МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Математический факультет

Кафедра функционального анализа

Отчет по дисциплине:

«Программирование криптографических алгоритмов»

Направление 02.04.01 Математика и компьютерные науки

Преподаватель _____ к.ф.-м.н. М.Г. Завгородний
подпись

Обучающийся _____ А.А. Уткин
подпись

Воронеж 2021

Содержание

1 Постановка задачи

Составьте алгоритм (в виде блок-схемы) и напишите (на любом языке программирования) соответствующую ему программу, позволяющую выполнять арифметические операции (сложение, вычитание, умножение и деление) над длинными целыми числами.

2 Используемые инструменты

Для решения вышеуказанной задачи были использованы следующие инструменты:

- Основным ЯП был выбран Python версии 3.9.2;
- Для компиляции программы в бинарный файл .exe использован конвертер файлов Auto PY to EXE, который использует для своей работы PyInstaller.

3 Общая структура библиотеки целых длинных чисел

В библиотеке целых длинных содержится класс «BigInt», внутри которого находятся следующие методы:

- Сложение целых длинных чисел.
  Программная реализация представляет собой сложение чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания.

- Вычитание целых длинных чисел.
  Программная реализация представляет собой вычитание чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания.

- Умножение целых длинных чисел.
  Программная реализация представляет собой умножение чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания.

- Целочисленное деление целых длинных чисел.
  Программная реализация представляет собой деление чисел с помощью метода вычитания. Данный способ реализации был выбран по причине простоты его работы и написания.

Класс «BigInt» содержит в себе два основных поля:

1. Поле хранения числа «value».
   Представляет собой переменную типа строка, в котором содержится число экземпляра класса;

2. Поле хранения знака числа «is_neg».
   Представляет собой переменную типа bool, в которой содержится информация о знаке числа. Значение True эквивалентно отрицательному числу, значение False - положительному;

Создания экземпляра класса «BigInt» происходит следующие способами:

- Создание экземпляра класса без передачи аргументов. Числовое значение такого экземпляра будет равно нулю.

```
1 a = BigInt()
```

- Создание экземпляра класса с передачей в аргумент строки, которая может валидно быть приведена к типу целого числа.

```
1 a = BigInt('-1234567890') # a = -1234567890
2 b = BigInt('1234567890') # b = 1234567890
3 d = BigInt('0') # d = 0
```

- Создание экземпляра класса с передачей в аргумент целого числа.

```
1 a = BigInt(-1234567890) # a = -1234567890
2 b = BigInt(1234567890) # b = 1234567890
3 d = BigInt(0) # d = 0
```

- Создание экземпляра класса с передачей в аргумент экземпляра класса «BigInt».

```
1 a = BigInt(-1234567890) # a = -1234567890
2 b = BigInt(a) # b = -1234567890
```

4 Примеры работы библиотеки

В качестве примера работы будут использоваться прямые вызовы методов класса «BigInt».

Пусть даны два целых длинных числа a и b, сохраненных в экземпляр класса «BigInt». А так же, создадим экземпляр класса «BigInt» с нулевым значением.

```
1 a = BigInt('-12345678909876543 21')
2 b = BigInt('9876543210123456789')
3 zero = BigInt()
```

- Выполним сложение:

```
1 print(a + b)
```

Вывод: 86419753191358024 68

- Выполним вычитание:

```
1 print(a - b)
```

Вывод: −11111111011111111110

- Выполним умножение:

```
1 print(a * b)
```

Вывод: −121932631211705532655235482511126352 69

- Выполним целочисленное деление:

```
1 print(a / b)
```

Вывод: −8

- Выполним деление на ноль:

```
1 print(a / zero)
```

Вывод: *ZeroDivisionError*

- Выполним деление нуля:

```
1 print(zero / b)
```

Вывод: 0

- Выполним умножение на ноль:

```
1 print(a * zero)
```

Вывод: 0

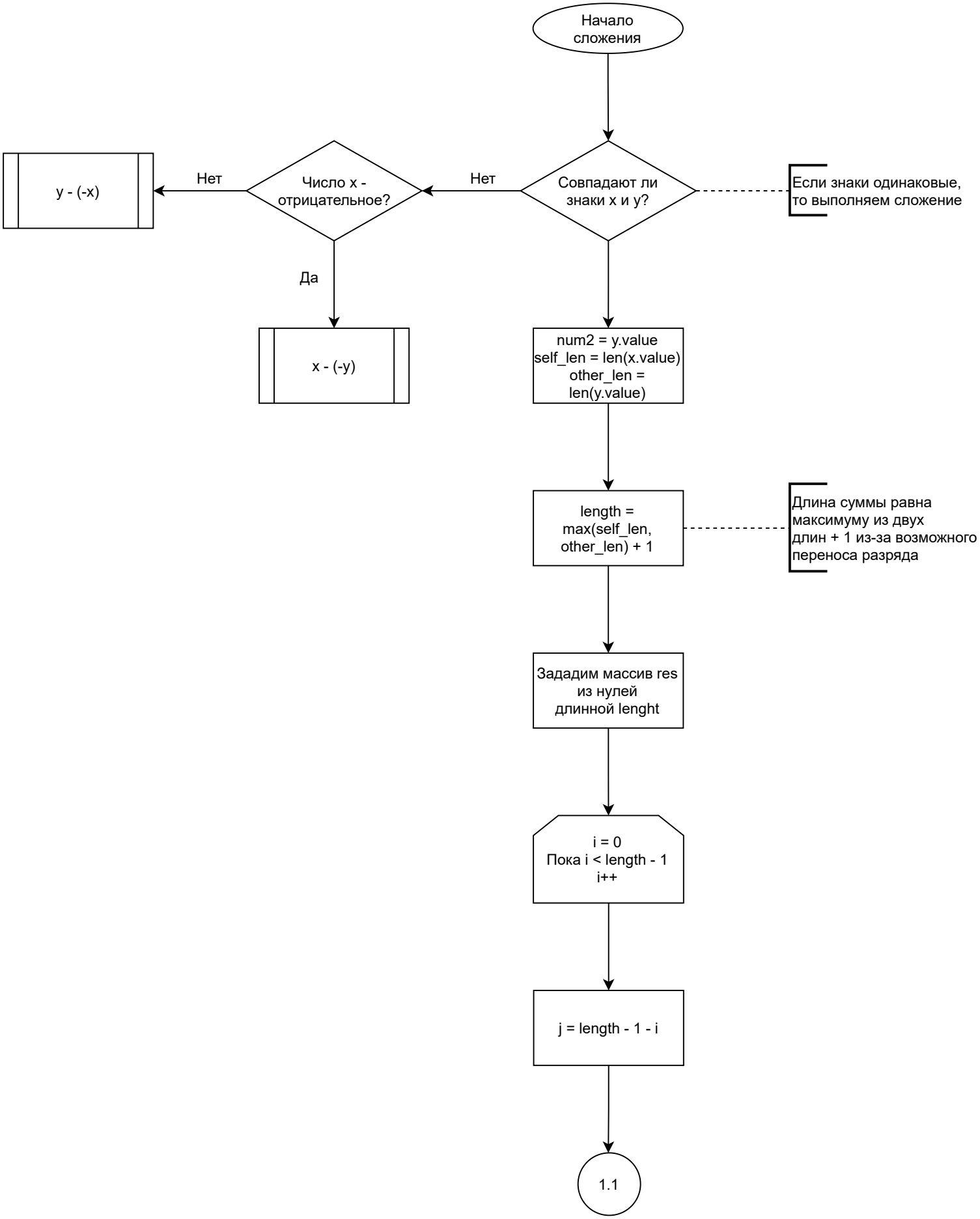
5 Вывод

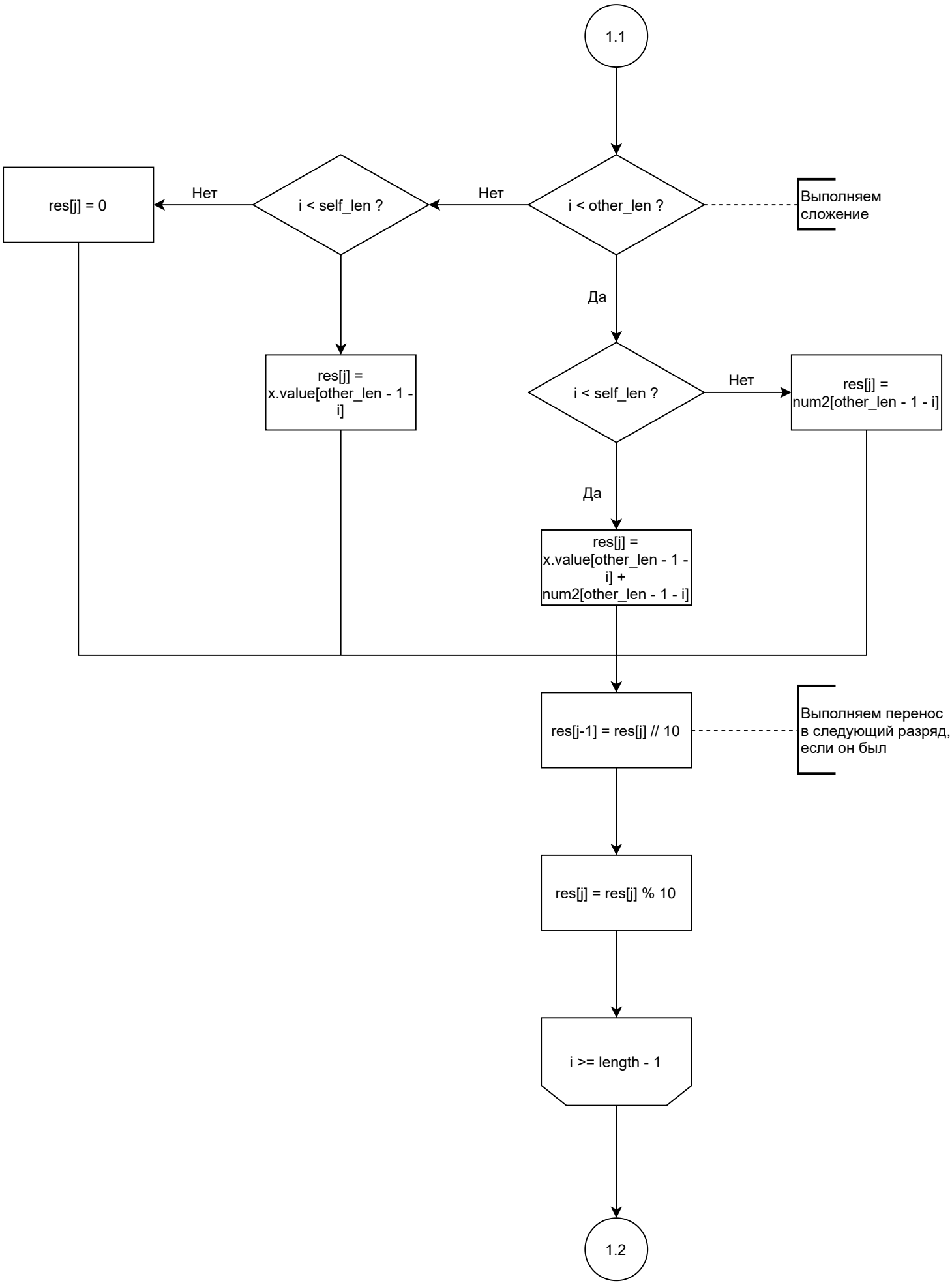
Мною был составлен алгоритм (в виде блок-схемы) и написана на языке Python программа, позволяющая выполнять арифметические операции (сложение, вычитание, умножение и деление) над длинными целыми числами.

