

# 학습 내용

1부. 프로그래밍 언어 기본

1장. 파이썬 개요 및 개발환경 구성

2장. 자료형과 연산자



3장. 데이터 구조

- 1. 리스트
- 2. 튜플
- 3. 딕셔너리
- 4. 셋
- 5. enumerate

4장. 제어문

5장. 함수

# 1절. 리스트

1장. 데이터 구조 > 1절. 리스트

- 리스트를 이용하면 여러 개 값을 저장
- 리스트를 만들려면 대괄호('['와 ']')를 이용
- 인덱스를 이용해 읽기와 쓰기를 지원
- 부분 데이터셋을 뽑아내는 슬라이싱(slicing)을 지원
- 파이썬의 인덱스는 0부터 시작

# 표 1. 파이썬에서 리스트 다루기

1절. 리스트

방법	설명
<code>listData = [ ]</code>	리스트를 만들어 줌
<code>len(listData)</code>	리스트의 항목의 수를 반환
<code>mix(listData), max(listData)</code>	리스트에서 가장 작은(min) 항목과 가장 큰(max) 항목을 반환
<code>listData[start:stop]</code>	리스트의 start 위치부터 stop 위치까지 부분 데이터를 추출(stop 위치의 항목은 포함 안 됨)
<code>listData.append(value)</code>	list에 value를 추가
<code>listData.clear()</code>	list의 모든 항목을 삭제
<code>listData.count(value)</code>	리스트에서 value의 개수를 반환
<code>listData.extend(newList)</code>	list에 newList를 추가
<code>+</code>	두 리스트를 연결함
<code>listData.index(value, position=0)</code>	position위치 이후에서 value의 값이 있는 인덱스를 반환
<code>listData.insert(index, value)</code>	list의 index위치에 value를 삽입
<code>listData.remove(value)</code>	리스트에서 해당 값을 삭제
<code>del listData[index]</code>	리스트에서 인덱스를 이용해 항목을 삭제
<code>listData.pop()</code>	리스트에서 가장 마지막 항목을 반환하고 삭제
<code>listData.reverse()</code>	리스트의 항목들의 순서를 반대로 함
<code>listData.sort(reverse=False)</code>	리스트의 항목들을 정렬. reverse 속성을 True로 하면 내림차순으로 정렬

## 1.2. 기본 정보 조회

1절. 리스트

- `len()` : 길이(항목의 수)
- `min()`, `max()` : 최댓값, 최솟값

```
1 numbers = [1,2,3,4,5]
2 numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

```
1 len(numbers), len(numbers_2d), len(numbers_2d[0])
```

(5, 4, 5)

```
1 min(numbers), max(numbers)
```

(1, 5)

```
1 max(numbers_2d)
```

[10, 20, 30, 40, 50]

# 1) +에 의한 연결, 2) \*에 의한 반복

1절. 리스트 > 1.3. 항목 추가하기

- + : 두 리스트를 연결
- \* : 리스트를 곱한 수만큼 반복

1	<code>numbers = [1,2,3,4,5]</code>
2	<code>numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]</code>

1	<code>new_numbers = numbers + numbers</code>
2	<code>new_numbers</code>

[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

1	<code>3*numbers</code>
---	------------------------

[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

# 3) 항목 추가하기, 4) 리스트 추가하기, 5) 중간에 삽입하기

1절. 리스트 > 1.3. 항목 추가하기

- `append()`
  - 단일 항목을 맨 뒤에 추가
  - 리스트를 `append` 하면 리스트가 항목으로 추가됨
- `extend()`
  - 리스트를 항목별로 맨 뒤에 추가
- `insert()`
  - 지정한 인덱스 위치에 삽입

1	<code>numbers.append(10)</code>
2	<code>numbers</code>

[1, 2, 3, 4, 5, 10]

1	<code>numbers.append([20, 30, 40, 50])</code>
2	<code>numbers</code>

[1, 2, 3, 4, 5, 10, [20, 30, 40, 50]]

1	<code>numbers = [1, 2, 3, 4, 5]</code>
2	<code>numbers.extend([10, 20, 30, 40, 50])</code>
3	<code>numbers</code>

[1, 2, 3, 4, 5, 10, 20, 30, 40, 50]

1	<code>numbers.insert(5, 100)</code>
2	<code>numbers</code>

[1, 2, 3, 4, 5, 100, 10, 20, 30, 40, 50]

# 1) 데이터 수 세기, 2) 항목의 위치 반환하기

1절. 리스트 > 1.4. 인덱싱

- `count()` : 리스트에서 데이터의 개수를 반환
- `index()` : 해당 항목의 위치 반환
- 항목을 찾지 못하면 에러 발생

