Python

프로그래밍기초와실습

CH.05

반복문과 컴프리헨션(3)

보조 제어문

- 제어문(조건문, 반복문) 내에서 사용
- 조건문과 반복문을 보조하는 기능 수행
- break문, continue문

break

- 반복문을 강제로 종료하기 위한 보조 제어문
- 주로 반복문 내부에서 조건문과 함께 사용

break

무한반복 프로그램1 : while문 내부에 식의 상태 변경을 위한 문장이 없는 경우

```
1  # while문 내부에 식의 상태 변경을 위한 문장이 없는 경우
2  # sum_err.py
3
4  value = int(input("양의 정수를 입력하세요 : "))
5
6  sum_value = 0
7  count = 1
8
9  while count <= value:
10     sum_value += count
11
12 print("1부터 %d까지의 합은 %d입니다." % (value, sum_value))
```

실행 결과

```
양의 정수를 입력하세요 : 10
Traceback (most recent call last):
 File "C:/Python/sum_err.py", line 10, in <module>
  sum_value += count
KeyboardInterrupt
```

break

무한반복 프로그램2 : while문의 식을 True로 하여 강제 무한 반복

```
1  # 무한루프 프로그램
2  # infinite_loop.py
3
4  while True:
5      print("무한루프")
```

실행 결과

```
무한루프
무한루프
...
무한루프
무한루프
무한루프
Traceback (most recent call last):
 File "C:/Python/infinite_loop.py", line 5, in <module>
  print("무한루프")
KeyboardInterrupt
```

break

❖ 특정 조건이 발생했을 때 무한반복 프로그램을 강제 종료

```
1  # 입력한 숫자까지의 합을 구하는 프로그램
2  # 무한루프 사용
3  # sum_infinite_loop1.py
4
5  value = int(input("정수를 입력하세요 : "))
6
7  count = 1
8  sum_value = 0
9
10 while True:
11     sum_value += count
12     count += 1
13
14     if count == value:
15         break
16
17 print("1부터 %d까지의 합은 %d입니다." % (value, sum_value))
```

count가 value와 같아지면
break문을 만나서 반복문 종료

실행 결과

```
정수를 입력하세요 : 10
1부터 10까지의 합은 45입니다.
```

break

```
1  # for문에서 break 사용
2  # break_for.py
3
4  value = int(input("정수를 입력하세요 : "))
5
6  sum_value = 0
7  for i in range(1, value + 1):
8      sum_value += i
9
10     if i == 5:
11         break
12
13 print("1부터 %d까지의 합은 %d입니다." % (value, sum_value))
```

실행 결과 1

```
정수를 입력하세요 : 10
1부터 10까지의 합은 15입니다.
```

실행 결과 2

```
정수를 입력하세요 : 100
1부터 100까지의 합은 15입니다.
```

break, continue

break

- 반복문에 break문이 있을 경우 반복문에 else: 사용 가능
- 반복문 전체를 반복할 때 break문이 실행되지 않을 경우 else: 이하의 문장 실행

```
1  # for문에서 else를 이용한 break 사용 검사
2  # break_for.py
3
4  value = int(input("정수를 입력하세요 : "))
5
6  sum_value = 0
7  for i in range(1, value + 1):
8      sum_value += i
9
10     if i == 5:
11         break
12 else:
13     print("5보다 낮은 수가 입력되었습니다.")
14
15 print("1부터 %d까지의 합은 %d입니다." % (value, sum_value))
```

실행 결과 1

```
정수를 입력하세요 : 10
1부터 10까지의 합은 15입니다.
```

실행 결과 2

```
정수를 입력하세요 : 4
5보다 낮은 수가 입력되었습니다.
1부터 4까지의 합은 10입니다.
```

break

실행 결과 1	정수를 입력하세요 : 10 1부터 10까지의 합은 15입니다.
실행 결과 2	정수를 입력하세요 : 4 5보다 낮은 수가 입력되었습니다. 1부터 4까지의 합은 10입니다.

continue

- 반복을 중단하지는 않지만 특별한 이유로 반복 실행하는 문장을 건너뛰고 다음 반복을 수행하고 싶을 때 사용
- 반복문 내에서 if문과 함께 사용

```
1  # for문에서 continue 사용
2  # continue_for.py
3
4  value = int(input("정수를 입력하세요 : "))
5
6  sum_value = 0
7  for i in range(1, value + 1):
8
9      if i == 5:
10         continue
11
12     sum_value += i
13
14 print("1부터 %d까지 5를 제외한 합은 %d입니다." % (value, sum_value))
```

실행 결과

```
정수를 입력하세요 : 10
1부터 10까지 5를 제외한 합은 50입니다.
```

continue

❖ 주의! : continue문을 만나면 반복문의 나머지 문장을 건너뛰기 때문에 while문에서 사용할 때 상태 변경 부분이 실행되지 않을 수 있다!

```
1  # while문에서 continue문 사용 주의
2  # continue_while_err.py
3
4  value = int(input("정수를 입력하세요 : "))
5
6  sum_value = 0
7  count = 1
8  while count <= value:
9
10     if count == 5:
11         continue
12
13     sum_value += count
14     count += 1
15
16 print("1부터 %d까지 5를 제외한 합은 %d입니다." % (value, sum_value))
```

