7장 코드를 함수로 모아봅시다.

드디어 함수까지 왔습니다...

함수는 '활용을 어떻게 하느냐' 에 따라 코드 작성도 내용도 차이가 많이 나는 부분입니다.

결국은 어렵다는 얘기입니다.

어려우니 배우는 것이고 그러니 모르면 이해가 안 되면 질문을 하셔야 합니다!

중간고사에도 반드시 들어가는 부분이니 이해해야 합니다~

우리 성신인, 수정이들 모두 오늘도 파이팅입니다! ^__________^

함수의 개념

- 수학 속 함수
  - 수학에서 함수는 특정 수식에 x 값을 대입하면 y 값이 정해지는 것으로, y = f(x)라고 표현함

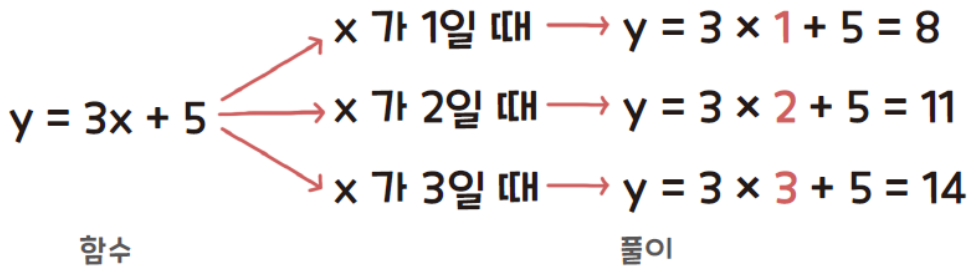


그림 10-1 수학에서의 함수

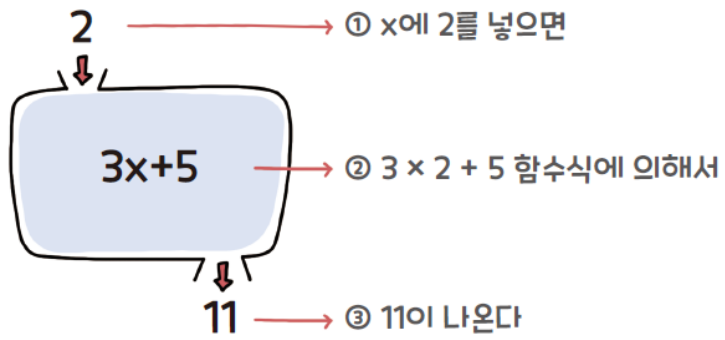


그림 10-2 함수의 구조

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수의 종류 : 내장 함수 vs 사용자 정의 함수

- 함수의 종류
  - 내장 함수 : 파이썬에서 기본으로 제공하는 함수로 파이썬을 설치하면 바로 사용할 수 있음
  - 사용자 함수 : 사용자가 필요할 때 직접 만들어 사용함

(a) 내장 함수

(b) 사용자 함수

그림 10-5 파이썬 함수의 종류

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수의 종류 : 내장 함수 vs 사용자 정의 함수

- 대표적인 내장 함수
  - print(), len() 등

#화면출력

input() → 입력함수

int() → 정수형 함수

코드 10-1

```
01  userName = 'Hong gil dong'
02
03  print('이름 : ', userName)
04  print('이름의 길이 : ', len(userName))
```

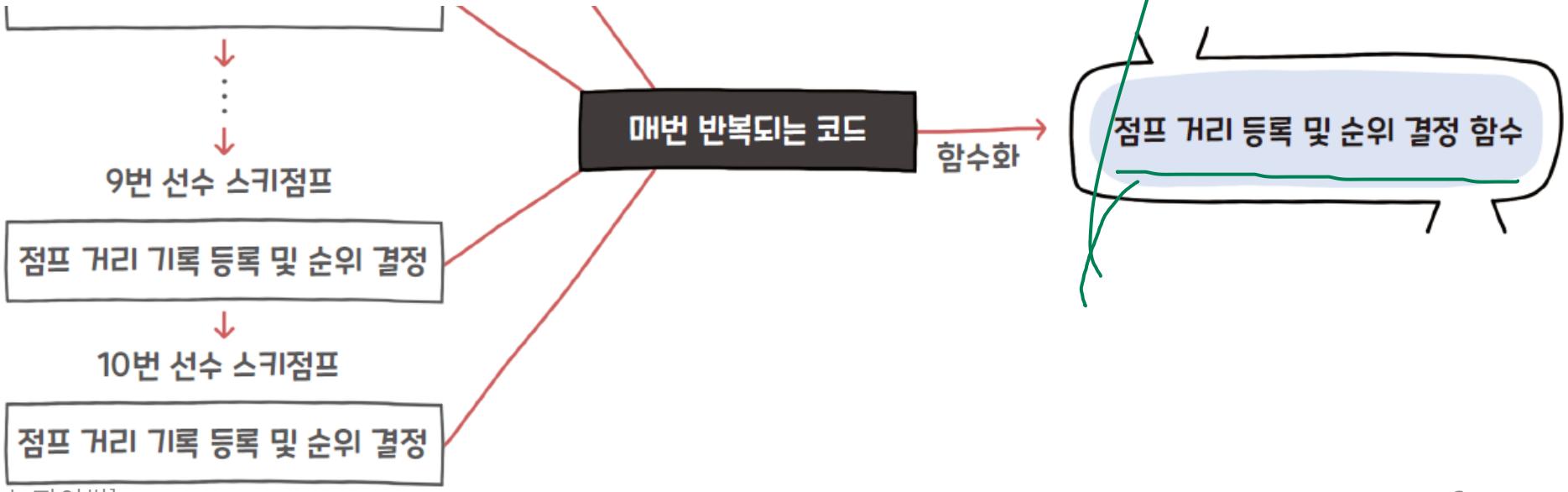
```
이름 :  Hong gil dong
이름의 길이 :  13
```

- 사용자 함수
  - 사용자가 직접 만들어 사용함
  - 개발자가 특정 함수가 필요하다고 판단할 때 직접 함수를 만들어 프로그램에 삽입함

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수는 왜 사용할까?

- 코드 재사용
  - 10명의 스키점프 선수가 점프할 때마다 점프 거리를 입력 받아 1등부터 10등까지 순위를 결정하는 프로그램을 만든다고 가정함
  - 이 프로그램에서 점프 거리를 등록하고 순위를 계산하는 코드가 10번 중복됨
  - 이렇게 매번 중복되는 코드를 함수로 만들어 사용하면 프로그램의 코드 양이 줄어들고 간결해짐



[출처 : 난생처음 컴퓨팅사고 with 파이썬]

그림 10-7 매번 반복되는 코드를 함수로 정리

함수는 왜 사용할까?

