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 2020

Содержание

1 Постановка задачи

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

2. Составить алгоритм и написать соответствующую ему программу, позволяющую возводить целое число в квадрат;

3. Составить алгоритм и написать соответствующую ему программу, позволяющую возводить натуральное число в натуральную степень;

4. Составить алгоритм и написать соответствующую ему программу, позволяющую вычислить целую часть квадратного корня из натурального числа;

5. Составить алгоритм и написать соответствующую ему программу, позволяющую вычислить целую часть кубического корня из натурального числа;

6. Используя один из предложенных выше алгоритмов, составить блок схему и написать соответствующую ей программу, позволяющую вычислять наибольший общий делитель двух больших натуральных чисел.

2 Используемые инструменты

Для решения вышеуказанных задач были использованы следующие инструменты:

- Основным ЯП был выбран Python версии 3.8.1;
- Для создания интерфейса был использован фреймворк Qt5, а также его расширение PyQt5;
- Для построение основы интерфейса была использована кроссплатформенная свободная среда для разработки графических интерфейсов программ использующих библиотеку Qt - Qt Designer;
- Для компиляции программы в бинарный файл .exe использован конвертер файлов Auto PY to EXE, который использует для своей работы PyInstaller.

3 Общая структура программы

Условно программу, написанную для решения вышеуказанных задач, можно разделить на две основных логических части:

1. Интерфейс пользователя.
   Содержит в себе логику обработки команд, поступающих от пользователя. Содержит в себе код, отвечающий за разметку элементов интерфейса в окне, а также код, отвечающий за поведение программы, при использовании этих элементов;

2. Библиотека работы целых длинных чисел.
   Содержит в себе обособленную часть кода, которая может быть подключена как отдельная библиотека к любой программе на ЯП Python.

4 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. При этом данный способ не является медленно работающим;

- Вычитание целых длинных чисел.
  Программная реализация представляет собой вычитание чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Умножение целых длинных чисел.
  Программная реализация представляет собой умножение чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Целочисленное деление целых длинных чисел.
  Программная реализация представляет собой деление чисел в «столбик» без дробной части. Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Выделение корня из простого длинного числа любой положительной целой степени.
  Программная реализация представляет подбор наиболее близкого числа, возведенного в данную из аргументов степень, при котором результат возведения в степень не будет превышать число, из которого выделяется корень. Выбор данного способа обусловлен простотой его

реализации, а также отсутствием предполагаемых альтернатив. При этом, скорее всего, альтернативы есть;

- Возведение в степень простого длинного числа.
  Программная реализация представляет умножение данного числа на самого себя, используя рекурсивные вызовы этой же функции. Размер этого повторного умножение равно числу, в степень которого необходимо возвести некоторое число. Выбор данного способа реализации обусловлено желанием опробовать рекурсию на практике.

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
```

Также в данной библиотеке содержится функция «GCD», реализующая возможность нахождения наибольшего общего делителя. Эта функция может работать как с экземплярами класса «BigInt», дак и с численными типами данных ЯП Python.

5 Примеры работы библиотеки

В качестве примера работы будут использоваться прямые вызовы методов класса «BigInt». При этом, при работе с графической программой результаты будут идентичны.

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 нахождение остатка от деления:

```
1 print(b % a)
```

Вывод: 82222222221

- Выполним нахождение НОД:

```
1 print(GCD(b, a))
```

Вывод: -9

- Выполняем возведение в степень:

```
1 print(a.bipow(20))
```

Вывод:
67654945781131788253399139476950939867213847384221510782372183
38736383554932818216005379411615896402318839463975841663187950
47266740645217094738013218419327830527872057771151857381511749
91352856101226216668855950857925749095871686783571452199421977
46524667992025003348862025953101533163689052346013027443912327
30289077240646312505876707771713512612446512064624 01

- 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

- Выполняем возведение в степень ноль:

```
1 print(b.bipow(0))
```

Вывод: 1

6 Руководство пользователя

1. В случае с сложением, вычитанием, умножением и делением программа работает по принципу: [первое число] [действие] [второе число]

2. В случае возведения в степень программа работает по принципу: [первое число] в степени [второе число]

3. В случае извлечения корня ($\sqrt{}$) программа работает по принципу: корень в степени[второе число] по [первое число]

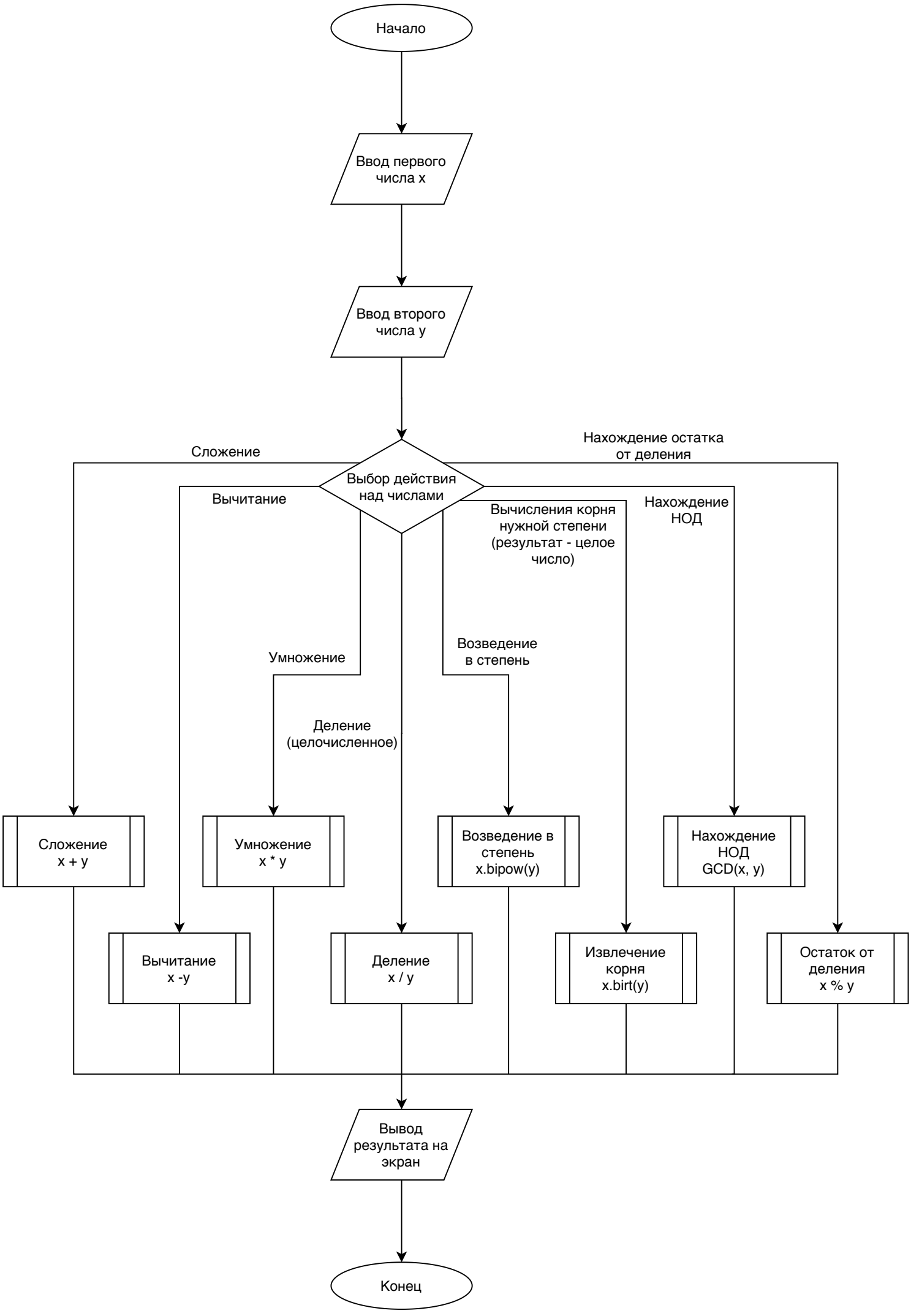
4. В случае нахождения НОД программа ищет наибольший общий делитель чисел.

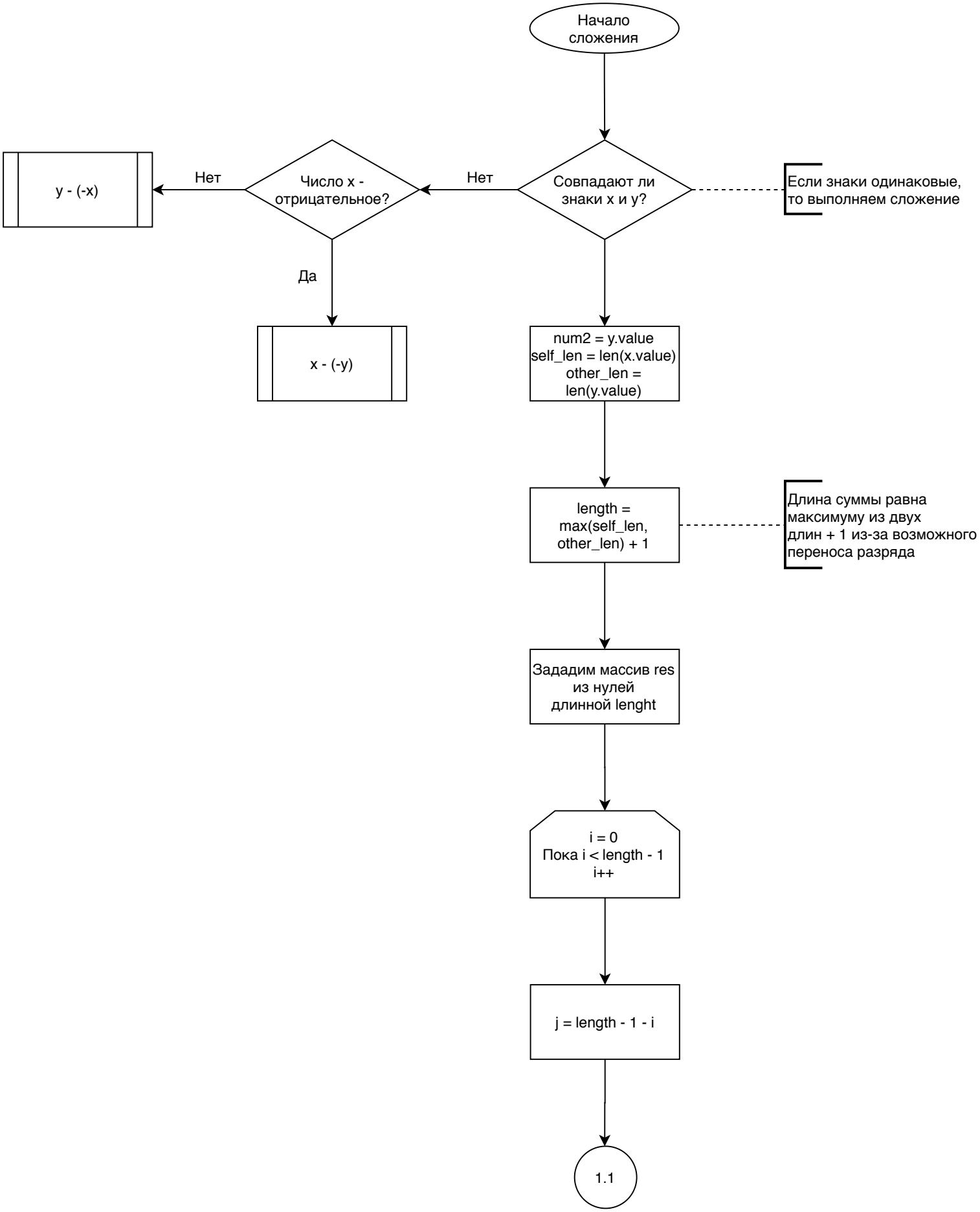
5. В случае нахождения НОД программа ищет остаток от деление первого числа на второе число.

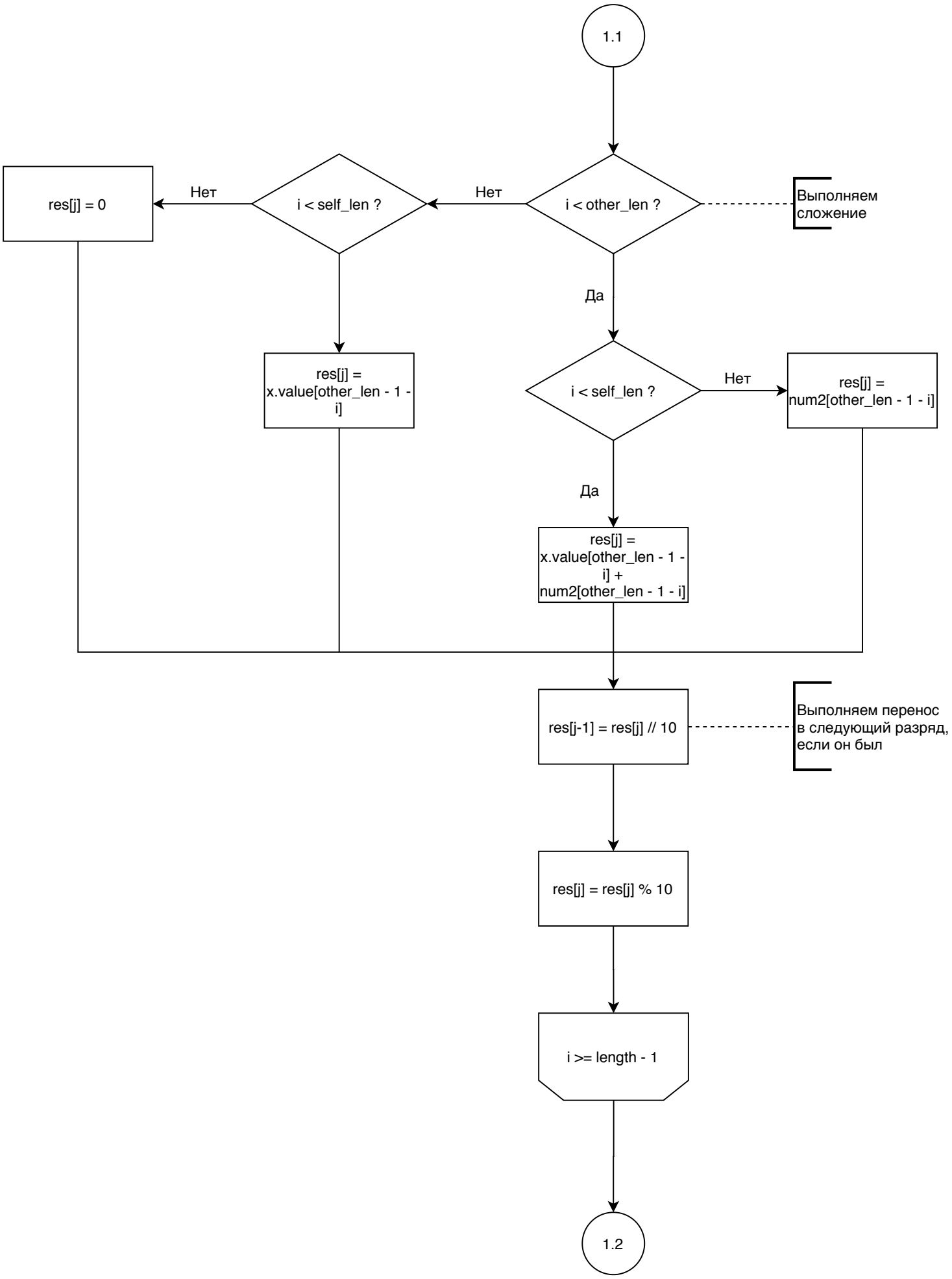
7 Блок-схема методов библиотеки

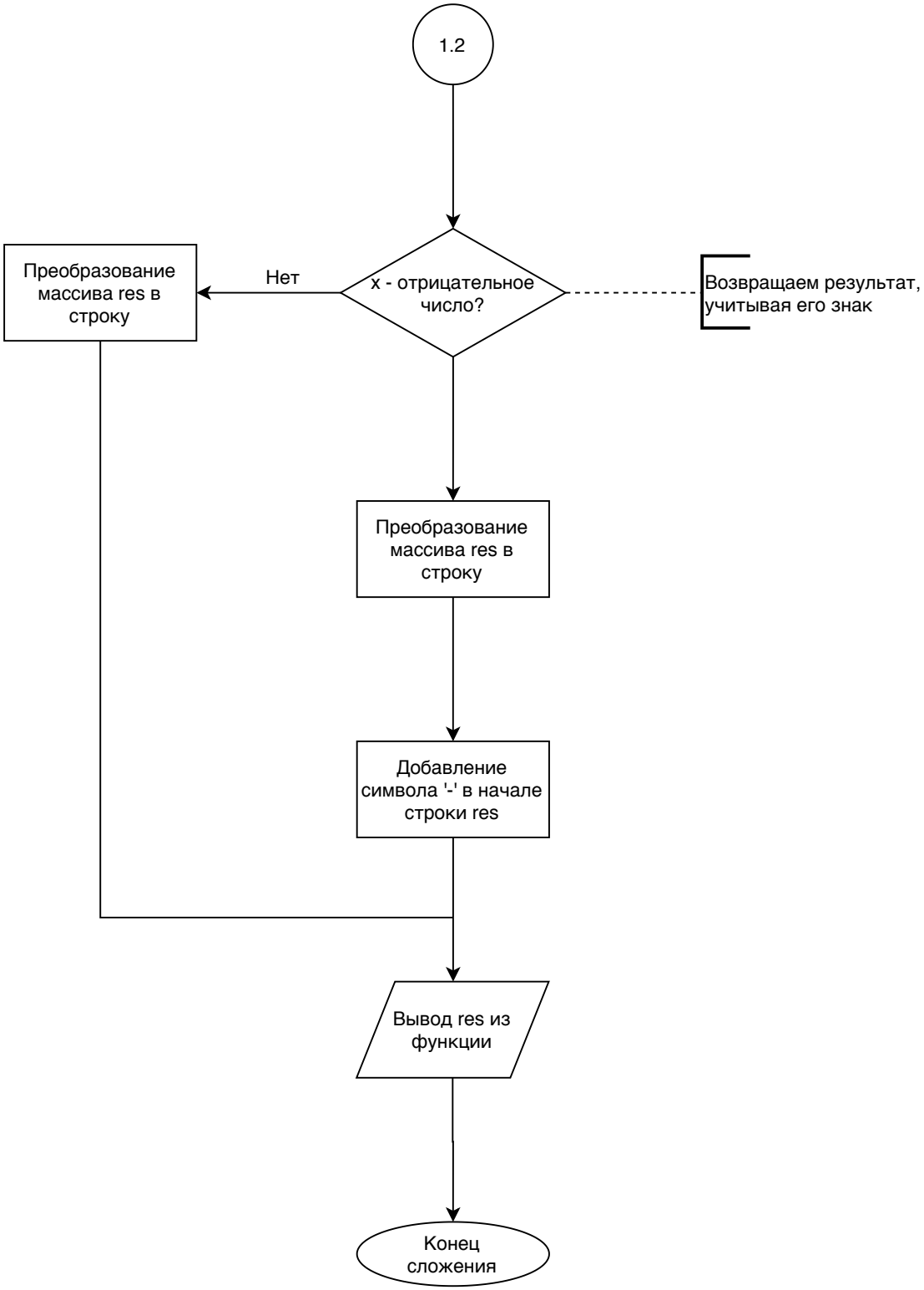
Ниже представлены блок-схемы методов в следующем порядке:

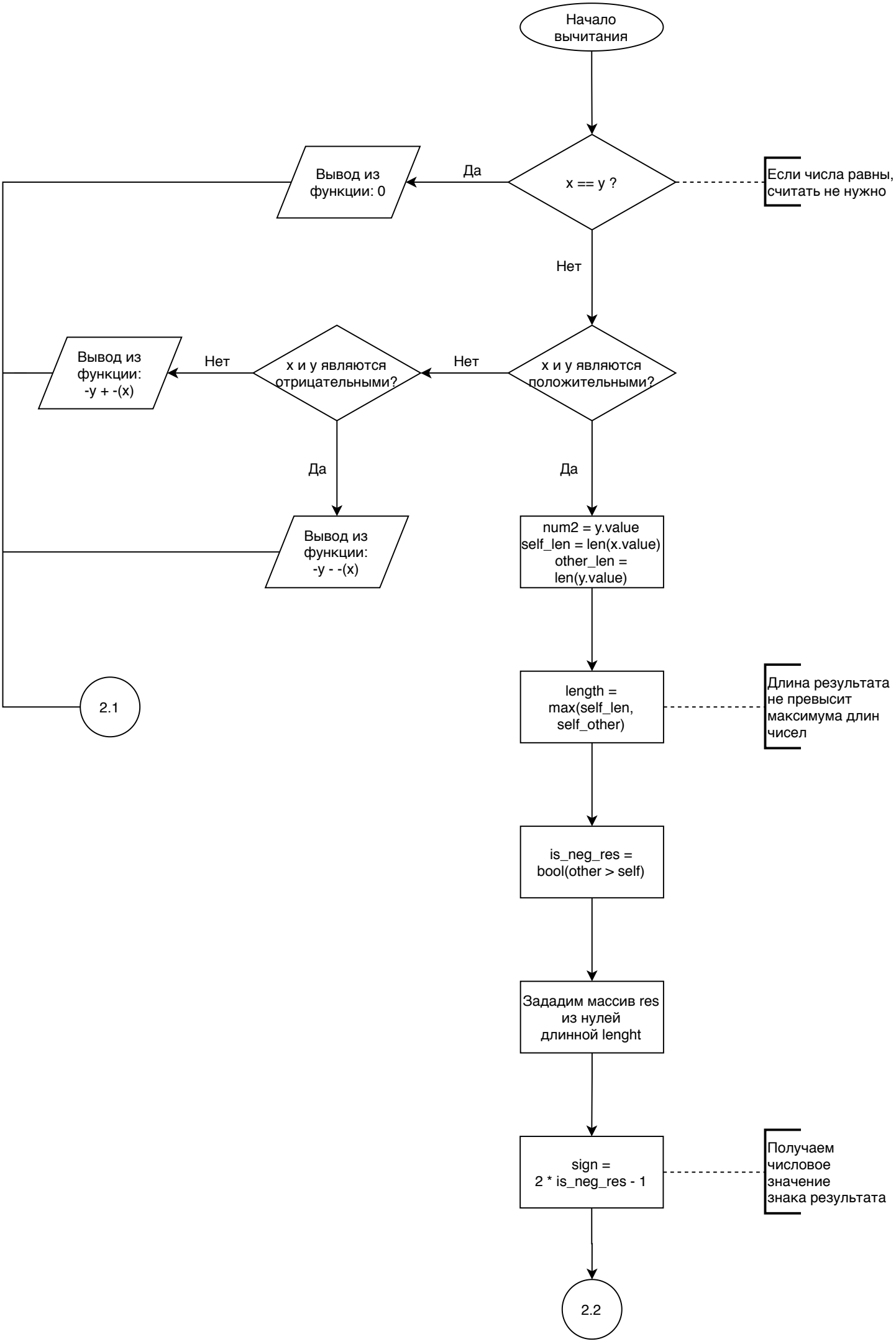
1. Логика работы интерфейса;
2. Сложение;
3. Вычитание;
4. Умножение;
5. Деление;
6. Возведение в степень;
7. Извлечение корня;
8. Нахождение НОД;
9. Нахождение остатка от деления.

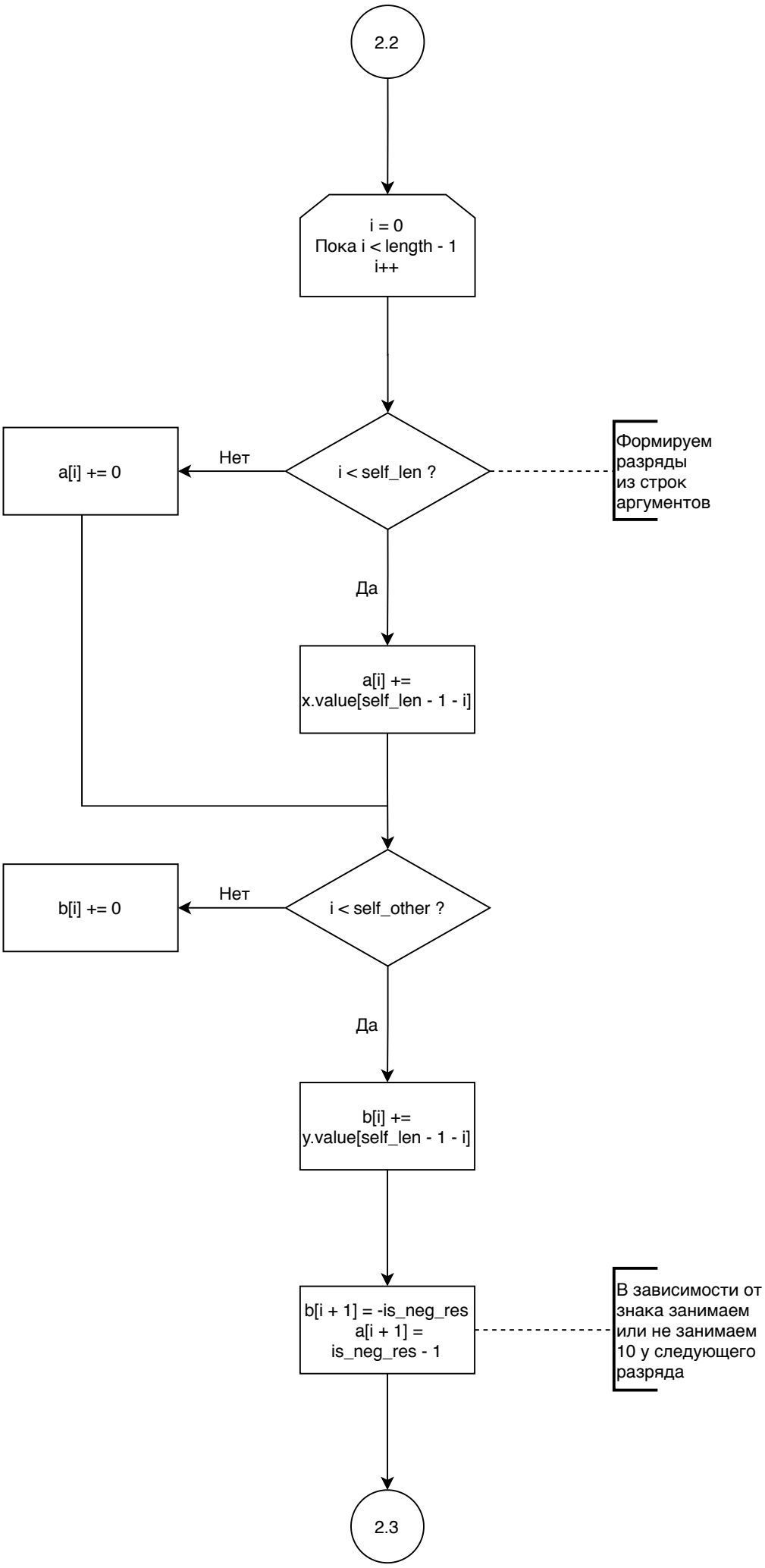

Начало
Ввод первого числа x
Ввод второго числа y
Выбор действия над числами
Сложение
Вычитание
Умножение
Деление (целочисленное)
Возведение в степень
Вычисления корня нужной степени (результат - целое число)
Нахождение НОД
Нахождение остатка от деления
Сложение x + y
Умножение x * y
Возведение в степень x.bipow(y)
Нахождение НОД GCD(x, y)
Вычитание x -y
Деление x / y
Извлечение корня x.birt(y)
Остаток от деления x % y
Вывод результата на экран
Конец

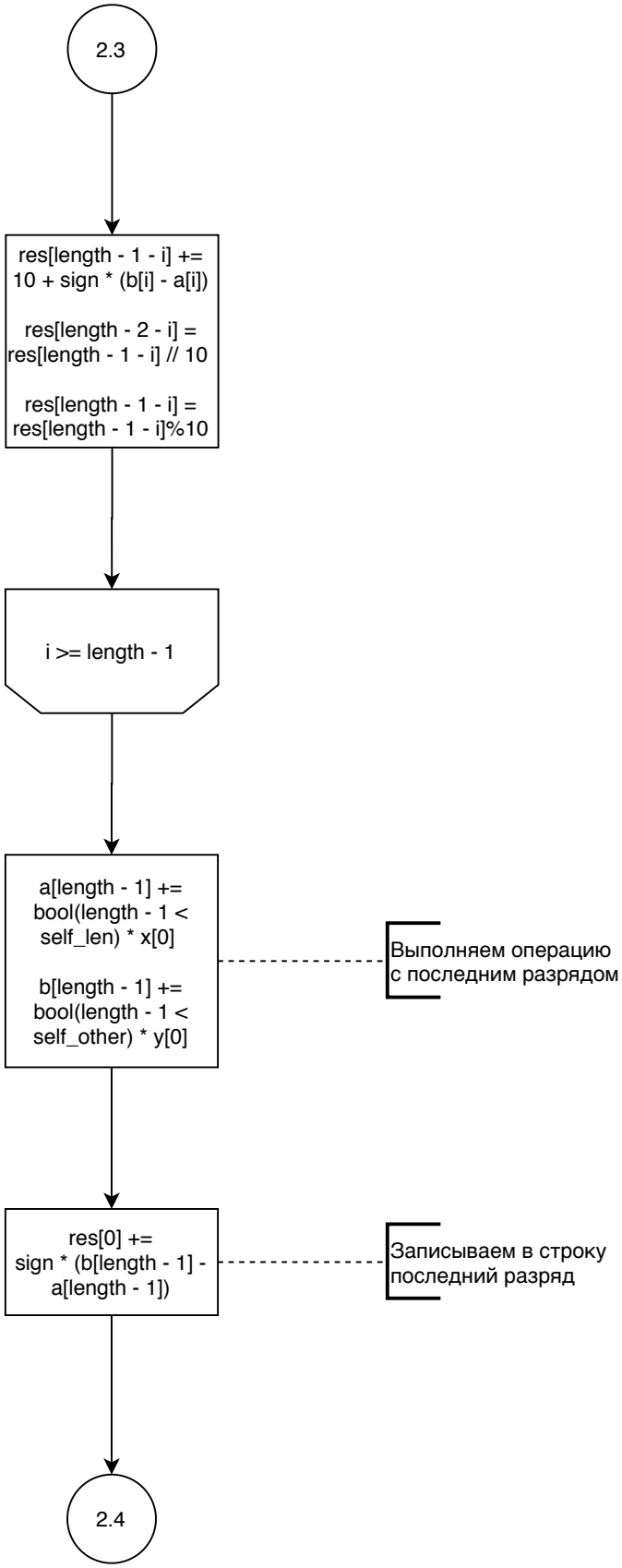

Начало сложения
Совпадают ли знаки x и y?
Нет
Число x - отрицательное?
Нет
y - (-x)
Да
x - (-y)
Если знаки одинаковые, то выполняем сложение
num2 = y.value
self_len = len(x.value)
other_len = len(y.value)
length = max(self_len, other_len) + 1
Длина суммы равна максимуму из двух длин + 1 из-за возможного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

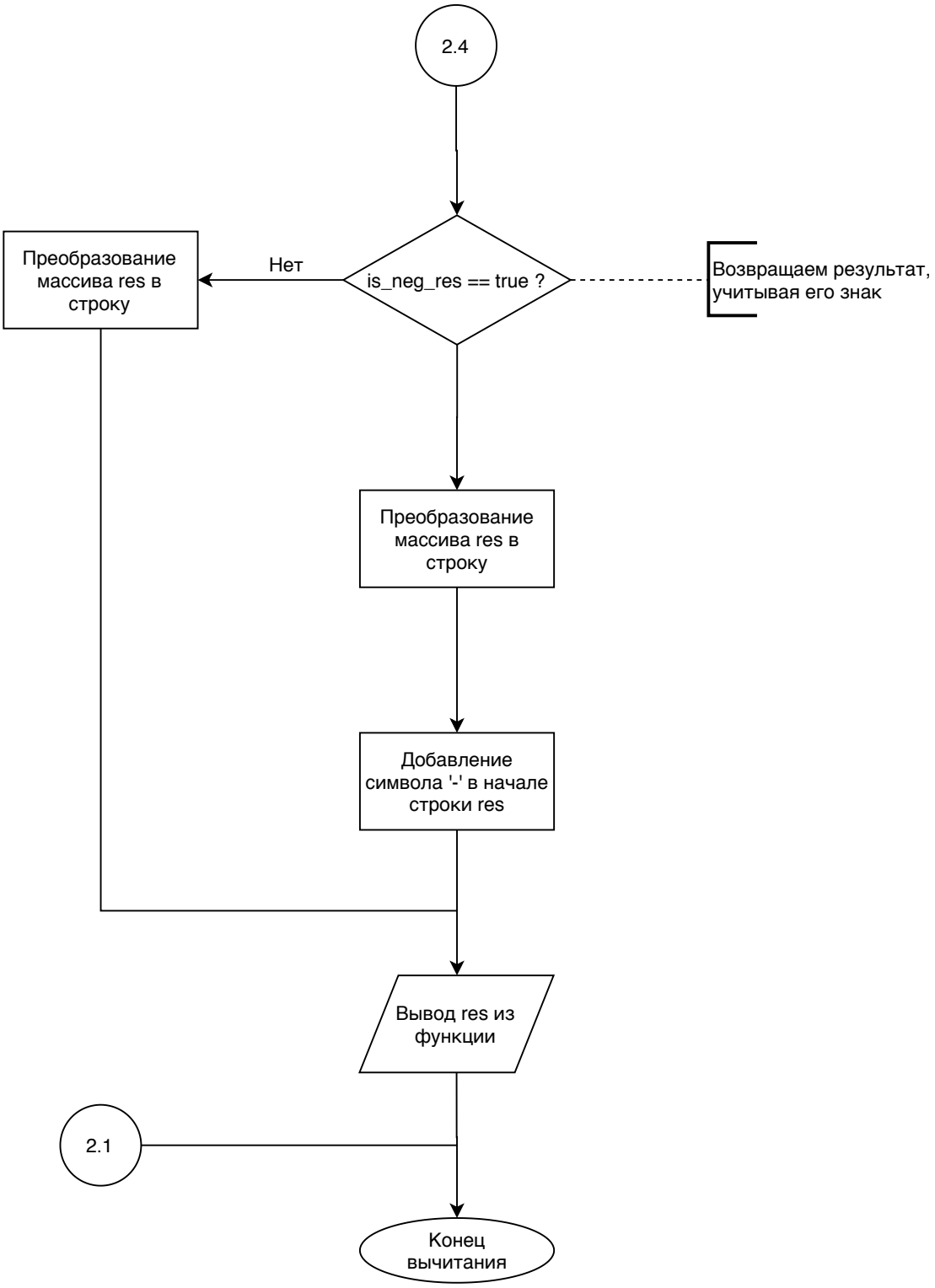

1.1
i < other_len ?
Нет
Выполняем сложение
Да
i < self_len ?
Нет
res[j] = num2[other_len - 1 - i]
Да
res[j] = x.value[other_len - 1 - i] + num2[other_len - 1 - i]
i < self_len ?
Нет
res[j] = x.value[other_len - 1 - i]
res[j] = 0
res[j-1] = res[j] // 10
Выполняем перенос в следующий разряд, если он был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

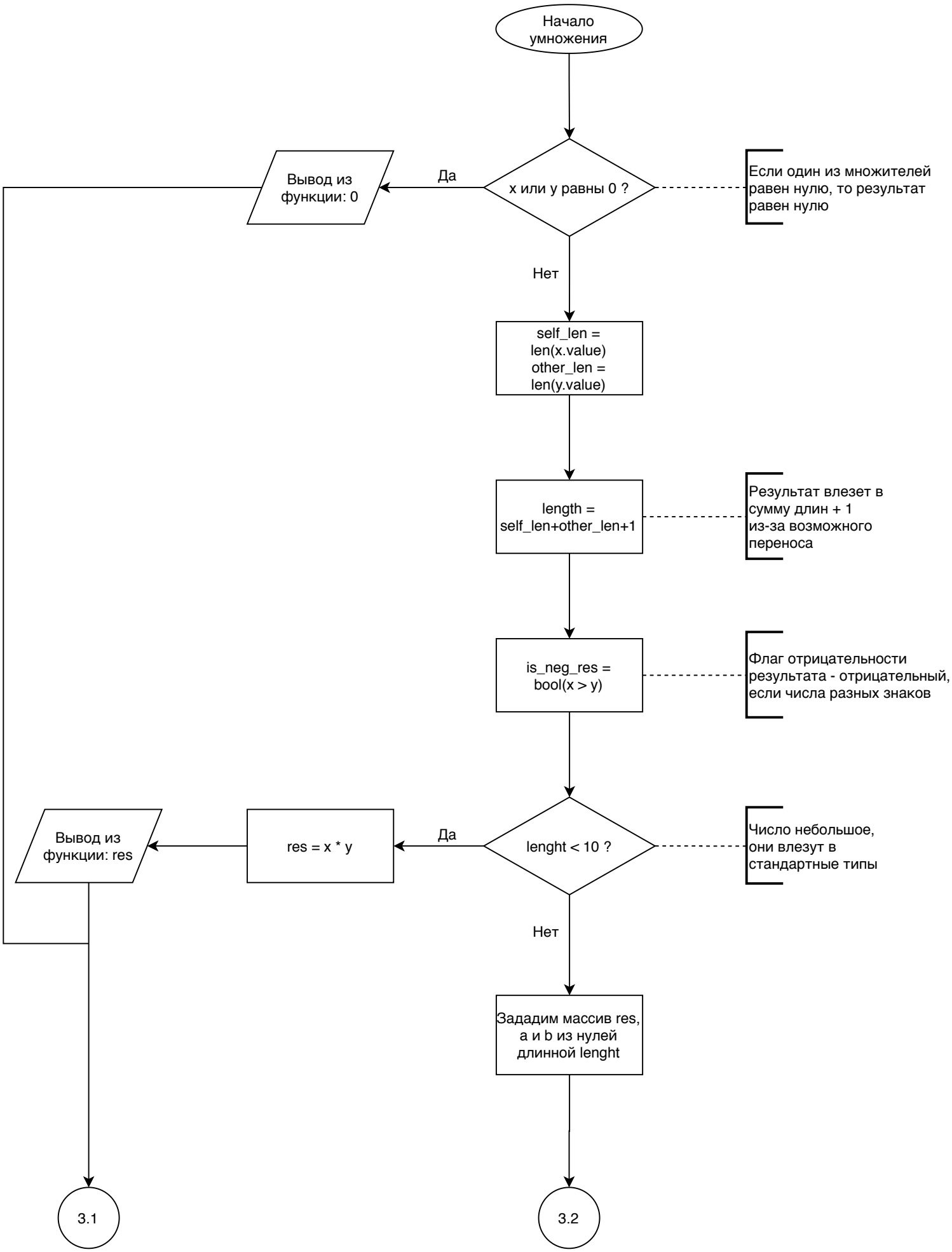

1.2
x - отрицательное число?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
Конец сложения

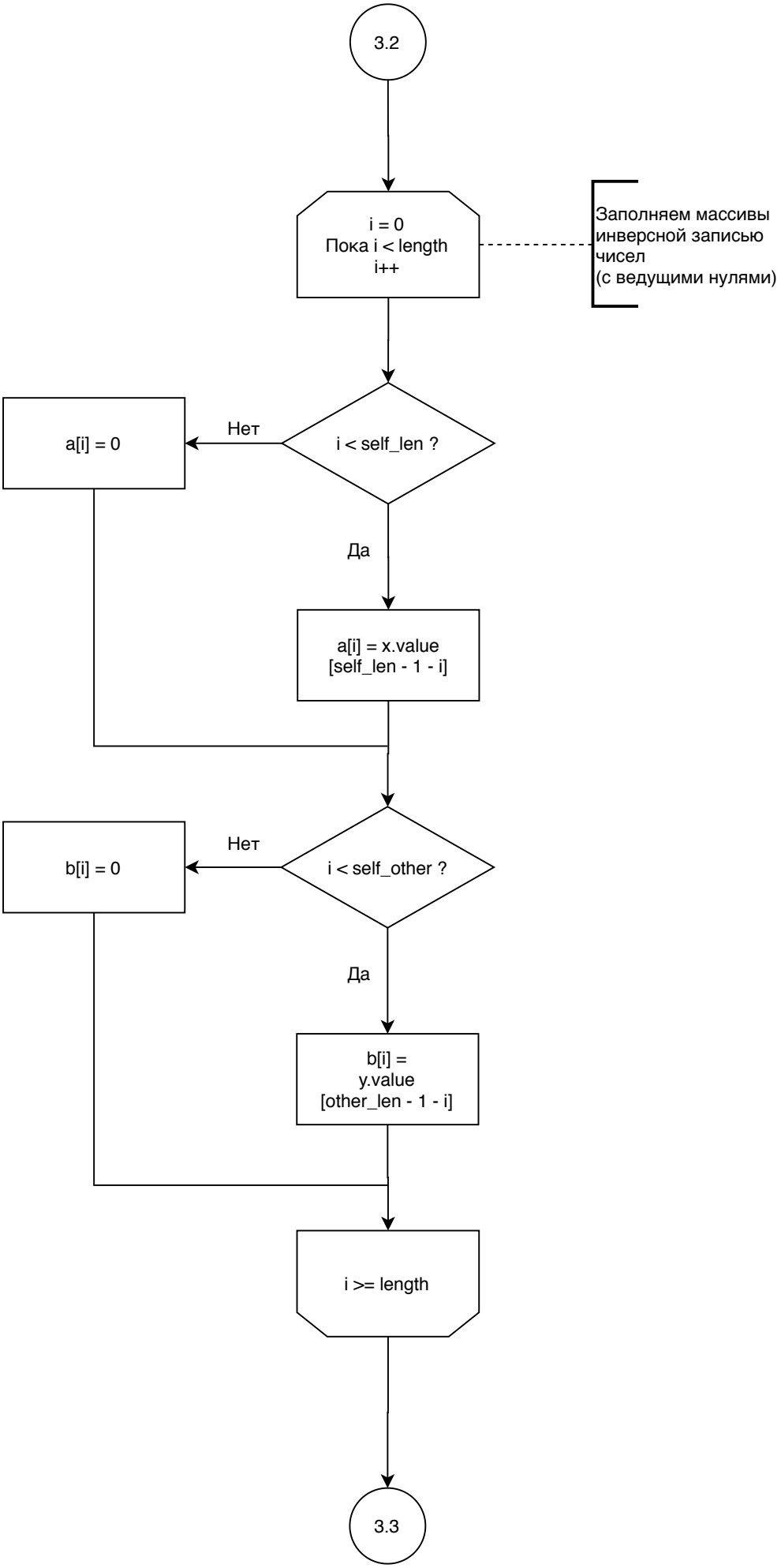

Начало вычитания
x == y ?
Да
Вывод из функции: 0
Если числа равны, считать не нужно
Нет
x и y являются положительными?
Нет
x и y являются отрицательными?
Нет
Вывод из функции: -y + -(x)
Да
Вывод из функции: -y - -(x)
Да
num2 = y.value
self_len = len(x.value)
other_len = len(y.value)
length = max(self_len, 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

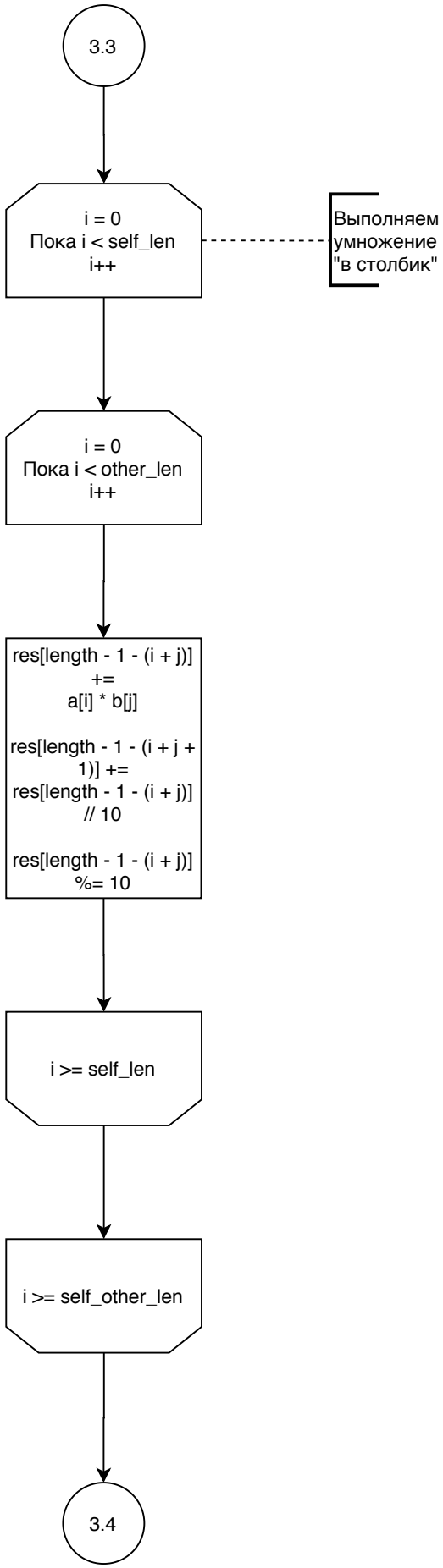

2.2
i = 0
Пока i < length - 1
i++
i < self_len ?
Нет
a[i] += 0
Да
Формируем разряды из строк аргументов
a[i] += x.value[self_len - 1 - i]
i < self_other ?
Нет
b[i] += 0
Да
b[i] += y.value[self_len - 1 - i]
b[i + 1] = -is_neg_res
a[i + 1] = is_neg_res - 1
В зависимости от знака занимаем или не занимаем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

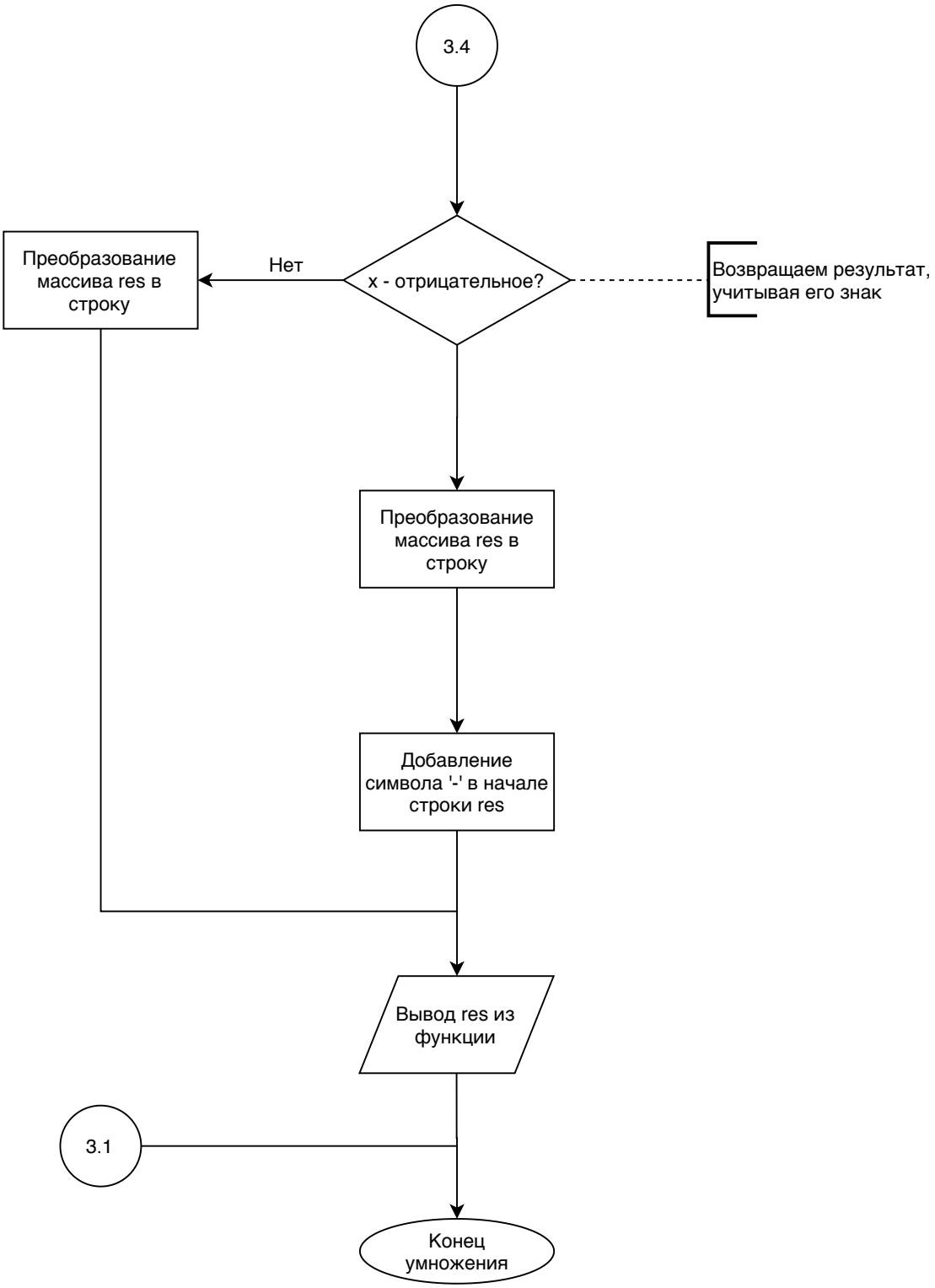

2.3
res[length - 1 - i] += 10 + sign * (b[i] - a[i])
res[length - 2 - i] = res[length - 1 - i] // 10
res[length - 1 - i] = res[length - 1 - i]%10
i >= length - 1
a[length - 1] += bool(length - 1 < self_len) * x[0]
b[length - 1] += bool(length - 1 < self_other) * y[0]
Выполняем операцию с последним разрядом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

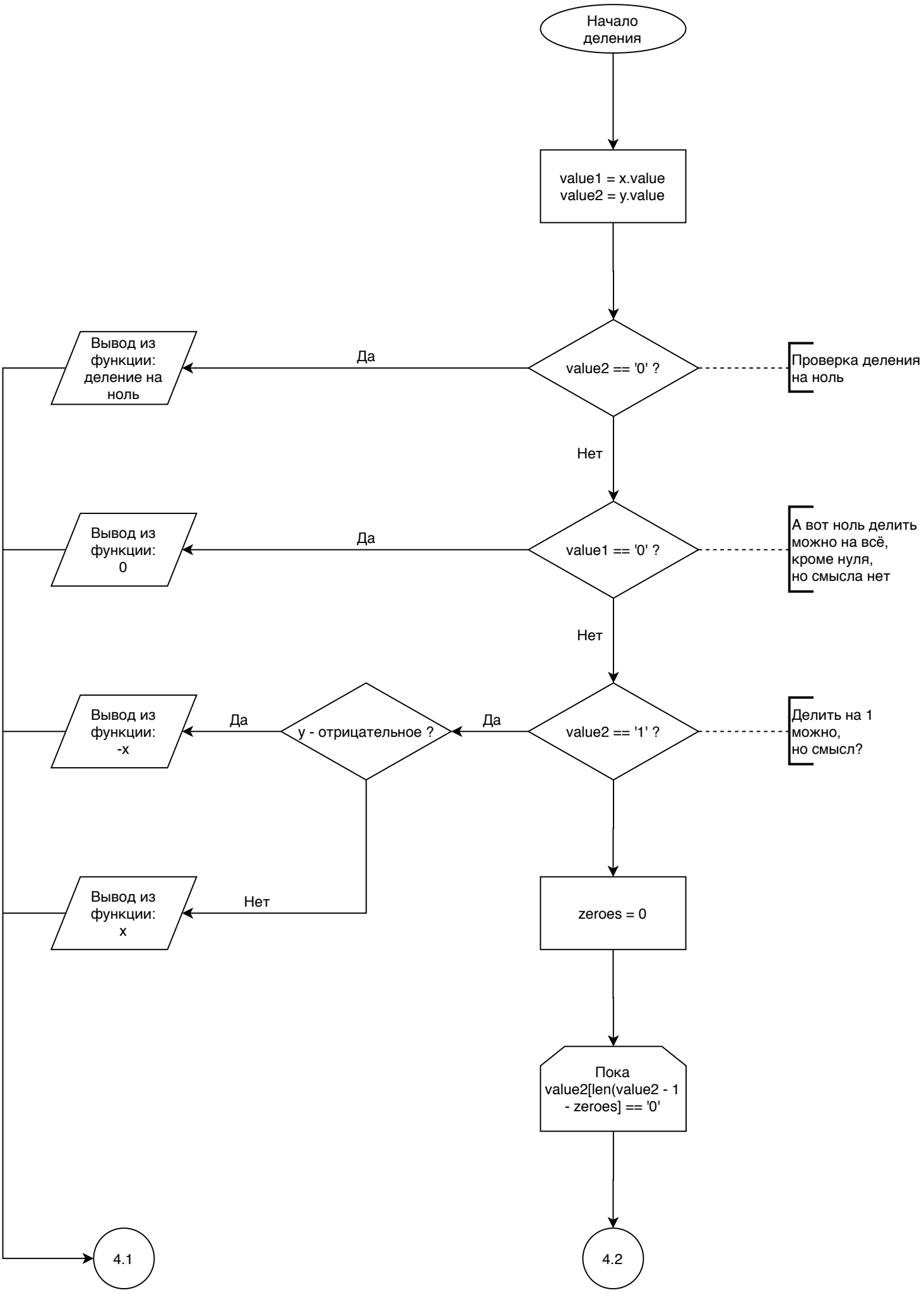

2.4
is_neg_res == true ?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
2.1
Конец вычитания

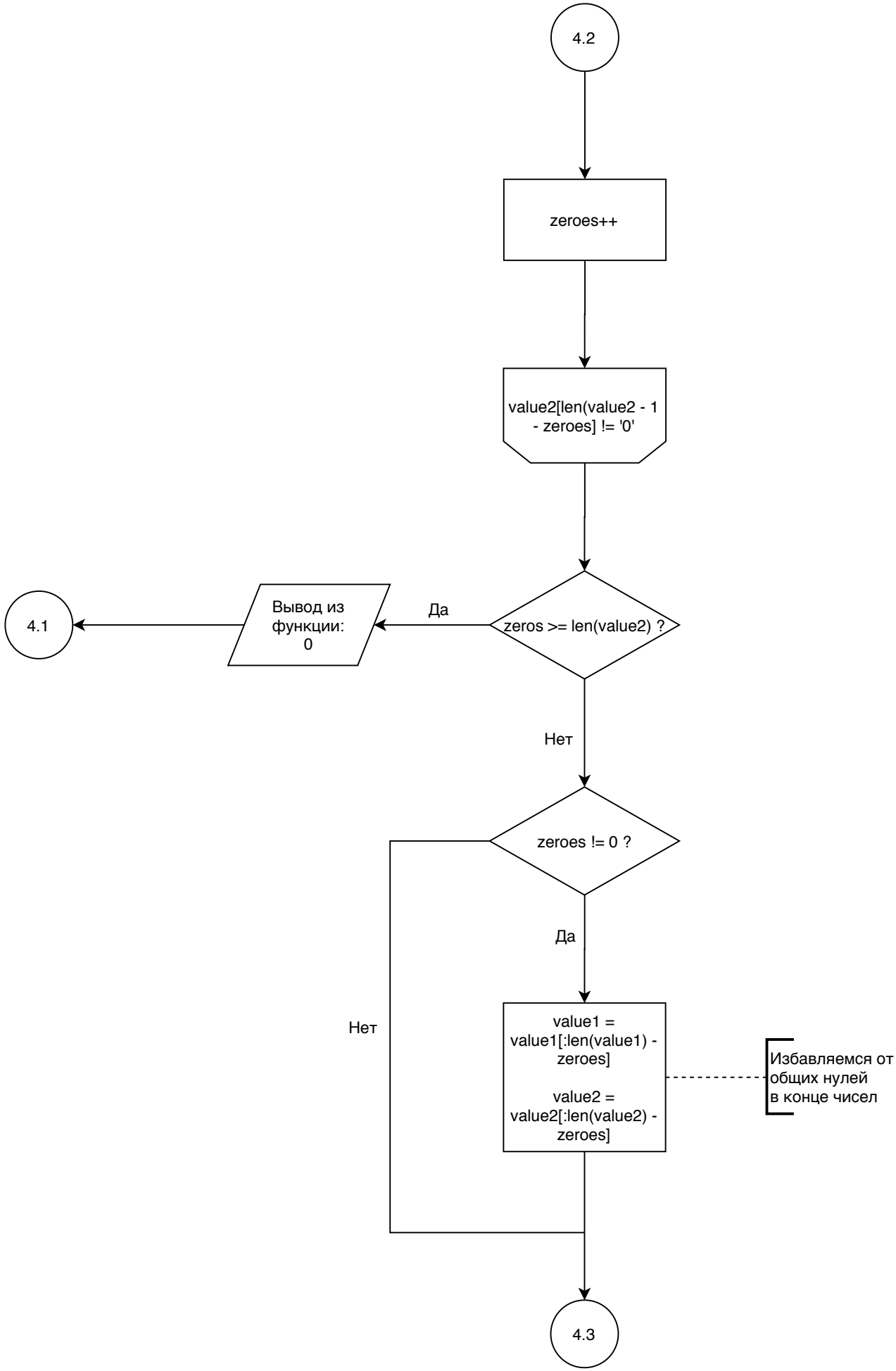

Начало умножения
x или y равны 0 ?
Да
Вывод из функции: 0
Если один из множителей равен нулю, то результат равен нулю
Нет
self_len = len(x.value) other_len = len(y.value)
length = self_len+other_len+1
Результат влезет в сумму длин + 1 из-за возможного переноса
is_neg_res = bool(x > y)
Флаг отрицательности 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

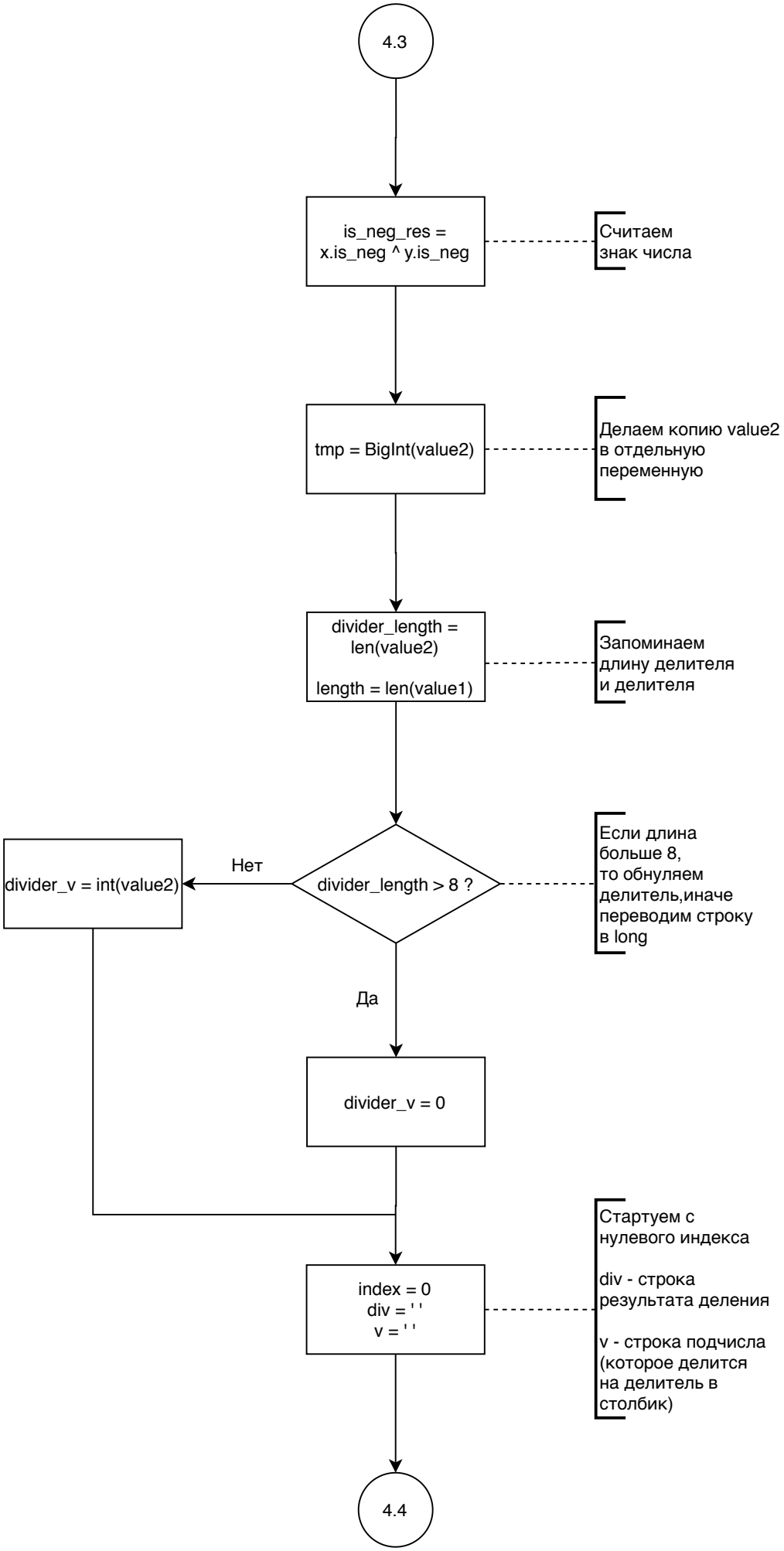

3.2
i = 0
Пока i < length
i++
Заполняем массивы
инверсной записью
чисел