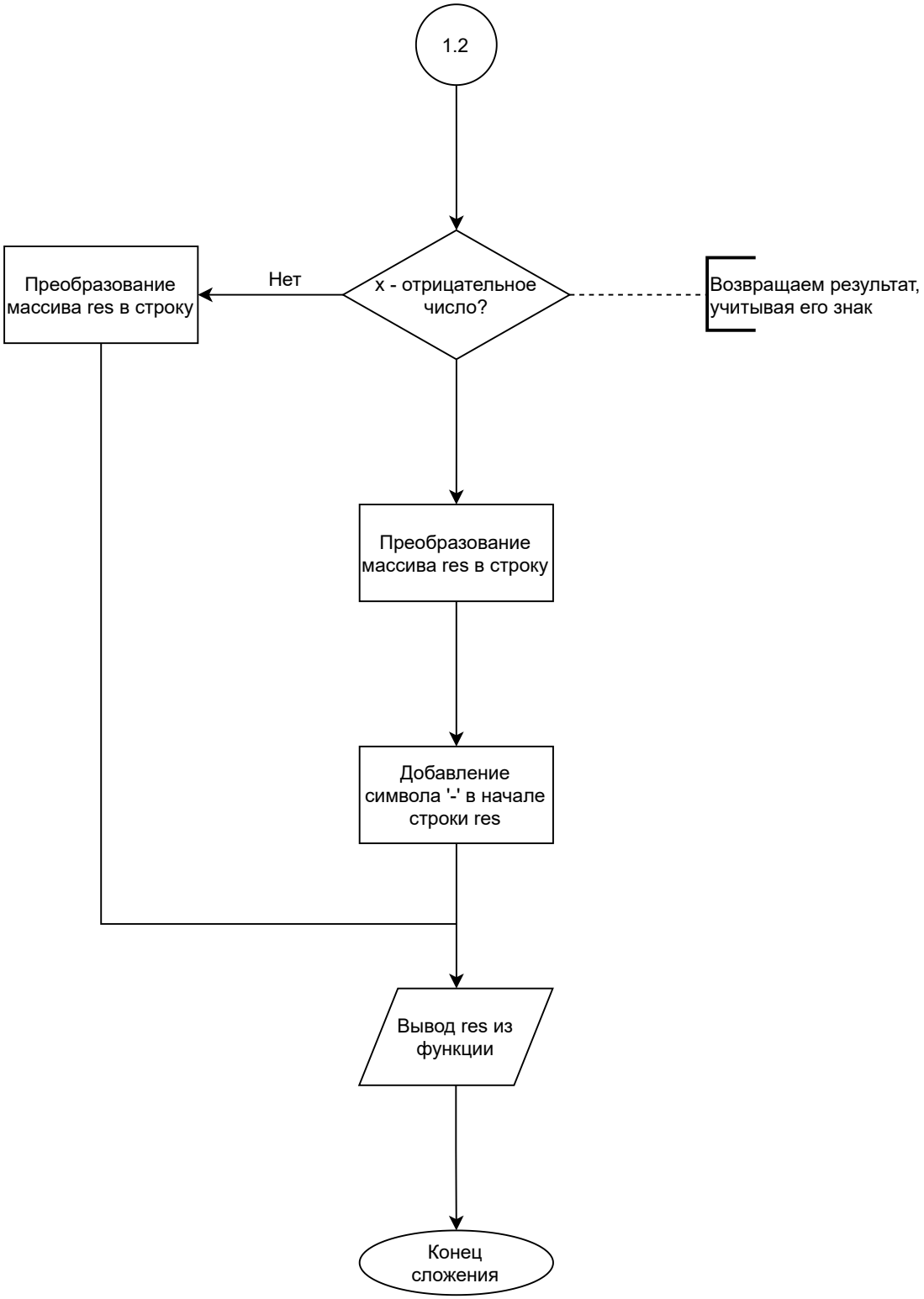
6 Блок-схема методов библиотеки

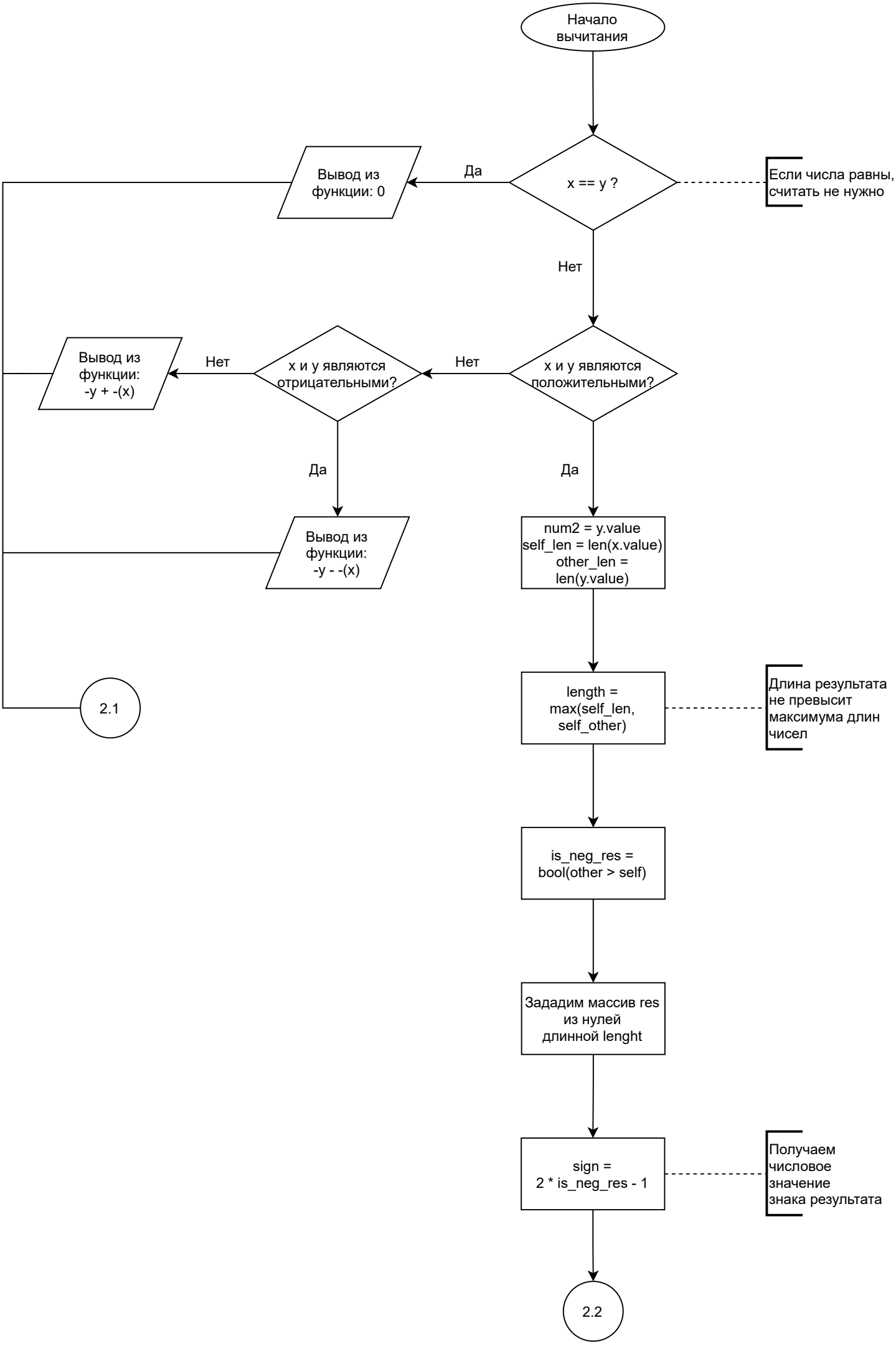
Ниже представлены блок-схемы методов в следующем порядке:

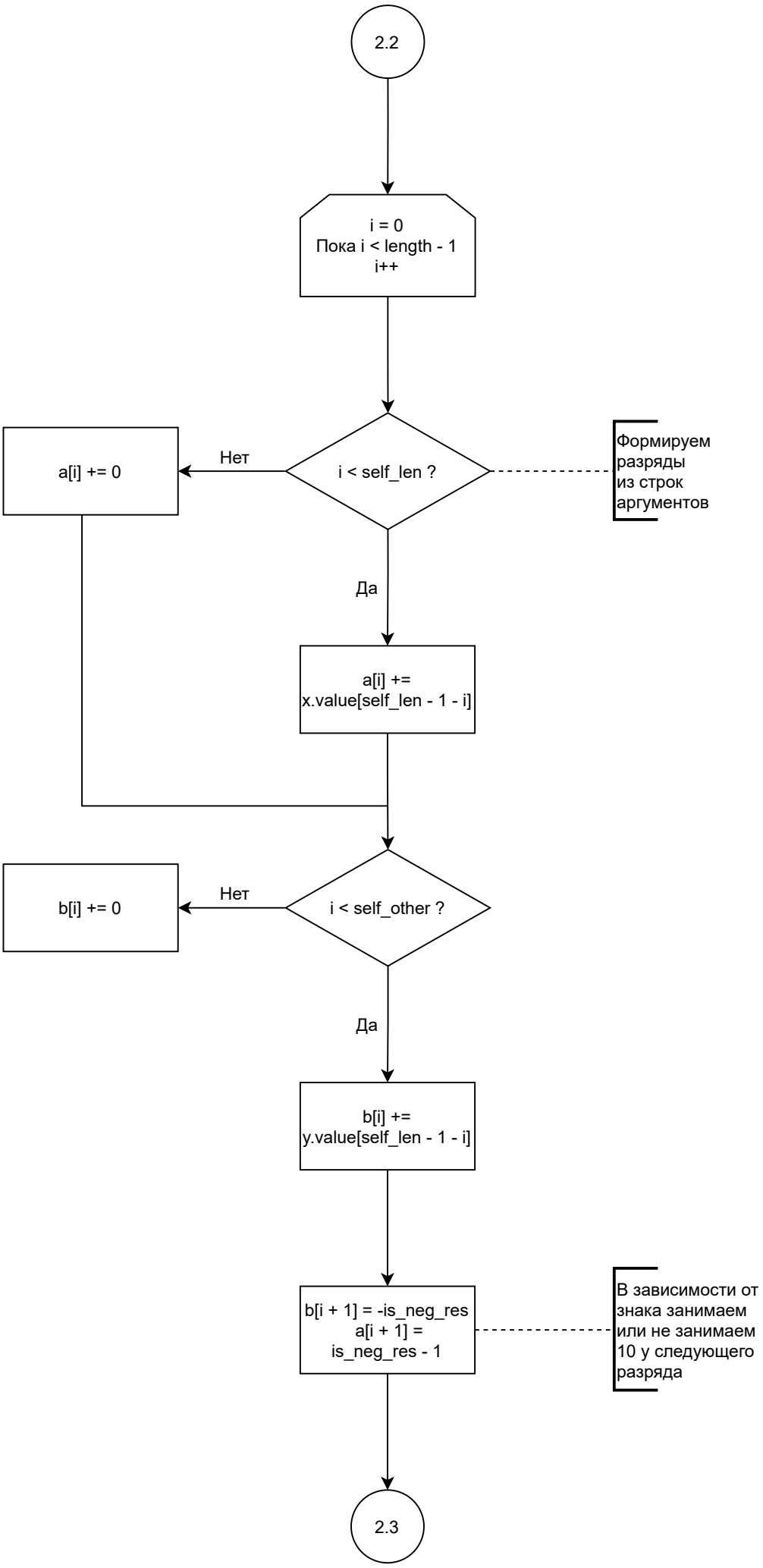
1. Сложение;
2. Вычитание;
3. Умножение;
4. Деление.

