

# 합수 만들기

# 목차

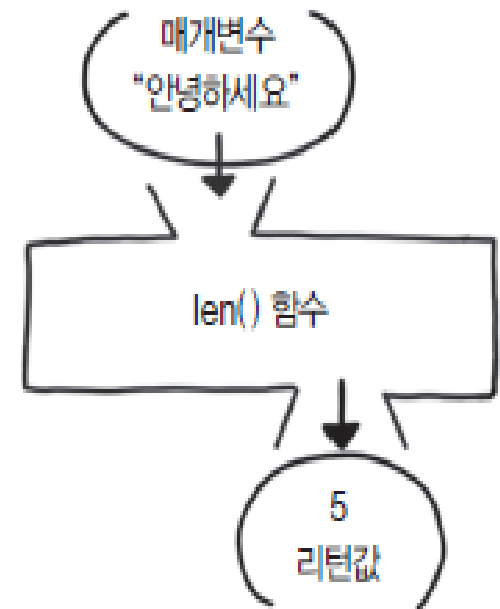
- 시작하기 전에
- 함수의 기본
- 함수에 매개변수 만들기
- 가변 매개변수
- 기본 매개변수
- 키워드 매개변수
- 리턴
- 기본적인 함수의 활용
- 키워드로 정리하는 핵심 포인트
- 확인문제

**[핵심 키워드]** : 호출, 매개변수, 리턴값, 가변 매개변수, 기본 매개변수

**[핵심 포인트]**

함수들을 어떻게 만들고 활용하는지 살펴본다.

- 함수를 호출
  - 함수 사용
- 매개변수
  - 함수 호출 시 괄호 내부에 넣는 여러 가지 자료
- 리턴값
  - 함수를 호출하여 최종적으로 나오는 결과



- 함수 = 코드의 집합

```
def 함수 이름():  
    문장
```

```
01 def print_3_times():  
02     print("안녕하세요")  
03     print("안녕하세요")  
04     print("안녕하세요")  
05  
06 print_3_times()
```

실행결과

안녕하세요  
안녕하세요  
안녕하세요

- 매개변수

```
def 함수 이름(매개변수, 매개변수, ...):  
    문장
```

print

print

```
def print(value, ..., sep=' ' *  
        ', end='\n', file=sys.stdout,  
        t, flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream);  
defaults to the current sys.stdout.

sep: string inserted between values,  
default a space.

end: string appended after the last

```
01 def print_n_times(value, n):  
02     for i in range(n):  
03         print(value)  
04  
05 print_n_times("안녕하세요", 5)
```

KID 실행결과

안녕하세요  
안녕하세요  
안녕하세요  
안녕하세요  
안녕하세요

- 매개변수와 관련된 TypeError
  - 매개변수를 넣지 않은 경우

```
def print_n_times(value, n): → 매개변수를 2개 지정했는데
    for i in range(n):
        print(value)

# 함수를 호출합니다.
print_n_times("안녕하세요") → 하나만 넣었습니다.
```

## 오류

```
Traceback (most recent call last):
  File "test5_01.py", line 6, in <module>
    print_n_times("안녕하세요")
TypeError: print_n_times() missing 1 required positional argument: 'n'
```



- 매개변수를 더 많이 넣은 경우

```
def print_n_times(value, n): → 매개변수를 2개 지정했는데
    for i in range(n):
        print(value)

# 함수를 호출합니다.
print_n_times("안녕하세요", 10, 20) → 3개를 넘었습니다.
```

## 오류

```
Traceback (most recent call last):
  File "test5_02.py", line 6, in <module>
    print_n_times("안녕하세요", 10, 20)
TypeError: print_n_times() takes 2 positional arguments but 3 were given
```

- 가변 매개변수

- 매개변수를 원하는 만큼 받을 수 있는 함수

```
def 함수 이름(매개변수, 매개변수, ..., *가변 매개변수):  
    문장
```

- 제약
  - 가변 매개변수 뒤에는 일반 매개변수 올 수 없음
  - 가변 매개변수는 하나만 사용할 수 있음

## 예시 - 가변 매개변수 함수

