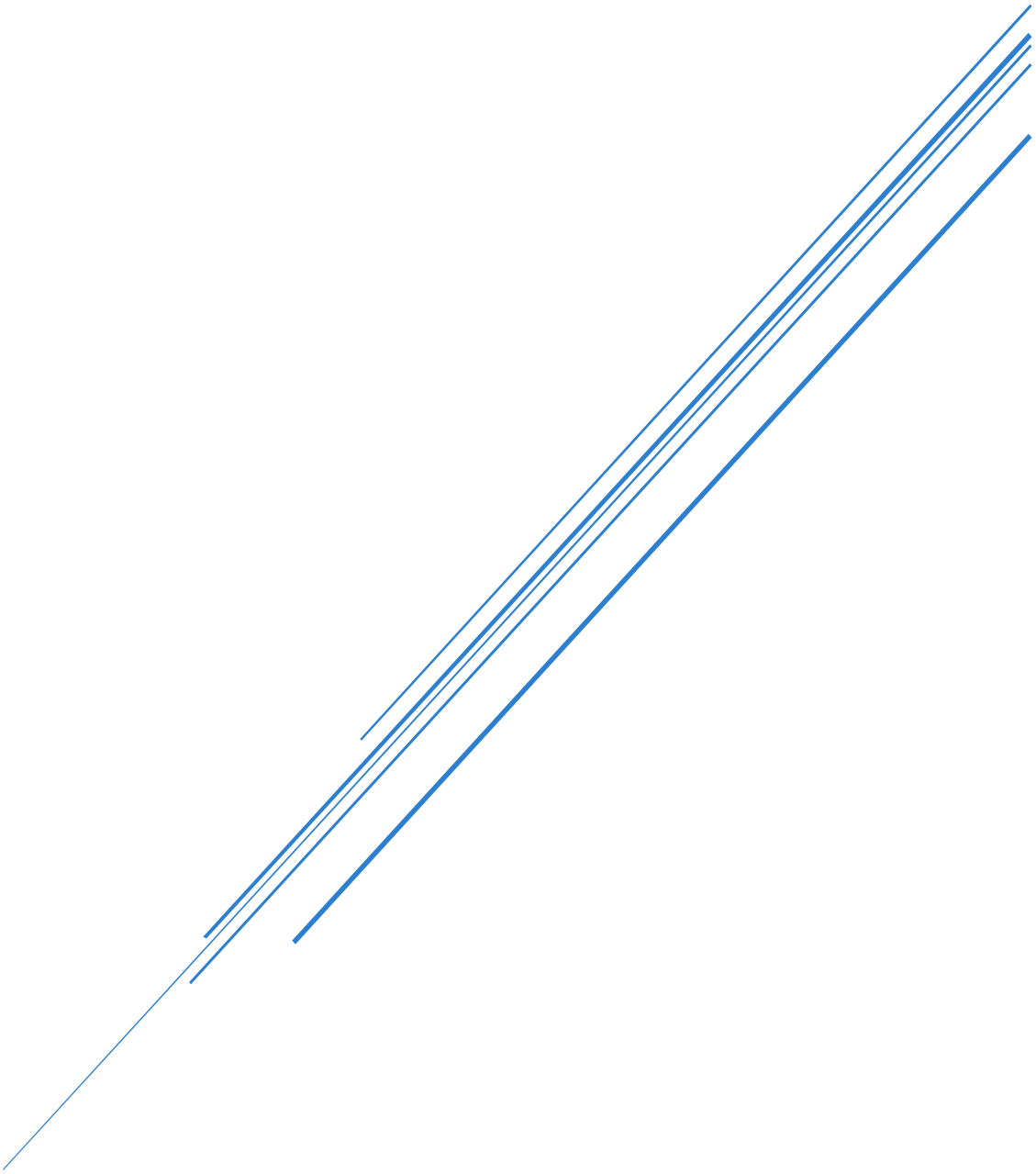
AI ESSENTIAL

실습가이드



24.10

목 차

01. Python

01-001 Python 코드

실습번호	01-001	실습명	Python 코드
실습 코드	 ``` # Python: 들여쓰기로 코드 블록 설정 def check_numbers(numbers): for number in numbers: if number % 2 == 0: print(f'{number} is even') else: print(f'{number} is odd') # 함수 호출 numbers = [1, 2, 3, 4, 5] check_numbers(numbers) ``` 		
실습 가이드	이 실습에서는 Python의 코드 구조에서 중요한 요소인 들여쓰기를 소개합니다. Python은 중괄호 대신 들여쓰기를 사용하여 코드 블록을 구분합니다. 들여쓰기는 가독성을 높이고, 코드의 논리적 흐름을 명확하게 나타내는 역할을 합니다. 함수 정의와 for문, if문을 작성하는 과정에서 들여쓰기를 어떻게 사용하는지 확인할 수 있습니다.		
실습 요약	 파이썬의 들여쓰기 규칙을 학습하고, 이를 이용해 짝수와 홀수를 구분하는 간단한 함수를 구현하여 코드 블록을 정의하는 방법을 이해합니다.		

01-002 Python: for문을 이용한 시퀀스 순회 구조

실습번호	01-002	실습명	Python: for문을 이용한 시퀀스 순회 구조
실습 코드	numbers = [0, 1, 2, 3, 4] # 리스트 생성 for number in numbers: # 리스트의 각 요소를 순차적으로 number 변수에 대입 print(number) # number의 값을 출력		
실습 가이드	이 실습에서는 Python의 `for`문을 사용하여 리스트의 각 요소를 순차적으로 변수에 대입하고 그 변수를 이용해 작업을 수행하는 기본적인 반복 구조를 학습합니다. `numbers` 리스트의 요소들이 `number` 변수에 하나씩 대입되며, `print()` 함수를 사용하여 각 값을 출력하는 방식입니다. 이 과정에서 Python의 `for`문이 반복적으로 데이터를 처리하는 방법을 이해할 수 있습니다.		
실습 요약	`for`문을 사용하여 리스트의 각 요소가 변수에 순차적으로 대입되는 원리를 학습하고, 이를 통해 각 요소를 출력하는 방법을 실습합니다.		

01-002 Python: for문을 이용한 시퀀스 순회 구조

실습번호	01-002	실습명	Python: for문을 이용한 시퀀스 순회 구조
실습 코드	numbers = [0, 1, 2, 3, 4] # 리스트 생성 for number in numbers: # 리스트의 각 요소를 순차적으로 number 변수에 대입 print(number) # number의 값을 출력		
실습 가이드	이 실습에서는 Python의 `for`문을 사용하여 리스트의 각 요소를 순차적으로 변수에 대입하고 그 변수를 이용해 작업을 수행하는 기본적인 반복 구조를 학습합니다. `numbers` 리스트의 요소들이 `number` 변수에 하나씩 대입되며, `print()` 함수를 사용하여 각 값을 출력하는 방식입니다. 이 과정에서 Python의 `for`문이 반복적으로 데이터를 처리하는 방법을 이해할 수 있습니다.		
실습 요약	`for`문을 사용하여 리스트의 각 요소가 변수에 순차적으로 대입되는 원리를 학습하고, 이를 통해 각 요소를 출력하는 방법을 실습합니다.		

01-003 Python: range() 함수를 이용한 반복

실습번호	01-003	실습명	Python: range() 함수를 이용한 반복
실습 코드	`# range() 함수를 사용한 반복문` `for number in range(5):  # 0부터 4까지의 숫자를 생성하여 number 변수에 대입` `    print(number)  # number의 값을 출력`		
실습 가이드	이 실습에서는 `range()` 함수를 사용하여 반복문을 구성하는 방법을 학습합니다. `range(5)`는 0부터 시작하여 5개의 숫자(0, 1, 2, 3, 4)를 생성합니다. `for`문에서 각 숫자가 `number` 변수에 순차적으로 대입되며, 이 변수의 값을 `print()` 함수를 통해 출력합니다. `range()` 함수는 특정 범위의 숫자를 반복할 때 매우 유용한 함수입니다.		
실습 요약	`range()` 함수를 사용하여 지정된 범위 내의 숫자들을 생성하고, 이를 `for`문에서 사용하여 반복적으로 작업을 수행하는 방법을 실습합니다.		

01-004 Python: 함수의 기본 구조

<table>
<tr><td>실습번호</td><td>01-004</td><td>실습명</td><td>Python: 함수의 기본 구조</td></tr>
<tr><td>실습
코드</td><td colspan="3">

```
# 함수 정의 예시
def 함수이름(매개변수1, 매개변수2):
    # 실행할 코드
    return 반환값  # 반환값은 선택 사항
```

</td></tr>
<tr><td>실습
가이드</td><td colspan="3">이 실습에서는 Python에서 함수를 정의하는 기본적인 구조를 학습합니다. 함수는 특정 작업을 수행하는 코드 블록이며, 매개변수를 통해 입력값을 받을 수 있습니다. `def` 키워드를 사용하여 함수를 정의하며, 함수의 이름과 매개변수 리스트를 지정합니다. 함수가 호출되면 지정된 코드가 실행되고, 필요한 경우 `return`을 사용하여 결과값을 반환할 수 있습니다. 반환값은 선택 사항입니다.</td></tr>
<tr><td>실습
요약</td><td colspan="3">Python에서 함수의 기본적인 정의 방법을 학습하고, 매개변수와 반환값의 개념을 이해합니다. 함수는 재사용 가능한 코드 블록으로, 프로그램 내에서 반복적으로 사용되는 작업을 캡슐화합니다.</td></tr>
</table>

01-005 Python: 함수 정의 및 호출

실습번호	01-005	실습명	Python: 함수 정의 및 호출
실습 코드	 `# 두 값을 더하는 함수 정의` `def add(a, b):  # a와 b라는 두 매개변수를 받음` `    return a + b  # 두 값을 더한 결과를 반환` `# 함수 호출 및 결과 출력` `result = add(5, 3)  # add 함수를 호출하여 5와 3을 더한 결과를 반환받음` `print(result)  # 결과값 출력 -> 8`		
실습 가이드	이 실습에서는 두 매개변수를 받아서 더한 값을 반환하는 간단한 함수를 정의하고 호출하는 방법을 학습합니다. `add` 함수는 두 매개변수 `a`와 `b`를 입력받아, 이 둘을 더한 값을 `return` 문을 통해 반환합니다. 이후, `add` 함수를 호출하여 `5`와 `3`을 인자로 전달한 후 결과값을 `print()`로 출력합니다.		
실습 요약	 매개변수를 사용하는 함수의 정의 방법과 함수 호출을 통해 값을 전달하고, 반환된 결과값을 출력하는 기본적인 함수 사용법을 실습합니다.		

01-006 Python: 기본 매개변수를 사용하는 함수

실습번호	01-006	실습명	Python: 기본 매개변수를 사용하는 함수
실습 코드	 `# 기본 매개변수를 사용하는 함수 정의` `def welcome(name='Guest'):  # 기본값이 'Guest'인 매개변수 name` `    return f'Welcome {name}!'  # name 값을 사용해 환영 메시지 반환` `# 함수 호출 및 결과 출력` `print(welcome())  # 매개변수를 전달하지 않으면 기본값 'Guest'가 사용됨 -> 'Welcome Guest!'` `print(welcome('to AI Essential'))  # 'to AI Essential' 값을 전달 -> 'Welcome to AI Essential!'`		
실습 가이드	이 실습에서는 함수 매개변수에 기본값을 설정하는 방법을 학습합니다. 함수의 매개변수에 기본값을 지정하면, 함수 호출 시 해당 매개변수를 생략해도 기본값이 사용됩니다. `welcome` 함수에서는 `name` 매개변수의 기본값을 'Guest'로 설정했으며, 이를 통해 매개변수가 전달되지 않으면 기본값으로 환영 메시지를 출력하고, 매개변수를 전달하면 그 값을 사용합니다.		
실습 요약	 기본 매개변수를 설정하여 함수 호출 시 유연하게 매개변수를 처리하는 방법을 실습합니다. 매개변수를 생략하거나 값을 전달할 때 함수의 동작이 어떻게 달라지는지 이해할 수 있습니다.		

01-007 Python: 모듈 생성 및 사용

실습번호	01-007	실습명	Python: 모듈 생성 및 사용
실습 코드	%%writefile mymodule.py # mymodule.py def welcome(name='Guest'): return f'Welcome {name}!' # 기본값을 사용하여 환영 메시지 반환 def add(a, b): return a + b # 두 값을 더한 결과를 반환		
실습 가이드	이 실습에서는 Python에서 모듈을 생성하고 사용하는 방법을 학습합니다. `mymodule.py`라는 모듈을 생성하여 그 안에 두 개의 함수를 정의합니다. 모듈은 여러 함수나 클래스를 파일로 묶어서 재사용 가능한 코드를 만드는 방식입니다. 이때, `%%writefile` 매직 명령어는 Colab 환경에서 Python 코드를 파일로 저장할 때 사용됩니다. 이 명령어는 셀 안의 코드를 지정한 파일명으로 저장하는 역할을 합니다.		
실습 요약	`%%writefile` 명령어를 사용하여 Python 모듈을 생성하고, 모듈 안에 여러 함수를 정의하는 방법을 학습합니다. 모듈은 코드의 재사용성을 높이며, 별도의 파일로 나누어 관리할 수 있습니다.		

01-008 Python: 모듈을 사용하여 함수 호출하기

실습번호	01-008	실습명	Python: 모듈을 사용하여 함수 호출하기
실습 코드	 `# mymodule 모듈을 import` `import mymodule` `# 모듈 내 함수 호출` `print(mymodule.welcome('to AI Essential'))  # 'Welcome to AI Essential!' 출력` `print(mymodule.add(3, 5))  # 3 + 5 = 8 출력`		
실습 가이드	이 실습에서는 Python 모듈을 `import`하여 그 안에 정의된 함수를 사용하는 방법을 학습합니다. 앞서 만든 `mymodule.py` 파일을 불러와서 `welcome` 함수와 `add` 함수를 호출합니다. Python에서는 모듈을 사용하여 코드를 재사용할 수 있으며, 모듈은 다른 파일에 정의된 함수나 클래스를 쉽게 불러와 사용할 수 있습니다.		
실습 요약	 `import` 키워드를 사용하여 모듈을 불러오고, 모듈 안에 정의된 함수를 호출하는 방법을 실습합니다. 모듈을 사용함으로써 코드의 재사용성과 유지보수가 개선됩니다.		

01-009 Python: 모듈을 별칭으로 불러오기

실습번호	01-009	실습명	Python: 모듈을 별칭으로 불러오기
실습 코드	```# mymodule 모듈을 별칭 'mm'으로 import import mymodule as mm # 모듈의 함수를 별칭을 사용하여 호출 print(mm.welcome('to AI Essential')) # 'Welcome to AI Essential!' 출력 print(mm.add(3, 5)) # 3 + 5 = 8 출력```		
실습 가이드	이 실습에서는 모듈을 `import`할 때 별칭을 사용하는 방법을 학습합니다. `mymodule` 모듈을 `mm`이라는 짧은 이름으로 불러와 사용함으로써, 코드 작성 시 더 간결하게 모듈의 함수를 호출할 수 있습니다. `as` 키워드를 사용하여 모듈에 별칭을 지정할 수 있으며, 이렇게 하면 모듈 이름을 짧게 줄여 사용할 수 있습니다.		
실습 요약	`import 모듈 as 별칭` 구문을 사용하여 모듈을 불러오고, 지정한 별칭으로 모듈의 함수를 호출하는 방법을 실습합니다. 이 방법은 긴 모듈 이름을 줄여서 간편하게 사용할 수 있는 장점이 있습니다.		

01-010 Python: 모듈에서 특정 함수만 불러오기

실습번호	01-010	실습명	Python: 모듈에서 특정 함수만 불러오기
실습 코드			

```
# mymodule 모듈에서 특정 함수만 불러오기
from mymodule import welcome, add

# welcome 함수 호출
print(welcome('to AI Essential'))  # 'Welcome to AI Essential!' 출력

# add 함수 호출
print(add(3, 5))  # 3 + 5 = 8 출력
```

실습 가이드	이 실습에서는 `from 모듈 import 함수` 구문을 사용하여 모듈에서 특정 함수만 선택적으로 불러오는 방법을 학습합니다. `mymodule` 모듈에서 `welcome` 함수와 `add` 함수만을 가져와서 사용할 수 있습니다. 이를 통해 불필요한 함수나 모듈 전체를 불러오지 않고, 필요한 함수만을 선택적으로 사용할 수 있습니다.
실습 요약	`from 모듈 import 함수` 구문을 사용하여 특정 함수만을 불러와 사용하는 방법을 실습합니다. 모듈 전체를 불러오는 대신 필요한 함수만 가져와서 코드의 효율성을 높일 수 있습니다.

02. PyTorch Tensor

02-001 PyTorch와 NumPy 라이브러리 임포트

실습번호	02-001	실습명	PyTorch와 NumPy 라이브러리 임포트
실습 코드	`import torch  # PyTorch 라이브러리 임포트` `import numpy as np  # NumPy 라이브러리 임포트`		
실습 가이드	이 실습은 PyTorch와 NumPy 라이브러리를 사용하는 프로젝트의 기본 설정 과정입니다. PyTorch는 딥러닝을 위한 주요 라이브러리이며, NumPy는 과학 계산을 위한 중요한 도구입니다. 이 단계에서는 두 라이브러리를 임포트하여 이후의 계산 및 작업에서 사용할 수 있도록 준비합니다.		
실습 요약	PyTorch와 NumPy 라이브러리를 임포트하여 딥러닝과 과학 계산을 위한 환경을 설정하는 과정입니다.		

02-002 1D 텐서 생성 및 출력

실습번호	02-002	실습명	1D 텐서 생성 및 출력
실습 코드	`tensor_1d = torch.tensor([1.0, 2.0, 3.0])  # 1차원 텐서를 생성 (값: 1.0, 2.0, 3.0)` `tensor_1d  # 생성된 텐서를 출력 -> tensor([1., 2., 3.])`		
실습 가이드	이 실습에서는 1차원 텐서를 생성하고 출력하는 방법을 학습합니다. torch.tensor() 함수를 사용하여 텐서를 생성하며, 이 함수는 다양한 형태의 텐서를 쉽게 만들 수 있습니다. 생성한 텐서는 바로 출력할 수 있습니다.		
실습 요약	1차원 텐서를 생성하고 이를 출력하는 방법을 실습합니다. 텐서는 딥러닝에서 데이터를 표현하는 기본 단위입니다.		

02-003 2D 텐서 생성 및 출력

실습번호	02-003	실습명	2D 텐서 생성 및 출력
실습 코드	`tensor_2d = torch.tensor([[1.0, 2.0],` `[3.0, 4.0]]) # 2차원 텐서를 생성 (값: [[1.0, 2.0], [3.0, 4.0]])` `tensor_2d # 생성된 텐서를 출력 -> tensor([[1., 2.], [3., 4.]])`		
실습 가이드	이 실습에서는 2차원 텐서를 생성하고 출력하는 방법을 학습합니다. 2D 텐서는 행렬을 나타내며, 딥러닝 및 과학적 계산에서 자주 사용됩니다. torch.tensor() 함수를 사용하여 2차원 텐서를 생성하고 이를 출력할 수 있습니다.		
실습 요약	2차원 텐서를 생성하고 출력하는 과정을 실습합니다. 2D 텐서는 딥러닝에서 데이터나 파라미터를 행렬 형태로 저장하는 데 유용합니다.		

02-004 텐서를 GPU로 이동

실습번호	02-004	실습명	텐서를 GPU로 이동
실습 코드	 # 텐서를 GPU로 이동 if torch.cuda.is_available(): # GPU 사용 가능 여부 확인 tensor_1d = tensor_1d.to('cuda') # 1D 텐서 GPU로 이동 tensor_2d = tensor_2d.to('cuda') # 2D 텐서 GPU로 이동		
실습 가이드	이 실습에서는 텐서를 GPU로 이동하는 방법을 다룹니다. 딥러닝 모델 학습 시 GPU를 사용하면 연산 속도가 크게 향상될 수 있습니다. 먼저 torch.cuda.is_available()로 GPU 사용 가능 여부를 확인한 후, 사용 가능하다면 텐서를 GPU 메모리로 이동시킵니다.		
실습 요약	이 실습은 텐서를 GPU로 이동하여 빠른 연산을 할 수 있도록 하는 방법을 보여줍니다. 이를 통해 GPU가 있는 환경에서의 효율적인 연산 처리가 가능합니다.		

02-005 NumPy 배열로부터 텐서 생성 및 dtype 지정

실습번호	02-005	실습명	NumPy 배열로부터 텐서 생성 및 dtype 지정
실습 코드	`numpy_array = np.array([1, 2, 3])  # NumPy 배열 생성` `tensor_from_numpy = torch.tensor(numpy_array, dtype=torch.float64)  # NumPy 배열로부터 텐서 생성 및 dtype을 float64로 지정` `tensor_from_numpy  # 텐서 출력 -> tensor([1., 2., 3.], dtype=torch.float64)`		
실습 가이드	이 실습에서는 NumPy 배열로부터 PyTorch 텐서를 생성하고, 생성 시 텐서의 데이터 타입(dtype)을 명시적으로 지정하는 방법을 학습합니다. dtype을 설정하면 텐서의 데이터 타입을 지정할 수 있으며, 이는 연산의 정확성과 성능에 영향을 미칩니다. 이 예제에서는 float64로 지정하여 높은 정밀도의 실수를 처리할 수 있도록 합니다.		
실습 요약	NumPy 배열을 사용해 PyTorch 텐서를 생성하고, dtype을 설정하여 텐서의 데이터 타입을 명시적으로 지정하는 방법을 실습합니다. 이 과정은 데이터 타입을 정확히 제어해야 할 때 유용합니다.		

02-006 균일하게 분포된 난수로 초기화된 텐서 생성

실습번호	02-006	실습명	균일하게 분포된 난수로 초기화된 텐서 생성
실습 코드	`# 2x3의 [0, 1) 범위에서 균일하게 분포된 난수로 초기화된 Tensor 생성` `rand_tensor = torch.rand((2, 3))  # 2x3 크기의 랜덤 텐서 생성` `rand_tensor  # 생성된 텐서를 출력 -> tensor([[random_values], [random_values]])`		
실습 가이드	이 실습에서는 2x3 크기의 텐서를 [0, 1) 범위에서 균일하게 분포된 난수로 초기화하는 방법을 학습합니다. torch.rand() 함수는 주어진 크기만큼의 텐서를 생성하며, 값은 0과 1 사이에서 랜덤하게 할당됩니다. 이 방법은 모델의 가중치 초기화 등에서 자주 사용됩니다.		
실습 요약	torch.rand()를 사용하여 2x3 크기의 랜덤 텐서를 생성하는 과정을 실습합니다. 생성된 값들은 [0, 1) 범위에서 균일하게 분포된 난수입니다		

02-007 표준 정규 분포를 따르는 난수로 초기화된 텐서 생성

실습번호	02-007	실습명	표준 정규 분포를 따르는 난수로 초기화된 텐서 생성
실습 코드	# 2x2의 표준 정규 분포를 따르는 난수로 초기화된 Tensor 생성 randn_tensor = torch.randn((2, 2)) # 2x2 크기의 표준 정규 분포를 따르는 랜덤 텐서 생성 randn_tensor # 생성된 텐서를 출력 -> tensor([[random_values], [random_values]])		
실습 가이드	이 실습에서는 2x2 크기의 텐서를 표준 정규 분포(N(0, 1))에 따라 난수로 초기화하는 방법을 학습합니다. torch.randn() 함수는 주어진 크기만큼의 텐서를 생성하며, 값은 평균이 0, 표준편차가 1인 정규 분포를 따릅니다. 이 방법은 모델 가중치 초기화에서 자주 사용됩니다.		
실습 요약	torch.randn()를 사용하여 2x2 크기의 표준 정규 분포를 따르는 랜덤 텐서를 생성하는 과정을 실습합니다. 이 과정은 딥러닝에서 가중치 초기화에 자주 활용됩니다.		

02-008 데이터 타입이 지정된 2x2 텐서 생성

실습번호	02-008	실습명	데이터 타입이 지정된 2x2 텐서 생성
실습 코드	`# 2x2 텐서 생성, 데이터 타입을 float32로 지정` `tensor_with_dtype = torch.zeros((2, 2), dtype=torch.float32)  # 2x2 크기의 텐서를 생성하고, 데이터 타입을 float32로 지정` `tensor_with_dtype  # 생성된 텐서를 출력 -> tensor([[0., 0.], [0., 0.]], dtype=torch.float32)`		
실습 가이드	이 실습에서는 2x2 크기의 텐서를 생성하고, 데이터 타입을 명시적으로 float32로 지정하는 방법을 학습합니다. torch.zeros() 함수를 사용하여 0으로 초기화된 텐서를 만들고, dtype 매개변수를 통해 데이터 타입을 설정할 수 있습니다. float32는 딥러닝 모델에서 자주 사용하는 데이터 타입입니다.		
실습 요약	2x2 크기의 텐서를 생성하면서 데이터 타입을 float32로 명시적으로 지정하는 방법을 실습합니다. 이 과정은 텐서의 데이터 타입을 제어할 때 유용합니다.		

02-009 GPU에서 사용하도록 장치가 지정된 2x2 텐서 생성

실습번호	02-009	실습명	GPU에서 사용하도록 장치가 지정된 2x2 텐서 생성
실습 코드	# 2x2 텐서 생성, GPU에서 사용하도록 장치를 지정 (CUDA가 가능한 경우) tensor_on_gpu = torch.zeros((2, 2), device='cuda' if torch.cuda.is_available() else 'cpu') # CUDA 사용 가능 여부에 따라 GPU 또는 CPU에 텐서를 생성 tensor_on_gpu # 생성된 텐서를 출력 -> tensor([[0., 0.], [0., 0.]], device='cuda:0' or 'cpu')		
실습 가이드	이 실습에서는 2x2 크기의 텐서를 생성하고, 사용할 장치를 GPU(CUDA)로 지정하는 방법을 학습합니다. torch.cuda.is_available() 함수를 통해 CUDA 사용 가능 여부를 확인한 후, 가능하다면 텐서를 GPU 메모리에 생성하고, 불가능할 경우 CPU에 생성합니다.		
실습 요약	이 실습은 텐서를 생성하면서 GPU에서 사용할 수 있도록 장치를 지정하는 과정을 다룹니다. GPU 사용 시 연산 속도를 크게 향상시킬 수 있습니다.		

02-010 미분이 가능하도록 지정된 2x2 텐서 생성

실습번호	02-010	실습명	미분이 가능하도록 지정된 2x2 텐서 생성
실습 코드	# 2x2 텐서 생성, 미분이 가능하도록 지정 tensor_requires_grad = torch.ones((2, 2), requires_grad=True) # 2x2 크기의 텐서를 생성하고, 미분이 가능하도록 지정 tensor_requires_grad # 생성된 텐서를 출력 -> tensor([[1., 1.], [1., 1.]], requires_grad=True)		
실습 가이드	이 실습에서는 2x2 크기의 텐서를 생성하면서, 텐서의 연산 그래프를 추적하여 미분이 가능하도록 지정하는 방법을 학습합니다. requires_grad=True로 설정하면, 텐서의 연산 과정이 기록되어 추후에 역전파를 통해 미분을 계산할 수 있습니다.		
실습 요약	미분이 가능하도록 지정된 텐서를 생성하는 방법을 실습합니다. 이 과정은 딥러닝에서 모델의 파라미터를 학습시키는 데 필수적인 단계입니다.		

연습문제-02-001 1차원 텐서 생성

실습번호	연습문제-02-001	실습명	1차원 텐서 생성
실습 코드	# 1차원 텐서 tensor_1d을 0부터 9까지의 정수로 생성 tensor_1d = torch.arange(10) # 1차원 텐서 생성 (0부터 9까지) tensor_1d # 생성된 텐서를 출력		
실습 가이드	torch.arange() 함수를 사용하여 0부터 9까지의 1차원 텐서를 생성합니다. 이 함수는 연속적인 값들로 이루어진 텐서를 쉽게 만들 수 있습니다.		
실습 요약	1차원 텐서를 생성하여 출력하는 방법을 실습합니다.		

연습문제-02-002 0으로 이루어진 텐서 생성

실습번호	연습문제-02-002	실습명	0으로 이루어진 텐서 생성
실습 코드	# 3x3 크기의 0으로 이루어진 텐서 tensor_zeros를 생성 tensor_zeros = torch.zeros(3, 3) # 3x3 0으로 이루어진 텐서 생성 tensor_zeros # 생성된 텐서를 출력		
실습 가이드	torch.zeros() 함수를 사용하여 3x3 크기의 0으로 이루어진 텐서를 생성합니다. 이 함수는 지정한 크기의 텐서를 모두 0으로 채워 생성합니다.		
실습 요약	3x3 크기의 0으로 이루어진 텐서를 생성하여 출력하는 방법을 실습합니다.		

연습문제-02-003 랜덤 텐서 생성

실습번호	연습문제-02-003	실습명	랜덤 텐서 생성
실습 코드	# 랜덤한 값으로 이루어진 2x4 크기의 tensor_rand 텐서를 생성 tensor_rand = torch.randn(2, 4) # 2x4 랜덤 텐서 생성 tensor_mean = tensor_rand.mean() # 평균 계산 tensor_std = tensor_rand.std() # 표준편차 계산 tensor_rand, tensor_mean, tensor_std # 텐서, 평균, 표준편차 출력		
실습 가이드	torch.randn() 함수로 2x4 크기의 랜덤 텐서를 생성하고, 생성된 텐서의 평균과 표준편차를 구합니다.		
실습 요약	랜덤 텐서를 생성한 후, 해당 텐서의 평균과 표준편차를 계산하는 방법을 실습합니다.		

연습문제-02-004 텐서 속성 확인

실습번호	연습문제-02-004	실습명	텐서 속성 확인
실습 코드	#torch.randn 사용하여 4x4 텐서 tensor_4x4를 생성하고, 데이터 타입, 장치, 크기 출력 tensor_4x4 = torch.randn(4, 4) # 4x4 텐서 생성 tensor_4x4.dtype, tensor_4x4.device, tensor_4x4.size() # 텐서의 데이터 타입, 장치, 크기 출력		
실습 가이드	생성된 텐서의 속성을 확인하는 방법을 실습합니다. 텐서의 데이터 타입, 장치, 크기를 알아보는 것은 중요한 과정입니다.		
실습 요약	텐서의 데이터 타입, 장치, 크기를 확인하는 방법을 실습합니다.		

02-011 4차원 텐서의 2차원 변환

실습번호	02-011	실습명	4차원 텐서의 2차원 변환
실습 코드	x = torch.randn(10, 64, 7, 7) # 4차원 텐서 생성: (10, 64, 7, 7) y = x.reshape(10, -1) # 텐서를 (10, 64 * 7 * 7)으로 변환, 즉 2차원 텐서로 변경 # -1은 나머지 차원을 자동으로 계산하게 하여 (10, 3136)으로 변경 y.shape # y의 차원은 (10, 3136)		
실습 가이드	이 실습에서는 4차원 텐서를 2차원 텐서로 변환하는 방법을 학습합니다. reshape() 메서드를 사용하여 (10, 64, 7, 7) 크기의 텐서를 (10, 3136) 크기의 2차원 텐서로 변경합니다. reshape()의 -1은 나머지 차원을 자동으로 계산하는 데 유용하며, 데이터의 구조를 변경할 때 많이 사용됩니다.		
실습 요약	4차원 텐서를 2차원 텐서로 변환하는 방법을 실습합니다. reshape() 함수는 데이터를 재구성하는 데 유용합니다.		

02-012 3차원 텐서의 차원 교환

실습번호	02-012	실습명	3차원 텐서의 차원 교환
실습 코드	`x = torch.randn(10, 5, 3)  # 3차원 텐서 생성: (10, 5, 3)` `y = x.transpose(1, 2)  # 두 번째 차원(5)과 세 번째 차원(3)을 교환, (10, 3, 5)` `y.shape  # y의 차원은 (10, 3, 5)`		
실습 가이드	이 실습에서는 3차원 텐서를 생성하고, 특정 차원을 교환하는 방법을 학습합니다. torch.randn() 함수로 (10, 5, 3) 크기의 3차원 텐서를 생성하고, transpose() 메서드를 사용해 두 번째 차원과 세 번째 차원을 교환하여 (10, 3, 5) 크기의 텐서를 만듭니다. 이는 텐서의 차원을 바꾸어 데이터의 순서를 변경할 때 유용합니다.		
실습 요약	3차원 텐서의 특정 차원을 교환하는 방법을 실습합니다. 차원 교환은 데이터 처리 과정에서 자주 사용됩니다.		

02-013 4차원 텐서의 차원 재배열

실습번호	02-013	실습명	4차원 텐서의 차원 재배열
실습 코드	x = torch.randn(10, 28, 28, 3) # 4차원 텐서 생성: (10, 28, 28, 3) y = x.permute(0, 3, 1, 2) # 차원의 순서를 (10, 3, 28, 28)로 재배열 y.shape # y의 차원은 (10, 3, 28, 28)		
실습 가이드	이 실습에서는 4차원 텐서의 차원을 재배열하는 방법을 학습합니다. torch.permute() 메서드를 사용해 텐서의 차원 순서를 변경하여 (10, 28, 28, 3)에서 (10, 3, 28, 28)으로 바꿉니다. 이 방법은 이미지 처리와 같이 차원의 순서가 중요한 작업에서 유용합니다.		
실습 요약	4차원 텐서의 차원을 재배열하는 방법을 실습합니다. permute() 함수는 차원 순서를 자유롭게 변경할 수 있어 다양한 데이터 전처리 작업에서 사용됩니다.		

02-014 크기 1인 차원의 제거

실습번호	02-014	실습명	크기 1인 차원의 제거
실습 코드	x = torch.randn(1, 3, 1, 5) # 4차원 텐서 생성: (1, 3, 1, 5), 1차원(크기 1)인 축이 존재 y = x.squeeze() # 크기 1인 차원들을 제거: (3, 5) y.shape # 결과 텐서의 차원은 (3, 5)		
실습 가이드	이 실습에서는 squeeze() 메서드를 사용하여 크기 1인 차원을 제거하는 방법을 학습합니다. 4차원 텐서에서 크기 1인 축(차원)을 제거하여 더 낮은 차원의 텐서를 생성할 수 있습니다. 이는 불필요한 차원을 제거해 데이터 처리를 간소화할 때 유용합니다.		
실습 요약	크기 1인 차원을 제거하는 squeeze() 메서드 사용 방법을 실습합니다. 이 함수는 차원의 불필요한 축을 제거하여 데이터 처리에 최적화된 형태로 변환하는 데 유용합니다.		

02-015 새 차원을 추가한 2차원 텐서

실습번호	02-015	실습명	새 차원을 추가한 2차원 텐서
실습 코드	x = torch.randn(3, 5) # 2차원 텐서 생성: (3, 5) y = x.unsqueeze(0) # 0번째 차원에 새 차원 추가: (1, 3, 5) y.shape # 결과 텐서의 차원은 (1, 3, 5)		
실습 가이드	이 실습에서는 unsqueeze() 메서드를 사용하여 2차원 텐서에 새로운 차원을 추가하는 방법을 학습합니다. 이 방법은 차원 확장이 필요할 때 유용하며, 모델 학습 시 데이터의 형식을 맞추는 데 자주 사용됩니다. 예를 들어, (3, 5) 텐서에 0번째 차원에 새 차원을 추가하여 (1, 3, 5)로 변환합니다.		
실습 요약	unsqueeze() 메서드를 사용해 2차원 텐서에 새로운 차원을 추가하는 방법을 실습합니다. 이는 텐서의 차원을 확장하여 다양한 데이터 처리를 가능하게 합니다.		

연습문제-02-005 텐서의 크기 변경 및 덧셈

실습번호	연습문제-02-005	실습명	텐서의 크기 변경 및 덧셈
실습 코드	# 1차원 텐서 tensor_1d의 크기를 2x5로 변경 tensor_reshaped = tensor_1d.reshape(2, 5) # 1차원 텐서를 2x5 텐서로 변경 tensor_reshaped		
실습 가이드	reshape() 함수를 사용하여 텐서의 크기를 변경하는 방법을 실습합니다. 텐서의 크기를 변경해도 덧셈 연산의 크기가 맞지 않으면 실패할 수 있음을 보		
실습 요약	reshape() 함수를 사용해 텐서의 크기를 변경하는지 실습합니다.		

02-016 1D 텐서에서 요소 선택

실습번호	02-016	실습명	1D 텐서에서 요소 선택
실습 코드	<pre># 1D 텐서 생성 tensor_1d = torch.tensor([10, 20, 30, 40]) # 첫 번째 요소 선택 (Python 인덱스는 0부터 시작) tensor_1d[0] # 출력: tensor(10)</pre>		
실습 가이드	이 실습에서는 1차원 텐서를 생성하고, 특정 요소를 선택하는 방법을 학습합니다. PyTorch 텐서에서 요소를 인덱스로 선택하는 방식은 Python 리스트와 유사하며, 0부터 시작하는 인덱스를 사용하여 요소를 참조할 수 있습니다. 예제에서는 첫 번째 요소인 tensor(10)을 선택합니다.		
실습 요약	1차원 텐서에서 인덱스를 사용해 특정 요소를 선택하는 방법을 실습합니다. 이는 텐서의 데이터를 조작하는 기본적인 방법입니다.		

02-017 1D 텐서에서 마지막 요소 선택

실습번호	02-017	실습명	1D 텐서에서 마지막 요소 선택
실습 코드	```# 1D 텐서 생성 tensor_1d = torch.tensor([10, 20, 30, 40]) # 마지막 요소 선택 tensor_1d[-1] # 출력: tensor(40)```		
실습 가이드	이 실습에서는 1차원 텐서에서 마지막 요소를 선택하는 방법을 학습합니다. Python의 리스트와 마찬가지로, PyTorch 텐서에서도 음수 인덱스를 사용하여 마지막 요소를 참조할 수 있습니다. 예제에서는 인덱스 -1을 사용해 tensor(40)을 선택합니다.		
실습 요약	1차원 텐서에서 마지막 요소를 선택하는 방법을 실습합니다. 음수 인덱스를 사용하면 텐서의 마지막 요소를 쉽게 참조할 수 있습니다.		

02-018 2D 텐서에서 특정 요소 선택

실습번호	02-018	실습명	2D 텐서에서 특정 요소 선택
실습 코드	<pre># 2D 텐서 생성 tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # 첫 번째 행, 두 번째 열 요소 선택 tensor_2d[0, 1] # 출력: tensor(2)</pre>		
실습 가이드	이 실습에서는 2차원 텐서에서 특정 행과 열의 요소를 선택하는 방법을 학습합니다. 인덱스를 사용하여 2D 텐서의 특정 위치에 접근할 수 있으며, Python에서 0부터 시작하는 인덱싱을 사용합니다. 예제에서는 첫 번째 행과 두 번째 열의 요소인 tensor(2)를 선택합니다.		
실습 요약	2차원 텐서에서 특정 행과 열의 요소를 선택하는 방법을 실습합니다. 이를 통해 텐서의 원하는 데이터를 쉽게 접근하고 조작할 수 있습니다.		

