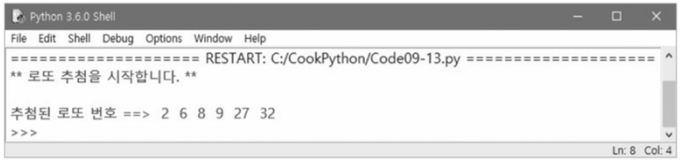
Section01 이 장에서 만들 프로그램

- **[프로그램] 로또 번호 추첨**



Section02 함수 기본

- **함수의 선언**

```
def 함수 이름 (매개변수 #1 …):
    수행문 1
    수행문 2
    return <반환값>
```

① **def :** 'definition'의 줄임말로, 함수를 정의하여 시작한다는 의미이다.

② **함수 이름 :** 함수 이름은 개발자가 마음대로 지정할 수 있지만, 파이썬에서는 일반적으로 다음과 같은 규칙을 사용한다.

- 소문자로 입력한다.
- 띄어쓰기를 할 경우에는 _ 기호를 사용한다. ex) save_model
- 행위를 기록하므로 동사와 명사를 함께 사용하는 경우가 많다. ex) find_number
- 외부에 공개하는 함수일 경우, 줄임말을 사용하지 않고 짧고 명료한 이름을 정한다.

Section02 함수 기본

- **함수의 선언**

```
def 함수 이름 (매개변수 #1 …):
    수행문 1
    수행문 2
    return <반환값>
```

③ **매개변수(parameter) :** 함수에서 입력값으로 사용하는 변수를 의미하며, 1개 이상의 값을 적을 수 있다.

④ **수행문 :** 수행문은 반드시 들여쓰기한 후 코드를 입력해야 한다. 수행해야 하는 코드는 일반적으로 작성하는 코드와 같다. if나 for 같은 제어문을 사용할 수도 있고, 고급 프로그래밍을 하게 되면 함수 안에 함수를 사용하기도 한다.

Section02 함수 기본

여기서 잠깐! **반환return**

- 약간 어렵게 느껴질 수 있는 부분이 바로 '반환'이라는 개념이다. 이는 수학에서의 함수와 같은 개념이라고 생각하면 된다. 예를 들어, 수학에서 f(x) = x + 1이라고 한다면 f(1)의 값은 얼마일까? 중학교 정도의 수학을 이해하고 있다면 f(1) = 2라는 것을 알 것이다. 즉, 함수 f(x)에서 x에 1이 들어가면 2가 반환되는 것이다. 파이썬의 함수도 같은 개념이다. 수학에서 x에 해당하는 것이 매개변수, 즉 입력값이고, x + 1의 계산 과정이 함수 안의 코드이며, 그 결과가 출력값이다.

Section02 함수 기본

여기서 잠깐! 매개변수와 인수

- 매개변수는 함수의 인터페이스 정의에 있어 어떤 변수를 사용하는지를 정의하는 것이다. 그에 반해 인수는 실제 매개변수에 대입되는 값을 뜻한다.

코드 5-3 parameter.py

```
1 def f(x):
2     return 2 * x + 7
3
4 print(f(2))
```

```
11
```

➡ [코드 5-3]에서 'def f(x):'의 x를 매개변수라고 한다. 일반적으로 함수의 입력값에 대한 정의를 함수 사용에 있어 인터페이스를 정의한다고 한다. 매개변수는 함수의 인터페이스 정의에 있어 어떤 변수를 사용하는지를 정의하는 것이다. 즉, 위 함수에서는 x가 해당 함수의 매개변수이다. 그에 반해, 인수는 실제 매개변수에 대입되는 값을 뜻한다. 매개변수가 설계도라면 인수는 그 설계도로 지은 건물 같은 것이다. 위 코드에서는 f(2)에서 2가 인수에 해당한다.

Section02 함수 기본

■ 함수의 형태

매개변수 유무 / 반환값 유무	매개변수 없음	매개변수 있음
반환값 없음	함수 안 수행문만 수행	매개변수를 사용하여 수행문만 수행
반환값 있음	매개변수 없이 수행문을 수행한 후, 결과값 반환	매개변수를 사용하여 수행문을 수행한 후, 결과값 반환

[함수의 형태]

Section02 함수 기본

■ 함수의 개념과 필요성

- 함수(Function) : '무엇'을 넣으면, '어떤 것'을 돌려주는 요술 상자
- 메서드(Method)와 차이점 : 함수는 외부에 별도로 존재, 메서드는 클래스 안에 존재
- 함수의 형식

```
함수명()
```

- print() 함수

```
print("CookBook-파이썬")
```


Section02 함수 기본

- 커피를 타는 과정

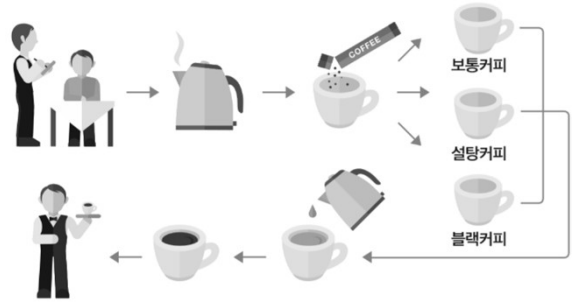


그림 9-1 **직접 커피를 타는 과정**

Section02 함수 기본

▪ 커피를 타는 과정의 코드

Code09-01.py

```
1  coffee = 0
2
3  coffee = int(input("어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙)") )
4
5  print()
6  print("#1. 뜨거운 물을 준비한다.");
7  print("#2. 종이컵을 준비한다.");
8
9  if coffee == 1 :
10     print("#3. 보통커피를 탄다.")
11 elif coffee == 2 :
12     print("#3. 설탕커피를 탄다.")
13 elif coffee == 3 :
14     print("#3. 블랙커피를 탄다.")
15 else :
16     print("#3. 아무거나 탄다.\n")
```

