단국대학교
SW중심대학

창의적 사고와 코딩

Lecture 6. 리스트

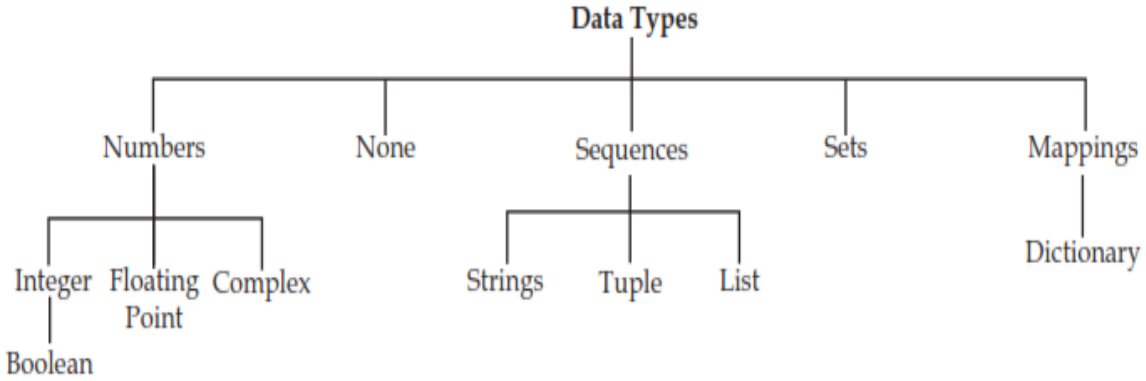
시퀀스 자료형

파이썬 자료형(data types)

- 수치 자료형
  - int(정수), float (실수), complex (복소수): immutable (객체를 수정할 수 없음)

```
>>> num1 = 10
>>> num2 = 10.0
>>> num3 = 10 + 5j
>>> type(num1), type(num2), type(num3)
(<class 'int'>, <class 'float'>, <class 'complex'>)
```

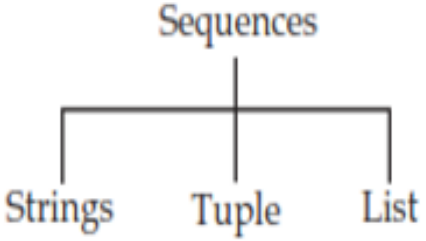
  - 부울 자료형: bool (True / False)
- 군집으로 다루는 자료형
  - str(문자열) : immutable
  - list (리스트) : mutable(수정 가능)
  - tuple (튜플) : immutable
  - set (집합) : mutable
  - dict(사전) : mutable



파이썬 자료형(data types)

시퀀스(Sequence)

- 요소(element)로 구성되어 있고 요소 간에 순서가 있는 자료구조
- 시퀀스 요소들은 번호(index)를 부여받으며, 인덱스는 0번 부터 부여
- 시퀀스에 속하는 자료 구조들은 동일한 연산을 지원
  - 인덱싱(indexing), 슬라이싱(Slicing), 덧셈 연산(Adding), 곱셈 연산(Multiplying)



함수나 연산자	설명	예	결과
len()	길이 계산	len([1, 2, 3])	3
+	2개의 시퀀스 연결	[1, 2] + [3, 4, 5]	[1, 2, 3, 4, 5]
*	반복	['Welcome!'] * 3	['Welcome!', 'Welcome!', 'Welcome!']
in	소속	3 in [1, 2, 3]	True
not in	소속하지 않음	5 not in [1, 2, 3]	True
[]	인덱스	myList[1]	myList의 1번째 요소
min()	시퀀스에서 가장 작은 요소	min([1, 2, 3])	1
max()	시퀀스에서 가장 큰 요소	max([1, 2, 3])	3
for 루프	반복	for x in [1, 2, 3]: print (x)	1 2 3

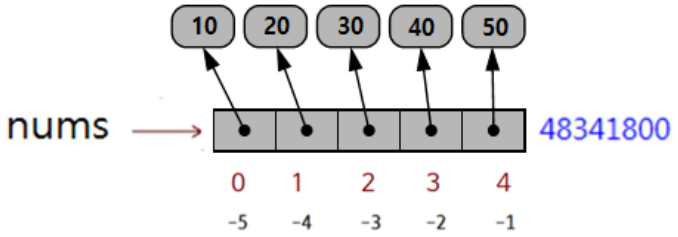
리스트

- 대괄호 [ ] 로 표현
- 하나의 변수에 다양한 종류의 데이터를 할당 관리하는 자료형
  - 다른 언어의 배열과 유사하나 배열의 크기는 고정
  - 리스트의 크기는 가변적
    → 요소의 개수에 따라 커지거나 작아질 수 있음

```
>>> myList = [1, 'computer', 3.4]
>>> myList = ["apple", [10, 20], ['a','b', 'c'], 0.01]
```

- 데이터 변경 가능: mutable
- 순서가 있고, 인덱스를 이용하여 데이터에 접근
- 리스트에 대한 +(연결), *(반복) 연산

```
>>> nums = [10, 20, 30, 40,50]
>>> print(nums)
[10, 20, 30, 40, 50]
>>> type(nums)
<class 'list'>
>>> id(nums)
48341800
```




```
>>> nums[1] = 15
>>> nums
[10, 15, 30, 40, 50]
>>> nums2 = [1,2,3]
>>> nums + nums2
[10, 15, 30, 40, 50, 1, 2, 3]
>>> nums2 * 2
[1, 2, 3, 1, 2, 3]
```

리스트

- In, not in, len()

```
>>> lista = [1,3,5,7,9]
>>> 7 in lista
True
>>> 10 not in lista
True
>>> 2 in lista
False
>>> len(lista)
5
```

- 리스트에서 사용할 수 있는 메소드

```
>>> dir(list)
['__add__', '__class__', …… '__subclasshook__', 'append',
'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

- 빈 리스트: [ ] , list()
- 리스트 비우기 : clear()

```
>>> list1 = list('Hello')
>>> list1
['H', 'e', 'l', 'l', 'o']
>>> list2 = list(range(0,5))
>>> list2
[0, 1, 2, 3, 4]
```

```
>>> L = []
>>> print(L)
[]
>>> A = list()
>>> print(A)
[]
>>> L.append(1)
>>> L.append(2)
>>> print(L)
[1, 2]
>>> L.clear()
>>> print(L)
[]
>>>
```

- 리스트명.메소드 형태로 사용

```
>>> nums = [ ]
>>> nums.append(100)
```

yunyi96@gmail.com

리스트 객체에서 사용할 수 있는 메소드

메소드	설명
clear()	리스트를 비운다.
copy()	리스트를 복사한다.
count (x)	데이터 x의 개수를 알아낸다.
append (x)	데이터 x를 리스트 끝에 추가한다.
extend (M)	리스트 M을 연결한다.
index (x)	데이터 x의 위치를 알아낸다 (인덱스를 알려준다).
insert (i,x)	데이터 x를 지정한 위치 (인덱스 i)에 삽입한다.
pop()	리스트의 지정한 값 하나를 읽어 내고 삭제한다.
remove (x)	리스트에서 데이터 x를 삭제한다.
reverse()	리스트의 순서를 역순으로 바꾼다.
sort()	리스트를 정렬한다.

리스트

```
>>> heroes = [ "스파이더맨", "슈퍼맨"]
>>> heroes.append('헐크')
>>> heroes
['스파이더맨', '슈퍼맨', '헐크']
>>> heroes.extend(["아이언맨", "배트맨"])
>>> heroes
['스파이더맨', '슈퍼맨', '헐크', '아이언맨', '배트맨']
>>> heroes.insert(1, '블랙팬서')
>>> heroes
['스파이더맨', '블랙팬서', '슈퍼맨', '헐크', '아이언맨', '배트맨']
>>> heroes.pop()
'배트맨'
>>> heroes.pop(1)
'블랙팬서'
>>> heroes
['스파이더맨', '슈퍼맨', '헐크', '아이언맨']
>>> heroes.remove('헐크')
>>> heroes
['스파이더맨', '슈퍼맨', '아이언맨']
>>> heroes.append('스파이더맨')
>>> heroes
['스파이더맨', '슈퍼맨', '아이언맨', '스파이더맨']
```

```
>>> heroes.count('스파이더맨')
2
>>> heroes.index('슈퍼맨')
1
>>> heroes.clear()
>>> heroes
[]
```