```
1 numbers = [1,3,5,7,9]
2 numbers.index(5)
```

2

```
1 numbers = [1,3,5,7,9,1,2,3,4,5]
2 numbers.index(5,3)
```

9

```
1 numbers = [1,3,5,7,9,1,2,3,4,5]
2 numbers.index(6,3)
```

```
-----
ValueError                                Traceback
<ipython-input-24-f30a67067ef4> in <module>()
      1 numbers = [1,3,5,7,9,1,2,3,4,5]
----> 2 numbers.index(6,3)
```

**ValueError**: 6 is not in list

### 3) 인덱스를 이용한 직접 접근

1절. 리스트 > 1.4. 인덱싱

- [index] : 인덱스를 이용한 직접 접근

```
1 numbers = [1,3,5,7,9,1,2,3,4,5]
```

```
1 numbers[0], numbers[2]
```

(1, 5)

```
1 numbers[20]
```

---

```
IndexError                                Traceback (most recent call last)
<ipython-input-32-48ce5c7b3e82> in <module>()
----> 1 numbers[20]
```

**IndexError:** list index out of range



# 1) [ start : stop ]

1절. 리스트 > 1.5. 슬라이싱

## ● [ start : stop ] : start부터 stop까지 부분 리스트 추출

```
1 numbers = [1,2,3,4,5,6,7,8,9,10]
2 numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

```
1 numbers[2:4]
```

[3, 4]

```
1 numbers[2:]
```

[3, 4, 5, 6, 7, 8, 9, 10]

```
1 numbers[:4]
```

[1, 2, 3, 4]

```
1 numbers[:]
```

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

```
1 numbers[-5:-1]
```

[6, 7, 8, 9]

```
1 numbers[:-1]
```

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

```
1 numbers[-1:-5]
```

[]

```
1 numbers_2d[1:3]
```

[[10, 20, 30, 40, 50], [1, 3, 5, 7, 9]]

```
1 numbers_2d[1][1:4]
```

[20, 30, 40]

## 2) [ start : stop : step ]

1절. 리스트 > 1.5. 슬라이싱

- [ *start* : *stop* : *step* ] 형식은 매 *step* 번째 아이템을 추출

1	numbers = [1,2,3,4,5,6,7,8,9,10]
2	numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]

1	numbers[::2]
---	--------------

[1, 3, 5, 7, 9]

1	numbers[::-1]
---	---------------

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

1	numbers[-5::-1]
---	-----------------

[6, 5, 4, 3, 2, 1]

## 2) remove()

1절. 리스트 > 1.7. 삭제하기

- remove() : 해당 항목 삭제

```
1 numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
1 numbers.remove(3)
2 numbers
```

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

```
1 numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

```
1 numbers_2d.remove([1,2,3,4,5])
2 numbers_2d
```

[[10, 20, 30, 40, 50], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]

### 3) del

1절. 리스트 > 1.7. 삭제하기

- del : 지정한 위치 항목 삭제, 변수 삭제

```
1 numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
1 del numbers[3]  
2 numbers
```

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

```
1 numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
1 del numbers[3:8]  
2 numbers
```

[1, 2, 3, 9, 10]