1행 : 커피의 종류를 입력받을 변수를 선언
3행 : 손님에게 커피의 종류 입력
6~7행 : 물과 컵을 준비하는 메시지를 출력
9~16행 : 손님이 주문한 커피 종류에 따른 메시지 출력

9/69

Section02 함수 기본

```
17
18 print("#4. 물을 붓는다.");
19 print("#5. 스푼으로 젓는다.");
20 print()
21 print("손님~ 커피 여기 있습니다.");
```

18~19행 : 물을 붓고 커피를 스푼으로 잘 저어 녹인 후
21행 : 완성된 커피를 손님에게 제공

출력 결과

```
어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙) 2
#1. 뜨거운 물을 준비한다.
#2. 종이컵을 준비한다.
#3. 설탕커피를 탄다.
#4. 물을 붓는다.
#5. 스푼으로 젓는다.
손님~ 커피 여기 있습니다.
```

10/69

Section02 함수 기본

▪ 손님 3명이 연속해서 들어온다고 했을 때(Code09-01.py의 3~21행을 2번 반복)

```
coffee = int(input("어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙)"))

print()
print("#1. 뜨거운 물을 준비한다.");
print("#2. 종이컵을 준비한다.");

if coffee == 1 :
    print("#3. 보통커피를 탄다.")
elif coffee == 2 :
    print("#3. 설탕커피를 탄다.")
elif coffee == 3 :
    print("#3. 블랙커피를 탄다.")
else :
    print("#3. 아무거나 탄다.\n")

print("#4. 물을 붓는다.");
print("#5. 스푼으로 젓는다.");
print()
print("손님~ 커피 여기 있습니다.");

    # 두 번째 손님용 코드 다시 반복

    # 세 번째 손님용 코드 다시 반복
```

11/69

Section02 함수 기본

▪ 손님 3명이 연속해서 들어온다고 했을 때(커피 자판기 만들기)

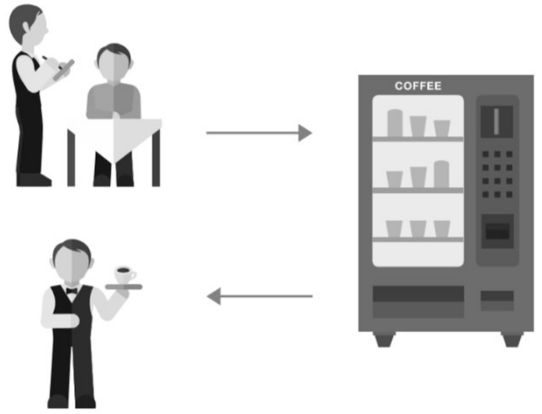


그림 9-2 커피 자판기 사용 예

12/69

Section02 함수 기본

- 커피 자판기 실제 프로그램으로 만들기(함수 개념 응용 Code09-01.py 수정)

Code09-02.py

```
## 전역 변수 선언 부분 ##
coffee = 0

## 함수 선언 부분 ##
def coffee_machine(button) :
    print()
    print("#1. (자동으로) 뜨거운 물을 준비한다.");
    print("#2. (자동으로) 종이컵을 준비한다.");

    if button == 1 :
        print("#3. (자동으로) 보통커피를 탄다.")
    elif button == 2 :
        print("#3. (자동으로) 설탕커피를 탄다.")
    elif button == 3 :
        print("#3. (자동으로) 블랙커피를 탄다.")
```


Section02 함수 기본

```
    else :
        print("#3. (자동으로) 아무거나 탄다.\n")

    print("#4. (자동으로) 물을 붓는다.");
    print("#5. (자동으로) 스푼으로 젓는다.");
    print()
## 메인 코드 부분 ##
coffee = int(input("어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙)"))
coffee_machine(coffee)
print("손님~ 커피 여기 있습니다.");
```

24행 : 손님에게 커피 종류를 물어봄
25행 : 커피 자판기만 있음
26행 : 손님에게 커피를 대접

출력 결과

```
어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙) 2
#1. (자동으로) 뜨거운 물을 준비한다.
#2. (자동으로) 종이컵을 준비한다.
#3. (자동으로) 설탕커피를 탄다.
#4. (자동으로) 물을 붓는다.
#5. (자동으로) 스푼으로 젓는다.
손님~ 커피 여기 있습니다.
```


Section02 함수 기본

- Code09-02.py를 이용 연속해서 방문한 손님(A, B, C) 3명에게 커피 대접

Code09-03.py

```
## 전역 변수 선언 부분 ##
coffee = 0

## 함수 선언 부분 ##
def coffee_machine(button) :
... # (중략) Code09-02.py의 6~21행과 동일

## 메인 코드 부분 ##
coffee = int(input("A손님, 어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙)"))
coffee_machine(coffee)
print("A손님~ 커피 여기 있습니다.")

coffee = int(input("B손님, 어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙)"))
coffee_machine(coffee)
print("B손님~ 커피 여기 있습니다.")

coffee = int(input("C손님, 어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙)"))
coffee_machine(coffee)
print("C손님~ 커피 여기 있습니다.")
```


Section02 함수 기본

출력 결과

```
A손님, 어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙) 2
#1. (자동으로) 뜨거운 물을 준비한다.
#2. (자동으로) 종이컵을 준비한다.
#3. (자동으로) 설탕커피를 탄다.
#4. (자동으로) 물을 붓는다.
#5. (자동으로) 스푼으로 젓는다.
A손님~ 커피 여기 있습니다.
B손님, 어떤 커피 드릴까요?(1:보통, 2:설탕, 3:블랙) 3
… 생략 …
```

Section02 함수 기본

SELF STUDY 9-1

Code09-03.py를 수정해서 커피 종류를 아메리카노, 카페라떼, 카푸치노, 에스프레소 중 하나를 선택할 수 있도록 하자. 그리고 로제, 리사, 지수, 제니라는 손님 4명의 주문을 받아 보자.

