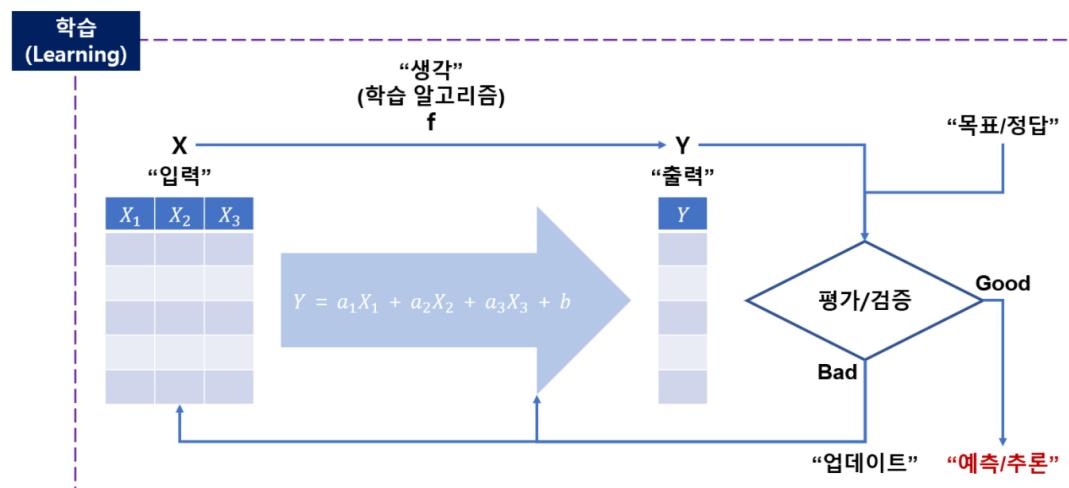
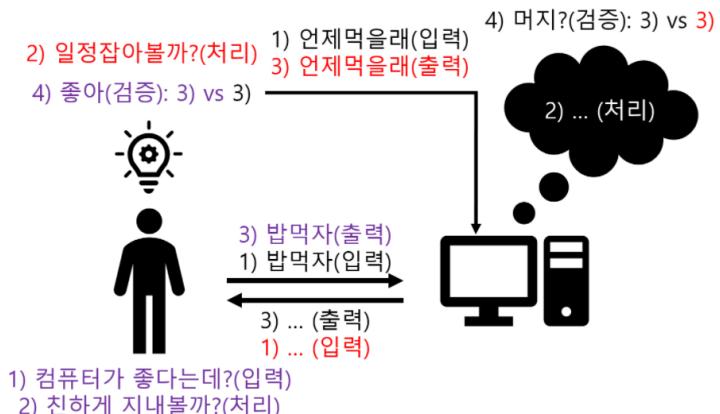


1 파이썬함수: 사람의 생각을 정교하게 구현/자동화하다

- 목적: 사람의 생각(f)을 컴퓨터로 구현하여 자동으로 입력에 대응되는 출력 반환

- 사람의 "생각" 또는 컴퓨터의 "처리" 과정을 "함수"로 관리하면 효율적이고 편리하게 반복사용 가능
- 입력(X)을 받아 "함수"를 거쳐 출력(Y) 된다: $Y = f(X)$
- print, input 등도 모두 파이썬에서 미리 만들어 둔 내장 함수
- 특정 기능을 수행하는 코드의 집합으로 여러 실행 문장을 하나로 묶는 기능
- 코드의 용도를 잘 정리해 둘 수 있고, 얼마든지 재사용 가능하며, 실수를 줄일 수 있음
- 복잡한 것들도 결국 "함수"로 작성할 수 있고, 결국 사람이 수작업으로 해야 할 것들이 줄어들고 자동화(인공지능) 가능



< 3단계 >

< 1단계 >

“문제정의”

 f $X \rightarrow Y$

“커뮤니케이션” “기획+평가”

< 2단계 >

“생각”

 f

“정리”

“평가”

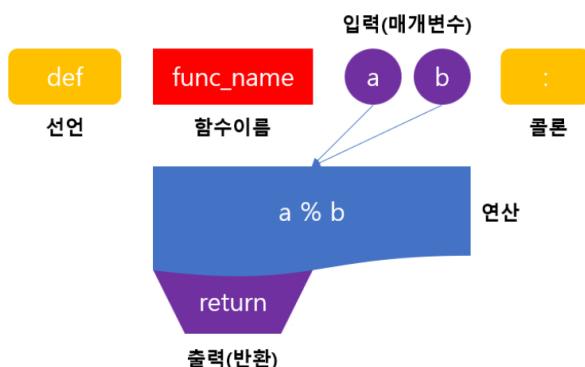
 f

“끝(Y)/시작(X)”

2 파이썬함수의 모양과 종류

- def 키워드를 사용하여 정의한다
- 함수에는 입력값과 출력값이 있으며 없을 수도 있다
- 입력은 매개변수라고도 하고 출력은 반환값이라고도 한다
- 함수 선언시 마지막에 콜론을 입력한다
- 입력값: 매개변수, 인수 등

- **매개변수**: 함수 선언시 입력 값이나 변수
- **인수**: 함수 사용(호출)시 입력 값이나 변수
- **출력값**: 결과값, 반환값, 돌려주는 값 등
- **return**: 값을 반환하는 + 함수를 종료하는 기능



```
In [1]: 5 % 2
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[1]: 1

```
In [2]: 7 % 4
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[2]: 3

```
In [3]: # 함수정의
# 함수의 이름(함수이름)은 func_name이고 입력으로 2개의 값을 받아 나눈 나머지를 출력
# 정의한 함수를 사용(호출)하여 5를 2로 나눈 나머지를 출력

def func_name(a, b):
    return a % b

func_name(5,2)
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[3]: 1

```
In [4]: # 변수값으로 출력 가능
def func_name(a, b):
    result = a % b
    return result

func_name(5,2)
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[4]: 1

```
In [5]: # 변수값으로 출력 가능
def func_name(a, b):
    result = a % b
    return result

y = func_name(5,2)
y
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[5]: 1

```
In [6]: # 예시 결과 확인
func_name(1,3)
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[6]: 1

```
In [7]: # 예시 결과 확인
func_name(50,4)
executed in 13ms, finished 02:02:08 2021-11-06
```

Out[7]: 2

```
In [8]: # print(a)
# print(b)
executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [9]: # 예시 결과 확인
def hello():
    print('Hello, world!')

hello()
executed in 14ms, finished 02:02:08 2021-11-06
```

Hello, world!

