Uni Programming Languages Notes

Felicitas Pojtinger (fp036)

2022-10-24

Introduction

Contributing

These study materials are heavily based on professor Ihler's "Aktuelle Programmiersprachen" lecture at HdM Stuttgart.

Found an error or have a suggestion? Please open an issue on GitHub (github.com/pojntfx/uni-programminglanguages-notes):

Figure 1: QR code to source repository

License

Figure 2: AGPL-3.0 license badge

SPDX-License-Identifier: AGPL-3.0

Overview

General Design

- "A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write."
- Inspired by Perl, Smalltalk, Eiffel, Ada, Lips
- Multi-paradigm from the beginning: Functional, imperative and object-oriented
- Radical object orientation: Everything is an object, there are no primitive types like in Java
  (5.times { print "We *love* Ruby -- it's outrageous!" })
- Very flexible, i.e. operators can be redefined
- Built-in blocks (closures) from the start, excellent mapreduce capabilities
- Prefers mixins over inheritance
- Syntax uses limited punctuation with some notable exceptions (instance variables with @, globals with $ etc.)

Implementation Details

- Exception handling similar to Java & Python, but no checked exceptions
- Garbage collection without reference counts
- Simple C/C++ extension interface
- OS independent threading & Fibers, even if OS is single-threaded (like MS-DOS)
- Cross-platform: Linux, macOS, Windows, FreeBSD etc.
- Many implementation (MRI/CRuby, JRuby for Ruby in the JVM, TruffleRuby on GraalVM, mruby for embedded uses, Artichoke for WebAssembly and Rust)

Users

- Twitter
- Mastodon
- GitHub
- Airbnb
- Shopify
- Twitch
- Stripe
- Etsy
- Soundcloud
- Basecamp
- Kickstarter

Timeline

- First concepts and prototypes ~1993
- First release ~1995, became most popular language in Japan by 2000
- Subsequent evolution and growth outside Japan
- Ruby 3.0 released ~2020, introducing a type system for static analysis, fibers (similar to Goroutines, asyncio etc.), and completing optimizations making it ~3x faster than Ruby 2.0 (from 2013)

Syntax

Logic

Typical logical operators:

```
>> 2 < 3
=> true
```

```
>> 1 == 2
=> false
```

Comparisons are type checked:

```
>> 1 == "1"
=> false
```

Trip equals can be used to check if if an instance belongs to a class:

```
>> String === "abc"
=> true
```


If, else, etc work as expected:

Loops

Ruby does not have traditional for loops, but multiple, more general constructs that allow for the usecases.

Ruby has the for loop that we are all used to, but also more specialized constructs that allow for more expressive usecases:

```
for i in 0..10
    p i
end
```

For example upto and downto methods:

```
10.downto 1 do |num|
  p num
end
```

```
17.upto 23 do |i|
  print "#{i}, "
```

Arrays

Arrays in Ruby can contain multiple types and work as expected; there is no array vs collection divide:

```
my_array = ["Something", 123, Time.now]
```

Instead of loops you can use the each method to iterate:

```
my_array.each do |element|
  puts element
end
```

We can use << to add things to an array:

```
>> countries << "India"
=> ["India"]
>> countries
=> ["India"]
>> countries.size
```

Hashes

Hashes can be used to store mapped information:

```
mark = {}
mark['English'] = 50
mark['Math'] = 70
mark['Science'] = 75
```

And we can define a default value:

```
mark = {}
mark.default = 0
mark['English'] = 50
mark['Math'] = 70
mark['Science'] = 75
```

The hash literal {} also allows us to create hashes with pre-filled information:

Ranges

Ranges are a cool concept in Ruby that we've used before. We can use them with the .. notation:

```
>> (1..5).each {|a| print "#{a}, " }
=> 1, 2, 3, 4, 5, => 1..5
```

We can also use them on strings:

```
>> ("bad".."bag").each {|a| print "#{a}, " }
=> bad, bae, baf, bag, => "bad".."bag"
```

They can be very useful in case statements, where you can replace lots of or operators with them:

```
grade = case mark
  when 80..100
    'A'
  when 60..79
```

Functions

As mentioned before, Ruby draws a lot of inspiration from functional programming languages, and functions are a primary building block in the language as a result.

We can define functions with def and call them without parentheses:

```
def print_line
  puts '_' * 20
end

print_line
```

It is also possible to define default arguments unlike in Java:

```
def print_line length = 20
  puts '_'*length
```