```
>>> listA = [3,5,6,4,10]
>>> sorted(listA) #원본은 유지되고 새로이 정렬된 리스트를 반환
[3, 4, 5, 6, 10]
>>>
>>> listA
[3, 5, 6, 4, 10]
```

#sored()내장함수

```
>>> listA = [3, 5, 6, 4, 10]
>>> listA.sort()
>>> listA
[3, 4, 5, 6, 10]
>>>
>>> listA.sort(reverse=True)
>>> listA
[10, 6, 5, 4, 3]
>>>
>>> listB = [3,5,7,4,2]
>>> listB.reverse()
>>> listB
[2, 4, 7, 5, 3]
```

리스트

- 비교 연산자 ==, !=, >, <를 사용하여서 2개의 리스트를 비교할 수 있음

```
>>> list1 = [1,2,3]
>>> list2 = [1,2,3]
>>> list1 = list2
>>> list1 > [1,2]
True
>>> list1>[1,2,3,4,5]
False
```

- 리스트 안에서 최소값과 최대값을 찾으려면 내장 메소드인 max()와 min()을 사용하면 된다.

```
>>> values = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
>>> min(values)
1
>>> max(values)
10
```

Lab: 성적 처리 프로그램

- 학생들의 성적을 처리하는 프로그램을 완성시켜보자. 사용자로부터 성적을 입력받아서 리스트에 저장한다. 성적의 평균을 구하고 80점 이상 성적을 받은 학생의 숫자를 계산하여 출력해보자.

```
score.py ===========
성적을 입력하시요: 20
성적을 입력하시요: 40
성적을 입력하시요: 60
성적을 입력하시요: 80
성적을 입력하시요: 100
성적 평균은 60.0 입니다.
80점 이상 성적을 받은 학생은 2 명입니다.
>>>
```

Solution

```
STUDENTS = 5

scores = []
scoreSum = 0

for i in range(STUDENTS):
    value = int(input("성적을 입력하시요: "))
    scores.append(value)
    scoreSum += value

scoreAvg = scoreSum / len(scores)

highScoreStudents = 0
for i in range(len(scores)):
    if scores[i] >= 80:
        highScoreStudents += 1

print("성적 평균은", scoreAvg, "입니다.")
print("80점 이상 성적을 받은 학생은", highScoreStudents, "명입니다.")
```

Solution2

```
STUDENTS = 5

scores = []
scoreSum = 0
highScoreStudents = 0

for i in range(STUDENTS):
    value = int(input("성적을 입력하시요: "))
    scores.append(value)
    scoreSum += value
    if scores[i] >= 80:
        highScoreStudents += 1

scoreAvg = scoreSum / len(scores)

print("성적 평균은", scoreAvg, "입니다.")
print("80점 이상 성적을 받은 학생은", highScoreStudents, "명입니다.")
```

튜플(tuple)

- 괄호 ( )로 표현
- 튜플은 요소를 추가하거나 삭제할 수 없는 리스트
- 리스트와 유사, 데이터를 변경할 필요가 없는 경우 사용
  - 인덱싱, 슬라이싱 이용할 수 있음
  - +, *, in, not in, len() 사용가능
  - 빈 튜플

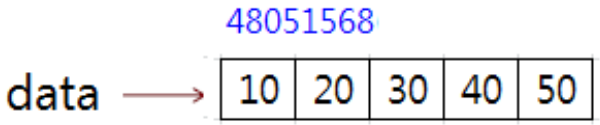
```
>>> t1 = ( )
>>> t2 = tuple()
>>> type(t1), type(t2)
(<class 'tuple'>, <class 'tuple'>)
```

```
>>> numbers = (1,2,3,4,5)
>>> colors = ('red', 'green', 'blue')
>>> t = numbers + colors
>>> t
(1, 2, 3, 4, 5, 'red', 'green', 'blue')
```

- immutable

```
>>> t1 = (1,2,3,4,5)
>>> print(t1[2])
3
>>> t1[0] = 100
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    t1[0] = 100
TypeError: 'tuple' object does not support item assignment
```

```
>>> data = (10, 20, 30, 40, 50)
>>> print(data)
(10, 20, 30, 40, 50)
>>> type(data)
<class 'tuple'>
>>> id(data)
48051568
>>>
```



튜플(tuple)

- 튜플 대입 연산 : 튜플은 여러 변수에 값을 동시에 할당할 수 있다.

