CHAPTER 12

함수와 클래스

학습 목표

기본 학습 목표

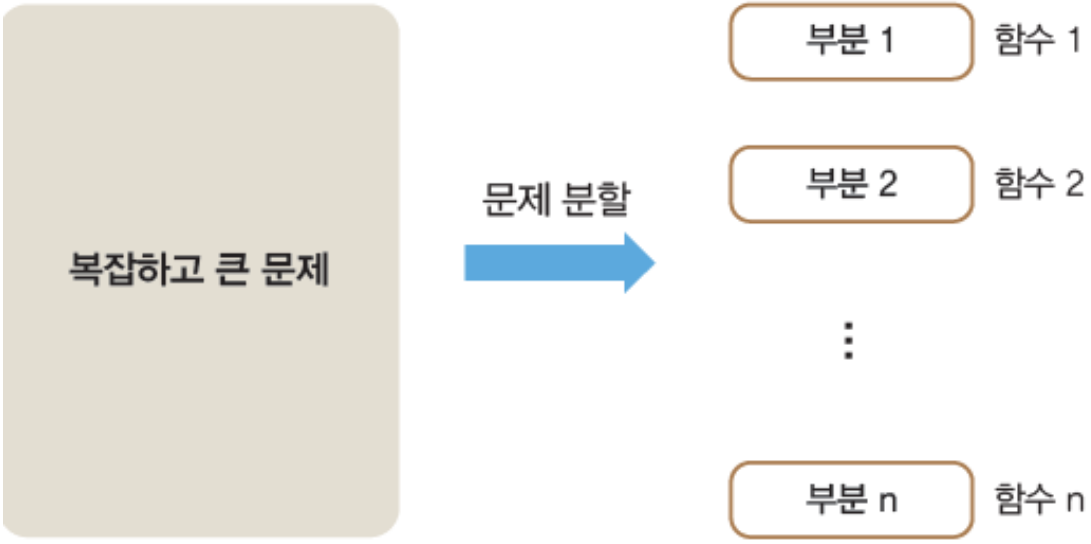
- 함수를 정의하고 호출할 수 있다.
- 함수 인자 전달과 리턴을 이해할 수 있다.
- 클래스를 정의하고 객체를 생성할 수 있다.
- 클래스와 객체의 차이를 이해할 수 있다.

심화 학습 목표

- 리턴 값이 하나 이상인 함수를 이용하여 프로그램을 작성할 수 있다.
- 전역 변수를 이해하고 필요시 사용할 수 있다.
- 클래스 생성자를 이해하고 사용할 수 있다.
- 예외 처리 상황을 효과적으로 처리하는 프로그램을 작성할 수 있다.

함수 –함수란?

