파이썬 기본

5. 함수

python

01. 함수란?

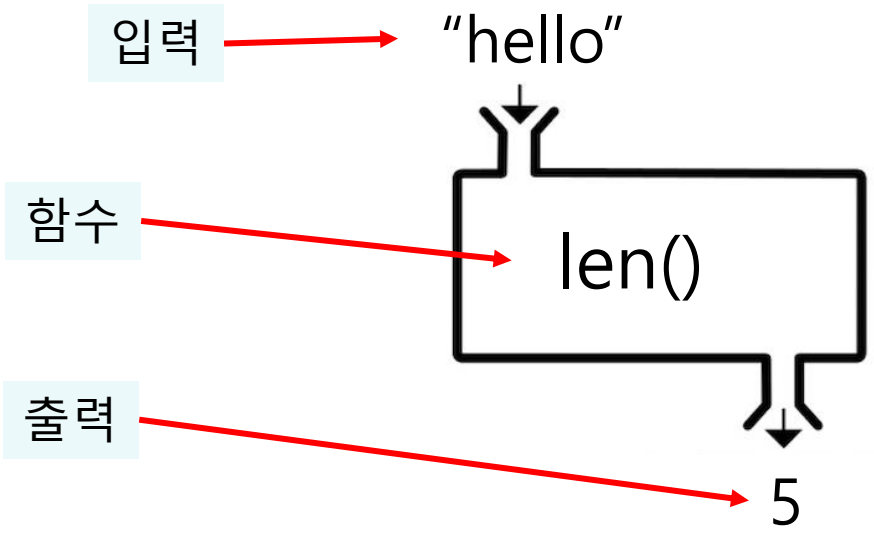
- 프로그램을 쪼개는 3가지 방법
  - ✓ 함수(function) : 반복적으로 사용하는 코드를 묶은 것
  - ✓ 객체(object) : 코드 중에서 독립적인 단위로 분리할 수 있는 조각
  - ✓ 모듈(module) : 프로그램의 일부를 가지고 있는 독립적인 파일

01. 함수란?

- 함수 : 일을 수행하는 코드의 덩어리

  def 키워드로 함수 정의 후 함수 호출하여 사용

  - ✓ 프로그램 안에서 중복된 코드를 제거한다
  - ✓ 코드를 간결하게 유지할 수 있다
  - ✓ 여러 번 호출하여 사용 가능. 다른 프로그램에서도 재사용될 수 있다
  - ✓ 하나의 큰 프로그램을 나누어 작성할 수 있어 구조화된 프로그래밍이 가능 → 가독성 증대, 유지관리 쉬워진다



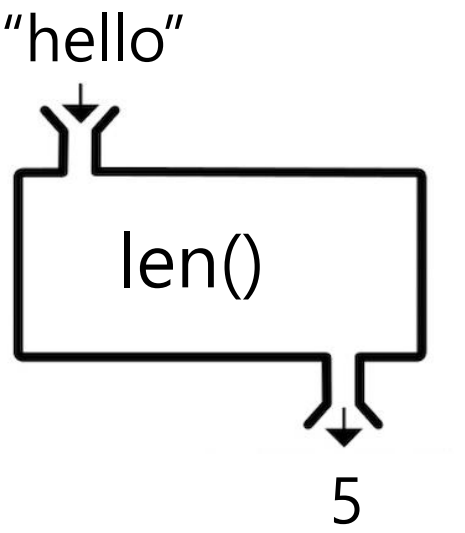
02. 함수 작성하고 호출하기

def 함수이름 (<매개변수1>, <매개변수2>,...) :

