

데이터 과학을 위한 파이썬 프로그래밍

2판



Chapter 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

학습목표

- 조건문의 개념과 if-else문, if-elif-else문에 대해 알아본다.
- 반복문의 개념과 for문, while문에 대해 학습하고, 반복문의 제어에 대해 이해한다.
- 문자열 역순 출력 및 십진수를 이진수로 변환하는 실습을 진행한다.
- 코드의 오류를 처리하는 방법에 대해 학습한다.

01 조건문

1. 조건문의 개념

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

- 위의 성적으로 학점을 부여하는 프로그램을 만들 때, 먼저 학점을 정하는 기준을 만들어야 함.

- ❶ 점수에 따른 학점의 기준을 정한다. ex) 95점 이상 'A+', 60점 미만 'F'
- ❷ 기준을 바탕으로 첫 번째 점수를 판단한다. ex) 38점은 60점 미만이므로 'F'
- ❸ 다음 점수로 이동하면서 ❷를 반복한다. ex) 37점은 60점 미만이므로 'F'
- ❹ 더 이상 판단할 점수가 없을 때 프로그램을 종료한다

- 프로그램을 만들 때 고려사항

- 👉 어떤 기준으로 결정해야 하는가? → 조건의 설정
- 👉 언제까지 해야 하는가? → 반복의 설정

1. 조건문의 개념

- **조건문(conditional statement)**: 조건에 따라 특정 동작을 하도록 하는 프로그래밍
- 조건문을 구성하기 위해서는 조건을 나타내는 기준과 실행해야 할 명령이 필요함.
예) 스마트폰 잠금 해제 패턴이 5회 틀리면, 20초 동안 대기 상태로 만들어라

└─> 조건의 기준

└─> 실행해야 할 명령어
- 조건문은 반드시 조건의 참(True)과 거짓(False)으로 구분되어야 함.



그림 4-1 조건문의 예: 스마트폰 잠금 해제 패턴

2. if-else문

■ if-else문의 기본 문법

```
if : <조건>                # if를 입력하고 조건 삽입 후 ':' 입력
    <수행 명령 1-1>         # 들여쓰기 후 수행 명령 입력
    <수행 명령 1-2>         # 같은 조건에서 계속 수행할 명령일 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 명령일 경우 들여쓰기 유지
```

- ① if 뒤에는 참과 거짓을 판단할 수 있는 조건문이 와야 하고, 조건문이 끝나면 반드시 콜론(:)을 붙여야 한다.
- ② 들여쓰기(indentation)를 사용하여 조건이 참일 경우 수행할 명령을 작성한다.
- ③ if의 조건이 거짓일 경우 else문이 수행된다. else문은 생략해도 상관없다.

2. if-else문

[코드 4-1]

```
1 print("Tell me your age?")
2 myage = int(input())           # 나이를 입력받아 myage 변수에 할당
3 if myage < 30:                 # myage가 30 미만일 때
4     print("Welcome to the Club.")
5 else:                         # myage가 30 이상일 때
6     print("Oh! No. You are not accepted.")
```

[실행결과]

Tell me your age?	← 입력 대기
20	← 사용자 입력
Welcome to the Club	← 출력

3. 조건의 판단

3.1 비교 연산자 (또는 조건 연산자)

- 어떤 것이 큰지, 작은지, 같은지를 비교하는 것으로 그 결과는 참이나 거짓이 됨

표 4-1 비교 연산자

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

3. 조건의 판단

- 파이썬에서는 같음을 의미할 때 == 연산자를 사용하고, 할당의 의미를 표현할 때는 = 연산자를 사용

- 만약 다음과 같은 코드가 있다면 결과는 True일까, 아니면 False일까?

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

☞ 두 값이 정확하게 일치하므로 결과는 True

3. 조건의 판단

- `is`는 `==` 연산자처럼 두 변수가 같음을 비교하지만, `==` 연산자와 다르게 메모리의 주소를 비교함.