• 함수의 실행 순서:

1. 파이썬 함수 작성 및 실행
2. 작성 함수 호출(함수 생성 전 호출 불가)
3. 호출된 함수 실행
4. 호출된 함수 출력
5. 호출된 함수 종료

```
In [10]: def hello():
    pass
executed in 14ms, finished 02:02:08 2021-11-06

In [11]: hello()
executed in 14ms, finished 02:02:08 2021-11-06
```

▼ 2.1 입력도 없고 출력도 없는 함수

```
In [12]: # 함수를 작성하고, 함수를 호출하여 결과 확인
def func1():
    print('My name is KK')
func1()
func1()
executed in 14ms, finished 02:02:08 2021-11-06

My name is KK
My name is KK
```

▼ 2.2 입력만 있는 함수

```
In [13]: # 함수를 작성하고, 함수를 호출하여 결과 확인
# a = 2, b = 4
def func2(a, b):
    print(f'{a} 곱하기 {b} = {a*b}')
func2(2,4)
executed in 14ms, finished 02:02:08 2021-11-06

2 곱하기 4 = 8
```

```
In [14]: # 예시 결과 확인
func2(5,4)
executed in 14ms, finished 02:02:08 2021-11-06
```

5 곱하기 4 = 20

```
In [15]: # 예시 결과 확인
func2(10,2)
executed in 15ms, finished 02:02:08 2021-11-06
```

10 곱하기 2 = 20

▼ 2.3 출력만 있는 함수

```
In [16]: # 함수를 작성하고, 함수를 호출하여 결과 확인
def func3():
    return 'What is your name? '
func3()
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[16]: 'What is your name? '

```
In [17]: # 예시 결과 확인
y = func3()
executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [18]: # 예시 결과 확인
y = func3()
y + 'KK'
executed in 15ms, finished 02:02:08 2021-11-06
```

Out[18]: 'What is your name? KK'

▼ 2.4 입력이나 출력이 여러개인 함수

```
In [19]: # 함수를 작성하고, 함수를 호출하여 결과 확인
# a=1, b=2, c=3, d=4
def input_several(a, b, c, d):
    summation = a + b + c + d
    return summation
input_several(1,2,3,4)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
Out[19]: 10
```

```
In [20]: # input_several(1,2,3,4,5)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [21]: # 함수를 작성하고, 함수를 호출하여 결과 확인
```

```
# 모든 입력값의 합과 차를 계산하는 함수
```

```
# a=1, b=2, c=3, d=4
```

```
def output_several(a, b, c, d):
```

```
    plus = a + b + c + d
```

```
    minus = -a - b - c - d
```

```
    return plus, minus
```

```
output_several(1,2,3,4)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
Out[21]: (10, -10)
```

```
In [22]: # 변수로 받아 출력 가능
```

```
x, y = output_several(1,2,3,4)
```

```
print(x, y)
```

```
executed in 12ms, finished 02:02:08 2021-11-06
```

```
10 -10
```

3 파이썬함수 응용

3.1 위치인수 활용

- 위치인수(Positional Argument): 함수의 입력을 순서대로 넣는 방식

```
In [23]: print(10, 20, 30)
```

```
executed in 13ms, finished 02:02:08 2021-11-06
```

```
10 20 30
```

```
In [24]: def print_numbers(a, b, c):
```

```
    print(a)
```

```
    print(b)
```

```
    print(c)
```

```
print_numbers(10, 20, 30)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10
```

```
20
```

```
30
```

```
In [25]: # 함수의 입력값은 코드의 입력 위치와 매핑되어 자동으로 인식
```

```
def print_numbers(a, b, c, d, e, f, g, h, i, j):
```

```
    print(a)
```

```
    print(b)
```

```
    print(c)
```

```
    print(d)
```

```
    print(e)
```

```
    print(f)
```

```
    print(g)
```

```
    print(h)
```

```
    print(i)
```

```
    print(j)
```

```
print_numbers(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10
```

```
20
```

```
30
```

```
40
```

```
50
```

```
60
```

```
70
```

```
80
```

```
90
```

```
100
```

3.2 아규먼트인수 활용

- 언팩킹(Unpacking): 입력변수(인수) 앞에 *** 기호를 붙이면 리스트 내부 값들을 모두 모아 튜플로 만들고 차례대로 반영
- 함수의 입력(매개변수)의 갯수와 리스트 내부 요소의 갯수는 같아야 함

```
In [26]: def print_numbers(a, b, c):
```

```
    print(a)
```

```
    print(b)
```

```
    print(c)
```

```
# print_numbers(10, 20, 30)
```

```
print_numbers(*[10, 20, 30])
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10  
20  
30
```

```
In [27]: # x에 [10, 20, 30]를 반영하고 이를 아규먼트를 사용하여 print_numbers 함수를 출력  
x = [10, 20, 30]  
print_numbers(*x)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10  
20  
30
```

```
In [28]: # 입력 갯수가 다르면 오류 발생  
# print_numbers(*[10, 20, 30, 40, 50])
```

```
executed in 15ms, finished 02:02:08 2021-11-06
```

- **가변인수(Variable Argument):** 입력의 갯수가 정해지지 않은 경우 사용
- 입력 값을 한개, 열개, 넣지 않아도 됨

```
def 함수이름(*매개변수):  
    문장
```

```
In [29]: # 가변인수를 사용해서 입력의 갯수를 조정 가능
```

```
# def print_numbers(*a):  
#     print(a)  
  
x = [10, 20, 30, 40, 50]  
print_numbers(*x)
```

```
executed in 15ms, finished 02:02:08 2021-11-06
```

```
(10, 20, 30, 40, 50)
```

```
In [30]: # 다양한 상황 활용 가능
```

```
x = [10, 20, 30, 40, 50, 60, 70]  
print_numbers(*x)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
(10, 20, 30, 40, 50, 60, 70)
```

```
In [31]: # 입력값들의 개별 출력 가능
```

```
# def print_numbers(*args):  
#     for arg in args:  
#         print(arg)  
  
print_numbers(10)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10
```

```
In [32]: # 예시 출력 확인
```

```
print_numbers()
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [33]: # 예시 출력 확인
```

```
print_numbers(10, 20, 30, 40)
```

```
executed in 13ms, finished 02:02:08 2021-11-06
```

```
10  
20  
30  
40
```

```
In [34]: # 예시 출력 확인
```

```
y = [10, 20, 30, 40, 50, 60, 70]  
print_numbers(*y)
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10  
20  
30  
40  
50  
60  
70
```

```
In [35]: # 값의 위치를 기반으로 출력 가능
```

```
# def print_numbers(args):  
#     length = len(args)  
#     for i in range(length):  
#         print(args[i])
```

```
print_numbers([10])
```

```
executed in 14ms, finished 02:02:08 2021-11-06
```

```
10
```

```
In [36]: # 예시 출력 확인
```

```
print_numbers([10, 20, 30])
```

```
executed in 15ms, finished 02:02:08 2021-11-06
```

```
10  
20
```

```
In [37]: # 입력값들의 합을 계산
def total_sum(*args):
    a = 0
    for i in args:
        a = a + i
    return a

total_sum(1,2)
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[37]: 3

```
In [38]: # 3,4,5,6의 합을 위해 함수를 사용하여 출력
total_sum(3,4,5,6)
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[38]: 18

```
In [39]: # 예시 출력 확인
total_sum()
executed in 14ms, finished 02:02:08 2021-11-06
```

Out[39]: 0

- ***args:** "arguments"의 약자로 함수를 호출할 때 입력의 갯수가 여러개여도 상관없이 받음
- args로 입력된 값들을 튜플 타입으로 변환하여 함수 내부로 모두 전달
- 고정인수와 가변인수를 함께 사용할 때는 고정 매개변수를 앞에 지정하고, 다음 매개변수로 *args

```
def 함수이름(고정 매개변수, *args):
    문장
```

```
In [40]: # 위치인수와 가변인수를 모두 입력으로 활용 가능
def print_numbers(a, *args):
    print(a, args)
    for arg in args:
        print(arg + a)

print_numbers(5, 2, 3)
executed in 14ms, finished 02:02:08 2021-11-06
```

5 (2, 3)
7
8