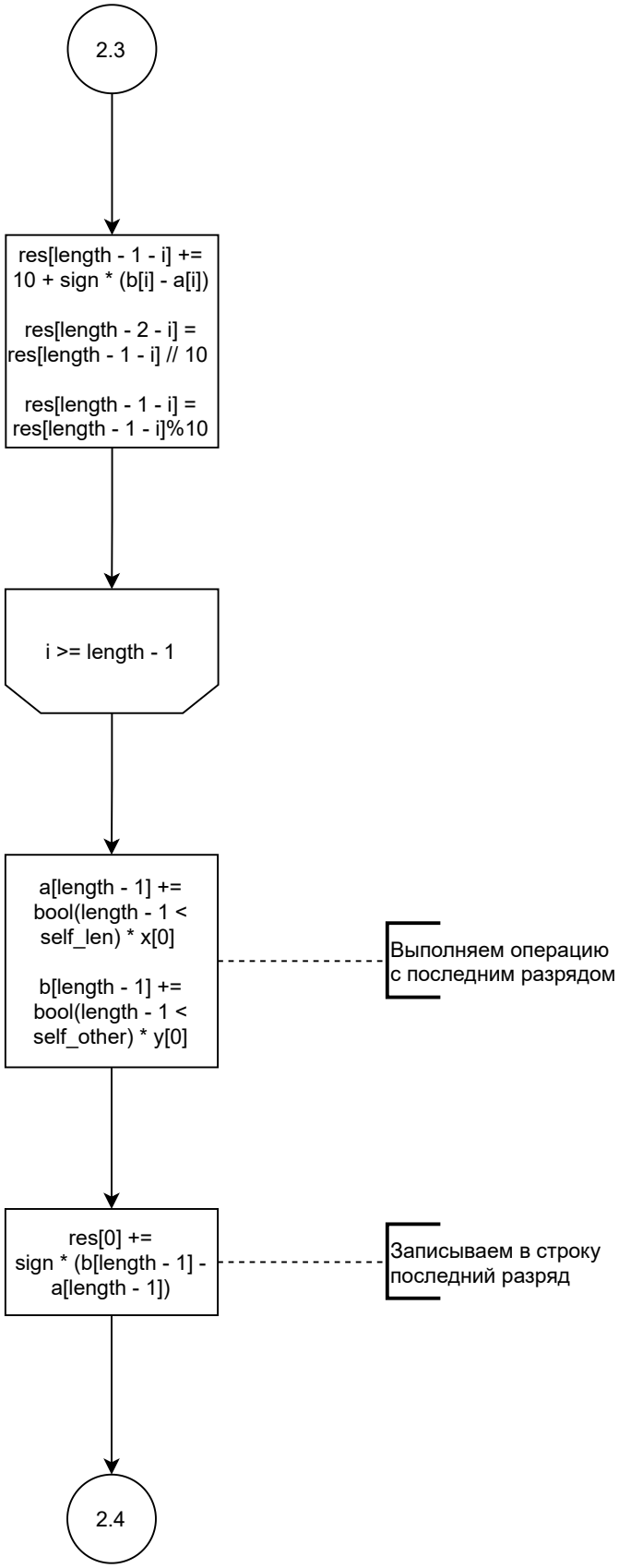

Начало сложения
Совпадают ли знаки x и y?
Нет
Число x - отрицательное?
Нет
y - (-x)
Да
x - (-y)
Если знаки одинаковые, то выполняем сложение
num2 = y.value
self_len = len(x.value)
other_len = len(y.value)
length = max(self_len, other_len) + 1
Длина суммы равна максимуму из двух длин + 1 из-за возможного переноса разряда
Зададим массив res из нулей длинной lenght
i = 0
Пока i < length - 1
i++
j = length - 1 - i
1.1


1.1
i < other_len ?
Нет
Выполняем сложение
Да
i < self_len ?
Нет
res[j] = 0
res[j] = x.value[other_len - 1 - i]
i < self_len ?
Нет
res[j] = num2[other_len - 1 - i]
Да
res[j] = x.value[other_len - 1 - i] + num2[other_len - 1 - i]
res[j-1] = res[j] // 10
Выполняем перенос в следующий разряд, если он был
res[j] = res[j] % 10
i >= length - 1
1.2

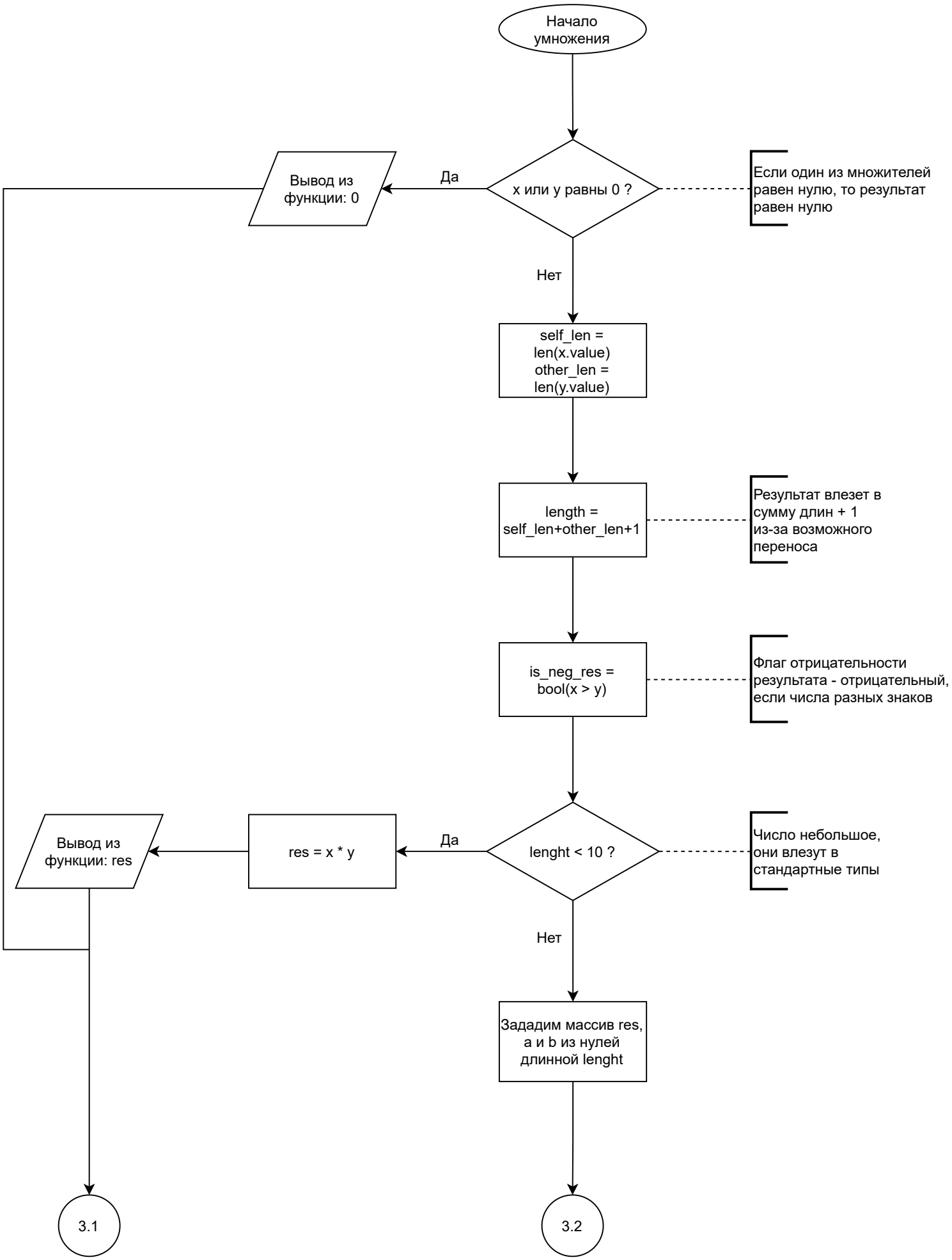

1.2
x - отрицательное число?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
Конец сложения

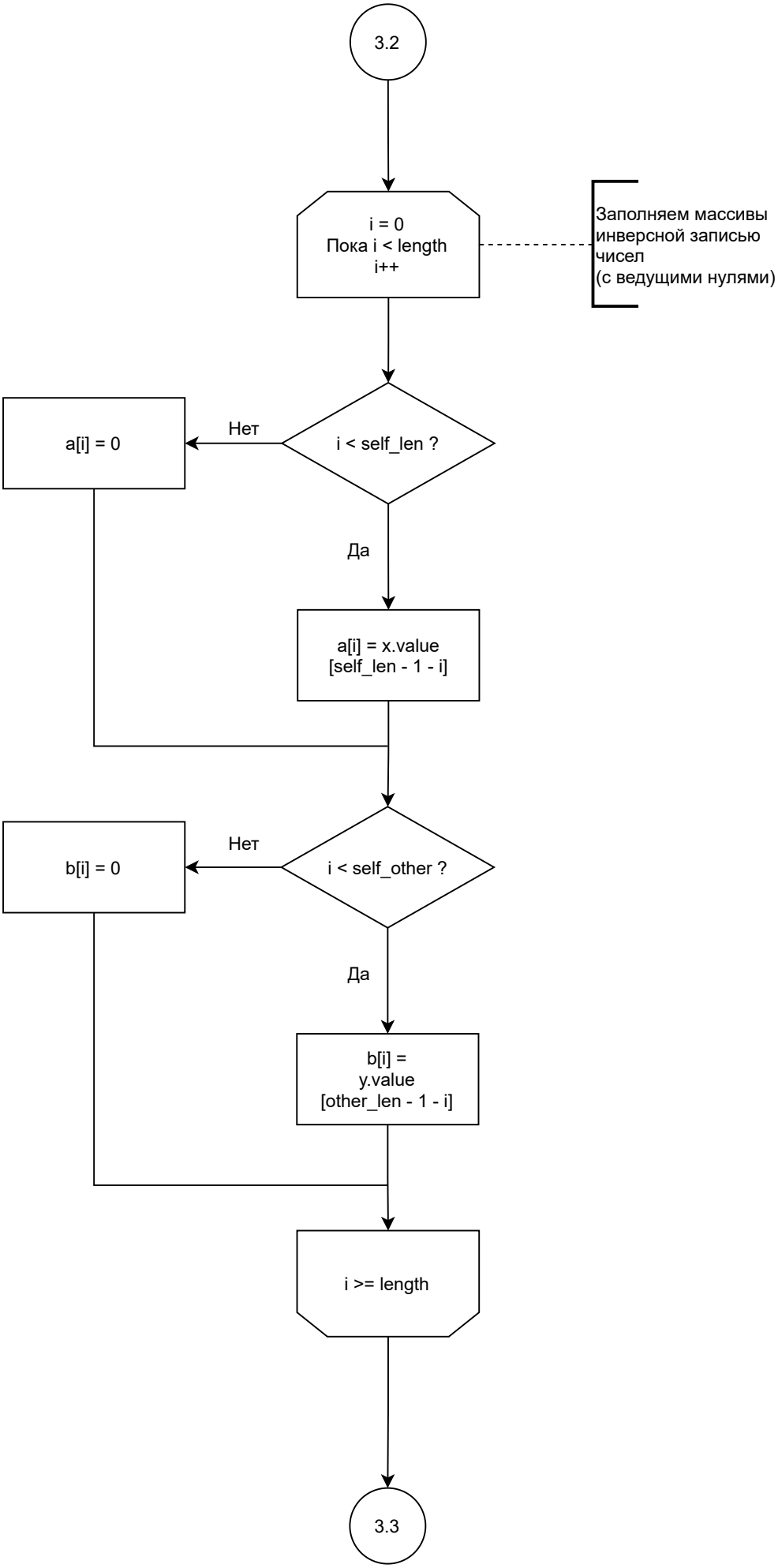

Начало вычитания
x == y ?
Да
Вывод из функции: 0
Если числа равны, считать не нужно
Нет
x и y являются положительными?
Нет
x и y являются отрицательными?
Нет
Вывод из функции: -y + -(x)
Да
Да
Вывод из функции: -y - -(x)
num2 = y.value
self_len = len(x.value)
other_len = len(y.value)
length = max(self_len, self_other)
Длина результата не превысит максимума длин чисел
is_neg_res = bool(other > self)
Зададим массив res из нулей длинной lenght
sign = 2 * is_neg_res - 1
Получаем числовое значение знака результата
2.1
2.2