- 모듈화
  - 모듈은 특정 기능의 작은 프로그램을 뜻함
  - 특정 기능이 함수로 모듈화되면 다른 프로그램에 쉽게 이식하여 사용할 수 있으며 그만큼 프로그램을 만드는 시간도 단축됨

ex) randint()

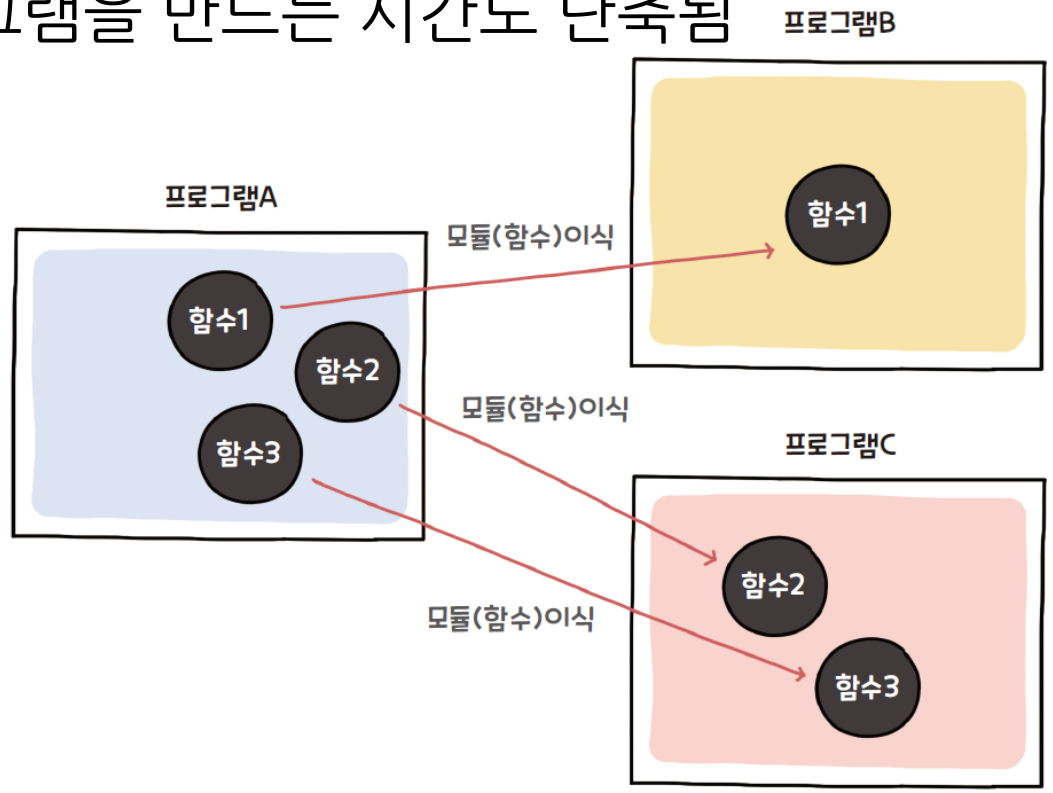


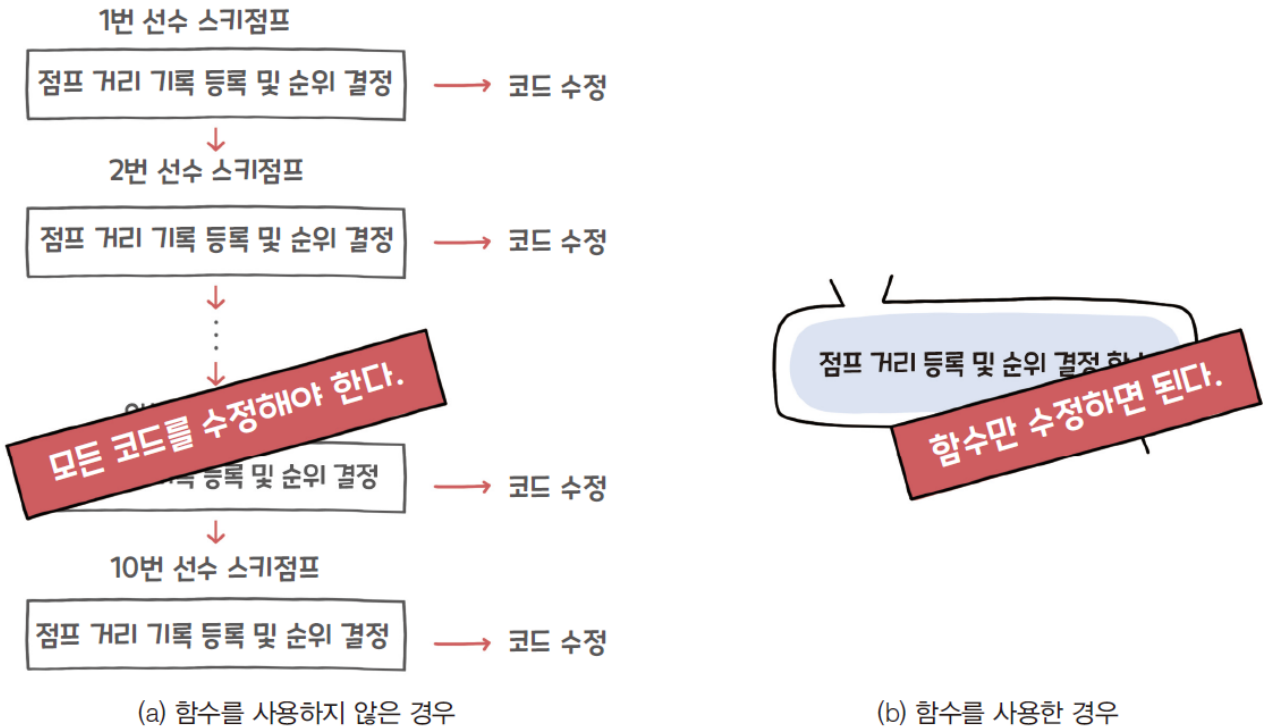
그림 10-8 함수를 여러 프로그램에 이식하여 사용

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수는 왜 사용할까?

재사용성↑
코드수정↑

- 코드 수정 용이
  - 코드에 수정 사항이 발생했을 때 함수를 사용하면 함수만 수정하면 됨
  - 만약 함수를 사용하지 않는다면 모든 코드를 수정해야 하는 불편함이 있고, 자칫 일부 코드가 수정되지 않아 프로그램에 심각한 오류가 발생할 수 있음



(a) 함수를 사용하지 않은 경우 (b) 함수를 사용한 경우

그림 10-9 코드 수정의 용이성

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수의 구성 요소

- 함수 정의(function definition)
  - 함수를 정의할 때는 def 키워드, 함수명, 콜론(:), 실행문으로 나누어 작성함
  - def
    - def는 definition의 약자로 '무언가를 정의한다.'는 뜻임
    - 여기서는 함수를 정의한다는 의미로 사용함