(с ведущими нулями)
i < self_len ?
Нет
a[i] = 0
Да
a[i] = x.value
[self_len - 1 - i]
i < self_other ?
Нет
b[i] = 0
Да
b[i] =
y.value
[other_len - 1 - i]
i >= length
3.3

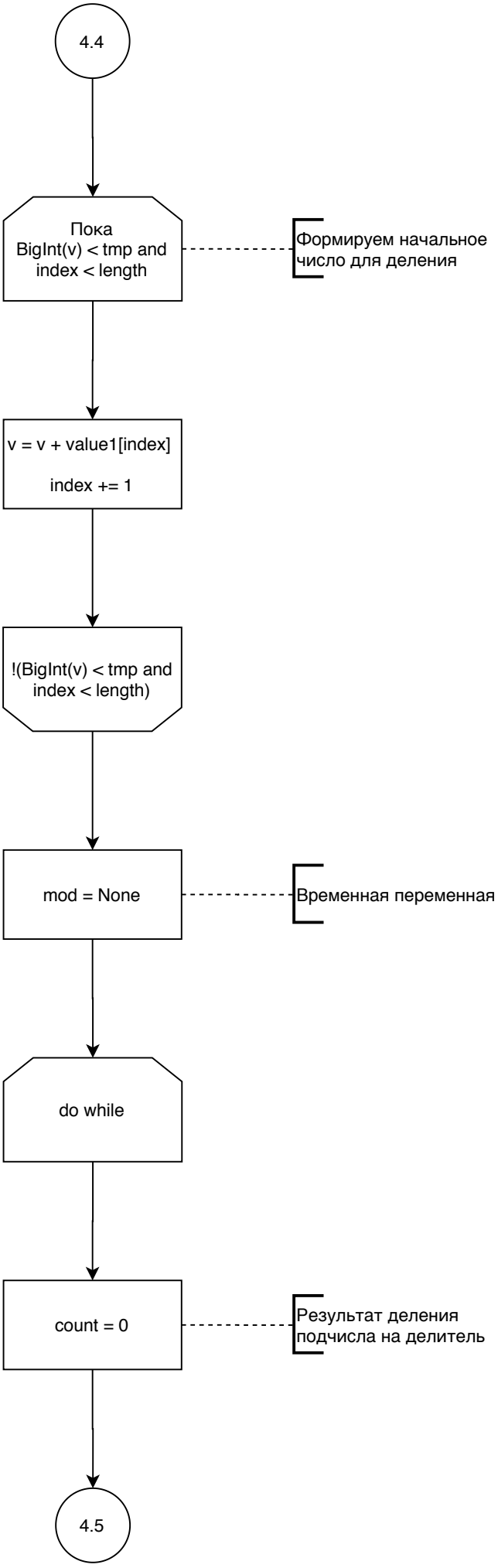

3.3
i = 0
Пока i < self_len
i++
Выполняем умножение "в столбик"
i = 0
Пока i < other_len
i++
res[length - 1 - (i + j)] += a[i] * b[j]
res[length - 1 - (i + j + 1)] += res[length - 1 - (i + j)] // 10
res[length - 1 - (i + j)] %= 10
i >= self_len
i >= self_other_len
3.4

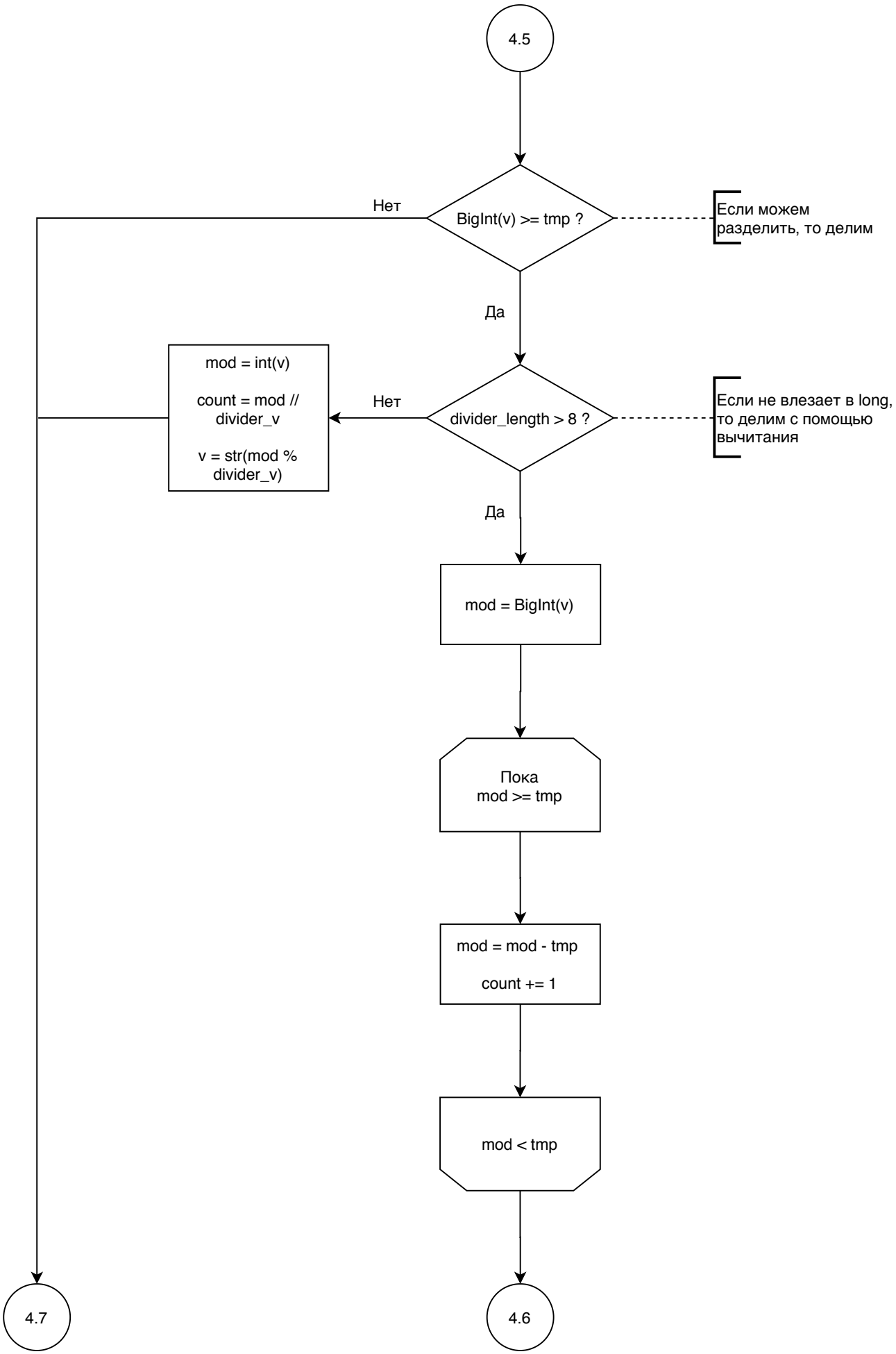

3.4
x - отрицательное?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
3.1
Конец умножения

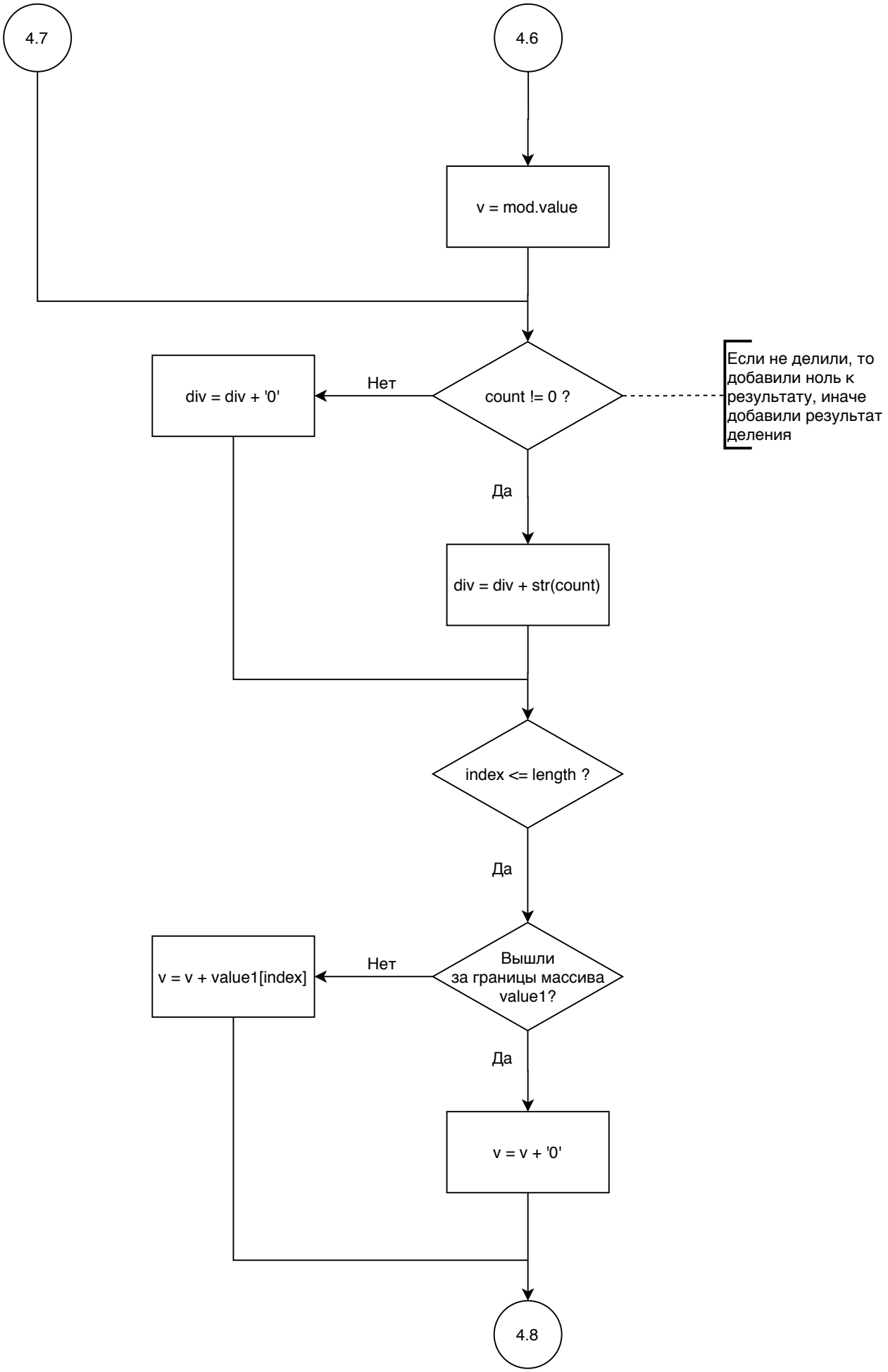

Начало деления
value1 = x.value
value2 = y.value
value2 == '0' ?
Да
Вывод из функции: деление на ноль
Проверка деления на ноль
Нет
value1 == '0' ?
Да
Вывод из функции: 0
А вот ноль делить можно на всё, кроме нуля, но смысла нет
Нет
value2 == '1' ?
Да
y - отрицательное ?
Да
Вывод из функции: -x
Делить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

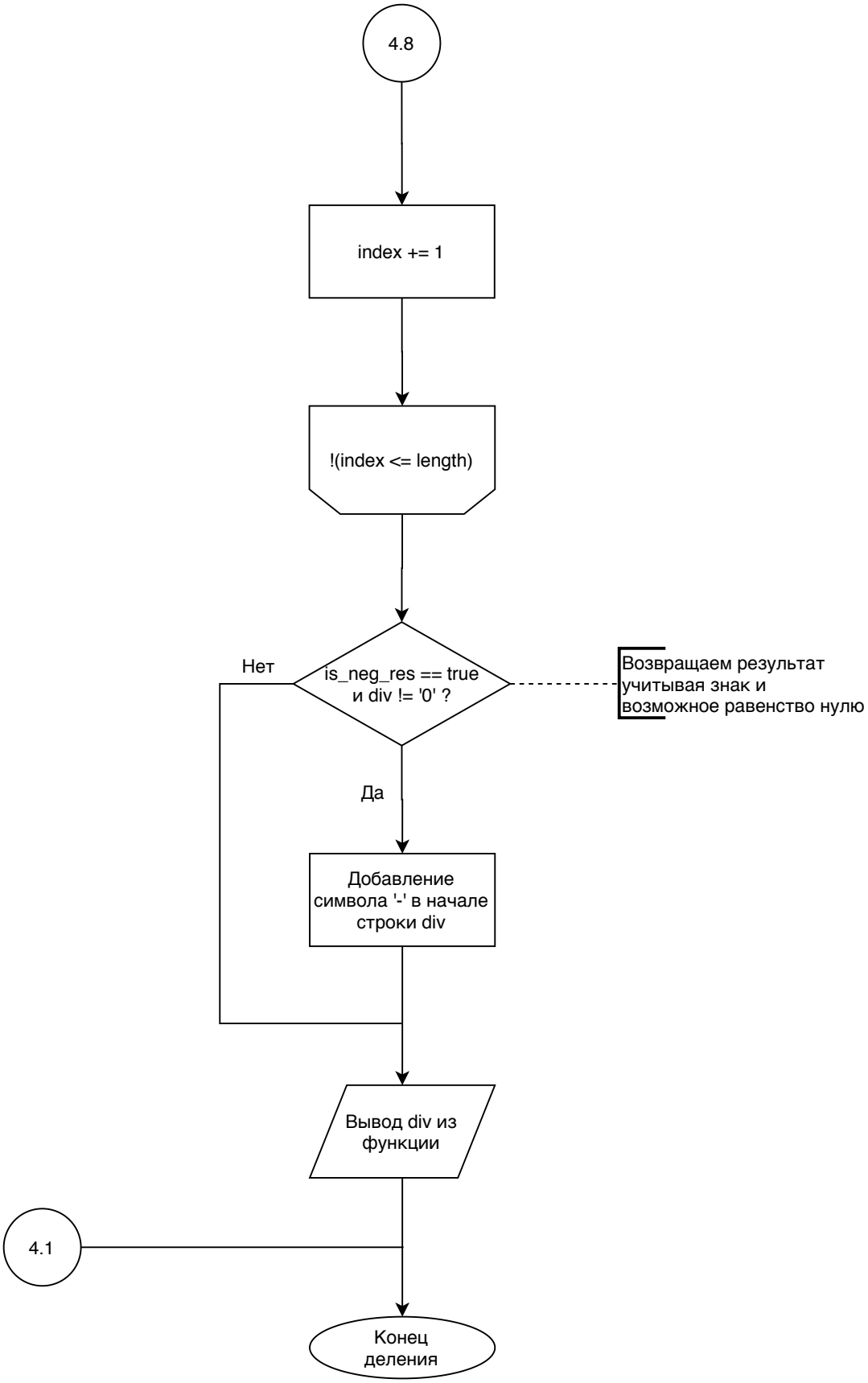

4.2
zeroes++
value2[len(value2 - 1 - zeroes] != '0'
zeros >= len(value2) ?
Да
Вывод из функции: 0
4.1
Нет
zeroes != 0 ?
Да
Нет
value1 = value1[:len(value1) - zeroes]
value2 = value2[:len(value2) - zeroes]
Избавляемся от общих нулей в конце чисел
4.3

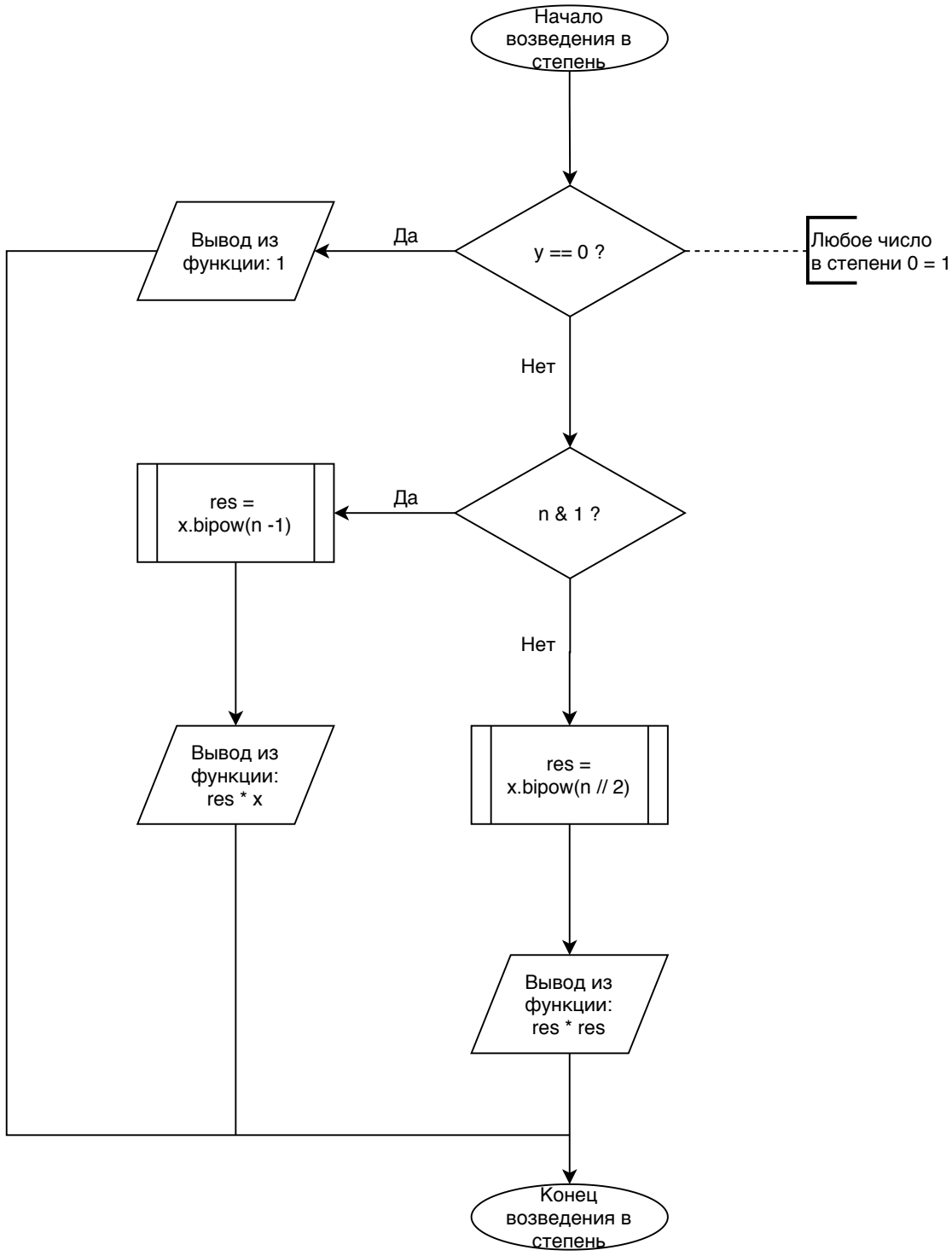

4.3
is_neg_res =
x.is_neg ^ y.is_neg
Считаем
знак числа
tmp = BigInt(value2)
Делаем копию value2
в отдельную
переменную
divider_length =
len(value2)
length = len(value1)
Запоминаем
длину делителя
и делителя
divider_length > 8 ?
Нет
divider_v = int(value2)
Если длина
больше 8,
то обнуляем
делитель,иначе
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

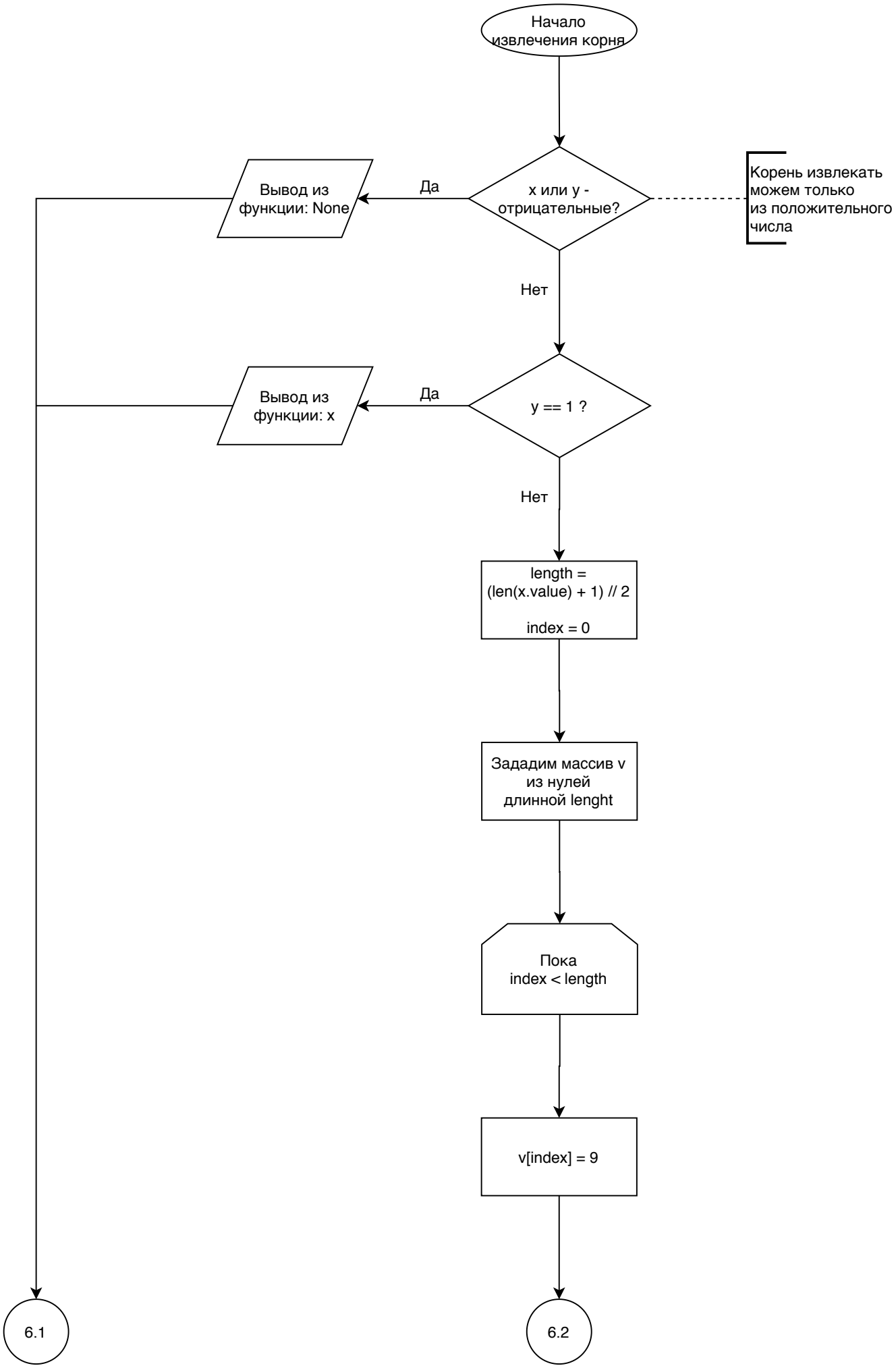

4.4
Пока
BigInt(v) < tmp and
index < length
Формируем начальное
число для деления
v = v + value1[index]
index += 1
!(BigInt(v) < tmp and
index < length)
mod = None
Временная переменная
do while
count = 0
Результат деления
подчисла на делитель
4.5

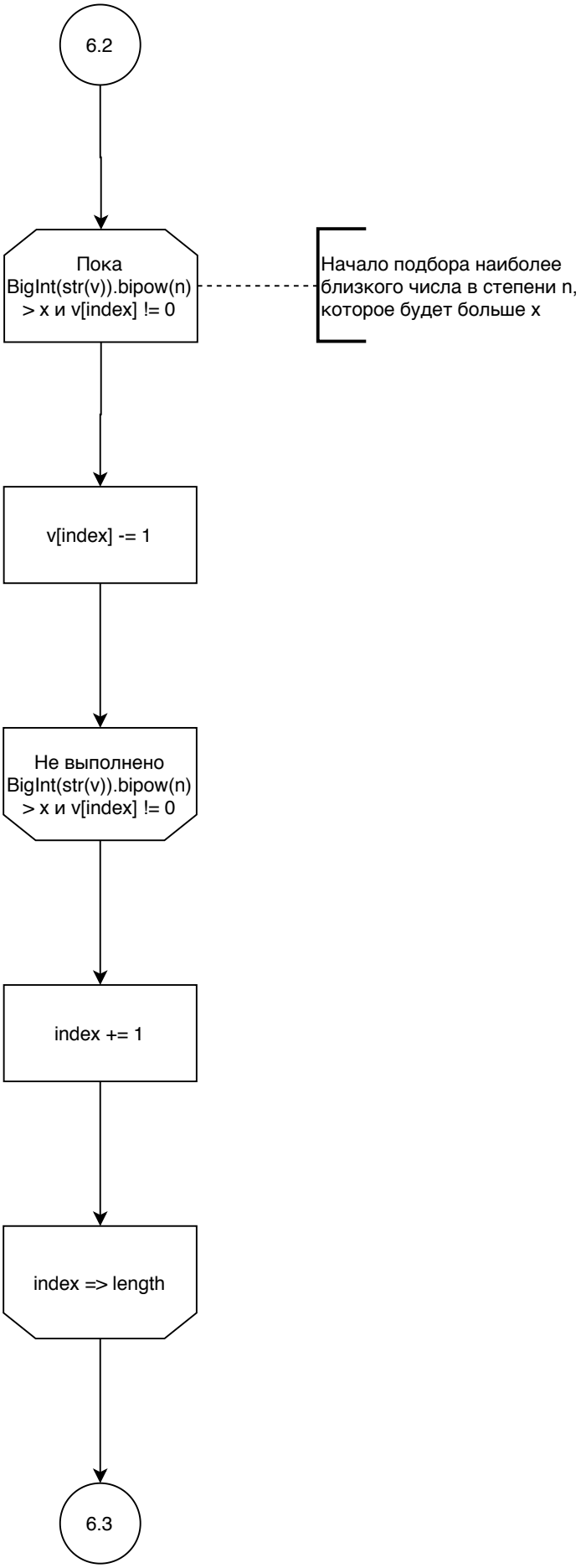

4.5
BigInt(v) >= tmp ?
Нет
Если можем разделить, то делим
Да
mod = int(v)
count = mod // divider_v
v = str(mod % divider_v)
Нет
divider_length > 8 ?
Если не влезает в long, то делим с помощью вычитания
Да
mod = BigInt(v)
Пока mod >= tmp
mod = mod - tmp
count += 1
mod < tmp
4.7
4.6

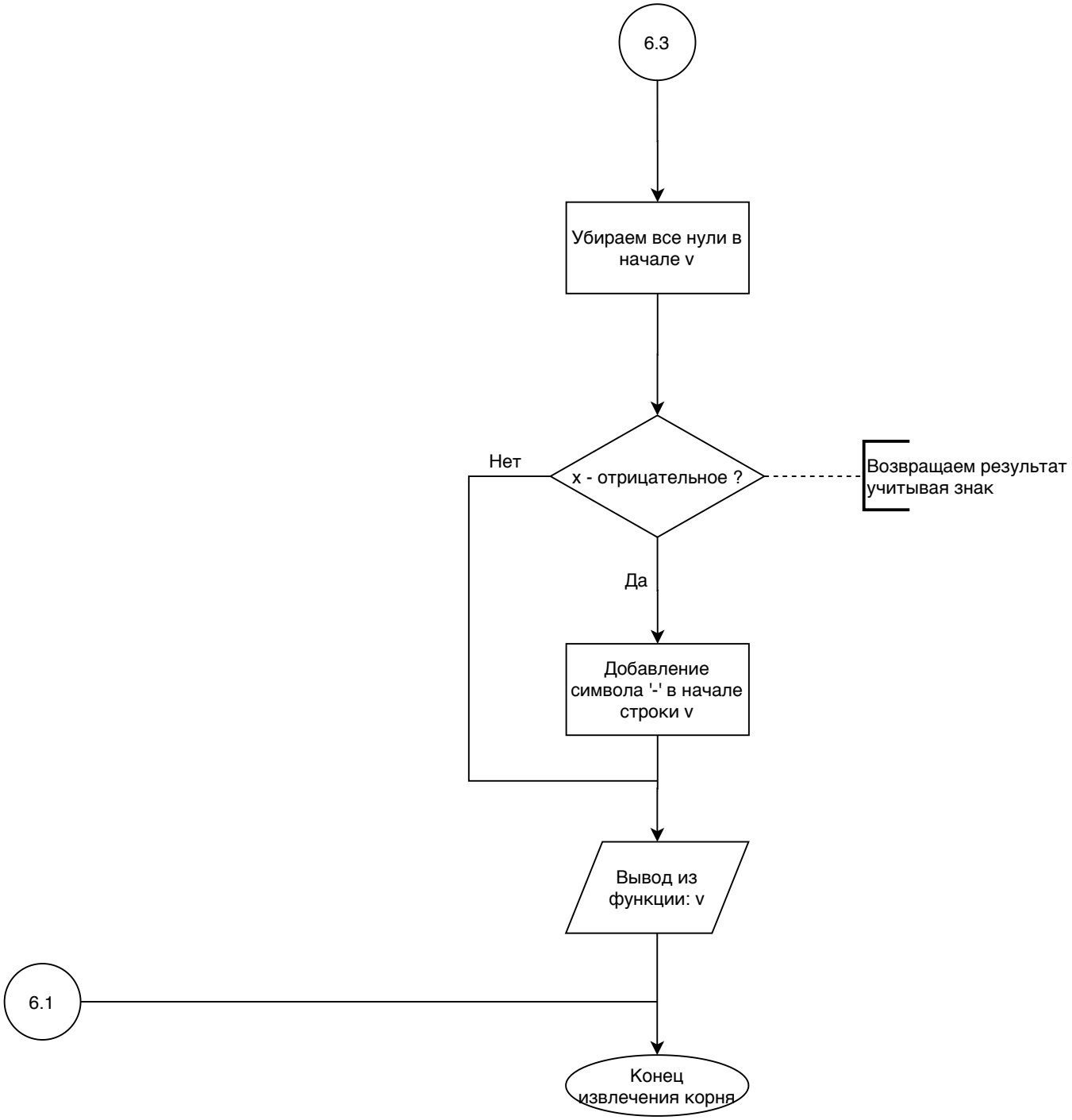

4.7
4.6
v = mod.value
count != 0 ?
Нет
div = div + '0'
Если не делили, то добавили ноль к результату, иначе добавили результат деления
Да
div = div + str(count)
index <= length ?
Да
Вышли за границы массива value1?
Нет
v = v + value1[index]
Да
v = v + '0'
4.8

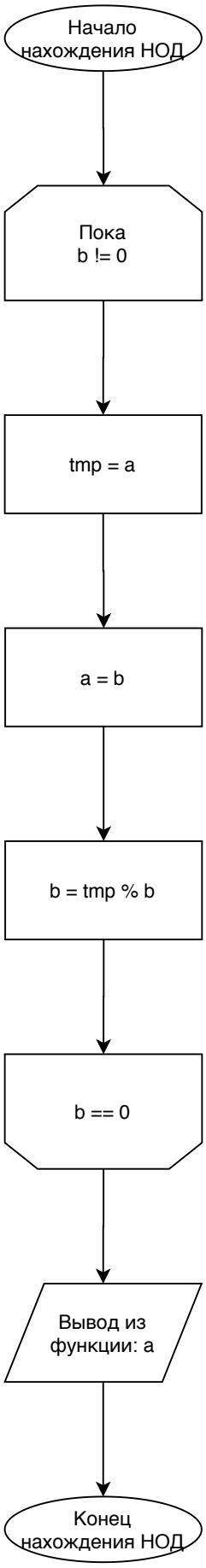

4.8
index += 1
!(index <= length)
Нет
is_neg_res == true
и div != '0' ?
Возвращаем результат
учитывая знак и
возможное равенство нулю
Да
Добавление
символа '-' в начале
строки div
Вывод div из
функции
4.1
Конец
деления

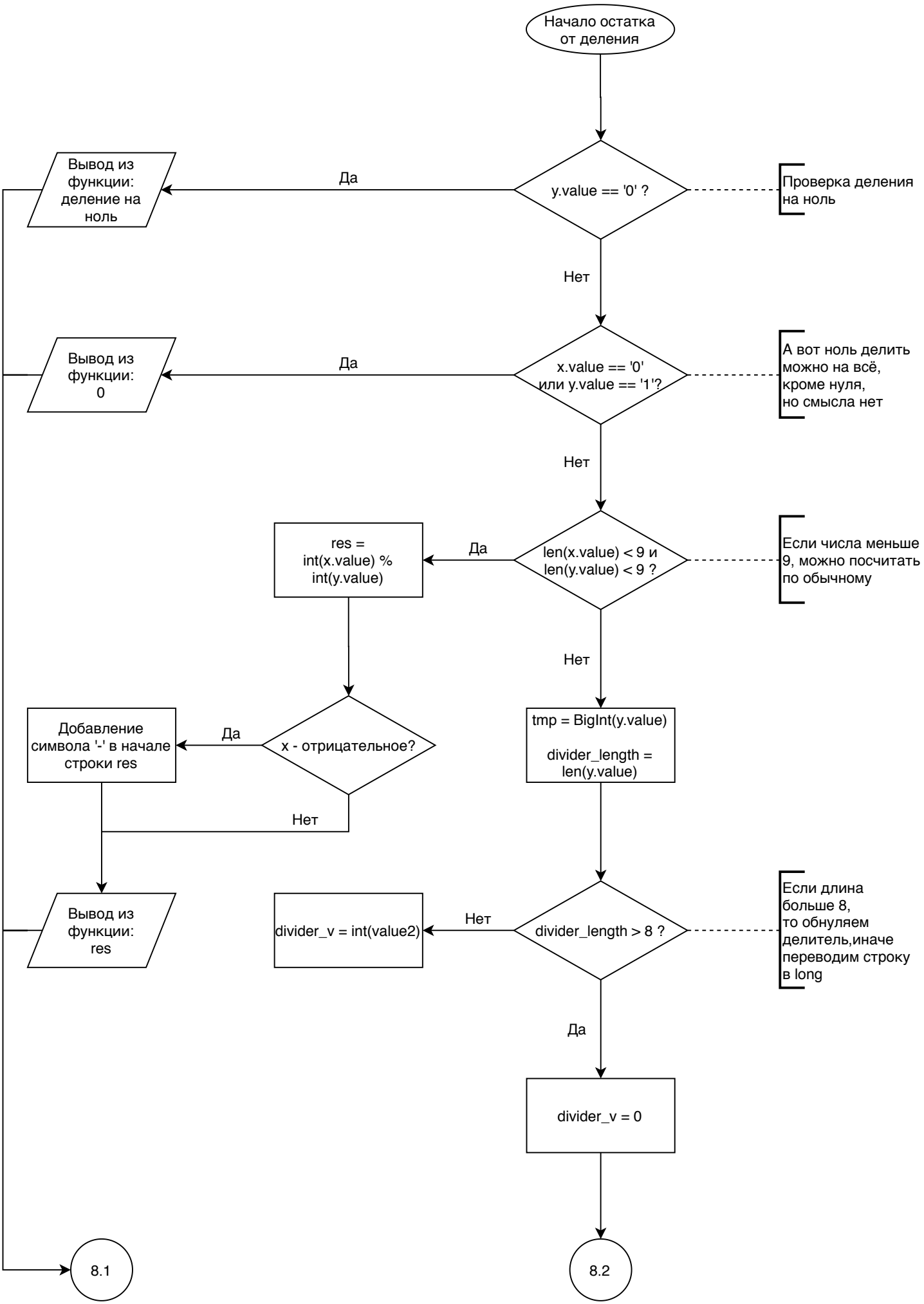

Начало возведения в степень
y == 0 ?
Да
Вывод из функции: 1
Любое число в степени 0 = 1
Нет
n & 1 ?
Да
res = x.bipow(n -1)
Нет
Вывод из функции: res * x
res = x.bipow(n // 2)
Вывод из функции: res * res
Конец возведения в степень

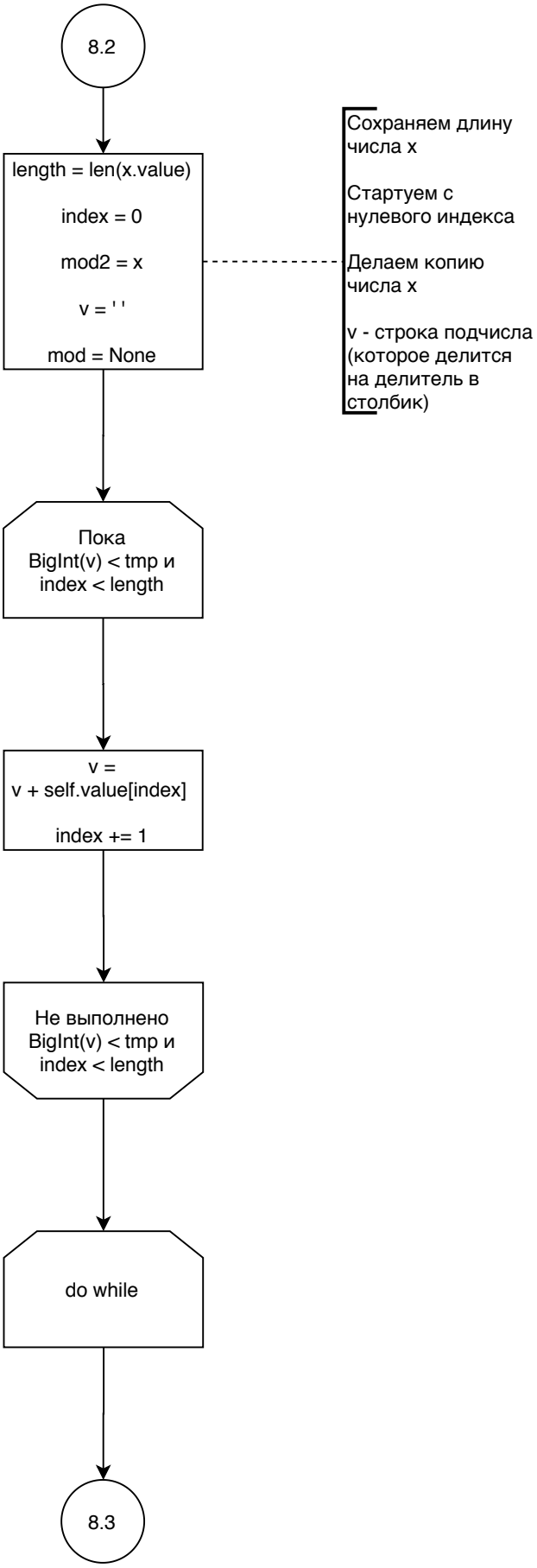

Начало извлечения корня
x или y - отрицательные?
Да
Вывод из функции: None
Корень извлекать можем только из положительного числа
Нет
y == 1 ?
Да
Вывод из функции: x
Нет
length = (len(x.value) + 1) // 2
index = 0
Зададим массив v из нулей длинной lenght
Пока index < length
v[index] = 9
6.1
6.2

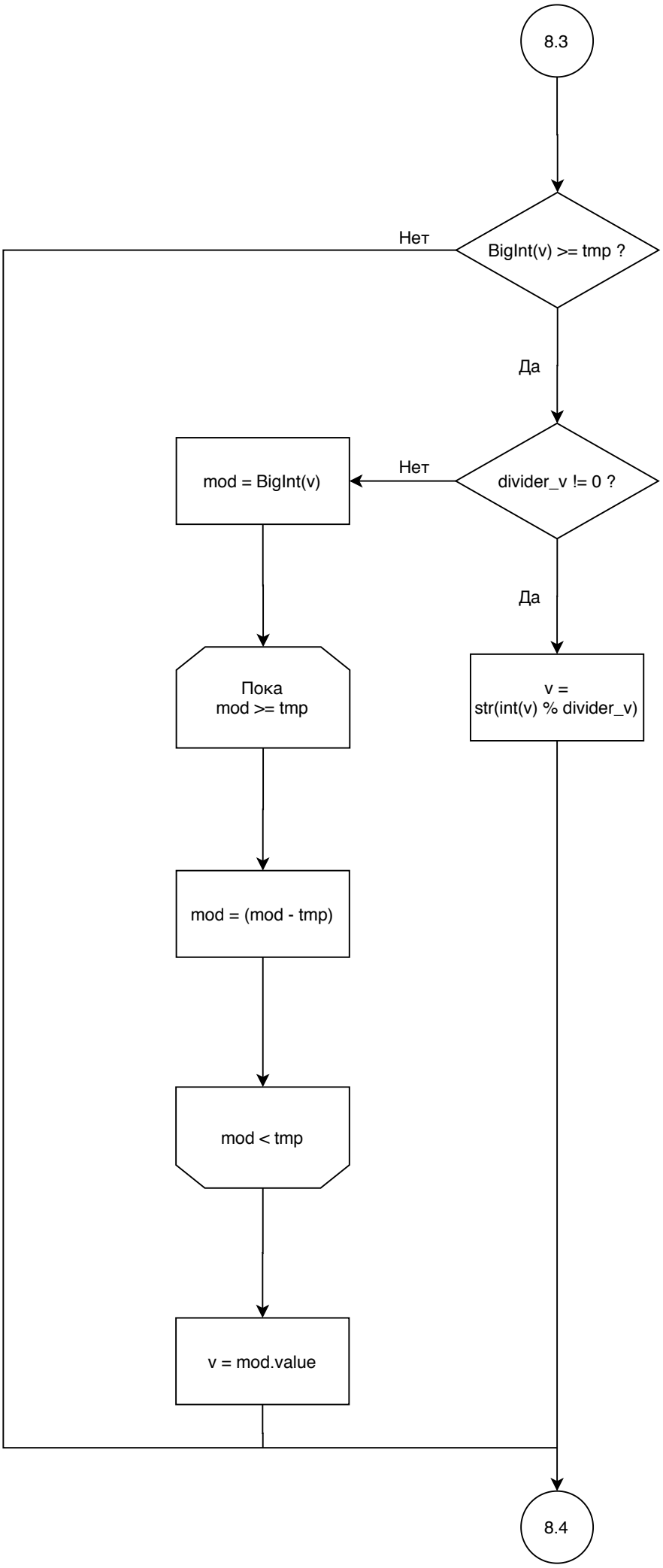

6.2
Пока BigInt(str(v)).bipow(n) > x и v[index] != 0
Начало подбора наиболее близкого числа в степени n, которое будет больше x
v[index] -= 1
Не выполнено BigInt(str(v)).bipow(n) > x и v[index] != 0
index += 1
index => length
6.3

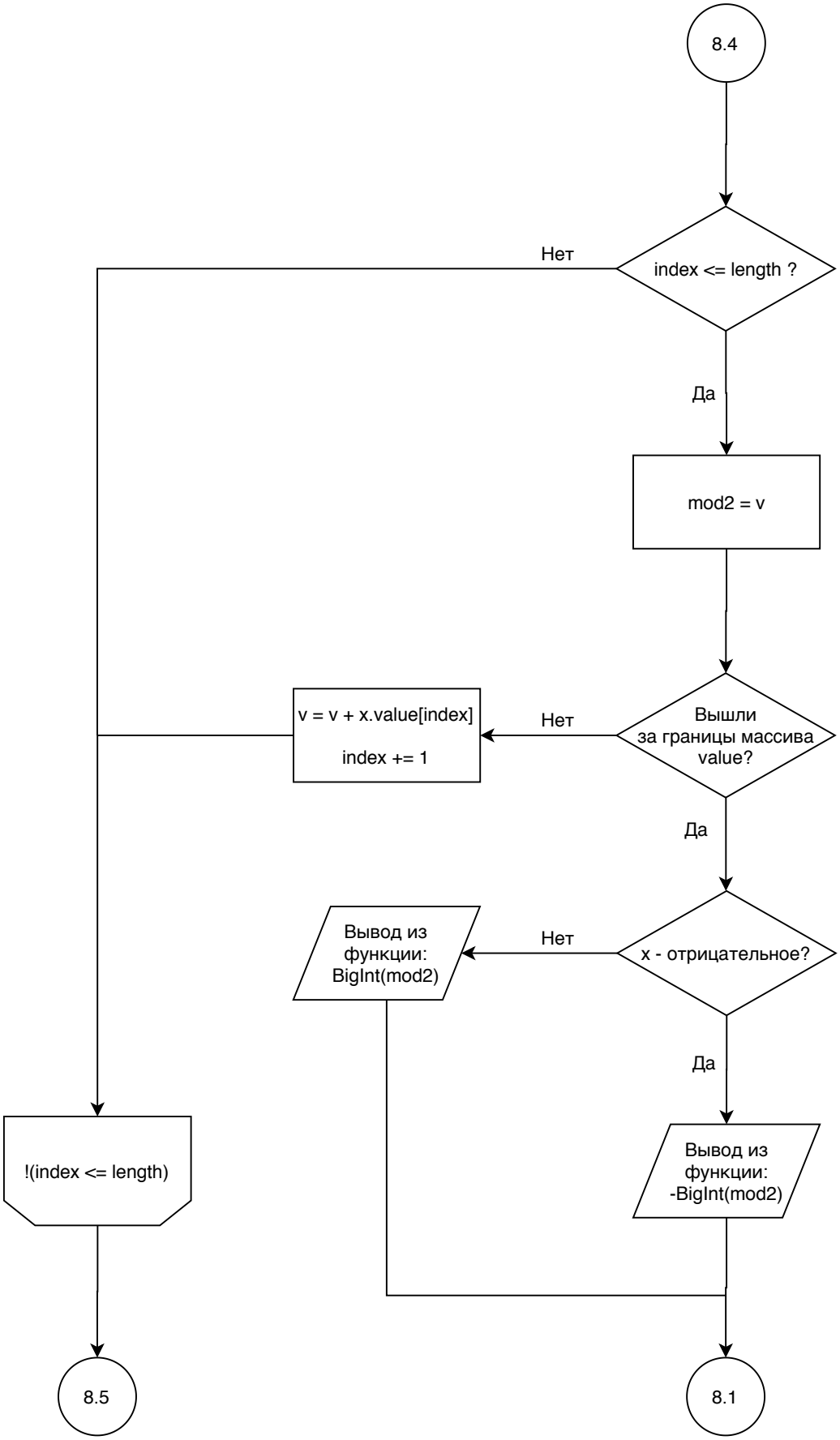

6.3
Убираем все нули в начале v
Нет
x - отрицательное ?
Возвращаем результат учитывая знак
Да
Добавление символа '-' в начале строки v
Вывод из функции: v
6.1
Конец извлечения корня

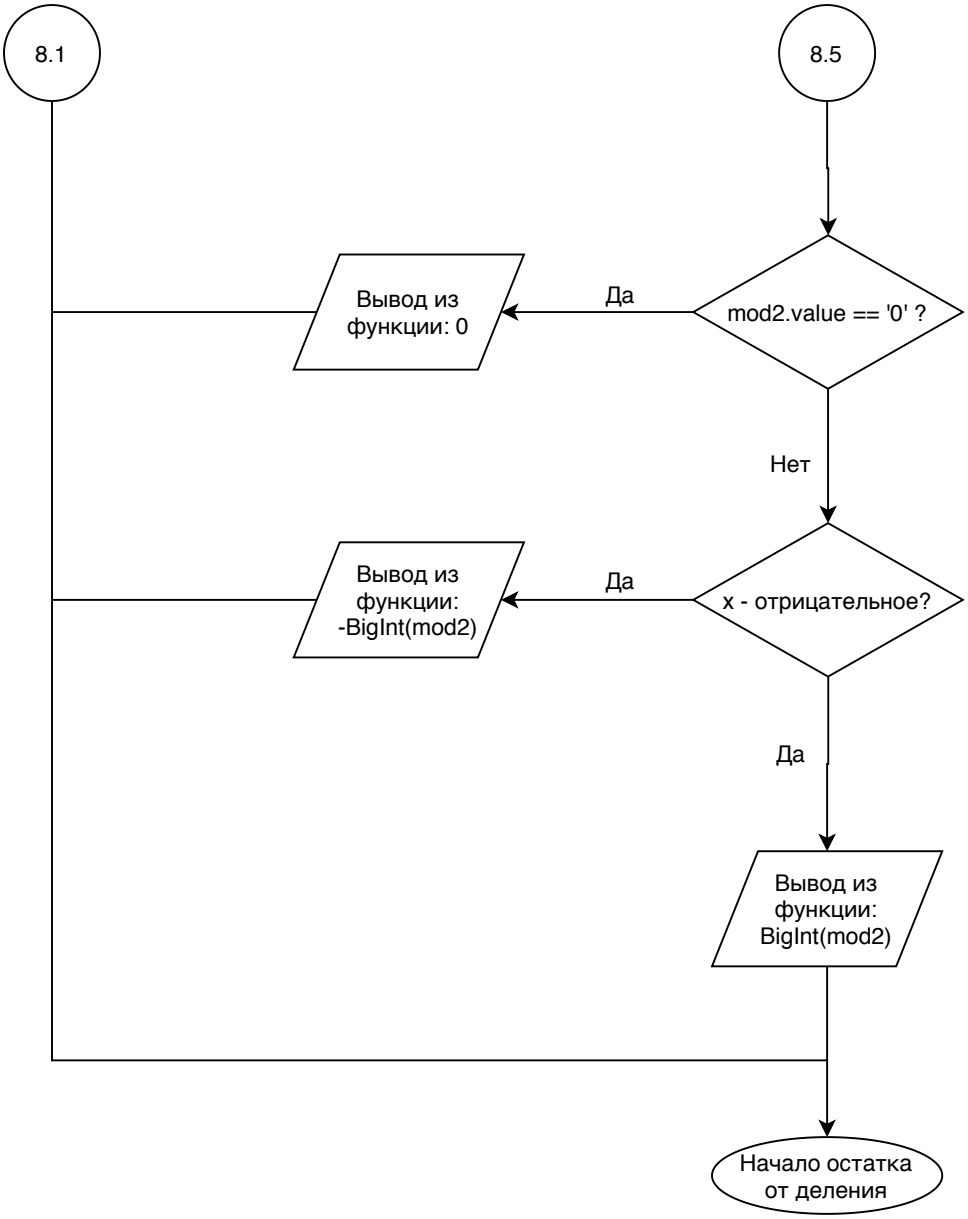

Начало
нахождения НОД
Пока
b != 0
tmp = a
a = b
b = tmp % b
b == 0
Вывод из
функции: a
Конец
нахождения НОД


Начало остатка от деления
y.value == '0' ?
Да
Вывод из функции: деление на ноль
Проверка деления на ноль
Нет
x.value == '0' или y.value == '1'?
Да
Вывод из функции: 0
А вот ноль делить можно на всё, кроме нуля, но смысла нет
Нет
