

Container

- list
- tuple
- range
- set

- 튜플(tuple)
 - 리스트와 동일하지만, 튜플은 불변객체이라는 차이점이 있음

```
>>> a=(1,2,3,4,5)
>>> a
(1,2,3,4,5)
>>> a[0]=0 #불변객체이므로 재할당이 되지 않는다.

>>> b=tuple(range(1,10))
>>> b
(1,2,3,4,5,6,7,8,9)
>>> b.count(7) # 아이템 7이 몇개인지 개수를 세어준다.
1
>>> b.index(7) # 아이템 7의 인덱스를 알려준다.
```

- 다양한 tuple 값 대입

```
>>> x=1,  
>>> type(x)  
tuple  
>>> x=1,2  
>>> type(x)  
tuple  
>>> x,y=1,2  
>>> type(x)  
??  
>>> x=1, (2,3)  
>>> x=(1,)
```

```
>>> x=(1, (2,3))  
>>> (x,y) =(1,2)  
>>> type(x)  
??  
>>> type((x,y))  
??
```

- 딕셔너리
 - key와 value 쌍으로 이루어진 자료구조

```
>>> a={'a' : 1, 'b': 2}  
>>> a  
{ 'a' : 1, 'b': 2 }
```

- 키-값 쌍(key-value pair)으로 저장
- 딕셔너리는 중괄호{ }로 생성
- 딕셔너리 생성

```
>>> phone_book={'신사임당': '010-1234-5678', '홍길동': '010-4532-4532'}  
>>> phone_book  
{'신사임당': '010-1234-5678', '홍길동': '010-4532-4532'}
```

- 딕셔너리 키-값 추가

```
>>> phone_book['이순신'] = '010-0987-6543'  
>>> phone_book  
{'신사임당': '010-1234-5678', '홍길동': '010-4532-4532', '이순신': '010-0987-6543'}
```

- 딕셔너리 키로 검색

```
>>> phone_book={'신사임당': '010-1234-5678', '홍길동': '010-4532-4532', '이순신': '010-0987-6543'}
>>> phone_book['이순신']
010-0987-6543
>>> '이순신' in phone_book
True
>>> '강감찬' in phone_book
False
>>> phone_book.get('이순신')
'010-0987-6543'
```

- 딕셔너리 keys 출력

```
>>> phone_book.keys()  
dict_keys(['신사임당', '홍길동', '이순신'])
```

- 딕셔너리 values 출력

```
>>> phone_book.values()  
dict_values(['010-1234-5678', '010-4532-4532', '010-0987-6543'])
```


- 딕셔너리 항목 삭제

```
>>> del phone_book['홍길동']  
>>> phone_book  
{'신사임당': '010-1234-5678', '이순신': '010-0987-6543'}  
>>> phone_book.clear() #모든 항목 삭제  
>>> phone_book  
{}
```

```
>>> phone_book={'신사임당': '010-1234-5678', '홍길동': '010-4532-4532', '이순신': '010-0987-6543'}
```

```
>>> for i in phone_book:  
    print(i)
```

신사임당

홍길동

이순신

```
>>> for i in phone_book:  
    print(i)  
    else:  
        print('end')
```

신사임당

홍길동

이순신

end

```
>>> for i in phone_book.keys():  
    print(i)  
신사임당  
홍길동  
이순신  
>>> for i in phone_book.values():  
    print(i)  
010-1234-5678  
010-4532-4532  
010-0987-6543  
>>> for i in phone_book.items():  
    print(i)  
( '신사임당', '010-1234-5678' )  
( '홍길동', '010-4532-4532' )  
( '이순신', '010-0987-6543' )
```

```
>>> for i, j in phone_book.items():  
    print(i, j)  
신사임당 010-1234-5678  
홍길동 010-4532-4532  
이순신 010-0987-6543
```

- 편의점 아이템

- items = {"커피": 7, "펜": 3, "종이컵": 2, "우유": 1, "콜라": 4, "책": 5}

```
-----메뉴-----
1. 재고확인  2. 판매  3. 종료
-----

1
재고 확인을 위한 물건명을 입력하세요. 커피
7
-----메뉴-----
1. 재고확인  2. 판매  3. 종료
-----

2
판매한 물건명을 입력하세요. 콜라
콜라의 판매 개수를 입력하세요. 2
-----메뉴-----
1. 재고확인  2. 판매  3. 종료
-----

1
재고 확인을 위한 물건명을 입력하세요. 콜라
2
-----메뉴-----
1. 재고확인  2. 판매  3. 종료
-----

3
>>>
```

```
-----메뉴-----  
1. 단어 추가  2. 사전 보기  3. 종료  
-----  
1  
한글 단어를 입력하세요: 사과  
영어 단어를 입력하세요: apple  
1  
한글 단어를 입력하세요: 기차  
영어 단어를 입력하세요: train  
2  
찾고자 하는 단어를 입력하세요:기차  
train  
3  
>>>
```

