를 처리하는 방법
+ 모듈의 작성, 설치와 활용

09. 코드의 오류를 처리하는 방법

■ 버그와 디버그

- 버그(bug) : 프로그래밍에서의 오류
- 디버그(debug) : 오류를 수정하는 과정
- 디버깅(debugging) : 코드에서 오류를 만났을 때, 프로그램의 잘못을 찾아내고 고치는 것

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법 : 문법적 오류

- 문법적 오류는 코딩했을 때, 인터프리터가 해석을 못 해 코드 자체를 실행시키지 못하는 오류이다. 문법적 오류는 비교적 쉬운 유형의 오류이며, 대표적으로 들여쓰기 오류와 오타자로 인한 오류가 있다.

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법 : 문법적 오류

- 들여쓰기 오류(indentation error)

코드 4-22 indentation.py

```
1 x = 2
2 y = 5
3 print(x + y)
```

```
D:\workspace\Ch04>python indentation.py
File "indentation.py", line 2
    y = 5
    ^
IndentationError: unexpected indent
```

[들여쓰기 오류 메시지]

09. 코드의 오류를 처리하는 방법

여기서 잠깐! 들여쓰기 오류가 발생하는 이유

- <Space> 키나 <Tab> 키로 들여쓰기를 하면 들여쓰기 오류가 자주 발생한다. 초기 파이썬으로 코딩할 때는 들여쓰기를 4 스페이스(4 space)로 하는 사람과 <Tab> 키로 하는 사람이 각각 따로 있어 함께 코딩하면 많은 문제가 발생하였다. 하지만 최근에는 이러한 문제가 많이 줄었다

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법 : 문법적 오류

- 오타자로 인한 오류

코드 4-23 name.py

```
1 pront (x + y)      # Print가 아닌 Pront로 작성
2 korean = "ACE"
3 print(Korean)      # k는 소문자
```

```
D:\workspace\Ch04>python name.py
Traceback (most recent call last):
  File "name.py", line 1, in <module>
    pront (x + y) # Print가 아닌 Pront로 작성
NameError: name 'pront' is not defined
```

[오타자로 인한 오류 메시지]

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법 : 논리적 오류

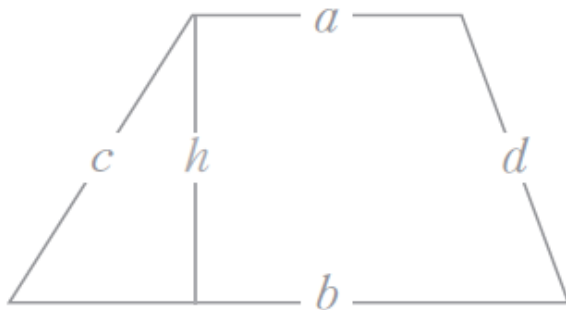
- 프로그램을 작성하다 보면 코드를 제대로 작성했다고 생각했음에도, 원하는 결과가 나오지 않는 경우가 종종 있다.
- 논리적 오류를 해결하는 방법은 다양한데, 당장 쉽게 사용할 수 있는 방법은 확인이 필요한 변수들에 `print()` 함수를 사용하여 값을 확인하는 것이다.

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- 사다리꼴 넓이를 구하는 프로그램을 작성하면서, 이 논리적 오류를 해결하는 연습을 해 보자.

$$A = \frac{a + b}{2} h \quad (a: \text{윗변}, b: \text{아랫변}, h: \text{높이})$$



[사다리꼴의 넓이 구하기]

09. 코드의 오류를 처리하는 방법

예제: trapezium_def.py
, 모듈로 활용됨

■ 오류의 종류와 해결 방법

- 사다리꼴의 넓이를 구하는 공식은 $\{(밑변 + 윗변) / 2\} * 높이$ 이다. 이 수식에 있는 각각의 과정을 하나씩 함수로 만들어 변환하는 연습을 할 것이다. 첫 번째는 두 변수를 더하는 `addition()` 함수, 두 번째는 두 값을 곱하는 `multiplication()` 함수, 세 번째는 2로 나누는 `divided_by_2()` 함수이다.

