

# 문자열 분리와 결합

```
>>> items = 'zero one two three'.split()           # 빈칸을 기준으로 문자열 분리하기
>>> print (items)
['zero', 'one', 'two', 'three']
```

```
>>> example = 'python,jquery,javascript'           # ","를 기준으로 문자열 나누기
>>> example.split(",")
['python', 'jquery', 'javascript']
>>> a, b, c = example.split(",")                   # 리스트에 있는 각 값을 a, b, c 변수로 언패킹
>>> print(a, b, c)
python jquery javascript
>>> example = 'theteamlab.univ.edu'
>>> subdomain, domain, tld = example.split('.')     # "."을 기준으로 문자열 나누기 → 언패킹
>>> print(subdomain, domain, tld)
theteamlab univ edu
```

# 문자열 분리와 결합

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''.join(colors)
>>> result
'redbluegreenyellow'
```

```
>>> result = ' '.join(colors)           # 연결 시, 1칸을 띄고 연결
>>> result
'red blue green yellow'
>>> result = ', '.join(colors)          # 연결 시 ", "으로 연결
>>> result
'red, blue, green, yellow'
>>> result = '-'.join(colors)           # 연결 시 "-"으로 연결
>>> result
'red-blue-green-yellow'
```

# 문자열 관련 주요 함수와 메서드



함수명	기능
len()	문자열의 문자 개수를 반환 
upper()	대문자로 변환
lower()	소문자로 변환
title()	각 단어의 앞글자만 대문자로 변환
capitalize()	첫 문자를 대문자로 변환
count('찾을 문자열')	'찾을 문자열'이 몇 개 들어 있는지 개수 반환
find('찾을 문자열')	'찾을 문자열'이 왼쪽 끝부터 시작하여 몇 번째에 있는지 반환
rfind('찾을 문자열')	find() 함수와 반대로 '찾을 문자열'이 오른쪽 끝부터 시작하여 몇 번째에 있는지 반환
startswith('찾을 문자열')	'찾을 문자열'로 시작하는지 여부 반환
endswith('찾을 문자열')	'찾을 문자열'로 끝나는지 여부 반환

# 문자열 관련 주요 함수와 메서드



strip()	좌우 공백 삭제
rstrip()	오른쪽 공백 삭제
lstrip()	왼쪽 공백 삭제
split()	문자열을 공백이나 다른 문자로 나누어 리스트로 반환
isdigit()	문자열이 숫자인지 여부 반환
islower()	문자열이 소문자인지 여부 반환
isupper()	문자열이 대문자인지 여부 반환

## ■ 리스트 컴프리헨션 다루기

- 리스트 컴프리헨션(list comprehension)의 기본 개념은 기존 리스트형을 사용하여 간단하게 새로운 리스트를 만드는 기법이다. 리스트와 for문을 한 줄에 사용할 수 있는 장점이 있다.

일반적인 반복문 + 리스트

```
>>> result = []
>>> for i in range(10):
...     result.append(i)
...
>>> result
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

리스트 컴프리헨션

```
>>> result = [i for i in range(10)]
>>> result
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# 리스트 컴프리헨션

## ■ 리스트 컴프리헨션 용법 : 조건체크

- 필터링은 if문과 함께 사용하는 리스트 컴프리헨션이다. 일반적으로 짝수만 저장하기 위해서는 다음과 같은 코드를 작성해야 한다.

일반적인 반복문 + 리스트

```
>>> result = []
>>> for i in range(10):
...     if i % 2 == 0:
...         result.append(i)
...
>>> result
[0, 2, 4, 6, 8]
```

리스트 컴프리헨션

```
>>> result = [i for i in range(10) if i % 2 == 0]
>>> result
[0, 2, 4, 6, 8]
```

## ■ 리스트 컴프리 헨션 용법 : 조건체크

- 다음 코드를 보면, 기존 리스트 컴프리헨션문 끝에 `if i % 2 == 0` 을 삽입하여 해당 조건을 만족할 때만 `i`를 추가할 수 있게 한다. 만약 `else`문과 함께 사용하여 해당 조건을 만족하지 않을 때는 다른 값을 할당할 수 있다. 다음 코드를 보자.

```
>>> result = [i if i % 2 == 0 else 10 for i in range(10)]
>>> result
[0, 10, 2, 10, 4, 10, 6, 10, 8, 10]
```

- 위 코드처럼 `if`문을 앞으로 옮겨 `else`문과 함께 사용하면, 조건을 만족하지 않을 때 `else` 뒤에 `i`의 값을 할당하는 코드를 작성할 수 있다.

## ■ 리스트 컴프리헨션 용법 : 중첩 반복문

- 리스트 컴프리헨션에서도 기존처럼 리스트 2개를 섞어 사용할 수 있다.
- 다음 코드와 같이 2개의 for문을 만들 수 있다.

```
>>> word_1 = "Hello"
>>> word_2 = "World"
>>> result = [i + j for i in word_1 for j in word_2]           # 중첩 반복문
>>> result
['HW', 'Ho', 'Hr', 'Hl', 'Hd', 'eW', 'eo', 'er', 'el', 'ed', 'lW', 'lo', 'lr', 'll', 'ld', 'lW',
'lo', 'lr', 'll', 'ld', 'oW', 'oo', 'or', 'ol', 'od']
```

- 위 코드를 보면, word\_1에서 나오는 값을 먼저 고정한 후, word\_2의 값을 하나씩 가져와 결과를 생성한다.



## ■ 리스트 컴프리헨션 용법: 중첩 반복문

- 중첩 반복문에서도 필터링을 적용할 수 있다. 다음과 같이 반복문 끝에 `if` 문을 추가하면 된다.

