

4장. 함수(Function)



함수(Function)

- **함수란?**

- 특정 작업을 수행하는 코드 블록을 의미한다.
- 함수를 사용하면 코드를 재사용하고, 프로그램을 더 모듈화하며, 가독성을 높일 수 있다.
- **def** 키워드를 사용함

- **함수(Function)의 종류**

- **사용자 정의 함수** – 사용자가 직접 만든 함수
- **내장 함수** – 파이썬이 제공하는 함수

함수(Function)

- 사용자 정의 함수의 형태

def 함수 이름():
함수의 내용

return 반환값이 없는 함수

def 함수 이름(매개 변수):
함수의 내용
return 반환값

return 반환값이 있는 함수

사용자 정의 함수

- return이 없는 함수

```
def greet():
    print("Hello, World!")
```

함수의 정의

```
def greet2(name):
    print(f"Hello, {name}!")
```

```
greet() # Hello, World!
greet2("Elsa") # Hello, Elsa!
greet2("명제") # Hello, 명제!
```

함수의 호출

사용자 정의 함수

- return이 없는 함수

```
def get_gugu(dan):
    for i in range(1, 10):
        # print(dan, 'x', i, '=', (dan * i))
        print(f"{dan} x {i} = {dan*i}")
```

```
# 구구단 호출
get_gugu(6)
```

```
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
```

함수 정의하고 호출하기

- `return`이 있는 함수

```
def message():
    return "Good Luck!"

def square(x):
    return x * x

def add(a, b):
    return a + b

print("5의 제곱:", square(5))  # 5의 제곱: 25
result = add(3, 5)
print("3 + 5 =", result)  # 3 + 5 = 8
```

도형의 면적 계산

- 도형의 면적을 계산하는 함수 정의와 사용

```
def square_area(w, h):
    return w * h

def triangle_area(b, h):
    return (b * h) / 2

print("사각형 면적:", square_area(5, 4)) # 사각형 면적: 20
print("삼각형 면적:", triangle_area(5, 4)) # 삼각형 면적: 10.0
```

함수의 반환값

- **return** 반환값 – 여러 개인 경우

```
# 구조 할당
a, b = 10, 20    #변수
print("a=", a)
print("b=", b)

x, y = (10, 20) #튜플
print("x=", x)
print("y=", y)
```

함수의 반환값

- **return** 반환값 – 여러 개인 경우

```
# 반환값 - 여러 개
def calculate(x, y):
    return x + y, x - y

# 구조 분해 할당
sum, diff = calculate(10, 4)
print("합:", sum) # 합: 14
print("차:", diff) # 차: 6
```

간단한 규칙기반 챗봇

▪ 챗봇(chatbot) 함수 만들기

```
print(" *** 간단한 규칙 기반 챗봇 ***")
from datetime import datetime

def my_chatbot():
    while True:
        user_input = input("챗봇에게 질문하세요 (종료하려면 'exit' 입력): ")
        if user_input.lower() in ["exit", "quit"]:
            print("챗봇을 종료합니다.")
            break
        elif "안녕" in user_input:
            print("챗봇: 안녕하세요! 무엇을 도와드릴까요?")
        elif "이름" in user_input:
            print("챗봇: 제 이름은 챗봇입니다.")
        elif "시간" in user_input:
            now = datetime.now()
            print(f"챗봇: 현재 시간은 {now.hour}시 {now.minute}분입니다.")
        else:
            print("챗봇: 죄송합니다, 잘 모르겠어요.")

my_chatbot() # 챗봇 실행(함수 호출)
```

매개변수로 리스트 전달

- 리스트를 매개변수로 전달하여 합계, 평균 계산하기

```
def calc_sum(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

def average(numbers):
    total = calc_sum(numbers)
    return total / len(numbers)

num_list = [1, 2, 3, 4, 5]
print("리스트의 합:", calc_sum(num_list)) # 리스트의 합: 15
print("리스트의 평균:", average(num_list)) # 리스트의 평균: 3.0
```

