5장
여러 번 반복하는 일을 하자

이장에서 배울 것들

- 반복문의 필요성을 이해해 보아요.
- while 문을 사용하여 조건으로 반복하는 방법을 학습해 보아요.
- while 문과 반복 블록의 사용법을 익혀 보아요.
- for 문을 사용하여 정해진 횟수만큼 반복하는 방법을 학습해 보아요.
- 반복문을 사용하여 다양한 문제를 해결해 보아요.
- 이제 많은 횟수의 동작을 해야 하는 프로그램도 쉽게 작성할 수 있어요.
- 오랫동안 동작하며 사용자와 상호작용하는 프로그램을 작성할 수 있어요.

5. 1 왜 반복이 중요한가

- 반복은 어떤 단계를 반복하게 하는 것으로 반복 구조를 사용하면 프로그램이 간단하고 빠르게 된다.
- 예를 들어서 동일한 작업을 반복하기 위하여 똑같은 문장을 복사하여 붙여넣기 하는 것보다 반복 구조를 사용하는 편이 프로그램을 간결하게 만든다.
- 또 프로그래밍에 필요한 시간도 단축할 수 있다.

5. 1 왜 반복이 중요한가

- 하나의 예로 화면에 회사에 중요한 손님이 오셔서 대형 전광판에 다음과 같이 환영인사를 5번 출력한다고 하자.

```
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
```

- 반복 구조를 사용하지 않는다면 다음과 같이 동일한 문장을 Ctrl+C를 사용하여 복사한 후에 Ctrl+V로 붙여넣기 하여야 한다.

```
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
```

5. 1 왜 반복이 중요한가

- 물론 반복 횟수가 몇 번 안 되는 경우에는 위와 같이 "복사해서 붙여넣기" 하여도 된다.
- 하지만 100번 반복해야 한다면 어떻게 할 것인가?
- 이런 경우에 for 문을 사용한다면 다음과 같이 간단하게 작성할 수 있다.

```
for i in range(100):
    print("파이썬 주식회사의 방문을 환영합니다!")
```

- 이와 같이 반복문은 인간이 **반복적으로 수행하는 번거로운 작업을 손쉽게 해결**해 줄 수 있으므로 매우 중요한 제어문으로 널리 사용되고 있다.

도전문제 5.1

for - in range(): 를 이용하여 '####################'를 5줄에 걸쳐 출력하여라.

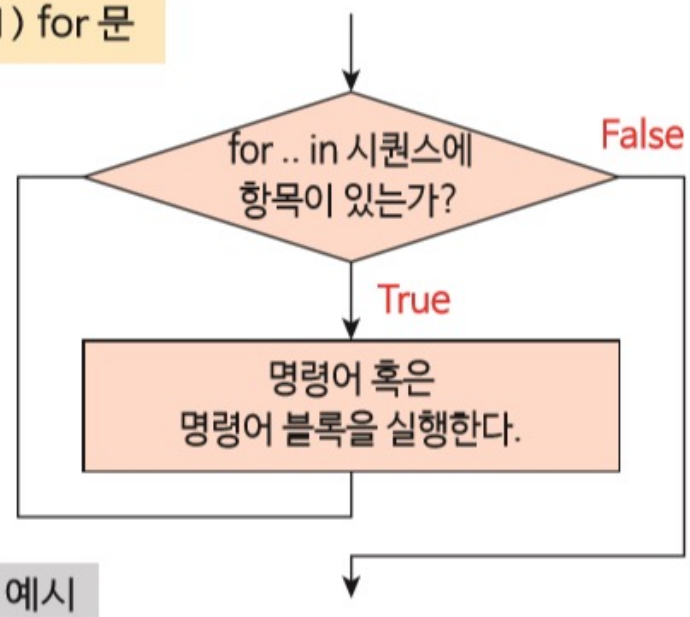
```
####################
####################
####################
####################
####################
```

5. 2 반복의 종류

- 파이썬에서는 2 가지 종류의 반복이 있다.
  1. 횟수 제어 반복(**for 문**): 보통 횟수를 정해 놓고 반복한다.
  2. 조건 제어 반복(**while 문**): 특정한 조건이 만족되면 계속 반복한다.
- 횟수 제어 반복은 반복을 시작하기 전에 반복의 횟수를 미리 아는 경우에 사용한다.
- 예를 들어서 "환영합니다" 문장을 5번 반복하여 출력한다면 for 문을 사용하면 자연스럽다.
- 파이썬에서는 항목들을 모아 놓은 시퀀스라는 객체가 있고 여기에서 항목을 하나씩 가져와서 반복 할 때도 **for** 문이 적합하다. 시퀀스에 항목이 더 이상 없으면 반복이 종료된다.

5. 2 반복의 종류

1) for 문



예시

```
for i in range(5):
        print('환영합니다')
```

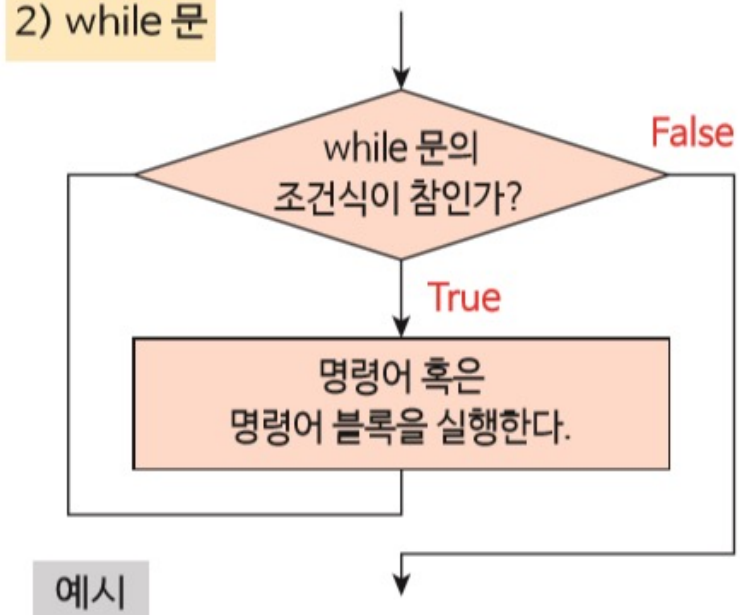
0, 1, 2, 3, 4 시퀀스를 생성함

- 왼쪽의 그림에서 range(5)라는 함수는 0, 1, 2, 3, 4의 숫자 시퀀스를 자동으로 생성하는 함수이다.
- 따라서 이 문장은 0에서 4까지의 숫자를 반환하는 작업을 끝낼 때까지 반복 실행된다.

5. 2 반복의 종류

- 반면 오른쪽의 while문은 i < 5의 조건이 참일 경우 환영합니다를 출력하고 i 값을 1 증가시킨다.
- 최초의 i 값이 0이므로 이 반복문 역시 5회 반복 수행하게 된다.

2) while 문



예시

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

잠깐 – 루프loop

프로그래밍에서 반복은 흔히 올가미 같은 동그란 고리를 의미하는 **루프**loop라고 한다. 왜냐하면 프로그램이 반복할 때 이전 단계로 되돌아가는데 이것이 동그라미를 그리는 것처럼 보이기 때문이다. 이 고리를 빠져나오지 않고 계속해서 반복을 하는 경우를 **무한루프**infinite loop라고 한다.

도전문제 5.2

`for t in range(10):` 를 이용하여 블록 내부에서 `t` 값을 출력해 보자.

5. 3 횟수를 정해 놓고 반복시키자

- 파이썬에서 횟수 제어 반복은 for 루프를 이용하는 것이 가장 간편하다. for 루프는 반복할 횟수를 정해 두고, 이 횟수가 만족될 때까지 같은 동작을 반복하도록 하는 일을 쉽게 표현할 수 있다.

for 반복문을 사용한 반복 실행의 일반적인 흐름

5. 3 횟수를 정해 놓고 반복시키자

- for 루프를 사용할 때는 다음과 같이 리스트를 이용하여 사용하는 방식을 먼저 연습해 보자. 아래와 같은 코드로 반복을 할 수 있다. 여기서 [...]은 **리스트**list이다.
- 이것은 7장에서 더욱 자세히 다룰 것인데, 여러 개의 값들을 담을 수 있는 장바구니와 같은 개념으로 이해하면 된다. 리스트 [1, 2, 3, 4, 5] 안에는 정수 1, 2, 3, 4, 5가 담겨있다고 생각하면 된다.

```
for i in [1, 2, 3, 4, 5]:          #  반복문의 끝에 :이 있어야 함
    print("방문을 환영합니다!")      # if 문과 마찬가지로 들여쓰기를 하여야 함
```

5. 3 횟수를 정해 놓고 반복시키자

- 반복해서 실행될 문장은 반드시 들여쓰기를 하여야 한다.
- for 문에 담긴 **블록**block이라 생각하면 된다. 위의 코드를 실행하면 다음과 같은 출력을 얻을 수 있다.

```
방문을 환영합니다!
방문을 환영합니다!
방문을 환영합니다!
방문을 환영합니다!
방문을 환영합니다!
```

- 첫 번째 반복에서 변수 i의 값은 리스트의 첫 번째 숫자인 1이 되고 print(...) 문장이 실행된다.
- 두 번째 반복에서 변수 i의 값은 리스트의 두 번째 숫자인 2가 되고 print(...) 문장이 실행된다.
- 세 번째 반복에서 변수 i의 값은 리스트의 세 번째 숫자인 3이 되고 print(...) 문장이 실행된다.
- 네 번째 반복에서 변수 i의 값은 리스트의 네 번째 숫자인 4가 되고 print(...) 문장이 실행된다.
- 다섯 번째 반복에서 변수 i의 값은 리스트의 다섯 번째 숫자인 5가 되고 print(...) 문장이 실행된다.

5. 4 for – in 다음에는 리스트나 문자열도 올 수 있다

- 앞 페이지에서는 변수 i의 값을 전혀 이용하지 않았다! 이번에는 반복하면서 print() 함수를 이용해서 변수 i의 값을 출력하여 보자.

```
for i in [1, 2, 3, 4, 5]:      # in 뒤에 리스트를 넣고 끝에 :을 넣자
    print("i =", i)            # i 값을 출력해보자
```

- 위의 코드를 실행하면 다음과 같이 출력된다.

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

5. 4 for – in 다음에는 리스트나 문자열도 올 수 있다

- 이 문장은 다음과 같은 문자열에 대해서도 적용할 수 있다.

```
for i in "Hello":        # 끝에 :이 있어야 함
    print("i =", i)      # i 값을 출력해보자
```

- 위의 코드를 실행하면 다음과 같이 출력된다.

```
i = H
i = e
i = l
i = l
i = o
```

5. 4 for – in 다음에는 리스트나 문자열도 올 수 있다

- 이번에는 이러한 출력 방식을 조금더 개선해서 다음과 같이 구구단 9단 중의 일부를 for 루프를 이용하여 출력하여 보자.

```
for i in [1, 2, 3, 4, 5]:          # 1에서 5까지의 리스트
    print("9 *", i, "=", 9 * i )  # 9*i 값을 출력해보자
```

- 위의 코드를 실행하면 다음과 같이 출력된다.

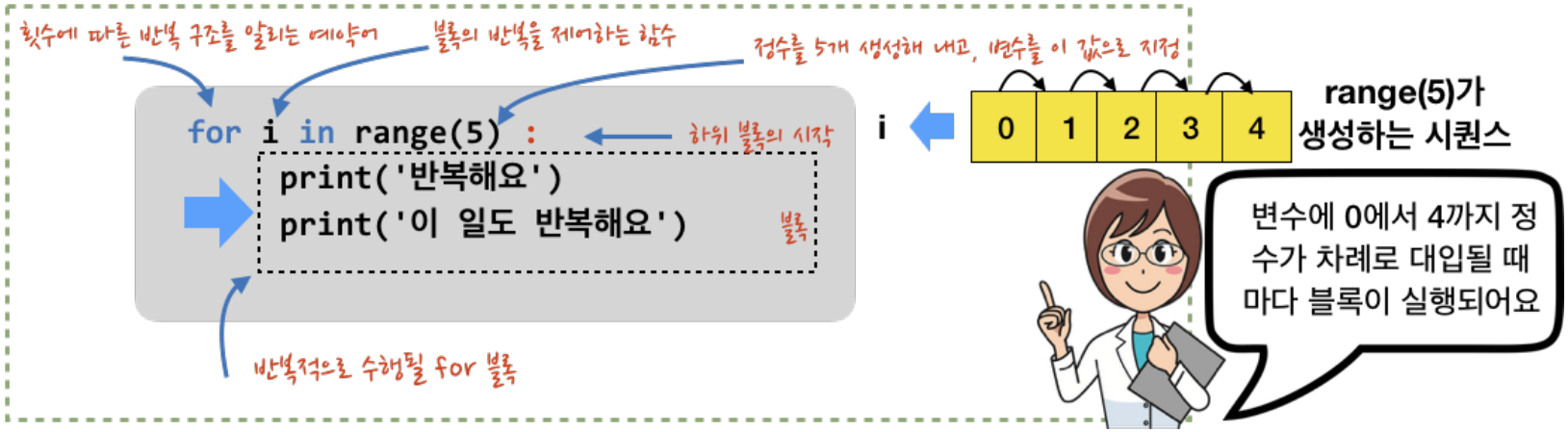
```
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
```

도전문제 5.3

`bts = ['V', 'J-Hope', 'RM', 'Jungkook', 'Jin', 'Jimin', 'Suga']` 라는 리스트를 선언한 후 이 리스트 내부의 모든 항목을 출력하는 코드를 작성하여라.