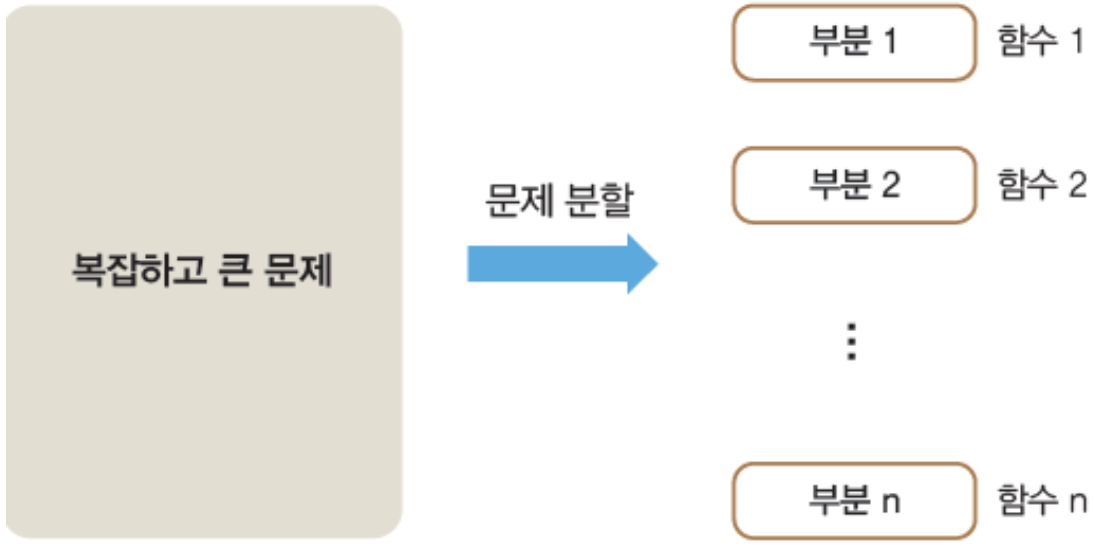
- 복잡한 문제를 분할
- 분할한 각 부분을 함수로 정의



[그림 12-1] 문제 분할과 함수 관계

함수 –함수란?

- 문제 분할 방법
  - 결합도(coupling) 낮고 응집도(cohesion) 높게
- 응집도
  - 하나의 함수 내에서문장들 간에 밀접한 정도
- 결합도
  - 각 함수 간에 관계 정도



[그림 12-1] 문제 분할과 함수 관계

함수 –함수란?

- 함수의 좋은 점
  - 프로그램을 수정, 재사용 용이
  - 프로그램 이해 용이
  - 함수 반복 호출시 프로그램 크기 감소 효과

묻고 답하기

Q 묻고 답하기

문 함수의 장단점은 무엇일까?

함수 –함수란?

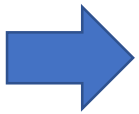
- 함수 정의
  - 키워드 def 다음에 함수명()

```
def 함수명():
      프로그램 문장1
      프로그램 문장2
      :
```

함수 –함수란?

- 함수 정의 예

```
print("합=", 50+60)
print("평균", (50+60)/2)
print("합=", 50+60)
print("평균", (50+60)/2)
```



```
def sumandavg():
  print("합=", 50+60)
  print("평균", (50+60)/2)
```

- 함수 호출 예

```
sumandavg()
sumandavg()
```

묻고 답하기

Q 묻고 답하기

문 함수명은 어떻게 만드는 것이 좋을까?

실습해보기

실습해보기 12-1

위의 프로그램을 for 문을 사용하여 동일한 수행 결과를 얻도록 수정해 보자.

함수 – 인자 전달

- 함수정의시 함수명 괄호 안에 전달받는 인자 나타냄
- 인자 전달 함수 정의

```
def sumandavg(a, b):
  print("합=", a+b)
  print("평균", (a+b)/2)
```

- 인자 전달 함수 호출

```
a=50
b=60
sumandavg(a, b)
sumandavg(a, b)
```

묻고 답하기

Q 묻고 답하기

문 인자 개수는 많은 게 좋을까? 아니면 적은 게 좋을까?

실습해보기

실습해보기 12-2

세 수의 합과 평균을 구하도록 인자가 a, b, c 세 개인 경우로 프로그램을 수정해 보자.

함수 – 리턴 값

- 함수에서 실행한 결과를 호출한 쪽으로 넘겨줌
- 함수 정의시 리턴의 예

```
def avg(a, b):
  return (a+b)/2
```

- 함수 호출후 리턴값 받아서 출력의 예

```
a=50
b=60
print("합=", a + b)
print("평균=", avg(a, b))
```

실습해보기

실습해보기 12-3

합에 대해서 리턴하는 함수를 정의하고 위의 프로그램을 이용해 작성해 보자.

함수 – 리턴 값 2개

- 2개 이상의 리턴 값이 가능함
- 2개의 값을 리턴하는 예

```
def sumandavg(a, b):
  sum = a+b
  avg = (a+b)/2
  return sum, avg
```

함수 – 리턴 값 2개

- 함수 호출로 2개의 리턴 값을 받는 예

```
a1=50
b1=60
sum1, avg1 = sumandavg(a1, b1)
print("합=", sum1)
print("평균=", avg1)

a2 =70
b2 =80

sum2, avg2 = sumandavg(a2, b2)

print("합=", sum2)
print("평균=", avg2)
```

수행 결과

```
합= 110
평균= 55.0
```

묻고 답하기

Q 묻고 답하기

문 리턴 값이 2개 이상 여러개 가능한가?