```
>>> student1 = ('철수', 19, 'CS')
>>> name, age, major = student1
>>> name
'철수'
>>> age
19
>>> major
'CS'
```

```
>>> x, y, z = 1,2,3
>>> (x,y,z) = (1,2,3)
```

=

```
>>> x = 1
>>> y = 2
>>> z = 3
```

튜플(tuple)

- 튜플을 이용한 swap
- 리스트로부터 튜플 생성

```
>>> listA = [1,2,3]
>>> t = tuple(listA)
>>> t
(1, 2, 3)
```

```
>>> a = 10
>>> b = 20
>>> print(a, b)
10 20
>>> a, b = b, a
>>> print(a,b)
20 10
```

- 퓨플도 리스트와 같이 내부에 다른 튜플을 가질 수 있다.

```
>>> t = (1, 2, 'hello')
>>> u = t, ( 4,5,6,7)
>>> u
((1, 2, 'hello'), (4, 5, 6, 7))
```

Lab: 함수의 튜플 반환 예제

- 원의 넓이와 둘레를 동시에 반환하는 함수를 작성, 테스트해보자.

```
원의 반지름을 입력하시오: 10
원의 넓이는 314.1592653589793이고 원의 둘레는 62.831853071795864이다.
```

Solution

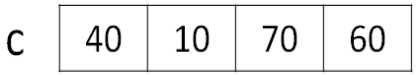
```
import math

def calCircle(r):
    # 반지름이 r인 원의 넓이와 둘레를 동시에 반환하는 함수(area, circum)
    area  = math.pi * r * r
    circum = 2 * math.pi * r
    return (area, circum)

radius = float(input("원의 반지름을 입력하시오: "))
(a, c) = calCircle(radius)
print("원의 넓이는 "+str(a)+"이고 원의 둘레는"+str(c)+"이다.")
```

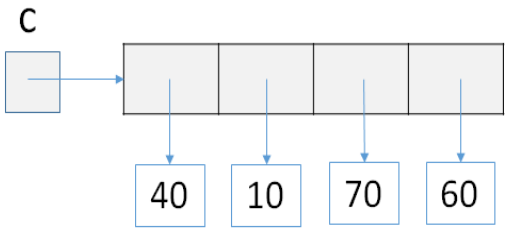
리스트 복사하기

파이썬의 리스트는 동적 배열(Dynamic Array) 개념으로 구현

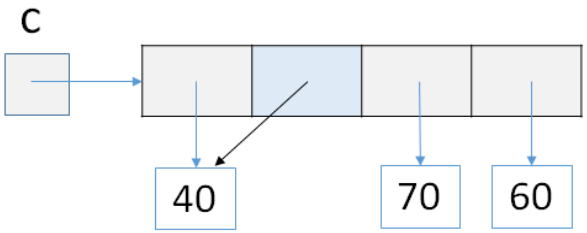
- 리스트 객체를 직접 저장하고 있지 않음 → 리스트 자체는 다른 곳에 저장되고 리스트의 참조값(메모리에서 리스트 객체의 위치)만 저장 ➔ **리스트는 값을 저장하는 것이아니라 값이 있는 주소를 저장하는 방식**

c = [40, 10, 70, 60]





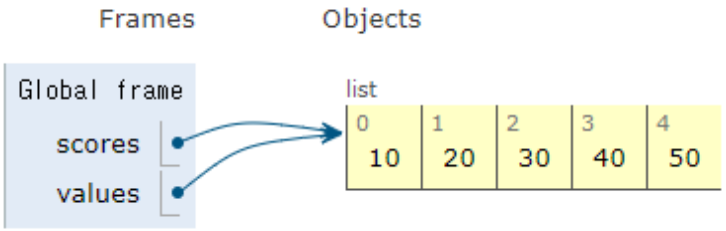
c[1] = 40이 수행된 후의 리스트 c

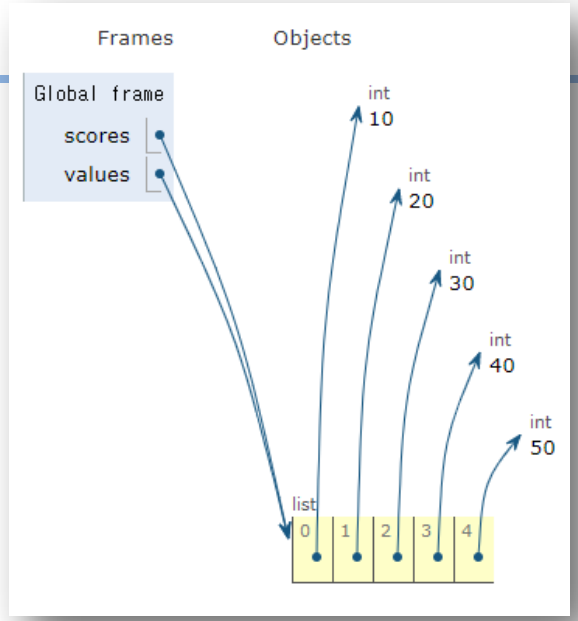



