[3주차] 머신러닝을 위한 첫걸음: 파이썬 기초와 데이터 표현 🐍

안녕하세요! 3주차 수업에 오신 것을 환영합니다.

지난 2주간 우리는 머신러닝의 놀라운 가능성을 눈으로 확인했습니다.

앞으로 4주간은, 이 멋진 모델들을 우리 손으로 직접 만들기 위한 가장 중요한 도구,

'파이썬(Python)' 프로그래밍 언어를 배우는 시간을 갖겠습니다.

오늘은 그 첫 시간으로, 왜 머신러닝 세계에서 파이썬이 필수인지 알아보고,

'데이터'를 파이썬으로 어떻게 표현하고 저장하는지를 배워보겠습니다.

🌐 모듈 1: 왜 머신러닝에 파이썬을 사용할까?

- **쉽고 직관적인 문법**:
  - 다른 언어에 비해 배우기 쉽고 코드가 간결
  - 복잡한 수학적 이론을 구현하는 데 집중
- **강력하고 풍부한 라이브러리**:
  - **NumPy**: 과학 계산 및 다차원 배열을 위한 필수 라이브러리
  - **Pandas**: 표(테이블) 형태의 데이터를 다루는 데 최적화된 라이브러리
  - **Matplotlib**: 데이터 시각화를 위한 라이브러리
  - **Scikit-Learn**: 고전적인 머신러닝 모델 라이브러리
  - **TensorFlow, PyTorch**: 딥러닝 모델을 위한 대표적인 라이브러리
- **강력한 커뮤니티**:
  - 전 세계 수많은 개발자와 연구자들이 파이썬을 사용
  - 문제 해결에 필요한 자료나 도움을 얻기 쉽다

📦 모듈 2: 데이터의 가장 작은 단위, 변수(Variables)

- **변수란?:** 데이터(값)를 저장하기 위한 '이름표'가 붙은 상자
- **머신러닝에서의 변수:**
  - 데이터셋의 한 칸에 들어가는 값을 의미
  - 예를 들어, 한 학생의 '공부 시간'이나 '시험 점수'
- **자료형 (Data Types)**: 변수에 담는 데이터의 형태
  - `int` : **정수** (e.g., 92, -5, 0)
  - `float` : **실수** (소수점이 있는 숫자, e.g., 10.5, 3.14)
  - `str` : **문자열** (따옴표로 감싼 글자들, e.g., "Kim", "Pass")
  - `bool` : **불리언** (참/거짓 두 가지만 표현, e.g., `True`, `False`)

In [ ]:
```
# '한 학생'의 데이터를 변수에 저장해봅시다.

# 1. '공부 시간'  이라는 이름표(변수)에 10.5 라는 실수(float) 값을 저장
study_hours = 10.5

# 2. '시험 점수' 라는 이름표(변수)에 92 라는 정수(int) 값을 저장
exam_score = 92

# 저장된 값 확인
print("공부 시간:", study_hours)
print("시험 점수:", exam_score)

# type() 함수로 변수의 자료형 확인
print("\n'study_hours' 변수의 자료형:", type(study_hours))
print("'exam_score' 변수의 자료형:", type(exam_score))
```

```
공부 시간: 10.5
시험 점수: 92

'study_hours' 변수의 자료형: <class 'float'>
'exam_score' 변수의 자료형: <class 'int'>
```

In [ ]:
```
# 학생의 추가 정보 저장

# '학생 이름' 이라는 변수에 "Kim" 이라는 문자열(str) 값을 저장
student_name = "Kim"

# '합격 여부' 라는 변수에 True 라는 불리언(bool) 값을 저장
is_passed = True

# 저장된 값 확인
print("학생 이름:", student_name)
print("합격 여부:", is_passed)

# 자료형 확인
print("\n'student_name' 변수의 자료형:", type(student_name))
print("'is_passed' 변수의 자료형:", type(is_passed))
```

```
학생 이름: Kim
합격 여부: True

'student_name' 변수의 자료형: <class 'str'>
'is_passed' 변수의 자료형: <class 'bool'>
```

In [ ]:
```
# 변수는 계산에도 활용할 수 있습니다.
# 예를 들어,  이 학생이 5점짜리 보너스 문제를 맞혔다고 가정해봅시다.

print("기존 점수:", exam_score)

# 기존 점수에 5를 더해서 새로운 점수로 업데이트
final_score = exam_score + 5

print("최종 점수:", final_score)
```

```
기존 점수: 92
최종 점수: 97
```

🦦 모듈 3: 데이터셋을 담는 그릇, 자료구조

여러 학생의 데이터를 한 번에 다루려면 어떻게 해야 할까요?

여러 개의 값을 하나의 변수에 담을 수 있는 더 큰 그릇, **자료구조**가 필요합니다.