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

3. 조건의 판단

3.2 True와 False의 치환

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

- 위 코드를 실행하면 True가 출력됨

☞ 컴퓨터는 기본적으로 이진수만 처리할 수 있으므로 True는 1로, False는 0으로 처리

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

- 위 코드를 실행하면 True가 출력
- 하나하나 치환해보면 먼저, $3 > 5$ 는 False이고 결국 0으로 치환됨. 이것을 다시 치환하면 $(0) < 10$ 이 되고, 이 값은 당연히 참이므로 True가 반환되는 것임.

3. 조건의 판단

3.3 논리 연산자

표 4-2 논리 연산자

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

- and는 둘 다 참이어야 True
- or는 둘 중 하나만 참이어도 True
- not은 참이면 False이고 거짓이면 True

3. 조건의 판단

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

- `a == 8 and b == 4`의 경우, `a == 8`은 참, `b == 4`는 거짓이므로 `True and False`로 치환되고, 둘 중 하나라도 거짓이 있을 경우 `and` 구문은 `False`를 반환하므로 `False`가 출력됨
- `a > 7 or b > 7`의 경우, `a > 7`은 참, `b > 7`은 거짓이므로 `True or False`로 치환되고, 둘 중 하나라도 참이 있으므로 `True`가 출력됨
- `not (a > 7)`의 경우, `not True`로 치환되고, `True`는 `not`에 의해 `False`가 출력됨

4. if-elif-else문

- 중첩 if문을 간단히 표현하면 if-elif-else문임
- [표 4-3]과 같은 점수판이 있다고 가정

표 4-3 점수판

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

4. if-elif-else문

[코드 4-2]

```
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)
```

[실행결과]

Enter your score: 98

← 사용자 점수 입력

D

← 잘못된 값 출력

4. if-elif-else문

- 실제 코드로 만들어 실행하면 모든 값이 'D'나 'F'로 나옴. ➡ 코드가 한 줄씩 차례대로 실행되기 때문.

```
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)
```

그림 4-2 if문만을 이용한 학점 계산기에 98을 입력했을 때 결과 산출 방식

4. if-elif-else문

[코드 4-3]

```
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)
```

[실행결과]

```
Enter your score: 98          ← 사용자 점수 입력
A                             ← 올바른 값 출력
```

02

Lab: 어떤 종류의 학생인지 맞추기

1. 실습 내용

- 사용자가 태어난 연도를 입력하면 어떤 종류의 학생인지 맞추는 프로그램
- 프로그램은 매년 연도가 바뀔 때마다 변경해야 하는데, 여기서는 2020년을 기준으로 프로그램을 작성함.
- 이 프로그램을 작성하는 규칙
 - 나이는 $(2020 - \text{태어난 연도} + 1)$ 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

2. 실행 결과

[실행결과]

당신이 태어난 연도를 입력하세요.

1982

학생이 아닙니다.

← 입력 대기


← 자신이 태어난 연도 입력

← 어떤 종류의 학생인지 출력

3. 문제해결

[코드 4-4]

```
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.")
```

여기서  잠깐! 논리 연산자 없이 비교 연산자를 사용할 경우

코드에서 `age < 14 and age >= 8` 조건과 수학적식에서 사용하는 `8 <= age < 14` 조건은 서로 같을까? 전통적인 프로그램과 달리 파이썬에서는 수학적 조건도 잘 실행된다.

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

사실 이 코드는 좋은 코드가 아니다. 이 코드가 순서대로 실행되면 먼저 `1 <= 100`을 해석하니 True로 반환되고, True는 1과 같으므로 `1 < 10`도 True로 반환되어야 한다. 하지만 파이썬에서는 False가 나온다.

