Focus on Functions . . .

CMSC 122

1. Introduction
   - How we use functions & procedures
   - Mechanical considerations . . .

Definitions

- Recall that a *function* is an object that takes elements (inputs) from a class of objects (a set) to an output, which is an object in a class of objects (a set).
- Buried in that last statement is the assumption that the function applies some kind of "transformation" to the object(s) chosen from its input(s) to its output.
- *Procedures*, on the other hand, take elements (inputs) from a class of objects but do not **return** any object as their output.

Functions are executed for their values, procedures are executed for their effects.

Examples from JavaScript

Consider the following examples:

```
 /* Define a function that computes an arithemtic sum. */
 function sum( n ) {
     var total=0;
     while( n > 0 ) {
        // accumulate total ..
     }
     return total;
}
/* define a procedure that posts the result of
 * computing a sum to a document ...
 */
function showSum() {
    var toShow = sum( 5 );
    document.getElementById(‘‘output’’).innerHTML=
    ‘‘Sum(5) is ‘’ + toShow;
}
```

How are parameters passed?

- Functions and procedures often require additional data to complete their tasks.
- These additional data are usually provided by *parameters* or arguments to the function or procedure. (Give some examples.)
- In most modern languages, functions do not modify their arguments.
- The mechanics of how this is arranged are subtle.

Some questions . . .

- How does JavaScript ensure that functions do not change the values of their arguments?
- Are there instances where the value of an argument passed to a function can be changed? (Think about the case where we pass an array as an argument.)