len(x.value) < 9 и len(y.value) < 9 ?
Да
res = int(x.value) % int(y.value)
Если числа меньше 9, можно посчитать по обычному
Нет
x - отрицательное?
Да
Добавление символа '-' в начале строки res
Нет
tmp = BigInt(y.value)
divider_length = len(y.value)
divider_length > 8 ?
Нет
divider_v = int(value2)
Вывод из функции: res
Если длина больше 8, то обнуляем делитель,иначе переводим строку в long
Да
divider_v = 0
8.1
8.2


8.2
length = len(x.value)
index = 0
mod2 = x
v = ' '
mod = None
Сохраняем длину числа x
Стартуем с нулевого индекса
Делаем копию числа x
v - строка подчисла (которое делится на делитель в столбик)
Пока BigInt(v) < tmp и index < length
v = v + self.value[index]
index += 1
Не выполнено BigInt(v) < tmp и index < length
do while
8.3


8.3
BigInt(v) >= tmp ?
Нет
Да
divider_v != 0 ?
Нет
Да
mod = BigInt(v)
Пока
mod >= tmp
v =
str(int(v) % divider_v)
mod = (mod - tmp)
mod < tmp
v = mod.value
8.4


8.4
index <= length ?
Нет
Да
mod2 = v
Вышли
за границы массива
value?
Нет
v = v + x.value[index]
index += 1
Да
x - отрицательное?
Нет
Вывод из
функции:
BigInt(mod2)
Да
Вывод из
функции:
-BigInt(mod2)
!(index <= length)
8.5
8.1


8.1
8.5
mod2.value == '0' ?
Да
Вывод из функции: 0
Нет
x - отрицательное?
Да
Вывод из функции: -BigInt(mod2)
Да
Вывод из функции: BigInt(mod2)
Начало остатка от деления

8 Исходный код программы

8.1 Исходный код bigint.py

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  from sys import setrecursionlimit
5
6  setrecursionlimit(1500)  # Максимальный стек рекурсии
7
8
9  class BigInt(object):
10     """Класс работы с большими целыми числами"""
11     is_neg = False  # Флаг отрицательности числа
12     value = ''      # Число в виде стоки
13
14     def __init__(self, x=0):
15         """Без аргументов конструктор задает значение равное нулю.
16         Если ‘x‘ - число, конструктор заполняет экземпляр этим числом.
17         Если ‘x‘ - строка, которая могла бы быть числом, конструктор заполня
       ет экземпляр этим числом.
18         Если ‘x‘ - экземпляр того же класса, конструктор копирует его содерж
       имое в новый экземпляр."""
19         self.value = '0'
20         # Если в конструктор передано целое число
21         if isinstance(x, int):
22             self.is_neg = x < 0
23             self.value = str(x if x >= 0 else -x)
24         # Если в конструктор передана строка
25         elif isinstance(x, str):
26             # Если вдруг пришла пустая строка, пропускаем, оставляем value =
       0
27             if len(x):
28                 self.is_neg = x[0] == '-'
29                 # Значением будет все, после минуса, если он был. И убираем
       ведущие нули, если они были
30                 self.value = x[self.is_neg:].lstrip('0')
31                 # Проверяем, является ли строка числом, если нет, value = 0
32                 if not self.value.isdigit():
33                     self.value = '0'
34                 if self.value == '0':
```

```
35                      self.is_neg = False
36          # Если в конструктор передан экземпляр того же класса, копируем его
    содержимое
37          elif isinstance(x, BigInt):
38              self.value = x.value
39              self.is_neg = x.is_neg
40
41      # Является ли число четным
42      def is_even(self):
43          """Является ли число четным"""
44          return not (int(self.value[-1]) & 1)
45
46      # Перегрузка числа по модулю
47      def __abs__(self):
48          return BigInt(self.value)
49
50      def bipow(self, n):
51          """Возведение числа в степень `n`"""
52          # Любое число в степени 0 = 1
53          if not n:
54              return BigInt(1)
55          if n & 1:
56              return BigInt(self.bipow(n - 1)) * self
57          tmp = BigInt(self.bipow(n // 2))
58          return BigInt(tmp * tmp)
59
60      def birt(self, n):
61          """Вычисление корня степени `n` из числа"""
62          # Корень извлекать можем только из положительного числа
63          if (n < 0) or self.is_neg:
64              return None
65          if n == 1:
66              return BigInt(self)
67          length = (len(self.value) + 1) // 2
68          index = 0
69          v = [0] * length
70          while index < length:
71              v[index] = 9
72              while BigInt(''.join(str(x) for x in v)).bipow(n) > self and v[
    index]:
73                  v[index] -= 1
74              index += 1
```

```
