GENERAL ASSEMBLY

BEWD
INTRO TO RUBY

AGENDA

The Ruby Programming Language	20 MIN
Data Types	15 MIN
Basic Operators	15 MIN
IRB (Interactive Ruby Shell)	30 MIN
Variables	20 MIN
Control Structures	50 MIN

PHILOSOPHY 101

HUMANS COMMUNICATE THOUGHTS AND IDEAS THROUGH IMAGES, SOUNDS, WORDS AND MANY OTHER FORMS OF EXPRESSION.

COMPUTERS UNDERSTAND ZEROS AND ONES.

IN ORDER TO UTILIZE THE POWER OF A COMPUTER WE MUST LEARN TO EXPRESS OURSELVES IN A WAY THAT A COMPUTER CAN UNDERSTAND.

PROGRAMMING LANGUAGES ARE OUR VEHICLE FOR EXPRESSION.

A TECHNICAL DEFINITION

> *A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer.*

EN.WIKIPEDIA.ORG/WIKI/PROGRAMMING_LANGUAGE

A WORD ON TECHNICAL JARGON

Never forget that everything we discuss in this course was imagined and created by people just like you and me. Technical definitions can be confusing and demoralizing, but if you look deeply are more often than not grounded in common sense principles you've known your whole life.

> *Modern [theory], bound up with computers and equations, hides the fact that it traffics in truths known at some level by everyone.*

"THINKING IN SYSTEMS, A PRIMER", DONELLA H. MEADOWS

SO, SO MANY PROGRAMMING LANGUAGES

WHY SO MANY?

- New programming languages emerge when a developer or group of developers run up against the limitations of existing languages to solve problems in a particular problem space.
- “Run up against the limitations of” usually means that for a specific set of problems, designing, creating, and maintaining a new set of specialized tools will be more efficient and expressive than using the existing ones.
- If you scoop and spear your food, a spoon will scoop and a fork will spear. If you’re scooping and spearing all day everyday, maybe you should invest in a spork. Maybe. If you’re cool with being that kid at the lunch table with a spork.

BEWD BOSTON

DATA TYPES AND OPERATORS

DATA TYPES

- A frequent programming task is handling data (storage, manipulation, etc.)
- Programming languages have data types for the purposes of standardizing that data
- Data types are transparent in Ruby (we don't specify them), but it's very important to know which ones exist.

DATA TYPES IN RUBY

- Boolean
- Numeric
- String
- Symbol
- Array
- Hash
- Range

RUBY DATA TYPES

BOOLEAN

- **Boolean** represents the idea of true and false (yes and no, one and zero)
- Booleans are typically used to control the flow of a program
- Ruby provides the keywords `true` and `false` for our Boolean needs

IF RAMEEN WERE A PROGRAM

RUBY DATA TYPES

NUMERIC

- **Numeric** is a data type used for numbers
- Numbers can be integers (e.g. `4`, `8`, `15`, `16`, `23`, `42`)
- Numbers can be floating-point (e.g. `4.35678`, `34.5696854`, `138583.5`)
- Powers of 10 can be expressed by using the letter `e` or `E` (e.g. `6.24e19`)
- Floating-point numbers must have a leading zero (i.e. `0.1`, not `.1`)

RUBY DATA TYPES

STRING

- The **String** data type is for representing text
- Strings can be used with single quotes (e.g. `'hello world'`)
- Or double quotes (e.g. `"hello world"`)
- Double quotes provide a few perks, including what's called **string interpolation**
- String interpolation allows us to insert expressions into our strings that are evaluated when the code is executed. We use `#{}` for this and our code goes between the curly braces.

```
>> "I am #{23 + 1}" # returns "I am 24"
```

```
>> 'I am #{23 + 1}' # returns "I am #{23 + 1}"
```