```
>>> c = [40, 10, 70, 60]
>>> id(c[0]), id(c[1]), id(c[2]), id(c[3])
(1492048672, 1492048192, 1492049152, 1492048992)
>>> c[1] = 40
>>> c
[40, 40, 70, 60]
>>> id(c[0]), id(c[1]), id(c[2]), id(c[3])
(1492048672, 1492048672, 1492049152, 1492048992)
>>>
```

리스트 복사하기

```
scores = [ 10, 20, 30, 40, 50 ]
values = scores
```





- 리스트는 복사되지 않고, scores와 values는 모두 동일한 리스트 객체를 가리킴
- values는 scores 리스트의 별칭과 마찬가지

➔ **얕은 복사(shallow copy)**

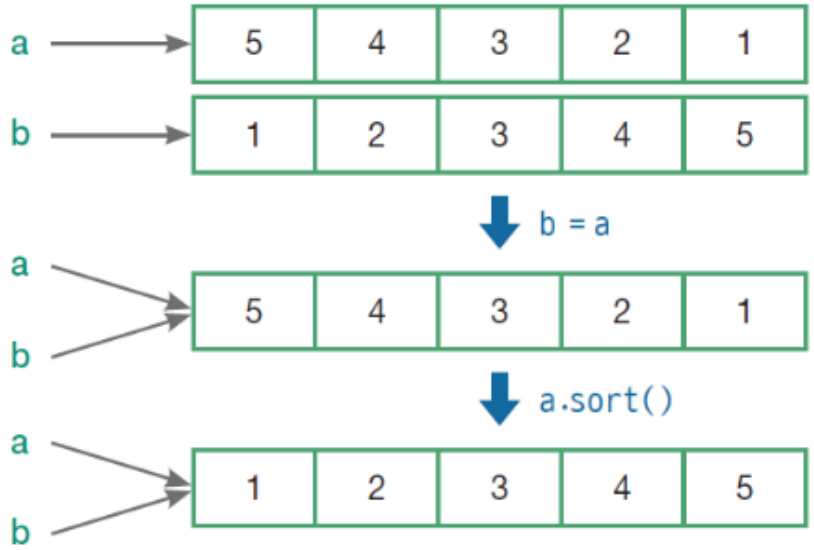
```
>>> values is scores
```

결과는?

yunyi96@gmail.com

리스트 복사하기

- 얕은 복사: 실제 메모리 블록이 복사되는 것이 아니라 단순 참조값의 복사

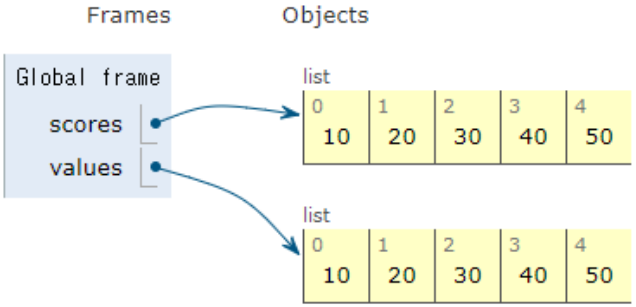



