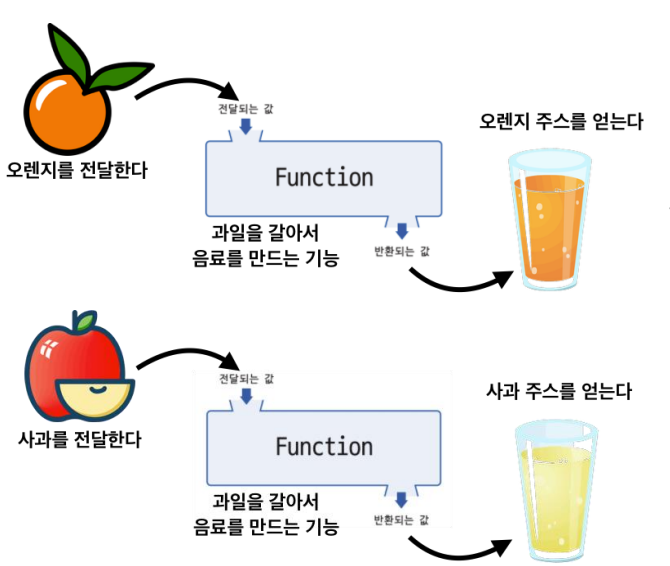
데이터분석 프로그래밍
함수

임현기

함수

- 파이썬 프로그램의 모듈화
  - 함수: 반복적으로 사용하는 코드를 묶은 것
  - 객체: 코드 중에서 독립적인 단위로 분리할 수 있는 조각
  - 모듈: 프로그램의 일부를 가진 독립적인 파일

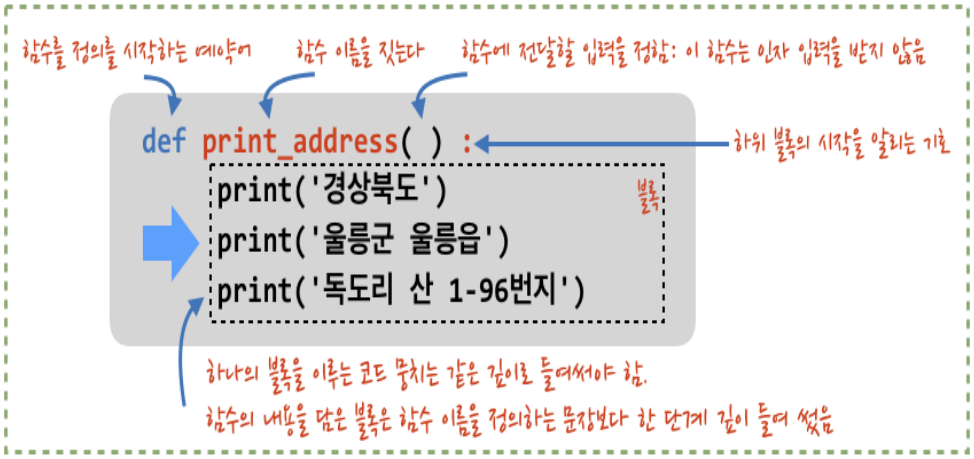
함수



함수는 자료를 전달받아서 일을 하고 그 결과를 돌려준다

전달되는 값이 다르면 결과도 다를 것이다. (예: 사과, 오렌지)

함수 작성 및 호출



- def 키워드를 이용하여 함수를 정의
- 함수의 이름을 적은 후에 소괄호 ()와 콜론: 을 붙임
- 함수 내 문장들은 반드시 들여쓰기를 해야 함

함수 작성 및 호출

- 함수 안에 코드들은 자동으로 실행되지 않음
- 호출되어야 함수 내의 코드가 실행 됨

- 함수 호출: 함수의 이름을 적어주어야 함

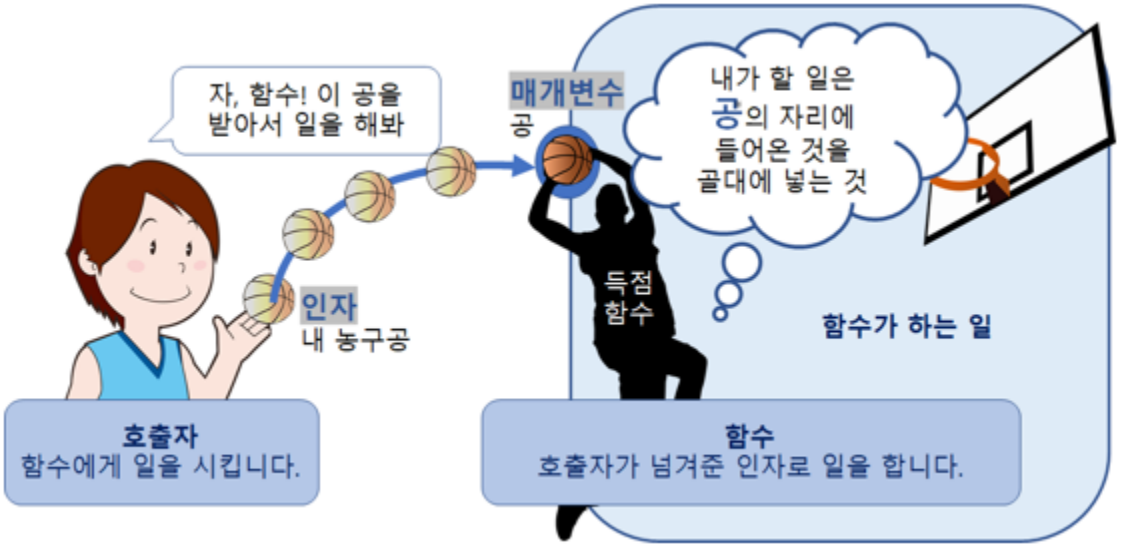
```
def print_address() :
    print('경상북도')
    print('울릉군 울릉읍')
    print('독도리 산 1-96번지')

print_address()  # 정의한 함수를 호출
```

```
경상북도
울릉군 울릉읍
독도리 산 1-96번지
```

함수 호출로 인한 수행결과.

반드시 함수 호출문이 있어야만 함수가 수행된다.

함수 인자


자, 함수! 이 공을
받아서 일을 해봐
매개변수
공
내가 할 일은
공의 자리에
들어온 것을
골대에 넣는 것
득점
함수
인자
내 농구공
함수가 하는 일
호출자
함수에게 일을 시킵니다.
함수
호출자가 넘겨준 인자로 일을 합니다.

함수 인자

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

print_address("홍길동")
print_address("널널한 교수")
```

함수의 매개변수 name을 통해서 그 결과를 다르게 만들 수 있다

함수에 전달되는 실제값으로 인자라고 함 : "홍길동" 대신 여러분의 이름도 가능하다

- 함수 정의부 소괄호 안에 변수 name
- name을 통해 함수로 값이 전달 됨

- 인자: 함수 호출 시 전달되는 실제 값
- 매개변수: 함수 내부에서 전달받는 변수

함수 인자

함수 내부에서 전달 받는 변수 : 매개변수

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

print_address("홍길동")
print_address("널널한 교수")
```

```
서울 특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
서울 특별시 종로구 1번지
파이썬 빌딩 7층
널널한 교수
```

함수에 넘겨지는 값 : 인자

도전문제 6.2

print_address2(name, address)와 같이 2개의 매개변수를 가지는 함수를 정의하여라. 이 함수를 호출할 때 print_address("홍길순", "부산광역시 남구 광안로 10번길 1-1")라고 호출하면 다음과 같은 출력이 나타나도록 함수 몸체를 작성하여라.

```
이름 : 홍길순
주소: 부산광역시 남구 광안로 10번길 1-1
```

함수 반환 값

- 반환 값: 함수로부터 되돌아오는 값
- return 키워드: 함수 내부의 값을 외부로 보낼 때

함수 반환 값

- 원의 반지름을 보내면 원의 면적을 계산해서 반환하는 함수

```
def calculate_area(radius):
    area = 3.14 * radius**2
    return area                 # 이전 줄에서 구한 area 값을 호출문에 돌려준다
```

```
c_area = calculate_area(5.0)    # calculate_are() 함수가 계산한 값을 c_area에 저장
```

```
>>> print(calculate_area(5.0))
78.5
>>> area_sum = calculate_area(5.0) + calculate_area(10.0)
>>> print(area_sum)
```