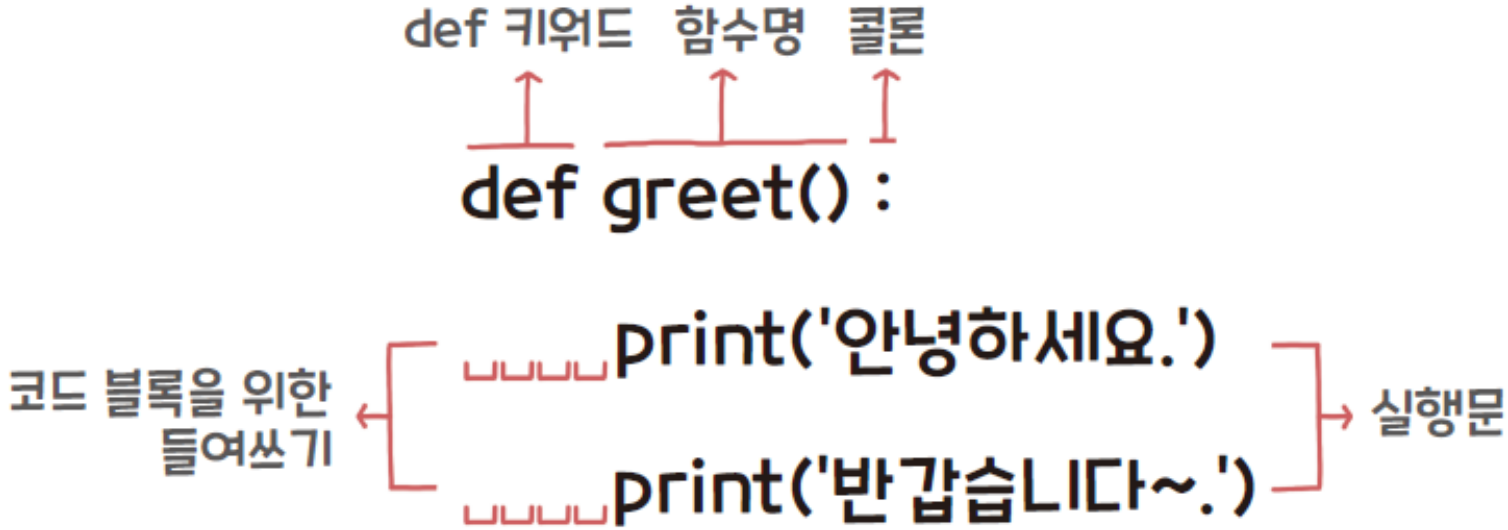


그림 10-10 함수 정의 구문

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수의 구성 요소

- 함수명
  - def 키워드 다음에는 함수명을 명시함
  - 함수명을 지을 때의 유의 사항

① 내장 함수명과 동일하면 안 됨
  - print(), len() 등과 같이 기존의 내장 함수와 같은 이름은 사용이 불가능함
  - 이 경우 myPrint(), myLen()처럼 내장 함수 이름과 중복되지 않도록 함

② 첫 글자는 주로 소문자로 시작함
  - 예를 들어 MyCalculator()는 첫 글자가 대문자이므로 권장하지 않음
  - 이 경우 myCalculator()처럼 첫 글자를 소문자로 수정하여 사용함

③ 첫 글자로 숫자를 사용할 수 없음
  - 첫 글자 외에는 숫자를 사용해도 되지만 2myCalculator()처럼 숫자로 시작하는 함수명은 사용이 불가능함

④ 특수문자는 사용할 수 없지만 언더바(_)는 사용 가능함
  - 특수 문자 하이픈(-)이 포함된 my-Calculator()는 사용 불가능함

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수의 구성 요소

- 콜론(:)과 실행문
  - 콜론(:)은 실행문(코드 블록)의 시작을 나타내는 것으로 앞에서 살펴본 if문, for문과 사용 방법이 같음
  - 실행문 역시 들여쓰기로 구분함

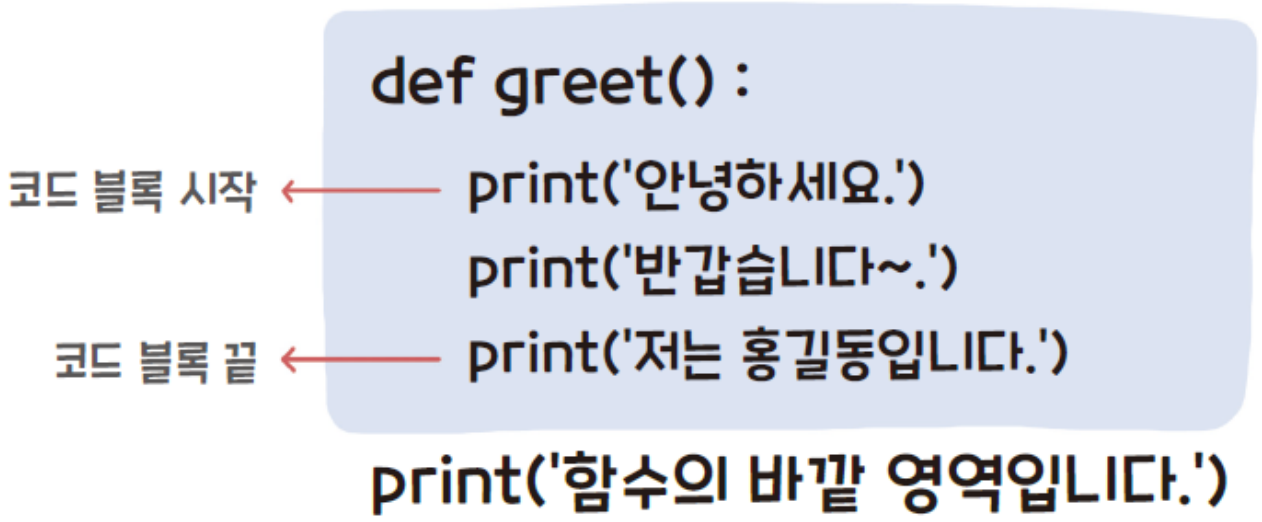


그림 10-11 실행문은 들여쓰기로 구분

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수 호출

- 함수 호출
  - 함수를 사용하는 것을 의미함
  - 함수 호출은 함수 실행이 필요한 곳에서 함수 이름을 적으면 됨

  - 인사 문구를 출력하는 greet() 함수를 정의하고 호출하기

코드 10-2

```
01  def greet():
02      print('Hello.')
03      print('Nice to meet you.')
04
05  greet()
```

함수 정의

함수 호출

```
Hello.
Nice to meet you.
```

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수 작성하고 호출하기

```
def print_address():
        print("서울특별시 종로구 1번지")
        print("파이썬 빌딩 7층")
        print("홍길동")

print_address()
```

함수 정의

함수 호출

```
========== RESTART: C:/User
서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
>>>
```

[출처 : 두근두근 파이썬, 생능]

함수의 장점

- 한 번만 함수를 정의하면 언제든지 필요할 때면 함수를 불러서 일을 시킬 수 있다.

```
print_address()
print_address()
print_address()
```

```
서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
```

[출처 : 두근두근 파이썬, 생능]

함수 내부에서 다른 함수 호출

- 함수 내부에서 또 다른 함수를 호출하기
  - fun3()을 호출하면 fun3() 내부에서는 fun1()과 fun2()를 호출함
  - 따라서 fun3()을 호출하면 fun1()과 fun2()가 모두 호출됨