```
>>> case_1 = ["A", "B", "C"]
>>> case_2 = ["D", "E", "A"]
>>> result = [i + j for i in case_1 for j in case_2 if not(i==j)]
>>> result
['AD', 'AE', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
```

# 리스트 컴프리헨션

## ■ 리스트 컴프리헨션 용법 : 이차원 리스트

- 비슷한 방식으로 이차원 리스트(two-dimensional list) 를 만들 수 있다. 앞 중첩 반복문의 예시 코드 결과는 일차원 리스트(one-dimensional list) 였다. 그렇다면 하나의 정보를 열row 단위로 저장하는 이차원 리스트는 어떻게 만들 수 있을까? 먼저 다음 코드를 보자.

```
>>> words = 'The quick brown fox jumps over the lazy dog'.split()
>>> print(words)
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
>>> stuff = [[w.upper(), w.lower(), len(w)] for w in words]      # 리스트의 각 요소를 대문
자, 소문자, 길이로 변환하여 이차원 리스트로 변환
```

- ➡ 가장 간단한 방법은 위 코드처럼 대괄호 2개를 사용하는 것이다. 이 코드는 기존 문장을split() 함수로 분리하여 리스트로 변환한 후, 각 단어의 대문자, 소문자, 길이를 하나의 리스트로 따로 저장하는 방식이다.

## ■ 리스트 컴프리헨션 용법 : 이차원 리스트

- 저장한 후 결과를 출력하면, 다음과 같이 이차원 리스트를 생성할 수 있다.

```
>>> for i in stuff:  
...     print (i)  
...  
['THE', 'the', 3]  
['QUICK', 'quick', 5]  
['BROWN', 'brown', 5]  
['FOX', 'fox', 3]  
['JUMPS', 'jumps', 5]  
['OVER', 'over', 4]  
['THE', 'the', 3]  
['LAZY', 'lazy', 4]  
['DOG', 'dog', 3]
```

## ■ 리스트 컴프리헨션 용법 : 이차원 리스트

- 다른 방법으로 for문 2개를 붙여 사용할 수도 있다. 여기서 한 가지 주의할 점은 for문 2개를 붙여 사용하면 대괄호의 위치에 따라 for문의 실행이 달라진다는 것이다. 이전에 배웠던 for문은 앞에 있는 for문이 먼저 실행된 후, 뒤의 for문이 실행되었다. 그래서 다음 코드의 경우 A가 먼저 고정되고, D, E, A를 차례대로 붙여 결과가 출력된다.

```
>>> case_1 = ["A", "B", "C"]
>>> case_2 = ["D", "E", "A"]
>>> result = [i + j for i in case_1 for j in case_2]
>>> result
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
```

## ■ 리스트 컴프리헨션 용법 : 이차원 리스트

- 이차원 리스트를 만들기 위해서는 대괄호를 하나 더 사용해야 한다. 그리고 그와 동시에 먼저 작동하는 for문의 순서가 달라진다. 다음 코드는 위의 코드와 달리 리스트 안에 `[i + j for i in case_1]`이 하나 더 존재한다. 따라서 먼저 나온 for문이 고정되는 것이 아니라, 뒤의 for문이 고정된다. 즉, A부터 고정되는 것이 아니라 case\_2의 첫 번째 요소인 D가 고정되고 A, B, C가 차례로 D 앞에 붙는다. 결과를 보면 이차원 리스트 형태로 출력된 것을 확인할 수 있다.

```
>>> result = [[i + j for i in case_1] for j in case_2]
>>> result
[['AD', 'BD', 'CD'], ['AE', 'BE', 'CE'], ['AA', 'BA', 'CA']]
```

## ■ 리스트 컴프리헨션 용법 : 이차원 리스트

- 다음 두 코드는 꼭 구분해야 한다.

①	②
① [i + j for i in case_1 for j in case_2]	
② [[i + j for i in case_1] for j in case_2]	
②	①

- ➔ 첫 번째 코드는 일차원 리스트를 만드는 코드로, 앞의 for문이 먼저 실행된다. 두 번째 코드는 이차원 리스트를 만드는 코드로, 뒤의 for문이 먼저 실행된다. 이 두 코드의 차이를 꼭 이해하고 넘어가자.

## ■ 리스트 컴프리헨션의 성능

- 두 코드의 성능을 비교하기 위해 리눅스 계열에서 사용하는 time 명령어의 결과를 캡처한 것이다. 코드의 총 실행 시간을 확인할 수 있다.

```
real    0m10.230s
user    0m10.203s
sys     0m0.023s
```

(a) 일반적인 반복문 + 리스트

```
real    0m7.788s
user    0m7.762s
sys     0m0.021s
```

(b) 리스트 컴프리헨션

[코드의 실행 시간을 통한 성능 비교]

## ❖ 목차

- 1. 컬렉션 관리 함수
- 2. 람다 함수
- 3. 컬렉션의 사본



# 1. 컬렉션 관리 함수

## ❖ enumerate

- 순서값과 요소값을 한꺼번에 구하는 내장함수

- Ex) 여러 학생의 성적을 출력할 경우

- for문 이용
  - 순서값을 알 수 없음
  - 별도 변수 활용한 복잡한 과정 통해 순서값 구함

```
score = [ 88, 95, 70, 100, 99 ]  
for no, s in enumerate(score, 1):  
    print(str(no) + "번 학생의 성적 :", s)
```

sequence

```
score = [ 88, 95, 70, 100, 99 ]  
for s in score:  
    print("성적 :", s)
```

sequence3

```
score = [ 88, 95, 70, 100, 99 ]  
for no in range(len(score)):  
    print(str(no + 1) + "번 학생의 성적 :", score[no])
```

# 1. 컬렉션 관리 함수



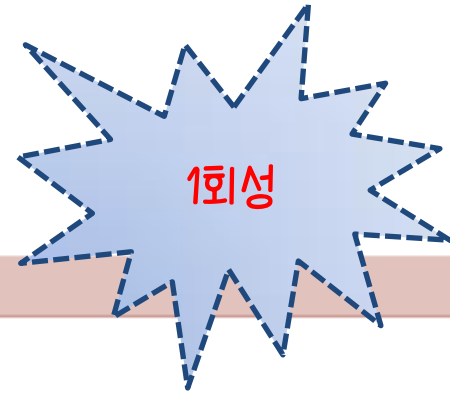
```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):    # 리스트에 있는 인덱스와 값을 언패킹
...     print(i, v)
...
0 tic
1 tac
2 toe
```