파이썬은 사람에게 친숙하게 프로그래밍을 하겠다는 철학을 가지고 있으므로 이러한 처리가 가능하도록 지원한다. 사람의 생각 그대로 그냥 조건을 작성하면 파이썬 인터프리터가 다 해결해준다. 문제는 다른 언어들이 지원하지 않는 경우가 있으므로, 이렇게 작성하는 것은 좋은 코드가 아니라는 걸 기억해야 한다. 물론 사람이 보기 쉬운 코드가 좋은 코드인지, 다른 프로그래머를 고려하는 것이 좋은 코드인지에 따라 의견이 달라질 수 있지만, 저자는 권장하지 않는다.

03 반복문

1. 반복문의 개념

- **반복문(loop):** 문장을 반복하도록 만드는 것으로, 정해진 동작을 반복적으로 수행할 때 사용하는 명령어
- **반복문의 구성:** 반복 시작 조건, 종료 조건, 수행 명령으로 구성되어 있으며, 들여쓰기와 블록(block)으로 구분
- 조건문에 if라는 키워드가 있듯이, 반복문은 for와 while이라는 명령 키워드를 사용

2. for문

- for문 예시

[코드 4-5]

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

[실행결과]

```
hello  
hello  
hello  
hello  
hello
```

2. for문

- 변수 자체를 출력할 수도 있음

[코드 4-6]

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

[실행결과]

```
1  
2  
3  
4  
5
```

2. for문

- range 키워드를 사용한 코드


[코드 4-7]

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

[실행결과]

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

2. for문

여기서  잠깐! 반복문에서 알아두면 좋은 상식

1. 반복문의 변수는 대부분 i, j, k로 지정한다. 이것은 수학에서 변수를 x, y, z로 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 대부분의 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.

2. for문

- 문자열도 리스트와 같은 연속적인 데이터이므로 각 문자를 변수 i에 할당하여 화면에 출력할 수 있음

[코드 4-8]

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

[실행결과]

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

2. for문

- 숫자를 화면에 출력하듯 문자열로 이루어진 리스트 값들도 사용할 수 있음

[코드 4-9]

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

[실행결과]

```
americano  
latte  
frappuccino
```


2. for문

- 1부터 9 까지 2씩 증가시키는 for문 확인

[코드 4-10]

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

[실행결과]

```
1
3
5
7
9
```

2. for문

- 10부터 2까지 1씩 감소시키는 반복문

[코드 4-11]

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

[실행결과]

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

3. while문

- 어떤 조건이 만족하는 동안 명령 블록을 수행하고, 해당 조건이 거짓일 경우 더이상 반복 명령문을 수행하지 않는 구문


[코드 4-12]

```
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
```

[실행결과]

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

3. while문

여기서  잠깐! for문과 while문의 사용

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 하는데 '이기면 종료하라.'라는 조건을 준다고 가정했을 때 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

그림 4-3 for문과 while문의 사용

4. 반복문의 제어

4.1 break문

- 반복문에 break문에 있으면 반복문을 강제로 종료 시킬 수 있음.

[코드 4-13]

```
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
```

[실행결과]

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

4. 반복문의 제어

4.2 continue문

- break문과 달리 특정 조건에서 남은 명령을 건너뛰고 다음 반복문을 수행.

[코드 4-14]

```
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
```

[실행결과]

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

4. 반복문의 제어

4.3 else문

- else문은 어떤 조건이 완전히 끝났을 때 한번 더 실행해주는 역할을 함.

[코드 4-15]

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

[실행결과]

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

4. 반복문의 제어

여기서  **잠깐!** 반복문을 제어하는 구문의 사용

개인적으로 앞에서 설명한 제어 구문들은 되도록 사용하지 않을 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

1. 실습 내용

- 반복문을 이용하여 구구단 계산기 만들기
- 구구단 계산기 프로그램은 사용자가 계산하고 싶은 구구단의 단수를 입력하면 프로그램이 구구단을 출력하는 매우 간단한 프로그램임.
- 이 프로그램을 작성하는 규칙
 - 프로그램이 시작되면 '구구단 몇 단을 계산할까?'가 출력된다.
 - 사용자는 계산하고 싶은 구구단 단수를 입력한다.
 - 프로그램은 '구구단 n단을 계산한다.'라는 메시지와 함께 구구단의 결과를 출력한다.

