04장 함수

4.1 함수의 정의

- 입력값을 가지고 어떤 일을 수행한 다음에 그 결과물을 내어놓은 것, 이것이 바로 함수가 하는 일이다.

- 함수를 사용하는 이유는 반복된 코드를 줄이고 프로그램의 흐름을 일목요연하게 볼 수 있기 때문이다.
- 함수를 선언하는 문법은 아래와 같다.

```
def 함수명(인자1, 인자2, …인자N):
    구문
    return 반환값
```

```
# 파이썬 함수의 구조
def add(a, b):
    return a + b
print(add)  # 함수가 생성될 때마다 객체의 메모리 주소는 매번 다르다.

a = 3
b = 4
c = add(a, b)
print(c)

# 내장함수 목록 확인하기
print(dir(__builtins__))  # __builtins__는 내장 영역의 이름이 저장돼 있는 리스트.
x = [1,2,3]
print(sum(x))  # 내장함수 sum()을 사용함
```

4.2 함수의 형태

```
# 입력값과 결과값에 따른 함수의 형태
# 1. 일반적인 함수
def sum(a, b):
    result = a + b
    return result

# 2. 입력값이 없는 함수
def say():
    return 'Hi'

a = say()
print(a)

# 3. 결과값이 없는 함수
def sum1(a, b):
    print("%d, %d의 합은 %d입니다." % (a, b, a+b))

a = sum1(3,4)
```

```
print(a)

# 4. 입력값도 결과값도 없는 함수
def say1():
    print('Hi')

say1()
# Hi
```

4.3 함수의 인자

4.3.1 기본 인자 값

■ 함수를 호출할 때 인자를 지정해 주지 않아도 기본 값이 할당되게 하는 방법이다.

```
# 기본 인자 값
print('-'*20)
def Times(a=10, b=20):
    return a*b
print(Times())
print(Times(5))  # a에만 5가 할당된다.

def say_myself(name, old, man=True):
    print("나의 이름은 %s입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
say_myself("박응용",27)
say_myself("박응용",27,True)
say_myself("박응선",27,False)

# 인자 전달 방식
a = 10
b = 20
def sum1(x, y):
   return x + y
print(sum1(a, b))
# 30

# 함수 안에서 선언된 변수의 효력 범위
x = 10
def sum2(x, y):
   x = 1  # 이 부분에서 값이 1인 객체가 생성되고 x에 레퍼런스를 할당한다. (scoping rule 참조)
   return x + y
print(sum2(x, b))  # x를 인자로 넣어 준다.
# 21
print(x)  # 함수 내부에서 변경한 사항이 외부에 영향을 미치지 않는다.
# 10

a = 1  # 전역변수
def vartest(b):
    a = b + 1  # 지역변수
vartest(a)
print(a)


# 함수 안에서 함수 밖의 변수를 변경하는 방법
```

```
a = 1
def vartest():
    global a
    a = a+1
vartest()
print(a)
```

4.3.2 키워드 인자

- C,C++와 JAVA에서는 반드시 변수의 전달 순서를 맞춰줘야 하지만, 파이썬에서는 키워드 인자 이름으로 값을 전달할 수 있다.

```
# 키워드 인자
def connectURI(server, port):
    str = "http://" + server + ":" + port
    return str
print(connectURI("test.com", "8080"))
print(connectURI(port="8080", server="test.com"))
```

4.3.3 가변 인자 리스트

- 함수를 호출할 때 인자의 개수가 정해지지 않은 가변 인자를 전달받는 방법이 있다.
- *를 함수 인자 앞에 붙이면 가변 인자를 받겠다는 의미이다.

```
# 가변 인자 리스트
def sum_many(*args):
    sum = 0
    for i in args:
        sum = sum + i
    return sum

result = sum_many(1,2,3)
print(result)
result = sum_many(1,2,3,4,5)
print(result)

def sum_mul(choice, *args):
    if choice == "sum":
        result = 0
        for i in args:
            result = result + i
    elif choice == "mul":
        result = 1
        for i in args:
            result = result * i
    return result

result = sum_mul('sum', 1,2,3,4)
print(result)
result = sum_mul('mul', 1,2,3,4,5)
print(result)
```