```
>>> {i:j for i,j in enumerate('TEAMLAB is an academic institute located in South
Korea.'.split())}
{0: 'TEAMLAB', 1: 'is', 2: 'an', 3: 'academic', 4: 'institute', 5: 'located', 6: 'in',
7: 'South', 8: 'Korea.'}
```

# 1. 컬렉션 관리 함수

## ❖ zip

- 여러 개 컬렉션 합쳐 하나로 만들
- 두 리스트의 대응되는 요소끼리 짝지어 **튜플 리스트** 생성



zip

```
yoil = ["월", "화", "수", "목", "금", "토", "일"]  
food = ["갈비탕", "순대국", "칼국수", "삼겹살"]  
menu = zip(yoil, food)  
for y, f in menu:  
    print("%s요일 메뉴 : %s" % (y, f))
```

실행결과

```
월요일 메뉴 : 갈비탕  
화요일 메뉴 : 순대국  
수요일 메뉴 : 칼국수  
목요일 메뉴 : 삼겹살
```

- 합쳐지는 두 리스트의 길이는 무관
- 생성되는 튜플의 순서는 원본 리스트의 순서와 같음

# 1. 컬렉션 관리 함수



```
>>> alist = ['a1', 'a2', 'a3']
>>> blist = ['b1', 'b2', 'b3']
>>> for a, b in zip(alist, blist):           # 병렬로 값을 추출
...     print(a, b)
...
a1 b1
a2 b2
a3 b3
```

```
>>> a, b, c = zip((1, 2, 3), (10, 20, 30), (100, 200, 300))
>>> print (a, b, c)
(1, 10, 100) (2, 20, 200) (3, 30, 300)
>>> [sum(x) for x in zip((1, 2, 3), (10, 20, 30), (100, 200, 300))]
[111, 222, 333]
```

# 1. 컬렉션 관리 함수



```
>>> alist = ['a1', 'a2', 'a3']
>>> blist = ['b1', 'b2', 'b3']
>>> for i, (a, b) in enumerate(zip(alist, blist)):
...     print(i, a, b)                                # (인덱스, alist[인덱스], blist[인덱스]) 표시
...
0 a1 b1
1 a2 b2
2 a3 b3
```

## 2. 람다 함수

### ❖ filter 함수

- 리스트 요소 중 조건에 맞는 것만을 골라냄
- 첫 번째 인수 : 조건 지정하는 함수
- 두 번째 인수 : 대상 리스트

filter

```
def flunk(s):  
    return s < 60  
  
score = [ 45, 89, 72, 53, 94 ]  
for s in filter(flunk, score):  
    print(s)
```

실행결과

45  
53

### ■ flunk 함수

- 점수 s를 인수로 받아 60 미만인지 조사

## 2. 람다 함수

### ❖ map 함수

- 모든 요소에 대한 변환함수 호출, 새 요소 값으로 구성된 리스트 생성
- 첫 번째 인수 : 조건 지정하는 함수
- 두 번째 인수 : 대상 리스트

map

```
def half(s):  
    return s / 2  
  
score = [ 45, 89, 72, 53, 94 ]  
for s in map(half, score):  
    print(s, end = ', ')
```

실행결과

22.5, 44.5, 36.0, 26.5, 47.0,

### ■ half 함수

- 인수로 전달받은 s를 절반으로 나누어 리턴

## 2. 람다 함수

### ❖ 람다 함수

- 이름 없고 입력과 출력만으로 함수를 정의하는 축약된 방법

- `lambda` 인수 : 식

- 인수는 여러 개 가질 수 있음

호출 시 전달할 리턴 값

호출 시 전달받을  
아규먼트(생략 가능)

```
lambda x:x + 1
```

```
lambda
```

```
score = [ 45, 89, 72, 53, 94 ]  
for s in filter(lambda x:x < 60, score):  
    print(s)
```



### 3. 컬렉션의 사본

#### ❖ 리스트의 사본

- 기본형 변수는 대입 이후 둘 중 하나 바꾸어도 다른 쪽에 영향 없음
- 컬렉션의 경우, 같은 리스트를 두 변수가 가리키는 것이라 영향 있음

##### varcopy

```
a = 3
b = a
print("a = %d, b = %d" % (a, b))

a = 5
print("a = %d, b = %d" % (a, b))
```

a  b 




##### 실행결과

```
a = 3, b = 3
a = 5, b = 3
```

##### listcopy

```
list1 = [ 1, 2, 3 ]
list2 = list1

list2[1] = 100
print(list1)
print(list2)
```

list1   



list2

##### 실행결과

```
[1, 100, 3]
[1, 100, 3]
```

### 3. 컬렉션의 사본

- **copy 메서드**로 두 리스트를 완전히 독립 사본으로 만들 수 있음

listcopy2	
<pre>list1 = [ 1, 2, 3 ] list2 = list1.copy()  list2[1] = 100 print(list1) print(list2)</pre>	<div>list1 </div> <div>list2 </div>
실행결과	<pre>[1, 2, 3] [1, 100, 3]</pre>

- list[:] 으로 전체 범위에 대한 사본 만드는 방법도 가능

### 3. 컬렉션의 사본

#### deepcopy

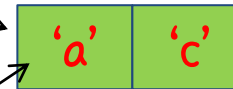
```
list0 = [ 'a', 'b' ]  
list1 = [ list0, 1, 2 ]  
list2 = list1.copy()
```

```
list2[0][1] = 'c'  
print(list1)  
print(list2)
```

실행결과

```
[[ 'a', 'c' ], 1, 2]  
[[ 'a', 'c' ], 1, 2]
```

list1



list2



#### deepcopy2