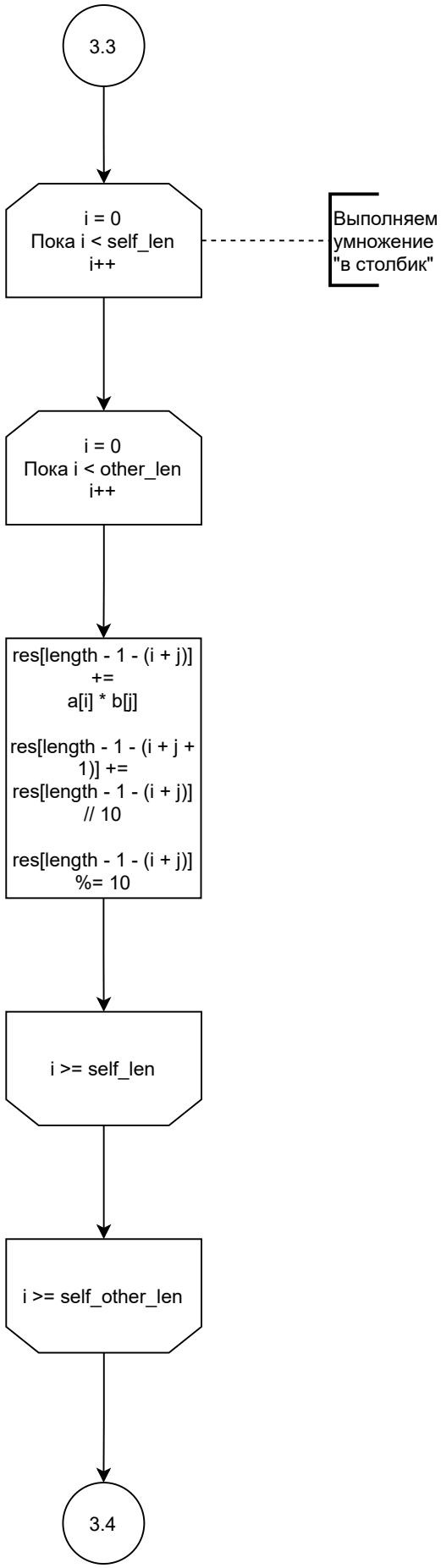

2.2
i = 0
Пока i < length - 1
i++
i < self_len ?
Нет
a[i] += 0
Формируем разряды из строк аргументов
Да
a[i] += x.value[self_len - 1 - i]
i < self_other ?
Нет
b[i] += 0
Да
b[i] += y.value[self_len - 1 - i]
b[i + 1] = -is_neg_res
a[i + 1] = is_neg_res - 1
В зависимости от знака занимаем или не занимаем 10 у следующего разряда
2.3

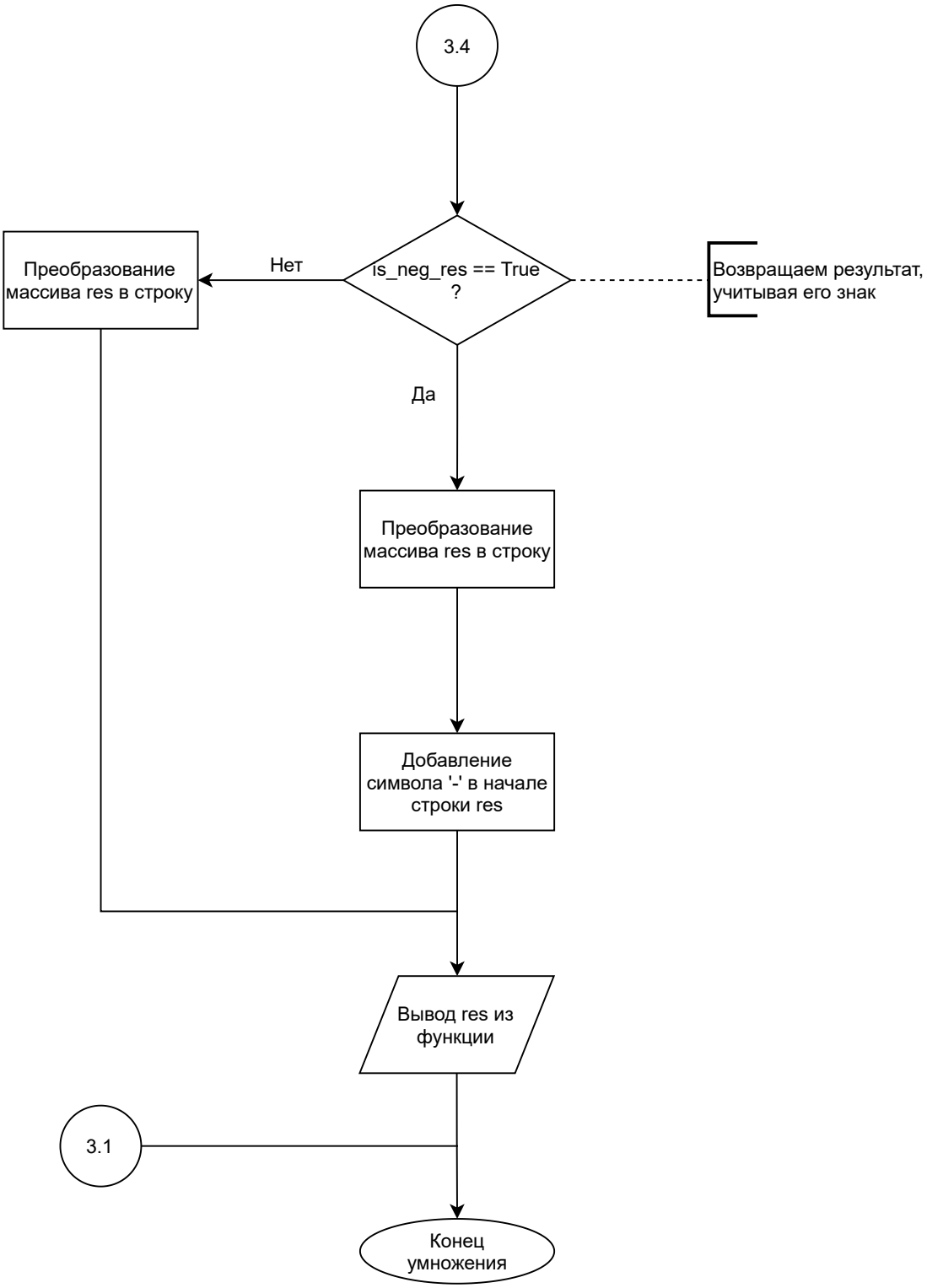

2.3
res[length - 1 - i] += 10 + sign * (b[i] - a[i])
res[length - 2 - i] = res[length - 1 - i] // 10
res[length - 1 - i] = res[length - 1 - i]%10
i >= length - 1
a[length - 1] += bool(length - 1 < self_len) * x[0]
b[length - 1] += bool(length - 1 < self_other) * y[0]
Выполняем операцию с последним разрядом
res[0] += sign * (b[length - 1] - a[length - 1])
Записываем в строку последний разряд
2.4

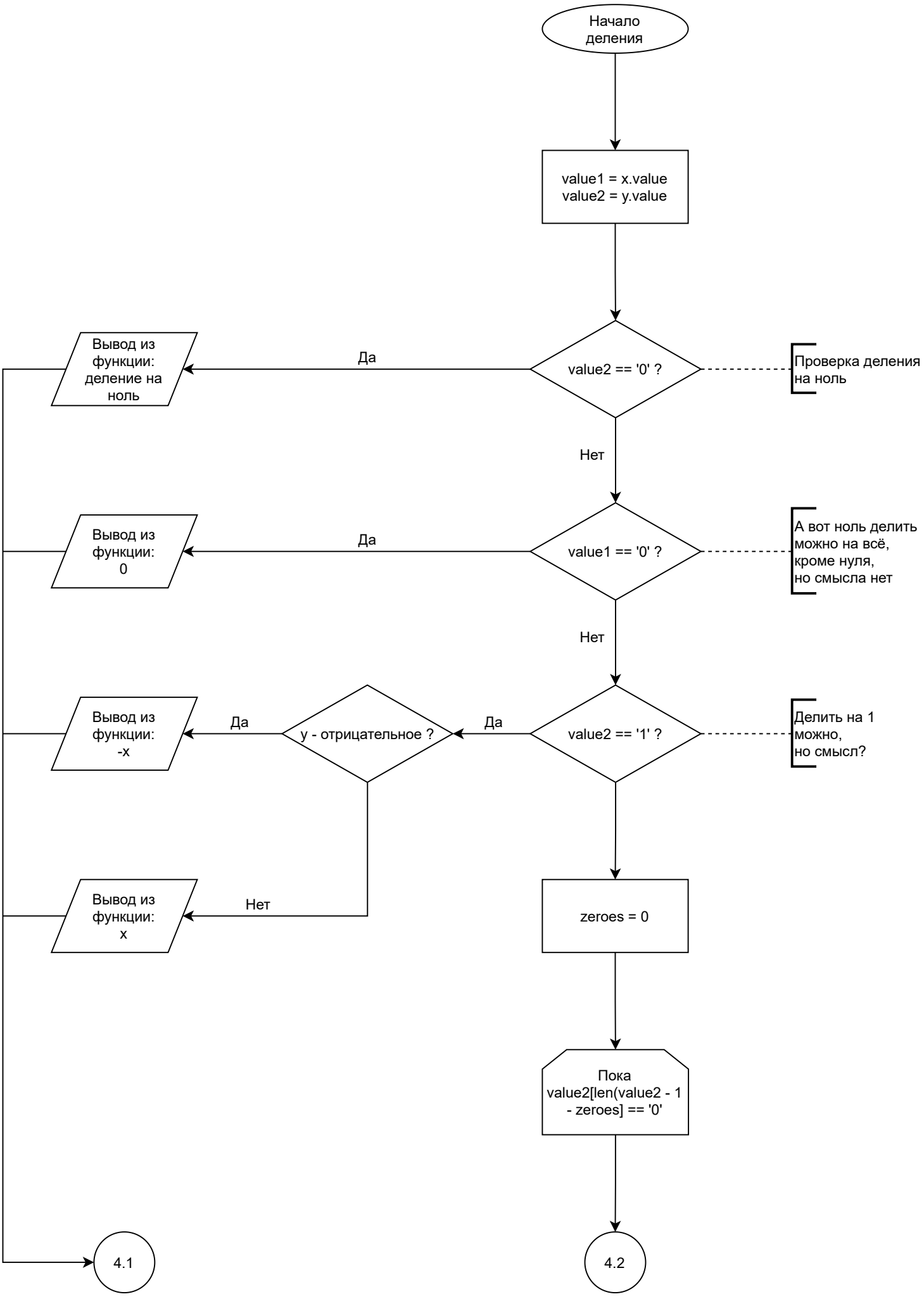

2.4
is_neg_res == true ?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
2.1
Конец вычитания

