20250410 데이터 구조

1절. 리스트(List)

1.1 리스트 만들기 및 접근하기

개념	설명	예시
생성 방법	`[]`를 이용해서 만들며, 여러 타입의 값을 하나의 변수에 저장/관리 가능	`numbers = [1, 2, 3, 4, 5]` `mixed = [1, "hello", 3.14, True]`
`list()` 함수	list() 함수를 이용한 리스트 생성	`empty = list()` `converted = list("hello")` → `['h', 'e', 'l', 'l', 'o']`

인덱스	각 요소의 위치를 나타내는 숫자
	• 0부터 시작
	• 맨 마지막 인덱스는 -1
		`numbers = [10, 20, 30, 40]`
		`numbers[0]` → `10`
		`numbers[-1]` → `40`

슬라이싱	`[from:stop:step]` 형식으로 부분 데이터셋 추출	`numbers = [10, 20, 30, 40, 50]`
		`numbers[1:4]` → `[20, 30, 40]`
		`numbers[::2]` → `[10, 30, 50]`

읽기/쓰기	인덱스와 슬라이싱을 이용해서 읽기/쓰기 지원	`numbers = [10, 20, 30]` • 읽기: `numbers[1]` → `20` • 쓰기: `numbers[1] = 25` → `[10, 25, 30]`
여러 타입 저장	리스트는 여러 타입의 데이터를 담을 수 있음	`mixed = [0, True, '2', 3, [4, 5, 6]]`

1.2 리스트 관련 함수와 특징

함수/개념	설명	예시

`range(from, to, by)`	from부터 to 앞까지 by씩 증가하는 범위의 정수 생성
	• from 생략 시: 0부터 시작
	• by 생략 시: 1씩 증가
		`list(range(5))` → `[0, 1, 2, 3, 4]`
		`list(range(2, 8))` → `[2, 3, 4, 5, 6, 7]`
		`list(range(1, 10, 2))` → `[1, 3, 5, 7, 9]`

`len(리스트)`	리스트의 요소 개수 반환	`len([1, 2, 3, 4])` → `4`

`enumerate(나열 가능한 자료)`	(인덱스, 값) 쌍의 집합으로 반환	`for idx, val in enumerate(['a', 'b', 'c']):`
		`    print(idx, val)`
		결과:
		`0 a`
		`1 b`
		`2 c`

1.3 리스트 요소 추가

연산자/메서드	설명	예시
`+`	두 리스트 연결	`[1, 2] + [3, 4]` → `[1, 2, 3, 4]`
`*`	리스트를 지정한 수만큼 반복	`[1, 2] * 3` → `[1, 2, 1, 2, 1, 2]`
`append(값)`	맨 뒤에 요소 추가	`numbers = [1, 2, 3]` `numbers.append(4)` → `[1, 2, 3, 4]`
`extend(리스트)`	맨 뒤에 리스트를 요소별로 추가	`numbers = [1, 2, 3]` `numbers.extend([4, 5])` → `[1, 2, 3, 4, 5]`
`insert(idx, 값)`	idx번째 인덱스 위치에 값 추가 (기존 데이터는 오른쪽으로 이동)	`numbers = [1, 2, 4]` `numbers.insert(2, 3)` → `[1, 2, 3, 4]`

1.4 인덱싱과 요소 갯수

메서드/연산	설명	예시
`변수[index]`	인덱스를 이용한 특정 위치 요소 접근	`fruits = ['apple', 'banana', 'orange']` `fruits[1]` → `'banana'`

`index(찾을 데이터)`	찾을 데이터가 있는 요소의 인덱스 반환
	찾을 데이터가 없을 경우 오류 발생
		`fruits = ['apple', 'banana', 'orange']`
		`fruits.index('banana')` → `1`
		`fruits.index('grape')` → `ValueError`

`count(찾을 데이터)`	리스트에서 찾을 데이터의 갯수 반환	`numbers = [1, 2, 2, 3, 2, 4]` `numbers.count(2)` → `3`