```
In [41]: def print_numbers(a, *args):
    print(a)
    print(args)
executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [42]: print_numbers(1)
executed in 14ms, finished 02:02:08 2021-11-06
```

1
()

```
In [43]: print_numbers(1, 10, 20)
executed in 14ms, finished 02:02:08 2021-11-06
```

1
(10, 20)

```
In [44]: print_numbers(*[10, 20, 30])
executed in 14ms, finished 02:02:08 2021-11-06
```

10
(20, 30)

- **Question:** 리스트 언팩킹(*args)을 사용하여 "get_max_score" 함수를 만들고, 아래의 코드를 사용하여 가장 높은 점수가 출력되도록 만드시오

```
korean, english, mathematics, science = 100, 86, 81, 91

max_score = get_max_score(korean, english, mathematics, science)
print('높은점수: ', max_score)

max_score = get_max_score(english, science)
print('높은점수: ', max_score)
```

```
In [45]: # 평균값
# 아래코드 실행하여 최고점수 100 출력
# max_score = get_max_score(korean, english, mathematics, science)
# print('높은점수: ', max_score)

# 아래코드 실행하여 최고점수 91 출력
# max_score = get_max_score(english, science)
# print('높은점수: ', max_score)
executed in 14ms, finished 02:02:08 2021-11-06
```

- **Question:** 입력의 갯수는 정해지지 않았으나 모든 입력의 평균값을 계산해주는 함수를 만드시고, 임의의 입력이 2개일때 그리고 입력이 5개일때 평균값

In []:

executed in 18ms, finished 13:07:33 2021-10-07

3.3 키워드인수 활용

- **키워드 인수(Keyword Argument)**: 함수의 값 입력시 매개변수의 위치와 용도를 함께 기억해야하지만, 각 인수마다 이름을 붙여 그럴 필요를 줄여줌
- 키워드 인수 사용시 매개변수의 입력 순서를 맞출 필요가 없음

```
def 함수이름(키워드=값):
    문장
```

```
In [46]: # 키워드 인수를 사용하여 입력값을 고정 가능
def personal_info(name, age, address):
    print('이름:', name)
    print('나이:', age)
    print('주소:', address)

executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [47]: # 위치 인수 사용하여 본인 이름, 나이, 주소를 출력
personal_info('KK', 30, '서울시 용산구')

executed in 14ms, finished 02:02:08 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

```
In [48]: # 입력값의 위치를 바꾸면 결과도 변경
personal_info('서울시 용산구', 'KK', 30)

executed in 15ms, finished 02:02:08 2021-11-06
```

이름: 서울시 용산구
나이: KK
주소: 30

```
In [49]: # 입력값의 위치를 바꾸더라도 결과가 달라지지 않도록 고정 가능
personal_info(name='KK', age=30, address='서울시 용산구')

executed in 15ms, finished 02:02:08 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

```
In [50]: # 다양한 방식으로 사용 가능
personal_info(name='KK', address='서울시 용산구', age=30)

executed in 14ms, finished 02:02:08 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

3.4 딕셔너리인수 활용

- 매개변수가 많아지게 되면 함수의 가로 길이가 길어져 가독성이 떨어지고 입력을 이해하기 어려움
- 딕셔너리 언패킹(Dictionary Unpacking): 키와 값으로 이루어진 딕셔너리 활용 함수 입력 제어
- 함수의 매개변수 이름과 갯수가 딕션너리의 키의 이름과 갯수와 같아야 함

```
In [51]: def personal_info(name, age, address):
    print('이름:', name)
    print('나이:', age)
    print('주소:', address)

executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [52]: personal_info(name='KK', address='서울시 용산구', age=30)

executed in 14ms, finished 02:02:08 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

```
In [53]: x = {'name': 'KK', 'age': 30, 'address': '서울시 용산구'}

executed in 14ms, finished 02:02:08 2021-11-06
```

```
In [54]: # 임의의 딕셔너리로 입력값을 반영 가능
x = {'name': 'KK',
      'age': 30,
      'address': '서울시 용산구'}

personal_info(**x)

executed in 14ms, finished 02:02:08 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

```
In [55]: # 임의의 딕셔너리라도 입력값의 갯수는 함수의 입력 갯수와 같아야 함
# x = {'name': 'KK',
#       'age': 30,
#       'address': '서울시 용산구'}
```

```

#      'temp': '블라블라'}

# personal_info(**x)
executed in 14ms, finished 02:02:08 2021-11-06

In [56]: # 다양하게 입력 가능
personal_info(**{'name': 'KK', 'age': 30, 'address': '서울시 용산구'})
executed in 14ms, finished 02:02:08 2021-11-06
이름: KK
나이: 30
주소: 서울시 용산구

In [57]: # 일반 변수로 입력시 key만 인식
personal_info(*x) # *기호를 1번만 사용하면 키값만 사용, 2번 사용하면 키와 값을 사용한다는 뜻
executed in 13ms, finished 02:02:08 2021-11-06
이름: name
나이: age
주소: address

• **kwargs: 매개변수 이름은 자유지만 관례적으로 keyword arguments를 줄여서 표현

def 함수이름(**kwargs):
    문장

In [58]: # 키워드 인수의 갯수에 무관하게 함수 구성 가능
def personal_info(**kwargs):
    for kw, arg in kwargs.items():
        print(kw, ':', arg, sep=' ')
executed in 14ms, finished 02:02:08 2021-11-06

In [59]: # 키워드 인수가 하나일 때 출력
personal_info(name='KK')
executed in 13ms, finished 02:02:08 2021-11-06
name:KK

In [60]: # 키워드 인수가 세개일 때 출력
personal_info(name='KK', age=30, address='서울시 용산구')
executed in 14ms, finished 02:02:08 2021-11-06
name:KK
age:30
address:서울시 용산구

In [61]: # 다양하게 입력 가능
y = {'name': 'KK', 'age': 30, 'address': '서울시 용산구'}
personal_info(**y)
executed in 15ms, finished 02:02:08 2021-11-06
name:KK
age:30
address:서울시 용산구

In [62]: # 다양하게 입력 가능
y = {'name': 'KK',
     'age': 30,
     'address': '서울시 용산구',
     'temp': '블라블라'}

personal_info(**y)
executed in 14ms, finished 02:02:08 2021-11-06
name:KK
age:30
address:서울시 용산구
temp:블라블라

In [63]: # 조건문이 포함된 함수에 키워드 인수 활용 가능
def personal_info(**kwargs):
    if 'name' in kwargs: # in으로 딕셔너리 안에 특정 키가 있는지 확인
        print('이름: ', kwargs['name'])
    if 'age' in kwargs:
        print('나이: ', kwargs['age'])
    if 'address' in kwargs:
        print('주소: ', kwargs['address'])
executed in 14ms, finished 02:02:08 2021-11-06

In [64]: # 세개의 키워드 인수 활용 가능
y = {'name': 'KK', 'age': 30, 'address': '서울시 용산구'}
personal_info(**y)
executed in 14ms, finished 02:02:08 2021-11-06
이름: KK
나이: 30
주소: 서울시 용산구

In [65]: # 네개의 키워드 인수 활용 가능
y = {'name': 'KK',
     'age': 30,
     'address': '서울시 용산구',
     'temp': '블라블라'}