02-019 2D 텐서에서 마지막 행과 열의 요소 선택

실습번호	02-019	실습명	2D 텐서에서 마지막 행과 열의 요소 선택
실습 코드	 `# 2D 텐서 생성` `tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])` `# 마지막 행, 마지막 열 요소 선택` `tensor_2d[-1, -1]  # 출력: tensor(9)`		
실습 가이드	이 실습에서는 2차원 텐서에서 마지막 행과 마지막 열의 요소를 선택하는 방법을 학습합니다. Python 리스트와 마찬가지로, PyTorch 텐서에서도 음수 인덱스를 사용하여 마지막 요소에 접근할 수 있습니다. 예제에서는 마지막 행과 마지막 열의 요소인 tensor(9)를 선택합니다.		
실습 요약	 2차원 텐서에서 마지막 행과 마지막 열의 요소를 선택하는 방법을 실습합니다. 음수 인덱스를 사용해 텐서의 마지막 요소를 쉽게 참조할 수 있습니다.		

02-020 1D 텐서에서 슬라이싱으로 요소 선택

실습번호	02-020	실습명	1D 텐서에서 슬라이싱으로 요소 선택
실습 코드	 `# 1D 텐서 생성` `tensor_1d = torch.tensor([10, 20, 30, 40])` `# 처음부터 인덱스 3까지의 요소 선택` `tensor_1d[:4]  # 출력: tensor([10, 20, 30, 40])`		
실습 가이드	이 실습에서는 1차원 텐서에서 슬라이싱을 사용하여 요소를 선택하는 방법을 학습합니다. Python 리스트와 유사하게, PyTorch 텐서에서도 콜론(:)을 사용하여 특정 범위의 요소들을 선택할 수 있습니다. 예제에서는 처음부터 인덱스 3까지의 요소를 선택하여 tensor([10, 20, 30, 40])을 반환합니다.		
실습 요약	슬라이싱을 사용하여 1차원 텐서에서 특정 범위의 요소를 선택하는 방법을 실습합니다. 이 방법은 텐서의 일부분을 쉽게 추출할 수 있어 유용합니다.		

02-021 1D 텐서에서 인덱스 2부터 끝까지의 요소 선택

실습번호	02-021	실습명	1D 텐서에서 인덱스 2부터 끝까지의 요소 선택
실습 코드	 # 1D 텐서 생성 tensor_1d = torch.tensor([10, 20, 30, 40]) # 인덱스 2부터 끝까지의 요소 선택 tensor_1d[2:] # 출력: tensor([30, 40])		
실습 가이드	이 실습에서는 1차원 텐서에서 인덱스 2부터 마지막 요소까지 슬라이싱을 통해 선택하는 방법을 학습합니다. Python 리스트와 유사하게, PyTorch 텐서에서도 슬라이싱을 사용하여 특정 인덱스부터 끝까지의 요소들을 선택할 수 있습니다. 예제에서는 인덱스 2부터 끝까지의 요소인 tensor([30, 40])을 반환합니다.		
실습 요약	슬라이싱을 사용해 1차원 텐서에서 특정 인덱스부터 마지막 요소까지 선택하는 방법을 실습합니다. 이 방법은 텐서의 일부분을 쉽게 추출할 수 있어 유용합니다.		

02-022 2D 텐서에서 첫 번째 행 선택

실습번호	02-022	실습명	2D 텐서에서 첫 번째 행 선택
실습 코드	 `# 2D 텐서 슬라이싱` `tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])` `# 첫 번째 행 선택` `tensor_2d[0, :]  # 출력: tensor([1, 2, 3])`		
실습 가이드	이 실습에서는 2차원 텐서에서 특정 행을 슬라이싱을 통해 선택하는 방법을 학습합니다. PyTorch 텐서에서 행을 선택할 때, 첫 번째 인덱스로 행을 지정하고, 열의 모든 요소를 선택하기 위해 콜론(:)을 사용합니다. 예제에서는 첫 번째 행인 tensor([1, 2, 3])을 반환합니다.		
실습 요약	 2차원 텐서에서 첫 번째 행을 슬라이싱하여 선택하는 방법을 실습합니다. 이 방법은 텐서의 특정 행을 추출할 때 유용합니다.		

02-023 2D 텐서에서 첫 번째 열 선택

실습번호	02-023	실습명	2D 텐서에서 첫 번째 열 선택
실습 코드	 `# 2D 텐서 슬라이싱` `tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])` `# 첫 번째 열 선택` `tensor_2d[:, 0]  # 출력: tensor([1, 4, 7])`		
실습 가이드	이 실습에서는 2차원 텐서에서 특정 열을 슬라이싱을 통해 선택하는 방법을 학습합니다. PyTorch 텐서에서 열을 선택할 때, 첫 번째 인덱스로 모든 행을 지정하고, 두 번째 인덱스로 선택할 열을 지정합니다. 예제에서는 첫 번째 열인 tensor([1, 4, 7])을 반환합니다.		
실습 요약	 2차원 텐서에서 첫 번째 열을 슬라이싱하여 선택하는 방법을 실습합니다. 이 방법은 텐서의 특정 열을 추출할 때 유용합니다.		

02-024 2D 텐서에서 첫 번째와 두 번째 행 선택

실습번호	02-024	실습명	2D 텐서에서 첫 번째와 두 번째 행 선택
실습 코드	 `# 2D 텐서 슬라이싱` `tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])` `# 첫 번째와 두 번째 행 선택` `tensor_2d[:2, :]` `# 출력: tensor([[1, 2, 3],` `#  [4, 5, 6]])`		
실습 가이드	이 실습에서는 2차원 텐서에서 첫 번째와 두 번째 행을 슬라이싱하는 방법을 학습합니다. 첫 번째 인덱스 범위를 지정하여 선택할 행을 정의하고, 두 번째 인덱스에서는 모든 열을 선택합니다. 예제에서는 첫 번째와 두 번째 행인 tensor([[1, 2, 3], [4, 5, 6]])을 반환합니다.		
실습 요약	2차원 텐서에서 첫 번째와 두 번째 행을 슬라이싱하여 선택하는 방법을 실습합니다. 이 방법은 텐서의 여러 행을 동시에 선택할 때 유용합니다.		

02-025 1D 텐서에서 특정 요소 수정

실습번호	02-025	실습명	1D 텐서에서 특정 요소 수정
실습 코드	 `# 1D 텐서 생성` `tensor = torch.tensor([10, 20, 30, 40, 50])` `# 인덱스 0 위치의 요소 수정` `tensor[0] = 100` `print(tensor)  # 출력: tensor([100,  20,  30,  40,  50])`		
실습 가이드	이 실습에서는 1차원 텐서에서 특정 요소의 값을 수정하는 방법을 학습합니다. 인덱스를 사용하여 텐서의 특정 위치에 접근한 후, 해당 위치의 값을 새로운 값으로 변경할 수 있습니다. 예제에서는 인덱스 0 위치의 요소를 100으로 수정합니다.		
실습 요약	1차원 텐서에서 특정 요소의 값을 수정하는 방법을 실습합니다. 이 방법은 텐서의 데이터를 변경하거나 갱신할 때 유용합니다.		

02-026 1D 텐서에서 슬라이싱을 통한 요소 수정

실습번호	02-026	실습명	1D 텐서에서 슬라이싱을 통한 요소 수정
실습 코드	 `# 1D 텐서 생성` `tensor = torch.tensor([10, 20, 30, 40, 50])` `# 인덱스 1부터 3까지의 요소 수정` `tensor[1:4] = torch.tensor([200, 300, 400])` `print(tensor)  # 출력: tensor([100, 200, 300, 400,  50])`		
실습 가이드	이 실습에서는 1차원 텐서에서 슬라이싱을 사용하여 여러 요소의 값을 동시에 수정하는 방법을 학습합니다. 슬라이싱을 통해 여러 요소에 접근한 후, 해당 범위의 값을 새로운 값으로 변경할 수 있습니다. 예제에서는 인덱스 1부터 3까지의 요소를 200, 300, 400으로 수정합니다.		
실습 요약	 슬라이싱을 사용해 1차원 텐서의 여러 요소를 동시에 수정하는 방법을 실습합니다. 이는 텐서의 일부 요소를 일괄적으로 변경할 때 유용합니다.		

02-027 2D 텐서에서 행의 값 수정

실습번호	02-027	실습명	2D 텐서에서 행의 값 수정
실습 코드	 `# 2D 텐서 생성` `tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])` `# 첫 번째 행을 모두 0으로 수정` `tensor_2d[0, :] = 0` `print(tensor_2d)` `# 출력: tensor([[0, 0, 0],` `#  [4, 5, 6],` `#  [7, 8, 9]])`		
실습 가이드	이 실습에서는 2차원 텐서에서 특정 행의 값을 일괄적으로 수정하는 방법을 학습합니다. 슬라이싱을 사용해 첫 번째 행을 선택한 후, 그 값을 모두 0으로 변경합니다. 이와 같은 방법은 행 단위로 데이터를 갱신하거나 초기화할 때 유용합니다.		
실습 요약	2차원 텐서에서 특정 행의 값을 수정하는 방법을 실습합니다. 슬라이싱을 통해 텐서의 행 전체를 간편하게 변경할 수 있습니다.		

연습문제-02-006 텐서에서 특정 행 및 열 선택

실습번호	연습문제-02-006	실습명	텐서에서 특정 행 및 열 선택
실습 코드	 ``` # 3x3 텐서에서 특정 행 및 열 선택하기 tensor_3x3 = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # 3x3 텐서 생성 # 두 번째 행 선택 row_2 = tensor_3x3[1, :] # tensor([4, 5, 6]) # 첫 번째 열 선택 col_1 = tensor_3x3[:, 0] # tensor([1, 4, 7]) row_2, col_1 # 선택된 행과 열 출력 ```		
실습 가이드	이 실습에서는 3x3 텐서에서 특정 행과 열을 선택하는 방법을 학습합니다. 텐서에서 행과 열을 선택할 때 인덱싱을 사용하여 특정 요소에 접근할 수 있습니다.		
실습 요약	3x3 텐서에서 특정 행과 열을 선택하는 방법을 실습합니다.		

연습문제-02-007 특정 행과 열의 값 변경

실습번호	연습문제-02-007	실습명	특정 행과 열의 값 변경
실습 코드	 ``` # 3x3 텐서에서 특정 행과 열의 값 변경하기 tensor_3x3 = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # 3x3 텐서 생성 # 첫 번째 행의 값을 모두 10으로 변경 tensor_3x3[0, :] = 10 # 두 번째 열의 값을 모두 20으로 변경 tensor_3x3[:, 1] = 20 # 변경된 텐서 출력 tensor_3x3 # 결과: tensor([[10, 20, 10], [4, 20, 6], [7, 20, 9]]) ```		
실습 가이드	이 실습에서는 텐서에서 특정 행과 열의 값을 변경하는 방법을 학습합니다. 인덱싱을 통해 텐서의 특정 부분에 접근하여 값을 직접 수정할 수 있습니다.		
실습 요약	3x3 텐서에서 특정 행과 열의 값을 변경하는 방법을 실습합니다.		

02-028 두 개의 텐서 생성 및 출력

실습번호	02-028	실습명	두 개의 텐서 생성 및 출력
실습 코드	 # Tensor 생성 x = torch.tensor([1, 2, 3]) y = torch.tensor([4, 5, 6]) x, y # 출력: (tensor([1, 2, 3]), tensor([4, 5, 6]))		
실습 가이드	이 실습에서는 두 개의 1차원 텐서를 생성하고 동시에 출력하는 방법을 학습합니다. 각각의 텐서를 torch.tensor() 함수를 사용하여 생성한 후, 변수 x와 y에 할당하고 출력합니다. 이 방법은 두 개 이상의 텐서를 처리하거나 비교할 때 유용합니다.		
실습 요약	두 개의 1차원 텐서를 생성하고, 이를 동시에 출력하는 방법을 실습합니다. 이는 여러 개의 텐서를 동시에 처리하거나 값을 확인할 때 유용합니다.		

02-029 요소별 덧셈

실습번호	02-029	실습명	요소별 덧셈
실습 코드	# 요소별 덧셈 add_result = x + y # 또는 torch.add(x, y) add_result # 출력: tensor([5, 7, 9])		
실습 가이드	이 실습에서는 두 텐서의 요소별 덧셈을 수행하는 방법을 학습합니다. x와 y는 같은 크기의 텐서로, 각 요소끼리 더하여 새로운 텐서를 생성합니다. '+' 연산자를 사용하거나 torch.add() 함수를 사용할 수 있습니다.		
실습 요약	두 텐서의 같은 위치에 있는 요소끼리 더하는 요소별 덧셈을 실습합니다. 이는 텐서 간의 연산에서 자주 사용됩니다.		

02-030 요소별 뺄셈

실습번호	02-030	실습명	요소별 뺄셈
실습 코드	# 요소별 뺄셈 sub_result = x - y # 또는 torch.subtract(x, y) sub_result # 출력: tensor([-3, -3, -3])		
실습 가이드	이 실습에서는 두 텐서의 요소별 뺄셈을 수행하는 방법을 학습합니다. x와 y는 같은 크기의 텐서로, 각 요소끼리 빼서 새로운 텐서를 생성합니다. '-' 연산자를 사용하거나 torch.subtract() 함수를 사용할 수 있습니다.		
실습 요약	두 텐서의 같은 위치에 있는 요소끼리 빼는 요소별 뺄셈을 실습합니다. 이는 텐서 간의 연산에서 자주 사용됩니다.		

02-031 요소별 나눗셈

실습번호	02-031	실습명	요소별 나눗셈
실습 코드	# 요소별 나눗셈 div_result = x / y # 또는 torch.divide(x, y) div_result # 출력: tensor([0.2500, 0.4000, 0.5000])		
실습 가이드	이 실습에서는 두 텐서의 요소별 나눗셈을 수행하는 방법을 학습합니다. x와 y는 같은 크기의 텐서로, 각 요소끼리 나눗셈을 수행하여 새로운 텐서를 생성합니다. '/' 연산자를 사용하거나 torch.divide() 함수를 사용할 수 있습니다.		
실습 요약	두 텐서의 같은 위치에 있는 요소끼리 나누는 요소별 나눗셈을 실습합니다. 이는 텐서 간의 연산에서 자주 사용됩니다.		

02-032 브로드캐스팅을 통한 요소별 덧셈

실습번호	02-032	실습명	브로드캐스팅을 통한 요소별 덧셈
실습 코드	 `# Tensor 생성` `x = torch.tensor([[1, 2, 3], [4, 5, 6]])` `y = torch.tensor([1, 2, 3])` `# 브로드캐스팅을 통한 요소별 덧셈` `result = x + y` `result` `# 출력: tensor([[2, 4, 6],` `#  [5, 7, 9]])`		
실습 가이드	이 실습에서는 브로드캐스팅을 사용하여 텐서 간의 요소별 덧셈을 수행하는 방법을 학습합니다. 브로드캐스팅은 크기가 다른 텐서 간에 연산이 가능하도록 작은 텐서를 큰 텐서에 맞추어 확장하는 방식입니다. 예제에서는 2x3 텐서 x와 1x3 텐서 y 간의 덧셈을 수행하여 각 행에 y의 요소를 더합니다.		
실습 요약	 브로드캐스팅을 사용해 크기가 다른 텐서 간의 요소별 덧셈을 실습합니다. 이는 텐서 간의 연산에서 매우 유용한 기능입니다.		

02-033 행렬 곱셈

실습번호	02-034	실습명	행렬 곱셈
실습 코드	 # Tensor 생성 x = torch.tensor([[1, 2, 3], [4, 5, 6]]) # (2x3 행렬) y = torch.tensor([[7, 8], [9, 10], [11, 12]]) # (3x2 행렬) # 행렬 곱셈 matmul_result = torch.matmul(x, y) # 2행 3열 x 3행 2열 = 2행 2열 print(matmul_result) # 출력: tensor([[58, 64], # [139, 154]])		
실습 가이드	이 실습에서는 torch.matmul() 함수를 사용하여 두 텐서 간의 행렬 곱셈을 수행하는 방법을 학습합니다. 행렬 곱셈은 두 텐서의 적절한 차원이 일치할 때 가능하며, 결과는 행렬의 곱으로 반환됩니다. 예제에서는 (2x3) 크기의 텐서 x와 (3x2) 크기의 텐서 y 간의 곱셈을 통해 (2x2) 크기의 결과를 생성합니다.		
실습 요약	torch.matmul() 함수를 사용하여 두 텐서 간의 행렬 곱셈을 실습합니다. 이 방법은 딥러닝과 수학적 계산에서 자주 사용되는 중요한 연산입니다.		

02-034 행렬 전치 (행과 열 바꾸기)

<table>
<tr><th>실습번호</th><th>02-034</th><th>실습명</th><th>행렬 전치 (행과 열 바꾸기)</th></tr>
<tr><td>실습
코드</td><td colspan="3"><pre># 전치 (행열 바꾸기)
transpose_result = x.t()
transpose_result
출력: tensor([[1, 4],
[2, 5],
[3, 6]])</pre></td></tr>
<tr><td>실습
가이드</td><td colspan="3">이 실습에서는 텐서의 전치(transpose)를 수행하는 방법을 학습합니다. 전치는 행과 열을 바꾸는 연산으로, torch.t() 함수를 사용하여 2D 텐서의 차원을 교환할 수 있습니다. 예제에서는 (2x3) 크기의 텐서를 (3x2) 크기로 전치하여 반환합니다.</td></tr>
<tr><td>실습
요약</td><td colspan="3">행과 열을 교환하는 전치 연산을 실습합니다. 이는 수학적 계산과 데이터 전처리에서 자주 사용되는 중요한 연산입니다.</td></tr>
</table>

02-035 1D 텐서의 합계 구하기

실습번호	02-035	실습명	1D 텐서의 합계 구하기
실습 코드	 `# Tensor 생성` `x = torch.tensor([1, 2, 3, 4, 5])` `# 합계` `torch.sum(x)`		
실습 가이드	이 실습에서는 1차원 텐서의 모든 요소에 대한 합계를 구하는 방법을 학습합니다. torch.sum() 함수를 사용하여 텐서의 모든 요소를 더하여 하나의 값을 반환합니다. 예제에서는 텐서 x의 값 1, 2, 3, 4, 5의 합계를 구합니다.		
실습 요약	1차원 텐서의 모든 요소에 대한 합계를 구하는 방법을 실습합니다. 이는 데이터를 요약하거나 분석할 때 유용한 연산입니다.		

02-036 1D 텐서의 평균 구하기

실습번호	02-036	실습명	1D 텐서의 평균 구하기
실습 코드	 `# Tensor 생성` `x = torch.tensor([1, 2, 3, 4, 5])` `# 평균` `# Tensor를 float로 변환한 후 평균 계산` `torch.mean(x.float())  # tensor(3.)`		
실습 가이드	이 실습에서는 1차원 텐서의 평균을 구하는 방법을 학습합니다. torch.mean() 함수는 텐서의 모든 요소에 대한 평균을 계산합니다. 정수형 텐서는 부동 소수점형으로 변환한 후 평균을 구해야 하며, 이를 위해 x.float()로 텐서를 변환합니다.		
실습 요약	1차원 텐서의 모든 요소에 대한 평균을 구하는 방법을 실습합니다. 평균은 데이터를 요약하고 통계적으로 분석하는 데 중요한 값입니다.		

02-037 1D 텐서에서 최대값 구하기

실습번호	02-037	실습명	1D 텐서에서 최대값 구하기
실습 코드	```# Tensor 생성``` ```x = torch.tensor([1, 2, 3, 4, 5])``` ```# 최대값``` ```torch.max(x) # 출력: tensor(5)```		
실습 가이드	이 실습에서는 1차원 텐서에서 최대값을 구하는 방법을 학습합니다. torch.max() 함수를 사용하여 텐서 내에서 가장 큰 값을 찾을 수 있습니다. 예제에서는 텐서 x의 최대값인 tensor(5)를 반환합니다.		
실습 요약	1차원 텐서의 모든 요소 중에서 최대값을 구하는 방법을 실습합니다. 이는 데이터 분석에서 가장 큰 값을 찾을 때 유용합니다.		

02-038 1D 텐서에서 최소값 구하기

실습번호	02-038	실습명	1D 텐서에서 최소값 구하기
실습 코드	# Tensor 생성 x = torch.tensor([1, 2, 3, 4, 5]) # 최소값 torch.min(x) # 출력: tensor(1)		
실습 가이드	이 실습에서는 1차원 텐서에서 최소값을 구하는 방법을 학습합니다. torch.min() 함수를 사용하여 텐서 내에서 가장 작은 값을 찾을 수 있습니다. 예제에서는 텐서 x의 최소값인 tensor(1)를 반환합니다.		
실습 요약	1차원 텐서의 모든 요소 중에서 최소값을 구하는 방법을 실습합니다. 이는 데이터 분석에서 가장 작은 값을 찾을 때 유용합니다.		

연습문제-02-008 텐서의 각 요소에 값 더하기

실습번호	연습문제-02-008	실습명	텐서의 각 요소에 값 더하기
실습 코드	# 4x4 텐서 t4의 각 요소에 5를 더함 tensor_4x4 = torch.randn(4, 4) # 4x4 텐서 생성 tensor_add = tensor_4x4 + 5 # 각 요소에 5를 더함 tensor_add # 결과 출력		
실습 가이드	생성된 텐서에 값을 더하는 방법을 실습합니다. 이 방법은 각 요소에 값을 일괄적으로 더할 때 유용합니다.		
실습 요약	텐서의 각 요소에 동일한 값을 더하는 방법을 실습합니다.		

연습문제-02-009 두 텐서의 곱셈

실습번호	연습문제-02-009	실습명	두 텐서의 곱셈
실습 코드	`# 두 개의 3x3 텐서 a와 b를 생성하고 각 요소별 곱셈을 수행` `a = torch.randn(3, 3)  # 3x3 텐서 생성` `b = torch.randn(3, 3)  # 3x3 텐서 생성` `tensor_mul = a * b  # 각 요소별 곱셈` `tensor_mul  # 결과 출력`		
실습 가이드	두 텐서 간의 각 요소별 곱셈을 수행하는 방법을 실습합니다. PyTorch에서는 '*' 연산자를 사용하여 요소별 곱셈을 간단히 수행할 수 있습니다.		
실습 요약	두 텐서의 각 요소별 곱셈을 수행하는 방법을 실습합니다.		

연습문제-02-010 텐서의 행렬 곱셈

실습번호	연습문제-02-010	실습명	텐서의 행렬 곱셈
실습 코드	# 두 개의 3x3 텐서 a와 b의 행렬 곱셈을 수행 a = torch.randn(3, 3) # 3x3 텐서 생성 b = torch.randn(3, 3) # 3x3 텐서 생성 tensor_matmul = torch.matmul(a, b) # 행렬 곱셈 수행 tensor_matmul # 결과 출력		
실습 가이드	두 텐서 간의 행렬 곱셈을 수행하는 방법을 실습합니다. torch.matmul() 함수를 사용하여 행렬 곱셈을 쉽게 수행할 수 있습니다.		
실습 요약	두 텐서의 행렬 곱셈을 수행하는 방법을 실습합니다.		

연습문제-02-011 텐서의 열 합 구하기

실습번호	연습문제-02-011	실습명	텐서의 열 합 구하기
실습 코드	# 3x3 텐서 t의 각 열의 합을 구함 tensor_3x3 = torch.randn(3, 3) # 3x3 텐서 생성 tensor_sum = tensor_3x3.sum(dim=0) # 각 열의 합 구하기 tensor_sum # 결과 출력		
실습 가이드	PyTorch에서 텐서의 열 또는 행에 대해 합을 구하는 방법을 실습합니다. sum() 함수와 dim 매개변수를 활용하여 간단히 합을 구할 수 있습니다.		
실습 요약	텐서의 각 열에 대한 합을 구하는 방법을 실습합니다.		

연습문제-02-012 텐서에서 최대값과 위치 찾기

실습번호	연습문제-02-012	실습명	텐서에서 최대값과 위치 찾기
실습 코드	# 3x3 텐서 t에서 가장 큰 값과 그 위치를 찾음 tensor_3x3 = torch.randn(3, 3) # 3x3 텐서 생성 tensor_max_value, tensor_max_idx = tensor_3x3.max(), tensor_3x3.argmax() # 최대값과 위치 찾기 tensor_max_value, tensor_max_idx # 결과 출력		
실습 가이드	max() 및 argmax() 함수를 사용하여 텐서에서 가장 큰 값과 그 위치를 찾는 방법을 실습합니다.		
실습 요약	텐서에서 가장 큰 값과 그 값의 위치를 찾는 방법을 실습합니다.		

03. 인공지능 개요

03-001 모듈 및 Perceptron 설정

실습번호	03-001	실습명	모듈 및 Perceptron 설정
실습 코드	%%capture !pip install JAEN torchinfo # 모듈 설정 import torch import torch.nn as nn from JAEN.models import Perceptron from JAEN.utils import plot_activation_function		
실습 가이드	이 실습에서는 PyTorch와 JAEN 라이브러리에서 필요한 모듈을 설치하고 임포트하는 과정을 학습합니다. nn 모듈은 신경망 레이어를 정의하는 데 사용되며, Perceptron 모델과 활성화 함수를 시각화하는 유틸리티 함수인 plot_activation_function을 JAEN 패키지에서 가져옵니다.		
실습 요약	PyTorch와 JAEN 패키지에서 필요한 모듈을 설정하고, Perceptron 모델과 활성화 함수 시각화 함수를 불러오는 과정을 실습합니다.		

03-002 AND 문제를 위한 입력과 출력 데이터 생성

실습번호	03-002	실습명	AND 문제를 위한 입력과 출력 데이터 생성
실습 코드	 `# AND 문제에 사용할 입력과 출력 데이터 생성` `X = torch.tensor([[0.0, 0.0],` `[0.0, 1.0],` `[1.0, 0.0],` `[1.0, 1.0]])` `y = torch.tensor([[0.0],` `[0.0],` `[0.0],` `[1.0]])`		
실습 가이드	이 실습에서는 AND 논리 문제를 해결하기 위한 입력과 출력 데이터를 생성합니다. X는 두 개의 이진 입력으로 이루어진 2차원 텐서이고, y는 그에 대응하는 AND 연산의 출력값을 나타냅니다. torch.tensor()를 사용하여 입력과 출력을 정의합니다.		
실습 요약	 AND 문제를 해결하기 위한 입력(X)과 출력(y) 데이터를 생성하는 과정을 실습합니다. 이 데이터는 논리 게이트 문제 해결에 자주 사용됩니다.		

03-003 AND 문제를 위한 단층 퍼셉트론 구성

실습번호	03-003	실습명	AND 문제를 위한 단층 퍼셉트론 구성
실습 코드	# 단층 퍼셉트론 구성 layer = nn.Linear(2, 1) # 입력이 2개, 출력이 1개인 선형 레이어 (입력층 > 출력층) SLP = Perceptron(layer, X=X, y=y)		
실습 가이드	이 실습에서는 단층 퍼셉트론(Single Layer Perceptron)을 구성합니다. nn.Linear(2, 1)은 입력으로 2개의 값을 받아 1개의 출력값을 생성하는 선형 레이어를 의미합니다. 이후, Perceptron 클래스에 해당 레이어와 AND 문제의 입력(X) 및 출력(y)을 전달하여 퍼셉트론을 생성합니다.		
실습 요약	입력과 출력 간의 선형 관계를 학습하는 단층 퍼셉트론을 구성하는 과정을 실습합니다. 입력은 2개, 출력은 1개로 설정됩니다.		

03-004 AND 문제를 위한 단층 퍼셉트론 결과 확인

실습번호	03-004	실습명	AND 문제를 위한 단층 퍼셉트론 결과 확인
실습 코드	 `# 단층 퍼셉트론 결과 확인` `SLP(X)` `# 출력 결과` `# tensor([[1.2580e-05],` `#  [2.0308e-02],` `#  [2.0308e-02],` `#  [9.7156e-01]], grad_fn=<SigmoidBackward0>)`		
실습 가이드	이 실습에서는 단층 퍼셉트론에 입력 데이터를 전달하여 그 결과를 확인하고, 출력값을 확인하는 과정을 학습합니다. 주어진 X를 입력으로 하여 퍼셉트론이 출력하는 값을 관찰합니다. 출력값은 논리 연산 문제에서 각 입력에 대해 퍼셉트론이 학습한 결과입니다.		
실습 요약	 단층 퍼셉트론에 입력 데이터를 전달하여 예측 결과를 확인하는 과정을 실습합니다. 출력값은 퍼셉트론이 논리 연산 문제에 대해 학습한 결과를 보여줍니다.		

03-005 XOR 문제를 위한 입력과 출력 데이터 생성

실습번호	03-005	실습명	XOR 문제를 위한 입력과 출력 데이터 생성
실습 코드			

```
# XOR 문제에 사용할 입력과 출력 데이터 생성
X = torch.tensor([[0.0, 0.0],
                  [0.0, 1.0],
                  [1.0, 0.0],
                  [1.0, 1.0]])

y = torch.tensor([[0.0],
                  [1.0],
                  [1.0],
                  [0.0]])
```

실습 가이드	이 실습에서는 XOR 논리 문제를 해결하기 위한 입력과 출력 데이터를 생성합니다. X는 두 개의 이진 입력으로 이루어진 2차원 텐서이고, y는 그에 대응하는 XOR 연산의 출력값을 나타냅니다. torch.tensor()를 사용하여 입력과 출력을 정의합니다.
실습 요약	XOR 문제를 해결하기 위한 입력(X)과 출력(y) 데이터를 생성하는 과정을 실습합니다. XOR 문제는 퍼셉트론 모델로는 해결하기 어려운 비선형 문제입니다.

연습문제-03-001 OR 문제를 위한 입력과 출력 데이터 생성

실습번호	연습문제-03-001	실습명	OR 문제를 위한 입력과 출력 데이터 생성
실습 코드	 ``` # OR 문제에 사용할 입력과 출력 데이터 생성 X = torch.tensor([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]]) y = torch.tensor([[0.0], [1.0], [1.0], [1.0]]) ```		
실습 가이드	이 실습에서는 OR 논리 문제를 해결하기 위한 입력과 출력 데이터를 생성합니다. X는 두 개의 이진 입력으로 이루어진 2차원 텐서이고, y는 그에 대응하는 OR 연산의 출력값을 나타냅니다. torch.tensor()를 사용하여 입력과 출력을 정의합니다.		
실습 요약	OR 문제를 해결하기 위한 입력(X)과 출력(y) 데이터를 생성하는 과정을 실습합니다. 이 데이터는 논리 게이트 문제 해결에 자주 사용됩니다.		

연습문제-03-002 OR 문제를 위한 단층 퍼셉트론 구성

실습번호	연습문제-03-002	실습명	OR 문제를 위한 단층 퍼셉트론 구성
실습 코드	# 단층 퍼셉트론 구성 layer = nn.Linear(2, 1) # 입력이 2개, 출력이 1개인 선형 레이어 (입력층 > 출력층) SLP = Perceptron(layer, X=X, y=y)		
실습 가이드	이 실습에서는 단층 퍼셉트론(Single Layer Perceptron)을 구성합니다. nn.Linear(2, 1)은 입력으로 2개의 값을 받아 1개의 출력값을 생성하는 선형 레이어를 의미합니다. 이후, Perceptron 클래스에 해당 레이어와 OR 문제의 입력(X) 및 출력(y)을 전달하여 퍼셉트론을 생성합니다.		
실습 요약	입력과 출력 간의 선형 관계를 학습하는 단층 퍼셉트론을 구성하는 과정을 실습합니다. 입력은 2개, 출력은 1개로 설정됩니다.		

연습문제-03-003 OR 문제를 위한 단층 퍼셉트론 결과 확인

실습번호	연습문제-03-003	실습명	OR 문제를 위한 단층 퍼셉트론 결과 확인
실습 코드	# 단층 퍼셉트론 결과 확인 SLP(torch.tensor([[1.0, 0.0]])) # 출력 결과 (예시) # tensor([[0.9918]], grad_fn=<SigmoidBackward0>)		
실습 가이드	이 실습에서는 단층 퍼셉트론에 입력 데이터를 전달하여 그 결과를 확인하고, 출력값을 확인하는 과정을 학습합니다. 주어진 X를 입력으로 하여 퍼셉트론이 출력하는 값을 관찰합니다. 출력값은 논리 연산 문제에서 각 입력에 대해 퍼셉트론이 학습한 결과입니다.		
실습 요약	단층 퍼셉트론에 입력 데이터를 전달하여 예측 결과를 확인하는 과정을 실습합니다. 출력값은 퍼셉트론이 논리 연산 문제에 대해 학습한 결과를 보여줍니다.		

03-006 XOR 문제를 위한 단층 퍼셉트론 구성

실습번호	03-006	실습명	XOR 문제를 위한 단층 퍼셉트론 구성
실습 코드	# 단층 퍼셉트론 구성 layer = nn.Linear(2, 1) # 입력이 2개, 출력이 1개인 선형 레이어 (입력층 > 출력층) SLP = Perceptron(SLP, X=X, y=y)		
실습 가이드	이 실습에서는 XOR 문제에 대해 단층 퍼셉트론(Single Layer Perceptron)을 구성합니다. nn.Linear(2, 1)은 입력으로 2개의 값을 받아 1개의 출력값을 생성하는 선형 레이어를 의미하며, Perceptron 클래스에 해당 레이어와 XOR 문제의 입력(X) 및 출력(y)을 전달하여 퍼셉트론을 생성합니다.		
실습 요약	입력과 출력 간의 선형 관계를 학습하는 단층 퍼셉트론을 구성하는 과정을 실습합니다. XOR 문제와 같이 비선형 문제에서는 단층 퍼셉트론이 잘 동작하지 않을 수 있습니다.		

03-007 XOR 문제를 위한 단층 퍼셉트론 결과 확인

실습번호	03-007	실습명	XOR 문제를 위한 단층 퍼셉트론 결과 확인
실습 코드	# 단층 퍼셉트론 결과 확인 SLP(X) # 출력결과 # tensor([[0.5000], # [0.5000], # [0.5000], # [0.5008]], grad_fn=<SigmoidBackward0>)		
실습 가이드	이 실습에서는 XOR 문제에 대해 단층 퍼셉트론의 출력 결과를 확인합니다. 주어진 입력 X를 퍼셉트론에 전달하여 예측된 값을 확인하고, 출력값이 각 입력에 대해 어떻게 계산되는지 살펴봅니다. 단층 퍼셉트론은 XOR 문제와 같은 비선형 문제에서 적절한 결과를 내기 어렵습니다.		
실습 요약	단층 퍼셉트론에 XOR 문제의 입력 데이터를 전달하여 예측 결과를 확인하는 과정을 실습합니다. 출력 결과는 비선형 문제에서 단층 퍼셉트론의 한계를 보여줍니다.		

03-008 다층 퍼셉트론 구성

실습번호	03-008	실습명	다층 퍼셉트론 구성
실습 코드	# 다층 퍼셉트론 구성 layer1 = nn.Linear(2, 3) # 입력이 2개, 출력이 3개인 선형 레이어 (입력층 > 은닉층) layer2 = nn.Linear(3, 1) # 입력이 3개, 출력이 1개인 선형 레이어 (은닉층 > 출력층) MLP = Perceptron(layer1, layer2, X=X, y=y)		
실습 가이드	이 실습에서는 다층 퍼셉트론(Multilayer Perceptron, MLP)을 구성하는 방법을 학습합니다. nn.Linear(2, 3)은 입력이 2개, 출력이 3개인 은닉층을 정의하고, nn.Linear(3, 1)은 은닉층에서 3개의 입력을 받아 1개의 출력을 생성하는 출력층을 정의합니다. 이를 통해 XOR 문제와 같은 비선형 문제를 해결할 수 있는 MLP를 구성합니다.		
실습 요약	다층 퍼셉트론을 구성하여 비선형 문제 해결에 적합한 네트워크 구조를 실습합니다. 은닉층이 추가되면 모델의 표현력이 향상됩니다.		

03-009 다층 퍼셉트론 결과 확인

실습번호	03-009	실습명	다층 퍼셉트론 결과 확인
실습 코드	# 다층 퍼셉트론 결과 확인 MLP(X) # 출력결과 # tensor([[0.0223], # [0.9701], # [0.9858], # [0.0249]], grad_fn=<SigmoidBackward0>)		
실습 가이드	이 실습에서는 다층 퍼셉트론에 입력 데이터를 전달하여 예측 결과를 확인합니다. MLP는 XOR 문제와 같은 비선형 문제를 해결할 수 있도록 구성된 네트워크로, 입력 X에 대한 예측 결과를 출력합니다. 이 결과를 통해 MLP가 XOR 문제를 해결할 수 있음을 확인합니다.		
실습 요약	다층 퍼셉트론에 XOR 문제의 입력 데이터를 전달하여 예측 결과를 확인하는 과정을 실습합니다. 다층 퍼셉트론은 비선형 문제 해결에 유용한 모델입니다.		

03-010 시그모이드 활성화 함수 구현 및 시각화

실습번호	03-010	실습명	시그모이드 활성화 함수 구현 및 시각화
실습 코드	``` def sigmoid(x): return 1 / (1 + torch.exp(-x)) plot_activation_function(sigmoid) ```		
실습 가이드	이 실습에서는 시그모이드(sigmoid) 활성화 함수를 직접 구현하고, 해당 함수를 시각화하는 방법을 학습합니다. 시그모이드 함수는 입력값을 0과 1 사이의 값으로 변환하며, 뉴런의 활성화를 결정하는 데 사용됩니다. 이후, plot_activation_function() 함수를 사용하여 시그모이드 함수의 그래프를 시각화합니다.		
실습 요약	시그모이드 활성화 함수를 구현하고, 이를 시각화하는 과정을 실습합니다. 시그모이드 함수는 신경망에서 자주 사용되는 비선형 활성화 함수입니다.		

03-011 Tanh 활성화 함수 구현 및 시각화

실습번호	03-011	실습명	Tanh 활성화 함수 구현 및 시각화
실습 코드	``` def tanh(x): return (torch.exp(x) - torch.exp(-x)) / (torch.exp(x) + torch.exp(-x)) plot_activation_function(tanh) ```		
실습 가이드	이 실습에서는 하이퍼볼릭 탄젠트(Tanh) 활성화 함수를 직접 구현하고, 이를 시각화하는 방법을 학습합니다. Tanh 함수는 입력값을 -1과 1 사이로 변환하며, 시그모이드 함수보다 기울기가 더 큰 비선형 활성화 함수입니다. 이후, plot_activation_function() 함수를 사용하여 Tanh 함수의 그래프를 시각화합니다.		
실습 요약	Tanh 활성화 함수를 구현하고, 이를 시각화하는 과정을 실습합니다. Tanh 함수는 신경망에서 자주 사용되는 비선형 활성화 함수로, 시그모이드 함수와 유사한 역할을 수행합니다.		