    명령어

    명령어

    return <값>



- 인수(argument) : 호출 프로그램에 의해 함수에 전달되는 값(정보)
- 매개변수(parameter) : 외부에서 전달되는 데이터를 함수로 전달하는 변수
- 반환 값 : 함수로 부터 되돌아오는 값

    return 키워드 사용

02. 함수 작성하고 호출하기

- 함수 선언

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

- 함수 호출

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

02. 함수 작성하고 호출하기

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


result = get_area(3)
print("반지름이 3인 원의 면적 = ",result)
result = get_area(20)
print("반지름이 20인 원의 면적 = ",result)
```

참고. 인수와 매개변수

- 인수(argument) : 호출 프로그램에 의해 함수에 전달되는 값(정보)
- 매개변수(parameter) : 이 함수에 전달되는 값을 전달받는 변수

  ➔ 함수가 호출될 때마다 인수는 함수의 매개 변수로 전달된다.
- 반환 값(return value) : 함수가 호출한 곳으로 반환하는 작업의 결과값

  return 키워드 사용

  수식 또는 값을 뒤에 쓸 수 있다

  함수로부터 반환된 값은 변수에 저장하여 사용

  함수가 값을 반환하지 않을 경우 None 반환

```
>>> def func(msg):
        s = msg
        return s
>>> print(func("Hello"))
```

매개변수

인수

02. 함수의 정의와 호출

- 입력 값도 없고 반환 값(리턴 값)도 없는 함수

```
>>> def  say1( ) :
        print('Hi!!')
>>> say1( )
>>> a = say1( )
```

- 입력 값이 없는 함수

```
>>> def say2( ) :
        return 'Hi!!'
>>> say2( )
>>> print(say2( ))
>>> b=say2( )
```

02. 함수의 정의와 호출

- 반환 값(리턴 값)이 없는 함수

```
>>> def  say3(name) :
        print('Hi!!', name)
>>> say3('kim')
>>> say3('lee')
```

- 입력 값과 반환 값(리턴 값) 모두 있는 함수

```
>>> def  say4(num) :
        print('number : ', num)
        return  num * 2
>>> c = say4(10)
>>> c
```

실습문제. 소수 찾기

1. 입력 받은 숫자가 소수인지 확인하는 프로그램을 작성하세요.

   소수 : 1과 자기자신만을 약수로 가지는 수

   프로그램 종료 – 0 입력

   예) 소수인지 확인할 숫자를 입력하세요 : 2

   2는 소수입니다

   소수인지 확인할 숫자를 입력하세요 : 10

   10는 소수가 아닙니다

2. 2부터 100 사이의 수 중에서 소수를 모두 출력하세요

03. 여러 개의 함수가 있는 프로그램

- 파이썬은 인터프리트 언어이기 때문에 함수의 순서가 중요
- 함수를 호출하기 전에 반드시 함수를 정의해야 한다

```
result = get_area(3)    ➔ 에러
print("반지름이 3인 원의 면적=", result)

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

03. 여러 개의 함수가 있는 프로그램

- 함수 내에서는 아직 정의되지 않은 함수를 호출할 수는 있다

```
def main() :
    result1 = get_area(3)
    print("반지름이 3인 원의 면적=", result1)

def get_area(radius):
    area = 3.14*radius**2
    return area

main()
```

04. 매개 변수 전달

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

print_address("홍길동")
print_address("김철수")
```

매개변수

인수

```
>>> def square(n) :
        return n*n
>>> print(square(10))
>>> print(square(100))
```

05. 함수에 여러 개의 입력 전달

- 인수(argument) : 호출 프로그램에 의하여 실제로 함수에 전달되는 값
- 매개변수(parameter) : 이 값(함수로 전달되는 인수 값)을 전달받는 변수

➔ 인수와 매개변수의 개수는 같아야 한다

```
def   get_sum(start, end):
      print(start)
      print(end)

get_sum(1,10)
get_sum(5,100)
```

05. 함수에 여러 개의 입력 전달

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

value = get_sum(1, 10)

x = get_sum(1, 100)
```

6. 디폴트 인수

- 함수의 매개변수가 가지는 기본 값

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

print("철수", "좋은 아침!")
print("영희")
```

➔ 에러

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

print("영희")
print("철수", "좋은 아침!")
```