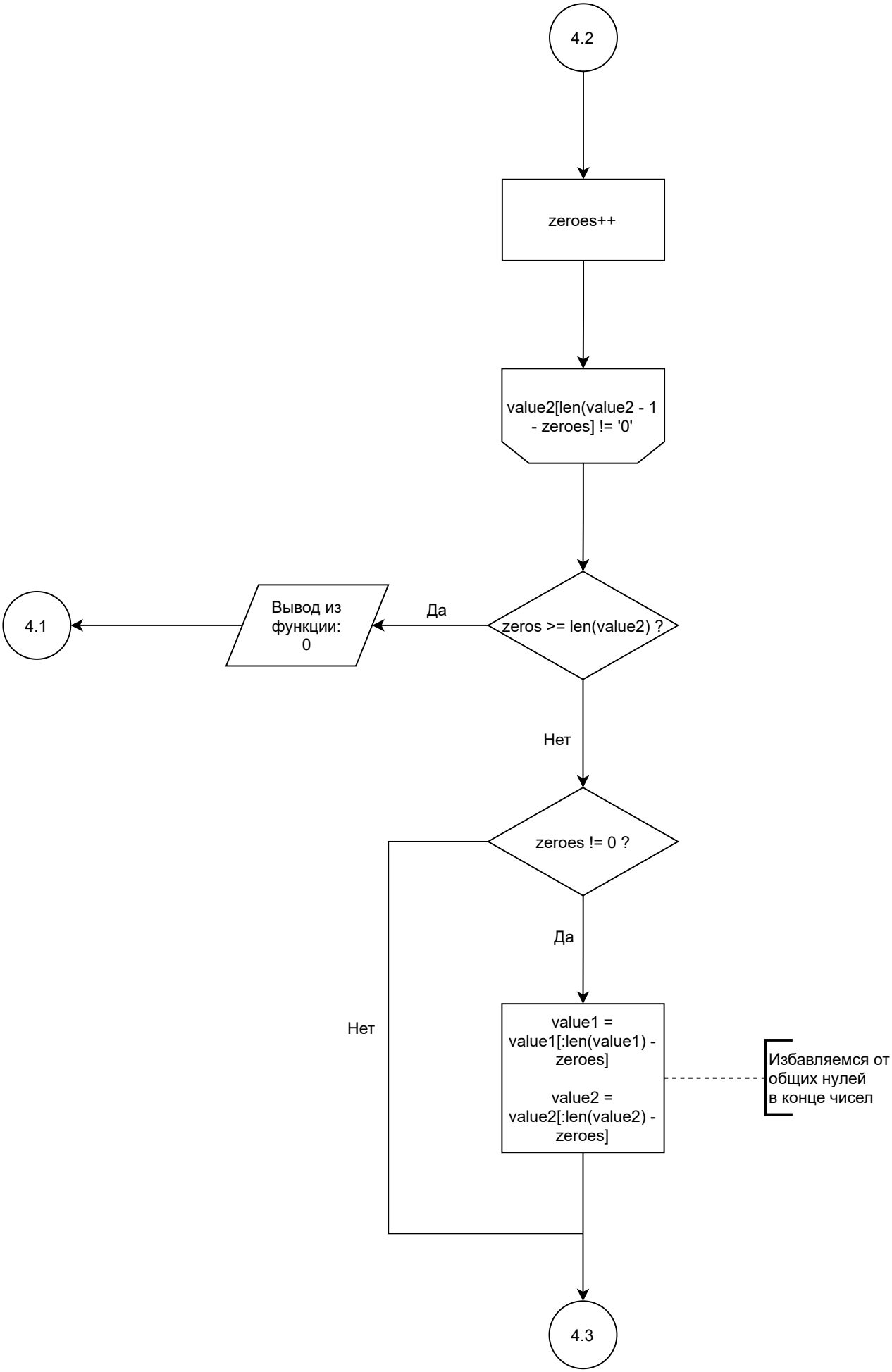

Начало умножения
x или y равны 0 ?
Да
Вывод из функции: 0
Если один из множителей равен нулю, то результат равен нулю
Нет
self_len = len(x.value) other_len = len(y.value)
length = self_len+other_len+1
Результат влезет в сумму длин + 1 из-за возможного переноса
is_neg_res = bool(x > y)
Флаг отрицательности результата - отрицательный, если числа разных знаков
lenght < 10 ?
Да
res = x * y
Вывод из функции: res
Число небольшое, они влезут в стандартные типы
Нет
Зададим массив res, a и b из нулей длинной lenght
3.1
3.2

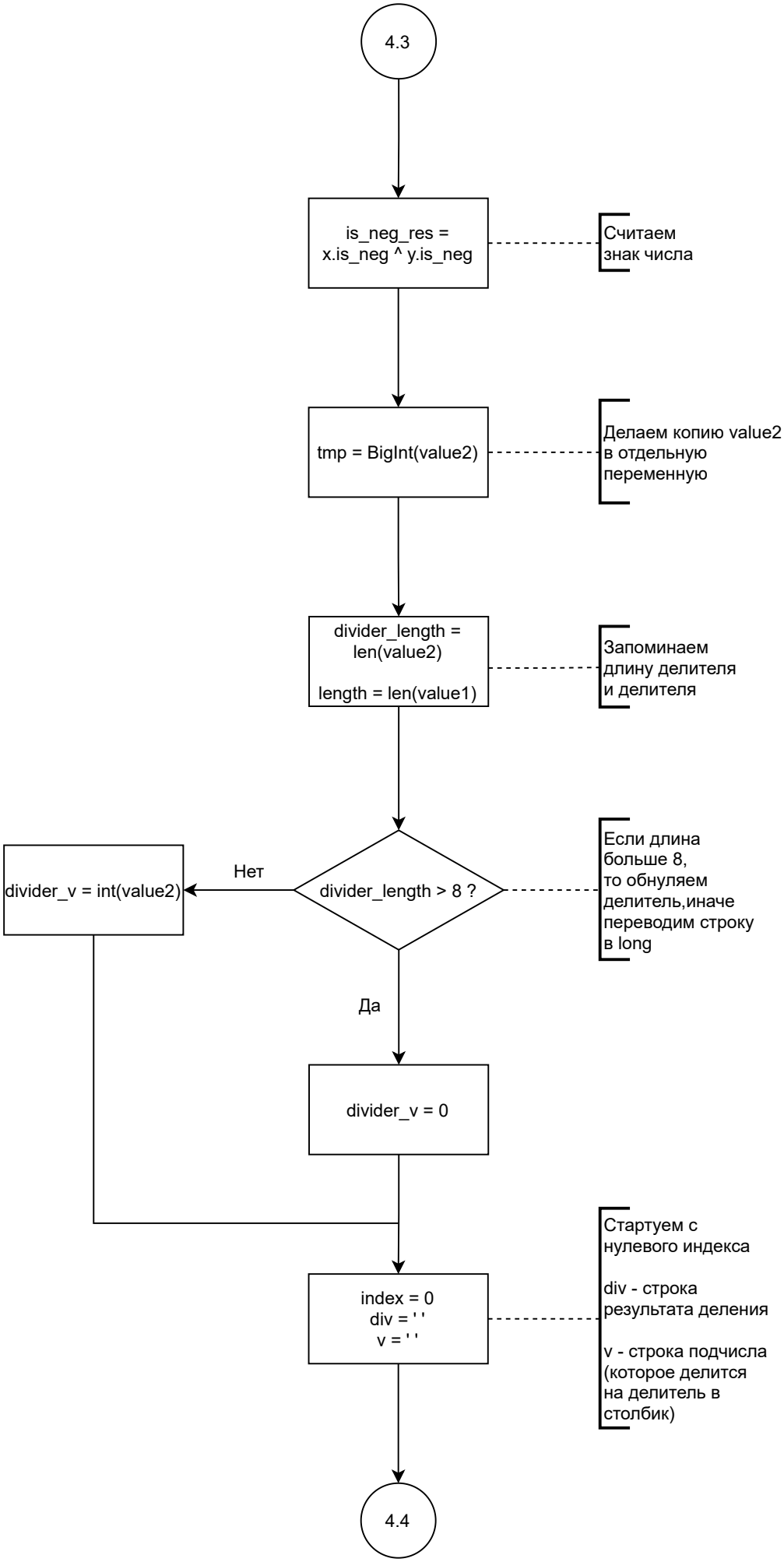

3.2
i = 0
Пока i < length
i++
Заполняем массивы
инверсной записью
чисел
(с ведущими нулями)
i < self_len ?
Нет
a[i] = 0
Да
a[i] = x.value
[self_len - 1 - i]
i < self_other ?
Нет
b[i] = 0
Да
b[i] =
y.value
[other_len - 1 - i]
i >= length
3.3

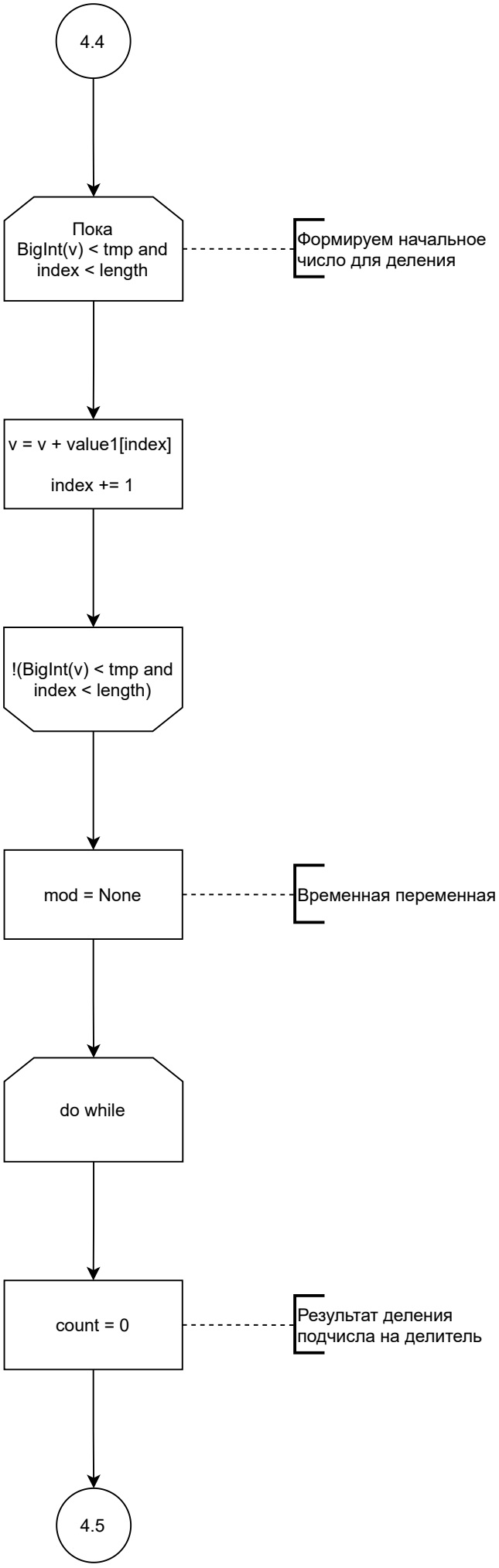

3.3
i = 0
Пока i < self_len
i++
Выполняем умножение "в столбик"
i = 0
Пока i < other_len
i++
res[length - 1 - (i + j)] += a[i] * b[j]
res[length - 1 - (i + j + 1)] += res[length - 1 - (i + j)] // 10
res[length - 1 - (i + j)] %= 10
i >= self_len
i >= self_other_len
3.4

