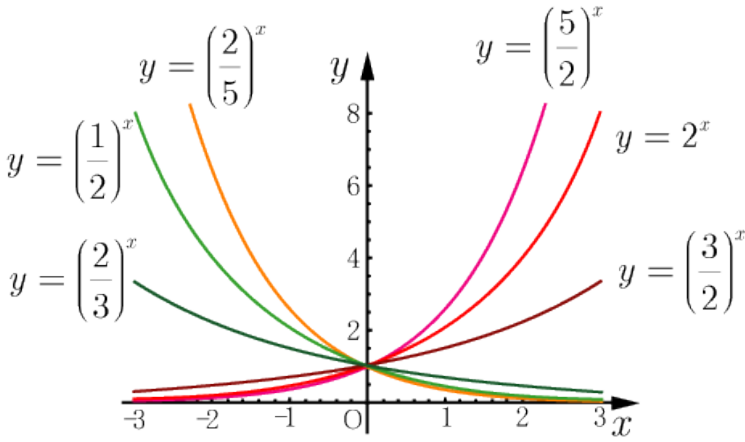
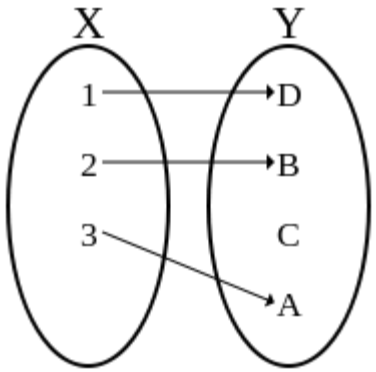
파이썬의 함수

함수란?

- 입력 → 결과
- 함수를 사용하는 이유
  - 똑같은 내용을 반복
  - 프로그램을 구조화





함수의 구조 (일반적)

```
def 함수명(입력인수):
    <수행할 문장1>
    <수행할 문장2>
    return 결과값
```

```
In [1]: def sum(a,b):
   ...:     return a+b
   ...:

In [2]: sum(3, 4)
Out[2]: 7
```

함수 예제 – 1

- 절대값 구하기

```
def absolute(n):
    if n < 0:
        n = -n
    return n

def absolute(n):
    if n < 0: return -n
    else: return n
```

- 리스트 더하기

```
def sum_list(ls):
    sum = 0
    for e in ls:
        sum += e
    return sum
```

함수 예제 – 2

- N! 구하기

```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result

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

- 리스트의 최대값

```
def max_list(ls):
    max = ls[0]
    for e in ls[1:]:
        if e > max:
            max = e
    return max
```

함수 예제 – 3

- 피보나치 수열 구하기
  - fib(n) = fib(n-2) + fib(n-1)
  - fib(0) = 0, fib(1) = 1

```
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-2)+fib(n-1)
```

- 리스트의 평균값

```
def avg_list(ls):
    avg = 0.0
    for e in ls:
        avg += e
    avg = avg / len(ls)
    return avg
```

함수 예제 – 4

- 홀수만 남기기

```
def cut_list(ls):
    result = []
    for e in ls:
        if e % 2 == 1:
            result.append(e)
    return result
```

- 5보다 큰 수만 남기기

```
def cut_list(ls):
    result = []
    for e in ls:
        if e > 5:
            result.append(e)
    return result
```

함수 예제 – 5

- x^n 구하기

```
def power(x, n):
    result = 1
    for i in range(n):
        result = result * x
    return result

def power(x, n):
    if n <= 0: return 1
    else: return x * power(x, n-1)
```

- 1~n의 합

```
def sum(n):
    result = 0
    for i in range(1, n+1):
        result += i
    return result

def sum(n):
    if n <= 0: return 0
    else: return n + sum(n-1)
```

함수 예제 – 6

- 구글 입사문제 중에서
  - 1부터 10,000까지 8이라는 숫자가 총 몇번 나오는가?
  - 1부터 n까지 x라는 숫자가 총 몇번 나오는가?

```
def count(n, x):
    result = 0
    for i in range(1, n+1):
        m = i
        while m > 0:
            if m % 10 == x:
                result += 1
            m = m / 10
    return result
```

함수 예제 – 7

- 다음 입사문제 중에서
  - 1차원의 점들이 주어졌을 때, 그 중 가장 거리가 짧은 것의 쌍을 출력하는 함수를 작성하시오. (단 점들의 배열은 모두 정렬되어 있다고 가정한다.)
  - 예를 들어 S=[1, 3, 4, 8, 13, 17, 20] 이 주어졌다면, 결과값은 (3, 4)가 될 것이다.