출력 결과

```
로제씨, 어떤 커피 드릴까요?(1:아메리카노, 2:카페라떼, 3:카푸치노, 4:에스프레소) 4
#1. (자동으로) 뜨거운 물을 준비한다.
#2. (자동으로) 종이컵을 준비한다.
#3. (자동으로) 에스프레소를 탄다.
#4. (자동으로) 물을 붓는다.
#5. (자동으로) 스푼으로 젓는다.
로제씨~ 커피 여기 있습니다.
… 생략 …
```


Section02 함수 기본

▪ **함수의 형식과 활용**

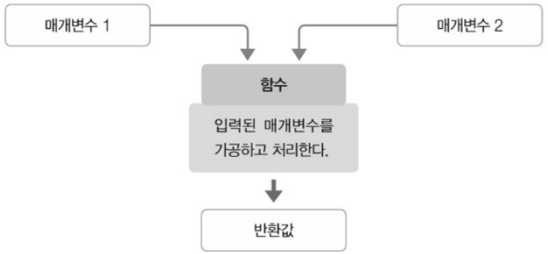


그림 9-3 **함수의 기본 형식**

Section02 함수 기본

▪ plus() 함수

Code09-04.py

```
1  ## 함수 선언 부분 ##
2  def plus(v1, v2) :
3      result = 0
4      result = v1 + v2
5      return result
6
7  ## 전역 변수 선언 부분 ##
8  hap = 0
9
10 ## 메인 코드 부분 ##
11 hap = plus(100, 200)
12 print("100과 200의 plus() 함수 결과는 %d" % hap)
```

출력 결과

```
100과 200의 plus() 함수 결과는 300
```


Section02 함수 기본

▪ plus() 함수

Code09-04.py

```
1  ## 함수 선언 부분 ##
2  def plus(v1, v2) :
3      result = 0
4      result = v1 + v2
5      return result
6
7  ## 전역 변수 선언 부분 ##
8  hap = 0
9
10 ## 메인 코드 부분 ##
11 hap = plus(100, 200)
12 print("100과 200의 plus() 함수 결과는 %d" % hap)
```

2~5행 : plus() 함수를 정의
4행 : 매개변수로 받은 두 값의 합계를 구함
5행 : 반환
11행 : 100, 200 두 값 을 전달하면서 plus() 함수를 호출해 hap에 대입
12행 : plus() 함수에서 반환된 값 출력

