CSE M226: Programming Languages

Lecture 4: Introduction to Ruby and OOP

Rokan Uddin Faruqui

Dept. of Computer Science and Engineering
University of Chittagong, Bangladesh
Email: rufaruqui@cu.ac.bd

Blocks

A block

- is the same thing as a method, except it does not have a name, and does not belong to an object.
- is an anonymous piece of code, it can accept input in form of arguments (if it needs any), and it will return a value, but it does not have a name.
- can only be created by the way of passing them to a method when the method is called.

Block

A block is a piece of code that accepts arguments, and returns a value. A block is always passed to a method call.

Blocks

```
5.times do
puts "Oh, hello!"
end

5.times { puts "hello!" }
```

Blocks with arguments

```
1  [1, 2, 3, 4, 5].each do |number|
     puts "#{number} was passed to the block"
3  end
```

Block arguments are listed between pipes ¡code¿—¡/code¿, instead of parentheses.

Block return values

```
1  p [1, 2, 3, 4, 5].collect { |number| number + 1 }

3  p [1, 2, 3, 4, 5].select { |number| number.odd? }

5
```

- Use the method *collect* to transform an array into another array.
- Use the method *select* to select a new array with values that match a criteria defined by the block.

Inversion of Control

- In Ruby there are a lot more methods that accept blocks, and they do very different things. However, they have one thing in common:

By accepting a block, from you as a programmer, the method can pass control to you.

- This design is an example for the principle of *inversion of control*, and it's the real reason why Rubyists love blocks so much.

Inversion of Control

- Ruby allow methods to pass control to its users

"Control" in this context refers to the question who gets to decide how things work.

- In older languages, where there was no such tool, people either had to implement lots of very specific methods.
- For example, in Ruby, we don't have to define lots of methods like 'select_odd', 'select_even', 'select_lesser_than', 'select_greater_than' and so on,
- Instead, the class 'Array' only has to implement one single, very generic method for arrays: 'select'
- Since Ruby has blocks, the method can allow you (as a programmer) to specify the criterion yourself: *by passing a piece of code, in the form of block to the method.*

Iterators

- In Ruby iterators are ?chainable?, adding functionality on top of each other.
- That means that, if you do not pass a block to an iterator method, such as each, collect, select, then you?ll get an iterator object back. You can then call more methods on these iterator objects, and finally pass a block. Like so:

Iterators

```
numbers = [1, 2, 3, 4, 5].collect.with_index do |
    number, index|
  number + index
end

```

Conditionals

```
1  number = 5

3  if number.between?(1, 10)
     puts "The number is between 1 and 10"
5  elsif number.between?(11, 20)
     puts "The number is between 11 and 20"
7  else
     puts "The number is bigger than 20"
9  end
```

Nothingness and the truth

Nil

"Nothing" is a special thing in Ruby

Operators

- Arithmetical Operators
- Logical Operators
- Comparison Operators

Operators are methods

$$number = 2 + 3 * 4$$

translates as

$$number = 2. + (3. * (4))$$

- String Interpolation

- String Interpolation
- Top-level object

- String Interpolation
- Top-level object
- Some useful methods

- String Interpolation
- Top-level object
- Some useful methods
- Using Libraries

- String Interpolation
- Top-level object
- Some useful methods
- Using Libraries
- Modules

- String Interpolation
- Top-level object
- Some useful methods
- Using Libraries
- Modules
- Private methods

- String Interpolation
- Top-level object
- Some useful methods
- Using Libraries
- Modules
- Private methods
- Regular Expressions

- String Interpolation
- Top-level object
- Some useful methods
- Using Libraries
- Modules
- Private methods
- Regular Expressions