코드 4-24 trapezium_def.py

```
1 def addition(x, y):
2     return x + y
3
4 def multiplication(x, y):
5     return x * y
6
7 def divided_by_2(x):
8     return x / 2
```

* 본 예제 이후로는 별도의 자료 “**모듈의 설계, 설치와 활용**”을 먼저 학습한 후 진행할 것을 권한다.

모듈을 만들고 현재의 폴더에 설치한 것으로 가정하자.

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- ① 함수가 잘 작동하는지 확인하는 방법 : 파이썬 셸에서 실행하기

```
>>> import trapezium_def as ta          # trapezium_def 파일을 ta라는 이름으로 부름
>>> ta.addition(10, 5)
15
>>> ta.multiplication(10, 5)
50
>>> ta.divided_by_2(50)
25.0
```

09. 코드의 오류를 처리하는 방법

예제: trapezium_area.py

■ 오류의 종류와 해결 방법

- ② 함수가 잘 작동하는지 확인하는 방법 : 파일에서 체크할 수 있도록 if `_name_ == "_main_"`: 을 써 주는 구조로, if 문 안에 테스트할 코드를 작성하기

코드 4-25 trapezium_area.py

```
1 def addition(x, y):
2     return x + y
3
4 def multiplication(x, y):
5     return x * y
6
7 def divided_by_2(x):
8     return x / 2
9
10 # 파이썬 셸에서 호출할 경우 실행되지 않음
11 if __name__ == '__main__':
12     print(addition(10, 5))
13     print(multiplication(10, 5))
14     print(divided_by_2(50))
```

if문 때문에 아래 루틴은 이 프로그램을 메인 자격으로 수행할 때만 처리된다. 이 프로그램을 import 할 때는 메인 자격이 아니기 때문에 11~14는 수행되지 않는다.

09. 코드의 오류를 처리하는 방법

예제: trapezium_area.py

■ 오류의 종류와 해결 방법



```
15          # 12행의 실행 결과
50          # 13행의 실행 결과
25.0        # 14행의 실행 결과
```

➡ if `__name__ == '__main__'`을 넣는 가장 큰 이유는 해당 파일을 파이썬 셸에서 불러올 import 때 함수 안에 들어 있지 않은 코드들이 작동되지 않게 하기 위해서이다. 만약 해당 구문 없이 `print()` 함수로 구문을 작성한다면, 해당 파일을 파이썬 셸에서 호출할 때 그 구문이 화면에 출력되는 것을 확인할 수 있다.

따라서 어떤 것을 테스트하기 위해서는 반드시 if `__name__ == '__main__'` 안에 코드를 넣는 것이 좋다. => 파이썬이 어떤 프로그램을 main 자격으로 처음 수행할 때는 그 프로그램의 `__name__` 변수의 값을 `__main__`이라고 설정한다.

참고자료: [import 와 namespace](#)

09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- 실제 사다리꼴의 넓이 구하기 프로그램을 작성

코드 4-26 trapezium_area_final.py

```
1 def addition(x, y):
2     return x + y
3
4 def divided_by_2(x):
5     return x / 2
6
7 def main():
8     base_line = float(input("밑변의 길이는? "))
9     upper_edge = float(input("윗변의 길이는? "))
10    height = float(input("높이는? "))
11
12    print("넓이는:", divided_by_2(addition(base_line, upper_edge) * height))
13
14 if __name__ == '__main__':
15     main()
```


09. 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- 실제 사다리꼴의 넓이 구하기 프로그램을 작성



```
밑변의 길이는? 5
```

```
윗변의 길이는? 4
```

