Lecture 1 : 함수 정의와 활용

1. 함수 개요

- 파이썬 함수는 내장함수(built-in function)과 사용자 정의 함수(user-defined function)으로 나눌 수 있음
- **내장함수(built-in function)** : 파이썬 설치와 함께 이미 설치된 함수

> **len([1, 2, 3])**

- **사용자 정의 함수(user-defined function)** : 사용자가 직접 만들어 활용하는 함수

```
def add(x, y) :
    s = x + y
    return s


add(3, 5)
```

2. 내장함수(built-in function)

- **내장함수(built-in function)** : 파이썬 설치와 함께 이미 설치된 함수

> **len([1, 2, 3])**

내장함수(built-in function) / LAB-1

번호	사용 예	출력(결과)	설 명
1	dir(**builtins**)	['ArithmeticError', 'AssertionError', ...]	내장 함수 이름 목록을 튜플로 출력
2	help(len)	Help on built-in function len in module builtins:	내장함수 len() 활용법 요청
		len(obj, /)	
		Return the number of items in a container.	
3	len([1, 2, 3])	3	튜플 [1, 2, 3]의 원소 수 반환
4	min([1, 2, 3])	1	튜플 [1, 2, 3]의 원소 중 최소 값 반환
5	print([1, 2, 3])	[1, 2, 3]	튜플 [1, 2, 3] 출력

LAB-1 내장함수(built-in function)

In [ ]:
```
dir(__builtins__)
```

In [ ]:
```
help(len)
```

In [ ]:
```
len([1, 2, 3])
```

In [ ]:
```
min([1, 2, 3])
```

In [ ]:
```
print([1, 2, 3])
```

3. 사용자 정의 함수(user-defined function)

- **사용자 정의 함수(user-defined function)** : 사용자가 직접 만들어 활용하는 함수

> def **add(x, y)** :
> s = x + y
> return s
>
> **add(3, 5)**

- **지역변수** : 함수 내부에서 처음으로 대입되어 사용된 변수

> 함수 내부에서만 사용 가능

- **전역변수** : 함수 외부에서 처음으로 대입되어 사용된 변수

> 함수 외부는 물론 내부에서도 참조 가능
> 다만, 함수 내부에서 전역 변수 수정은 기본적으로 불가

LAB-2 사용자 정의 함수(user-defined function)

In [ ]:
```
# 함수의 정의 와 호출
def add(x, y) :
    s = x + y
    return s

add(3, 5)
```

In [ ]:
```
# 함수를 변수에 저장
def add(x, y) :
    s = x + y
    return s

f = add
f(3, 5)
```

In [ ]:
```
# 지역 변수와 전역 변수
g1 = 10
```

```
def func() :
    a = 20
    print(f'g1 = {g1}, a = {a}')

func()
```

In [ ]:
```
# 지역 변수와 전역 변수

def func() :
    a = 20
    print(f'g1 = {g1}, a = {a}')

g1 = 10

func()
```

In [ ]:
```
# 지역 변수와 전역 변수
# 함수 func()에서만 사용 가능한 지역 변수 a 접근(오류)
g1 = 10

def func() :
    a = 20
    print(f'g1 = {g1}, a = {a}')

func()
print(a)    # (오류) 함수 func()에서만 사용 가능한 지역 변수 a 참조
```

In [ ]:
```
# 지역 변수와 전역 변수
# 기본적으로 함수 func()에서는 지역 변수만 수정 가능
g1 = 10

def func() :
    a = 20
    g1 = g1 + 1    # (오류) 함수 func()에서 전역 변수 수정 시도
    print(f'func : g1 = {g1}, a = {a}')

func()
```

In [ ]:
```
# 지역 변수와 전역 변수
# 기본적으로 함수 func()에서는 지역 변수만 수정 가능
# 함수 func()에서 전역변수 g1을 수정하기 위해서는 g1이 전역변수임을 명시하면 사용

g1 = 10

def func() :
    global g1   # g1이 전역 변수임을 명시적으로 선언

    a = 20
    g1 = g1 + 1    # 전역변수로 명시된 g1은 함수 func()에서 수정 가능
    print(f'func : g1 = {g1}, a = {a}')

func()
```

In [ ]:
```
# 지역변수와 전역변수 이름이 같은 경우

def addone() :
    i = 30
    i += 1
```

```
    print(f'\t지역변수 i : {i}')

i = 20

print(f'호출 전-전역변수 i : {i}')
addone()
print(f'호출 후-전역변수 i : {i}')
```

In [ ]:
```
# 지역변수와 전역변수 이름이 같은 경우
i = 20
def addone() :
    i = 30
    i += 1

    print(f'\t지역변수 i : {i}')

print(f'호출 전-전역변수 i : {i}')
addone()
print(f'호출 후-전역변수 i : {i}')
```

LAB-3 수행 과제

1. 문제 : 지역변수와 전역변수의 차이 이해하기

- 아래의 요구사항을 만족하는 파이썬 프로그램을 작성하세요.래 조건에 따라 로그 데이터를 분석하세요.