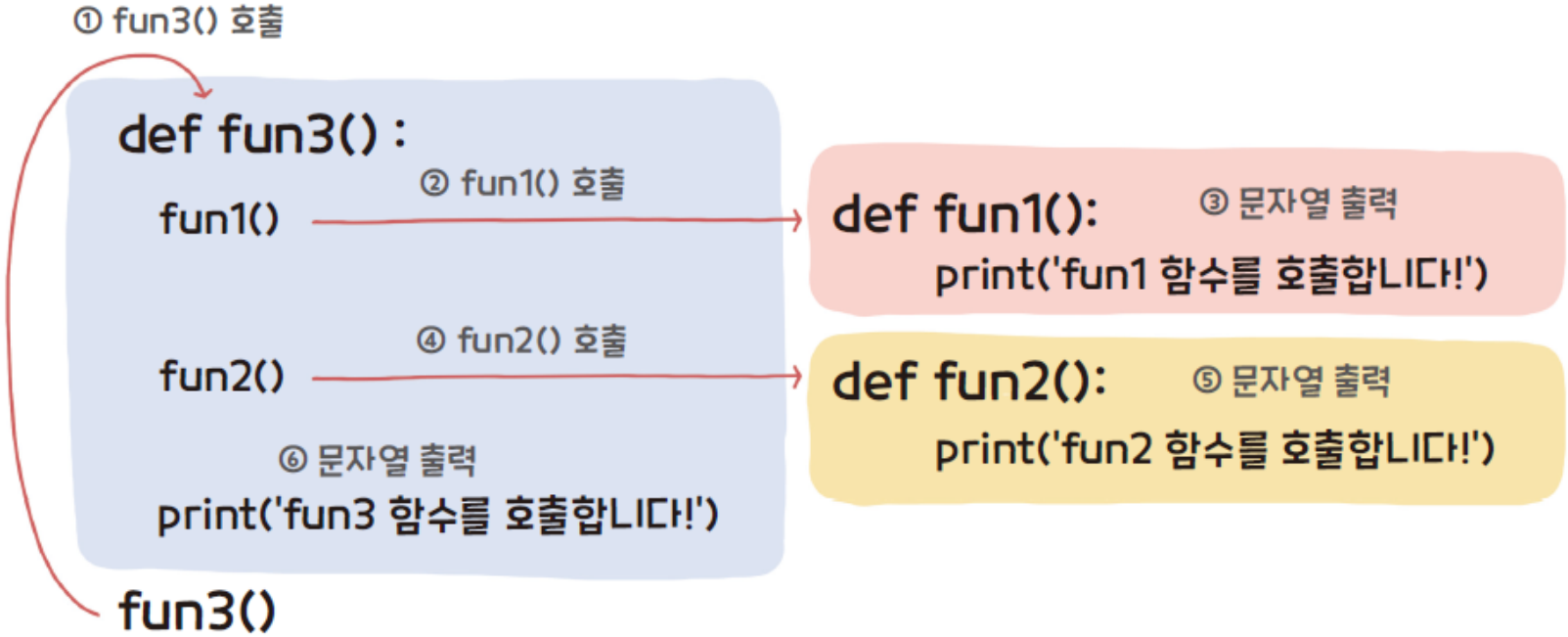


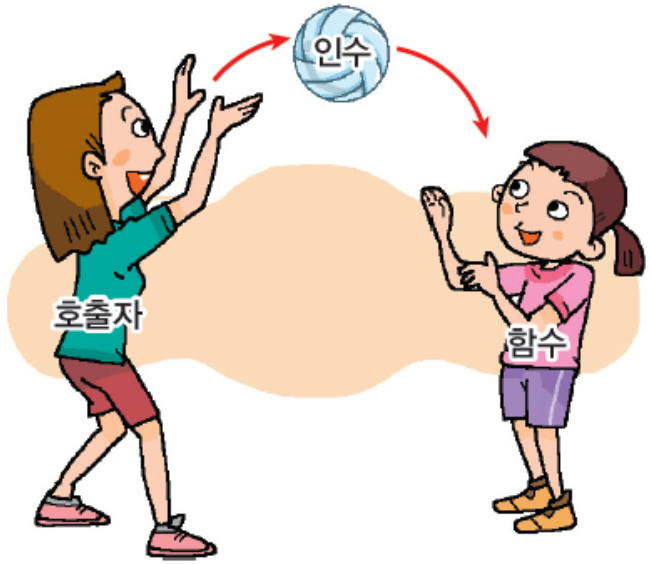
그림 10-12 함수 호출 및 실행 순서

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수에 입력 전달하기

- 우리는 함수에 값(정보)을 전달할 수 있다. 이 값을 인수(argument)라고 한다.

y=f(x)



[출처 : 두근두근 파이썬, 생능]

인수 전달

name을 통하여
함수로 값이 전달됨

```
def print_address(name):
        print("서울특별시 종로구 1번지")
        print("파이썬 빌딩 7층")
        print(name)

print_address("성춘향")
```

```
========== RESTART: C:/Users
서울특별시 종로구 1번지
파이썬 빌딩 7층
성춘향
>>>
```

코드 11-4

```
01  def greet(name):
02      print(name, '씨, 안녕하세요.')
03
04  greet('홍길동')
05  greet('박찬호')
06  greet('박지성')
```

❷ 매개변수에 저장
❸ 출력
❶ 인수 전달

```
홍길동 씨, 안녕하세요.
박찬호 씨, 안녕하세요.
박지성 씨, 안녕하세요.
```

[출처 : 두근두근 파이썬, 생능]

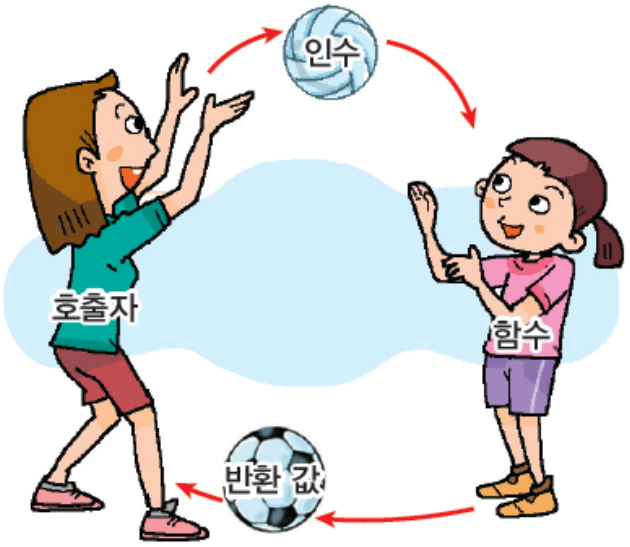
함수 내부에서 다른 함수 호출 + 인수 전달

```
import time

def inputNumber(cnt):
    for i in range(1, cnt+1):
        print(i, "단계 : ","~"*i)
        print_game()
        time.sleep(i) <- 실행 잠시 멈춤

def print_game():
    print("안녕! 클레오파트라!", end=" ")
    print("세상에서 제일 가는 포테이토칩!")
    print()

print("게임을 시작하지!")
count = int(input("몇 명인가요?"))
inputNumber(count)
print("게임 종료!")
```

```
====== RESTART: C:₩Users₩Administrator₩De
게임을 시작하지!
몇 명인가요?3
1 단계 :  ~
안녕! 클레오파트라! 세상에서 제일 가는 포테이토칩!

2 단계 :  ~~
안녕! 클레오파트라! 세상에서 제일 가는 포테이토칩!

3 단계 :  ~~~
안녕! 클레오파트라! 세상에서 제일 가는 포테이토칩!

게임 종료!
>>>
```