출력 결과

```
100과 200의 plus() 함수 결과는 300
```

Section02 함수 기본

- plus() 함수의 형식과 호출 순서

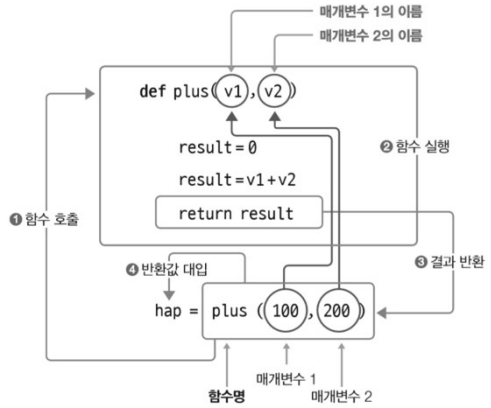


그림 9-4 plus( ) 함수의 형식과 호출 순서

Section02 함수 기본

- plus() 함수의 형식과 호출 순서

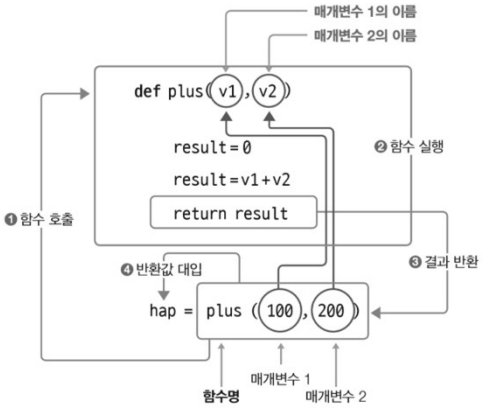


그림 9-4 plus( ) 함수의 형식과 호출 순서

Section02 함수 기본

- plus() 함수의 호출 간략 표현

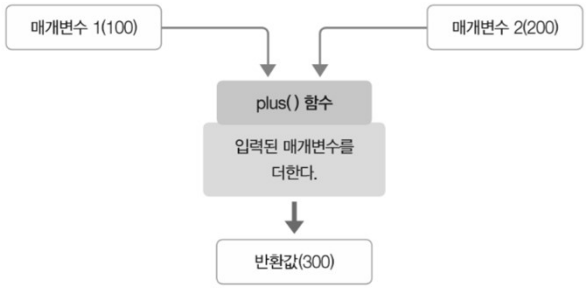


그림 9-5 plus( ) 함수의 호출 간략 표현

Section02 함수 기본

- 덧셈, 뺄셈, 곱셈, 나눗셈을 하는 계산기 함수를 작성

Code09-05.py

```
1  ## 함수 선언 부분 ##
2  def calc(v1, v2, op) :
3      result = 0
4      if op == '+' :
5          result = v1 + v2
6      elif op == '-' :
7          result = v1 - v2
8      elif op == '*' :
9          result = v1 * v2
10     elif op == '/' :
11         result = v1 / v2
12
13     return result
14
15 ## 전역 변수 선언 부분 ##
16 res = 0
17 var1, var2, oper = 0, 0, ""
18
19 ## 메인 코드 부분 ##
20 oper = input("계산을 입력하세요(+, -, *, /) : ")
21 var1 = int(input("첫 번째 수를 입력하세요 : "))
```

2~13행 : 매개변수를 3개 받는 calc() 함수를 정의
20행 : 어떤 연산을 할지 연산자를 입력
21~22행 : 계산할 숫자 2개를 입력
24행 : calc() 함수에 매개 변수 3개를 넘겨주며 호출
4~11행 : 사용자가 입력한 연산자에 따라 필요한 연산을 수행
13행 : 계산된 값을 return으로 반환
24행 : calc() 함수에서 반환한 값을 res에 넣음
26행 : 계산식 출력

Section02 함수 기본

```
22  var2 = int(input("두 번째 수를 입력하세요 : "))
23
24  res = calc(var1, var2, oper)
25
26  print("## 계산기 : %d %s %d = %d" % (var1, oper, var2, res))
```

출력 결과

```
계산을 입력하세요(+, -, * , /) : *
첫 번째 수를 입력하세요 : 7
두 번째 수를 입력하세요 : 8
## 계산기 : 7 * 8 = 56
```

Tip • 매개변수(파라미터)는 지역 변수로 취급. Code09-05.py의 calc( ) 함수가 받는 매개변수 v1, v2, op는 모두 calc() 함수 안에서만 사용되는 지역 변수. 지역 변수와 전역 변수는 바로 이어서 설명

Section02 함수 기본

SELF STUDY 9-2

Code09-05.py에 다음 기능을 추가해 보자.

① 숫자1, 연산자, 숫자2 순서로 입력받는다.

② 제곱(**) 연산자를 추가한다.

③ 0으로 나누려고 하면 메시지를 출력하고 계산되지 않도록 한다.

힌트 메인 코드 부분에 if~else 문을 활용한다.

출력 결과

```
첫 번째 수를 입력하세요 : 2
계산을 입력하세요(+, -, *, /, **) : **
두 번째 수를 입력하세요 : 4
## 계산기 : 2 ** 4 = 16
```

출력 결과

```
첫 번째 수를 입력하세요 : 8
계산을 입력하세요(+, -, *, /, **) : /
두 번째 수를 입력하세요 : 0
0으로는 나누면 안 됩니다.ㅠㅠ
```


Section03 지역 변수, 전역 변수

▪ 함수의 호출 방식

- 함수 밖에 있는 변수 x의 메모리 주소와 함수 안에 있는 변수 x의 메모리 주소가 같은지 다른지 확인할 필요가 있다.
- 함수 안에 변수가 인수로 들어가 사용될 때, 변수를 호출하는 방식을 전통적인 프로그래밍에서는 다음과 같이 크게 두 가지로 나눈다.

종류	설명
값에 의한 호출 (call by value)	• 함수에 인수를 넘길 때 값만 넘김 • 함수 안의 인수값 변경 시, 호출된 변수에 영향을 주지 않음
참조 호출 (call by reference)	• 함수에 인수를 넘길 때 메모리 주소를 넘김 • 함수 안의 인수값 변경 시, 호출된 변수값도 변경됨

[함수가 변수를 호출하는 방식]

Section03 지역 변수, 전역 변수

▪ 지역 변수와 전역 변수의 이해

- 지역 변수 : 한정된 지역에서만 사용
- 전역 변수 : 프로그램 전체에서 사용

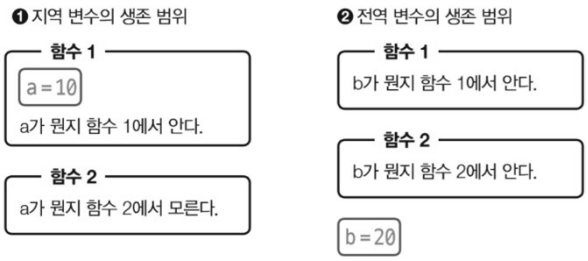


그림 9-6 **지역 변수와 전역 변수의 생존 범위**

Section03 지역 변수, 전역 변수

- **지역 변수와 전역 변수의 이해**
  - 지역 변수 : 한정된 지역에서만 사용
  - 전역 변수 : 프로그램 전체에서 사용

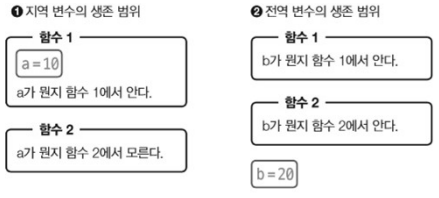


그림 9-6 **지역 변수와 전역 변수의 생존 범위**

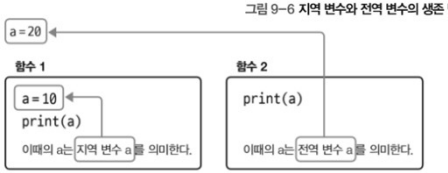


그림 9-7 **지역 변수와 전역 변수의 공존**

Section03 지역 변수, 전역 변수

Code09-06.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      a = 10    # 지역 변수
4      print("func1()에서 a값 %d" % a)
5
6  def func2() :
7      print("func2()에서 a값 %d" % a)
8
9  ## 전역 변수 선언 부분 ##
10 a = 20        # 전역 변수
11
12 ## 메인 코드 부분 ##
13 func1()
14 func2()
```

2~4행 : 한 func1() 함수 정의
3행 : a를 선언(지역 변수)
10행 : a는 선언(전역 변수)
13행 : func1() 함수 호출
14행 : func2() 함수 호출

출력 결과

```
func1()에서 a값 10
func2()에서 a값 20
```


Section03 지역 변수, 전역 변수

- 10행의 전역 변수가 없다면 7행은?

출력 결과

```
func1()에서 a의 값 10
Traceback (most recent call last):
    File "C:/파이썬코드/09-06.py", line 14, in <module>
        func2()
    File "C:/파이썬코드/09-06.py", line 7, in func2
        print("func2()에서 a값 %d" % a)
NameError: name 'a' is not defined
```


Section03 지역 변수, 전역 변수

- **글로벌 예약어**

Code09-07.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      global a   # 이 함수 안에서 a는 전역 변수
4      a = 10
5      print("func1()에서 a값 %d" % a)
6
7  def func2() :
8      print("func2()에서 a값 %d" % a)
9
10 ## 함수 변수 선언 부분 ##
11 a = 20          # 전역 변수
12
13 ## 메인 코드 부분 ##
14 func1()
15 func2()
```

출력 결과

```
func1()에서 a값 10
func2()에서 a값 10
```

Section03 지역 변수, 전역 변수

■ **글로벌 예약어**

Code09-07.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      global a    # 이 함수 안에서 a는 전역 변수
4      a = 10
5      print("func1()에서 a값 %d" % a)
6
7  def func2() :
8      print("func2()에서 a값 %d" % a)
9
10 ## 함수 변수 선언 부분 ##
11 a = 20           # 전역 변수
12
13 ## 메인 코드 부분 ##
14 func1()
15 func2()
```

3행 : global 예약어로 a 변수를 전역 변수로 지정
4행 : 전역 변수 a값을 10으로 변경하므로 func1()과 func2() 함수에서 모두 전역 변수 a값을 10으로 출력

출력 결과

```
func1()에서 a값 10
func2()에서 a값 10
```

33/69

Section04 함수의 반환값과 매개변수

■ **함수의 반환값**

Tip • 반환값은 return 문으로 반환되므로 리턴값이라고도 함. 매개변수는 파라미터라고도 함

▪ 반환값이 있는 함수

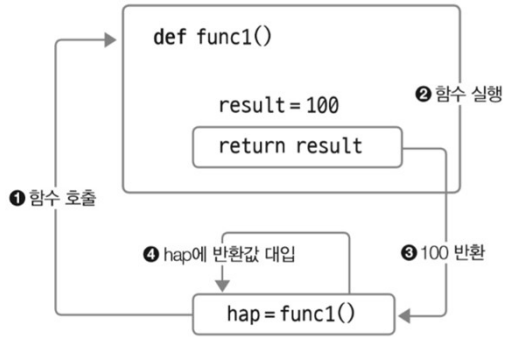


그림 9-8 **값의 반환**

34/69

Section04 함수의 반환값과 매개변수

▪ 반환값이 없는 함수

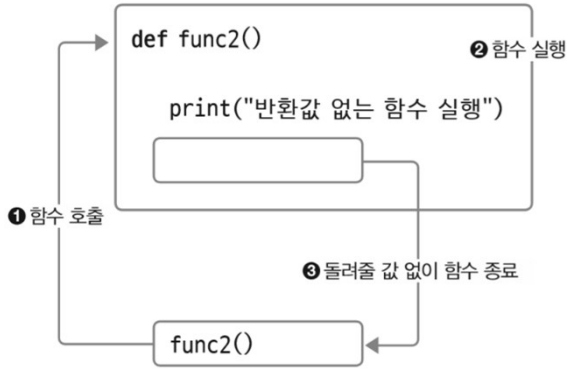


그림 9-9 **반환값이 없는 함수의 작동**

35/69

Section04 함수의 반환값과 매개변수

▪ 반환값이 없는 함수

Code09-08.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      result = 100
4      return result
5
6  def func2() :
7      print("반환값이 없는 함수 실행")
8
9  ## 전역 변수 선언 부분 ##
10 hap = 0
11
12 ## 메인 코드 부분 ##
13 hap = func1()
14 print("func1()에서 돌려준 값 ==> %d" % hap)
15 func2()
```

13행 : 반환값이 있는 함수인 func1()을 호출하면 func1() 실행 후 func1()의 반환값을 hap에 넣고
14행 : 출력
15행 : 반환값이 없는 함수인 func2()를 호출하면 반환 않음

출력 결과

```
func1()에서 돌려준 값 ==> 100
반환값이 없는 함수 실행
```

36/69

Section04 함수의 반환값과 매개변수

- 반환값이 여러 개인 함수

Code09-09.py

```
1  ## 함수 선언 부분 ##
2  def multi(v1, v2) :
3      retList = []      # 반환할 리스트
4      res1 = v1 + v2
5      res2 = v1 - v2
6      retList.append(res1)
7      retList.append(res2)
8      return retList
9
10 ## 전역 변수 선언 부분 ##
11 myList = []
12 hap, sub = 0, 0
13
14 ## 메인 코드 부분 ##
15 myList = multi(100, 200)
16 hap = myList[0]
17 sub = myList[1]
18 print("multi()에서 돌려준 값 ==> %d, %d" % (hap, sub))
```