  - 함수 호출만 하면 그 반환 값은 사용되지 않음

```
>>> calculate_area(10.0)    # 함수를 호출만 하고 그 반환값을 사용안함
```

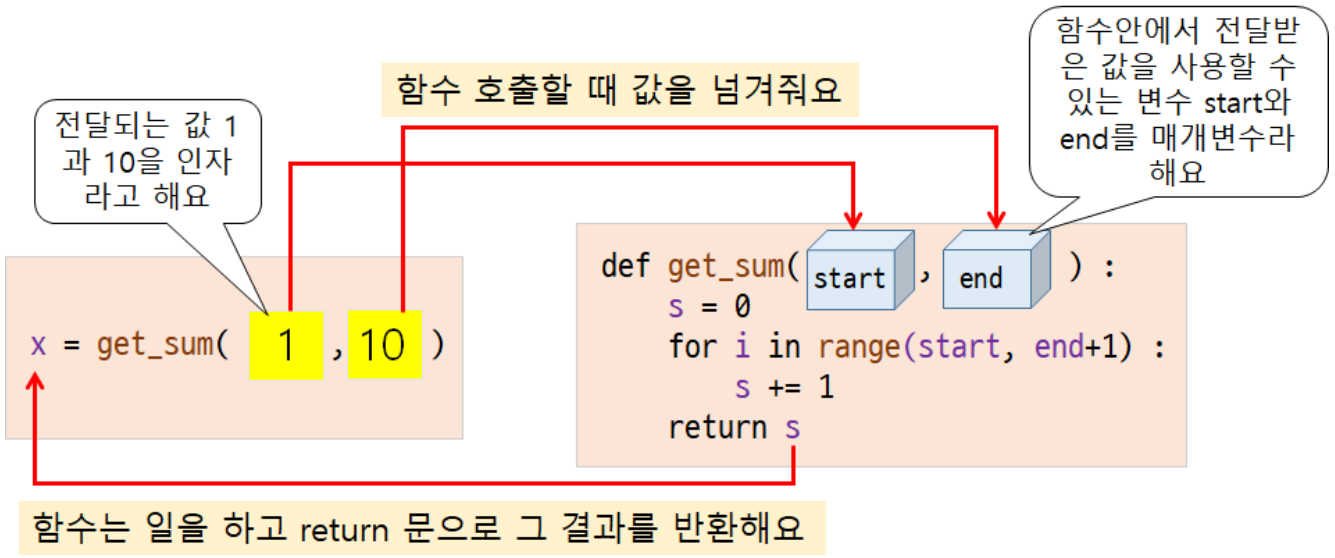
복수 개의 인자

- 2개의 정수 start와 end를 받아서 start에서 end까지의 합을 계산하는 함수 get_sum()

```
def get_sum(start, end):                # start, end를 매개변수로하여 인자를 받는다
    s = 0
    for i in range(start, end+1):       # start부터 end까지 정수의 합을 구함
        s += i
    return s                            # start부터 end까지 수의 합을 반환한다

print(get_sum(1, 10))                   # 1에서 10까지 정수의 합 55를 출력한다
```

```
55
```

복수 개의 인자

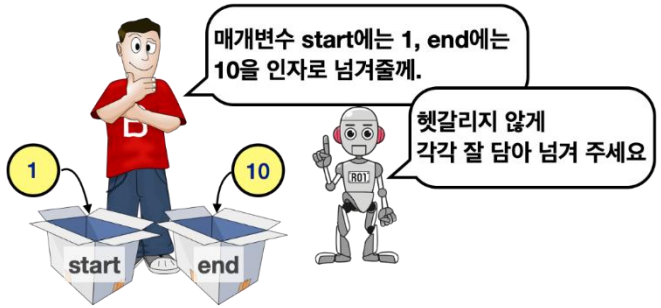

함수 호출할 때 값을 넘겨줘요
전달되는 값 1과 10을 인자라고 해요
함수안에서 전달받은 값을 사용할 수 있는 변수 start와 end를 매개변수라 해요
x = get_sum(1 , 10)
def get_sum(start , end) :
s = 0
for i in range(start, end+1) :
s += 1
return s
함수는 일을 하고 return 문으로 그 결과를 반환해요