4.3.4 정의되지 않은 인자 처리하기

- **를 붙이면 정의되지 않은 인자를 딕셔너리 형식으로 전달받을 수 있다. 인자를 딕셔너리 형식으로 만들어 전달할 수 있다는 것은 정말 큰 매력이다.

```
# 정의되지 않은 인자 처리하기
def userURIBuilder(server, port, **user):
    str = "http://" + server + ":" + port + "/?"
    for key in user.keys():
        str += key + "=" + user[key] + "&"
    return str
print(userURIBuilder("test.com","8080",id='userid',password='1234'))
print(userURIBuilder("test.com","8080",id='userid',password='1234',name='mike',age='20'))
```

4.4 람다 함수

- 람다 함수는 함수 이름이 없고, 함수 객체만 존재하는 익명 함수를 의미한다.
- 일반 함수와 마찬가지로 여러 개의 인자를 전달 받을 수 있고, return 구문을 적지 않아도 하나의 반환값을 돌려 준다.
- 한 줄의 간단한 함수가 필요한 경우나 프로그램의 가독성을 위해 혹은 함수를 인자로 넘겨줄 때 람다 함수를 쓸 수 있다.
- 문법

    lambda 인자 : 구문

```
print("\n*** 람다 함수 ***")
g = lambda x,y : x*y
print(g(2,3))
print((lambda x:x*x)(3))

def sqrt(x):
    x = x*x
    return x
print(list(map(sqrt,[1,2,3,4,5])))
# [1, 4, 9, 16, 25]
print(list(map(lambda x:x*x,[1,2,3,4,5])))
# [1, 4, 9, 16, 25]

foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
print(list(filter(lambda x:x%3==0, foo)))
# [18, 9, 24, 12, 27]
```

4.5 재귀적 함수 호출

- 재귀적(recursive) 함수 호출은 함수 내부에서 자기 자신을 호출하는 것을 말한다.

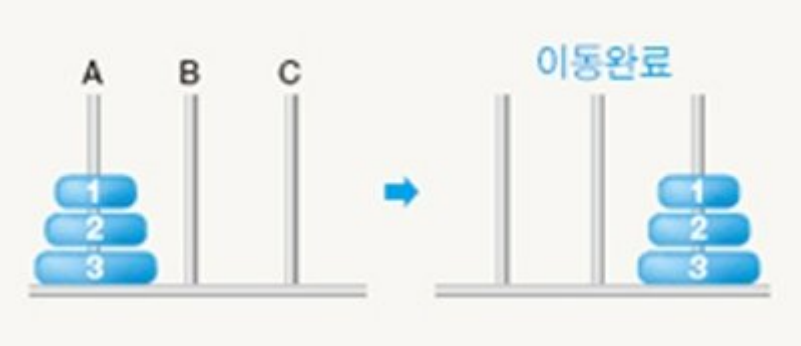
```
print("\n*** 재귀적 함수 호출 ***")
def factorial(x):
    if x == 1:
        return 1
    return x * factorial(x-1)
print(factorial(10))
```

[과제] 하노이 탑(Tower of Hanoi) 게임

아래와 같이 하노이 탑 퍼즐을 재귀적 함수를 사용하여 파이썬 코드를 작성해 보자.

세 개의 기둥과 이 기둥에 꽂을 수 있는 크기가 다양한 원판이 있고, 퍼즐을 시작하기 전에는 한 기둥에 원판들이 작은 것이 위에 있도록 순서대로 쌓여 있다. 게임의 목표는 다음 두 가지 조건을 만족시키면서, 한 기둥에 꽂힌 원판을 그 순서 그래도 다른 기둥으로 옮겨서 다시 쌓는 것이다.

- 한 번에 하나의 원판만 옮길 수 있다.
- 큰 원판이 작은 원판에 위에 있어서는 안 된다.



[힌트]
풀잇법은 아래와 같다.

- 기둥1에서 N-1개의 원반을 기둥3을 이용해 기둥2로 옮긴다.
- 기둥1에서 1개의 원반을 기둥3으로 옮긴다.
- 기둥2에서 N-1개의 원반을 기둥1을 이용해 기둥3으로 옮긴다.

