SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

데이터분석 이론 및 실습

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

9월10일 실습 및 과제 알고리즘

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

실습 알고리즘

실습1. 각 변수의 타입 출력

• 1-1. 각 변수의 타입 출력

1st. temperature, humidity, pollutants, location, sensor_readings, status 변수에 값을 할당하여 초기화합니다

2nd. 각 변수의 타입을 확인하기 위해 type() 함수를 사용합니다

3rd. print() 함수를 사용하여 각 변수의 타입을 출력합니다

실습1. 각 변수의 타입 출력

- 1-2. 리스트 pollutants 조작

1st. 리스트에 새로운 요소를 추가하기 위해 append("CO2") 메서드를 사용하여 "CO2"를 리스트의 마지막에 추가합니다

2nd. 리스트의 첫 번째 요소를 삭제하기 위해 del 키워드를 사용하여 pollutants[0]를 제거합니다

3rd. 리스트의 길이를 확인하기 위해 len() 함수를 사용하고, 이를 출력합니다

실습1. 각 변수의 타입 출력

- 1-3. 튜플 sensor_readings에서 숫자 20

1st. index() 메서드를 사용하여 튜플 sensor_readings에서 숫자 20의 인덱스를 찾습니다

2nd. 찾은 인덱스 값을 출력합니다

• 2-1. SensorData 클래스 정의 및 사용

1st. SensorData 클래스를 정의하고, __init__ 메서드를 사용하여 위치(location), 오염물질(pollutant), 측정값(value)을 속성으로 초기화합니다

2nd. 클래스에 print_info() 메서드를 추가하여 객체의 속성을 출력할 수 있도록 합니다

3rd. sensor1 객체를 생성하여 초기값으로 "Downtown", "PM2.5", 35.4를 할당합니다

4th. sensor1 객체의 print_info() 메서드를 호출하여 정보를 출력합니다.

5th. sensor2 객체를 생성하여 초기값으로 "Suburb", "O3", 20.1을 할당합니다.
6th. sensor2 객체의 print_info() 메서드를 호출하여 정보를 출력합니다.

• 2-2. SensorData 클래스에 alarm() 메서드 추가

1st. SensorData 클래스에 alarm() 메서드를 추가하여 주어진 임계값(threshold)과 측정값을 비교하도록 합니다

2nd. 측정값이 임계값을 초과할 경우 경고 메시지를 출력하고, 그렇지 않을 경우 안전 메시지를 출력합니다

3rd. sensor1 객체에 대해 alarm() 메서드를 호출하여 임계값 25.0과 비교합니다

- 2-3. SensorData 클래스에 update_value() 메서드 추가

1st. SensorData 클래스에 update_value() 메서드를 추가하여 새로운 측정값(new_value)을 받아서 기존의 값을 업데이트합니다

2nd. 업데이트된 측정값을 출력하여 변경 사실을 확인할 수 있도록 합니다

3rd. sensor1 객체를 생성한 후, update_value() 메서드를 호출하여 new_value를 40.0으로 설정합니다

4th. sensor1 객체의 print_info() 메서드를 호출하여 업데이트된 정보를 출력합니다

실습3. 연산자 오버로딩을 활용한 벡터 데이터 연산

• 3-1. Vector2D 클래스 정의 및 연산자 오버로딩

1st. Vector2D 클래스를 정의하고, 생성자 __init__을 사용하여 x와 y 좌표를 초기화합니다

2nd. 벡터 간 덧셈을 지원하기 위해 __add__ 메서드를 정의하여 두 벡터의 합을 반환하도록 합니다

3rd. 벡터 간 뺄셈을 지원하기 위해 __sub__ 메서드를 정의하여 두 벡터의 차를 반환하도록 합니다

4th. 스칼라 곱을 지원하기 위해 __mul__ 메서드를 정의하여 벡터의 x, y 값을 스칼라와 곱한 결과를 반환하도록 합니다

5th. sensor1_vector를 (3, 4)로, sensor2_vector를 (1, 2)로 초기화하고 연산자 오버로딩을 활용하여 덧셈, 뺄셈, 스칼라 곱 연산을 수행한 결과를 출력합니다

실습3. 연산자 오버로딩을 활용한 벡터 데이터 연산

• 3-2. 벡터 길이 계산 메서드 추가

1st. Vector2D 클래스에 length() 메서드를 추가하여 벡터의 길이를 계산하도록 합니다

2nd. length() 메서드는 피타고라스 정리(math.sqrt(self.x**2 + self.y**2))를 사용하여 벡터의 크기를 반환합니다

3rd. sensor1_vector를 (3, 4)로 초기화한 후, length() 메서드를 호출하여 벡터의 길이를 출력합니다

실습3. 연산자 오버로딩을 활용한 벡터 데이터 연산

• 3-3. 벡터 내적 계산 메서드 추가

1st. Vector2D 클래스에 내적 계산을 지원하는 __mul__ 메서드를 수정하여 벡터와 벡터 간 내적 연산도 지원하도록 합니다

2nd. 두 벡터의 내적은 각 성분의 곱을 더한 값(self.x * other.x + self.y * other.y)으로 계산됩니다

3rd. sensor1_vector를 (3, 4)로, sensor2_vector를 (1, 2)로 초기화하고, 두 벡터 간의 내적을 계산하여 결과를 출력합니다

실습4. 상속을 통한 데이터 카테고리 분류

• 4-1. 기본 클래스와 상속 클래스 정의

1st. PollutionSource 클래스를 정의하고, 생성자 __init__을 사용하여 오염원 이름(name)과 배출량(emission)을 초기화합니다