5. 5 for 문과 찰떡 궁합인 range() 함수

- 앞에서는 리스트에 정수들을 저장해두고 하나씩 꺼내서 반복하였다.
- 하지만 반복 횟수가 1000번이라면 이 방법이 불가능하다.
- 반복해야 하는 횟수가 큰 경우를 대비하여 range() 함수를 사용하는 방법이 준비되어 있다.
- range() 함수로 반복 횟수를 전달하면 range() 함수가 자동으로 정수들을 생성해준다.



5. 5 for 문과 찰떡 궁합인 range() 함수

- 예를 들어서 '파이썬 주식회사의'와 '방문을 환영합니다!'를 3번 출력하는 문장을 for 문으로 작성하면 다음과 같다. (2)는 반복 수행되는 블록임!

```
for i in range(3):                  # (1)
    print('파이썬 주식회사의')       # (2)
    print('방문을 환영합니다!')      # (2)
```

```
파이썬 주식회사의
방문을 환영합니다!
파이썬 주식회사의
방문을 환영합니다!
파이썬 주식회사의
방문을 환영합니다!
```

5. 5 for 문과 찰떡 궁합인 range() 함수

1) range(3) 함수는 0, 1, 2까지의 **숫자열**sequence을 반환한다. 반복할 때마다 변수 i에 이 값들을 대입하면서 문장을 반복한다. 즉 첫 번째 반복에서는 i는 0이고 되고 두 번째 반복에서는 1이 된다. 마지막 반복에서 i는 2가 된다.

2) 하나 이상의 반복되는 문장이 올 수 있는데, 반복되는 문장은 if 문과 같이 **반드시 동일한 간격의 들여쓰기를 해야 한다**. 이 때 들여쓰기가 있는 문장들만이 반복된다.

파이썬 쉘에서 range() 함수에 list() 함수를 적용시키면 range() 함수가 생성하는 정수들을 볼 수 있다. 이와 같이 range() 함수는 연속적인 값들을 생성하는 일을 한다.

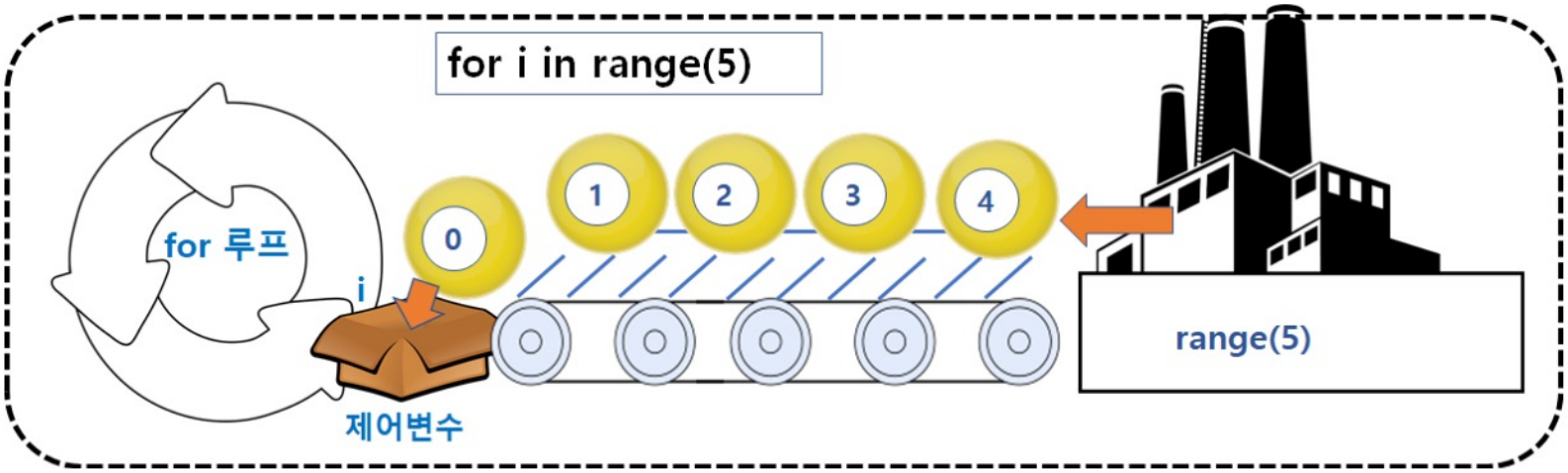
```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

잠깐 – range() 함수가 반환하는 값은 range 형이다

파이썬의 range() 함수는 range 형의 자료형을 반환한다. range 형의 자료형은 호출이 발생할 때마다 매번 연속된 값을 생성하여 반환하는 특별한 일을 한다. 따라서 주로 for 문과 함께 사용된다.

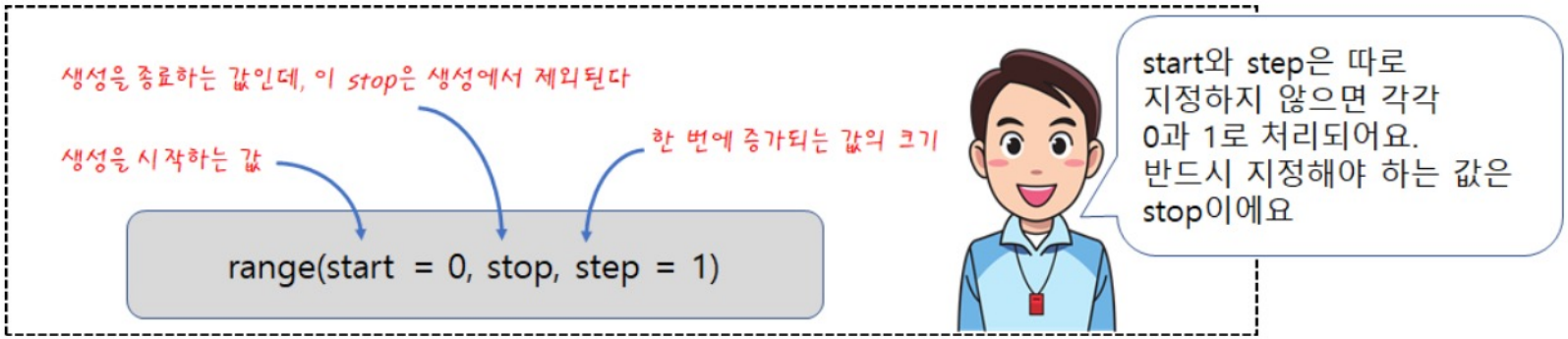
5. 6 range() 함수는 숫자를 생산하는 공장이다.

- range() 함수는 숫자들을 생산하는 공장으로 생각하면 된다.
- 그림과 같이 range(5)은 5개의 정수를 생성한다.
- 0, 1, 2, 3, 4가 바로 그것이다. 그리고 for 루프는 이 시퀀스의 갯수만큼 반복 수행한다.



5. 6 range() 함수는 숫자를 생산하는 공장이다.

- range() 함수는 여러개의 인자를 이용하여 좀 더 다양한 정수 시퀀스를 생성할 수 있으며, 일반적인 형식은 다음과 같다.
- range(start, stop, step)이라고 호출하면 start에서 시작하여 (stop-1)까지 step 간격으로 정수들이 생성된다.
- 0, 1, 2, 3, 4가 바로 그것이다. 그리고 for 루프는 이 시퀀스의 갯 수만큼 반복 수행한다.
- 여기서 start와 step가 생략될 수 있는데 이 경우 start는 0으로 간주되고 step은 1로 간주된다.
- 하지만 **stop 값은 반드시 지정해야만 루프가 수행된다.**



5. 6 range() 함수는 숫자를 생산하는 공장이다.

- 예를 들어서 range(0, 5, 1)이라고 하면 0, 1, 2, 3, 4까지의 정수가 반환된다. range(5)라고 하면 start와 step은 생략된 것으로 range(0, 5, 1)와 같다.
- 만약 0이 아닌 1부터 시작하여서 5까지 반복하고 싶다면 어떻게 하면 될까? range(1, 6, 1)을 사용하면 될 것이다. 반복하면서 변수 i의 값을 출력하여 보자.