```
>>> a = [5,4,3,2,1]
>>> b = [1,2,3,4,5]
>>> id(a), id(b)
(46188944, 46189184)
>>> b=a
>>> a
[5, 4, 3, 2, 1]
>>> b
[5, 4, 3, 2, 1]
>>> id(a), id(b)
(46188944, 46188944)
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
>>> b
[1, 2, 3, 4, 5]
>>> id(a), id(b)
(46188944, 46188944)
>>>
```

깊은 복사

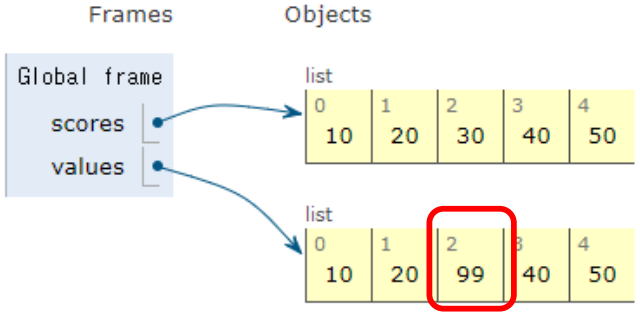
- 새로운 메모리 공간 생성
- 방법 1 : list() 내장 함수 사용

```
>>> scores = [ 10, 20, 30, 40, 50 ]
>>> values = list(scores)
>>> values[2]=99
>>> scores
[10, 20, 30, 40, 50]
>>> values
[10, 20, 99, 40, 50]
```

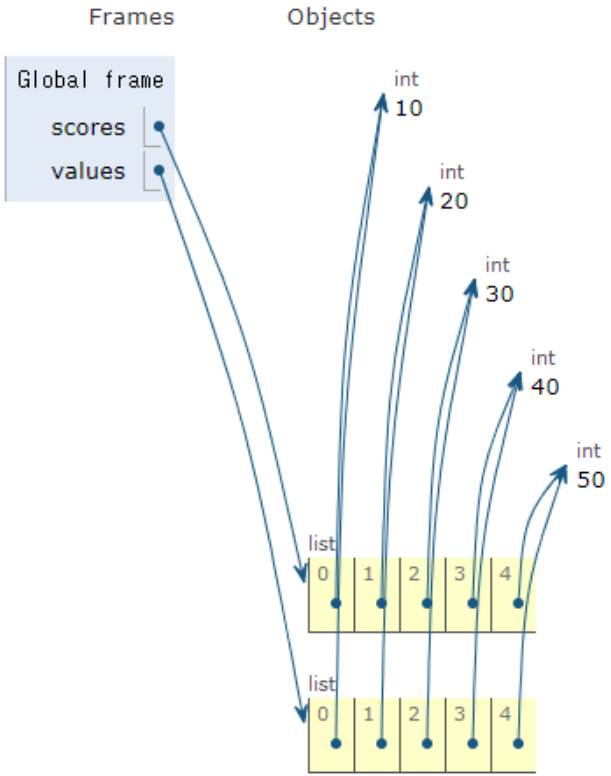



