Functions

What is a function? Remember this syntax from high school math?

$f(x) = x^2$
$f(1) = 1$
$f(2) = 2$
$f(3) = 9$

f(x) is a function. If values are substituted, the results can be plotted on a graph, showing the relationship between inputs and outputs.

Computer functions work similarly. There is an input, the function does something to the input, and results in an output.

Let's imagine we want to calculate the area of a room, given the width and length.

```
a = w * h
```

How would you turn this into a function? Functions can have no inputs up to an infinite number of inputs. A function will only return one output, or no output (nil). The inputs are called parameters.

```
function(parameters)
```

In Ruby, functions start with **`def`** and end with **`end`**. You can call the function whatever you like, but function names in Ruby use the same naming conventions as variables. Parameters are separated by commas.

```
def area(length, width)
  x = length * width
  x
end
```

When you call a function, the parameters are passed into the function. Inside the function, variables within the function are visible. So the area function can read the length, width and x variables. Unless you tell it otherwise, the last thing that gets executed in a function is the output. In this case, that is the 'x' on the line before end.

But you don't actually need either x, since when the inputs are multiplied together, the last thing that gets executed is this calculation, so this is what will be returned.

```
def area(length, width)
  length * width
end
```

Functions are like tools, in that they are only useful when you use them. In order to use a function, you have to call it. To do this, call the function name and provide the parameters.

```
area(3, 5)
```

What if you wanted to store the output somewhere? You can create a variable, or many variables, since once you create a function, you can use it over and over.

```
x = area(3, 5)
y = area(9, 9)
z = area(3, 7)
```

Let's use this by saving the output and using it later.

```
a = area(3, 5)
```

Once this function is done, a is equal to 15.

```
a = 15
```

Now imagine that we want to compute volume.

```
def vol (l, w, h)
  l * w * h
end
```

Assign some values to the variables.

```
l = 3
w = 5
h = 3
```

Now to call the volume function, use:

```
vol(l, w, h)
```

What if you wanted to compute this a different way? You could use the a variable that has the value of the area function.

```
def vol2 (a, h)
  a * h
end
```

We know the length because we assigned the variable, and we can use the area function to compute that value. To use the original volume formula, change the calculation to use the area function.

```
def vol(l, w, h)
  l * area(w, h)
end
```

Let's do some work in the terminal to elucidate some of these ideas.

```ruby
def area(length, width)
  length * width
end
```

```ruby
def volume(length, width, height)
  length * width * height
end
```

Save the file and enter pry in the terminal, then load in the file (in single quotes).

```
~/Documents/wdi3/ruby/lectures ✗ master $ pry

[1] pry(main)> load '2013-01-29.rb'

=> true
```

We can call the functions, but we get errors because the function is looking for parameters.

```
[2] pry(main)> area

ArgumentError: wrong number of arguments (0 for 2)

from 2013-01-29.rb:47:in `area'

[3] pry(main)> volume

ArgumentError: wrong number of arguments (0 for 3)

from 2013-01-29.rb:51:in `volume'
```

Try again and include parameters.

```
[4] pry(main)> area(3,5)

=> 15

[5] pry(main)> area(3,7)

=> 21

[6] pry(main)> area(3,9)

=> 27

[7] pry(main)> volume(3,9,3)

=> 81
```

Now let's imagine that we want to calculate the area of a triangle. A triangle is half the area of a square, so divide the result of the area function by 2.

```
[8] pry(main)> area(6,6)/2

=> 18
```

Back in Sublime, let's define functions for square and cube. Make sure to save the file.

```ruby
def square(x)
  x * x
end
```

```ruby
def cube(x)
  x**3
end
```

Now go back into pry, exit it, re-enter and re-load the file. Pry returns true if the file loaded correctly.

```
~/Documents/wdi3/ruby/lectures ✗ master $ pry

[1] pry(main)> load '2013-01-29.rb'

=> true
```

Set some variables, then run the functions passing the variables into the functions.

```
[2] pry(main)> a = 3

=> 3

[3] pry(main)> b = 4

=> 4

[4] pry(main)> c = 5

=> 5


[5] pry(main)> square(a)

=> 9

[6] pry(main)> cube(c)

=> 125
```

Now imagine that you want to get the area of a^2 and b^2

```
[7] pry(main)> area(square(a), square(b))

=> 144
```

What's happening here? The area function will call the square function for a (3 * 3 or 9) and for b (4 * 4 or 16) and multiply them together (9 * 16 or 144).

Now figure out the volume of the squares of a, b and c.

```
[8] pry(main)> volume(square(a), square(b), square(c))

=> 3600
```

Now find the volume of the same thing square the cubes of each of the variables.

```
[9] pry(main)> volume(square(cube(a)), square(cube(b)), square(cube(c)))

=> 46656000000
```

The point here is that you can call functions within functions.

Functions are not just for math!

```ruby
def name_tag_generator(first, last, gender, age)
  if gender == "f"
    if age <19
      puts "Miss #{first} #{last}"
    else
      puts "Ms #{first} #{last}"
    end
  else
    puts "Mr #{first} #{last}"
  end
end
```

Enter the first and last names, gender and age. If the gender is 'f' and age is less than 19, use one greeting; if age is 19 or more, use another greeting; and if gender is not 'f', use yet another greeting. Try it out in the terminal.

```
[2] pry(main)> name_tag_generator("chyld", "medford", 'm', 20)

Mr chyld medford

=> nil
```

```
[3] pry(main)> name_tag_generator("lucy", "medford", 'f', 12)

Miss lucy medford

=> nil

[4] pry(main)> name_tag_generator("lucy", "medford", 'f', 23)

Ms lucy medford

=> nil
```

Why is there a line with nil? This is because **`puts`** is also a function and the last line of a function is always returned, which in this case returns nil.

Save the code we've worked on to GitHub.

```
~/Documents/wdi3/ruby/lectures ✗ master $ b

~/Documents/wdi3/ruby ✗ master $ git status

# On branch master

#

# Initial commit

#

# Untracked files:

#   (use "git add <file>..." to include in what will be committed)

#

#  lectures/

nothing added to commit but untracked files present (use "git add" to
track)

~/Documents/wdi3/ruby ✗ master $ git add .

~/Documents/wdi3/ruby ✗ master $ git status

# On branch master

#

# Initial commit

#
```

```
# Changes to be committed:

#   (use "git rm --cached <file>..." to unstage)

#

#  new file:   lectures/2013-01-29.rb

#

~/Documents/wdi3/ruby ✗ master $ git commit -m 'lecture 2013-01-29'

[master (root-commit) 1032dcf] lecture 2013-01-29

 1 file changed, 73 insertions(+)

 create mode 100644 lectures/2013-01-29.rb

~/Documents/wdi3/ruby ✓ master $ git push origin master

Counting objects: 4, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (4/4), 817 bytes, done.

Total 4 (delta 0), reused 0 (delta 0)

To https://github.com/username/wdi3-ruby.git

 * [new branch]      master -> master
```

Get in the habit of saving all your code to GitHub so at the end of the class, you have all your work documented.