Function

Structure & Definition

송지수

Structure

Overview

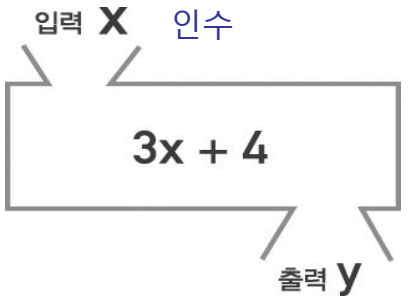
✦함수?

여기서 이런 일을 하고 싶다, 귀찮은 일을 다른 사람에게 맡김

- 어떤 입력을 대해 작업을 수행한 후 결과물을 만드는 기능

✦프로그램에서의 함수

- 프로그램에서 자주 사용되는 특정 기능을 따로 만들어 정의함
- 주로 특정 기능을 수행하는 코드들을 함수로 정의하여 반복적으로 수행함
  - 함수는 외부에서 호출하여 입력을 전달하고 처리가 완료된 후 결과를 반환
- 프로그래머는 이미 정의 되어 있는 함수를 사용하거나 필요한 함수를 정의하여 사용함



인수

반환 값
: 함수를 실행한 결과로 반환된 데이터

```
ret = Function(argument)
ret : 반환값
Function : 함수이름
argument : 함수에 입력하는 값
```

Function

Types

✦함수의 종류

- 내장 함수 (Built-In Function)
  - 파이썬에서 기본적으로 제공하는 함수
- 라이브러리 함수 (Library Function)
  - 이미 외부에서 미리 정의된 패키지를 사용한 라이브러리 함수
- 사용자 함수 (User Defined Function)
  - 사용자가 필요한 기능을 직접 정의한 함수

Function

Library

✦라이브러리 혹은 패키지 함수

- 파이썬은 이미 정의되어 있는 함수들이 기능별로 패키지 혹은 모듈에 포함하여 제공함
- 모듈이란 파이썬 코드를 포함하고 있는 파일을 말함
  - 모듈들을 묶어서 패키지라 함
- 파이썬을 설치하면 많은 패키지와 모듈들이 설치되고, 프로그램에서 함수를 사용하기 위해서는 해당 라이브러리를 포함한 후에 사용할 수 있음
- 기본 라이브러리에서 제공되지 않는 것은 추가의 라이브러리를 설치해서 사용할 수 있음
  - 예 : numpy, scipy, matplotlib 등
- 라이브러리나 모듈을 포함시키기 위해서는 import 예약어를 사용한다.

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```

```
>>> import random
>>> random.randint(1, 10)
8
```

Function

User Defined Function

✦사용자 정의 함수

- 사용자가 자신이 필요로 하는 기능을 수행하는 함수를 직접 작성함
- 특정 기능을 수행하는 코드들을 하나의 묶음으로 사용
- 동일한 기능을 수행하는 코드들의 재 사용을 위해 함수를 작성
- 코드의 통일된 관리를 위해 함수를 작성

Function

Structure

✦함수의 구조

- 함수의 선언은 def로 시작하고 콜론(:)으로 끝냄
- 함수의 코드 블록은 함수의 내용을 처리하는 부분이며, 들여쓰기로 구분함
- 인수 (Argument or Parameter)
  - 함수에 전달하는 정보를 말하며, 함수에 입력 값이 필요할 경우 인수를 사용함
- 반환 값 (Return Value)
  - 함수가 실행 후 결과로 반환된 데이터
  - 반환 값이 발생할 경우 return문을 사용하며, return문을 사용하지 않을 경우 None을 반환함

함수가 수행하는데 필요한 자료

인수가 없을 경우 괄호만

```
def 함수명(인수1, 인수2, ...):
    함수의 내용(수행할 문장)
    ........
    return 반환값
```