매개변수로 리스트 전달

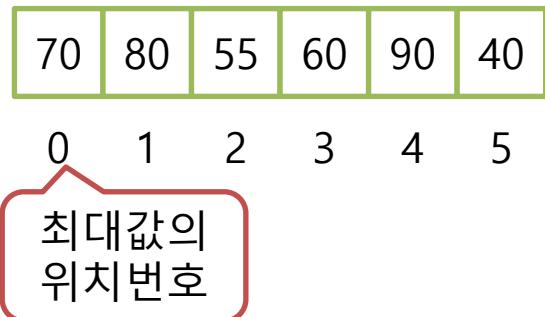
- 리스트를 매개변수로 새로운 리스트 만들기

```
def times(a):
    a2 = [] #복사할 빈 리스트 생성
    for i in a:
        a2.append(4 * i)
    return a2

# 리스트를 4의 배수로 저장
arr = [1, 2, 3, 4]
# 함수 호출
arr2 = times(arr)
print(arr2)
```

매개변수로 리스트 전달

- 최대값과 최대값의 위치 구하기



- 첫번째 숫자 70을 최대값으로 기억한다.
- 두 번째 숫자 80을 최대값 70과 비교하여 최대값은 80이 된다.
- 계속 다음 숫자와 비교과정을 반복하여 최대값을 결정한다.

매개변수로 리스트 전달

- 최대값과 최대값의 위치 구하기

```
# 최대값 구하기
def find_max(a):
    max_v = a[0]

    for i in a:
        if max_v < i:
            max_v = i

    return max_v

# 최대값 위치
def find_max_idx(a):
    max_idx = 0
    n = len(a)

    for i in range(1, n):
        if a[max_idx] < a[i]:
            max_idx = i

    return max_idx
```

```
v = [70, 80, 55, 60, 90, 40]
max_v = find_max(v)
max_idx = find_max_idx(v)

print("최대값 : ", max_v)
print("최대값의 위치 : ", max_idx)
```

실습 문제 – 사용자 정의 함수

절대값을 계산하는 함수를 구현하세요.

(10을 입력하면 10이 출력되고, -10을 입력해도 10이 출력됨)

☞ 실행 결과

절댓값: 10

변수의 메모리 영역

■ 변수의 메모리 영역 분석

데이터 영역 : 전역 변수가 저장되는 영역



고정된 영역
(전역, 정적 변수 등)

스택 영역 : 매개 변수 및 중괄호(블록)
내부에 정의된 변수들이
저장되는 영역



Stack
(지역, 매개 변수)

힙 영역 : 동적으로 메모리를 할당하는
변수들이 저장되는 영역
(클래스 객체 - 인스턴스 변수)



heap
(객체)

변수의 유효 범위 - 전역변수

- **전역 변수(global variable)의 유효 범위**

전역 변수는 메인 함수 영역에 선언하여 사용하고, 영향 범위가 전체로 미친다. 프로그램이 종료되면 메모리에서 소멸한다.

- **지역 변수(local variable)의 유효 범위**

지역변수는 함수나 명령문(조건, 반복)의 블록 안에서 생성되며 블록{ }을 벗어나면 메모리에서 소멸한다.

변수의 유효 범위 - 지역변수

- 변수의 유효 범위

```
def get_price():
    price = 1000 * quantity #price - 지역 변수
    print(f"{quantity}개에 {price}원 입니다.")

# 전역 변수
quantity = 2 # 수량
get_price() # 호출(사용)

# 변수 출력
# print(price) # 소멸된 변수임(오류 발생)
```

변수의 유효 범위 - 정적변수

▪ 정적 변수의 유효 범위

지역 변수에 **global** 키워드를 붙이면 정적 변수가 되어 값을 유지하고, 프로그램이 종료되면 메모리에서 소멸한다.

```
def click():
    global x #지역 변수가 전역변수로 됨
    # x = 0 #지역 변수
    x = x + 1

    print(f"x = {x}")

x = 0 #전역 변수

# 함수 호출
click()
click()
click()
```

x = 1
x = 2
x = 3

배송비 계산 프로그램

- 상품 가격이 40000원 미만이면 배송비 3000원을 포함하고, 40000원 이상이면 배송비를 포함하지 않는 프로그램 작성.

```
def get_price(unit_price, quantity):
    delivery_fee = 3000 # 배송비
    price = unit_price * quantity # 가격 = 단위당 가격 * 수량
    if price < 40000:
        price += delivery_fee
    else:
        price
    return price

# 메인 영역 - 함수 호출
price1 = get_price(25000, 2)
price2 = get_price(30000, 1)

print("상품1 가격 : " + str(price1) + "원")
print(f"상품1 가격 : {price1}원")
print("상품2 가격 : " + str(price2) + "원")
print(f"상품2 가격 : {price2}원")
```