- 집합 (set)
 - set은 순서가 없으며, 수정이 가능하고, 값이 중복되지 않음

```
>>> a={1,2,3,4,5}
>>> a
{1,2,3,4,5}
>>> b=set([1,1,4,2,2,6,6,6,6,7,8,10,4,2,8,6,3])
>>> b
{1,2,3,4,6,7,8,10}
```

```
>>> a={1,2,3,4,5}
```

```
>>> a
```

```
{1,2,3,4,5}
```

```
>>> a.add(6) #아이템을 추가한다.
```

```
>>> a.remove(6) #아이템을 제거한다.
```

```
>>> a.clear() #아이템을 모두 제거한다.
```

```
>>> a.pop() #가장 먼저 있는 아이템을 제거한다.
```

- set은 집합이므로, 합집합, 차집합, 교집합, 대칭차집합을 표현할 수 있음

```
>>> a={1,4,7}
>>> b={1,3,6,8,10}
>>> a | b   #합집합
{1, 3, 4, 6, 7, 8, 10}
>>> a-b    #차집합
{4, 7}
>>> a&b    #교집합
{1}
>>> a^b    #대칭차집합   #(합집합)-(교집합)
{3, 4, 6, 7, 8, 10}
```


Operation	Result
s.add(item)	Add item to s (mutates s)
s.clear()	Remove elements from s (mutates s)
s.copy()	Shallow copy
s.difference(s2)	Return set with elements from s and not s2
s.difference_update(s2)	Remove s2 items from s (mutates s)
s.discard(item)	Remove item from s (mutates s). No error on missing item
s.intersection(s2)	Return set with elements from both sets
s.intersection_update(s2)	Update s with members of s2 (mutates s)
s.isdisjoint(s2)	True if there is no intersection of these two sets
s.issubset(s2)	True if all elements of s are in s2
s.issuperset(s2)	True if all elements of s2 are in s
s.pop()	Remove arbitrary item from s (mutates s). KeyError on missing item
s.remove(item)	Remove item from s (mutates s). KeyError on missing item
s.symmetric_difference(s2)	Return set with elements only in one of the sets
s.symmetric_difference_update(s2)	Update s with elements only in one of the sets (mutates s)
s.union(s2)	Return all elements of both sets
s.update(s2)	Update s with all elements of both sets (mutates s)

- Container에서의 값 할당의 여러 가지 예
 - *는 나머지 값의 의미로 사용한다.

```
>>> a, *b, c = 1,2,3,4,5
```

```
>>> a
```

```
1
```

```
>>> b
```

```
[2,3,4]
```

```
>>> c
```

```
5
```

```
>>> a, b= {1,2}
```

```
>>> a
```

```
1
```

```
>>> {3,1,2,3,1}
```

```
{1,2,3}
```

```
>>> *a, b, c = "안녕하세요"
```

```
>>> a
```

```
['안','녕','하']
```

```
>>> b
```

```
['세']
```

```
>>> c
```

```
['요']
```

```
>>> a, b= {1,2}
```

```
>>> a
```

```
1
```

```
>>> {3,1,2,3,1}
```

```
{1,2,3}
```

- 해당 요소가 있으면 True, 없으면 False를 반환함

```
>>> a='Hello'
>>> 'h' in a
True
>>> 'b' in a
False
>>> b={1,3,6,8,10}
>>> 1 in b
True
>>> c = [1,2,3,4,5]
>>> 3 in c
True
>>> d=[[4,6], '1', '2']
>>> [4] in d
False
```

```
>>> [4,6] in d
True
```

```
>>> not True
False
>>> not False
True
>>> True not 
SyntaxError: invalid syntax
```

- 0이외의 값은 모두 True이다.

```
>>> if 2:  
    print('True')  
True  
>>> if [2]:  
    print('True')  
True
```

```
>>> if [ ]:  
    print('True')  
else:  
    print('False')  
False
```

```
>>> a=3  
  
>>> if 1 < a < 5:  
    print('True')  
else:  
    print('False')  
True
```

```
>>> a=-3  
>>> b=3 if a>0 else 8  
>>> a  
-3  
>>> b  
8
```

- 비교연산자 is는 값 비교가 아닌 레퍼런스 비교를 수행함

```
>>> 1 is 3
False
>>> a = '1'
>>> b = 1
>>> a is b
False
>>> a = 10
>>> b = 10
>>> a is b
True
>>> id(a), id(b)
(1516360656, 1516360656)
```

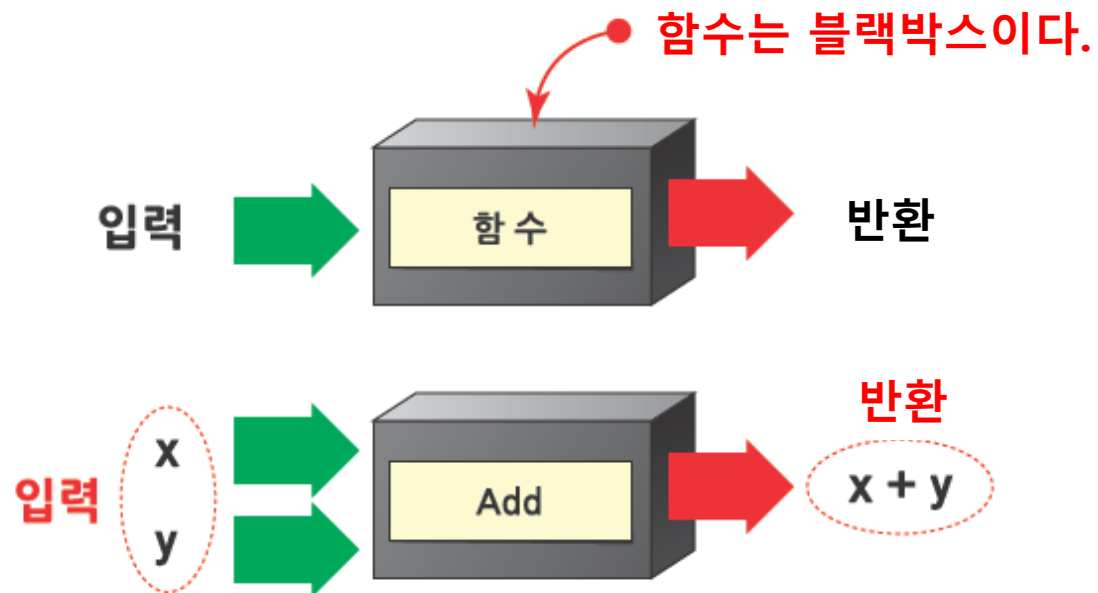
```
>>> a = -5
>>> b = -5
>>> a is b
True
>>> a = -6
>>> b = -6
>>> a is b
False
>>> a = 256
>>> b = 256
>>> a is b
True
>>> a = 257
>>> b = 257
>>> a is b
False
```

파이썬에서는 -5~256까지는 메모리에 캐시하여 동일한 주소를 참조하도록 하고 있다.

함수

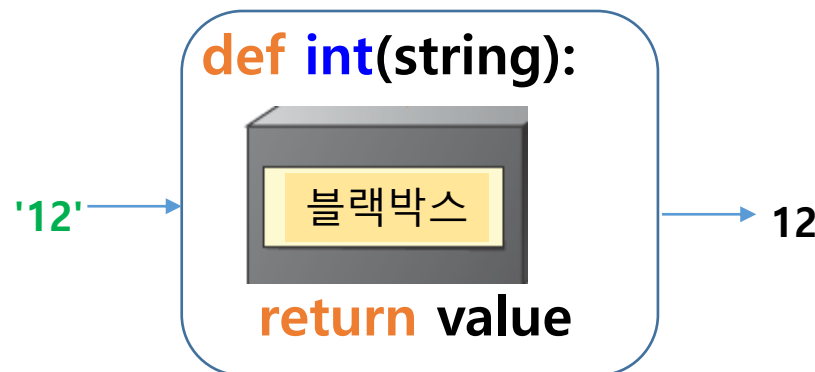
- 함수

- 프로그래밍에서 자주 사용되는 코드를 따로 만들어 두고 필요할 때마다 불러서(호출해서) 사용하는 기능



- 지금까지 프로그래밍에서 사용한 함수에는 어떤 것들이 있었을까?
 - 입력함수
 - `input('Input the number :')`
 - `input()`
 - 출력함수
 - `print('Total number' + total)`
 - `print()`
 - 형변환함수
 - `int('12')`
 - `str(31)`
 - `float(15)`
 - 랜덤함수
 - `random.randint(1,20)`

