제어문

for

제어문 (Control Statement)

- 반복 구문 - for

  for 변수(들) in 순회 가능한 자료 또는 실행 결과 :
  변수에 값이 할당되는 동안 수행할 구문1
  변수에 값이 할당되는 동안 수행할 구문2
  . . .

  – 키워드 in 뒤에 기재된 자료 구조 또는 실행 결과로부터 항목을 처음부터 마지막까지 하나씩 꺼내어 변수에 넣어, for 블록의 구문들을 순서대로 반복하여 실행한다.
  – 변수에 들어갈 값이 없게 되면 for 블록은 더 이상 실행되지 않고 빠져 나온다.

제어문 (Control Statement)

- 반복 구문 - for

  for 변수(들) in 순회 가능한 자료 또는 실행 결과 :
  변수에 값이 할당되는 동안 수행할 구문1
  변수에 값이 할당되는 동안 수행할 구문2
  …

  – 정수 1부터 5까지 화면에 출력하시오.

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

```
1
2
3
4
5
```

제어문 (Control Statement)

- 반복 구문 - for

  for 변수(들) in 순회 가능한 자료 또는 실행 결과 :
  변수에 값이 할당되는 동안 수행할 구문1
  변수에 값이 할당되는 동안 수행할 구문2
  . . .

  – 튜플 (1, 2)와 (3, 4)가 항목인 리스트 a에서 각 튜플 별로 앞의 값만 화면에 출력하시오.

```
1  a = [(1, 2), (3, 4)]
2
3  for i, j in a:
4      print(i)
```

```
1
3
```

제어문 (Control Statement)

- 반복 구문 while 및 for의 비교
  - while 구문과 for 구문은 둘 다 반복을 실행하므로 상황에 따라 더 편하거나 적절한 것을 선택할 수 있다.
  - 다만 while 구문은 참/거짓으로 논리를 검사할 때, for 구문은 여러 차례 순서대로 내용을 추출할 때 사용하는 것이 더 일반적이고 바람직하다.

```
1  a = [1, 2, 3]
2  i = 0
3
4  while i < len(a):
5      print(a[i])
6      i += 1
```

```
1
2
3
```

```
1  a = [1, 2, 3]
2
3  for i in a:
4      print(i)
```

```
1
2
3
```

제어문 (Control Statement)

- 반복 구문 - for / else

```
for 변수(들) in 순회 가능한 자료 또는 실행 결과 :
        변수에 값이 할당되는 동안 수행할 구문1
        …
else :
        변수에 값이 할당되지 않는 경우 수행할 구문1
        …
```

  - 변수에 값이 할당되는 동안 for 블록의 구문들을 순서대로 반복하여 실행한다.
  - 변수에 더 이상 들어갈 값이 없으면 else 블록의 구문들을 순서대로 한번 실행하고 빠져 나온다.

제어문 (Control Statement)

- 반복 구문 - for / else
  - 튜플 (“a”, “b”, “c”, “d”, “e”)에 들어 있는 값을 순서대로 결합하여 문자열 s를 만들고, 튜플에 있는 모든 값을 사용하였으면 문자열 s를 출력한 뒤 프로그램을 종료하시오.

```
1  t = ("a", "b", "c", "d", "e")
2  s = ""
3
4  for c in t:
5      s += c
6  else:
7      print(s)
```

```
abcde
```

제어문 (Control Statement)

- 반복을 강제로 종료하기 - **break**
  - 명령어 break는 자신이 속해 있는 반복 구문을 빠져 나온다.
  - 아직 변수에 할당할 값이 남아 있더라도, break를 만나면 반복 구문은 종료된다.

- 반복의 처음 위치로 돌아가기 - **continue**
  - 명령어 continue는 자신이 속해 있는 반복 구문의 맨 처음 위치로 돌아 간다.
  - continue를 만나면 반복 구문의 맨 처음 위치로 돌아가서 그 다음 값을 변수에 할당한다.

제어문 (Control Statement)

- for 구문과 함께 사용되는 범위 지정 - **range**

  for 변수(들) in range(시작 위치, 끝 위치, 간격) :
  변수에 값이 할당되는 동안 수행할 구문1
  . . .

  – 명령어 range를 키워드 in 뒤에 기재하여, 정수의 목록을 만들어 사용할 수 있다. '시작 위치'부터 '끝 위치'가 되기 전까지 '간격' 단위로 순서대로 정수를 목록화한다.

숫자의 수	형태	의미
1	range(b)	0부터 정수 b 전까지 1씩 증가하는 정수들 (0, 1, 2, 3, ..., b-1)
2	range(a, b)	정수 a부터 정수 b 전까지 1씩 증가하는 정수들 (a, a+1, a+2, ..., b-1)
3	range(a, b, c)	정수 a부터 정수 b 전까지 c씩 증가하는 정수들 (a, a+c, a+2c, ..., b-1)

제어문 (Control Statement)

- for 구문과 함께 사용되는 범위 지정 - range

```
for i in range(3):
    print(i)
```

```
0
1
2
```

```
for i in range(1, 5):
    print(i)
```

```
1
2
3
4
```

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

```
1
3
5
7
9
```

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

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

```
for i in range(0, -7, -3):
    print(i)
```

```
0
-3
-6
```

제어문 (Control Statement)

- range의 결과로 리스트 만들기

  list(range(시작 위치, 끝 위치, 간격))