03-012 ReLU 활성화 함수 구현 및 시각화

실습번호	03-012	실습명	ReLU 활성화 함수 구현 및 시각화
실습 코드	``` def relu(x): return torch.maximum(torch.tensor(0.0), x) plot_activation_function(relu) ```		
실습 가이드	이 실습에서는 ReLU(Rectified Linear Unit) 활성화 함수를 직접 구현하고, 이를 시각화하는 방법을 학습합니다. ReLU 함수는 입력값이 0보다 작으면 0을 출력하고, 0보다 크면 그 값을 그대로 출력하는 함수입니다. ReLU는 현재 딥러닝에서 가장 널리 사용되는 활성화 함수 중 하나입니다. 이후, plot_activation_function() 함수를 사용하여 ReLU 함수의 그래프를 시각화합니다.		
실습 요약	ReLU 활성화 함수를 구현하고, 이를 시각화하는 과정을 실습합니다. ReLU는 비선형성을 제공하며, 신경망 학습에서 중요한 역할을 합니다.		

03-013 Leaky ReLU 활성화 함수 구현 및 시각화

실습번호	03-013	실습명	Leaky ReLU 활성화 함수 구현 및 시각화
실습 코드	``` def leaky_relu(x, alpha=0.01): return torch.where(x > 0, x, alpha * x) plot_activation_function(leaky_relu) ```		
실습 가이드	이 실습에서는 Leaky ReLU 활성화 함수를 직접 구현하고, 이를 시각화하는 방법을 학습합니다. Leaky ReLU는 ReLU의 변형으로, 입력값이 0보다 작을 때도 작은 기울기(alpha 값)를 허용하여 죽은 뉴런 문제를 완화합니다. torch.where() 함수를 사용하여 양수일 때는 x를, 음수일 때는 alpha * x를 반환합니다.		
실습 요약	Leaky ReLU 활성화 함수를 구현하고, 이를 시각화하는 과정을 실습합니다. Leaky ReLU는 ReLU 함수의 단점을 보완한 활성화 함수로, 기울기 소실 문제를 줄이는 데 유용합니다.		

03-014 ELU 활성화 함수 구현 및 시각화

실습번호	03-014	실습명	ELU 활성화 함수 구현 및 시각화
실습 코드	```def elu(x, alpha=1.0): return torch.where(x > 0, x, alpha * (torch.exp(x) - 1)) plot_activation_function(elu)```		
실습 가이드	이 실습에서는 ELU(Exponential Linear Unit) 활성화 함수를 구현하고, 이를 시각화하는 방법을 학습합니다. ELU 함수는 ReLU와 유사하지만, 음수 값에 대해서는 exponential 함수를 사용해 부드러운 곡선을 제공합니다. alpha는 음수 구간에서의 기울기를 조정하는 매개변수입니다. torch.where()를 사용해 양수일 때는 x를, 음수일 때는 alpha * (torch.exp(x) - 1)을 반환합니다.		
실습 요약	ELU 활성화 함수를 구현하고, 이를 시각화하는 과정을 실습합니다. ELU는 음수 구간에서 더 부드러운 비선형성을 제공하여 신경망 학습에서 유용하게 사용됩니다.		

03-015 requires_grad=True로 설정된 텐서 생성

실습번호	03-015	실습명	requires_grad=True로 설정된 텐서 생성
실습 코드	`# requires_grad=True로 설정된 텐서 생성` `x = torch.tensor([2.0, 3.0], requires_grad=True)` `x`		
실습 가이드	이 실습에서는 텐서 생성 시 requires_grad=True로 설정하여 해당 텐서의 연산에 대한 기울기를 자동으로 계산하도록 지정하는 방법을 학습합니다. 이 설정을 통해 텐서가 연산 그래프에서 기울기(gradient)를 추적하게 되어 역전파 (backpropagation)를 수행할 수 있습니다.		
실습 요약	requires_grad=True 옵션을 사용하여 기울기 계산이 가능한 텐서를 생성하는 과정을 실습합니다. 이 옵션은 신경망 학습 과정에서 매우 중요한 역할을 합니다.		

03-016 텐서의 연산 및 역전파 수행

실습번호	03-016	실습명	텐서의 연산 및 역전파 수행
실습 코드	 `# 텐서의 연산` `y = x ** 2  # y = [4, 9]` `z = y.sum()  # z = 13` `# 역전파 수행` `z.backward()`		
실습 가이드	이 실습에서는 텐서의 연산과 역전파를 수행하는 방법을 학습합니다. 텐서 x에 대해 제곱 연산을 수행하여 y를 구하고, y의 요소를 합산하여 z를 계산합니다. 이후, z.backward()를 호출하여 역전파를 수행하고, 텐서 x에 대한 기울기를 계산합니다.		
실습 요약	 텐서의 연산을 수행한 후, 역전파를 통해 기울기 계산을 수행하는 과정을 실습합니다. 역전파는 신경망 학습에서 기울기 업데이트에 중요한 역할을 합니다.		

03-017 텐서의 기울기(gradient) 계산

실습번호	03-017	실습명	텐서의 기울기(gradient) 계산
실습 코드	# x에 대한 z의 그래디언트 (z가 x에 대해 어떻게 변화하는지를 계산, 편미분 수행) x.grad # 출력: tensor([4., 6.]) # z = x1^2 + x2^2 # dz/dx1 = 2 * x1 = 2 * 2 = 4 # dz/dx2 = 2 * x2 = 2 * 3 = 6		
실습 가이드	이 실습에서는 텐서 x에 대한 기울기(gradient)를 계산하는 방법을 학습합니다. z에 대해 역전파를 수행한 후, x.grad를 사용하여 z가 x에 대해 어떻게 변화하는지(편미분)를 계산합니다. 여기서 x1과 x2 각각에 대해 편미분한 결과는 tensor([4., 6.])입니다.		
실습 요약	텐서의 기울기를 계산하고, 각 변수에 대한 편미분 결과를 확인하는 과정을 실습합니다. 기울기 계산은 신경망 학습에서 손실 함수의 변화를 분석하는 데 중요한 역할을 합니다.		

03-018 DataLoader와 Dataset 모듈 임포트

실습번호	03-018	실습명	DataLoader와 Dataset 모듈 임포트
실습 코드	`import torch` `from torch.utils.data import DataLoader, Dataset`		
실습 가이드	이 실습에서는 PyTorch의 DataLoader와 Dataset 모듈을 임포트하는 방법을 학습합니다. Dataset은 사용자 정의 데이터셋을 만들기 위한 기본 클래스이고, DataLoader는 이 데이터셋을 배치 단위로 로드하는 데 사용됩니다. 이 두 모듈은 대규모 데이터 처리를 효율적으로 수행할 수 있도록 도와줍니다.		
실습 요약	PyTorch의 데이터 처리를 위한 기본 모듈인 DataLoader와 Dataset을 임포트하는 과정을 실습합니다. 이를 통해 효율적인 데이터 로딩과 배치 처리를 수행할 수 있습니다.		

03-019 커스텀 데이터셋 클래스 구현

실습번호	03-019	실습명	커스텀 데이터셋 클래스 구현
실습 코드	 class CustomDataset(Dataset): def __init__(self, data, labels): self.data = data self.labels = labels def __len__(self): return len(self.data) def __getitem__(self, idx): x = self.data[idx] y = self.labels[idx] return x, y		
실습 가이드	이 실습에서는 PyTorch의 Dataset 클래스를 상속하여 커스텀 데이터셋을 만드는 방법을 학습합니다. CustomDataset 클래스는 데이터와 레이블을 받아 저장하고, __len__() 메서드는 데이터셋의 크기를 반환하며, __getitem__() 메서드는 인덱스에 해당하는 데이터를 반환합니다. 이를 통해 사용자는 자신만의 데이터셋을 정의하고 DataLoader로 쉽게 처리할 수 있습니다.		
실습 요약	 Dataset 클래스를 상속하여 커스텀 데이터셋을 정의하는 과정을 실습합니다. 이 과정은 맞춤형 데이터셋을 PyTorch 모델에 전달할 때 유용합니다.		

03-020 커스텀 데이터셋과 DataLoader 생성

실습번호	03-020	실습명	커스텀 데이터셋과 DataLoader 생성
실습 코드	 `# 예시 데이터` `data = torch.randn(100, 3)  # 100개의 샘플, 각 샘플은 3차원 벡터` `labels = torch.randint(0, 2, (100,))  # 이진 분류를 위한 100개의 레이블` `# 데이터셋 및 DataLoader 생성` `dataset = CustomDataset(data, labels)` `dataloader = DataLoader(dataset, batch_size=3)`		
실습 가이드	이 실습에서는 예시 데이터를 사용하여 커스텀 데이터셋과 DataLoader를 생성하는 방법을 학습합니다. 100개의 샘플로 구성된 3차원 벡터 데이터를 생성하고, 이에 대응하는 이진 레이블을 만듭니다. 그런 다음, CustomDataset 클래스를 사용하여 데이터셋을 만들고, DataLoader를 통해 데이터를 배치 단위로 로드할 수 있도록 설정합니다.		
실습 요약	 예시 데이터를 사용해 커스텀 데이터셋과 DataLoader를 생성하고, 데이터를 배치 단위로 로드하는 과정을 실습합니다. 이 과정은 대규모 데이터셋을 효율적으로 처리할 때 필수적입니다.		

03-021 DataLoader에서 배치 조회

실습번호	03-021	실습명	DataLoader에서 배치 조회
실습 코드	`# DataLoader에서 모든 배치를 리스트로 변환` `all_batches = list(dataloader)` `batch_data, batch_labels = all_batches[0]  # 첫번째 배치 조회` `batch_data, batch_labels`		
실습 가이드	이 실습에서는 DataLoader에서 로드된 데이터를 배치 단위로 조회하는 방법을 학습합니다. DataLoader를 리스트로 변환하여 각 배치를 확인할 수 있으며, 첫 번째 배치를 조회하여 해당 배치의 데이터와 레이블을 확인합니다. 이를 통해 데이터셋에서 배치 단위로 데이터를 쉽게 가져올 수 있습니다.		
실습 요약	DataLoader에서 로드된 데이터를 배치 단위로 조회하고, 첫 번째 배치의 데이터를 확인하는 과정을 실습합니다. 배치 처리는 신경망 학습에서 중요한 역할을 합니다.		

03-022 DataLoader에서 마지막 배치 조회

실습번호	03-022	실습명	DataLoader에서 마지막 배치 조회
실습 코드	`batch_data, batch_labels = all_batches[-1]  # 마지막 배치 조회` `batch_data, batch_labels`		
실습 가이드	이 실습에서는 DataLoader에서 마지막 배치를 조회하는 방법을 학습합니다. 리스트의 음수 인덱스를 사용하여 마지막 배치를 가져올 수 있으며, 이를 통해 해당 배치의 데이터와 레이블을 확인할 수 있습니다. 마지막 배치는 전체 데이터셋의 크기와 배치 크기에 따라 일부 요소만 포함될 수 있습니다.		
실습 요약	DataLoader에서 마지막 배치를 조회하고, 해당 배치의 데이터를 확인하는 과정을 실습합니다. 이는 배치 처리에서 특정한 위치의 데이터를 조회할 때 유용합니다.		

03-023 DataLoader에서 변경된 배치 크기로 마지막 배치 조회

실습번호	03-023	실습명	DataLoader에서 변경된 배치 크기로 마지막 배치 조회
실습 코드	`dataloader = DataLoader(dataset, batch_size=7)` `all_batches = list(dataloader)` `batch_data, batch_labels = all_batches[-1]  # 마지막 배치 조회` `batch_data, batch_labels`		
실습 가이드	이 실습에서는 DataLoader의 배치 크기를 변경한 후, 마지막 배치를 조회하는 방법을 학습합니다. 배치 크기를 7로 설정하여 DataLoader에서 데이터를 로드한 뒤, 마지막 배치를 확인합니다. 변경된 배치 크기는 전체 데이터셋을 나누는 방식에 영향을 미치며, 마지막 배치의 크기는 나머지 데이터에 따라 달라질 수 있습니다.		
실습 요약	DataLoader의 배치 크기를 변경한 후, 마지막 배치를 조회하여 해당 데이터를 확인하는 과정을 실습합니다. 이는 배치 크기에 따른 데이터 처리의 변화를 이해하는 데 유용합니다.		

03-024 셔플된 DataLoader에서 마지막 배치 조회

실습번호	03-024	실습명	셔플된 DataLoader에서 마지막 배치 조회
실습 코드	`dataloader = DataLoader(dataset, batch_size=7, shuffle=True)` `all_batches = list(dataloader)` `batch_data, batch_labels = all_batches[-1]  # 마지막 배치 조회` `batch_data, batch_labels`		
실습 가이드	이 실습에서는 DataLoader에서 데이터를 셔플한 후 마지막 배치를 조회하는 방법을 학습합니다. DataLoader의 shuffle=True 옵션을 사용하여 데이터를 무작위로 섞은 뒤, 배치 크기를 7로 설정하여 데이터를 로드합니다. 이후, 마지막 배치를 조회하여 셔플된 데이터가 제대로 처리되는지 확인합니다.		
실습 요약	셔플된 DataLoader에서 마지막 배치를 조회하여 데이터를 확인하는 과정을 실습합니다. 셔플은 모델 학습 시 데이터의 순서에 의한 편향을 줄이는 데 유용합니다.		

03-025 마지막 배치를 버리는 DataLoader에서 마지막 배치 조회

실습번호	03-025	실습명	마지막 배치를 버리는 DataLoader에서 마지막 배치 조회
실습 코드	dataloader = DataLoader(dataset, batch_size=7, shuffle=True, drop_last=True) all_batches = list(dataloader) batch_data, batch_labels = all_batches[-1] # 마지막 배치 조회 batch_data, batch_labels		
실습 가이드	이 실습에서는 DataLoader에서 배치 크기에 맞지 않는 마지막 배치를 버린 후 남은 마지막 배치를 조회하는 방법을 학습합니다. drop_last=True 옵션을 사용하여 전체 데이터셋에서 남는 데이터가 있을 경우 이를 버리고, 나머지 데이터를 배치로 처리하지 않도록 설정합니다. 이후, 마지막 배치를 조회하여 데이터를 확인합니다.		
실습 요약	DataLoader에서 drop_last=True 옵션을 사용하여 마지막 남은 데이터를 버린 후, 남은 마지막 배치를 조회하는 과정을 실습합니다. 이는 특정 배치 크기에 맞추어 데이터를 균일하게 처리할 때 유용합니다.		

연습문제-03-004 커스텀 데이터셋과 DataLoader 생성

실습번호	연습문제-03-004	실습명	커스텀 데이터셋과 DataLoader 생성
실습 코드	# 새로운 랜덤 데이터 생성 data = torch.randn(150, 4) # 150개의 샘플, 각 샘플은 4차원 벡터 labels = torch.randint(0, 3, (150,)) # 3개의 클래스로 분류되는 150개의 레이블 # 커스텀 데이터셋과 DataLoader 생성 # 커스텀 데이터셋 클래스는 03-019 재활용 dataset = CustomDataset(data, labels) # 미니 배치 크기는 5 dataloader = DataLoader(dataset, batch_size=5)		
실습 가이드	이 실습에서는 150개의 샘플과 3개의 클래스 레이블을 가진 새로운 랜덤 데이터를 사용하여 커스텀 데이터셋을 생성하고, DataLoader를 설정하는 방법을 학습합니다. 데이터셋은 4차원 벡터로 구성되며, 이 데이터를 DataLoader를 통해 배치 단위로 처리할 수 있습니다.		
실습 요약	새로운 랜덤 데이터를 사용해 커스텀 데이터셋과 DataLoader를 생성하고, 데이터를 배치 단위로 로드하는 방법을 실습합니다.		

연습문제-03-005 DataLoader에서 배치 조회

실습번호	연습문제-03-005	실습명	DataLoader에서 배치 조회
실습 코드	 `# DataLoader에서 모든 배치를 리스트로 변환` `all_batches = list(dataloader)` `# 첫 번째 배치 조회` `batch_data, batch_labels = all_batches[0]  # 첫번째 배치의 데이터와 레이블 조회` `batch_data, batch_labels`		
실습 가이드	이 실습에서는 새로운 랜덤 데이터로 생성한 DataLoader에서 첫 번째 배치를 조회하는 방법을 학습합니다. 리스트로 변환된 DataLoader에서 첫 번째 배치의 데이터를 가져와 출력합니다.		
실습 요약	 DataLoader에서 새로운 랜덤 데이터를 사용하여 첫 번째 배치의 데이터를 확인하는 방법을 실습합니다.		

연습문제-03-006 DataLoader에서 마지막 배치 조회 (새로운 배치 크기)

실습번호	연습문제-03-006	실습명	DataLoader에서 마지막 배치 조회 (새로운 배치 크기)
실습 코드	# DataLoader에서 배치 크기를 7로 설정 dataloader = DataLoader(dataset, batch_size=7) # DataLoader에서 마지막 배치 조회 all_batches = list(dataloader) batch_data, batch_labels = all_batches[-1] # 마지막 배치의 데이터와 레이블 조회 batch_data, batch_labels		
실습 가이드	이 실습에서는 DataLoader의 배치 크기를 7로 설정하여 마지막 배치를 조회하는 방법을 학습합니다. 배치 크기가 변경됨에 따라 마지막 배치의 데이터가 어떻게 구성되는지 확인합니다.		
실습 요약	배치 크기를 7로 설정한 DataLoader에서 마지막 배치를 조회하는 방법을 실습합니다.		

연습문제-03-007 셔플된 DataLoader에서 마지막 배치 조회

실습번호	연습문제-03-007	실습명	셔플된 DataLoader에서 마지막 배치 조회
실습 코드	# DataLoader에서 shuffle=True로 설정하여 셔플된 데이터 로드 dataloader = DataLoader(dataset, batch_size=7, shuffle=True) # 셔플된 DataLoader에서 마지막 배치 조회 all_batches = list(dataloader) batch_data, batch_labels = all_batches[-1] # 마지막 배치의 데이터와 레이블 조회 batch_data, batch_labels		
실습 가이드	이 실습에서는 DataLoader에서 데이터를 셔플한 후 마지막 배치를 조회하는 방법을 학습합니다. shuffle=True 옵션을 사용하여 데이터를 무작위로 섞어 로드하고, 마지막 배치를 조회하여 셔플된 데이터를 확인합니다.		
실습 요약	셔플된 DataLoader에서 마지막 배치를 조회하여 데이터를 확인하는 과정을 실습합니다.		

연습문제-03-008 셔플되지 않은 DataLoader에서 마지막 배치 조회

실습번호	연습문제-03-008	실습명	셔플되지 않은 DataLoader에서 마지막 배치 조회
실습 코드	 # 셔플되지 않은 DataLoader에서 마지막 배치 조회 dataloader = DataLoader(dataset, batch_size=7, shuffle=False) # 셔플되지 않은 DataLoader에서 마지막 배치 조회 all_batches = list(dataloader) batch_data, batch_labels = all_batches[-1] # 마지막 배치의 데이터와 레이블 조회 batch_data, batch_labels		
실습 가이드	이 실습에서는 shuffle=False로 설정된 DataLoader에서 마지막 배치를 조회하는 방법을 학습합니다. 데이터가 셔플되지 않은 상태에서 로드된 마지막 배치를 확인합니다.		
실습 요약	셔플되지 않은 DataLoader에서 마지막 배치를 조회하고, 데이터를 확인하는 방법을 실습합니다.		

04. 인공지능-DNN

04-001 PyTorch 및 관련 모듈 임포트와 device 설정

실습번호	04-001	실습명	PyTorch 및 관련 모듈 임포트와 device 설정
실습 코드			

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torchinfo import summary
from JAEN.utils import plot_training_results

# device 설정 (GPU가 사용 가능하면 GPU로, 그렇지 않으면 CPU 사용)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

실습 가이드	이 실습에서는 PyTorch와 관련된 모듈을 임포트하고, 신경망을 GPU 또는 CPU에서 실행할 수 있도록 device를 설정하는 방법을 학습합니다. torch.device()를 사용하여 GPU 사용 가능 여부에 따라 적절한 장치를 선택합니다.
실습 요약	PyTorch의 다양한 모듈을 임포트하고, 학습에 사용할 장치(GPU 또는 CPU)를 설정하는 과정을 실습합니다. 이를 통해 GPU 가속을 활용한 빠른 연산이 가능해집니다.

04-002 FashionMNIST 데이터 변환 및 정규화

실습번호	04-002	실습명	FashionMNIST 데이터 변환 및 정규화
실습 코드	`# FashionMNIST 데이터 변환 (이미지를 텐서로 변환하고 [0, 1] 범위로 정규화)` `transform = transforms.Compose([` `  transforms.ToTensor(),` `])`		
실습 가이드	이 실습에서는 FashionMNIST 데이터를 신경망에 입력할 수 있도록 텐서로 변환하고, 데이터를 정규화하는 방법을 학습합니다. transforms.ToTensor()는 이미지를 텐서로 변환하고, transforms.Normalize()는 평균과 표준편차를 이용해 데이터를 정규화하여 모델 학습 시 안정성을 높입니다.		
실습 요약	FashionMNIST 데이터를 텐서로 변환하고, 정규화를 적용하는 과정을 실습합니다. 데이터 정규화는 모델의 학습 성능을 향상시키는 중요한 전처리 과정입니다.		

04-003 FashionMNIST 학습 및 테스트 데이터셋 로드

실습번호	04-003	실습명	FashionMNIST 학습 및 테스트 데이터셋 로드
실습 코드	# 학습 및 테스트 데이터셋 로드 train_dataset = datasets.FashionMNIST(root='./data', train=True, transform=transform, download=True) test_dataset = datasets.FashionMNIST(root='./data', train=False, transform=transform, download=True)		
실습 가이드	이 실습에서는 FashionMNIST 데이터셋을 학습용과 테스트용으로 로드하는 방법을 학습합니다. train=True로 설정된 데이터셋은 학습용 데이터로, train=False로 설정된 데이터셋은 테스트용 데이터로 로드됩니다. transform을 통해 앞서 정의한 이미지 변환 및 정규화가 적용됩니다.		
실습 요약	FashionMNIST 데이터셋을 학습용과 테스트용으로 로드하는 과정을 실습합니다. 이는 신경망 학습 및 평가를 위한 필수적인 단계입니다.		

04-004 데이터 로더 생성

실습번호	04-004	실습명	데이터 로더 생성
실습 코드	# 데이터 로더 생성 train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True) test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)		
실습 가이드	이 실습에서는 학습 및 테스트 데이터셋을 배치 단위로 로드하기 위해 DataLoader를 생성하는 방법을 학습합니다. train_loader는 배치 크기가 64이고 데이터를 무작위로 섞으며, test_loader는 배치 크기가 64이고 데이터를 순차적으로 로드합니다. DataLoader는 대규모 데이터셋을 처리할 때 매우 유용합니다.		
실습 요약	학습 및 테스트 데이터셋을 처리할 DataLoader를 생성하는 과정을 실습합니다. 배치 처리를 통해 메모리 효율성을 높이고, 데이터셋을 효과적으로 로드할 수 있습니다.		

04-005 4차원 텐서 Flatten

<table>
<tr><td>실습번호</td><td>04-005</td><td>실습명</td><td>4차원 텐서 Flatten</td></tr>
<tr><td>실습 코드</td><td colspan="3"><pre># 4차원 텐서 예시 (배치 크기 1, 채널 1, 높이 2, 너비 2)
x = torch.tensor([[[[1, 2], [3, 4]]]])
flatten = nn.Flatten() # Flatten 모듈 생성
output = flatten(x) # Flatten 적용
output # 출력 텐서: [[1, 2, 3, 4]]</pre></td></tr>
<tr><td>실습 가이드</td><td colspan="3">이 실습에서는 4차원 텐서를 2차원 텐서로 변환하는 Flatten 모듈의 사용 방법을 학습합니다. nn.Flatten()을 사용하여 배치 크기와 채널을 유지하면서 나머지 차원을 평탄화합니다. 예제에서는 4개의 요소로 구성된 2x2 텐서가 1x4 텐서로 변환됩니다.</td></tr>
<tr><td>실습 요약</td><td colspan="3">4차원 텐서를 Flatten 모듈을 사용하여 2차원 텐서로 변환하는 과정을 실습합니다. Flatten은 신경망의 전방향 전파에서 유용하게 사용됩니다.</td></tr>
</table>

04-006 nn.Sequential 기반 신경망 모델 구성

실습번호	04-006	실습명	nn.Sequential 기반 신경망 모델 구성
실습 코드	``` model = nn.Sequential(nn.Flatten(), # 28x28 이미지를 1차원 벡터로 펼침 nn.Linear(28*28, 128), # 입력: 28*28 픽셀, 출력: 128개의 노드 nn.ReLU(), # ReLU 활성화 함수 nn.Linear(128, 10) # 출력층: 10개의 클래스) # 모델 인스턴스 생성 model.to(device) summary(model, (32, 1, 28, 28)) ```		
실습 가이드	이 실습에서는 nn.Sequential()을 사용하여 간단한 신경망 모델을 구성하는 방법을 학습합니다. 이 모델은 입력 이미지의 크기가 28x28인 경우, Flatten을 통해 1차원 벡터로 변환한 뒤, 두 개의 선형 레이어와 ReLU 활성화 함수를 사용합니다. 마지막으로 10개의 클래스를 출력하는 레이어를 추가합니다. summary() 함수를 사용하여 모델의 구조를 요약하고, 각 레이어의 파라미터 수를 확인합니다.		
실습 요약	신경망 모델을 구성하고, 요약 정보를 통해 모델의 구조와 파라미터 수를 확인하는 과정을 실습합니다. 이 과정은 모델의 설계를 이해하고 조정하는 데 중요한 단계입니다.		

04-007 nn.Module 기반 신경망 모델 구성

실습번호	04-007	실습명	nn.Module 기반 신경망 모델 구성

실습 코드

```
class FashionMNISTModel(nn.Module):
    def __init__(self):
        super(FashionMNISTModel, self).__init__()
        self.flatten = nn.Flatten()  # 28x28 이미지를 1차원 벡터로 펼침
        self.fc1 = nn.Linear(28*28, 128)  # 입력: 28*28 픽셀, 출력: 128개의 노드
        self.fc2 = nn.Linear(128, 10)     # 출력층: 10개의 클래스

    def forward(self, x):
        x = self.flatten(x)
        x = torch.relu(self.fc1(x))  # ReLU 활성화 함수 적용
        x = self.fc2(x)  # 마지막 출력에는 활성화 함수를 사용하지 않음 (CrossEntropyLoss에서 처리)
        return x

# 모델 인스턴스 생성
model = FashionMNISTModel().to(device)
summary(model, (10, 1, 28, 28))
```

실습 가이드

이 실습에서는 nn.Module을 상속하여 FashionMNIST를 위한 신경망 모델 클래스를 구현합니다. __init__() 메서드에서 Flatten 레이어와 두 개의 선형 레이어를 정의하며, forward() 메서드에서 데이터 흐름을 정의합니다. 마지막 출력 레이어는 CrossEntropyLoss에서 직접 활성화를 처리하므로 활성화 함수를 사용하지 않습니다. summary() 함수를 사용하여 모델의 구조를 요약합니다.

실습 요약

FashionMNIST를 위한 신경망 모델 클래스를 구현하고, 요약 정보를 통해 모델의 구조와 파라미터 수를 확인하는 과정을 실습합니다. 이 과정은 모델 설계를 이해하고 조정하는 데 중요합니다.

04-008 모델 학습 함수 구현

실습번호	04-008	실습명	모델 학습 함수 구현
실습 코드	def train(model, train_loader, criterion, optimizer, epoch, device): model.train() # 모델을 학습 모드로 설정 running_loss = 0.0 correct = 0 total = 0 for images, labels in train_loader: images, labels = images.to(device), labels.to(device) outputs = model(images) loss = criterion(outputs, labels) # 역전파 및 옵티마이저 스텝 optimizer.zero_grad() loss.backward() optimizer.step() running_loss += loss.item() # 정확도 계산 _, predicted = torch.max(outputs, 1) total += labels.size(0) correct += (predicted == labels).sum().item() train_loss = running_loss / len(train_loader) train_accuracy = 100 * correct / total print(f'Epoch [{epoch+1}]') print(f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%') return train_loss, train_accuracy		
실습 가이드	이 실습에서는 PyTorch 모델을 학습하기 위한 train() 함수를 구현합니다. 모델을 학습 모드로 설정하고, 데이터 로더에서 배치 단위로 이미지를 가져와 모델의 출력과 손실을 계산합니다. 손실에 대해 역전파를 수2행하고 옵티마이저를 사용하여 파라미터를 업데이트합니다. 마지막으로, 전체 손실과 정확도를 계산하여 출력합니다.		
실습 요약	모델 학습을 위한 train() 함수를 구현하고, 각 에포크에서 손실과 정확도를 출력하는 과정을 실습합니다. 이 함수는 신경망을 학습시키는 데 필수적인 구성 요소입니다.		

04-009 모델 평가 함수 구현

실습번호	04-009	실습명	모델 평가 함수 구현

실습 코드

```
# 평가 함수 정의
def evaluate(model, test_loader, criterion, device):
    model.eval()  # 모델을 평가 모드로 설정
    test_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():  # 평가 중에는 기울기 계산을 하지 않음
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()

            # 예측 정확도 계산
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    test_loss /= len(test_loader)
    test_accuracy = 100 * correct / total
    print(f'Test Loss: {test_loss:.4f}, Test Accuracy:
{test_accuracy:.2f}%')

    return test_loss, test_accuracy
```

실습 가이드

이 실습에서는 모델을 평가하기 위한 evaluate() 함수를 구현합니다. 모델을 평가 모드로 설정하고, 기울기 계산을 하지 않도록 torch.no_grad() 블록을 사용합니다. 테스트 데이터 로더에서 배치 단위로 데이터를 가져와 모델의 출력과 손실을 계산합니다. 마지막으로, 전체 테스트 손실과 정확도를 계산하여 출력합니다.

실습 요약

모델 평가를 위한 evaluate() 함수를 구현하고, 테스트 데이터에서 손실과 정확도를 출력하는 과정을 실습합니다. 이 함수는 모델의 성능을 측정하는 데 필수적입니다.

04-010 학습 및 평가 과정 관리 함수 구현

실습번호	04-010	실습명	학습 및 평가 과정 관리 함수 구현

실습 코드

```
# 학습 및 평가 과정 관리
def train_and_evaluate(model, train_loader, test_loader, criterion, optimizer, num_epochs, device):
    train_losses = []
    train_accuracies = []
    test_losses = []
    test_accuracies = []

    for epoch in range(num_epochs):
        # 모델 학습(학습데이터)
        train_loss, train_accuracy = train(model, train_loader, criterion, optimizer, epoch, device)
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)

        # 모델 평가 (평가데이터)
        test_loss, test_accuracy = evaluate(model, test_loader, criterion, device)
        test_losses.append(test_loss)
        test_accuracies.append(test_accuracy)

    return train_losses, train_accuracies, test_losses, test_accuracies
```

실습 가이드

이 실습에서는 모델의 학습과 평가 과정을 관리하는 train_and_evaluate() 함수를 구현합니다. 주어진 에포크

수만큼 학습을 반복하고, 매 에포크마다 학습 손실 및 정확도를 기록합니다. 학습 후에는 테스트 데이터셋에서 모델을 평가하고 손실과 정확도를 기록합니다. 최종적으로 학습 및 평가 결과를 리스트로 반환합니다.

실습 요약

모델의 학습 및 평가 과정을 관리하는 train_and_evaluate() 함수를 구현하는 과정을 실습합니다. 이 함수는 모델 훈련 및 성능 분석에 중요한 역할을 합니다.

04-011 손실 함수 및 옵티마이저 설정과 학습 수행

실습번호	04-011	실습명	손실 함수 및 옵티마이저 설정과 학습 수행
실습 코드	 `# 손실 함수와 옵티마이저 설정` `criterion = nn.CrossEntropyLoss()  # 다중 클래스 분류를 위한 손실 함수` `optimizer = optim.Adam(model.parameters(), lr=0.01)  # Adam 옵티마이저` `train_losses, train_accuracies, test_losses, test_accuracies =` `train_and_evaluate(` `    model, train_loader, test_loader, criterion, optimizer, num_epochs=10,` `device=device` `)`		
실습 가이드	이 실습에서는 모델 학습을 위한 손실 함수와 옵티마이저를 설정합니다. nn.CrossEntropyLoss()는 다중 클래스 분류 문제에 적합한 손실 함수로, 각 클래스의 확률을 기반으로 손실을 계산합니다. Adam 옵티마이저는 학습률(lr)을 0.01로 설정하여 모델의 파라미터를 업데이트합니다. 이후, train_and_evaluate() 함수를 호출하여 모델을 학습하고 평가합니다.		
실습 요약	 손실 함수와 옵티마이저를 설정한 후, 모델을 학습하고 평가하는 과정을 실습합니다. 이 과정은 신경망 모델의 성능을 최적화하는 데 필수적입니다.		

04-012 드롭아웃을 포함한 FashionMNIST 모델 클래스 구현

실습번호	04-012	실습명	드롭아웃을 포함한 FashionMNIST 모델 클래스 구현

실습 코드

```
# 신경망 모델 정의 (Dropout 포함)
class FashionMNISTDropoutModel(nn.Module):
    def __init__(self):
        super(FashionMNISTDropoutModel, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 10)
        self.dropout = nn.Dropout(0.1)  # 10%의 드롭아웃 적용

    def forward(self, x):
        x = self.flatten(x)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)  # 첫 번째 은닉층 뒤에 드롭아웃 적용
        x = self.fc2(x)  # 출력층에는 드롭아웃을 사용하지 않음
        return x

# 모델 인스턴스화
model = FashionMNISTDropoutModel().to(device)
summary(model, (10, 1, 28, 28))
```

실습 가이드

이 실습에서는 드롭아웃(Dropout) 기법을 포함한 FashionMNIST 모델 클래스를 구현합니다. nn.Dropout()을 사용하여 과적합을 방지하기 위해 첫 번째 은닉층 뒤에 드롭아웃을 적용합니다. 드롭아웃 비율은 0.1로 설정하여 학습 중 무작위로 뉴런을 비활성화합니다. 모델의 구조는 summary() 함수를 통해 확인합니다.

실습 요약

드롭아웃을 포함한 FashionMNIST 신경망 모델을 구현하고, 요약 정보를 통해 모델의 구조와 파라미터 수를 확인하는 과정을 실습합니다. 드롭아웃은 신경망의 일반화 성능을 향상시키는 데 효과적인 기법입니다.

04-013 손실 함수 및 옵티마이저 설정과 학습 수행

실습번호	04-013	실습명	손실 함수 및 옵티마이저 설정과 학습 수행
실습 코드	 `# 손실 함수와 옵티마이저 설정` `criterion = nn.CrossEntropyLoss()  # 다중 클래스 분류를 위한 손실 함수` `optimizer = optim.Adam(model.parameters(), lr=0.01)  # Adam 옵티마이저` `train_losses, train_accuracies, test_losses, test_accuracies =` `train_and_evaluate(` `    model, train_loader, test_loader, criterion, optimizer, num_epochs=10,` `device=device` `)`		
실습 가이드	이 실습에서는 모델 학습을 위한 손실 함수와 옵티마이저를 설정합니다. nn.CrossEntropyLoss()는 다중 클래스 분류 문제에 적합한 손실 함수로, 각 클래스의 확률을 기반으로 손실을 계산합니다. Adam 옵티마이저는 학습률(lr)을 0.01로 설정하여 모델의 파라미터를 업데이트합니다. 이후, train_and_evaluate() 함수를 호출하여 모델을 학습하고 평가합니다.		
실습 요약	 손실 함수와 옵티마이저를 설정한 후, 모델을 학습하고 평가하는 과정을 실습합니다. 이 과정은 신경망 모델의 성능을 최적화하는 데 필수적입니다.		

04-014 Batch Normalization 및 Dropout 포함 모델 정의

실습번호	04-014	실습명	Batch Normalization 및 Dropout 포함 모델 정의
실습 코드	# 신경망 모델 정의 (Batch Normalization 및 Dropout 포함) class FashionMNISTBNModel(nn.Module): def __init__(self): super(FashionMNISTBNModel, self).__init__() self.flatten = nn.Flatten() self.fc1 = nn.Linear(28*28, 128) self.bn1 = nn.BatchNorm1d(128) # 첫 번째 배치 정규화 레이어 self.fc2 = nn.Linear(128, 10) self.dropout = nn.Dropout(0.1) # 10%의 드롭아웃 적용 def forward(self, x): x = self.flatten(x) x = torch.relu(self.bn1(self.fc1(x))) # 첫 번째 배치 정규화 + ReLU x = self.dropout(x) # 드롭아웃 적용 x = self.fc2(x) # 출력층에는 배치 정규화 및 드롭아웃을 사용하지 않음 return x # 모델 인스턴스화 model = FashionMNISTBNModel().to(device) summary(model, (10, 1, 28, 28))		
실습 가이드	이 실습에서는 Batch Normalization과 Dropout을 포함한 FashionMNIST 모델을 정의합니다. nn.BatchNorm1d()를 사용하여 첫 번째 선형 레이어의 출력을 정규화하고, nn.Dropout()을 통해 과적합을 방지하기 위해 드롭아웃을 적용합니다. 모델의 구조는 summary() 함수를 통해 확인합니다.		
실습 요약	Batch Normalization 및 Dropout을 포함한 FashionMNIST 신경망 모델을 구현하고, 요약 정보를 통해 모델의 구조와 파라미터 수를 확인하는 과정을 실습합니다. 이 기법들은 신경망의 학습 성능을 향상시키는 데 도움을 줍니다.		

04-015 손실 함수 및 옵티마이저 설정과 학습 수행

실습번호	04-015	실습명	손실 함수 및 옵티마이저 설정과 학습 수행
실습 코드	 `# 손실 함수와 옵티마이저 설정` `criterion = nn.CrossEntropyLoss()  # 다중 클래스 분류를 위한 손실 함수` `optimizer = optim.Adam(model.parameters(), lr=0.01)  # Adam 옵티마이저` `train_losses, train_accuracies, test_losses, test_accuracies =` `train_and_evaluate(` `    model, train_loader, test_loader, criterion, optimizer, num_epochs=10,` `device=device` `)`		
실습 가이드	이 실습에서는 모델 학습을 위한 손실 함수와 옵티마이저를 설정합니다. nn.CrossEntropyLoss()는 다중 클래스 분류 문제에 적합한 손실 함수로, 각 클래스의 확률을 기반으로 손실을 계산합니다. Adam 옵티마이저는 학습률(lr)을 0.01로 설정하여 모델의 파라미터를 업데이트합니다. 이후, train_and_evaluate() 함수를 호출하여 모델을 학습하고 평가합니다.		
실습 요약	 손실 함수와 옵티마이저를 설정한 후, 모델을 학습하고 평가하는 과정을 실습합니다. 이 과정은 신경망 모델의 성능을 최적화하는 데 필수적입니다.		

