RUBY (1/2)

- Historic
- Variables / Constants
- Conditions
- Loops
- Functions
- Hash

A LITTLE CONTEXT

- Created in 1995 by Yukihiro “Matz” Matsumoto
- Open Source
- OS Independent
- Open Source
  - Written in C and Ruby
  - https://github.com/ruby/ruby/tree/ruby25
  - https://ruby-doc.org/stdlib-2.5.2/
- Great documentation (https://www.ruby-lang.org/en/documentation/)

SPECS

- Interpreted (vs compiled) -- needing a VM (requiring to install ruby)
- Untyped (will change in 3.0)
- 100% Objects (everything has methods)
- Excel in data manipulation (like python) - hash first
- Easy memory management (mark-and-sweep garbage collector)

IRB

IRB (Interactive RuBy) is a tool to interactively execute Ruby expressions read from the standard input
— https://github.com/ruby/irb

```
irb(main):005:0> 1 + 1
=> 2
irb(main):006:0> %w(hello world).map(&:capitalize).join(" ")
=> "Hello World"
```

VARIABLES

VARIABLES

- No declaration
- Untyped
- Scope (var, @var, $var)
- `nil`

SAMPLES

```ruby
# this is a comment
GREATING = 'Hello'
username = 'Hugues'
aString = "#{GREATING} #{username}" # interpolation
anotherString = 'Hello,\n #{username}'
anInt = 7
anArray = [aString, anotherString, anInt, false]
anArray[2] *= 6
anArray << anInt.to_s
anArray += ["last"]
anArray.pop
anArray # => ["Hello Hugues", "Hello,\\n \#{username}", 42, false, "7"]
```

TYPES

- Float (https://ruby-doc.org/core-2.5.2/Float.html)
- Integer (https://ruby-doc.org/core-2.5.2/Integer.html)
- Boolean (https://ruby-doc.org/core-2.5.1/TrueClass.html, https://ruby-doc.org/core-2.5.1/FalseClass.html)
- String (https://ruby-doc.org/core-2.5.2/String.html)
- Array (https://ruby-doc.org/core-2.5.2/Array.html)
- Hash (https://ruby-doc.org/core-2.5.2/Hash.html)
- Symbol (https://ruby-doc.org/core-2.5.2/Symbol.html)
- File (https://ruby-doc.org/core-2.5.2/File.html)

CONDITIONS

COMPARISON OPERATORS

```
1 != 2
3 > 2
6/2 >= 3
true != false
true == !false

3 & 1 # bitwise operators (&, |, ^)
1 << 2 # bits shifting (<<, >>)
```

IF/ELSE - TERNARY OPERATOR

```ruby
age = 15
if age < 18
  puts "You're underage"
elsif age > 80
  puts "You're a senior"
else
  puts "You just another person..."
end # don't forget this one

understood = true
puts understood ? "For sure!" : "Not really!?" # Ternary Operator
puts "I will be damn!" unless understood # inline
```

CASE/WHEN

```ruby
age = 21
case age
when 0...18
  puts "You're underage"
when 80..Float::INFINITY
  puts "You're a senior"
else
  puts "You just another person..."
end # don't forget this one
```

```ruby
age = 83
case # to use as a if/elsif statements
when age < 18
  puts "You're underage"
when age > 80
  puts "You're a senior"
else
  puts "You just another person..."
end # don't forget this one
```

LOOPS

FOR / NEXT

Iterate over a list

```
for i in [0, 1, 2, 3, 4, 5]
  print "#{i} " # 0 1 2 3 4 5
end

for i in 0..5
  next if (i % 2) == 0
  print "#{i} " # 1 3 5
end
```

WHILE / LOOP

Conditional loop

```ruby
i = 0
while i <= 5
  print "#{i} " # 0 1 2 3 4 5
  i += 1
end

i = 0
loop do
  print "#{i} " # 0 1 2 3 4 5
  i = i + 1
  break if i > 5
end
```

BREAK

Any loop can be break

```
i = 0
while true
  print "#{i} " # 0 1 2 3 4 5
  i = i + 1
  break if i > 5
end
```

EACH / TIMES

theses are actually a method (with so many others)

```
(0..5).each do |i|
  print "#{i} " # 0 1 2 3 4 5
end

6.times { |i| print "#{i} " } # 0 1 2 3 4 5
```

MAP

Map is the most common method used in Ruby (on array / hash)

```ruby
(0..5).map do |i|
  i * 2
end.join(' ') # => "0 2 4 6 8 10"
```

More on this in a minute...

FUNCTIONS

```ruby
def power(a, n)
  case
  when n.zero?
    1
  when n == 1
    a
  when n.positive?
    (n - 1).times { a = a * a }
    a
  else
    1.0 / power(a, n.abs)
  end
end
power(4, 0) # => 1
power(4, 1) # => 1
power(4, 2) # => 16
power(4, -1) # => 0.25
```

PASS REFERENCE BY VALUE!

```
def foo(bar)
  puts bar.object_id # => 70283441049480
  bar =  "new value"
  puts bar.object_id # => 70283441015700
end

bar = "value"
puts bar.object_id # => 70283441049480
foo(bar)
puts bar.object_id # => 70283441049480
```

```
def double(a)
  a *= 2
end
b = 2
double(b) # => 4
b # => 2

def upcase!(a)
  a.upcase!
end
b = 'hello'
upcase!(a) # => 'HELLO'
b # => 'HELLO'
```

ARGS VS NAMED ARGS VS HASH

```
def double(a)
  a *= 2
end
double(2)

def multiply(a, by:)
  a *= by
end
multiply(3, by: 2)
multiply(3, by: 3)

def divide(args)
  args[:numerator] /= args[:denominator]
end
divide({ numerator: 10, denominator: 2 })
divide(numerator: 10, denominator: 2)

def divide(numerator:, denominator: )
  numerator /= denominator
end
divide(numerator: 10, denominator: 2)
```

HASH

A Hash is a dictionary-like collection of unique keys and their values. Also called associative arrays, they are similar to Arrays, but where an Array uses integers as its index, a Hash allows you to use any object type.
– https://ruby-doc.org/core-2.5.2/Hash.html

>80% of Web Development is data manipulation

```
grades = { "Jane Doe" => 10, "Jim Doe" => 6 }
options = { font_size: 10, font_family: "Arial" }
grades = Hash.new
grades["Dorothy Doe"] = 9
```

```ruby
data = [
        {
                city: 'Paris',
                country: 'France',
                population: 2_161_000,
        },
        {
                city: 'Marseille',
                country: 'France',
                population: 861_635,
        },
        {
                city: 'Lyon',
                country: 'France',
                population: 496_343
        },
        {
                city: 'Toulouse',
                country: 'France',
                population: 453_317
        },
        {
                city: nil,
                country: 'France',
                population: 63_033_705
        },
        {
                city: 'New York',
                country: 'USA',
                population: 8_419_000
        },
]
data.select { |n| n[:country] == 'France' }.reduce(0) { |sum, n| sum + n[:population] } # 67006000
data.select { |n| n[:population] > 1_000_000 && n[:city] }.map { |n| n[:city] }.join(', ') # Paris, New York
data.group_by { |n| n[:country] }.map { |country, n| [country, n.reduce(0) { |sum, n| sum + n[:population] }] }.to_h # => {"France"=>67006000, "USA"=>8419000}
```