```
for i in range(1, 6, 1):
    print(i, end = " ")      # end = " "로 지정하면 줄바꿈을 하지 않고 공백으로 나열한다.
```

```
1 2 3 4 5
```

- 만약 10부터 시작하여서 1까지 1씩 감소하며 반복하고 싶다면 어떻게 하면 될까? range(10, 0, -1)을 사용하면 될 것이다. 반복하면서 변수 i의 값을 출력하여 보자.

```
for i in range(10, 0, -1):
    print(i, end = " ")
```

```
10 9 8 7 6 5 4 3 2 1
```

잠깐 – range(n)은 range(0, n, 1)과 같다

range(10) 함수를 사용할 때 가장 혼동하는 부분이 1에서 10까지의 정수가 생성된다고 생각하는 것이다. 반복 횟수로 생각하면 10번 반복은 맞다. 하지만 생성되는 정수는 0부터 9까지이다. 이것은 컴퓨팅의 오랜 논쟁거리였다. 지금은 0부터 시작하는 것이 대세가 되었다. 아무튼 range(10)하면 10번 반복되고 생성되는 정수는 0부터 9까지이다. 만약 1부터 10까지의 정수가 필요하면 range(1, 11)로 하면 된다.

LAB5-5 반복을 이용하여 팩토리얼을 계산하기

사용자로부터 임의의 정수 n을 입력받은 뒤에 for 문을 이용하여서 팩토리얼을 계산해보자. 팩토리얼 n!은 1부터 n까지의 정수를 모두 곱한 것을 의미한다. 즉, n! = 1×2×3×.....×(n-1)×n이다.

원하는 결과

```
정수를 입력하시오: 10
10!은 3628800 이다.
```

LAB5-5 반복을 이용하여 팩토리얼을 계산하기

```
n = int(input("정수를 입력하시오: "))
fact = 1
for i in range(1, n+1):
    fact = fact * i

print(n, "!은", fact, "이다.")
```

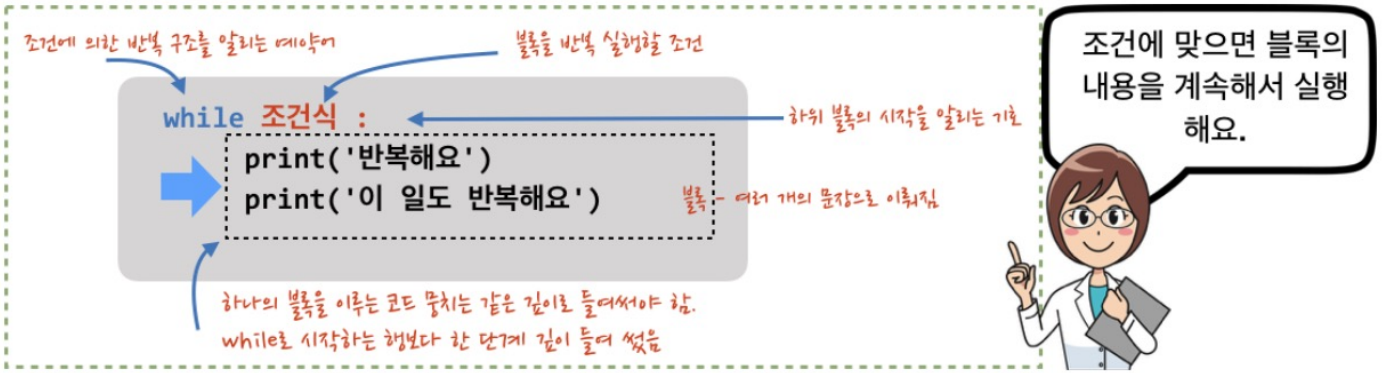
5. 7 조건에 따라 반복해서 실행하는 while 문

- 조건 제어 반복은 어떤 조건이 만족되는 동안 반복하기 때문에 붙여진 이름이다.
- 예를 들어서 자전거를 타고 가는데 앞에서 공사가 벌어져서 진행할 수가 없다. 공사가 진행 중인 동안은 같은 구간을 계속해서 반복하는 것이다.
- 이 자전거는 한 바퀴 돌 때마다 공사가 끝났는지 확인을 한다. 공사중인 상태이면 반복을 지속하고 공사가 끝나면 탈출하는 것이다.
- 이런 경우에 조건 제어 반복을 사용한다. 조건 제어 반복은 while 반복문을 사용하는 것이 편리하다.

while 반복문을 사용한 반복 실행의 일반적인 흐름

5. 7 조건에 따라 반복해서 실행하는 while 문

- while 반복문은 다음과 같은 구조를 갖는다.



위의 예를 프로그램으로 작성해보면 다음과 같다.