7. 키워드 인수

- 인수의 이름을 명시적으로 지정해서 전달하는 방법
- 위치 인수(positional argument)와 같이 사용하는 경우는 위치 인수가 먼저 나와야 한다

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


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

08. 가변 인수 함수

- 인수의 개수가 정해지지 않은 가변인수에 사용
  매개변수 앞에 *을 붙여 사용
  함수 안에서는 반복문으로 처리

```
def  함수이름 ( *매개변수) :
    명령어
```

```
>>> def  add_many(*args) :
        result = 0
        for  i  in  args :
            result = result + i
        return  result

>>> add_many(20, 30)
>>> add_many(1,2,3,4,5,6,7,8,9,10)
>>> add_many(1,3,5,7,9)
>>> add_many()
```

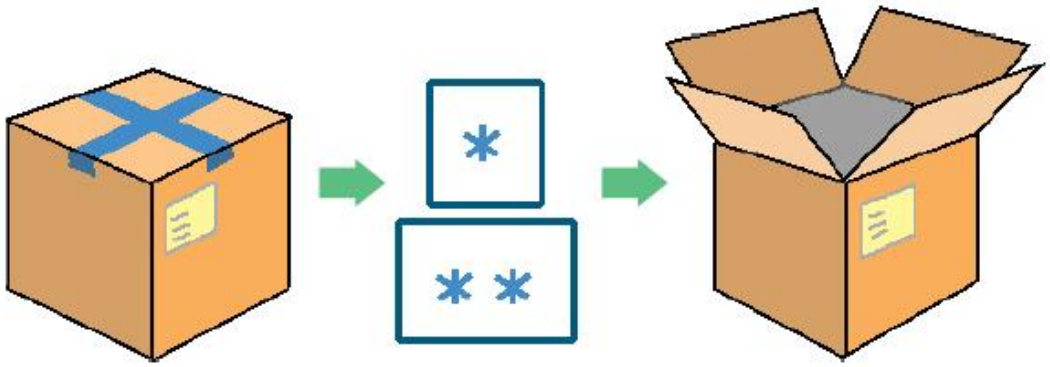
08. 가변 인수 함수

- 고정인수와 가변인수를 함께 사용할 수 있다.

```
>>> def   print_num(a, *args) :
        print(a)
        for   arg   in   args :
            print(arg)

>>> print_num(1)
>>> print_num(1, 10, 20)
>>> print_num(2, 3, 4, 5, 6)
```

참고. * 연산자로 언패킹하기

- 단일 별표 연산자 *는 파이썬이 제공하는 모든 반복 가능한 객체(iterable)을 언패킹할 수 있고 이중 별표 연산자 **는 딕셔너리 객체를 언패킹할 수 있다.




```
>>> alist = [ 1 , 2 , 3 ]
>>> print(*alist)
1 2 3

>>> alist = [ 1 , 2 , 3 ]
>>> print(alist)
[1, 2, 3]
```

09. 값 반환하기

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


print(cal_area(5.0))
c_area = cal_area(5.0)
print(c_area)
area_sum = cal_area(5.0) + cal_area(10.0)
print(area_sum)
```

실습문제. 패스워드 생성기

3. 알파벳 소문자와 숫자를 랜덤하게 조합하여 일회용 패스워드를 생성하는 프로그램을 만드세요. 패스워드의 길이는 6자리로 합니다. genPass()라는 함수를 작성하고 이 함수가 랜덤하게 생성된 패스워드를 반환하게 합니다. 모두 3개의 패스워드를 생성하여 출력하도록 합니다

예) q2ni4s

kw9nxi

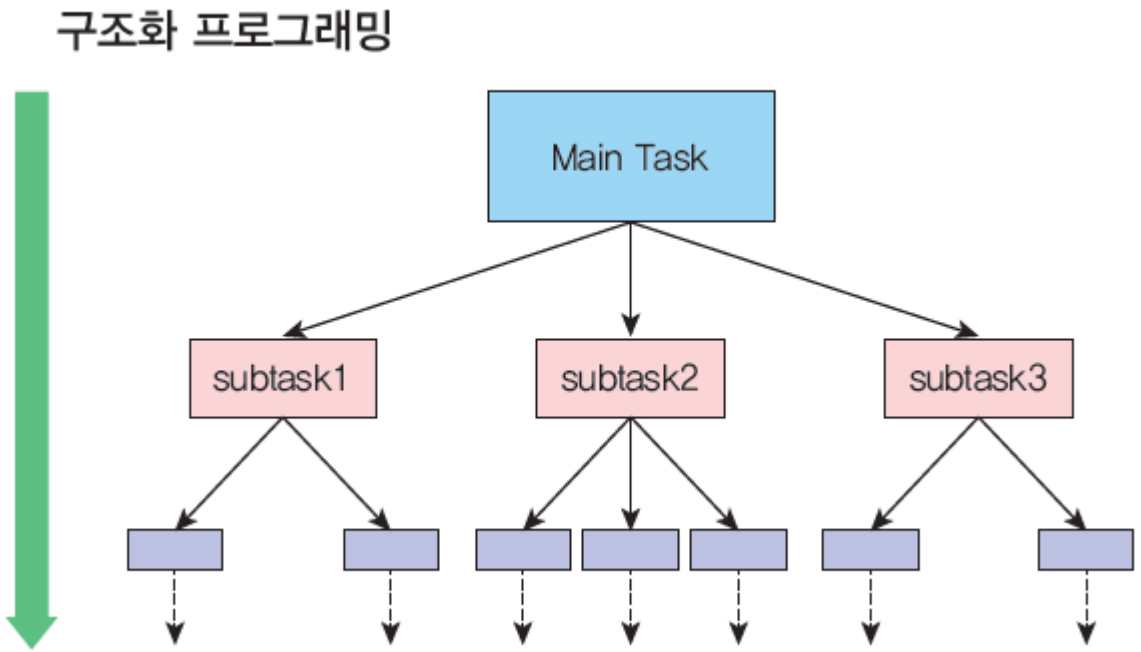
0axyes

4. 사칙 연산을 수행하는 4개의 함수(add(), sub(), mul(), div())를 작성한다. 이들 함수를 이용하여 10 + 20 * 30 을 계산해보자.

10. 함수를 사용하는 이유

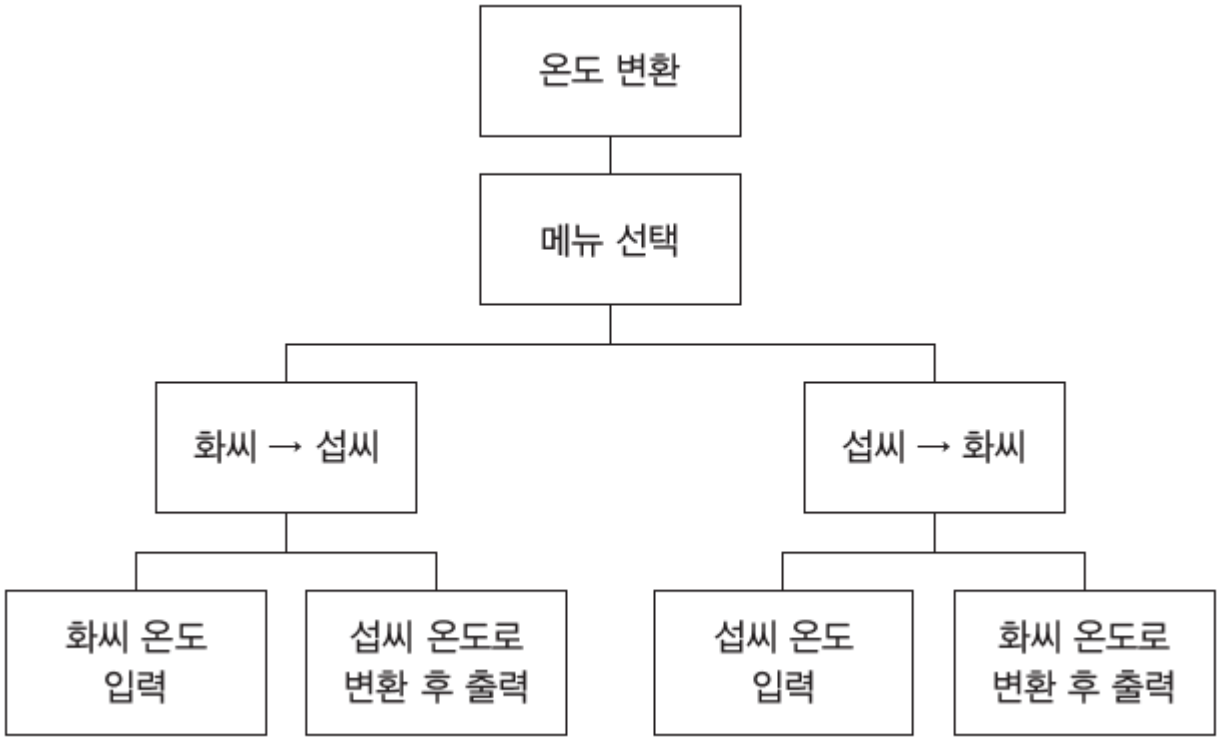
- 소스 코드의 중복성을 없애준다.
- 한번 제작된 함수는 다른 프로그램을 제작할 때도 사용이 가능하다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.

  ➔ 소스의 가독성이 좋아진다.



Lab. 구조화 프로그램 실습

- 온도를 변환하는 프로그램



Lab. 구조화 프로그램 실습

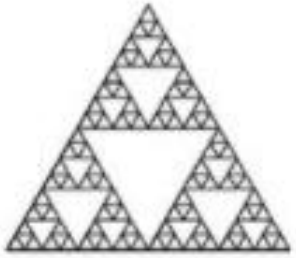
```
# conv_temp.py

