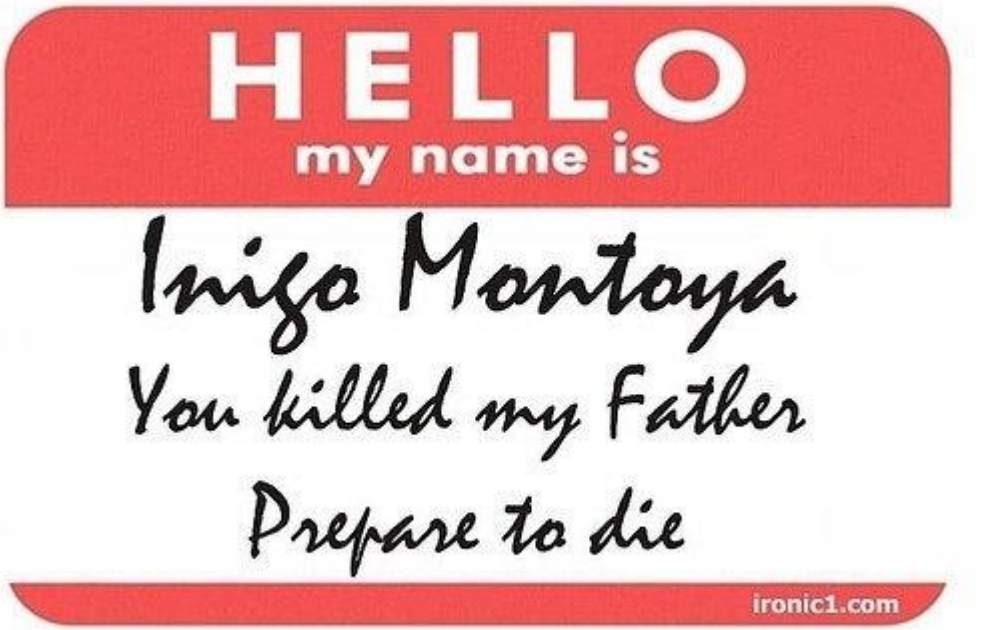
More about functions

(named parameters, optional parameters)



A normal function call

```
def score_info(score1, score2, name):
    avg = (score1 + score2) / 2.0
    print "%s: %f" % (name, avg)

score_info(100, 50, "enne")
```

Named parameters

Normally, values that you specify when calling a function are stored in parameters **in the same order** in the function declaration, e.g.

```
def score_info(score1, score2, name)
score_info(100, 50, "enne")
```

100 gets stored in score1 (first param)
50 gets stored in score2 (second param)
"enne" gets stored in name (third param)

Named parameters

Rather than just picking an order, Python lets you specify parameters by name.

```
score_info(name="enne", score1=100, score2=50)
```

...you also don't have to specify them all:

```
score_info(100, 50, name="enne")
```

Named parameters

However, if you mix named and unnamed, the named ones have to go last:

```
>>> score_info(name="enne", 100, 50)
SyntaxError: non-keyword arg after keyword arg
```

Python-ese (Parseltongue?)

named parameter => keyword arg

unnamed parameter => non-keyword arg

(arg here means "argument", which is a synonym for parameter)

Why bother naming?

One easy reason is that it can make your code easier to read. Compare these two:

```
print_scores(True, False, True, True)
```

```
print_scores(sorted=True,
             show_averages=False,
             show_individual=True,
             show_totals=True)
```

Optional function parameters

Sometimes a function takes the same value for a parameter the vast majority of the time and something "special" a little bit of the time.

To save time typing, you make some parameters optional and give them default values that get used.

Optional example

```
def calc_tip(amount, percent=0.15):
    return amount * percent
```

Optional example

```
def calc_tip(amount, percent=0.15):
    return amount * percent

# All params a normal function call
print calc_tip(100, 0.20) # 20
# Uses the default value of 0.15
print calc_tip(100) # 15
# Can do named args too...
print calc_tip(amount=200) # 30
```

Optional Errors #1

```
def my_func(optional_first=3, second):
    pass # do something here
```

SyntaxError: non-default argument follows default argument

Optional arguments must all come last. You can have more than one optional argument with a default value, but they must **all** come after the non-optional ones.

Optional Errors #2

Don't ever specify a mutable type (list, dictionary, file, etc) as a default value.

Just trust me on this one.

```
# WRONG
def function(a = []):
    pass
```

Part 3: Back to Scrabble

Write This Function:

```
def string_for_row(d, row):
    """ For a given board 'd',
    this function returns a string
    that represents all the characters
    in row 'row' appended together.
    d is a dictionary with tuple
    (column, row) keys and str values,
    row is an integer """
    pass
```

If you get done:

```
def string_for_column(d, col):
    pass

def load_word_list():
    """ Returns word list as a list
    of strings from word_list.txt """
    pass

def validate_row_string(s, list):
    pass
```