[정답]

```
# Tower of Hanoi

def hanoi(ndisks, startPeg=1, endPeg=3):
    if ndisks:
        hanoi(ndisks-1, startPeg, 6-startPeg-endPeg)
        print(startPeg, "번 기둥의", ndisks, "번 원반을", endPeg, "번 기둥에 옮깁니다.")
        hanoi(ndisks-1, 6-startPeg-endPeg, endPeg)

hanoi(ndisks=3)

# 실행 결과
1 번 기둥의 1 번 원반을 3 번 기둥에 옮깁니다.
1 번 기둥의 2 번 원반을 2 번 기둥에 옮깁니다.
3 번 기둥의 1 번 원반을 2 번 기둥에 옮깁니다.
1 번 기둥의 3 번 원반을 3 번 기둥에 옮깁니다.
2 번 기둥의 1 번 원반을 1 번 기둥에 옮깁니다.
2 번 기둥의 2 번 원반을 3 번 기둥에 옮깁니다.
1 번 기둥의 1 번 원반을 3 번 기둥에 옮깁니다.
```

4.6 함수 안에서 선언된 변수의 효력 범위(scoping rule)

- 함수는 별도의 이름공간을 가진다. 함수 내부에서 사용되는 (지역)변수는 일단 함수 내부의 이름공간을 참조한다. 만약 함수 내부의 이름공간에서 차지 못하면 상위 이름공간에서 (전역)변수을 찾습니다.
- 함수가 끝나면 함수 내부에서 생성하고 사용한 지역 변수들은 메모리 공간에서 모두 반환(제거)되는 반면, 전역 변수들은 함수가 종료해도 메모리 공간에 남아 있게 된다. 이는 서로 다른 이름공간이기 때문이다.

```
print("\n*** 함수 안에서 선언된 변수의 효력 범위 (스코핑 룰) ***")
# 전역 변수
x = 1
def func1(a):
    return a+x
print(func1(1))
# 2

# 지역 변수
def func2(a):
    x=2
    return a+x
print(func2(1))
# 3

g = 1
def func3(a):
    global g  # 변수에 global 키워드를 지정해 전역 영역에 존재하는 변수임을 지정한다.
    g = 2
    return g + a
print(func3(1))
# 3
print(g)
# 2
```

4.7 요약

- 정의: def 함수명(인자1, 인자2,...):
  실행문
  return 반환값
- 사용: 변수 = 함수명(인자1,인자2)
- 키워드 인자: def 함수명(k1=v1, k2=v2):
- 가변 인자: def 함수명(*args):
- 정의되지 않은 인자 처리: def 함수명(**user):
- 람다 함수: 익명 함수, 예) lambda x:x*x
- 재귀 함수: 함수내부에서 자기 자신을 호출하는 함수

[과제] 연습문제

Q1 주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수(is_odd)를 작성해 보자.

```
# Q1 주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수(is_odd)를 작성해 보자.
def is_odd(number):
    if ___________  # 2로 나누었을 때 1이면 홀수이다.
        return True
    else:
```

```
        return False

print(is_odd(3))
# True
print(is_odd(4))
# False
```

Q2 입력으로 들어오는 모든 수의 평균 값을 계산해 주는 함수를 작성해 보자. (단 입력으로 들어오는 수의 개수는 정해져 있지 않다.)

```
def avg_numbers(___________):
    result = 0
    for i in args:
        result += i
    return ___________________

print(avg_numbers(1, 2))
# 1.5
print(avg_numbers(1,2,3,4,5))
# 3.0
```

Q3 첫 번째 항의 값이 0이고 두 번째 항의 값이 1일 때, 이후에 이어지는 항들은 이전의 두 항을 더한 값으로 이루어지는 수열을 피보나치 수열이라고 한다. 예) 0, 1, 1, 2, 3, 5, 8, 13, 21, ... 입력을 정수 n으로 받았을 때, n 이하까지의 피보나치 수열을 출력하는 함수를 작성해 보자.

```
def fib(n):
    if _____ __ __    # n이 0일 때는 0을 리턴
    if _____ _____    # n이 1일 때는 1을 리턴
    return ____ __  # n이 2이상일 때는 그 이전의 두 값을 더하여 리턴

for i in range(10):
    print(fib(i), end=' ')
# 0 1 1 2 3 5 8 13 21 34
```