- 아래에 주어진 요구사항을 고려하여 Python 코드를 작성하여 제출하세요.
- 코드 작성은 Google Colab에서 주피터 노트북 파일 형태로 작성
- 파일 이름은 “학번_이름.ipynb”로 할 것

1. 아래와 같은 세부 요구사항을 고려해서 주어진 점수 리스트에서 최고점과 최저점을 제외한 나머지 점수의 평균을 구하는 Python 함수를 작성하고 동작을 확인하시오.

■ 구현 요구사항

- 함수명: calc_avg_ex_minmax(scores)
  - ✓ scores 인자: 각 요소 값이 정수 또는 실수 값으로 이루어진 Numpy 배열
- 가정
  - ✓ 입력 배열에서 최고점과 최저점은 각각 하나씩만 존재
  - ✓ scores에 포함된 요소 값들은 모두 0보다 큰 정수 또는 실수
- 동작
  - ✓ 최고점 1개와 최저점 1개를 제외한 나머지 점수들의 평균값을 반환
  - ✓ 최고점, 최저점 선택과 평균 계산은 모두 numpy 배열 연산 함수를 활용하여 구현
  - ✓ 평균 계산을 위해서는 함수 내에서 최고점과 최저점을 제외한 새로운 numpy 배열을 만들되, list comprehension을 활용할 것.
  - ✓ 반환값은 float 형태의 평균값임.

■ 예외 처리

- 함수 내에서 인자로 전달된 scores 배열의 원소 개수가 3개 미만일 때 예외 처리
  - ✓ Python에 익숙한 학생은 ValueError exception으로 에러 처리
  - ✓ Python이 익숙하지 않은 학생은 -1을 반환하여 호출한 곳에서 에러 처리하도록 함.

■ 예시 실행 출력 결과

```
입력= [100  95  75  80  50  90]
최고점과 최저점을 하나씩만 제외한 평균: 85.0
```

```
입력= [10 20]
입력 점수 개수 부족
```

2. Python Class를 활용하여 간단한 자동차 동작 시뮬레이션 프로그램 작성

■ 구현 요구사항

- Car 클래스 생성
  - ✓ 자동차는 속성으로 '속도 (speed)'와 '위치 (position)' 정보를 가지도록 함.
  - ✓ 초기 속도와 위치는 각각 0으로 설정
- 아래 기능을 포함한 method 구현
  - ✓ accelearate(): 가속양 (amount)를 인자로 받아 속도를 증가시킴.
  - ✓ brake(): 감속양 (amount)를 인자로 받아 속도를 감소시키되, 최소 속도는 0이어야 함.
  - ✓ move(): 현재 속도만큼 자동차 위치를 업데이트하고 출력
- Car 클래스 타입 객체 car를 생성한 후, 무한반복하면서 사용자 입력을 받아 자동차를 조작
  - ✓ 'a' 키 입력 후 1만큼 가속하고 현재 속도 출력 (accelerate( ) method 사용)
  - ✓ 'b' 키 입력 후 1만큼 감속하고 현재 속도 출력 (brake( ) method 사용)
  - ✓ 'a' 키 또는 'b' 키가 입력되면 속도 조정 후 move()
  - ✓ 'q' 키 입력시 무한루프 탈출
  - ✓ 'a', 'b', 'q' 외의 키가 입력되면 에러 메시지를 출력하고 현재 속도에 따라 move() 후 다시 사용자 입력 대기

■ 힌트

- 사용자 입력은 Python 내장 함수인 input() 함수 사용

```
[예시 코드]
        user_input = input("입력: ")
        print("입력값: ", user_input)
[예시 실행결과]
        입력: Hello world!
        입력값: Hello world!
```

■ 예시 실행 출력 결과

```
Press 'a' to accelerate, 'b' to brake, 'q' to quit: a
Accelerating: New speed = 1
Moving: New position = 1
Press 'a' to accelerate, 'b' to brake, 'q' to quit: a
Accelerating: New speed = 2
Moving: New position = 3
Press 'a' to accelerate, 'b' to brake, 'q' to quit: b
Braking: New speed = 1
Moving: New position = 4
Press 'a' to accelerate, 'b' to brake, 'q' to quit: r
Invalid input. Use 'a', 'b', or 'q'.
Moving: New position = 5
Press 'a' to accelerate, 'b' to brake, 'q' to quit: q
Exiting...
```

3. 자율주행을 하는 이동체에서 센서 측정값을 이용해서 전방의 장애물을 감지하는 예시 코드를 작성하고 실행해 보자.