실습해보기

실습해보기 12-4

리턴 값의 순서를 바꿔서 평균과 합 순서로 리턴하도록 수정해 보자.

함수 – 변수의 참조 범위와 전역 변수

- 함수 내의 변수는 함수 내에서만 참조될 수 있음
- 다음 예제에서 함수 내 두 변수 a,b와 함수 밖 두 변수 a, b는 별개임

```
def avg():
  a=0
  b=0
  return (a+b)/2

a=80
b=60
print("평균=", avg())
print("합=", a + b)
```

수행 결과

```
평균= 0.0
합= 140
```

묻고 답하기

Q 묻고 답하기

문 변수의 참조 범위가 있으면 어떤 점이 좋을까?

함수 – 변수의 참조 범위와 전역 변수

- 함수 호출시 다음과 같이 인자로 전달하는 경우
  - 함수 내에서의 a,b에 대한 연산 결과가 함수 밖의 a, b에 영향을 주지 못함

```
def avg(a, b):
  a= a + 10
  b= b + 10
  return (a+b)/2

a=50
b=60

print("평균=", avg(a, b))
print("합=", a + b)
```

수행 결과

```
평균= 65.0
합= 110
```

함수 – 변수의 참조 범위와 전역 변수

- 두번째 예제
  - 함수 내에서의 연산 결과 값이 함수 밖에 영향을 주지 못함

```
def sumandavg(a, b):
  sum = a+b
  avg = (a+b)/2

sum =0
avg =0
a =50
b =60
```

```
sumandavg(a, b)
print("합=", sum)
print("평균=", avg)
```

수행 결과

```
합= 0
평균= 0
```

함수 – 변수의 참조 범위와 전역 변수

- 전역 변수
  - 변수 앞에 global 키워드를 붙임
- 예제

```
def sumandavg(a, b):
  global sum, avg
  sum = a+b
  avg = (a+b)/2

sum =0
avg =0
a =50
b =60

sumandavg(a, b)
print("합=", sum)
print("평균=", avg)
```

수행 결과

```
합= 110
평균= 55.0
```

실습해보기

실습해보기 12-5

위의 프로그램을 인자 전달을 통해 동일한 기능을 하도록 수정해 보자.

클래스 – 클래스란?

- 객체지향프로그래밍에서 객체를 만들기 위한 틀
- 데이터와 메소드(method)로 구성됨
- 함수와 클래스의 차이점
  - 데이터와 메소드들이 한 틀로 묶여 있음
  - 메소드가 접근할 수 있는 데이터를 제한시켜 데이터를 보호하거나 숨길 수 있음
  - 문제에 포함된 객체를 정확히 표현 가능

묻고 답하기

Q 묻고 답하기

문 문제를 해결할 때 언제 클래스를 사용하나요?

클래스 – 클래스란?

- 클래스 정의

```
class 클래스명:
   데이터 형 멤버명1
        :
   def 메소드명1():
     문장 1
     문장 2
        :
```

- 클래스 정의 예제

```
class Student:
   pass
```

클래스 – 객체 생성

- 객체
  - 클래스를 이용하여 생성됨
- 클래스와 객체 간의 관계
  - 클래스 –붕어빵 틀
  - 객체 - 붕어빵

클래스 객체

[그림 12-2] 클래스와 객체 간의 관계

- 객체 생성 방법

```
a=클래스명()
```

클래스 – 객체 생성

- 학생 클래스에 대한 객체 생성 예제

```
student1 = Student()
student1.name='Kim'
student1.no = '2019-123456'

print("학생이름=", student1.name)
print("학번=", student1.no)
```

수행 결과

```
학생이름= Kim
학번= 2019-123456
```

실습해보기

실습해보기 12-6

또 다른 학생을 student2라는 레퍼런스 명으로 객체를 생성해 보자.