```
import copy
```

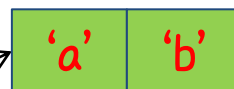
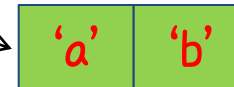
```
list0 = [ "a", "b" ]  
list1 = [ list0, 1, 2 ]  
list2 = copy.deepcopy(list1)
```

```
list2[0][1] = "c"  
print(list1)  
print(list2)
```

실행결과

```
[[ 'a', 'b' ], 1, 2]  
[[ 'a', 'c' ], 1, 2]
```

list1



list2



### 3. 컬렉션의 사본

#### ❖ is 연산자

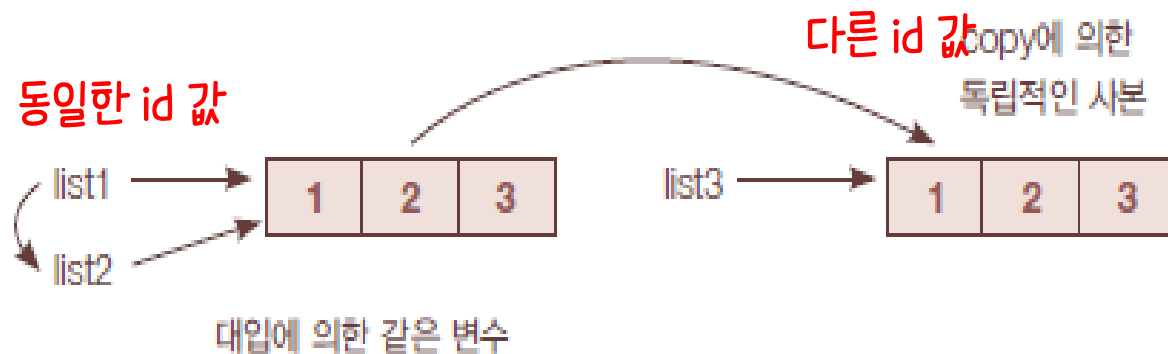
- is 구문 통해 두 변수가 같은 객체 가리키는지 조사

is

```
list1 = [ 1, 2, 3 ]  
list2 = list1  
list3 = list1.copy()  
  
print("1 == 2" , list1 is list2)  
print("1 == 3" , list1 is list3)  
print("2 == 3" , list2 is list3)
```

실행결과

```
1 == 2 True  
1 == 3 False  
2 == 3 False
```



## ■ 리스트의 메모리 저장

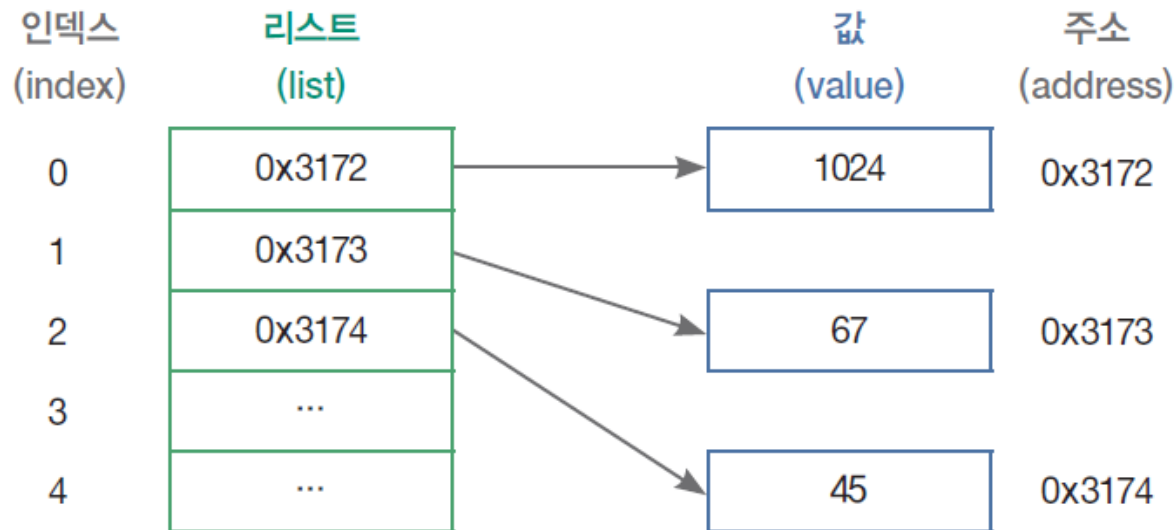
- 다음 코드에서 가장 핵심 코드는 `math_score[0] = 1000`이다. 분명히 `math_score`의 값을 변경하였는데 `midterm_score` 두 번째 행의 첫 번째 값이 변경되었다. 이는 파이썬 리스트가 값을 저장하는 방식 때문에 발생하는 현상이다.

```
>>> kor_score = [49, 79, 20, 100, 80]
>>> math_score = [43, 59, 85, 30, 90]
>>> eng_score = [49, 79, 48, 60, 100]
>>>
>>> midterm_score = [kor_score, math_score, eng_score]
>>> midterm_score
[[49, 79, 20, 100, 80], [43, 59, 85, 30, 90], [49, 79, 48, 60, 100]]
>>>
>>> math_score[0] = 1000
>>> midterm_score
[[49, 79, 20, 100, 80], [1000, 59, 85, 30, 90], [49, 79, 48, 60, 100]]
```

# 리스트의 메모리 관리 방식

## 리스트의 메모리 저장

- 파이썬은 리스트를 저장할 때 값 자체가 아니라, 값이 위치한 메모리 주소(reference)를 저장한다.



[ 리스트의 메모리 저장 ]

## ■ 리스트의 메모리 저장

- `==` 은 값을 비교하는 연산이고, `is`는 메모리의 주소를 비교하는 연산이다.
- 아래 코드에서 `a`와 `b`는 값은 같지만, 메모리의 저장 주소는 다른 것이다.

