Оглавление

[1. Язык программирования Squirrel 1](#_Toc530563232)

[1.1 Общая характеристика и история создания 1](#_Toc530563233)

[1.2 Система типов и особенности архитектуры языка + 1.3 Примеры кода и интересные факты 2](#_Toc530563234)

[2. Язык программирования Fantom 4](#_Toc530563235)

[2.1 Общая характеристика и история создания 4](#_Toc530563236)

[2.2 Система типов и особенности архитектуры языка 4](#_Toc530563237)

[2.3 Примеры кода и интересные факты 4](#_Toc530563238)

[3. Язык программирования Elixir 4](#_Toc530563239)

[3.1 Общая характеристика и история создания 4](#_Toc530563240)

[3.2 Система типов и особенности архитектуры языка 5](#_Toc530563241)

[3.3 Примеры кода и интересные факты 5](#_Toc530563242)

[Список использованной литературы 6](#_Toc530563243)

# 1. Язык программирования Squirrel

## 1.1 Общая характеристика и история создания

Squirrel - [интерпретируемый](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), рис.1

разработанный специально для использования в качестве скриптового языка в приложениях реального времени, таких как компьютерные игры. Является [свободно распространяемым](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), с открытыми исходными текстами. Язык разработан и поддерживается Альберто Демикелисом. Впервые опубликован в 2003 году под лицензией zlib/libpng. Позднее, в ноябре 2010-го, лицензия была изменена на MIT для публикации на Google Code. Главной проблемой Lua для разработчиков того времени была непредсказуемость поведения сборщика мусора, существенно влиявшая на производительность программного продукта. Современный стандарт Lua подразумевает инкрементальную сборку мусора, предоставляемую функцией lua\_gc() с параметром LUA\_GCSTEP, однако оптимальное значение этого параметра сильно зависит от конкретной задачи и может меняться в ходе разработки. В 2003, для решения проблемы отсутствия инкрементальной сборки мусора в Lua, Демикелис приступил к реализации сборщика на основе принципа подсчёта ссылок. В последствии, вокруг нового сборщика вырос и новый язык, решавший также некоторые типичные для разработчиков проблемы синтаксиса Lua. Несмотря на то, что язык вдохновлён Lua, его синтаксис гораздо ближе к [C++](https://ru.bmstu.wiki/index.php?title=C%2B%2B&action=edit&redlink=1)\[C#](https://ru.bmstu.wiki/index.php?title=C&action=edit&redlink=1). На данный момент, разработка ядра языка завершена, лишь сам разработчик продолжает вносить изменения в стандарт на уровне калибровки производительности и документации.

## 1.2 Система типов и особенности архитектуры языка + 1.3 Примеры кода и интересные факты

Squirrel - динамически-типизированный язык, переменные не имеют явного типа, хотя содержащиеся в них данные строго типизированы. Базовый набор типов языка: integer, float, string, null, table, array, function, generator, class, instance, bool, thread и userdata.

Примитивы

* local a = 123 // десятичное
* local b = 0x0012 // шестнадцатеричное
* local c = 075 // восьмеричное
* local d = 'w' // символ
* local a = 1.0 // число с плавающей точкой
* local b = 0.234 //
* local a = true // логическая переменная

Массив

Массивы - простые, пронумерованные с нуля последовательности объектов.

local a = ["I'm","an","array"]

local b = [null]

b[0] = a[2];

Функции

Функции в Squirrel похожи на таковые во всех остальных Си-подобных языках, с некоторыми синтаксическими отличиями. Так как язык встраиваем, реализована поддержка вызова функций с интерфейсом ANSI C.

local a = function(a,b,c) {return a+b-c;}

// или

function ciao(a,b,c)

{

return a+b-c;

}

// или

this.ciao <- function(a,b,c)

{

return a+b-c;

}

// или анонимная функция

local myexp = @(a,b,c) a + b - c

Класс

Класс - ассоциативный контейнер, набор пар ключ\значение. Классы создаются с помощью выражения class. Squirrel поддерживает механизмы наследования полей. Новые поля могут быть добавлены в класс, пока не создан хоть один экземпляр.

class Foo {

//конструктор

constructor(a)

{

testy = ["stuff",1,2,3,a];

}

//метод

function PrintTesty()

{

foreach(i,val in testy)

{

::print("idx = "+i+" = "+val+" \n");

}

}

//свойство

testy = null;

}

# 2. Язык программирования Fantom

## 2.1 Общая характеристика и история создания

**Fantom** объектно ориентированный язык программирования общего назначения созданный Брайаном и Энди Фрэнком (Brian Frank, Andy Frank)[[4]](https://ru.wikipedia.org/wiki/Fantom#cite_note-4) который компилируется в [Java Runtime Environment](https://ru.wikipedia.org/wiki/Java_Runtime_Environment" \o "Java Runtime Environment) (JRE), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript), и .NET [Common Language Runtime](https://ru.wikipedia.org/wiki/Common_Language_Runtime" \o "Common Language Runtime) (CLR). Главной целью является предоставление стандартного [API](https://ru.wikipedia.org/wiki/API) который абстрагирует от вопроса в каком окружении будет выполняться код: в JRE или CLR. Язык поддерживает функциональное программирование через замыкания и многопоточность через акторы.

## 2.2 Система типов и особенности архитектуры языка

Fantom Default Java .NET

------ ------- ---- ------

Bool false boolean bool

Bool? null java.lang.Boolean bool?

Int 0 long long

Int? null java.lang.Long long?

Float 0.0f double double

Float? null java.lang.Double double?

## 2.3 Примеры кода и интересные факты

Int func(Int x) { ... }

Int i := 5

Num n := 5

Str s := "foo"

// statically correct as is: Int.fits(Int)

func(i) => func(i)

// implicit cast inserted: Int.fits(Num)

func(n) => func((Int)n)

// compile time error: !Int.fits(Str)

func(s) => error

# 3. Язык программирования Elixir

## 3.1 Общая характеристика и история создания

Elixir — функциональный, распределённый язык программирования общего назначения, который работает на виртуальной машине Erlang (BEAM). Построен поверх Erlang, что обеспечивает распределённость, отказоустойчивость, исполнение в режиме мягкого реального времени, метапрограммирование с макросами и полиморфизмом, реализованным через протоколы. Использует Erlang/OTP для работы с деревьями процессов.

Создан Жозе Валимом (José Valim), ранее являвшимся одним из основных разработчиков фреймворка Ruby on Rails и сооснователем компании Plataformatec. Его целью было включить более высокую расширяемость и производительность в Erlang VM, сохраняя совместимость с инструментами и экосистемой Erlang.

## 3.2 Система типов и особенности архитектуры языка

На низком уровне Elixir использует примитивные типы, доступные в виртуальной машине Erlang. Так, [список](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) — всего лишь упорядоченный набор значений. Кроме того, соответствующие модули и [синтаксический сахар](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%81%D0%B0%D1%85%D0%B0%D1%80) позволяют использовать более высокоуровневые абстракции. Например, тип Keyword является модулем Elixir, а его реализация основана на списке [кортежей](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D1%82%D0%B5%D0%B6_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), и, естественно, со значениями типа Keyword можно работать и как со списками[[10]](https://ru.wikipedia.org/wiki/Elixir_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-_770e8b9501798b63-10).

Elixir имеет следующие встроенные типы данных (см. также [типы данных Erlang](https://ru.wikipedia.org/wiki/Erlang#%D0%A2%D0%B8%D0%BF%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85))[[11]](https://ru.wikipedia.org/wiki/Elixir_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-_56046769a14f4499-11):

* Целые произвольной разрядности
* [Числа с плавающей запятой](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D1%81_%D0%BF%D0%BB%D0%B0%D0%B2%D0%B0%D1%8E%D1%89%D0%B5%D0%B9_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%BE%D0%B9)
* [Атомы](https://ru.wikipedia.org/w/index.php?title=%D0%90%D1%82%D0%BE%D0%BC_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)&action=edit&redlink=1)
* Интервальные типы ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *ranges*)
* [Регулярные выражения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)
* Идентификаторы процессов (PID) и портов
* Ссылки
* Кортежи
* Списки
* Отображения ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *maps*)
* Бинарные данные ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *binaries*)
* Функции

## 3.3 Примеры кода и интересные факты

**defmodule** **Fun** **do**

**def** fib(0), do: 0

**def** fib(1), do: 1

**def** fib(n) **do**

fib(n-2) + fib(n-1)

**end**

**end**

**quote** do: (k = 1; k + 2) *# в результате даёт*

{:\_\_block\_\_, [],

[{:=, [], [{:k, [], **Elixir**}, 1]},

{:+, [context: **Elixir**, import: **Kernel**], [{:k, [], **Elixir**}, 2]}]}

# Список использованной литературы

<https://ru.wikipedia.org/wiki/Squirrel>

<https://ru.bmstu.wiki/Squirrel>

<https://ru.wikipedia.org/wiki/Fantom>

<https://ru.wikipedia.org/wiki/Elixir_(язык_программирования)>

рис.1