```

```

personal_info(**y)
executed in 15ms, finished 02:09 2021-11-06
이름: KK
나이: 30
주소: 서울시 용산구

In [66]: # 입력을 자동으로 dictionary로 인식
def personal_info(**kwargs):
    print(kwargs)

personal_info(name='KK')
executed in 14ms, finished 02:09 2021-11-06
{'name': 'KK'}
```

```

In [67]: # 입력을 자동으로 dictionary로 인식
personal_info(name='KK', age=30)
executed in 8ms, finished 02:09 2021-11-06
{'name': 'KK', 'age': 30}
```

```

In [68]: # 위치인수와 키워드인수를 함께 사용 가능
def personal_info(name, **kwargs):
    print(name)
    print(kwargs)
executed in 12ms, finished 02:09 2021-11-06
```

```

In [69]: personal_info('KK')
executed in 14ms, finished 02:09 2021-11-06
KK
{}
```

```

In [70]: # 입력 갯수에 따라 위치 인수와 키워드 인수로 자동 인식
personal_info('KK', age=30, address='서울시 용산구')
executed in 14ms, finished 02:09 2021-11-06
KK
{'age': 30, 'address': '서울시 용산구'}
```

```

In [71]: # 다양하게 입력 가능
personal_info(**{'name': 'KK', 'age': 30, 'address': '서울시 용산구'})
executed in 14ms, finished 02:09 2021-11-06
KK
{'age': 30, 'address': '서울시 용산구'}
```

3.5 아규먼트 입력과 딕셔너리 입력 동시 활용

- 내장함수들은 보통 위치입력과 키워드입력을 함께 사용하는 편
- 복잡한 함수의 경우 위치입력/키워드입력/딕셔너리입력 모두 사용
- 매개변수 순서에서 *args, **kwargs 순서대로 사용해야 함
- *args는 리스트를 사용하고, **kwargs는 딕셔너리를 사용

```
def 함수이름(arg, *args, **kwargs):
    문장
```

Docstring:

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type: builtin_function_or_method
```

```

In [72]: # 다양한 입력 결합 가능
def custum_print(*args, **kwargs):
    print(*args, **kwargs)

custum_print(1, 2, 3, sep=':', end='')

executed in 14ms, finished 02:09 2021-11-06
1:2:3
```

3.6 특정 매개변수 자동입력(생략)

- 함수의 매개변수에 초기값을 할당하여 함수를 저장하면 생략 가능
- 여러개의 입력 중 특정 입력값이 입력되지 않으면 함수내부에서 지정한 값으로 사용
- 초깃값으로 고정된 입력은 함수입력 표현에서 뒤에 배치해야 함
- 초깃값으로 고정되더라도 함수 활용시 매개변수에 값을 재할당해도 적용 가능

```
def 함수이름(매개변수=값):
```

```
In [73]: def personal_info(name, age, address):
    print('이름:', name)
    print('나이:', age)
    print('주소:', address)

personal_info('KK', 30, '서울시 용산구')
executed in 14ms, finished 02:02:09 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

```
In [74]: # 입력의 값수가 맞지 않으면 에러 출력
# def personal_info(name, age, address):
#     print('이름:', name)
#     print('나이:', age)
#     print('주소:', address)

# personal_info('KK', 30)
executed in 14ms, finished 02:02:09 2021-11-06
```

```
In [75]: # 특정 입력값을 고정 가능
def personal_info(name, age, address='비공개'):
    print('이름:', name)
    print('나이:', age)
    print('주소:', address)
executed in 14ms, finished 02:02:09 2021-11-06
```

```
In [76]: # 고정된 입력값은 별도 입력하지 않아도 됨
personal_info('KK', 30)
executed in 14ms, finished 02:02:09 2021-11-06
```

이름: KK
나이: 30
주소: 비공개

```
In [77]: # 고정 입력값이 있더라도 별도 입력시 별도 입력값을 우선함
personal_info('KK', 30, '서울시 용산구')
executed in 14ms, finished 02:02:09 2021-11-06
```

이름: KK
나이: 30
주소: 서울시 용산구

```
In [78]: # 고정입력값은 마지막에 와야만 함
# def personal_info(name, address='비공개', age):
#     print('이름:', name)
#     print('나이:', age)
#     print('주소:', address)
executed in 15ms, finished 02:02:09 2021-11-06
```

```
def personal_info(name, age, address='비공개'):
def personal_info(name, age=0, address='비공개'):
def personal_info(name='비공개', age=0, address='비공개'):
```

3.7 조건문 결합

- 생각하는 조건이 참(True)이면 명령을 실행하고 거짓(False)이면 명령을 실행하지 않음

```
In [79]: # if ~: 특정 조건을 만족하는 상황 반영
a = 4

if a % 2 == 0:
    print('짝수입니다!')
    print('2의 배수입니다!')
executed in 14ms, finished 02:02:09 2021-11-06
```

짝수입니다!
2의 배수입니다!

```
In [80]: # //if 출력 확인
a = 5

if a % 2 == 0:
    print('짝수입니다!')
    print('2의 배수입니다!')
executed in 14ms, finished 02:02:09 2021-11-06
```

```
In [81]: # if ~ else: 특정 조건을 만족하는 경우와 그렇지 않은 경우를 모두 반영
a = 5

if a % 2 == 0:
    print('짝수입니다!')
else:
    print('홀수입니다!')
executed in 14ms, finished 02:02:09 2021-11-06
```

홀수입니다!

```
In [82]: # 예시 출력 확인
a = 4

if a % 2 == 0:
    print('짝수입니다!')
else:
    print('홀수입니다!')
executed in 13ms, finished 02:02:09 2021-11-06
```

짝수입니다!

```
In [83]: # if ~ elif ~ else: 여러가지 조건을 종합하는 경우 반영
a = 10

if a >= 100:
    print('큰 양수')
elif a >= 10:
    print('적당한 양수')
elif a < 0:
    print('음수')
else:
    print('0에 가깝구만유!')
executed in 14ms, finished 02:02:09 2021-11-06
```

적당한 양수

```
In [84]: # 예시 출력 확인
a = 1000

if a >= 100:
    print('큰 양수')
elif a >= 10:
    print('적당한 양수')
elif a < 0:
    print('음수')
else:
    print('0에 가깝구만유!')
executed in 14ms, finished 02:02:09 2021-11-06
```

큰 양수

```
In [85]: # 예시 출력 확인
a = -1000

if a >= 100:
    print('큰 양수')
elif a >= 10:
    print('적당한 양수')
elif a < 0:
    print('음수')
else:
    print('0에 가깝구만유!')
executed in 14ms, finished 02:02:09 2021-11-06
```

음수

```
In [86]: # 예시 출력 확인
a = 5

if a >= 100:
    print('큰 양수')
elif a >= 10:
    print('적당한 양수')
elif a < 0:
    print('음수')
else:
    print('0에 가깝구만유!')
executed in 14ms, finished 02:02:09 2021-11-06
```

0에 가깝구만유!

```
In [87]: # 함수와 결합
def num_spec(a):
    if a >= 100:
        print('큰 양수')
    elif a >= 10:
        print('적당한 양수')
    elif a < 0:
        print('음수')
    else:
        print('0에 가깝구만유!')

y = num_spec(5)
y
executed in 14ms, finished 02:02:09 2021-11-06
```

