6장. 함수(메서드)

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 정의
  - **내장 함수** – 파이썬이 제공하는 함수

함수(Function)

- **사용자 정의 함수의 형태**
  - return 반환값이 있는 함수
  - return 반환값이 없는 함수

```
def 함수 이름():
    함수의 내용
```

```
def 함수 이름(매개 변수):
    함수의 내용
    return 반환값
```

사용자 정의 함수

- **return이 없는 함수**

```
def say_hello():
    print("안녕~")

def say_hello2(name):
    print(name + "님 안녕~")

# 함수 호출
say_hello()

say_hello2("민준")
say_hello2("Elsa")
```

함수의 정의

함수의 호출

함수 정의하고 호출하기

- return이 있는 함수

```
# 제곱수 구하는 함수
def square(x):
    return x * x;

# 절대값 구하는 함수
def my_abs(x):
    if x < 0:
        return -x
    else:
        return x

val1 = square(8)
print(val1)

val2 = my_abs(-5)
print(val2)
```

구구단을 출력하는 함수

- 구구단을 출력하는 함수

```
def print_gugudan(dan):
    for i in range(1, 10):
        print(dan, 'x', i, '=', (dan * i))

print_gugudan(5)
```

도형의 면적 계산

- 도형의 면적을 계산하는 함수 정의와 사용

```
def square(w, h):
    area = w * h
    return area

def triangle(n, h):
    area = n * h / 2
    return area

print('사각형의 면적 : ', square(5, 4))
print('삼각형의 면적 : ', triangle(4, 7))
```

간단한 규칙기반 챗봇

- 챗봇(chatbot) 함수 만들기

```python
def my_chatbot():
    print("안녕하세요! 저는 간단한 챗봇입니다.(종료: exit)")

    while True:
        user_input = input("사용자: ")

        if user_input == "exit":
            print("챗봇: 대화를 종료합니다. 안녕히 가세요!")
            break

        elif "안녕" in user_input:
            print("챗봇: 안녕하세요!. 반가워요")

        elif "이름" in user_input:
            print("챗봇: 저는 Python 챗봇입니다.")

        elif "기분" in user_input:
            print("챗봇: 저는 항상 기분이 좋아요.")

        else:
            print("챗봇: 죄송해요, 잘 이해하지 못했어요.")

# 실행
my_chatbot()
```

```
안녕하세요! 저는 간단한 챗봇입니다.(종료: exit)
사용자: 안녕~
챗봇: 안녕하세요!. 반가워요
사용자: 이름이 뭐니?
챗봇: 저는 Python 챗봇입니다.
사용자: 꿈이 있어?
챗봇: 죄송해요, 잘 이해하지 못했어요.
사용자: exit
챗봇: 대화를 종료합니다. 안녕히 가세요!
```

매개변수로 리스트 전달

- 리스트를 매개변수로 전달하여 평균 계산하기

```
def get_avg(a):
    sum_v = 0
    for i in a:
        sum_v += i  #합계

    avg = sum_v / len(a) #평균

    return avg


v = [1, 2, 3, 4]
average = get_avg(v)

print("평균 :", average)
```

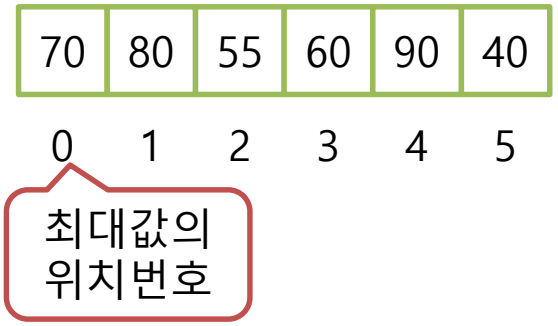
매개변수로 리스트 전달

- 리스트를 매개변수로 새로운 리스트 만들기

```
def times(a):
    a2 = []
    for i in a:
        a2.append(i * 4)
    return a2

v = [1, 2, 3, 4]
v2 = times(v)
print(v2)
```

