단원 03
함수의 기초와 제어문

인공지능소프트웨어학과

강환수 교수

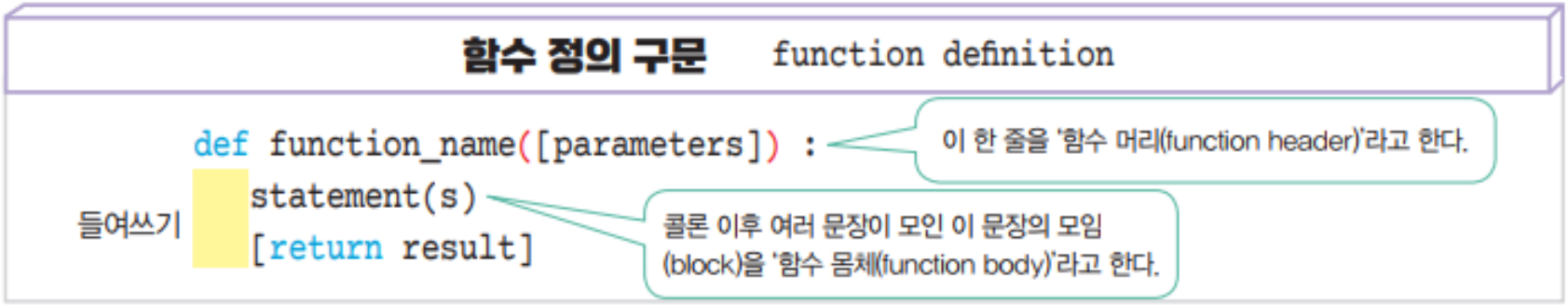
Section 1. 함수의 기초

-

함수 필요성과 기본 구조

특정 작업을 수행하는 코드의 묶음

- **필요성**
  - 반복적인 작업을 효율적으로 처리하고 코드의 가독성을 높이는 데 필수적인 요소
- **정의 구문**
  - def 키워드를 사용하여 정의
    - 함수 이름, 매개변수, 실행할 코드 블록, 그리고 return 문을 통해 반환값을 지정



함수 정의와 호출

- **함수 정의**

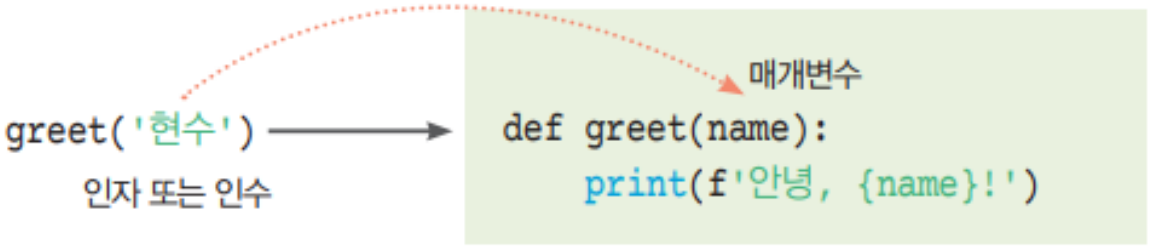
```
>>> def greet():
...     print('안녕, 현수!')
...     print('안녕, 수희!')
...     print('안녕, 지수!')
...
>>>
```

- **함수 호출**
  - 함수 이름과 함께 필요한 인자를 전달하여 호출
    - **비로소 실행**

```
>>> greet()
안녕, 현수!
안녕, 수희!
안녕, 지수!
```

매개변수와 인자

- **매개변수(parameters)**
  - 함수 정의 시 사용되는 변수로, 함수 내부에서 사용될 값을 받아들이는 역할
- **인자(arguments)**
  - 함수를 호출할 때 실제로 전달되는 값
- **함수 호출 시 전달되는 인자는 함수 정의 시 지정된 매개변수에 순서대로 대입**



반환 값

- **반환 값 None**
  - 함수는 return 문을 사용하여 결과값을 반환
  - return 문이 없는 함수는 None 값을 반환하는 것과 같음
    - None은 파이썬에서 " 아무것도 없음 " 을 나타내는 특별한 값

```
>>> def greet(name):
...     print(f'안녕, {name}!')
...
```

```
>>> print(greet('모든 친구들'))
안녕, 모든 친구들!
None
```