내장함수:
프로그래밍에서 자주 사용하는 기능을 미리 만들어놓은 함수



- 필요한 기능이 내장함수에 없으면 어떻게 하지?
 - 예:
 - "섭씨온도를 입력하면 화씨온도를 계산해주는 기능이 있으면 좋겠어"
 - "dino를 클릭할 때마다 공룡그림을 출력해주는 기능이 필요해"

```
def process_c_f(c):  
    f=9.0/5.0*c+32.0  
    print (f)
```

```
def dino( ):  
    print (''  
    <img alt="A green dashed outline of a dinosaur, specifically a T-Rex, facing left." data-bbox="175 695 365 910"/>  
    ''')
```

사용자정의함수:
프로그래머가 필요에 의해서 만든
함수

- 함수를 사용하지 않은 예

```
num = 1
total = 0
while num <= 10 :
    total = total + num
    num = num + 1
print ('1부터 10까지의 합은' + str(total) + '입니다.')
```

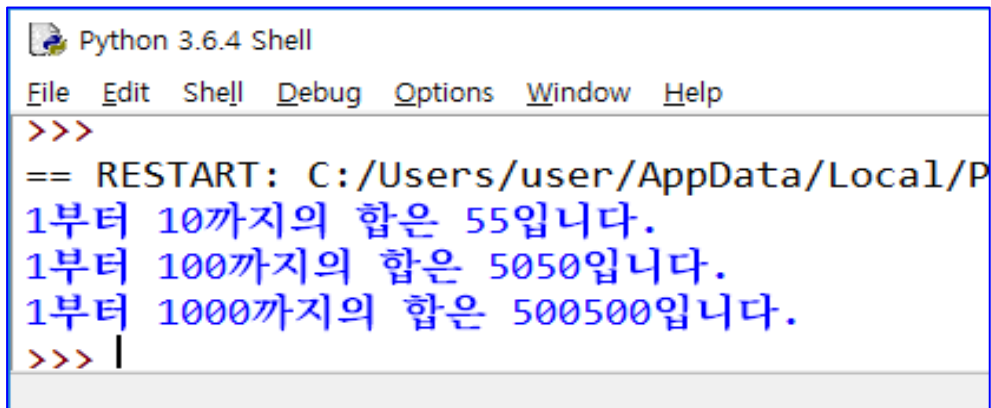
```
num = 1
total = 0
while num <= 100 :
    total = total + num
    num = num + 1
print ('1부터 100까지의 합은' + str(total) + '입니다.')
```

```
num = 1
total = 0
while num <= 1000 :
    total = total + num
    num = num + 1
print ('1부터 1000까지의 합은' + str(total) + '입니다.')
```

- 함수를 사용한 예

```
def summation(y) :  
    num = 1  
    total = 0  
    while num <= y :  
        total = total + num  
        num = num + 1  
    print ('1부터 ' + str(y) + '까지의 합은 ' + str(total) + '입니다.')
```

```
summation(10)  
summation(100)  
summation(1000)
```



Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

```
>>>  
== RESTART: C:/Users/user/AppData/Local/P  
1부터 10까지의 합은 55입니다.  
1부터 100까지의 합은 5050입니다.  
1부터 1000까지의 합은 500500입니다.  
>>> |
```

- 코드의 간결성
 - 코드가 중복되지 않고 간결해진다.
- 코드의 재사용성
 - 한 번 작성해둔 코드를 여러 번 사용하므로 코드를 재사용할 수 있다.
- 프로그램의 모듈화
 - 기능별로 함수를 작성하므로 프로그램 모듈화가 증대된다.
- 코드 수정의 용이성
 - 프로그램 기능을 함수로 나누어 묶기 때문에 코드 수정이 쉽다.