```
under_construction = True
while under_construction:
    response = input("공사가 완료 되었습니까?")
    if response == "예" :
        under_construction = False

print("루프를 탈출했습니다.")
```

LAB5-6 사용자로부터 암호를 받아 로그인하기

어떤 경우에 반복되는 횟수를 알 수 없을까? 가장 대표적인 경우가 사용자로부터 어떤 값을 받아서 처리할 때이다. 사용자가 입력하는 값을 예측하기 힘들기 때문이다. 예를 들어서 사용자가 암호를 입력하고 프로그램에서 암호가 맞는지 체크한다고 하자.

우리가 작성할 코드는 사용자가 바른 암호를 입력할 때까지 질문을 계속한다. 암호가 "pythonisfun"이라고 가정하고 간단한 알고리즘을 생각해 보자. 다음과 같이 틀린 암호를 하나 생성해 두고, 사용자의 입력으로 대체한다. 그리고 이것이 옳은 암호인지를 체크해서 아닌 경우에는 이 일을 반복한다. 반복을 빠져나가면 로그인 성공을 표시해 준다.

1. 암호=""
2. 암호가 "pythonisfun"이 아니면 다음을 반복한다.
    - 사용자로부터 암호를 입력받는다.
3. "로그인 성공"을 출력한다.

원하는 결과

```
암호를 입력하시오: idontknow
암호를 입력하시오: 12345678
암호를 입력하시오: pythonisfun
** 로그인 성공 **
```

LAB 5-6 사용자로부터 암호를 받아 로그인하기

```
password = ""
while password != "pythonisfun":
    password = input("암호를 입력하시오: ")
print("** 로그인 성공 **")
```

잠깐 – 내 패스워드가 뭐였지?

인스타그램, 카카오톡, 페이스북, 이메일 등 전자기기를 이용한 인터넷 서비스는 삶의 필수적인 부분이 되었다. 하지만 많은 서비스를 이용하는 만큼 인증을 위한 패스워드 역시 서비스의 종류만큼 많이 필요하게 되었다. 이러한 패스워드를 기억하지 못해서 혼란에 빠지는 증상이나 스트레스를 "**패스워드 증후군**"이라 한다.

쉽고 기억하기 쉬우면서도 해커가 깨뜨리기 어려운 패스워드를 만들기 위해서는 자신이 좋아하는 단어를 하나 고르고 앞뒤로 특수문자와 숫자를 삽입하는 것이 좋다. 예들들어 자신이 Daisy(민들레)를 좋아한다고 가정하고 야구선수 이대호의 등번호 10을 좋아한다고 하자. 이 경우 **[Daisy^^10]** 이라는 암호를 사용한다면 나름 기억하기 좋고 강력한 암호가 될 것이다. 이 경우 대문자를 조합하는 것이 더 좋기 때문에 첫 글자를 대문자로 사용했으나 **[daiSY^^10]**과 같이 한다면 더욱 더 강력한 암호가 될 것이다. 혹은 one+three==4과 같은 자신이 좋아하는 두 수를 이용한 계산식을 사용하는 것도 기억하기 좋으면서 강력한 암호일 것이다.

5. 8 일정한 횟수 반복에 while 사용하기

- 횟수를 알 수 없는 경우에만 while 루프를 사용할 수 있는 것은 아니다. 횟수를 알고 있는 경우에도 while 루프를 사용할 수 있다.
- 예를 들어서 1부터 10까지의 합을 계산하는 코드를 while 루프로 작성해보자. 이를 위하여 1부터 10까지 증가하는 변수 count를 1로 초기화하여 만들자.
- 그리고 이 값들을 저장할 변수 s를 선언하고 0으로 초기화하자. 이 count에 s를 누적해서 더하는 코드를 다음과 같이 작성할 수 있을 것이다.

```
count = 1
s = 0          # s는 누적하여 더한 값을 담을 변수로 0으로 초기화 함
while count <= 10 :
    s = s + count          # 매번 count 값을 s에 더함
    count = count + 1      # 매번 count 값을 1씩 증가시킴
print("합계는", s)
```

```
합계는 55
```

5. 8 일정한 횟수 반복에 while 사용하기

- 이 코드의 count와 s 값의 변화 그리고 count <= 10 조건문의 반환값은 다음과 같다.

count	s	count <= 10	반복 여부
1	1	True	반복
2	1+2	True	반복
3	1+2+3	True	반복
4	1+2+3+4	True	반복
5	1+2+3+4+5	True	반복
6	1+2+3+4+5+6	True	반복
7	1+2+3+4+5+6+7	True	반복
8	1+2+3+4+5+6+7+8	True	반복
9	1+2+3+4+5+6+7+8+9	True	반복
10	1+2+3+4+5+6+7+8+9+10	True	반복
11	1+2+3+4+5+6+7+8+9+10	False	반복 중단!

LAB5-7 입력받은 수를 사용하는 구구단 출력

사용자로부터 특정한 수를 입력받아 구구단의 특정 단을 while 반복문을 이용하여 출력하여 보자. 예를 들어 다음과 같이 9를 입력받을 경우 9*1, 9*2, 9*3, .., 9*9까지 9번 반복시켜 출력하면 될 것이다.

원하는 결과

```
원하는 단은: 9
9*1=9
9*2=18
9*3=27
9*4=36
9*5=45
9*6=54
9*7=63
9*8=72
9*9=81
```

LAB 5-7 입력받은 수를 사용하는 구구단 출력

```
dan = int(input("원하는 단은: "))
i = 1

while i <= 9:
    print("%s*%s=%s" % (dan, i, dan*i))
    i = i + 1
```

도전문제 5.7

구구단의 1단부터 9단까지를 모두 출력하도록 위의 프로그램을 수정해보자.

LAB5-10 사용자가 입력하는 숫자의 합을 계산하자

사용자가 입력한 숫자들을 더하는 프로그램을 작성해보자. 사용자가 yes라고 답한 동안에만 숫자를 입력받는다.

원하는 결과