- ❖ 출력 내용인 '안녕, 모든 친구들!'이 보이고 다음 줄 에 None도 함께 출력
- ❖ 이유는 greet( ) 자체가 반환 값이 없어 None이 반환되기 때문

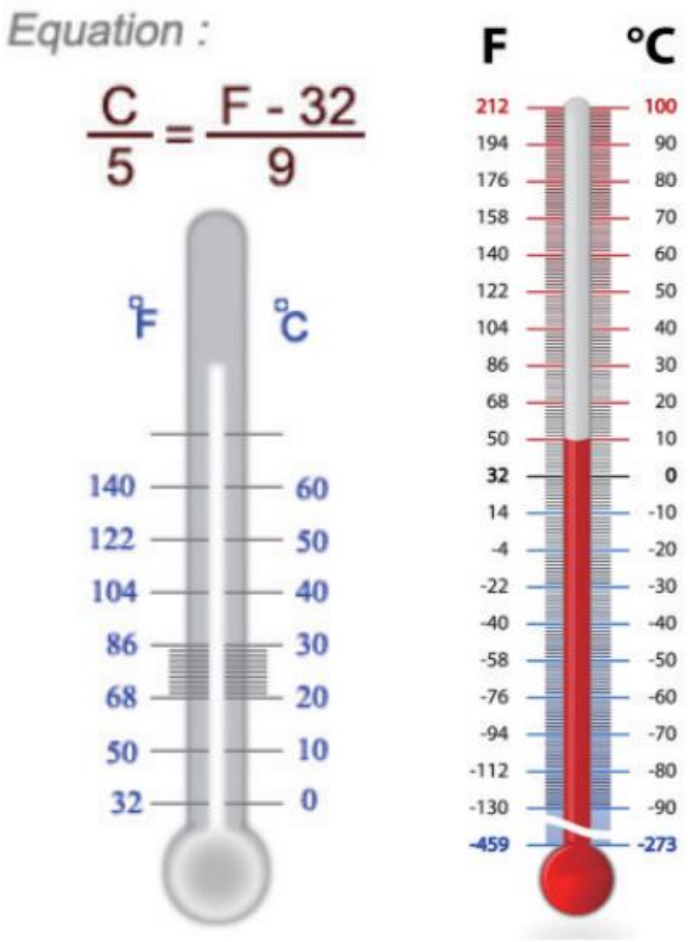
- **결과 반환 return**

```
>>> def toArea(pg):
...     return pg * 3.3
...
```

```
>>> toArea(16)
52.8
```

섭씨 온도와 화씨 온도의 관계와 변환 함수

- **3-1 코딩**



2. 간단한 암산 방법

1. **섭씨 → 화씨 (약식 변환)**
   - 섭씨에 **2를 곱하고 30을 더하기**
   - 예: 20℃ → (20 × 2) + 30 ≈ 70℉ (정확한 값: 68℉)
2. **화씨 → 섭씨 (약식 변환)**
   - 화씨에서 **30을 빼고 2로 나누기**
   - 예: 80℉ → (80 - 30) ÷ 2 ≈ 25℃ (정확한 값: 26.7℃)

Fahrenheit to Celsius formula	Celsius to Fahrenheit formula
$F = C \cdot \frac{9}{5} + 32$	$C = (F - 32) \cdot \frac{5}{9}$

람다 함수

람다 함수 == 익명 함수

- **익명 함수**
  - 간단한 함수를 한 줄로 정의하는 방법을 제공
  - lambda 키워드를 사용하여 정의
    - 주로 다른 함수에 인자로 전달되거나, 간단한 연산을 수행하는 데 사용

람다 함수 정의 구문 lambda function definition

```
lambda [parametereters] : statement_or_expression
```

parameters 옵션(없으면 생략 가능)인 매개변수는 콤마로 구분해 여러 개 가능

statement_or_expression 하나의 함수 몸체로 특정한 작업을 수행하는 문장이나 반환되는 연산식

- **람다 함수를 정의하면서 호출**

```
>>> # 람다 함수 정의와 호출
>>> (lambda : print('안녕'))()
안녕
```

다양한 내장 함수

파이썬은 다양한 내장 함수를 제공하며, 이를 활용하여 코드를 간결하고 효율적으로 작성

- **'파이썬 내장 함수 목록'으로 검색해 연결**
  - https://docs.python.org/ko/3.13/library/functions.html