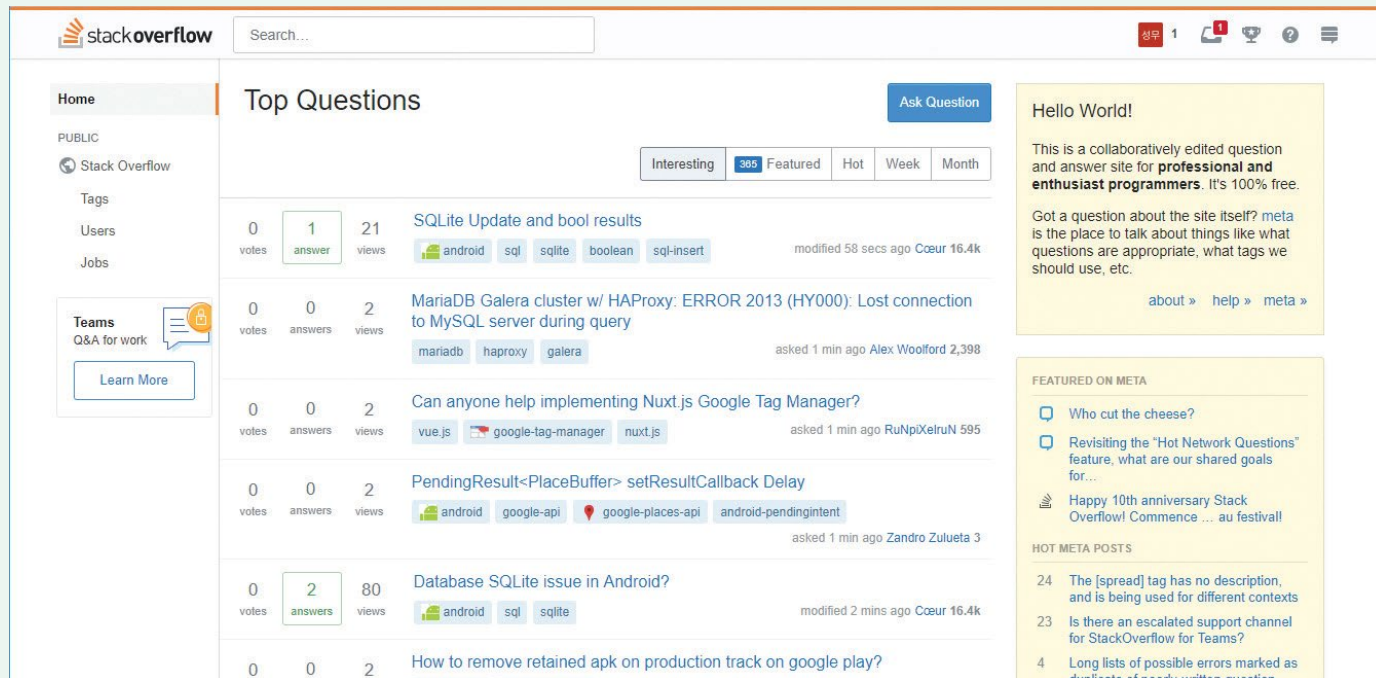
```
높이는? 3
```

```
넓이는: 13.5
```

09. 코드의 오류를 처리하는 방법

여기서 잠깐! 오류를 스스로 해결하기

- 많은 사람이 구글과 프로그래밍계의 지식인인 Stack Overflow를 통해 문제를 해결하고 있다. Stack Overflow는 전 세계 개발자들이 사용하는 Q&A 사이트이다.



