Day 11

Finding Common Patterns
Announcements

- Lab #1 due Monday @ 11:59PM
- Lab #2 will be released Monday
Exercises from Last Time

1. Write a function, \texttt{sumTo}, that sums all numbers up to (and including) a given number. ie: \texttt{sumTo(3)} would sum 1, 2, and 3.

2. Write a function, \texttt{sumSquaresTo}, that sums all perfect squares up to a given number. ie: \texttt{sumSquaresTo(19)} would sum 1, 4, 9, and 16.

3. Write a function, \texttt{countChars}, that takes a string and a character, and counts how many times the character appears in that string. ie: \texttt{countChars("Hello World!", "l")} would return 3.
A Few Minor Details

JavaScript

```javascript
function printSequence(seq) {
    for(let x of seq) {
        console.log(x);
    }
}
```

The `for...of` loop in JavaScript is used to iterate over elements of a sequence. *(Just like `for...in` in Python)*
A Few Minor Details

JavaScript

```javascript
function printSequence(seq) {
  for(let x of seq) {
    console.log(x);
  }
}
```

The `for...of` loop in JavaScript is used to iterate over elements of a sequence. (Just like `for...in` in Python)

JavaScript also allows you to iterate over the `indices` of a sequence using `for...in`. (Yes...it's a bit confusing)
A Few Minor Details

JavaScript

function printSequence(seq) {
    for(let x of seq) {
        console.log(x);
    }
}

The `for...of` loop in JavaScript is used to iterate over elements of a sequence. 
*(Just like `for...in` in Python)*

function printSequence(seq) {
    for(let i in seq) {
        console.log(seq[i]);
    }
}

JavaScript also allows you to iterate over the *indices* of a sequence using `for...in`. 
*(Yes...it's a bit confusing)*
A Few Minor Details

**JavaScript**

```javascript
function printSequence(seq) {
    for(let x of seq) {
        console.log(x);
    }
}

function printSequence(seq) {
    for(let i in seq) {
        console.log(seq[i]);
    }
}
```

The `for...of` loop in JavaScript is used to iterate over elements of a sequence. *(Just like `for...in` in Python)*

JavaScript also allows you to iterate over the **indices** of a sequence using `for...in`. *(Yes...it's a bit confusing)*
<table>
<thead>
<tr>
<th>Python</th>
<th>Construct/Concept</th>
<th>JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Types</td>
<td>Boolean</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td>Number</td>
</tr>
<tr>
<td>float</td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>str</td>
<td></td>
<td></td>
</tr>
<tr>
<td>range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>Literals</td>
<td>true</td>
</tr>
<tr>
<td>False</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>3.5</td>
<td></td>
<td>3.5</td>
</tr>
<tr>
<td>4.99</td>
<td></td>
<td>4.99</td>
</tr>
<tr>
<td>&quot;Hello&quot;</td>
<td></td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>'Goodbye'</td>
<td></td>
<td>'Goodbye'</td>
</tr>
<tr>
<td>+ - * / // %</td>
<td>Operators</td>
<td>+ - * / % **</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;= == !=</td>
<td></td>
<td>&lt; &lt;= &gt; &gt;= == !=</td>
</tr>
<tr>
<td>and or not</td>
<td></td>
<td>and or not</td>
</tr>
<tr>
<td>Must be assigned before use. ie:</td>
<td>Variables</td>
<td>Must be declared before use. ie:</td>
</tr>
<tr>
<td>x = 10</td>
<td></td>
<td>let x;</td>
</tr>
<tr>
<td>y = x * 2</td>
<td></td>
<td>x = 10;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>let y = x * 2;</td>
</tr>
<tr>
<td>True</td>
<td>Keywords</td>
<td>true false</td>
</tr>
<tr>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>def</td>
<td></td>
<td>let function</td>
</tr>
<tr>
<td>return</td>
<td></td>
<td>return</td>
</tr>
<tr>
<td>if</td>
<td></td>
<td>if</td>
</tr>
<tr>
<td>elif</td>
<td></td>
<td>else for</td>
</tr>
<tr>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and or not</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Common Pattern

How would we write a function that takes a string, and returns a string with every other letter?
How would we write a function that takes a string, and returns a string with every other letter?

```javascript
function everyOther(s) {
    let newString = "";
    for(let i=0;i<s.length;i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}
```
How would we write a function that takes a string, and returns a string with every other letter?

function everyOther(s) {
    let newString = "";
    for(let i=0; i<s.length; i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}

This function looks a bit familiar...
How would we write a function that takes a string, and returns a string with every other letter?

```javascript
function everyOther(s) {
    let newString = "";
    for(let i=0; i<s.length; i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}
```

This function looks a bit familiar...