상품1 가격 : 50000원
상품1 가격 : 50000원
상품2 가격 : 33000원
상품2 가격 : 33000원

함수의 기본 매개변수

- 기본 매개변수

매개변수를 초기화하여 선언하고 함수 호출시 매개변수를 생략하면 기본 값으로 출력된다.

def 함수 이름(변수1, 변수2=1):

코드블럭

함수의 기본 매개변수

● 기본 매개변수

```
# 버스 요금 계산 함수
def take_bus(fare=1200):
    print(f"버스 요금은 {fare}원입니다.")

take_bus()  # 버스 요금은 1200원입니다.
take_bus(1500)  # 버스 요금은 1500원입니다.

# 거듭제곱 계산 함수
def power(base, exponent=2):
    return base ** exponent

print("3의 제곱:", power(3))  # 3의 제곱: 9
print("2의 세제곱:", power(2, 3))  # 2의 세제곱: 8
```

함수의 가변 매개변수

● 가변 매개변수

매개변수의 입력값이 정해지지 않고 변경해야 할 때 사용하는 변수이다.
변수 이름 앞에 *를 붙인다.

def 함수 이름(*변수):
코드블럭

함수의 가변 매개변수

● 가변 매개변수

```
# 평균을 계산하는 함수 정의
def calc_avg(*number):
    total = 0 #합계
    for i in number: #number(iterable-리스트, 튜플)
        total += i; #total = total + i
    avg = total / len(number) # 평균 = 합계 / 개수
    return avg

print(calc_avg(1, 2)) #1.5

average = calc_avg(1, 2, 3, 4)
print(average) #2.5
```

lambda Expressions(람다식)

● 람다함수(Lambda Function)

- 이름 없는(익명) 한 줄 함수이다.
- lambda 키워드를 사용한다.
- return 문을 사용하지 않는다.

lambda 매개변수 : 반환할 표현식

map(람다함수, 순서열) : 순서열의 모든 값에 연산 적용

filter(람다함수, 순서열) : 조건에 맞는 값만 순서열에서 추출

lambda Expressions(람다 표현식)

● 람다함수(Lambda Function)

```
# 일반 함수  
def add(x, y):  
    return x + y  
  
print(add(1, 2))
```



```
#lambda 함수  
add = lambda x, y : x + y  
  
print(add(1, 2))
```

```
# 제곱 계산 lambda 함수  
square = lambda x: x ** 2  
  
print(square(8)) # 64
```

lambda(람다) 프로그래밍

- map(), filter() 함수와 함께 사용

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

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

# filter 함수와 lambda 함수
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers)) # [2, 4]
```

내장 함수(Built in Function)

❖ 내장 함수(Built in Function)

특정한 기능을 수행하는 프로그램의 일부분을 함수(Function)라 한다.

Built-in Functions			
A abs() aiter() all() anext() any() ascii()	E enumerate() eval() exec()	L len() list() locals()	R range() repr() reversed() round()
B bin() bool() breakpoint() bytearray() bytes()	F filter() float() format() frozenset()	M map() max() memoryview() min()	S set() setattr() slice() sorted()
C callable() chr() classmethod()	H hasattr() hash() help()	N next() object() oct() open() ord()	O super()
			T tuple() type()

내장 함수(Built in Function)

❖ 내장 함수(Built in Function)

함수	설명	사용 예
sum(iterable)	리스트나 튜플의 모든 요소의 합을 반환	sum([1, 2, 3]) 6 sum((1.2.3)) 6
max(iterable)	리스트나 튜플의 최대값을 반환	max([1, 2, 3]) 3 max((1.2.3)) 3
round(n, digit)	숫자를 입력받아 반올림하여 돌려줌	round(4.6) 5 round(4.4) 4
eval(expression)	문자열 표현식을 숫자로 변환	eval('1+2') 3
list(s)	반복 가능한 문자열을 입력받아 리스트로 반환	list("python") ['p', 'y', 't', 'h', 'o', 'n']

내장 함수(Built in Function)

❖ 내장 함수(Built in Function)