```
def min_dist(ls):
    min = ls[1] – ls[0]
    pair = (ls[0], ls[1])
    len_ls = len(ls)
    i = 2
    while i < len_ls:
        dist = ls[i] – ls[i-1]
        if dist < min_dist:
            min_dist = dist
            pair = (ls[i-1], ls[i])
        i += 1
    return pair
```

함수 예제 – 8

- 아마존 면접문제 중에서
  - 다음과 같은 형태의 배열을
  - [a1,a2,a3...,an,b1,b2...bn]
  - 다음과 같은 형태로 바꾸시오
  - [a1,b1,a2,b2.....an,bn]

```
def change_list(ls):
    ls_a = ls[:len(ls)/2]
    ls_b = ls[len(ls)/2:]
    result = []
    for i in range(len(ls)/2):
        result.append(ls_a[i])
        result.append(ls_b[i])
    return result
```

입력 값이 없는 함수

```
def 함수명():
    <수행할 문장1>
    <수행할 문장2>
    return 결과값
```

```
In [3]: def say():
   ...:     return 'hello'
   ...:

In [4]: a = say()

In [5]: print a
hello
```

리턴 값이 없는 함수

```
def 함수명(입력인수):
    <수행할 문장1>
    <수행할 문장2>
```

```
In [3]: def say():
   ...:     print 'hello'
   ...:

In [4]: a = say()
Hello

In [9]: print a
None
```

입력 값이 몇 개인지 모를 경우 – 1

```
def 함수명(*입력인수):
    <수행할 문장1>
    <수행할 문장2>
```

```
In [10]: def sum_many(*args):
   ....:     sum = 0
   ....:     for i in args:
   ....:         sum += i
   ....:     return sum
   ....:

In [11]: sum_many(1,2,3,4,5)
Out[11]: 15
```

입력 값이 몇 개인지 모를 경우 – 2

```
def 함수명(입력인수, *입력인수):
        <수행할 문장1>
        <수행할 문장2>
```

```
In [12]: def sum_mul(choice, *args):
   ....:     if choice == 'mul':
   ....:         result = 1
   ....:         for i in args:
   ....:             result *= i
   ....:     else:
   ....:         result = 0
   ....:         for i in args:
   ....:             result += i
   ....:     return result
   ....:

In [13]: sum_mul('mul', 1, 2, 3, 4, 5)
Out[13]: 120
```

이름으로 구분하는 입력 인수

- 일반적으로 여러 개의 입력 인수(parameters)는 위치로 구분함
- 파이썬에서는 이름으로 입력 인수를 구분할 수도 있음
- 단, 위치로 구분하는 입력 인수를 먼저 써주고, 그 다음에 이름으로 구분하는 입력 인수를 사용함

```
In [59]: def foo(a, b, c):
   ....:     print a, b, c
   ....:

In [60]: foo(1, 2, 3)
1 2 3

In [61]: foo(c=3, b=2, a=1)
1 2 3

In [62]: foo(1, c=3, b=2)
1 2 3
```

리턴 값이 둘 이상일 경우

```
def 함수명(입력인수):
        <수행할 문장1>
        <수행할 문장2>
        return (결과1, 결과2)
```

```
In [14]: def sum_and_mul(a, b):
   ....:     return (a+b, a*b)
   ....:

In [15]: sum_and_mul(3, 4)
Out[15]: (7, 12)
```

Return의 또 다른 쓰임새

- 함수를 빠져나가기를 원할 때

```
In [16]: def say_nick(nick):
   ....:     if nick == 'fool':
   ....:         return
   ....:     print nick
   ....:

In [17]: say_nick('genius')
genius

In [18]: say_nick('fool')

In [19]:
```

입력 값에 초기값 설정

```
def 함수명(입력인수=초기값):
        <수행할 문장1>
        <수행할 문장2>
```

```
In [19]: def say_name_nick(name, nick='genius'):
   ....:     print name, nick
   ....:

In [20]: say_name_nick('Dave')
Dave genius

In [21]: say_name_nick('Dave', 'fool')
Dave fool
```

함수의 지역 변수

지역변수: 함수 내에서 선언된 변수, 혹은 함수의 입력 인수

```
In [22]: def var_test(a):
   ....:     a += 1
   ....:     b = a
   ....:     return a
   ....:
In [23]: a = 1
In [24]: var_test(a)
Out[24]: 2
In [25]: a
Out[25]: 1
In [26]: b
NameError: name 'b' is not defined
```

```
In [27]: a = 1

In [28]: a = var_test(a)

In [29]: a
Out[29]: 2
```

전역 변수

전역변수: 어떤 영역 내에서도 접근할 수 있는 변수
global: 전역변수를 함수 내에서 사용

```
In [30]: a = 1

In [31]: def var_test():
   ....:     global a
   ....:     a += 1
   ....:

In [32]: var_test()

In [33]: a
Out[33]: 2
```

```
In [34]: a = 1

In [35]: def var_test():
   ....:     a += 1
   ....:

In [36]: var_test()
UnboundLocalError: local variable 'a' referenced before assignment
```

함수의 입력인수 전달방식 – 1

- 입력 인수의 타입이 **immutable** 인 경우
  - 정수, 실수, 문자열, 튜플 등
  - Call-by-value와 동일 → 함수 내에서 바뀐 결과가 원래의 입력 변수에 반영 안됨

```
In [39]: t = (1, 2, 3)

In [40]: def test(a):
   ....:     a += (4, 5, 6)
   ....:

In [41]: test(t)

In [42]: t
Out[42]: (1, 2, 3)
```

```
In [46]: t = 1

In [47]: def test(a):
   ....:     a += 1
   ....:

In [48]: test(t)

In [49]: t
Out[49]: 1
```

함수의 입력인수 전달방식 – 2

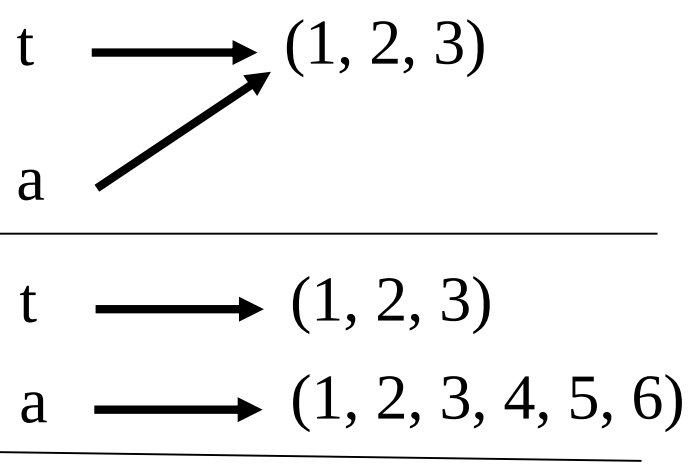
- 입력 인수의 타입이 **mutable** 인 경우
  - 리스트, 딕셔너리
  - Call-by-reference와 동일 → 함수 내에서 바뀐 결과가 원래의 입력 변수에 반영 됨

```
In [51]: t = [1, 2, 3]

In [52]: def test(a):
   ....:     a += [4, 5, 6]
   ....:

In [53]: test(t)

In [54]: t
Out[54]: [1, 2, 3, 4, 5, 6]
```

```
In [55]: t = {1:'a'}

In [56]: def test(a):
   ....:     a[2] = 'b'
   ....:

In [57]: test(t)

In [58]: t
Out[58]: {1: 'a', 2: 'b'}
```

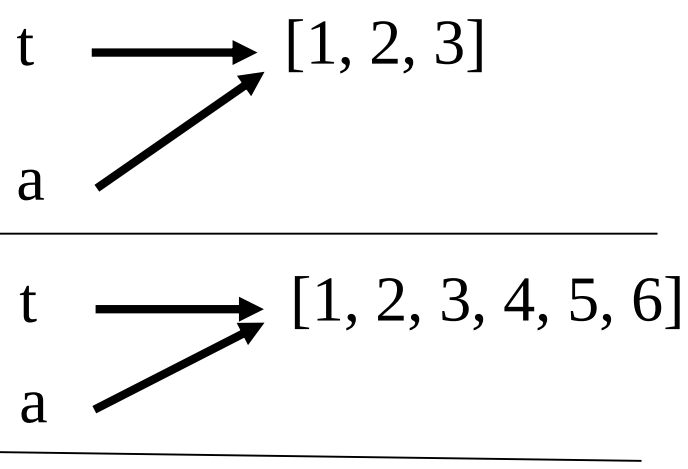
Tuples are *immutable*