```
숫자를 입력하시오: 10
계속?(yes/no): yes
숫자를 입력하시오: 20
계속?(yes/no): no
합계는 : 30
```

LAB5-10 사용자가 입력하는 숫자의 합을 계산하자

```
total = 0
answer = 'yes'
while answer == 'yes':
    number = int(input('숫자를 입력하시오: '))
    total = total + number
    answer = input('계속?(yes/no): ')

print('합계는 : ', total)
```

잠깐 – for와 while

모든 **for** 문은 **while** 문으로 고쳐 쓸 수 있고, 모든 **while** 문 역시 **for** 문으로 고쳐 쓸 수 있다. 하지만, **for** 문의 형식은 횟수가 정해져 있는 반복에 적합하고, while은 횟수를 정해 놓지 않고 변화하는 조건을 체크하며 반복하는 일에 적합하다.

LAB5-11 무한 반복문으로 숫자 맞추기 게임을 만들자

사용자가 답을 제시하면 프로그램은 자신이 저장한 정수와 비교하여 제시된 정수가 더 높은지 낮은지 만을 알려준다. 정수의 범위를 1부터 100까지로 한정하면 최대 7번이면 누구나 알아맞힐 수 있다. 정수의 범위를 1부터 1,000,000까지 확대하더라도 최대 20번이면 맞출 수 있다. 왜 그럴까? 이진 탐색의 원리 때문이다. 중간값과 한 번씩 비교할 때마다 탐색의 범위는 1/2로 대폭 줄어든다. 게임이 끝나면 몇 번 만에 맞추었는지도 함께 출력한다. 다음의 경우 87이 정답이지만 매번 임의로 생성된 수가 정답이 될 수 있음에 유의하자.

원하는 결과

```
1부터 100 사이의 숫자를 맞추시오
숫자를 입력하시오: 50
낮음!
숫자를 입력하시오: 86
낮음!
숫자를 입력하시오: 87
축하합니다. 총 시도횟수 = 3
```

LAB 5-11 무한 반복문으로 숫자 맞추기 게임을 만들자

```
import random

tries = 0
guess = 0
answer = random.randint(1, 100)
print("1부터 100 사이의 숫자를 맞추시오")
while guess != answer:
    guess = int(input("숫자를 입력하시오: "))
    tries = tries + 1
    if guess < answer:
        print("낮음!")
    elif guess > answer:
        print("높음!")

print("축하합니다. 총 시도횟수=", tries)
```

도전문제 5.10

시도 횟수를 최대 10번으로 제한하려면 위의 프로그램을 어떻게 변경하여야 하는가?

LAB 5-12 암산 문제를 만들어보자

초등학생들을 위하여 산수 문제를 발생시키는 프로그램을 작성해보자. 질문은 다음과 같이 1에서 100 사이의 무작위적인 두 수가 나타나고 사용자가 두 수의 합을 입력하도록 되어 있다. 이때 정답을 맞힐 경우 '잘했어요'를 출력하고 그렇지 않을 경우 '정답은 oo입니다. 다음 번에는 잘할 수 있죠?'를 출력하여라. 이 프로그램은 Ctrl + C를 입력하여 사용자가 종료하기 전까지 멈추지 않는다.

원하는 결과

```
9 + 48 = 57
잘했어요!!
65 + 11 = 76
잘했어요!!
91 + 31 = 121
정답은 122 입니다. 다음번에는 잘할 수 있죠?
38 + 4 =
```

LAB5-12 암산 문제를 만들어보자

```
import random

while True:
    x = random.randint(1, 100)
    y = random.randint(1, 100)
    print(x, '+', y, '=', end = ' ')
    answer = int(input())
    if answer == x + y:
        print('잘했어요!!')
    else:
        print('정답은',x+y,'입니다. 다음 번에는 잘할 수 있죠?')
```

도전문제 5.11

덧셈 뿐만 아니라 무작위적으로 뺄셈 문제도 출제할 수 있도록 위의 프로그램을 수정하라.

```
23 - 21 = 2
잘했어요!!
38 - 89  = 2
정답은 -51 입니다. 다음 번에는 잘할 수 있죠?
```

LAB5-13 창업자를 위한 기능 : 모든 샌드위치 종류 출력하기

달수는 새로 샌드위치 가게를 차렸다. 달수는 자신의 가게에서 제공가능한 빵, 고기, 야채, 소스의 조합을 통해서 만들 수 있는 모든 샌드위치의 종류를 출력하고 싶다. 달수네 가게의 빵 종류는 "호밀빵", "위트", "화이트"가 가능하며 고기로는 "미트볼", "소시지", "닭가슴살", 야채로는 "양상추", "토마토", "오이", 소스로는 "마요네즈", "허니 머스타드", "칠리" 등이 가능하다. 각 재료들은 한 가지씩만 선택이 가능하다고 하자. 가능한 조합은 어떻게 될까? 다음과 같이 모든 조합을 출력해 보자.

원하는 결과

```
달수네 샌드위치 가게의 가능한 조합
호밀빵 + 미트볼 + 양상추 + 마요네즈
호밀빵 + 미트볼 + 양상추 + 허니 머스타드
호밀빵 + 미트볼 + 양상추 + 칠리
...
화이트 + 닭가슴살 + 오이 + 마요네즈
화이트 + 닭가슴살 + 오이 + 허니 머스타드
화이트 + 닭가슴살 + 오이 + 칠리
```

LAB 5-13 창업자를 위한 기능 : 모든 샌드위치 종류 출력하기

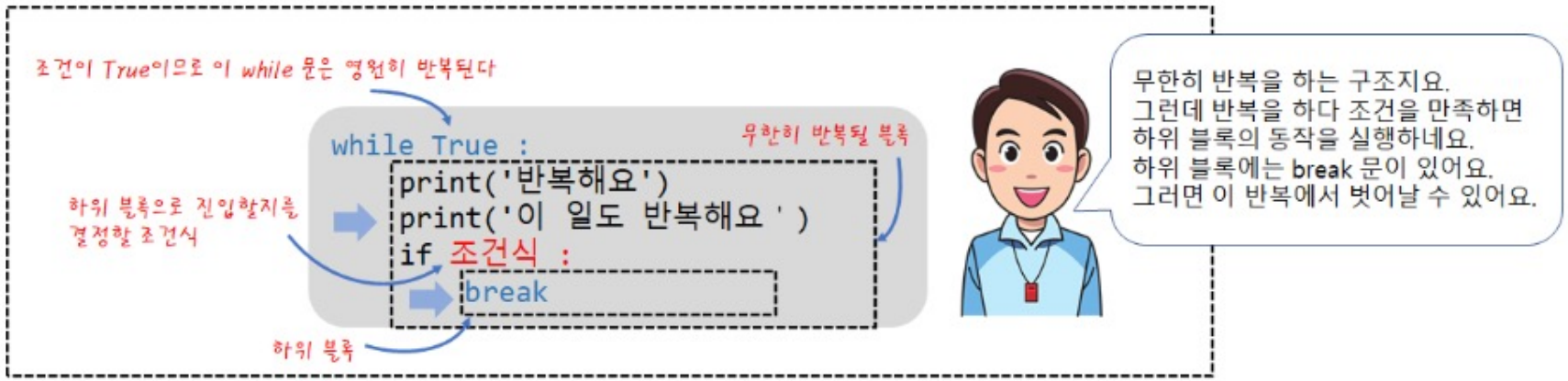
```
breads = ["호밀빵", "위트", "화이트" ]
meats = ["미트볼", "소시지", "닭가슴살"]
vegis = ["양상추", "토마토", "오이"]
sauces = ["마요네즈", "허니 머스타드", "칠리"]