04-016 L2 정칙화를 위한 옵티마이저 설정

실습번호	04-016	실습명	L2 정칙화를 위한 옵티마이저 설정
실습 코드	`# L2 정칙화를 위한 옵티마이저 설정 (weight_decay가 L2 정칙화)` `optimizer = torch.optim.SGD(model.parameters(), lr=0.01,` `weight_decay=0.001)`		
실습 가이드	이 실습에서는 L2 정칙화(L2 regularization)를 적용하기 위해 SGD(Stochastic Gradient Descent) 옵티마이저를 설정합니다. weight_decay 매개변수를 사용하여 L2 정칙화 항을 추가함으로써 과적합을 방지할 수 있습니다. 이는 모델의 일반화 성능을 향상시키는 데 중요한 역할을 합니다.		
실습 요약	L2 정칙화를 적용한 옵티마이저를 설정하는 과정을 실습합니다. 이는 신경망 학습에서 모델의 일반화 성능을 개선하는 데 효과적입니다.		

연습문제-04-001 Sequential 방식으로 2차원 더미 데이터를 이용한 다중 분류 모델 설계

실습번호	연습문제-04-001	실습명	Sequential 방식으로 2차원 더미 데이터를 이용한 다중 분류 모델 설계
실습 코드	 `# Sequential 방식으로 다중 분류 모델 설계` `model = nn.Sequential(` `  nn.Linear(2, 32),` `  nn.ReLU(),` `  nn.Linear(32, 16),` `  nn.ReLU(),` `  nn.Linear(16, 3),` `  nn.Softmax(dim=1)` `)`		
실습 가이드	이 실습에서는 PyTorch의 Sequential API를 사용하여 다중 분류 모델을 설계하는 방법을 학습합니다. Sequential 방식은 모델의 각 계층을 순차적으로 정의하며, 간단한 네트워크를 설계할 때 유용합니다. 먼저, 2차원 입력을 받는 네트워크를 설계하게 됩니다. 첫 번째 층에서는 입력 데이터(2차원)를 32개의 노드로 확장합니다. 이어서 ReLU 활성화 함수를 적용하여 비선형성을 추가합니다. 두 번째 층에서는 32개의 노드를 16개의 노드로 줄이며, 마찬가지로 ReLU 활성화 함수를 적용합니다. 마지막으로, 출력층에서 3개의 클래스를 분류하는 Softmax 함수가 적용됩니다. 이 실습을 통해 다중 분류 문제에서 Sequential API를 어떻게 활용하는지, 각 층의 역할과 활성화 함수가 모델 성능에 어떻게 기여하는지에 대해 학습할 수 있습니다. 또한, Softmax 함수가 분류 문제에서 어떻게 확률 기반의 출력을 생성하는지 이해하게 됩니다.		
실습 요약	PyTorch Sequential API를 사용하여 2차원 더미 데이터를 기반으로 다중 분류 DNN 모델을 설계하고, 각 계층의 역할과 모델 구조를 학습합니다.		

연습문제-04-002 Module 방식으로 2차원 더미 데이터를 이용한 회귀 모델 설계

실습번호	연습문제-04-002	실습명	Module 방식으로 2차원 더미 데이터를 이용한 회귀 모델 설계
실습 코드			

```
# Module 방식으로 회귀 모델 설계
class RegressionModel(nn.Module):
    def __init__(self):
        super(RegressionModel, self).__init__()
        self.layer1 = nn.Linear(2, 64)
        self.layer2 = nn.Linear(64, 32)
        self.layer3 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = self.layer3(x)
        return x
```

실습 가이드	이 실습에서는 PyTorch의 Module 방식을 사용하여 회귀 문제를 해결하는 모델을 설계하는 방법을 학습합니다. Module 방식을 사용하면, 모델의 계층을 더 세밀하게 정의할 수 있으며, 복잡한 네트워크 구조나 맞춤형 로직을 적용할 때 유리합니다. 모델은 2차원 입력을 받으며, 첫 번째 층에서 64개의 노드로 확장한 뒤 ReLU 활성화 함수를 적용합니다. 두 번째 층에서는 64개의 노드를 32개의 노드로 줄이고, 마찬가지로 ReLU 활성화 함수를 사용하여 비선형성을 부여합니다. 마지막으로, 출력층에서 단일 회귀 값을 예측합니다. 이 실습을 통해 PyTorch의 Module 클래스를 사용하여 회귀 모델을 설계하고, 각 계층의 역할 및 활성화 함수가 어떻게 회귀 문제 해결에 기여하는지 학습할 수 있습니다. 또한, ReLU 활성화 함수가 네트워크 학습에서 어떤 역할을 하는지, 회귀 문제에서 출력층이 어떻게 동작하는지 이해하게 됩니다.
실습 요약	PyTorch Module 클래스를 사용하여 2차원 더미 데이터를 기반으로 회귀 모델을 설계하고, 각 계층의 역할과 활성화 함수의 중요성을 학습합니다.

05. 인공지능-CNN

05-001 2D 컨볼루션 출력 크기 계산 함수 구현

실습번호	05-001	실습명	2D 컨볼루션 출력 크기 계산 함수 구현
실습 코드	```def conv2d_output_size(input_size, kernel_size, stride=1, padding=0): height, width = input_size # Convolution 공식 적용 out_height = (height + 2 * padding - kernel_size) // stride + 1 out_width = (width + 2 * padding - kernel_size) // stride + 1 return out_height, out_width```		
실습 가이드	이 실습에서는 2D 컨볼루션 레이어의 출력 크기를 계산하는 함수를 구현합니다. 입력 크기, 커널 크기, 스트라이드, 패딩을 매개변수로 받아 컨볼루션 계산 공식을 사용하여 출력 높이와 너비를 반환합니다. 이 함수는 CNN 모델 설계 시 출력 크기를 예측하는 데 유용합니다.		
실습 요약	2D 컨볼루션의 출력 크기를 계산하는 함수를 구현하는 과정을 실습합니다. 이 함수는 신경망 설계 시 중요한 역할을 합니다.		

05-002 컨볼루션 출력 크기 계산

실습번호	05-002	실습명	컨볼루션 출력 크기 계산
실습 코드	`# 입력 크기 (Height, Width), 커널 크기, 스트라이드, 패딩` `output_size = conv2d_output_size((28, 28), 3, 1, 0)` `print(f"Output feature map size: {output_size}")`		
실습 가이드	이 실습에서는 앞서 구현한 conv2d_output_size() 함수를 사용하여 컨볼루션 레이어의 출력 크기를 계산합니다. 입력으로 28x28 크기의 이미지와 3x3 커널, 스트라이드 1, 패딩 0을 설정하여 출력 feature map의 크기를 확인합니다.		
실습 요약	입력 이미지와 커널의 크기를 사용하여 컨볼루션의 출력 크기를 계산하고 확인하는 과정을 실습합니다. 이를 통해 CNN 모델의 구조를 이해하는 데 도움이 됩니다.		

05-003 패딩을 포함한 컨볼루션 출력 크기 계산

실습번호	05-003	실습명	패딩을 포함한 컨볼루션 출력 크기 계산
실습 코드	`# 입력 크기 (Height, Width), 커널 크기, 스트라이드, 패딩` `output_size = conv2d_output_size((28, 28), 3, 1, 1)` `print(f"Output feature map size: {output_size}")`		
실습 가이드	이 실습에서는 conv2d_output_size() 함수를 사용하여 패딩을 포함한 컨볼루션 레이어의 출력 크기를 계산합니다. 입력으로 28x28 크기의 이미지와 3x3 커널, 스트라이드 1, 패딩 1을 설정하여 출력 feature map의 크기를 확인합니다. 이 과정은 CNN 설계 시 출력 크기를 예측하는 데 중요합니다.		
실습 요약	패딩을 포함하여 컨볼루션의 출력 크기를 계산하고 확인하는 과정을 실습합니다. 패딩의 효과를 이해하는 데 도움이 됩니다.		

05-004 스트라이드를 포함한 컨볼루션 출력 크기 계산

실습번호	05-004	실습명	스트라이드를 포함한 컨볼루션 출력 크기 계산
실습 코드	`# 입력 크기 (Height, Width), 커널 크기, 스트라이드, 패딩` `output_size = conv2d_output_size((28, 28), 3, 2, 0)` `print(f"Output feature map size: {output_size}")`		
실습 가이드	이 실습에서는 conv2d_output_size() 함수를 사용하여 스트라이드를 포함한 컨볼루션 레이어의 출력 크기를 계산합니다. 입력으로 28x28 크기의 이미지와 3x3 커널, 스트라이드 2, 패딩 0을 설정하여 출력 feature map의 크기를 확인합니다. 이 과정은 CNN 모델 설계 시 중요한 요소입니다.		
실습 요약	스트라이드를 포함하여 컨볼루션의 출력 크기를 계산하고 확인하는 과정을 실습합니다. 스트라이드의 효과를 이해하는 데 도움이 됩니다.		

05-005 스트라이드 및 패딩을 포함한 컨볼루션 출력 크기 계산

실습번호	05-005	실습명	스트라이드 및 패딩을 포함한 컨볼루션 출력 크기 계산
실습 코드	`# 입력 크기 (Height, Width), 커널 크기, 스트라이드, 패딩` `output_size = conv2d_output_size((28, 28), 3, 2, 1)` `print(f"Output feature map size: {output_size}")`		
실습 가이드	이 실습에서는 conv2d_output_size() 함수를 사용하여 스트라이드와 패딩을 모두 포함한 컨볼루션 레이어의 출력 크기를 계산합니다. 입력으로 28x28 크기의 이미지와 3x3 커널, 스트라이드 2, 패딩 1을 설정하여 출력 feature map의 크기를 확인합니다. 이 과정은 CNN 설계 시 중요한 요소입니다.		
실습 요약	스트라이드와 패딩을 포함하여 컨볼루션의 출력 크기를 계산하고 확인하는 과정을 실습합니다. 이를 통해 CNN의 동작을 더 깊이 이해할 수 있습니다.		

05-006 CNN 모델 클래스 정의

실습번호	05-006	실습명	CNN 모델 클래스 정의

실습 코드

```
# CNN 모델 정의
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        # 첫 번째 컨볼루션 레이어
        # 입력 채널: 1 (흑백 이미지), 출력 채널: 16, 커널 크기: 3x3, 패딩:
1
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16,
kernel_size=3, padding=1)

        # 두 번째 컨볼루션 레이어
        # 입력 채널: 16, 출력 채널: 32, 커널 크기: 3x3, 패딩: 1
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32,
kernel_size=3, padding=1)

        # MaxPool 레이어 (다운샘플링)
        # 커널 크기: 2x2, 스트라이드: 2
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # 첫 번째 완전 연결 (Fully Connected) 레이어
        # 입력 크기: 32 * 7 * 7 (Conv2d 출력을 펼친 크기), 출력 크기: 128
        self.fc1 = nn.Linear(32 * 7 * 7, 128)

        # 두 번째 완전 연결 레이어
        # 입력 크기: 128, 출력 크기: 10 (클래스 개수)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        # 입력 데이터 크기: (batch_size, 1, 28, 28)

        # 첫 번째 Conv + ReLU + MaxPool
        # Conv 후 크기: (batch_size, 16, 28, 28)
        # MaxPool 후 크기: (batch_size, 16, 14, 14)
        x = self.pool(torch.relu(self.conv1(x)))

        # 두 번째 Conv + ReLU + MaxPool
        # Conv 후 크기: (batch_size, 32, 14, 14)
        # MaxPool 후 크기: (batch_size, 32, 7, 7)
        x = self.pool(torch.relu(self.conv2(x)))
```

	`` # Flatten: Conv 출력을 1차원 벡터로 펼침 # Flatten 후 크기: (batch_size, 32 * 7 * 7) x = x.reshape(-1, 32 * 7 * 7) # 첫 번째 Fully Connected + ReLU # 출력 크기: (batch_size, 128) x = torch.relu(self.fc1(x)) # 두 번째 Fully Connected (출력층) # 출력 크기: (batch_size, 10) x = self.fc2(x) return x # 모델 요약 출력 model = CNN().to(device) summary(model, input_size=(64, 1, 28, 28)) ```
실습 가이드	이 실습에서는 CNN(Convolutional Neural Network) 모델을 정의합니다. 두 개의 컨볼루션 레이어, 각 레이어 뒤에 배치 정규화 및 드롭아웃을 적용하고, 최종적으로 두 개의 완전 연결 레이어를 사용하여 FashionMNIST 데이터셋의 분류 문제를 해결합니다. summary() 함수를 통해 모델 구조를 확인합니다.
실습 요약	CNN 모델 클래스를 구현하고, 요약 정보를 통해 모델의 구조와 파라미터 수를 확인하는 과정을 실습합니다. CNN은 이미지 분류 문제에 효과적인 신경망 구조입니다.

05-007 손실 함수 및 옵티마이저 설정과 학습 수행

실습번호	05-007	실습명	손실 함수 및 옵티마이저 설정과 학습 수행
실습 코드	 # 손실 함수와 옵티마이저 설정 criterion = nn.CrossEntropyLoss() # 다중 클래스 분류를 위한 손실 함수 optimizer = optim.Adam(model.parameters(), lr=0.0001) # Adam 옵티마이저 train_losses, train_accuracies, test_losses, test_accuracies = train_and_evaluate(model, train_loader, test_loader, criterion, optimizer, num_epochs=10, device=device) # 결과 시각화 plot_training_results(train_losses, train_accuracies, test_losses, test_accuracies)		
실습 가이드	이 실습에서는 모델 학습을 위한 손실 함수와 옵티마이저를 설정합니다. nn.CrossEntropyLoss()는 다중 클래스 분류 문제에 적합한 손실 함수로, 각 클래스의 확률을 기반으로 손실을 계산합니다. Adam 옵티마이저는 학습률(lr)을 0.0001로 설정하여 모델의 파라미터를 업데이트합니다. 이후, train_and_evaluate() 함수를 호출하여 모델을 학습하고 평가합니다.		
실습 요약	손실 함수와 옵티마이저를 설정한 후, 모델을 학습하고 평가하는 과정을 실습합니다. 이 과정은 신경망 모델의 성능을 최적화하는 데 필수적입니다.		

연습문제-05-001 (배치, 3, 32, 32) 입력을 받는 CNN 모델 설계 (Sequential 방식)

실습번호	연습문제-05-001	실습명	(배치, 3, 32, 32) 입력을 받는 CNN 모델 설계 (Sequential 방식)
실습 코드	 # (배치, 3, 32, 32) 입력을 받는 CNN 모델 설계 model = nn.Sequential(nn.Conv2d(3, 16, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(kernel_size=2, stride=2), nn.Conv2d(16, 32, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(kernel_size=2, stride=2), nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(kernel_size=2, stride=2), nn.Flatten(), nn.Linear(64 * 4 * 4, 128), nn.ReLU(), nn.Linear(128, 10))		
실습 가이드	이 실습에서는 PyTorch Sequential 방식을 사용하여 (배치, 3, 32, 32) 형태의 이미지 입력을 받는 CNN 모델을 설계하는 방법을 학습합니다. 이 CNN 모델은 세 개의 합성곱 층을 사용하며, 각 층의 뒤에는 ReLU 활성화 함수와 Max Pooling을 적용합니다. 모델은 64개의 필터를 가진 마지막 합성곱 층에서 특징을 추출하고, 이를 완전히 연결된 층으로 전달합니다. 마지막 출력은 10개의 클래스를 나타냅니다. 학습할 때, 이 모델은 (배치, 3, 32, 32) 형태의 컬러 이미지를 입력으로 받아 분류 작업을 수행할 수 있습니다. 각 이미지의 높이와 너비는 32x32 픽셀이며, 채널은 RGB(3개)로 구성됩니다.		
실습 요약	(배치, 3, 32, 32) 형태의 입력을 받는 CNN 모델을 Sequential 방식으로 설계하여 이미지 분류 문제를 해결하는 방법을 실습합니다. CNN의 기본 구조와 작동 원리에 대해 익히며, 합성곱, ReLU 활성화 함수, Max Pooling, 그리고 완전 연결 층의 역할을 이해합니다.		

연습문제-05-002 (배치, 3, 224, 224) 입력을 받는 CNN 모델 설계 (Module 방식)

실습번호	연습문제-05-002	실습명	(배치, 3, 224, 224) 입력을 받는 CNN 모델 설계 (Module 방식)

실습 코드

```
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()

        # 첫 번째 합성곱층 및 활성화 함수, 풀링층
        self.conv1 = nn.Conv2d(3, 256, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        # 두 번째 합성곱층 및 풀링층
        self.conv2 = nn.Conv2d(256, 128, kernel_size=3, padding=1)
        # 세 번째 합성곱층 및 풀링층
        self.conv3 = nn.Conv2d(128, 64, kernel_size=3, padding=1)
        # 네 번째 합성곱층 및 풀링층
        self.conv4 = nn.Conv2d(64, 32, kernel_size=3, padding=1)

        # Flatten 레이어
        self.flatten = nn.Flatten()

        # 완전 연결층
        self.fc1 = nn.Linear(32 * 14 * 14, 256)
        self.fc2 = nn.Linear(256, 2)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
```

	`` x = self.relu(self.conv2(x)) x = self.pool(x) x = self.relu(self.conv3(x)) x = self.pool(x) x = self.relu(self.conv4(x)) x = self.pool(x) x = self.flatten(x) x = self.relu(self.fc1(x)) x = self.fc2(x) return x # 모델 인스턴스 생성 model = CNNModel() ```
실습 가이드	이 실습에서는 PyTorch Sequential 방식을 사용하여 (배치, 3, 224, 224) 형태의 이미지 입력을 받는 CNN 모델을 설계하는 방법을 학습합니다. 이 모델은 더 큰 이미지(224x224 픽셀)를 처리할 수 있도록 설계되었으며, 네 개의 합성곱 층을 사용하여 이미지에서 점차적으로 더 추상적인 특징을 추출합니다. 각 합성곱 층 뒤에는 ReLU 활성화 함수와 Max Pooling을 적용하여 공간 정보를 줄이면서도 유용한 특징을 남깁니다. 마지막 완전 연결 층에서 추출된 특징을 기반으로 최종적으로 2개의 클래스를 분류합니다. 이 모델은 매우 큰 이미지(224x224)를 처리할 수 있도록 최적화되어 있으며, 이러한 이미지들은 일반적으로 고해상도 이미지를 나타냅니다. 이미지 분류 문제에서 이러한 모델은 큰 특징 맵을 효과적으로 다룰 수 있는 방법을 제공합니다.
실습 요약	(배치, 3, 224, 224) 형태의 입력을 받는 CNN 모델을 Sequential 방식으로 설계하여 고해상도 이미지 분류 문제를 해결하는 방법을 실습합니다. 더 깊은 네트워크 구조를 통해 고해상도 이미지를 처리하며, 각 층이 추출하는 특징과 네트워크의 학습 방식에 대해 이해합니다.

05-008 JAEN 패키지에서 CNN 모델 불러오기

실습번호	05-008	실습명	JAEN 패키지에서 CNN 모델 불러오기
실습 코드			

```
# JAEN 패키지에서 CNN 모델 가져오기
from JAEN.models import CNNModel

# CNN 모델 불러오기 (pretrained=True)
model = CNNModel(pretrained=True)

# 모델 정보 확인
summary(model, (64, 1, 28, 28))
```

실습 가이드	이 실습에서는 JAEN 패키지에서 제공하는 CNN 모델을 불러오고, 사전 학습된 (pretrained) 가중치를 사용하여 모델을 초기화하는 방법을 학습합니다. summary() 함수를 사용하여 모델의 구조와 파라미터 수를 확인하여 모델이 올바르게 로드되었는지 검증합니다.
실습 요약	JAEN 패키지에서 CNN 모델을 불러오고, 모델 정보를 요약하여 확인하는 과정을 실습합니다. 이는 사전 학습된 모델을 사용할 때 유용합니다.

05-009 기존 Conv Block 동결

실습번호	05-009	실습명	기존 Conv Block 동결
실습 코드	```# 기존 Conv Block 동결 for param in model.conv_layers.parameters(): param.requires_grad = False```		
실습 가이드	이 실습에서는 사전 학습된 CNN 모델의 일부 레이어(Conv Block)의 파라미터를 동결하여 학습 중에 업데이트되지 않도록 설정하는 방법을 학습합니다. 이는 transfer learning 기법의 일환으로, 특정 레이어의 가중치를 고정하여 새로운 데이터셋에 대해 학습할 때 유용합니다.		
실습 요약	CNN 모델의 기존 Conv Block을 동결하여 해당 레이어의 파라미터가 학습되지 않도록 설정하는 과정을 실습합니다. 이는 전이 학습 시 일반적인 기법입니다.		

05-010 새로운 Fully Connected Block 설정

실습번호	05-010	실습명	새로운 Fully Connected Block 설정
실습 코드			

```
# 새로운 Fully Connected Block 설정
model.fc_layers = nn.Sequential(
    nn.Linear(32 * 7 * 7, 64),  # 첫 번째 은닉층
    nn.ReLU(),
    nn.Dropout(p=0.5),           # 드롭아웃 추가
    nn.Linear(64, 10)           # 출력층 (활성화 함수 없음)
)

# 디바이스 설정 (GPU 또는 CPU)
model = model.to(device)
summary(model, input_size=(64, 1, 28, 28))
```

실습 가이드	이 실습에서는 CNN 모델의 Fully Connected Block을 새롭게 설정합니다. nn.Sequential()을 사용하여 새로운 은닉층과 드롭아웃 레이어를 추가하고, 출력층을 정의합니다. 모델을 디바이스(GPU 또는 CPU)로 이동한 후, summary() 함수를 사용하여 모델의 구조를 확인합니다.
실습 요약	새로운 Fully Connected Block을 설정하고, 모델의 구조와 파라미터 수를 요약하여 확인하는 과정을 실습합니다. 이는 모델 구조를 조정하고 최적화하는 데 중요한 단계입니다.

05-011 손실 함수 및 최적화 도구 정의와 학습 수행

실습번호	05-011	실습명	손실 함수 및 최적화 도구 정의와 학습 수행
실습 코드	 `# 손실 함수와 최적화 도구 정의` `criterion = nn.CrossEntropyLoss()` `optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=0.0001)` `train_losses, train_accuracies, test_losses, test_accuracies = train_and_evaluate(` `  model, train_loader, test_loader, criterion, optimizer, num_epochs=10, device=device` `)`		
실습 가이드	이 실습에서는 손실 함수와 최적화 도구를 정의합니다. nn.CrossEntropyLoss()는 다중 클래스 분류에 적합한 손실 함수이며, Adam 옵티마이저는 requires_grad가 True인 파라미터만 업데이트하도록 설정합니다. 이후, train_and_evaluate() 함수를 호출하여 모델을 학습하고 평가합니다.		
실습 요약	손실 함수와 최적화 도구를 설정한 후, 모델을 학습하고 평가하는 과정을 실습합니다. 이 과정은 신경망 모델의 성능을 최적화하는 데 필수적입니다.		

05-012 마지막 두 Conv 레이어만 학습하도록 설정

실습번호	05-012	실습명	마지막 두 Conv 레이어만 학습하도록 설정
실습 코드	```# 마지막 두 Conv 레이어만 학습하도록 설정 for name, p in model.conv_layers.named_parameters(): if name in ['5.weight', '5.bias', '7.weight', '7.bias']: p.requires_grad = True```		
실습 가이드	이 실습에서는 CNN 모델에서 마지막 두 컨볼루션 레이어의 파라미터만 학습하도록 설정합니다. named_parameters() 메서드를 사용하여 각 레이어의 이름과 파라미터를 반복하고, 특정 레이어에 대해서만 requires_grad를 True로 설정합니다. 이를 통해 모델의 특정 부분만 학습하여 과적합을 방지하고 성능을 최적화할 수 있습니다.		
실습 요약	CNN 모델의 마지막 두 Conv 레이어만 학습하도록 설정하는 과정을 실습합니다. 이는 전이 학습 시 효과적인 접근 방식입니다.		

05-013 손실 함수 및 최적화 도구 정의와 학습 수행

실습번호	05-013	실습명	손실 함수 및 최적화 도구 정의와 학습 수행
실습 코드	 `# 손실 함수와 최적화 도구 정의` `criterion = nn.CrossEntropyLoss()` `optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=0.0001)` `train_losses, train_accuracies, test_losses, test_accuracies = train_and_evaluate(` `  model, train_loader, test_loader, criterion, optimizer, num_epochs=10, device=device` `)`		
실습 가이드	이 실습에서는 손실 함수와 최적화 도구를 정의합니다. nn.CrossEntropyLoss()는 다중 클래스 분류에 적합한 손실 함수이며, Adam 옵티마이저는 requires_grad가 True인 파라미터만 업데이트하도록 설정합니다. 이후, train_and_evaluate() 함수를 호출하여 모델을 학습하고 평가합니다.		
실습 요약	손실 함수와 최적화 도구를 설정한 후, 모델을 학습하고 평가하는 과정을 실습합니다. 이 과정은 신경망 모델의 성능을 최적화하는 데 필수적입니다.		

06. 인공지능-RNN

06-001 Tokenizer 불러오기

실습번호	06-001	실습명	Tokenizer 불러오기
실습 코드	`from transformers import AutoTokenizer` `tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")`		
실습 가이드	이 실습에서는 Hugging Face의 Transformers 라이브러리를 사용하여 BERT 모델의 토크나이저를 불러오는 방법을 학습합니다. AutoTokenizer를 사용하여 'bert-base-cased' 모델에 맞는 토크나이저를 자동으로 가져오며, 이 토크나이저는 입력 텍스트를 BERT 모델에서 처리할 수 있는 형식으로 변환하는 데 사용됩니다.		
실습 요약	BERT 모델에 사용할 토크나이저를 불러오는 과정을 실습합니다. 이는 NLP 작업에서 텍스트를 준비하는 데 중요한 단계입니다.		

06-002 토큰화 실습

실습번호	06-002	실습명	토큰화 실습
실습 코드	`tokenizer("AI Essential")`		
실습 가이드	이 실습에서는 불러온 BERT 토크나이저를 사용하여 주어진 텍스트 'AI Essential'을 토크나이징하는 방법을 학습합니다. 토크나이저는 입력 문자열을 BERT 모델이 이해할 수 있는 형식으로 변환하며, 각 단어를 고유한 ID로 매핑합니다.		
실습 요약	BERT 토크나이저를 사용하여 텍스트를 토크나이즈하는 과정을 실습합니다. 이는 NLP 모델에서 입력 데이터를 준비하는 데 필수적인 과정입니다.		

06-003 토큰화된 텍스트 확인

실습번호	06-003	실습명	토큰화된 텍스트 확인
실습 코드	`tokens = tokenizer.tokenize("AI Essential")` `tokens`		
실습 가이드	이 실습에서는 BERT 토크나이저를 사용하여 텍스트 'AI Essential'을 토큰화한 결과를 확인합니다. tokenizer.tokenize() 메서드를 통해 입력 텍스트를 개별 토큰으로 분리하며, 각 토큰은 BERT 모델이 처리할 수 있는 형태로 변환됩니다.		
실습 요약	주어진 텍스트를 BERT 토크나이저를 사용하여 토큰화하고, 생성된 토큰을 확인하는 과정을 실습합니다. 이는 NLP 모델에서 데이터를 준비하는 중요한 단계입니다.		

06-004 토큰을 ID로 변환

실습번호	06-004	실습명	토큰을 ID로 변환
실습 코드	ids = tokenizer.convert_tokens_to_ids(tokens) ids		
실습 가이드	이 실습에서는 BERT 토크나이저를 사용하여 토큰화된 텍스트를 고유한 정수 ID로 변환하는 방법을 학습합니다. tokenizer.convert_tokens_to_ids() 메서드를 통해 각 토큰을 모델에서 사용하는 입력 형태로 변환합니다.		
실습 요약	토큰화된 텍스트를 BERT 모델의 입력으로 사용할 수 있도록 정수 ID로 변환하는 과정을 실습합니다. 이는 NLP 모델에 데이터를 입력하는 데 필수적인 과정입니다.		

06-005 토큰 ID를 텍스트로 디코딩

실습번호	06-005	실습명	토큰 ID를 텍스트로 디코딩
실습 코드	`tokenizer.decode(ids)`		
실습 가이드	이 실습에서는 BERT 토크나이저를 사용하여 토큰 ID를 다시 원래의 텍스트로 디코딩하는 방법을 학습합니다. tokenizer.decode() 메서드를 사용하여 ID를 입력하면, 모델이 이해할 수 있는 형태의 원본 문자열로 복원됩니다.		
실습 요약	토큰 ID를 사용하여 원래의 텍스트로 디코딩하는 과정을 실습합니다. 이는 NLP 작업에서 데이터를 확인하고 검증하는 데 유용합니다.		

06-006 임베딩 모듈 생성 및 사용

실습번호	06-006	실습명	임베딩 모듈 생성 및 사용
실습 코드			

```
import torch
import torch.nn as nn

# 임베딩 모듈 생성 (정수 인덱스 10개, 각 인덱스는 5차원의 벡터로 매핑)
embedding_layer = nn.Embedding(num_embeddings=10, embedding_dim=5)

# 임베딩에 사용할 예시 입력 데이터 (정수 인덱스)
# 여기서 [2, 5, 7]은 단어나 토큰에 해당한다고 가정
input_data = torch.tensor([2, 5, 7])

# 임베딩 레이어에 입력 데이터를 전달하여 벡터 변환
embedded_output = embedding_layer(input_data)

print("입력 데이터 (정수 인덱스):", input_data)
print("임베딩 결과 (벡터):\n", embedded_output)
```

실습 가이드	이 실습에서는 PyTorch의 nn.Embedding 모듈을 사용하여 정수 인덱스를 고차원 벡터로 변환하는 방법을 학습합니다. 임베딩 레이어를 생성하고, 입력 데이터로 주어진 정수 인덱스를 벡터로 변환합니다. 이 과정은 자연어 처리(NLP) 작업에서 단어 또는 토큰을 임베딩하는 데 사용됩니다.
실습 요약	임베딩 모듈을 생성하고, 정수 인덱스를 벡터로 변환하는 과정을 실습합니다. 이는 NLP 모델에서 단어 표현을 얻는 데 필수적인 단계입니다.

06-007 코퍼스 및 레이블 정의

실습번호	06-007	실습명	코퍼스 및 레이블 정의
실습 코드	corpus = ['very good nice quality', 'stop lying', 'ugly terrible', 'excellent work', 'adorable lovely', 'bad', 'great nice'] y = torch.FloatTensor([1, 0, 0, 1, 1, 0, 1]).reshape(-1, 1) # 0: 부정, 1: 긍정		
실습 가이드	이 실습에서는 간단한 문장 리스트(corpus)와 각 문장에 대한 레이블(y)을 정의합니다. y는 부정(0)과 긍정(1)으로 이루어져 있으며, 각 문장의 감정 평가를 나타냅니다. 이러한 데이터는 감정 분석 모델을 학습시키는 데 사용됩니다.		
실습 요약	감정 분석을 위한 문장 목록과 레이블을 정의하는 과정을 실습합니다. 이는 NLP 모델 학습을 위한 데이터 준비의 기초 단계입니다.		

06-008 GPT2 토크나이저로 문장 변환

실습번호	06-008	실습명	GPT2 토크나이저로 문장 변환
실습 코드	 from transformers import AutoTokenizer # GPT2 토크나이저 불러오기 tokenizer = AutoTokenizer.from_pretrained('gmteacher/simple-word-tokenizer', clean_up_tokenization_spaces=True) seqs = tokenizer(corpus)['input_ids'] seqs # 변환된 시퀀스		
실습 가이드	이 실습에서는 GPT2 모델의 토크나이저를 사용하여 문장 목록(corpus)을 토큰화합니다. AutoTokenizer를 통해 사전 학습된 토크나이저를 불러오고, clean_up_tokenization_spaces=True 옵션을 통해 불필요한 공백을 제거합니다. 변환된 시퀀스는 각 문장을 토큰 ID의 리스트로 표현합니다.		
실습 요약	 GPT2 토크나이저를 사용하여 문장 목록을 토큰화하고, 변환된 시퀀스를 확인하는 과정을 실습합니다. 이는 NLP 모델의 입력 데이터를 준비하는 데 필수적인 단계입니다.		

06-009 토큰 ID를 단어로 복원

실습번호	06-009	실습명	토큰 ID를 단어로 복원
실습 코드	`# 시퀀스를 단어로 복원` `[tokenizer.decode(seq) for seq in seqs]`		
실습 가이드	이 실습에서는 GPT2 토크나이저를 사용하여 토큰 ID를 원래의 단어로 복원합니다. tokenizer.decode() 메서드를 사용하여 각 시퀀스를 입력하면, 모델이 이해할 수 있는 형태의 원본 문자열로 변환됩니다.		
실습 요약	토큰 ID를 사용하여 원래의 단어로 복원하는 과정을 실습합니다. 이는 NLP 작업에서 데이터를 확인하고 검증하는 데 유용합니다.		

06-010 시퀀스를 텐서로 변환하고 패딩

실습번호	06-010	실습명	시퀀스를 텐서로 변환하고 패딩
실습 코드	 from torch.nn.utils.rnn import pad_sequence # 각 시퀀스를 텐서로 변환 seqs = [torch.tensor(seq) for seq in seqs] # 패딩 x = pad_sequence(seqs, batch_first=True) x		
실습 가이드	이 실습에서는 토큰 ID로 변환된 시퀀스들을 PyTorch 텐서로 변환한 후, pad_sequence() 함수를 사용하여 패딩을 적용합니다. 패딩을 통해 각 시퀀스의 길이를 동일하게 맞추어 배치 처리를 용이하게 합니다. batch_first=True로 설정하여 배치가 첫 번째 차원에 오도록 합니다.		
실습 요약	 각 시퀀스를 텐서로 변환하고, 패딩을 적용하여 배치 처리를 위한 텐서를 준비하는 과정을 실습합니다. 이는 시퀀스 데이터 처리에서 매우 중요한 단계입니다.		

06-011 텍스트 분류기 모델 정의

실습번호	06-011	실습명	텍스트 분류기 모델 정의
실습 코드			

```
class TextClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, seq_len, num_class):
        super().__init__()
        # 임베딩 계층 추가
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.flat = nn.Flatten()
        self.fc = nn.Linear(embed_dim*seq_len, num_class)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.embedding(x)
        out = self.flat(out)
        out = self.fc(out)
        out = self.sigmoid(out)
        return out
```

실습 가이드	이 실습에서는 텍스트 분류를 위한 신경망 모델을 정의합니다. TextClassifier 클래스는 입력 데이터를 임베딩 레이어를 통해 고차원 벡터로 변환하고, 평탄화(flatten)한 후 완전 연결 레이어를 사용하여 클래스를 예측합니다. 마지막으로, Sigmoid 활성화 함수를 사용하여 출력을 변환합니다.
실습 요약	텍스트 분류를 위한 신경망 모델을 정의하는 과정을 실습합니다. 이 모델은 NLP 작업에서 입력 데이터를 분류하는 데 사용됩니다.

06-012 텍스트 분류기 모델 인스턴스 생성 및 출력 테스트

실습번호	06-012	실습명	텍스트 분류기 모델 인스턴스 생성 및 출력 테스트
실습 코드	``` torch.manual_seed(0) # 모델 인스턴스 생성 model = TextClassifier(tokenizer.vocab_size, 2, x.shape[1], 1).to(device) # 출력 테스트 output = model(x[:1].to(device)) output.shape ```		
실습 가이드	이 실습에서는 TextClassifier 모델의 인스턴스를 생성하고, 입력 데이터에 대한 출력 결과의 형태를 확인합니다. torch.manual_seed(0)으로 랜덤 시드를 설정하여 결과의 재현성을 보장합니다. 모델의 출력이 올바르게 형성되는지를 테스트합니다.		
실습 요약	텍스트 분류기 모델을 인스턴스화하고, 입력 데이터를 사용하여 출력의 형태를 확인하는 과정을 실습합니다. 이는 모델이 기대한 대로 작동하는지 검증하는 데 중요합니다.		

06-013 장치 적용

실습번호	06-013	실습명	장치 적용
실습 코드	# 장치 적용 x = x.to(device) y = y.to(device)		
실습 가이드	이 실습에서는 입력 데이터 x와 타겟 데이터 y를 설정한 장치(GPU 또는 CPU)로 이동합니다. PyTorch에서는 모델과 데이터가 동일한 장치에 있어야 연산이 가능하므로, 이를 통해 학습 및 추론을 위한 준비를 합니다.		
실습 요약	입력 데이터와 타겟 데이터를 장치에 적용하는 과정을 실습합니다. 이는 PyTorch에서 효율적인 계산을 위한 필수적인 단계입니다.		

06-014 모델 학습 및 손실 계산

실습번호	06-014	실습명	모델 학습 및 손실 계산
실습 코드			

```
loss_fn = nn.BCELoss()  # 손실 함수
optimizer = optim.Adam(model.parameters())  # 최적화 도구(optimizer)

epochs = 20000  # 최대 에폭 지정
results = {'cost':[]}

model.train()  # 학습 모드 설정

for epoch in range(epochs):
    h = model(x)  # 예측 값 생성(추론)
    loss = loss_fn(h, y)  # 손실 계산

    optimizer.zero_grad()  # 미분 값 초기화
    loss.backward()  # 역전파(미분 계산)
    optimizer.step()  # 업데이트 진행

    results['cost'].append(loss.item())

    if epoch % 2000 == 0:
        print(f'epoch: {epoch:4d}, cost: {results["cost"][-1]:.10f}')
```

실습 가이드	이 실습에서는 정의한 텍스트 분류기 모델을 학습시키고 손실을 계산하는 과정을 구현합니다. BCELoss를 손실 함수로 사용하고, Adam 옵티마이저를 통해 모델의 파라미터를 업데이트합니다. 지정된 최대 에폭 수만큼 모델을 학습시키며, 주기적으로 손실 값을 출력합니다.
실습 요약	모델을 학습시키고 손실을 계산하여 업데이트하는 과정을 실습합니다. 학습 과정의 손실 변화를 모니터링하는 것은 모델의 성능을 이해하는 데 중요합니다.

06-015 예측 값을 클래스로 변환

실습번호	06-015	실습명	예측 값을 클래스로 변환
실습 코드	 `# 예측 값(시그모이드 값)을 클래스로 변환` `pred = (h.reshape(-1).detach().cpu() > 0.5).to(torch.float32)` `print(f'실제 값: {y.reshape(-1)}')` `print(f'예측 값: {pred}')`		
실습 가이드	이 실습에서는 모델의 예측 값을 시그모이드 결과를 기반으로 클래스로 변환하는 방법을 학습합니다. h.reshape(-1)로 예측 결과를 평탄화한 후, 0.5를 기준으로 이진 클래스로 변환합니다. 실제 값과 예측 값을 출력하여 모델의 성능을 평가합니다.		
실습 요약	모델의 예측 값을 클래스 형태로 변환하고, 실제 값과 비교하여 출력하는 과정을 실습합니다. 이는 모델의 예측 정확도를 확인하는 데 중요합니다.		