```
>>> a = 300  
>>> b = 300  
>>> a is b  
False  
>>> a == b  
True
```

## ■ 리스트의 메모리 저장

- 이전 코드와 다르게 is와 == 연산자는 모두 True를 반환한다. 그렇다면 a와 b의 메모리 주소는 같은 것일까? 이것은 파이썬의 정수형 저장 방식의 특성 때문이다. 파이썬은 인터프리터가 구동될 때, -5부터 256까지의 정수값을 특정 메모리 주소에 저장한다. 그리고 해당 숫자를 할당하려고 하면 해당 변수는 그 숫자가 가진 메모리 주소로 연결한다.

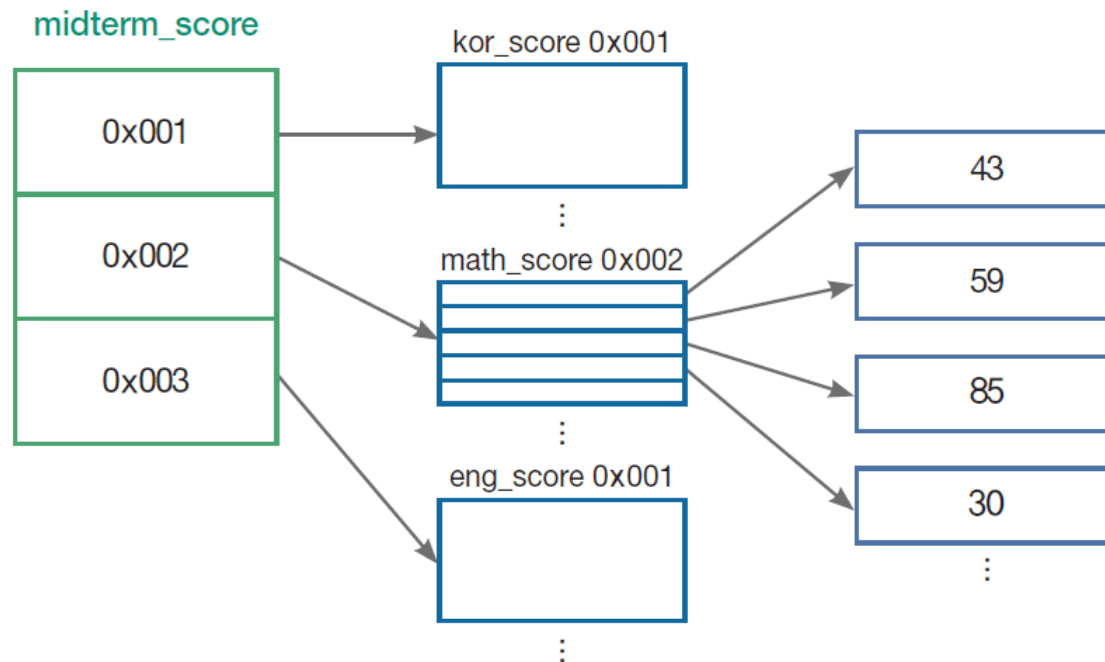
```
>>> a = 1
>>> b = 1
>>> a is b
True
>>> a == b
True
```



# 리스트의 메모리 관리 방식

## ■ 리스트의 메모리 저장

- 리스트는 기본적으로 값을 연속으로 저장하는 것이 아니라, 값이 있는 주소를 저장하는 방식이다.



[리스트의 메모리 저장 연결 관계]

# 리스트의 메모리 관리 방식

## ■ 메모리 저장 구조로 인한 리스트의 특징

- 다양한 형태의 변수가 하나의 리스트에 들어갈 수 있다.

```
>>> a = ["color", 1, 0.2]
```

- 기존 변수들과 함께 리스트 안에 다른 리스트를 넣을 수 있다. 흔히 이를 중첩 리스트라고 한다. 이러한 특징은 파이썬의 리스트가 값이 아닌 메모리의 주소를 저장해 메모리에 새로운 값을 할당하는 데 있어 매우 높은 자유도를 보장하므로 가능하다.

```
>>> a = ["color", 1, 0.2]
>>> color = ['yellow', 'blue', 'green', 'black', 'purple']
>>> a[0] = color                # 리스트 안에 리스트도 입력 가능
>>> print(a)
[['yellow', 'blue', 'green', 'black', 'purple'], 1, 0.2]
```

## ■ 메모리 저장 구조로 인한 리스트의 특징

- 리스트의 저장 방식
- b와 a 변수를 각각 다른 값으로 선언한 후, b에 a를 할당하였다. 그리고 b를 출력하면, a 변수와 같은 값이 화면에 출력된다.

```
>>> a = [5, 4, 3, 2, 1]
>>> b = [1, 2, 3, 4, 5]
>>> b = a
>>> print(b)
[5, 4, 3, 2, 1]
```

# 리스트의 메모리 관리 방식

## ■ 메모리 저장 구조로 인한 리스트의 특징

- 리스트의 저장 방식

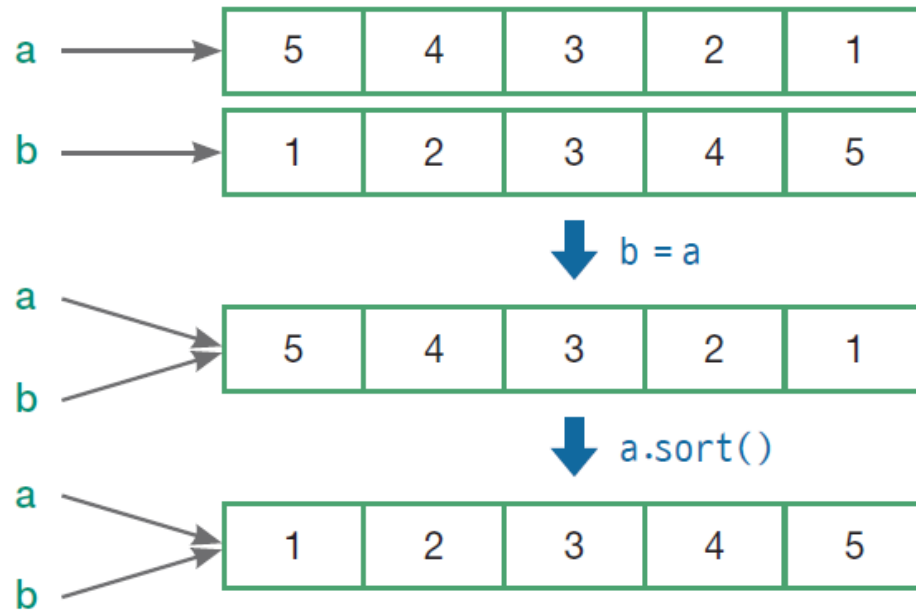
- $a$ 만 정렬하고  $b$ 를 출력했을 때  $b$ 도 정렬되었다. 두 변수가 같은 메모리 주소와 연결되어 있으므로, 하나의 변수값만 바뀌더라도 둘 다 영향을 받는다.