- 함수는 어떻게 만들지?
 - 함수 정의 파트와 함수 호출 파트로 이루어진다.
 - 함수 정의 파트가 함수 호출 파트보다 먼저 작성되어 있어야 한다.
 - 함수 호출 파트(외부) 에서 함수 정의 파트(내부)로 값을 전달할 때 전달인자를 사용한다.
 - 함수 정의 파트에서는 값을 받을 때 매개변수로 받는다.
 - 전달인자나 매개변수를 파라미터라고도 부름
 - 함수 정의 파트에서 함수 호출 파트로 값을 내 보낼 때에는 반환(return)을 사용한다.
 - 전달인자, 매개변수, 반환이 없는 함수도 있다.
 - 함수안에서 사용하는 지역변수와 함수 밖에서 사용하는 전역변수가 있다.

- 함수 정의와 호출 방법
 - 함수 정의 부분과 함수 호출 부분으로 이루어진다.
 - 함수 정의 부분이 함수 호출 부분보다 먼저 작성되어 있어야 한다.

함수 정의 방법

keyword 매개변수

```
def bird ( ):
```

↑ ↑
함수명 colon

함수 호출 방법

함수명() 을 적는다.

함수 정의와 호출의 예

```
def bird():  
    print (''  
           {.,.}  
           |_)_  
           _"_"  
           ... )  
bird()
```

들여쓰기

함수 정의 (내부)

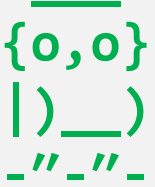

함수 호출 (외부)


- 함수 정의 위치
 - 함수 정의 파트가 함수 호출 파트보다 먼저 작성되어 있어야 한다.

```
sayGoodBye ()  
  
def sayGoodBye () :  
    print 'Good bye! '
```

```
Traceback (most recent call last):  
  File "<pyshell#3>", line 2, in <module>  
    sayGoodBye ()  
NameError: name 'sayGoodBye' is not defined
```


- 실습해 보기

```
def bird1():  
    print (''  
          
        '' )  
def bird2():  
    print (''  
          
        '' )
```

```
def bird3():  
    print (''  
          
        '' )  
  
    count = 1  
    while count <= 10:  
        bird1()  
        bird2()  
        bird3()  
        count = count + 1
```

- 함수를 10번
호출해 보기

- Intro 출력하기
 - `print()`를 이용하여 출력

```
print ('You are in a land full of dragons.')
```

```
print ('In front of you, you see two caves. ')
```

```
print ('In one cave, the dragon is friendly, ')
```

```
print (' and will share his treasure with you. ')
```

```
print ('The other dragon is greedy and hungry, ')
```

```
print ('and will eat you on sight. ')
```

- 으سس~효과를 주기
 - 맨 윗줄에 **import time**을 추가
 - 시간 지연을 넣고자 하는 곳에 **time.sleep(1)**을 추가

```
import time
print ('You are in a land full of dragons.')
time.sleep(1)
print ('In front of you, you see two caves. ')
time.sleep(1)
print ('In one cave, the dragon is friendly, ')
time.sleep(1)
print (' and will share his treasure with you. ')
time.sleep(1)
print ('The other dragon is greedy and hungry, ')
time.sleep(1)
print ('and will eat you on sight. ')
```

- 동굴번호 입력받기
 - 동굴번호를 입력받아 저장할 변수 : cave

```
cave = input ('Which cave will you go into? (1 or 2) ')
```

- 입력받은 동굴번호가 1도 아니고, 2도 아니면 계속 입력해달라고 반복해야 함

- 소스코드 작성

```
cave = input ('Which cave will you go into? (1 or 2) ')
```

- 동굴번호 입력 후 메시지
 - 메시지를 출력하는 소스코드를 작성하세요.
 - 시간 지연을 넣고자 하는 곳에 **time.sleep(1)**을 추가하세요.

- 소스코드 작성

You approach the cave...
It is dark and spooky...
A large dragon jumps out in front of you!
He opens his jaws and...

- 랜덤값 발생

- ❖ 컴퓨터도 랜덤값 발생, 랜덤값을 저장할 변수이름 : friendlycave

- 소스코드 작성

```
#랜덤모듈 import
#랜덤값 발생
if friendlycave == int(cave):
    print ('Gives you his treasure!')
else :
    print ('Gobbles you down in one bite!')
```

- 게임 계속 여부 질문 넣기

```
playagain = input ('Do you want to play again?(yes or no)')
```

- 'yes' or 'y'이면 게임이 계속되도록 반복문을 넣어보세요.
 - 반복해야 할 문장은? 문장 전체

• 소스코드 작성

```
.....  
:   
print ('You are in a land full of dragons.')  
time.sleep(1)  
print ('In front of you, you see two caves. ' )  
.....  
playagain = input ('Do you want to play again?(yes or no)')
```