```
a = [1, 2, 3, 4]
b = (1, 2, 3, 4)

print(sum(a)) # 합계
print(sum(b))

print(max(a)) # 최대값
print(max(b))

print(min(a)) # 최소값
print(min(b))
```

내장 함수(Built in Function)

❖ 내장 함수(Built in Function)

```
# 반올림  
print(round(2.74)) #3  
print(round(2.14)) #2  
  
# 소수 자리수  
x = 706.351  
print(round(x, 1)) # 소수첫째자리, 706.4  
print(round(x, 0)) # 706.0  
print(round(x)) # 정수 706  
print(int(round(x, -1))) # 일의 자리 710
```

내장 함수(Built in Function)

❖ 내장 함수(Built in Function)

```
# 문자열 표현식을 숫자로 변환
print("1 + 2")
print(eval("1 + 2")) # 3

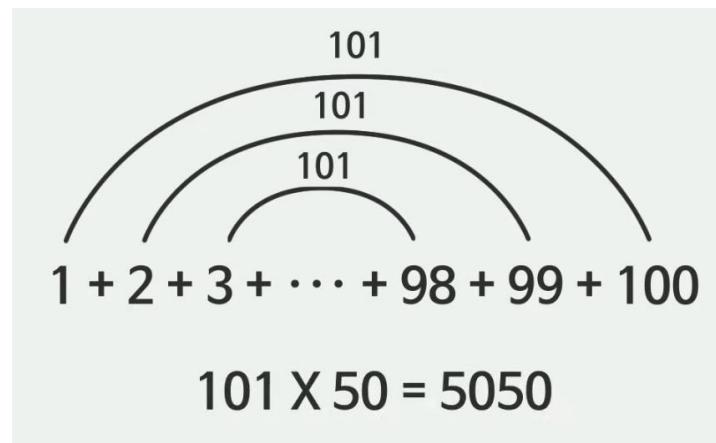
# 리스트(list), 튜플(tuple), 집합(set), 딕셔너리(dict) 변환
print(list("Hello")) # ['H', 'e', 'l', 'l', 'o']
print(tuple([1, 2, 3])) # (1, 2, 3)
print(set([1, 2, 2, 3, 3, 3])) # {1, 2, 3}
print(dict([('a', 1), ('b', 2)])) # {'a': 1, 'b': 2}
```

내장 함수(Built in Function)

❖ 합계 함수 만들고 비교하기

$$1 + 2 + 3 + \cdots + n \longrightarrow \text{방법 1}$$

$$n \times (n + 1) \div 2 \longrightarrow \text{방법 2}$$



내장 함수(Built in Function)

❖ 합계 함수 만들고 비교하기

```
...
    - 계산 복잡도 비교
    sum_n(n) - 덧셈 n번 -> O(n) : Big O 표기
    sum_n2(n) - 덧셈, 곱셈, 나눗셈(총 3번) -> O(1)
...

def sum_n(n):
    total = 0
    for i in range(1, n+1):
        total = total + i
    return total

def sum_n2(n):
    total = (n * (n + 1)) / 2
    return int(total)
```

내장 함수(Built in Function)

❖ 합계 함수 만들고 비교하기

```
# 호출  
print("합계1:", sum_n(10)) #55  
print("합계2:", sum_n2(10)) #55  
  
# 내장 함수 - sum()  
v = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
print("합계3:", sum(v)) #55
```

내장 함수(Built in Function)

❖ 거듭 제곱 함수 만들고 비교하기

```
def my_pow(x, y):
    num = 1
    for i in range(0, y):
        num = num * x
    return num
...
x=2, y=4 일때
반복
i=0, num = 1 * 2
i=1, num = 2 * 2
i=2, num = 4 * 2
i=3, num = 8 * 2
i=4, 반복 종료
...
```

```
print(my_pow(2, 4)) #16
print(my_pow(3, 3)) #27

# 내장 함수 - pow()와 비교
print(pow(2, 4)) #16
print(pow(3, 3)) #27
```