매개변수로 리스트 전달

- 최대값과 최대값의 위치 구하기



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

매개변수로 리스트 전달

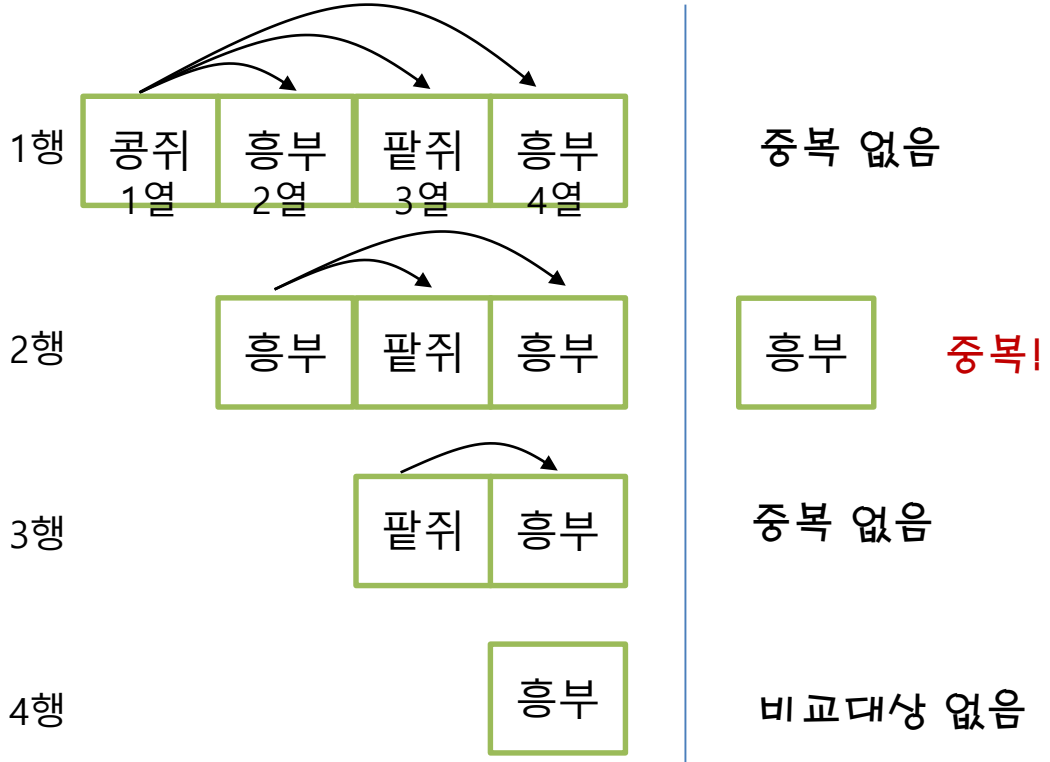
- 최대값과 최대값의 위치 구하기

```
# 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)
```

동명이인 찾기 – 중복 검사

♥ 같은 이름 찾기



동명이인 찾기 – 중복 검사

```
# 동명이인 찾기
def find_same_name(a):
    same_name = []
    n = len(a)
    for i in range(0, n-1):
        for j in range(i+1, n):
            if a[i] == a[j]:
                same_name.append(a[i])
    return same_name

'''
1행 a[0] == a[1], a[0] == a[2], a[0] == a[3]
2행 a[1] == a[2], a[0] == a[3] 중복!
3행 a[2] == a[3]
'''

name = ['콩쥐', '흥부', '팥쥐', '흥부']
result = find_same_name(name)

print(result)
```