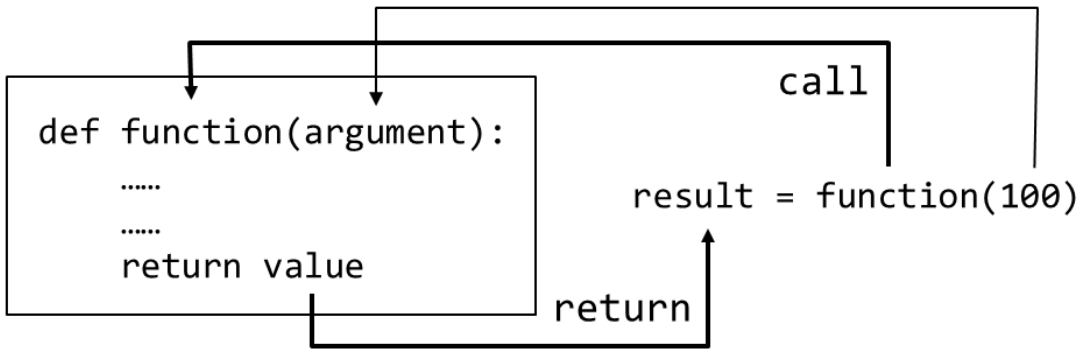
Function

Structure

✦함수의 호출과 반환

- 함수를 호출(Call)한다는 말은 함수 안의 코드를 수행한다는 의미
- 함수를 호출할 때는 함수 이름 다음에 소괄호를 사용하여 표현
  - 함수를 호출하는 코드는 함수의 정의보다 나중에 작성되어야 함
- 함수에 전달할 값이 있다면 인수를 통해 전달
- 함수가 수행되고 난 후 함수의 결과를 반환 값 이라고 함
  - 반환 값은 함수를 호출한 지점으로 돌아가 자신을 호출한 코드에게 값을 돌려줌
  - 파이썬에서는 여러 개의 결과를 반환할 수 있음

함수의 호출
함수명( )




```
>>> def message():정의
        print("hello python")

>>> message()  인수가 없을 경우
hello python   괄호만

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

>>> add(10, 20)  인수를 전달
30
```

Practice

Example 11-2

```
>>> def compute_add(number1, number2):
        output = number1 + number2
        return output

>>> result = compute_add(10, 20)
>>> print(result)
30
>>> result = compute_add("hello", "python")
>>> print(result)
hellopython
>>> result = compute_add([10, 20], [30, 40])
>>> print(result)
[10, 20, 30, 40]
>>> result = compute_add(10, "hello")
Traceback (most recent call last):
  File "<pyshell#148>", line 1, in <module>
    result = compute_add(10, "hello")
  File "<pyshell#141>", line 2, in compute_add
    output = number1 + number2
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Function

Structure

✦함수의 형태

- 함수의 인수와 반환 값이 모두 없는 경우
  - 그냥 인수 없이 괄호만 사용하며, return 키워드를 사용하지 않음

```
>>> def function1():
        return

>>> print(function1())
None
```

```
>>> def function2():
        pass    파이썬에서 아무 일도 안 한다는 의미

>>> print(function2())
None    함수의 결과값이 없을 경우 None 객체를 반환
```

- 함수의 인수는 존재하나 반환 값이 없는 경우
  - 인수로 전달된 값은 함수 내에서 처리되고 함수 밖에는 영향을 주지 않음
- 함수의 인수는 존재하지 않지만 반환 값이 존재하는 경우
  - 함수 내에서 생성하여 처리한 값을 함수 밖으로 반환함
- 함수의 인수도 존재하며 반환 값도 존재하는 경우
  - 가장 많이 사용하는 함수의 형태
- 함수의 반환 값이 두 개 이상인 경우
  - 다른 프로그래밍 언어에 비해 파이썬에서는 여러 개의 반환 값을 정의할 수 있음

Practice

Example 11-4

✦3개의 인수를 전달받아 최대값 구하는 함수의 작성

```
def get_max(num1, num2, num3):
    if num1 > num2:
        maximum = num1
    else:
        maximum = num2

    if num3 > maximum:
        maximum = num3

    return maximum

print("최대값 :", get_max(20, 30, 10))
```

```
최대값 : 30
```

Practice

Example 11-5

✦문자열을 인수로 찾는 문자의 개수 구하는 함수의 작성

```
def check_char(message, search):
    count = 0
    for letter in message:
        if letter != search:
            continue
        count += 1
    return count

message = input("문자열 입력 : ")
search = input("찾을 문자 : ")
result = check_char(message, search)
print("%c 문자 개수 : %d" %(search, result))
```

```
문자열 입력 : hello python
찾을 문자 : o
o 문자 개수 : 2
```

Function

Scope

✦변수의 유효범위

- 프로그램에서 변수가 사용되는 유효범위
  - 변수의 생명주기와 밀접한 관련
- 변수를 생성할 때 변수가 사용되는 범위에 대해서 고려해야 함
- 내부 영역 : 함수 내의 영역
- 외부 영역 : 함수 밖의 영역

Function

Local Variable

✦지역 변수

- 함수 내부와 같은 한정된 지역의 범위에서만 유효한 변수
- 함수 내에서 정의한 변수는 함수가 실행중인 동안만 유효
- 함수 밖에서는 해당 변수가 존재하지 않기 때문에 사용할 수 없음

```
def display():
    message = "python"
    print(message) ① → python 출력

display()
print(message) ② → error 접근 불가
```

함수 안에서 선언된 변수

함수 밖에서는 사용할 수 없음

Function

Global Variable

✦전역 변수

- 함수 외부에서 정의되어 프로그램 전체에서 사용되는 변수
  - 프로그램 내의 모든 함수에서 다 접근이 가능함
  - 함수들 간의 데이터 교환이 필요할 때 주로 사용
- 전역 변수는 프로그램이 실행되는 동안 유효하며, 종료하면 사라짐
- 같은 이름의 전역 변수와 지역 변수가 존재할 경우 지역 변수를 먼저 접근
- 함수 안에서 외부에 정의된 전역 변수에 접근은 가능하지만 값을 변경할 수는 없음
  - 함수 내에서 전역 변수의 값을 변경하기 위한 방법
    - 함수 내에 `global` 키워드를 사용하여 변수를 사용

Function

Global Variable

✦전역 변수

```
def display():
    .........  ③ → message = "hello" 이 정의된다면?
    print(message) ① → python 출력

display()
message = "python"
print(message) ② → python 출력
```

③ 같은 이름의 지역 변수가 정의되면
지역 변수가 전역 변수보다 우선

함수 안에서 선언된 변수

함수 안과 밖 모두 사용할 수 있음

Practice

Example 11-8

✦지역 변수와 전역 변수의 예제 프로그램

```
def display():
    lnumber = 0
    print("함수안 지역변수 :", lnumber)
    print("함수안 전역변수 :", gnumber)

lnumber = 10
gnumber = 100
print("함수밖 지역변수 :", lnumber)
print("함수밖 전역변수 :", gnumber)
display()
```

```
함수밖 지역변수 : 10
함수밖 전역변수 : 100
함수안 지역변수 : 0
함수안 전역변수 : 100
```

Practice

Example 11-9

✦계좌를 통한 입출금 프로그램

```
def deposit(money):
    global balance
    balance += money

def withdraw(money):
    global balance
    balance -= money

def display():
    print("잔액 : %d원" %balance)

