**WDI - Day 2 Notes**

**AGENDA**

**1) Types of objects in Ruby (or classes) 🡪 array, hash, string**

**2) Conditions and flow control**

**3) Git Intro**

**4) Functions**

----------------------------------------------------------------------------------------------------------------------

**1) TYPES OF OBJECTS IN RUBY (OR CLASSES)**

Hashable Real World Things:

* bag
* garage
* dictionary
* unsorted heap of clotheswww.gmai
* index of information

Bag = Hash.new

A [Hash](http://www.ruby-doc.org/core-2.1.1/Hash.html) is a dictionary-like collection of unique keys and their values. Also called associative arrays, they are similar to Arrays, but where an [Array](http://www.ruby-doc.org/core-2.1.1/Array.html) uses integers as its index, a [Hash](http://www.ruby-doc.org/core-2.1.1/Hash.html) allows you to use any object type. Hash is unsorted vs. Array is sorted.

Bag [: name]

* Name is a symbol that serves as a key.
* If we do not put a key there, hash will return nil.

Type vs. Instance

Band vs. RadioHead

**ADDITIONAL HASH NOTES FROM WEB**

A [Hash](http://www.ruby-doc.org/core-2.1.1/Hash.html) can be easily created by using its implicit form:

grades = { "Jane Doe" => 10, "Jim Doe" => 6 }

Hashes allow an alternate syntax form when your keys are always symbols.

options = { :font\_size => 10, :font\_family => "Arial" }

You could also write it as:

options = { font\_size: 10, font\_family: "Arial" }

Each named key is a symbol you can access in hash:

options[:font\_size] **# => 10**

A [Hash](http://www.ruby-doc.org/core-2.1.1/Hash.html) can also be created through its [::new](http://www.ruby-doc.org/core-2.1.1/Hash.html#method-c-new) method:

grades = Hash.new

grades["Dorothy Doe"] = 9

bedroom[:dresser] [:drawer1] = {}

bedroom[:dresser] [:drawer2] = {}

bedroom[:wall\_paint\_color] = “blueish”

bedroom[:wall\_number] = 4

bedroom[:dresser] [:drawer\_number] = 2

**2) CONDITIONS AND WORKFLOW**

DRY

Don’t Repeat Yourself

Repeating yo’self means

* hard to find where changes must be made
* code runs faster or more smoothly
* easier to read by other

Conditionals

* if
* else
* elseif
* case/when

**PERSONAL SIDENOTE: DIFFERENCE BETWEEN PUTS & PRINTS**

Print will print everything in an array, puts will not, for example:

Puts also adds a new (blank) line after the thing you want it to print.

print [nil, 1, 2]

gives

[nil, 1, 2]

but

puts [nil, 1, 2]

gives

1,2

Boolean Expression

Ends in true or false

user\_age = gets.chomp.to\_i

if user\_age >= 21

elseif user\_age >= 18 && user\_age < 21

puts “not 21 but hang in there, almost there. Ps I know a place that makes good fake id’s”

else user age < 18

puts “stay in school kiddo”

end

**3) GIT OVERVIEW**

Allows you to save code

Also allows you to create a server and send stuff to it

GitHub is essentially a really suped up git server.

**4) FUNCTIONS**

Functions store blocks of code (or procedures) for later use

Definition:

* def func\_name(parameter1, ….)

# insert your desired code here

result = parameter1+4

# the last line is automatically returned

result

end

* def square(number)

number\*number

end

Call or invocation:

- func\_name(10) # => 14

arguments 🡪 **function** 🡪 return value

(input) (output)

def sqrt(number)

return false if number < 0

math.sqrt(number)

end

require\_relative 'functions.rb' # opens file in same file source directory

Spec = test

describe “the ‘square’ method” do

it “should return the square of a number” do

square(5).should eq(25)

end

Loop

* creates a loop in your code
* Example: imbedding a function inside the same function