06-016 코퍼스 및 레이블 정의

실습번호	06-016	실습명	코퍼스 및 레이블 정의
실습 코드	corpus = ['very good nice quality', 'stop lying', 'ugly terrible', 'excellent work', 'adorable lovely', 'bad', 'great nice'] y = torch.FloatTensor([1, 0, 0, 1, 1, 0, 1]).reshape(-1, 1) # 0: 부정, 1: 긍정		
실습 가이드	이 실습에서는 간단한 문장 리스트(corpus)와 각 문장에 대한 레이블(y)을 정의합니다. y는 부정(0)과 긍정(1)으로 이루어져 있으며, 각 문장의 감정 평가를 나타냅니다. 이러한 데이터는 감정 분석 모델을 학습시키는 데 사용됩니다.		
실습 요약	감정 분석을 위한 문장 목록과 레이블을 정의하는 과정을 실습합니다. 이는 NLP 모델 학습을 위한 데이터 준비의 기초 단계입니다.		

06-017 GPT2 토크나이저로 문장 변환

실습번호	06-017	실습명	GPT2 토크나이저로 문장 변환
실습 코드	 from transformers import AutoTokenizer # GPT2 토크나이저 불러오기 tokenizer = AutoTokenizer.from_pretrained('gmteacher/simple-word-tokenizer', clean_up_tokenization_spaces=True) seqs = tokenizer(corpus)['input_ids'] seqs # 변환된 시퀀스		
실습 가이드	이 실습에서는 GPT2 모델의 토크나이저를 사용하여 문장 목록(corpus)을 토큰화합니다. AutoTokenizer를 통해 사전 학습된 토크나이저를 불러오고, clean_up_tokenization_spaces=True 옵션을 통해 불필요한 공백을 제거합니다. 변환된 시퀀스는 각 문장을 토큰 ID의 리스트로 표현합니다.		
실습 요약	 GPT2 토크나이저를 사용하여 문장 목록을 토큰화하고, 변환된 시퀀스를 확인하는 과정을 실습합니다. 이는 NLP 모델의 입력 데이터를 준비하는 데 필수적인 단계입니다.		

06-018 토큰 ID를 단어로 복원

실습번호	06-018	실습명	토큰 ID를 단어로 복원
실습 코드	`# 시퀀스를 단어로 복원` `[tokenizer.decode(seq) for seq in seqs]`		
실습 가이드	이 실습에서는 GPT2 토크나이저를 사용하여 토큰 ID를 원래의 단어로 복원합니다. tokenizer.decode() 메서드를 사용하여 각 시퀀스를 입력하면, 모델이 이해할 수 있는 형태의 원본 문자열로 변환됩니다.		
실습 요약	토큰 ID를 사용하여 원래의 단어로 복원하는 과정을 실습합니다. 이는 NLP 작업에서 데이터를 확인하고 검증하는 데 유용합니다.		

06-019 시퀀스를 텐서로 변환하고 패딩

실습번호	06-019	실습명	시퀀스를 텐서로 변환하고 패딩
실습 코드	 from torch.nn.utils.rnn import pad_sequence # 각 시퀀스를 텐서로 변환 seqs = [torch.tensor(seq) for seq in seqs] # 패딩 x = pad_sequence(seqs, batch_first=True) x		
실습 가이드	이 실습에서는 토큰 ID로 변환된 시퀀스들을 PyTorch 텐서로 변환한 후, pad_sequence() 함수를 사용하여 패딩을 적용합니다. 패딩을 통해 각 시퀀스의 길이를 동일하게 맞추어 배치 처리를 용이하게 합니다. batch_first=True로 설정하여 배치가 첫 번째 차원에 오도록 합니다.		
실습 요약	 각 시퀀스를 텐서로 변환하고, 패딩을 적용하여 배치 처리를 위한 텐서를 준비하는 과정을 실습합니다. 이는 시퀀스 데이터 처리에서 매우 중요한 단계입니다.		

06-020 텍스트 분류기 모델 정의

실습번호	06-020	실습명	텍스트 분류기 모델 정의

실습 코드

```
from torch import nn

class TextClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_size, num_class):
        super().__init__()
        # 임베딩 계층 추가
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_class)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.embedding(x)
        out, _ = self.lstm(out)
        out = self.fc(out[:, -1, :])
        out = self.sigmoid(out)
        return out
```

실습 가이드

이 실습에서는 텍스트 분류를 위한 신경망 모델을 정의합니다. TextClassifier 클래스는 입력 데이터를 임베딩 레이어를 통해 고차원 벡터로 변환한 후, LSTM(Long Short-Term Memory) 계층을 사용하여 순차적 특성을 학습합니다. LSTM의 마지막 출력(hidden state)을 이용해 완전 연결 레이어로 클래스를 예측하고, Sigmoid 활성화 함수를 통해 최종 출력을 얻습니다. 이 모델은 자연어 처리 작업에서 시퀀스 데이터를 기반으로 한 분류 작업에 사용됩니다.

실습 요약

이 실습에서는 LSTM 계층을 포함한 텍스트 분류 모델을 정의하는 과정을 다룹니다. 모델은 임베딩 계층과 LSTM 계층을 활용하여 입력 텍스트의 순차적 특성을 반영한 분류 작업을 수행합니다. LSTM의 최종 출력을 사용하여 여러 클래스로의 분류를 예측합니다.

06-021 텍스트 분류기 모델 인스턴스 생성 및 출력 테스트

실습번호	06-021	실습명	텍스트 분류기 모델 인스턴스 생성 및 출력 테스트
실습 코드	``` torch.manual_seed(0) hidden_size = 128 # 모델 인스턴스 생성 model = TextClassifier(tokenizer.vocab_size, 2, hidden_size, 1).to(device) # 출력 테스트 output = model(x[:1].to(device)) output.shape ```		
실습 가이드	이 실습에서는 TextClassifier 모델의 인스턴스를 생성하고, 입력 데이터에 대한 출력 결과의 형태를 확인합니다. 모델 생성 시, 임베딩 차원과 LSTM의 은닉 상태 크기(hidden_size)를 설정합니다. torch.manual_seed(0)을 사용해 랜덤 시드를 고정하여 재현성을 보장합니다. 모델이 입력 데이터를 기반으로 출력이 올바르게 형성되는지를 테스트합니다.		
실습 요약	텍스트 분류기 모델을 인스턴스화하고, 입력 데이터를 사용하여 출력의 형태를 확인하는 과정을 실습합니다. 이는 모델이 기대한 대로 작동하는지 검증하는 데 중요한 단계입니다.		

06-022 장치 적용

실습번호	06-022	실습명	장치 적용
실습 코드	`# 장치 적용` `x = x.to(device)` `y = y.to(device)`		
실습 가이드	이 실습에서는 입력 데이터 x와 타겟 데이터 y를 설정한 장치(GPU 또는 CPU)로 이동합니다. PyTorch에서는 모델과 데이터가 동일한 장치에 있어야 연산이 가능하므로, 이를 통해 학습 및 추론을 위한 준비를 합니다.		
실습 요약	입력 데이터와 타겟 데이터를 장치에 적용하는 과정을 실습합니다. 이는 PyTorch에서 효율적인 계산을 위한 필수적인 단계입니다.		

06-023 모델 학습 및 손실 계산

실습번호	06-023	실습명	모델 학습 및 손실 계산

실습 코드

```
loss_fn = nn.BCELoss()  # 손실 함수
optimizer = optim.Adam(model.parameters())  # 최적화 도구(optimizer)

epochs = 20000  # 최대 에폭 지정
results = {'cost':[]}

model.train()  # 학습 모드 설정

for epoch in range(epochs):
    h = model(x)  # 예측 값 생성(추론)
    loss = loss_fn(h, y)  # 손실 계산

    optimizer.zero_grad()  # 미분 값 초기화
    loss.backward()  # 역전파(미분 계산)
    optimizer.step()  # 업데이트 진행

    results['cost'].append(loss.item())

    if epoch % 2000 == 0:
        print(f'epoch: {epoch:4d}, cost: {results["cost"][-1]:.10f}')
```

실습 가이드

이 실습에서는 정의한 텍스트 분류기 모델을 학습시키고 손실을 계산하는 과정을 구현합니다. BCELoss를 손실 함수로 사용하고, Adam 옵티마이저를 통해 모델의 파라미터를 업데이트합니다. 지정된 최대 에폭 수만큼 모델을 학습시키며, 주기적으로 손실 값을 출력합니다.

실습 요약

모델을 학습시키고 손실을 계산하여 업데이트하는 과정을 실습합니다. 학습 과정의 손실 변화를 모니터링하는 것은 모델의 성능을 이해하는 데 중요합니다.

06-024 예측 값을 클래스로 변환

실습번호	06-024	실습명	예측 값을 클래스로 변환
실습 코드	```# 예측 값(시그모이드 값)을 클래스로 변환 pred = (h.reshape(-1).detach().cpu() > 0.5).to(torch.float32) print(f'실제 값: {y.reshape(-1)}') print(f'예측 값: {pred}')```		
실습 가이드	이 실습에서는 모델의 예측 값을 시그모이드 결과를 기반으로 클래스로 변환하는 방법을 학습합니다. h.reshape(-1)로 예측 결과를 평탄화한 후, 0.5를 기준으로 이진 클래스로 변환합니다. 실제 값과 예측 값을 출력하여 모델의 성능을 평가합니다.		
실습 요약	모델의 예측 값을 클래스 형태로 변환하고, 실제 값과 비교하여 출력하는 과정을 실습합니다. 이는 모델의 예측 정확도를 확인하는 데 중요합니다.		

연습문제-06-001 10개 문서 유형 분류를 위한 모델 정의 (Module 방식)

실습번호	연습문제-06-001	실습명	10개 문서 유형 분류를 위한 모델 정의 (Module 방식)
실습 코드			
실습 가이드	이 실습에서는 문서 유형 10개를 분류하기 위한 신경망 모델을 Module 방식으로 정의합니다. 이 모델은 입력된 문서 데이터를 임베딩 레이어와 LSTM 계층을 거쳐 처리한 후, 10개의 출력값을 가지는 완전 연결층을 통해 각 문서가 어느 유형에 속하는지 예측합니다. 활성화 함수는 적용되지 않으며, 출력값을 통해 각 클래스에 대한 로짓 값을 반환합니다.		
실습 요약	이 실습에서는 PyTorch의 Module 방식을 사용해 문서 유형 10개를 분류하는 모델을 정의합니다. 이 모델은 LSTM을 통해 문서의 순차적 특성을 학습하고, 완전 연결층을 사용하여 10개의 범주로 문서를 분류합니다.		

```
from torch import nn

class DocumentClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_size, num_class=10):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_class)

    def forward(self, x):
        out = self.embedding(x)
        out, _ = self.lstm(out)
        out = self.fc(out[:, -1, :])  # LSTM의 마지막 출력 사용
        return out  # 활성화 함수 없이 10개의 출력값 반환
```

06-025 텍스트 분류 파이프라인 사용

실습번호	06-025	실습명	텍스트 분류 파이프라인 사용
실습 코드	`from transformers import pipeline` `classifier = pipeline('text-classification')` `classifier("I've been waiting for a HuggingFace course my whole life.")`		
실습 가이드	이 실습에서는 Hugging Face의 Transformers 라이브러리를 사용하여 텍스트 분류 파이프라인을 생성합니다. pipeline() 함수를 통해 'text-classification' 작업을 설정하고, 주어진 문장에 대해 모델의 예측 결과를 확인합니다.		
실습 요약	텍스트 분류 파이프라인을 사용하여 입력 문장에 대한 분류 결과를 얻는 과정을 실습합니다. 이는 NLP 작업에서 텍스트의 감정이나 주제를 분류하는 데 유용합니다.		

06-026 DistilBERT로 텍스트 분류

실습번호	06-026	실습명	DistilBERT로 텍스트 분류

실습 코드

```
import torch
from transformers import DistilBertTokenizer,
DistilBertForSequenceClassification

model_name = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = DistilBertTokenizer.from_pretrained(model_name)
model = DistilBertForSequenceClassification.from_pretrained(model_name)

text = "I've been waiting for a HuggingFace course my whole life."
inputs = tokenizer(text, return_tensors="pt")
with torch.no_grad():
    logits = model(**inputs).logits

predicted_class_id = logits.argmax().item()
label = model.config.id2label[predicted_class_id]
score = torch.softmax(logits, dim=1).squeeze()[predicted_class_id].item()
print(f'label:{label}, score:{score}')
```

실습 가이드

이 실습에서는 Hugging Face의 DistilBERT 모델을 사용하여 텍스트를 분류하는 방법을 보여줍니다. 사전 학습된 DistilBERT 모델을 로드하고, 주어진 문장을 토크나이즈하여 입력 텐서를 생성합니다. 모델에 입력을 전달하여 예측 결과를 얻고, 확률을 계산하여 최종 레이블과 점수를 출력합니다.

실습 요약

DistilBERT 모델을 사용하여 입력 텍스트의 감정을 분류하고, 예측 결과와 확률을 출력하는 과정을 실습합니다. 이는 NLP에서 텍스트 분류 작업의 기본적인 방법론입니다.

06-027 파이프라인 모델 지정 및 예측

실습번호	06-027	실습명	파이프라인 모델 지정 및 예측
실습 코드	# 파이프라인 모델 지정 classifier = pipeline('text-classification', model=model, tokenizer=tokenizer) classifier("I've been waiting for a HuggingFace course my whole life.")		
실습 가이드	이 실습에서는 Hugging Face의 pipeline을 사용하여 이전에 로드한 DistilBERT 모델과 토크나이저를 지정합니다. 지정된 파이프라인을 통해 주어진 문장에 대한 텍스트 분류를 수행하고 결과를 출력합니다.		
실습 요약	파이프라인을 사용하여 DistilBERT 모델로 텍스트 분류를 수행하고 결과를 출력하는 과정을 실습합니다. 이는 모델을 간편하게 사용하여 예측을 수행하는 방법을 보여줍니다.		

06-028 여러 문장 분류

실습번호	06-028	실습명	여러 문장 분류
실습 코드	 # 여러 문장 분류 classifier(["I've been waiting for a HuggingFace course my whole life.", "I hate this so much!"])		
실습 가이드	이 실습에서는 Hugging Face의 파이프라인을 사용하여 여러 개의 문장에 대해 텍스트 분류를 수행합니다. classifier() 함수를 호출하여 입력된 문장 목록을 전달하면, 각 문장에 대한 분류 결과를 얻을 수 있습니다.		
실습 요약	여러 문장에 대해 텍스트 분류를 수행하고 결과를 출력하는 과정을 실습합니다. 이는 NLP 모델이 동시에 여러 입력을 처리할 수 있는 방법을 보여줍니다.		

06-029 한국어 감정 분류

실습번호	06-029	실습명	한국어 감정 분류
실습 코드	 `# 한국어 감정 분류` `classifier_ko = pipeline('text-classification',` `model='matthewburke/korean_sentiment')` `classifier_ko(` `["그녀가 먼저 말을 걸어와서 나는 무척 기뻤다.",` `"듣기 싫어 죽겠네, 짜증나니까 그만 좀 닥쳐줄래"])`		
실습 가이드	이 실습에서는 Hugging Face의 파이프라인을 사용하여 한국어 감정 분류 모델을 로드하고, 여러 문장에 대한 감정을 분류합니다. 입력된 문장에 대해 긍정적 또는 부정적인 감정을 예측하여 결과를 출력합니다.		
실습 요약	한국어 감정 분류를 수행하고 결과를 출력하는 과정을 실습합니다. 이는 NLP 모델이 한국어 텍스트의 감정을 분석하는 데 유용한 방법입니다.		

06-030 제로 샷 분류

<table>
<tr><th>실습번호</th><th>06-030</th><th>실습명</th><th>제로 샷 분류</th></tr>
<tr><td>실습
코드</td><td colspan="3"><pre><code># zero-shot classification
classifier = pipeline("zero-shot-classification")
classifier(
    "This is a course about the Transformers library",
    candidate_labels=["education", "politics", "business"],
)</code></pre></td></tr>
<tr><td>실습
가이드</td><td colspan="3">이 실습에서는 Hugging Face의 제로 샷 분류 파이프라인을 사용하여 주어진 문장이 어떤 클래스에 속하는지를 예측합니다. candidate_labels로 제공된 여러 후보 레이블 중에서 가장 적합한 레이블을 선택하여 결과를 출력합니다.</td></tr>
<tr><td>실습
요약</td><td colspan="3">제로 샷 분류를 통해 주어진 문장의 분류를 수행하는 과정을 실습합니다. 이는 사전 학습된 모델을 사용하여 새로운 태스크에 대해 적응할 수 있는 방법을 보여줍니다.</td></tr>
</table>

06-031 한국어 제로 샷 분류

실습번호	06-031	실습명	한국어 제로 샷 분류
실습 코드	 `# 한국어 zero-shot classification` `model_name = 'MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7'` `classifier_ko = pipeline("zero-shot-classification", model_name)` `classifier_ko(` `  "이 강의는 트랜스포머 라이브러리에 대한 것입니다.",` `  candidate_labels=["교육", "정치", "사업"],` `)`		
실습 가이드	이 실습에서는 한국어 문장에 대해 제로 샷 분류를 수행합니다. mDeBERTa 모델을 사용하여 주어진 문장이 어떤 클래스에 속하는지를 예측합니다. 후보 레이블로 제공된 여러 항목 중에서 가장 적합한 레이블을 선택하여 결과를 출력합니다.		
실습 요약	한국어 문장에 대한 제로 샷 분류를 수행하고 결과를 출력하는 과정을 실습합니다. 이는 다국어 모델을 사용하여 다양한 태스크에 적응할 수 있는 방법을 보여줍니다.		

06-032 한국어 제로 샷 분류 예시

실습번호	06-032	실습명	한국어 제로 샷 분류 예시
실습 코드	classifier_ko("이 소총의 유효사거리는 5km를 넘습니다.", candidate_labels=["예술", "정치", "군사"],)		
실습 가이드	이 실습에서는 한국어 문장에 대해 제로 샷 분류를 수행하여 주어진 문장이 어떤 클래스에 속하는지를 예측합니다. 후보 레이블로 제공된 '예술', '정치', '군사' 중에서 가장 적합한 레이블을 선택하여 결과를 출력합니다.		
실습 요약	주어진 한국어 문장에 대한 제로 샷 분류를 수행하고, 결과를 출력하는 과정을 실습합니다. 이는 NLP 모델이 새로운 태스크에 대해 적응할 수 있는 방법을 보여줍니다.		

06-033 한국어 제로 샷 분류 예시

실습번호	06-033	실습명	한국어 제로 샷 분류 예시
실습 코드	classifier_ko("선과 색의 조화가 탁월한 그 작품은 과거의 어느 작가의 그것과 닮아 있다.", candidate_labels=["예술", "정치", "군사"],)		
실습 가이드	이 실습에서는 주어진 한국어 문장에 대해 제로 샷 분류를 수행합니다. 문장은 예술 작품에 대한 설명이며, 후보 레이블로 제공된 '예술', '정치', '군사' 중에서 가장 적합한 레이블을 선택하여 결과를 출력합니다.		
실습 요약	주어진 문장에 대한 제로 샷 분류를 수행하고, 출력 결과를 확인하는 과정을 실습합니다. 이는 다양한 주제에 대해 모델의 분류 능력을 평가하는 데 유용합니다.		

06-034 질문-답변 파이프라인 사용

<table>
<tr><th>실습번호</th><th>06-034</th><th>실습명</th><th>질문-답변 파이프라인 사용</th></tr>
<tr><td>실습
코드</td><td colspan="3"><pre>
Q & A
question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in Brooklyn",
)
</pre></td></tr>
<tr><td>실습
가이드</td><td colspan="3">이 실습에서는 Hugging Face의 질문-답변 파이프라인을 사용하여 주어진 질문에 대한 답변을 찾습니다. 질문과 함께 제공된 문맥(context)을 기반으로 모델이 질문에 대한 적절한 답변을 생성합니다.</td></tr>
<tr><td>실습
요약</td><td colspan="3">질문-답변 파이프라인을 사용하여 주어진 질문에 대한 답변을 찾는 과정을 실습합니다. 이는 NLP 모델이 문맥에서 정보를 추출하는 데 유용한 방법입니다.</td></tr>
</table>

06-035 한국어 질문-답변 파이프라인 사용

실습번호	06-035	실습명	한국어 질문-답변 파이프라인 사용
실습 코드	# 한국어 Q & A question_answerer = pipeline("question-answering", model='timpal0l/mdeberta-v3-base-squad2') question_answerer(question="홍길동은 어디에 있지?", context="홍길동씨는 잠실에 있는 숨겨진 국가정보원 빌딩에서 청소를 하며 살아 가고있다.",)		
실습 가이드	이 실습에서는 한국어 질문-답변 파이프라인을 사용하여 주어진 질문에 대한 답변을 찾습니다. 특정 모델을 지정하여 문맥(context)에서 질문에 대한 적절한 답변을 생성합니다.		
실습 요약	한국어 질문-답변 파이프라인을 사용하여 주어진 질문에 대한 답변을 찾는 과정을 실습합니다. 이는 NLP 모델이 한국어 문맥에서 정보를 추출하는 데 유용한 방법입니다.		

06-036 한국어 질문-답변 추가 예시

실습번호	06-036	실습명	한국어 질문-답변 추가 예시
실습 코드	question_answerer(question="홍길동은 무얼하고 있지?", context="홍길동씨는 잠실에 있는 숨겨진 국가정보원 빌딩에서 청소를 하며 살아 가고있다.",)		
실습 가이드	이 실습에서는 한국어 질문-답변 파이프라인을 사용하여 새로운 질문에 대한 답변을 찾습니다. 주어진 문맥(context)을 바탕으로 질문에 대한 적절한 답변을 생성합니다.		
실습 요약	한국어 질문-답변 파이프라인을 사용하여 특정 질문에 대한 답변을 찾는 과정을 실습합니다. 이는 모델이 한국어 문맥에서 정보를 추출하는 능력을 평가하는 데 유용합니다.		

06-037 GPT 텍스트 생성

실습번호	06-037	실습명	GPT 텍스트 생성
실습 코드	 # GPT 텍스트 생성 from transformers import pipeline generator = pipeline("text-generation", model='gpt2') generator("In this course, we will teach you how to")		
실습 가이드	이 실습에서는 Hugging Face의 텍스트 생성 파이프라인을 사용하여 GPT-2 모델을 기반으로 텍스트를 생성합니다. 주어진 입력 문장에 이어서 모델이 자동으로 텍스트를 생성하도록 합니다.		
실습 요약	GPT-2 모델을 사용하여 입력 문장에 이어지는 텍스트를 생성하는 과정을 실습합니다. 이는 자연어 생성 모델의 활용 방법을 보여줍니다.		

06-038 KoGPT 텍스트 생성

실습번호	06-038	실습명	KoGPT 텍스트 생성
실습 코드	`# KoGPT 텍스트 생성` `generator = pipeline("text-generation", model='skt/kogpt2-base-v2')` `output = generator("감기에 걸리지 않으려면")` `print(output[0]['generated_text'])`		
실습 가이드	이 실습에서는 KoGPT 모델을 사용하여 주어진 한국어 문장에 이어지는 텍스트를 생성합니다. '감기에 걸리지 않으려면'이라는 입력 문장에 대해 모델이 생성한 텍스트를 출력하여 확인합니다.		
실습 요약	KoGPT 모델을 사용하여 입력 문장에 따른 텍스트를 생성하고, 결과를 출력하는 과정을 실습합니다. 이는 한국어 자연어 생성 모델의 활용을 보여줍니다.		

07. Streamlit

07-001 PYNGROK API 키 설정

실습번호	07-001	실습명	PYNGROK API 키 설정
실습 코드	! %%capture !pip install pyngrok streamlit -qU PYNGROK_API_KEY = ''		
실습 가이드	이 실습에서는 PYNGROK_API_KEY 변수를 설정하여 ngrok API 키를 저장합니다. ngrok은 로컬 서버를 인터넷에서 접근할 수 있도록 터널링을 제공하는 서비스입니다. API 키는 ngrok 서비스를 사용하기 위해 필요합니다.		
실습 요약	PYNGROK_API_KEY를 설정하는 과정을 실습합니다. 이는 ngrok 서비스를 사용하여 로컬 환경을 외부에서 접근할 수 있도록 하는 데 필요합니다.		

07-002 Streamlit 앱 생성

실습번호	07-002	실습명	Streamlit 앱 생성
실습 코드	%%writefile app1.py # Streamlit 라이브러리를 st라는 이름으로 불러옵니다. import streamlit as st # 앱의 제목을 설정 st.title('Hello, Streamlit!') # 웹 페이지 상단에 'Hello, Streamlit!'이라는 제목이 표시됩니다. # 텍스트를 화면에 출력 st.write('Welcome to your first Streamlit app.') # 'Welcome to your first Streamlit app.'라는 문구가 화면에 표시됩니다.		
실습 가이드	이 실습에서는 Streamlit을 사용하여 간단한 웹 애플리케이션을 생성하는 코드를 작성합니다. 파일 이름은 app1.py로 설정하며, 앱의 제목과 환영 메시지를 화면에 표시합니다.		
실습 요약	Streamlit을 사용하여 간단한 웹 애플리케이션을 생성하는 과정을 실습합니다. 이는 데이터 시각화 및 웹 앱 개발의 기초를 다지는 데 유용합니다.		

07-003 ngrok을 통한 Streamlit 앱 실행

실습번호	07-003	실습명	ngrok을 통한 Streamlit 앱 실행
실습 코드			

```
from pyngrok import ngrok

# ngrok 서비스 인증
ngrok.set_auth_token(PYNGROK_API_KEY)

# app1.py 백그라운드 프로세스 실행
!nohup streamlit run app1.py --server.port 5011 &

# ngrok 터널링 실행
ngrok_tunnel = ngrok.connect(addr='5011', proto='http', bind_tls=True)

# ngrok 터널링 결과
print(' * Tunnel URL:', ngrok_tunnel.public_url)
```

실습 가이드	이 실습에서는 ngrok을 사용하여 Streamlit 앱을 외부에서 접근할 수 있도록 설정합니다. ngrok.set_auth_token()을 통해 인증 토큰을 설정한 후, app1.py를 백그라운드에서 실행하고, ngrok을 통해 터널링을 설정합니다. 최종적으로 터널 URL을 출력하여 외부에서 접근할 수 있는 링크를 제공합니다.
실습 요약	ngrok을 통해 Streamlit 앱을 실행하고, 외부에서 접근할 수 있는 URL을 생성하는 과정을 실습합니다. 이는 로컬 서버를 외부에서 테스트할 수 있는 유용한 방법입니다.

07-004 Streamlit 및 ngrok 서비스 종료

실습번호	07-004	실습명	Streamlit 및 ngrok 서비스 종료
실습 코드	# Streamlit 서비스 종료 !pkill -f streamlit # ngrok 터널링 종료 ngrok.disconnect(ngrok_tunnel.public_url)		
실습 가이드	이 실습에서는 실행 중인 Streamlit 서비스와 ngrok 터널을 종료합니다. !pkill -f streamlit 명령을 통해 모든 Streamlit 프로세스를 종료하고, ngrok.disconnect()를 사용하여 활성화된 ngrok 터널을 끊습니다.		
실습 요약	실행 중인 Streamlit 서비스와 ngrok 터널을 종료하는 과정을 실습합니다. 이는 자원을 정리하고 서비스를 안전하게 종료하는 데 필요한 과정입니다.		

07-005 Streamlit 챗봇 앱 UI 생성

실습번호	07-005	실습명	Streamlit 챗봇 앱 UI 생성
실습 코드	%%writefile app2.py import streamlit as st # 사이드 바 생성 st.sidebar.title('API Key 설정') # 사이드 바에 API Key 입력을 위한 텍스트 인풋 생성 api_key_input = st.sidebar.text_input('OpenAI API Key', type='password') if api_key_input: st.title('가전제품 고객지원 응답봇') user_question = st.chat_input('고장 사항을 입력하세요:') if user_question: with st.chat_message('user'): st.write(user_question) with st.chat_message('assistant'): st.write(f'안녕하세요. 저는 Gauss입니다. \n {user_question}을 입력하셨습니다.') else: st.sidebar.warning('API Key를 입력하세요.')		
실습 가이드	이 실습에서는 Streamlit 앱에 사이드바를 추가하고 API 키를 입력할 수 있는 텍스트 인풋을 생성합니다. 사용자가 API 키를 입력하면, 응답봇이 사용자 질문을 받아 처리하는 기본적인 인터페이스를 구현합니다.		
실습 요약	Streamlit 앱에 사이드바를 추가하여 API 키를 입력받고, 사용자 질문에 대한 응답을 표시하는 과정을 실습합니다. 이는 사용자와의 상호작용을 통해 앱 기능을 확장하는 데 유용합니다.		

07-006 app2.py 실행 및 ngrok 터널링

실습번호	07-006	실습명	app2.py 실행 및 ngrok 터널링
실습 코드			

```
from pyngrok import ngrok

# ngrok 서비스 인증
ngrok.set_auth_token(PYNGROK_API_KEY)

# app1.py 백그라운드 프로세스 실행
!nohup streamlit run app2.py --server.port 5011 &

# ngrok 터널링 실행
ngrok_tunnel = ngrok.connect(addr='5011', proto='http', bind_tls=True)

# ngrok 터널링 결과
print(' * Tunnel URL:', ngrok_tunnel.public_url)
```

실습 가이드	이 실습에서는 ngrok을 사용하여 Streamlit 앱(app2.py)을 외부에서 접근할 수 있도록 설정합니다. ngrok.set_auth_token()을 통해 인증 토큰을 설정한 후, app2.py를 백그라운드에서 실행하고, ngrok을 통해 터널링을 설정합니다. 최종적으로 터널 URL을 출력하여 외부에서 접근할 수 있는 링크를 제공합니다.
실습 요약	ngrok을 통해 Streamlit 앱을 실행하고, 외부에서 접근할 수 있는 URL을 생성하는 과정을 실습합니다. 이는 로컬 서버를 외부에서 테스트할 수 있는 유용한 방법입니다.

07-007 Streamlit 및 ngrok 서비스 종료

실습번호	07-007	실습명	Streamlit 및 ngrok 서비스 종료
실습 코드	# Streamlit 서비스 종료 !pkill -f streamlit # ngrok 터널링 종료 ngrok.disconnect(ngrok_tunnel.public_url)		
실습 가이드	이 실습에서는 실행 중인 Streamlit 서비스와 ngrok 터널을 종료합니다. !pkill -f streamlit 명령을 통해 모든 Streamlit 프로세스를 종료하고, ngrok.disconnect()를 사용하여 활성화된 ngrok 터널을 끊습니다.		
실습 요약	실행 중인 Streamlit 서비스와 ngrok 터널을 종료하는 과정을 실습합니다. 이는 자원을 정리하고 서비스를 안전하게 종료하는 데 필요한 과정입니다.		

08. LangChain

08-001 OpenAI API 키 설정

실습번호	08-001	실습명	OpenAI API 키 설정
실습 코드	`import os` `os.environ["OPENAI_API_KEY"]= ''`		
실습 가이드	이 실습에서는 OpenAI API 키를 환경 변수로 설정합니다. os.environ을 사용하여 OPENAI_API_KEY에 값을 할당함으로써, 이후 API를 사용할 때 이 키를 참조할 수 있습니다.		
실습 요약	OpenAI API 키를 환경 변수로 설정하는 과정을 실습합니다. 이는 OpenAI API를 안전하게 사용할 수 있도록 하는 데 필요합니다.		

08-002 GPT-4o-mini 모델 객체 생성

실습번호	08-002	실습명	GPT-4o-mini 모델 객체 생성
실습 코드	`# gpt-4o-mini 모델 객체 생성` `from langchain_openai import ChatOpenAI` `llm = ChatOpenAI(model="gpt-4o-mini")` `llm`		
실습 가이드	이 실습에서는 langchain_openai 라이브러리를 사용하여 gpt-4o-mini 모델 객체를 생성합니다. ChatOpenAI 클래스를 통해 모델을 초기화하고, 이후에 이 모델을 사용하여 텍스트 생성 또는 질문 응답 등의 작업을 수행할 수 있습니다.		
실습 요약	GPT-4o-mini 모델 객체를 생성하는 과정을 실습합니다. 이는 LangChain 라이브러리를 사용하여 OpenAI의 모델을 활용하는 기본적인 방법을 보여줍니다.		

08-003 GPT-4o-mini 모델에 메세지 전달

실습번호	08-003	실습명	GPT-4o-mini 모델에 메세지 전달
실습 코드	```# gpt-4o-mini 모델 메세지 전달 from langchain_core.messages import HumanMessage, SystemMessage messages = [SystemMessage(content="Translate the following from Korean into Python"), HumanMessage(content="주어진 리스트를 병합 정렬(Merge Sort) 알고리즘을 사용하여 오름차순으로 정렬하는 파이썬 코드를 작성하세요."),] # messages 전달 결과 확인 result = llm.invoke(messages) result```		
실습 가이드	이 실습에서는 gpt-4o-mini 모델에 대해 시스템 메시지와 사용자 메시지를 전달하여 응답을 받습니다. SystemMessage를 통해 작업의 목표를 설정하고, HumanMessage를 통해 모델에 전달할 내용을 입력합니다. 최종적으로 모델의 응답 결과를 확인합니다.		
실습 요약	GPT-4o-mini 모델에 메세지를 전달하고, 그 결과를 확인하는 과정을 실습합니다. 이는 OpenAI 모델을 활용하여 특정 작업을 수행하도록 하는 기본적인 방법을 보여줍니다.		

08-004 JAEN에서 파일 다운로드

실습번호	08-004	실습명	JAEN에서 파일 다운로드
실습 코드	`from JAEN import download_file` `download_file('샘플사진')`		
실습 가이드	이 실습에서는 JAEN 라이브러리를 사용하여 특정 파일을 다운로드합니다. download_file 함수에 '샘플사진'이라는 파일명을 전달하여 해당 파일을 다운로드하고, 이후에 사용할 수 있도록 준비합니다.		
실습 요약	JAEN 라이브러리를 사용하여 파일을 다운로드하는 과정을 실습합니다. 이는 데이터나 리소스를 가져오는 데 유용한 방법입니다.		

08-005 이미지 표시

실습번호	08-005	실습명	이미지 표시
실습 코드	`from IPython.display import Image` `Image('sample.jpg')`		
실습 가이드	이 실습에서는 IPython.display 라이브러리의 Image 클래스를 사용하여 이미지를 표시합니다. 'sample.jpg' 파일을 입력으로 전달하여 해당 이미지를 Jupyter Notebook 또는 IPython 환경에서 시각적으로 확인할 수 있습니다.		
실습 요약	이미지를 표시하는 과정을 실습합니다. 이는 데이터 시각화 및 결과 확인에 유용한 방법입니다.		

08-006 이미지 인코딩

실습번호	08-006	실습명	이미지 인코딩
실습 코드	 import base64 def encode_image(image_path): with open(image_path, "rb") as image_file: return base64.b64encode(image_file.read()).decode("utf-8") encoded_image = encode_image('sample.jpg')		
실습 가이드	이 실습에서는 이미지를 Base64 형식으로 인코딩하는 함수를 정의합니다. encode_image 함수는 주어진 이미지 경로를 사용하여 이미지를 바이너리 형식으로 읽고, 이를 Base64로 인코딩하여 문자열로 반환합니다.		
실습 요약	이미지를 Base64 형식으로 인코딩하는 과정을 실습합니다. 이는 웹에서 이미지를 전송하거나 저장하는 데 유용한 방법입니다.		

08-007 이미지 분석 메시지 구성

실습번호	08-007	실습명	이미지 분석 메시지 구성
실습 코드	messages = [{"role": "system", "content": "당신은 비주얼 분석가입니다. 주어진 이미지를 보고 이미지 속에 포함된 요소들을 면밀히 관찰하고, 각 요소가 어떤 의미를 가지고 있는지 분석하세요. 이미지의 색상, 구성, 인물의 표정과 자세, 배경 요소, 조명 등 모든 시각적 요소를 분석하여 이미지가 전달하고자 하는 메시지나 감정을 설명하세요. 또한, 이 이미지가 제작된 문화적, 사회적, 역사적 배경을 고려하여 추가적인 해석을 한국어로 제공하세요."}, {"role": "user", "content": [{ "type": "image_url", "image_url": {"url": f"{encoded_image}"}}]}]		
실습 가이드	이 실습에서는 이미지 분석을 위한 메시지를 구성합니다. 시스템 메시지를 통해 비주얼 분석가의 역할을 정의하고, 사용자 메시지로 이미지의 URL을 포함합니다. base64 인코딩된 이미지를 사용하여 주어진 내용을 전달할 수 있습니다.		
실습 요약	이미지 분석을 위한 메시지를 구성하는 과정을 실습합니다. 이는 NLP 모델이 비주얼 콘텐츠를 이해하고 해석하는 데 필요한 방법을 보여줍니다.		

08-008 이미지 분석 요청

실습번호	08-008	실습명	이미지 분석 요청
실습 코드	`llm.invoke(messages)`		
실습 가이드	이 실습에서는 이전에 구성한 메시지를 사용하여 GPT 모델에 이미지 분석 요청을 수행합니다. llm.invoke() 함수를 호출하여 주어진 메시지를 전달하면, 모델이 이미지에 대한 분석 결과를 생성하여 반환합니다.		
실습 요약	구성된 메시지를 통해 이미지 분석 요청을 수행하고, 그 결과를 확인하는 과정을 실습합니다. 이는 NLP 모델이 비주얼 콘텐츠에 대한 통찰을 제공하는 방법을 보여줍니다.		

08-009 ChatOpenAI 인스턴스 생성

실습번호	08-009	실습명	ChatOpenAI 인스턴스 생성
실습 코드	 from langchain_openai import ChatOpenAI # ChatOpenAI 인스턴스를 생성하면서 모델과 파라미터를 설정. model = ChatOpenAI(model="gpt-4o-mini", # 사용할 모델을 "gpt-4o-mini"로 지정. max_tokens=2048, # 모델이 반환할 최대 토큰 수를 2048로 설정 temperature=0.1, # 모델의 응답 창의성을 조절하는 파라미터. 0.1로 설정하여 더 예측 가능한 응답을 유도.) 		
실습 가이드	이 실습에서는 ChatOpenAI 클래스를 사용하여 모델 인스턴스를 생성합니다. 사용할 모델, 최대 토큰 수, 응답의 창의성을 조절하는 온도를 설정하여 인스턴스를 초기화합니다. 이러한 설정은 모델의 응답 특성에 큰 영향을 미칩니다.		
실습 요약	ChatOpenAI 인스턴스를 생성하고 설정하는 과정을 실습합니다. 이는 모델을 활용하여 특정 작업을 수행하는 데 필요한 초기 설정을 보여줍니다.		