balance = 1000
display()
deposit(2000)
display()
withdraw(500)
display()
```

```
잔액 : 1000원
잔액 : 3000원
잔액 : 2500원
```

Function

Global Variable

✦global 명령어 이용하기

- vartest 함수 안의 global a라는 문장은 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻.

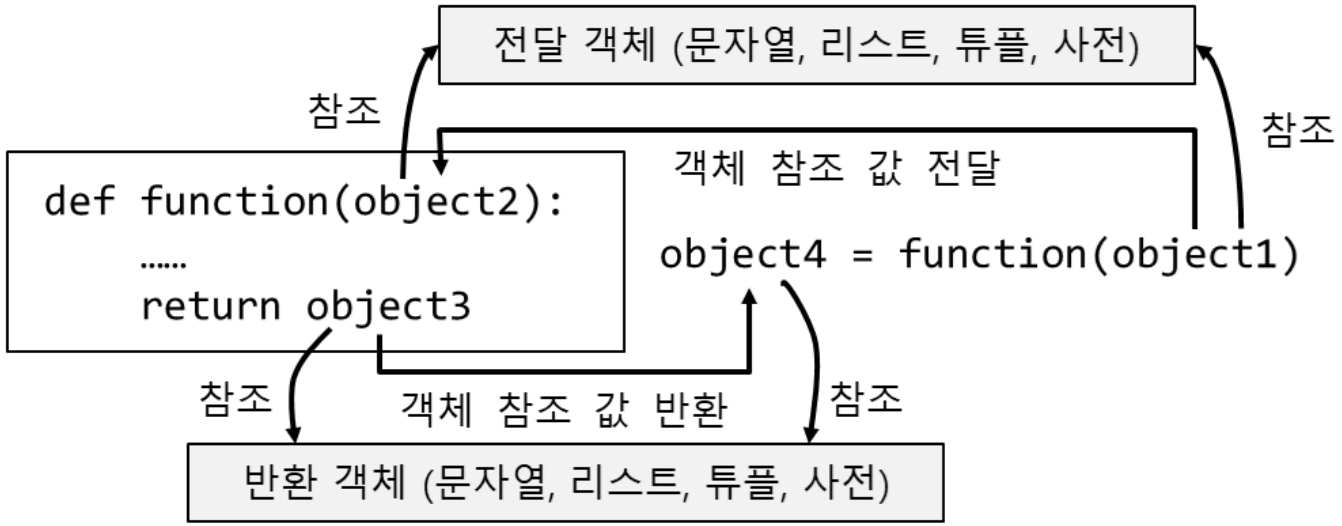
```
# vartest_global.py
a = 1
def vartest():
    global a
    a = a+1

vartest()
print(a)
```

Function

Argument

✦함수의 고정 인수 지정

- 함수 인수 개수를 정확히 지정해서 전달
- 함수에 정의된 인수의 개수와 전달할 값의 개수가 정확히 일치해야 함
- 인수는 어떤 자료형이라도 가능
  - 일반 숫자, 문자, 문자열, 리스트, 튜플, 사전
  - 인수로 전달하는 값은 객체 참조 값



Function

Argument

✦함수로 전달하는 인수 개수가 가변 적인 경우

- 함수도 인수 개수에 맞게 각각 새로 정의해야 하거나
- 하나의 함수에 가변적인 개수의 값을 전달할 수 있는 방법이 필요

✦함수의 기본 인수 지정

- 함수를 정의할 때 인수에 기본 값을 초기값으로 설정하는 것
- 함수를 호출할 때 인수를 지정하지 않으면 기본 값이 인수로 전달됨
- 사용자는 인수의 개수와 상관없이 꼭 필요한 인수만 전달할 수 있음
- 기본 인수는 필요한 개수 만큼 두는 것이 좋음
- 반드시 고정 인수를 먼저 나열하고 기본 인수를 나열해야 함

Practice

Example 11-14

```
>>> def add(number1, number2 = 10):
        return number1 + number2

>>> print(add(10))
20
>>> print(add(10, 20))
30
>>> def display(name = "none", age = 0):
        print("이름 : %s, 나이 : %d" %(name, age))

>>> display()
이름 : none, 나이 : 0
>>> display("홍길동")
이름 : 홍길동, 나이 : 0
>>> display("홍길동", 25)
이름 : 홍길동, 나이 : 25
```

고정 인수를 먼저 작성하고
기본 인수를 뒤에 나열함

인수를 하나만 지정하고 두 번째 인수는 지정하지 않더라도
두 번째 인수는 자동으로 기본 값인 10으로 지정됨

두 번째 인자를 사용해도 되며, 기본 인자 값은 무시됨

Function

Argument

✦고정 인수와 기본 인수의 한계

- 고정 인수 : 인수의 개수가 정해져 있음
- 기본 인수 : 인수의 최대 개수가 정해져 있음

✦함수의 가변 인수 지정

- 인수의 개수와 상관없이 값을 전달하는 방법
  - 인수를 가변적인 인수로 두고 여러 개의 데이터를 전달