2. 요구사항:

- 전역 변수 사용: 은행 계좌의 잔액을 나타내는 balance 변수를 전역 변수로 선언하세요.
- 지역 변수 사용: 입금(deposit)과 출금(withdraw) 기능을 **함수**로 구현하되, **함수 내부에서 지역 변수로 계산하여 전역 변수를 수정**하세요.
- 출력 요구사항:
  - 입금과 출금이 정상적으로 이루어졌을 때, 현재 잔액을 출력하세요.
  - 출금할 금액이 잔액보다 많으면 "잔액이 부족합니다"라는 메시지를 출력하세요.

```
# 전역 변수 선언
balance = 1000  # 초기 잔액 1000원

def deposit(amount):
    # 코드 작성
    #

def withdraw(amount):
    # 코드 작성
    #


# 실행 예제
deposit(500)   # 입금 실행
```

```
withdraw(200)  # 출금 실행
withdraw(2000) # 잔액 부족 상황 테스트
```

In [ ]:
```
# 전역 변수 선언
balance = 1000  # 초기 잔액 1000원

def deposit(amount):
    """입금 함수: 전역 변수 balance를 수정"""
    global balance  # 전역 변수 balance 사용 선언
    balance += amount
    print(f"{amount}원이 입금되었습니다. 현재 잔액: {balance}원")

def withdraw(amount):
    """출금 함수: 전역 변수 balance를 수정"""
    global balance  # 전역 변수 balance 사용 선언
    if amount > balance:
        print("잔액이 부족합니다.")
    else:
        balance -= amount
        print(f"{amount}원이 출금되었습니다. 현재 잔액: {balance}원")

# 실행 예제
deposit(500)   # 입금 실행
withdraw(200)  # 출금 실행
withdraw(2000) # 잔액 부족 상황 테스트
```

LAB-4 수행 과제(심화 과제)

1. 문제 : 지역변수와 전역변수의 차이 이해하기

- global 키워드를 사용하지 않고 함수에서 전역 변수를 수정할 수 있는 방법을 찾아보세요.

> Hint : global 키워드 없이 전역 변수를 수정하려면, 함수의 반환값을 활용

- **전역 변수를 직접 변경하는 것보다 유지보수가 쉽고, 함수가 독립적으로 동작할 수 있습니다.**

```
# 전역 변수 선언
balance = 1000  # 초기 잔액 1000원

def deposit(balance, amount):
    # 코드 작성
    #
    return balance

def withdraw(balance, amount):
    # 코드 작성
    #
    return balance

# 실행 예제
balance = deposit(balance, 500)
balance = withdraw(balance, 200)
balance = withdraw(balance, 2000)  # 잔액 부족 상황 테스트
```

LAB-5 수행 과제(내친김에 - 심화 과제)

1. 문제 : 지역 변수만을 사용하여 잔액을 관리하는 함수형 프로그래밍 방식

- 전역 변수를 아예 사용하지 않고, 지역 변수만을 사용하여 상태를 관리할 수도 있습니다.
- **전역 변수 없이 지역 변수로 상태를 유지하고 balance를 함수의 매개변수로 전달**
- **함수의 부작용을 최소화하고, 유지보수가 쉬운 코드 스타일을 제공**

```
def bank_operations():
    """함수형 프로그래밍 스타일로 잔액 관리"""
    balance = 1000  # 지역 변수로 잔액 관리

    def deposit(balance, amount):
        #
        # 코드 작성
        #
        return balance

    def withdraw(balance, amount):
        #
        # 코드 작성
        #
        return balance
```

```
    # 함수 실행
    balance = deposit(balance, 500)
    balance = withdraw(balance, 200)
    balance = withdraw(balance, 2000)  # 잔액 부족 테스트

bank_operations()
```

LAB-6 수행 과제(이왕에 그냥 가보자!!! - 심화 과제)

1. 문제 : 클래스를 사용하여 BankAccount 객체 만들기

- 객체 지향 프로그래밍(OOP)을 활용하여 은행 계좌를 BankAccount 클래스로 구현
- **self.balance를 사용하여 각 객체별로 독립적인 상태를 유지**
- **함수형 프로그래밍 방식과 달리, 객체 단위로 데이터를 관리하기 때문에 재사용성이 높아짐**

```
class BankAccount:
    def __init__(self, initial_balance=1000):
        """계좌 객체 초기화"""
        self.balance = initial_balance  # 인스턴스 변수로 잔액 관리

    def deposit(self, amount):
        """입금 메서드"""
        self.balance += amount
        print(f"{amount}원이 입금되었습니다. 현재 잔액: {self.balance}원")

    def withdraw(self, amount):
        """출금 메서드"""
        if amount > self.balance:
            print("잔액이 부족합니다.")
        else:
            self.balance -= amount
            print(f"{amount}원이 출금되었습니다. 현재 잔액:
{self.balance}원")

# 실행 예제
account = BankAccount()  # 계좌 객체 생성
account.deposit(500)  # 입금
account.withdraw(200)  # 출금
account.withdraw(2000)  # 잔액 부족 테스트
```

In [ ]: