Управляющие структуры в ruby: циклы

Информатика
10-11 классы

20 октября 2011 г.

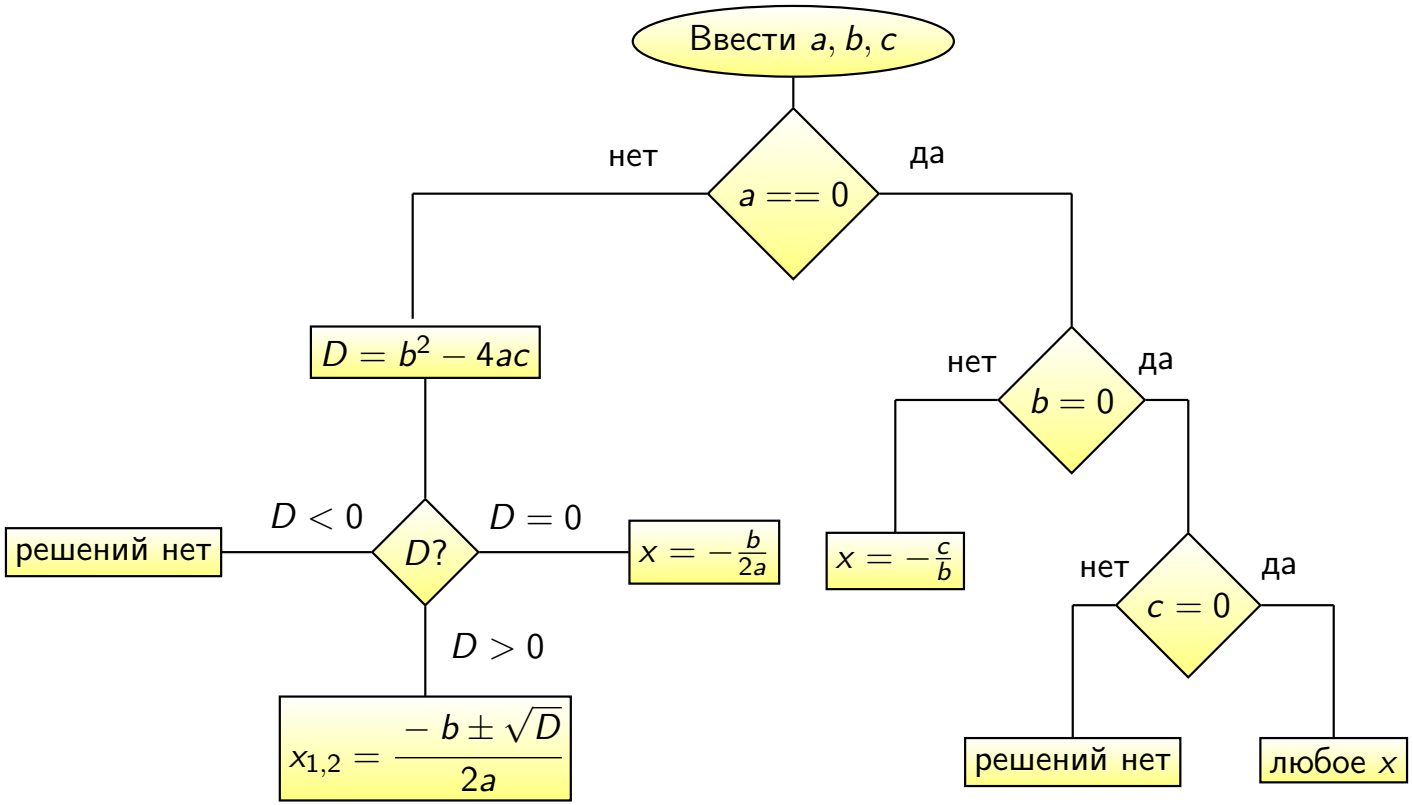
Описание

- Итак, вернёмся к квадратному уравнению. Напишем программу, высчитывающую все корни (если таковые имеются) квадратного уравнения $ax^2 + bx + c = 0$.
- Если $a \neq 0$, то:
  1. Вычислим дискриминант уравнения по формуле: $D = b^2 - 4ac$.
  2. Если дискриминант меньше нуля, то решений нет.
  3. Если дискриминант равен нулю, то корень — один. Он равен: $-\frac{b}{2a}$.
  4. Если дискриминант больше нуля, то существует два вещественных корня:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- В случае $a = 0$ уравнение из квадратного превращается в линейное, которое мы уже умеем решать.

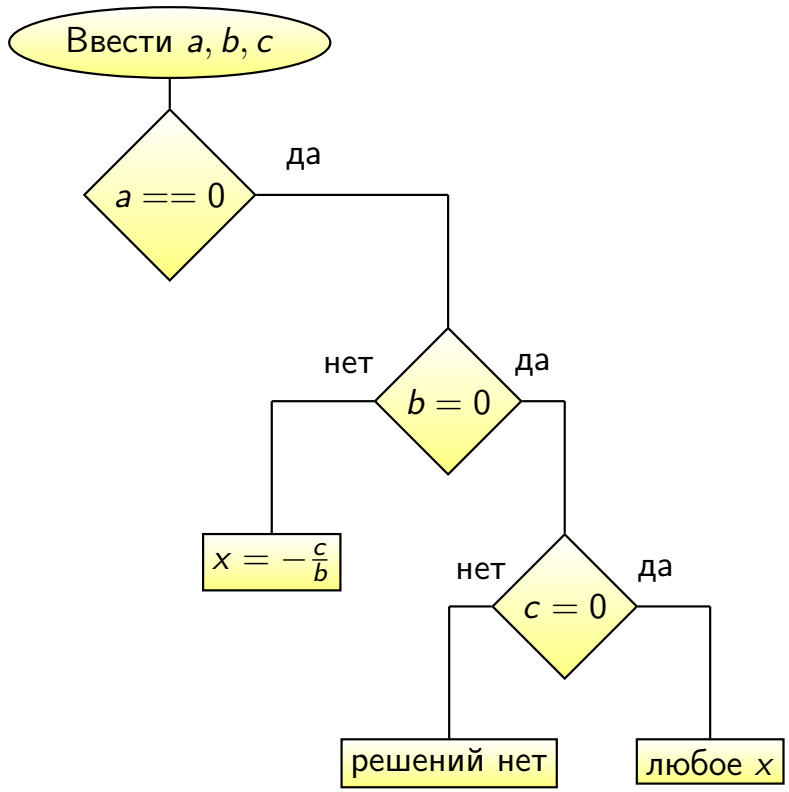
Блок–схема


Ввести a, b, c
$a == 0$
нет
да
$D = b^2 - 4ac$
$D?$
$D < 0$
$D = 0$
$D > 0$
решений нет
$x = -\frac{b}{2a}$
$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$
$b = 0$
нет
да
$x = -\frac{c}{b}$
$c = 0$
нет
да
решений нет
любое x

Программа

Listing 1: Квадратное уравнение

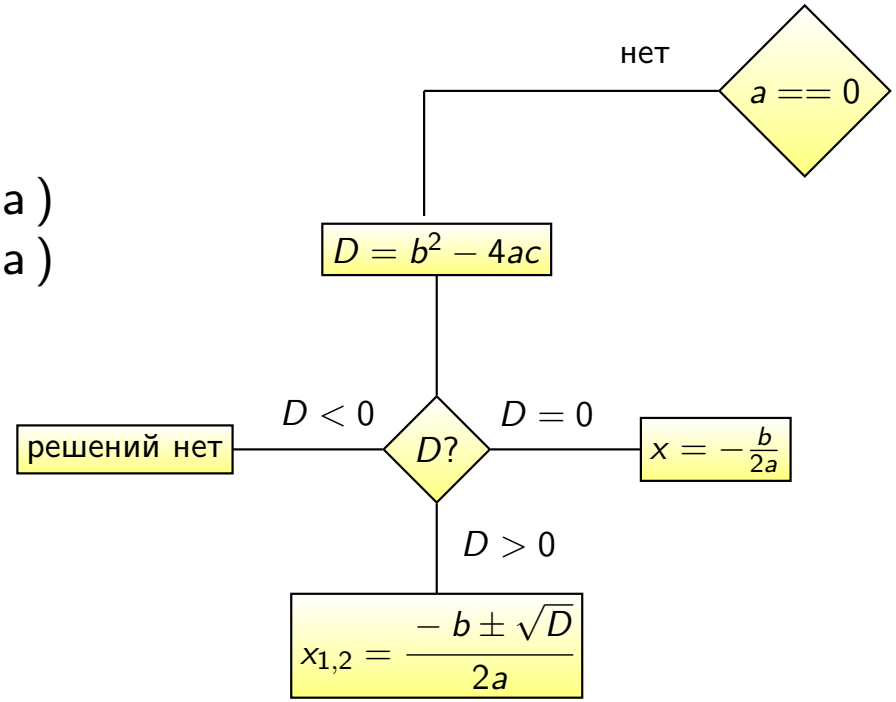
```
a = 2.0
b = 4.0
c = 2.0
if (a == 0)
  if (b == 0)
    if (c == 0)
      puts "any x"
    else
      puts "no solutions"
    end
  else
    x = -c/b
    puts x
  end
else
```



Программа

Listing 2: Квадратное уравнение

```
else
  D = b*b-4*a*c
  if (D > 0)
    x1 = (-b+D**0.5)/(2.0*a)
    x2 = (-b-D**0.5)/(2.0*a)
    puts x1,x2
  elsif (D == 0)
    x = -b/(2*a)
    puts x
  else
    puts "no solutions"
  end
end
```



Основы

- Часто встречаются ситуации, когда какое-либо действие надо повторить несколько раз.
- Например, вывести на экран первые 5 натуральных чисел.
- Простейшая программа выглядела бы следующим образом:

Listing 3: 5 последовательных чисел

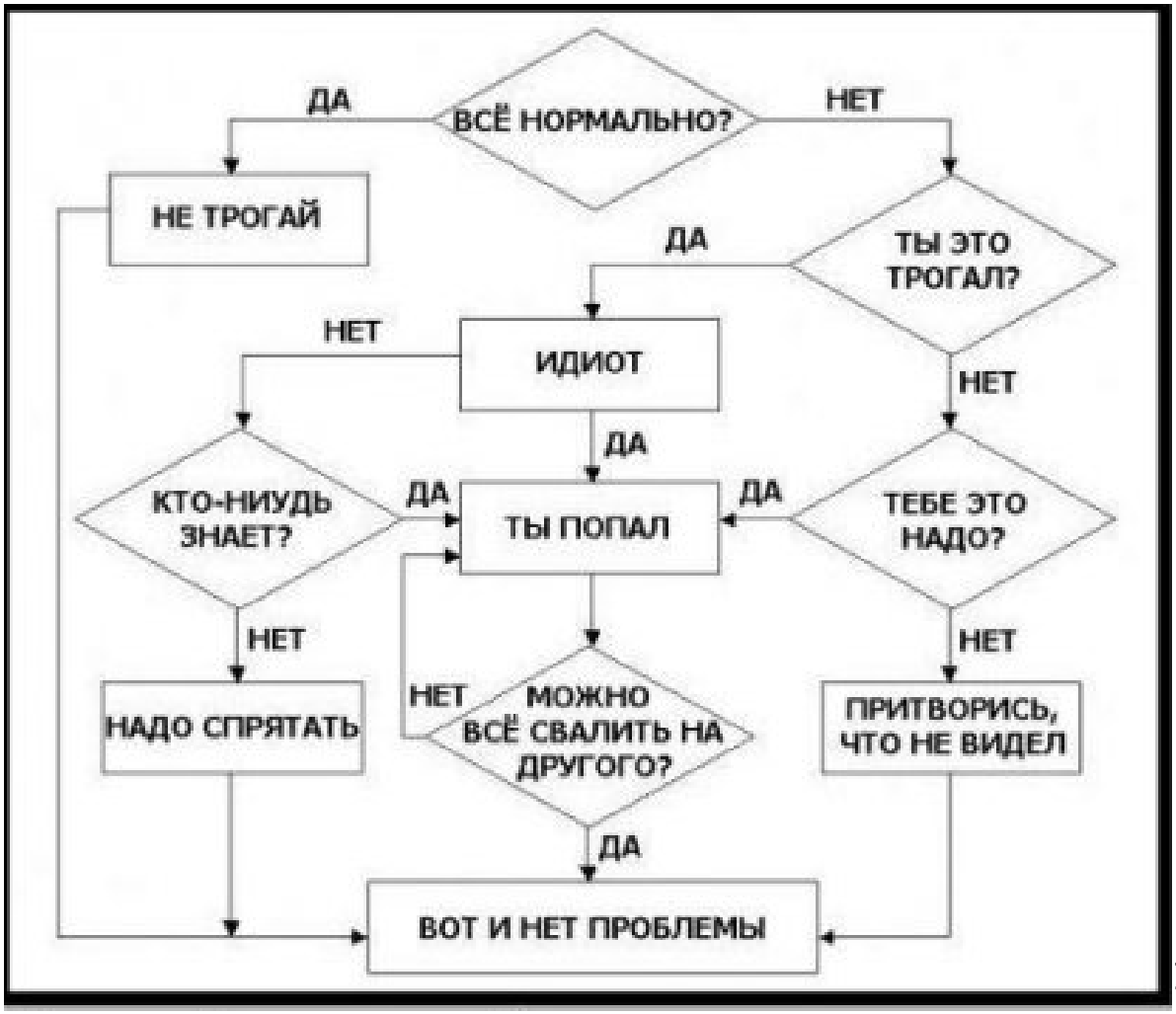
```
puts "1,2,3,4,5"
```

- А если чисел — 100? 1000? 10000?

Основы

- На помощь приходят *циклы*.
- *Цикл* — управляющая структура в алгоритме/программе, позволяющая повторять какую-либо операцию несколько раз.
- Два основных типа циклов:
  1. с заданным количеством шагов–итераций (*явный цикл*)
  2. без явно заданного количества итераций (*неявный цикл*)
- Первый тип состоит из начала цикла, номера итерации, тела цикла (что делаем), конца цикла.
- Второй тип — из начала цикла, условия выхода из цикла, номера итерации (опционально), тела цикла, конца цикла.

Пример цикла



[1]

[1] Где здесь цикл и какому типу он принадлежит?

For и while

- Двум типам циклов соответствует два оператора: *for* и *while*.
- Напишем программу, выводящую на экран числа от 1 до 100.

Listing 4: for

```
for i in 1..100
  puts i
end
```

Listing 5: while

```
i = 0
while (i < 100)
  i = i+1
  puts i
end
```

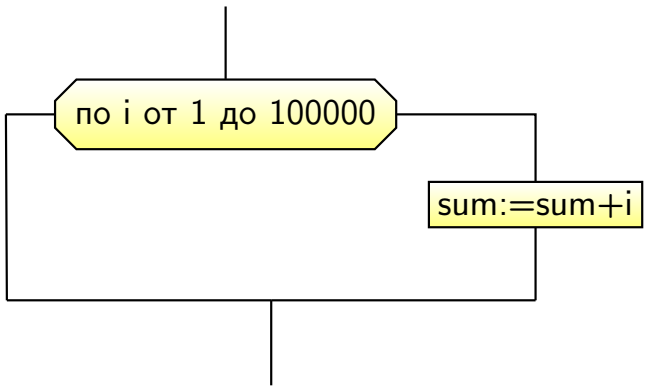
For и while

- Рассмотрим программы более детально.
- Оба оператора образуют следующую структуру:
  1. Ключевое слово (**for** или **while**).
  2. Условие повторения (явное или неявное).
  3. Конец (оператор **end**).
- В определении цикла for используется специальная переменная *i*, которая указывает на номер итерации.
- Она “пробегает” значения в заданном интервале: for i in 1..100 последовательно пробегает значения 1, 2, 3, ... 99, 100.
- В цикле while с помощью оператора присваивания (i = i+1) мы *эмулируем* такую переменную.

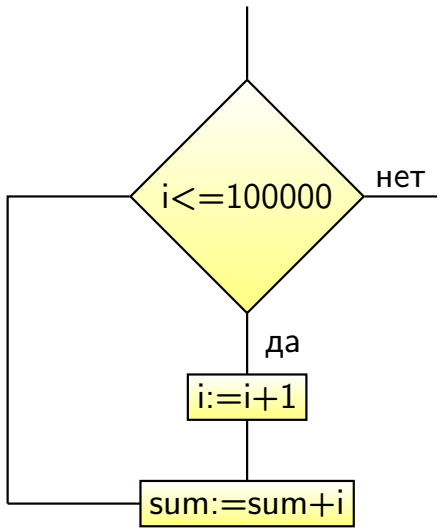
Суммирование чисел: алгоритм

- Посчитаем сумму чисел от 1 до 100000 с помощью циклов.

For



While



Программа

Listing 6: for

```
sum = 0
for i in 1..100000
  sum = sum + i
end
puts sum
```

Listing 7: while

```
i   = 0
sum = 0
while (i < 100000)
  i = i+1
  sum = sum + i
end
puts sum
```

Логарифмирование

- Задача: найти наименьшее целое число, для которого выполнено неравенство:

$$2^x > 1000000$$

- Цикл *for* здесь не поможет, ведь мы заранее не знаем, сколько итераций будет.
- Решаем через *while*. Как?
- Пробежим все степени двойки, начиная с нулевой, фиксируя на каждом шаге значение показателя степени.
- Если текущее значение меньше 1000000, увеличим показатель на 1.
- Когда-нибудь наступит ситуация, когда 2 в какой-либо степени станет больше, чем 1000000.
- Последнее значение показателя степени и будет искомым числом.

Логарифмирование

Listing 8: Вычисление наименьшей степени

```
num = 0
i = 0
while (num <= 1000000)
   i = i+1
   num = 2**i
end
puts i
```

- Задание: дан алфавит, состоящий из N букв. Написать программу, которая считает, сколько бит занимает один символ этого алфавита.

Выход из цикла

- Может возникнуть ситуация, когда нам нужно прекратить выполнение до цикла даже несмотря на то, что не все итерации пройдены.
- Для окончания цикла нужно вызвать оператор **break**.
- Пример: если бы в задаче $2^x > 1000000$ мы использовали цикл for, то нам следовало бы остановиться в ситуации, когда 2^i стало бы больше 1000000, где i — номер итерации.

Listing 9: Break

```
for i in 1..1000000
  ...
  break if (2**i > 1000000)
end
```

- Для перехода к следующей итерации без выполнения дальнейшего кода из тела цикла — оператор **next**.

Оператор times

- Оператор **times** (*англ.* кол-во раз) очень похож на **for**. Он повторяет определённое заданное количество раз определённое действие.
- Его зачастую используют, когда цикл — очень простой и содержит всего одно действие.
- Приведём пример: выведем на экран квадраты чисел от 0 до 9.

Listing 10: Вычисление наименьшей степени

```
10.times { |i| puts i**2 }
```

- Лаконично, не правда ли?
- NB! Нумерация начинается с 0, а не с 1!

Числа Фибоначчи

- *Числа Фибоначчи* задаются *рекуррентной* формулой:

$$\varphi_n = \varphi_{n-1} + \varphi_{n-2}$$

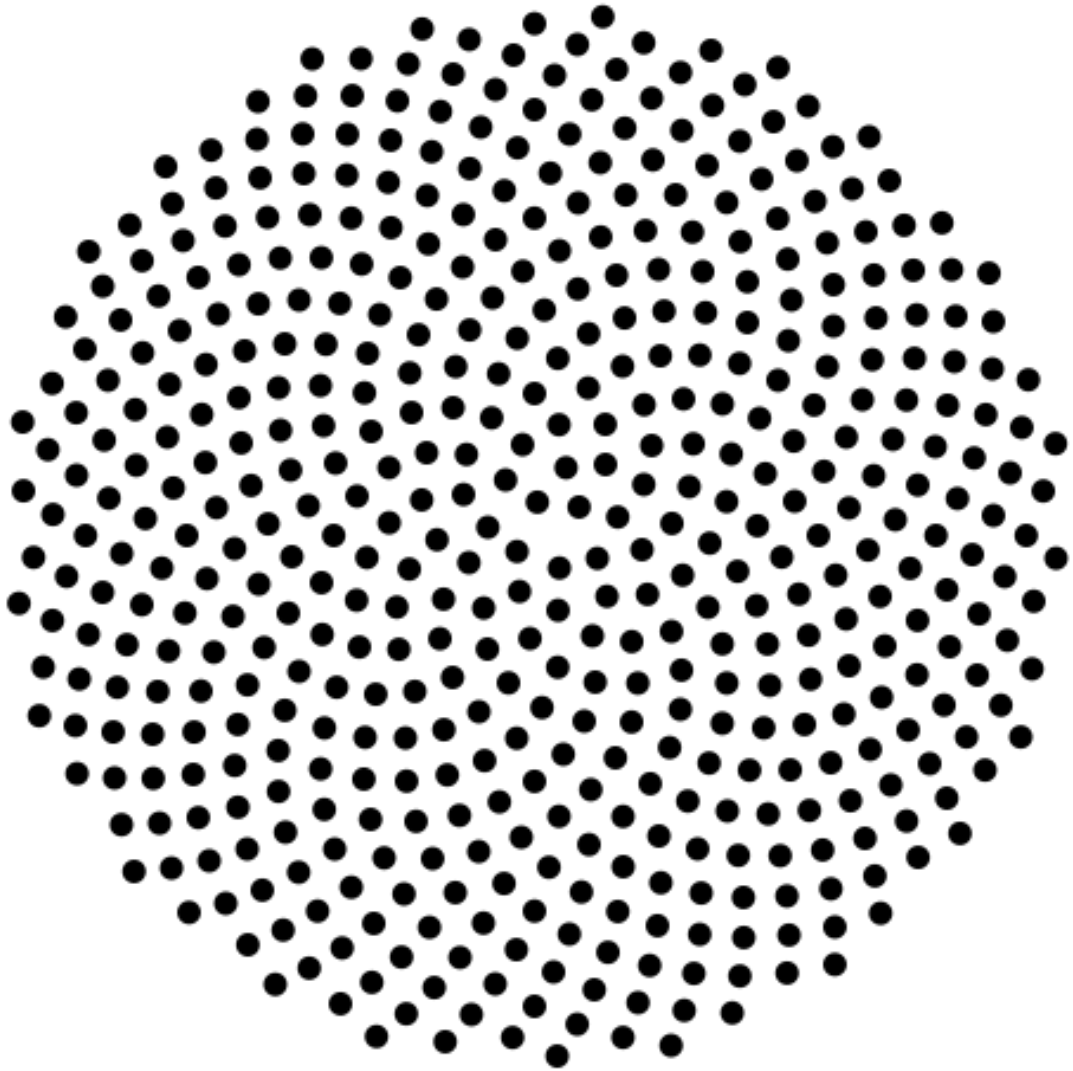
где $\varphi_0 = 1, \varphi_1 = 1$.

- Первые несколько чисел Фибоначчи:

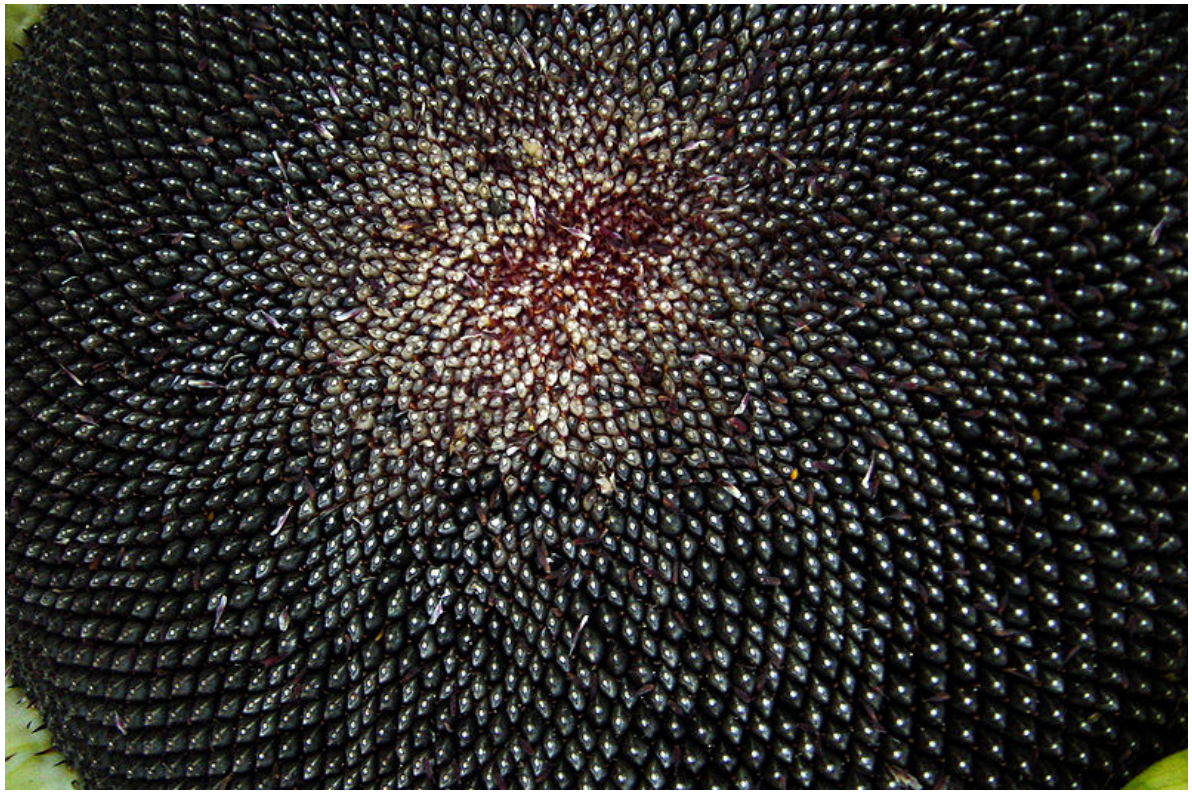
φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	φ_9	φ_{10}
1	1	2	3	5	8	13	21	34	55	89

- Числа Фибоначчи встречаются и в природе: филлотаксис (листорасположение) у растений описывается последовательностью Фибоначчи.
- Зерна подсолнуха, сосновые шишки, лепестки цветков располагаются также по числам Фибоначчи.

Числа Фибоначчи



Числа Фибоначчи



Вычисление 100 числа Фибоначчи

- Итак, чтобы вычислить число Фибоначчи, нужно знать два предыдущих.
- Пойдём последовательно. Нам известны значения двух первых чисел: $\varphi_0 = 1, \varphi_1 = 1$.
- Чтобы вычислить второе число, надо сложить первое и нулевое.
- Прибавив к полученному числу первое — получим третье:

$$\varphi_3 = \varphi_2 + \varphi_1 = (\varphi_1 + \varphi_0) + \varphi_0.$$

- Для вычисления четвёртого числа воспользуемся значениями третьего и второго.
- Итого: нам надо хранить два последних числа. Складывая их, мы получаем новое число. “Сдвигаемся” на единицу дальше.

Программа

Listing 11: 100 число Фибоначчи

```
a0 = 1
a1 = 1
a_new = 0
for i in 2..100
  a_new = a0 + a1
  a0 = a1
  a1 = a_new
end
puts a_new
```

- Задание: подготовить блок–схему программы.
- Задание: сделать аналогично с циклом *while*.

References

- Все презентации доступны на http://school.smirik.ru!
- Вопросы, предложения, д/з: smirik@gmail.com