2. 실행 결과

[실행결과]

구구단 몇 단을 계산할까?

5

구구단 5단을 계산한다.

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

⋮

$$5 \times 8 = 40$$

$$5 \times 9 = 45$$

3. 문제 해결

[코드 4-16]

```
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)
```

05

조건문과 반복문 실습

1. 문자열 역순 출력

- **reverse_sentence 변수:** 입력된 문자열을 역순으로 출력하기 위한 변수

[코드 4-17]

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

[실행결과]

```
uoy evol I
```

1. 문자열 역순 출력

- 이 코드의 핵심은 reverse_sentence 변수
- 기존 sentence 변수에 있는 글자를 char라는 변수에 하나씩 저장한 후, 역순으로 reverse_sentence에 붙여넣는 구조
- 반복문의 순서

<u>Loop</u>	<u>reverse_sentence¹</u>	<u>reverse_sentence²</u>	<u>char</u>
0		I	I
1	I	I	
2	I	II	I
3	II	oI I	o
4	oI I	voI I	v
5	voI I	evol I	e
6	evol I	evol I	
7	evol I	y evol I	y
8	y evol I	oy evol I	o
9	oy evol I	uoy evol I	u

그림 4-4 reverse_sentence를 사용한 반복문의 순서

2. 십진수를 이진수로 변환

- 십진수 숫자를 2로 계속 나눈 후 그 나머지를 역순으로 취하면 이진수가 됨

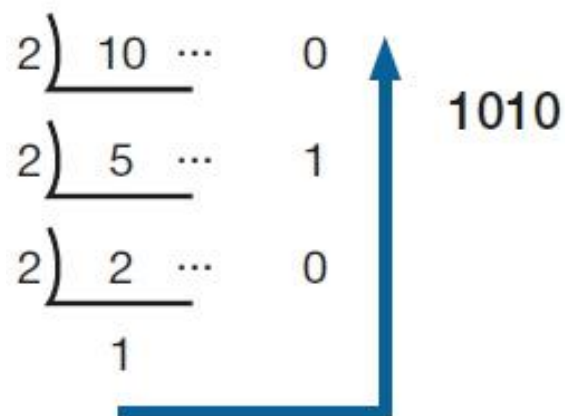


그림 4-5 십진수를 이진수로 변환하는 방법

2. 십진수를 이진수로 변환

[코드 4-18]

```
1 decimal = 10
2 result = ' '
3 while (decimal > 0):
4     remainder = decimal % 2
5     decimal = decimal // 2
6     result = str(remainder) + result
7 print(result)
```

[실행결과]

1010

2. 십진수를 이진수로 변환

■ [코드 4-18]에서 핵심 코드는 3개

- ① 4행의 `remainder = decimal % 2` 코드는 나머지를 구해 `remainder` 변수에 저장하는 것
- ② `decimal = decimal // 2` 코드는 현재의 십진수를 2로 나눈 몫을 다시 `decimal` 변수에 저장
- ③ 값의 역순을 `result` 변수에 저장하는 코드 `result = str(remainder) + result`

<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

그림 4-6 반복문이 진행될 때마다 진행되는 변수의 변화

06

Lab: 숫자 찾기 게임

1. 실습 내용

- **숫자 찾기 게임(guess number game):** 컴퓨터가 임의로 생성한 숫자를 사용자가 추측하는 숫자를 입력해 맞추는 간단한 프로그램
- 이 프로그램을 작성하는 규칙
 - 먼저 컴퓨터가 1에서 100까지 중 임의의 숫자를 생성한다.
 - 다음으로 사용자가 추측하는 숫자를 입력하면 컴퓨터가 생성한 임의의 숫자보다 큰지, 작은지를 계속 비교해준다.
 - 정답을 맞힐 때까지 반복하다가 맞혔을 때 '정답입니다. 입력한 숫자는 n입니다.'를 출력한다.