값 반환하기

- 함수는 값을 반환할 수 있음
- return 키워드
  - 함수는 실행이 끝난 후에 나온 결과물(데이터)을 호출부로 반환할 수 있음



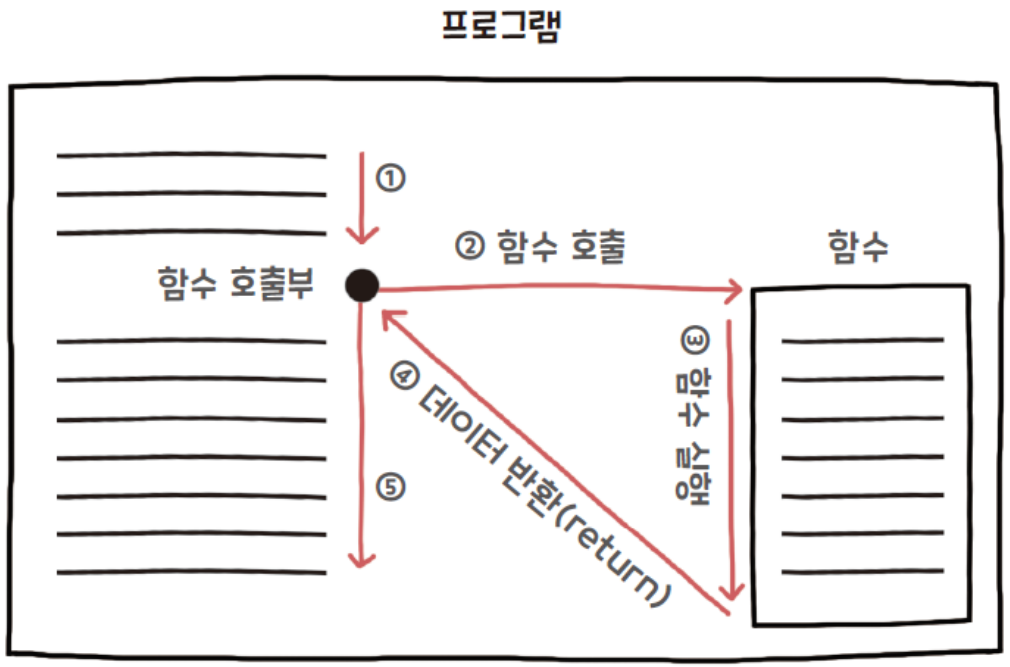


그림 11-3 함수가 포함된 프로그램의 실행 순서

[출처 : 두근두근 파이썬, 난생처음 컴퓨팅사고 with 파이썬]

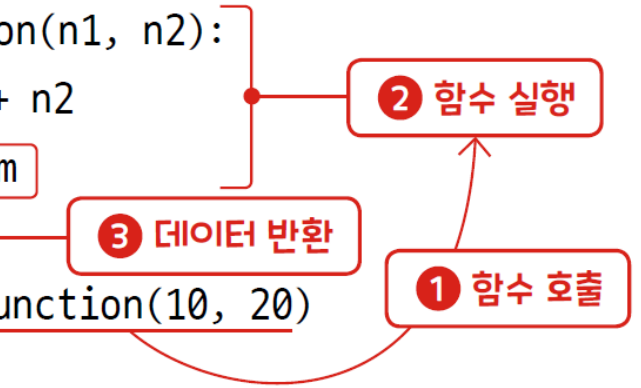
값 반환

```
def calculate_area(radius) :
    area = 0
    area = 3.14 * radius **2
    return area

c_area = calculate_area(5.0) # 실행 결과 값이 # c_area에 대입됨
                             = area 값 같음
print('반지름 5.0 원의 면적은 ', c_area)
```

```
========== RESTART: C:/Use
반지름 5.0 원의 면적은  78.5
>>>
```

코드 11-9

```
01  def addFunction(n1, n2):
02      sum = n1 + n2
03      return sum
04
05  result = addFunction(10, 20)
06  print(result)
```




```
30
```

[출처 : 두근두근 파이썬, 난생처음 컴퓨팅사고 with 파이썬]

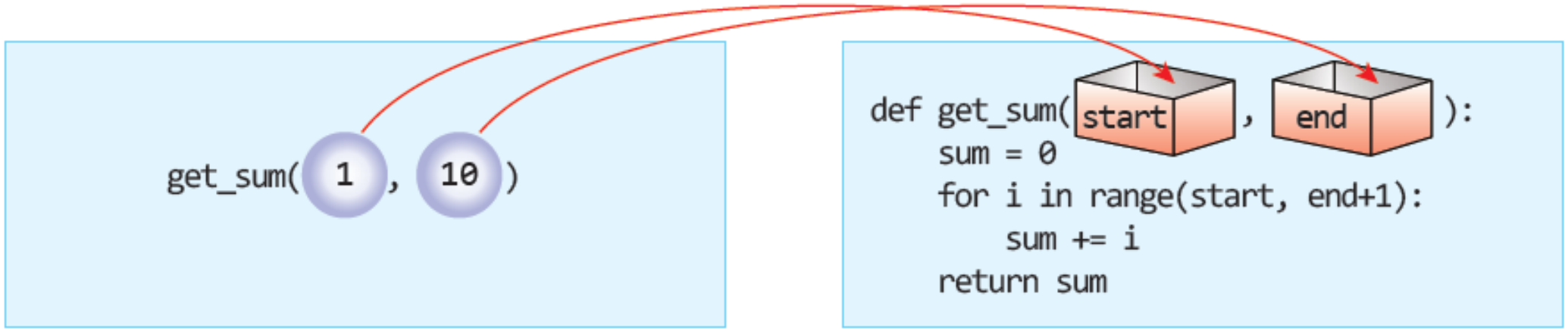
함수에 여러 개의 입력 전달하기

```
def get_sum(start, end):
    sum = 0
    for i in range(start, end+1):
        sum += i

    return sum

print(get_sum(1, 10))
```

코드 11-9

```
01  def addFunction(n1, n2):
02      sum = n1 + n2
03      return sum
04
05  result = addFunction(10, 20)
06  print(result)
```

② 함수 실행
③ 데이터 반환
① 함수 호출



[출처 : 두근두근 파이썬, 생능]

함수에 여러 개의 입력 전달하기

- 함수 호출부에서 전달할 인수가 두 개 이상인 경우
  - 매개변수는 호출부에서 전달하는 인수의 개수와 순서에 맞춰서 선언

코드 11-5 ch11_05.py

```
01  def forecastWeather(temp, humi, rain):      # 인수 3개를 매개변수 3개에 저장
02      print('날씨 예보입니다.')
03      print('최고 온도 : ', temp, '도')
04      print('평균 습도 : ', humi, '%')
05      print('비올 확율 : ', rain, '%')
06
07  temperature = 32
08  humidity = 67
09  rainPercent = 50
10
11  forecastWeather(temperature, humidity, rainPercent)  # 함수 호출 시 인수 3개 전달
```

함수 정의

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

함수에 여러 개의 입력 전달하기

하나 더 알기 **인수의 개수와 매개변수의 개수가 일치하지 않는 경우**

인수의 개수와 매개변수의 개수가 일치하지 않으면 에러가 발생합니다. 따라서 전달하는 인수의 개수와 매개변수의 개수는 반드시 일치해야 합니다.

■ 전달하는 인수의 개수가 매개변수 개수보다 적은 경우 → 에러 발생

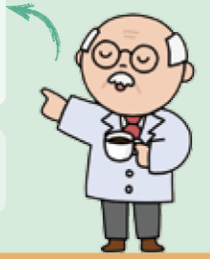
```
def fun1(n1, n2, n3):        ← 매개변수의 개수 3개
    print(n1, n2, n3)

fun1(10, 20)                 ← 인수의 개수 2개
```

```
TypeError: fun1() missing 1 required positional argument: 'n3'
```

■ 전달하는 인수의 개수가 매개변수 개수보다 많은 경우 → 에러 발생

```
def fun1(n1, n2, n3):        ← 매개변수의 개수 3개
    print(n1, n2, n3)

fun1(10, 20, 30, 40)         ← 인수의 개수 4개
```

```
TypeError: fun1() takes 3 positional arguments but 4 were given
```



[출처 : 난생처음 컴퓨팅사고 with 파이썬]

인수의 개수를 모르는 경우

- 함수를 호출할 때 전달하는 인수의 개수가 수시로 변경되는 경우
  - '*' 기호를 이용 : 매개변수의 개수를 변경할 필요 없이 해결할 수 있음

코드 11-6 ch11_06.py

```
01  def printAverageScore(*scores):
02      print(type(scores))
03
04      totalScore = 0
05      cnt = len(scores);
06
07      for score in scores:
08          totalScore += score
09
10      print('총점: ', totalScore, '점')
11      print('평점: ', totalScore / cnt, '점')
12      print('----------------------------')
13
14  printAverageScore(80, 90, 70)
15  printAverageScore(90, 85, 90, 100)
16  printAverageScore(95, 80, 100, 95, 85)
```

