Homework 1 notes

- general
  - printing a result vs. returning a result
    - what is `return`? A function? No, it's a "keyword", like `if`, `for`, `in`, etc.
    - normally you write `return something`, not `return(something)`. The latter works, but only by accident
    - parentheses are used for other things besides function calls, such as controlling order of operations, which in this case does nothing (there's no operations to order)
    - when defining or calling a function, never leave space between the `functionname` and the opening `(`.
    - Bad: `functionname (arg1, arg2)`. Good: `functionname(arg1, arg2)`.
    - bad example works, but the good example makes it visually obvious that you're defining or calling a function, and not doing something else
    - BTW, what's the difference between defining a function and calling a function?
  - no need for semicolons at the end of each line! although adding one doesn't raise an error, it's a Matlab habit that's best to break free of
  - style: leave one blank line between the end of one function and the start of the next, easier to see where each function ends

1. Write a function called `vowelcount()` that takes a string as an argument, and returns the number of vowels in the string. Test it, e.g. `vowelcount('hEllo')`, `vowelcount('wOrld')`. It should ignore whether the vowels are captial or lowercase.

   - `s.lower()` doesn't affect `s` "in-place", it returns a new string - have to overwrite existing string: `s = s.lower()`
   - use a loop whenever possible to reduce the amount of duplicated code
     - loop over a string of vowels instead of calling `s.count()` separately for each vowel
     - use `lower()` and loop over the vowels once, regardless of case
   - if possible, do calculation/operation (in this case, `.lower()`) once outside of the loop and reuse it, instead of unnecessarily re-calculating on every iteration of the loop
   - loop over a string directly, instead of the indices of the string. Compare:

     ```
     for c in s:
         # do stuff
     ```

     ```
     for i in range(len(s)):
         c = s[i]
         # do stuff
     ```

     - this is a very common Matlab habit, and is much harder to read, and more prone to error
   - no real need to check the type of the input. If it's not a string, you'll usually get an error that gives you a hint, e.g. if you give an int instead of a string, you'll get `AttributeError: 'int' has no attribute 'lower()'` when trying to get the lowercase version
   - if you really do want to check the type, it's better to exit ASAP:

   ```
   if type(s) != str:
       print('s is not a string')
   ```

```
    return
# do stuff
return count
```

at the very top, instead of:

```
if type(s) == str:
    # do stuff
    return count
else:
    print('s is not a string')
    return
```

The second one forces all the code that actually does stuff to be indented one extra level, unnecessarily complicated

- what happens when you don't return anything? What's the returned value?
- most people iterated over the input string. What if instead you iterated over a list of all vowels?
- `.count()` is more useful than `.find()` or `.index()` in this case. All you care about is the vowel count, not *where*the vowels happen to be

2. Write a function called `metric()` that takes two numbers `x` and `y`, prints their difference and sum in a single clear message (e.g. `difference is 1, sum is 5`), and returns the difference divided by the sum. Test it, e.g. `metric(2, 3)`, `metric(10, 0.1)`. What happens if the sum is 0? What can you do to handle that case?

   - difference/sum, not sum/difference! The 1st is commonly used (goes from -1 to 1), the 2nd is unbounded and probably meaningless
   - can do `abs()` on the difference, which is fine, but this changes the output metric
   - nice, but not always necessary to assign your result to a variable (e.g. `result`) before returning it, you can leave out the assignment and save a line

3. Write a function called `multtable()` that takes a number `n` and prints out the multiplication table for integers 1 through `n`. Hint: use two `for` loops, each with a different loop variable. Bonus: check the help for `print()` to figure out how to print each row in the table in a single horizontal line.

   - what's a multiplication table?
   - do you need to check with an `if` statement if you've reached the end of the inner loop before printing a newline character?
   - what's the difference in output between `print("")` and `print()`?