Output, Input statements and Functions Notes

Note that item 2. **prompt** command, and item 3. **parseInt** method are good to know as they are used often in sample code found in this folder.

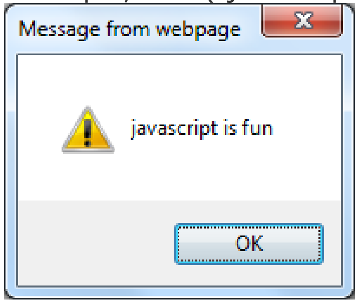
1. **OUTPUT -- alert statement.** Both the **alert** and **document.write** statements are output statements. And they both were mentioned in the document called **JavaScript Notes.** The form of the alert is:

   window.alert("text goes here but no HTML");

   The first part or the expression, "window", may be dropped and so it can be expressed as:

   alert("text goes here but no HTML");

   The alert statement displays the text inside the quotes in a pop-up alert window. For example, alert("javascript is fun"); or window.alert("javascript is fun"); appears as:

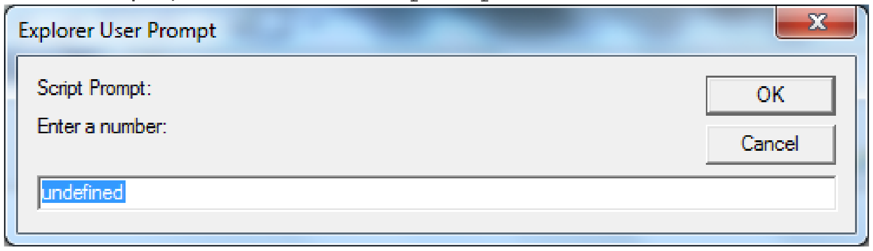


2. **INPUT** -- **prompt** statement. (`Repeated from` **Some Tips for tracing the sample code**.) The **prompt** executes a pop-up window that prompts the user for an input value (number, string, etc.). It has similar form to that of the alert or the document.write. The form is:

   ```
   prompt("text",[default value]);
   ```

   where the text in the quotation marks will appear in the pop up window. The default value is optional, thus the square brackets. If a default value is given it will appear as the user's input value in the input box.

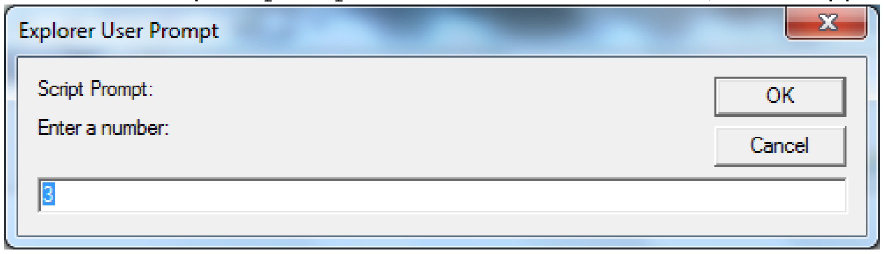
   For example, the statement: `prompt("Enter a number: ");` appears as:



   In the example above, the user will replace "undefined" with his chosen number, say 8 and then click OK or press <enter>. Furthermore the 8 may be "*captured*", and stored in variable x with the assignment statement:

   ```
   x = prompt("Enter a number: ");
   ```

Another example: `prompt("Enter a number: ", 3);` appears as:



Explorer User Prompt

Script Prompt:
Enter a number:

3

The default value 3, appears in advance in the pop-up window and will be used if the user doesn't input any other value in its place. A zero is used if the user clicks Cancel.

3. **parseInt** and **parseFloat**

If you prompt a user to input numbers from the keyboard and then follow that with a statement to add the numbers, the "+" sign in the expression actually concatenates the numbers instead of adding. (Ex: Numbers inputted 3 and 4, result in 34 instead of 7.) What happens is that the x and y have become string values and not numbers in this context. To remedy this, use code (A) below instead of the code (B). Use "parseInt" which takes care of turning the strings back into numbers to be added mathematically.

