






함수

 PDF	비어 있음
 비교	함수 기초, 내장 함수
 숫자	12
 실습문제	비어 있음
 실습문제답안	비어 있음

01. 함수 (function)

01-01. 함수 개요

01-01-01. 함수란

- 💡 특정 기능을 하기 위한 코드의 집합을 의미한다. 함수를 정의해 놓으면 해당 기능이 필요한 위치에서 함수를 호출하여 간편하게 사용할 수 있다.

01-01-02. 함수 표현식

- def 키워드를 사용하여 함수를 정의한다.
- 이때, 함수 이름 뒤에 소괄호와 콜론을 작성하며 함수에서 실행할 내용은 "반드시" 들여쓰기 한다.

```
def 함수명(): (함수가 실행하는 코드 내용)
```

- 함수 선언과 호출 예시

```
def basic_function(): print("Hello World") basic_function() # Hello World
```

01-02. 매개변수

01-02-01. 파라미터

- 함수 생성 시 함수명 옆에 붙이는 소괄호 안에 매개변수로 전달받을 값의 변수명을 넣을 수 있다. 이를 파라미터라고 한다.

```
def greeting(name, greet): print(name, ":", greet) greeting("다람쥐", "감사합니다다람쥐~") # 다람쥐 : 감사합니다다람쥐~
```

- 함수를 정의할 때 매개변수의 기본값을 지정해 줄 수 있다.

함수 호출 시 인자를 보내지 않으면 함수 선언부에서 미리 지정한 기본값을 사용하여 함수를 실행한다.

```
def greeting(name, greet="처음 뵙겠습니다!!!"): print(name, ":", greet) greeting("다람쥐") # 다람쥐 : 처음 뵙겠습니다!!!
```

01-02-02. 인자

- 함수 호출 시 함수명에 소괄호를 붙여 호출하며, 이때 소괄호 안에 전달하는 값은 인자라고 한다.

- 함수 인자를 보내는 방식은 다음 2가지가 있다.
 1. 위치 인자: 위치로 매칭하는 방법
 2. 키워드 인자: 매개변수 이름으로 매칭하는 방법
 - 이때, 위치 인자를 먼저 사용하면 뒤에 키워드 인자를 쓸 수 있다.
 - 단, 위치 인자와 키워드 인자를 함께 사용할 때 키워드 인자를 먼저 사용하는 것은 불가하다.

```
def information(name, color): print(name + '의 최애 색상', color, sep=' : ')
```

```
# 위치 인자 information('tiger', 'orangered') # tiger의 최애 색상 : orangered
```

```
# 키워드 인자 information(color='black', name='monkey') # monkey의 최애 색상 : black i
nformation('monkey', color='black') # monkey의 최애 색상 : black # information(color
='black', 'monkey') # 불가능 # information(name='monkey', 'black') # 불가능
```

01-03. 반환값

01-03-01. return

- 함수 블록 내부에서 return 키워드를 만나면 함수의 작동이 중단된다.
- return 키워드 뒤에 변수 또는 값이 오면, 해당 자료를 함수 호출 위치에 반환하며 중단된다.

```
def introduce(name): return f'안녕하세요 {name} 감사 인사드립니다~!' result =
introduce('사슴') print(result) # 안녕하세요 사슴 감사 인사드립니다~!
```

- return 키워드는 선택적으로 사용할 수 있다. (반드시 작성하지 않아도 된다.)

▼ 추가 설명 (return 값 여러 개)

```
# 두 개 이상의 값을 리턴할 수 있다. (튜플 사용) def calc2(a,b): return a+b, a-b, a*b,
a/b print(calc2(7,3)) result = calc2(8,3) print(result) print(result[0]) # 리턴값을
각각의 변수에 담기 plus, minus, multiply, divide = calc2(8,3) print(plus, minus, multi
ply, divide)
```

01-04. 패키지과 언패키지

01-04-01. 패키징

- 위치 인자가 많을 때 *을 붙여서 패키징하여 하나의 객체로 처리할 수 있다.

```
def add_many(*args): result = 0 for i in args: result = result + i return result add_
many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) # 55
```

- 키워드 인자인 경우에는 * 두 개(**)를 붙여서 작성할 수 있다.

