PROG8020

Week 4a

Keeping it neat: Functions and JavaScript Source Files

Functions

Built-in Functions

- Built-in functions are like subprograms
- They contain one or more parameters (a value to be sent to the function)
- They return (send back to the program) at least one value
- When a built-in function is called, the function name is assigned a value (of the type specified for that function)
- A built-in function is called by using the function name anywhere in the program that a constant of its type is allowed

Some JavaScript global functions:

Function	Description
isFinite()	determines whether a value is a finite, legal number
isNan()	determines whether a value is an illegal number
parseFloat()	returns a floating point number
parseInt()	returns an integer
String()	converts an object's value to a string

User-Defined Functions

- Built-in functions return a single value to the program, such as:

```
age = parseInt(34.56);
```

- **age** will have the value `34` after the statement is executed
- You set the value of a variable (**age**) equal to the result of calling the function when the value you send it is `34.56`
- We can create that type of function ourselves: a user-defined function

Example: The output, after clicking a button to call `clickit()`, will be `9`, the sum of `4` and `5`

```
1.  function addIt(x,y)
2.  {
3.      return x + y;
4.  }
5.  function clickIt()
6.  {
7.      document.write(addIt(4, 5));
8.  }
```

The Scope of a Variable

Global vs Local Variables

- When a variable is input, processed, or output, we say that it has been **referenced**.
- The **scope of a variable** refers to the part of the program in which a given variable can be referenced.
- Global scope: If a variable is defined outside of all functions then the variable will exist from that point until the end of the program.
- Local scope: any variable defined within a function will only exist while the function is running.

Beware the Global Variable!

```
1.    <html><head><title>Beware Global Variables</title>
2.    <script>
3.    function getAges()
4.    {
5.          var age = 0;
6.          age = parseInt(prompt("How old is your grandmother?"));
7.          pet();
8.          function pet()
9.          {
10.               age = parseInt(prompt("How old is your puppy?"));
11.               document.getElementById('puppy').innerHTML = (age + 10);
12.         }
13.         document.getElementById('granny').innerHTML = (age + 10);
14.   }
15.   </script></head>
16.   <body>
17.      <input type="button" onclick="getAges()" value="Find ages"></button><br />
18.      <h3>Granny's age in 10 years: <span id = "granny"> </span></h3>
19.      <h3>Puppy's age in 10 years: <span id = "puppy"> </span></h3>
20.   </body></html>
```

Results

If 105 is entered for your grandmother's age and 3 for your puppy's age, output would look as shown.

- The value of **age** on line 6 is 105
- It is changed on line 10 to 3
- Since **age** has global scope within the `getAges()` function, it displays this value as requested on line 11
- It retains this value even after the `pet()` function is exited
- This is what is displayed when line 13 is executed.

Using Local Variables

The problem caused by a global variable can be fixed.

Changing the line from the previous program:

```
age = parseInt(prompt("How old is your puppy?"));
```

to:

```
var age = parseInt(prompt("How old is your puppy?"));
```

This fixes the problem! It creates a new variable, albeit with the same name (**age**) which is only local to the `pet()` function.

Sending Information to a Function

Arguments and Parameters

- Functions often need to receive information from the statement that calls them. These values, sent to a function, are **arguments**.
- The values in the parentheses in a function's name are **parameters**.
- Arguments can be variables, expressions, or values.
- Parameters must be variables.
- When sending an argument to a function, it must be the same data type as the parameter that receives it.