■ 가정

- 장애물 감지 센서는 매 1초당 10개의 데이터를 탐지해서 장애물 유무 판단 알고리즘에 입력으로 전달
- 전원이 켜지면 센서는 무한히 측정값을 전달
- 데이터는 장애물까지의 거리 (m)를 나타내며, 0~10 (m) 사이의 값을 전송한다.
- 장애물 회피를 위해 센서 측정값 10개 중 하나라도 1 이하의 값이 들어오면 장애물이 있다고 판단

■ 구현 요구사항

- 실제 센서가 없으므로 np.random 모듈의 uniform 함수를 이용해서 매번 랜덤한 데이터를 발생시켜 센서 측정값을 흉내내도록 함.
  - ✓ np.random.uniform(low, high, size): low~high 범위에 있는 임의의 값 size 개를 생성해서 numpy array로 반환
- 전달된 센서값들을 검사하여 1 m 이하의 측정값이 들어오면 “Obstacle Detected!” 메시지를 출력하고, 그렇지 않으면 “No obstacle detected.” 메시지 출력
  - ✓ 반복문 없이 numpy 함수를 이용하여 구현하도록 함.
- 1초 대기 후 반복
  - ✓ 특정 시간 동안 대기 동작은 time 모듈의 sleep() 함수 활용

■ 예시 실행 출력 결과

```
Sensor Data: [0.58122311 7.66388484 7.91008528 0.54312503 9.68440738 7.40200084
 5.42648062 7.61585406 2.66015664 4.45897909]
Obstacle Detected!
Sensor Data: [6.60523878 8.37464685 1.35502827 4.07999489 5.07712297 2.53674133
 7.04952192 4.82233399 0.50545966 6.39133157]
Obstacle Detected!
Sensor Data: [4.89711029 8.42595702 7.69003336 1.85202039 3.24235471 8.54275117
 6.93230929 6.33554764 3.84508888 8.70618103]
No obstacle
Sensor Data: [8.98681147 7.9824454  0.5797094  5.62903197 4.98161617 0.22146836
 6.10388325 4.93747919 9.3710777  5.94381693]
Obstacle Detected!
Sensor Data: [5.62463841 8.21842605 2.58853297 3.0462069  8.06884672 8.65182421
 6.33927113 2.15990805 5.19921949 8.84795216]
No obstacle
...
```

4. 3차원 좌표공간에서 이동하는 물체의 자세는 보통 Roll, Pitch, Yaw 세 개의 회전 값으로 표현된다. ROS에서는 이 값이 기본적으로 radian 단위로 주어지지만, degree 단위가 더 직관적일 때가 많다. Radian 값을 degree로 변환하는 방법에는 다음 두 가지가 있다:

(1) 직접 변환: 기본 수식을 이용해 radian을 degree로 변환 (degree = radian × 180 / π)

(2) NumPy 내장 함수 사용: numpy.degrees() 함수 활용

■ 구현 요구사항

- rad_to_degree_manual(rads) 함수 구현
  - ✓ 주어진 radian 값 배열을 수식을 사용하여 degree로 변환
- rad_to_degree_builtin(rads) 함수 구현
  - ✓ 같은 변환을 NumPy의 np.degrees() 함수를 사용하여 수행
- 랜덤한 Roll, Pitch, Yaw 값 생성 후 각 함수의 인자로 전달
  - ✓ numpy.random.uniform(-π, π, size=(3,))을 사용하여 -π ~ π 범위에서 랜덤한 3개의 radian 값을 생성
  - ✓ 생성된 numpy 배열을 각 함수의 인자로 전달
- 변환 결과 출력
  - ✓ 랜덤하게 생성된 각 radian 값과 두 개의 함수를 이용해 각각 변환된 degree 값을 출력
  - ✓ 출력시 모든 값은 소숫점 아래 4자리까지의 실수로 출력
- 두 변환 방식의 실행 시간 비교
  - ✓ time.time()을 사용하여 두 함수의 실행 시간을 측정하고 비교
  - ✓ 실행시간을 출력할 때, 시간 단위는 microsecond (us) 단위로 보정
  - ✓ 실행시간을 비교한 후, 어떤 방식이 더 빠른지 측정 결과와 간단한 토의를 코멘트로 코드에 추가

■ 예시 실행 출력 결과

```
Roll (radian): -0.9848, Roll (degree): -56.4233
Pitch (radian): 0.9665, Pitch (degree): 55.3735
Yaw (radian): 0.5538, Yaw (degree): 31.7305
Execution time using arithmetic operation:  ... usec

Roll (radian): -0.9848, Roll (degree): -56.4233
Pitch (radian): 0.9665, Pitch (degree): 55.3735
Yaw (radian): 0.5538, Yaw (degree): 31.7305
Execution time using np.degrees():  ... usec
```