Ruby Basics Two

Gathering data from a user

If you're running your Ruby program from the command line, you can use gets to have the user type in some input:

```
puts "What is your name?"

name = gets

puts "Your name is"

puts name
```

Symbols vs. Strings

Why are symbols and strings different? A symbol is immutable. Take the following code sample:

```
"hello" << "world"

> "helloworld"

:hello << :world

>NoMethodError: undefined method '<<' for :hello:Symbol
```

It's impossible to concatenate one symbol to another. This means that symbols are inherently more efficient as the Ruby interpreter always knows what their value is

Symbols vs. Strings

- Two symbols with the same value can be stored at the same location in memory, saving resources
- Since their value cannot be changed - an identical symbol simply becomes a reference to the already stored value of the first symbol
- It also makes comparing two symbols far more efficient - the Ruby interpreter can compare the symbols' object id to see if it matches rather than comparing the actual value of the symbol, as would be necessary to compare two mutable strings

Comments

Comments, a way to write in your notes or disable code, are demarcated in Ruby using the #

```ruby
# This won't be run

puts "But this line will be run"

# puts "And this one will not be"
```

Exercise: Basic Data Types

Try to use the basic data types inside of a Ruby file. Your file should include comments that describe what each data type is and what it is used for, for example:

```
# A string is used to store a collection
# of characters. It is demarcated using
# quotes.

"Hello World"
```

Conditions

A condition in Ruby is a test for something that evaluates to the boolean `true` or `false`

```
2 == 2

> true

"Hello" == "Hey"

> false
```

Exercise

Try Running a few conditions inside of IRB to see what they evaluate to.

For example:

```
$ irb
001> 2 == 2
> true
```

Logic

- A logical statement evaluates a number of conditions and executes code based on their truthiness
- Think of it as two paths diverging in a wood, if one condition is true, follow the left path, if false, follow the right path

The `if` statement

The `if` statement allows us to run code only if a certain test evaluates to `true`

```
if 5 > 10
  puts "You'll never see this returned because 5 is not greater than 10"
end

if 5 < 10
  puts "But you'll definitely see this"
end
```

The `else` statement

The `else` statement runs only if the statement in the `if` statement is false

```
if 5 < 10
  puts "You'll never see this because 5 is not greater than 10"
else
  puts "You will see this though, since 5 < 10 evaluates to false"
end
```

The `else if` statement

If you want to run another test before getting to `else`, you'll want to use `else if`

```
if 5 > 10
  puts "You'll never see this because 5 is not greater than 10"
elsif 5 == 5
  puts "Yes, 5 really is equal to 5, so this will be returned"
else
  puts "We won't get here because our else if evaluates to true"
end
```

Logic

Another logic example:

```
a_number = 10

if false
  puts "you’ll never see this output"
elsif a_number == 30.1
  puts "Your number is equal to 30.1"
else
  puts "None of the other conditions are true, so we end up here"
end
```

Logic - AND (&&)

Use the && operator (AND) if you'd like both conditions to determine the final value:

```
a = 5

a == 5 && a == 6

> false
```

Logic - OR (||)

Use the || operator (OR) to tack on another condition to test. If either condition is true, the entire statement returns true.

```
a = 5

a == 5 || a == 6

>true
```

Ruby syntax note

Ruby is different from many other programming languages in that the(,), {, and } are either optional or completely unnecessary to write it

example of "block" syntax in Ruby for logical statement

```
if true
  puts "this is true"
end
```

example of "block" syntax in JavaScript for logical statement

```
if(true){
  console.log("This is true")
}
```

Exercise

Add some examples of conditionals and logic to the "cheat sheet" file you created for the last exercise. Be sure to include comments explaining what each line of code is doing. For example:

```ruby
# This is an example of a conditional. Ruby will

# evaluate this code to true, which is of the

# boolean data type

2 == 2
```

Loops

- Are repetitive blocks of code where at least one variable inside the block typically changes each time the loop is run
- In Ruby, there are a few basic kinds:
  - `for`
  - `while`
  - `each`

for loop

```
for i in 0..4
  puts "currently on the number " + i + " iteration of this loop."
end
```

For the variable `i`, representing the range of numbers from 0 to 4, loop over the range assigning i to whichever variable in the range we're currently "on"

each loop

```ruby
#setup an array of strings
awesome_array = ["This", "is", "awesome"]

#for each item in the array
awesome_array.each do |item|

  #run this block of code - the current item is
  #aliased to the variable name "item"
  puts item
end
```

while loop

```ruby
some_number = 0 # set a counter up


while some_number < 5 # until this condition is no longer true
  # execute the code inside this block

  puts some_number

  # add one to some_number each time to get closer to >= 5
  # which will end the loop
  some_number = some_number + 1
end
```

Functions

- Functions are a way to alias a block of code with a name and call it later on
- They can take arguments, which are like variables used only inside the function
- Typically, they return a value

Function syntax

```ruby
def add_two(n)
  n + 2
end

a = add_two(5)

puts a
# This would print out 7
```

Exercises

Create the following functions:

- Adds five to argument given
- Multiplies argument given by 15
- Performs a mathematical operation using four integer and/or float arguments
- Prints the argument given four times
- Prints an uppercase version of the argument given