- **주요 함수**
  - abs(): 숫자의 절댓값을 반환합니다.
  - pow(): 거듭제곱을 계산합니다.
  - round(): 숫자를 반올림합니다.
  - divmod(): 몫과 나머지를 튜플 형태로 반환합니다.
  - min(), max(): 최솟값과 최댓값을 찾습니다.
  - sum(): 숫자들의 합을 계산합니다.
  - bin(), oct(), hex(): 숫자를 2진수, 8진수, 16진수 문자열로 변환합니다.
  - format(): 숫자 또는 문자열을 지정된 형식으로 변환합니다.
  - sorted(): 리스트를 정렬합니다. reverse=True 인자를 사용하면 내림차순 정렬이 가능합니다.
  - reversed(): 리스트의 순서를 뒤집습니다.
  - all(): 모든 요소가 True인지 판별합니다.
  - any(): 하나라도 True인 요소가 있는지 판별합니다.
  - help() 함수를 사용하여 내장 함수에 대한 자세한 정보를 확인할 수 있습니다.

내장 함수 사용

- **도움말 함수 help()**
  - 함수 help([object])
    - 대화형 모드에서 도움말을 위한 함수

```
>>> help(abs)
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

- **절대 값 함수 abs()**
  - abs(num)
    - 정수 또는 실수 등 num의 절대 값을 반환

```
>>> abs(-7)
7
>>> abs(-3.875)
3.875
```

- **min(), max(), sum()**

```
>>> min(3.4, 5, 2)
2
>>> max(3.4, 5, 2)
5
```

진수로 변환된 문자열을 반환하는 함수 bin(), oct(), hex()

- **함수 bin(num)**
  - 정수 num을 "0b" 가 앞에 붙은 2진수 형태 문자열을 반환
- **함수 oct(num), hex(num)**
  - 각각 "0o"와 "0x"가 앞에 붙은 8진수, 16진수의 문자열을 반환

```
>>> bin(20)
'0b10100'
>>> oct(20)
'0o24'
>>> hex(20)
'0x14'
```

- **형식 함수 format()**
  - 진수를 표현하는 접두어 "0b", "0o", "0x"가 필요 없는 경우 사용
  - 두 번째 인자에 형식 지정자(format specifiers)인 'b', 'o', 'x'를 사용

```
>>> format(15) # 그대로 문자열로
'15'
>>> format(15, 'b') # 2진수 문자열로
'1111'
>>> format(15, 'o') # 8진수 문자열로
'17'
>>> format(15, 'x') # 16진수 문자열로
'f'
```

함수 sorted( )

- 내장 함수 sorted( )의 키워드 인자로 key가 없으면 기본 정렬을 수행
  - 즉, 문자열이라면 알파벳 순, 수라면 크기순으로 정렬한 결과를 반환
  - 키워드 인자 reverse=True로 내림차 순(descending order)으로 정렬

```
>>> lst = [5, 3, 9, 8]
>>> sorted(lst)
[3, 5, 8, 9]
>>> sorted(lst, reverse=True)
[9, 8, 5, 3]
```

  - 집합, 튜플도 sorted(집합)에 인자로 사용 가능

- 함수 reversed(sequence)
  - sequence 객체의 역순인 이터레이터(itearator)를 반환
  - 반환된 결과를 리스트나 튜플로 변환하여 사용하면 편리

```
>>> reversed(lst)
<list_reverseiterator object at 0x000001702CB516C0>
>>> list(reversed(lst))
[8, 9, 3, 5]
```

함수 sorted( )의 키워드 인자 key

- 키워드 인자 key의 값으로 정렬에 사용될 키를 지정
  - key=len
    - 각각의 항목 길이 함수인 len을 기준으로 오름차순으로 정렬

```
>>> # 길이를 기준으로 문자열 리스트 정렬
>>> sorted_length = sorted(words, key=len)
>>> print(sorted_length)
['fig', 'date', 'apple', 'cherry']
```

- 키워드 인자 key에 람다 함수의 사용
  - 각 항목에서 두 번째인 개발년도를 기준으로 정렬

```
# 프로그래밍 언어와 개발 연도를 튜플로 리스트 구성
pl = [('basic', 1964), ('java', 1995), ('kotlin', 2016), ('python', 1991),
('c', 1972)]

