Assginment 1: Black Box Testing (10 pts)

This assignment is to practice black box testing techniques. It has two questions (5pts each). Each question gives you a faulty black box program. You cannot read its source. Each program contains 5 bugs. You earn 0.5 pt for every bug your test cases detect. A test case detects a bug when its expected output, taken from the specification, differs from what the program returns. A test counts only if its expected output matches the specification.

You can play with the black box program and try your test cases here: https://sh1k4ku-python-playground.static.hf.space/index.html

General rules:

- Expected outputs must be computed from the specification, not from the faulty program.
- A bug is counted once even if several tests expose it.
- Categorical strings are case sensitive.
- Type checking is strict: integer fields must receive Python int values, float fields must receive Python float values, and categorical fields must receive an exact string from the listed domain.

Question 1: Boundary Value Testing (5 pts)

Introduction

In this exercise you will craft a set of boundary focused test cases for the **UNSW Room Allocator**. The allocator decides whether a group of students can be seated in the booked rooms, based on four input parameters: `students`, `rooms`, `capacity`, `mode`. The supplied program is faulty and contains 5 bugs, and you need to design boundary value test cases to detect them.

Input Specification

Input	Type / Range
students	Integer 1 to 300
rooms	Integer 1 to 10
capacity	Integer 5 to 50, seats per room
mode	Categorical {Lecture, Tutorial, Exam}

An out of range or wrong type input returns `INVALID`.

The **usable seats** per room equal capacity. The **total seats** equal rooms times usable seats per room. The result is `SEATED` when students is less than or equal to **total seats**, otherwise it is `OVERFLOW`.

Your Task

Q1.1 (2.5 pts) Identify which input fields are suitable for Boundary Value Testing, and for each suitable field list the boundary values you would test for **robust boundary values testing**. Put your answer in `Q1.txt`.

Q1.2 (2.5 pts) You may test any combination of inputs you want. The program contains 5 bugs and each bug you detect is worth 0.5 pt. Write a set of Robust Boundary Value test cases and submit them in `Q1_test.py` as a list named `TESTS`, one case per line, in the order students, rooms, capacity, mode, each paired with the expected output from the specification.

Hint: In our lecture, we assume that the input parameters are indepdendent and only consider the boundaries per input parameter. That's the ideal cases for "textbook" scenarios. In reality, inputs often have dependencies. Consider boundaries caused by the combinations of input fields.

Example Submission Format

Q1.txt (format only, not the answer)

```
inputA[boundaryValue1, boundaryValue2, ...], inputB[boundaryValue1, boundaryValue2, ...]
```

Q1_test.py (format only, not the answer, just copy/paste from the playground)

```
TESTS = [
    ((50,  2, 30, "Lecture"), "SEATED"),
    ((200, 5, 25, "Lecture"), "OVERFLOW"),
    ((50,  2, 30, "Online"),  "INVALID"),
]
```

Files to Submit

Item	File name	What to include
Suitable Inputs	Q1.txt	Fields suitable for Robust Boundary Value Testing, with the boundary values for each
Test case List	Q1_test.py	Your RBVT test cases, one case per line in the order students, rooms, capacity, mode, each paired with its expected output

---

Question 2: Equivalence Class Testing (5 pts)

Introduction

In this exercise you will explore the equivalence classes of the **UNSW Casual Workload Approval System**. The function determines whether a proposed booking is `APPROVED`, `REQUIRES_APPROVAL`, `DENIED`, or `ERROR`, based on four independent input parameters: `role`, `fte`, `weekly_hours`, `period`. The supplied program is faulty and contains 5 bugs, and your need to design equivalence class test cases to detect them.

Input Specification

#	Input Name	Type / Domain
1	role	Categorical {Tutor, Demonstrator, Marker, Coordinator}
2	fte	Float, greater than 0.0 and at most 1.0
3	weekly_hours	Integer 1 to 60
4	period	Categorical {Teaching, Break}

- An out of range or wrong type input returns `ERROR`.

Each role has a full time weekly base (hours). `Tutor` is 13, `Demonstrator` is 16, `Marker` is 9, and `Coordinator` is 21. The **cap** equals floor of fte times base (`floor(fte * times_base)`).

- In the `Break` period, `weekly_hours` <= cap returns `APPROVED`, otherwise `DENIED`.
- In the `Teaching` period, `weekly_hours` <= cap returns `APPROVED`, `weekly_hours` > cap and <= 2 times cap returns `REQUIRES_APPROVAL`, and `weekly_hours` > 2 times cap returns `DENIED`.

Your Task

Q2.1 (2.5 pts) Identify the valid and invalid equivalence classes from the input specification table above. Use braces for categorical values. Submit your answer in Q2.txt.

Q2.2 (2.5 pts) You may test any combination of inputs you want. The program contains 5 bugs and each bug you detect is worth 0.5 pt. Generate a Equivalence Class test suite and submit it in Q2_test.py as a list named `TESTS`, one input tuple per line, in the order `role`, `fte`, `weekly_hours`, `period`, each paired with the expected output according to the specification.

Hint: In our lecture, we assume that the input parameters are indepdendent. That's the ideal cases for "textbook" scenarios. In reality, inputs often have dependencies. Test the special program behaviours caused by combinations equivalence classes, not just one value inside each.

Example Submission Format

Q2.txt (format only, not the answer, "[" and "]" mean inclusive, "(" and ")" mean exclusive)

```
inputA: valid {categoryA, categoryB, ...}  invalid {any other string}
inputB: valid [0.0, 1) invalid [-inf, 0.0)  invalid [1, inf]
inputC: valid [1, 100] invalid [-inf, 1)  invalid (100, inf]
```

Q2_test.py

```
TESTS = [
    (("Tutor", 1.0, 5, "Teaching"),        "APPROVED"),
    (("Marker", 0.5, 3, "Break"),          "APPROVED"),
    (("Coordinator", 1.0, 50, "Teaching"), "DENIED"),
]
```

Files to Submit

Item	File name	What to include
Equivalence Classes	Q2.txt	All valid and invalid equivalence classes
Test case List	Q2_test.py	Your test cases, one input tuple per line in the order role, fte, weekly_hours, period, each paired with its expected output

---

Put all 4 files into one folder (don't use subfolders). Compress the folder into a single zip and submit that file on webcms using `give`.