08-010 Output Parser 생성

실습번호	08-010	실습명	Output Parser 생성
실습 코드	`from langchain_core.output_parsers import StrOutputParser` `parser = StrOutputParser()` `parser`		
실습 가이드	이 실습에서는 StrOutputParser 클래스를 사용하여 문자열 출력을 처리하는 파서를 생성합니다. 이 파서는 모델의 출력을 문자열 형식으로 변환하고, 이후 작업에 사용할 수 있도록 준비합니다.		
실습 요약	Output Parser를 생성하여 모델의 출력을 처리하는 과정을 실습합니다. 이는 NLP 모델의 결과를 적절한 형식으로 변환하는 데 유용합니다.		

08-011 ChatPromptTemplate 생성 및 사용

실습번호	08-011	실습명	ChatPromptTemplate 생성 및 사용
실습 코드	 `from langchain_core.prompts import ChatPromptTemplate` `system_template = "Translate the following into {language}:"` `prompt_template = ChatPromptTemplate.from_messages(` `[("system", system_template),` `("user", "{text}")]` `)` `result = prompt_template.invoke({"language": "spanish", "text": "hello"})` `result`		
실습 가이드	이 실습에서는 ChatPromptTemplate을 사용하여 사용자와 시스템 간의 대화 형식의 프롬프트를 생성합니다. 시스템 메시지를 통해 번역할 언어를 설정하고, 사용자 메시지를 통해 번역할 텍스트를 입력합니다. 최종적으로 invoke() 메서드를 사용하여 결과를 얻습니다.		
실습 요약	 ChatPromptTemplate을 생성하고 이를 사용하여 텍스트 번역 요청을 수행하는 과정을 실습합니다. 이는 NLP 모델과의 상호작용을 관리하는 데 유용한 방법입니다.		

08-012 체인 구성 및 실행

실습번호	08-012	실습명	체인 구성 및 실행
실습 코드	`chain = prompt_template ¦ model ¦ parser` `chain.invoke({"language": "spanish", "text": "hello."})`		
실습 가이드	이 실습에서는 ChatPromptTemplate, 모델, 및 출력 파서를 연결하여 체인을 구성합니다. 이 체인은 입력 텍스트를 번역 요청으로 변환하고, 모델을 통해 번역을 수행한 후 결과를 문자열로 파싱합니다. 마지막으로 invoke() 메서드를 사용하여 스페인어로 'hello.'라는 텍스트를 번역합니다.		
실습 요약	체인을 구성하고 이를 통해 번역 요청을 수행하는 과정을 실습합니다. 이는 다양한 구성 요소를 연결하여 NLP 작업을 수행하는 방법을 보여줍니다.		

08-013 체인 구성 및 실행

실습번호	08-013	실습명	체인 구성 및 실행
실습 코드	`chain = prompt_template ¦ model ¦ parser` `chain.invoke({"language": "korean", "text": "hello."})`		
실습 가이드	이 실습에서는 앞서 구성한 체인을 사용하여 한국어로 'hello.'라는 텍스트를 번역합니다. 체인은 프롬프트 템플릿, 모델, 출력 파서를 연결하여 입력 텍스트를 처리합니다.		
실습 요약	체인을 사용하여 입력 텍스트를 한국어로 번역하는 과정을 실습합니다. 이는 다양한 NLP 작업을 효과적으로 수행하기 위한 체인 구성 방법을 보여줍니다.		

08-014 PromptTemplate 및 체인 구성

실습번호	08-014	실습명	PromptTemplate 및 체인 구성
실습 코드	``` from langchain_openai import ChatOpenAI from langchain_core.prompts import PromptTemplate from langchain_core.output_parsers import StrOutputParser llm = ChatOpenAI(model="gpt-4o-mini") prompt = PromptTemplate.from_template("{topic} 에 대하여 한 문장으로 설명해 줘.") chain = prompt ¦ llm ¦ StrOutputParser() ```		
실습 가이드	이 실습에서는 ChatOpenAI 모델과 PromptTemplate을 사용하여 주제를 설명하는 체인을 구성합니다. 사용자가 입력한 주제에 대해 한 문장으로 설명해주는 프롬프트 템플릿을 설정하고, 이를 모델과 연결하여 최종 출력 결과를 문자열로 파싱합니다.		
실습 요약	PromptTemplate을 사용하여 주제를 설명하는 체인을 구성하는 과정을 실습합니다. 이는 사용자 정의 프롬프트를 활용하여 모델과 상호작용하는 방법을 보여줍니다.		

08-015 토픽에 대한 스트림 생성 및 출력

실습번호	08-015	실습명	토픽에 대한 스트림 생성 및 출력
실습 코드	# '인공지능' 토픽에 대한 스트림을 생성하고 반복합니다. for token in chain.stream({"topic": "인공지능"}): # 스트림에서 받은 데이터의 내용을 출력합니다. 줄바꿈 없이 이어서 출력하고, 버퍼를 즉시 비웁니다. print(token, end="", flush=True)		
실습 가이드	이 실습에서는 구성한 체인을 사용하여 '인공지능'이라는 토픽에 대한 설명을 스트림 형태로 생성하고, 각 토큰을 반복하여 출력합니다. 줄바꿈 없이 내용을 이어서 출력하고, 버퍼를 즉시 비워 실시간으로 결과를 보여줍니다.		
실습 요약	주어진 토픽에 대한 설명을 스트림 형태로 생성하고 출력하는 과정을 실습합니다. 이는 모델의 출력 결과를 실시간으로 확인하는 방법을 보여줍니다.		

08-016 주제에 대한 invoke 호출

실습번호	08-016	실습명	주제에 대한 invoke 호출
실습 코드	# chain 객체의 invoke 메서드를 호출하고, '삼성전자'라는 주제로 딕셔너리를 전달합니다. chain.invoke({"topic": "삼성전자"})		
실습 가이드	이 실습에서는 구성한 체인의 invoke 메서드를 사용하여 '삼성전자'라는 주제에 대한 설명을 요청합니다. 주제 정보를 딕셔너리 형식으로 전달하여 모델이 해당 주제에 대해 응답하도록 합니다.		
실습 요약	체인의 invoke 메서드를 호출하여 특정 주제에 대한 설명을 요청하는 과정을 실습합니다. 이는 모델이 주제에 따라 적절한 출력을 생성하는 방법을 보여줍니다.		

08-017 주제 리스트 배치 처리

실습번호	08-017	실습명	주제 리스트 배치 처리
실습 코드	`# 주어진 토픽 리스트를 배치 처리하는 함수 호출` `chain.batch([{"topic": "온디바이스AI"}, {"topic": "가우스AI"}])`		
실습 가이드	이 실습에서는 주어진 주제 리스트를 배치 처리하여 여러 주제에 대한 설명을 동시에 요청합니다. chain.batch() 메서드를 사용하여 각 주제 정보를 포함한 딕셔너리 리스트를 전달하면, 모델이 각각의 주제에 대한 응답을 생성합니다.		
실습 요약	주제 리스트를 배치 처리하여 여러 주제에 대한 설명을 동시에 요청하는 과정을 실습합니다. 이는 효율적으로 여러 요청을 처리하는 방법을 보여줍니다.		

08-018 주제 리스트 배치 처리 (동시 처리 설정)

실습번호	08-018	실습명	주제 리스트 배치 처리 (동시 처리 설정)
실습 코드	 chain.batch([{"topic": "머신러닝"}, {"topic": "인공지능"}, {"topic": "LangChain"}, {"topic": "ChatGPT"}, {"topic": "Llama"},], config={"max_concurrecy": 3},)		
실습 가이드	이 실습에서는 주어진 주제 리스트를 배치 처리하면서 동시 처리 개수를 설정합니다. chain.batch() 메서드를 사용하여 여러 주제 정보를 전달하고, max_concurrecy 파라미터를 통해 동시에 처리할 요청의 수를 제한합니다.		
실습 요약	주제 리스트를 배치 처리하고 동시 처리 개수를 설정하는 과정을 실습합니다. 이는 효율적으로 여러 요청을 관리하고, 처리 성능을 최적화하는 방법을 보여줍니다.		

08-019 한국어에서 영어로 번역 요청

실습번호	08-019	실습명	한국어에서 영어로 번역 요청

실습 코드

```
from langchain_core.messages import HumanMessage, SystemMessage
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o-mini",  # 사용할 모델을 "gpt-4o-mini"로 지정.
    max_tokens=2048,      # 모델이 반환할 최대 토큰 수를 2048로 설정
    temperature=0.1,      # 모델의 응답 창의성을 조절하는 파라미터. 0.1로 설정하여 더 예측 가능한 응답을 유도.
)

messages = [
    SystemMessage(content="Translate the following from Korean into English"),
    HumanMessage(content="안녕하세요. 만나서 반갑습니다."),
]

result = llm.invoke(messages)
result
```

실습 가이드

이 실습에서는 ChatOpenAI 모델을 사용하여 주어진 한국어 문장을 영어로 번역 요청합니다. SystemMessage를 통해 번역 요청의 맥락을 설정하고, HumanMessage로 번역할 문장을 입력합니다. invoke() 메서드를 통해 번역 결과를 얻습니다.

실습 요약

한국어 문장을 영어로 번역하는 요청을 수행하는 과정을 실습합니다. 이는 NLP 모델의 번역 능력을 활용하는 방법을 보여줍니다.

08-020 PromptTemplate 객체 생성

실습번호	08-020	실습명	PromptTemplate 객체 생성
실습 코드	 `from langchain_core.prompts import PromptTemplate` `# {country}는 이후에 값이 들어갈 자리를 의미` `template = "{country}의 수도는 어디인가요?"` `# from_template 메소드를 이용하여 PromptTemplate 객체 생성` `prompt = PromptTemplate.from_template(template)` `prompt`		
실습 가이드	이 실습에서는 PromptTemplate을 사용하여 특정 형식의 질문을 생성하는 객체를 만듭니다. {country}는 사용자가 입력할 값이 들어갈 자리 표시자로 사용되며, 이를 통해 동적으로 질문을 구성할 수 있습니다.		
실습 요약	PromptTemplate 객체를 생성하여 동적인 질문 형식을 구성하는 과정을 실습합니다. 이는 사용자 입력을 기반으로 모델과 상호작용하는 방법을 보여줍니다.		

08-021 PromptTemplate을 통한 프롬프트 생성

실습번호	08-021	실습명	PromptTemplate을 통한 프롬프트 생성
실습 코드	`# format 메소드를 이용하여 변수에 값을 넣어 프롬프트 생성` `prompt = prompt.format(country="대한민국")` `prompt`		
실습 가이드	이 실습에서는 생성한 PromptTemplate 객체의 format 메소드를 사용하여 {country} 자리 표시자에 '대한민국' 값을 넣어 최종 질문을 생성합니다. 이 방식으로 동적인 질문을 쉽게 만들 수 있습니다.		
실습 요약	PromptTemplate의 format 메소드를 사용하여 사용자 지정 값을 포함한 질문을 생성하는 과정을 실습합니다. 이는 입력값에 따라 질문을 동적으로 구성하는 방법을 보여줍니다.		

08-022 PromptTemplate 객체 생성

실습번호	08-022	실습명	PromptTemplate 객체 생성
실습 코드	 `# PromptTemplate 객체를 활용하여 prompt_template 생성` `prompt = PromptTemplate(` `  template=template,` `  input_variables=["country"],` `)` `prompt`		
실습 가이드	이 실습에서는 PromptTemplate 클래스를 사용하여 template과 input_variables를 명시하여 PromptTemplate 객체를 생성합니다. input_variables는 나중에 값을 대입할 수 있는 변수 목록입니다.		
실습 요약	PromptTemplate 객체를 생성하여 입력 변수를 정의하는 과정을 실습합니다. 이는 질문 형식을 동적으로 구성하는 데 필요한 기본 설정을 제공합니다.		

08-023 프롬프트 생성 및 변수 삽입

실습번호	08-023	실습명	프롬프트 생성 및 변수 삽입
실습 코드	`# prompt 생성` `prompt.format(country="대한민국")`		
실습 가이드	이 실습에서는 생성된 PromptTemplate 객체를 사용하여 format 메소드를 통해 '대한민국' 값을 넣어 최종 질문을 생성합니다. 이 과정은 동적 프롬프트 생성의 기초적인 예입니다.		
실습 요약	PromptTemplate을 사용하여 특정 변수에 값을 넣어 프롬프트를 생성하는 과정을 실습합니다. 이는 사용자의 입력에 따라 질문을 유연하게 구성하는 방법을 보여줍니다.		

08-024 PromptTemplate 객체 생성 (부분 변수 포함)

실습번호	08-024	실습명	PromptTemplate 객체 생성 (부분 변수 포함)
실습 코드			

```
# template 정의
template = "{country1}과 {country2}의 수도는 각각 어디인가요?"
# PromptTemplate 객체를 활용하여 prompt_template 생성
prompt = PromptTemplate(
    template=template,
    input_variables=["country1"],
    partial_variables={
        "country2": "미국"  # dictionary 형태로 partial_variables를 전달
    },
)
prompt
```

실습 가이드	이 실습에서는 PromptTemplate 객체를 생성할 때 부분 변수를 사용하여 template에 포함된 변수 중 일부에 값을 미리 설정합니다. country2에 대한 값은 '미국'으로 설정하고, country1은 이후에 입력될 수 있도록 합니다.
실습 요약	부분 변수를 포함한 PromptTemplate 객체를 생성하는 과정을 실습합니다. 이는 여러 변수를 효율적으로 관리하는 방법을 보여줍니다.

08-025 부분 변수를 사용한 프롬프트 생성

실습번호	08-025	실습명	부분 변수를 사용한 프롬프트 생성
실습 코드	`prompt.format(country1="대한민국")`		
실습 가이드	이 실습에서는 생성된 PromptTemplate 객체를 사용하여 format 메소드를 통해 '대한민국' 값을 넣어 최종 질문을 생성합니다. country2는 미리 설정된 값인 '미국'으로 사용됩니다.		
실습 요약	부분 변수를 사용하여 최종 프롬프트를 생성하는 과정을 실습합니다. 이는 여러 입력 변수를 효과적으로 활용하는 방법을 보여줍니다.		

08-026 부분 변수로 PromptTemplate 수정

실습번호	08-026	실습명	부분 변수로 PromptTemplate 수정
실습 코드	`prompt_partial = prompt.partial(country2="일본")` `prompt_partial`		
실습 가이드	이 실습에서는 기존의 PromptTemplate 객체에서 partial 메소드를 사용하여 country2 변수의 값을 '일본'으로 수정합니다. 이를 통해 새로운 PromptTemplate 객체를 생성할 수 있습니다.		
실습 요약	부분 변수를 사용하여 PromptTemplate을 수정하는 과정을 실습합니다. 이는 기존 프롬프트를 기반으로 새로운 프롬프트를 유연하게 구성하는 방법을 보여줍니다.		

08-027 수정된 PromptTemplate으로 프롬프트 생성

실습번호	08-027	실습명	수정된 PromptTemplate으로 프롬프트 생성
실습 코드	`prompt_partial.format(country1="대한민국")`		
실습 가이드	이 실습에서는 수정된 PromptTemplate 객체를 사용하여 format 메소드를 통해 '대한민국' 값을 넣어 최종 질문을 생성합니다. country2는 이전에 설정된 값인 '일본'으로 사용됩니다.		
실습 요약	수정된 PromptTemplate을 사용하여 최종 프롬프트를 생성하는 과정을 실습합니다. 이는 변수 값을 변경하여 다양한 질문을 생성하는 방법을 보여줍니다.		

08-028 현재 날짜 가져오기

실습번호	08-028	실습명	현재 날짜 가져오기
실습 코드	 from datetime import datetime def get_date(): return datetime.now().strftime("%m월%d일") get_date() 		
실습 가이드	이 실습에서는 현재 날짜를 가져오는 함수를 정의합니다. get_date() 함수는 현재 날짜를 '월 일' 형식으로 반환합니다.		
실습 요약	현재 날짜를 특정 형식으로 가져오는 함수를 실습합니다. 이는 날짜 형식을 다루는 기본적인 방법을 보여줍니다.		

08-029 PromptTemplate 생성 (함수 사용)

실습번호	08-029	실습명	PromptTemplate 생성 (함수 사용)
실습 코드	 prompt = PromptTemplate(template="오늘은 {date} 입니다. 과거에 있었던 주요한 이벤트를 {n}개를 알려주세요.", input_variables=["n"], partial_variables={ "date": get_date })		
실습 가이드	이 실습에서는 현재 날짜를 반환하는 함수를 사용하여 PromptTemplate을 생성합니다. template 문자열에서 {date}는 get_date() 함수의 결과로 대체되며, {n}은 사용자가 입력할 값으로 설정됩니다.		
실습 요약	함수를 사용하여 부분 변수를 포함한 PromptTemplate을 생성하는 과정을 실습합니다. 이는 동적인 날짜 정보를 프롬프트에 포함하는 방법을 보여줍니다.		

08-030 프롬프트 생성 및 변수 삽입

실습번호	08-030	실습명	프롬프트 생성 및 변수 삽입
실습 코드	`# prompt 생성` `prompt.format(n=3)`		
실습 가이드	이 실습에서는 생성된 PromptTemplate 객체를 사용하여 format 메소드를 통해 {n} 자리에 3을 넣어 최종 질문을 생성합니다. {date}는 get_date() 함수의 결과로 대체됩니다.		
실습 요약	PromptTemplate을 사용하여 특정 변수에 값을 넣어 프롬프트를 생성하는 과정을 실습합니다. 이는 사용자의 입력에 따라 질문을 유연하게 구성하는 방법을 보여줍니다.		

08-031 체인 실행 및 결과 확인

실습번호	08-031	실습명	체인 실행 및 결과 확인
실습 코드	`# chain 을 실행 후 결과를 확인합니다.` `chain = prompt | llm` `print(chain.invoke(3).content)`		
실습 가이드	이 실습에서는 PromptTemplate과 ChatOpenAI 모델을 연결하여 체인을 생성합니다. chain.invoke(3)을 호출하여 입력값으로 3을 전달하고, 최종 결과를 출력합니다.		
실습 요약	체인을 실행하여 모델의 응답을 확인하는 과정을 실습합니다. 이는 입력값에 따른 모델의 동작을 검토하는 데 유용합니다.		

08-032 JAEN에서 파일 다운로드

실습번호	08-032	실습명	JAEN에서 파일 다운로드
실습 코드	`from JAEN import download_file` `download_file('capitalyaml')`		
실습 가이드	이 실습에서는 JAEN 라이브러리를 사용하여 'capitalyaml'이라는 파일을 다운로드합니다. download_file 함수를 호출하여 해당 파일을 다운로드하고, 이후에 사용할 수 있도록 준비합니다.		
실습 요약	JAEN 라이브러리를 사용하여 파일을 다운로드하는 과정을 실습합니다. 이는 데이터나 리소스를 가져오는 데 유용한 방법입니다.		

08-033 YAML 파일에서 프롬프트 로드

실습번호	08-033	실습명	YAML 파일에서 프롬프트 로드
실습 코드	``` from langchain_core.prompts import load_prompt prompt = load_prompt("capital_info.yaml", encoding="utf-8") prompt ```		
실습 가이드	이 실습에서는 YAML 파일에서 프롬프트를 로드합니다. load_prompt 함수를 사용하여 'capital_info.yaml' 파일을 읽고, 해당 파일에 정의된 프롬프트를 사용하여 모델과 상호작용할 수 있게 합니다.		
실습 요약	YAML 파일에서 프롬프트를 로드하는 과정을 실습합니다. 이는 외부 파일에서 설정된 프롬프트를 효과적으로 활용하는 방법을 보여줍니다.		

08-034 ChatPromptTemplate 생성 및 메시지 생성

실습번호	08-034	실습명	ChatPromptTemplate 생성 및 메시지 생성
실습 코드	 from langchain_core.prompts import ChatPromptTemplate chat_template = ChatPromptTemplate.from_messages([# role, message ("system", "당신은 친절한 AI 어시스턴트입니다. 당신의 이름은 {name}입니다."), ("human", "반가워요!"), ("ai", "안녕하세요! 무엇을 도와드릴까요?"), ("human", "{user_input}"),]) # 챗 message 를 생성합니다. messages = chat_template.format_messages(name="가우스", user_input="당신의 이름은 무엇입니까?") messages		
실습 가이드	이 실습에서는 ChatPromptTemplate을 사용하여 여러 역할에 대한 메시지를 구성합니다. format_messages 메소드를 사용하여 {name} 및 {user_input} 변수에 값을 삽입하여 최종 메시지를 생성합니다.		
실습 요약	ChatPromptTemplate을 사용하여 메시지를 생성하는 과정을 실습합니다. 이는 여러 역할 간의 상호작용을 정의하는 데 유용합니다.		

08-035 메시지를 통해 모델에 요청

실습번호	08-035	실습명	메시지를 통해 모델에 요청
실습 코드	`llm.invoke(messages).content`		
실습 가이드	이 실습에서는 생성된 메시지를 사용하여 ChatOpenAI 모델에 요청을 보냅니다. llm.invoke() 메서드를 호출하여 메시지를 전달하고, 모델의 응답 내용을 확인합니다.		
실습 요약	생성된 메시지를 통해 모델에 요청하고, 그 결과를 확인하는 과정을 실습합니다. 이는 NLP 모델과의 상호작용을 효과적으로 관리하는 방법을 보여줍니다.		

08-036 체인 실행 및 모델에 요청

실습번호	08-036	실습명	체인 실행 및 모델에 요청
실습 코드	`chain = chat_template ¦ llm` `chain.invoke({"name": "가우스", "user_input": "당신의 이름은 무엇입니까?"}).content`		
실습 가이드	이 실습에서는 ChatPromptTemplate과 ChatOpenAI 모델을 연결하여 체인을 구성합니다. chain.invoke() 메서드를 호출하여 사용자 이름과 입력 문장을 전달하고, 모델의 응답 내용을 확인합니다.		
실습 요약	체인을 실행하여 입력값에 대한 모델의 응답을 확인하는 과정을 실습합니다. 이는 모델과의 상호작용을 효과적으로 관리하는 방법을 보여줍니다.		

08-037 퓨샷 예시 데이터 정의

실습번호	08-037	실습명	퓨샷 예시 데이터 정의

실습 코드

```
# 퓨샷 예시
# 예제 데이터: 가전제품 고장과 대처 방법
examples = [
    {
        "question": "세탁기가 작동하지 않아요. 어떻게 해야 하나요?",
        "answer": """
1. 세탁기의 전원 케이블이 콘센트에 제대로 연결되어 있는지 확인해 주세요.
2. 세탁기의 전원 스위치가 "ON" 상태인지 확인해 주세요.
3. 세탁기의 문이 완전히 닫혀 있는지 확인해 주세요. 세탁기는 문이 열려 있을 때 작동하지 않을 수 있습니다.
4. 위의 방법으로 문제가 해결되지 않으면, 삼성 서비스 웹사이트를 방문하거나 가까운 서비스 센터에 문의하여 전문적인 점검을 받으세요.
""",
    },
    {
    "question": "냉장고가 시원하지 않아요. 어떻게 해야 하나요?",
    "answer": """
1. 냉장고의 온도 조절기가 적절히 설정되어 있는지 확인해 주세요. 너무 높은 온도 설정은 냉장고의 성능을 저하시킬 수 있습니다.
2. 냉장고의 문이 제대로 닫혀 있는지 확인해 주세요. 문이 제대로 닫히지 않으면 냉기가 새어나갈 수 있습니다.
3. 냉장고의 팬과 통풍구가 막히지 않았는지 확인해 주세요. 공기 흐름이 원활해야 냉각이 제대로 이루어집니다.
4. 냉장고의 뒷면과 측면에 먼지나 이물질이 쌓여 있는지 확인하고 청소해 주세요. 먼지가 쌓이면 냉각 성능이 저하될 수 있습니다.
5. 위의 방법으로 문제가 해결되지 않으면, 삼성 서비스 웹사이트를 방문하거나 가까운 서비스 센터에 문의하여 전문적인 점검을 받으세요.
"""
    },
    {
    "question": "에어컨이 작동하지 않아요. 어떻게 해야 하나요?",
    "answer": """
1. 에어컨의 전원 코드가 제대로 연결되어 있는지 확인해 주세요.
2. 에어컨의 전원 스위치가 "ON" 상태인지 확인해 주세요.
3. 리모컨의 배터리가 충분한지 확인하고, 배터리를 교체해 보세요.
4. 에어컨의 필터가 막히거나 더러워졌는지 확인하고 청소해 주세요.
5. 위의 방법으로 문제가 해결되지 않으면, 삼성 서비스 웹사이트를 방문하거나 가까운 서비스 센터에 문의하여 전문적인 점검을 받으세요.
"""    }
]
```

실습 가이드	이 실습에서는 가전제품 고장과 관련된 질문과 그에 대한 대처 방법을 포함하는 예제 데이터를 정의합니다. 각 예제는 질문과 해당 질문에 대한 적절한 답변으로 구성됩니다.
실습 요약	퓨샷 예시 데이터를 정의하는 과정을 실습합니다. 이는 특정 주제에 대한 질문과 답변을 구조화하여 모델 훈련에 활용하는 방법을 보여줍니다.

08-038 예제 프롬프트 생성 및 출력

실습번호	08-038	실습명	예제 프롬프트 생성 및 출력
실습 코드	`example_prompt = PromptTemplate.from_template(` `  "Question:\n{question}\nAnswer:\n{answer}"` `)` `print(example_prompt.format(**examples[0]))`		
실습 가이드	이 실습에서는 예제 데이터에서 첫 번째 항목을 사용하여 질문과 답변 형식의 프롬프트를 생성합니다. format 메소드를 통해 'question'과 'answer' 변수를 채워 최종 프롬프트를 출력합니다.		
실습 요약	예제 데이터를 기반으로 질문과 답변 형식의 프롬프트를 생성하고 출력하는 과정을 실습합니다. 이는 모델에게 제공할 질문과 그에 대한 답변 형식을 구성하는 방법을 보여줍니다.		

08-039 FewShotPromptTemplate 생성 및 출력

실습번호	08-039	실습명	FewShotPromptTemplate 생성 및 출력
실습 코드	 `from langchain_core.prompts.few_shot import FewShotPromptTemplate` `prompt = FewShotPromptTemplate(` `  examples=examples,` `  example_prompt=example_prompt,` `  suffix="Question:\n{question}\nAnswer:",` `  input_variables=["question"],` `)` `question = "공기청정기를 작동해도 미세먼지 농도가 줄어들지 않아요."` `final_prompt = prompt.format(question=question)` `print(final_prompt)`		
실습 가이드	이 실습에서는 FewShotPromptTemplate을 사용하여 주어진 예제 데이터를 기반으로 질문에 대한 답변 형식의 프롬프트를 생성합니다. 입력 변수로 질문을 받고, 기존 예제와 함께 최종 프롬프트를 출력합니다.		
실습 요약	FewShotPromptTemplate을 생성하고, 주어진 질문을 바탕으로 최종 프롬프트를 출력하는 과정을 실습합니다. 이는 모델이 특정 질문에 대해 학습된 예제에 기반하여 응답할 수 있도록 돕습니다.		

08-040 이메일 대화 예시 정의

실습번호	08-040	실습명	이메일 대화 예시 정의
실습 코드	email_conversation = """From: 김민수 (minsoo.kim@samsung.com) To: 이서윤 (seoyoon.lee@samsung.com) Subject: "GALAXY" 노트북 유통 협력 및 미팅 일정 제안 안녕하세요, 이서윤 프로님, GALAXY 모델에 대한 상세한 브로슈어를 요청드립니다. 특히 기술 사양, 배터리 성능, 그리고 디자인 측면에 대한 정보가 필요합니다. 이를 통해 저희가 제안할 유통 전략과 마케팅 계획을 보다 구체화할 수 있을 것입니다. 또한, 관련 내용을 더 깊이 논의하기 위해 다음 주 화요일(1월 15일) 오전 10시에 이야기를 나눌 수 있을까요? 감사합니다. 김민수 프로 삼성전자 """		
실습 가이드	이 실습에서는 이메일 대화를 문자열로 정의합니다. 이메일의 발신자, 수신자, 제목, 내용 등이 포함되어 있으며, 이는 후속 작업에서 텍스트 분석이나 모델 학습 등에 활용할 수 있습니다.		
실습 요약	이메일 대화 예시를 정의하는 과정을 실습합니다. 이는 텍스트 데이터 세트를 생성하여 모델의 입력으로 사용할 수 있는 방법을 보여줍니다.		

08-041 이메일 내용 추출 체인 실행

실습번호	08-041	실습명	이메일 내용 추출 체인 실행
실습 코드	 from itertools import chain from langchain_core.prompts import PromptTemplate from langchain_core.output_parsers import StrOutputParser prompt = PromptTemplate.from_template("다음의 이메일 내용중 중요한 내용을 추출해 주세요.\n\n{email_conversation}") llm = ChatOpenAI(temperature=0, model_name="gpt-4o") parser = StrOutputParser() # 문자열 출력 파서 생성 chain = prompt ¦ llm ¦ parser # 문자열 출력 파서를 llm 출력 뒤에 체인 연결 result = chain.invoke({"email_conversation": email_conversation}) print(result)		
실습 가이드	이 실습에서는 PromptTemplate을 사용하여 이메일 내용에서 중요한 정보를 추출하는 요청을 생성합니다. 문자열 출력 파서를 통해 모델의 결과를 처리하고, 최종 결과를 출력합니다.		
실습 요약	이메일 내용에서 중요한 정보를 추출하는 체인을 실행하는 과정을 실습합니다. 이는 모델을 활용하여 특정 정보를 효과적으로 추출하는 방법을 보여줍니다.		

08-042 PydanticOutputParser 생성

실습번호	08-042	실습명	PydanticOutputParser 생성
실습 코드			

```
from langchain_core.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field

class EmailSummary(BaseModel):
    person: str = Field(description="메일을 보낸 사람")
    email: str = Field(description="메일을 보낸 사람의 이메일 주소")
    subject: str = Field(description="메일 제목")
    summary: str = Field(description="메일 본문을 요약한 텍스트")
    date: str = Field(description="메일 본문에 언급된 미팅 날짜와 시간")

# PydanticOutputParser 생성
parser = PydanticOutputParser(pydantic_object=EmailSummary)
```

실습 가이드	이 실습에서는 Pydantic을 사용하여 이메일 요약 모델을 정의하고, 이를 기반으로 PydanticOutputParser를 생성합니다. 이 파서는 모델의 출력을 Pydantic 모델 형식으로 변환하여 구조화된 데이터를 제공합니다.
실습 요약	PydanticOutputParser를 생성하고 이를 통해 구조화된 데이터를 정의하는 과정을 실습합니다. 이는 모델의 출력을 더 쉽게 다룰 수 있도록 돕습니다.

08-043 PydanticOutputParser의 형식 지침 출력

실습번호	08-043	실습명	PydanticOutputParser의 형식 지침 출력
실습 코드	`# instruction 을 출력합니다.` `print(parser.get_format_instructions())`		
실습 가이드	이 실습에서는 PydanticOutputParser에서 제공하는 형식 지침을 출력합니다. 이 지침은 모델의 출력을 Pydantic 객체에 맞게 형식화하는 방법을 설명합니다.		
실습 요약	PydanticOutputParser의 형식 지침을 확인하는 과정을 실습합니다. 이는 데이터 형식화 및 처리 방법을 이해하는 데 도움이 됩니다.		

08-044 프롬프트 템플릿 및 체인 생성

실습번호	08-044	실습명	프롬프트 템플릿 및 체인 생성
실습 코드	# 프롬프트 템플릿 작성 prompt = PromptTemplate.from_template(""" You are a helpful assistant. Please answer the following questions in KOREAN. QUESTION: {question} EMAIL CONVERSATION: {email_conversation} FORMAT: {format} """) # format 에 PydanticOutputParser의 부분 포맷팅(partial) 추가 prompt = prompt.partial(format=parser.get_format_instructions()) # chain 을 생성합니다. chain = prompt ¦ llm ¦ parser			
실습 가이드	이 실습에서는 KOREAN으로 질문에 답하는 보조 도우미를 위한 프롬프트 템플릿을 작성합니다. 또한, PydanticOutputParser에서 제공하는 형식 지침을 부분 포맷팅하여 최종 프롬프트에 추가합니다. 이후, 프롬프트 템플릿과 모델, 파서를 연결하여 체인을 생성합니다.		
실습 요약	프롬프트 템플릿을 작성하고, 이를 기반으로 체인을 생성하는 과정을 실습합니다. 이는 특정 형식으로 정보를 요청하고 처리하는 방법을 보여줍니다.		

08-045 체인 실행 및 결과 출력

실습번호	08-045	실습명	체인 실행 및 결과 출력
실습 코드	``` # chain 을 실행하고 결과를 출력합니다. result = chain.invoke({ "email_conversation": email_conversation, "question": "이메일 내용중 주요 내용을 추출해 주세요.", }) ```		
실습 가이드	이 실습에서는 앞서 생성한 체인을 사용하여 이메일 대화와 질문을 입력으로 제공하고, 모델의 응답을 받아 출력합니다. chain.invoke() 메서드를 통해 이메일 내용 중 주요 정보를 추출하는 요청을 수행합니다.		
실습 요약	체인을 실행하여 이메일 내용에서 주요 정보를 추출하는 요청을 수행하는 과정을 실습합니다. 이는 모델의 분석 능력을 활용하는 방법을 보여줍니다.		

08-046 JAEN에서 PDF 파일 다운로드

실습번호	08-046	실습명	JAEN에서 PDF 파일 다운로드
실습 코드	from JAEN import download_file download_file('PDF') #온디바이스 AI 기술동향 및 발전방향.pdf		
실습 가이드	이 실습에서는 JAEN 라이브러리를 사용하여 '온디바이스 AI 기술동향 및 발전방향.pdf' 파일을 다운로드합니다. download_file 함수를 호출하여 해당 PDF 파일을 다운로드하고, 이후에 사용할 수 있도록 준비합니다.		
실습 요약	JAEN 라이브러리를 사용하여 PDF 파일을 다운로드하는 과정을 실습합니다. 이는 데이터나 리소스를 가져오는 데 유용한 방법입니다.		

08-047 PDF 파일 로딩 설정

실습번호	08-047	실습명	PDF 파일 로딩 설정
실습 코드	 from langchain_community.document_loaders import PyPDFLoader # 예제 파일 경로 FILE_PATH = "온디바이스 AI 기술동향 및 발전방향.pdf" # 로더 설정 loader = PyPDFLoader(FILE_PATH)		
실습 가이드	이 실습에서는 PyPDFLoader를 사용하여 특정 PDF 파일을 로드하는 설정을 합니다. FILE_PATH 변수를 통해 로드할 PDF 파일의 경로를 지정하고, 해당 파일을 로드할 수 있는 로더 객체를 생성합니다.		
실습 요약	PDF 파일을 로드하는 설정 과정을 실습합니다. 이는 문서 데이터 처리의 첫 번째 단계로, 파일을 프로그램에서 사용할 수 있도록 준비하는 방법을 보여줍니다.		

08-048 PDF 파일 로딩 및 문서 수 확인

실습번호	08-048	실습명	PDF 파일 로딩 및 문서 수 확인
실습 코드	```# PDF 로더 docs = loader.load() # 로드된 문서의 수 확인 len(docs)```		
실습 가이드	이 실습에서는 설정한 PDF 로더를 사용하여 PDF 파일을 로드하고, 로드된 문서의 수를 확인합니다. loader.load() 메서드를 호출하여 PDF 파일의 내용을 읽어오고, len() 함수를 사용하여 읽어온 문서의 수를 출력합니다.		
실습 요약	PDF 파일을 로드하고, 로드된 문서의 수를 확인하는 과정을 실습합니다. 이는 문서 데이터 처리의 두 번째 단계로, 읽어온 내용을 효과적으로 관리하는 방법을 보여줍니다.		

08-049 첫 번째 문서 확인

실습번호	08-049	실습명	첫 번째 문서 확인
실습 코드	# 첫번째 문서 확인 docs[0]		
실습 가이드	이 실습에서는 로드한 PDF 문서 중 첫 번째 문서의 내용을 확인합니다. docs 리스트의 첫 번째 요소를 출력하여 해당 문서의 정보를 확인합니다.		
실습 요약	로드된 PDF 파일의 첫 번째 문서를 확인하는 과정을 실습합니다. 이는 문서의 내용을 검토하고 필요한 정보를 추출하는 데 유용합니다.		

08-050 PDF 파일을 generator 방식으로 로드

실습번호	08-050	실습명	PDF 파일을 generator 방식으로 로드
실습 코드	`# generator 방식으로 문서 로드` `for doc in loader.lazy_load():` `  print(doc.metadata)`		
실습 가이드	이 실습에서는 PDF 파일을 generator 방식으로 로드하여 각 문서의 메타데이터를 출력합니다. lazy_load() 메서드를 사용하여 메모리 효율적으로 문서를 로드하고, 각 문서의 메타데이터를 확인합니다.		
실습 요약	PDF 파일을 generator 방식으로 로드하고 각 문서의 메타데이터를 확인하는 과정을 실습합니다. 이는 대량의 문서 데이터를 효율적으로 처리하는 방법을 보여줍니다.		

08-051 PDF 파일을 Async 방식으로 로드

실습번호	08-051	실습명	PDF 파일을 Async 방식으로 로드
실습 코드	`# 문서를 async 방식으로 로드` `adocs = loader.aload()`		
실습 가이드	이 실습에서는 PDF 파일을 비동기(async) 방식으로 로드합니다. aload() 메서드를 사용하여 문서를 로드하고, 이를 통해 비동기적으로 파일을 처리할 수 있습니다.		
실습 요약	PDF 파일을 비동기 방식으로 로드하는 과정을 실습합니다. 이는 문서 로딩 시 효율성을 높이고, 다른 작업과 병행하여 진행할 수 있는 방법을 보여줍니다.		

08-052 비동기 문서 로드

실습번호	08-052	실습명	비동기 문서 로드
실습 코드	`# 문서 로드` `await adocs`		
실습 가이드	이 실습에서는 비동기(async) 방식으로 로드한 문서를 실제로 가져오기 위해 await 키워드를 사용합니다. 이는 비동기적으로 로드된 문서를 기다리고, 해당 문서를 사용할 수 있게 합니다.		
실습 요약	비동기 방식으로 로드한 문서를 가져오는 과정을 실습합니다. 이는 효율적인 문서 처리를 위한 비동기 작업의 활용 방법을 보여줍니다.		