2nd. describe() 메서드를 정의하여 오염원의 정보를 출력하도록 합니다

3rd. PollutionSource 클래스를 상속받아 Traffic, Industry, Natural 클래스를 정의합니다

- 4-2. move() 메서드 오버라이딩

1st. PollutionSource 클래스에 move() 메서드를 추가하여 기본적으로 "오염원은 고정되어 있습니다."라는 메시지를 출력합니다

2nd. Traffic, Industry, Natural 클래스에서 move() 메서드를 오버라이딩하여 각각 고유한 이동 방식을 설명하는 메시지를 출력하도록 합니다

3rd. traffic, industry, natural 객체의 move() 메서드를 호출하여 출력을 확인합니다

• 4-3. 상위 클래스 메서드 호출

1st. 자식 클래스에서 부모 클래스의 describe() 메서드를 호출하여 자식 클래스의 설명에 부모 클래스의 설명을 추가합니다

2nd. super()를 사용하여 부모 클래스의 describe() 메서드를 호출합니다

3rd. 부모와 자식 클래스의 설명이 명확히 구분되도록 출력 형식을 지정합니다

4th. traffic, industry, natural 객체의 describe() 메서드를 호출

실습5. 파일 입출력을 통한 데이터 기록 및 분석

• 5-1. 단어 수 계산 후 파일에 저장

1st. "data.txt" 파일을 읽기 모드로 열어 각 줄을 lines 리스트에 저장합니다

2nd. 각 줄에서 단어 수를 계산하기 위해 split() 메서드를 사용하여 단어를 분리한 후, len() 함수를 사용해 단어 수를 구합니다

3rd. 각 줄의 단어 수를 [줄 번호]번째 줄: [단어 수]개의 단어 형식으로 print() 함수를 사용해 출력합니다

4th. 계산된 단어 수를 "word_count.txt" 파일에 기록합니다

실습5. 파일 입출력을 통한 데이터 기록 및 분석

• 5-2. 가장 긴 단어 찾기 및 파일에 기록

1st. "data.txt" 파일을 읽고 각 줄을 lines 리스트에 저장합니다

2nd. 각 줄의 단어를 분리하고, 가장 긴 단어와 그 단어가 위치한 줄 번호를 추적합니다

3rd. 가장 긴 단어와 그 단어가 위치한 줄 번호를 print() 함수를 사용해 출력합니다

4th. 가장 긴 단어와 그 위치를 "longest_word.txt" 파일에 기록합니다

• 5-3. 파일 내용을 거꾸로 읽어 파일에 저장

1st. "data.txt" 파일을 읽고 각 줄을 lines 리스트에 저장합니다

2nd. 각 줄의 순서뿐만 아니라 줄 안의 문자 순서도 거꾸로 뒤집어 reversed_lines 리스트에 저장합니다

3rd. 거꾸로 변환된 내용을 print() 함수를 사용해 출력합니다

4th. 변환된 내용을 "reversed.txt" 파일에 저장합니다

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

과제 알고리즘

과제1. 사용자 정의 예외 클래스 생성 및 사용

• 1-1. 사용자 정의 예외 클래스 생성

1st. InvalidValueError라는 사용자 정의 예외 클래스를 정의합니다

2nd. InvalidValueError 클래스는 Python의 기본 예외 클래스 ValueError를 상속받아 정의됩니다

3rd. 생성자 __init__ 메서드를 정의하여, 예외가 발생할 때 전달된 값과 함께 예외 메시지를 초기화합니다

과제1. 사용자 정의 예외 클래스 생성 및 사용

• 1-2. 예외 발생 함수 작성

1st. check_positive_value(value) 함수를 정의합니다

2nd. 함수에서 전달된 값이 0보다 작을 경우 InvalidValueError를 발생시킵니다

3rd. 값이 0 이상이면 해당 값이 유효하다는 메시지를 출력합니다

과제1. 사용자 정의 예외 클래스 생성 및 사용

• 1-3. 예외 발생 확인 및 처리

1st. try-except 블록을 사용하여 check_positive_value(value) 함수를 호출하고, 예외가 발생하는지 확인합니다

2nd. 예외가 발생하면 except 블록에서 InvalidValueError 예외를 잡아 해당 예외 메시지를 출력합니다

3rd. 정상적인 값이 전달되면 예외가 발생하지 않고, 부적절한 값이 전달되면 예외가 발생하여 메시지가 출력됩니다

• 2-1. 패키지 및 모듈 작성

1st. calculator라는 패키지를 생성합니다. 패키지는 폴더로 구성되며, __init__.py 파일을 포함하여 패키지로 인식되도록 합니다

2nd. calculator 폴더 내에 각각의 연산을 담당하는 네 개의 모듈 파일(add.py, subtract.py, multiply.py, divide.py)을 작성합니다

3rd. 각 모듈 파일에 두 숫자를 입력받아 해당 연산을 수행하는 함수를 정의합니다:

- add.py: 두 숫자를 더하는 add(x, y) 함수 작성.
- subtract.py: 두 숫자를 빼는 subtract(x, y) 함수 작성.
- multiply.py: 두 숫자를 곱하는 multiply(x, y) 함수 작성.
- divide.py: 두 숫자를 나누는 divide(x, y) 함수 작성. 단, 0으로 나누는 경우 ValueError를 발생시킵니다.

• 2-2. main.py 파일 작성 및 패키지 사용

1st. main.py 파일을 작성하여 calculator 패키지 내의 모듈을 임포트합니다

2nd. from calculator.add import add 등과 같이 각 모듈에서 함수를 임포트합니다

3rd. 각 임포트한 함수를 사용하여 각 연산을 수행하고, 결과를 출력합니다:

- add(10, 5)는 15를 반환
- subtract(10, 5)는 5를 반환
- multiply(10, 5)는 50을 반환
- divide(10, 5)는 2를 반환

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미