print('달수네 샌드위치 가게의 가능한 조합')
for b in breads:
    for m in meats:
        for v in vegis:
            for s in sauces:
                print(b + " + " + m + " + " + v + " + " + s)
```

잠깐 – 반복문이 중첩될 수록 코드를 읽기가 어려워 진다

위의 경우와 같이 반복문이 4번 반복 중첩될 경우 들여쓰기 레벨이 증가하게 되고 루프제어 변수도 4개가 필요하다. 이렇게 되면 루프를 제어하는 변수가 많아져서 코드를 읽기가 매우 어려워지게 된다. 그리고 코드를 수정하거나 업데이트 하는 작업 역시 쉽지 않으므로 가급적 3중 루프 이상의 중첩루프는 사용하지 않는 것이 좋다.

5. 9 무한 루프와 break로 빠져나가기

- 조건 제어 루프에서 가끔은 프로그램이 무한히 반복하는 일이 발생한다. 이것은 **무한 루프**infinite loop라고 한다.
- 무한 반복이 발생하면 프로그램은 빠져 나올 수 없기 때문에 문제가 된다.
- 하지만 가끔은 무한 루프가 사용되는데 예를 들면 신호등 제어 프로그램은 무한 반복하여야 하기 때문이다.
- 무한 반복 루프는 다음과 같은 형태를 가진다.



5. 9 무한 루프와 break로 빠져나가기

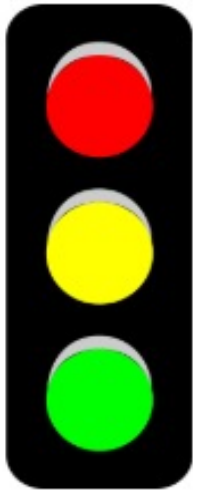
- while 루프의 조건에 True가 있다. 따라서 조건이 항상 참이므로 무한히 반복된다.
- 하지만 무한 루프라고 하더라도 어떤 조건이 성립하면 무한 루프를 빠져나와야 하는 경우도 많다.
- 이런 경우는 if 문장을 사용하여서 루프를 빠져나오게 된다. break 문장은 루프를 강제적으로 빠져 나올 때 사용하는 문장이다.
- 간단한 예제를 작성해보면 다음과 같다.

```
while True:
    light = input('신호등 색상을 입력하시오:')
    if light == 'green':
        break
print('전진!!')
```

5. 9 무한 루프와 break로 빠져나가기

- 위의 코드에서는 화면에 '신호등 색상을 입력하시오:'을 출력하고 사용자의 입력을 기다린다.
- while True:로 되어 있으므로 무한 루프이다.
- 사용자가 'green'를 입력하면 break 문장을 실행하여서 무한 루프를 빠져나간다.
- 사용자가 입력한 색상이 'green'이 아니면 이 루프는 계속 반복한다.

```
신호등 색상을 입력하시오: red
신호등 색상을 입력하시오: yellow
신호등 색상을 입력하시오: green
전진!!
```



도전문제 5.12

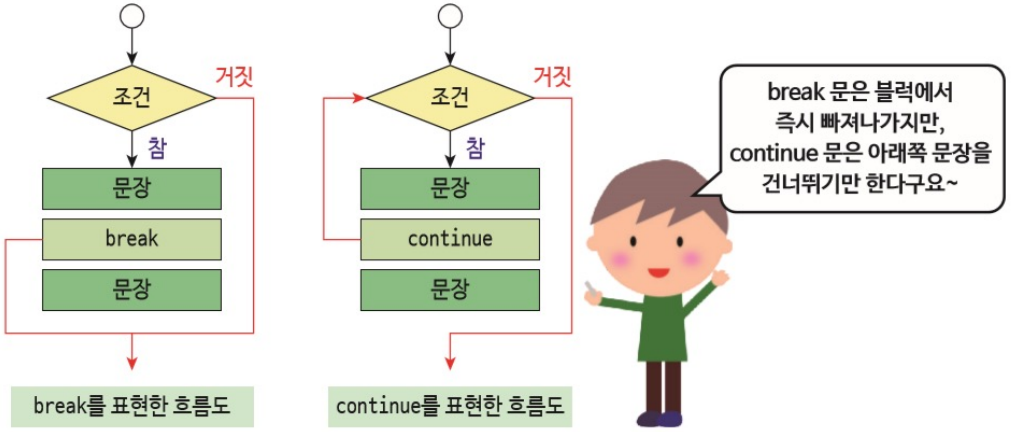
사용자로부터 하나의 단어를 입력받은 다음 이 단어에서 모음이 나타나기 전까지의 모든 자음을 출력하는 프로그램을 작성하여라. 예를 들어 다음과 같이 'programming'이 입력되면 'o'가 나타나기 이전인 pr만 출력하도록 하여라.