# 프로그래밍 언어의 개발순으로 정렬
print(sorted(pl, key= lambda lang: lang[1]))
```

all()과 any() 함수

리스트의 모든 요소 또는 일부 요소에 대한 조건을 판별하는 데 유용하게 활용

- **all()**
  - 이터러블의 모든 항목이 참(True)이면 True를 반환
    - 하나라도 False이면 False 반환
  - 이터러블의 항목이 비어 있으면 True를 반환
    - 이터러블이 비어 있다면, False를 발견할 기회가 없으므로 True를 반환
- **any()**
  - 하나의 항목이라도 참(True)이면 True를 반환
  - 항목이 비어 있으면 False를 반환
    - 이터러블이 비어 있다면, True를 발견할 기회가 없으므로 False를 반환

```
>>> all([3, 7 >= 4, 'py' in 'python', 3 % 2])
True

>>> all('')
True
>>> all([])
True
>>> all({})
True
```

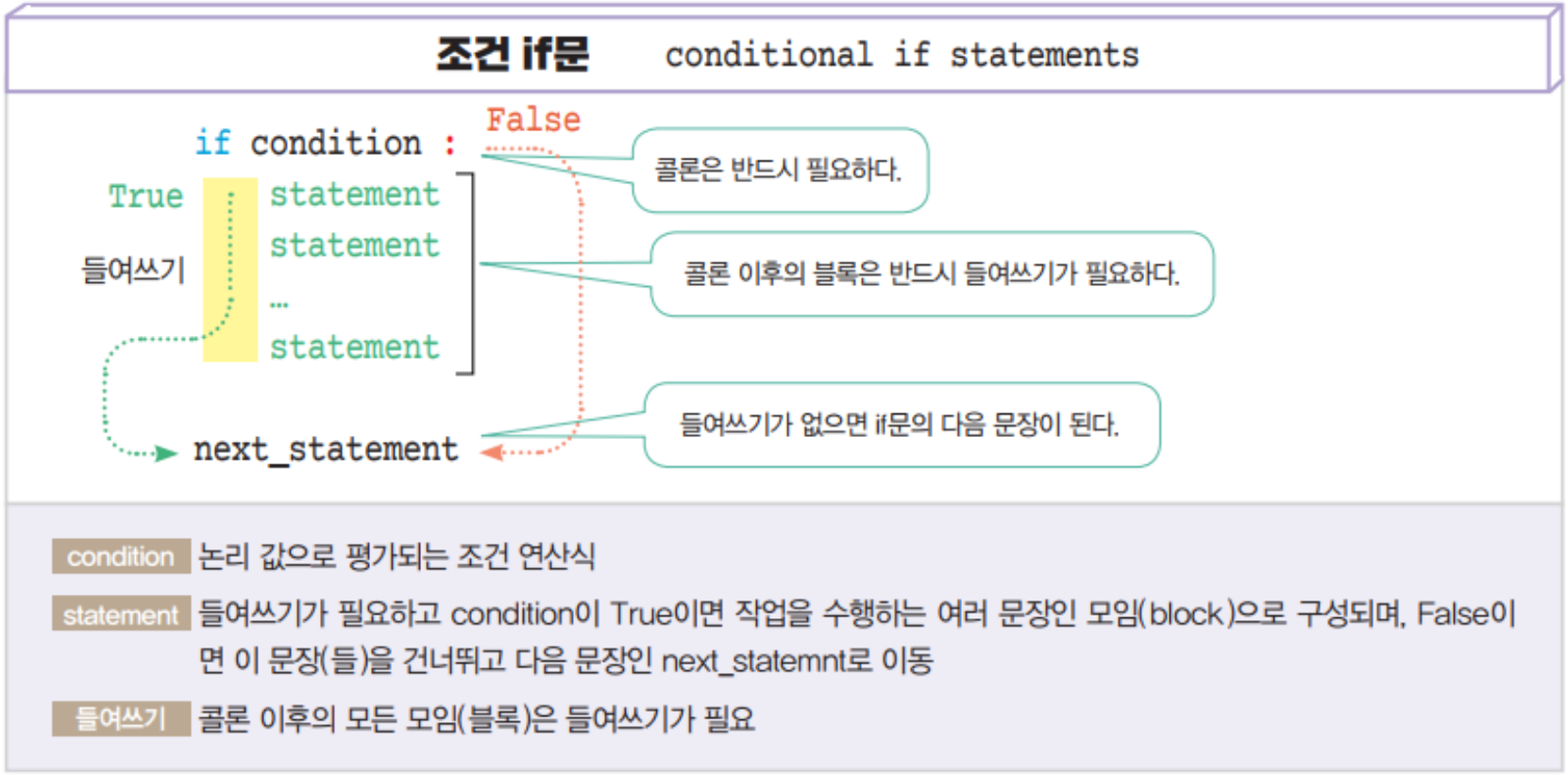
항목이 비어 있는 시퀀스는 True를 반환한다.

Section 2. 조건에 따른 선택 if

-

조건에 따른 선택 if 문

- **순차적인 코드 흐름을 바꾸는 역할**
  - 프로그램 실행 순서를 조건에 따라 변경
  - 특정 조건이 참일 때만 실행되는 코드 블록을 정의



if else 문

- 조건에 따라 다른 실행
  - if condition:
    - 조건이 True일 때 실행되는 코드 블록
  - else:
    - if 조건이 만족되지 않을 경우 실행될 코드를 지정

```
>>> temperature = 34
>>> if 33 <= temperature:
...     print(f'폭염에 주의하시오.')
... else:
...     print(f'기온 {temperature}도입니다.')
...
폭염에 주의하시오.
```

if elif else 문

- **다중 조건 처리**
  - 여러 개의 조건을 순차적으로 검사
    - 첫 번째 조건(if)이 False이면 다음 조건(elif)을 검사
    - 모든 조건이 False일 경우 else 블록이 실행

```
>>> age = 20
>>> if age >= 70:
...     print('노인')
... elif age >= 20:
...     print('성인')
... elif age >= 12:
...     print('청소년')
... elif age >= 6:
...     print('어린이')
... elif age >= 1:
...     print('유아')
... else:
...     print('영아')
...
성인
```

먼저 검사한 age >= 70 조건이 False인 경우, 이 age >= 20 조건을 실행하므로 결과적으로 조건 70 > age >= 20을 검사하는 의미가 됨.

마찬가지로 조건 6 > age >= 1을 검사하는 의미가 됨.

조건 연산자 if else

- **파이썬의 조건 연산자**
  - 조건 condition이 True이면 exp1, False이면 exp2를 반환

조건 연산자 conditional operator or ternary operator

```
exp1 if condition else exp2
```

조건 condition이 True로 반환되는 exp1이 맨 앞에 위치

condition 논리 값으로 평가되는 조건 연산식

exp1 condition이 True면 실행되거나 반환되는 연산식이나 문장

exp2 condition이 False면 실행되거나 반환되는 연산식이나 문장

```
>>> is_sunny = True
>>> play = '놀이공원 가기' if is_sunny else '집에서 놀기'
>>> print(play)
놀이공원 가기
```

한 줄에 쓰는 조건문

- **코드 간결화**
  - 간단한 조건문을 한 줄로 표현: 가독성을 높이고 코드를 간결
  - 간단한 구문에 사용

한 줄 if one line if

```
if condition : statement1; statement2; …; statement(n)
```

condition 논리 값으로 평가되는 조건 연산식

statement(n) 한 줄에 여러 문장을 세미콜론으로 구분해 코딩 가능하며, 마지막 문장은 세미콜론 생략 가능

- **일반적으로 전통적인 여러 줄의 if elif else문을 사용하는 것이 보편적**
  - 복잡한 if elif else문에서 한 줄에 여러 문장을 사용하면 가독성이 나빠질 수 있음

Section 3. 반복문 for in과 while

-

반복문 for in

주어진 컬렉션의 각 항목에 대해 코드 블록을 반복 실행하는 데 사용

- **컬렉션**
  - list, tuple, dict, set, frozenset, range 등 가능
- **for 루프는 컬렉션의 모든 항목을 순차적으로 처리**

for in 반복문 for in loop statements

