Ruby!

- Useful as a **scripting language**
  - **script**: A small program meant for one time use
  - Targeted towards small to medium size projects
- Use by:
  - Amazon, Twitter, Yahoo!, White Pages, Reddit

Interpreted

- C/C++
  - Compiled to assembly/Run directly on machine
- Java
  - Compiled to bytecode/Interpreted by JVM
- Ruby
  - Interpreted (no compilation)

irb (Ruby interpreter)

- Allows you to type commands one at a time and see results

attu.cs.washington.edu - PuTTY

```
[ngarrett@attu2 ~]$ irb
irb(main):001:0> puts "Hello World"
Hello World
=> nil
irb(main):002:0> 1 + 2 + 3
=> 6
irb(main):003:0>
```

Our First Program

- Ruby does not have a main method like Java
  - Just write your code directly in a file
- Ruby statements do not end with semicolons
- Method calls don't need parenthesis

```
File Edit Options Buffers Tools
def say_hello
  puts "Hello World"
end

# main calls say_hello twice
say_hello
say_hello
```

Expressions

- Arithmetic is similar to Java
  - Operators similar to Java
    - + - * / % (plus ** for exponentiation)
- Precedence
  - () before ** before * / % before + -
- Integers vs Real Numbers

```
irb(main):008:0> 7 / 2
=> 3
irb(main):009:0> 7.0 / 2
=> 3.5
```

Unlimited Precision

- Java
  - There is a maximum value for integers
  - There is a maximum value for longs
- Ruby
  - There is no maximum!

```
irb(main):120:0> 2 ** 200
=> 1606938044258990275541962092341162602522202993782792835301376
```

    - Fixnum
    - Bignum
  - Why the distinction?

Declaring Strings

- “” allows escape sequences

```
irb(main):123:0> i = 9000
=> 9000
irb(main):124:0> puts "Over #{i}"
Over 9000
```

```
irb(main):128:0> puts "Only \' \n\tEscape"
Only '
        Escape
```

- ” does not allow escapes (excpet for \’)

```
irb(main):125:0> puts 'Over #{i}'
Over #{i}
```

```
irb(main):127:0> puts 'Only \' \n\tEscape'
Only ' \n\tEscape
```

Variables/Types

- Don’t declare types
- Ruby is looser about Types than Java
  - Type of variable can change throughout program

```
irb(main):024:0> i = 2
=> 2
irb(main):025:0> i = "hello"
=> "hello"
```

String Multiplication

- Strings can be multiplied by integers
  - Concatenates string repeatedly

```
irb(main):026:0> "hello" * 3
=> "hellohellohello"
irb(main):027:0> "yo " * 4
=> "yo yo yo yo "
```

Strings and Ints

- Integers and Strings cannot be concatenated in Ruby
  - to_s – converts to string
  - to_i – converts to integer

```
irb(main):029:0> "hello" + 4 + 3
TypeError: can't convert Fixnum into String
        from (irb):29:in `+'
        from (irb):29
        from :0
irb(main):030:0> "hello" + 4.to_s + 3.to_s
=> "hello43"
irb(main):031:0> "42".to_i
=> 42
irb(main):032:0> 42.to_s
=> "42"
```

Loops

- The for loop
  - Java

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

  - Ruby

```ruby
5.times { |i|
  puts i
}
```

Loops

- The while loop
  - Java

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

  - Ruby

```
i = 0
while i < 5
  puts i
  i += 1
end
```

```
i = 0
until i == 5
  puts i
  i += 1
end
```

Constants

- Ruby doesn't really have constants
  - Instead declare a variable at the top of your code and it will be accessible everywhere

```
irb(main):044:0> MY_CONSTANT = "w00t!"
=> "w00t!"
irb(main):045:0> MY_CONSTANT = "different"
(irb):45: warning: already initialized constant MY_CONSTANT
=> "different"
```

  - You will get a warning if you change a constant, but you can change it anyway (bad style)

Parameters

- Parameters are declared by writing their names (no types)

```ruby
def my_method(a, b)
  puts a + b
end

my_method(1, 2)
my_method("wt", "f")
my_method([1,2], [3,4])
```

```
[ngarrett@attu4 ~]$ ruby test.rb
3
wtf
1
2
3
4
```

- May seem odd that we can pass ints, strings, or arrays

Duck Typing

- Actually, we can pass anything that has a + method
  - This is called Duck Typing
  - Why would we limit our method to only operating on objects of type Duck?
    - If it looks like a Duck and quacks like a Duck, then it's a Duck
- This allows us to write flexible, reusable code

Inspecting Objects

- How do I know whether an object has a + method?
  - You can ask the object (with the “methods” method)
  - Everything is an object in Ruby (no primatives)

```
irb(main):058:0> (5.methods - Object.methods).sort
=> ["%", "&", "*", "**", "+", "+@", "-", "-@", "/", "<<", ">>", "[]", "^", "abs", "between
?", "ceil", "chr", "coerce", "div", "divmod", "downto", "even?", "fdiv", "floor", "id2name
", "integer?", "modulo", "next", "nonzero?", "odd?", "ord", "prec", "prec_f", "prec_i", "p
red", "quo", "remainder", "round", "singleton_method_added", "size", "step", "succ", "time
s", "to_f", "to_i", "to_int", "to_sym", "truncate", "upto", "zero?", "|", "~"]
```

