Function Junction

what's your... I already used the word "function"

Recap!

- Dictionaries
- Lists
- Tuples

```
>>> foo = [1, 2, 3]
>>> bar = foo
>>> bar
```

```
>>> foo[2] = 5
>>> bar
```

Even short programs can get complicated.

Controlling complexity

- At least you're not writing in machine code!
- For loops and if statements avoid goto
- Next method: functions

Don't
Repeat
Yourself

Okay where did I repeat myself?

Anatomy of a function

```
def pluralize(word):
    if word[-1] == 'y':
        return word[0:-1] + "ies"
    elif word[-1] == 's' or word[-1] == 'o':
        return word + "es"
    else:
        return word + "s"
```

`def` is the keyword for defining a function

Anatomy of a function

```
def pluralize(word):
    if word[-1] == 'y':
        return word[0:-1] + "ies"
    elif word[-1] == 's' or word[-1] == 'o':
        return word + "es"
    else:
        return word + "s"
```

`pluralize` is the name of this function. You can name a function anything you can name a variable.

Anatomy of a function

```
def pluralize(word):
    if word[-1] == 'y':
        return word[0:-1] + "ies"
    elif word[-1] == 's' or word[-1] == 'o':
        return word + "es"
    else:
        return word + "s"
```

This is the place for arguments the function accepts as input. They go in parens. In this case there is only one, but there could be a number of them separated by commas.

Anatomy of a function

```
def pluralize(word):
    if word[-1] == 'y':
        return word[0:-1] + "ies"
    elif word[-1] == 's' or word[-1] == 'o':
        return word + "es"
    else:
        return word + "s"
```

The **body** of the function runs whenever you call it. You would call this function like so:

```
pluralize("potato")
```

Anatomy of a function

```
def pluralize(word):
    if word[-1] == 'y':
        return word[0:-1] + "ies"
    elif word[-1] == 's' or word[-1] == 'o':
        return word + "es"
    else:
        return word + "s"
```

The **`return`** keyword specifies the value that comes out of a call to this function. The first **`return`** you hit immediately leaves the function with that value.

Okay so that's at least shorter, with less repetition. But wait, there's more...

Think less.

While you're writing the code that uses the function, think about **what the function does**, not how.

ABSTRACTION BARRIER DO NOT CROSS

While you're writing the function, think about **how to accomplish the function's goal**, not all the surrounding code.

ABSTRACTION BARRIER DO NOT CROSS

The abstraction barrier is the function's **specification**, or **spec** for short.

- What **inputs** are valid?
- When is it okay to call? (**precondition**)
- What **return** value is valid for a given input?
- How is the world different after? (**postcondition**)

Where to write it down

```
def pluralize(word):
    """ This text in triple quotes is
        the function's "docstring".  It
        is where we write the specification
        in English.
    """
    if word[-1] == 'y':
        return word[0:-1] + "ies"
    elif word[-1] == 's' or word[-1] == 'o':
        return word + "es"
    else:
        return word + "s"
```

Let's do it.

"Factor out" the function
Write the specification