NLA Serie 1 Documentation

Release 1.0

Tom Lambert, Yuuma Odaka-Falush

Nov 27, 2017

CONTENTS

CHAPTER

ONE

WELCOME TO NLA SERIES 1'S DOCUMENTATION!

This is our implementation for Series 1, Numerical Linear Algebra. The task was to create a class which can create and execute operations on fractions. Operations include addition, subtraction, multiplication, division, equality and inequality.

The main topic in the problem is object-oriented programming in Python, more specifically working with fractions. The following documentation explains how our class Fraction and its associated programs work.

A number of automated unit tests were carried out to guarantee the correct execution of the Fraction and Prime classes. The tests were carried out in the implemented main program.

We imported a number of python modules to support our implementation:

Console provided us with functions which make it easier to accept and process user inputs.

Numpy is a package that contains many useful mathematical functions.

1.1 Modules

1.1.1 bruch module

1.1.1.1 Bruch class

Bruch class exists only to fulfill the task. The actual implementation is in Fraction class. Bruch inherits from Fraction, thus it has all of the members of Fraction. See *Members of Fraction class*.

class bruch.**Bruch**(*numerator*, *denominator*)

Bases: *fraction.Fraction*

A derivative of the Fraction-class without other implementations. For UnitTests see FractionTests.

bruch.**main**()

The main program. It runs unittests to test the main-modules.

bruch.**run_test**(*class_name*, *fx*)

Runs a given unit-test and prints the result.

Parameters

- **class_name** – The class name which will be tested. It will be printed ith the results.
- **fx** – The test-function to execute.

Returns A result-object with the result of the tests.

1.1.2 fraction module

1.1.2.1 Mathematical background

A Fraction objects consists of its instance, a numerator and a denominator. All calculations obey the rules governing adding, subtracting, multiplying and dividing fractions. If the denominator of a fraction is 0, any calculations involivng it will return "NaN". Subtraction and division default to modified addition and multiplication, and the greatest common divisors are always reduced.

1.1.2.2 Members of Fraction class

class fraction.**Fraction**(*numerator*, *denominator*)

Represents a fraction with two integers.

__add__(*other*)

Adds an integer or another fraction to this instance and returns the result.

Parameters **other** – The other value to add; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the addition.

__div__(*other*)

Divides this instance by an integer or another fraction and returns the result.

Parameters **other** – The other value to divide with; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the division.

__eq__(*other*)

Compares this Fraction with another Fraction, float, int or long for value-equality.

Parameters **other** – The other value to compare with; it can be another Fraction, float, int or long

Returns True, if the values are equal; otherwise False.

__float__()

Converts the value of this instance into a float-value. The result must not be exact.

Returns A approximated float-value of this instances value.

__ge__(*other*)

Checks if the value of this instance is greater or equal than another value.

Parameters **other** – The other value to compare with; it can be another Fraction float, int or long

Returns True if this instances value is greater or equal then the other value; otherwise False.

__gt__(*other*)

Checks if the value of this instance is greater than another value.

Parameters **other** – The other value to compare with; it can be another Fraction float, int or long

Returns True if this instances value is greater then the other value; otherwise False.

__init__(*numerator*, *denominator*)

Initializes a new Fraction-Instance with a value.

Parameters

- **numerator** – The numerator in the instance to created.

- **denominator** – The denominator in the instance to create.

__le__(*other*)
Checks if the value of this instance is less or equal than another value.

Parameters **other** – The other value to compare with; it can be another Fraction float, int or long

Returns True if this instances value is less or equal then the other value; otherwise False.

__lt__(*other*)
Checks if the value of this instance is less then another value.

Parameters **other** – The other value to compare with; it can be another Fraction float, int or long

Returns True if this instances value is less then the other value; otherwise False.

__mul__(*other*)
Multiplies an integer or another fraction with this instance and returns the result.

Parameters **other** – The other value to multiply with; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the multiplication.

__ne__(*other*)
Compares this Fraction with another Fraction, float, int or long for value-inequality.

Parameters **other** – The other value to compare with; it can be another Fraction, float, int or long

Returns False, if the values are equal; otherwise True.

__neg__()
Negates the value of this instance and returns it.

Returns A new Fraction instance with the negated value of this instance

__radd__(*other*)
Adds an integer or another fraction to this instance and returns the result.

Parameters **other** – The other value to add; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the addition.

__rdiv__(*other*)
Divides an integer or another fraction with this instance and returns the result.

Parameters **other** – The other value to divide; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the division.

__rmul__(*other*)
Multiplies an integer or another fraction with this instance and returns the result.

Parameters **other** – The other value to multiply with; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the multiplication.

__rsub__(*other*)
Subtracts this instance from an integer or another fraction and returns the result.

Parameters **other** – The other value to subtract from; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the subtraction.

__str__()
Creates a string representation for this instance.

Returns

- "NaN" if the denominator is 0;
- "0" if the denominator is 0;
- the numerator-value as a string if the denominator is 1;
- otherwise "numerator / denominator"

__sub__(*other*)
Subtracts an integer or another fraction from this instance and returns the result.

Parameters **other** – The other value to subtract; it can be a Fraction, int or long.

Returns A new Fraction instance with the result of the subtraction.

clone()
Creates a copy of this instance.

Returns A new instance with the same value as this fraction.

reduce()
Reduces the fraction by removing all common prime factors.

1.1.3 prime module

1.1.3.1 Mathematical background

Prime numbers are a special subset of the naturals. They can be effectively utilized to find the greatest common divisor of two numbers.

1.1.3.2 Members of Prime class

class prime.**Prime**
Provides methods to obtain prime numbers and use them.

__init__()
This class should not be initialized. All substantial members are static.

static append_next_to_cache()
Calculates the next prime number which is not in the cache.

Returns The added prime number.

cache = [2, 3, 5, 7]

static get_greatest_common_divisor(*a*, *b*)
Calculates the greatest common divisor

Parameters

- **a** – The first number.
- **b** – The second number.

Returns The greatest common divisor of a and b.

static get_prime(*index*)
Returns the prime number at the given index. The index starts with 0.

Parameters **index** (*int*) – The index of the requested prime number.

Returns The prime number at position index.

static get_prime_factors(*num*)
Returns the prime factors of the given number.

Parameters **num** (*long*) – The number to split in prime factors.

Returns An array of prime factors of num.

Raises **ValueError** – if num is <= 1

1.1.3.3 Remarks

The generation of prime numbers is accelerated with a cache of already known prime numbers in the RAM.

PYTHON MODULE INDEX

INDEX