## Exercise 06: Random Virtual Sequence

***Objective:*** *Introduction to the randomization and constraint concepts using the PyVSC library.*

***Task:*** *Create a new virtual sequence that randomizes the number of sequence items generated by the agent.*

Use the provided template 0.1 as the starting point to create the virtual sequence while following the next steps:

1. Inside the folder `<ROOT>/sat_filter/src/tb/vseqs`, create a file called `sat_filter_rnd_itr_seq`. Use the file `sat_filter_default_seq` from the same directory as reference.
2. The new virtual sequence must be extended from the `sat_filter_tb_base_seq` (file located in the testbench directory: `<ROOT>/sat_filter/src/tb`).
3. The new virtual sequence must contain a random variable that shall be used to control the number of items sent to the *uVC*. This variable must be called `itr_nbr`. Read more in the *PyVSC* Data Types documentation.
4. Define a constraint for the `itr_nbr` variable. This constraint should limit the number of sequences generated to:
   - a positive number,
   - below a user defined thresold.

   Read more in the *PyVSC* Constraints documentation.
5. In the virtual sequence `body` two coroutines must be implemented: one for the producer agent and a second for the consumer agent of the *sSDT*. In this loop, over the `itr_nbr` value, the coroutines must be launched in parallel.

```python
@vsc.randobj
class sat_filter_rnd_itr_seq(sat_filter_tb_base_seq):

    def __init__(self, name="sat_filter_rnd_itr_seq"):

        super().__init__(name)

        # TODO: Add missing sequences declaration
        # TODO: Add itr_nbr parameter

    async def body(self):

        # Launch sequences
        await super().body()

        # TODO: Fill in with the missing steps

    @vsc.constraint
    def itr_nbr_c(self):

        # TODO: Add itr_nbr constraint here
```

Listing 0.1: Template for the `sat_filter_rnd_itr_seq` sequence.

For the execution of the virtual sequence, a test case must be created. Use the provided template 0.2 for the following steps:

1. Create a new test case, named `test_sat_filter_rnd_itr`, inside `<ROOT>/sat_filter/src/tb/tests`.
2. This test case must be extended from `sat_filter_tb_base_test` provided inside `<ROOT>/sat_filter/src/tb`.
3. The sequence must be added in the test using the factory override method in the `start_of_simulation_phase`.
4. The `run_phase` must also be implemented accordingly: first randomizing the virtual sequence and then starting the virtual sequence on the virtual sequencer.
5. Improve the test case by adding an inline constraint to control the randomization from the test case. e.g.: Always generate at least 10 sequences.

```python
@pyuvm.test(timeout_time=_TIMEOUT_TIME, timeout_unit=_TIMEOUT_UNIT)
class test_sat_filter_rnd_itr(sat_filter_tb_base_test):

    def __init__(self, name="test_sat_filter_rnd_itr", parent=None):

        super().__init__(name, parent)

    def start_of_simulation_phase(self):

        super().start_of_simulation_phase()

        # TODO: Complete (...)
        # uvm_factory().set_type_override_by_type(...)

    async def run_phase(self):

        self.raise_objection()
        await super().run_phase()

        # Randomize sequence
        self.virt_sequence.randomize()

        # Run sequence
        await self.virt_sequence.start(self.tb_env.virtual_sequencer)

        self.drop_objection()
```

Listing 0.2: Template for the `test_sat_filter_rnd_itr` test case.