복수 개의 인자

- 매개변수의 개수와 넘어가는 인자의 개수가 일치해야 함
- 매개변수의 개수와 인자의 개수가 일치하지 않으면 오류가 발생

```
x = get_sum(1, 10) # 1과 10이 인자가 된다.
print('x =', x)

y = get_sum(1, 20) # 1과 20이 인자가 된다.
print('y =', y)
```

```
x = 55
y = 210
```



복수 개의 반환 값

- 파이썬은 여러 개의 값을 반환 가능

```
def sort_num(n1, n2):           # 2개의 값을 받아오는 함수
    if n1 < n2:
        return n1, n2           # n1이 더 작으면 n1, n2 순서로 반환
    else:
        return n2, n1           # n2가 더 작으면 n2, n1 순서로 반환

print(sort_num(110, 210))       # 110과 210을 함수의 인자로 전달하고 반환되는 값을 출력
print(sort_num(2100, 80))
```

```
(110, 210)
(80, 2100)
```

항상 작은 수, 큰 수 쌍이 반환된다

복수 개의 반환 값

```
def calc(n1, n2):
    return n1 + n2, n1 - n2, n1 * n2, n1 / n2  # 덧셈, 뺄셈, 곱셈, 나눗셈 결과를 반환

n1, n2 = 200, 100
t1, t2, t3, t4 = calc(n1, n2)  # 네 개의 값을 반환받기 위해 4개의 변수를 사용함
print(n1, '+', n2, '=', t1)
print(n1, '-', n2, '=', t2)
print(n1, '*', n2, '=', t3)
print(n1, '/', n2, '=', t4)
```

사칙연산의 결과를 반환받는다.

```
200 + 100 = 300
200 - 100 = 100
200 * 100 = 20000
200 / 100 = 2.0
```

도전문제 6.3

a, b, c의 세 인자를 받아서 이 세 수의 제곱을 반환하는 함수 get_square(a, b, c)를 작성하여라. 그리고 이 함수가 반환하는 세 개의 결과 a_sq, b_sq, c_sq를 출력하도록 하여라.

```
a, b, c = 1, 2, 3
a_sq, b_sq, c_sq = get_square(a, b, c)
print(a, '제곱 :', a_sq, ', ', b,'제곱 :', b_sq,', ',c,'제곱 :',c_sq)
```

```
1 제곱 : 1 ,  2 제곱 : 4 ,  3 제곱 : 9
```

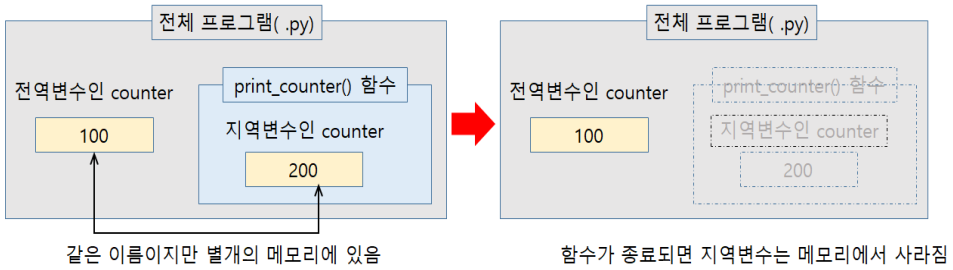
변수의 범위

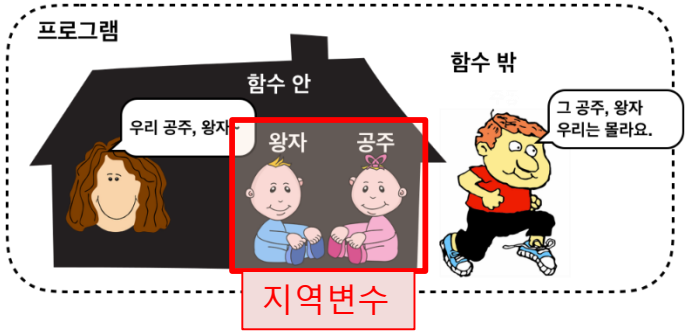
```
def print_counter():
    counter = 200
    print('counter =', counter)  # 함수 내부의 counter 값

counter = 100
print_counter()
print('counter =', counter)      # 함수 외부의 counter 값
```

```
counter = 200
counter = 100
```

- 지역변수: 함수 안에서 생성되는 변수
  - 함수 종료 시 사라짐
  - 전역변수 counter = 100
  - 지역변수 counter = 200

변수의 범위





함수가 호출될 때 생성되어 호출이 완료되면 메모리에서 사라지는 지역변수

```
def print_counter():
    counter = 200
    print('counter =', counter)

counter = 100
print_counter()
print('counter =', counter)
```

print_counter() 함수의 호출에 관계없이 메모리에서 유지되는 전역변수

전역 변수

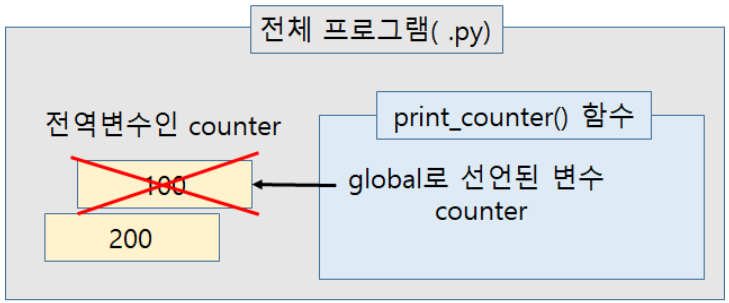
- 함수 내에서 전역 변수를 사용하려면
  - global 키워드

```
def print_counter():
    global counter      # 함수 외부의 전역변수 counter를 사용하겠다는 선언
    counter = 200
    print('counter =', counter)  # 함수 내부의 counter 값

counter = 100
print_counter()
print('counter =', counter)      # 함수 외부의 counter 값
```

```
counter = 200
counter = 200
```

전역 변수

- 전역변수를 너무 많이 사용하면 문제 가능성이 커짐
  - 꼭 필요한 경우에만 사용



global로 선언된 변수는 전역변수를 참조함

```
def print_counter():
    global counter
    counter = 200
    print('counter =', counter)

counter = 100
print_counter()
print('counter =', counter)
```

지역변수가 아닌 전역변수 counter를 사용하게 된다. 함수 내부에서 값을 바꾸면 외부에서도 그 값이 바뀌게 된다

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

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

```
원의 반지름: 5.0
78.5
```

area 값을 반환하지 않아도 외부에서 그 값이 0에서 78.5로 바뀐다

디폴트 인자

- 디폴트 인자: 함수의 매개변수가 가지는 기본 값

```
def order(num, pickle, onion) :
    print('햄버거 {0} 개 - 피클 {1}, 양파 {2}'.format(num, pickle, onion))

order(1, False, True)
```

```
햄버거 1 개 - 피클 False, 양파 True
```

  - order() 함수에 3개의 인자를 전달하지 않으면 오류 발생

```
>>> order(1)     # 인자를 1개만 전달하여 order()을 호출함
...
TypeError: order1() missing 2 required positional arguments: 'pickle' and 'onion'
```

디폴트 인자

```
def order(num, pickle = True, onion = True) :
    print('햄버거 {0} 개 - 피클 {1}, 양파 {2}'.format(num, pickle, onion))

order(1, pickle = False, onion = True)
order(2)     # 햄버거 2개를 주문, 디폴트로 pickle, onion 값은 True임
```

```
햄버거 1 개 - 피클 False, 양파 True
햄버거 2 개 - 피클 True, 양파 True
```

- pickle과 onion 매개변수에 디폴트 인자로 True 설정
- order(2)와 같이 호출해도 오류없이 수행
- num에 대한 디폴트 값을 주지 않음