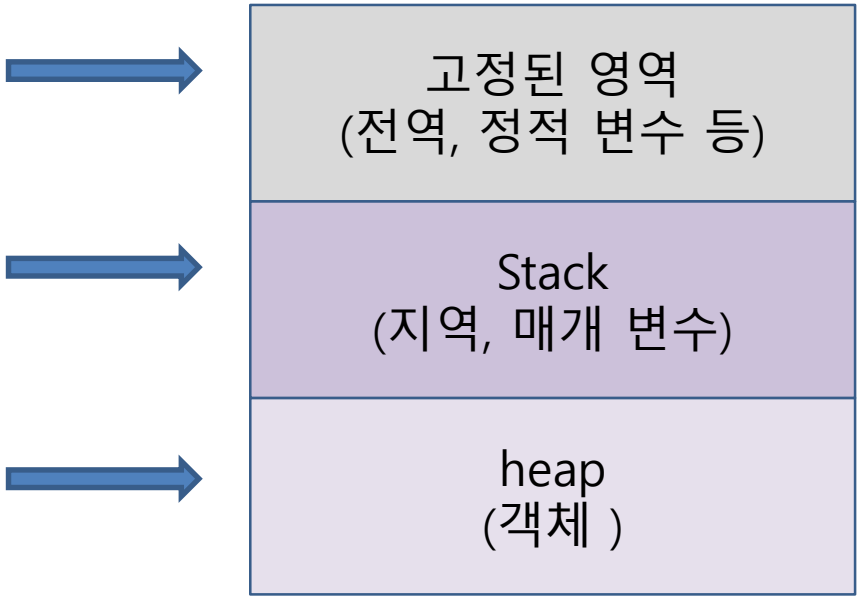
변수의 메모리 영역

- **변수의 메모리 영역 분석**

데이터 영역 : **전역 변수**가 저장되는 영역

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

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



변수의 유효 범위 – 전역변수

- **전역 변수의 유효 범위**

  전역 변수는 메인 함수의 위쪽에서 선언하여 사용하고, 영향 범위가 전체로 미친다. 프로그램이 종료되면 메모리에서 소멸한다.

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

quantity = 2 #전역 변수
get_price()  #함수 호출
```

변수의 유효 범위 – 지역변수

- **지역 변수(local variable)의 유효 범위**

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

```
def one_up():
    x = 1   #지역 변수
    x += 1
    return x

print(one_up())
print(one_up())
print(one_up())
# print(x)  - 오류 발생
```

```
2
2
2
Traceback (most recent call last):
  File "d:\korea_IT\pyworks\function\function1.py"
    print(x)  # 오류 발생
          ^
NameError: name 'x' is not defined
```

변수의 유효 범위 – 정적변수

- **정적 변수의 유효 범위**

  지역변수에 global 키워드를 붙이면 정적 변수가 되어 값을 유지하고,
  프로그램이 종료되면 메모리에서 소멸한다.

```
def one_up2():
    global x  # 정적변수(전역 변수화 함)
    x += 1
    return x

x = 1  # 전역 변수
print(one_up2())
print(one_up2())
print(one_up2())
```

```
2
3
4
```

배송비 계산하기

- 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

```
# 기본매개 변수
def print_string(text, count=1):
    for i in range(count):
        print(text)

print_string("Hello")
print_string("Hello", 3)
```

함수의 가변 매개변수

- **가변 매개변수**

  매개변수의 입력값이 정해지지 않고 변경해야 할때 사용하는 변수이다.

  변수이름 앞에 *를 붙인다.

def 함수 이름**(*변수):**

    코드블럭

```
def calc_avg(*numbers):
    sum_v = 0
    avg = 0.0
    for i in numbers:
        sum_v += i
    avg = sum_v / len(numbers)
    return avg

avg1 = calc_avg(1, 2)
avg2 = calc_avg(1, 2, 3, 4)
print(avg1)
print(avg2)
```

내장 함수(Built in Function)

❖ 내장 함수(Built in Function)

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

Built-in Functions

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

```
# 반올림
print(round(2.74))
print(round(2.24))

# 문자열 표현식 -> 숫자로 변환
# print('1 + 2')
print(eval('1 + 2'))

# 리스트로 변환
print(list('korea'))
```

```
['k', 'o', 'r', 'e', 'a']
```

```
10
10
4
4
4
1
1
1
3
2
3
```

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

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
```

재귀 함수(recursive function)

- **재귀 함수(recursive function)**

  어떤 함수 안에서 자기 자신을 부르는 것을 말한다.