```
>>> values is scores
```

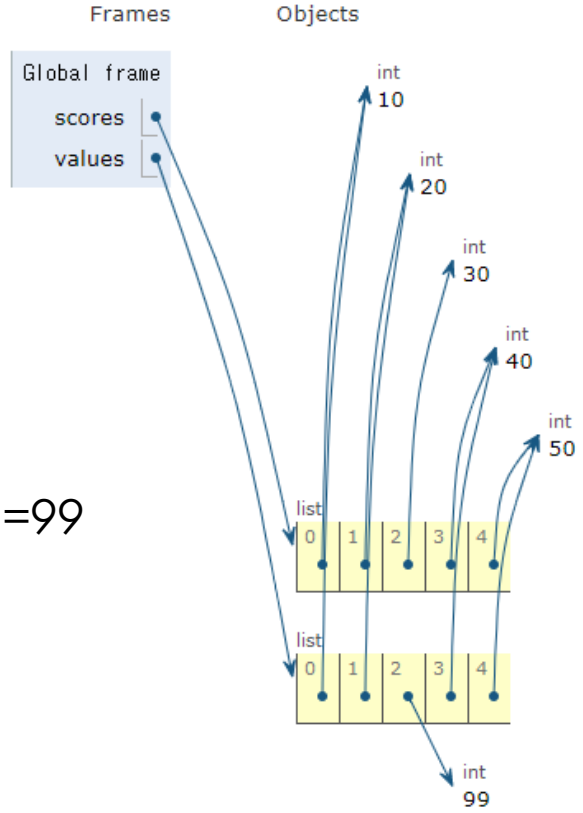
결과는?



깊은 복사



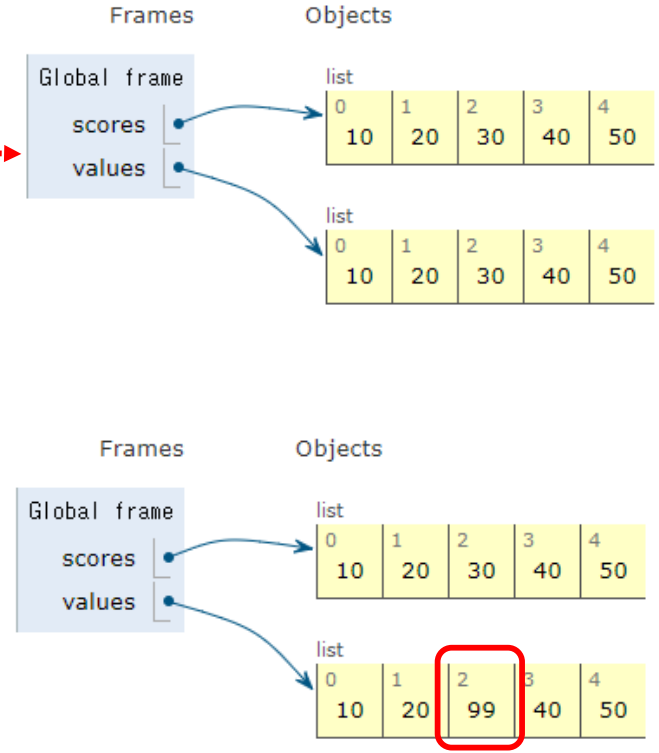
values[2]=99



깊은 복사

- 방법 2 : 리스트 자료형의 메소드 copy() 사용

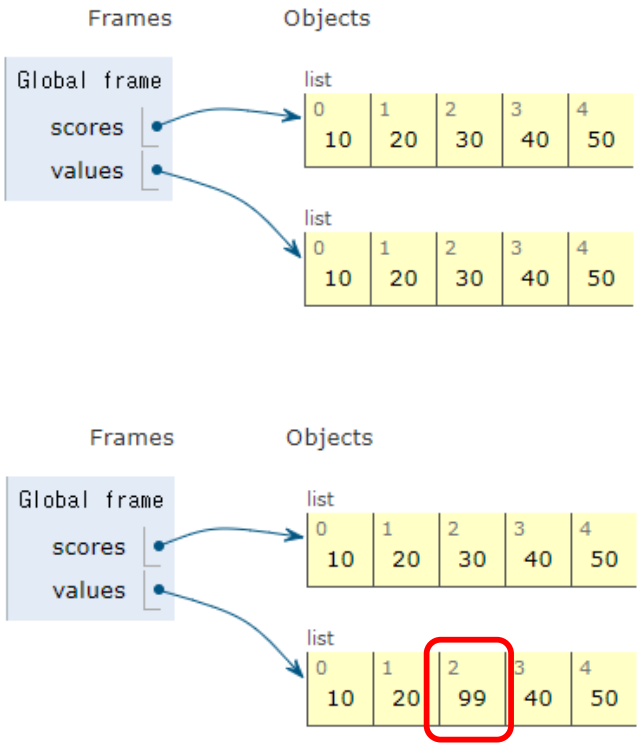
```
>>> scores = [ 10, 20, 30, 40, 50 ]
>>> values = scores.copy()
>>> values[2]=99
>>> scores
[10, 20, 30, 40, 50]
>>> values
[10, 20, 99, 40, 50]
```



깊은 복사

- 방법 3 : 리스트 슬라이스의 할당 사용 [:]

