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»;

- Вычитание целых длинных чисел.
  Программная реализация представляет собой вычитание чисел в «столбик»;

- Умножение целых длинных чисел.
  Программная реализация представляет собой умножение чисел в «столбик»;

- Целочисленное деление целых длинных чисел.
  Программная реализация представляет собой деление чисел в «столбик» без дробной части;

- 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;

- 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;

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.

```
a = BigInt()
```

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

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

- Создание экземпляра класса с передачей в аргумент целого числа.

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

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

```
a = BigInt(-1234567890)  # a = -1234567890
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.

```
a = BigInt('-1234567890987654321')
b = BigInt('9876543210123456789')
zero = BigInt()
```

- Выполним сложение:

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

  Вывод: 8641975319135802468

- Выполним вычитание:

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

  Вывод: -11111111101111111110

- Выполним умножение:

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

  Вывод: $-12193263121170553265523548251112635269$

- Выполним целочисленное деление:

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

  Вывод: -8

- Выполним нахождение остатка от деления:

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

  Вывод: 82222222221

- Выполним нахождение НОД:

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

Вывод: -9

- Выполняем возведение в степень:

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

Вывод:
676549457811317882533991394769509398672138473842215107823721833873638355493281821600537941161589640231883946397584166318795047266740645217094738013218419327830527872057771151857381511749913528561012262166688559508579257490958716867835714521994219774652466799202500334886202595310153316368905234601302744391232730289077240646312505876707771713512612446512064624010

- Выполним деление на ноль:

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

Вывод: *ZeroDivisionError*

- Выполним деление нуля:

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

Вывод: 0

- Выполним умножение на ноль:

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

Вывод: 0

- Выполняем возведение в степень ноль:

```
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]

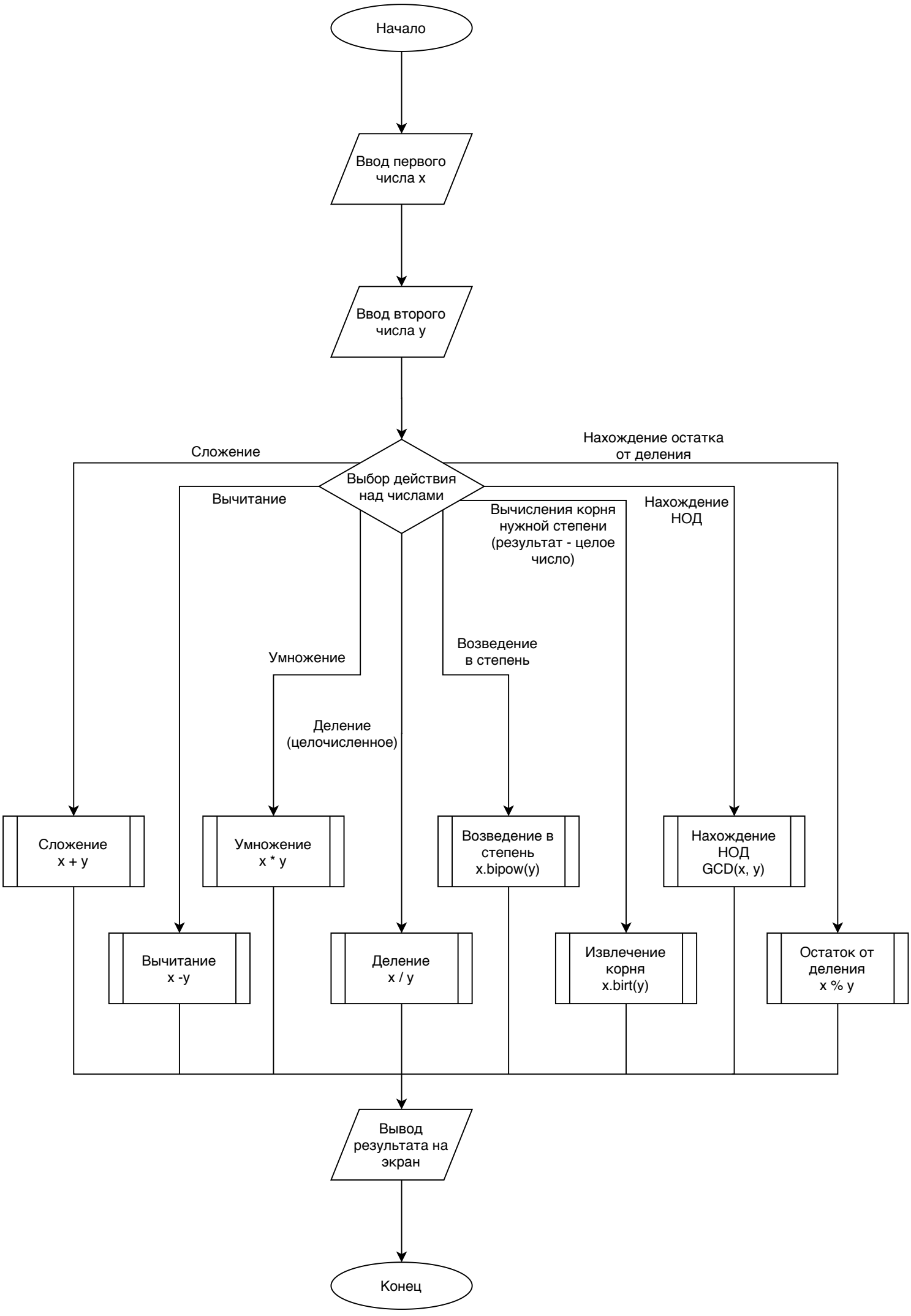
4. В случае нахождения НОД программа ищет наибольший общий делитель чисел.

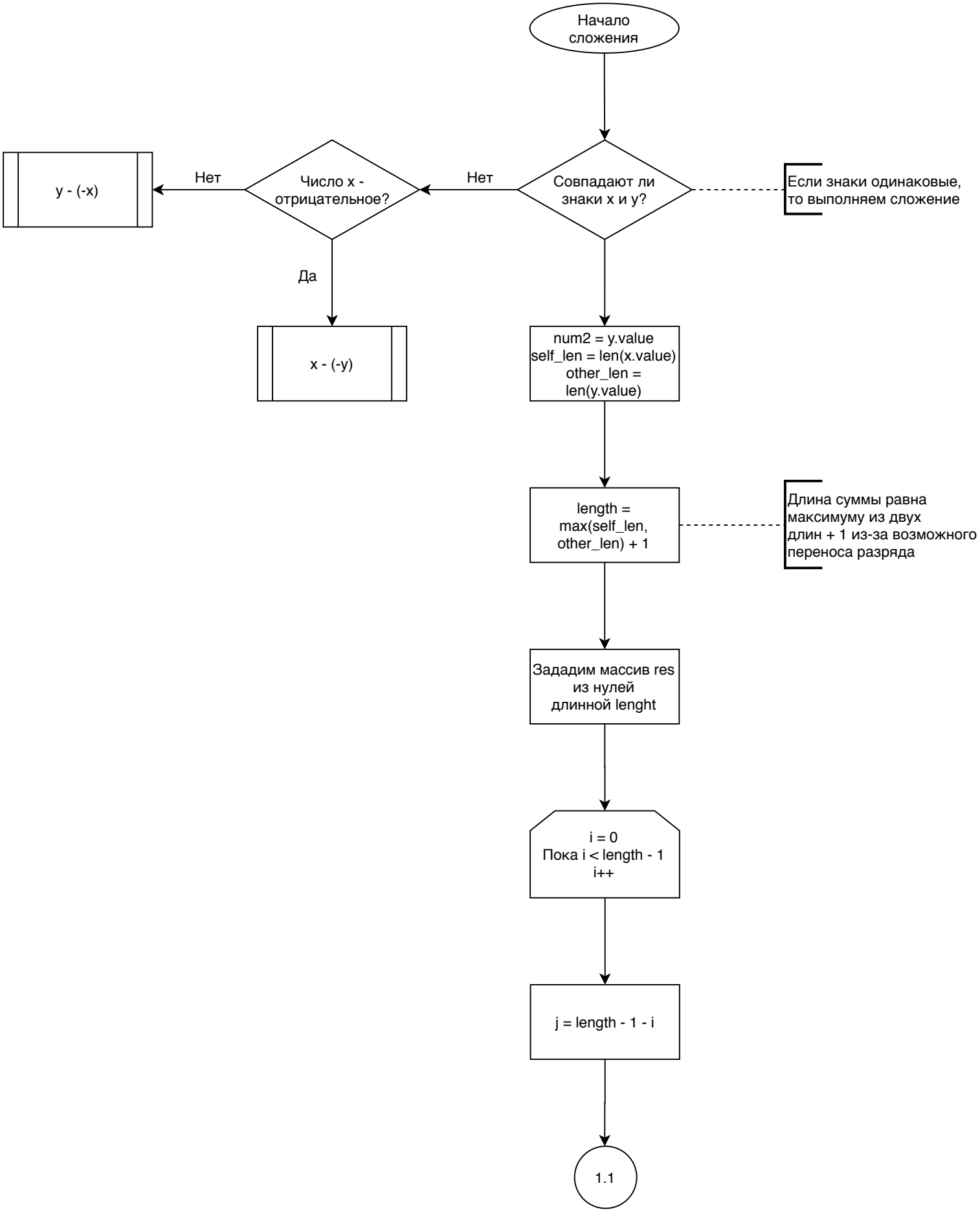
5. В случае нахождения НОД программа ищет остаток от деление первого числа на второе число.