def menu() :
    print("1. 섭씨 온도->화씨 온도")
    print("2. 화씨 온도->섭씨 온도")
    print("3. 종료")
    selection = int(input("메뉴를 선택하세요: "))
    return selection

def ctof(c) :
    temp = c*9.0/5.0 + 32
    return temp

def ftoc(f) :
    temp = (f-32.0)*5.0/9.0
    return temp

def input_f() :
    f = float(input("화씨 온도를 입력하시오: "))
    return f
```

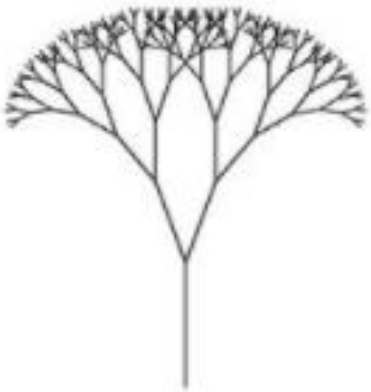
Lab. 구조화 프로그램 실습

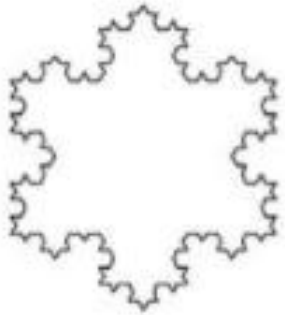
```
def input_c() :
    c = float(input("섭씨 온도를 입력하시오: "))
    return c