1.5 원하는 요소 추출

방법	설명	예시
리스트 슬라이싱	리스트에서 원하는 index의 데이터 추출 • [from:stop] 형식으로 사용 • from부터 stop 앞까지 step씩 증가하는 인덱스의 값을 추출 • from 생략 시 처음, stop 생략 시 끝, step 생략 시 1 • 음수 인덱스 사용 가능 • 인덱스 범위를 벗어나도 에러 발생 안 함	`numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]` • 기본: `numbers[2:7]` → `[2, 3, 4, 5, 6]` • step 지정: `numbers[1:8:2]` → `[1, 3, 5, 7]` • 음수 인덱스: `numbers[-3:]` → `[7, 8, 9]` • 역순: `numbers[::-1]` → `[9, 8, 7, ..., 0]`

리스트 컴프리헨션	리스트에서 원하는 데이터를 추출하는 간결한 방법
	• [표현식 for 변수 in 리스트 if 조건식]
		• 짝수만 추출:
		`[x for x in range(10) if x % 2 == 0]` → `[0, 2, 4, 6, 8]`
		• 제곱 값 생성:
		`[x**2 for x in range(5)]` → `[0, 1, 4, 9, 16]`
		• 문자열 처리:
		`[s.upper() for s in ['a', 'b', 'c']]` → `['A', 'B', 'C']`

1.6 요소 수정하기

방법	설명	예시
인덱스로 수정	특정 위치의 요소 값 변경	`numbers = [1, 2, 3, 4]` `numbers[1] = 20` → `[1, 20, 3, 4]`

슬라이싱 수정
(step 미포함)
	부분 리스트를 통째로 수정
	원본과 교체할 리스트의 길이가 달라도 됨
		`numbers = [1, 2, 3, 4, 5]`
		`numbers[1:4] = [20, 30]` → `[1, 20, 30, 5]`
		`numbers[1:3] = [200, 300, 400]` → `[1, 200, 300, 400, 5]`

슬라이싱 수정 (step 포함)	특정 간격의 요소들을 교체 **원본과 교체할 리스트의 길이가 같아야 함**	`numbers = [1, 2, 3, 4, 5, 6]` `numbers[::2] = [10, 30, 50]` → `[10, 2, 30, 4, 50, 6]`

1.7 요소 삭제

메서드/방법	설명	예시
`pop()`	가장 마지막 인덱스 요소를 반환하고 삭제	`numbers = [1, 2, 3, 4]` `last = numbers.pop()` → `last = 4, numbers = [1, 2, 3]`
`pop(idx)`	지정한 idx번 요소를 반환하고 삭제	`numbers = [1, 2, 3, 4]` `second = numbers.pop(1)` → `second = 2, numbers = [1, 3, 4]`
`remove(값)`	리스트에서 첫 번째로 등장하는 해당 값을 삭제 **값이 없으면 ValueError 발생**	`numbers = [1, 2, 2, 3, 4]` `numbers.remove(2)` → `[1, 2, 3, 4]` (첫 번째 2만 삭제)
`del 변수[idx]`	idx번째 요소를 삭제	`numbers = [1, 2, 3, 4]` `del numbers[1]` → `[1, 3, 4]`
`del 변수[from:to]`	슬라이싱 범위의 요소들을 삭제	`numbers = [1, 2, 3, 4, 5]` `del numbers[1:3]` → `[1, 4, 5]`
`clear()`	리스트의 모든 요소를 삭제	`numbers = [1, 2, 3, 4]` `numbers.clear()` → `[]` (빈 리스트)

1.8 정렬하기

메서드	설명	예시
`sort()`	리스트 자체를 정렬된 결과로 변경 **기본 정렬: 오름차순**	`numbers = [3, 1, 4, 2]` `numbers.sort()` → `numbers = [1, 2, 3, 4]`
`sort(reverse=True)`	내림차순으로 정렬	`numbers = [3, 1, 4, 2]` `numbers.sort(reverse=True)` → `numbers = [4, 3, 2, 1]`
`reverse()`	리스트 자체를 역순으로 변경 **(정렬은 하지 않고 순서만 뒤집음)**	`numbers = [3, 1, 4, 2]` `numbers.reverse()` → `numbers = [2, 4, 1, 3]`

1.9 복제하기

복제 방법	설명	예시

스칼라 데이터의 복제	단일 값(정수, 문자열 등)은 직접 할당으로 독립적인 복제 가능	`i = 10 # 원본`
		`copy_i = i # 복제본`
		`# 현재 두 변수는 같은 값을 가리킴`
		`copy_i = 99 # 복제본 값 변경`
		`# 원본 값은 그대로 유지 (i = 10)`