3행 : 빈 리스트를 준비
6~7행 : 리스트에 값을 추가
8행 : 리스트 반환
15~17행 : 반환한 리스트의 값을 각 변수에 대입

출력 결과

```
multi()에서 돌려준 값 ==> 300, -100
```


Section04 함수의 반환값과 매개변수

- pass 예약어

```
def myFunc() :
    pass
```

- True일 때 아무런 할 일이 없다고 빈 줄로 둘 때 오류 발생

```
if True :

else :
    print('거짓이네요')
```

- 오류 해결

```
if True :
    pass
else :
    print('거짓이네요')
```


Section04 함수의 반환값과 매개변수

▪ 함수의 인수

- 파이썬에서 인수를 사용하는 방법에 대해 알아보자

종류	내용
키워드 인수	함수의 인터페이스에 지정된 변수명을 사용하여 함수의 인수를 지정하는 방법
디폴트 인수	별도의 인수값이 입력되지 않을 때, 인터페이스 선언에서 지정한 초깃값을 사용하는 방법
가변 인수	함수의 인터페이스에 지정된 변수 이외의 추가 변수를 함수에 입력할 수 있게 지원하는 방법
키워드 가변 인수	매개변수의 이름을 따로 지정하지 않고 입력하는 방법

[파이썬에서 인수를 사용하는 방법]

Section04 함수의 반환값과 매개변수

▪ 함수의 매개변수 전달

- 매개변수의 개수를 지정해 전달하는 방법
  - 숫자 2개의 합과 숫자 3개의 합을 구 하는 코드

Code09-10.py

```
1  ## 함수 선언 부분 ##
2  def para2_func( v1, v2) :
3      result = 0
4      result = v1 + v2
5      return result
6
7  def para3_func( v1, v2, v3) :
8      result = 0
9      result = v1 + v2 + v3
10     return result
11
12 ## 전역 변수 선언 부분 ##
13 hap = 0
```

2~5행은매개변수를 2개, 7~10행은 매개변수를 3개 받아 합계를 반환하는 함수 정의

Section04 함수의 반환값과 매개변수

16~19행 : 각 함수를 호출하고 결과 출력

```
14
15  ## 메인 코드 부분 ##
16  hap = para2_func(10, 20)
17  print("매개변수가 2개인 함수를 호출한 결과 ==> %d" % hap)
18  hap = para3_func(10, 20, 30)
19  print("매개변수가 3개인 함수를 호출한 결과 ==> %d" % hap)
```

출력 결과

```
매개변수가 2개인 함수를 호출한 결과 ==> 30
매개변수가 3개인 함수를 호출한 결과 ==> 60
```


Section04 함수의 반환값과 매개변수

- 매개변수에 기본값을 설정해 놓고 전달하는 방법

Code09-11.py

```
1  ## 함수 선언 부분 ##
2  def para_func( v1, v2, v3 = 0 ) :
3      result = 0
4      result = v1 + v2 + v3
5      return result
6
7  ## 전역 변수 선언 부분 ##
8  hap = 0
9
10 ## 메인 코드 부분 ##
11 hap = para_func(10, 20)
12 print("매개변수가 2개인 함수를 호출한 결과 ==> %d" % hap)
13 hap = para_func(10, 20, 30)
14 print("매개변수가 3개인 함수를 호출한 결과 ==> %d" % hap)
```


Section04 함수의 반환값과 매개변수

SELF STUDY 9-3

Code09-11.py에서 2에서 10개까지 몇 개를 매개변수로 사용하든지 합계를 구하도록 para_func() 함수를 수정해 보자.

출력 결과

```
매개변수가 2개인 함수를 호출한 결과 ==> 30
매개변수가 10개인 함수를 호출한 결과 ==> 550
```


Section04 함수의 반환값과 매개변수

- 매개변수의 개수를 지정하지 않고 전달하는 방법

Code09-12.py

```
1  ## 함수 선언 부분 ##
2  def para_func (*para) :
3      result = 0
4      for num in para :
5          result = result + num
6
7      return result
8
9  ## 전역 변수 선언 부분 ##
10 hap = 0
11
12 ## 메인 코드 부분 ##
13 hap = para_func(10, 20)
14 print("매개변수가 2개인 함수를 호출한 결과 ==> %d" % hap)
15 hap = para_func(10, 20, 30)
16 print("매개변수가 3개인 함수를 호출한 결과 ==> %d" % hap)
```

2행 : *para로 매개변수 설정
13행 : 호출한 매개변수는 (10, 20) 형식의 튜플로 전달
15행 : 호출한 매개변수는 (10, 20, 30) 형식의 튜플로 전달

Section04 함수의 반환값과 매개변수

- (10, 20, 30)을 매개 변수로 받았을 때 4~5행의 반복
  - 1회 : num에 10을 저장한 후 result = result + 10 result에 10 저장됨
  - 2회 : num에 20을 저장한 후 result = result + 20 result에 30 저장됨
  - 3회 : num에 30을 저장한 후 result = result + 30 result에 60 저장됨
- 매개변수 10개 이상일 때

```
hap = para_func(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
print(hap)
```