2. 실행 결과

[실행결과]

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.

- 이 게임의 가장 큰 특징은 반복문에 값이 변하는 변수가 들어가 그 값을 맞힐 때까지 계속 if 문으로 비교하는 것

3. 문제 해결

[코드 4-19]

```
01 import random                                # 난수 발생 함수 호출
02 guess_number = random.randint(1, 100)        # 1~100 사이 정수 난수 발생
03 print("숫자를 맞춰 보세요. (1 ~ 100)")
04 users_input = int(input())                    # 사용자 입력을 받음
05 while (users_input is not guess_number):      # 사용자 입력과 난수가 같은지 판단
06     if users_input > guess_number:            # 사용자 입력이 클 경우
07         print("숫자가 너무 큼니다.")
08     else:                                     # 사용자 입력이 작을 경우
09         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
```

07

Lab: 연속적인 구구단 계산기

1. 실습 내용

- **연속적인 구구단:** 사용자가 종료할 때까지 입력한 숫자에 대한 구구단 결과를 출력하는 게임
- 이 프로그램을 작성하는 규칙:
 - 프로그램이 시작되면 '구구단 몇 단을 계산할까요(1~9)?'가 출력된다.
 - 사용자는 계산하고 싶은 구구단 단수를 입력한다.
 - 프로그램은 '구구단 n단을 계산합니다.'라는 메시지와 함께 구구단의 결과를 출력한다.
 - 기존 예제와 다르게 이번에는 프로그램이 계속 실행되다가 종료 조건에 해당하는 숫자(여기에서는 0)를 입력하면 종료된다.

2. 실행 결과

[실행결과]

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

2. 실행 결과

[실행결과]

구구단 5단을 계산합니다.

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

$$5 \times 3 = 15$$

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

$$5 \times 6 = 30$$

$$5 \times 7 = 35$$

$$5 \times 8 = 40$$

$$5 \times 9 = 45$$

구구단 몇 단을 계산할까요(1~9)?

0

구구단 게임을 종료합니다.

3. 문제 해결

[코드 4-20]

```
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.")
```

08

Lab: 평균 구하기

1. 실습 내용

- **평균 구하기 프로그램:** 이차원 리스트에 있는 값들의 평균을 구하는 프로그램
- 학생들의 과목별 평균 점수가 있는 표를 활용하여 평균 구하기

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 프로그램을 작성하는 규칙

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

2. 실행 결과

[실행결과]

```
[47.0, 74.0, 51.0, 60.0, 90.0]
```

3. 문제 해결

[코드 4-21]

```
01 kor_score = [49, 80, 20, 100, 80]
02 math_score = [43, 60, 85, 30, 90]
03 eng_score = [49, 82, 48, 50, 100]
04 midterm_score = [kor_score, math_score, eng_score]
05
06 student_score = [0, 0, 0, 0, 0]
07 i = 0
08 for subject in midterm_score:
09     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)
```

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

student_score[0]

student_score[1]

student_score[2]

student_score[3]

student_score[4]

그림 4-7 student_score[i]

09

코드의 오류를 처리하는 방법

1. 버그와 디버그

- **버그(bug):** 'ValueError'나 'Invalid sentence'와 같은 오류들을 프로그래밍에서 일컫는 말
- **디버그(debug):** 오류를 수정하는 과정
- **디버깅(debugging):** 코드에서 오류를 만났을 때 잘못을 찾아내고 고치는 것

2.1 문법적 오류

- 코딩 과정에서 인터프리터가 해석을 못해 코드 자체를 실행시키지 못하는 오류
- 비교적 쉬운 유형의 오류이며, 대표적으로 들여쓰기 오류와 오타자로 인한 오류가 있음.