얕은 복사(할당)	리스트를 직접 할당하면 같은 객체를 참조
	하나를 수정하면 다른 하나도 변경됨
		`original = [1, 2, 3] # 원본`
		`shallow = original # 얕은 복사`
		`shallow[0] = 99 # 복사본 수정`
		`# 원본도 변경됨 (original = [99, 2, 3])`

깊은 복사(복제)	`copy()` 메서드를 사용하여 독립적인 복제본 생성
	하나를 수정해도 다른 하나는 영향 없음
		`original = [1, 2, 3] # 원본`
		`deep = original.copy() # 깊은 복사`
		`deep[0] = 99 # 복사본 수정`
		`# 원본은 변경 안됨 (original = [1, 2, 3])`

1.10 다차원 리스트

개념	설명	예시

2차원 리스트	리스트 안에 리스트가 있는 구조	`numbers_2d = [`
		`[90, 91, 92, 93],`
		`[85, 81, 82, 83],`
		`[71, 70, 72, 73]`
		`]`

접근 방법	이중 인덱스로 접근	`numbers_2d[0][2]` → `92`
크기 확인	행과 열의 개수	`len(numbers_2d)` → `3` (행 개수) `len(numbers_2d[0])` → `4` (열 개수)
`min()/max()`	2차원 리스트에서 적용 시: • min(): 0번 인덱스 요소가 가장 작은 리스트 반환 • max(): 0번 인덱스 요소가 가장 큰 리스트 반환	`min([[3, 5], [1, 9]])` → `[1, 9]` `max([[3, 5], [1, 9]])` → `[3, 5]`
문자 리스트 비교	문자 리스트일 경우 아스키코드 값으로 비교	`min(['abc', 'def', 'ABC'])` → `'ABC'` (A의 아스키코드: 65, a의 아스키코드: 97)

1.11 리스트 정렬과 집계 함수

함수	설명	예시

`sorted(리스트)`	리스트를 정렬하여 새로운 리스트 반환
	원본 리스트는 변경되지 않음
		`numbers = [1, 12, 3, 5, 4]`
		`sorted(numbers)` → `[1, 3, 4, 5, 12]`
		`numbers` → `[1, 12, 3, 5, 4]` (원본 유지)

`sorted(리스트, reverse=True)`	리스트를 내림차순으로 정렬	`sorted(numbers, reverse=True)` → `[12, 5, 4, 3, 1]`
`sum(리스트)`	리스트 요소들의 합계 반환	`sum([1, 12, 3, 5, 4])` → `25`

2절. 튜플(Tuple)

특징	설명	예시
기본 특성	• 리스트와 유사하지만 **읽기 전용** • 수정이 필요 없는 데이터에 사용 • 수정 불가하므로 데이터 수정, 추가, 삭제 불가 • 제공되는 함수가 많지 않음	`coordinates = (10, 20)` `person = ('John', 25, 'Developer')`

생성 방법	• 소괄호 `()` 사용
	• `tuple()` 함수 사용
	• 괄호 없이 콤마로만 구분해도 튜플로 인식
		`empty = ()`
		`single = (1,)`
		`numbers = (1, 2, 3, 4)`
		`from_list = tuple([1, 2, 3])`
		`implicit = 1, 2, 3  # 괄호 없이도 튜플`

접근 방법	리스트와 동일하게 인덱싱, 슬라이싱 사용 가능	`numbers = (1, 2, 3, 4, 5)`
		`numbers[0]` → `1`
		`numbers[1:3]` → `(2, 3)`

다중 할당	파이썬은 여러 값이 전달될 때 튜플 타입으로 전달	`a, b = (10, 20)` `x, y, z = 1, 2, 3  # 튜플 자동 할당`

지원 메서드	읽기 전용이므로 제한된 메서드만 제공	`numbers = (1, 2, 2, 3, 4)`
		`numbers.count(2)` → `2`
		`numbers.index(3)` → `3`

3절. 딕셔너리(Dictionary)

특징	설명	예시
기본 특성	• 중괄호 `{}`를 이용해서 키(key)-값(value) 쌍으로 구성된 자료 구조 • 키는 중복 불가(유일한 값만 가능) • 키에 리스트 등 변경 가능한 자료형은 사용 불가 • 값은 중복 가능 • 인덱스를 이용한 접근 지원 안 함	`person = {'name': 'John', 'age': 30, 'job': 'Developer'}`

생성 방법	• 중괄호 `{}` 사용
	• `dict()` 함수 사용
		`empty = {}`
		`scores = {'math': 90, 'science': 85}`
		`from_tuples = dict([('a', 1), ('b', 2)])`
		`with_keywords = dict(name='John', age=30)`

접근 방법	키를 이용해 값에 접근	`person = {'name': 'John', 'age': 30}` `person['name']` → `'John'`

4절. 셋(Set)

특징	설명	예시
기본 특성	• 중복을 허용하지 않는 집합 • 순서가 없음(인덱스 접근 불가) • 수학의 집합 개념과 동일	`unique_numbers = {1, 2, 3, 2, 1}` 출력: `{1, 2, 3}` (중복 제거됨)

생성 방법	• 중괄호 `{}` 사용 (빈 셋은 `set()`으로 생성해야 함)
	• `set()` 함수 사용
		`numbers = {1, 2, 3, 4}`
		`from_list = set([1, 2, 2, 3])`
		`empty = set()` (빈 중괄호 `{}`는 빈 딕셔너리임)

요소 추가	• `add()`: 기본자료형, 튜플 등 하나의 요소로 추가 • `update()`: 리스트, 튜플, 딕셔너리, 셋의 요소들을 개별적으로 추가 (딕셔너리 사용 시 key만 추가됨)	`numbers = {1, 2, 3}` • `numbers.add(4)` → `{1, 2, 3, 4}` • `numbers.add((5, 6))` → `{1, 2, 3, 4, (5, 6)}` (튜플은 하나의 요소) • `numbers.update([7, 8])` → `{1, 2, 3, 4, (5, 6), 7, 8}` • `numbers.update({"a": 1, "b": 2})` → `{1, 2, 3, 4, (5, 6), 7, 8, "a", "b"}` (키만 추가)

집합 연산	• `&`: 교집합
	• `|`: 합집합
	• `-`: 차집합
	• `^`: 대칭차집합
	
	• `intersection()`: 교집합
	• `union()`: 합집합
	• `difference()`: 차집합
		`A = {1, 2, 3, 4}`
		`B = {3, 4, 5, 6}`
		• `A & B` → `{3, 4}` (교집합)
		• `A

요소 삭제	• `remove()`: 요소 삭제 (없으면 오류)
	• `discard()`: 요소 삭제 (없어도 오류 없음)
	• `pop()`: 임의의 요소 반환 및 삭제
		`numbers = {1, 2, 3, 4}`
		`numbers.remove(3)` → `{1, 2, 4}`
		`numbers.discard(5)` → `{1, 2, 4}` (변화 없음)
		`x = numbers.pop()` → x는 셋의 임의 요소, 셋에서는 제거됨

5절. enumerate 함수

특징	설명	예시

기본 개념	• 반복자 또는 순서 객체로 반복문을 처리할 때 사용하는 함수
	• 리스트, 튜플, 셋, 딕셔너리 등을 for문에 사용할 때 인덱스와 값을 함께 얻을 수 있음
		`fruits = ['apple', 'banana', 'cherry']`
		`for idx, fruit in enumerate(fruits):`
		`    print(idx, fruit)`
		
		출력:
		`0 apple`
		`1 banana`
		`2 cherry`

인덱스 시작값 지정	• 두 번째 매개변수로 시작 인덱스 지정 가능	`fruits = ['apple', 'banana', 'cherry']`
		`for idx, fruit in enumerate(fruits, 1):`
		`    print(idx, fruit)`
		
		출력:
		`1 apple`
		`2 banana`
		`3 cherry`

다양한 자료형 활용	• 리스트, 튜플, 문자열 등 순회 가능한 자료형에 사용 가능	`# 문자열 활용`
		`for idx, char in enumerate("Python"):`
		`    print(idx, char)`
		
		`# 튜플 활용`
		`points = ((1, 2), (3, 4), (5, 6))`
		`for idx, (x, y) in enumerate(points):`
		`    print(f"Point {idx}: ({x}, {y})")`