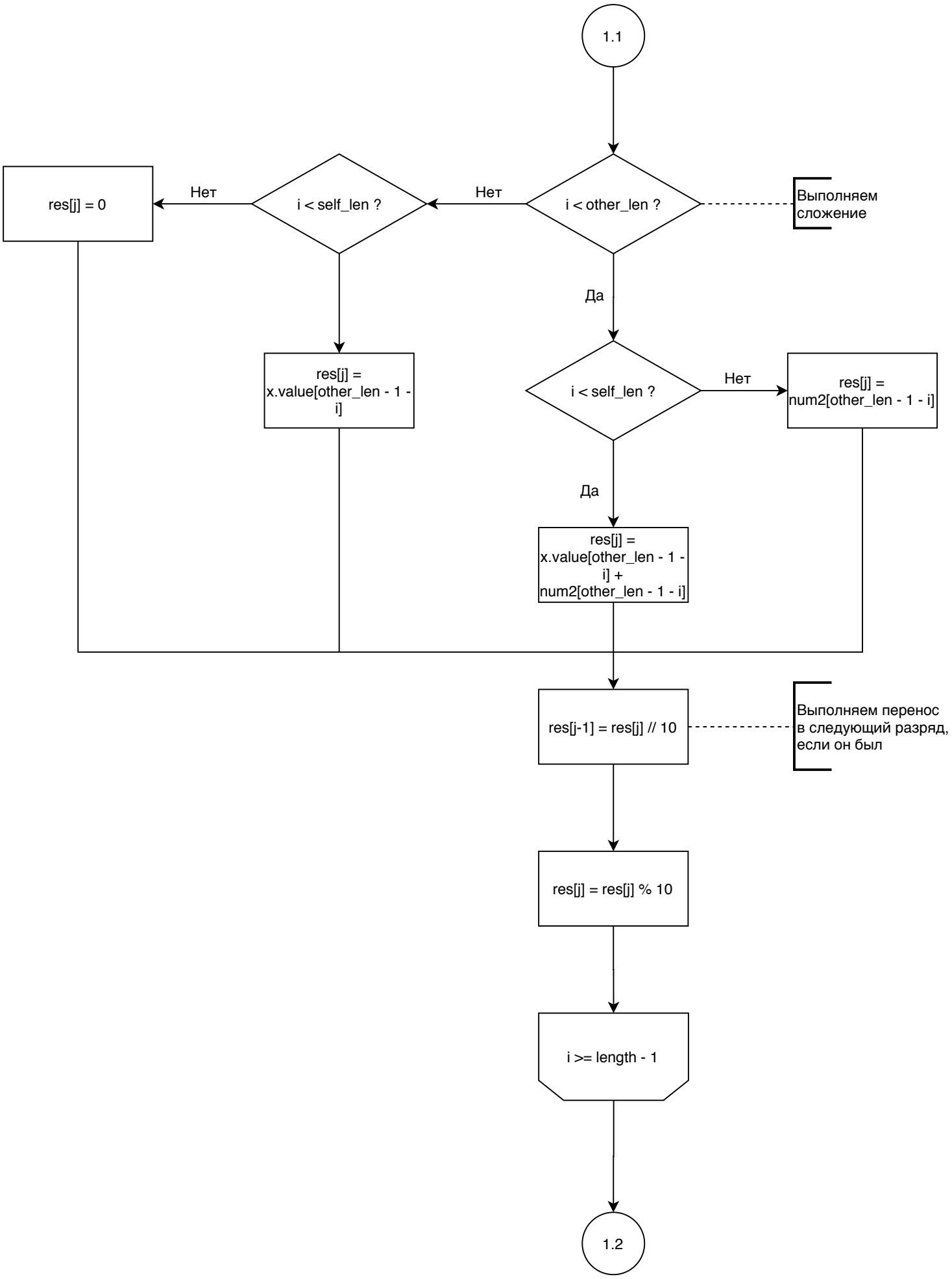
7 Блок-схема методов библиотеки

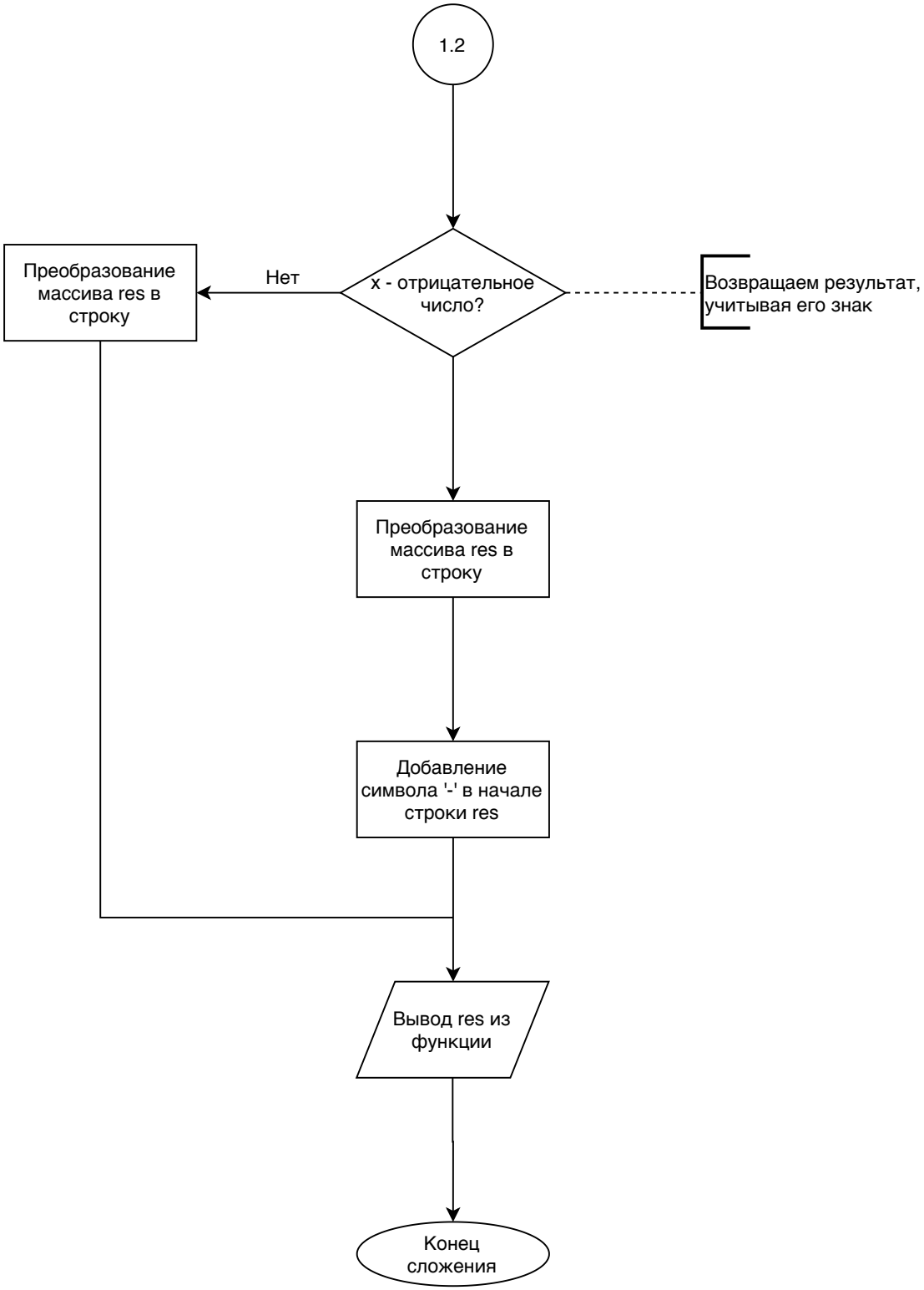
Ниже представлены блок-схемы методов в следующем порядке:

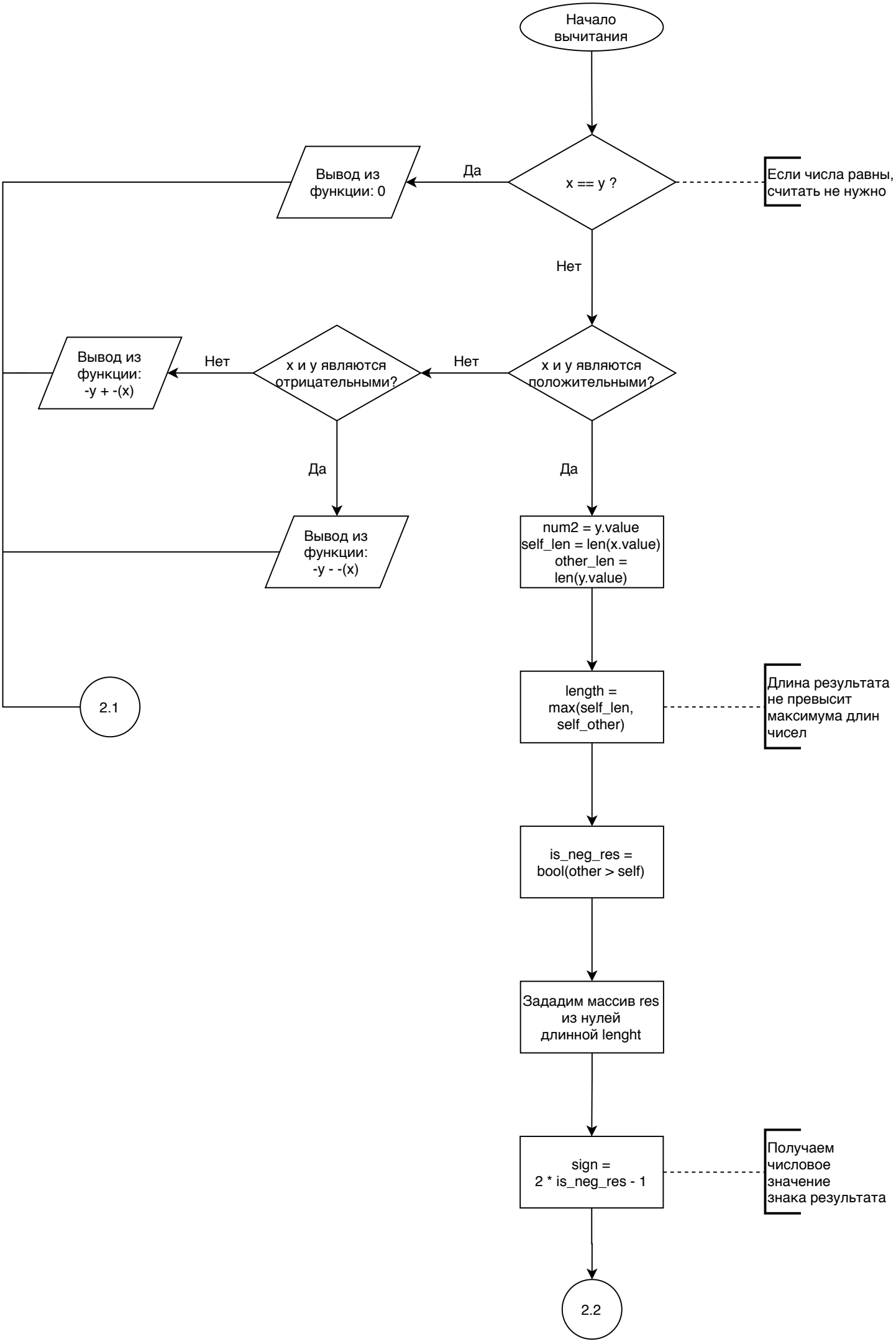
1. Логика работы интерфейса;
2. Сложение;
3. Вычитание;
4. Умножение;
5. Деление;
6. Возведение в степень;
7. Извлечение корня;
8. Нахождение НОД;
9. Нахождение остатка от деления.

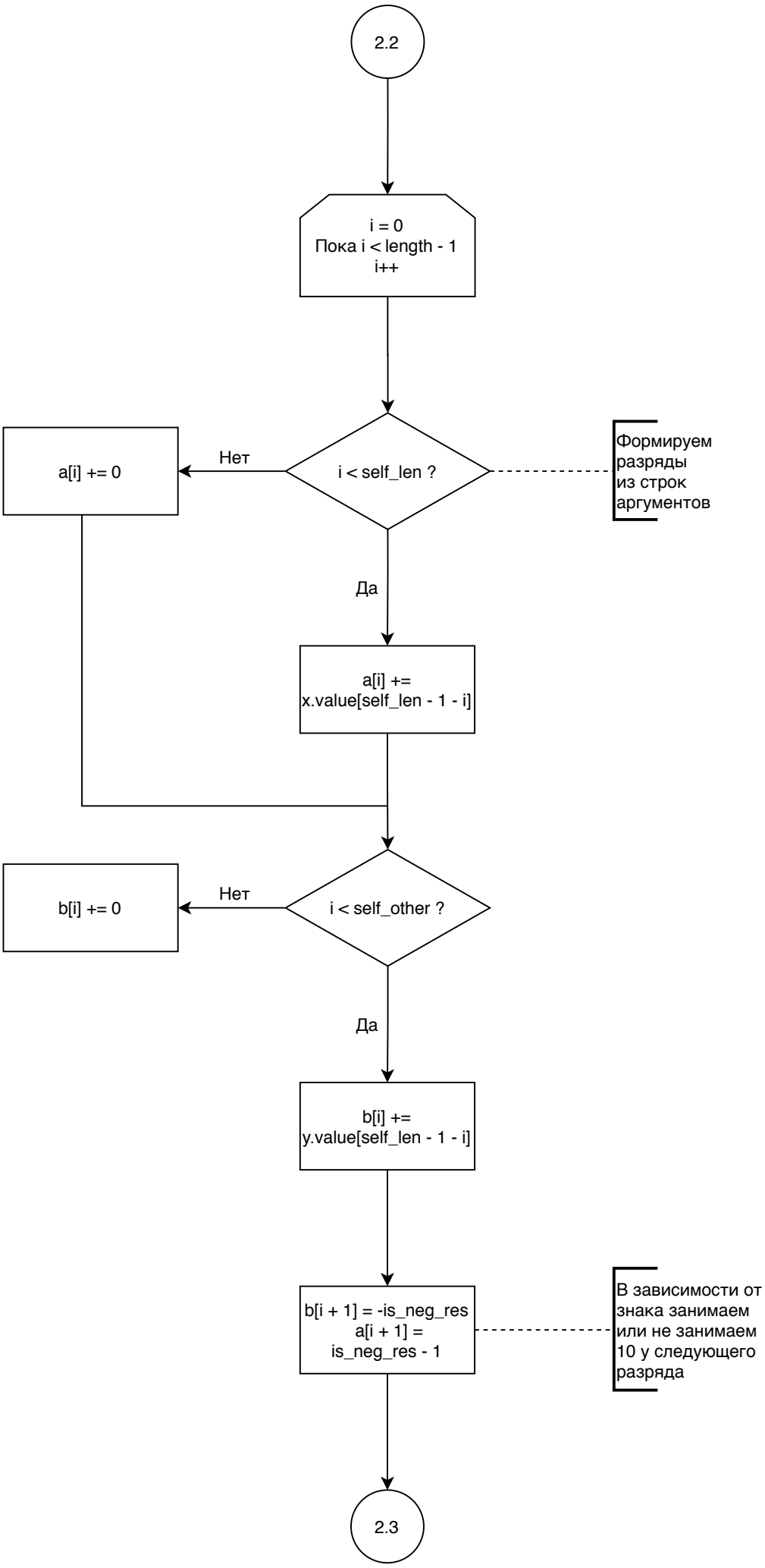

Начало
Ввод первого числа x
Ввод второго числа y
Выбор действия над числами
Сложение
Вычитание
Умножение
Деление (целочисленное)
Возведение в степень
Вычисления корня нужной степени (результат - целое число)
Нахождение НОД
Нахождение остатка от деления
Сложение x + y
Умножение x * y
Возведение в степень 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

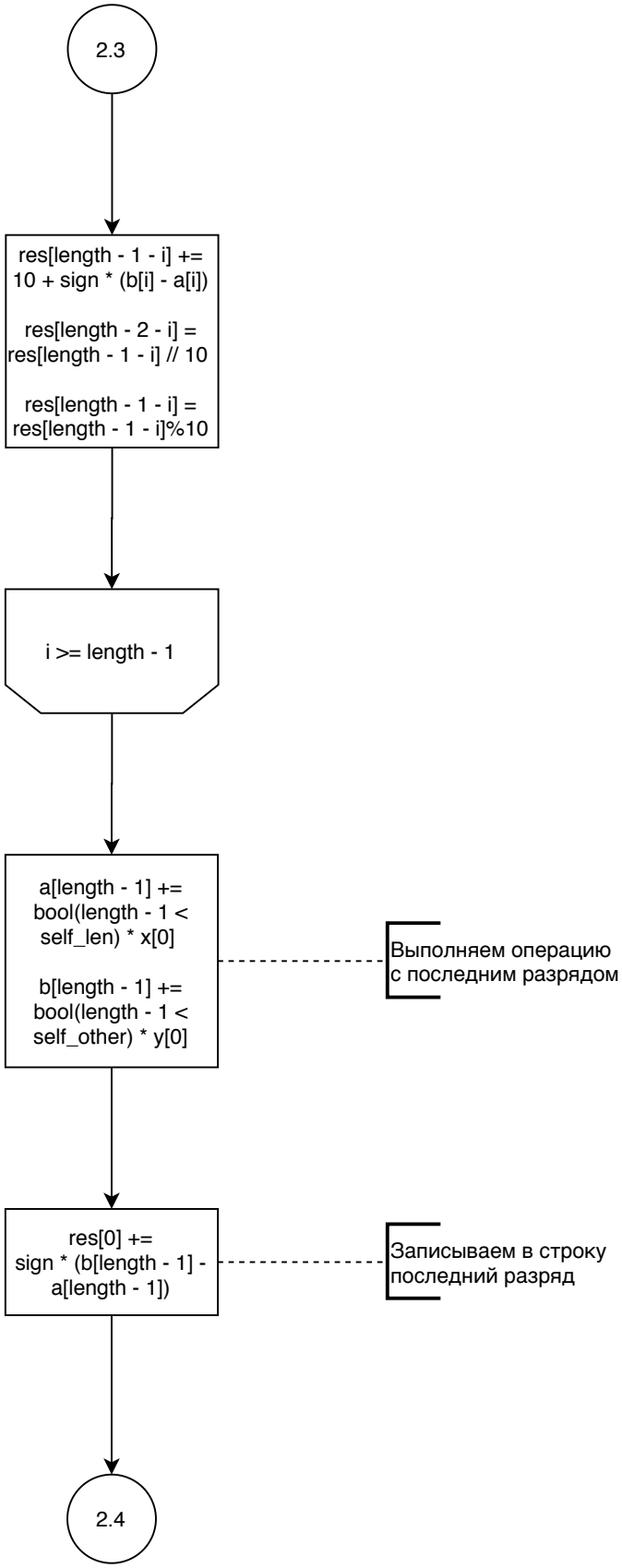

Начало сложения
Совпадают ли знаки x и y?
Нет
Число x - отрицательное?
Нет
y - (-x)
Да
x - (-y)
Если знаки одинаковые, то выполняем сложение
num2 = y.value
self_len = len(x.value)
other_len = len(y.value)
length = max(self_len, other_len) + 1
Длина суммы равна максимуму из двух длин + 1 из-за возможного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

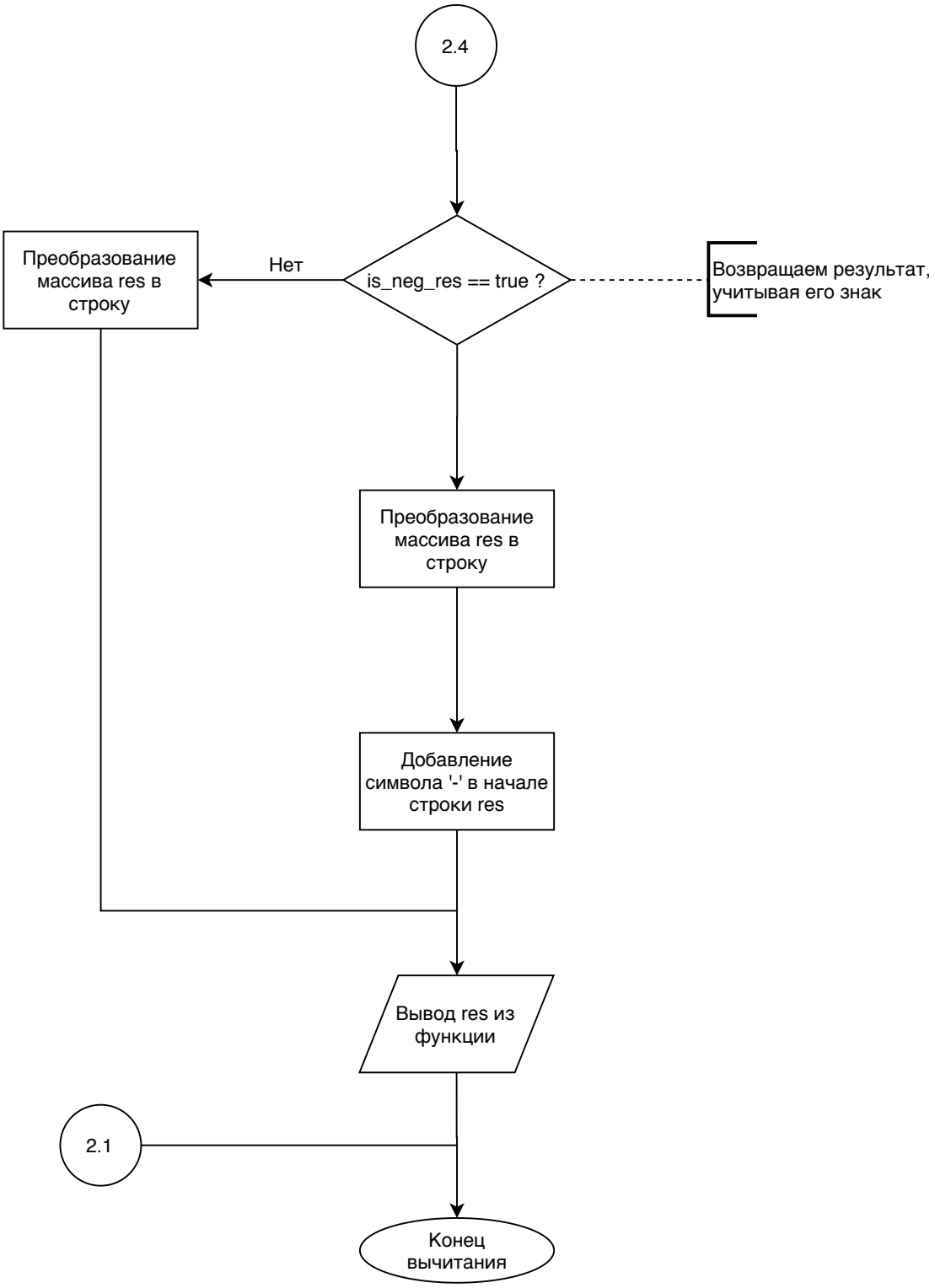

1.1
i < other_len ?
Нет
Выполняем сложение
Да
i < self_len ?
Нет
res[j] = num2[other_len - 1 - i]
Да
res[j] = x.value[other_len - 1 - i] + num2[other_len - 1 - i]
i < self_len ?
Нет
res[j] = x.value[other_len - 1 - i]
res[j] = 0
res[j-1] = res[j] // 10
Выполняем перенос в следующий разряд, если он был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

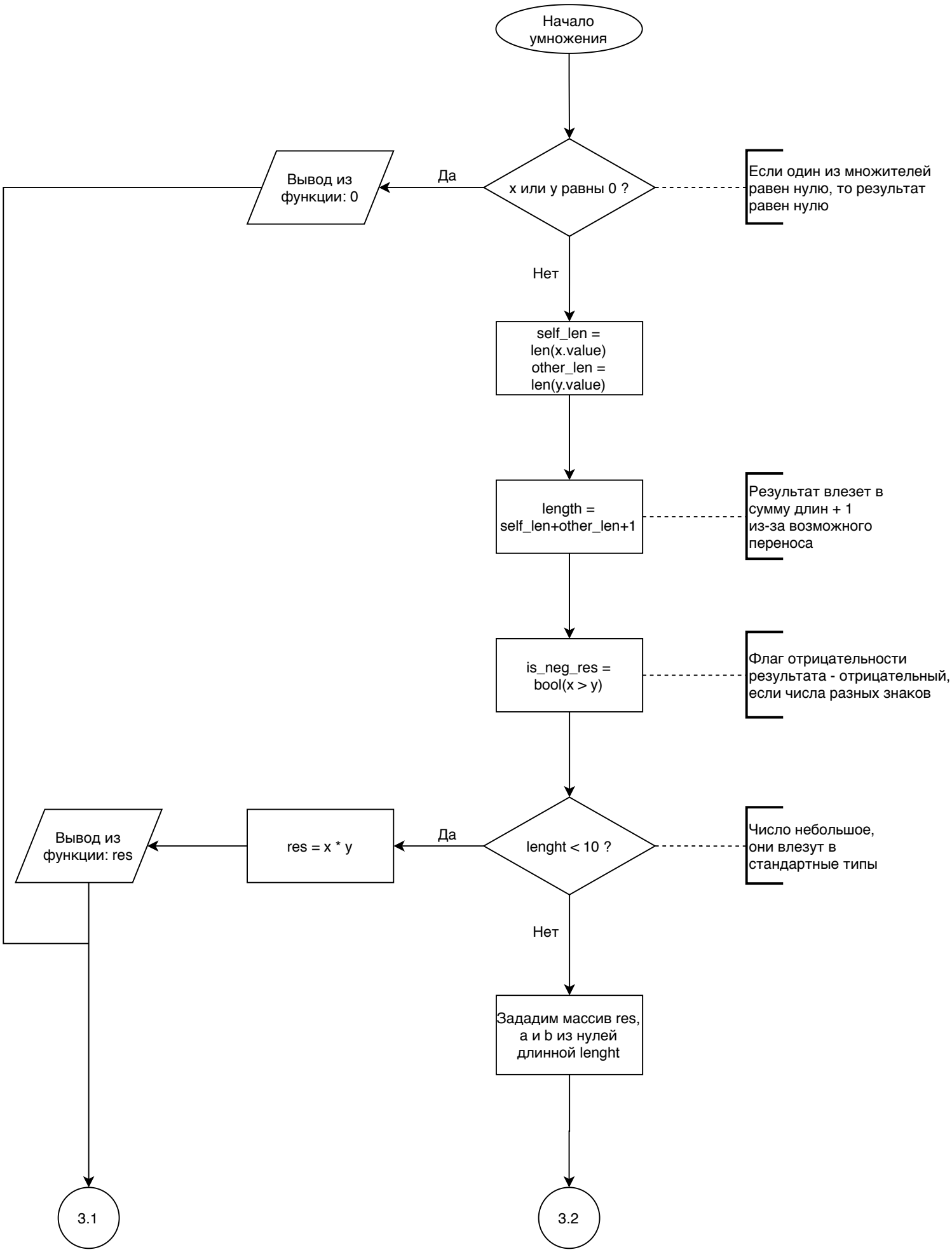

1.2
x - отрицательное число?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
Конец сложения

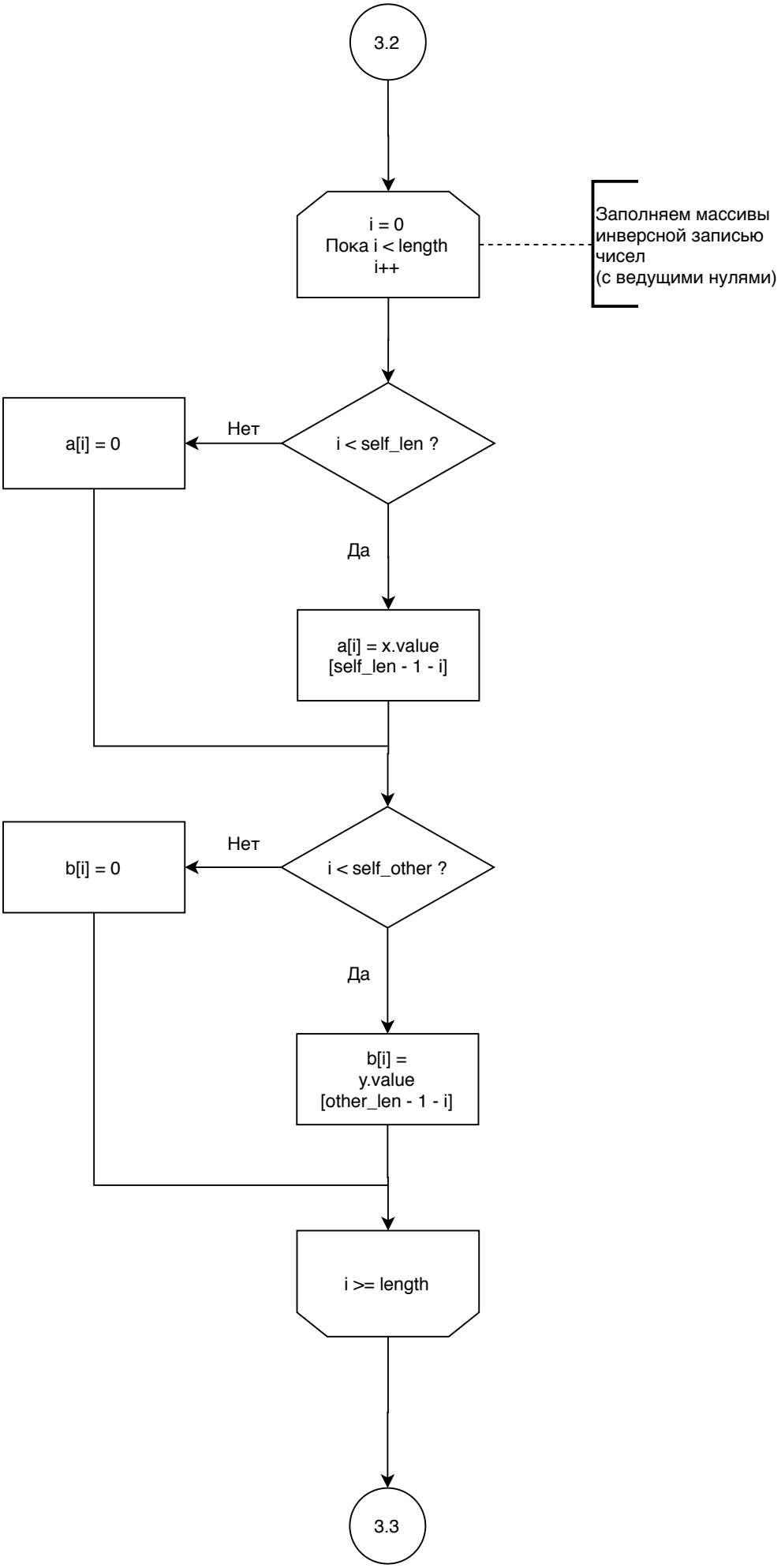

Начало вычитания
x == y ?
Да
Вывод из функции: 0
Если числа равны, считать не нужно
Нет
x и y являются положительными?
Нет
x и y являются отрицательными?
Нет
Вывод из функции: -y + -(x)
Да
Вывод из функции: -y - -(x)
Да
num2 = y.value
self_len = len(x.value)
other_len = len(y.value)
length = max(self_len,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