2. 오류의 종류와 해결 방법

2.1 문법적 오류

➤ 들여쓰기 오류(indentation error)

[코드 4-22]

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

- 2행의 `y = 5` 앞에 띄어쓰기가 되어 있는데, 파이썬은 이유 없는 띄어쓰기를 허용하지 않음
- 특히 if문, for문, while문 등을 작성하면서 들여쓰기를 실수할 때가 많은데 오류가 발생하면 파이썬에서는 'IndentationError'라는 오류 메시지를 출력함.

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


그림 4-8 들여쓰기 오류 메시지

2. 오류의 종류와 해결 방법

- 문법적 오류는 파이썬 파일을 실행시켰을 때 곧바로 문법적 오류임을 알려줌.
- 오류 메시지의 내용: 오류가 발생한 파일 경로와 줄(line) 번호를 출력하고 오류가 발생한 부분에 꺾쇠 표시(^)를 하고, 오류 종류와 함께 'unexpected indent'라고 나타냄.

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

그림 4-8 들여쓰기 오류 메시지

여기서  **잠깐!** 들여쓰기 오류가 발생하는 이유

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

2. 오류의 종류와 해결 방법

➤ 오타자로 인한 오류

[코드 4-23]

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

- 1행에서 명령어를 'print'가 아닌 'pront'라고 입력하였고, 3행에서 변수의 대소문자를 구분하지 않아 오타자로 인한 오류 발생.
- 이러한 오류가 발생하면 파이썬에서는 'NameError'라는 오류 메시지를 출력함.

```
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
```

그림 4-9 오타자로 인한 오류 메시지

2. 오류의 종류와 해결 방법

2.2 논리적 오류

- 사다리꼴 넓이 구하는 프로그램 작성하면서 논리적 오류 해결 연습하기

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

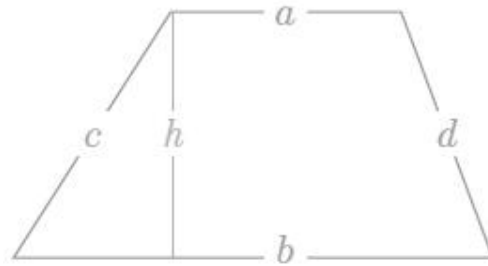


그림 4-10 사다리꼴의 넓이 구하기

- 각각의 과정을 3개의 함수로 만들어 변환하기
 - ① 두 변수를 더하는 addition() 함수
 - ② 두 값을 곱하는 multiplication() 함수
 - ③ 2로 나누는 divided_by_2() 함수
- [코드 4-24]처럼 3개의 함수를 작성하여 'trapezium_def.py'라는 파일에 저장

2. 오류의 종류와 해결 방법

[코드 4-24]

```
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
```

2. 오류의 종류와 해결 방법

- 함수가 잘 작동하는지 확인하는 방법:

① 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
```


2. 오류의 종류와 해결 방법

- 함수가 잘 작동하는지 확인하는 방법:

② if 문 안에 테스트할 코드를 작성하는 방식 (파일에서 체크할 수 있도록 if __name__ == '__main__':을 추가)

[코드 4-25]

```
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))
```

2. 오류의 종류와 해결 방법

[실행결과]

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

2. 오류의 종류와 해결 방법

- 테스트를 모두 마치고 실제 사다리꼴의 넓이 구하기 프로그램을 작성하기

[코드 4-26]