여러개 값을 받아올 수 있음

함수 정의

```
<class 'tuple'>
총점:  240 점
평점:  80.0 점
----------------------------
<class 'tuple'>
총점:  365 점
평점:  91.25 점
----------------------------
<class 'tuple'>
총점:  455 점
평점:  91.0 점
----------------------------
```

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

재귀 함수

- 재귀(Recursive) 함수
  - 함수 안에서 자신을 다시 호출하는 함수
  - 재귀 함수는 자신 안에서 자기 자신을 계속 호출하기 때문에 함수 호출을 종료할 수 있는 코드가 꼭 필요함

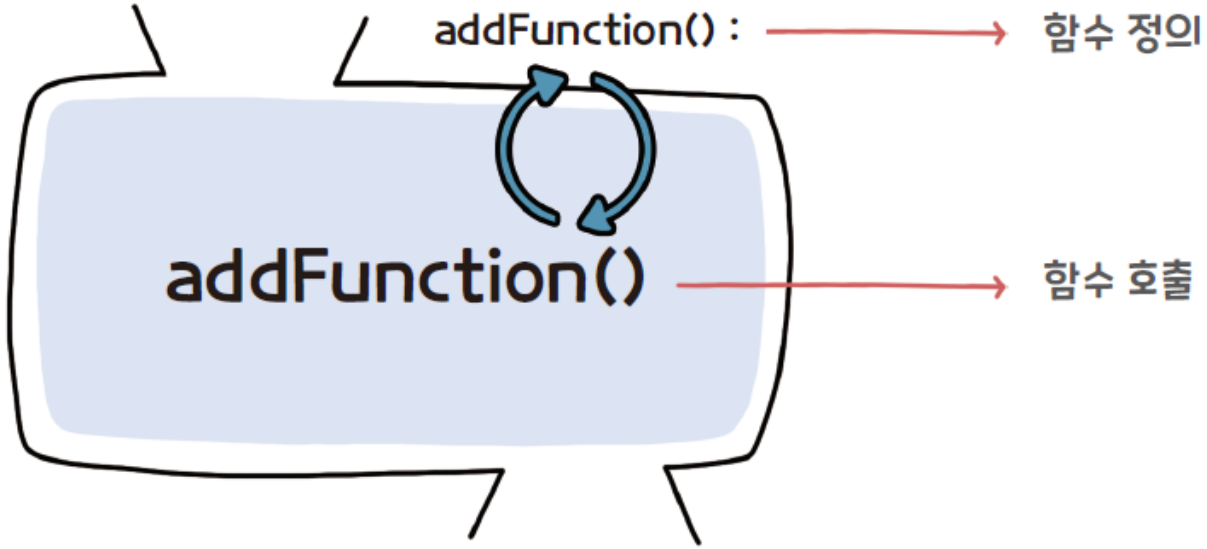


그림 11-5 재귀 함수의 개념

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

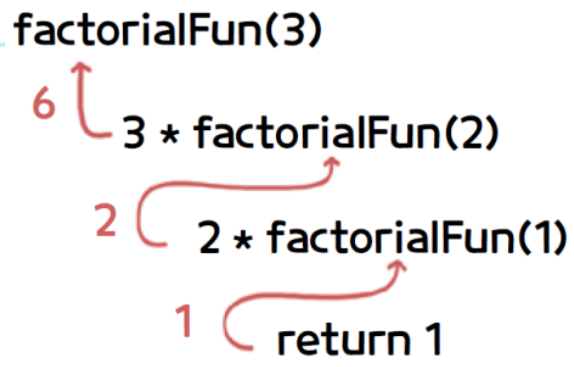
재귀 함수

문제 해결 11-4 재귀 함수로 팩토리얼 구현하기 ch11_sol_04.py

사용자가 입력한 정수를 이용하여 팩토리얼 계산을 실행하는 프로그램을 재귀 함수를 이용하여 만들어봅시다.

```
01  # 재귀 함수 정의
02  def fatorialFun(num):
03      if num == 1:
04          return 1
05      else:
06          return num * fatorialFun(num - 1)
07
08  inputData = int(input('0보다 큰 숫자를 입력하세요. '))
09  result = fatorialFun(inputData)
10  print( inputData, '팩토리얼은 ', result,'입니다.')
```

재귀 함수 호출



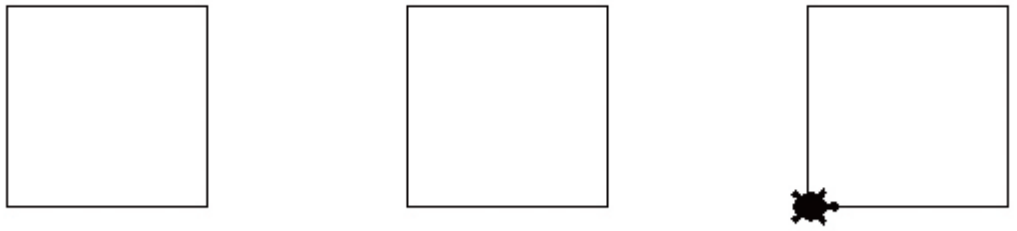
[출처 : 난생처음 컴퓨팅사고 with 파이썬]

Lab: 사각형을 그리는 함수 작성하기

- 정사각형을 그리는 함수는 다음과 같다.

```
def square(length):          # length는 한 변의 길이
    for i in range(4):
        t.forward(length)
        t.left(90)
```

- 위의 함수를 호출하여 3개의 정사각형을 그려 보자.



[출처 : 두근두근 파이썬, 생능]

Solution

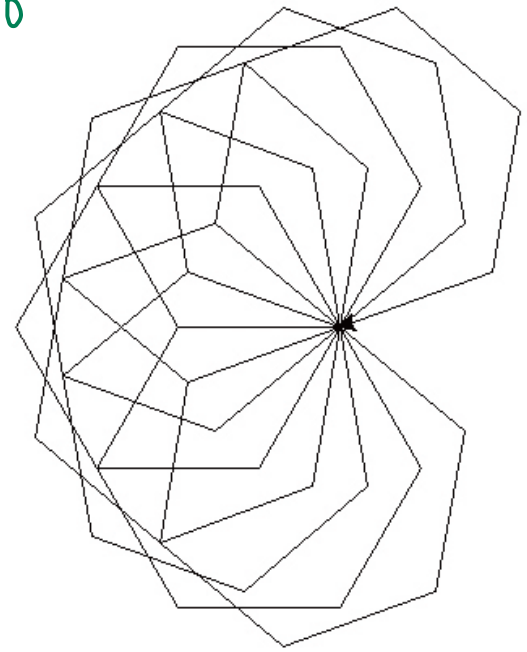
- 사용자에게 한 변의 길이를 입력 받아 사각형을 그리는 함수를 호출하여 사각형을 3개 그려보자.

```
import turtle
t = turtle.Turtle()
t.shape("turtle")

def square(length):                    # length는 한 변의 길이
    for i in range(4):
        t.forward(length)
        t.left(90)

def drawSquare(varX, varY, size): # varX, varY는 x, y 좌표, size는 한 변의 길이
    t.up()
    t.goto(varX, varY)
    t.down()
    square(size)
```

[출처 : 두근두근 파이썬, 생능]

Lab: n-각형을 그리는 함수 작성하기

- 도형의 길이와 그릴 도형을 숫자로 입력 받아 그림을 그려보자.