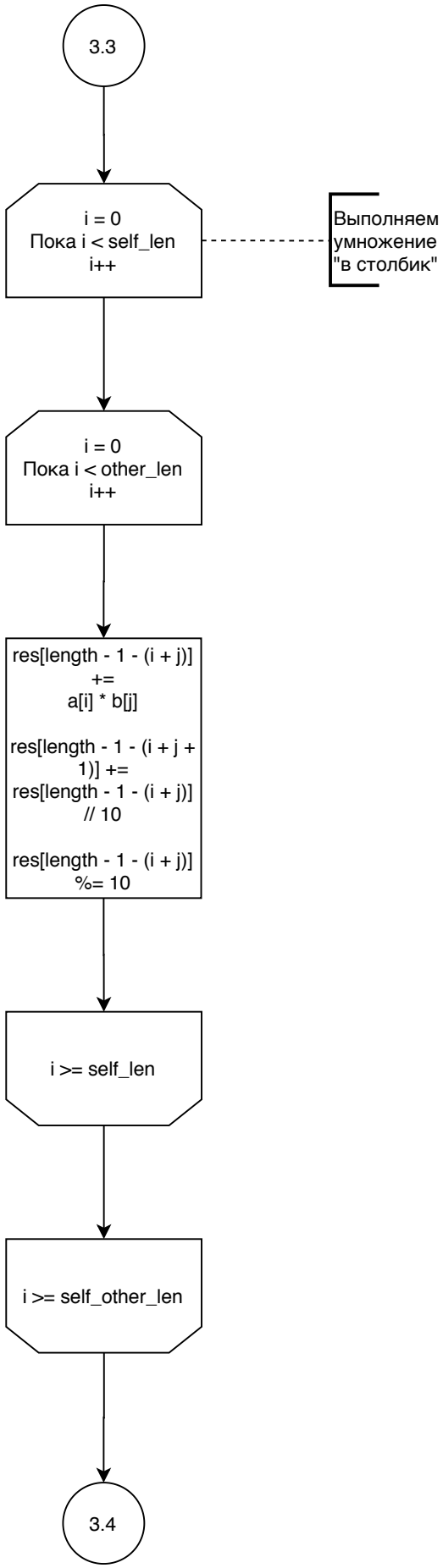

2.2
i = 0
Пока i < length - 1
i++
i < self_len ?
Нет
a[i] += 0
Да
Формируем разряды из строк аргументов
a[i] += x.value[self_len - 1 - i]
i < self_other ?
Нет
b[i] += 0
Да
b[i] += y.value[self_len - 1 - i]
b[i + 1] = -is_neg_res
a[i + 1] = is_neg_res - 1
В зависимости от знака занимаем или не занимаем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

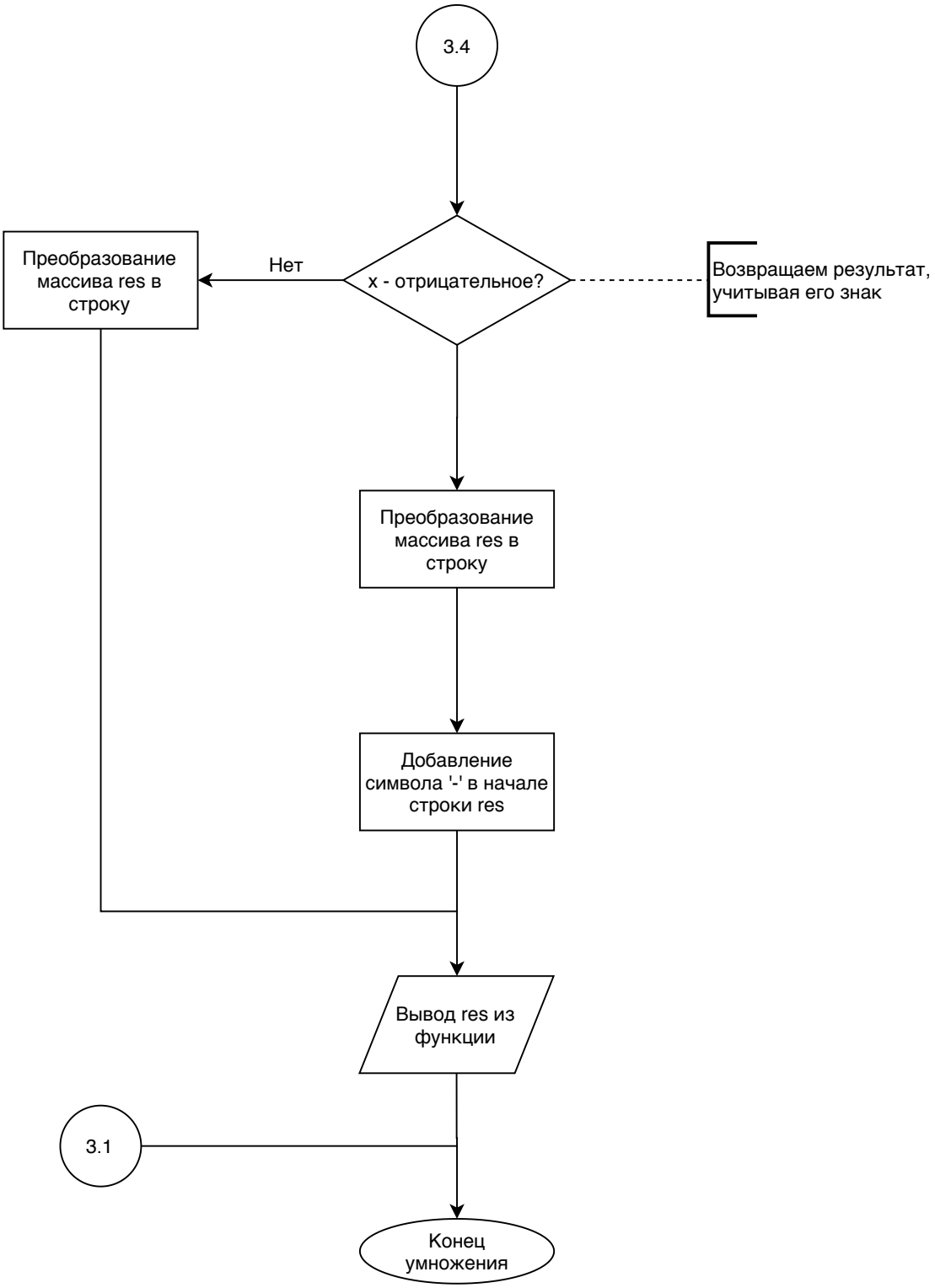

2.3
res[length - 1 - i] += 10 + sign * (b[i] - a[i])
res[length - 2 - i] = res[length - 1 - i] // 10
res[length - 1 - i] = res[length - 1 - i]%10
i >= length - 1
a[length - 1] += bool(length - 1 < self_len) * x[0]
b[length - 1] += bool(length - 1 < self_other) * y[0]
Выполняем операцию с последним разрядом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

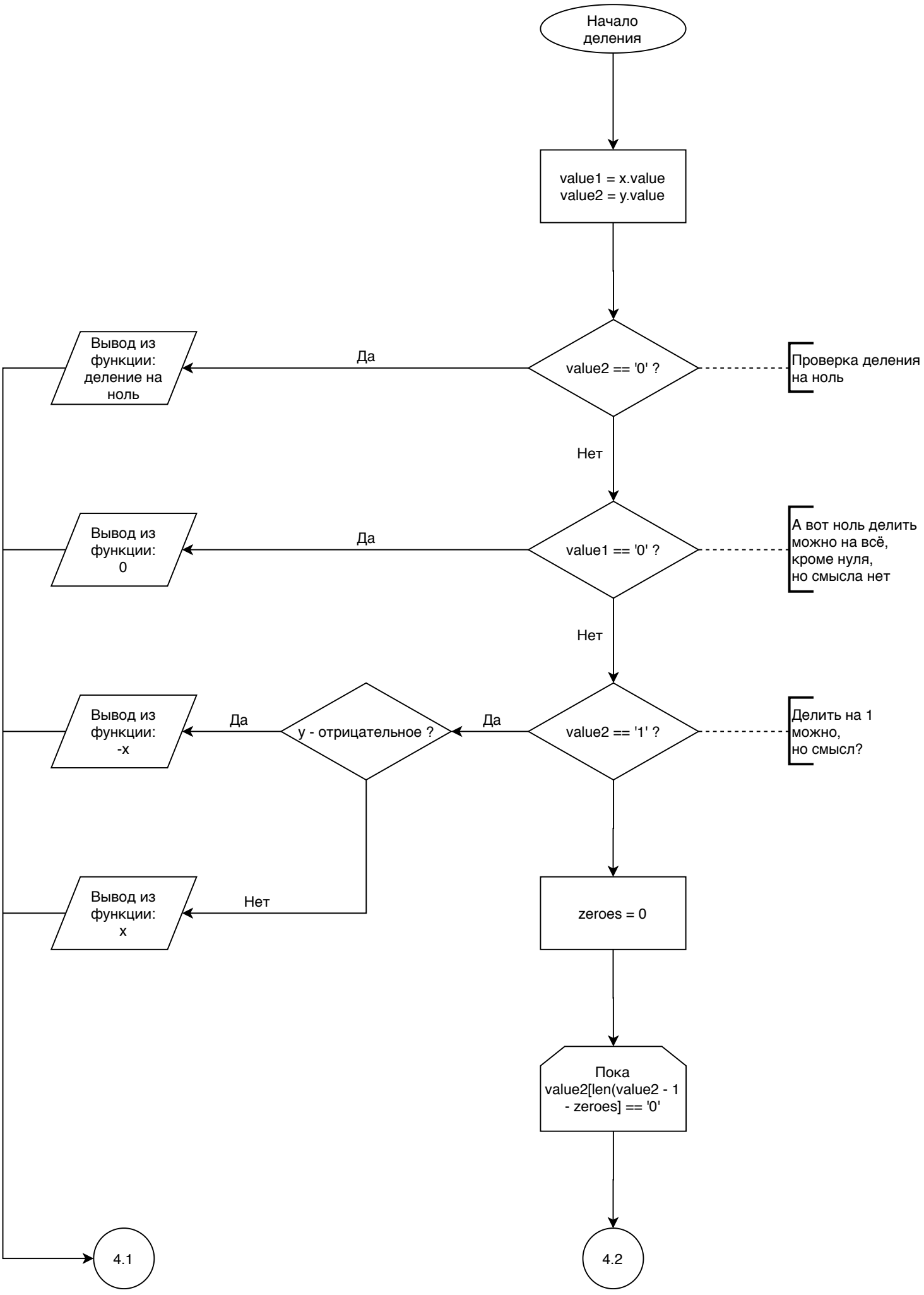

2.4
is_neg_res == true ?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
2.1
Конец вычитания

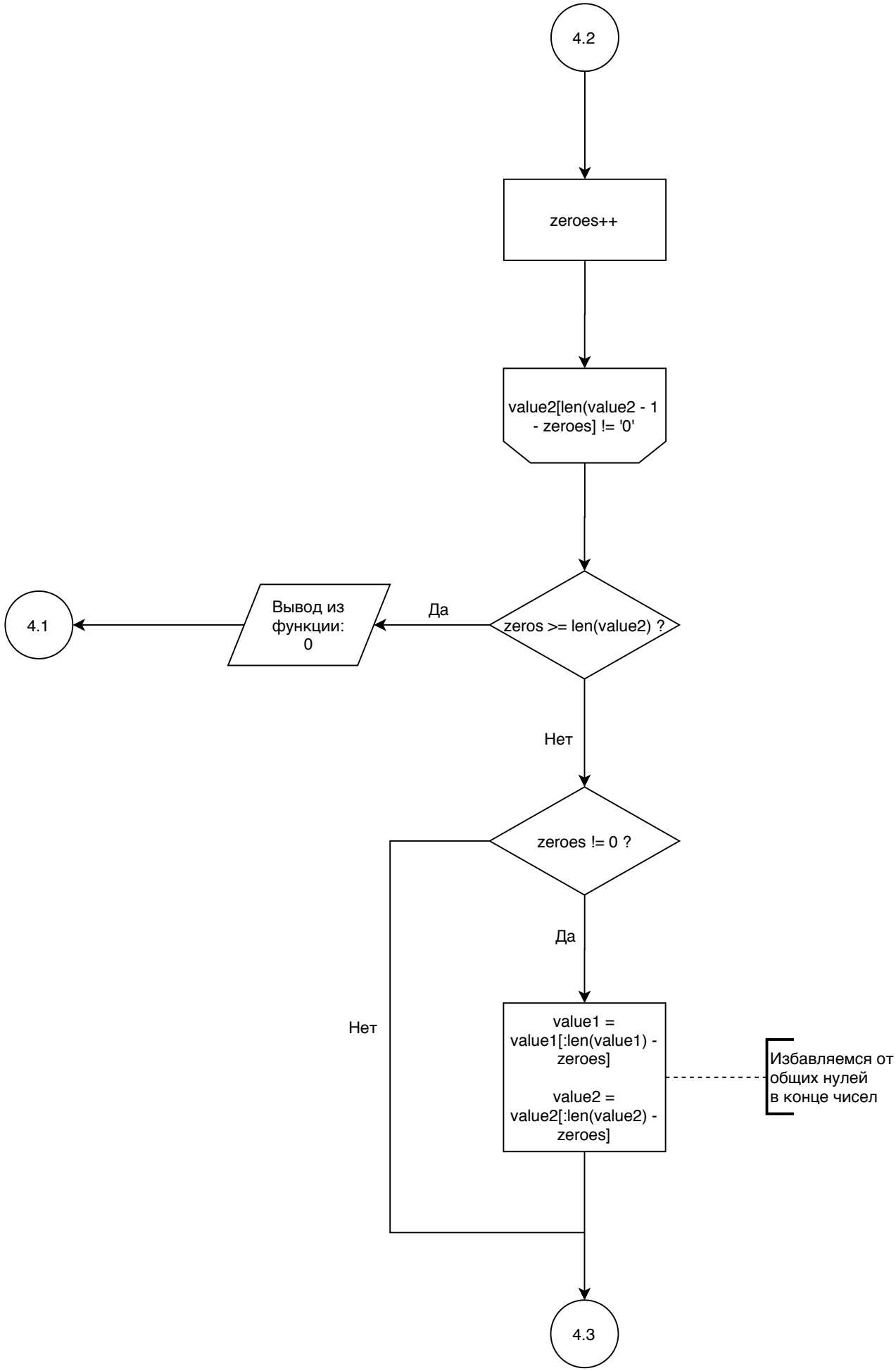

Начало умножения
x или y равны 0 ?
Да
Вывод из функции: 0
Если один из множителей равен нулю, то результат равен нулю
Нет
self_len = len(x.value) other_len = len(y.value)
length = self_len+other_len+1
Результат влезет в сумму длин + 1 из-за возможного переноса
is_neg_res = bool(x > y)
Флаг отрицательности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