Passing Arguments to Parameters

- The variables, **name** and **age** are the arguments that are passed to the parameters in the function `newAge()`. Note that the string variable, **name**, must be passed to the variable **person** and the numeric variable, **age**, must be passed to the variable, **num1**.

```
1. <script>
2.    var name = "Lizzy";
3.    var age = 25;
4.    newAge(name, age);
5.    function newAge(person, num1);
6.    {
7.       num1 = num1 + 100;
8.       document.write(person + ", next century you'll be " + num1 + "!");
9.    }
10. </script>
```

- The output will be:
  - `Lizzy, next century you'll be 125!`
- If line 4 was written as: `newAge(age, name);` the output would be:
  - `25, next century you'll be NaN!`

The `return` Statement

- The `return` statement allows functions to send a value back to the expression or statement that called them.
- A `return` statement can only return a single value.
- This program takes the value of **age** (sent to the function) and returns a new value which is then used in the output.

```
1. <script>
2.    var name = "Lizzy";
3.    var age = 25;
4.    agePlus = newAge(age);
5.    document.write(name + ", next century you'll be " + agePlus + "!");
5.    function newAge(num1);
6.    {
7.       return (num1 + 100);
8.    }
9. </script>
```

Passing Values

- When a variable is declared, a location in the computer's memory is identified.
- When the name of that variable is used in the program, the computer goes to that location to retrieve the value.
- If a variable named `cars` is assigned the value of 12 and a new variable, named `trucks` and is set equal to `cars`, then `trucks` has the value of 12 also.
- Now two areas in the computer have the value of 12—one referenced by `cars` and the other referenced by `trucks`.
- Anything done to `cars` to change its value will not affect the value of `trucks`.
- This is the process that happens when a variable is **passed by value**.
  - The function that received the variable makes a new copy of that variable and manipulates the new copy. That's one way to pass variables in a program.
- When a variable is **passed by reference**, the function receives the location in the computer's memory of the variable.
- The variable in the function probably has a different name but the new variable points to the same location as the original variable.
  - From that point on, any changes made to the new

Value and Reference Parameters

- **Value parameters**: Changes to their values in the function *do not* affect the value of the corresponding (argument) variables in the calling function. These parameters can only be used to import data.
- **Reference parameters**: Changes in their values *do* affect the corresponding arguments in the calling function. They can be used to import data into and export data from a function.

In JavaScript,

- primitive types are manipulated by value
  - numbers, booleans, and strings are considered primitive types in JavaScript
- reference types are manipulated by reference
  - objects are considered reference types
  - arrays and functions, which are specialized types of objects, are also reference types

Objects and Object-Oriented Concepts

The `Math` Object

- Properties and Methods

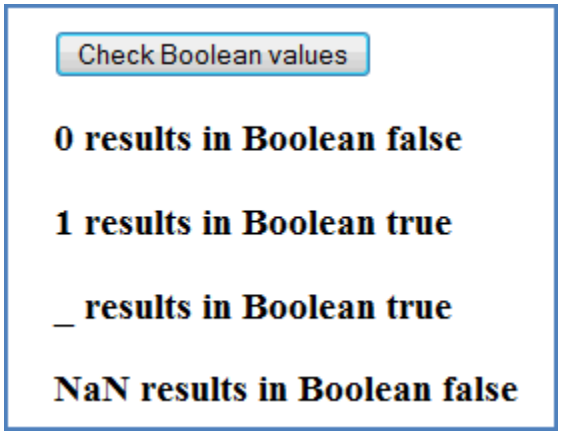
Some Properties of the Math Object	
Property	Description
LN10	returns the natural logarithm of 10 (≈ 2.302)
PI	returns π (≈ 3.14)
E	returns Euler's number (≈ 2.718)
SQRT2	returns $\sqrt{2}$ (≈ 1.414)
SQRT1_2	returns $\sqrt{1/2}$ (≈ 0.707)
Some Methods of the Math Object	
Method	Description
abs(x)	returns the absolute value of x
floor(x)	returns the integer value of x, rounded down
random()	returns a random number between 0 and 1
pow(x,y)	returns the value x^y
round(x)	rounds x to the nearest integer
sqrt(x)	returns the square root of x