클래스 – 객체 생성

- 데이터 속성을 클래스 멤버로 포함 예제
  - self는 인스턴스 객체 자신을 가리킴

```
class Student:
    name ="서진"
    no = "2019-123456"

    def prStudent(self):
        print("학생이름=", self.name)
        print("학번=", self.no)
student1 = Student()
student1.prStudent()
```

수행 결과

```
학생이름= Kim
학번= 2019-123456
```

묻고 답하기

Q 묻고 답하기

문 호출시 실제로 인자 전달이 없는데 메소드 인자에 self는 꼭 필요한가?

클래스 – 객체 초기화

- 객체 생성시 데이터 속성 초기화
  - 생성자 사용
    - __init__(self)
  - 예제

```
class Student:
    def __init__(self):
        self.name ="서진"
        self.no = "2019-123456"

    def prStudent(self):
        print("학생이름=", self.name)
        print("학번=", self.no)

student1 = Student()
student2 = Student()
```

```
student1.prStudent()
student2.prStudent()

student1.name ="준성"
student1.no ="2019-456789"

student1.prStudent()
```

수행 결과

```
학생이름= 서진
학번= 2019-123456
학생이름= 서진
학번= 2019-123456
학생이름= 준성
학번= 2019-456789
```

클래스 – 객체 초기화

- 객체 생성시 인자 전달 가능
- student1과 student2 인자 전달로 객체 생성

```
class Student:
    def __init__(self, name, studentNo):
        self.name = name
        self.no = studentNo

    def prStudent(self):
        print("학생이름=", self.name)
        print("학번=", self.no)

student1 = Student("준성", "2019-456789")
student1.prStudent()
```

수행 결과

```
학생이름= 준성
학번= 2019-456789
```

실습해보기

실습해보기 12-7

위의 프로그램에 다른 학생을 한 명 더 추가해 보자.

예외 처리

- 프로그램 수행 중에 에러 또는 예외 상황 발생할 수 있음
  - 예를 들면 0으로 나누는 경우
- 파이썬은 예외 상황 처리를 위해 try문 제공

```
try:
   예외 상황 발생 가능 코드

except [예외 유형]:
   try 절 수행 중에 지정된 유형의 예외 상황이 발생했을 경우 수행되는
   예외 처리 코드

else:
   try 절 수행 중에 예외 상황이 발생하지 않았을 경우 수행되는 코드
```

예외 처리

- 다음 예제는 0으로 나눈 경우 Zero-Division Error 이벤트 발생
- 예외 이벤트가 발생하지 않으면 else 절이 실행됨

```
try:
    dividend= int(input("피젯수를 입력하세요"))
    divisor = int(input("젯수를 입력하세요"))

    result = dividend/divisor

except ZeroDivisionError:

    print("피젯수가 0이므로 나눌 수 없습니다.")

except ValueError:
    print("피젯수나 젯수가 숫자가 아닙니다.")

else:
    print(result)
```

수행 결과 1

```
피젯수를 입력하세요7
젯수를 입력하세요x
피젯수나 젯수가 숫자가 아닙니다.
```

수행 결과 2

```
피젯수를 입력하세요7
젯수를 입력하세요0
피젯수가 0이므로 나눌 수 없습니다.
```

예외 처리

- 예외 처리 문법 구조

```
try:
   예외 상황 발생 가능 코드

except [예외 유형]:
   try 절 수행 중에 지정된 유형의 예외 상황이 발생했을 경우 수행되는
   예외 처리 코드

except [예외 유형1], [예외 유형2], ....:
   try 절 수행 중에 나열된 유형 중 어느 한 유형의
   예외 상황이 발생했을 경우 수행되는 예외 처리 코드

except [예외 유형] as 변수명:
   try 절 수행 중에 지정된 유형의 예외 상황이 발생했을 경우 수행되는
   예외 처리 코드

else:
   try 절 수행 중에 예외 상황이 발생하지 않았을 경우 수행되는 코드

finally:
   예외 상황 발생 여부와 상관없이 try 문 수행 완료 직전에 반드시 수행되는 코드
```