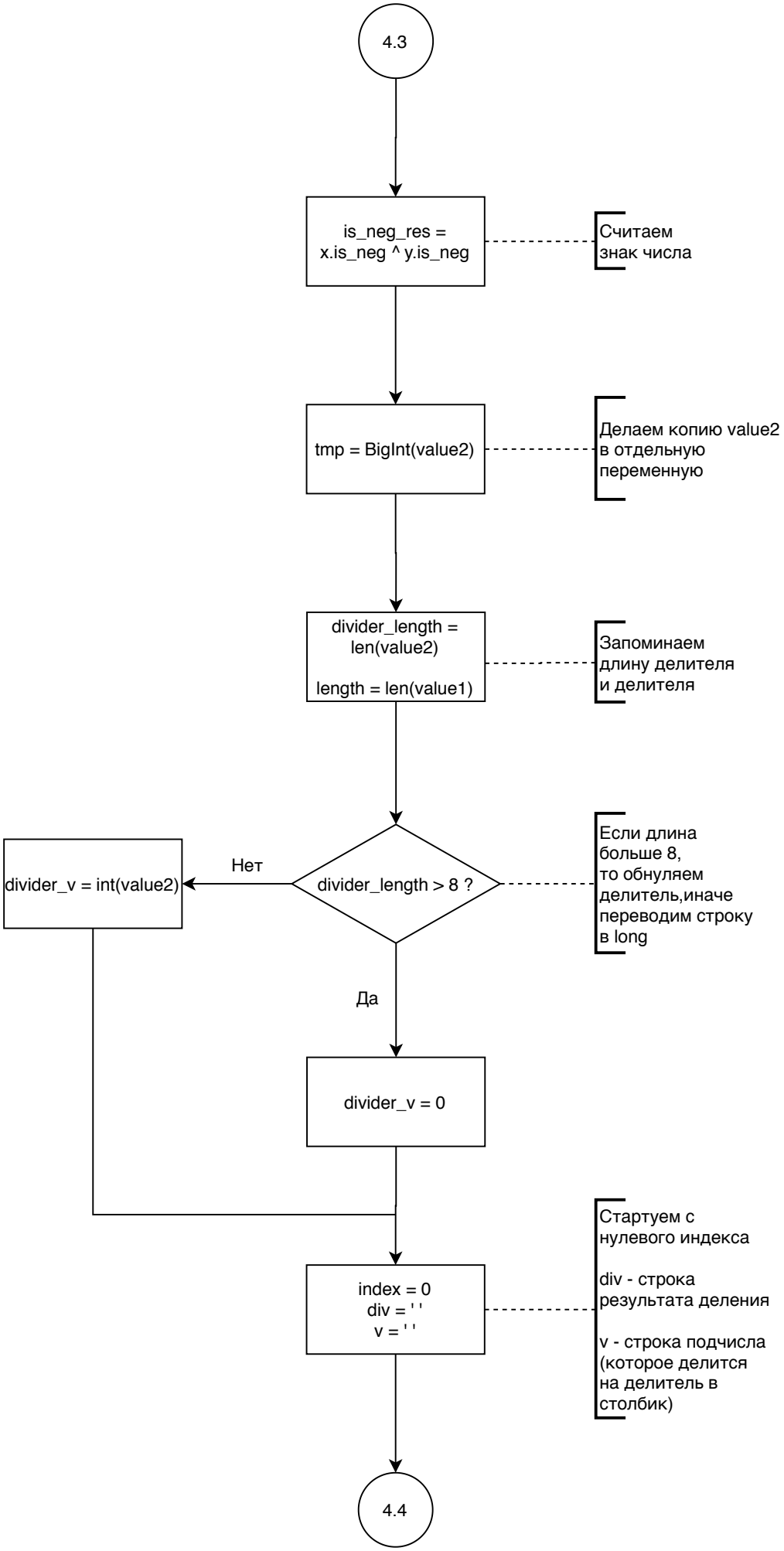

3.2
i = 0
Пока i < length
i++
Заполняем массивы
инверсной записью
чисел
(с ведущими нулями)
i < self_len ?
Нет
a[i] = 0
Да
a[i] = x.value
[self_len - 1 - i]
i < self_other ?
Нет
b[i] = 0
Да
b[i] =
y.value
[other_len - 1 - i]
i >= length
3.3

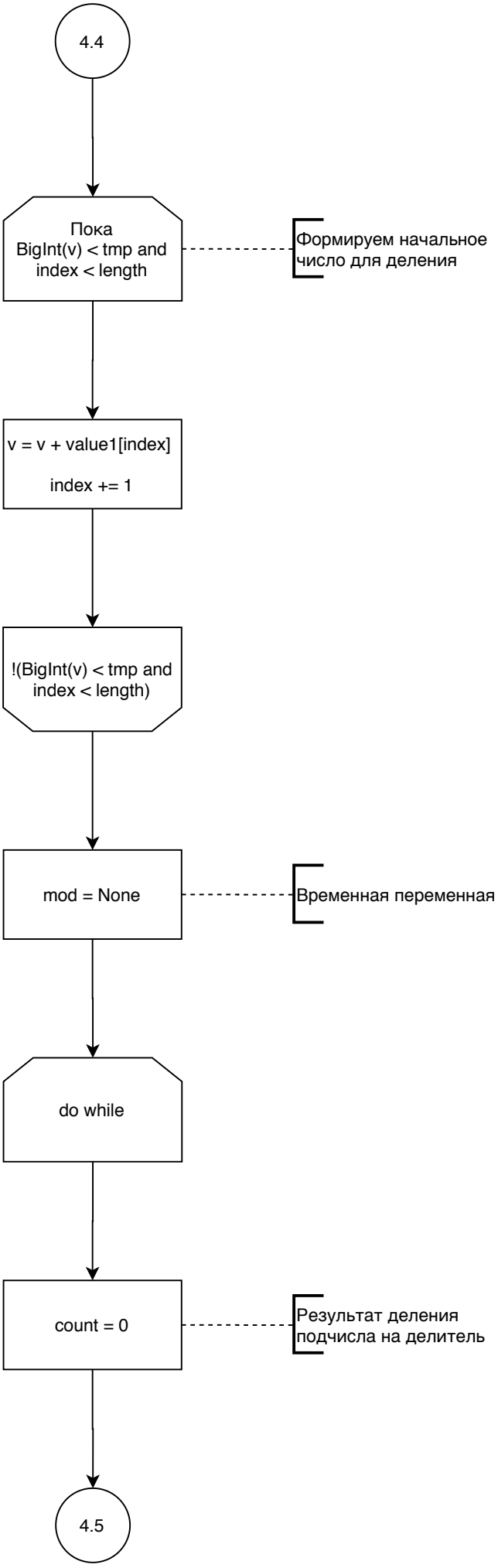

3.3
i = 0
Пока i < self_len
i++
Выполняем умножение "в столбик"
i = 0
Пока i < other_len
i++
res[length - 1 - (i + j)] += a[i] * b[j]
res[length - 1 - (i + j + 1)] += res[length - 1 - (i + j)] // 10
res[length - 1 - (i + j)] %= 10
i >= self_len
i >= self_other_len
3.4

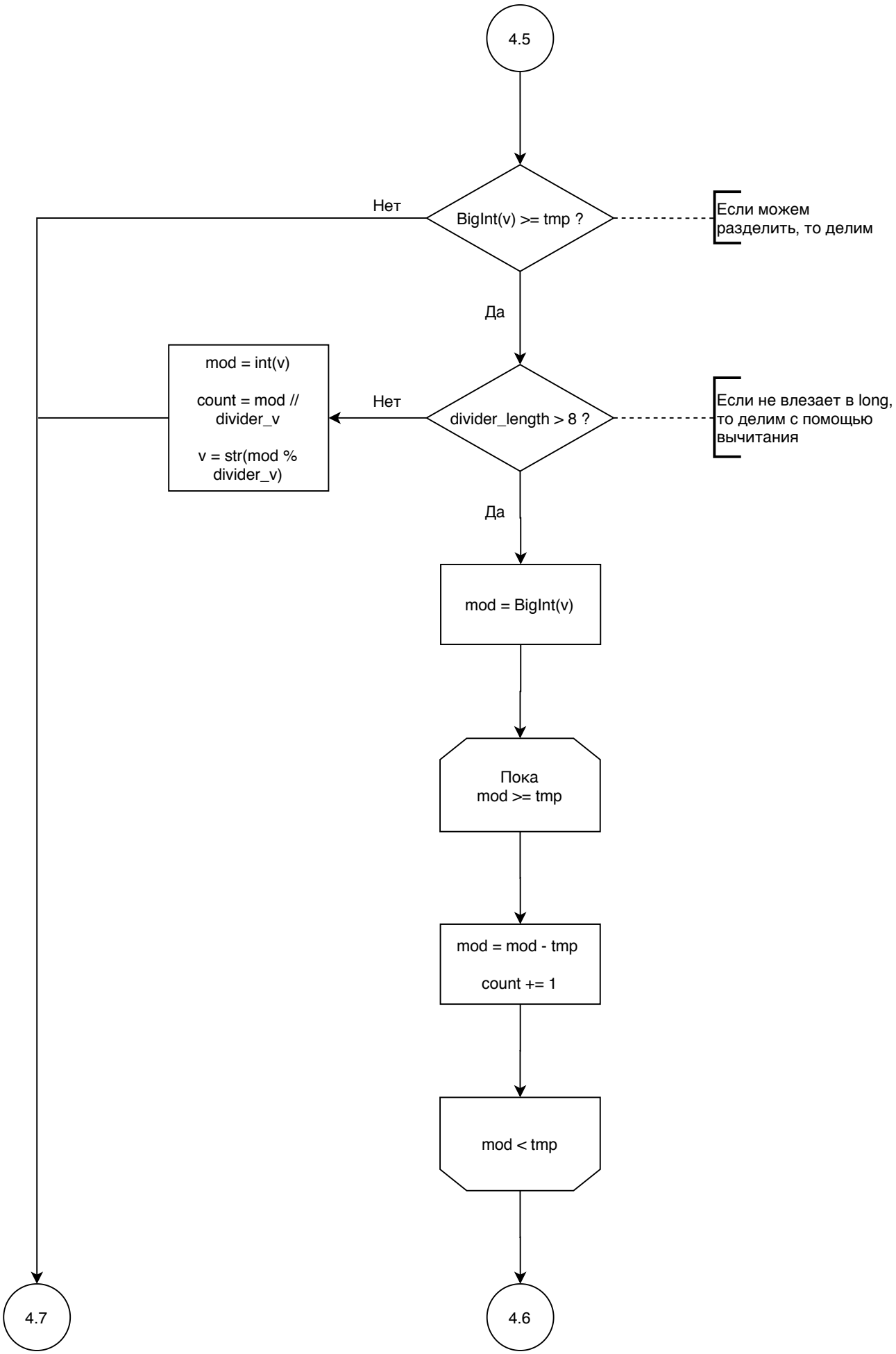

3.4
x - отрицательное?
Нет
Преобразование массива res в строку
Возвращаем результат, учитывая его знак
Преобразование массива res в строку
Добавление символа '-' в начале строки res
Вывод res из функции
3.1
Конец умножения

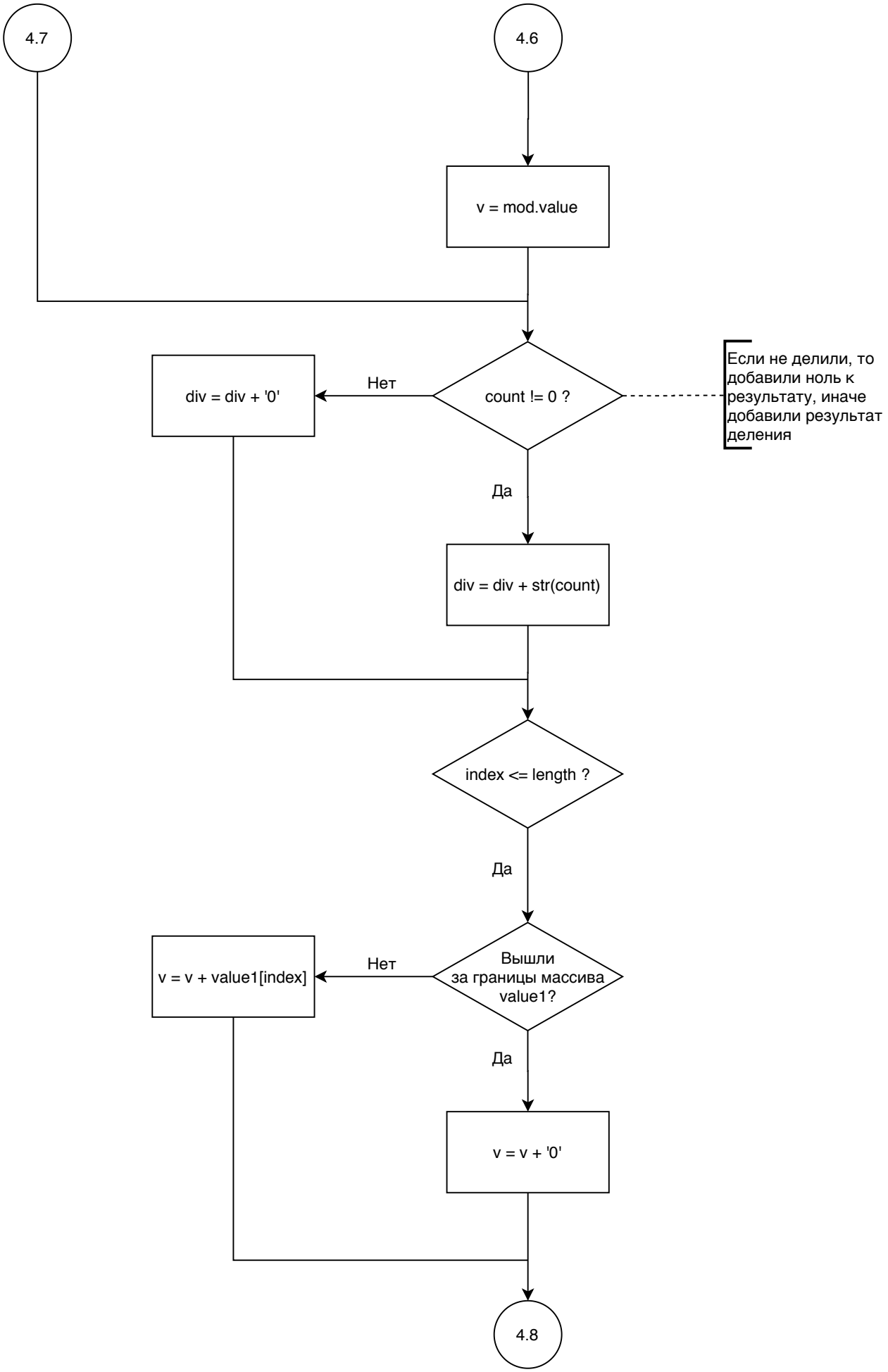

Начало деления
value1 = x.value
value2 = y.value
value2 == '0' ?
Да
Вывод из функции: деление на ноль
Проверка деления на ноль
Нет
value1 == '0' ?
Да
Вывод из функции: 0
А вот ноль делить можно на всё, кроме нуля, но смысла нет
Нет
value2 == '1' ?
Да
y - отрицательное ?
Да
Вывод из функции: -x
Делить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