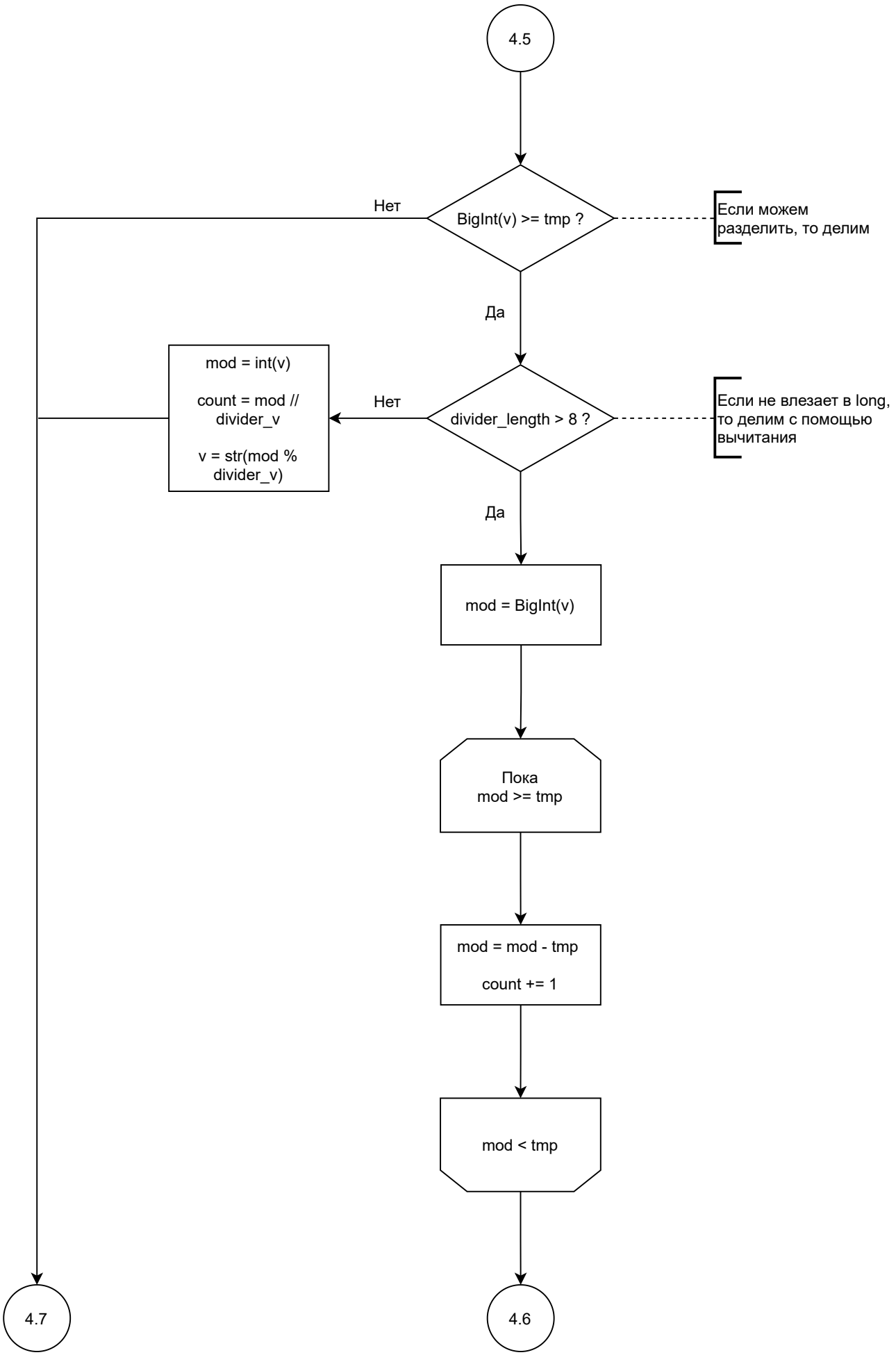

3.4
Преобразование массива res в строку
Нет
Is_neg_res == True ?
Возвращаем результат, учитывая его знак
Да
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
3.1
Конец умножения

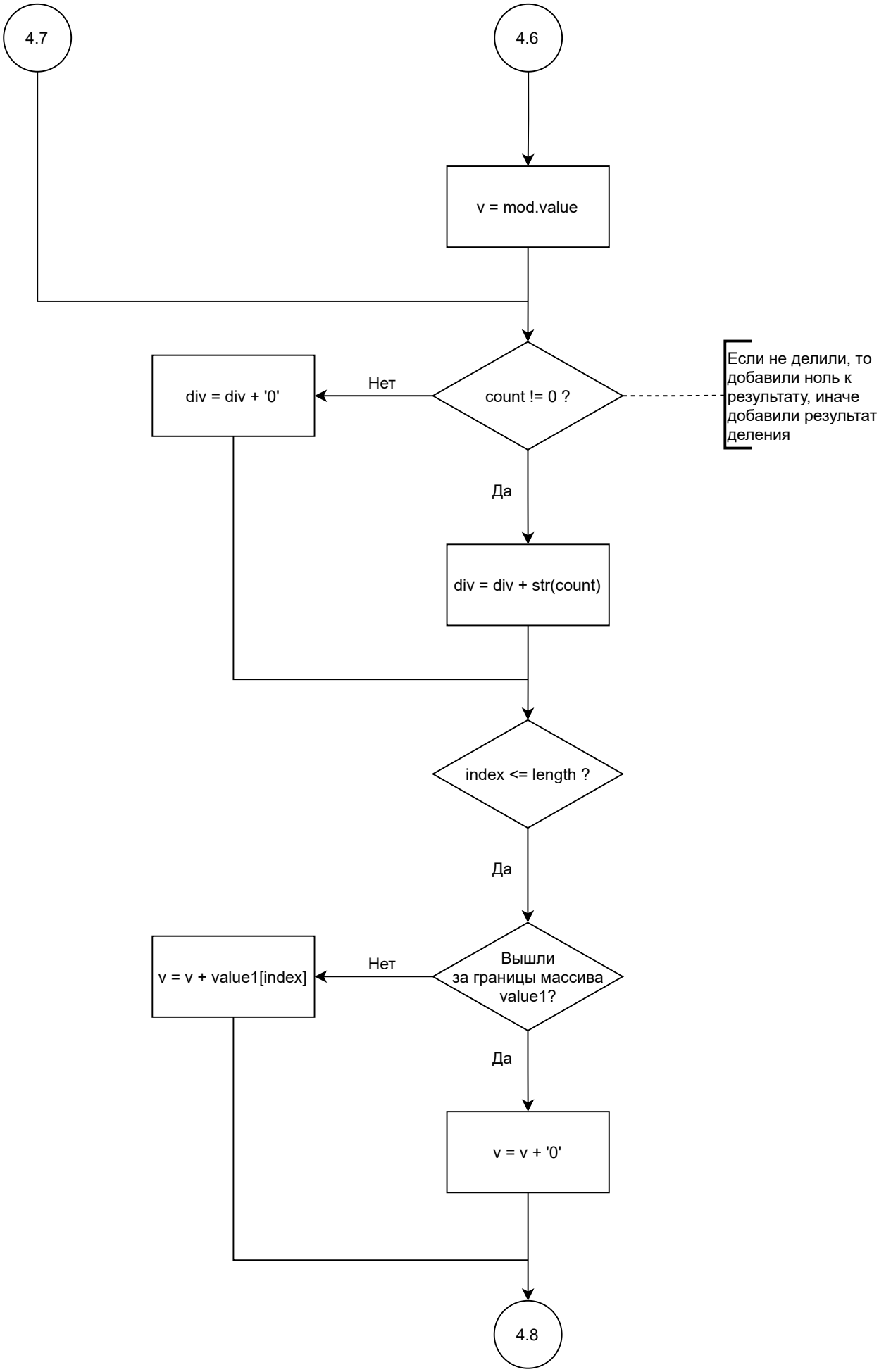

Начало деления
value1 = x.value
value2 = y.value
value2 == '0' ?
Да
Вывод из функции: деление на ноль
Проверка деления на ноль
Нет
value1 == '0' ?
Да
Вывод из функции: 0
А вот ноль делить можно на всё, кроме нуля, но смысла нет
Нет
value2 == '1' ?
Да
y - отрицательное ?
Да
Вывод из функции: -x
Делить на 1 можно, но смысл?
Нет
Вывод из функции: x
zeroes = 0
Пока value2[len(value2 - 1 - zeroes] == '0'
4.1
4.2