0에 가깝구만유!

```
In [88]: # 1000 입력 후 결과 확인
num_spec(1000)
executed in 14ms, finished 02:02:09 2021-11-06
```

큰 양수

```
In [89]: # -1000 입력 후 결과 확인
num_spec(-1000)
executed in 14ms, finished 02:02:09 2021-11-06
```

음수

```
In [90]: # 5 입력 했을 때 확인
num_spec(5)
executed in 15ms, finished 02:02:09 2021-11-06
0에 가깝구만유!
```

3.8 반복문 결합

- 반복 횟수는 입력된 항목의 원소 개수만큼으로 보통 유한개

```
In [91]: # for문으로 숫자 출력
num_list = [1,2,3,5,7,10]

for num in num_list:
    print(num)
executed in 14ms, finished 02:02:09 2021-11-06

1
2
3
5
7
10
```

```
In [92]: # if문과의 결합
num_list = [1,2,3,5,7,10]

for num in num_list:
    if num >= 5:
        print(num)
executed in 14ms, finished 02:02:09 2021-11-06

5
7
10
```

```
In [93]: # if문과의 결합
num_list = [1,2,3,5,7,10]

for num in num_list:
    if num >= 5:
        print(num)
        print(num*2)
        print(num**3)
        print('\n')
executed in 14ms, finished 02:02:09 2021-11-06

5
10
125

7
14
343
```

10
20
1000

```
In [94]: # 값의 저장
blank = []
for num in num_list:
    if num >= 5:
        print(num)
        blank.append(num*2)

blank
executed in 14ms, finished 02:02:09 2021-11-06
```

5
7
10

Out[94]: [10, 14, 20]

```
In [95]: # 합수와 결합
def multiple_list(num_list, criteria):
    blank = []
    for num in num_list:
        if num >= criteria:
            blank.append(num*2)
    return blank

multiple_list([1,2,3,5,7,10], 5)
executed in 14ms, finished 02:02:09 2021-11-06
```

Out[95]: [10, 14, 20]

```
In [96]: # 예시 결과 확인
multiple_list([1,3,5,7,9,11], 3)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
Out[96]: [6, 10, 14, 18, 22]
```

```
In [97]: # 예시 결과 확인  
multiple_list([1,3,5,7,9,11], 10)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
Out[97]: [22]
```

```
In [98]: # while ~  
num = 1
```

```
while num < 5:  
    print(num)  
    num = num + 1
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
1  
2  
3  
4
```

```
In [99]: # for + if 를 결합하여 결과 복제  
for num in [1,2,3,4,5,6,7,8,9]:  
    if num < 5:  
        print(num)
```

```
executed in 15ms, finished 02:02:09 2021-11-06
```

```
1  
2  
3  
4
```

```
In [100]: num = 1
```

```
while num < 5:  
    print('before: ', num)  
    print('after: ', num*2)  
    print('\n')  
    num = num + 1
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
before: 1  
after: 2
```

```
before: 2  
after: 4
```

```
before: 3  
after: 6
```

```
before: 4  
after: 8
```

```
In [101]: # 함수와의 결합
```

```
def num2(num):  
    while num < 5:  
        print('before: ', num)  
        print('after: ', num*2)  
        print('\n')  
        num = num + 1
```

```
num2(1)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
before: 1  
after: 2
```

```
before: 2  
after: 4
```

```
before: 3  
after: 6
```

```
before: 4  
after: 8
```

```
In [102]: # 예시 결과 확인
```

```
num2(2)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
before: 2  
after: 4
```

```
before: 3  
after: 6
```

```
before: 4  
after: 8
```

```
In [103]: # while ~ break: 특정 상황에서 조건문을 빠져나오기 위해 break 명령을 사용  
num = 1
```

```
blank = []  
while True:  
    blank.append(num*2)  
    print(blank)  
    if len(blank) == 5:  
        break  
    num = num + 1
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
[2]  
[2, 4]  
[2, 4, 6]  
[2, 4, 6, 8]  
[2, 4, 6, 8, 10]
```

```
In [104]: # 함수와의 결합
```

```
def numseq2(num):  
    blank = []  
    while True:  
        blank.append(num*2)  
        print(blank)  
        if len(blank) == 5:  
            break  
        num = num + 1
```

```
numseq2(1)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
[2]  
[2, 4]  
[2, 4, 6]  
[2, 4, 6, 8]  
[2, 4, 6, 8, 10]
```

```
In [105]: # 예시 결과 확인
```

```
numseq2(3)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
[6]  
[6, 8]  
[6, 8, 10]  
[6, 8, 10, 12]  
[6, 8, 10, 12, 14]
```

```
In [106]: # 랜덤 값 생성 내장함수 사용하여 임의 값 출력
```

```
import random  
  
samples = list(range(1,46))  
print(samples)  
random.shuffle(samples)  
print(samples)
```

```
executed in 14ms, finished 02:02:09 2021-11-06
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45]  
[42, 41, 27, 21, 2, 3, 32, 39, 5, 35, 34, 40, 31, 15, 14, 25, 20, 17, 1, 7, 24, 10, 8, 4, 44, 26, 22, 38, 37, 28, 23, 18, 45, 36, 11, 29, 13, 19, 16, 30, 43, 9, 33, 12, 6]
```

```
In [107]: # 계산값을 random.shuffle에 바로 사용시 사용 불가
```

```
samples_test = random.shuffle(list(range(1,46)))  
print(samples_test)
```

```
executed in 13ms, finished 02:02:09 2021-11-06
```

```
None
```

```
In [108]: # 중복된 값이 가능한 로또 번호 6개 출력
```

```
samples = list(range(1,10))  
  
lotto = []  
while len(lotto) < 6:  
    random.shuffle(samples)  
    num_selected = samples[0]  
    lotto.append(num_selected)  
    print(num_selected)  
print(lotto)
```

```
executed in 13ms, finished 02:02:09 2021-11-06
```

```
9  
4  
1  
4  
9  
5  
[9, 4, 1, 4, 9, 5]
```

- **Question:** 위 로또번호 생성기에, 1) continue/break 문을 사용하여 중복되지 않는 번호 6개를 출력하는 함수명 "lotto"를 만들고, 2) 아래 코드 실행시 10이내의 정수 중 중복되지 않는 로또번호가 6개를 생성하시오

```
y = lotto(10)  
y
```

In [109]: # 로또 번수 작성

executed in 14ms, finished 02:02:09 2021-11-06

3.9 파이썬 내장함수 활용

- def를 사용해서 python언어로 이미 anaconda에 만들어진 함수

In [110]: # 합계
samples = [1,2,3,4,5,6,7,8,9,10]
sum(samples)

executed in 14ms, finished 02:02:09 2021-11-06

Out[110]: 55

In [111]: # 합계 내장함수와 동일한 결과가 나오는 반복문 생성
samples = list(range(1,11))
print(samples)

```
num_sum = 0  
for i in samples:  
    num_sum = num_sum + i  
print(num_sum)
```

executed in 14ms, finished 02:02:09 2021-11-06

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
55

In [112]: # 최대값
max(samples)

executed in 14ms, finished 02:02:09 2021-11-06

Out[112]: 10

In [113]: # 최소값
min(samples)

executed in 14ms, finished 02:02:09 2021-11-06

Out[113]: 1

In [114]: # 길이
len(samples)

executed in 14ms, finished 02:02:09 2021-11-06

Out[114]: 10

In [115]: samples = list(range(10,20))

```
for num in samples:  
    print(num)
```

executed in 14ms, finished 02:02:09 2021-11-06

10
11
12
13
14
15
16
17
18
19

In [116]: # 개별 순서와 값을 반환하는 함수 작성

```
samples = list(range(10,20))  
  
for i, num in enumerate(samples):  
    print(i, num)
```

executed in 14ms, finished 02:02:09 2021-11-06

0 10
1 11
2 12
3 13
4 14
5 15
6 16
7 17
8 18
9 19

In [117]: # 특정 범위의 배열을 만드는 함수: range(시작, 끝)

range(0,10)

executed in 14ms, finished 02:02:09 2021-11-06

Out[117]: range(0, 10)

In [118]: # range 내장함수 사용 값 출력
for i in range(0,10):

```

print(i)
executed in 14ms, finished 02:02:09 2021-11-06
0
1
2
3
4
5
6
7
8
9

In [119]: # 정수형으로 변환
int(3.14)
executed in 14ms, finished 02:02:09 2021-11-06
Out[119]: 3

In [120]: # 문자열로 변환
str(3.14)
executed in 14ms, finished 02:02:09 2021-11-06
Out[120]: '3.14'

In [121]: # 반올림
round(3.14)
executed in 14ms, finished 02:02:09 2021-11-06
Out[121]: 3

In [122]: # 값들의 순서를 바꿈
samples = list(range(1,10))
print(samples)
reverse_samples = list(reversed(samples))
print(reverse_samples)
executed in 14ms, finished 02:02:09 2021-11-06
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1]

In [123]: # 오름차순
sorted(reverse_samples)
executed in 14ms, finished 02:02:09 2021-11-06
Out[123]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [124]: # 값들의 짝을 이루게하여 반복문에 많이 사용됨
# 짝을 이루는 값들의 길이는 서로 같아야 함
chars = ['a', 'b', 'c']
nums = [1, 2, 3]
zip(chars, nums)
executed in 14ms, finished 02:02:09 2021-11-06
Out[124]: <zip at 0x1c573c2e880>

In [125]: # 짤을 이용한 값들 출력
list(zip(chars, nums))
executed in 14ms, finished 02:02:09 2021-11-06
Out[125]: [('a', 1), ('b', 2), ('c', 3)]

In [126]: # 다양한 출력 가능
dict(zip(chars, nums))
executed in 14ms, finished 02:02:09 2021-11-06
Out[126]: {'a': 1, 'b': 2, 'c': 3}

```

3.10 내장함수처럼 활용하기

```

In [127]: # 5 나누기 3의 나머지 출력 함수 작성
def calculator(a, b):
    result = a % b
    return result

calculator(5,3)
executed in 14ms, finished 02:02:09 2021-11-06
Out[127]: 2

In [128]: # 예시 확인
calculator(2,2)
executed in 14ms, finished 02:02:09 2021-11-06
Out[128]: 0

In [129]: # 예시 확인
calculator(10,3)
executed in 14ms, finished 02:02:09 2021-11-06
Out[129]: 1

In [130]: # 위 함수 활용 3가지 케이스 나머지 모두 출력
nairs = [(5,3), (2,2), (10,3)]

```

```

for a, b in pairs:
    modulo = calculator(a,b)
    print(modulo)

```

executed in 14ms, finished 02:02:09 2021-11-06

2
0
1

In [131]: # 합수의 중첩

```

def mul(a, b):
    c = a * b
    return c

def add(a, b):
    c = a + b
    print(c)
    d = mul(a, b)
    print(d)

```

x = 10
y = 20
add(x, y)

executed in 14ms, finished 02:02:09 2021-11-06

30
200

In [132]: # 중첩 합수로 계산기 만들기

```

def calculator(a, b):
    result = a % b
    return result

def calculator_final(pairs):
    result = dict()
    for a, b in pairs:
        modulo = calculator(a,b)
        result[(a,b)] = modulo
    return result

pairs = [(5,3), (2,2), (10,3)]
calculator_final(pairs)

```

executed in 14ms, finished 02:02:10 2021-11-06

Out[132]: {(5, 3): 2, (2, 2): 0, (10, 3): 1}

- **Question:** 아래 2개의 변수값을 입력받아 1)몫을 구하는 함수, 2)나머지를 구하는 함수, 3) 그리고 1)과2)를 중첩함수로 한꺼번에 몫과 나머지를 출력하는 함수를 작성하고, 3) 함수를 이용해 몫과 나머지를 출력하시오

x = 10
y = 3

In [133]: # 합수 3개 작성

결과 출력

executed in 14ms, finished 02:02:10 2021-11-06

3.11 내장함수처럼 설명하기

- **독스트링(문서화 문자열, documentation strings, docstring):** 함수의 콜론(:) 다음줄에 """ """(큰따옴표 3개 묶음)으로 문자열을 입력하면 함수에 대한 설명 반영
- '(작은따옴표), " "(큰따옴표), """(작은따옴표 세 개)로 만들어도 되지만, 파이썬 코딩 스타일 가이드(PEP 8)에서는 """ """(큰따옴표 세 개)를 권장

```

def 함수이름(매개변수):
    """독스트링"""
    문장

```

```

def 함수이름(매개변수):
    ...
    여러 줄로 된
    독스트링
    ...
    문장

```

In [134]: # 위 계산기 중첩 합수에서 설명 추가 하기

```

def calculator(a, b):
    result = a % b
    return result

def calculator_final(pairs):
    """이 함수는 쌍의 값을 받아 각 쌍별 나머지를 출력하는 함수입니다."""
    result = dict()
    for a, b in pairs:
        modulo = calculator(a,b)
        result[(a,b)] = modulo
    return result

pairs = [(5,3), (2,2), (10,3)]
calculator_final(pairs)

```

```
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[134]: {(5, 3): 2, (2, 2): 0, (10, 3): 1}
```

```
In [135]: # 함수 설명 확인하기
```

```
calculator_final.__doc__
```

```
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[135]: '이 함수는 쌍의 값을 받아 각 쌍별 나머지를 출력하는 함수입니다.'
```

```
In [136]: # 내장함수의 설명도 확인 가능
```

```
print.__doc__
```

```
executed in 14ms, finished 02:02:10 2021-11-06
```

```
print(value, ..., sep=' ', end='\\n', file=sys.stdout, flush=False)
```

```
Prints the values to a stream, or to sys.stdout by default.
```

```
Optional keyword arguments:
```

```
file: a file-like object (stream); defaults to the current sys.stdout.
```

```
sep: string inserted between values, default a space.
```

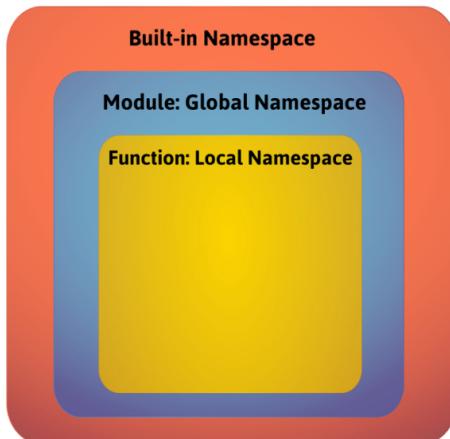
```
end: string appended after the last value, default a newline.
```

```
flush: whether to forcibly flush the stream.
```