Default Parameter Values

- You can give a default value to parameters
  - The caller doesn't have to pass a value

```
def my_method(a=3)
  puts a
end

my_method
my_method 10
```

```
[ngarrett@attu4 ~]$ ruby test.rb
3
10
```

Math

- The Math module has methods and constants that you can use

```
irb(main):001:0> (Math.methods - Object.methods).sort
=> ["acos", "acosh", "asin", "asinh", "atan", "atan2", "atanh", "cos", "cosh", "erf", "erf
c", "exp", "frexp", "hypot", "ldexp", "log", "log10", "sin", "sinh", "sqrt", "tan", "tanh"
]
irb(main):002:0> Math.constants
=> ["PI", "E"]
irb(main):003:0> Math::PI
=> 3.14159265358979
```

- Has many of the same methods as Java

Returning Values

- Methods in Ruby return the last value evaluated (only do this if you're an expert)

```
irb(main):004:0> def my_method(a, b)
irb(main):005:1>   a + b
irb(main):006:1> end
=> nil
irb(main):007:0> my_method(2,3)
=> 5
irb(main):008:0> my_method("wt","f")
=> "wtf"
```

- You can also explicitly return values, and this is less error prone

```
irb(main):009:0> def my_method(a, b)
irb(main):010:1>   return a + b
irb(main):011:1> end
```

Reading from the Console

- Java

```java
Scanner s = new Scanner(System.in);
String line = s.nextLine();
System.out.println(line);
```

- Ruby

```ruby
line = STDIN.gets
puts line
```

If Statements

- Java

```java
if (1 < 2) {
    System.out.println("1 < 2");
}
```

- Ruby

```ruby
if 1 < 2
then
  puts "1 < 2"
end

puts "1 < 2" if 1 < 2
```

```ruby
unless 1 >= 2
then
  puts "1 < 2"
end

puts "1 < 2" unless 1 >= 2
```

elsif

- Java

```java
if (1 > 2) {
    System.out.println("1 > 2");
} else if (1 == 2) {
    System.out.println("1 == 2");
} else {
    System.out.println("1 < 2");
}
```

- Ruby

```ruby
if 1 > 2
then
  puts "1 > 2"
elsif 1 == 2
  puts "1 == 2"
else
  puts "1 < 2"
end
```

Logical Operators

- == != >= <= < > (just like Java)
- <=> (not in Java)
  - Remember, because of Duck Typing these are applicable to more than just numbers

```
irb(main):051:0> 1 <=> 2
=> -1
irb(main):052:0> 2 <=> 1
=> 1
irb(main):053:0> 1 <=> 1
=> 0
```

  - What might <=> be useful for?
- && || ! (just like Java)

Arrays

- Arrays
  - More flexible than Java, can mix types

```
irb(main):062:0> ['a', 1, 2, 'b', [4, 5, 6], 'yo']
=> ["a", 1, 2, "b", [4, 5, 6], "yo"]
```

  - Many useful methods
    - map, sort, delete, each, min, max, include?, select, shuffle, slice
  - Negative Indexing

```
irb(main):065:0> a = ['a', 'b', 'c']
=> ["a", "b", "c"]
irb(main):066:0> a[-1]
=> "c"
irb(main):067:0> a[-2]
=> "b"
```

Hashes

- In Java these are Maps
  - (you will learn about them in 143)
  - Ruby's are more flexible; you can mix types
- Kind of like Arrays, but instead of indexing by numbers, you index by whatever you want

```
irb(main):068:0> a = {1 => 'a', 'foo' => 'bar'}
=> {1=>"a", "foo"=>"bar"}
irb(main):069:0> a[1]
=> "a"
irb(main):070:0> a['foo']
=> "bar"
```

Multiple Assignment

- Can assign to and return multiple items at a time (uses arrays under the covers)

```
irb(main):087:0> def my_method(a, b, c, d)
irb(main):088:1>   return a + b, c + d
irb(main):089:1> end
=> nil
irb(main):090:0> i, j = my_method(1, 2, 3, 4)
=> [3, 7]
irb(main):091:0> i
=> 3
irb(main):092:0> j
=> 7
```

Reading Files

- Java

```java
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class Test {
    public static void main(String[] args) throws IOException{
        Scanner s = new Scanner(new File("Test.java"));
        while (s.hasNextLine())
            System.out.println(s.nextLine());
    }
}
```

- Ruby

```ruby
File.open("test.rb", "r").each_line do |line|
  puts line
end
```

Writing Files

- Java

```java
import java.io.File;
import java.io.IOException;
import java.io.PrintStream;

public class Test {
    public static void main(String[] args) throws IOException {
        PrintStream out = new PrintStream(new File("out.txt"));
        out.println("first line");
        out.println("second line");
    }
}
```

- Ruby

```ruby
File.open("out.txt", "w").puts(["first line", "second line"])
```