1. 리스트 (List)

- **정의**:
  - 여러 개의 값을 **순서대로** 저장하는 자료구조
  - 대괄호 `[]` 로 구분
- **머신러닝에서의 리스트**: 데이터를 순서대로 묶어 처리할 때 유용
- **예시**: 모든 학생의 '공부 시간' / '시험점수' 리스트

In [ ]:
```
# 여러 학생의 '공부 시간'과 '시험 점수' 데이터를 리스트로 만들어 봅시다.
study_hours_list = [2.0, 4.5, 6.0, 8.5, 10.0]
exam_scores_list = [65, 78, 85, 95, 98]

print("공부 시간 리스트:", study_hours_list)
print("시험 점수 리스트:", exam_scores_list)

# 리스트는 '순서'가 있기 때문에, 인덱스(index)로 특정 값에 접근할 수 있습니다.
# 파이썬은 0부터 숫자를 셉니다!
print("\n첫 번째 학생의 공부 시간:", study_hours_list[0])
print("세 번째 학생의 시험 점수:", exam_scores_list[2])

# len() 함수로 리스트에 몇 개의 데이터가 있는지 확인할 수 있습니다.
print("\n총 학생 수:", len(study_hours_list))
```

```
공부 시간 리스트: [2.0, 4.5, 6.0, 8.5, 10.0]
시험 점수 리스트: [65, 78, 85, 95, 98]

첫 번째 학생의 공부 시간: 2.0
세 번째 학생의 시험 점수: 85

총 학생 수: 5
```

In [ ]:
```
# 새로운 학생의 데이터가 추가되었다고 가정해봅시다.
# .append()를 이용해 리스트의 맨 뒤에 값을 추가할 수 있습니다.

print("추가 전 공부 시간 리스트:", study_hours_list)

study_hours_list.append(12.0)
exam_scores_list.append(100)

print("추가 후 공부 시간 리스트:", study_hours_list)
print("추가 후 총 학생 수:", len(study_hours_list))
```

```
추가 전 공부 시간 리스트: [2.0, 4.5, 6.0, 8.5, 10.0]
추가 후 공부 시간 리스트: [2.0, 4.5, 6.0, 8.5, 10.0, 12.0]
추가 후 총 학생 수: 6
```

2. 딕셔너리 (Dictionary)

- **정의**:
  - '키(Key)'와 '값(Value)'을 하나의 쌍으로 묶어서 저장하는 자료구조
  - 중괄호 `{}` 로 구분
  - 순서가 없음
- **머신러닝에서의 딕셔너리**: 데이터간 관계에 따라 묶어 처리할 때 유용
- **예시**: 한 학생의 '이름', '공부 시간', '시험 점수', '합격 여부'를 모두 담고 있는 정보 묶음

In [ ]:
```
# 한 학생의 모든 정보를 딕셔너리로 만들어 봅시다.
student_A_info = {
    'name': 'Park',
    'study_hours': 8.5,
    'exam_score': 95,
    'is_passed': True
}

print("학생 A의 정보:", student_A_info)

# 딕셔너리는 순서(index) 대신 '키(Key)'를 이용해 값에 접근합니다.
print("\n학생 A의 이름:", student_A_info['name'])
print("학생 A의 시험 점수:", student_A_info['exam_score'])
```

```
학생 A의 정보: {'name': 'Park', 'study_hours': 8.5, 'exam_score': 95, 'is_passed': True}

학생 A의 이름: Park
학생 A의 시험 점수: 95
```

최종 형태: 리스트와 딕셔너리의 조합

- 머신러닝에서 다루는 대부분의 데이터는 **'딕셔너리의 리스트'** 또는 **'리스트의 딕셔너리'** 형태로 표현
- 특히 **"각 학생의 정보(딕셔너리)들을 하나의 리스트로 묶는"** 방식은 가장 흔하게 사용
  - 마치 엑셀의 표와 같은 형태

In [ ]:
```
# 여러 학생의 정보(딕셔너리)를 하나의 리스트로 묶어 전체 데이터셋을 표현
full_dataset = [
    { 'name': 'Kim', 'study_hours': 2.0, 'exam_score': 65, 'is_passed': True },
    { 'name': 'Lee', 'study_hours': 4.5, 'exam_score': 78, 'is_passed': True },
    { 'name': 'Park', 'study_hours': 8.5, 'exam_score': 95, 'is_passed': True },
    { 'name': 'Choi', 'study_hours': 1.5, 'exam_score': 55, 'is_passed': False }
]

# 첫 번째 학생('Kim')의 정보에 접근하기
first_student = full_dataset[0]
print("첫 번째 학생의 정보:", first_student)

# 첫 번째 학생의 '시험 점수'에만 접근하기
first_student_score = full_dataset[0]['exam_score']
print("첫 번째 학생의 시험 점수:", first_student_score)
```

```
첫 번째 학생의 정보: {'name': 'Kim', 'study_hours': 2.0, 'exam_score': 65, 'is_passed': True}
첫 번째 학생의 시험 점수: 65
```