```
for variable in iterable :
    statement(s)
[else :
    additional_statement(s)]
```

variable 매 반복마다 모음인 iterable에서 하나의 요소인 항목을 저장해 반복 수행

iterable 반복 가능한(iterable) 자료형으로는 str, list, tuple, dict, set, frozenset, range가 있으며, 반복할 여러 자료의 모음(colletion)으로 자신의 항목을 한 번에 하나씩 전달할 수 있는 기능인 __iter( )__ 메서드를 지원하는 객체가 위치

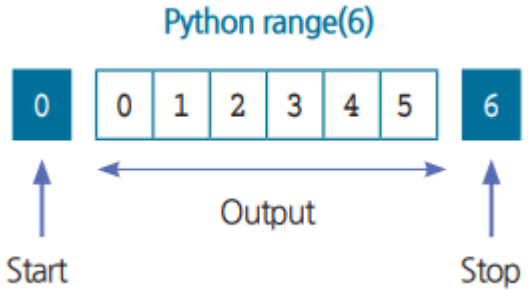
statement(s) 반복 몸체라는 모임(블록)으로 매 반복마다 한 번 실행되는 문장(들)이며, 콜론 이후의 모임(블록)이므로 들여쓰기가 필요한 여러 문장으로 구성

else 옵션으로 모음인 iterable에서 모든 항목에 대해 반복이 실행된 이후에만 실행되는 블록 additional_statement(s)

자료형 range와 반복 for in

- **정수형 범위 반환 range() 함수 활용**
  - range(): 정수 시퀀스를 생성하는 데 사용되며, 시작 값, 종료 값, 단계 값을 지정

  - range(stop)
  - range(start, stop)
  - range(start, stop, step)
    - Start: 시작 값 (포함). 기본값은 0
    - Stop: 종료 값 (제외).
    - Step: 단계 값, 기본값은 1



- **반복문에서 활용**

```
>>> for i in range(5):
...     print(i, end = ' ')
...
0 1 2 3 4
```

내장 함수 range(5)처럼 인자가 1개이면 반복이 5회 수행한다고 생각하자. 다만 반복 항목 시작이 1부터가 아니라 0부터이므로 마지막도 5가 아니라 4라는 것을 잊지 말자.

while 반복문

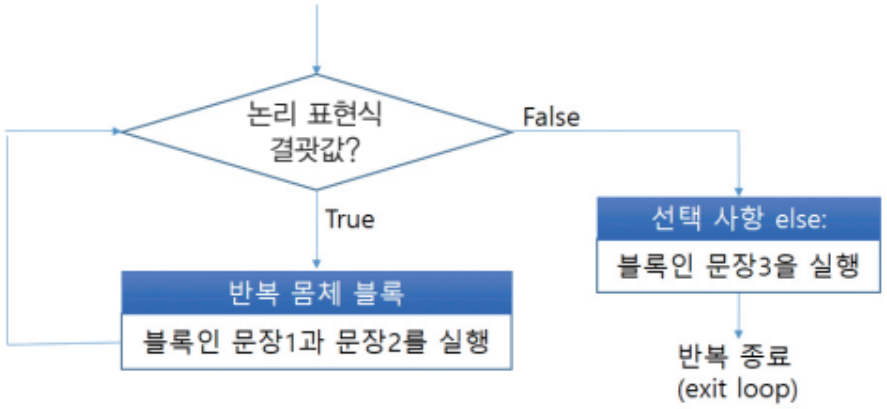
- **조건식이 True인 동안 코드 블록을 반복 실행**
  - 조건식이 False이면 else: 블록 실행 후 종료

