**Документация к классу LongInt**

Описание Класс **LongInt** представляет длинное целое число и предоставляет методы для работы с ним, включая арифметические операции, такие как сложение, вычитание, умножение и деление.

**Элементы класса**

**Конструкторы**

1. **LongInt(value=0)**
   * **Описание:** Создает объект LongInt со значением, указанным в параметре value.
   * **Параметры:**
     + **value**: Значение для инициализации объекта. По умолчанию 0.
   * **Возвращаемое значение:** Объект LongInt.

**Методы**

1. **\_init\_from\_string(value\_str)**
   * **Описание:** Инициализирует объект LongInt из строки.
   * **Параметры:**
     + **value\_str:** Строка, содержащая представление длинного целого числа.
   * **Возвращаемое значение:** Отсутствует.
2. **\_init\_from\_integer(value\_int)**
   * **Описание:** Инициализирует объект LongInt из целого числа**.**
   * **Параметры:**
     + **value\_int:** Целое число для инициализации объекта.
   * **Возвращаемое значение:** Отсутствует.
3. **\_remove\_leading\_zeros()**
   * **Описание:** Удаляет ведущие нули из числа.
   * **Параметры:** Отсутствуют.
   * **Возвращаемое значение:** Отсутствует.

**Перегруженные операторы**

1. **\_\_str\_\_()**
   * **Описание: Возвращает строковое представление числа.**
   * **Параметры: Отсутствуют.**
   * **Возвращаемое значение: Строковое представление числа.**
2. **\_\_neg\_\_()**
   * **Описание: Возвращает число с противоположным знаком.**
   * **Параметры: Отсутствуют.**
   * **Возвращаемое значение: Число с измененным знаком.**
3. **\_\_eq\_\_(other)**
   * **Описание: Проверяет равенство двух чисел.**
   * **Параметры:**
     + **other: Другой объект LongInt для сравнения.**
   * **Возвращаемое значение: True, если числа равны, иначе False.**

**Методы (продолжение)**

1. **\_\_lt\_\_(other)**
   * **Описание:** Проверяет, меньше ли текущее число, чем другое число.
   * **Параметры:**
     + **other:** Другой объект **LongInt** для сравнения.
   * **Возвращаемое значение:** True, если текущее число меньше, чем other, иначе False.
2. **\_\_le\_\_(other)**
   * **Описание:** Проверяет, меньше или равно ли текущее число, чем другое число.
   * **Параметры:**
     + **other:** Другой объект **LongInt** для сравнения.
   * **Возвращаемое значение:** True, если текущее число меньше или равно other, иначе False.
3. **\_\_gt\_\_(other)**
   * **Описание:** Проверяет, больше ли текущее число, чем другое число.
   * **Параметры:**
     + **other:** Другой объект LongInt для сравнения.
   * **Возвращаемое значение:** True, если текущее число больше other, иначе False.
4. **\_\_ge\_\_(other)**
   * **Описание:** Проверяет, больше или равно ли текущее число, чем другое число.
   * **Параметры:**
     + **other:** Другой объект LongInt для сравнения.
   * **Возвращаемое значение:** True, если текущее число больше или равно other, иначе False.
5. **\_\_add\_\_(other)**
   * **Описание:** Выполняет операцию сложения с другим числом.
   * **Параметры:**
     + **other:** Другой объект LongInt для сложения**.**
   * **Возвращаемое значение:** Результат сложения.
6. **\_\_sub\_\_(other)**
   * **Описание:** Выполняет операцию вычитания с другим числом.
   * **Параметры:**
     + **other:** Другой объект LongInt для вычитания.
   * **Возвращаемое значение:** Результат вычитания.
7. **\_\_mul\_\_(other)**
   * **Описание:** Выполняет операцию умножения на другое число.
   * **Параметры:**
     + **other:** Другой объект LongInt для умножения.
   * **Возвращаемое значение:** Результат умножения.
8. **\_\_truediv\_\_(other)**
   * **Описание:** Выполняет операцию деления на другое число.
   * **Параметры:**
     + **other:** Другой объект **LongInt** для деления.
   * **Возвращаемое значение:** Результат деления.
9. **\_\_floordiv\_\_(other)**
   * **Описание:** Выполняет операцию целочисленного деления на другое число.
   * **Параметры:**
     + **other:** Другой объект **LongInt** для деления.
   * **Возвращаемое значение:** Результат целочисленного деления.
10. **\_\_mod\_\_(other)**
    * **Описание:** Выполняет операцию взятия остатка от деления на другое число.
    * **Параметры:**
      + **other:** Другой объект **LongInt** для деления.
    * **Возвращаемое значение:** Результат операции взятия остатка.
11. **odd()**
    * **Описание:** Проверяет, является ли число нечетным.
    * **Параметры:** Отсутствуют.
    * **Возвращаемое значение:** True, если число нечетное, иначеFalse.
12. **even()**
    * **Описание:** Проверяет, является ли число четным.
    * **Параметры:** Отсутствуют.
    * **Возвращаемое значение:** True, если число четное, иначе False.
13. **pow(n)**
    * **Описание:** Возвращает число, возведенное в степень n.
    * **Параметры:**
      + **n:** Степень, в которую нужно возвести число.
    * **Возвращаемое значение:** Результат возведения числа в степень.