08-053 문서 분할 및 첫 번째 문서 확인

실습번호	08-053	실습명	문서 분할 및 첫 번째 문서 확인
실습 코드	 `from langchain.text_splitter import CharacterTextSplitter` `# 문자열 분할기 설정` `text_splitter = CharacterTextSplitter(chunk_size=200, chunk_overlap=0)` `# 문서 분할` `docs = loader.load_and_split(text_splitter=text_splitter)` `# 로드된 문서의 수 확인` `print(len(docs))` `# 첫번째 문서 확인` `docs[0]`		
실습 가이드	이 실습에서는 CharacterTextSplitter를 사용하여 로드한 문서를 지정된 크기(200자)로 분할합니다. chunk_overlap을 0으로 설정하여 겹치지 않는 문서 조각을 생성합니다. 문서 분할 후, 로드된 문서의 수를 확인하고 첫 번째 문서의 내용을 출력합니다.		
실습 요약	문서를 특정 크기로 분할하는 과정을 실습합니다. 이는 대량의 텍스트 데이터를 효율적으로 처리하는 방법을 보여줍니다.		

08-054 PDF 로더 초기화 및 페이지 내용 확인

실습번호	08-054	실습명	PDF 로더 초기화 및 페이지 내용 확인
실습 코드	 `# 설치` `!pip install -qU rapidocr-onnxruntime` `# PDF 로더 초기화, 이미지 추출 옵션 활성화` `loader = PyPDFLoader("https://arxiv.org/pdf/1706.03762.pdf", extract_images=True)` `# PDF 페이지 로드` `docs = loader.load()` `# 페이지 내용 접근` `print(docs[3].page_content)`		
실습 가이드	이 실습에서는 rapidocr-onnxruntime 라이브러리를 설치하고, PyPDFLoader를 초기화하여 이미지 추출 옵션을 활성화합니다. 이후 PDF 페이지를 로드하고, 특정 페이지(5페이지)의 내용을 출력하여 확인합니다.		
실습 요약	PDF 로더를 초기화하고 페이지 내용을 확인하는 과정을 실습합니다. 이는 PDF 문서에서 텍스트 및 이미지를 추출하는 방법을 보여줍니다.		

08-055 PyMuPDF 로더 초기화 및 문서 내용 확인

실습번호	08-055	실습명	PyMuPDF 로더 초기화 및 문서 내용 확인
실습 코드			

```
# 설치
!pip install -qU pymupdf

from langchain_community.document_loaders import PyMuPDFLoader

# PyMuPDF 로더 인스턴스 생성
loader = PyMuPDFLoader(FILE_PATH)

# 문서 로드
docs = loader.load()

# 문서의 내용 출력
print(docs[10].page_content[:300])
```

실습 가이드	이 실습에서는 pymupdf 라이브러리를 설치하고, PyMuPDFLoader를 사용하여 PDF 파일을 로드합니다. 이후 특정 페이지(11페이지)의 내용을 출력하여 문서의 내용을 확인합니다.
실습 요약	PyMuPDF 로더를 사용하여 PDF 문서를 로드하고 특정 페이지의 내용을 확인하는 과정을 실습합니다. 이는 PDF 문서의 내용 추출을 보여줍니다.

08-056 OpenAI 임베딩 생성 및 쿼리 결과 확인

실습번호	08-056	실습명	OpenAI 임베딩 생성 및 쿼리 결과 확인
실습 코드	 from langchain_openai import OpenAIEmbeddings # OpenAI의 "text-embedding-3-large" 모델을 사용하여 임베딩을 생성합니다. embeddings = OpenAIEmbeddings(model="text-embedding-3-small") # 텍스트를 임베딩하여 쿼리 결과를 생성합니다. query_result = embeddings.embed_query("임베딩 테스트를 하기 위한 샘플 문장입니다.") query_result[:5]		
실습 가이드	이 실습에서는 OpenAIEmbeddings를 사용하여 주어진 텍스트에 대한 임베딩을 생성합니다. 'text-embedding-3-small' 모델을 사용하며, 생성된 임베딩의 일부를 출력하여 결과를 확인합니다.		
실습 요약	OpenAI 모델을 사용하여 텍스트 임베딩을 생성하고, 그 결과를 확인하는 과정을 실습합니다. 이는 텍스트 데이터를 벡터 형식으로 변환하는 방법을 보여줍니다.		

08-057 여러 텍스트 일괄 임베딩 생성

실습번호	08-057	실습명	여러 텍스트 일괄 임베딩 생성
실습 코드	 `# 여러 텍스트를 일괄 임베딩하여 벡터를 생성합니다.` `doc_result = embeddings.embed_documents(` `['임베딩 테스트를 하기 위한 샘플 문장입니다.',` `'AI Essential']` `)` `# 문서 결과의 첫 번째 요소의 길이를 반환합니다.` `len(doc_result[0])`		
실습 가이드	이 실습에서는 OpenAIEmbeddings를 사용하여 여러 텍스트에 대한 임베딩을 일괄 생성합니다. embed_documents() 메서드를 사용하여 주어진 문서 리스트에 대한 임베딩을 생성하고, 첫 번째 문서의 임베딩 벡터 길이를 확인합니다.		
실습 요약	여러 텍스트를 일괄 임베딩하여 벡터를 생성하는 과정을 실습합니다. 이는 대량의 텍스트 데이터를 효율적으로 처리하는 방법을 보여줍니다.		

08-058 캐시 지원 임베딩 생성

실습번호	08-058	실습명	캐시 지원 임베딩 생성
실습 코드	 from langchain.storage import InMemoryByteStore from langchain_openai import OpenAIEmbeddings from langchain.embeddings import CacheBackedEmbeddings # OpenAI의 "text-embedding-3-large" 모델을 사용하여 임베딩을 생성합니다. embeddings = OpenAIEmbeddings(model="text-embedding-3-small") # 인메모리 스토어를 생성합니다. store = InMemoryByteStore() # 캐시를 지원하는 임베딩 생성 cached_embeding = CacheBackedEmbeddings.from_bytes_store(underlying_embeddings=embeddings, document_embedding_cache=store, namespace=embeddings.model, # 기본 임베딩과 저장소를 사용하여 캐시 지원 임베딩을 생성) 		
실습 가이드	이 실습에서는 OpenAI의 임베딩 모델을 사용하여 임베딩을 생성하고, 인메모리 스토어를 사용하여 캐시 지원 임베딩을 생성합니다. CacheBackedEmbeddings를 사용하여 메모리 내에 임베딩 데이터를 캐시하여 성능을 개선합니다.		
실습 요약	캐시 지원 임베딩을 생성하는 과정을 실습합니다. 이는 임베딩 결과를 효율적으로 저장하고 재사용하는 방법을 보여줍니다.		

08-059 캐시 지원 임베딩으로 쿼리 결과 생성

실습번호	08-059	실습명	캐시 지원 임베딩으로 쿼리 결과 생성
실습 코드	# 텍스트를 임베딩하여 쿼리 결과를 생성합니다. query_result = cached_embeding.embed_query("임베딩 테스트를 하기 위한 샘플 문장입니다.") query_result		
실습 가이드	이 실습에서는 캐시 지원 임베딩을 사용하여 특정 텍스트에 대한 임베딩을 생성합니다. embed_query 메서드를 통해 텍스트를 임베딩하고 결과를 출력하여 확인합니다.		
실습 요약	캐시 지원 임베딩을 사용하여 텍스트 임베딩을 생성하는 과정을 실습합니다. 이는 임베딩 결과를 효율적으로 생성하는 방법을 보여줍니다.		

08-060 1024차원 임베딩 생성 및 길이 확인

실습번호	08-060	실습명	1024차원 임베딩 생성 및 길이 확인
실습 코드	# OpenAI의 "text-embedding-3-small" 모델을 사용하여 1024차원의 임베딩을 생성하는 객체를 초기화합니다. embeddings_1024 = OpenAIEmbeddings(model="text-embedding-3-small", dimensions=1024) # 주어진 텍스트를 임베딩하고 첫 번째 임베딩 벡터의 길이를 반환합니다. len(embeddings_1024.embed_documents(['AI Essential'])[0])		
실습 가이드	이 실습에서는 OpenAI의 임베딩 모델을 사용하여 1024차원의 임베딩을 생성하는 객체를 초기화합니다. 이후 주어진 텍스트에 대한 임베딩을 생성하고, 첫 번째 임베딩 벡터의 길이를 확인합니다.		
실습 요약	1024차원 임베딩을 생성하고, 임베딩 벡터의 길이를 확인하는 과정을 실습합니다. 이는 임베딩의 차원 수를 설정하고 결과를 다루는 방법을 보여줍니다.		

08-061 임베딩 대상 텍스트 정의

실습번호	08-061	실습명	임베딩 대상 텍스트 정의
실습 코드	 `# 임베딩 대상 텍스트` `sentence1 = "안녕하세요? 반갑습니다."` `sentence2 = "안녕하세요? 반갑습니다!"` `sentence3 = "안녕하세요? 만나서 반가워요."` `sentence4 = "Hi, nice to meet you."` `sentence5 = "I like to eat apples."`		
실습 가이드	이 실습에서는 임베딩을 생성할 텍스트 문장을 정의합니다. 다양한 인사말 및 문장을 포함하여 나중에 임베딩을 생성할 준비를 합니다.		
실습 요약	임베딩할 텍스트 문장을 정의하는 과정을 실습합니다. 이는 다양한 문장을 사용하여 임베딩의 성능을 테스트하는 방법을 보여줍니다.		

08-062 임베딩 수행

실습번호	08-062	실습명	임베딩 수행
실습 코드	# 임베딩 수행 sentences = [sentence1, sentence2, sentence3, sentence4, sentence5] embedded_sentences = embeddings_1024.embed_documents(sentences)		
실습 가이드	이 실습에서는 정의한 여러 문장을 임베딩하여 벡터로 변환합니다. embed_documents 메서드를 사용하여 문장 리스트에 대한 임베딩을 수행하고 결과를 저장합니다.		
실습 요약	여러 텍스트 문장을 임베딩하는 과정을 실습합니다. 이는 텍스트 데이터를 벡터 형태로 변환하는 방법을 보여줍니다.		

08-063 코사인 유사도 계산 함수 정의

실습번호	08-063	실습명	코사인 유사도 계산 함수 정의
실습 코드	`from sklearn.metrics.pairwise import cosine_similarity` `def similarity(a, b):` `  return cosine_similarity([a], [b])[0][0]`		
실습 가이드	이 실습에서는 두 임베딩 벡터 간의 코사인 유사도를 계산하는 함수를 정의합니다. sklearn의 cosine_similarity 함수를 사용하여 두 벡터의 유사도를 반환합니다.		
실습 요약	코사인 유사도를 계산하는 함수를 정의하는 과정을 실습합니다. 이는 벡터 간의 유사도를 평가하는 방법을 보여줍니다.		

08-064 유사도 계산 및 결과 출력

실습번호	08-064	실습명	유사도 계산 및 결과 출력
실습 코드	```# 유사도 계산 for i, sentence in enumerate(embedded_sentences): for j, other_sentence in enumerate(embedded_sentences): if i < j: print(f"[유사도 {similarity(sentence, other_sentence):.4f}] {sentences[i]} \t <=====> \t {sentences[j]}")```		
실습 가이드	이 실습에서는 정의된 문장 리스트에서 각 문장 쌍 간의 코사인 유사도를 계산하고 결과를 출력합니다. 두 문장 간의 유사도를 확인하기 위해 이중 루프를 사용합니다.		
실습 요약	여러 문장 간의 유사도를 계산하고 출력하는 과정을 실습합니다. 이는 문장 간의 의미적 유사성을 평가하는 방법을 보여줍니다.		

08-065 JAEN에서 키워드 파일 다운로드

실습번호	08-065	실습명	JAEN에서 키워드 파일 다운로드
실습 코드	`from JAEN import download_file` `download_file('nlp-keywords') # nlp-keywords.txt` `download_file('finance-keywords') # finance-keywords.txt`		
실습 가이드	이 실습에서는 JAEN 라이브러리를 사용하여 'nlp-keywords.txt'와 'finance-keywords.txt' 파일을 다운로드합니다. download_file 함수를 호출하여 해당 파일들을 다운로드하고, 이후에 사용할 수 있도록 준비합니다.		
실습 요약	JAEN 라이브러리를 사용하여 두 개의 키워드 파일을 다운로드하는 과정을 실습합니다. 이는 특정 주제에 대한 키워드를 쉽게 가져오는 방법을 보여줍니다.		

08-066 텍스트 파일 로드 및 문서 분할

실습번호	08-066	실습명	텍스트 파일 로드 및 문서 분할
실습 코드	 from langchain_community.document_loaders import TextLoader from langchain.text_splitter import RecursiveCharacterTextSplitter # 텍스트 분할 text_splitter = RecursiveCharacterTextSplitter(chunk_size=600, chunk_overlap=0) # 텍스트 파일을 load -> List[Document] 형태로 변환 loader1 = TextLoader("./nlp-keywords.txt", encoding='utf-8') loader2 = TextLoader("./finance-keywords.txt", encoding='utf-8') # 문서 분할 split_doc1 = loader1.load_and_split(text_splitter) split_doc2 = loader2.load_and_split(text_splitter) # 문서 개수 확인 len(split_doc1), len(split_doc2)		
실습 가이드	이 실습에서는 RecursiveCharacterTextSplitter를 사용하여 텍스트 파일을 문서로 로드하고 분할합니다. 각 텍스트 파일을 읽어와서 지정한 크기로 문서 조각으로 나누고, 분할된 문서의 개수를 확인합니다.		
실습 요약	텍스트 파일을 로드하고, 이를 문서로 분할하는 과정을 실습합니다. 이는 대량의 텍스트 데이터를 효과적으로 관리하는 방법을 보여줍니다.		

08-067 FAISS 벡터 스토어 및 임베딩 차원 크기 계산

실습번호	08-067	실습명	FAISS 벡터 스토어 및 임베딩 차원 크기 계산
실습 코드	import faiss from langchain_community.vectorstores import FAISS from langchain_community.docstore.in_memory import InMemoryDocstore from langchain_openai import OpenAIEmbeddings # 임베딩 embeddings = OpenAIEmbeddings(model="text-embedding-3-small") # 임베딩 차원 크기를 계산 dimension_size = len(embeddings.embed_query("hello world")) print(dimension_size)		
실습 가이드	이 실습에서는 OpenAIEmbeddings를 사용하여 임베딩을 생성하고, 특정 텍스트에 대한 임베딩 차원 크기를 계산합니다. embed_query 메서드를 호출하여 임베딩 벡터의 크기를 확인합니다.		
실습 요약	FAISS 벡터 스토어와 함께 임베딩의 차원 크기를 계산하는 과정을 실습합니다. 이는 벡터 데이터를 다루는 데 필요한 차원 정보를 확인하는 방법을 보여줍니다.		

08-068 FAISS 벡터 스토어 생성

실습번호	08-068	실습명	FAISS 벡터 스토어 생성
실습 코드	# DB 생성 db = FAISS.from_documents(documents=split_doc1, embedding=OpenAIEmbeddings(model="text-embedding-3-small"))		
실습 가이드	이 실습에서는 분할된 문서 리스트를 사용하여 FAISS 벡터 스토어를 생성합니다. from_documents 메서드를 호출하여 문서와 임베딩을 사용하여 벡터 스토어를 초기화합니다.		
실습 요약	FAISS 벡터 스토어를 생성하는 과정을 실습합니다. 이는 문서 데이터의 벡터화를 통해 검색 및 유사도 계산을 효율적으로 처리하는 방법을 보여줍니다.		

08-069 FAISS DB 문서 저장소 ID 확인

실습번호	08-069	실습명	FAISS DB 문서 저장소 ID 확인
실습 코드	`# 문서 저장소 ID 확인` `db.index_to_docstore_id`		
실습 가이드	이 실습에서는 생성된 FAISS 벡터 스토어에서 문서 저장소 ID를 확인합니다. index_to_docstore_id 속성을 호출하여 각 문서의 ID와 관련된 정보를 출력합니다.		
실습 요약	FAISS DB에서 문서 저장소 ID를 확인하는 과정을 실습합니다. 이는 벡터 스토어 내의 문서 식별을 확인하는 데 유용합니다.		

08-070 FAISS DB에서 저장된 문서 확인

실습번호	08-070	실습명	FAISS DB에서 저장된 문서 확인
실습 코드	`# 저장된 문서의 ID: Document 확인` `db.docstore._dict`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어의 내부 문서 저장소에서 저장된 문서들을 확인합니다. _dict 속성을 호출하여 각 문서의 ID와 내용을 확인할 수 있습니다.		
실습 요약	FAISS DB에서 저장된 문서를 확인하는 과정을 실습합니다. 이는 벡터 스토어 내의 문서 내용을 검토하는 데 유용합니다.		

08-071 FAISS DB 생성 - 문자열 리스트로

실습번호	08-071	실습명	FAISS DB 생성 - 문자열 리스트로
실습 코드	 `# 문자열 리스트로 생성` `db2 = FAISS.from_texts(` `["안녕하세요. 정말 반갑습니다.", "AI Essential 과정입니다"],` `embedding=OpenAIEmbeddings(model="text-embedding-3-small"),` `metadatas=[{"source": "텍스트문서"}, {"source": "텍스트문서"}],` `ids=["doc1", "doc2"],` `)`		
실습 가이드	이 실습에서는 문자열 리스트를 사용하여 FAISS 벡터 스토어를 생성합니다. from_texts 메서드를 사용하여 텍스트 데이터를 벡터화하고, 메타데이터 및 ID를 추가하여 문서 저장소를 초기화합니다.		
실습 요약	문자열 리스트를 기반으로 FAISS 벡터 스토어를 생성하는 과정을 실습합니다. 이는 간단한 텍스트 데이터 처리 방법을 보여줍니다.		

08-072 FAISS DB에서 저장된 내용 확인

실습번호	08-072	실습명	FAISS DB에서 저장된 내용 확인
실습 코드	# 저장된 내용 db2.docstore._dict		
실습 가이드	이 실습에서는 FAISS 벡터 스토어 db2에 저장된 내용을 확인합니다. docstore._dict 속성을 호출하여 각 문서의 ID 및 관련 내용을 출력합니다.		
실습 요약	FAISS DB에서 저장된 내용을 확인하는 과정을 실습합니다. 이는 생성된 문서의 내용을 검토하는 데 유용합니다.		

08-073 FAISS DB에서 유사도 검색

실습번호	08-073	실습명	FAISS DB에서 유사도 검색
실습 코드	# 유사도 검색 db.similarity_search("TF IDF 에 대하여 알려줘")		
실습 가이드	이 실습에서는 FAISS 벡터 스토어를 사용하여 주어진 질문에 대한 유사한 문서를 검색합니다. similarity_search 메서드를 호출하여 입력된 질문과 가장 유사한 문서들을 찾습니다.		
실습 요약	FAISS DB에서 유사도 검색을 수행하는 과정을 실습합니다. 이는 벡터 스토어를 활용하여 문서 간의 유사성을 평가하는 방법을 보여줍니다.		

08-074 FAISS DB에서 k 값 지정하여 유사도 검색

실습번호	08-074	실습명	FAISS DB에서 k 값 지정하여 유사도 검색
실습 코드	`# k 값 지정` `db.similarity_search("TF IDF 에 대하여 알려줘", k=2)`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어를 사용하여 주어진 질문에 대해 유사한 문서를 검색할 때 k 값을 지정합니다. similarity_search 메서드의 k 매개변수를 사용하여 반환할 유사한 문서의 수를 설정합니다.		
실습 요약	k 값을 지정하여 FAISS DB에서 유사도 검색을 수행하는 과정을 실습합니다. 이는 원하는 수의 결과를 제어하는 방법을 보여줍니다.		

08-075 FAISS DB에 문서 추가

실습번호	08-075	실습명	FAISS DB에 문서 추가
실습 코드	 `from langchain_core.documents import Document` `# page_content, metadata 지정` `db.add_documents(` `  [` `  Document(` `  page_content="안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼께요 ",` `  metadata={"source": "mydata.txt"},` `  )` `  ],` `  ids=["new_doc1"],` `)`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에 새로운 문서를 추가합니다. Document 객체를 생성할 때 page_content와 metadata를 지정하고, add_documents 메서드를 사용하여 문서를 추가합니다.		
실습 요약	FAISS DB에 문서를 추가하는 과정을 실습합니다. 이는 벡터 스토어에 데이터를 확장하는 방법을 보여줍니다.		

08-076 FAISS DB에서 추가된 데이터 확인

실습번호	08-076	실습명	FAISS DB에서 추가된 데이터 확인
실습 코드	`# 추가된 데이터를 확인` `db.similarity_search("안녕하세요", k=1)`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에 추가된 데이터를 확인하기 위해 similarity_search 메서드를 사용합니다. '안녕하세요'라는 쿼리를 입력하여 가장 유사한 문서를 검색하고 결과를 출력합니다.		
실습 요약	FAISS DB에서 추가된 데이터를 확인하는 과정을 실습합니다. 이는 추가한 문서의 유사성을 평가하는 방법을 보여줍니다.		

08-077 FAISS DB에 텍스트 데이터 추가

실습번호	08-077	실습명	FAISS DB에 텍스트 데이터 추가
실습 코드	 # 신규 데이터를 추가 db.add_texts(["이번엔 텍스트 데이터를 추가합니다.", "추가한 2번째 텍스트 데이터 입니다."], metadatas=[{"source": "mydata.txt"}, {"source": "mydata.txt"}], ids=["new_doc2", "new_doc3"],)		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에 텍스트 데이터를 추가합니다. add_texts 메서드를 사용하여 텍스트 리스트와 메타데이터, ID를 함께 추가합니다.		
실습 요약	FAISS DB에 텍스트 데이터를 추가하는 과정을 실습합니다. 이는 다양한 형태의 데이터를 벡터 스토어에 저장하는 방법을 보여줍니다.		

08-078 FAISS DB에서 추가된 데이터의 ID 확인

실습번호	08-078	실습명	FAISS DB에서 추가된 데이터의 ID 확인
실습 코드	# 추가된 데이터를 확인 db.index_to_docstore_id		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에서 추가된 데이터의 ID를 확인합니다. index_to_docstore_id 속성을 호출하여 각 문서의 ID와 관련된 정보를 출력합니다.		
실습 요약	FAISS DB에서 추가된 데이터의 ID를 확인하는 과정을 실습합니다. 이는 벡터 스토어 내의 문서 식별을 확인하는 데 유용합니다.		

08-079 FAISS DB에 삭제용 데이터 추가

실습번호	08-079	실습명	FAISS DB에 삭제용 데이터 추가
실습 코드	 `# 삭제용 데이터를 추가` `ids = db.add_texts(` `  ["삭제용 데이터를 추가합니다.", "2번째 삭제용 데이터입니다."],` `  metadatas=[{"source": "mydata.txt"}, {"source": "mydata.txt"}],` `  ids=["delete_doc1", "delete_doc2"],` `)`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에 삭제할 데이터를 추가합니다. add_texts 메서드를 사용하여 텍스트 리스트와 메타데이터, ID를 함께 추가합니다.		
실습 요약	FAISS DB에 삭제용 데이터를 추가하는 과정을 실습합니다. 이는 나중에 데이터를 삭제하기 위해 식별할 수 있도록 데이터를 준비하는 방법을 보여줍니다.		

08-080 FAISS DB에서 삭제할 ID 확인

실습번호	08-080	실습명	FAISS DB에서 삭제할 ID 확인
실습 코드	`# 삭제할 id 를 확인` `print(ids)`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에 추가한 삭제용 데이터의 ID를 확인합니다. print() 함수를 사용하여 최근에 추가한 데이터의 ID 리스트를 출력합니다.		
실습 요약	FAISS DB에서 삭제할 ID를 확인하는 과정을 실습합니다. 이는 삭제할 데이터의 식별을 확인하는 데 유용합니다.		

08-081 FAISS DB에서 ID로 데이터 삭제

실습번호	08-081	실습명	FAISS DB에서 ID로 데이터 삭제
실습 코드	`# id 로 삭제` `db.delete(ids)`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에서 지정한 ID를 사용하여 데이터를 삭제합니다. delete 메서드를 호출하여 삭제할 ID 리스트를 전달합니다.		
실습 요약	FAISS DB에서 특정 ID를 사용하여 데이터를 삭제하는 과정을 실습합니다. 이는 벡터 스토어에서 불필요한 데이터를 관리하는 방법을 보여줍니다.		

08-082 FAISS DB에서 삭제된 결과 확인

실습번호	08-082	실습명	FAISS DB에서 삭제된 결과 확인
실습 코드	`# 삭제된 결과를 출력` `db.index_to_docstore_id`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어에서 삭제된 결과를 확인합니다. index_to_docstore_id 속성을 호출하여 현재 문서의 ID 리스트를 출력하고, 삭제된 데이터가 반영된 결과를 확인합니다.		
실습 요약	FAISS DB에서 삭제된 결과를 확인하는 과정을 실습합니다. 이는 데이터 삭제가 제대로 이루어졌는지 검토하는 방법을 보여줍니다.		

08-083 FAISS DB를 로컬 Disk에 저장

실습번호	08-083	실습명	FAISS DB를 로컬 Disk에 저장
실습 코드	`# 로컬 Disk 에 저장` `db.save_local(folder_path="faiss_db", index_name="faiss_index")`		
실습 가이드	이 실습에서는 FAISS 벡터 스토어를 로컬 디스크에 저장합니다. save_local 메서드를 사용하여 지정한 폴더와 인덱스 이름으로 데이터를 저장합니다.		
실습 요약	FAISS DB를 로컬 디스크에 저장하는 과정을 실습합니다. 이는 데이터 지속성을 확보하고 나중에 사용할 수 있도록 저장하는 방법을 보여줍니다.		

08-084 로컬 Disk에서 FAISS DB 로드

실습번호	08-084	실습명	로컬 Disk에서 FAISS DB 로드
실습 코드	 `# 저장된 데이터를 로드` `loaded_db = FAISS.load_local(` `  folder_path="faiss_db",` `  index_name="faiss_index",` `  embeddings=embeddings,` `  allow_dangerous_deserialization=True,` `)`		
실습 가이드	이 실습에서는 로컬 디스크에 저장된 FAISS 벡터 스토어를 로드합니다. load_local 메서드를 사용하여 저장된 폴더와 인덱스 이름을 지정하고, 임베딩과 함께 로드합니다.		
실습 요약	로컬 디스크에서 FAISS DB를 로드하는 과정을 실습합니다. 이는 저장된 데이터를 불러와 사용할 수 있도록 하는 방법을 보여줍니다.		

08-085 로드된 FAISS DB에서 데이터 확인

실습번호	08-085	실습명	로드된 FAISS DB에서 데이터 확인
실습 코드	# 로드된 데이터를 확인 loaded_db.index_to_docstore_id		
실습 가이드	이 실습에서는 로드된 FAISS 벡터 스토어에서 문서 저장소 ID를 확인합니다. index_to_docstore_id 속성을 호출하여 로드된 데이터의 ID 리스트를 출력합니다.		
실습 요약	로드된 FAISS DB에서 데이터를 확인하는 과정을 실습합니다. 이는 데이터가 정상적으로 로드되었는지 검토하는 방법을 보여줍니다.		

08-086 새로운 FAISS 벡터 저장소 생성

실습번호	08-086	실습명	새로운 FAISS 벡터 저장소 생성
실습 코드	```# 새로운 FAISS 벡터 저장소 생성 db = FAISS.from_documents(documents=split_doc1 + split_doc2, embedding=OpenAIEmbeddings(model="text-embedding-3-small"))```		
실습 가이드	이 실습에서는 두 개의 분할된 문서 리스트(split_doc1과 split_doc2)를 사용하여 새로운 FAISS 벡터 저장소를 생성합니다. from_documents 메서드를 호출하여 문서와 임베딩을 기반으로 벡터 스토어를 초기화합니다.		
실습 요약	두 개의 문서 리스트를 결합하여 FAISS 벡터 저장소를 생성하는 과정을 실습합니다. 이는 여러 데이터 소스를 통합하여 벡터화를 수행하는 방법을 보여줍니다.		

08-087 FAISS DB를 검색기로 변환 및 검색 수행

실습번호	08-087	실습명	FAISS DB를 검색기로 변환 및 검색 수행
실습 코드	`# 검색기로 변환` `retriever = db.as_retriever()` `# 검색 수행` `retriever.invoke("Word2Vec 에 대하여 알려줘")`		
실습 가이드	이 실습에서는 FAISS 벡터 저장소를 검색기로 변환하고, 주어진 쿼리를 사용하여 검색을 수행합니다. as_retriever 메서드를 호출하여 검색 기능을 활성화하고, invoke 메서드를 사용하여 검색 쿼리를 입력합니다.		
실습 요약	FAISS DB를 검색기로 변환하고 검색을 수행하는 과정을 실습합니다. 이는 벡터 스토어를 활용하여 정보 검색을 가능하게 하는 방법을 보여줍니다.		

08-088 FAISS DB에서 MMR 검색 수행

실습번호	08-088	실습명	FAISS DB에서 MMR 검색 수행
실습 코드	 `# MMR 검색 수행` `# k: 최종 문서 수` `# lambda_mult: query와의 유사성과 문서 간의 다양성 조절, 1이면 유사성 only, 0이면 다양성 only` `# fetch_k: 후보 문서 수` `retriever = db.as_retriever(` `  search_type="mmr", search_kwargs={"k": 6, "lambda_mult": 0.25, "fetch_k": 10}` `)` `retriever.invoke("Word2Vec 에 대하여 알려줘")`		
실습 가이드	이 실습에서는 FAISS 벡터 저장소에서 MMR(Maximum Marginal Relevance) 검색을 수행합니다. 검색기를 생성할 때 search_type에 'mmr'을 지정하고, k, lambda_mult, fetch_k와 같은 검색 매개변수를 설정하여 유사성과 다양성을 조절합니다.		
실습 요약	 FAISS DB에서 MMR 검색을 수행하는 과정을 실습합니다. 이는 검색 결과의 다양성과 관련성을 동시에 고려하는 방법을 보여줍니다.		

08-089 FAISS DB에서 MMR 검색 수행 (상위 2개만 반환)

실습번호	08-089	실습명	FAISS DB에서 MMR 검색 수행 (상위 2개만 반환)
실습 코드	# MMR 검색 수행, 상위 2개만 반환 retriever = db.as_retriever(search_type="mmr", search_kwargs={"k": 2, "fetch_k": 10}) retriever.invoke("Word2Vec 에 대하여 알려줘")		
실습 가이드	이 실습에서는 FAISS 벡터 저장소에서 MMR(Maximum Marginal Relevance) 검색을 수행하여 상위 2개의 문서만 반환합니다. search_kwargs를 설정하여 반환할 문서 수를 k로 지정합니다.		
실습 요약	FAISS DB에서 MMR 검색을 수행하고 상위 2개의 결과만 반환하는 과정을 실습합니다. 이는 검색 결과의 개수를 조정하여 더 구체적인 정보를 얻는 방법을 보여줍니다.		

08-090 FAISS DB에서 임계 값 기반 검색 수행

실습번호	08-090	실습명	FAISS DB에서 임계 값 기반 검색 수행
실습 코드	 `# 임계 값 기반 검색 수행` `retriever = db.as_retriever(` `  search_type="similarity_score_threshold",` `search_kwargs={"score_threshold": 0.4}` `)` `retriever.invoke("Word2Vec 에 대하여 알려줘")`		
실습 가이드	이 실습에서는 FAISS 벡터 저장소에서 임계 값 기반 검색을 수행합니다. similarity_score_threshold 검색 유형을 사용하여 특정 유사도 점수 이상인 문서만 반환합니다. score_threshold 매개변수를 설정하여 필터링 기준을 지정합니다.		
실습 요약	 FAISS DB에서 임계 값 기반 검색을 수행하는 과정을 실습합니다. 이는 유사도 점수를 기반으로 문서를 필터링하여 검색 결과를 최적화하는 방법을 보여줍니다.		

08-091 FAISS DB에서 가장 유사한 문서 검색

실습번호	08-091	실습명	FAISS DB에서 가장 유사한 문서 검색
실습 코드	`# k=1 로 설정하여 가장 유사한 문서만 검색` `retriever = db.as_retriever(search_kwargs={"k": 1})` `retriever.invoke("Word2Vec 에 대하여 알려줘")`		
실습 가이드	이 실습에서는 FAISS 벡터 저장소에서 가장 유사한 문서 하나만 검색합니다. search_kwargs를 설정하여 k 값을 1로 지정하여 최상위 결과만 반환합니다.		
실습 요약	FAISS DB에서 가장 유사한 문서 하나를 검색하는 과정을 실습합니다. 이는 특정 쿼리에 대해 가장 관련성 높은 문서를 찾는 방법을 보여줍니다.		

08-092 JAEN에서 키워드 파일 다운로드

실습번호	08-092	실습명	JAEN에서 키워드 파일 다운로드
실습 코드	`from JAEN import download_file` `download_file('appendix-keywords') # appendix-keywords.txt`		
실습 가이드	이 실습에서는 JAEN 라이브러리를 사용하여 'appendix-keywords.txt' 파일을 다운로드합니다. download_file 함수를 호출하여 해당 파일을 다운로드하고, 이후에 사용할 수 있도록 준비합니다.		
실습 요약	JAEN 라이브러리를 사용하여 추가 키워드 파일을 다운로드하는 과정을 실습합니다. 이는 특정 주제에 대한 키워드를 쉽게 가져오는 방법을 보여줍니다.		

08-093 appendix-keywords.txt 파일 로드 및 FAISS 벡터 데이터베이스 생성

실습번호	08-093	실습명	appendix-keywords.txt 파일 로드 및 FAISS 벡터 데이터베이스 생성
실습 코드	from langchain_community.vectorstores import FAISS from langchain_openai import OpenAIEmbeddings from langchain_text_splitters import CharacterTextSplitter from langchain_community.document_loaders import TextLoader # TextLoader를 사용하여 파일을 로드합니다. loader = TextLoader("appendix-keywords.txt", encoding='utf-8') # 문서를 로드합니다. documents = loader.load() # 문자 기반으로 텍스트를 분할하는 CharacterTextSplitter를 생성합니다. 청크 크기는 300이고 청크 간 중복은 없습니다. text_splitter = CharacterTextSplitter(chunk_size=300, chunk_overlap=0) # 로드된 문서를 분할합니다. split_docs = text_splitter.split_documents(documents) # OpenAI 임베딩을 생성합니다. embeddings = OpenAIEmbeddings(model="text-embedding-3-small") # 분할된 텍스트와 임베딩을 사용하여 FAISS 벡터 데이터베이스를 생성합니다. db = FAISS.from_documents(split_docs, embeddings)		
실습 가이드	이 실습에서는 appendix-keywords.txt 파일을 로드하여 내용을 문서로 읽고, CharacterTextSplitter를 사용하여 문서를 지정된 크기로 분할합니다. 이후 OpenAIEmbeddings를 사용하여 문서의 임베딩을 생성하고, FAISS 벡터 데이터베이스를 생성합니다.		
실습 요약	텍스트 파일을 로드하고, 문서로 분할한 후 FAISS 벡터 데이터베이스를 생성하는 과정을 실습합니다. 이는 다양한 문서 데이터를 벡터화하여 검색 및 유사도 평가를 가능하게 하는 방법을 보여줍니다.		

08-094 FAISS DB를 검색기로 변환

실습번호	08-094	실습명	FAISS DB를 검색기로 변환
실습 코드	`# 데이터베이스를 검색기로 사용하기 위해 retriever 변수에 할당` `retriever = db.as_retriever()`		
실습 가이드	이 실습에서는 FAISS 벡터 데이터베이스를 검색기로 변환합니다. as_retriever 메서드를 호출하여 검색 기능을 활성화하고, 검색 작업에 사용할 retriever 변수를 설정합니다.		
실습 요약	FAISS DB를 검색기로 변환하는 과정을 실습합니다. 이는 저장된 데이터를 효과적으로 검색할 수 있는 방법을 보여줍니다.		

08-095 질문-답변 프롬프트 템플릿 생성

실습번호	08-095	실습명	질문-답변 프롬프트 템플릿 생성
실습 코드	from langchain_core.prompts import PromptTemplate prompt = PromptTemplate.from_template("""당신은 질문-답변(Question-Answering)을 수행하는 친절한 AI 어시스턴트입니다. 당신의 임무는 주어진 문맥(context) 에서 주어진 질문(question) 에 답하는 것입니다. 검색된 다음 문맥(context) 을 사용하여 질문(question) 에 답하세요. 만약, 주어진 문맥(context) 에서 답을 찾을 수 없다면, 답을 모른다면 주어진 정보에서 질문에 대한 정보를 찾을 수 없습니다 라고 답하세요. 한글로 답변해 주세요. 단, 기술적인 용어나 이름은 번역하지 않고 그대로 사용해 주세요. #Question: {question} #Context: {context} #Answer:"		
실습 가이드	이 실습에서는 질문-답변을 위한 프롬프트 템플릿을 생성합니다. 주어진 질문과 문맥에 따라 적절한 답변을 유도하는 형식을 설정합니다.		
실습 요약	질문-답변 프롬프트 템플릿을 생성하여 AI 어시스턴트가 사용자 질문에 대한 응답을 제공하는 방법을 실습합니다.		

08-096 RAG 체인 생성

실습번호	08-096	실습명	RAG 체인 생성
실습 코드	 `from langchain_core.runnables import RunnablePassthrough` `from langchain_core.output_parsers import StrOutputParser` `from langchain_openai import ChatOpenAI` `llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)` `# 체인을 생성합니다.` `rag_chain = (` `{"context": retriever, "question": RunnablePassthrough()}` `| prompt` `| llm` `| StrOutputParser()` `)`		
실습 가이드	이 실습에서는 질문-답변 체인을 생성하기 위해 RAG(Retrieval-Augmented Generation) 체인을 설정합니다. retriever와 prompt를 결합하여 질문과 관련된 문맥을 바탕으로 답변을 생성합니다.		
실습 요약	RAG 체인을 생성하여 검색된 문맥을 활용하여 질문에 대한 응답을 생성하는 과정을 실습합니다. 이는 정보 검색과 생성적 응답을 통합하는 방법을 보여줍니다.		

08-097 RunnablePassthrough 사용 예제

실습번호	08-097	실습명	RunnablePassthrough 사용 예제
실습 코드	`from langchain_core.runnables import RunnablePassthrough` `result = RunnablePassthrough().invoke({'name': 'AI Essential', 'Class': 1})` `print(result)` `result = RunnablePassthrough().invoke([10, 20, 30, 40, 50])` `print(result)`		
실습 가이드	이 실습에서는 RunnablePassthrough를 사용하여 다양한 형식의 입력 데이터를 그대로 통과시키는 예제를 보여줍니다. invoke 메서드를 호출하여 데이터의 원본 형태를 유지하며 출력합니다.		
실습 요약	RunnablePassthrough를 사용하여 입력 데이터를 그대로 반환하는 과정을 실습합니다. 이는 데이터 흐름을 유지하면서 추가 처리를 하지 않고 원본 데이터를 사용할 수 있도록 하는 방법을 보여줍니다.		