Code (A): (correct way – use parseInt)

```
x = prompt("Enter a integer: ", 0);
y = prompt("Enter another integer: ", 0);
z = parseInt(x) + parseInt(y);
document.write("The answer is: " + z);
```

If the inputs 3 and 4 are used in the above code, the output you get is "The answer is: 7"

Code (B): (logical but incorrect way)

```
x = prompt("Enter a integer: ", 0);
y = prompt("Enter another integer: ", 0);
z = x + y;
document.write("The answer is: " + z)
```

If the inputs 3 and 4 are used in the above code, the output you get is "The answer is: 34"

Note the following presents no issues with the other operations of multiply, divide and subtract, and will produce the excepted numerical results:

```
x =prompt("Enter a integer: ", 0);
y = prompt("Enter another integer: ", 0);
z =x*y;
d = x/y;
s = x-y;
document.write("The answer is: " + z + "<br />");
document.write("The answer is: " + d + "<br />");
document.write("The answer is: " + s+ "<br />");
```

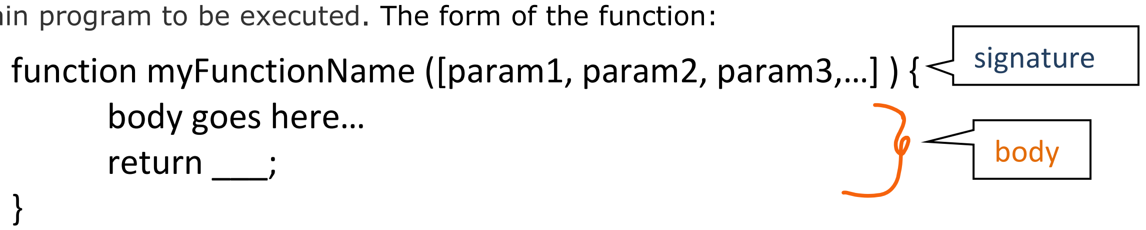
For the last set of code above with inputs of 3 and 4, the output will be:

```
The answer is: 12
The answer is: 0.75
The answer is: -1
```

4. **Functions**
   a. **Some notes:**
      i. **function** - is small program. A program (or **algorithm**) includes a precise step-by-step solution to a problem. It could, in terms of the few statements learned so far, include an input of data, instructions to process that data, and an output of the results. The main difference between a program and a function is size, and syntax.

      ii. A good programmer will want to take the programming problem at hand that requires several sub-problems to be processed and write corresponding functions, one for each sub-problem. He does this instead of writing one long program. An advantage, ideally, is that he may be able to reuse these functions in the future.

      iii. A function has a definition (= signature + body). A function needs to be "called" in the main program to be executed. The form of the function:



```
function myFunctionName ([param1, param2, param3,…] ) {
        body goes here…
        return ___;
}
```

Notes:
- note the word "function" in the first line is all lowercase.
- you make up a name for it, **myFunctionName**
- the **signature** is the first line. The **body** is inside squiggly brackets { }.
- [ ] indicate that these parameters are optional. Don't actually write []'s in the code.
- return statement – returns the value that results back to the function call and the function call expression is replaced with that value. The return statement is optional, and its inclusion depends on your desired goal.
- list function definition first and then the function call later in main program
- function only runs at a function call.

   b. **ex1:**

--a function called add2nums() , with no parameters

```
1   <script>
3   function add2nums() {
4   var x = prompt("Enter number 1 of 2: ", 0);
5   var y = prompt("Enter number 1 of 2: ", 0);
6   document.write("x = " + x + "   y = " + y);
7   document.write("<br />");
8   var z = parseInt(x) + parseInt(y);     <<    -- See #4 above for explanation of parseInt
9   document.write("their sum is " + z);
10  document.write("<br />");
11  }
12  add2nums()    // function call
13  </script>
```

