Functions
Functions

• A **function** is a piece of code you can use over and over again
  • Treat it like a black box
• You pass it (optional) values, it does some work, and it (optionally) returns values
• You “call it”, ”invoke it”, or “use it” by using its name and parentheses
  • The things you pass it go inside the parentheses
  • `function( input ) => output`
Using Simple Functions

- Functions for mathematical functions
  - `Math.random();`
  - `Math.floor(number);`
- Pass in arguments
- Execute in sequential order
  - flow of execution
Writing Simple Functions

- **Defining** functions
  - Creates function
  - Does not execute/run them
- Function body indicated by curly braces `{ }`.
  - In general, code inside `{ }` is called a “block” of code
- Call functions from event handlers or other functions

```javascript
function myFunction() {
    alert('assigning a value to x');
    x = 24;
    alert('x is ' + x);
}
```
Function function-name() {
    statement;
    statement;
    ...
    statement;
}

Format of a function definition
Format of a Function Definition with Parameters

function function-name(list-of-params) {
  statement;
  statement;
  ...
  statement;
}

function myFunction(myNum) {
    alert('Value of myNum is' + myNum);
}

....
<input type="button" value="Click for 42!"
onclick="myFunction(42);"/>
Parameters are Variables

• When you pass values into functions as parameters, they get assigned to the variable names declared in the definition line of the function.
• For example, when you call `myFunction(0.2)`
  The myNum variable is assigned (points to) the value 0.2
• When the code in the function refers to the myNum variable, it evaluates to the number 0.2
• So, when you call `myFunction(0.2)` and the alert function calls uses the expression
  'The value for myNum is' + myNum
  it's the same as if it called alert('The value for myNum is 0.2');
Passing variables to a functions

- If you pass a variable to a function, the function gets the value that the variable is pointing at.

```javascript
x = 4;
x = Math.floor(x);
alert('The value of x is: ' + x);
```
Using Functions that Return Values

```javascript
x = Math.random();
alert('Random number is' + x);
alert(Math.floor(15.67));
```
Return Statements

• The return statement is used to return a value from a function.
• Examples:
  return 3.14159;

  x = 6;
  return x;
function function-name(list-of-params) {
    statement;
    statement;
    ...
    statement;
    return value;
}
<script>
function area(radius) {
  return 3.14 * Math.pow(radius, 2);
}

function circumference(diameter) {
  return 3.14 * diameter;
}

function circleStuff() {
  document.getElementById('outputDiv').innerHTML = 'The area of a circle with radius 2 is ' + area(2) + ' and its circumference is ' + circumference(4);
}
</script>

<input type="button" value="Circle calculations" onclick="circleStuff();">
Composing Functions

• You can use the output (return value) of one function as the input (parameter) to another function.

```javascript
alert(Math.floor(15.67));
```

• In this example, the Math.floor(15.67) function executes first (things inside parentheses execute before things outside parentheses)
• The Math.floor function returns a number, which is then given to the alert function as a parameter.
Functions with Local Variables

```javascript
function area(radius) {
    var a = 3.14 * Math.pow(radius,2);
    return a;
}

function circumference(diameter) {
    var c = 3.14 * diameter;
    return c;
}
```
Variables in a Function are Local

• Variables declared using the `var x, y, z;` syntax in a function are **private** meaning they can't be accessed or changed by other functions/code.
  – Includes the parameters

• Allows each function has **its own** variables
  – Even when the names are the same

• Allows you to write functions independently without worrying about using the same name
Different Variables - Same Name

function area(radius) {
    var a = 3.14 * Math.pow(radius,2);
    return a;
}

function circumference(radius) {
    var a = 3.14 * 2 * radius;
    return a;
}