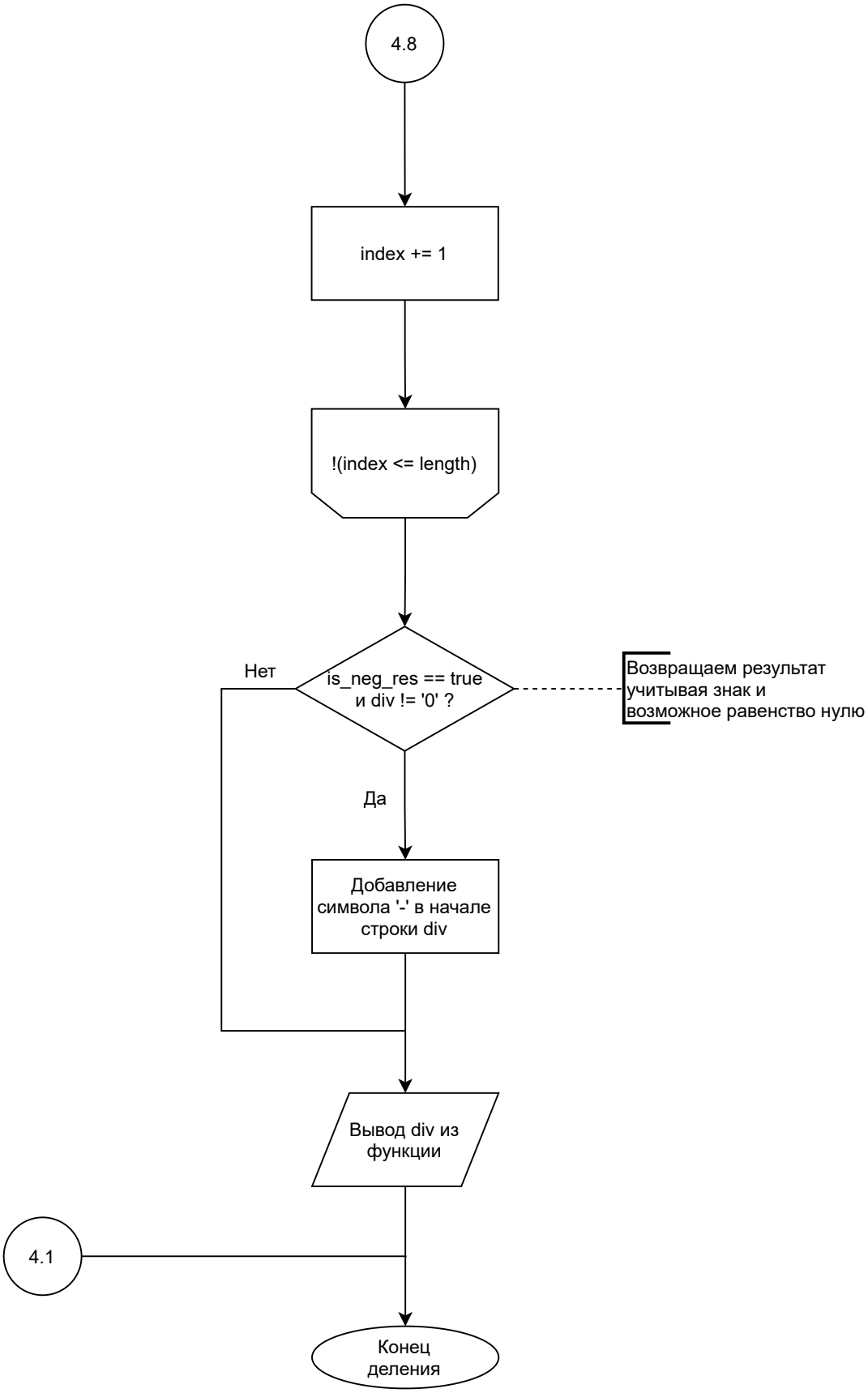

4.2
zeroes++
value2[len(value2 - 1 - zeroes] != '0'
zeros >= len(value2) ?
Да
Вывод из функции: 0
4.1
Нет
zeroes != 0 ?
Да
Нет
value1 = value1[:len(value1) - zeroes]
value2 = value2[:len(value2) - zeroes]
Избавляемся от общих нулей в конце чисел
4.3


4.3
is_neg_res =
x.is_neg ^ y.is_neg
Считаем
знак числа
tmp = BigInt(value2)
Делаем копию value2
в отдельную
переменную
divider_length =
len(value2)
length = len(value1)
Запоминаем
длину делителя
и делителя
divider_length > 8 ?
Нет
divider_v = int(value2)
Если длина
больше 8,
то обнуляем
делитель,иначе
переводим строку
в long
Да
divider_v = 0
index = 0
div = ' '
v = ' '
Стартуем с
нулевого индекса
div - строка
результата деления
v - строка подчисла
(которое делится
на делитель в
столбик)
4.4


4.4
Пока
BigInt(v) < tmp and
index < length
Формируем начальное
число для деления
v = v + value1[index]
index += 1
!(BigInt(v) < tmp and
index < length)
mod = None
Временная переменная
do while
count = 0
Результат деления
подчисла на делитель
4.5


4.5
BigInt(v) >= tmp ?
Нет
Если можем разделить, то делим
Да
divider_length > 8 ?
Нет
Если не влезает в long, то делим с помощью вычитания
mod = int(v)
count = mod // divider_v
v = str(mod % divider_v)
Да
mod = BigInt(v)
Пока mod >= tmp
mod = mod - tmp
count += 1
mod < tmp
4.7
4.6


4.7
4.6
v = mod.value
div = div + '0'
Нет
count != 0 ?
Если не делили, то добавили ноль к результату, иначе добавили результат деления
Да
div = div + str(count)
index <= length ?
Да
v = v + value1[index]
Нет
Вышли за границы массива value1?
Да
v = v + '0'
4.8


4.8
index += 1
!(index <= length)
Нет
is_neg_res == true
и div != '0' ?
Возвращаем результат
учитывая знак и
возможное равенство нулю
Да
Добавление
символа '-' в начале
строки div
Вывод div из
функции
4.1
Конец
деления