RUBY DATA TYPES

SYMBOLS

- **Symbols** are generally used as unique identifiers in place of strings where the exact content of the string isn't really that important.
- They are denoted with a `:` followed by the content of the symbol and shouldn't include spaces
- They are immutable (can't be changed), unlike strings which are mutable
- Their immutability provides gains in efficiency (faster program execution) in certain situations
- For the purposes of this course, efficiency gains are negligible, so don't worry about it!

```
>> :bewd
   => :bewd
```

BASIC OPERATORS

- Ruby defines basic **operators** for numerics including addition, multiplication, and others

Operator	Meaning	Example	Return value
+	Addition	10 + 8	18
-	Subtraction	10 - 8	2
*	Multiplication	10 * 8	80
/	Division	10 / 10	1
%	Modulus (remainder)	10 % 8	2
**	Exponent	10 ** 2	100

OF INTEREST

SOME BASIC ARITHMETIC OPERATORS WORK ON STRINGS AS WELL. GIVE IT A SHOT.

INTERACTIVE RUBY SHELL (IRB)

RUBY IN THE COMMAND LINE

- Because Ruby is interpreted, we need an interpreter to run our Ruby code
- IRB is a Command-line program that allows us to write Ruby to be evaluated by the Ruby interpreter
- Each line of code is passed to the interpreter when we press the `return` key
- The result of our code is **returned** by the interpreter to IRB and displayed in the Terminal

```
~/Desktop $ irb
2.0.0p247 :001 > 2 + 2
 => 4
```

Starting IRB from the Terminal

Entering Ruby code and pressing return

The result

USEFUL COMMANDS IN IRB

EXIT

```
exit
```

CLEAR

```
system 'clear'
```
or
```
cmd + k
```

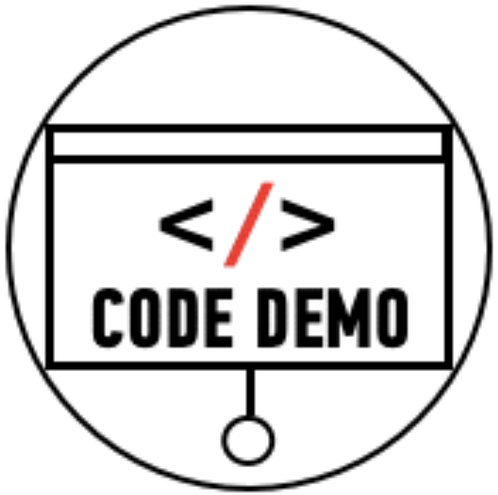
- Clears the window of old entries

INTERRUPT COMMAND EXECUTION

```
ctrl + c
```

- Stops execution of code by the interpreter
- Very useful if code is taking a really long time to execute (e.g. infinite loop)

DATA TYPES AND OPERATORS CODE DEMO



TO GET ROLLING

Type Command ⌘ + Space and open Terminal

A BETTER WAY TO WRITE/RUN RUBY PROGRAMS

- IRB is great for learning and experimentation but we don't really use it beyond that.
- We almost always write programs in a text editor and save them with the extension **.rb** (for Ruby)

EXECUTING A .RB FILE

- To execute a .rb file, simply navigate to the file in Terminal using standard UNIX commands
- Then, run the file by issuing the `ruby` command, followed by a space and the name of the file

```
~/Desktop $ ruby a_ruby_file.rb
```

VARIABLES

VARIABLES

KEEPING THE RESULTS

- When we were working in IRB, we entered expressions like `2 + 2` and `4 * 3`
- This code was interpreted and executed by the interpreter, and the result was returned to us (remember that IRB automatically prints the result to the window after receiving it.)
- But what if we'd like to hold on to the result of one of those expressions for future reference?
- Just like in algebra, Ruby has **variables** for referencing expression results

THE FOUR VARIABLE TYPES

RUBY DEFINES FOUR TYPES OF VARIABLES. THEY ARE LOCAL, INSTANCE, CLASS, AND GLOBAL.