```
01 def print_n_times(n, *values):  
02     # n번 반복합니다.  
03     for i in range(n):  
04         # values는 리스트처럼 활용합니다.  
05         for value in values:  
06             print(value)  
07         # 단순한 줄바꿈  
08         print()  
09  
10 # 함수를 호출합니다.  
11 print_n_times(3, "안녕하세요", "즐거운", "파이썬 프로그래밍")
```

**실행결과**

안녕하세요  
즐거운  
파이썬 프로그래밍

안녕하세요  
즐거운  
파이썬 프로그래밍

안녕하세요  
즐거운  
파이썬 프로그래밍

- 기본 매개변수

- 매개변수 값 입력하지 않았을 경우 매개변수에 들어가는 기본값

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
01 def print_n_times(value, n=2):  
02     # n번 반복합니다.  
03     for i in range(n):  
04         print(value)  
05  
06     # 함수를 호출합니다.  
07     print_n_times("안녕하세요")
```

실행결과

안녕하세요  
안녕하세요

# 키워드 매개변수

- 기본 매개변수가 가변 매개변수보다 앞에 올 때
  - 기본 매개변수의 의미가 사라짐

```
def print_n_times(n=2, *values):  
    # n번 반복합니다.  
    for i in range(n):  
        # values는 리스트처럼 활용합니다.  
        for value in values:  
            print(value)  
        # 단순한 줄바꿈  
        print()  
  
# 함수를 호출합니다.  
print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍")
```

오류

Traceback (most recent call last):

File "test5\_03.py", line 11, in <module>

print\_n\_times("안녕하세요", "즐거운", "파이썬 프로그래밍")

File "test.py", line 3, in print\_n\_times

for i in range(n):

TypeError: 'str' object cannot be interpreted as an integer

- 가변 매개변수가 기본 매개변수보다 앞에 올 때
  - 가변 매개변수가 우선됨

```
def print_n_times(*values, n=2):  
    # n번 반복합니다.  
    for i in range(n):  
        # values는 리스트처럼 활용합니다.  
        for value in values:  
            print(value)  
        # 단순한 줄바꿈  
        print()  
  
# 함수를 호출합니다.  
print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", 3)
```

안녕하세요

즐거운

파이썬 프로그래밍

3

안녕하세요

즐거운

파이썬 프로그래밍

3



- 키워드 매개변수

- 매개변수 이름을 지정해서 입력하는 매개변수

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
# while 반복문을 사용합니다.
```

```
while True:
```

```
    # "."을 출력합니다.
```

```
    # 기본적으로 end가 "\n"이라 줄바꿈이 일어나는데,
```

```
    # 빈 문자열 ""로 바꿔서 줄바꿈이 일어나지 않게 합니다.
```

```
    print(".", end="") → 키워드 매개변수입니다.
```

```
01 def print_n_times(*values, n=2):  
02     # n번 반복합니다.  
03     for i in range(n):  
04         # values는 리스트처럼 활용합니다.  
05         for value in values:  
06             print(value)  
07         # 단순한 줄바꿈  
08         print()  
09  
10     # 함수를 호출합니다.  
11 print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", n=3)
```

실행결과

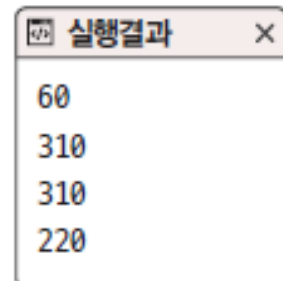
안녕하세요  
즐거운  
파이썬 프로그래밍

안녕하세요  
즐거운  
파이썬 프로그래밍

안녕하세요  
즐거운  
파이썬 프로그래밍

- 기본 매개변수 중에서 필요한 값만 입력하기
  - 예시 – 여러 함수 호출 형태

```
01 def test(a, b=10, c=100):  
02     print(a + b + c)  
03  
04 # 1) 기본 형태  
05 test(10, 20, 30)  
06 # 2) 키워드 매개변수로 모든 매개변수를 지정한 형태  
07 test(a=10, b=100, c=200)  
08 # 3) 키워드 매개변수로 모든 매개변수를 마구잡이로 지정한 형태  
09 test(c=10, a=100, b=200)  
10 # 4) 키워드 매개변수로 일부 매개변수만 지정한 형태  
11 test(10, c=200)
```



실행결과

60
310
310
220

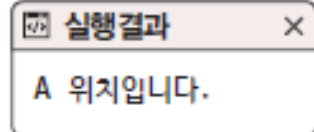
- 첫 번째 매개변수  $a$  : 일반 매개변수이므로 해당 위치에 반드시 입력해야 함
- 8행 3번 : 키워드 지정하여 매개변수 입력하는 경우 매개변수 순서를 원하는 대로 입력할 수 있음
- 10행 4번 :  $b$ 를 생략한 형태. 키워드 매개변수 사용하여 필요한 매개변수에만 값을 전달

- 리턴값 (return value)
  - 함수의 결과

```
# input() 함수의 리턴값을 변수에 저장합니다.  
value = input("> ")  
  