출력 결과

```
550
```


Section04 함수의 반환값과 매개변수

- 함수 호출할 때 딕셔너리 형식의 매개변수를 키=값 형식으로 사용

```
def dic_func(**para) :
    for k in para.keys() :
        print("%s  --> %d명입니다." % (k, para[k]))

dic_func(트와이스 = 9, 소녀시대 = 7, 걸스데이 = 4, 블랙핑크 = 4)
```

출력 결과

```
트와이스  --> 9명입니다.
소녀시대  --> 7명입니다.
걸스데이  --> 4명입니다.
블랙핑크  --> 4명입니다.
```


Section05 모듈

- **모듈 : 함수의 집합**

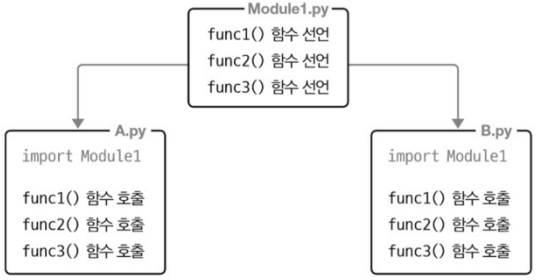


그림 9-10 **모듈 사용 예**

Section05 모듈

- **모듈의 생성과 사용**

Module1.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      print("Module1.py의 func1()이 호출됨.")
4
5  def func2() :
6      print("Module1.py의 func2()가 호출됨.")
7
8  def func3() :
9      print("Module1.py의 func3()이 호출됨.")
```

Section05 모듈

A.py

```
1 import Module1
2
3 ## 메인 코드 부분 ##
4 Module1.func1()
5 Module1.func2()
6 Module1.func3()
```

출력 결과

```
Module1.py의 func1()이 호출됨.
Module1.py의 func2()가 호출됨.
Module1.py의 func3()이 호출됨.
```

- 모듈명을 생략하고 함수명만 쓸 때 1행 형식

```
from 모듈명 import 함수1, 함수2, 함수3
또는
from 모듈명 import *
```


Section05 모듈

- 4~6행 함수명으로만 호출

B.py

```
1 from Module1 import func1, func2, func3    # 또는 from Module1 import *
2
3 ## 메인 코드 부분 ##
4 func1()
5 func2()
6 func3()
```


Section05 모듈

- **모듈의 종류**
  - 표준 모듈, 사용자 정의 모듈, 서드 파티 모듈로 구분
  - 표준 모듈 : 파이썬에서 제공하는 모듈
  - 사용자 정의 모듈 : 직접 만들어서 사용하는 모듈
  - 서드 파티(3rd Party) 모듈 : 파이썬이 아닌 외부 회사나 단체에서 제공하는 모듈
    - 파이썬 표준 모듈이 모든 기능을 제공 않음
    - 서드 파티 모듈 덕분에 파이썬에서도 고급 프로그래밍 가능
    - 게임 개발 기능이 있는 pyGame, 윈도창을 제공 하는 PyGTK, 데이터베이스 기능의 SQLAlchemy 등

Section05 모듈

- 파이썬에서 제공하는 표준 모듈의 목록을 일부 확인

```
import sys
print(sys.builtin_module_names)
```

출력 결과

```
('_ast', '_bisect', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022', '_codecs_
jp', '_codecs_kr', '_codecs_tw', '_collections', '_csv', '_datetime', '_functools',
'_heapq', '_imp', '_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec',
'_opcode', '_operator', '_pickle', '_random', '_sha1', '_sha256', '_sha512', '_signal',
'_sre', '_stat', '_string', '_struct', '_symtable', '_thread', '_tracemalloc',
'_warnings', '_weakref', '_winapi', 'array', 'atexit', 'audioop', 'binascii', 'builtins',
'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt',
'nt', 'parser', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib')
```

Section05 모듈

- 파이썬에서 제공하는 표준 모듈의 목록을 일부 확인

```
import sys
print(sys.builtin_module_names)
```

출력 결과

```
('_ast', '_bisect', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022', '_codecs_
jp', '_codecs_kr', '_codecs_tw', '_collections', '_csv', '_datetime', '_functools',
'_heapq', '_imp', '_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec',
'_opcode', '_operator', '_pickle', '_random', '_sha1', '_sha256', '_sha512', '_signal',
'_sre', '_stat', '_string', '_struct', '_symtable', '_thread', '_tracemalloc',
'_warnings', '_weakref', '_winapi', 'array', 'atexit', 'audioop', 'binascii', 'builtins',
'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt',
'nt', 'parser', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib')
```

Tip • dir(__builtins__) 명령어로도 제공하는 모듈과 예약어 확인

53/69

Section05 모듈

- 수학 계산 모듈인 math 모듈이 제공하는 함수의 목록 보기

```
import math
dir(math)
```

출력 결과

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

54/69

Section06 함수의 심화 내용

- **패키지**
  - 모듈이 하나의 *.py 파일 안에 함수가 여러 개 들어 있는 것이라면, 패키지(Package)는 여러 모듈을 모아 놓은 것으로 폴더의 형태로 나타냄
  - 모듈을 주제별로 분리할 때 주로 사용

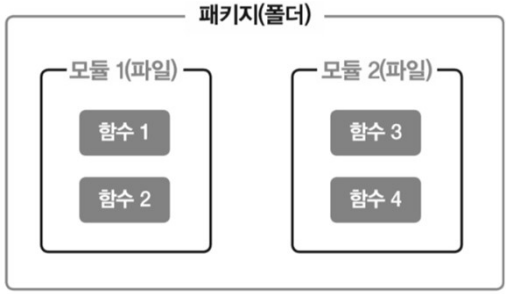


그림 9-11 패키지의 개념

55/69

Section06 함수의 심화 내용

- 임포트 형식