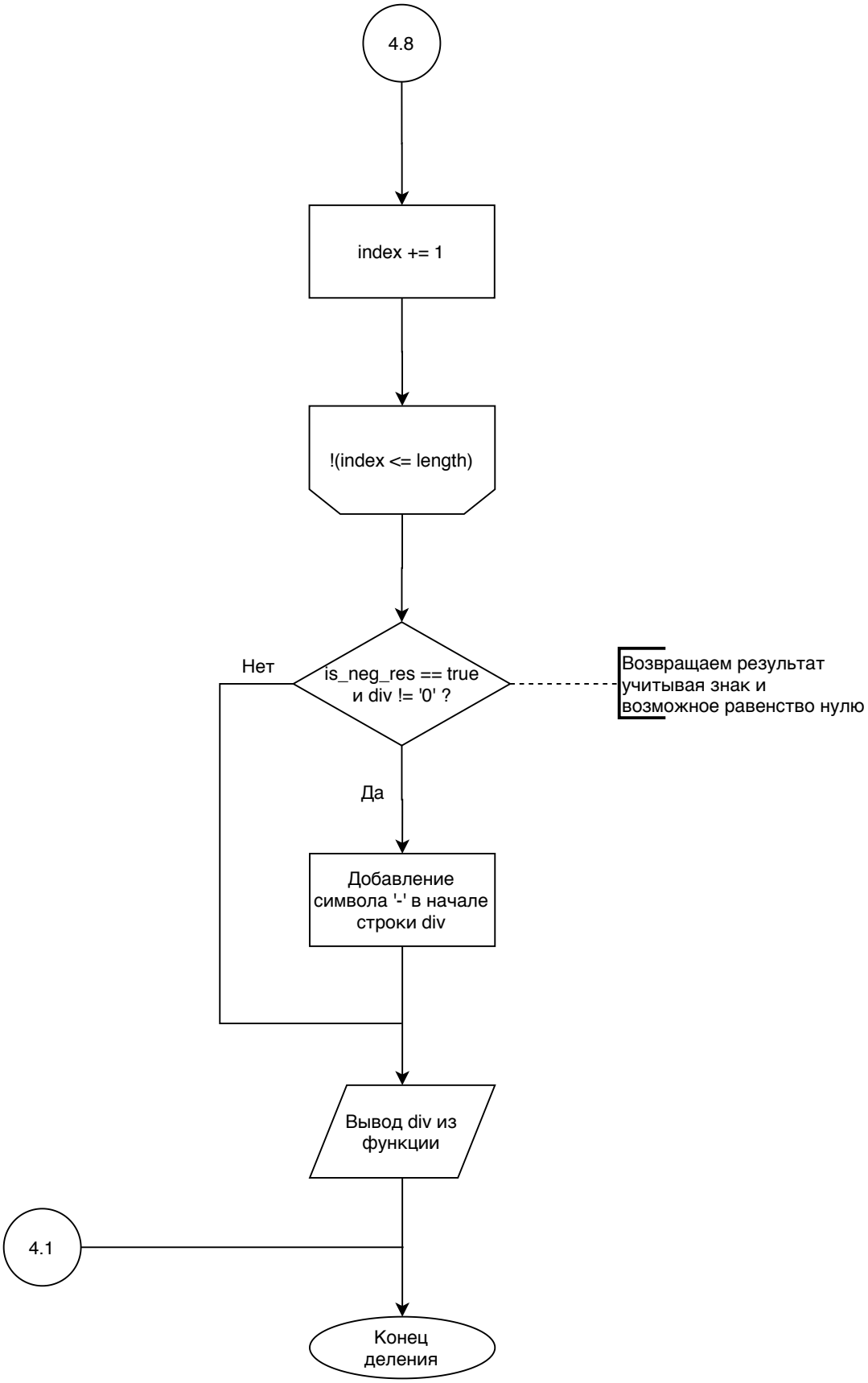

4.2
zeroes++
value2[len(value2 - 1 - zeroes] != '0'
zeros >= len(value2) ?
Да
Вывод из функции: 0
4.1
Нет
zeroes != 0 ?
Да
Нет
value1 = value1[:len(value1) - zeroes]
value2 = value2[:len(value2) - zeroes]
Избавляемся от общих нулей в конце чисел
4.3

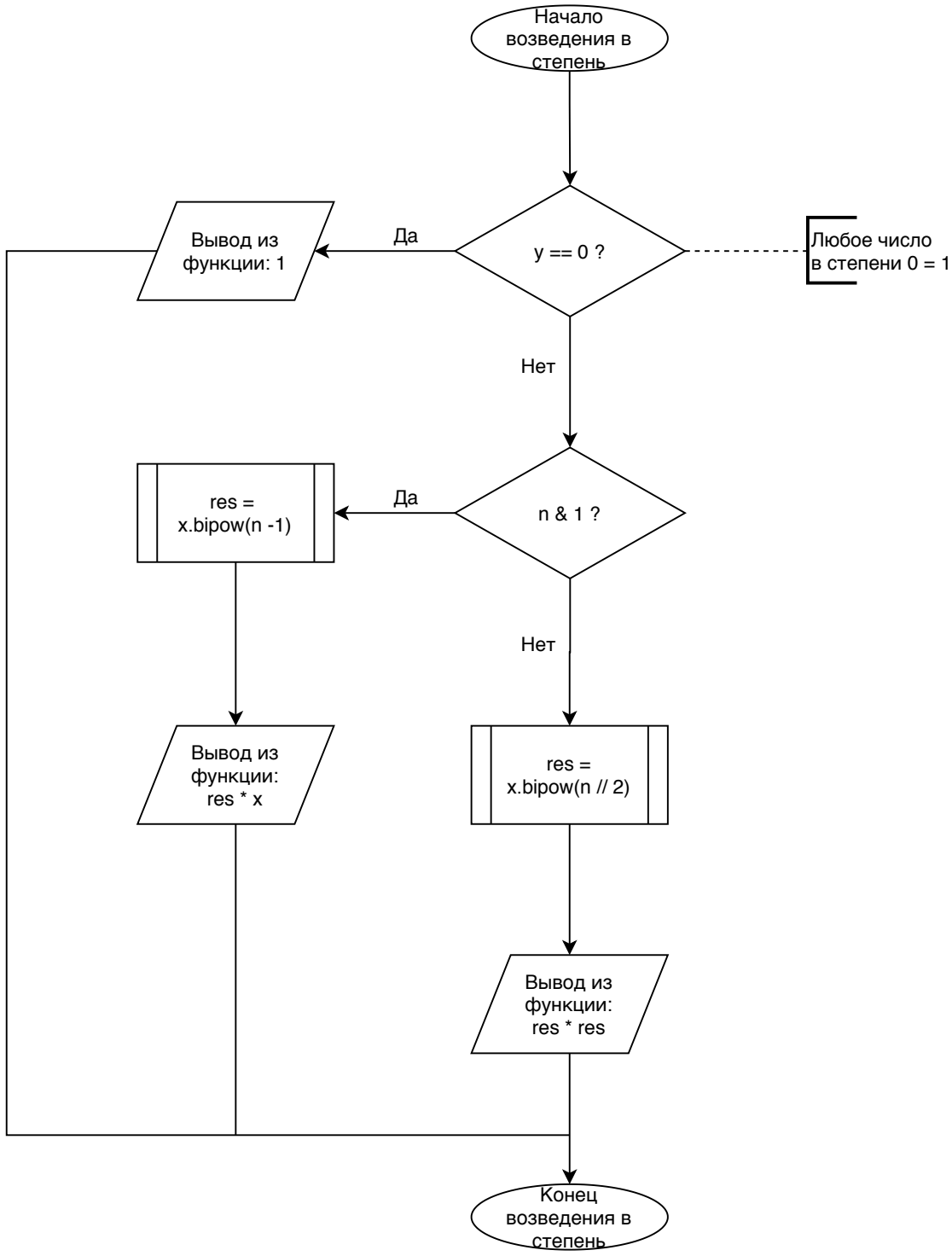

4.3
is_neg_res =
x.is_neg ^ y.is_neg
Считаем
знак числа
tmp = BigInt(value2)
Делаем копию value2
в отдельную
переменную
divider_length =
len(value2)
length = len(value1)
Запоминаем
длину делителя
и делителя
divider_length > 8 ?
Нет
divider_v = int(value2)
Если длина
больше 8,
то обнуляем
делитель,иначе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

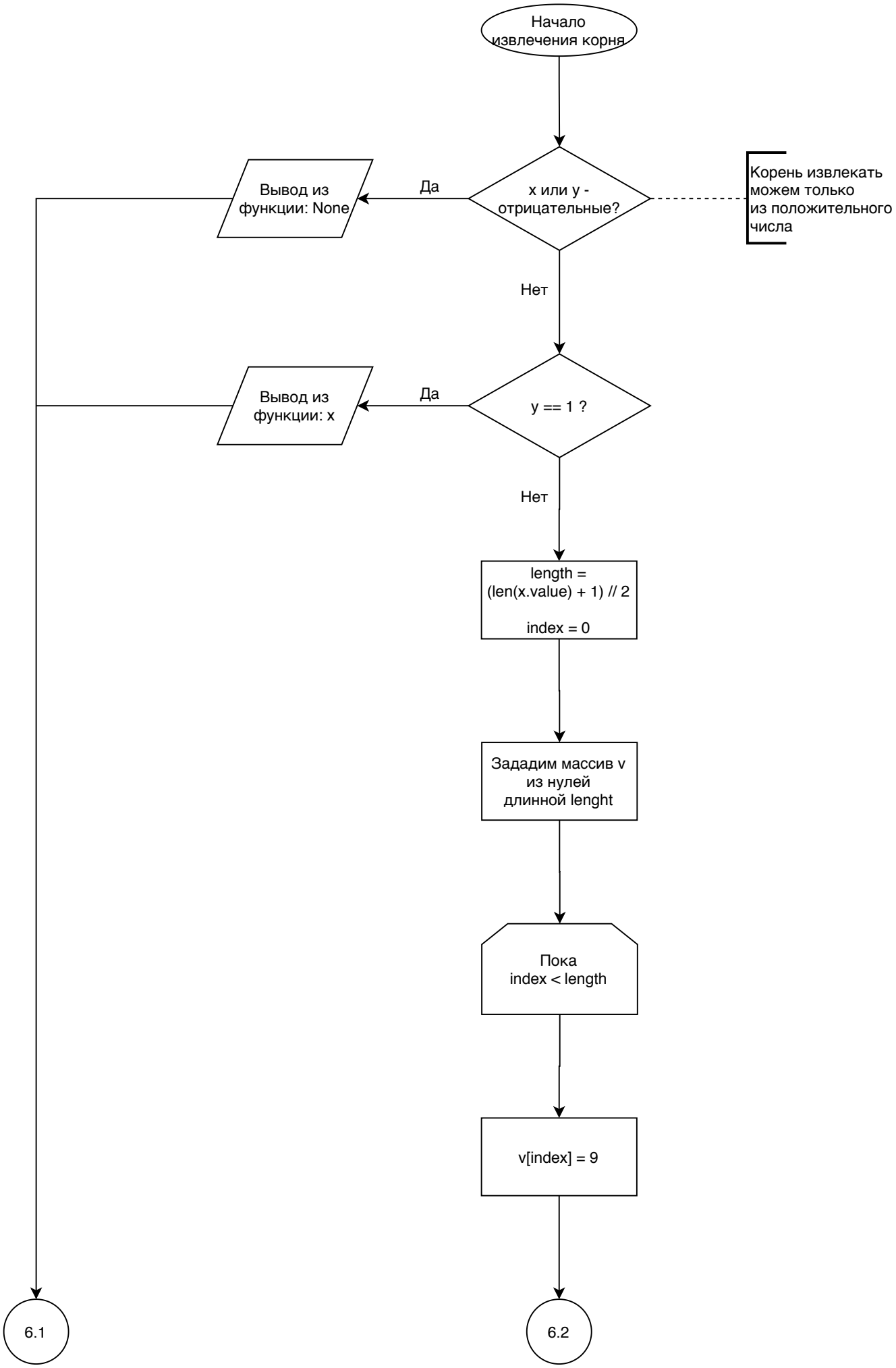

4.4
Пока
BigInt(v) < tmp and
index < length
Формируем начальное
число для деления
v = v + value1[index]
index += 1
!(BigInt(v) < tmp and
index < length)
mod = None
Временная переменная
do while
count = 0
Результат деления
подчисла на делитель
4.5

