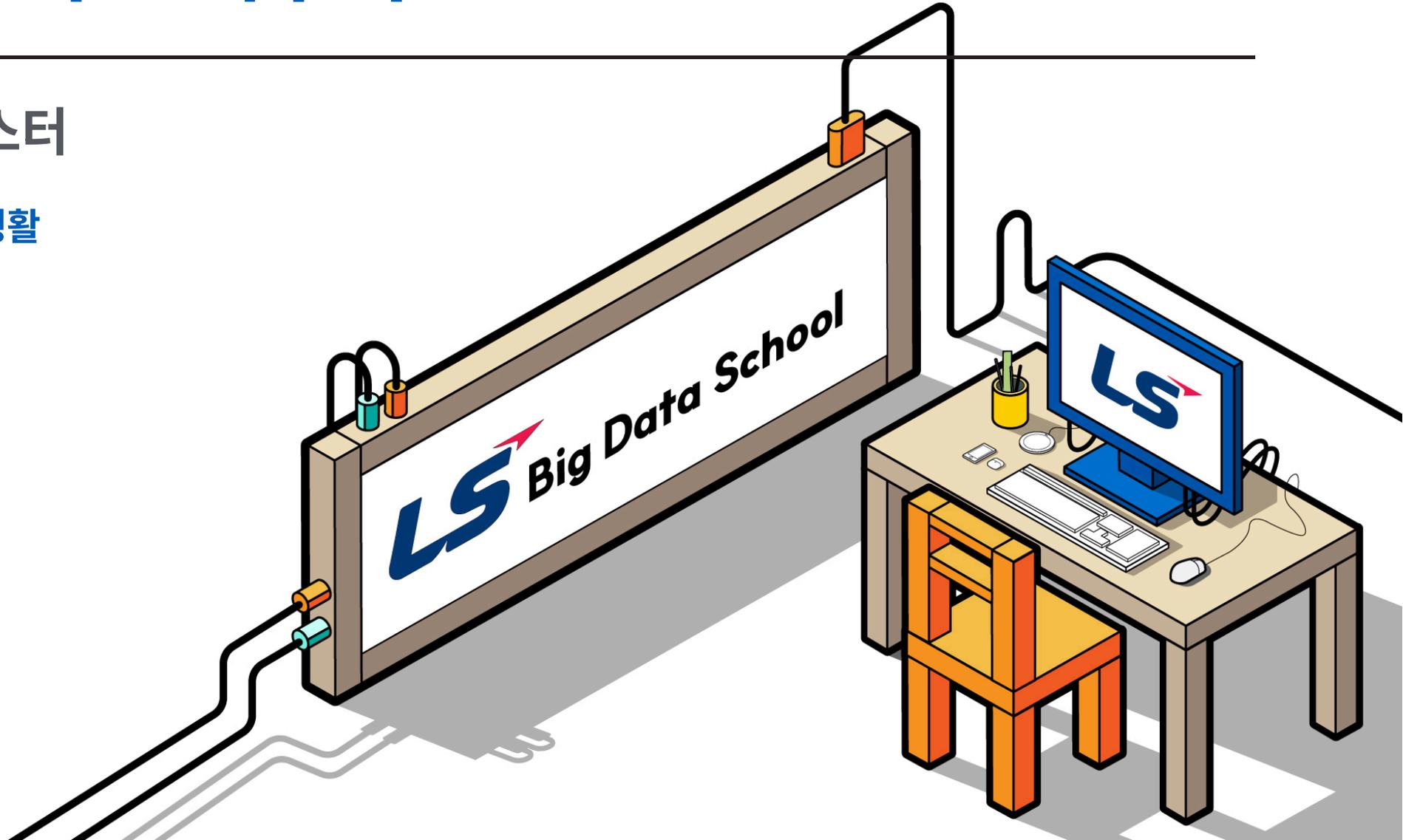


# 파이썬 기초 배우기

리스트 마스터

슬기로운통계생활



# 코스 훑어보기.

파이썬의 리스트에 대해 심화 학습합니다.