```
from 패키지명.모듈명 import 함수명
```

- [그림 9-11]과 같은 형태로 구현 임포트

```
from package.Module1 import *
```

56/69

Section06 함수의 심화 내용

- **내부 함수, lambda, map()**
  - 내부 함수 : 함수 안에 함수가 있는 형태

```
def outFunc(v1, v2) :
    def inFunc(num1, num2) :
        return num1 + num2
    return inFunc(v1, v2)
print(outFunc(10, 20))
```

출력 결과

```
30
```

  - outFunc() 함수 밖에서 호출하면 오류 발생

```
print(inFunc(10, 20))
```

출력 결과

```
NameError: name 'inFunc' is not defined
```


Section06 함수의 심화 내용

- 람다 함수 : 함수를 한 줄로 간단하게 만들어 줌

```
def hap(num1, num2) :
    res = num1 + num2
    return res
print(hap(10, 20))
```

출력 결과

```
30
```

```
hap2 = lambda num1, num2 : num1 + num2
print(hap2(10, 20))
```


Section06 함수의 심화 내용

- 매개변수에 기본값을 설정

```
hap3 = lambda num1 = 10, num2 = 20 : num1 + num2
print(hap3())
print(hap3(100, 200))
```

출력 결과

```
30
300
```

  - 매개변수를 지정하지 않으면 기본값으로 설정, 매개변수를 넘겨주면 기본값 무시

Section06 함수의 심화 내용

- 리스트에 모두 10을 더하는 코드

```
myList = [1, 2, 3, 4, 5]
def add10(num) :
    return num + 10

for i in range(len(myList)) :
    myList[i] = add10(myList[i])
print(myList)
```

출력 결과

```
[11, 12, 13, 14, 15]
```

  - 람다 함수와 map() 함수로 간단히

```
1 myList = [1, 2, 3, 4, 5]
2 add10 = lambda num : num + 10
3 myList = list(map(add10, myList))
4 print(myList)
```

Section06 함수의 심화 내용

▪ 2행과 3행을 합친 코드

```
myList = [1, 2, 3, 4, 5]
myList = list(map(lambda num : num + 10, myList))
print(myList)
```

▪ 두 리스트 의 각 자릿수를 합쳐서 새로운 리스트로 만들기

```
list1 = [1, 2, 3, 4]
list2 = [10, 20, 30, 40]
hapList = list(map(lambda n1, n2 : n1 + n2, list1, list2))
print(hapList)
```

출력 결과

```
[11, 22, 33, 44]
```


Section06 함수의 심화 내용

■ **재귀 함수**

- **재귀 함수(recursive function) :** 함수가 자기 자신을 다시 부르는 함수이다.

$$n! = n \cdot (n-1) \cdots 2 \cdot 1 = \prod_{i=1}^{n} i$$

$$\begin{aligned} 1! &= 1 \\ 2! &= 2(1) = 2 \\ 3! &= 3(2)(1) = 6 \\ 4! &= 4(3)(2)(1) = 24 \\ 5! &= 5(4)(3)(2)(1) = 120 \end{aligned}$$

[점화식]

- 위 수식이 팩토리얼(factorial) 함수이다. 정확히는 'n!'로 표시하면 n! = n × (n - 1)!로 선언할 수 있다. 자신의 숫자에서 1씩 빼면서 곱하는 형식이다. 보통은 점화식이라고 한다.

Section06 함수의 심화 내용

■ **재귀 함수**

▪ 자신이 자신을 호출

```
def selfCall() :
    print('하', end = '')
    selfCall()
selfCall()
```

출력 결과

```
하하하하하하하하…
```


Section06 함수의 심화 내용

▪ 입력한 숫자를 1까지 세는 함수를 재귀 함수

```
def count(num) :
    if num >= 1 :
        print(num, end = ' ')
        count(num - 1)
    else :
        return
count(10)
count(20)
```

출력 결과

```
10 9 8 7 6 5 4 3 2 1
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Section06 함수의 심화 내용

- 팩토리얼(Factorial)값을 구하는 함수

```
def factorial(num) :
    if num <= 1 :
        return num
    else :
        return num * factorial(num - 1)
print(factorial(4))
print(factorial(10))
```

출력 결과

```
24
3628800
```

65/69

Section06 함수의 심화 내용

- **제너레이터와 yield**
  - yield 문 : 함수를 종결하지 않으면서 값을 계속 반환

```
def genFunc() :
    yield 1
    yield 2
    yield 3

print(list(genFunc()))
```

출력 결과

```
[1, 2, 3]
```

66/69

Section06 함수의 심화 내용

- 제너레이터(Generator : 생성자) 함수 : yield가 포함된 함수
  - yield 문으로 값을 반환한 후 계속 진행

```
1 def genFunc(num) :
2     for i in range(0, num) :
3         yield i
4         print('제너레이터 진행 중')
5 for data in genFunc(5) :
6     print(data)
```

1행의 genFunc() 함수는 매개변수 num에 5를 대입
2행에서 range() 함수로 0~4를 반복
3행에서 0 반환

5행에서 for 문이 실행될 때 genFunc() 함수 호출

6행에서 0 출력

출력 결과

제너레이터는 계속 진행되어 4행의 메시지 출력
아직 2행의 for 문이 끝나지 않았으므로 계속 반복

```
0
제너레이터 진행 중
1
제너레이터 진행 중
2
제너레이터 진행 중
… 생략 …
```

67/69