# W4 제어구조 - 반복문

| | |
|---|---|
| content | 반복문 응용 |
| check | ☑ |
| date | @2023년 3월 1일 |
| 상태 | To edit |

## #1 반복문

### 반복문(loop)의 개념

- 정해진 동작을 반복적 수행할 때 사용하는 명령어
- 일상생활의 예시
  - 학생 100명 성적 산출
  - 쇼핑몰에서 상품 추천 → 사용자의 상품 구매 특징과 다른 상품 간의 특징 무한 비교

▼ **구성**

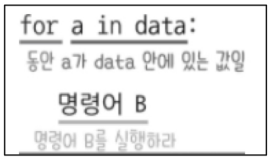
- 반복 시작 조건
- 반복 종료 조건
- 수행 명령
- 구분기준 : 들여쓰기 & 블록block
- 명령 키워드 : for, while

### for문

- 기본적인 반복문

▼ **기본 형식**

```
for a in data:
명령어 B
```




⇒ ‘a가 data안에 있는 값일 동안 반복해서 명령어 b 실행하라’

- 반복 범위 지정하여 수행

### ▼ 범위 지정 방법

#### ▼ 1. 리스트

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

- in 다음의 리스트의 각각의 값을 하나씩 가져와 looper 변수에 할당
- 한 번 할당할 때마다 다음 줄의 명령 print('hello') 실행
- [1,2,3,4,5]의 모든 값 한 번씩 수행 → 총 다섯 번의 반복 발생
- 'hello' 다섯 번 출력

```
hello
hello
hello
hello
hello
```

+) 변수 자체 출력

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

```
1
2
3
4
5
```

- 리스트의 각각 값이 하나씩 looper 변수에 할당 → 그 값이 출력됨
- 하지만 100번 반복해야 할 경우, 이 방법은 매우 비효율적!

#### ▼ 2. range

```
for looper in range(100):
    print('hello')
```


실행 결과

- looper 변수에 0~99까지 총 100개의 값 할당
- range 기본 구조

> **for 변수 in range(시작 번호, 마지막 번호, 증가값)**

- 리스트의 인덱싱 문법과 동일
  - 0부터 시작!
- 마지막 번호에 입력한 숫자 - 1까지 리스트 생성
  - ex. range(0,5) → [0,1,2,3,4]
- 시작 번호 & 증가값 → 생략 가능
  - 이경우, 시작 번호 = 0, 증가값 = 1

### ▼ 2.1 range 활용

- 2 칸씩 증가

```
for i in range(1,10,2):
    print(i)
```

```
1
3
5
7
9
```

- 1 칸씩 감소

```
for i in range(10,1,-1):
    print(i)
```

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

### ▼ 3. 문자열

- 단순 문자열

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

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

- 문자열로 이루어진 리스트

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

```
americano
latte
frappuccino
```

**알아두면 좋은 것**

1. 반복문 변수 대부분 i,j,k로 지정
2. 반복문 대부분 0부터 반복 시작
3. 반복문 잘못 작성시, 무한 루프 오류 발생

▼ **발생가능한 에러**

- 콜론(:)을 입력하지 않았을 때
  - [형식의 오류] SyntaxError : invalid syntax
- 들여쓰기 잘못됐을 때
  - 반복문 내부, 4칸 들여쓰기 하지 않았을 때
  - [들여쓰기 오류] IndentationError : expected an indented block
- 변수관련

```
for b in range(1,5):
    print(k, 'hello')
```

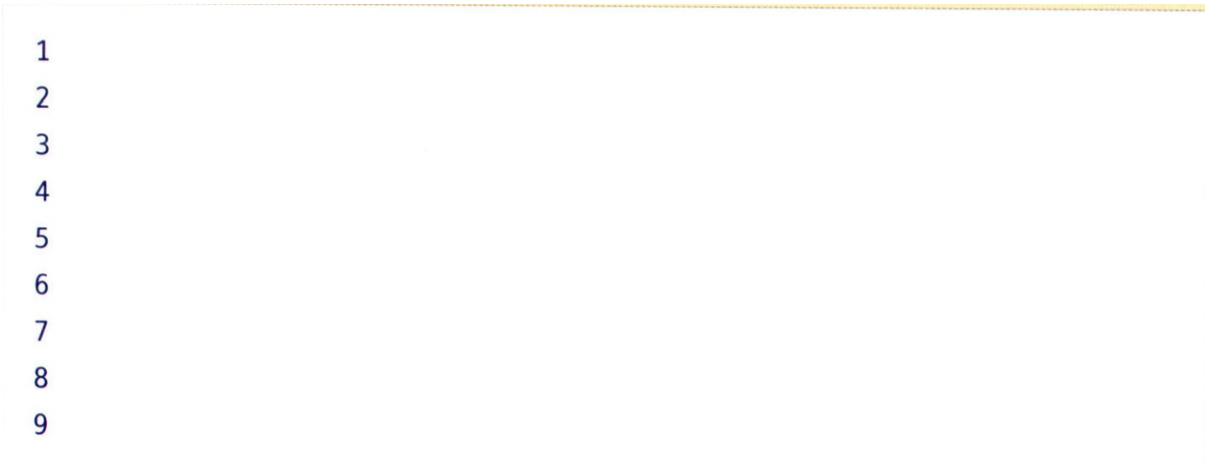
  - [변수 오류] NameError : name ‘k’ is not defined
  - k → b로 수정하기!

