Intro to Ruby

Languages (and learning new languages)

- Try to isolate what is different about different languages
- What abstractions are used in the framework
- How can we exploit the specific abstractions

8 Steps to X-ray a new language

1. Types and typing
2. Primitives
3. Methods / functions (class vs instance)
4. Abstractions and encapsulation of classes
5. Idioms (symbols and blocks) specific to a language
    - Diff between word for word translation and "thinking in a language"
6. Libraries and packages
    - Documentation
    - Management
    - Installation
7. Debugging (breakpoint, REPL)
8. Testing

Pair Programming

- Need to prevent one person from doing all of the work at any given time
- Goal: crosschecking during a task can ensure good code and help people come up with solutions quicker
- One person is the driver and one person is the observer
    - Driver actively codes and explains what they're doing
    - Observer is thinking a bit in advance to move on from there
- For simple tasks, it is generally quicker
- For complex tasks, it generally just yields higher quality code
- Swap roles frequently

Basic Intro to Ruby

- Everything is an expression (everything has a value)
- Dynamically typed
    - Weak typing
- Everything is passed by reference

Primitives: Variables and naming

- Class names use UpperCamelCase
- `<` indicates simple inheritance
- `@@` indicates class variable (class scope)
- `@` indicates instance variable (class scope)
- Class scope $\triangleq$ protected
- Class scope = anywhere in a class
- Methods use lower_snake_case
    - Often end in `?` or `!` if question or important
- Methods can be `public` or `private`
- Constants are in UPPER_SNAKE_CASE
    - `$` indicates global
- Symbol: like an immutable sting whose value is itself

  - Indicated by `:`
  - Usually used as an enumeration
- `nil` and `false` are falsy, everything else is truthy
- Weekly typed arrays
  - Undefined array elements are `nil`
- Double quotes can be subbed in with `#{}`
- Can regex match with "=~" and the regex is between `//`
  - Trailing `i` indicates ignore case
  - Global vars `$GROUP_NUMBER` are set to the match value of that group

Methods (function) class vs. instance

- Value of method is the last expression evaluated in the func
  - Return is optional
- Default values can be defined as `foo(3, y: 5`
- Parentheses can be omitted if the correct parsing is ambiguous and will not be confusing
  - Never at the expense of clarity
  - Idiomatic
- Class methods prepended with `self.`
- Defined with `def`
- End with `end`

Everything Is an Object

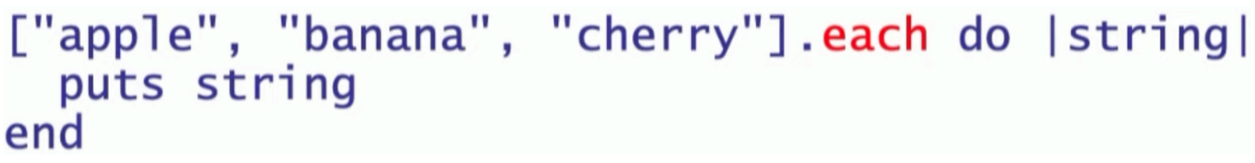
- `responds_to?` Object method checks if it has a method
- `some_array.length()` ≙ `some_array.send(:length)`
- `1 + 2` ≙ `1.send(:+, 2)`
- `array[3] = 4` ≙ `array.send(:[]=. 3, 4)`
- `a.b` ≙ "call the method b on the object a"
  - Does not mean that "b is an instance variable / attribute of a"

Abstractions and Encapsulation of classes

- `class SavingsAccount < Account` ≙ `SavingsAccount` inherits `Account`
- `initialize()` ≙ init function in ruby
  - Instance variables can initially be described in `initialize`
- `def balance=(new_ammount)` defines setter for `balance`
  - Need to define getters and setters b/c everything is protected
- Getters and setters can be both defined by `attr_accessor :value`
  - `:value` can be a string or a symbol

Blocks

- Type of idiom
- Not loops; they are lambdas
- Args go in `||`



```
["apple", "banana", "cherry"].each do |string|
  puts string
end
```

Figure 1: Screenshot_2023-08-29_at_3.08.54_PM.png

- - `each`is defined on most collections
  - `do` indicates an anonymous lambda
- Essentially map reduce
- `for i in (1..10) do |i| ... end` does the same
  - Should avoid for loops
- `1.upto 10 do |num| ... end`
- `3.times do ... end` will do something multiple times
- Collection idioms are ubiquitous
  - `x.sort` should apply to all collections
- `x.uniq.reverse` $\triangleq$ get the unique elements of x and then reverse the return of `.uniq`
  - Most make copies and do not mutate
    - Ending with `!` generally indicates mutate egs:

```
x.filter do |f|
    f.include?('e')
end.sort
```

```
    x.any? { |f| f.length > 5}
```

```
x.map(&:method)
```

- Blocks are closures
  - A closure is the set of all variable bindings in scope
  - They have their own environment
- Separate what to do vs when and where to do it
- eg shortened version: `(1..10).each { |x| ... }`