START UP INSTITUTE RAMPUP

INTRO TO RUBY
WEEK 2

REVIEW

1. Ruby Strings, Fixnum, Float, Boolean, and Nil.
2. Create and assign local variable with assignment operators.
3. Define and Identify methods.
4. Write and execute code in IRB or with a Text Editor.
5. Allow user input and print or put.

QUESTIONS?

SHOW & TELL TIME!

RUBY COMPARISON OPERATORS

Ruby comparison operators compares the two operands and returns either true or false.

1. == : compares if the value of the two operands are equal. If yes, it will return true **eg. 1 == 1**
2. != : compares if the value of the two operands are equal. If not, it will return true **eg. 1 != 2**
3. > : checks if the value of the left operand is greater than the right operand. If yes, it will return true **eg. 3 > 2**
4. < : checks if the value of the left operand is less than the right operand. If yes, it will return true **eg. 3 < 4**
5. >= : checks if the value of the left operand is greater than or equal to the right operand. If yes, it will return true **eg. 3 >= 3 and 3 >= 1**
6. <= : checks if the value of the left operand is less than or equal to the right operand. If yes, it will return true eg. **4 <= 5 and 4 <= 4**

Now go into IRB and try these out!

CONTROL FLOW - IF CONDITIONAL

The IF statement is one of the first type of branching you learn when programming. If it is true, do one thing; If it's not, do something else. Like this:

```
puts "what is your name?"
name = gets.chomp
if name == "bob"
  puts "Great name!"
elsif name == "joe"
   puts "That's a pretty common name"
else
   puts "Never heard of that name before"
end
```

If as a modifier: When you have an if statement without elsif or else Ruby allows you to write it like this:

```
puts "Great name!" if name == "bob"
```

Isn't Ruby just so eloquent? YES! And as a bonus you can also write **If** statements like this:

```
if name == "joe" then puts "That's a pretty common name"
```

UNLESS

Unless is the opposite of the **If** statement. In most programming languages, we want to reverse the return of the conditional, we have to negate it, usually with the !(bang) method.

```
engine_on = true
if !engine_on
  puts "The engine is off."
end
```

we can use the unless method instead.

```
unless engine_on
  puts "The engine is off"
end
```

Or even better. We can write it with one line:

```
puts "The engine is off" unless engine_on
```

CASE / WHEN STATEMENT

Case statements are similar to the switch/case statements in other languages. If you have many elsifs within your if statement. You can clean it up by using the case/when statement. Like this:

```
age = 15
case age
when age == 15
  puts “I’m 15 too!”
when age == 14
  puts “You’re one year younger than me”
when age == 16
  puts “you’re one year older than me”
when age < 14
  puts “you’re just too young”
end
```

RUBY LOGICAL OPERATORS

1. || - Returns true if either side is true. If the left side is true it will never check the right side.
    a. **true** || **true** is true
    b. **false** || **true** is true
    c. **true** || **false** is true
    d. **false** || **false** is false

2. && - Returns true if both side is true.
    a. **true** && **true** is true
    b. **false** && **true** is false
    c. **true** && **false** is false
    d. **false** && **false** is false

3. or - same as ||
4. and - same as &&
5. not - reverse logical state
6. ! - reverse logical state

The **or** and **and** are a little bit different from **||** and **&&**

when you have something convoluted like this:

age = ***15 && age / 2***

It gives you an error because it's reads

age = ***(15 && age) / 2***

what you really want is: ***age = 15 and age / 2***

which is the same as: ***(age = 15) && (age / 2)***

WHILE LOOP

What if you want to do something over and over and over again until a condition is met? Ruby has the **while** loop for that. A while loop will continue to execute until the condition is false. If it never returns false it will never stop and that's when we're in an of infinite loop.

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

You can also use the break keyword to stop and get out of the loop

UNTIL LOOP

The **until** loop is the opposite of the while loop. It will continue the loop until the condition is true.

```
days_left = 7

until days_left == 0
  puts “There are still #{days_left} days left in the week”
  days_left -= 1
end
```

NUMBER GUESSING GAME

Now that you have a pretty good idea of conditionals and comparison operators let's build a number guessing game! Break into pairs and write a program that does the following

1. Randomly chooses a number between 1 and 100 (hint: use rand(100))
2. User has 5 chances to guess the right number
3. Ask for a guess. If they guess right then congratulate them and end the game.
4. If they guess wrong then tell them if it's higher or lower, tell them how many chances they have left and let them guess again.
5. once all 5 of their chances are used up tell them they have no more chances, show the answer and end the game.

GIT

Git is a version control system. It allows you to “save” the current version of your code. Technically it takes a snapshot of your code and a pointer to the previous version. To check if you have git installed type ‘***git --version***’. You should see git version 2.3.1

To start tracking your files make sure you’re in the right directory. We recommend that you create a folder called rampup and work from there throughout the course. Once you’re inside the rampup folder git can start tracking your folder by typing ‘***git init***.’

git status - shows the status of the files. They can be untracked, modified, unstaged or staged and ready to commit. You will use this frequently to check the status.

git add *filename* - puts the file into staging area so that it can be committed(saved). It’s usually green. You can add as many files as you like. If you want to add all the changed files you can use **git add .**

git commit -m “descriptive message”. This command commits the changes of the files you added. The -m is used to add a message to your commit so you can easily reference what that commit is about.

Github

Git and Github are two different things. Github is a social network platform that allows users to share their code which can be either open source or private. You can use git to push code you've committed to Github.

1. Create a new repository called RampUp
2. Look for the url that looks like https://github.com/username/rampup.git
3. Go to your console make sure you're in the rampup directory
4. Type in **git remote add origin https://github.com/username/rampup.git**
5. Now you can push your committed code to Github by using **git push -u origin master**

Lab

1. Write a program that prints out the complete lyrics to “99 bottles of beer on the wall.”
2. Write a Deaf Grandma program. Whatever you say to grandma(user input) she should respond with HUH?!, SPEAK UP SONNY!, unless you shout it(type in all CAPS). If you shout, she can hear you and yells back NO, NOT SINCE 1938! Grandma should shout a different year each time; random between 1930 to 1980. You can’t stop talking to grandma until you shout BYE.
3. Add on to the above. Grandma really loves your company and doesn’t want you to go unless you shout BYE three times in a row. So if you say BYE twice and then something else you have to say BYE three times again.