01-04-02. 언패키징

- 패키지와 반대되는 개념으로 여러 개의 객체를 포함하는 하나의 객체를 풀어준다.

```
def sum(a, b, c): return a + b + c numbers = [1, 2, 3] # sum(numbers) # 배열로 하나만
넘겨주면 에러 발생 print(sum(*numbers)) # 6
```

01-05. 내장 함수

01-05-01. 내장 함수란

💡 파이썬이 제공하여 직접 작성하지 않고도 사용할 수 있는 함수이다.

01-05-02. 내장 함수 종류

- 내장 함수의 종류는 다양하지만 자주 쓰는 것, 지금까지 썼던 것은 다음과 같은 것이 있다.

1. `print()` : 콘솔 출력 기능을 하는 함수
2. `input()` : 콘솔을 통해 사용자 입력을 받아 문자열로 반환하는 함수
3. `type()` : 인자로 전달된 값의 자료형을 반환하는 함수
4. `len()` : 인자로 전달된 시퀀스 자료형의 요소의 갯수(길이)를 반환하는 함수
5. `int()`, `float()`, `str()`, `list()`, `dict()` : 인자로 전달된 자료형을 특정 자료형으로 형 변환하여 반환하는 함수
6. `range()` : 전달받은 인자에 따라 일정 범위의 숫자 시퀀스를 생성하여 반환하는 함수
7. `min()`, `max()` : 인자로 전달된 시퀀스의 요소 중 가장 크거나 작은 값을 반환하는 함수
8. `sum()` : 인자로 전달된 시퀀스 내 모든 요소의 합계를 반환하는 함수
9. `abs()` : 절댓값을 반환하는 함수
10. `round()` : 인자로 전달된 숫자를 반올림하는 함수
11. `map()` : 함수와 시퀀스를 인자로 받아, 시퀀스의 각 요소에 함수를 적용한 결과를 반환하는 함수
12. `filter()` : 함수와 시퀀스를 인자로 받아, 시퀀스에서 함수의 조건을 만족하는 요소만 반환하는 함수
13. `pow()` : 인자의 첫 번째 숫자를 두 번째 숫자로 거듭 제곱한 값을 반환하는 함수

01-06. 람다 (lambda)

01-06-01. 람다란

💡 람다는 일회성의 간단한 함수를 정의할 때 유용하게 사용할 수 있는, 함수를 보다 간단하고 쉽게 선언하는 방법이다. 매개변수로 함수를 전달해야 할 때, 함수 구문을 작성하는 것이 번거롭고 코드가 지나치게 길어지는 경우 사용한다. 즉, 함수 기능을 매개변수로 전달하는 코드를 더 효율적으로 작성할 수 있도록 한다.

01-06-02. 람다 표현식

변수명 = `lambda` (매개변수) : (함수 실행 내용)

01-06-03. 람다 사용 예시

1. 람다 기본

```
add = lambda x, y: x + y
print(add(3, 5)) # 8
```

2. 람다 함수를 고차함수의 인자로 전달 (배열 요소 처리)

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x**2, numbers)
print(list(squared_numbers)) # [1, 4, 9, 16, 25]
```

3. 람다 함수를 고차함수의 인자로 전달 (배열 요소 필터링)

```
numbers = [1, 2, 3, 4, 5, 6] even_numbers = filter(lambda x: x % 2 == 0, numbers) print(list(even_numbers)) # [2, 4, 6]
```

4. 배열 정렬 기준을 람다 함수로 전달

```
points = [(1, 2), (3, 1), (5, -1)] sorted_points = sorted(points, key=lambda x: x[1]) print(sorted_points) # [(5, -1), (3, 1), (1, 2)]
```