7 Исходный код программы

```
# -*- coding: utf-8 -*-

import time
from sys import setrecursionlimit

setrecursionlimit(1500) # Максимальный стек рекурсии


class BigInt(object):
    is_neg = False # Флаг отрицательности числа
    value = ''# Число в виде стоки

    def __init__(self, x=0):
        self.value = '0'
        # Если в конструктор передано целое число
        if isinstance(x, int):
            self.is_neg = x < 0
            self.value = str(x if x >= 0 else -x)
        # Если в конструктор передана строка
        elif isinstance(x, str):
            # Если вдруг пришла пустая строка, пропускаем, оставляем value = 0
            if len(x):
                self.is_neg = x[0] == '-'
                # Значением будет все, после минуса, если он был. И убираем ведущие нули, если они были
                self.value = x[self.is_neg:].lstrip('0')
                # Проверяем, является ли строка числом, если нет, value = 0
                if not self.value.isdigit():
                    self.value = '0'
                if self.value == '0':
                    self.is_neg = False
        # Если в конструктор передан экземпляр того же класса, копируем его содержимое
        elif isinstance(x, BigInt):
            self.value = x.value
            self.is_neg = x.is_neg

    # Является ли число четным
    def is_even(self):
        return not (int(self.value[-1]) \& 1)
```

```
39
40      # Перегрузка числа по модулю
41      def __abs__(self):
42          return BigInt(self.value)
43
44      def bipow(self, n):
45          # Любое число в степени 0 = 1
46          if n < 0:
47              return None
48          if not n:
49              return BigInt(1)
50          b = bin(n)[2:]
51          res = self
52          for i in range(1, len(b)):
53              res = res * res
54              if b[i] == '1':
55                  res = res * self
56          return res
57
58      def birt(self, n):
59          # Корень извлекать можем только из положительного числа
60          if (n < 0) or self.is_neg:
61              return None
62          if n == 1:
63              return self
64          length = (len(self.value) + 1) // 2
65          index = 0
66          v = [0] * length
67          while index < length:
68              v[index] = 9
69              while BigInt(''.join(str(x) for x in v)).bipow(n) > self and v[
                    index]:
70                  v[index] -= 1
71              index += 1
72          v = ''.join(str(x) for x in v).lstrip('0')
73          return BigInt('-' + v) if self.is_neg else BigInt(v)
74
75      # Перегрузка перевода в bool
76      def __bool__(self):
77          return self.value != '0'
78
```

```
79      # Перегрузка x < y
80      def __lt__(self, other):
81          if isinstance(other, int):
82              other = BigInt(other)
83          self_len = len(self.value) # Запоминаем длину первого числа
84          self_other = len(other.value) # Запоминаем длину второго числа
85          # Если знаки одинаковые, то проверяем значения
86          if self.is_neg == other.is_neg:
87              # Если длины не равны
88              if self_len != self_other:
89                  # Меньше число с меньшей длинной для положительных и с
                        большей длиной для отрицательных
90                  return (self_len < self_other) ^ self.is_neg
91              i = 0
92              # Ищем разряд, в котором значения отличаются
93              while (i < self_len and self.value[i] == other.value[i]):
94                  i += 1
95              # Если разряд найден, то меньше число с меньшей цифрой для
                    положительных и с большей цифрой для отрицательных, иначе
                    числа равны
96              return (i < self_len) and ((self.value[i] < other.value[i]) ^
                    self.is_neg)
97          return self.is_neg # Знаки разные, если число отрицательное, то оно
                меньше, если положительное, то больше
98
99      # Перегрузка x <= y
100     def __le__(self, other):
101         return self < other or self == other
102
103     # Перегрузка x == y
104     def __eq__(self, other):
105         if isinstance(other, int):
106             other = BigInt(other)
107         return (self.value == other.value) and (self.is_neg == other.is_neg)
108
109     # Перегрузка x != y
110     def __ne__(self, other):
111         return not self == other
112
113     # Перегрузка x > y
114     def __gt__(self, other):
115         return not (self < other or self == other)
116
```

```
117     # Перегрузка x >= y
118     def __ge__(self, other):
119         return self > other or self == other
120 
121     # Унарный плюс (просто копируем значение числа)
122     def __pos__(self):
123         return self
124 
125     # Унарный минус
126     def __neg__(self):
127         if self == 0:
128             return self
129         return BigInt(self.value if self.is_neg else '-' + self.value)
130 
131     # Число в бинарный вид (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
132     def to_bin(self):
133         return bin(int(self.value))
134 
135     # Битовый сдвиг вправо (x » y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
136     def __rshift__(self, n):
137         if n < 0:
138             raise ValueError
139         self_bin = self.to_bin()
140         if n >= len(self_bin) - 2 or (self == 0):
141             return BigInt(0)
142         return BigInt(int(self_bin[:len(self_bin) - n], 2))
143 
144     # Битовый сдвиг вслево (x « y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
145     def __lshift__(self, n):
146         if n < 0:
147             raise ValueError
148         if self == 0:
149             return BigInt(0)
150         self_bin = self.to_bin()
151         return BigInt(int(self_bin + ('0' * n), 2))
152 
153     # Побитовое И (x & y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
154     def __and__(self, other):
155         if isinstance(other, int):
156             other = BigInt(other)
157         self_bin = self.to_bin()[2:]
```

```
158         other_bin = other.to_bin()[2:]
159         self_len = len(self_bin)
160         other_len = len(other_bin)
161         if self_len > other_len:
162             other_bin = other_bin.zfill(self_len)
163         elif self_len < other_len:
164             self_bin = self_bin.zfill(other_len)
165         res = int('0b' + ''.join(['1' if (x, y) == ('1', '1') else '0' for x,
                y in zip(self_bin, other_bin)]), 2)
166         return BigInt(res)
167 
168     # Побитовое ИЛИ (x | y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
169     def __or__(self, other):
170         if isinstance(other, int):
171             other = BigInt(other)
172         self_bin = self.to_bin()[2:]
173         other_bin = other.to_bin()[2:]
174         self_len = len(self_bin)
175         other_len = len(other_bin)
176         if self_len > other_len:
177             other_bin = other_bin.zfill(self_len)
178         elif self_len < other_len:
179             self_bin = self_bin.zfill(other_len)
180         res = int('0b' + ''.join(['0' if (x, y) == ('0', '0') else '1' for x,
                y in zip(self_bin, other_bin)]), 2)
181         return BigInt(res)
182 
183     # Возврат копии
184     def copy(self):
185         return BigInt(('-' if self.is_neg else '') + self.value)
186 
187     # Сложение двух чисел
188     def __add__(self, other):
189         if isinstance(other, int):
190             other = BigInt(other)
191         if self == 0:
192             return other
193         if other == 0:
194             return self
195         # Если знаки одинаковые, то выполняем сложение
196         if other.is_neg == self.is_neg:
```

```
197             num2 = other.value # Запоминаем значение второго операнда
198             self_len = len(self.value) # Длинна первого операнда
199             other_len = len(num2) # Длинна второго операнда
200             # Длина суммы равна максимуму из двух длин + 1 из-за возможного
                    переноса разряда
201             length = max(self_len, other_len)
202             res = [0] * (length + 1)
203             for i in range(length):
204                 j = length - i
205                 # Выполняем сложение разрядов
206                 res[j] += int((num2[other_len - 1 - i] if i < other_len else '
                        0')) + int((self.value[self_len - 1 - i] if i < self_len
                        else '0'))
207                 res[j - 1] = res[j] // 10 # Выполняем перенос в следующий
                        разряд, если он был
208                 res[j] = res[j] % 10 # Оставляем только единицы от возможного
                        переноса и превращаем символ в цифру
209                 # Возвращаем результат, учитывая его знак
210             return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for x
                     in res))
211         # Если одно из чисел отрицательное, а другое положительное,
                отправляем на вычитание, меняя знак
212         return (self - (-BigInt(other))) if self.is_neg else (other - (-
                BigInt(self)))
213
214     # Вычитание одного числа из другого
215     def __sub__(self, other):
216         if isinstance(other, int):
217             other = BigInt(other)
218         # Если числа равны, считать не нужно
219         if self == other:
220             return BigInt(0)
221         if self == 0:
222             return -other
223         if other == 0:
224             return self
225         # Если оба числа положительные, выполняем вычитание
226         if not self.is_neg and not other.is_neg:
227             self_len = len(self.value) # Запоминаем длину первого числа
228             self_other = len(other.value) # Запоминаем длину второго числа
229             length = max(self_len, self_other) - 1 # Длина результата не
                    превысит максимума длин чисел
230             is_neg_res = other > self # Определяем знак результата
231             # Массивы аргументов
```

```
232             new_length = length + 1
233             a = [0] * new_length
234             b = [0] * new_length
235             res = [0] * new_length
236             sign = 2 * is_neg_res - 1 # Получаем числовое значение знака
                    результата
237             for i in range(length):
238                 a[i] += int(self.value[self_len - 1 - i]) if i < self_len
                        else 0 # Формируем
                        разряды
239                 b[i] += int(other.value[self_other - 1 - i]) if i <
                        self_other else 0 # Из строк
                        аргументов
240                 b[i + 1] = -is_neg_res # В зависимости от знака занимаем или
                        не занимаем
241                 a[i + 1] = is_neg_res - 1 # 10 у следующего разряда
242                 res[length - i] += 10 + sign * (b[i] - a[i])
243                 res[length - 1 - i] = res[length - i] // 10
244                 res[length - i] = res[length - i] % 10
245             # Выполняем операцию с последним разрядом
246             a[length] += (length < self_len) * int(self.value[0])
247             b[length] += (length < self_other) * int(other.value[0])
248             # Записываем в строку последний разряд
249             res[0] += sign * (b[length] - a[length])
250             # Возвращаем результат, учитывая его знак
251             return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
                    in res))
252         return -BigInt(other) - (-BigInt(self)) if self.is_neg and other.
                is_neg else self + -BigInt(other)
253
254     # Умножение двух чисел
255     def __mul__(self, other):
256         if isinstance(other, int):
257             other = BigInt(other)
258         # Если один из множителей равен нулю, то результат равен нулю
259         if self.value == '0' or other.value == '0':
260             return BigInt(0)
261         self_len = len(self.value) # Запоминаем длину первого числа
262         other_len = len(other.value) # Запоминаем длину второго числа
263         length = self_len + other_len # Результат влезет в сумму длин + 1
                из-за возможного переноса
264         # Флаг отрицательности результата - отрицательный, если числа разных
                знаков
265         is_neg_res = self.is_neg ^ other.is_neg
266         if length < 10: # Число небольшое, можно по нормальному
```

```
267             res = int(self.value) * int(other.value)
268             return BigInt(-res if is_neg_res else res)
269         else: # Умножаем в столбик
270             # Массивы аргументов
271             new_length = length + 1
272             a = [0] * new_length
273             b = [0] * new_length
274             res = [0] * new_length
275             # Заполняем массивы инверсной записью чисел (с ведущими нулями)
276             for i in range(new_length):
277                 a[i] = int(self.value[self_len - 1 - i]) if i < self_len else
                        0
278                 b[i] = int(other.value[other_len - 1 - i]) if i < other_len
                       else 0
279             # Выполняем умножение "в столбик"
280             for i in range(self_len):
281                 for j in range(other_len):
282                     res[length - (i + j)] += a[i] * b[j]
283                     res[length - (i + j + 1)] += res[length - (i + j)] // 10
284                     res[length - (i + j)] %= 10
285             # Возвращаем результат, учитывая его знак
286             return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
                   in res))
287
288     # Деление одного числа на другое
289     def __truediv__(self, other):
290         if isinstance(other, int):
291             other = BigInt(other)
292         value1 = self.value # Запоминаем значение первого числа
293         value2 = other.value # Запоминаем значение второго числа
294         if value2 == '0':
295             raise ZeroDivisionError # Нельзя делить на ноль
296         if value1 == '0':
297             return BigInt(0) # А вот ноль делить можно на всё, кроме нуля, но
                   смысл
298         if value2 == '1':
299             return -BigInt(self) if other.is_neg else BigInt(self) # Делить
                   на 1 можно, но смысл?
300         zeroes = 0
301         while value2[len(value2) - 1 - zeroes] == '0':
302             zeroes += 1
303         if zeroes >= len(value1):
304             return BigInt(0)
```

```
305         # если у нас 13698 / 1000, то мы можем делить 13 / 1
306         if zeroes:
307             value1 = value1[:len(value1) - zeroes]
308             value2 = value2[:len(value2) - zeroes]
309         is_neg_res = self.is_neg ^ other.is_neg # Считаем знак числа
310         tmp = BigInt(value2)
311         divider_length = len(value2) # Запоминаем длину делителя
312         # Если длина больше 8, то обнуляем делитель, иначе переводим строку
              в long
313         # Можно не обнулять, но мы думаем, что Python не умеет в большие
              числа
314         divider_v = 0 if divider_length > 8 else int(value2)
315         length = len(value1) # Получаем длину делимого
316         index = 0 # Стартуем с нулевого индекса
317         div = ''# Строка результата деления
318         v = ''# Строка подчисла (которое делится на делитель в столбик)
319         index = len(value2)
320         v = value1[:index]
321         mod = None
322         while True:
323             count = 0 # Результат деления подчисла на делитель
324             # Если можем разделить, то делим
325             if BigInt(v) >= tmp:
326                 # Если не входит в long, то делим с помощью вычитания
327                 if divider_length > 8:
328                     mod = BigInt(v)
329                     while mod >= tmp:
330                         mod = (mod - tmp).copy()
331                         count += 1
332                     v = mod.value
333                 else:
334                     mod = int(v)
335                     count = mod // divider_v
336                     v = str(mod % divider_v)
337             # Если не делили, то добавили ноль к результату, иначе добавили
                  результат деления
338             div = div + (str(count) if count else '0')
339             if index <= length:
340                 try: # Тот самый ноль, лучше не спрашивать
341                     v = v + value1[index]
342                 except IndexError:
343                     v = v + '0'
344                 index += 1 # Формируем новое значение для подчисла
```

```
345             if not (index <= length):
346                 break
347         # Возвращаем результат учитывая знак и возможное равенство нулю
348         return BigInt('-' + div if is_neg_res and div != '0' else div)
349 
350     # Обработка для выходных данных
351     def __str__(self):
352         return str('-' if self.is_neg else '') + self.value
353 
354     # Остаток от деления
355     def __mod__(self, other):
356         if isinstance(other, int):
357             other = BigInt(other)
358         if other.value == '0':
359             return None
360         if self.value == '0' or other.value == "1":
361             return BigInt(0)
362         # Если числа меньше 9, можно посчитать по нормальному
363         if len(self.value) < 9 and len(other.value) < 9:
364             return BigInt(int(str('-' if self.is_neg else '') + self.value) %
                      int(str('-' if other.is_neg else '') + other.value))
365         tmp = BigInt(other.value)
366         divider_length = len(other.value) # запоминаем длину делителя
367         # Если длина больше 8, то обнуляем long'овый делитель, иначе
              переводим строку в long
368         divider_v = 0 if divider_length > 8 else int(other.value)
369         length = len(self.value)
370         index = 0
371         mod2 = self.copy()
372         v = ''
373         mod = None
374         while BigInt(v) < tmp and index < length:
375             v = v + self.value[index]
376             index += 1
377         while True:
378             if BigInt(v) >= tmp:
379                 if divider_v:
380                     v = str(int(v) % divider_v)
381                 else:
382                     mod = BigInt(v)
383                     while mod >= tmp:
384                         mod = (mod - tmp).copy()
```

```
                    v = mod.value
            if index <= length:
                mod2 = v
                try:
                    v = v + self.value[index]
                except IndexError:
                    break
                index += 1
            if not (index <= length):
                break
        if isinstance(mod2, BigInt):
            if mod2.value == '0':
                return BigInt(0)
        res = -BigInt(mod2) if self.is_neg else BigInt(mod2)
        if self.is_neg ^ other.is_neg and res != 0:
            return other + res
        return res


def GCD(a, b):
    if a < 0:
        a = -a
    if b < 0:
        b = -b
    while b:
        a, b = b, a % b
    return a


def binary_GCD(num1, num2):
    if num1 < 0:
        num1 = -num1
    if num2 < 0:
        num2 = -num2
    shift = 0
    # Если одно из чисел равно нулю, делитель - другое число
    if num1 == 0:
        return num2
    if num2 == 0:
        return num1
    # Если num1 = 1010, а num2 = 0100, то num1 | num2 = 1110
```

```
426     # 1110 & 0001 == 0, тогда происходит сдвиг, который фиксируется в shift
427     while (num1 | num2) \& 1 == 0:
428         shift += 1
429         num1 = num1 >> 1
430         num2 = num2 >> 1
431     # Если True, значит num1 - четное, иначе - нечетное
432     while num1 \& 1 == 0:
433         # если нечетное, сдвигаем на один бит
434         num1 = num1 >> 1
435     while num2 != 0:
436         # пока число нечётное, сдвигаем на один бит
437         while num2 \& 1 == 0:
438             num2 = num2 >> 1
439         # если первое число больше второго
440         if num1 > num2:
441             # меняем их местами
442             num1, num2 = num2, num1
443         # теперь первое число меньше второго, вычитаем
444         num2 = num2 - num1
445     # возвращаем число, перед этим сдвинув его биты на shift
446     return num1 << shift
447
448
449 if __name__ == '__main__':
450     while True:
451         menu_text = '\n'.join([
452             'Выберитедействие:',
453             '1) x + y',
454             '2) x - y',
455             '3) x * y',
456             '4) x / y',
457             'q) Выход'
458         ])
459         print(menu_text)
460         choice = input()
461         if choice == '1':
462             x = BigInt(input('Введите первоечисло(x): '))
463             y = BigInt(input('Введите второечисло(y): '))
464             t = time.time()
465             res = x + y
466             print(f'\nВремя расчета: {time.time() - t} сек.')
```

```
467             print('x + y =', res, '\n\n')
468         elif choice == '2':
469             x = BigInt(input('Введите первоечисло(x): '))
470             y = BigInt(input('Введите второечисло(y): '))
471             t = time.time()
472             res = x - y
473             print(f'\nВремя расчета: {time.time() - t} сек.')
474             print('x - y =', res, '\n\n')
475         elif choice == '3':
476             x = BigInt(input('Введите первоечисло(x): '))
477             y = BigInt(input('Введите второечисло(y): '))
478             t = time.time()
479             res = x * y
480             print(f'\nВремя расчета: {time.time() - t} сек.')
481             print('x * y =', res, '\n\n')
482         elif choice == '4':
483             x = BigInt(input('Введите первоечисло(x): '))
484             y = BigInt(input('Введите второечисло(y): '))
485             t = time.time()
486             res = x / y
487             print(f'\nВремя расчета: {time.time() - t} сек.')
488             print('x / y =', res, '\n\n')
489         else:
490             exit()
```

Список литературы

1. *Акритас А.* Основы компьютерной алгебры с приложениями. — М. : МИР, 1994.
2. *Майрова С. П.*, *Завгородний М. Г.* Программирование. Криптографические алгоритмы: учебное пособие. — Воронеж : Издательский дом ВГУ, 2018.
3. *Саммерфилд М.* Программирование на Python 3. Подробное руководство. — Символ-Плюс, 2009.