**Тестирование класса LongInt**

Для тестирования класса **LongInt** на корректность работы всех его методов используется функция **test\_long\_int()**. Она генерирует случайные числа и проверяет правильность выполнения арифметических операций.

**import random def test\_long\_int(): random.seed() # Инициализация генератора**

**Код программы**

import random

class LongInt:

BASE = 1000000000

def \_\_init\_\_(self, value=0):

self.\_digits = []

self.\_is\_negative = False

if isinstance(value, str):

self.\_init\_from\_string(value)

elif isinstance(value, int):

self.\_init\_from\_integer(value)

else:

raise TypeError("Unsupported type for initializing LongInt")

def \_init\_from\_string(self, value\_str):

if len(value\_str) == 0:

self.\_is\_negative = False

else:

if value\_str[0] == '-':

value\_str = value\_str[1:]

self.\_is\_negative = True

else:

self.\_is\_negative = False

for i in range(len(value\_str), 0, -9):

if i < 9:

self.\_digits.append(int(value\_str[0:i]))

else:

self.\_digits.append(int(value\_str[i - 9:i]))

self.\_remove\_leading\_zeros()

def \_init\_from\_integer(self, value\_int):

if value\_int < 0:

self.\_is\_negative = True

value\_int = abs(value\_int)

while value\_int != 0:

self.\_digits.append(value\_int % self.BASE)

value\_int //= self.BASE

def \_remove\_leading\_zeros(self):

while len(self.\_digits) > 1 and self.\_digits[-1] == 0:

self.\_digits.pop()

if len(self.\_digits) == 1 and self.\_digits[0] == 0:

self.\_is\_negative = False

def \_\_str\_\_(self):

result = ""

if self.\_is\_negative:

result += "-"

if self.\_digits:

result += str(self.\_digits[-1])

for i in range(len(self.\_digits) - 2, -1, -1):

result += str(self.\_digits[i]).zfill(9)

else:

result += "0"

return result

def \_\_neg\_\_(self):

result = LongInt()

result.\_digits = self.\_digits[:]

result.\_is\_negative = not self.\_is\_negative

return result

def \_\_pos\_\_(self):

result = LongInt()

result.\_digits = self.\_digits[:]

result.\_is\_negative = self.\_is\_negative

return result

def \_\_eq\_\_(self, other):

if self.\_is\_negative != other.\_is\_negative:

return False

if not self.\_digits:

return not other.\_digits or (len(other.\_digits) == 1 and other.\_digits[0] == 0)

if not other.\_digits:

return len(self.\_digits) == 1 and self.\_digits[0] == 0

if len(self.\_digits) != len(other.\_digits):

return False

for i in range(len(self.\_digits)):

if self.\_digits[i] != other.\_digits[i]:

return False

return True

def \_\_lt\_\_(self, other):

if self == other:

return False

if self.\_is\_negative:

if other.\_is\_negative:

return -other < -self

else:

return True

elif other.\_is\_negative:

return False

else:

if len(self.\_digits) != len(other.\_digits):

return len(self.\_digits) < len(other.\_digits)

else:

for i in range(len(self.\_digits) - 1, -1, -1):

if self.\_digits[i] != other.\_digits[i]:

return self.\_digits[i] < other.\_digits[i]

return False

def \_\_le\_\_(self, other):

return self < other or self == other

def \_\_gt\_\_(self, other):

return not self <= other

def \_\_ge\_\_(self, other):

return not self < other

def \_\_add\_\_(self, other):

if self.\_is\_negative:

if other.\_is\_negative:

return -(-self + (-other))

else:

return other - (-self)

elif other.\_is\_negative:

return self - (-other)

carry = 0

result\_digits = []

for i in range(max(len(self.\_digits), len(other.\_digits))):