3.12 스코핑 룰: Scoping Rule

- 변수의 생존 범위에 관련된 규칙
- 파이썬 함수는 별도의 변수들이 인식될수 있는 "이름공간" 존재

- 지역변수(Local Variable): 함수 내부에서만 인식되는 변수
- 전역변수(Global Variable): 함수 외부에서도 인식되는 변수
- 내장변수(Built-in Variable): 파이썬 자체에서 정의되고 인식되는 변수



```
In [137]: # def foo():
#     xx = 10      # foo의 지역 변수
#     print(xx)    # foo의 지역 변수 출력
```

```
# foo()
# print(xx)      # 예외. foo의 지역 변수는 출력할 수 없음
```

```
executed in 14ms, finished 02:02:10 2021-11-06
```

```
In [138]: # 예시 결과 확인
xx = 10      # 전역 변수
```

```
def foo():
    print(xx)    # 전역 변수 출력
```

```
foo()
print(xx)    # 전역 변수 출력
```

```
executed in 14ms, finished 02:02:10 2021-11-06
```

```
10
```

```
10
```

```
In [139]: # 예시 결과 확인
xx = 10      # 전역 변수
```

```
def foo():
    xx = 20      # x는 foo의 지역 변수
    print(xx)    # foo의 지역 변수 출력
```

```
foo()
print(xx)    # 전역 변수 출력
```

```
executed in 14ms, finished 02:02:10 2021-11-06
```

```
20
```

```
10
```

```
In [140]: # 예시 결과 확인
xx = 10      # 전역 변수
```

```
def foo():
    global xx    # 전역 변수 x를 사용하겠다고 설정
    xx = 20      # x는 전역 변수
```

```

    print(xx)      # 선언 변수 출력

foo()
print(xx)      # 전역 변수 출력

```

executed in 14ms, finished 02:02:10 2021-11-06

20

20

```

In [141]: ▾ # 예시 결과 확인
           # 전역 변수 x가 없는 상태
           def foo():
               global xx      # x를 전역 변수로 만들
               xx = 200       # x는 전역 변수
               print(xx)     # 전역 변수 출력

foo()
print(xx)      # 전역 변수 출력

```

executed in 14ms, finished 02:02:10 2021-11-06

200

200

```

In [142]: ▾ # 예시 결과 확인
           def A():
               x = 10          # A의 지역 변수 x
               ▾ def B():
                   x = 20      # x에 20 할당

                   B()
                   print(x)     # A의 지역 변수 x 출력

A()

```

executed in 14ms, finished 02:02:10 2021-11-06

10

```

In [143]: ▾ # 예시 결과 확인
           def A():
               x = 10          # A의 지역 변수 x
               ▾ def B():
                   nonlocal x  # 현재 함수의 바깥쪽에 있는 지역 변수 사용
                   x = 20      # A의 지역 변수 x에 20 할당

                   B()
                   print(x)     # A의 지역 변수 x 출력

A()

```

executed in 14ms, finished 02:02:10 2021-11-06

20

- nonlocal은 현재 함수의 바깥쪽에 있는 지역 변수를 찾을 때 가장 가까운 함수부터 먼저 찾음
- 함수의 단계와 무관하게 global 키워드를 사용하면 무조건 전역변수를 사용
- 전역변수는 가급적 사용하지 않고 지역변수를 유지하는게 오류를 줄이는 방법

- **Question:** 아래의 코드의 출력결과를 이해하여 왜 50과 400이 출력되었는지 주석으로 작성하시오

```

In [144]: ▾ def A():
           x = 10
           y = 100
           ▾ def B():
               x = 20
               ▾ def C():
                   nonlocal x
                   nonlocal y
                   x = x + 30
                   y = y + 300
                   print(x)
                   print(y)
                   C()
               B()
           A()

```

executed in 15ms, finished 02:02:10 2021-11-06

50

400

```

In [145]: ▾ # 결과 설명
           #
           #

```

executed in 14ms, finished 02:02:10 2021-11-06

- **Question:** 아래의 코드의 출력결과를 이해하여 왜 31이 출력되었는지 주석으로 작성하시오

```

In [146]: x = 1
           def A():
               x = 10
               ▾ def B():
                   x = 20
                   ▾ def C():
                       global x
                       x = x + 30

```

```

    C()
B()
A()

executed in 14ms, finished 02:02:10 2021-11-06
31

In [147]: # 결과 설명
#
#
executed in 15ms, finished 02:02:10 2021-11-06

```

3.13 Others

3.13.1 경제적인 코드작성: lambda

- `lambda` 입력: 출력
- `lambda`는 함수를 생성하는 예약어 `def`와 동일한 역할이나 한줄로 간결하게 작성 가능
- 주로 `def`를 사용해야 할 정도로 복잡하지 않거나 `def`를 사용할 수 없는 곳에 사용
- 식의 형태로 되어 있어 람다 표현식(Lambda Expression)이라고도 하며 다른 함수의 인수로 주로 사용
- 람다 표현식은 이름이 없는 익명함수라서 호출할 수 없고 사용후 메모리에서 사라지므로 경제적

`lambda` 매개변수1 입력, 매개변수2 입력, ... : 매개변수들 표현식 연산값 출력

```

In [148]: def add(a,b):
           return a+b

add(1,2)

executed in 14ms, finished 02:02:10 2021-11-06

```

Out[148]: 3

```

In [149]: # 람다 기능으로 위 함수를 변환
add_id = lambda a,b:a+b

add_id(1,2)

executed in 14ms, finished 02:02:10 2021-11-06

```

Out[149]: 3

```

In [150]: # 람다 함수 변수 저장 가능
result = []
add_func = lambda x:result.append(x+1)

add_func(1)
result

executed in 14ms, finished 02:02:10 2021-11-06

```

Out[150]: [2]

```

In [151]: # 람다 함수도 활용 가능
result = []
for x in [1,2,3,4,5]:
    add_func(x)
    print(result)

executed in 14ms, finished 02:02:10 2021-11-06

```

[2]
[2, 3]
[2, 3, 4]
[2, 3, 4, 5]
[2, 3, 4, 5, 6]

- 람다 표현식 자체를 호출할 수 있음

(`lambda` 매개변수 입력: 매개변수식 출력)(인수들)

```

In [152]: # 다양한 활용 가능
(lambda x: x*10)(1)    # f(x)
executed in 14ms, finished 02:02:10 2021-11-06

```

Out[152]: 11

```

In [153]: # 내부 변수 생성 불가
# (lambda x: y=10; x+y)(1)
executed in 14ms, finished 02:02:10 2021-11-06

```

```

In [154]: # 다양한 활용 가능
y = 10
(lambda x: x+y)(1)
executed in 14ms, finished 02:02:10 2021-11-06

```

Out[154]: 11

- 특히 함수의 인수부분을 간단하게 함수로 반영할 때 많이 사용

```

In [155]: # 예시 결과 확인
def plus_ten(x):

```

```
    return x + 10
list(map(plus_ten, [1,2,3]))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[155]: [11, 12, 13]
```

```
In [156]: # 예시 결과 확인
list(map(lambda x:x*10, [1,2,3]))
executed in 15ms, finished 02:02:10 2021-11-06
```

```
Out[156]: [11, 12, 13]
```