```
def 함수명(인수1, 인수2, *가변인수):
    함수의 처리문장
    .........
```

- 가변 인수는 인수 앞에 '*' 를 붙임
- 가변 인수로 지정된 값들은 튜플로 정의하여 함수 내부로 값이 전달
- 반드시 고정 인수를 먼저 나열하고 가변 인수를 나중에 나열

Function

Argument

✦함수의 가변 인수 정의

- 가변 인수가 포함된 함수의 호출
  - 고정 인수가 있을 경우 그대로 개수만큼 인수를 지정
  - 가변 인수 자리에는 여러 개의 데이터를 콤마로 나열 (튜플로 전달)
- 가변 인수로 정의된 함수 내에서의 처리
  - for문과 같은 반복문을 이용하여 가변 인수를 처리함
- `def function(*numbers): → function(1, 2, 3, 4)`
  - 한 개의 가변 인수가 주어졌을 때 인수는 그냥 콤마로 값을 나열
- `def function(arg1, arg2, *numbers): → function(1, 2, 3, 4)`
  - 가변 인수 앞에 고정 인수 두 개가 있음
  - arg1과 arg2에는 1과 2가 각각 전달, 3, 4는 가변 인수로 전달
- `def function(**numbers): → function(one=1,two=2)`
  - 가변 인수 앞에 '**' 가 붙을 경우 튜플이 아닌 사전으로 전달받음
  - 함수를 호출할 때는 (키1=값1, 키2=값2)의 형식으로 사용한다.

Practice

Example 11-17

✦가변 인수를 사용한 덧셈 곱셈 프로그램

```
def calculator(op, *numbers):
    if op == "+":
        result = 0
        for item in numbers:
            result += item
    elif op == "*":
        result = 1
        for item in numbers:
            result *= item
    else:
        print("덧셈과 곱셈만 가능")

    return result

print("결과 : %d" %calculator("+", 1,2,3,4))
print("결과 : %d" %calculator("*", 1,2,3,4))
```

```
결과 : 10
결과 : 24
```

Function

Structure

✦매개변수 지정하여 호출하기

- 함수를 호출 할 때 매개변수를 지정하여 호출할 수도 있다.

```
>>> def sum(a, b):
        return a+b

>>> sum(a=3,b=7)
10
>>> sum(b=5,a=3)
8
>>>
```

Function

연습문제

✦[문제1] 홀수 짝수 판별

- 주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수(is_odd)를 작성.

✦[문제2] 평균값 계산

- 입력으로 들어오는 모든 수의 평균값을 계산해 주는 함수를 작성해 보자. (단, 입력으로 들어오는 수의 개수는 정해져 있지 않다.)

✦[문제3] 구구단 출력

- 입력을 자연수 n(2부터 9까지의 자연수)으로 받았을 때, n에 해당되는 구구단을 출력하는 함수를 작성해 보자.

Function

연습문제 풀이

✦[문제1] 홀수 짝수 판별

- 주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수(is_odd)를 작성.

```
>>> def is_odd(number):
...     if number % 2 == 1:  # 2로 나누었을 때 나머지가 1이면 홀수이다.
...         return True
...     else:
...         return False
...
>>> is_odd(3)
True
>>> is_odd(4)
False
```

Function

연습문제 풀이

✦ [문제2] 평균값 계산

- 입력으로 들어오는 모든 수의 평균값을 계산해 주는 함수를 작성해 보자. (단, 입력으로 들어오는 수의 갯수는 정해져 있지 않다.)

```
>>> def avg_numbers(*args):  # 입력 갯수에 상관없이 사용하기 위해 *args를 이용
...     result = 0
...     for i in args:
...         result += i
...     return result / len(args)
...
>>> avg_numbers(1, 2)
1.5
>>> avg_numbers(1,2,3,4,5)
3.0
```

Function

연습문제 풀이

✦[문제3] 구구단 출력

- 입력을 자연수 n(2부터 9까지의 자연수)으로 받았을 때, n에 해당되는 구구단을 출력하는 함수를 작성해 보자.

```
>>> def gugu(n):
...     for i in range(1, 10):
...         print(n*i)
...
>>> gugu(2)
```