# 출력합니다.  
print(value)
```

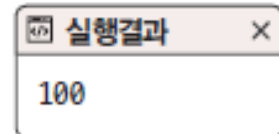
- 자료 없이 리턴하기
  - return 키워드 : 함수를 실행했던 위치로 돌아가게 함

```
01  # 함수를 정의합니다.  
02  def return_test():  
03      print("A 위치입니다.")  
04      return                # 리턴합니다.  
05      print("B 위치입니다.")  
06  
07  # 함수를 호출합니다.  
08  return_test()
```



- 자료와 함께 리턴하기
  - 리턴 뒤에 자료 입력하면 자료 가지고 돌아감

```
01  # 함수를 정의합니다.  
02  def return_test():  
03      return 100  
04  
05  # 함수를 호출합니다.  
06  value = return_test()  
07  print(value)
```

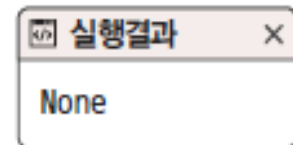


실행결과

100

- 아무것도 리턴하지 않기
  - None : '없다'라는 의미

```
01  # 함수를 정의합니다.  
02  def return_test():  
03      return  
04  
05  # 함수를 호출합니다.  
06  value = return_test()  
07  print(value)
```





- 일반적인 형태
  - 값을 만들어 리턴

```
def 함수(매개변수):
```

```
    변수 = 초깃값
```

```
    # 여러 가지 처리
```

```
    # 여러 가지 처리
```

```
    # 여러 가지 처리
```

```
    return 변수
```

- 예시 - 범위 내부의 정수를 모두 더하는 함수

```
01  # 함수를 선언합니다.
02  def sum_all(start, end):
03      # 변수를 선언합니다.
04      output = 0
05      # 반복문을 돌려 숫자를 더합니다.
06      for i in range(start, end + 1):
07          output += i
08      # 리턴합니다.
09      return output
10
11  # 함수를 호출합니다.
12  print("0 to 100:", sum_all(0, 100))
13  print("0 to 1000:", sum_all(0, 1000))
14  print("50 to 100:", sum_all(50, 100))
15  print("500 to 1000:", sum_all(500, 1000))
```

**실행결과**

```
0 to 100: 5050
0 to 1000: 500500
50 to 100: 3825
500 to 1000: 375750
```

- 예시 - 기본 매개변수와 키워드 매개변수를 활용해 범위의 정수를 더하는 함수

```
01  # 함수를 선언합니다.
02  def sum_all(start=0, end=100, step=1):
03      # 변수를 선언합니다.
04      output = 0
05      # 반복문을 돌려 숫자를 더합니다.
06      for i in range(start, end + 1, step):
07          output += i
08      # 리턴합니다.
09      return output
10
11  # 함수를 호출합니다.
12  print("A.", sum_all(0, 100, 10))
13  print("B.", sum_all(end=100))
14  print("C.", sum_all(end=100, step=2))
```

실행결과	
A.	550
B.	5050
C.	2550

- **호출** : 함수를 실행하는 행위
- **매개변수** : 함수의 괄호 내부에 넣는 것
- **리턴값** : 함수의 최종적인 결과
- **가변 매개변수 함수** : 매개변수를 원하는 만큼 받을 수 있는 함수
- **기본 매개 변수** : 매개변수에 아무 것도 넣지 않아도 들어가는 값

- 다음과 같이 방정식을 파이썬 함수로 만들어보세요.

예:  $f(x)=x$

```
def f(x):  
    return x  
print(f(10))
```

①  $f(x)=2x+1$

```
def f(x):  
    return   
print(f(10))
```

②  $f(x)=x^2+2x+1$

```
def f(x):  
    return   
print(f(10))
```

- 다음 빈칸을 채워 매개변수로 전달된 값들을 모두 곱해서 리턴하는 가변 매개변수 함수를 만들어보세요.

```
def mul(*values):
```

```
# 함수를 호출합니다.
```

```
print(mul(5, 7, 9, 10))
```

실행결과

3150

- 다음 중 오류가 발생하는 코드를 고르세요.

①

```
def function(*values, valueA, valueB):  
    pass  
function(1, 2, 3, 4, 5)
```

②

```
def function(*values, valueA=10, valueB=20):  
    pass  
function(1, 2, 3, 4, 5)
```

③

```
def function(valueA, valueB, *values):  
    pass  
function(1, 2, 3, 4, 5)
```

④

```
def function(valueA=10, valueB=20, *values):  
    pass  
function(1, 2, 3, 4, 5)
```