3.13.2 경제적인 중첩함수: map

- map(function, input)
- 함수를 입력으로 받을 수 있으며, 입력된 값이 함수에 적용된 값들이 출력

```
In [157]: # 함수와 값의 결합 가능
add_one = lambda x:x+1
data = list(range(0,10))

list(map(add_one, data))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[157]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [158]: # 함수와 2개 이상의 값 결합 가능
add = lambda x,y:x+y
a = [1,2,3,4,5]
b = [5,4,3,2,1]

list(map(add, a,b))
executed in 13ms, finished 02:02:10 2021-11-06
```

```
Out[158]: [6, 6, 6, 6, 6]
```

```
In [159]: # 다양한 활용 가능
a = [1,2,3,4,5]
b = [5,4,3,2,1]
list(map(lambda x,y:x+y, a,b))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[159]: [6, 6, 6, 6, 6]
```

```
In [160]: # 다양한 활용 가능
list(map(lambda x,y:x+y, [1,2,3,4,5], [5,4,3,2,1]))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[160]: [6, 6, 6, 6, 6]
```

- 조건문을 lambda 함수나 map 함수와 결합 가능
- 조건문 표현식에 물론() 사용하지 않음
- if를 사용했다면 반드시 else 사용해야 함
- elif 사용할 수 없고 if를 연속적으로 사용해야 함

lambda 매개변수들: 매개변수 식1(조건식 참일때) if 조건식 else 식2(조건식 거짓일때)
lambda 매개변수들: 매개변수 식1 if 조건식1 else 식2 if 조건식2 else 식3

```
In [161]: # 조건문 결합 가능
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

list(map(lambda x: str(x) if x % 3 == 0 else x, a))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[161]: [1, 2, '3', 4, 5, '6', 7, 8, '9', 10]
```

```
In [162]: # else가 포함된 원복한 형태여야 오류 미발생
# a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# list(map(lambda x: str(x) if x % 3 == 0, a))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
In [163]: # 조건문 여러개도 결합 가능
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

list(map(lambda x: str(x) if x % 3 == 0 else float(x) if x % 5 == 0 else x+10, a))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[163]: [11, 12, '3', 14, 5.0, '6', 17, 18, '9', 10.0]
```

- **Question:** 위와 동일한 결과를 출력하기 위해 def 명령을 사용하여 함수 "f"를 작성하여 아래 코드를 실행시 위의 결과와 동일하게 출력하시오

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list(map(f, a))
```

```
In [164]: # 함수 작성
```

```
# 코드 실행 및 결과 확인
# a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
# list(map(f, a))  
executed in 14ms, finished 02:02:10 2021-11-06
```

3.13.3 경제적인 조건반영 중첩함수: filter

- filter(function, input)
- map과 동일한 구조지만 조건식이 참인 입력 요소들만 출력

```
In [165]: # 값의 필터링 기능 확인  
even_num = lambda x: x%2==0  
data = list(range(0,10))  
  
list(filter(even_num, data))  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[165]: [0, 2, 4, 6, 8]
```

```
In [166]: # map 함수에서는 참/거짓의 로직만 출력됨  
list(map(even_num, data))  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[166]: [True, False, True, False, True, False, True, False, True, False]
```

```
In [167]: # 예시 결과 확인  
f = lambda x: x>5 and x<10  
data = [8, 3, 2, 10, 15, 7, 1, 9, 0, 11]  
  
list(filter(f, data))  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[167]: [8, 7, 9]
```

- map, filter와 lambda 중첩보다 리스트 표현식이 훨씬 가독성 좋고 속도도 빠름

```
In [168]: # 리스트 표현식으로 동일 결과 작성 가능  
a = [8, 3, 2, 10, 15, 7, 1, 9, 0, 11]  
  
[i for i in a if i>5 and i<10]  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[168]: [8, 7, 9]
```

- Question: 람다 표현식을 사용하여 아래 파일 명 중 jpg, .png인 이미지 파일만 리스트 형식으로 출력하시오

```
files = ['font', '1.png', '10.jpg', '11.gif', '2.jpg', '3.png', 'table.xlsx', 'spec.docx']
```

```
In [ ]:  
executed in 7ms, finished 15:30:31 2021-10-07
```

3.13.4 경제적인 반복값 생성함수: Comprehension

- 반복값을 생성하는 방법중 간편하고 유용한 기능으로 list 외 tuple/set/dict에서도 활용가능

```
In [169]: # List Type  
result = []  
for x in range(11):  
    result.append(x*2)  
result  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[169]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
In [170]: # 리스트 표현으로 위와 동일 결과 작성  
[x*2 for x in range(11)]  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[170]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
In [171]: result = []  
for i in range(2,9):  
    for j in range(i*2, 50, i):  
        result.append(j)  
print(len(result), result)  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
75 [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 10, 15, 20, 25, 30, 35, 40, 45, 12, 18, 24, 30, 36, 42, 48, 14, 21, 28, 35, 42, 49, 16, 24, 32, 40, 48]
```

```
In [172]: # 리스트 표현으로 위와 동일 결과 작성  
result = [] for i in range(2,9) for j in range(i*2, 50, i)  
print(len(result), result)  
executed in 14ms, finished 02:02:10 2021-11-06
```

```
75 [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 10, 15, 20, 25, 30, 35, 40, 45, 12, 18, 24, 30, 36, 42, 48, 14, 21, 28, 35, 42, 49, 16, 24, 32, 40, 48]
```

```
In [173]: # 집합으로 값을 정리하면 중복값들은 사라짐
```

```
result = []
for i in range(2,3) for j in range(i*2, 50, 1):
    print(len(result), result)
executed in 14ms, finished 02:02:10 2021-11-06
33 {4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 35, 36, 38, 39, 40, 42, 44, 45, 46, 48, 49}
```

```
In [174]: # Dict Type
subjects = ['math', 'history', 'english', 'computer engineering']
scores = [90, 80, 95, 100]

result = dict()
for key, value in zip(subjects, scores):
    result[key] = value
result
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[174]: {'math': 90, 'history': 80, 'english': 95, 'computer engineering': 100}
```

```
In [175]: # 리스트 표현으로 위와 동일 결과 작성
{key:value for key, value in zip(subjects, scores)}
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[175]: {'math': 90, 'history': 80, 'english': 95, 'computer engineering': 100}
```

```
In [176]: # zip 함수로 위와 동일 결과 작성
dict(zip(subjects, scores))
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[176]: {'math': 90, 'history': 80, 'english': 95, 'computer engineering': 100}
```

```
In [177]: # 리스트 표현으로 위와 동일 결과 작성
score_tuples = [('math', 90), ('history', 80), ('english', 95), ('computer engineering', 100)]
{t[0]: t[1] for t in score_tuples}
executed in 14ms, finished 02:02:10 2021-11-06
```

```
Out[177]: {'math': 90, 'history': 80, 'english': 95, 'computer engineering': 100}
```

```
In [178]: # dict 같은 key & value가 동시에 반복적 입력됨
score_tuples = [('math', 90), ('history', 80), ('english', 95), ('computer engineering', 100)]
for t in score_tuples:
    print(t)
executed in 14ms, finished 02:02:10 2021-11-06
```

```
('math', 90)
('history', 80)
('english', 95)
('computer engineering', 100)
```