def main() :
    while True:
        index = menu()
        if index == 1 :
            t = input_c()
            t2 = ctof(t)
            print("화씨 온도 = ", t2, "₩n")
        elif index == 2 :
            t = input_f()
            t2 = ftoc(t)
            print("섭씨 온도 = ", t2, "₩n")
        else :
            break
main()
```

11. 순환 호출

- **순환(recursion)**이란 어떤 알고리즘이나 함수가 자기 자신을 호출하여 문제를 해결하는 프로그래밍 기법이다.







11. 순환 호출

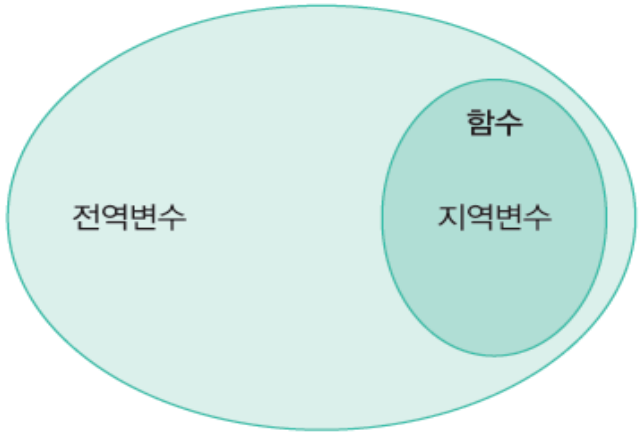
- 팩토리얼(n!) 계산

$$n! = \begin{cases} 1 & n = 0 \\ n*(n-1)! & n >= 1 \end{cases}$$

```
def   factorial(n) :
    if   n == 1 :
        return   1
    else :
        return   n * factorial(n-1)
```

12. 변수의 범위

- 지역변수(local variable)
  - 함수 안에서 생성되는 변수
  - 함수 안에서만 사용 가능.
  - 함수가 호출될 때 생성되고 함수 호출 종료 시 사라짐
- 전역변수(global variable) :
  - 함수의 외부에서 생성된 변수
  - 프로그램 어디서나 사용 가능



12. 변수의 범위

- 지역변수(local variable)

```
def  sub() :
    s = "I like a banana."
    print(s)


sub()
print(s)
```

- 전역변수(global variable)

```
def  sub() :
    print(s)
s = "I like a apple."
sub()
print(s)
```

12. 변수의 범위

- 지역변수(local variable)
  - 함수 안에서 생성되는 변수
  - 함수 안에서만 사용 가능.
  - 함수가 호출될 때 생성되고 함수 호출 종료 시 사라짐
- 전역변수(global variable) :
  - 함수의 외부에서 생성된 변수
  - 프로그램 어디서나 사용 가능

```
def  sub() :
    s = "I like a banana."
    print(s)


sub()
print(s)
```

```
def  sub() :
      print(s)

s = "I like a apple."
sub()
print(s)
```

12. 변수의 범위

Filename : scope1.py

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

r = float(input("원의 반지름 : "))
area = calculate_area(r)
print(area)
print(result)
```

→ result 는 지역변수 : 에러

Filename : scope2.py

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

r = float(input("원의 반지름 : "))
area = calculate_area()
print(area)
```

→ r은 전역변수

13. 함수 안에서 전역변수 사용하기

- 전역변수와 지역변수는 같은 이름을 가질 수 있다
- 함수 안에서 전역변수의 값을 변경하면 지역변수로 처리한다.
- global 키워드로 함수 안에서 전역변수를 사용할 수 있다.

Filename : scope3.py

```
def   calculate_area(radius):
    area = 3.14 * radius ** 2   → 새로운 area 변수(지역변수) 만들어짐
    return


area = 0
r = float(input("원의 반지름 : "))
calculate_area(r)
print(area)
```

13. 함수 안에서 전역변수 사용하기

Filename : scope4.py

```
def   calculate_area(radius):
    global   area
    area = 3.14 * radius ** 2   → 전역변수 area에 계산 결과 저장
    return


area = 0
r = float(input("원의 반지름 : "))
calculate_area(r)
print(area)
```