  재귀호출은 무한 반복하므로 **종료 조건**이 필요함

```
def func(입력 값):
    if 입력값이 충분히 작으면: #종료 조건
        return 결과값
    else:      # 더 작은 값으로 호출
        return 결과값
```

재귀 함수(recursive function)

- 재귀 호출: 다시 돌아가 부르기

```
def sos(i):
    print("Help me!")
    if i <= 1:
        return ''
    else:
        return sos(i - 1)
'''
    i=4, Help me!, sos(3)
    i=3, Help me!, sos(2)
    i=2, Help me!, sos(1)
    i=1, Help me!, sos(0)
    i=0, 공백 문자
'''
sos(4)
```

재귀 호출

- 1부터 n까지 곱하기

```
# 1부터 n까지의 곱 구하기(1x2x3x...xn)
def facto(n):
    gob = 1
    for i in range(1, n+1):
        gob *= i
        #print(i, gob)
    return gob

print(facto(1))
print(facto(4))
print(facto(5))
```

팩토리얼(factorial)

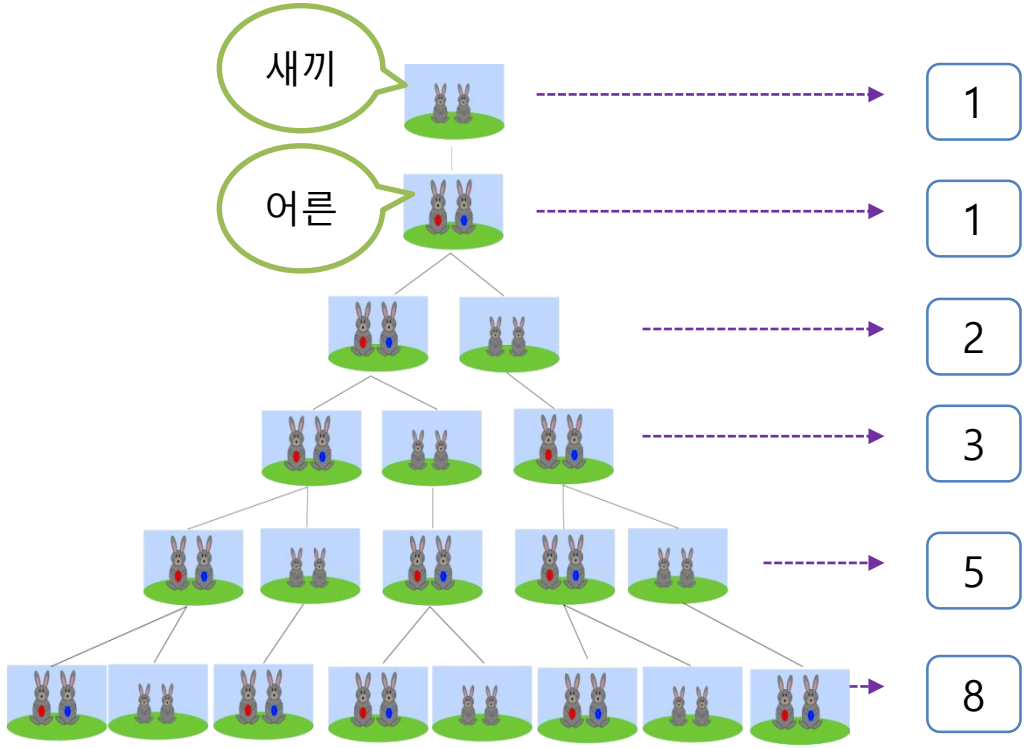
- 팩토리얼을 구하는 재귀 함수

```
def facto(n):
    if n <= 1:
        return 1
    else:
        return n * facto(n-1)

'''
    n = 4, 4 * facto(3)
           4 * 3 * facto(2)
           4 * 3 * 2 * facto(1)
           4 * 3 * 2 * 1  <24>
'''

print(facto(1)) # 1!
print(facto(2)) # 2!
print(facto(3)) # 3!
print(facto(4)) # 4!
```

피보나치 수열

- **피보나치(Fibonacci) 수열**

  수학에서 피보나치 수는 첫째 및 둘째 항이 1이며, 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열이다. 처음 여섯 항은 각각 1, 1, 2, 3, 5, 8이다.