```
>>> a.sort()
>>> print(b)
[1, 2, 3, 4, 5]
```

# 리스트의 메모리 관리 방식

## ■ 메모리 저장 구조로 인한 리스트의 특징

### • 리스트의 저장 방식



[ `a`를 정렬했는데 `b`도 정렬되는 이유 ]

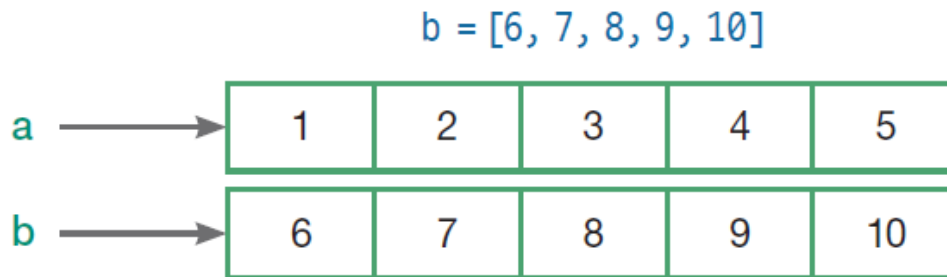
# 리스트의 메모리 관리 방식

## ■ 메모리 저장 구조로 인한 리스트의 특징

- 리스트의 저장 방식

- b에 새로운 값을 할당하면 b는 이제 새로운 메모리 주소에 새로운 값을 할당할 수 있는 것이다.

```
>>> b = [6, 7, 8, 9, 10]
>>> print(a, b)
[1, 2, 3, 4, 5] [6, 7, 8, 9, 10]
```



[ b에 새로운 값을 할당하는 경우 ]

## ❖ 모듈의 활용

### ■ 모듈

- 파이썬 코드를 저장하는 기본 단위
- 파이썬 소스 파일로서 .py 빼고 파일명으로 불림
- 파이썬에서 자주 사용하는 기능은 표준 모듈로 제공됨
- 내가 만들 파이썬 파일도 모듈이 될 수 있음
- 다른 파이썬 소스 즉 모듈의 함수나 변수 등을 현재 파이썬 소스에서 사용하려면 import 구문으로 포함시켜야 함

`import 함수명 [ as 별칭 ]`

`from 모듈 import 함수명 [ as 별칭 ] [, 함수명 [ as 별칭 ]]`

# 1. 수학

## ■ from 모듈 import 함수명

- 모듈의 함수 호출

fromimport

```
from math import sqrt
```

```
print(sqrt(2))
```

- 이 경우 sqrt 외 math에 속한 다른 함수는 사용할 수 없음



# 1. 수학

## ❖ math 모듈

- 정밀한 계산을 위한 복잡한 수학 연산 함수

상수	설명
pi	원주율 상수
tau	원주율의 2배 되는 상수. pi는 지름과 원둘레의 비율인 데 비해 tau는 반지름과 원둘레의 비율이다. pi 보다 계산식이 단순해져 타우를 쓰자고 주장하는 수학자들이 있다. 파이썬 3.6에서 추가되었다.
e	자연 대수 상수
inf	무한대 값
nan	숫자가 아닌 값을 의미한다.

# 1. 수학



함수	설명
<code>sqrt(x)</code>	x의 제곱근을 구한다. 세제곱근은 1/3승을 계산하여 구한다.
<code>pow(x, y)</code>	x의 y승을 계산한다. ** 연산자와 기능은 같지만 인수를 모두 실수로 바꾼 후 연산한다는 차이가 있다.
<code>hypot(x, y)</code>	피타고라스의 정리에 의거 x제곱 + y제곱의 제곱근을 구한다.
<code>factorial(x)</code>	x의 계승을 구한다. 인수 x는 양의 정수만 가능하다.
<code>sin(x), cos(x), tan(x)</code>	삼각함수를 계산한다. 인수 x는 라디안 값이다.
<code>asin(x), acos(x), atan(x), atan2(y,x)</code>	역삼각함수를 계산한다. 인수 x는 라디안 값이다.
<code>sinh(x), cosh(x), tanh(x)</code>	쌍곡선 삼각함수를 계산한다. 인수 x는 라디안 값이다.
<code>asinh(x), acosh(x), atanh(x)</code>	쌍곡선 역삼각함수를 계산한다. 인수 x는 라디안 값이다.
<code>degrees(x)</code>	라디안 값을 각도로 바꾼다.
<code>radians(x)</code>	각도를 라디안 값으로 바꾼다.
<code>ceil(x)</code>	수직선 오른쪽의 올림 값을 찾는다.
<code>floor(x)</code>	수직선 왼쪽의 내림 값을 찾는다.
<code>fabs(x)</code>	x의 절대값을 구한다.
<code>trunc(x)</code>	x의 소수점 이하를 버린다.
<code>log(x, base)</code>	base에 대한 x의 로그를 구한다. base가 생략되면 자연 로그를 구한다.
<code>log10(x)</code>	10의 로그를 구한다. <code>log(x, 10)</code> 과 같다.
<code>gcd(a, b)</code>	a, b의 최대공약수를 구한다.

# 1. 수학

## ■ 삼각함수, 제곱근 등 연산 예시

sin