```
1 del numbers
```

## 4) clear()

1절. 리스트 > 1.7. 삭제하기

- clear() : 모든 항목 삭제

1	<code>numbers = [1,2,3,4,5,6,7,8,9,10]</code>
---	---

1	<code>numbers.clear()</code>
2	<code>numbers</code>

`[]`

# 1.8. 정렬하기

1절. 리스트

- `sort()` : 정렬(`reverse=True` 속성을 이용하면 내림차순 정렬)
- `reverse()` : 역순으로 나열(내림차순 정렬이 아님)
- `[::-1]` : 역순으로 나열
- ❖ `sort()`와 `reverse()`는 원본데이터를 변경하지만 `[::-1]`은 역순으로 출력하고 원본데이터는 바꾸지 않습니다.

```
1 numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
1 numbers.sort()
2 numbers
```

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

```
1 numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
1 numbers.sort(reverse=True)
2 numbers
```

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

```
1 numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
1 numbers.reverse()
2 numbers
```

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

```
1 numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
1 new_numbers = numbers[::-1]
2 new_numbers
```

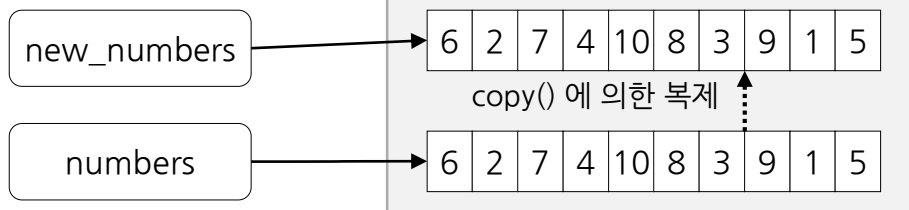
[5, 1, 9, 3, 8, 10, 4, 7, 2, 6]

# 리스트 복제

1절. 리스트

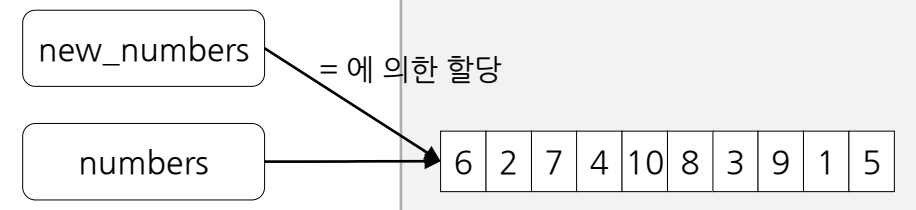
- `copy()` : 복제된 새로운 객체를 생성
- `=` : 주소를 복사해 같은 객체를 참조

```
1 numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

새로운 객체의  
주소를 가리킴

copy() 함수를 이용한 복사

기존 객체와 같은 객체를 참조함



할당연산자(=)를 이용한 복사

```
1 new_numbers = numbers.copy()
2 print(numbers)
3 print(new_numbers)
```

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

```
1 new_numbers = numbers
2 print(numbers)
3 print(new_numbers)
```

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

## 2.1. 튜플 만들기

2절. 튜플

- 튜플(tuple)은 소괄호('('와 ')')를 이용해 만듦
- 읽기 전용
  - 튜플은 속도가 빨라 수정이 필요 없는 배열 형태의 데이터 타입에 사용
  - 데이터를 수정할 수 없기 때문에 제공되는 함수가 많지 않음

방법	설명
<code>tupleData = ( )</code>	튜플을 만들어 줍니다.
<code>len(tupleData)</code>	튜플의 항목 수를 반환합니다.
<code>min(tuple), max(tuple)</code>	튜플에서 가장 작은 값(min)과 가장 큰 값(max)을 반환합니다.
<code>tupleData.count(value)</code>	튜플에서 value의 개수를 반환합니다.
<code>tupleData.index(value, position)</code>	position 위치 이후에서 value가 있는 인덱스를 반환합니다.

```
1 numbers1 = (1,2,3,4,5)
2 type(numbers1)
```

tuple

```
1 numbers2 = (1,)
2 type(numbers2)
```

tuple

```
1 numbers_2d = ((1,2,3), (4,5,6))
2 numbers_2d
```

((1, 2, 3), (4, 5, 6))