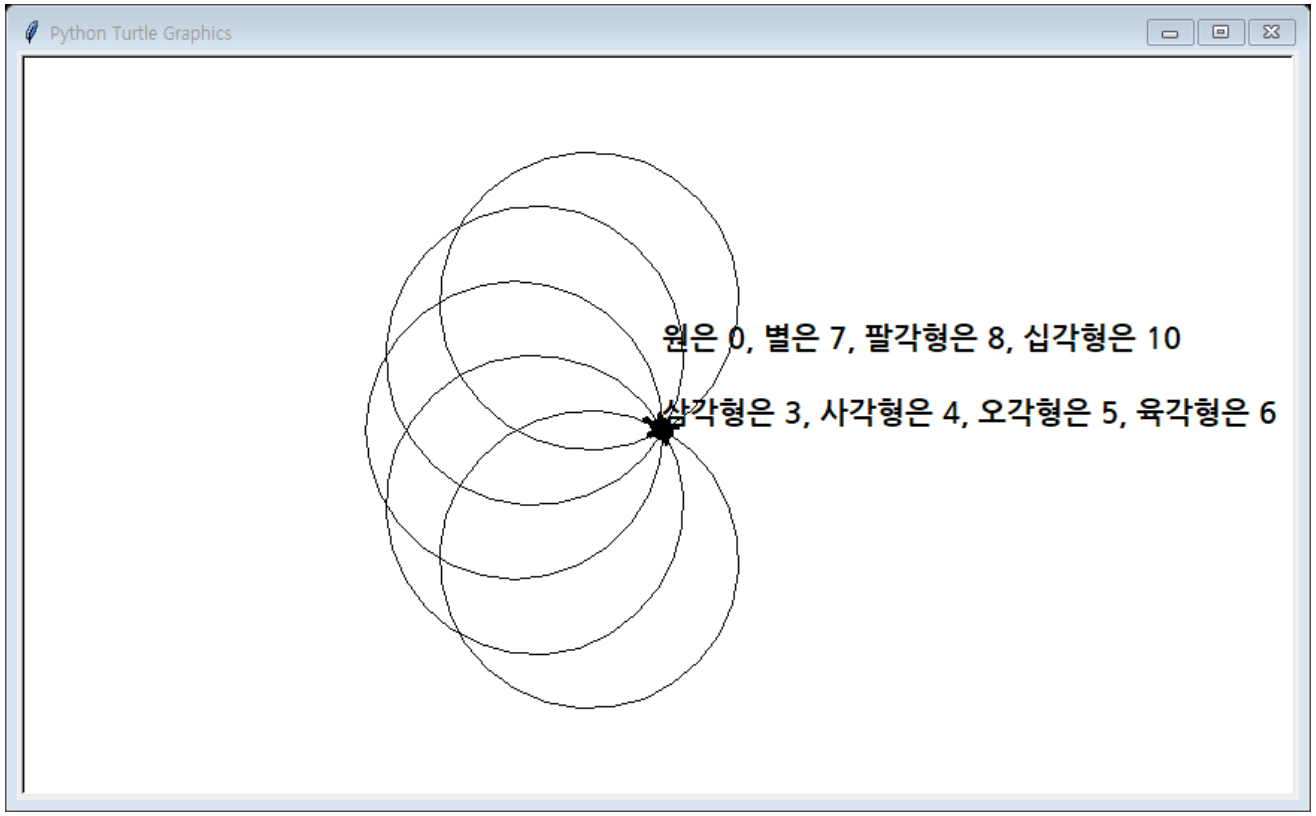


```
import turtle
t = turtle.Turtle()
t.shape("turtle")

def n_polygon(n, length):
    for i in range(n):
        t.forward(length)
        t.left(360//n)

for i in range(10):
    t.left(20)
    n_polygon(6, 100)
```

[출처 : 두근두근 파이썬, 생능]

Lab: n-각형을 그리는 함수 작성하기

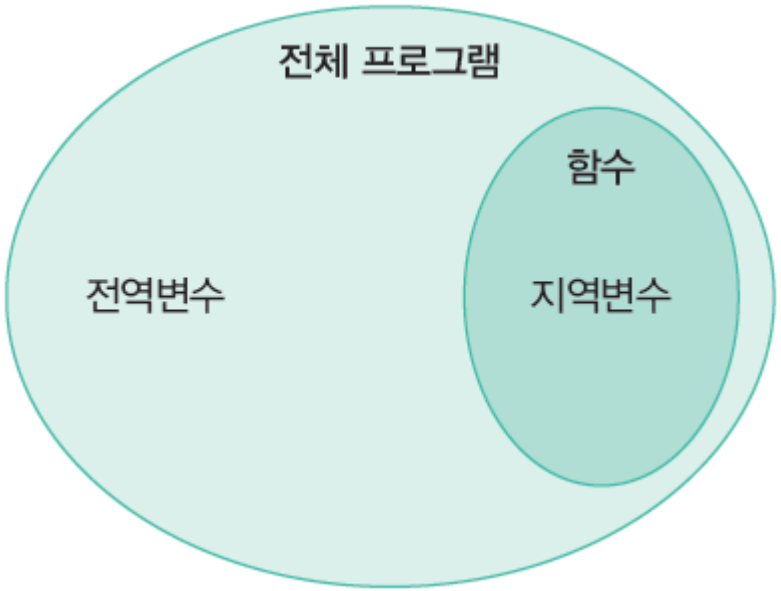
- 29 page의 코드를 참고하여 다음과 같은 결과가 나오도록 코드를 작성하세요.




```
turtle.write("삼각형은 3, 사각형은 4, 오각형은 5, 육각형은 6",font=("나눔고딕",15,"bold"))
```

변수의 종류

- 지역 변수(local variable): 함수 안에서 선언되는 변수
- 전역 변수(global variable): 함수 외부에서 선언되는 변수



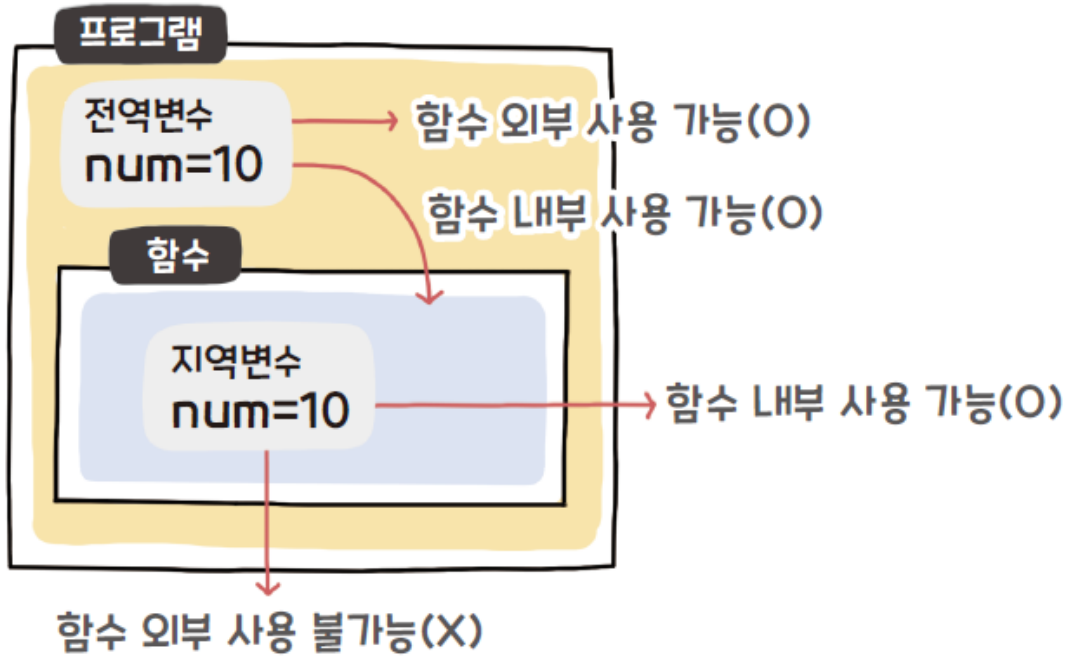


그림 11-1 지역변수와 전역변수의 사용 가능 범위

[출처 : 두근두근 파이썬, 생능]

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

변수의 종류

- 지역변수와 전역변수의 차이
  - fun1() 함수 내부에서 num을 다시 선언하는 경우
  - 4행에서 선언된 num : 지역변수
  - 1행의 num과 4행의 num은 이름만 같을 뿐 전혀 다른 변수임

함수가 실행되는 동안에만 살아있음.