sum\_ = (self.\_digits[i] if i < len(self.\_digits) else 0) + (

other.\_digits[i] if i < len(other.\_digits) else 0) + carry

carry = sum\_ // self.BASE

result\_digits.append(sum\_ % self.BASE)

if carry:

result\_digits.append(carry)

result = LongInt()

result.\_digits = result\_digits

return result

def \_\_iadd\_\_(self, other):

return self + other

def \_\_sub\_\_(self, other):

if other.\_is\_negative:

return self + (-other)

elif self.\_is\_negative:

return -(-self + other)

elif self < other:

return -(other - self)

result\_digits = []

borrow = 0

for i in range(len(self.\_digits)):

diff = self.\_digits[i] - (other.\_digits[i] if i < len(other.\_digits) else 0) - borrow

if diff < 0:

diff += self.BASE

borrow = 1

else:

borrow = 0

result\_digits.append(diff)

result = LongInt()

result.\_digits = result\_digits

result.\_remove\_leading\_zeros()

return result

def \_\_isub\_\_(self, other):

return self - other

def \_\_mul\_\_(self, other):

result\_digits = [0] \* (len(self.\_digits) + len(other.\_digits))

for i in range(len(self.\_digits)):

carry = 0

for j in range(len(other.\_digits)):

temp = self.\_digits[i] \* other.\_digits[j] + result\_digits[i + j] + carry

result\_digits[i + j] = temp % self.BASE

carry = temp // self.BASE

result\_digits[i + len(other.\_digits)] = carry

result = LongInt()

result.\_digits = result\_digits

result.\_is\_negative = self.\_is\_negative != other.\_is\_negative

result.\_remove\_leading\_zeros()

return result

def \_\_imul\_\_(self, other):

return self \* other

def \_\_truediv\_\_(self, other):

if other == LongInt("0"):

raise ZeroDivisionError("division by zero")

divisor = other if other.\_is\_negative == False else -other

dividend = self if self.\_is\_negative == False else -self

quotient = LongInt()

remainder = LongInt()

for i in range(len(dividend.\_digits) - 1, -1, -1):

remainder \*= LongInt.BASE

remainder += LongInt(dividend.\_digits[i])

quotient\_digit = 0

while remainder >= divisor:

remainder -= divisor

quotient\_digit += 1

quotient.\_digits.insert(0, quotient\_digit)

quotient.\_remove\_leading\_zeros()

quotient.\_is\_negative = self.\_is\_negative != other.\_is\_negative

return quotient

def \_\_itruediv\_\_(self, other):

return self / other

def \_\_abs\_\_(self):

result = LongInt()

result.\_digits = self.\_digits.copy()

result.\_is\_negative = False

if self.\_is\_negative:

result.\_is\_negative = False

return result

def \_\_int\_\_(self):

if not self.\_digits:

return 0

value = self.\_digits[-1]

if len(self.\_digits) > 1:

for digit in reversed(self.\_digits[:-1]):

value = value \* 10 \*\* 9 + digit

return value

def \_\_floordiv\_\_(self, other):

if other == LongInt(0):

raise ZeroDivisionError("division by zero")

result = LongInt()

dividend = abs(self)

divisor = abs(other)

quotient = dividend // divisor

result.\_digits = quotient.\_digits

result.\_is\_negative = self.\_is\_negative != other.\_is\_negative

return result

def \_\_mod\_\_(self, other):

return self - (self // other) \* other

def \_\_imod\_\_(self, other):

return self % other

def odd(self):

if not self.\_digits:

return False

return self.\_digits[0] & 1

def even(self):

return not self.odd()

def pow(self, n):

if n == LongInt("0"):

return LongInt("1")

result = LongInt("1")

base = self

while n > LongInt("0"):

if n.odd():

result \*= base

base \*= base

n //= LongInt("2")

return result

def test\_long\_int():

def generate\_random\_number(min\_val, max\_val):

return random.randint(min\_val, max\_val)

def test\_long\_int():

random.seed() # Инициализация генератора случайных чисел

for \_ in range(1000000):

aa = generate\_random\_number(-45000, 45000)

bb = generate\_random\_number(-45000, 45000)

cc = aa \* bb

dd = cc // bb

a = LongInt(aa)

b = LongInt(bb)

c = a \* b

c2 = LongInt(aa \* bb)

d = c // b

if d != a:

print("UNCORRECT d != a")

if dd != aa:

print("UNCORRECT dd != aa")

if d != dd:

print("UNCORRECT d != dd")

if \_\_name\_\_ == "\_\_main\_\_":

test\_long\_int()