```
import math

print(math.sin(math.radians(45)))
print(math.sqrt(2))
print(math.factorial(5))
```

실행결과

```
0.7071067811865475
1.4142135623730951
120
```

## 2. 시간

### ❖ time 모듈

- 날짜와 시간 관련 기능 제공
- 에폭(Epoch) / 유닉스 시간

time

```
import time  
  
print(time.time())
```

실행결과     1515549457.5692239

- 일상 시간 문자열로 변환 가능

ctime

```
import time  
  
t = time.time()  
print(time.ctime(t))
```

실행결과     Wed Jan 10 10:58:24 2018

## 2. 시간

- 보다 편리한 형태로 조립하려면

- `localtime` 함수

- 지역 시간 고려하여 현지 시간 구함
- 시간 요소 멤버로 가지는 `struct_time`형 객체 반환
- 정보 분리하여 문자열로 조립

```
time.struct_time(tm_year=2020, tm_mon=7, tm_mday=22,  
tm_hour=13, tm_min=30, tm_sec=47, tm_wday=2, tm_yday=204,  
tm_isdst=0)
```

## 2. 시간

### ❖ 실행 시간 측정

- time 함수 호출하는 시점에 따라 구해지는 시간이 다름을 이용
- 두 지점 간의 경과 시간 측정

ellapse

```
import time

start = time.time()
for a in range(1000):
    print(a)
end = time.time()
print(end - start)
```

실행결과

```
....
998
999
3.6027116775512695
```

- 소수점 이하 값까지 지정 가능

## 2. 시간

### ❖ calendar 모듈

- 달력 기능
- 인수로 받은 연도의 달력 객체 반환
- month 함수
  - 연도와 달을 인수로 받아 해당 월 달력 객체 반환
- weekday 함수
  - 특정 날짜가 어떤 요일인지 조사

## 2. 시간

### calendar

```
import calendar

print(calendar.calendar(2018))
print(calendar.month(2019, 1))
#calendar.prcal(2018)
#calendar.prmonth(2019, 1)
```

실행결과

```

                                     2018
      January                February                March
Mo Tu We Th Fr Sa Su    Mo Tu We Th Fr Sa Su    Mo Tu We Th Fr Sa
Su
  1  2  3  4  5  6  7          1  2  3  4          1  2  3  4
  8  9 10 11 12 13 14      5  6  7  8  9 10 11      5  6  7  8  9 10 11
 15 16 17 18 19 20 21      12 13 14 15 16 17 18      12 13 14 15 16 17 18
 22 23 24 25 26 27 28      19 20 21 22 23 24 25      19 20 21 22 23 24 25
 29 30 31                  26 27 28                  26 27 28 29 30 31
....
      January 2019
Mo Tu We Th Fr Sa Su
      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
```



### 3. 난수

#### ❖ random 모듈

- 난수 생성 기능
- 어떤 수가 나올 지 예측할 수 없는 무작위 동작 구현

random

```
import random  
  
for i in range(5):  
    print(random.random())
```

실행결과

```
0.05560178175601582  
0.7483996952798034  
0.054579188940304335  
0.22688047568249736  
0.9948204231812777
```

### 3. 난수

#### ■ randint(begin, end)

- 일정 범위의 정수 난수 범위 설정

randint	
<pre>import random  for i in range(5):     print(random.randint(1,10))</pre>	
실행결과	1 4 5 10 3

#### ■ randrange(begin, end)

- end는 범위에서 제외

#### ■ choice 함수

- 리스트에서 임의의 요소 하나 골라 반환

### 3. 난수

#### ■ shuffle 함수

- 리스트의 요소 무작위로 섞음

shuffle

```
import random

food = ["짜장면", "짬뽕", "탕수육", "군만두"]
print(food)
random.shuffle(food)
print(food)
```

실행결과

```
['짜장면', '짬뽕', '탕수육', '군만두']
['군만두', '짬뽕', '짜장면', '탕수육']
```

#### ■ sample 함수

- 리스트 항목 중 n개를 무작위로 뽑아 새 리스트 만들기

### 3. 난수

#### ❖ 산수 문제 내기

- 난수로 무작위 숫자 두 개를 골라 산수 문제 출제하는 프로그램

mathquiz

```
import random

a = random.randint(1, 9)
b = random.randint(1, 9)
question = "%d + %d = ? " % (a, b)
c = int(input(question))

if c == a + b:
    print("정답입니다.")
else:
    print("틀렸습니다.")
```

실행결과

1 + 7 = ? 8  
정답입니다.

## 4. sys 모듈

### ❖ sys 모듈

- 파이썬 해석기가 실행되는 환경과 해석기의 여러 기능 조회 및 관리

sys

```
import sys

print("버전 :", sys.version)
print("플랫폼 :", sys.platform)
if (sys.platform == "win32"):
    print(sys.getwindowsversion())
print("바이트 순서 :", sys.byteorder)
print("모듈 경로 :", sys.path)
sys.exit(0)
```

실행결과

```
버전 : 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32
bit (Intel)]
플랫폼 : win32
sys.getwindowsversion(major=10, minor=0, build=15063, platform=2,
service_pack='')
바이트 순서 : little
모듈 경로 : ['C:\\PyStudy\\CharmTest', 'C:\\PyStudy\\CharmTest',
'C:\\Python\\python36.zip', 'C:\\Python\\DLLs', 'C:\\Python\\lib',
'C:\\Python', 'C:\\Python\\lib\\site-packages']
```

## 4. sys 모듈

### ❖ 명령행 인수

- 파이썬에서 실행 파일 뒤에 인수를 전달할 수 있음

```
copy a.txt b.txt
```

- a.txt / b.txt
  - 명령행 인수
  - 명령 수행할 대상이나 옵션 지정

sysarg

```
import sys  
  
print(sys.argv)
```

- **sys.argv** 읽어 명령행 인수의 값 읽을 수 있음

```
C:\PyStudy>sysarg.py korea option  
['C:\\PyStudy\\sysarg.py', 'korea', 'option']
```

## 4. sys 모듈

### ❖ 경과일 계산

- 특정 날짜로부터 오늘까지 며칠이 경과되었는지 계산하여 출력

datecalc

```
import sys
import time

if (len(sys.argv) != 2):
    print("시작 날짜를 yyyyymmdd로 입력하십시오.")
    sys.exit(0)

birth = sys.argv[1]
if (len(birth) != 8 or birth.isnumeric() == False):
    print("날짜 형식이 잘못되었습니다.")
    sys.exit(0)

tm = (int(birth[:4]), int(birth[4:6]), int(birth[6:8]), 0, 0, 0, 0, 0, 0)
ellapse = int((time.time() - time.mktime(tm)) / (24 * 60 * 60))
print(ellapse)
```

- 명령행으로 인수 전달할 경우 사용자가 사용법 확실히 숙지하여야 함
  - 인수보다는 질문을 하고 직접 입력받아 사용하는 것이 보다 정확함



## ❖ 목차

- 1. 예외 처리
- 2. 자원 정리



# 1. 예외 처리

## ❖ 예외 (Exception)

- 프로그램 코드는 이상이 없으나 실행 중에 불가피하게 발생하는 문제

exception

```
str = "89점"  
score = int(str)  
print(score)  
print("작업완료")
```

실행결과

```
Traceback (most recent call last):  
  File "C:/Python/test.py", line 2, in <module>  
    score = int(str)  
ValueError: invalid literal for int() with base 10: '89점'
```

- 에러 발생 직후 프로그램 종료되어 이후 명령 무시

# 1. 예외 처리

## ❖ 예외 처리

```
try:  
    실행할 명령  
except 예외 as 변수:  
    오류 처리문  
else:  
    예외가 발생하지 않을 때의 처리
```

- 간단하게 **try: except:** 까지만 사용하는 것도 가능

tryexcept

```
str = "89점"  
try:  
    score = int(str)  
    print(score)  
except:  
    print("예외가 발생했습니다.")  
print("작업완료")
```

실행결과

예외가 발생했습니다.  
작업 완료

# 1. 예외 처리

---

- 예외 발생하지 않으면 except 블록 무시하고 다음 문장 실행
- 상황 자체를 해결하지는 않으나 예외 인식 / 처리 기회 제공
- 예외 처리 구문을 루프로 감싸면 무엇이 잘못되었는지 알려줌
  - 전체를 무한 루프로 감싸 예외 발생 시 다시 루프 선두로 돌아감

# 1. 예외 처리

## ❖ 예외의 종류

예외	설명
NameError	명칭이 발견되지 않는다. 초기화하지 않은 변수를 사용할 때 발생한다.
ValueError	타입은 맞지만 값의 형식이 잘못되었다.
ZeroDivisionError	0으로 나누었다.
IndexError	첨자가 범위를 벗어났다.
TypeError	타입이 맞지 않다. 숫자가 필요한 곳에 문자열을 사용한 경우 등이다.

# 1. 예외 처리

excepts

```
str = "89"
try:
    score = int(str)
    print(score)
    a = str[5]
except ValueError:
    print("점수의 형식이 잘못되었습니다.")
except IndexError:
    print("첨자 범위를 벗어났습니다.")
print("작업완료")
```

실행결과

```
89
첨자 범위를 벗어났습니다.
작업완료
```

- str[5] → IndexError 예외 발생
- except 블록 아무리 많아도 먼저 발생한 예외에 맞는 하나만 선택됨
- 두 개 이상 괄호로 묶어 except 블록에서 동시에 받기 가능

```
except (ValueError, IndexError):
    print("점수의 형식이나 첨자가 잘못되었습니다.")
```

# 1. 예외 처리

## ❖ raise 명령

- 고의적으로 예외 발생시킴
- 작업 위한 선결조건 맞지 않거나 문제 발생할 경우 호출원으로 사용

### raise

```
def calcsun(n):  
    if (n < 0):  
        raise ValueError  
    sum = 0  
    for i in range(n+1):  
        sum = sum + i  
    return sum  
  
try:  
    print("~10 =", calcsun(10))  
    print("~-5 =", calcsun(-5))  
except ValueError:  
    print("입력값이 잘못되었습니다.")
```

### 실행결과

```
~10 = 55  
입력값이 잘못되었습니다.
```

# 1. 예외 처리

- 특이값 리턴하는 방식도 가능
  - 리턴값 점검에 철저할 것

errorret

```
def calcsun(n):  
    if (n < 0):  
        return -1  
    sum = 0  
    for i in range(n+1):  
        sum = sum + i  
    return sum  
  
result = calcsun(10)  
if result == -1:  
    print("입력값이 잘못되었습니다.")  
else:  
    print("~10 =", result)  
  
result = calcsun(-5)  
if result == -1:  
    print("입력값이 잘못되었습니다.")  
else:  
    print("~10 =", result)
```

실행결과

~10 = 55  
입력값이 잘못되었습니다.

# 1. 예외 처리

---

- 예외 블록은 평이하게 작성하되 예외 발생 시에만 except 블록 점프
- 함수별로 비정상 상황 처리 방법 다름
- 예외 던지는 구문은 반드시 try 블록으로 감쌀 것



## 2. 자원 정리

### ❖ finally 블록

- 예외 발생 여부와 무관하게 반드시 실행해야 할 명령 지정

finally

```
try:
    print("네트워크 접속")
    a = 2 / 0
    print("네트워크 통신 수행")
finally:
    print("접속 해제")
print("작업 완료")
```

실행결과

```
네트워크 접속
접속 해제
Traceback (most recent call last):
  File "C:/Python/test.py", line 3, in <module>
    a = 2 / 0
ZeroDivisionError: division by zero
```

## 2. 자원 정리

### ❖ assert 문장

- 프로그램의 현재 상태가 맞는지 확인
- assert 조건, 메시지

```
assert
```

```
score = 128  
assert score <= 100, "점수는 100 이하여야 합니다"  
print(score)
```