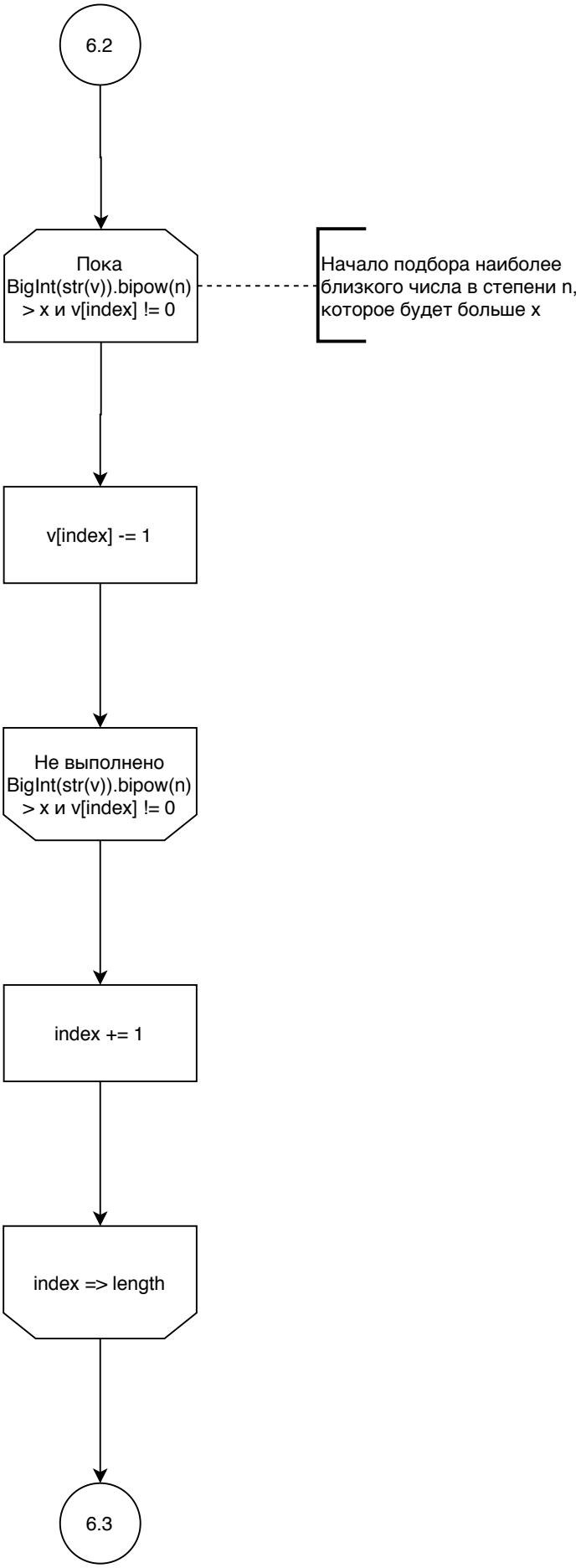

4.5
BigInt(v) >= tmp ?
Нет
Если можем разделить, то делим
Да
mod = int(v)
count = mod // divider_v
v = str(mod % divider_v)
Нет
divider_length > 8 ?
Если не влезает в long, то делим с помощью вычитания
Да
mod = BigInt(v)
Пока mod >= tmp
mod = mod - tmp
count += 1
mod < tmp
4.7
4.6

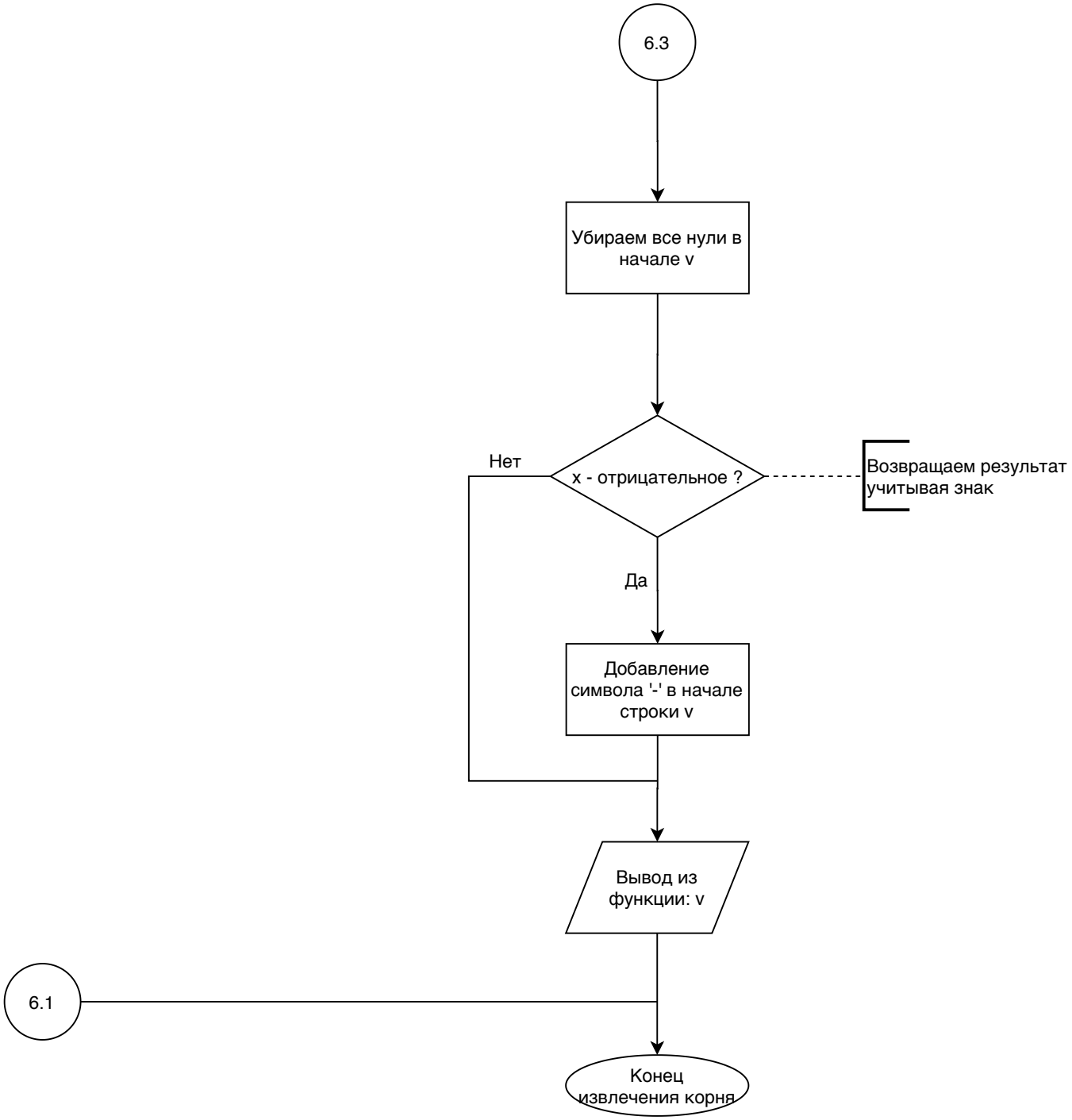

4.7
4.6
v = mod.value
count != 0 ?
Нет
div = div + '0'
Если не делили, то добавили ноль к результату, иначе добавили результат деления
Да
div = div + str(count)
index <= length ?
Да
Вышли за границы массива value1?
Нет
v = v + value1[index]
Да
v = v + '0'
4.8

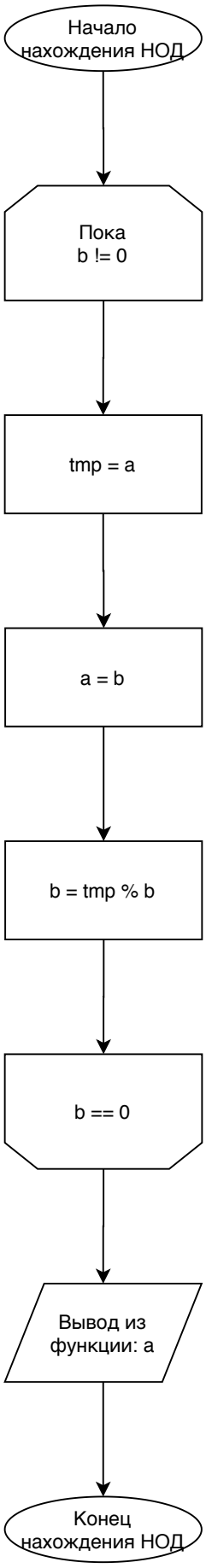

4.8
index += 1
!(index <= length)
Нет
is_neg_res == true
и div != '0' ?
Возвращаем результат
учитывая знак и
возможное равенство нулю
Да
Добавление
символа '-' в начале
строки div
Вывод div из
функции
4.1
Конец
деления

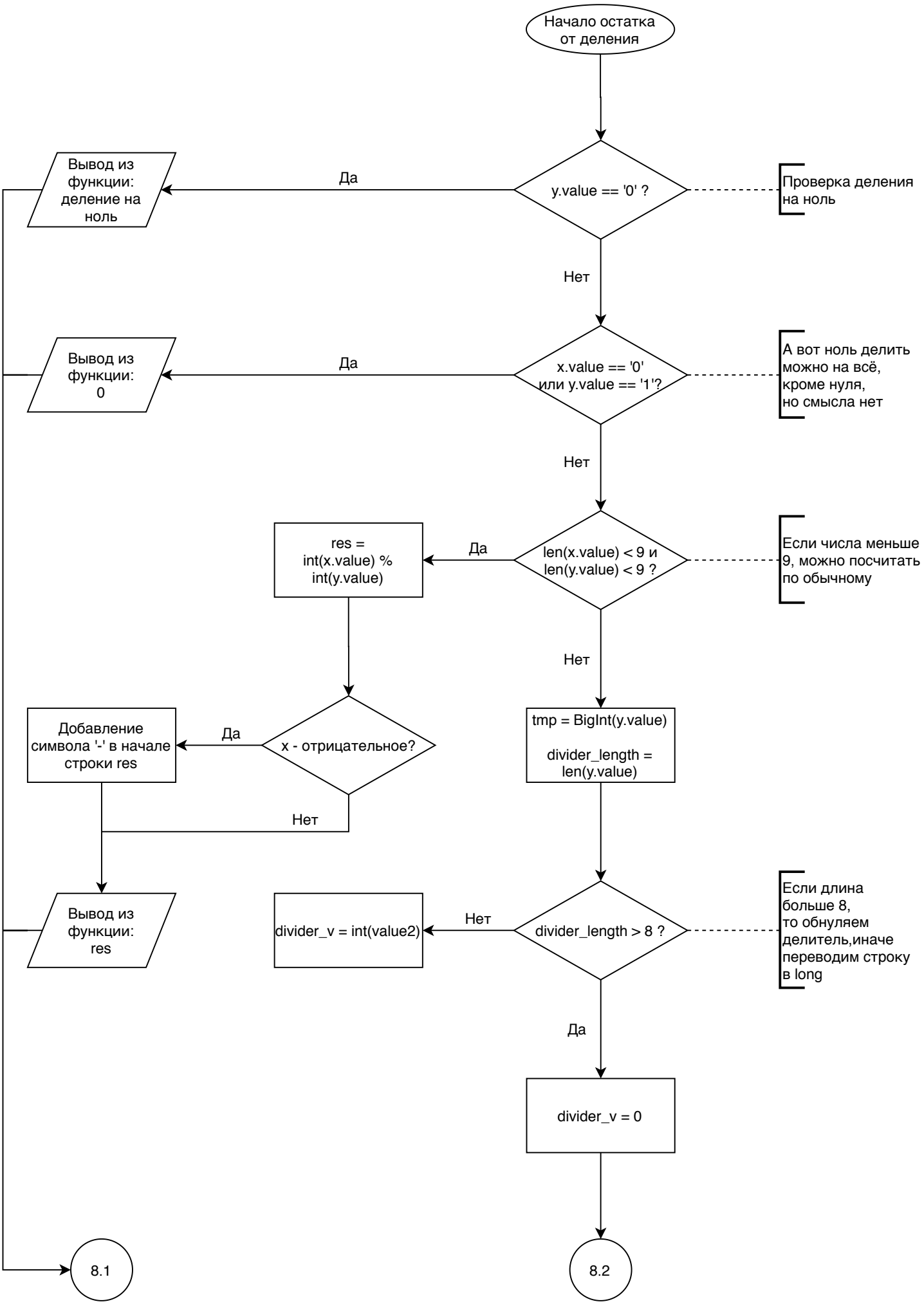

Начало возведения в степень
y == 0 ?
Да
Вывод из функции: 1
Любое число в степени 0 = 1
Нет
n & 1 ?
Да
res = x.bipow(n -1)
Нет
Вывод из функции: res * x
res = x.bipow(n // 2)
Вывод из функции: res * res
Конец возведения в степень