```
In [63]: def test(a):
   ....:     a += (4, 5, 6)
   ....:

In [64]: t = (1, 2, 3)

In [65]: test(t)  # a = t
         # a += (4,5,6)
In [66]: t
Out[66]: (1, 2, 3)
```



Lists are *mutable*

```
In [63]: def test(a):
   ....:     a += [4, 5, 6]
   ....:

In [64]: t = [1, 2, 3]

In [65]: test(t)  # a = t
         # a += [4,5,6]
In [66]: t
Out[66]: [1, 2, 3, 4, 5, 6]
```



함수는 객체(object)임 - 1

- 함수는 다른 변수와 똑같이 취급될 수 있음
  - **함수는 변수에 배정(assign)할 수 있음**
  - 함수는 인수로 다른 함수에 전달할 수 있음
  - 다른 함수로부터 함수는 리턴 될 수 있음
- def 문은 함수 몸체를 변수(함수이름)에 할당하는 것임

```
In [71]: def foo(a, b):
   ....:     print a, b
   ....:

In [72]: x = foo

In [73]: x(1, 2)
1 2

In [74]: foo
Out[74]: <function __main__.foo>

In [75]: x
Out[75]: <function __main__.foo>
```

함수는 객체(object)임 - 2

- 함수는 다른 변수와 똑같이 취급될 수 있음
  - 함수는 변수에 배정(assign)할 수 있음
  - **함수는 인수로 다른 함수에 전달할 수 있음**
  - 다른 함수로부터 함수는 리턴 될 수 있음
- def 문은 함수 몸체를 변수(함수이름)에 할당하는 것임

```
In [76]: def foo(f, a):
   ....:     return f(a)
   ....:

In [77]: def square(x):
   ....:     return x*x
   ....:

In [78]: foo(square, 3)
Out[78]: 9
```

함수는 객체(object)임 - 3

- 함수는 다른 변수와 똑같이 취급될 수 있음
  - 함수는 변수에 배정(assign)할 수 있음
  - 함수는 인수로 다른 함수에 전달할 수 있음
  - **다른 함수로부터 함수는 리턴 될 수 있음**
- def 문은 함수 몸체를 변수(함수이름)에 할당하는 것임

```
In [79]: def foo(x):
   ....:     def bar(y):
   ....:         return x+y
   ....:     return bar
   ....:

In [80]: f = foo(3)

In [81]: f
Out[81]: <function __main__.bar>

In [82]: f(2)
Out[82]: 5
```

이름 없는 함수 (Anonymous Function)

- 람다 표현식 (lambda expression) 은 함수 객체를 리턴
- 람다 표현식의 몸체(body)는 단순한 수식만 가능

```
>>> f = lambda x,y : x + y
>>> f(2,3)
5
>>> lst = ['one', lambda x : x * x, 3]
>>> lst[1](4)
16
```

Higher-Order Functions - 1

- **map(func, seq)**
  - for all i, applies func(seq[i]) and returns the corresponding sequence of the calculated results.

```
In [84]: def double(x):
   ....:     return 2*x
   ....:

In [85]: lst = range(10)

In [86]: lst
Out[86]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [87]: map(double, lst)
Out[87]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Higher-Order Functions – 2

- **filter(bool_func, seq)**
  - returns a sequence containing all those items in seq for which bool_func is True.

```
In [88]: def even(x):
   ....:     return (x%2 == 0)
   ....:

In [89]: lst = range(10)

In [90]: lst
Out[90]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [91]: filter(even, lst)
Out[91]: [0, 2, 4, 6, 8]
```

Higher-Order Functions – 3

- **reduce(func, seq)**
  - applies func to the items of seq, from left to right, two-at-time, to reduce the seq to a single value.

```
In [91]: def plus(x, y):
   ....:     return (x + y)
   ....:

In [92]: lst = range(10)

In [93]: lst
Out[93]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [94]: reduce(plus, lst)
Out[94]: 45
```