디폴트 인자

order(3) 만하면 자동으로 order(3, True, True) 가 된다

- 디폴트 값이 있는 매개 변수는 순서로 바꾸어 인자를 넣을 수 있음

```
>>> order(3, onion = False, pickle = True)     # 인자의 순서를 바꾸었다.
햄버거 3 개 - 피클 True, 양파 False
```

키워드 인자

- 여러 인자들은 위치에 따라 구별
  - 예: power(2, 10), power(10, 2)

- 키워드 인자
  - 인자들 앞에 키워드를 두어서 인자들 구분
  - 혼동을 줄이고 안전한 프로그래밍 가능

```
def power(base, exponent):
    return base**exponent    # base가 밑이고, exponent가 지수값이다.
```

```
>>> power(2, 10)  # 2의 10승을 반환한다
1024
>>> power(10, 2)  # 10의 2승을 반환한다
100
```

어떤 값이 밑이고 지수인지 헷갈릴 수 있다 -> 오류의 원인이 된다

키워드 인자

```
def power(base, exponent):
    return base**exponent    # base가 밑이고, exponent가 지수값이다.
```

```
>>> power(2, 10)  # 2의 10승을 반환한다
1024
>>> power(10, 2)  # 10의 2승을 반환한다
100
```

어떤 값이 밑이고 지수인지 헷갈릴 수 있다 -> 오류의 원인이 된다

```
>>> power(base=2, exponent=10)  # 2의 10승을 반환한다
1024
>>> power(exponent=10, base=2)  # 2의 10승을 반환한다
1024
```

밑(base)과 지수(exponent)를 명시를 해주므로 헷갈리지 않는다.

키워드 인자

- 위치 인자가 먼저 나와야 함
- 키워드 인자가 나온 뒤에 위치 인자가 나올 수 없음

```
>>> power(base = 10, 2)    # 문법 오류
SyntaxError: positional argument follows keyword argument
```

주단위로 봉급을 받는 아르바이트생이 있다고 하자. 현재 시급과 일한 시간을 입력하면 주급을 계산해주는 함수 weeklyPay(rate, hour)를 만들고 이 함수를 호출하여 주급을 출력하는 프로그램을 작성해보자. 30시간이 넘는 근무 시간에 대해서는 시급의 1.5배를 지급한다고 하자. 이 함수를 구현하고 입력받은 값을 r와 h 변수에 넣어서 키워드 인자로 함수 weeklyPay()를 호출하는 프로그램을 작성하여라.

원하는 결과

```
시급을 입력하시오:10000
근무 시간을 입력하시오:38
주급은 420000.0
```

```
def weeklyPay(rate, hour):
    if (hour > 30):
        money = rate*30 + 1.5*rate*(hour-30)
    else:
        money = rate*hour
    return money

r = int(input("시급을 입력하시오: "))        # 시급입력받기
h = int(input("근무 시간을  입력하시오: "))  # 근무시간 입력받기
print("주급은 " + str(weeklyPay(rate = r, hour = h)))
```

[27, 90, 30, 87, 56]와 같은 리스트에서 최댓값이나 최솟값을 찾는 getMinMax() 함수를 만드려고 한다. 이 함수는 리스트를 첫 번째 매개변수로 하고, 두 번째 매개변수는 method라는 이름을 가지는 키워드 인자를 받는다고 하자. 이 method에 주어지는 인자가 'max'이면 최댓값, 'min'이면 최솟값을 찾아 출력하도록 하는 것이다.

원하는 결과

```
[27, 90, 30, 87, 56]
최댓값을 원하면 max, 최솟값을 원하면 min을 입력하시오: max
90
[27, 90, 30, 87, 56]
최댓값을 원하면 max, 최솟값을 원하면 min을 입력하시오: min
27
[27, 90, 30, 87, 56]
최댓값을 원하면 max, 최솟값을 원하면 min을 입력하시오: maximum
illegal method
```

```
def getMinMax(mylist, method = 'max'):
    minValue = 99999999999999999999999999999999
    maxValue = -minValue

    if method == 'max' :
        for value in mylist:
            if value > maxValue:
                maxValue = value;
        return maxValue
    elif method == 'min' :
        for value in mylist:
            if value < minValue:
                minValue = value;
        return minValue
    else :
        print('illegal method')

list_data = [27, 90, 30, 87, 56]
while(True) :
    print(list_data)
    s = input('최댓값을 원하면 max, 최솟값을 원하면 min을 입력하시오: ')
    print(getMinMax(list_data, s))
```