  – 명령어 range의 실행 결과인 정수의 목록은 그 자체가 리스트인 것은 아니지만, 리스트로 변환할 수 있다.

```
1  a = list(range(3))
2
3  print(a)
```

```
[0, 1, 2]
```

```
1  c = list(range(1, 10, 2))
2
3  print(c)
```

```
[1, 3, 5, 7, 9]
```

```
1  b = list(range(1, 5))
2
3  print(b)
```

```
[1, 2, 3, 4]
```

```
1  d = list(range(10, 0, -1))
2
3  print(d)
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

※ 마찬가지로, 명령어 tuple을 사용하여 튜플로, 또는 명령어 set을 사용하여 집합으로 변환할 수 있다.

제어문 (Control Statement)

- 반복 구문의 여러 가지 예

```
1  t = (1, "python", 3.14, -999, False)
2
3  for i in t:
4      if type(i) == int:
5          print(i)
```

```
1
-999
```

```
1  s = "Hello, my name is 'Hong, Kil-Dong.'"
2
3  for c in s:
4      if c != " ":
5          print(c, end=" ")
```

```
H e l l o , m y n a m e i s ' H o n g , K i l - D o n g . '
```

※ print 할 때 마지막 부분에 end="원하는 문자열"을 추가하면, 줄바꿈을 하지 않고 end= 뒤에 기재한 문자열을 출력한다.

제어문 (Control Statement)

- 반복 구문의 여러 가지 예

```
1  d = {2017: "닭", 2018: "개", 2019: "돼지"}
2
3  for k in d:
4      print(k)
```

```
2017
2018
2019
```

```
1  for v in d.values():
2      print(v)
```

```
닭
개
돼지
```

```
1  for i in d.items():
2      print(i)
```

```
(2017, '닭')
(2018, '개')
(2019, '돼지')
```

제어문 (Control Statement)

- 반복 구문의 중첩 (nested statement)
  - 구문은 필요에 따라서 얼마든지 중첩되어(nested) 사용할 수 있다.
  - 중첩된 하위 블록은 상위 블록에 종속(dependent)된다.

```
1  for i in range(2):
2      print("i가 0부터 1까지인 동안 출력됩니다. 지금 i는", i, "입니다.")
3      for j in range(3):
4          print("₩t그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는", j, "입니다.")
```

```
i가 0부터 1까지인 동안 출력됩니다. 지금 i는 0 입니다.
        그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는 0 입니다.
        그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는 1 입니다.
        그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는 2 입니다.
i가 0부터 1까지인 동안 출력됩니다. 지금 i는 1 입니다.
        그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는 0 입니다.
        그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는 1 입니다.
        그 상황에서 j가 0부터 2까지인 동안 출력됩니다. 지금 j는 2 입니다.
```

제어문 (Control Statement)

- 반복 구문의 중첩 (nested statement)

```
1  primes = []
2
3  for i in range(2, 11):
4      for j in range(2, i):
5          if i % j == 0:
6              break
7      else:
8          primes.append(i)
9
10 print("2부터 10 사이의 소수는", primes, "입니다.")
```

2부터 10 사이의 소수는 [2, 3, 5, 7] 입니다.

i	j
2	-

i	j
3	2

i	j
4	2
4	3

i	j
5	2
5	3
5	4

. . .

i	j
9	2
9	3
9	4

i	j
9	5
9	6
9	7

. . .

리스트 함축 기법 (List Comprehension)

- 리스트 내에 for 구문을 함축하여 리스트 만들기
  - **리스트의 이름 = [연산 for 변수 in 기존자료 if 조건문]**
  - for 구문을 리스트 자료 구조의 내부에 직접 기재하여 새로운 리스트를 만든다.
  - '기존자료'에서 '조건문'을 만족하는 항목을 순서대로 하나씩 '변수'에 할당한 뒤, 그 변수에 대해 '연산'을 하여 새로운 리스트의 항목으로 추가한다.

```
1  a = [1, 2, 3]
2
3  b = [i * 2 for i in a]
4
5  print(b)
```

```
[2, 4, 6]
```

```
1  c = [57, -9, 3.14, 0, -123.45]
2
3  d = [x for x in c if x > 0]
4
5  print(d)
```

```
[57, 3.14]
```

※ if 조건문 부분은 생략할 수 있다.

사전 함축 기법 (Dictionary Comprehension)

- 사전 내에 for 구문을 함축하여 사전 만들기
  - **사전의 이름 = {키:값 for 변수 in 기존자료 if 조건문}**
  - for 구문을 사전 자료 구조의 내부에 직접 기재하여 새로운 사전을 만든다.
  - '기존자료'에서 '조건문'을 만족하는 항목을 순서대로 하나씩 '변수'에 할당한 뒤, 그 변수에 대해 '키:값' 처리를 하여 새로운 사전의 항목으로 추가한다.

```
1  a = {"a": 1, "b": 2, "c": 3}
2
3  b = {v:k for k, v in a.items()}
4
5  print(b)
```

```
{1: 'a', 2: 'b', 3: 'c'}
```

```
1  c = (1, 2, 3, 4, 5)
2
3  d = {x:x * 3 for x in c if x % 2 == 1}
4
5  print(d)
```

```
{1: 3, 3: 9, 5: 15}
```

※ if 조건문 부분은 생략할 수 있다.