For the above **ex1**:
- lines 3-11 is the *function definition*
- line 3 is the *signature*; the name of the function there is **add2nums().** It has no parameter(s).
- line 12 is the *function call*. Note it is the name of the function and with empty ()'s because there is no input value going into the function.

c. **ex2**

-- a function called **chatting** with a parameter called **name**:

```
2   <script>
3    function chatting(name) {
4   alert(“Hello there  “ +  name + “!”);
5               var x = confirm(“Are you  hungry?  ”);
6   if (x == true) alert(“go get a bite to eat!”);
7   }
8
9   N = prompt(“Please enter your name: “,  “none”);
10  chatting(N);    // function call
11
12  </script>
```

For the above **ex2**:

- lines 3 -7 are the *function definition*.
- line 3 is the *signature*; the name of the function there is **chatting(name).** The function has one parameter, **name**.
- line 10 is the *function call*. Note that the name of the function, *chatting*(N), is used. We say *"N is the value passed"* or *"N is the argument of the function"* and N is the name the user types in at the prompt statement at line 9. Alternate example: In place of lines 9 and 10 you could have: chatting("John");
- Notice the *if* statement. If statements have form *if (condition) statement(s);* If the condition is true the statements are executed. If the condition is false, the statements are not executed – they are skipped over.

d. To turn a program into a function is to essentially place the squiggly { }'s around the program statements and then precede all of that with a function signature. The signature includes the name and, if they exist, the parameters. **Try** these two items. Check your answers found at the bottom of the last page.

#1) Change this program below to a function:

```
for (x = 1;  x <= num; x++) {
   document.write("2 to the power of " + x + " = " + Math.pow(2,x) + "<br />");
}
```

Use the signature line: **function powers_of2(num)** and call the function with the following statement:

```
powers_of2(4);
```

#2) Change this program below to a function with parameters of b and h. To make the parameters b and h, you will not include line 1 and line 2 inside the function. Also you will need a return statement in the function, to return the value of A to the function call statement:

```
b = 8;  // line 1
h = 7;  // line 2
A = 1/2*b*h;
```

Use the signature line: **function funArea(b, h)** and call the function and "document.write" the result to the page:

```
x = funArea(8,7);
document.write("the area is: " + x);
```

5. **More on functions**
   a. **Other:**
      i. Nesting of functions is okay. Note **alert** is a method (which is like a function) and is nested inside of the function in ex2 above. Also for example you could list the function **chatting("John");** as a statement inside of the function **add2nums() above** if you wanted.
      ii. A function can return a value and that value goes in place of the function call. See the example below. Function interest(p, r, t) has 3 parameters p, r and t. The function is called with arguments 1000, 0.07 and 1 corresponding to the 3 parameters. Interest i is calculated and its value returned. Its value is returned and replaces the function call expression, `interest(1000, .07, 1)`. In turn it is assigned to the variable myInterest. Its is captured in the myInterest variable.

```
function interest(p, r, t) {
    i = p*r*t;
    return i;
}
myInterest = interest(1000, .07, 1);   // function call in gray
document.write(myInterest);
```

      iii. parseInt. See item #2 above. In paresInt(x), x is returned as an integer. Ex: parseInt("3"); returns 3. Note: "3" is a string and 3 is an integer.
   b. **Function samples code -** Take a look at the folder called **Function samples.** See if you can *trace* the functions -- *follow them step by step*. Use a pencil and paper and try to track it step by step as if you were the machine/interpreter.

- - - - - - - - - - - - - - - - -

answers to 5 d)

answer #1)

```
function powers_of2(num) {
   for (x = 1;  x <= num ; x++) {
      document.write("2 to the power of " + x + " = " + Math.pow(2,x) + "<br />");
   }
}
powers_of2(4);
```

answer #2)

```
function funArea(b, h){

  A = 1/2*b*h;
  return A;
}


x = funArea(8,7);
document.write("the area is: " + x);
```