

CS2006 Python Project 1 - Classes and Iterators in Python

Matriculation Numbers: 160001362, 160014817, 160013384 (Group 14)

06/03/2018

Contents

1	Summary of Functionality	3
1.1	Basic Specification:	3
1.2	Additional Requirements:	3
1.2.1	Easy	3
1.2.2	Medium	4
1.2.3	Hard	4
1.3	Further Features:	5
2	Design and Implementation	5
3	Evidence of Testing	5
4	Known Problems	5
5	Problems Overcome	5
6	Summary of Provenance	5
6.1	Code Implemented by the Group	5
6.2	Code Modified From the Provided Example Code	5
6.3	Code Sourced From Elsewhere	5
7	Conclusion	5

1 Summary of Functionality

This practical specified the development of a mathematical system to explore discrete mathematical structures using the programming language Python.

The provided README file gives detailed instructions on what the solution can do and how to run each command.

The following functionality has been implemented:

1.1 Basic Specification:

All requirements from the basic specification have been implemented. They are as follows:

- An implementation of the twisted integers data structure which supports addition and multiplication given by the following rules:
 - $a \oplus b = (a + b) \bmod n$
 - $a \otimes b = (a + b + a \cdot b) \bmod n$
- Exceptions that are used to check that user input is valid.
- Unit tests to ensure the correctness of the solution.

1.2 Additional Requirements:

From the suggested additional requirements, all of the easy, medium and hard requirements have been implemented:

1.2.1 Easy

- **Easy Requirement 1** - The function 'mulEqualToOne(n)' has been developed in the checker.py file which calculates for a given n all elements $x \in \mathbb{Z}_n$ such that $x \otimes x = 1$, where $1 \in \mathbb{Z}_n$
- **Easy Requirement 2** - For a given n functions (in checker.py) have been developed to check whether the following properties hold for all $x, y, z \in \mathbb{Z}_n$ (each of which returns a boolean signifying whether the properties hold):
 - $x \oplus y = y \oplus x$ - 'isCommutativeAdd(n)'
 - $x \otimes y = y \otimes x$ - 'isCommutativeMul(n)'
 - $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ - 'isAssociativeAdd(n)'
 - $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ - 'isAssociativeMul(n)'
 - $(x \oplus y) \otimes z = (x \otimes y) \oplus (y \otimes z)$ - 'isDistributive(n)'

- **Easy Requirement 3** - The file `twisted_integers.py` contains a class `TwistedIntegers` which implements a data structure representing \mathbb{Z}_n with respect to the operations $a \oplus b = (a \oplus b) \bmod n$ and $a \otimes b = (a \otimes b) \bmod n$ and contains the methods `__init__`, `__str__` and `size` where `size` returns the number of elements in \mathbb{Z}_n .

1.2.2 Medium

- **Medium Requirement 1** - In the `twisted_integers.py` file (along with the aforementioned `TwistedIntegers` class) the `IteratorOfTwistedIntegers` class has been implemented to iterate over instances of `TwistedIntegers` and contains an `__init__` function (to initialise it with a given instance of `TwistedIntegers`), a `hasNext()` boolean function (to indicate if the iterator has a next value) and a `next()` function (to return the next value in the iterator).
- **Medium Requirement 2** - The function `'findValAdd(n)'` in the `twisted_integers.py` file finds for a given n all elements τ of \mathbb{Z}_n such that $\tau \oplus x = x$ for all $x \in \mathbb{Z}_n$.
- **Medium Requirement 3** - The function `'findValMul(n)'` in the `twisted_integers.py` file finds for a given n all elements ε of \mathbb{Z}_n such that $\varepsilon \otimes x = x$ for all $x \in \mathbb{Z}_n$.

1.2.3 Hard

- **Hard Requirement 1** - In the `twisted_int_matrix.py` file the class `TwistedIntMatrix` has been implemented which represents a matrix of `TwistedInt` objects. The `__init__` function of the class takes an x and y dimension as well as a list of `TwistedInt` objects and creates a matrix containing the values in the list. The `__mul__` function implements matrix multiplication (between two given matrices) using the row by column rule and the \oplus and \otimes operators. This is achieved using the `calcDotProduct` function which calculates the dot product between a row and a column of the two matrices (which itself uses the `getCol` function to access a column of the second matrix).

For the same requirement, an algorithm has been implemented using the function `getPossibleMatrices(matrices)` which takes a list of M matrices and calculates all of the possible unique matrices that may be obtained by multiplying these matrices, i.e. the set $\{g_1 \ g_2 \ \dots \ g_k \mid k \in \mathbb{N}, g_i \in M\}$.

- **Hard Requirement 2** - The level of testing and documentation has been enhanced by including docstrings for every function and class, as well as example doctests within these docstrings that can be extracted and run to verify the correctness of the functions which they document.

1.3 Further Features:

The follow additional features have also been implemented:

- An iterator class, `IteratorOfTwistedIntMatrix`, has been implemented in `twisted_int_matrix.py` file which iterates over an instance of `TwistedIntMatrix` from top left to bottom right across the `TwistedInt` elements. As with the `IteratorOfTwistedIntegers` iterator, this iterator has a `hasNext()`, `next()` and `__init__` functions.
-

2 Design and Implementation

3 Evidence of Testing

4 Known Problems

5 Problems Overcome

6 Summary of Provenance

6.1 Code Implemented by the Group

6.2 Code Modified From the Provided Example Code

6.3 Code Sourced From Elsewhere

7 Conclusion