```
>>> scores = [ 10, 20, 30, 40, 50 ]
>>> values = scores[:]
>>> values[2]=99
>>> scores
[10, 20, 30, 40, 50]
>>> values
[10, 20, 99, 40, 50]
```



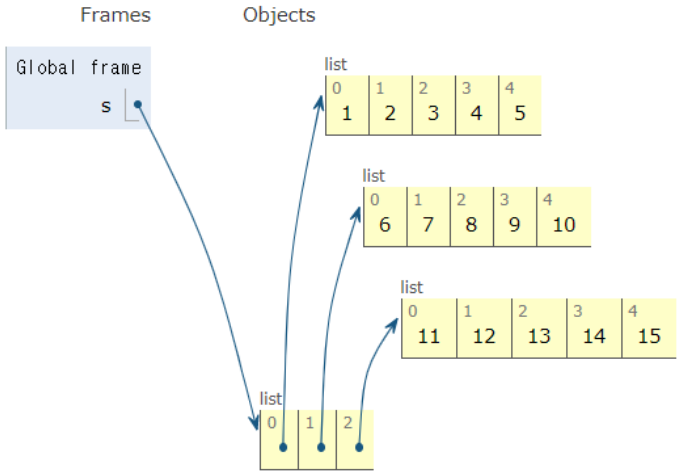
이차원 리스트

2차원 리스트

2차원 테이블 표시

학생	국어	영어	수학	과학	사회
김철수	1	2	3	4	5
김영희	6	7	8	9	10
최자영	11	12	13	14	15

```
# 2차원 리스트를 생성
s = [
        [ 1, 2, 3, 4, 5 ] ,
        [ 6, 7, 8, 9, 10 ],
        [11, 12, 13, 14, 15 ]
]
print(s)
```



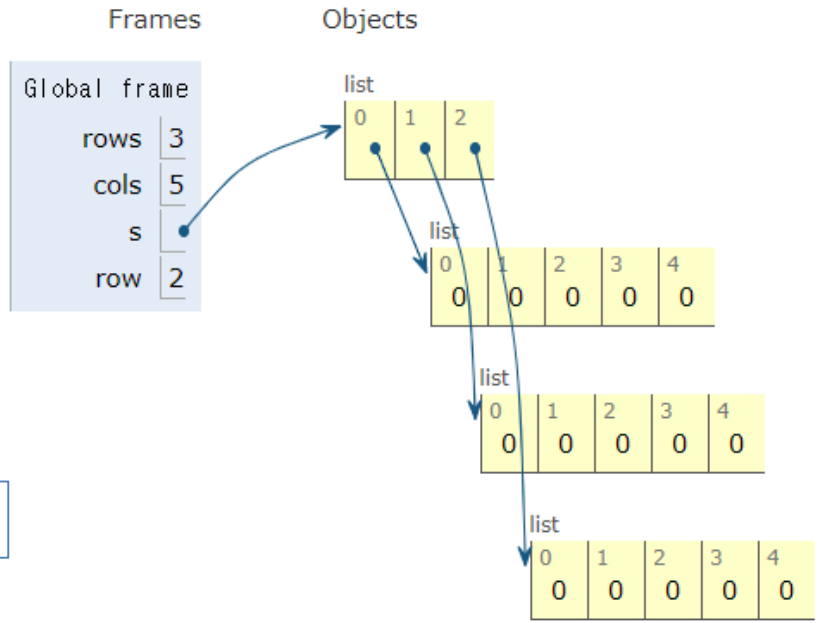
초기값이 미리 결정되어있어 정적으로 생성

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
```

동적으로 2차원 리스트 생성

```
# 동적으로 2차원 리스트를 생성한다.
rows = 3
cols = 5

s = [ ]
for row in range(rows):
        s += [[0]*cols]

print("s =", s)
```

```
s=[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```



2차원 리스트 요소 접근

- 2개의 인덱스 번호 지정 필요

```
s = [
        [ 1, 2, 3, 4, 5 ] ,
        [ 6, 7, 8, 9, 10 ],
        [11, 12, 13, 14, 15 ]
]

# 행과 열의 개수를 구한다.
rows = len(s)
cols = len(s[0])

for r in range(rows):
    for c in range(cols):
        print(s[r][c], end=",")
    print()
```

```
1,2,3,4,5,
6,7,8,9,10,
11,12,13,14,15,
```

Print output (drag lower right corner to resize)

```
1,2,3,4,5,
6,7,8,9,10,
11,12,13,14,15,
```