```
while condition :
    문장1
    문장2
else:
    문장3
```

논리 표현식이 위치하고 반드시 콜론이 필요하다.

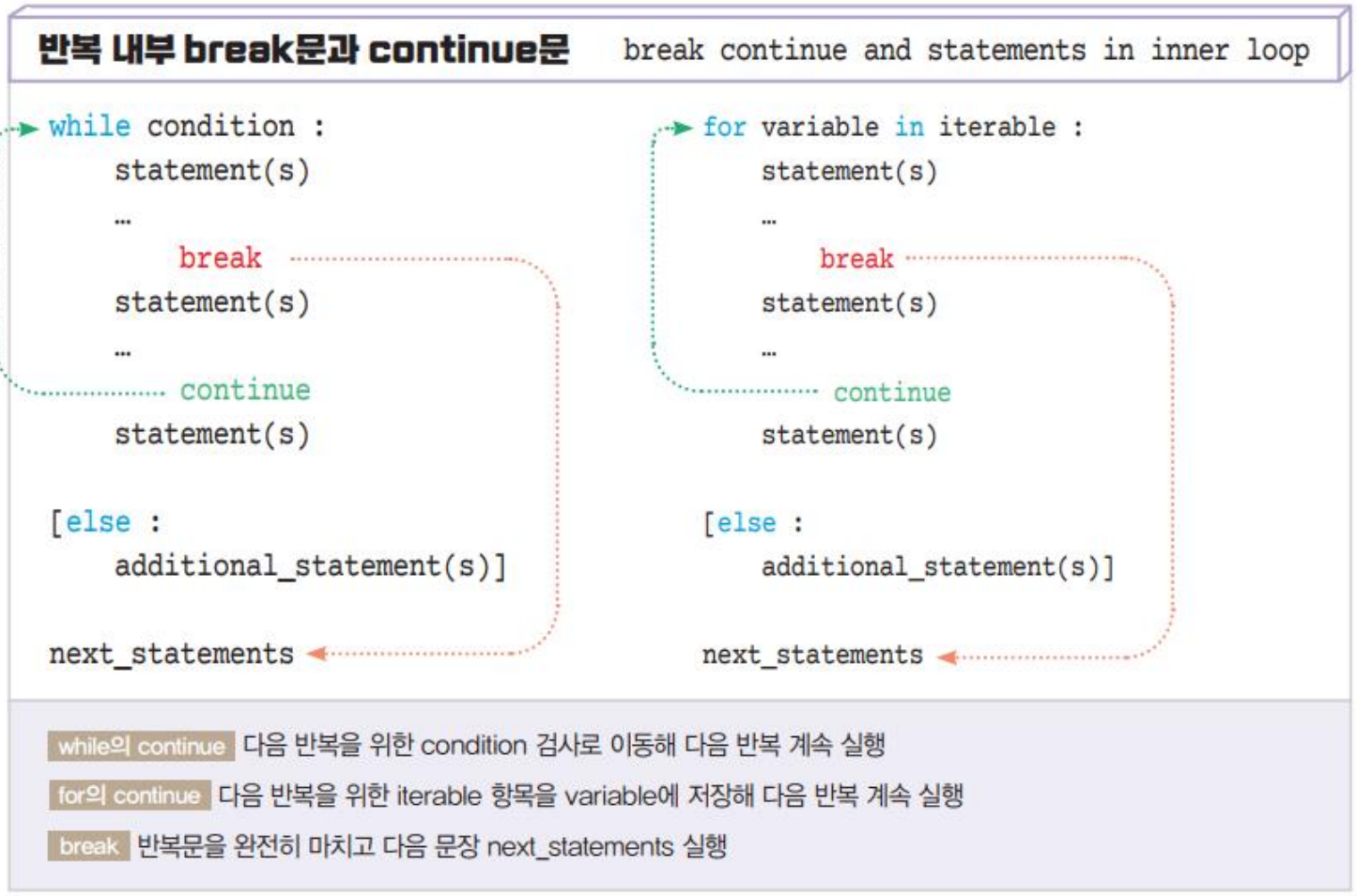
블록인 문장1과 문장2를 반복 몸체라고 하며, condition이 True이면 실행되는 반복 단위이다.

else: 이후의 블록인 문장3은 논리 표현식이 False가 되어 반복이 종료된 마지막에 실행된다.



반복 제어 break와 continue

- **break: 현재 루프를 즉시 종료**
- **continue: 현재 반복을 건너뛰고(이후 문장 미실행) 다음 반복으로 넘어감**



반복 for를 사용한 구구단

- 먼저 구구단의 7단을 출력
  - 변수 i에 7을 저장한 후 1에서 9까지의 수를 반복하기 위해
    - 반복 for에서 변수 j와 함수 range(1, 10) 객체를 사용

```
>>> i = 7
>>> for j in range(1, 10):
...     print(f'{i} * {j} = {i*j} ', end=' ')
...
7 * 1 = 7  7 * 2 = 14  7 * 3 = 21  7 * 4 = 28  …(중략)…  7 * 7 = 49  7 * 8 = 56
7 * 9 = 63
```

- 3-12 코딩
  - 전형적인 구구단 출력

```
for i in range(2, 10):
    for j in range(1, 10):
        print(f'{i} * {j} = {i*j:2d}', end=' ')
    print()
```