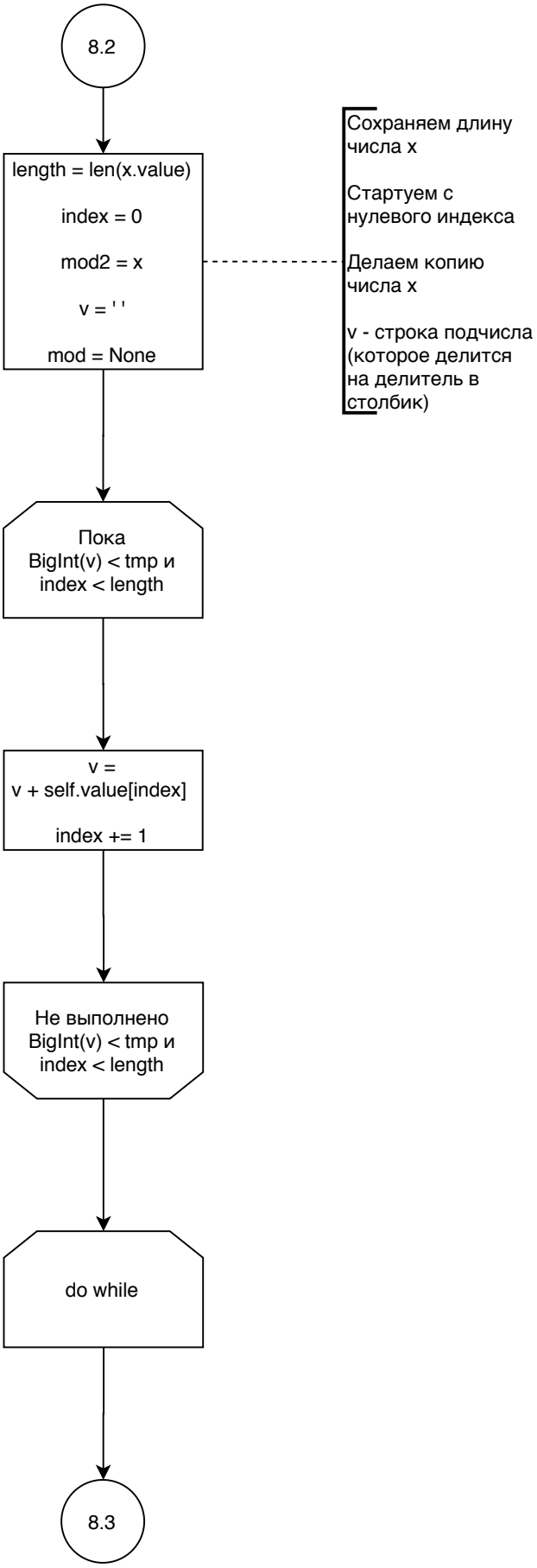

Начало извлечения корня
x или y - отрицательные?
Да
Вывод из функции: None
Корень извлекать можем только из положительного числа
Нет
y == 1 ?
Да
Вывод из функции: x
Нет
length = (len(x.value) + 1) // 2
index = 0
Зададим массив v из нулей длинной lenght
Пока index < length
v[index] = 9
6.1
6.2

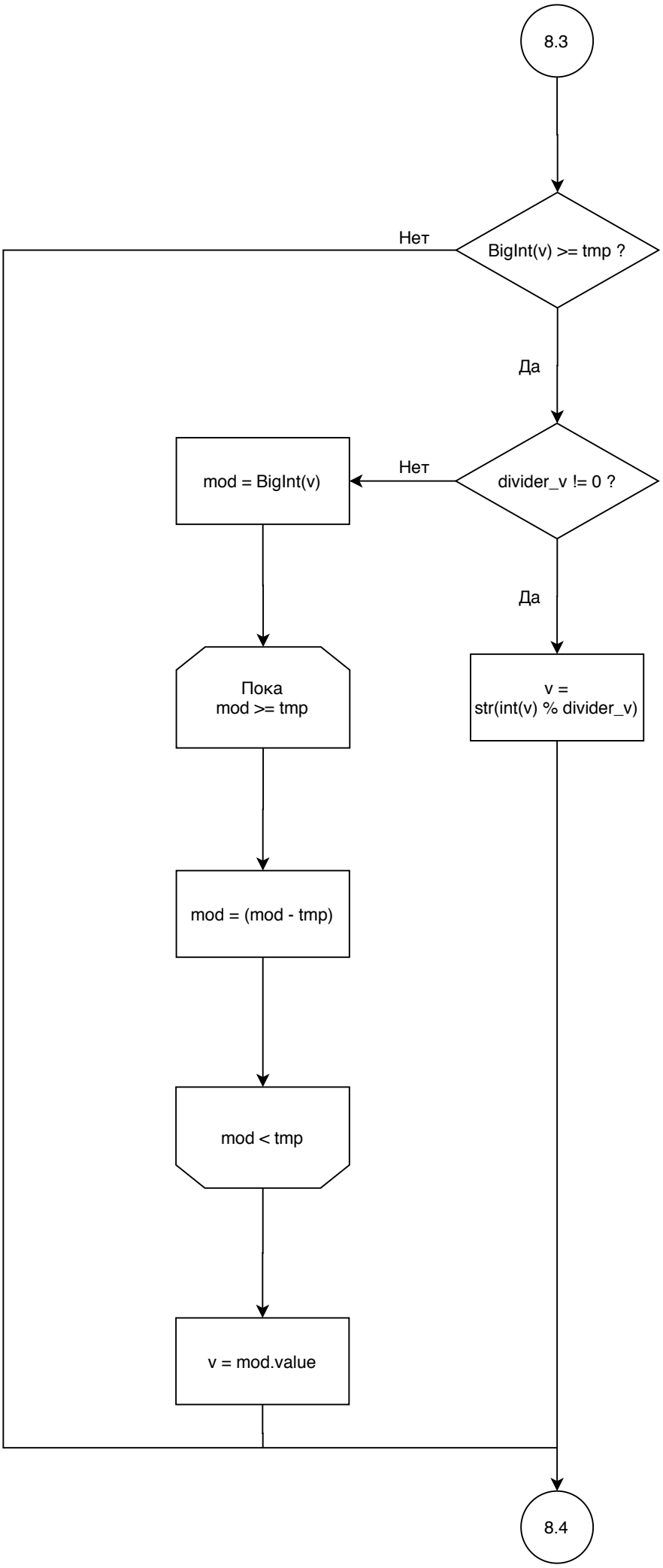

6.2
Пока BigInt(str(v)).bipow(n) > x и v[index] != 0
Начало подбора наиболее близкого числа в степени n, которое будет больше x
v[index] -= 1
Не выполнено BigInt(str(v)).bipow(n) > x и v[index] != 0
index += 1
index => length
6.3

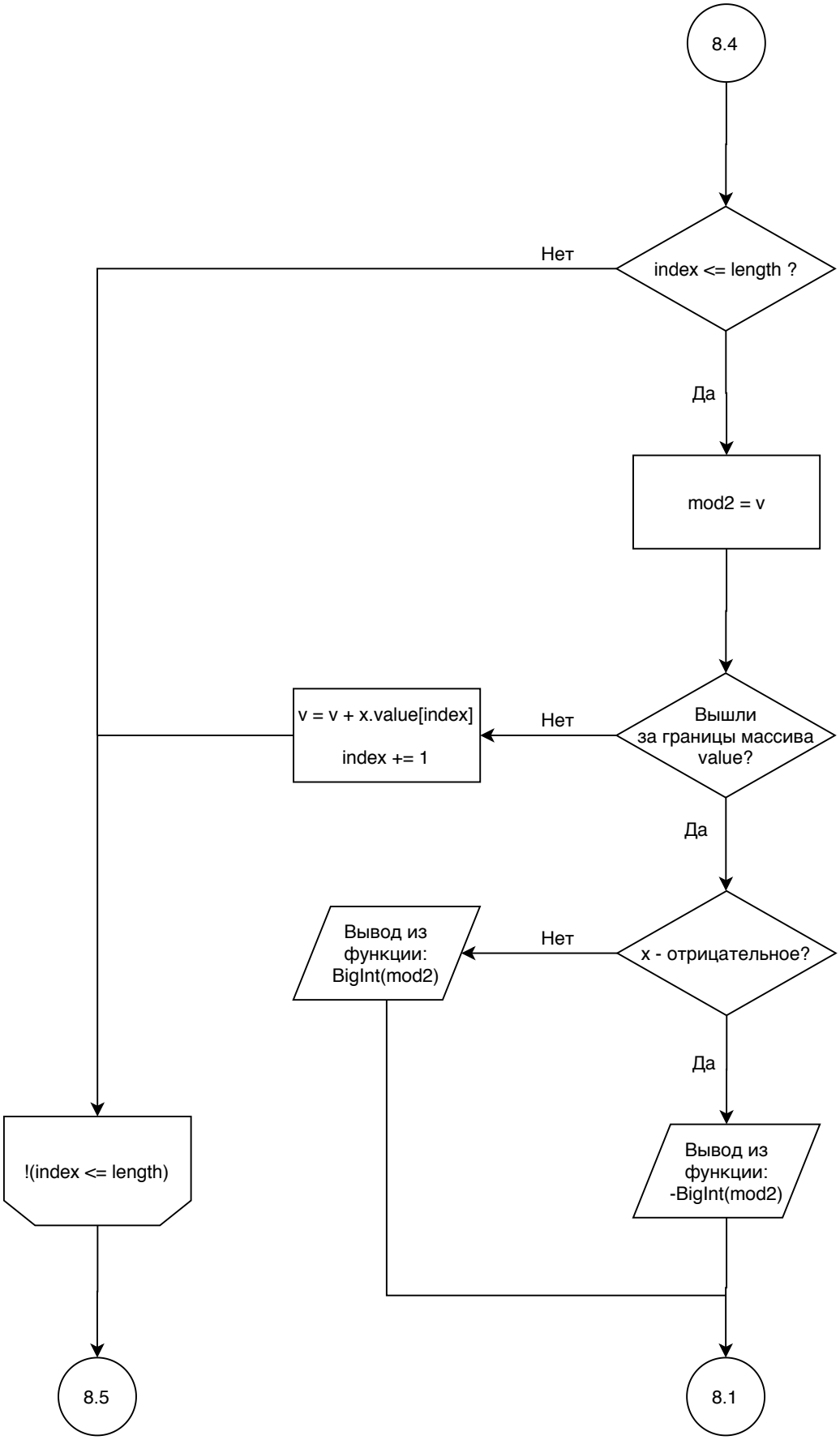

6.3
Убираем все нули в начале v
Нет
x - отрицательное ?
Возвращаем результат учитывая знак
Да
Добавление символа '-' в начале строки v
Вывод из функции: v
6.1
Конец извлечения корня

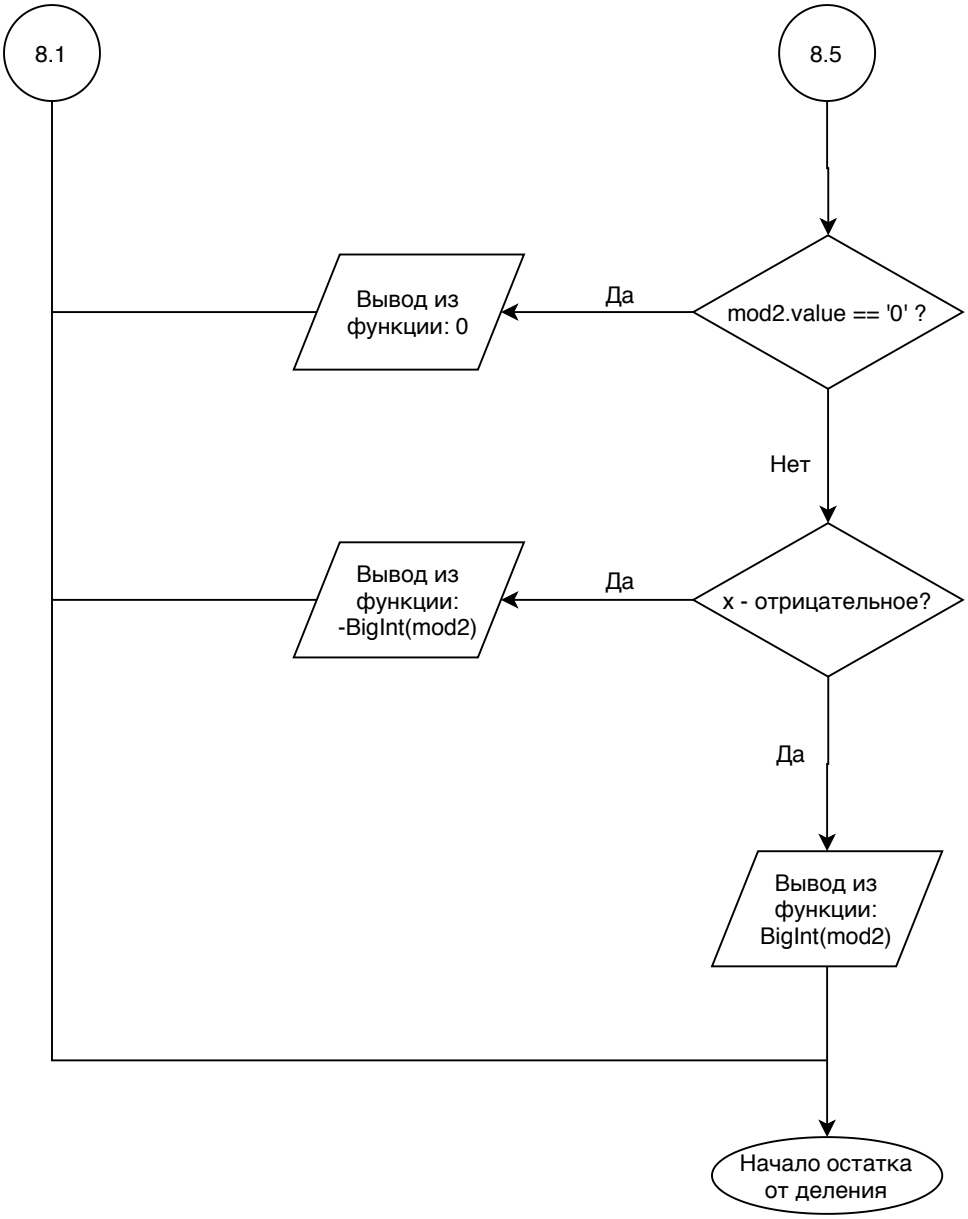

Начало
нахождения НОД
Пока
b != 0
tmp = a
a = b
b = tmp % b
b == 0
Вывод из
функции: a
Конец
нахождения НОД


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