# 리스트 개요

리스트는 파이썬에서 가장 많이 사용되는 자료형 중 하나로, 여러 개의 데이터를 순서대로 저장할 수 있습니다. 리스트는 대괄호 `[]`로 감싸서 정의하며, 각 원소는 쉼표로 구분됩니다. 리스트의 특징은 다음과 같습니다:

1. **순서가 있다:** 리스트는 순서가 있는 자료형으로, 각 원소는 인덱스를 통해 접근할 수 있습니다.
2. **변경 가능:** 리스트의 원소는 변경할 수 있습니다.
3. **중복 허용:** 리스트는 동일한 값을 여러 번 가질 수 있습니다.
4. **다른 데이터 타입 허용:** 리스트는 여러 데이터 타입 정보를 동시에 가질 수 있습니다.



## 리스트 예제

```
fruits = ["apple", "banana", "cherry"]  
numbers = [1, 2, 3, 4, 5]  
mixed = [1, "apple", 3.5, True]  
  
print("과일 리스트:", fruits)
```

```
## 과일 리스트: ['apple', 'banana', 'cherry']
```

```
print("숫자 리스트:", numbers)
```

```
## 숫자 리스트: [1, 2, 3, 4, 5]
```

```
print("혼합 리스트:", mixed)
```

```
## 혼합 리스트: [1, 'apple', 3.5, True]
```



# 리스트 생성

## 빈 리스트 생성 방법

빈 리스트를 생성하는 방법은 두 가지가 있습니다. 첫 번째는 대괄호 `[]`를 사용하는 것이고, 두 번째는 `list()` 함수를 사용하는 것입니다.

```
# 빈 리스트 생성
empty_list1 = []
empty_list2 = list()

print("빈 리스트 1:", empty_list1)
```

```
## 빈 리스트 1: []
```

```
print("빈 리스트 2:", empty_list2)
```

```
## 빈 리스트 2: []
```



## 초기값을 가진 리스트 생성 방법

생성을 할 때 초기값 부여해서 리스트를 생성하는 방법입니다.

```
# 초기값을 가진 리스트 생성
numbers = [1, 2, 3, 4, 5]
range_list = list(range(5))

print("숫자 리스트:", numbers)
```

```
## 숫자 리스트: [1, 2, 3, 4, 5]
```

```
print("range() 함수로 생성한 리스트:", range_list)
```

```
## range() 함수로 생성한 리스트: [0, 1, 2, 3, 4]
```



## 다양한 타입의 리스트 생성 방법

리스트의 특징 중 하나는 다양한 타입의 정보를 동시에 갖는 리스트를 생성할 수 있다는 것 입니다.

```
# 다양한 타입의 리스트 생성
mixed_list = [1, "apple", 3.5, True]

print("혼합 리스트:", mixed_list)
```

```
## 혼합 리스트: [1, 'apple', 3.5, True]
```

위와 같이 리스트는 동일한 컨테이너 안에 여러 타입의 데이터를 포함할 수 있어, 매우 유연하게 사용할 수 있습니다.



# 리스트 접근 및 수정

## 인덱싱과 슬라이싱

리스트의 특정 원소에 접근하거나 부분 리스트를 추출하는 방법을 알아보시다. 리스트는 인덱스를 사용하여 원소에 접근할 수 있으며, 슬라이싱을 사용하여 부분 리스트를 추출할 수 있습니다.

```
# 리스트 접근 및 슬라이싱
fruits = ["apple", "banana", "cherry", "date"]
first_fruit = fruits[0]
last_fruit = fruits[-1]
print("첫 번째 과일:", first_fruit)
```

```
## 첫 번째 과일: apple
```

```
print("마지막 과일:", last_fruit)
```

```
## 마지막 과일: elderberry
```

```
# 슬라이싱
some_fruits = fruits[1:4]
print("일부 과일:", some_fruits)
```

```
## 일부 과일: ['banana', 'cherry', 'date']
```



## 원소 수정하기

리스트의 원소를 수정하는 방법은 다음과 같습니다.

```
# 리스트 원소 수정
fruits = ["apple", "banana", "cherry"]
fruits[1] = "blueberry"

print("수정된 과일 리스트:", fruits)
```

```
## 수정된 과일 리스트: ['apple', 'blueberry', 'cherry']
```



## 내포(Comprehension)

리스트 내포는 조건이나 반복을 사용하여 리스트를 생성하는 방법입니다. 예를 들어, 0부터 9까지의 숫자를 포함하는 리스트를 생성하려면 다음과 같이 작성할 수 있습니다.

```
# 리스트 내포(comprehension)
squares = [x**2 for x in range(10)]
print("제공 리스트:", squares)
```

```
## 제공 리스트: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



# 리스트 연산

## 연결하기

리스트 연결은 `+` 연산자를 사용하여 두 리스트를 하나의 리스트로 결합할 수 있습니다.

```
# 리스트 연결
list1 = [1, 2, 3]
list2 = [4, 5, 6]

combined_list = list1 + list2

print("연결된 리스트:", combined_list)
```

```
## 연결된 리스트: [1, 2, 3, 4, 5, 6]
```



## 반복하기

리스트 반복은 \* 연산자를 사용하여 리스트를 반복하여 새로운 리스트를 생성할 수 있습니다.

```
# 리스트 반복
numbers = [1, 2, 3]
repeated_list = numbers * 3

print("반복된 리스트:", repeated_list)
```

```
## 반복된 리스트: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```



## 반복하기

각 원소를 반복하기 위해서는 어떻게 해야할까요? 앞에서 배운 내포를 사용하면 좋습니다.

```
# 리스트 각 원소별 반복
numbers = [5, 2, 3]
repeated_list = [x for x in numbers for _ in range(3)]

print("각 원소별 반복된 리스트:", repeated_list)
```

```
## 각 원소별 반복된 리스트: [5, 5, 5, 2, 2, 2, 3, 3, 3]
```

리스트 내포의 기본 구조는 [표현식 for 항목 in 반복 가능한 객체]입니다. 이를 통해 리스트의 각 항목을 새로운 리스트의 항목으로 변환할 수 있습니다.



## 원소 체크

리스트에 특정 원소가 포함되어 있는지 확인하려면 `in` 연산자를 사용할 수 있습니다.

```
# 리스트 원소 포함 여부 확인
fruits = ["apple", "banana", "cherry"]

print("banana가 리스트에 포함되어 있나요?", "banana" in fruits)
```

```
## banana가 리스트에 포함되어 있나요? True
```

```
print("grape가 리스트에 포함되어 있나요?", "grape" in fruits)
```

```
## grape가 리스트에 포함되어 있나요? False
```



## 원소 체크

각 원소별로 체크하고 싶은 경우 바로 할 수는 없고, 넘파이 배열로 변경 후 체크하거나 내포 (comprehension)를 사용하여 체크할 수 있습니다.

```
[x == "banana" for x in fruits]
```

```
## [False, True, False]
```



# 리스트 메서드

리스트 메서드를 사용하면 리스트를 쉽게 조작할 수 있습니다. 다음은 주요 리스트 메서드와 그 사용 예제입니다.

메서드	설명
<code>append()</code>	리스트의 끝에 원소를 추가
<code>extend()</code>	리스트의 끝에 다른 리스트의 원소를 추가
<code>insert()</code>	리스트의 특정 위치에 원소를 삽입
<code>remove()</code>	리스트에서 특정 원소를 제거
<code>pop()</code>	리스트의 특정 위치에 있는 원소를 제거하고 반환
<code>clear()</code>	리스트의 모든 원소를 제거

메서드	설명
<code>index()</code>	리스트에서 특정 원소의 인덱스를 반환
<code>count()</code>	리스트에서 특정 원소의 개수를 반환
<code>sort()</code>	리스트의 원소를 정렬
<code>reverse()</code>	리스트의 원소 순서를 반대로 변경
<code>copy()</code>	리스트를 얇은 복사



## 원소 추가하기

`append()`의 경우 리스트 뒤에 원소를 추가할 경우 사용 할 수 있습니다.

```
# append() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
fruits.append("date")

print("append() 후 리스트:", fruits)
```

```
## append() 후 리스트: ['apple', 'banana', 'cherry', 'date']
```



## 원소 추가하기

뒤에 리스트를 사용하여 붙이고 싶은 경우에는 `extend()` 를 사용해줍니다.

```
# extend() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
fruits.extend(["date", "elderberry"])

print("extend() 후 리스트:", fruits)
```

```
## extend() 후 리스트: ['apple', 'banana', 'cherry', 'date', 'elderberry']
```



## 원소 추가하기

특정 위치에 원소를 넣고 싶은 경우에는 `insert()`를 사용해서 넣을 인덱스와 내용을 넣습니다.

```
# insert() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
fruits.insert(1, "blueberry")

print("insert() 후 리스트:", fruits)
```

```
## insert() 후 리스트: ['apple', 'blueberry', 'banana', 'cherry']
```



## 원소 제거하기

특정 원소를 제거하고 싶은 경우에는 `remove()`를 사용합니다.

```
# remove() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
fruits.remove("banana")

print("remove() 후 리스트:", fruits)
```

```
## remove() 후 리스트: ['apple', 'cherry']
```



## 원소 빼내기

리스트에 있는 원소들을 빼내오고 싶은 경우 `pop()` 을 사용할 수 있습니다. 입력값은 어떤 위치의 값을 빼올 것인지 인덱스 정보를 넣어주면 됩니다.

```
# pop() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
popped_fruit = fruits.pop(1)

print("pop() 후 리스트:", fruits)
```

```
## pop() 후 리스트: ['apple', 'cherry']
```

```
print("제거된 원소:", popped_fruit)
```

```
## 제거된 원소: banana
```



## 원소 빼내기

여러 개의 원소들을 빼내고오고 싶은 경우, 내포를 사용해서 빼내올 수 있습니다.

```
# 리스트 생성
fruits = ["apple", "banana", "cherry", "date", "elderberry"]

# 제거할 인덱스 리스트
indices_to_remove = [1, 2, 0] # banana와 cherry를 제거

# 제거할 인덱스 리스트를 내림차순으로 정렬
indices_to_remove.sort(reverse=True)

# 각 인덱스에 대해 pop() 호출
popped_fruits = [fruits.pop(x) for x in indices_to_remove]

print("pop() 후 리스트:", fruits)
```

```
## pop() 후 리스트: ['date', 'elderberry']
```



## 원소 빼내기

리스트 안의 원소들을 다 지우고, 리셋하고 싶은 경우, `clear()` 메서드를 사용합니다.

```
# clear() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
fruits.clear()

print("clear() 후 리스트:", fruits)
```

```
## clear() 후 리스트: []
```



## 원소 찾기

특정 원소의 위치를 반환하고 싶은 경우, `index()` 메서드를 사용합니다. 단, 가장 처음 나오는 값의 인덱스를 반환해줍니다.

```
# index() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
index_of_cherry = fruits.index("cherry")

print("cherry의 인덱스:", index_of_cherry)
```

```
## cherry의 인덱스: 2
```



## 원소 찾기

특정 원소에 해당하는 모든 인덱스를 찾고 싶은 경우는 내포를 사용해야합니다.

```
# 리스트 생성
fruits = ["cherry", "apple", "banana", "cherry"]

# 특정 요소의 모든 인덱스를 찾기
target = "cherry"
indices_of_cherry = [index for index, value in enumerate(fruits) if value == target]

print("cherry의 모든 인덱스:", indices_of_cherry)
```

```
## cherry의 모든 인덱스: [0, 3]
```



## 원소 세어보기

리스트 안에 특정 원소가 몇개나 들어있는지 세기 위해서는 `count()` 메서드를 사용합니다.

```
# count() 메서드 사용 예제
fruits = ["apple", "banana", "cherry", "apple"]
count_of_apple = fruits.count("apple")

print("apple의 개수:", count_of_apple)
```

```
## apple의 개수: 2
```



## 원소 정렬하기

리스트 안의 원소들을 `sort()` 메서드를 사용하여 크기순으로 정리할 수 있습니다.

```
# sort() 메서드 사용 예제
numbers = [3, 1, 4, 1, 5, 9]
numbers.sort()

print("정렬된 리스트:", numbers)
```

```
## 정렬된 리스트: [1, 1, 3, 4, 5, 9]
```



## 원소 정렬하기

리스트 안의 원소들을 `reverse()` 메서드를 사용하여 역순으로 정리 할 수 있습니다.

```
# reverse() 메서드 사용 예제
numbers = [3, 1, 4, 1, 5, 9]
numbers.reverse()

print("역순 리스트:", numbers)
```

```
## 역순 리스트: [9, 5, 1, 4, 1, 3]
```



## 리스트 복사하기

리스트를 복사하려면 `copy()` 메서드를 사용합니다.

```
# copy() 메서드 사용 예제
fruits = ["apple", "banana", "cherry"]
new_fruits = fruits.copy()

print("원본 리스트:", fruits)
```

```
## 원본 리스트: ['apple', 'banana', 'cherry']
```

```
print("복사된 리스트:", new_fruits)
```

```
## 복사된 리스트: ['apple', 'banana', 'cherry']
```



# 리스트의 활용

## 요약

주제	예제
리스트의 합계, 최소값, 최대값	<code>sum(list)</code> , <code>min(list)</code> , <code>max(list)</code>
리스트 내 중복 제거	<code>list(set(list))</code>
리스트 변환	<code>",".join(list)</code> , <code>str.split()</code>
다차원 리스트	<code>[[1, 2], [3, 4]]</code>

리스트의 원소를 합산하거나, 최소값, 최대값을 구할 수 있습니다.



## 리스트의 합계, 최소값, 최대값 구하기

```
# 리스트 합계, 최소값, 최대값
numbers = [1, 2, 3, 4, 5]

sum_numbers = sum(numbers)
min_number = min(numbers)
max_number = max(numbers)
```

```
print("리스트 합계:", sum_numbers)
```

```
## 리스트 합계: 15
```

```
print("리스트 최소값:", min_number)
```

```
## 리스트 최소값: 1
```

```
print("리스트 최대값:", max_number)
```

```
## 리스트 최대값: 5
```



## 리스트 내 중복 제거

리스트에서 중복된 원소를 제거하려면 `set`을 사용할 수 있습니다.

```
# 리스트 중복 제거
numbers = [1, 2, 2, 3, 4, 4, 5]
unique_numbers = list(set(numbers))

print("중복 제거된 리스트:", unique_numbers)
```

```
## 중복 제거된 리스트: [1, 2, 3, 4, 5]
```



## 리스트 변환

리스트를 문자열로 변환하거나, 문자열을 리스트로 변환할 수 있습니다.

```
# 리스트를 문자열로 변환
fruits = ["apple", "banana", "cherry"]
fruits_str = ", ".join(fruits)

print("문자열로 변환된 리스트:", fruits_str)
```

```
## 문자열로 변환된 리스트: apple, banana, cherry
```

`", ".join(fruits)`는 `fruits` 리스트의 각 요소 사이에 `", "`(쉼표와 공백) 문자열을 삽입하여 하나의 문자열로 만들어 줍니다.



## 리스트 변환

```
# 문자열을 리스트로 변환
fruits_str = "apple, banana, cherry"
fruits_list = fruits_str.split(", ")

print("리스트로 변환된 문자열:", fruits_list)
```

```
## 리스트로 변환된 문자열: ['apple', 'banana', 'cherry']
```

`fruits_str.split(", ")`는 문자열을 `","`(쉼표와 공백) 기준으로 분할하여 리스트로 만듭니다.



## 다차원 리스트

다차원 리스트는 리스트 안에 리스트를 포함하는 형태입니다.

```
# 다차원 리스트 생성
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("다차원 리스트:", matrix)
```

```
## 다차원 리스트: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
# 다차원 리스트의 특정 원소 접근
print("첫 번째 행, 두 번째 열의 값:", matrix[0][1])
```

```
## 첫 번째 행, 두 번째 열의 값: 2
```



# 문제 1: 리스트 생성 및 기본 연산

1. 다음 두 리스트를 생성하세요:
  - list1 = [1, 2, 3]
  - list2 = [4, 5, 6]
2. 두 리스트를 결합하여 새로운 리스트 combined를 만드세요.
3. 결합된 리스트 combined를 2번 반복한 리스트 repeated를 생성하세요.
4. 결합된 리스트의 합계를 출력하세요.

출력 예시:

- 결합된 리스트: [1, 2, 3, 4, 5, 6]
- 반복된 리스트: [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
- 리스트 합계: 21



# 문제 1: 리스트 생성 및 기본 연산 해답

```
# 리스트 생성
list1 = [1, 2, 3]
list2 = [4, 5, 6]

# 리스트 결합
combined = list1 + list2
print("결합된 리스트:", combined)
```

```
## 결합된 리스트: [1, 2, 3, 4, 5, 6]
```

```
# 리스트 반복
repeated = combined * 2
print("반복된 리스트:", repeated)
```

```
## 반복된 리스트: [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```



# 문제 1: 리스트 생성 및 기본 연산 해답