실행 결과

```
정수를 입력하세요 : 10
Traceback (most recent call last):
 File "C:/Python/continue_while_err.py", line 11, in <module>
  continue
KeyboardInterrupt
```

continue문을 만나면
13, 14번 라인이 실행되지 않음
따라서 count가 5에서 변경되지 않음
무한루프

continue

실행 결과

```
정수를 입력하세요 : 10
Traceback (most recent call last):
 File "C:/Python/continue_while_err.py", line 11, in <module>
  continue
KeyboardInterrupt
```

continue

❖ 해결 방안 : continue 위에 상태를 변경하는 부분 추가

```
1  # while문에서 continue문 사용 주의
2  # continue_while.py
3
4  value = int(input("정수를 입력하세요 : "))
5
6  sum_value = 0
7  count = 1
8  while count <= value:
9
10     if count == 5:
11         count += 1
12         continue
13
14     sum_value += count
15     count += 1
16
17 print("1부터 %d까지 5를 제외한 합은 %d입니다." % (value, sum_value))
```

실행 결과

```
정수를 입력하세요 : 10
1부터 10까지 5를 제외한 합은 50입니다.
```

컴프리헨션(함축)

- 리스트, 딕셔너리, 셋과 같은 시퀀스 자료를 만들 때 for문과 if문을 사용
- 반복문과 조건문을 결합하여 비교적 간단한 구문으로 시퀀스 자료형 생성

자료형	형식
리스트	[표현식 for 변수 in 시퀀스 자료]
	[표현식 for 변수 in 시퀀스 자료 if 조건]
딕셔너리	{ 키_표현식 : 값_표현식 for 표현식 in 시퀀스 자료 }
	{ 키_표현식 : 값_표현식 for 표현식 in 시퀀스 자료 if 조건 }
셋	{ 표현식 for 변수 in 시퀀스 자료 }
	{ 표현식 for 변수 in 시퀀스 자료 if 조건 }

리스트 컴프리헨션

- 1, 2, 3, 4, 5의 다섯 개의 요소를 갖는 리스트를 만드는 여러가지 방법

```
>>> values = []
>>> values.append(1)
>>> values.append(2)
>>> values.append(3)
>>> values.append(4)
>>> values.append(5)
>>> values
[1, 2, 3, 4, 5]
```

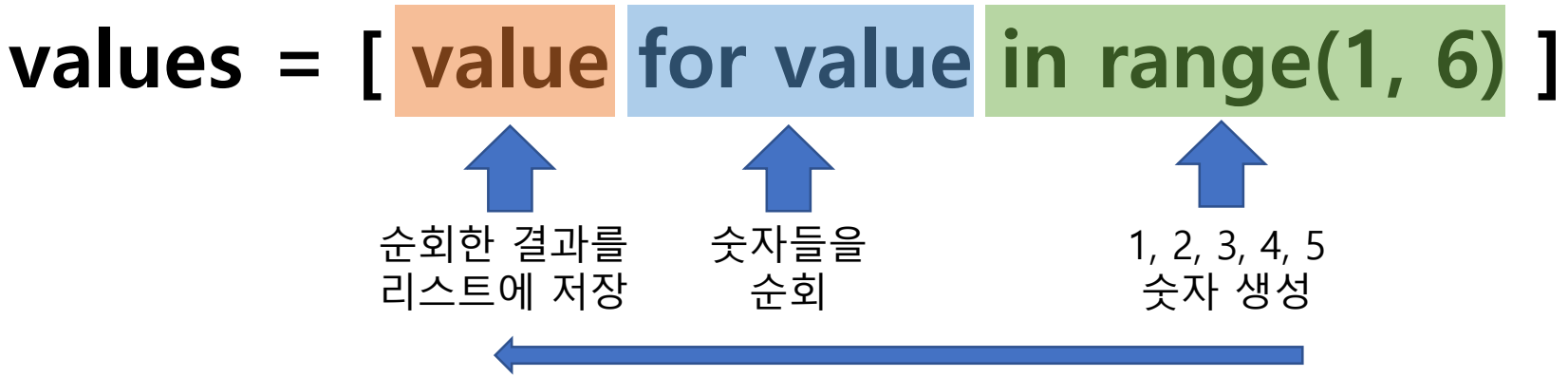
```
>>> values = []
>>> for i in range(1, 6):
        values.append(i)
>>> values
[1, 2, 3, 4, 5]
```

```
>>> values = list(range(1, 6))
>>> values
[1, 2, 3, 4, 5]
```

리스트 컴프리헨션

컴프리헨션을 이용한 리스트 생성 방법

```
>>> values = [value for value in range(1, 6)]
>>> values
[1, 2, 3, 4, 5]
```