75          v = ''.join(str(x) for x in v).lstrip('0')
76          return BigInt('-' + v) if self.is_neg else BigInt(v)
77
78      # Перегрузка перевода в bool
79      def __bool__(self):
80          return self.value != '0'
81
82      # Перегрузка x < y
83      def __lt__(self, other):
84          self_len = len(self.value)  # Запоминаем длину первого числа
85          self_other = len(other.value)  # Запоминаем длину второго числа
86          # Если знаки одинаковые, то проверяем значения
87          if self.is_neg == other.is_neg:
88              # Если длины не равны
89              if self_len != self_other:
90                  # Меньше число с меньшей длинной для положительных и с
    большей длиной для отрицательных
91                  return (self_len < self_other) ^ self.is_neg
92              i = 0
93              # Ищем разряд, в котором значения отличаются
94              while (i < self_len and self.value[i] == other.value[i]):
95                  i += 1
96              # Если разряд найден, то меньше число с меньшей цифрой для
    положительных и с большей цифрой для отрицательных, иначе числа равны
97              return (i < self_len) and ((self.value[i] < other.value[i]) ^
    self.is_neg)
98          return self.is_neg  # Знаки разные, если число отрицательное, то оно
    меньше, если положительное, то больше
99
100     # Перегрузка x <= y
101     def __le__(self, other):
102         return self < other or self == other
103
104     # Перегрузка x == y
105     def __eq__(self, other):
106         return (self.value == other.value) and (self.is_neg == other.is_neg)
107
108     # Перегрузка x != y
109     def __ne__(self, other):
110         return not self == other
111
112     # Перегрузка x > y
113     def __gt__(self, other):
```

```
114         return not (self < other or self == other)
115
116     # Перегрузка x >= y
117     def __ge__(self, other):
118         return self > other or self == other
119
120     # Унарный плюс (просто копируем значение числа)
121     def __pos__(self):
122         return BigInt(self)
123
124     # Унарный минус
125     def __neg__(self):
126         return BigInt(self.value if self.is_neg else '-' + self.value)
127
128     # Возврат копии
129     def copy(self):
130         """Возврат копии"""
131         return BigInt(('-' if self.is_neg else '') + self.value)
132
133     # Сложение двух чисел
134     def __add__(self, other):
135         new_int = BigInt()
136         # Если знаки одинаковые, то выполняем сложение
137         if other.is_neg == self.is_neg:
138             num2 = other.value  # Запоминаем значение второго операнда
139             self_len = len(self.value)  # Длинна первого операнда
140             other_len = len(num2)  # Длинна второго операнда
141             # Длина суммы равна максимуму из двух длин + 1 из-за возможного
    переноса разряда
142             length = max(self_len, other_len) + 1
143             res = [0] * length
144             for i in range(length - 1):
145                 j = length - 1 - i
146                 # Выполняем сложение разрядов
147                 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'))
148                 res[j - 1] = res[j] // 10  # Выполняем перенос в следующий
    разряд, если он был
149                 res[j] = res[j] % 10  # Оставляем только единицы от
    возможного переноса и превращаем символ в цифру
150                 # Возвращаем результат, учитывая его знак
151             return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for
    x in res))
```

```
152         # Если одно из чисел отрицательное, а другое положительное, отправляем на вычитание, меняя знак
153         return (self - (-BigInt(other))) if self.is_neg else (other - (-BigInt(self)))
154
155     # Вычитание одного числа из другого
156     def __sub__(self, other):
157         # Если числа равны, считать не нужно
158         if self == other:
159             return BigInt(0)
160         # Если оба числа положительные, выполняем вычитание
161         if not self.is_neg and not other.is_neg:
162             self_len = len(self.value)  # Запоминаем длину первого числа
163             self_other = len(other.value)  # Запоминаем длину второго числа
164             length = max(self_len, self_other)  # Длина результата не превысит максимума длин чисел
165             is_neg_res = other > self  # Определяем знак результата
166             # Массивы аргументов
167             a = [0] * length
168             b = [0] * length
169             res = [0] * length
170             sign = 2 * is_neg_res - 1  # Получаем числовое значение знака результата
171             for i in range(length - 1):
172                 a[i] += int(self.value[self_len - 1 - i]) if i < self_len else 0  # Формируем разряды
173                 b[i] += int(other.value[self_other - 1 - i]) if i < self_other else 0  # Из строк аргументов
174                 b[i + 1] = -is_neg_res  # В зависимости от знака занимаем или не занимаем
175                 a[i + 1] = is_neg_res - 1  # 10 у следующего разряда
176                 res[length - 1 - i] += 10 + sign * (b[i] - a[i])
177                 res[length - 2 - i] = res[length - 1 - i] // 10
178                 res[length - 1 - i] = res[length - 1 - i] % 10
179             # Выполняем операцию с последним разрядом
180             a[length - 1] += (length - 1 < self_len) * int(self.value[0])
181             b[length - 1] += (length - 1 < self_other) * int(other.value[0])
182             # Записываем в строку последний разряд
183             res[0] += sign * (b[length - 1] - a[length - 1])
184             # Возвращаем результат, учитывая его знак
185             return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x in res))
```

```
186         return -BigInt(other) - (-BigInt(self)) if self.is_neg and other.
    is_neg else self + -BigInt(other)
187
188     # Умножение двух чисел
189     def __mul__(self, other):
190         # Если один из множителей равен нулю, то результат равен нулю
191         if self.value == '0' or other.value == '0':
192             return BigInt(0)
193         self_len = len(self.value)  # Запоминаем длину первого числа
194         other_len = len(other.value)  # Запоминаем длину второго числа
195         length = self_len + other_len + 1  # Результат влезет в сумму длин +
    1 из-за возможного переноса
196         # Флаг отрицательности результата - отрицательный, если числа разных
    знаков
197         is_neg_res = self.is_neg ^ other.is_neg
198         if length < 10:  # Число небольшое, можно по нормальному
199             res = int(self.value) * int(other.value)
200             return BigInt(-res if is_neg_res else res)
201         else:  # Умножаем в столбик
202             # Массивы аргументов
203             a = [0] * length
204             b = [0] * length
205             res = [0] * length
206             # Заполняем массивы инверсной записью чисел (с ведущими нулями)
207             for i in range(length):
208                 a[i] = int(self.value[self_len - 1 - i]) if i < self_len
    else 0
209                 b[i] = int(other.value[other_len - 1 - i]) if i < other_len
    else 0
210             # Выполняем умножение "в столбик"
211             for i in range(self_len):
212                 for j in range(other_len):
213                     res[length - 1 - (i + j)] += a[i] * b[j]
214                     res[length - 1 - (i + j + 1)] += res[length - 1 - (i + j
    )] // 10
215                     res[length - 1 - (i + j)] %= 10
216             # Возвращаем результат, учитывая его знак
217             return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for
    x in res))
218
219     # Деление одного числа на другое
220     def __truediv__(self, other):
221         value1 = self.value  # Запоминаем значение первого числа
```

```
222         value2 = other.value  # Запоминаем значение второго числа
223         if value2 == '0':
224             raise ZeroDivisionError  # Нельзя делить на ноль
225         if value1 == '0':
226             return BigInt(0)  # А вот ноль делить можно на всё, кроме нуля,
    но смысл
227         if value2 == '1':
228             return BigInt(-BigInt(self) if other.is_neg else BigInt(self))
    # Делить на 1 можно, но
    смысл?
229         zeroes = 0
230         while value2[len(value2) - 1 - zeroes] == '0':
231             zeroes += 1
232         if zeroes >= len(value1):
233             return BigInt(0)
234         # Избавляемся от общих нулей в конце чисел
235         if zeroes:
236             value1 = value1[:len(value1) - zeroes]
237             value2 = value2[:len(value2) - zeroes]
238         is_neg_res = self.is_neg ^ other.is_neg  # Считаем знак числа
239         tmp = BigInt(value2)