## while문

- 특정 조건이 만족하는 동안 명령 블록 수행 → 제어구조
- 그 조건이 거짓일 경우 더이상 반복 명령블록 수행하지 않음
- if + 반복문
- **기본 문법**

> while 판단 조건: #판단 조건이 True인 동안
>
> 반복 실행할 내용 #이부분을 반복한다

```
i = 1               # i 변수에 1 할당
while i < 10:       # i가 10 미만인지 판단
    print(i)        # 조건을 만족할 때 i 출력, i에 1을 더함
    i += 1          # i가 10이 되면 반복 종료
```




▼ +) for vs. while

- 서로 변환가능, 유사함
- **쓰임새의 차이 존재**
  - for : 반복 횟수 명확히 앎 & 반복 횟수 불변 → **[반복할 구간]**
    - ex. 학생 성적 채점 프로그램 → 총 학생 수 앎
  - while : 반복 횟수 불명확 & 특정 조건 만족할 때 프로그램 종료 가능 → **[반복할 조건]**
    - ex. 가위바위보할 때 이기면 종료 → 시점 정확히 알 수 없음 ⇒ while문

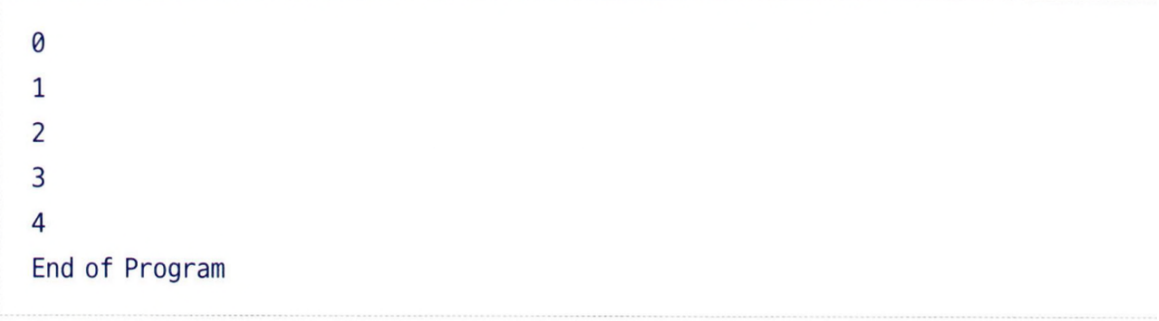
## 반복문의 제어

- 필요에 따라 제어 가능
- 중간에 반복 종료 or 실행 중 반복문 건너뛰기

▼ **break문**

- 논리적으로 강제 종료 가능
- ex. 'i == 5'가 참일 때 종료시키기

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



▼ **continue문**

- 특정 조건에서 남은 명령 스킵 → 다음 반복문 명령 수행
- ex. 'i == 5'일 경우, 다음 명령 건너뛰기

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

```
0
1
2
3
4
6
7
```

▼ **else문**

- 특정 조건 완전히 끝났을 때 한번 더 실행하는 역할
- break 등으로 코드 종료 후 else문 수행되지 않음 → 반복문 완벽히 수행됐는지 확인 용도

```
for i in range(10):
    print(i)
else:
    print("End of Program")
```

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

⇒ continue문, break문, else문 모두 while문에서도 사용가능

## 실습; 구구단 계산기

▼ **실습 내용**

- 반복문 활용
- 사용자 계산하고 싶은 구구단의 단수 입력 → 구구단 출력
- for문, input 활용

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

▼ **실행 결과**

```
구구단 몇 단을 계산할까?
5
구구단 5단을 계산한다.
5 × 1 = 5
5 × 2 = 10
  ⋮
5 × 8 = 40
5 × 9 = 45
```

▼ **문제 해결**

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

- #1 : "구구단 몇 단을 계산할까?" 문구 출력
- #2 : 사용자 입력 값 → user_input 할당
- #3 : 변수 이용 → 구구단 계산 문구 출력
- #4 : user_input이 문자열이므로 정수형으로 변환 → int_input 할당
- #5 ~ 7 : 입력된 숫자와 1~9까지 곱 반복문으로 표현
  - range 활용 → 1 ~ 9 숫자 i 할당
  - 사용자 입력값과 곱하여 결과를 result에 할당

# #2 조건문과 반복문 실습

## 문자열 역순 출력

- reverse_sentence.py

```
uoy evol I
```

: 'I love you' → 'uoy evol I'

```
sentence = "I love you"
reverse_sentence = ' '
for char in sentence:
    reverse_sentence = char + reverse_sentence
print(reverse_sentence)
```

- sentence의 글자를 char 에 하나씩 저장 → 역순으로 reverse_sentence에 붙여넣기

▼ **반복문 순서**

| Loop | reverse_sentence[1] | reverse_sentence[2] | char |
|---|---|---|---|
| 0 | | I | I |
| 1 | I | I | |
| 2 | I | I I | I |
| 3 | I I | ol I | o |
| 4 | ol I | vol I | v |
| 5 | vol 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

- reverse_sentence2 → 더해지는 reverse_sentence(우)
- reverse_sentence1 → 값 저장되는 reverse_sentence(좌)

## 십진수를 이진수로 변환

- 원리 : 십진수 숫자 → 2로 계속 나누기 → 그 나머지 역순 취하기

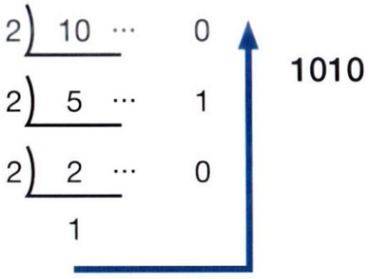



그림 4-5 십진수를 이진수로 변환하는 방법

- demical.py

```
#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)
```

```
1010
```

▼ **코드 설명**

- while 문 사용 → decimal이 어떤 값이든 'decimal > 0'이 참일 때가지 계속 실행
- #4 : remainder에 나머지 저장
- #5 : 현재 십진수를 2로 나눈 몫 → 다시 decimal에 저장
- #6 : result에 값의 역순 저장

▼ **변수 변화 세부 과정**

| Loop | decimal[1] | remainder | decimal[2] | result[1] | result[2] |
|---|---|---|---|---|---|
| 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

## 실습 ; 숫자 찾기 게임

### ▼ 실습 내용

- 반복문 조건문 활용
- 숫자 찾기 게임 : 컴퓨터가 임의로 생성한 숫자를 사용자가 추측하기
- 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.

- 맞힐 때까지 if문으로 비교하며 반복 ( =가변적 반복문)
  - 변수의 값에 따라 수행 여부 결정

### ▼ 실행 결과

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

### ▼ 문제 해결

```
#1
import random     # 난수 발생 함수 호출

#2
guess_number = random.randint(1, 100)              # 1~100 사이 정수 난수 발생
#3
print("숫자를 맞춰보세요. (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
        print("숫자가 너무 큽니다.")
    else:                                           # 사용자 입력이 작을 경우
        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
```

- #1 : random 모듈 사용 → 임의 수 생성
  - 모듈 : 다양한 함수 묶음
- #2 : random 안의 randint() 함수 사용 → 1,100 숫자 중 임의의 숫자 생성 → guess_number에 저장
- #3 : 사용자가 숫자 입력하도록 안내 문구 출력
- #4 : users_input에 정수형으로 변환하여 저장
- #5 ~ 10 : 사용자 입력값과 guess_number이 다를 동안 while 문 반복 수행

  ⇒ ‘users_input is not guess_number’ 충족할 동안
- #11 ~ 12 : 같아지면 반복문 탈출하여 안내 문구 출력

## 실습 ; 연속적인 구구단 계산기

▼ **실습 내용**

- 반복문 조건문 활용
- 중첩 반복문 이해
- 연속적인 구구단 : 사용자가 종료할 때 까지 입력한 숫자에 대한 구구단 결과 출력
- 0 입력 시, 프로그램 종료
- 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.**

- while문 안에 for문 사용

▼ 실행 결과

```
구구단 몇 단을 계산할까요(1~9)?
3
구구단 3단을 계산합니다.
3 × 1 = 3
3 × 2 = 6
3 × 3 = 9
3 × 4 = 12
```

```
3 × 5 = 15
3 × 6 = 18
3 × 7 = 21
3 × 8 = 24
3 × 9 = 27
구구단 몇 단을 계산할까요(1~9)?
5
구구단 5단을 계산합니다.
5 × 1 = 5
5 × 2 = 10
5 × 3 = 15
5 × 4 = 20
5 × 5 = 25
5 × 6 = 30
5 × 7 = 35
5 × 8 = 40
5 × 9 = 45
구구단 몇 단을 계산할까요(1~9)?
0
구구단 게임을 종료합니다.
```

▼ **문제 해결**

```
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.")
```

- #2 : while문 시작할 때 초기화 / ‘x is not 0’을 만족하기 위해
- #3 : while 반복문 시작
- #4 : 사용자에게 값 입력받아 x 할당
- #5 : 0 할당될 때 프로그램 종료 → #14행 실행
- #6 ~8 : x가 1 ~9가 아닌 경우 → 안내 문구 출력 → #4행으로 이동
- #9 ~ 13 : x가 1 ~ 9일 때 for문으로 구구단 결과 출력
- 중첩 반복문(nested loop)!