08-098 LLM과 StrOutputParser 결합 사용

실습번호	08-098	실습명	LLM과 StrOutputParser 결합 사용
실습 코드	`(llm ¦ StrOutputParser()).invoke("임베딩에 대해 알려줘")`		
실습 가이드	이 실습에서는 LLM(대형 언어 모델)과 StrOutputParser를 결합하여 주어진 질문에 대한 답변을 생성합니다. invoke 메서드를 호출하여 '임베딩에 대해 알려줘'라는 질문에 대한 응답을 출력합니다.		
실습 요약	LLM과 StrOutputParser를 결합하여 질문에 대한 응답을 생성하는 과정을 실습합니다. 이는 언어 모델의 출력을 간단하게 파싱하는 방법을 보여줍니다.		

08-099 RAG 체인 사용하여 질문에 대한 응답 생성

실습번호	08-099	실습명	RAG 체인 사용하여 질문에 대한 응답 생성
실습 코드	`rag_chain.invoke('임베딩에 대해 알려줘')`		
실습 가이드	이 실습에서는 RAG 체인을 사용하여 '임베딩에 대해 알려줘'라는 질문에 대한 응답을 생성합니다. invoke 메서드를 호출하여 검색된 문맥을 기반으로 언어 모델이 답변을 생성합니다.		
실습 요약	RAG 07체인을 통해 주어진 질문에 대한 응답을 생성하는 과정을 실습합니다. 이는 검색된 정보를 바탕으로 생성적 응답을 제공하는 방법을 보여줍니다.		

08-100 다양한 LangChain 모듈 임포트

실습번호	08-100	실습명	다양한 LangChain 모듈 임포트
실습 코드	`from langchain_community.tools.tavily_search import TavilySearchResults` `from langchain.text_splitter import RecursiveCharacterTextSplitter` `from langchain_community.vectorstores import FAISS` `from langchain_openai import OpenAIEmbeddings` `from langchain.document_loaders import PyPDFLoader` `from langchain.tools.retriever import create_retriever_tool` `from langchain_openai import ChatOpenAI` `from langchain import hub` `from langchain.agents import create_openai_functions_agent, AgentExecutor` `from langchain_community.chat_message_histories import ChatMessageHistory` `from langchain_core.runnables.history import RunnableWithMessageHistory`		
실습 가이드	이 실습에서는 LangChain의 다양한 모듈을 임포트합니다. 각 모듈은 텍스트 처리, 문서 로드, 임베딩 생성, 검색 도구, 에이전트 생성 등 여러 기능을 제공합니다.		
실습 요약	LangChain 모듈을 임포트하여 텍스트와 문서를 처리하고 검색 및 에이전트 기능을 사용할 수 있도록 준비하는 과정을 실습합니다.		

08-101 TavilySearchResults 인스턴스 생성

실습번호	08-101	실습명	TavilySearchResults 인스턴스 생성
실습 코드	`# TavilySearchResults 클래스의 인스턴스를 생성` `import os` `os.environ['TAVILY_API_KEY'] = ""` `search = TavilySearchResults(k=5, description='온디바이스 AI 기술 동향을 제외한 요청에 이 도구를 사용하세요.')`		
실습 가이드	이 실습에서는 TavilySearchResults 클래스의 인스턴스를 생성하여 검색 기능을 설정합니다. 환경 변수를 사용하여 TAVILY_API_KEY를 설정하고, k 값을 통해 반환할 결과의 수를 지정합니다.		
실습 요약	TavilySearchResults 인스턴스를 생성하여 검색 도구의 설정을 실습합니다. 이는 특정 검색 쿼리에 대한 응답을 효율적으로 처리하는 방법을 보여줍니다.		

08-102 TavilySearchResults 외부 검색 예시

실습번호	08-102	실습명	TavilySearchResults 외부 검색 예시
실습 코드	`# 외부 검색 예시` `search.invoke('삼성전자 비스포크에 대해서 알려줘')`		
실습 가이드	이 실습에서는 TavilySearchResults 인스턴스를 사용하여 외부 검색을 수행합니다. invoke 메서드를 호출하여 '삼성전자 비스포크에 대해서 알려줘'라는 쿼리로 검색을 수행하고, 결과를 반환받습니다.		
실습 요약	TavilySearchResults를 사용하여 특정 쿼리에 대한 외부 검색을 수행하는 과정을 실습합니다. 이는 외부 데이터 소스를 활용하여 정보를 검색하는 방법을 보여줍니다.		

08-103 TavilySearchResults 외부 검색 예시

실습번호	08-103	실습명	TavilySearchResults 외부 검색 예시
실습 코드	`# 외부 검색 예시` `search.invoke('2024년 리뷰가 있는 인계동 주변 맛집에 대해 알려줘')`		
실습 가이드	이 실습에서는 TavilySearchResults 인스턴스를 사용하여 외부 검색을 수행합니다. invoke 메서드를 호출하여 '2024년 리뷰가 있는 인계동 주변 맛집에 대해 알려줘' 라는 쿼리로 검색을 수행하고, 관련된 결과를 반환받습니다.		
실습 요약	TavilySearchResults를 활용하여 특정 쿼리에 대한 외부 검색을 수행하는 과정을 실습합니다. 이는 특정 조건을 만족하는 정보를 찾는 방법을 보여줍니다.		

08-104 PDF 파일 다운로드

실습번호	08-104	실습명	PDF 파일 다운로드
실습 코드	`from JAEN import download_file` `download_file('PDF') # 온디바이스 AI 기술동향 및 발전방향.pdf`		
실습 가이드	이 실습에서는 JAEN 라이브러리를 사용하여 지정된 PDF 파일을 다운로드합니다. download_file 함수를 호출하여 '온디바이스 AI 기술동향 및 발전방향.pdf' 파일을 로컬에 저장합니다.		
실습 요약	PDF 파일을 다운로드하는 과정을 실습합니다. 이는 외부 자료를 가져와 활용할 수 있도록 준비하는 방법을 보여줍니다.		

08-105 PDF 파일 로드 및 벡터 스토어 생성

실습번호	08-105	실습명	PDF 파일 로드 및 벡터 스토어 생성
실습 코드	# PDF 파일 로드 loader = PyPDFLoader("온디바이스 AI 기술동향 및 발전방향.pdf") # 텍스트 분할기를 사용하여 문서를 분할 text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100) # 문서를 로드하고 분할 split_docs = loader.load_and_split(text_splitter) # VectorStore를 생성 vector = FAISS.from_documents(split_docs, OpenAIEmbeddings(model="text-embedding-3-small"))		
실습 가이드	이 실습에서는 PyPDFLoader를 사용하여 PDF 파일을 로드하고, RecursiveCharacterTextSplitter를 사용하여 문서를 지정된 크기로 분할합니다. 이후 FAISS를 사용하여 분할된 문서로 벡터 스토어를 생성합니다.		
실습 요약	PDF 파일을 로드하고, 문서를 분할한 후 FAISS 벡터 스토어를 생성하는 과정을 실습합니다. 이는 PDF에서 정보를 효과적으로 추출하고 검색할 수 있는 방법을 보여줍니다.		

08-106 Retriever 생성 및 검색 도구 생성

실습번호	08-106	실습명	Retriever 생성 및 검색 도구 생성
실습 코드	 ``` # Retriever를 생성합니다. retriever = vector.as_retriever() # langchain 패키지의 tools 모듈에서 retriever 도구를 생성 retriever_tool = create_retriever_tool(retriever, name="pdf_search", # 도구에 대한 설명을 자세히 기입해야 합니다!!! description="온디바이스 AI 기술동향 및 발전방향을 PDF 문서에서 검색합니다. 온디바이스 AI의 전반적인 기술동향 또는 특정 국가의 온디바이스 AI 동향과 관련된 질문은 이 도구를 사용해야 합니다!",) retriever_tool ```		
실습 가이드	이 실습에서는 생성된 벡터 스토어를 기반으로 retriever를 생성합니다. 이후 langchain의 tools 모듈을 사용하여 PDF 문서에서 검색할 수 있는 도구를 생성합니다.		
실습 요약	Retriever를 생성하고 이를 사용하여 PDF 문서에서 정보를 검색하는 도구를 만드는 과정을 실습합니다. 이는 문서 검색 기능을 손쉽게 사용할 수 있도록 설정하는 방법을 보여줍니다.		

08-107 Tools 리스트에 검색 도구 추가

실습번호	08-107	실습명	Tools 리스트에 검색 도구 추가
실습 코드	`# tools 리스트에 search와 retriever_tool을 추가합니다.` `tools = [search, retriever_tool]`		
실습 가이드	이 실습에서는 기존에 생성한 검색 도구(search)와 PDF 문서 검색 도구(retriever_tool)를 하나의 리스트로 묶습니다. 이를 통해 두 개의 도구를 함께 사용할 수 있도록 설정합니다.		
실습 요약	search와 retriever_tool을 tools 리스트에 추가하여 다양한 검색 기능을 통합하는 과정을 실습합니다. 이는 여러 도구를 효율적으로 관리하고 사용할 수 있도록 하는 방법을 보여줍니다.		

08-108 LLM 모델 생성

실습번호	08-108	실습명	LLM 모델 생성
실습 코드	`# LLM 모델을 생성합니다.` `llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)`		
실습 가이드	이 실습에서는 ChatOpenAI 클래스를 사용하여 LLM(대형 언어 모델) 인스턴스를 생성합니다. 모델의 파라미터로는 'gpt-4o-mini'를 사용하고, temperature 값을 0으로 설정하여 더 예측 가능한 응답을 유도합니다.		
실습 요약	LLM 모델을 생성하여 후속 작업에서 사용할 준비를 하는 과정을 실습합니다. 이는 자연어 처리 작업을 수행하기 위한 기본 모델 설정을 보여줍니다.		

08-109 Hub에서 Prompt 가져오기

실습번호	08-109	실습명	Hub에서 Prompt 가져오기
실습 코드	# hub에서 prompt를 가져옴 prompt = hub.pull("hwchase17/openai-functions-agent")		
실습 가이드	이 실습에서는 LangChain Hub에서 지정된 prompt를 가져옵니다. hub.pull 메서드를 사용하여 미리 정의된 프롬프트를 쉽게 불러올 수 있습니다.		
실습 요약	Hub에서 prompt를 가져오는 과정을 실습합니다. 이는 사용자 정의 작업을 쉽게 수행하기 위해 사전 정의된 프롬프트를 활용하는 방법을 보여줍니다.		

08-110 OpenAI 함수 기반 에이전트 생성

실습번호	08-110	실습명	OpenAI 함수 기반 에이전트 생성
실습 코드	`# OpenAI 함수 기반 에이전트를 생성합니다.` `# llm, tools, prompt를 인자로 사용합니다.` `agents = create_openai_functions_agent(llm, tools, prompt)`		
실습 가이드	이 실습에서는 생성된 LLM, 검색 도구 리스트, 및 가져온 프롬프트를 사용하여 OpenAI 함수 기반 에이전트를 생성합니다. 이를 통해 다양한 질문에 대해 자동으로 응답할 수 있는 시스템을 구축합니다.		
실습 요약	OpenAI 함수 기반 에이전트를 생성하여 여러 도구와 LLM을 활용한 자동 응답 시스템을 구축하는 과정을 실습합니다. 이는 다양한 입력에 대해 적절한 행동을 수행하도록 에이전트를 설정하는 방법을 보여줍니다.		

08-111 AgentExecutor 설정

실습번호	08-111	실습명	AgentExecutor 설정
실습 코드	# AgentExecutor 클래스를 사용하여 agent와 tools를 설정하고, 상세한 로그를 출력하도록 verbose를 True로 설정합니다. agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)		
실습 가이드	이 실습에서는 AgentExecutor 클래스를 사용하여 생성한 에이전트와 도구를 설정합니다. verbose 모드를 True로 설정하여 에이전트 실행 과정에서 발생하는 로그를 자세히 출력합니다.		
실습 요약	AgentExecutor를 설정하여 질문-응답 시스템을 구축하는 방법을 실습합니다. 이는 에이전트의 동작을 모니터링할 수 있는 방법을 제공합니다.		

08-112 에이전트 실행 및 응답 출력

실습번호	08-112	실습명	에이전트 실행 및 응답 출력
실습 코드	`# AgentExecutor 클래스를 사용하여 agent와 tools를 설정하고, 상세한 로그를 출력하도록 verbose를 True로 설정합니다.` `response = agent_executor.invoke({` `  'input': '미국의 온디바이스 AI 정책에 대해 알려줘'` `})` `print(f'답변: {response["output"]}')`		
실습 가이드	이 실습에서는 AgentExecutor 클래스를 사용하여 생성한 에이전트와 도구를 설정합니다. verbose를 True로 설정하여 실행 과정에서 발생하는 로그를 상세히 출력하고, 주어진 질문에 대한 응답을 출력합니다.		
실습 요약	AgentExecutor를 사용하여 질문에 대한 응답을 출력하는 과정을 실습합니다. 이는 에이전트의 동작을 모니터링하고, 질문에 대한 정확한 답변을 제공하는 방법을 보여줍니다.		

08-113 에이전트 실행 및 응답 출력

실습번호	08-113	실습명	에이전트 실행 및 응답 출력
실습 코드	`response = agent_executor.invoke({` `  'input': '메타버스 기술 동향을 알려줘'` `})` `print(f'답변: {response["output"]}')`		
실습 가이드	이 실습에서는 설정한 AgentExecutor를 사용하여 주어진 질문에 대한 응답을 실행합니다. invoke 메서드를 호출하여 '메타버스 기술 동향을 알려줘'라는 질문에 대한 응답을 받고, 이를 출력합니다.		
실습 요약	에이전트를 실행하여 특정 질문에 대한 응답을 출력하는 과정을 실습합니다. 이는 질문에 대한 정보 검색과 응답 생성을 자동화하는 방법을 보여줍니다.		

08-114 RAGAS 평가를 위한 LLM 및 임베딩 모델 생성

실습번호	08-114	실습명	RAGAS 평가를 위한 LLM 및 임베딩 모델 생성
실습 코드	`# RAGAS 평가를 위한 LLM 및 임베딩 모델 생성` `from langchain_openai import ChatOpenAI` `from langchain_openai import OpenAIEmbeddings` `llm = ChatOpenAI(model='gpt-4o-mini', temperature=0)` `embeddings = OpenAIEmbeddings(model='text-embedding-3-small')`		
실습 가이드	이 실습에서는 RAGAS 평가를 위해 사용할 LLM(대형 언어 모델)과 OpenAI의 임베딩 모델을 생성합니다. LLM의 온도 값을 0으로 설정하여 더 예측 가능한 응답을 유도하고, 임베딩 모델은 'text-embedding-3-small'로 설정합니다.		
실습 요약	RAGAS 평가를 위한 LLM과 임베딩 모델을 생성하는 과정을 실습합니다. 이는 후속 작업에서 질문-응답 시스템을 강화하는 방법을 보여줍니다.		

08-115 데이터 샘플 생성

실습번호	08-115	실습명	데이터 샘플 생성
실습 코드	from datasets import Dataset data_samples = { 'question': ['Where and when was Einstein born?', 'Where and when was Einstein born?'], 'answer': ['Einstein was born in Germany on 14th March 1879.', 'Einstein was born in Germany on 20th March 1879.'], 'retrieved_contexts' : [['Albert Einstein (born 14 March 1879) was a German-born theoretical physicist, widely held to be one of the greatest and most influential scientists of all time'], ['Albert Einstein (born 14 March 1879) was a German-born theoretical physicist, widely held to be one of the greatest and most influential scientists of all time']], }		
실습 가이드	이 실습에서는 질문, 답변, 검색된 컨텍스트을 포함한 데이터 샘플을 생성합니다. 이를 통해 RAGAS 평가에 필요한 데이터 구조를 설정합니다.		
실습 요약	질문과 답변, 검색된 컨텍스트 및 정답을 포함하는 데이터 샘플을 생성하는 과정을 실습합니다. 이는 모델 평가 및 성능 분석에 필요한 데이터를 준비하는 방법을 보여줍니다.		

08-116 RAGAS 평가 수행 (Faithfulness)

실습번호	08-116	실습명	RAGAS 평가 수행 (Faithfulness)
실습 코드	`from ragas import evaluate` `from ragas.metrics import faithfulness` `dataset = Dataset.from_dict(data_samples)` `score = evaluate(dataset, metrics=[faithfulness], llm=llm, embeddings=embeddings)` `score.to_pandas()`		
실습 가이드	이 실습에서는 RAGAS를 사용하여 평가를 수행합니다. 생성한 데이터 샘플을 Dataset으로 변환하고, faithfulness 메트릭을 사용하여 모델의 응답을 평가합니다.		
실습 요약	RAGAS를 사용하여 데이터 샘플의 신뢰성을 평가하는 과정을 실습합니다. 이는 모델의 성능을 분석하는 중요한 단계를 보여줍니다.		

08-117 데이터 샘플 생성

실습번호	08-117	실습명	데이터 샘플 생성
실습 코드	 from datasets import Dataset data_samples = { 'question': ['Where is France and what is it’s capital?', 'Where is France and what is it’s capital?'], 'answer': ['France is in western Europe.', 'France is in western Europe and Paris is its capital.'], }		
실습 가이드	이 실습에서는 질문, 답변을 포함한 새로운 데이터 샘플을 생성합니다. 이를 통해 RAGAS 평가에 필요한 데이터 구조를 설정합니다.		
실습 요약	프랑스에 대한 질문과 답변을 포함하는 새로운 데이터 샘플을 생성하는 과정을 실습합니다. 이는 모델 평가 및 성능 분석에 필요한 데이터를 준비하는 방법을 보여줍니다.		

08-118 RAGAS 평가 수행 (Answer Relevancy)

실습번호	08-118	실습명	RAGAS 평가 수행 (Answer Relevancy)
실습 코드	`from copy import deepcopy` `from ragas import evaluate` `from ragas.metrics import answer_relevancy` `dataset = Dataset.from_dict(data_samples)` `score = evaluate(dataset, metrics=[answer_relevancy], llm=llm, embeddings=embeddings)` `score.to_pandas()`		
실습 가이드	이 실습에서는 RAGAS를 사용하여 주어진 데이터 샘플에 대한 평가를 수행합니다. answer_relevancy 메트릭을 사용하여 LLM의 응답을 평가하고, 그 결과를 pandas DataFrame으로 출력합니다.		
실습 요약	RAGAS를 사용하여 데이터 샘플의 응답 관련성과 정확성을 평가하는 과정을 실습합니다. 이는 모델의 성능을 분석하는 중요한 단계를 보여줍니다.		

08-119 데이터 샘플 생성

실습번호	08-119	실습명	데이터 샘플 생성
실습 코드	<pre>from datasets import Dataset data_samples = { 'question': ['Where is France and what is it’s capital?', 'Where is France and what is it’s capital?'], 'retrieved_contexts': [['France, in Western Europe, encompasses medieval cities, alpine villages and Mediterranean beaches. Paris, its capital, is famed for its fashion houses, classical art museums including the Louvre and monuments like the Eiffel Tower.'], ['France, in Western Europe, encompasses medieval cities, alpine villages and Mediterranean beaches. The country is also renowned for its wines and sophisticated cuisine. Lascaux’s ancient cave drawings, Lyon’s Roman theater and the vast Palace of Versailles attest to its rich history.']], 'ground_truth': ['France is in Western Europe and its capital is Paris.', 'France is in Western Europe and its capital is Paris.'] }</pre>		
실습 가이드	이 실습에서는 프랑스에 대한 질문, 검색된 컨텍스트, 정답을 포함한 데이터 샘플을 생성합니다. 이를 통해 RAGAS 평가에 필요한 데이터 구조를 설정합니다.		
실습 요약	프랑스의 위치와 수도에 관한 질문과 검색된 컨텍스트 및 정답을 포함하는 데이터 샘플을 생성하는 과정을 실습합니다. 이는 모델 평가 및 성능 분석에 필요한 데이터를 준비하는 방법을 보여줍니다.		

08-120 RAGAS 평가 수행 (Context Recall)

실습번호	08-120	실습명	RAGAS 평가 수행 (Context Recall)
실습 코드	`from ragas.metrics import context_recall` `dataset = Dataset.from_dict(data_samples)` `score = evaluate(dataset, metrics=[context_recall], llm=llm, embeddings=embeddings)` `score.to_pandas()`		
실습 가이드	이 실습에서는 RAGAS를 사용하여 주어진 데이터 샘플에 대한 평가를 수행합니다. context_recall 메트릭을 사용하여 LLM의 응답과 검색된 컨텍스트의 일치를 평가하고, 그 결과를 pandas DataFrame으로 출력합니다.		
실습 요약	RAGAS를 사용하여 데이터 샘플의 컨텍스트 회수를 평가하는 과정을 실습합니다. 이는 모델의 성능을 분석하는 중요한 단계를 보여줍니다.		

08-121 데이터 샘플 생성

실습번호	08-121	실습명	데이터 샘플 생성

실습 코드

```
from datasets import Dataset

data_samples = {
    'question': [
        'Where is France and what is it’s capital?',
        'Where is France and what is it’s capital?'
    ],
    'retrieved_contexts': [
        [
            "France, in Western Europe, encompasses medieval cities, alpine villages and Mediterranean beaches. Paris, its capital, is famed for its fashion houses, classical art museums including the Louvre and monuments like the Eiffel Tower",
            "The country is also renowned for its wines and sophisticated cuisine. Lascaux’s ancient cave drawings, Lyon’s Roman theater and the vast Palace of Versailles attest to its rich history."
        ],
        [
           "The country is also renowned for its wines and sophisticated cuisine. Lascaux’s ancient cave drawings, Lyon’s Roman theater and",
            "France, in Western Europe, encompasses medieval cities, alpine villages and Mediterranean beaches. Paris, its capital, is famed for its fashion houses, classical art museums including the Louvre and monuments like the Eiffel Tower"
        ]
    ],
    'ground_truth': [
        'France is in Western Europe and its capital is Paris.',
        'France is in Western Europe and its capital is Paris.'
    ]
}
```

실습 가이드	이 실습에서는 context recall을 평가하기 위한 질문, 검색된 컨텍스트, 정답을 포함한 데이터 샘플을 생성합니다. 이를 통해 RAGAS 평가에 필요한 데이터 구조를 설정합니다.
실습 요약	프랑스의 위치와 수도에 관한 질문 및 답변, 검색된 컨텍스트와 정답을 포함하는 데이터 샘플을 생성하는 과정을 실습합니다. 이는 모델 평가 및 성능 분석에 필요한 데이터를 준비하는 방법을 보여줍니다.

08-122 RAGAS 평가 수행 (Context Precision)

실습번호	08-122	실습명	RAGAS 평가 수행 (Context Precision)
실습 코드	 from ragas.metrics import context_precision dataset = Dataset.from_dict(data_samples) score = evaluate(dataset, metrics=[context_precision], llm=llm, embeddings=embeddings) score.to_pandas()		
실습 가이드	이 실습에서는 RAGAS를 사용하여 주어진 데이터 샘플에 대한 평가를 수행합니다. context_precision 메트릭을 사용하여 LLM의 응답과 검색된 컨텍스트의 관련성을 평가하고, 그 결과를 pandas DataFrame으로 출력합니다.		
실습 요약	RAGAS를 사용하여 데이터 샘플의 컨텍스트 정확성을 평가하는 과정을 실습합니다. 이는 모델의 성능을 분석하는 중요한 단계를 보여줍니다.		

09. LLM Fine-Tuning

09-001 Unsloth 설치

실습번호	09-001	실습명	Unsloth 설치
실습 코드	!pip install unsloth		
실습 가이드	이 실습에서는 Unsloth 패키지를 설치하여 딥러닝 모델 개발 환경을 설정합니다. 필요한 패키지를 확인하고 설치하여 개발을 시작할 준비를 합니다.		
실습 요약	Unsloth과 관련된 패키지를 설치하여 딥러닝 모델 개발 환경을 구축하는 과정을 실습합니다. 이를 통해 필요한 라이브러리를 활용할 수 있는 기반을 마련합니다.		

09-002 Google 드라이브 마운트 및 출력 디렉토리 설정

실습번호	09-002	실습명	Google 드라이브 마운트 및 출력 디렉토리 설정
실습 코드	 # from google.colab import drive # # Google 드라이브를 마운트합니다. # drive.mount('/content/drive') # # 결과물을 저장할 디렉터리 경로를 설정합니다. # output_dir = '/content/drive/MyDrive/outputs' output_dir = 'outputs'		
실습 가이드	이 실습에서는 Google 드라이브를 마운트하여 파일을 저장할 수 있는 환경을 설정합니다. 출력 결과를 저장할 디렉토리를 지정하여 나중에 결과를 쉽게 확인할 수 있도록 합니다.		
실습 요약	Google 드라이브를 마운트하고 결과물을 저장할 디렉토리를 설정하는 과정을 실습합니다. 이를 통해 파일을 효율적으로 관리할 수 있습니다.		

09-003 파인튜닝을 위한 모델 로드

실습번호	09-003	실습명	파인튜닝을 위한 모델 로드

실습 코드

```
# 파인튜닝을 위한 모델 로드
from unsloth import FastLanguageModel
import torch


# 모델 설정
max_seq_length = 2048  # 최대 시퀀스 길이
dtype = None  # 기본 dtype
load_in_4bit = True  # 4bit 양자화된 모델 사용 여부


# 4bit 모델 리스트
fourbit_models = [
    "unsloth/Meta-Llama-3.1-8B-bnb-4bit",      # Llama-3.1 모델 (15조 토큰 학습, 2배 더 빠름)
    "unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit",
    "unsloth/Meta-Llama-3.1-70B-bnb-4bit",
    "unsloth/Meta-Llama-3.1-405B-bnb-4bit",    # 405B 모델도 4bit로 업로드 됨
    "unsloth/Mistral-Nemo-Base-2407-bnb-4bit", # 새로운 Mistral 12b 모델 (2배 더 빠름)
    "unsloth/Mistral-Nemo-Instruct-2407-bnb-4bit",
    "unsloth/mistral-7b-v0.3-bnb-4bit",        # Mistral v3 (2배 더 빠름)
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/Phi-3.5-mini-instruct",           # Phi-3.5 모델 (2배 더 빠름)
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/gemma-2-2b-bnb-4bit",
    "unsloth/gemma-2-9b-bnb-4bit",
    "unsloth/gemma-2-27b-bnb-4bit",            # Gemma 모델 (2배 더 빠름)
```

	"unsloth/Llama-3.2-1B-bnb-4bit", # NEW! Llama 3.2 models "unsloth/Llama-3.2-1B-Instruct-bnb-4bit", "unsloth/Llama-3.2-3B-bnb-4bit", "unsloth/Llama-3.2-3B-Instruct-bnb-4bit",] # 더 많은 모델은 https://huggingface.co/unsloth 에서 확인 가능 # FastLanguageModel을 사용해 모델과 토크나이저 로드 model, tokenizer = FastLanguageModel.from_pretrained(model_name = fourbit_models[10], # 4bit 모델 중 11번째 모델 선택 (T4에서 훈련 가능) max_seq_length = max_seq_length, # 최대 시퀀스 길이 설정 dtype = dtype, # dtype 설정 load_in_4bit = load_in_4bit, # 4bit 모델 로드)
실습 가이드	이 실습에서는 파인튜닝된 모델을 로드하여 사용할 수 있는 환경을 설정합니다. 4bit 모델을 로드하여 메모리 효율성을 높이고, 모델 및 토크나이저를 준비하여 이후의 작업에 활용할 수 있도록 합니다.
실습 요약	4bit로 양자화된 파인튜닝된 모델과 토크나이저를 로드하여 딥러닝 환경을 설정하는 과정을 실습합니다. 이를 통해 메모리 사용량을 줄이면서도 모델의 성능을 유지할 수 있습니다.

09-004 학습 전 추론 결과 확인

실습번호	09-004	실습명	학습 전 추론 결과 확인
실습 코드	# 학습 전 추론 결과 확인 prompt = """Below is an instruction that describes a task. Write a response that appropriately completes the request. Answer in Korean. ### Instruction: {} ### Response: {} """ FastLanguageModel.for_inference(model) # 추론 모드 설정 # 입력 query query = prompt.format("삼성전자 캠퍼스에 대해서 알려주세요", "") # 입력 데이터 토큰화 input = tokenizer(query, return_tensors="pt").to('cuda') # GPU 필수 # 추론 output = model.generate(**input, max_new_tokens=512, use_cache=True, repetition_penalty=2.0) # 출력 토큰을 문자으로 변환 print(tokenizer.decode(output[0], skip_special_tokens=True))		
실습 가이드	이 실습에서는 사전 학습된 FastLanguageModel을 사용하여 특정 입력에 대한 추론 결과를 확인합니다. 주어진 프롬프트를 모델에 입력하여 생성된 응답을 출력합니다.		
실습 요약	FastLanguageModel을 사용하여 특정 질문에 대한 추론을 수행하고, 그 결과를 출력하는 과정을 실습합니다. 이를 통해 모델의 응답 생성 능력을 평가할 수 있습니다.		

09-005 모델에 Adapter 추가

실습번호	09-005	실습명	모델에 Adapter 추가
실습 코드	# 불러온 모델에 adapter 추가 model = FastLanguageModel.get_peft_model(model, r = 16, # Rank, 양의 정수 target_modules = ["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj",], # 대상 파라미터 모듈 lora_alpha = 16, # LoRA의 scaling factor lora_dropout = 0, # 0~1, 일반적으로 0이 최적 bias = "none", # bias 적용 여부, 'none', 'all', 'lora_only', 일반적으로 'none'이 최적 use_gradient_checkpointing = "unsloth", # 'unsloth' or True, 'unsloth' 권장 random_state = 1234, # 랜덤 상태를 고정하여 실험 재현성을 보장 use_rslora = False, # Rank Stabilized LoRA 적용 여부 loftq_config = None, # LoftQ 사용 여부)		
실습 가이드	이 실습에서는 FastLanguageModel에 LoRA(로우 랭크 어댑터)를 추가하여 모델의 성능을 향상시킵니다. 모델의 특정 모듈에 대해 어댑터 설정을 적용하여 메모리 효율성을 높이고 학습 속도를 개선합니다.		
실습 요약	모델에 LoRA 어댑터를 추가하여 성능 향상 및 메모리 효율성을 높이는 과정을 실습합니다. 이를 통해 어댑터를 활용한 파인튜닝 방법을 익힙니다.		

09-006 LLM JSON 파일 다운로드

실습번호	09-006	실습명	LLM JSON 파일 다운로드
실습 코드	`from JAEN import download_file` `download_file('llm') # llm.json`		
실습 가이드	이 실습에서는 JAEN 패키지를 사용하여 사전 정의된 LLM 구성 파일을 다운로드 합니다. 이후 해당 JSON 파일을 사용하여 LLM 설정을 조정할 수 있습니다.		
실습 요약	JAEN 패키지를 통해 LLM 구성 파일을 다운로드하는 과정을 실습합니다. 다운로드한 JSON 파일은 모델 설정에 사용될 수 있습니다.		

09-007 프롬프트 포매팅 함수 정의

실습번호	09-007	실습명	프롬프트 포매팅 함수 정의

실습 코드

```
# 명령어 형식
prompt = """Below is an instruction that describes a task. Write a response
that appropriately completes the request.
### Instruction:
{}

### Response:
{}
"""

EOS_TOKEN = tokenizer.eos_token  # eos 토큰 지정

def prompt_formatting(examples):
    instructions = examples["instruction"]
    outputs      = examples["output"]
    texts = []

    for instruction, output in zip(instructions, outputs):
        text = prompt.format(instruction, output) + EOS_TOKEN # eos을 완성
된 문장 뒤에 추가
        texts.append(text)
    return {"text" : texts} # 사전 형태로 반환
```

실습 가이드

이 실습에서는 주어진 예제의 지침(instruction)과 응답(output)을 기반으로 프롬프트를 포매팅하는 함수를 정의합니다. 이 함수를 사용하여 모델의 입력 형식을 통일할 수 있습니다.

실습 요약

프롬프트 포매팅 함수를 정의하여 명령어와 응답을 연결하는 과정을 실습합니다. 이를 통해 모델 입력을 일관되게 관리할 수 있습니다.

09-008 LLM JSON 데이터셋 로드 및 가공

실습번호	09-008	실습명	LLM JSON 데이터셋 로드 및 가공
실습 코드	 `from datasets import load_dataset` `dataset = load_dataset("json", data_files="llm.json", split='train')` `# 데이터셋 가공` `# prompt_formatting을 데이터 적용` `dataset = dataset.map(prompt_formatting, batched=True)`		
실습 가이드	 이 실습에서는 JSON 형식의 LLM 데이터를 로드한 후, 정의한 `prompt_formatting` 함수를 사용하여 데이터셋을 가공합니다. 이를 통해 각 데이터 포인트를 모델의 입력 형식에 맞게 변환합니다.		
실습 요약	 LLM JSON 데이터를 로드하고 가공하여 모델 입력 형식에 맞게 변환하는 과정을 실습합니다. 이를 통해 데이터셋의 일관성을 유지할 수 있습니다.		

09-009 파인튜닝을 위한 트레이너 설정

실습번호	09-009	실습명	파인튜닝을 위한 트레이너 설정

실습 코드

```
from trl import SFTTrainer  # Supervised 파인튜닝 트레이너

from transformers import TrainingArguments  # 트랜스포머 모델 훈련을 위한 설정 관리

from unsloth import is_bfloat16_supported  # 시스템이 bfloat16(브레인 플로트) 형식을 지원하는지 확인하는 함수


# 파인튜닝을 위한 트레이너 설정

trainer = SFTTrainer(

    model = model,

    tokenizer = tokenizer,

    train_dataset = dataset,  # 학습 데이터

    dataset_text_field = "text", # 학습 데이터의 key (사전 key)

    max_seq_length = max_seq_length,  # 최대 토큰 수

    dataset_num_proc = 2,  # 데이터세트 전처리 프로세스 수

    packing = False,  # ConstantLengthDataset을 이용한 sequence 묶음 기능 (학습 효율을 향상시키기 위한 방법)

    args = TrainingArguments(

        per_device_train_batch_size = 8,  # 장치에 전달할 small 배치 크기

        gradient_accumulation_steps = 4,  # grandient 누적 스텝

        warmup_steps = 5,  # warmup step 설정 (초기 학습률을 낮게 설정)

        num_train_epochs = 50,  # 학습 에폭

        learning_rate = 2e-4,  # 학습률 설"정

        fp16 = not is_bfloat16_supported(),  # FP16 사용 여부 (지원되지 않으면 False)

        bf16 = is_bfloat16_supported(),  # bfloat16 사용 여부 (지원되면 True)

        logging_steps = 1,  # 몇 스텝마다 로깅할지

        optim = "adamw_8bit",  # 최적화 방법 (8비트 AdamW 사용)
```

	weight_decay = 0.01, # 가중치 감소 설정 (정칙화) lr_scheduler_type = "linear", # 학습률 스케줄러 타입 (선형) seed = 1024, output_dir = output_dir, # 훈련 결과 저장 디렉토리 report_to = "none", # 로그 저장 연동))
실습 가이드	이 실습에서는 SFTTrainer를 사용하여 모델 파인튜닝을 위한 트레이너를 설정합니다. 다양한 훈련 매개변수를 설정하여 모델이 주어진 데이터셋에 적합하도록 학습할 수 있도록 합니다.
실습 요약	모델 파인튜닝을 위한 트레이너 설정 과정을 실습합니다. 이를 통해 다양한 학습 파라미터를 설정하여 효과적인 모델 학습을 진행할 수 있습니다.

09-010 CUDA 장치 상태 및 메모리 획득

실습번호	09-010	실습명	CUDA 장치 상태 및 메모리 획득
실습 코드	# CUDA 장치 상태 획득 gpu_stats = torch.cuda.get_device_properties(0) # 첫 번째 GPU 장치의 속성 정보를 가져옴 # 사용 중인 메모리 획득 start_gpu_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3) # 사용 중인 GPU 메모리(GB 단위) 계산 # GPU 최대 메모리 max_memory = round(gpu_stats.total_memory / 1024 / 1024 / 1024, 3) # GPU의 총 메모리 용량(GB 단위) 계산 # GPU 정보 및 메모리 상태 출력 print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.") # GPU 이름과 최대 메모리 출력 print(f"{start_gpu_memory} GB of memory reserved.") # 예약된 메모리 용량 출력		
실습 가이드	이 실습에서는 CUDA를 사용하여 GPU 장치의 상태와 메모리 사용량을 확인하는 방법을 보여줍니다. 이를 통해 모델 학습 시 GPU 자원 관리가 가능합니다.		
실습 요약	CUDA 장치 상태를 확인하고 GPU의 메모리 사용량을 측정하는 과정을 실습합니다. 이를 통해 GPU 자원의 효율적인 활용이 가능해집니다.		

09-011 모델 파인튜닝 수행

실습번호	09-011	실습명	모델 파인튜닝 수행
실습 코드	 import os # 파인튜닝 수행 trainer_stats = trainer.train()		
실습 가이드	이 실습에서는 설정한 트레이너를 사용하여 모델 파인튜닝을 수행합니다. 모델이 주어진 데이터셋을 기반으로 학습하게 됩니다.		
실습 요약	모델 파인튜닝을 수행하여 학습 결과를 얻는 과정을 실습합니다. 이를 통해 모델의 성능을 개선할 수 있습니다.		

09-012 학습 후 추론 결과 확인

실습번호	09-012	실습명	학습 후 추론 결과 확인
실습 코드	FastLanguageModel.for_inference(model) # 추론 모드 설정 # 입력 query query = prompt.format("삼성전자 캠퍼스에 대해서 알려주세요.", "") # 입력 데이터 토큰화 input = tokenizer(query, return_tensors="pt").to('cuda') # GPU 필수 # 추론 output = model.generate(**input, max_new_tokens=512, use_cache=True, repetition_penalty=2.0) # 출력 토큰을 문자으로 변환 print(tokenizer.decode(output[0], skip_special_tokens=True))		
실습 가이드	이 실습에서는 설정한 모델을 사용하여 입력 쿼리에 대한 추론을 수행합니다. 모델이 주어진 입력에 대해 적절한 응답을 생성하는 과정을 확인할 수 있습니다.		
실습 요약	입력 쿼리를 기반으로 모델이 생성한 출력을 확인하는 과정을 실습합니다. 이를 통해 모델의 응답 품질을 평가할 수 있습니다.		

09-013 학습된 모델 저장

실습번호	09-013	실습명	학습된 모델 저장
실습 코드	 `# 학습 모델 저장` `quantization_methods = ["f16", "q8_0", "q4_k_m"]  # 사용할 양자화 방법들을 정의` `output_dir = f"model_{quantization_methods[2]}"` `# GGUF 포맷으로 양자화된 모델을 저장` `model.save_pretrained_gguf(` `    output_dir,` `    tokenizer,  # 토크나이저 정보도 함께 저장` `    quantization_method=quantization_methods[2]  # 양자화 방법 지정` `)`		
실습 가이드	이 실습에서는 학습된 모델을 지정된 양자화 방법을 사용하여 GGUF 포맷으로 저장하는 방법을 보여줍니다. 저장된 모델은 추후 불러와 사용할 수 있습니다.		
실습 요약	학습된 모델을 다양한 양자화 방법으로 저장하여 모델의 활용도를 높이는 과정을 실습합니다.		