LOCAL VARIABLES

LOCAL VARIABLES

- The most basic kind of variable is called a **local variable**
- As a matter of convention, local variables should be **lower_snake_case**
- To assign to a variable (set its value), use the = operator

```
2.0.0p247 :001 > this_is_a_variable = 2 + 2
 => 4
```

- To use the underlying value of a variable (called “reading”), just reference the variable

```
2.0.0p247 :002 > this_is_a_variable * 2
 => 8
```

VARIABLES (CONT'D)

OTHER DATA TYPES

- Variables can hold all data types, not just numerics

```
2.0.0p247 :001 > a_string = “hi there”
 => “hi there”
2.0.0p247 :002 > a_string
 => “hi there”
```

GOTCHAS

- Variable names can't include spaces (i.e. `this is not ok = 2`)
- If you write code that reads a variable that doesn't exist, you'll get a **NameError**

CONTROL STRUCTURES

CONTROL STRUCTURES

- Let's consider the following three line Ruby program

```
x = 1
y = 3
x / y
```

direction of program execution

- The program is executed sequentially from top to bottom
- But what if we set y equal to zero?
- Womp, womp. We get a **ZeroDivisionError**
- **Control Structures** are tools for rerouting/controlling program flow

CONDITIONALS

- The most common control structure is called a **conditional**
- Conditionals allow us to execute some code if a condition is satisfied (true or false)

IF STATEMENTS

- **If** statements execute code if some condition is true
- Note the **end** statement which is required

```
if condition
   code
end
```

```
x = 1
y = 0
if y != 0
   x / y
end
```

CONDITIONALS (CONT'D)

IF/ELSE STATEMENTS

- **If/else** statements behave like if statements and also allow us to specify code if the primary condition is false

```
if condition
   code
else
   code
end
```

```
x = 1
y = 0
if y != 0
   x / y
else
   puts "undefined"
end
```

CONDITIONALS (CONT'D)

ELSIF STATEMENTS

- **Elsif** statements allow us to specify multiple conditions and code to execute

optional →

```
if condition_1
   code
elsif condition_2
   code
elsif condition_3
   code
else
   code
end
```

```
x = 1
y = 0
if y == 0
    puts "undefined"
elsif y < 0
    puts "y negative"
    x / y
elsif y > 0
    puts "y positive"
    x / y
end
```

BOOLEAN OPERATORS

- **Boolean operators** are very useful when used with conditionals

Operator	Meaning	Example	Return value
>	greater than	2 > 1	true
<	less than	4 < 5	true
>=	greater than or equal to	1 >= 1	true
<=	less than or equal to	2 <= 3	true
||	or	false || true	true
&&	and	true && true	true
==	equals	1 == 1	true
!=	does not equal	2 != 1	true

LOOPS

- Loops are used to execute the same chunk of code repeatedly until some condition is met

WHILE LOOP

- The most basic loop is called the **while loop**
- The while loop repeatedly executes a chunk of code while some condition is true

```
while condition do
   code
end
```

```
x = 0
while x <= 10
   puts x
   x = x + 1
end
```

CONTROL STRUCTURES EXERCISE

TO GET ROLLING

Type Command ⌘ + Space and open Terminal

THE BIG PICTURE

- Programming languages allow us to express ideas in a language that computers can understand
- Many programming languages exist to meet the specific needs of developers
- Ruby is an interpreted*, Object-Oriented Programming Language
- Ruby specifies several **data types**, including **Numeric**, **Boolean**, **String**, and **Symbol**
- We can store data for later use in **variables**
- To control the flow of execution of our program, we use **control structures**
- **Conditionals** (if, else, and elsif) and **Loops** (while) are examples of control structures
- To experiment with Ruby code in a sandbox where we have access to a Ruby Interpreter, we use **IRB** (Interactive Ruby Shell)

FOR NEXT CLASS

- Homework 1
- Enjoy the weekend!