More JavaScript Objects

JavaScript Object	Description
Array	allows you to store multiple values with a single variable name
Boolean	allows you to convert a non-Boolean value to a Boolean (true/false) value
Date	used to work with dates and times
Math	allows you to perform mathematical tasks
Number	used for primitive numeric values
String	allows you to manipulate and store text

The `Boolean` Object

- The program on the next slide creates four variables (**`one`**, **`two`**, **`three`**, and **`four`**) with various values.
- Four instances of the `Boolean` object are created.
- The values of each of the four variables are sent to the function that creates a new instance of the `Boolean` object.
- The object turns the value into a `Boolean` value.
  - Since variables are passed by reference, the value of the initial variable remains the same.
- If this program is coded and run the output will show that **`one`** still retains its initial value even though the `Boolean` object, **`bool1`**, now has the value of `false`.
- The output shows that a `0` is a Boolean `false`, a 1 is a Boolean `true`, an underscore is a Boolean `true`, and `NaN` is a Boolean `false`.



Boolean Objects

```
1. <script>
2. function begin()
3. {
4.     var one = 0; var two = 1; var three = "_"; var four = NaN;
5.     var bool1 = new Boolean(one);
6.     var bool2 = new Boolean(two);
7.     var bool3 = new Boolean(three);
8.     var bool4 = new Boolean(four);
9.     document.getElementById('1').innerHTML=(one+" results in Boolean "+bool1);
10.    document.getElementById('2').innerHTML=(two+" results in Boolean "+bool2);
11.    document.getElementById('3').innerHTML=(three+" results in Boolean "+bool3);
12.    document.getElementById('4').innerHTML=(four+" results in Boolean "+bool4);
13. }
14. </script></head>
15. <body>
16.    <input type ="button" onclick="begin()" value = "Check Boolean values"><br />
17.    <h3><span id = "1"> </span></h3>
18.    <h3><span id = "2"> </span></h3>
19.    <h3><span id = "3"> </span></h3>
20.    <h3><span id = "4"> </span></h3>
21. </body></html>
```

The `Date` Object

- Some methods of the Date object:

Some Methods of the Date() Object	
Method	Description
getDate()	returns the day of the month (numerical, from 1 to 31)
getDay()	returns the day of the week (from 0 to 6)
getFullYear()	returns the year in four digits
getHours()	returns the hour (from 0 to 23)
getMinutes()	returns the minutes (from 0 to 59)
getMonth()	returns the month (from 0 to 11)
getTime()	returns the number of milliseconds since midnight January 1, 1970
getTimezoneOffset()	returns the time difference, in minutes, between the local time and Greenwich Mean time (GMT)
setDate()	sets the day of the month of a Date object
setFullYear()	sets the year, using four digits, of a Date object
setHours()	sets the hour of a Date object
setMonth()	sets the month of a Date object
setTime()	sets a date and time by adding or subtracting a number of milliseconds that you specify to or from midnight, January 1, 1970
toString()	converts a Date object to a string
toTimeString()	converts the time portion of a Date object to a string

Using the `setTimeout()` Function

```
1. <html><head>
3. <title>The Day and Time</title>
4. <script>
5. function clockIt()
6. {
7.    var today = new Date(); var hour = today.getHours();var timer;
9.    var min = today.getMinutes(); var sec = today.getSeconds();
11.   min = checkTime(min);
12.   sec = checkTime(sec);
13.   document.getElementById('now').innerHTML=("Today's date: "+today);
14.   document.getElementById('clock').innerHTML=(hour+":"+min+":"+sec);
15.   timer = setTimeout('startClock()',500);
16. }
17. function checkTime(i)
18. {
19.   if (i < 10)
20.         i = "0" + i;
21.   return i;
22. }
23. </script></head>
25. <body>
26.   <input type="button" onclick="clockIt()" value="What's today?"><br />
27.   <h3><span id = "now"> </span></h3>
28.   <h3><span id = "clock"> </span></h3>
29.   </body></html>
```

Thank You!