A mask-based set for integers

- Total points: 2.5 (0.5 per task)
- Number of submissions allowed: 5

Introduction

We want to implement a class called `IntSet` that behaves mostly like a usual `set`, but is specialized to store non-negative integers, between `0` and some `m` (excluded).

Internally, we keep a “mask”, a list of fixed length `m` filled with boolean values. Each index represents an integer, and each boolean value denotes whether the integer is in the set or not.

For example, say `m=5`. The mask `[True, False, False, True, True]` represents the set `{0, 3, 4}`.

To add an element, we set the corresponding entry in the mask to `True`, and to remove it we set it to `False`.

We also keep track of the number of elements in the set, in an attribute called `n`.

The set is initialized with a fixed mask size, and it’s initially empty (all elements `False`, `n==0`). The size of the mask never changes.

Tasks

Each task is worth 0.5 points. They are all tested independently.

The test file `"runtest.py"` contains code that tests all tasks sequentially. You will keep getting errors until you complete all the tasks. But after each successful task, it prints a “test ok” message, so that you know you can proceed to the next task.

The initial file has 3 methods already implemented and you don’t need to modify them: the constructor, an internal `_check_arg` method and the `add` method.

1. Write a method such that the `len` built-in function can be applied to `IntSet` objects, returning the number of elements in the set.
2. Write a `remove` method. It should work exactly as the reverse of the (already written) `add` method, with only one difference: when we `add` an element that is already present, nothing happens, but when we `remove` an element that is not present we should get a `KeyError` instead. The error object should have the missing element as its argument, just like it happens for a regular set.
3. Write a method that checks whether a given integer `x` is contained in an `IntSet`. It should then be possible to write `x in s` or `x not in s` when `s` is an `IntSet`. As an additional requirement, the implementation must not check the validity of its argument directly. It must use the internal `_check_arg` method, and let an error be raised if the element is not an integer. But if the argument is an invalid (out-of-bounds) integer, it should catch the error and just return `False` instead.
4. Write a method called `pop_rand` that removes and returns a random element of the `IntSet`. If the set is empty, the method should return a `KeyError`. Otherwise, the method should work as follows: it should keep extracting random values in the range of the `IntSet`, and as soon as it finds one that is present in the set it should use the `remove` method to remove it, and then return it.
5. Write a method that allows to print the `IntSet` like a `set` with the same contents would be printed, for example `'{0, 2, 4}'`. One exception, for simplicity: an empty `IntSet` should be printed as `'{}'`. The method should allow not only `print` but also `repr` and `str` to work.

General rules

In all methods, you can use whatever constructs you want. Only the behavior matters, as long as you follow the indications.

The class methods must not print anything. When the file `"ex.py"` is loaded nothing should be printed. Do not write anything else in the file, just the class(es) and, if needed, `import` statements.

You can do tests by typing `python runtest.py` in the terminal, or pressing the `RUN` button. Edit the file `"runtest.py"` as much as you want when you test. That file is for your own use and it’s not used when grading.

Remember that copies of the original files are always present in the `"startercode"` directory, in case you need them.

As for examples of how everything should work, refer to the test script.