코드 11-2 ch11_02.py

```
01  num = 10                        # 전역변수 num 선언
02
03  def fun1():                     # 함수 정의
04      num = 20                    # 지역변수 num 선언
05      print('num : ', num)        # 지역변수 num 사용(함수 안에서 num을 먼저 찾는다.)
06
07  print('num : ', num)            # 전역변수 num 사용
08  fun1()                          # 함수 호출
```

```
num :  10    전역 변수 num 출력
num :  20    지역 변수 num 출력
```

변수의 종류

- global 키워드
  - '전역을 가리킨다'는 의미로 global 키워드를 사용하여 전역변수에 접근함
    → **'global num'이라고 하면 전역변수 num을 가리킴**

함수 안과 밖에서 모두 사용 가능

코드 11-3 ch11_03.py

```
01  num = 10                          # 전역변수 num 선언
02
03  def fun1():                       # 함수 정의
04      global num                    # 전역변수 num 설정
05      num = 20                      # 전역변수 num 변경
06      print('num : ', num)          # 전역변수 num 사용
07
08  print('num : ', num)              # 전역변수 num 사용
09  fun1()                            # 함수 호출
10  print('num : ', num)              # 전역변수 num 사용
```

```
num :  10
num :  20
num :  20
```

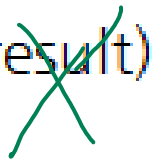
전역 변수 num 출력

[출처 : 난생처음 컴퓨팅사고 with 파이썬]

지역 변수의 범위

- 지역 변수는 함수 안에서만 사용이 가능하다.
- 아래의 코드에서 지역 변수를 찾아보자.

```
def calculate_area(radius) :
    result = 3.14 * radius **2
    return result

r = float(input("원의 반지름은 " ))
area = calculate_area(r)

print('반지름 ',r, '인 원의 면적은 ', result)
```



[출처 : 두근두근 파이썬, 생능]

지역 변수의 범위

```
def calculate_area(radius) :
    result = 3.14 * radius **2
    return result

r = float(input("원의 반지름은 " ))
area = calculate_area(r)

print('반지름 ',r, '인 원의 면적은 ', result)
```

지역변수는 함수 안에만 있음.

area로 바꾸면 됨

```
========== RESTART: C:/Users/Administrator/Desktop/성신-파이썬/ex-7week.py ====
원의 반지름은 5.0
Traceback (most recent call last):
  File "C:/Users/Administrator/Desktop/성신-파이썬/ex-7week.py", line 94, in <module>
    print('반지름 ',r, '인 원의 면적은 ', result)
NameError: name 'result' is not defined
>>>
```

[출처 : 두근두근 파이썬, 생능]

전역 변수

- 전역 변수는 어디서나 사용할 수 있다.
- 아래의 코드에서 전역 변수를 찾아보자.

```
def calculate_area(radius) :
    result = 3.14 * radius **2
    return result

r = float(input("원의 반지름은 " ))
area = calculate_area(r)

print('반지름 ',r, '인 원의 면적은 ', result)
```

[출처 : 두근두근 파이썬, 생능]

전역 변수

```
========== RESTART: C:/Users/Administratc
원의 반지름은 5.0
함수 내부 resultA =  100 result =  78.5
반지름  5.0 인 원의 면적은  78.5
resultA =  78.5 result =  0
>>>
```

```
def calculate_area(radius) :
    result = 3.14 * radius **2
    resultA = 100
    print("함수 내부 resultA = ", resultA, "result = ", result)
    return result

result = 0
resultA = 0
r = float(input("원의 반지름은 " ))
resultA = calculate_area(r)

print('반지름 ',r, '인 원의 면적은 ', resultA)
print("resultA = ", resultA, "result = ", result)
```

[출처 : 두근두근 파이썬, 생능]

함수 안에서 전역 변수 값 변경하기

```
def calculate_area(radius) :
    area = 3.14 * radius **2

area = 0
r = float(input("원의 반지름은 " ))
calculate_area(r)

print('반지름 ',r, '인 원의 면적은 ', area)
```

```
========== RESTART:
원의 반지름: 15
706.5
>>>
>>>
```

- global을 사용하여 전역 변수에 값을 저장한다고 알려야 한다.

```
def calculate_area(radius) :
    global area
    area = 3.14 * radius **2

area = 0
r = float(input("원의 반지름은 " ))
calculate_area(r)

print('반지름 ',r, '인 원의 면적은 ', area)
```

[출처 : 두근두근 파이썬, 생능]

디폴트 인수

- 파이썬에서는 함수의 매개변수가 기본값을 가질 수 있다. 이것을 디폴트 인수(default argument)라고 한다.

```
def greet(name, msg="별일없죠?"):
   print("안녕 ", name + ', ' + msg)

greet("영희")
```

```
안녕  영희, 별일없죠?
>>>
```

[출처 : 두근두근 파이썬, 생능]

키워드 인수

- 키워드 인수는 인수의 이름을 명시적으로 지정해서 전달하는 방법이다.

```
def calc(x, y, z):
    return x+y+z
```

```
>>> calc(y=20, x=10, z=30)
60
```

[출처 : 두근두근 파이썬, 생능]

예제 #1

- 계산기 프로그램을 함수를 사용하도록 코드를 수정하세요!

```
print("-"*60)
print("간단한 계산기 프로그램")
print("-"*60)

while True :
    exitText = input("시작은 아무키나, 종료는 x : ")
    if exitText == 'x' or exitText == "X":
        break
    else :
        firstOp = int(input("첫 번째 수를 입력하세요.: "))
        secondOp = int(input("두 번째 수를 입력하세요.: "))
        operator = input("계산할 연산자를 입력하세요.(+,-,*,/) : ")

    finalResult = operatorCal(firstOp, secondOp, operator)   # 사
    calResultPrint(operator, finalResult)
```

```
print("-"*60)
print("간단한 계산기 프로그램")
print("-"*60)

firstOp = float(input("첫 번째 수를 입력하세요.: "))
secondOp = float(input("두 번째 수를 입력하세요.: "))
operator = input("계산할 연산자를 입력하세요.(+,-,*,/) : ")
result = 0

if operator == '+':
    result = int(firstOp + secondOp)
elif operator == '-':
    result = int(firstOp - secondOp)
elif operator == '*':
    result = int(firstOp * secondOp)
elif operator == '/':
    result = firstOp / secondOp
else :
    result = operator

if operator == '/':
    print("연산 결과는 {:.2f}입니다.".format(result))
elif operator == '+' or operator == "-" or operator == "*":
    print("연산 결과는",result,"입니다.")
else:
    print("연산자의 입력이 잘못 되었습니다.")
```