```
단어를 입력하세요 : programming
pr
```

5. 10 루프를 제어하는 고급 기법 : continue와 break

```
st = 'I love Python Programming'  # 출력을 위한 문자열
for ch in st:
    if ch in ['a','e','i','o','u', 'A','E','I','O','U']:
        continue      # 모음일 경우 아래 출력을 건너뛴다
    print(ch, end='')
```

```
lv Pythn Prgrmmng
```



- 루프를 제어하는 첫번째 방법은 break 문으로 이 문장을 만나면 가장 가까운 블럭를 즉시 빠져나오게 된다.
- 그렇다면 continue는 어떻게 수행될까? 이 키워드는 루프를 빠져나오지 않고 continue 아래의 문장만을 건너뛰는 역할을 한다.
- 즉 반복문이 종료되는 것은 조건이 거짓일 때에만 해당한다. continue를 사용하는 예제코드가 아래에 있다.

잠깐 – continue와 break를 너무 많이 쓰면 코드를 읽기 어려워진다

break와 continue는 프로그램의 제어를 효율적으로 하는데 편리하게 사용할 수 있다. 하지만, break와 continue 문이 너무 많이 사용되는 경우 제어의 흐름에 일관성이 없어 프로그램을 이해하는 것이 어려워진다. 따라서 continue와 break는 필요한 경우에만 제한적으로 사용하는 것이 좋다.

5. 11 출력을 예쁘게 만드는 포매팅

- 지금까지 print()를 이용하여 다양한 출력을 해 보았다. 그런데 출력을 다듬는 일을 하지 않아 출력 메시지의 간격이 들쭉날쭉하였다.
- 문자열을 가지런히 출력하는 출력 방법에 대해 상세하게 알아보자. 문자열을 출력할 적에 매우 유용한 메소드가 바로 format() 메소드인데 다음과 같은 플레이스홀더를 이용하여 원하는 문자나 숫자를 적절한 포맷으로 출력할 수 있다.
- 여기서 나타나는 {}는 플레이스홀더로 format() 메소드의 인자로 들어오는 값이 출력되는 위치를 지정하는 역할을 한다.

```
>>> '{} Python!'.format('Hello')
Hello Python!
>>> 'I like {} and {}'.format('Python', 'Java')
'I like Python and Java'
```

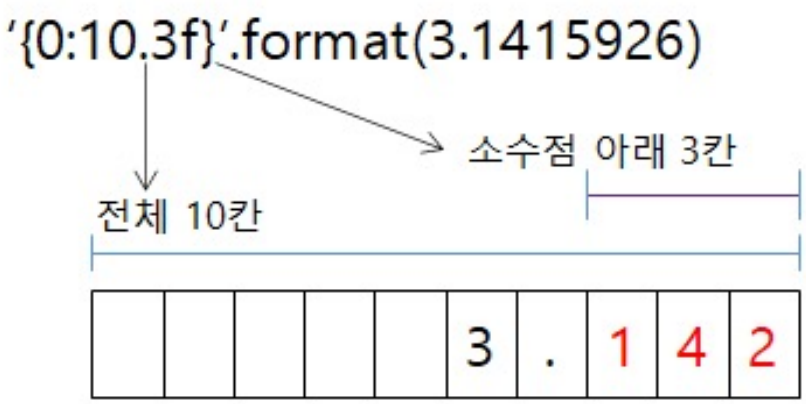
5. 11 출력을 예쁘게 만드는 포매팅

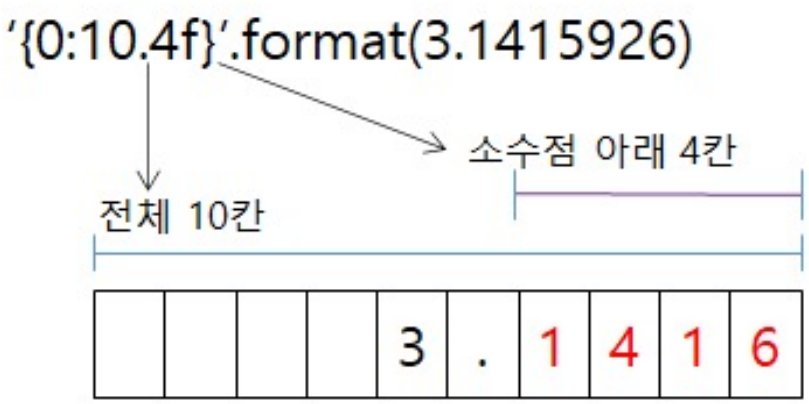
- 그리고 format() 메소드의 인자에 대해 0, 1, 2와 같은 인덱스를 이용하여 인자 값을 지정할 수 있다.

```
>>> 'I like {0} and {1}'.format('Python', 'Java')
'I like Python and Java'
>>> 'I like {1} and {0}'.format('Python', 'Java')
'I like Java and Python'
>>> 'I like {0}, {0}, {0}'.format('Python', 'Java')
'I like Python, Python, Python'
```

- 다음으로 {0:.2f}와 같이 소수점 아래 둘째자리 숫자를 출력하도록 할 수 있다. 만일 소수점 아래 셋째 자리까지 출력을 하려면 {0:.3f}와 같이 지정할 수 있다.
- 여기서 f를 **부동소수점**floating point을 의미한다. 이때, {0:5.2f}나 {0:6.2f}와 같이 마침표 앞에 전체 필드의 폭을 지정할 수도 있다.

```
>>> '소수점 아래 두자리 정밀도 : {0:.2f}, 세자리 정밀도 {0:.3f}'.format(1/3)
소수점 아래 두자리 정밀도 : 0.33, 세자리 정밀도 0.333
>>> '전체 10칸을 차지하는 실수 : {0:10.3f}, {0:10.4f}'.format(3.1415926)
전체 10칸을 차지하는 실수 :      3.142,     3.1416
```


'{0:10.3f}'.format(3.1415926)
소수점 아래 3칸
전체 10칸
3 . 1 4 2


'{0:10.4f}'.format(3.1415926)
소수점 아래 4칸
전체 10칸
3 . 1 4 1 6

5. 11 출력을 예쁘게 만드는 포매팅

- 또한 정수출력을 위해서는 {1:3d}, {1:4d}, {1:5d}와 같이 출력시에 차지할 칸의 수를 지정할 수 있다.
- 세 개의 인자에 대해 각각 3칸, 4칸, 5칸의 공간을 차지하도록 하여 세 개의 정수를 출력하도록 하는 코드를 만들어 보자.
- 아래와 같이 가지런하고 보기 좋은 출력을 만들 수 있다.

```
>>> for i in range(2, 11, 2):
...     print('{0:3d} {1:4d} {2:5d}'.format(i, i*i, i*i*i))
...
  2    4    8
  4   16   64
  6   36  216
  8   64  512
 10  100 1000
```

summary

핵심 정리

- 문장들을 반복 실행하려면 for나 while을 사용한다.
- 반복 실행되는 문장들을 들여쓰기 하여야 한다.
- **for** 문은 반복 횟수가 정해져 있을 때 적합하다.
- **while** 문은 반복 조건이 정해져 있을 때 유용하다.
- **while** 반복문은 문장의 초입에서 조건식이 검사된다.
- continue와 break는 루프제어를 위한 고급기능을 제공한다.

Questions?