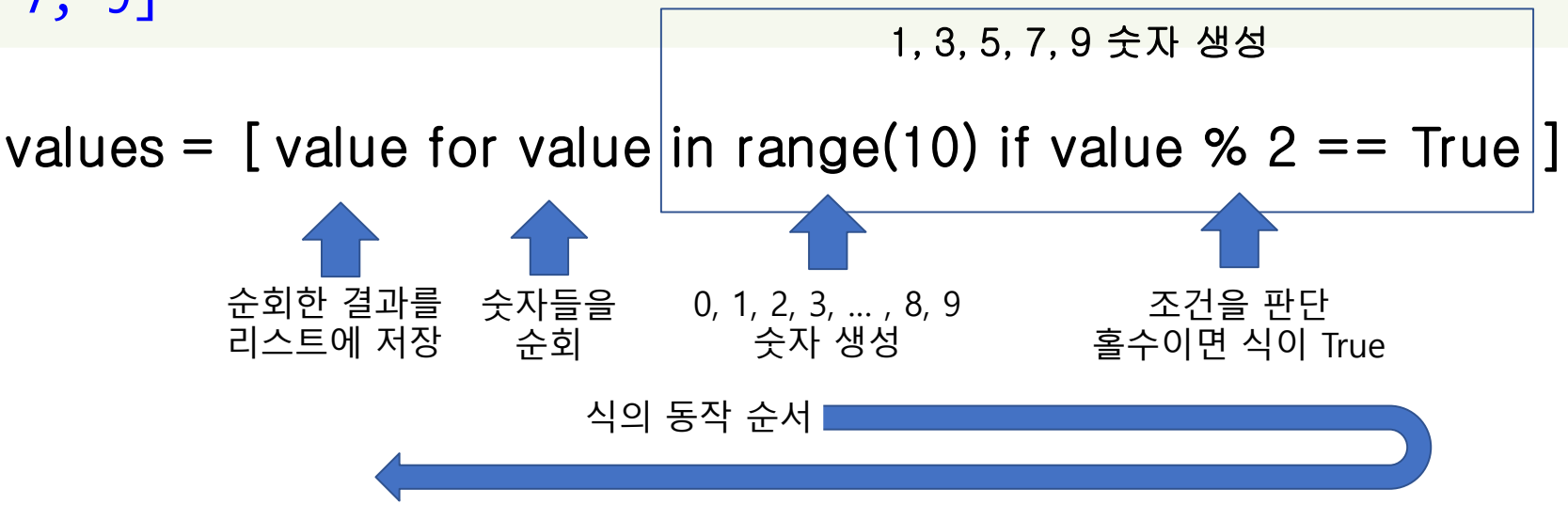



```
>>> values = [value + 1 for value in range(5)]
>>> values
[1, 2, 3, 4, 5]
```

리스트 컴프리헨션

컴프리헨션을 이용한 리스트 생성 시 조건 추가

```
>>> values = [value for value in range(10) if value % 2 == True]
>>> values
[1, 3, 5, 7, 9]
```



컴프리헨션의 중첩

❖ 중첩된 반복문을 이용하여 튜플을 요소로 갖는 리스트 생성

```
>>> rows = range(1, 4)
>>> cols = range(1, 3)
>>> matrix = []
>>> for r in rows:
        for c in cols:
            matrix.append((r, c))
>>> matrix
[(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2)]
```

❖ 컴프리헨션 중첩으로 해결

```
>>> rows = range(1, 4)
>>> cols = range(1, 3)
>>> matrix = [ (r, c) for r in rows for c in cols ]
>>> matrix
[(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2)]
```

딕셔너리 컴프리헨션

- 키와 값 모두를 저장하는 표현식 필요

```
>>> student = {"Kim":90, "Lee":80, "Park":95, "Choi":70}
>>> student_good = {k : v for k, v in student.items() if v > 80}
>>> student_good
{'Kim': 90, 'Park': 95}
```

```
>>> hello = "hello"
>>> l_count = {letter : hello.count(letter) for letter in hello}
>>> l_count
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

셋 컴프리헨션

- 리스트와 비슷한 방법으로 사용, 단 중괄호를 이용

```
>>> hello = "hello"
>>> set_hello = {letter for letter in hello}
>>> set_hello
{'l', 'o', 'e', 'h'}
```

제너레이터 컴프리헨션

- 컴프리헨션 문법을 소괄호를 사용할 경우 튜플 컴프리헨션이 아닌 제너레이터 컴프리헨션으로 동작

```
>>> numbers = (number for number in range(10))
>>> numbers
<generator object <genexpr> at 0x03F63EB0>
```

제너레이터

- 제너레이터는 객체를 생성하는 것이 아닌 이터레이터를 이용하여 단 한번만 반복할 수 있는 특별한 자료형

```
>>> numbers = (number for number in range(10))
>>> for i in numbers:
        print(i, end = " ")

0123456789
```

- 제너레이터를 순회하고 나면 더 이상 값의 접근이 불가

```
>>> numbers = (number for number in range(10))
>>> values1 = list(numbers)
>>> values1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> values2 = list(numbers)
>>> values2
[]
```

Thank you