The screenshot shows the Stack Overflow homepage. The top navigation bar includes the Stack Overflow logo, a search bar, and user avatars. The left sidebar contains links for Home, PUBLIC, Stack Overflow, Tags, Users, Jobs, Teams, and a Learn More button. The main content area is titled 'Top Questions' and features a list of questions with their respective votes, answers, and views. The questions are sorted by 'Interesting' (365), 'Featured', 'Hot', 'Week', and 'Month'. The first question is 'SQLite Update and bool results' with 0 votes, 1 answer, and 21 views. The second question is 'MariaDB Galera cluster w/ HAProxy: ERROR 2013 (HY000): Lost connection to MySQL server during query' with 0 votes, 0 answers, and 2 views. The third question is 'Can anyone help implementing Nuxt.js Google Tag Manager?' with 0 votes, 0 answers, and 2 views. The fourth question is 'PendingResult<PlaceBuffer> setResultCallback Delay' with 0 votes, 0 answers, and 2 views. The fifth question is 'Database SQLite issue in Android?' with 0 votes, 2 answers, and 80 views. The sixth question is 'How to remove retained apk on production track on google play?' with 0 votes, 0 answers, and 2 views. On the right side, there is a 'Hello World!' message and a 'FEATURED ON META' section with links to 'Who cut the cheese?', 'Revisiting the "Hot Network Questions" feature, what are our shared goals for...', 'Happy 10th anniversary Stack Overflow! Commence ... au festival!', and 'HOT META POSTS'.

[Stack Overflow(<https://stackoverflow.com>)]

Thank You !

■ 목적:

- `range()` 함수를 사용한 "`for ~ in range(N):`" 구문이 어떤 방식으로 반복 수행하는지 그 동작과정을 분석한다. 이 동작을 `iter()`, `next()` 함수를 사용하여 설명하고자 한다.

■ 배경 설명:

- 보통 `for` loop의 `in`에는 iterable objects가 쓰인다. = > "`for ~ in iterable_object:`"
- iterable_objects의 사례로는 `list`, `tuple`, `string`를 들 수 있는데 i 값에는 바로 이들 iterable_objects 원소 값이 순차적으로 바뀌어 모든 원소를 소진할 때까지 loop문을 수행한다.
- iterable object로 `range()` 함수가 사용될 수 있는데 `range()` 함수의 반환 값은 `<class 'range'>`로서 일반적으로 기대한 것처럼 이들이 `list`와 유사한 형태로 어떤 배열에 데이터를 모두 저장해 두고 있는 것은 아니다.
- "`for k in iterable_object`" 수행은 `in` 다음에 지정되는 object에 대해 iterator 생성하고 이후 loop를 반복할 때마다 `next(iterator)`를 수행하여 반환하는 원소를 `k`에 반환하는 것으로 이루어진다. 끝이면 `StopIteration exception`을 발생시켜 정지한다.

■ 결론:

- `range()` 함수는 지정한 회수를 loop를 수행하기 위해 사용된다.
- 그러나 이 함수는 호출 당시 그 회수 만큼의 원소를 가진 iterable object를 바로 반환하지 않는다.
- 이 동작을 원리적으로 설명한다면 다음과 같다. "for k in range(N):" loop에 대해;
- 1) `iterator = iter(iterable_objects)` # `iter(range())` 함수에 의해 iterator를 반환한다.
 - `iterable_objects`로는 원래 list, tuple, string를 들 수 있다.
 - `iter()` 함수는 "in ~" 에서 ~ 자리의 `iterable_objects`에게 자동으로 적용된다.
 - 아래의 `next(iterator)` 함수에서 이들의 실제 원소 값을 하나씩 반환할 수 있는 오브젝트(iterator)를 반환한다.
- 2) `element = next(iterator)` # 곧 이어 자동 호출되는 `next(iterator)`에 의해 loop를 수행하면서 새 원소를 k에 반환한다.
 - 자동으로 호출되는 `next()` 함수에 의해 호출될 때마다 iterable objects 안의 원소를 인덱스를 증가시켜 하나씩 반환한다.
 - 그런데 `iterable_objects`가 `range(N)`일 경우에는 `k=0, ..., N-1`을 순차적으로 반환한다고 해석된다.

■ 참고 링크:

- python iterable과 iterator의 의미: <https://bluese05.tistory.com/55>

Advanced Topic:

예제: `range_vs_list.py`

- “for ~ in iterable_object”에 `range()`와 `list` 자료 중 어떤 것의 수행 속도가 빠르지 실험 결과를 보이고, 그 결과를 분석하시오.