재귀 함수

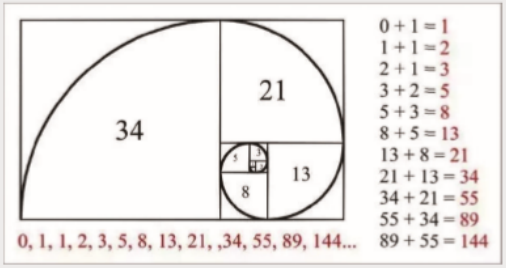
- 재귀 함수: 함수 내부에서 자기 자신을 호출하는 함수

- 팩토리얼: n! = n * (n-1)!

```
1! = 1
2! = 2 * 1
3! = 3 * 2 * 1
...
n! = n * (n-1) * (n-2) * ... * 2 * 1
```

```
def factorial(n):                       # n!의 재귀적 구현
    if n <= 1 :                         # 종료 조건이 반드시 필요하다
        return 1
    else :
        return n * factorial(n-1)       # n * (n-1)! 정의에 따른 구현

print('4! = ', factorial(4))            # 인자로 4를 넣어 호출
```

피보나치 수열이란 앞의 두 항을 더해 다음 항을 만드는 수열이다. 피보나치 수열의 첫 번째 항과 두 번째 항은 `0`, `1`로 설정한다. 이 수열에 속한 수를 피보나치 수라고 한다. 피보나치 수열의 n 번째 항을 계산하여 반환하는 함수 `fibonacci(n)`를 작성해보자.



원하는 결과

```
몇 번째 항: 9
21
```

```
def fibonacci(n):
    if n < 0:                                   # 입력 오류 검사
        print("잘못된 입력입니다.")
    elif n == 1:                                # 재귀호출 중단 조건
        return 0
    elif n == 2:                                # 재귀호출 중단 조건
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)  # 재귀호출

i = int(input("몇 번째 항: "))
print(fibonacci(i))
```

모듈

- 모듈: 파이썬 함수나 변수 또는 클래스들을 모아 놓은 스크립트 파일

- 스크립트나 대화창에서 모듈을 불러올 때

```
import 모듈이름1 [, 모듈이름2, ... ]
```

모듈

- 표준 라이브러리: 파이썬 설치와 함께 제공되는 모듈
  - 문자열와 텍스트 처리
  - 이진 데이터 처리
  - 날짜와 시간 처리

```
>>> import datetime                # 날짜와 시간을 다루는 모듈
>>> datetime.datetime.now()
datetime.datetime(2021, 1, 2, 6, 57, 27, 904565)
```

```
>>> today = datetime.date.today()
>>> print(today)
2021-09-14
>>> today
datetime.date(2021, 9, 14)
>>> today.year
2021
>>> today.month
9
>>> today.day
14
```

모듈 생성

```
# filename: my_func.py
def mf_print(msg, n = 1) :
    print(msg * n)
```

- 파일의 이름이 모듈의 이름이 됨
- 모듈이름.함수이름() 형태로 호출 가능

모듈 생성

```
# filename: main.py
import my_func

my_func.mf_print("my_func was imported ", 3)  # my_func 모듈의 mf_print() 함수 호출
```

```
my_func was imported my_func was imported my_func was imported
```

```
# filename: main.py
import my_func as mf

mf.mf_print('[alias]', 5)  # mf라는 별명을 사용해서 메시지를 5회 반복출력
```

```
[alias][alias][alias][alias][alias]
```

```
# filename: main.py
from my_func import mf_print

mf_print('-no module name-', 2)  # mf_print()를 호출할 때 my_func.을 사용안해도 됨
```

```
-no module name--no module name-
```

```
# filename: main.py
from my_func import *

mf_print('-no module name-', 2)
```

```
-no module name--no module name-
```