```
# 리스트 합계  
sum_combined = sum(combined)  
print("리스트 합계:", sum_combined)
```

```
## 리스트 합계: 21
```



## 문제 2: 리스트의 인덱싱과 슬라이싱

1. 다음 리스트를 생성하세요: `fruits = ["apple", "banana", "cherry", "date", "elderberry"]`
2. 리스트의 첫 번째 원소와 마지막 원소를 출력하세요.
3. 리스트에서 “banana”, “cherry”, “date”만 포함된 새로운 리스트 subset을 생성하세요.

출력 예시:

- 첫 번째 과일: `apple`
- 마지막 과일: `elderberry`
- 일부 과일: `['banana', 'cherry', 'date']`



## 문제 2: 리스트의 인덱싱과 슬라이싱 해답

```
fruits = ["apple", "banana", "cherry", "date", "elderberry"]  
print("첫 번째 과일:", fruits[0])
```

```
## 첫 번째 과일: apple
```

```
print("마지막 과일:", fruits[-1])
```

```
## 마지막 과일: elderberry
```

```
subset = fruits[1:4]  
print("일부 과일:", subset)
```

```
## 일부 과일: ['banana', 'cherry', 'date']
```



## 문제 3: 리스트 메서드 활용

1. 빈 리스트 `fruits`를 생성하세요.
2. `append()`를 사용하여 다음 과일을 순서대로 추가하세요: "apple", "banana", "cherry".
3. "banana"를 "blueberry"로 변경하세요.
4. 리스트에서 "cherry"를 제거하세요.
5. 리스트의 모든 원소를 제거하세요.

출력 예시:

- 리스트: ['apple', 'banana', 'cherry']
- 수정된 리스트: ['apple', 'blueberry', 'cherry']
- 제거된 리스트: ['apple', 'blueberry']
- 빈 리스트: []



# 문제 3: 리스트 메서드 활용 해답

```
# 빈 리스트 생성
```

```
fruits = []
```

```
# append()로 원소 추가
```

```
fruits.append("apple")
```

```
fruits.append("banana")
```

```
fruits.append("cherry")
```

```
print("리스트:", fruits)
```

```
## 리스트: ['apple', 'banana', 'cherry']
```

```
# 원소 수정
```

```
fruits[1] = "blueberry"
```

```
print("수정된 리스트:", fruits)
```

```
## 수정된 리스트: ['apple', 'blueberry', 'cherry']
```



# 문제 3: 리스트 메서드 활용 해답

```
# 원소 제거  
fruits.remove("cherry")  
print("제거된 리스트:", fruits)
```

```
## 제거된 리스트: ['apple', 'blueberry']
```

```
# 리스트 초기화  
fruits.clear()  
print("빈 리스트:", fruits)
```

```
## 빈 리스트: []
```



## 문제 4: 리스트 내포 활용

1. `range(1, 11)`에서 생성된 숫자 중 짝수의 제곱만 포함하는 리스트 `even_squares`를 생성하세요.
2. 생성된 리스트를 출력하세요.

출력 예시:

- 짝수의 제곱 리스트: [4, 16, 36, 64, 100]



# 문제 4: 리스트 내포 활용 해답

```
# 리스트 내포 사용
even_squares = [x**2 for x in range(1, 11) if x % 2 == 0]
print("짝수의 제곱 리스트:", even_squares)
```

```
## 짝수의 제곱 리스트: [4, 16, 36, 64, 100]
```



# 문제 5: 다차원 리스트

1. 다음과 같은 다차원 리스트를 생성하세요: `matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
2. `matrix`의 첫 번째 행과 두 번째 열의 값을 출력하세요.
3. `matrix`의 두 번째 행 전체를 출력하세요.

출력 예시:

- 첫 번째 행, 두 번째 열의 값: 2
- 두 번째 행: [4, 5, 6]



# 문제 5: 다차원 리스트 해답

```
# 다차원 리스트 생성
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# 특정 원소 접근
print("첫 번째 행, 두 번째 열의 값:", matrix[0][1])
```

```
## 첫 번째 행, 두 번째 열의 값: 2
```

```
# 특정 행 출력
print("두 번째 행:", matrix[1])
```

```
## 두 번째 행: [4, 5, 6]
```