240         divider_length = len(value2)  # Запоминаем длину делителя
241         # Если длина больше 8, то обнуляем делитель, иначе переводим строку
    в long
242         # Можно не обнулять, но мы думаем, что Python не умеет в большие
    числа
243         divider_v = 0 if divider_length > 8 else int(value2)
244         length = len(value1)  # Получаем длину делимого
245         index = 0  # Стартуем с нулевого индекса
246         div = ''  # Строка результата деления
247         v = ''  # Строка подчисла (которое делится на делитель в столбик)
248         # Формируем начальное число для деления
249         while BigInt(v) < tmp and index < length:
250             v = v + value1[index]
251             index += 1
252         mod = None
253         while True:
254             count = 0  # Результат деления подчисла на делитель
255             # Если можем разделить, то делим
256             if BigInt(v) >= tmp:
257                 # Если не входит в long, то делим с помощью вычитания
258                 if divider_length > 8:
259                     mod = BigInt(v)
260                     while mod >= tmp:
```

```
261                         mod = (mod - tmp).copy()
262                         count += 1
263                     v = mod.value
264                 else:
265                     mod = int(v)
266                     count = mod // divider_v
267                     v = str(mod % divider_v)
268             # Если не делили, то добавили ноль к результату, иначе добавили
    результат деления
269             div = div + (str(count) if count else '0')
270             if index <= length:
271                 try:  # Тот самый ноль, лучше не спрашивать
272                     v = v + value1[index]
273                 except IndexError:
274                     v = v + '0'
275                 index += 1  # Формируем новое значение для подчисла
276             if not (index <= length):
277                 break
278         # Возвращаем результат учитывая знак и возможное равенство нулю
279         return BigInt('-' + div if is_neg_res and div != '0' else div)
280 
281     # Обработка для выходных данных
282     def __str__(self):
283         return str('-' if self.is_neg else '') + self.value
284 
285     # Остаток от деления
286     def __mod__(self, other):
287         if other.value == '0':
288             return None
289         if self.value == '0' or other.value == "1":
290             return BigInt(0)
291         # Если числа меньше 9, можно посчитать по нормальному
292         if len(self.value) < 9 and len(other.value) < 9:
293             res = int(self.value) % int(other.value)
294             return BigInt(-res if self.is_neg else res)
295         tmp = BigInt(other.value)
296         divider_length = len(other.value)  # запоминаем длину делителя
297         # Если длина больше 8, то обнуляем long'овый делитель, иначе
    переводим строку в long
298         divider_v = 0 if divider_length > 8 else int(other.value)
299         length = len(self.value)
300         index = 0
```

```
301         mod2 = self.copy()
302         v = ''
303         mod = None
304         while BigInt(v) < tmp and index < length:
305             v = v + self.value[index]
306             index += 1
307         while True:
308             if BigInt(v) >= tmp:
309                 if divider_v:
310                     v = str(int(v) % divider_v)
311                 else:
312                     mod = BigInt(v)
313                     while mod >= tmp:
314                         mod = (mod - tmp).copy()
315                     v = mod.value
316             if index <= length:
317                 mod2 = v
318                 try:
319                     v = v + self.value[index]
320                 except IndexError:
321                     return -BigInt(mod2) if self.is_neg else BigInt(mod2)
322                 index += 1
323             if not (index <= length):
324                 break
325         if mod2.value == '0':
326             return BigInt(0)
327         return -BigInt(mod2) if self.is_neg else BigInt(mod2)
328
329
330 def GCD(a, b):
331     """Нахождение наибольшего общего делителя у чисел `a` и `b`"""
332     while b:
333         a, b = b, a % b
334     return a
335
336
337 if __name__ == '__main__':
338     a = BigInt('1111111111111111111111111111111111111111')
339     b = BigInt('1111111111111111')
340     print(a / b)
```