첫번째 달에 새로 태어난 토끼 한쌍이 있고, 둘째달에 토끼가 커서 그대로 어른토끼 한쌍, 세째달에는 새끼를 한쌍 낳아 어른, 새끼 두쌍, 네째달에는 어른이 새끼를 낳고, 새끼는 어른이 되어 총 세쌍, 이렇게 계속 새끼를 낳고, 죽지 않는다는 가정을 세우면 피보나치의 수가 된다.

피보나치 수열

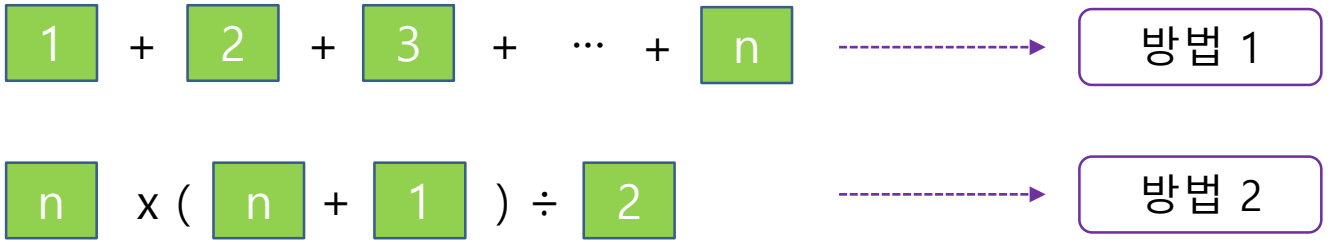
- 피보나치 수열 – 재귀 함수

```python
# fibonacci - 1, 1, 2, 3, 5, 8...
# 앞항 + 뒤항 = 항 반복
def fibo(n):
    if n <= 2:
        return 1
    else:
        return fibo(n-2) + fibo(n-1)

"""
    n = 1    fibo(1) = 1
    n = 2    fibo(2) = 1
    n = 3    fibo(1) + fibo(2) = 2
    n = 4    fibo(2) + fibo(3) = 3
    n = 5    fibo(3) + fibo(4) = 5
"""
print(fibo(1))
print(fibo(2))
print(fibo(3))
print(fibo(4))
```

```
1
1
2
3
```

알고리즘 계산 복잡도

- 1부터 n까지의 합을 구하기




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

print(sum_n(10))
```

```python
def sum_n2(n):
    sum_v = (n * (n + 1)) // 2
    return sum_v

print(sum_n2(10))
```

알고리즘 계산 복잡도

➢ 계산 복잡도

- **입력 크기와 계산 횟수**
  - 첫 번째 알고리즘 : 덧셈 n번
  - 두 번째 알고리즘 : 덧셈, 곱셈, 나눗셈(총 3번)

- **대문자 O표기법(Big O) : 계산 복잡도 표현**
  - O(n) : 필요한 계산횟수가 입력 크기 n과 비례할 때
  - O(1) : 필요한 계산횟수가 입력 크기 n과 무관할 때
- **판단**
  - 두번째 방법이 계산 속도가 더 빠름

실습 문제 – 사용자 정의 함수

두 수를 매개변수로 전달하여 서로 같으면 더하고, 서로 다르면 빼는 함수를 정의하고 호출하는 프로그램을 작성하세요.(파일: func_test.py)

<함수 호출>

```
result1 = my_func(8, 8)
print("result1 =", result1)

result2 = my_func(8, 9)
print("result2 =", result2)
```

☞ 실행 결과

```
result1 = 16
result2 = -1
```