## 3.1. 딕셔너리 만들기

### 3절. 딕셔너리

- 키(key)와 값(value)의 쌍으로 구성된 자료 구조
- 딕셔너리를 만들기 위해서는 중괄호('{와 }')를 이용
- **키**는
  - 중복이 없이 **유일**한 값이어야 함
  - 리스트 타입을 사용할 수 없지만 **튜플 타입은 사용할 수 있음**
- 값은 중복이 가능하며 모든 타입이 가능
- 인덱스를 이용한 데이터의 참조는 지원 안함
- 딕셔너리 키 목록에 없는 데이터를 사용하여 참조하면 에러가 발생

방법	설명
<code>dictData = {"key": "value", ... }</code>	딕셔너리를 만들어 줍니다.
<code>len(dictData)</code>	딕셔너리의 항목의 수를 반환합니다.
<code>dictData.items()</code>	딕셔너리의 각 항목들을 (key, value) 형식의 튜플들로 반환합니다.
<code>dictData.keys()</code>	딕셔너리의 키(key)들을 반환합니다.
<code>dictData.values()</code>	딕셔너리의 값(value)들을 반환합니다.

## 4.1. 셋 만들기

4절. 셋

- 순서가 정해지지 않고, 중복을 허용하지 않는 집합
- 중괄호('{와 }')를 이용하여 정의

```
1 fruits = {"apple", "orange", "banana", "apple", "bear"}
```

```
1 fruits
```

```
{'apple', 'banana', 'bear', 'orange'}
```

```
1 fruits.add("apple")
```

```
1 fruits
```

```
{'apple', 'banana', 'bear', 'orange'}
```

```
1 fruits.add("mango")
```

```
1 fruits
```

```
{'apple', 'banana', 'bear', 'mango', 'orange'}
```

# enumerate

5절. enumerate

- 반복자(iterator) 또는 순서(sequence) 객체를 인수로 받음
- enumerate(iter) 이라고 사용했을 경우 이 함수는 iter 객체를 (0, iter[0]), (1, iter[1]), (2, iter[2]), ... 이런 형식으로 반환

```
1 fruits = ['watermelon', 'orange', 'mango', 'grape', 'banana', 'apple']
```

```
1 ▾ for index, value in enumerate(fruits):  
2     print("{}번째 데이터는 {}입니다.".format(index+1, value))  
3
```

1번째 데이터는 watermelon입니다.

2번째 데이터는 orange입니다.

3번째 데이터는 mango입니다.

4번째 데이터는 grape입니다.

5번째 데이터는 banana입니다.

6번째 데이터는 apple입니다.

# 연습문제

- 다음 리스트가 주어졌을 경우 요구사항대로 코드를 작성하시오

```
numbers = [1,2,3,4,5,6,7,8,9,10]
```

1. 숫자 100을 맨 뒤에 추가하세요
2. 다음 리스트를 numbers리스트 맨 뒤에 추가하세요

```
data = [200, 300, 400, 500]
```

3. 처음 다섯 개 숫자만 출력하세요
4. 리스트에서 짝수 번째 데이터만 출력하세요
5. 짝수번째 데이터를 모두 0으로 바꾸세요
6. 데이터를 역순으로 나열하세요(내림차순 정렬이 아닙니다)

# 연습문제

- 다음 딕셔너리 데이터가 주어졌을 경우 요구사항대로 코드를 작성하세요.

```
member = {'name':"홍길동","age":20,"address":"서울시 금천구", 'score':90}
```

7. address값을 출력하세요
8. score 를 출력하고 member 딕셔너리에서 삭제하세요.
9. address를 '서울시 서대문구'로 변경하세요.
10. member 딕셔너리 데이터의 값을 리스트로 출력하세요