8.2 Исходный код main.py

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  from sys import argv
5
6  from PyQt5 import QtCore, QtGui, QtWidgets
7
8  import about
9  import helpme
10 import ui
11 from bigint import GCD, BigInt
12
13
14 # Класс главной формы
15 class LongArithmeticCalc(QtWidgets.QMainWindow, ui.Ui_MainWindow):
16     def __init__(self):
17         super().__init__()
18         self.setupUi(self)
19
20         self.error_message = 'Ошибка ввода :('
21
22         # Events кнопок меню описания и выбора файла
23         self.addition.clicked.connect(self.addition_clicked)
       #
       +
24         self.subtraction.clicked.connect(self.subtraction_clicked)
       #
       -
25         self.multiplication.clicked.connect(self.multiplication_clicked)
       #
       *
26         self.division.clicked.connect(self.division_clicked)
       #
       /
27         self.exponentiation.clicked.connect(self.exponentiation_clicked)
       #
       степень
28         self.root.clicked.connect(self.root_clicked)
       #
       корень
29         self.gdc.clicked.connect(self.gdc_clicked)
       #
       НОД
```

```
30          self.remainder.clicked.connect(self.remainder_clicked)
        #
        OCT
31          self.about.clicked.connect(self.about_clicked)
32          self.helpme.clicked.connect(self.helpme_clicked)
33
34      def get_nums(self):
35          return self.first_num_edit.toPlainText(), self.second_num_edit.
        toPlainText()
36
37      def data_validation(self, a):
38          try:
39              int(a)
40              return True
41          except ValueError:
42              return False
43
44      def addition_clicked(self):  # +
45          first_num, second_num = self.get_nums()
46          if self.data_validation(first_num) and self.data_validation(
        second_num):
47              res = str(BigInt(first_num) + BigInt(second_num))
48              self.result.setText(res)
49          else:
50              self.result.setText(self.error_message)
51
52      def subtraction_clicked(self):  # -
53          first_num, second_num = self.get_nums()
54          if self.data_validation(first_num) and self.data_validation(
        second_num):
55              res = str(BigInt(first_num) - BigInt(second_num))
56              self.result.setText(res)
57          else:
58              self.result.setText(self.error_message)
59
60      def multiplication_clicked(self):  # *
61          first_num, second_num = self.get_nums()
62          if self.data_validation(first_num) and self.data_validation(
        second_num):
63              res = str(BigInt(first_num) * BigInt(second_num))
64              self.result.setText(res)
65          else:
```

```
66              self.result.setText(self.error_message)
67
68      def division_clicked(self):  # /
69          first_num, second_num = self.get_nums()
70          if self.data_validation(first_num) and self.data_validation(
    second_num):
71              res = str(BigInt(first_num) / BigInt(second_num))
72              self.result.setText(res)
73          else:
74              self.result.setText(self.error_message)
75
76      def exponentiation_clicked(self):  # степень
77          first_num, second_num = self.get_nums()
78          if self.data_validation(first_num) and self.data_validation(
    second_num):
79              res = str(BigInt(first_num).bipow(int(second_num)))
80              self.result.setText(res)
81          else:
82              self.result.setText(self.error_message)
83
84      def root_clicked(self):  # корень
85          first_num, second_num = self.get_nums()
86          if self.data_validation(first_num) and self.data_validation(
    second_num):
87              res = str(BigInt(first_num).birt(int(second_num)))
88              self.result.setText(res)
89          else:
90              self.result.setText(self.error_message)
91
92      def gdc_clicked(self):  # НОД
93          first_num, second_num = self.get_nums()
94          if self.data_validation(first_num) and self.data_validation(
    second_num):
95              res = str(GCD(BigInt(first_num), BigInt(second_num)))
96              self.result.setText(res)
97          else:
98              self.result.setText(self.error_message)
99
100     def remainder_clicked(self):  # ОСТ
101         first_num, second_num = self.get_nums()
```

```
102         if self.data_validation(first_num) and self.data_validation(
    second_num):
103             res = str(BigInt(first_num) % BigInt(second_num))
104             self.result.setText(res)
105         else:
106             self.result.setText(self.error_message)
107 
108     def about_clicked(self):  # about
109         about_dialog = about()
110         about_dialog.exec_()
111 
112     def helpme_clicked(self):  # Помощь
113         helpme_dialog = helpme()
114         helpme_dialog.exec_()
115 
116 
117 class about(QtWidgets.QDialog, about.Ui_Dialog):
118     def __init__(self):
119         super().__init__()
120         self.setupUi(self)
121         pixmap = QtGui.QPixmap('./img/math_logo.png')
122         self.ejik.setPixmap(pixmap)
123         self.resize(pixmap.width(), pixmap.height())
124 
125 
126 class helpme(QtWidgets.QDialog, helpme.Ui_helpme):
127     def __init__(self):
128         super().__init__()
129         self.setupUi(self)
130 
131 
132 def main():
133     app = QtWidgets.QApplication(argv)
134     main_window = LongArithmeticCalc()
135     main_window.show()
136     app.exec_()
137 
138 
139 if __name__ == "__main__":
140     main()
```