```
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()
```

2. 오류의 종류와 해결 방법

[실행결과]

밑변의 길이는? 5
윗변의 길이는? 4
높이는? 3
넓이는: 13.5

2. 오류의 종류와 해결 방법

여기서 잠깐! 오류를 스스로 해결하기

오류를 발견했을 때 어떻게 스스로 해결할 수 있을까? 사실 스스로 문제를 해결하는 것이야말로 가장 빠르게 프로그래밍 실력을 향상시키는 방법이다. 프로그래밍 세계에서는 모든 것이 구글(Google)로 통한다. 많은 사람이 구글과 프로그래밍계의 지식인인 Stack Overflow를 통해 문제를 해결하고 있다. Stack Overflow는 전 세계 개발자들이 사용하는 Q&A 사이트이다. 영어와 코드가 조금만 익숙하면 많은 도움을 받을 수 있다.

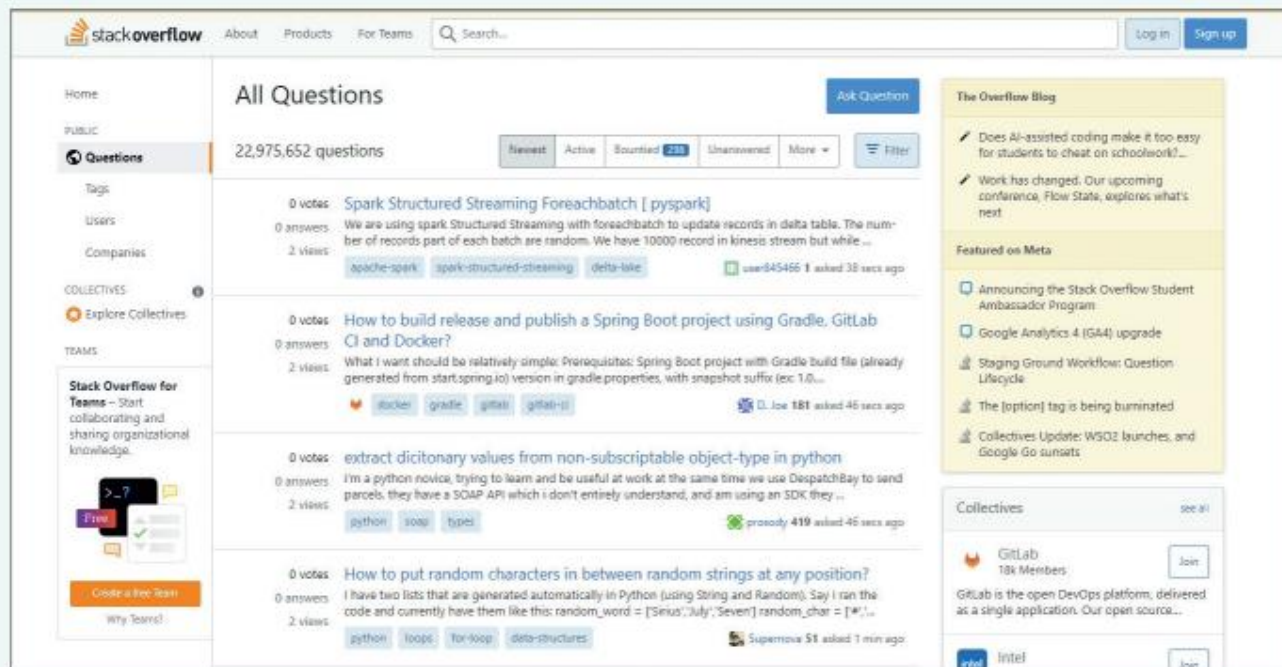


그림 4-11 Stack Overflow(<https://stackoverflow.com>)

프로그램을 만들다 보면 숏한 오류를 만난다. 그것을 해결하는 과정 하나하나가 공부이자 즐거움이라고 생각한다. 필자는 처음에 프로그래밍을 배웠을 때 문제 해결의 즐거움을 느끼며 서서히 이 세계로 빠져들었던 경험이 있다. 디버깅하는 과정을 통해 진짜 프로그래밍의 재미를 느껴보기 바란다.

Thank you!