It turns out it is a pretty common pattern.

```javascript
function sumTo(n) {
    let s = 0;
    for(let i=0; i<=n; i=i+1) {
        s = s + i;
    }
    return s;
}
```
function sumTo(n) {
    let s = 0;
    for(let i=0; i<=n; i=i+1) {
        s = s + i;
    }
    return s;
}

function everyOther(s) {
    let newString = "";
    for(let i=0; i<s.length; i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}

function factorial(n) {
    let f = 1;
    for(let i=1; i<=n; i=i+1) {
        f = f * n;
    }
    return f;
}
function everyOther(s) {
    let newString = "";
    for(let i=0;i<s.length;i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}

function factorial(n) {
    let f = 1;
    for(let i=1;i<=n;i=i+1) {
        f = f * n;
    }
    return f;
}

function sumTo(n) {
    let s = 0;
    for(let i=0;i<=n;i=i+1) {
        s = s + i;
    }
    return s;
}

1. Declare a variable to accumulate into
   a. (assigned the identity of your operation)
function sumTo(n) {
    let s = 0;
    for(let i=0; i<=n; i=i+1) {
        s = s + i;
    }
    return s;
}

function everyOther(s) {
    let newString = "";
    for(let i=0; i<s.length; i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}

function factorial(n) {
    let f = 1;
    for(let i=1; i<=n; i=i+1) {
        f = f * n;
    }
    return f;
}

1. Declare a variable to accumulate into
   a. (assigned the identity of your operation)
2. Loop some number of iterations
function sumTo(n) {
    let s = 0;
    for(let i=0;i<=n;i=i+1) {
        s = s + i;
    }
    return s;
}

function everyOther(s) {
    let newString = "";
    for(let i=0;i<s.length;i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}

function factorial(n) {
    let f = 1;
    for(let i=1;i<=n;i=i+1) {
        f = f * n;
    }
    return f;
}

1. Declare a variable to accumulate into
   a. (assigned the identity of your operation)
2. Loop some number of iterations
3. Accumulate into your declared variable
A Common Pattern

function sumTo(n) {
    let s = 0;
    for(let i=0; i<=n; i=i+1) {
        s = s + i;
    }
    return s;
}

function everyOther(s) {
    let newString = "";
    for(let i=0; i<s.length; i=i+2) {
        newString = newString + s[i];
    }
    return newString;
}

function factorial(n) {
    let f = 1;
    for(let i=1; i<=n; i=i+1) {
        f = f * n;
    }
    return f;
}

1. Declare a variable to accumulate into
   a. *(assigned the identity of your operation)*
2. Loop some number of iterations
3. Accumulate into your declared variable
4. After the loop you have the accumulated value
More on Arrays and Lists

Lists (Python)

\[ x = [] \]
\[ x.append(y) \]
Adds an item, y, to the end of list x.

\[ x.pop([i]) \]
Removes the item from position i of list x and returns it. If i is not given, it removes and returns the last item.

\[ len(x) \]
Returns the length of list x.

Arrays (JavaScript)

\[ let x = []; \]
\[ x.push(y) \]
Adds an item, y, to the end of array x.

\[ x.pop() \]
Removes and returns the last item in array x.

\[ x.length \]
Returns the length of array x.
Exercise

Write a function named `implode` which takes an array (in JS) or a list (in Python) of strings and returns a single string consisting of the characters of the argument strings in order.

For example, `implode(['a','b','c'])` must return `abc`.

What is the accumulation operation? What is the identity element?