Bruch Documentation

Release 1.0

Marvin Ertl

Oct 23, 2016

CONTENTS

Contents:

CHAPTER

ONE

BRUCH

class `bruch.`**`Bruch`**`(`**args*`)`

Bases: `object`

The class Bruch represents a fraction. Nearly all operator of this class are overloaded.

static **`_Bruch__makeBruch`**`(`*value*`)`

Creates a fraction :param value: int or a fraction :return: a fraction

`__abs__``()`

Returns absolute value of Bruch :return: float

`__add__``(`*other*`)`

Adds other to self :param other: int or a fraction :return: a fraction

`__dict__` = dict_proxy({'__int__': <function __int__ at 0x04395B70>, '__module__': 'bruch.Bruch', '__rtruediv__': <f

`__div__``(`*other*`)`

Divide ergebnis through other :param other: int or a fraction :return: float

`__eq__``(`*other*`)`

Test if self is equal to other :param other: int or a fraction :return: boolean

`__float__``()`

Returns float of Bruch :return: float

`__ge__``(`*other*`)`

Test if self is equal/bigger to other :param other: int or a fraction :return: boolean

`__gt__``(`*other*`)`

Test if self is bigger than other :param other: int or a fraction :return: boolean

`__iadd__``(`*other*`)`

Adds self to other :param other: int or a fraction :return: float

`__imul__``(`*other*`)`

Multiply self with other :param other: int or a fraction :return: float

`__init__``(`**args*`)`

Create a fraction or throw a exception if the parameters are not correct, for example Zero Division, Type Error. :param args: parameters for the fraction can be int, float or Bruch

`__int__``()`

Returns int of Bruch :return: int

`__invert__``()`

Divide nenner through zaehler :return: float

`__isub__``(`*other*`)`

Substract other from self :param other: int or a fraction :return: float

`__iter__`()
: Iterator of the fraction :return: iterator

`__itruediv__`(*other*)
: Divide self through other :param other: int or a fraction :return: float

`__le__`(*other*)
: Test if self is smaller/equal to other :param other: int or a fraction :return: boolean

`__lt__`(*other*)
: Test if self is smaller than other :param other: int or a fraction :return: boolean

`__module__` = 'bruch.Bruch'

`__mul__`(*other*)
: Multiply self with other :param other: int or a fraction :return: float

`__ne__`(*other*)
: Test if self is not equal to other :param other: int or a fraction :return: boolean

`__neg__`()
: Divide zaehler through zaehler and take it -1 times :return: float

`__pow__`(*power*, *modulo=None*)
: Takes self up to tje power :param power: int :param modulo: None :return: a fraction

`__radd__`(*other*)
: Adds self to other :param other: int or a fraction :return: float

`__rdiv__`(*other*)
: Divide self.zaehler through other :param other: int or a fration :return: float

`__rmul__`(*other*)
: Multiply other with self :param other: int or a fraction :return: float

`__rsub__`(*other*)
: Substract self from other :param other: int or a fraction :return: float

`__rtruediv__`(*other*)
: Multiply self to not other :param other: int or a fraction :return: float

`__str__`()
: Represented the fraction as (zaehler/nenner) :return: str

`__sub__`(*other*)
: Substract other from self :param other: int or a fraction :return: float

`__truediv__`(*other*)
: Multiply self to not other :param other: int or a fraction :return: float

`__weakref__`
: list of weak references to the object (if defined)

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX