Computer Science Correspondence School

discere domi

Lecture 19

CISC 131
Introduction to Programming and Problem Solving
Spring 2020
Processing Numbers

Due: **Monday, May 4, 2020, at start of class**
Points: **none**

Start of Today's Lecture

Here is on solution to the *MakingChange* problem. There are several ways to implement this function so if yours is different than mine, please study mine to get a different view of the problem.

```
function makeChangeFor(moneyAmount)
{
 var   unitNames = new Array ( "Hundred",      "Fifty",    "Twenty",
                               "Ten",          "Five",     "One",
                               "Half Dollar", "Quarter", "Dime",
                               "Nickel",       "Penny"
                             );
 var   unitValue = new Array(     10000,      5000,       2000,
                                   1000,       500,        100,
                                     50,        25,         10,
                                      5,         1
                             );
 var   units;
 var   i;

 // change parameter to pennies
 moneyAmount = Math.floor(moneyAmount * 100);

 result = '';
 for(i=0; i<unitValue.length; i++)
 {
  units = Math.floor(moneyAmount / unitValue[i]);
  if(units > 0)
  {
   result = result + units + " " + unitNames[i];

   if ( unitValue[i] >= 100 ) result = result + ' dollar bill';
   if(units > 1) { result = result + 's'; }
   result = result + '\n';
   moneyAmount = moneyAmount % unitValue[i];
  }
 }  // for

 // make Penny plural if necessary
 result = trim(result);
 i = result.indexOf("Pennys");
 if(i>0) { result = result.substring(0,i)+ 'Pennies' + result.substring(i+7); }

 return result;
}
```

Start of Today's Lecture/Assignment

A *Javascript* program can read information from the user in two ways:

- through the use of the *html input* tag or
- by using the *Javascript window.prompt* function.

We will discuss the input tag in a future lecture. For now, we will limit the user input method to the *window.prompt* Javascript statement. This is similar to how other programming languages acquire data from the user so there are some general lessons in studying this approach.

In both methods, the information entered by the user arrives at the program as a sequence of characters – a Javascript *String*. This means that even if the user enters numeric characters, the information sent to the *Javascript* program is not a number. Rather, it is a sequence of characters that can, possibly, be converted from a *String* into the internal number format.

It is actually quite useful to have all input data take the form of *Strings*. *Strings* have many built-in features that allow for their easy manipulation that numbers do not have. For example, *Strings* have a *length* property that tells you the number of characters in the *String*. Numbers do not have this property or any other property. Numbers are simply values that represent some quantity.

Arithmetic can be performed only on values that have the internal number format. We have discussed the internal format of integers – a decimal number represented in base two where the high order (leftmost) bit indicates the sign (positive or negative) of the number and the low order (rightmost) bit indicates whether the integer is odd or even. Negative integers are represented in the two's complement form. Non-integer numbers have a different, more complicated internal format and that format is beyond the scope of this class.

Programs will often need to convert *Strings* containing numeric characters into the internal format of actual numbers. This happens both when getting information from the user and also when accessing some properties of the *Document Object Model (DOM).* In this assignment you will develop some useful functions that can be used whenever *String* to number conversion is necessary.

Step 1

Create empty *html* and *Javascript* files. As you know, *Javascript* has a built-in function called *Number* that is used to convert *Strings* into internal numeric format. If the *Number* function is passed a number, it simply returns the value that was passed to it. If it is passed a *String*, it returns the numeric equivalent of the digit characters in the *String*. If the characters in the *String* do not represent a number, the *Number* function returns *NaN*. You can explicitly set the value of a variable to this special value using an assignment statement. For example:

```
result = NaN;
```

NaN is a special value that is an acronym for **n**ot **a** **n**umber. All modern programming languages have keywords such as *NaN, null, undefined, true,* and *false* that have special meaning and can be used as values for testing or assignment. The languages do not all use the same keywords and, when you learn a new language, it is important to become familiar with the keywords supported by that language. In Javascript, the keyword *NaN* differs from its other special values in that you must use a specific function in order to test whether a variable has the value *NaN*. This function is named *isNaN* and has a single formal parameter.

The *isNaN* function returns a *Boolean* value. For example, to determine whether the value of the variable *x* is *NaN*, you would do *isNaN(x).* If the function returns *true*, then the value of *x* is not a number. If the function returns *false*, then the value of *x* is a number. Doing *if (x === NaN)* will not work. You must always use the *isNaN* function to test to see if the value of a variable is a number or not a number.

Put the following statements in your program and note what is produced for each when the program is run.

```
window.alert(“ad: “ + Number("ad"));
window.alert(“6d: “ + Number("6d"));
window.alert(‘”23”: ‘ + Number("23"));
window.alert(“23: “ + Number(23));
window.alert(‘”-1.23”: ‘ + Number("-1.23"));
window.alert(“-1.23: “ + Number(-1.23));
```

The first two *alerts* should display *NaN* and the other *alerts* should display a numeric value. The Javascript *Number* function ignores leading and trailing *whitespace*[1] characters when doing the conversion. Try this:

```
window.alert(Number("   2.35   ")+1);
```

The value displayed will be 3.35. This was done by converting the *String* into a number and then adding one to the number. Internal *whitespace* characters are not ignored, however. Try this:

```
window.alert(Number("   87 41   "));
```

NaN will be displayed because of the space character between the seven and the four. Embedded *whitespace* characters such as this are treated as part of the data by the *Number* function.

Ignoring leading and trailing *whitespace* is a useful property of the *Number* function and would be even better except for one thing. Try this:

```
window.alert(Number(""));
window.alert(Number("      "));
```

Both of these will display the value zero. The *Number* function returns the numeric value *zero* if the *String* passed to it contains no characters or if the *String* passed to it contains all *whitespace* characters. In many situations, this can a very undesirable effect.

For programming purposes, the characters in a *String* represent a number only if these three criteria are all true:

1. there is at least one non-*whitespace* character in the *String* and
2. all the non-whitespace characters are consecutive and
3. the non-whitespace characters taken as a sequence represent a valid number

Most modern programming languages have a built-in function called *trim* that is associated with variables containing *Strings*. These *trim* functions are designed to take a *String* as input and produce as output the contents of the *String* after all leading and trailing *whitespace* characters have been removed. They do not remove internal whitespace. They only remove leading and

[1] Recall that *whitespace* is a common programming term used to describe a set of characters that have meaning but are not visible such as the space character or the tab character. Many languages represent these characters by using an escape character sequence – a character sequence whose first character is special. This special character is often the backslash character. For example, to represent a tab, the programmer would make a *String* containing: \t (inside quotes or apostrophes). The common set of whitespace characters is: *space* which does not need an escape character sequence to represent it, *tab* which is represented by \t, *new line* which is represented by \n, the *return* character which is represented by \r, and the *form feed* character which is represented by \f.

trailing whitespace. As you have no doubt discovered, *trim* is a very useful function and is often used when processing information from the user and it can be used to help determine if user input is numeric.

Step 2
Write the following function. The formal parameter value could be anything – number, *String*, or anything else. Do not use the *typeof* operator. It has not been introduced in class and it is not necessary for the logic.

```
function toNumber(data)
```

The function will return the numeric equivalent of the parameter if the parameter can be converted into a number using the three criteria shown above. If it cannot be so converted, the function returns the special *NaN* value. Test the function thoroughly.

Step 3
Write the following function. The formal parameter value could be anything. The function returns *Boolean true* if the parameter is a number or can be converted into a number. In all other cases, it returns *false*..

```
function isNumeric(data)
```

Remember that comparing a variable to *NaN* using the equality operator will not work. In order to determine if a variable contains a number, you must use the *isNaN* function. Passing a *String*, or most other things, to *isNaN* causes it to return *true*. Write the *isNumeric* function in such a way that it contains only a single statement: the *return* statement. You don't need *if* statements; you can use *Boolean* operators. Test your function thoroughly.

Step 4
Now write the function:

```
function isInteger(data)
```

This function returns *Boolean true* only when the passed value is an integer (either an internal numeric integer or a *String* containing characters that represent an integer value). In order to be an integer, the passed value must first be a number. The function must consist of a single Javascript statement: the *return* statement. Test your function thoroughly.

Step 5
Show that your functions work by running this test data :

```
test1("");
test1(null);
test1("hello");
test1("3");
test1("3.5");
test1(new Date());
test1(-.99);
test1(112);
test1(-112);
```

where *test1* is this function:

```
function test1(data)
{
 window.alert("value is: " + data +
             " isNumeric: " + isNumeric(data) +
             " isInteger: " + isInteger(data));
}
```

Test everything thoroughly.

Step 6
Now write a function that will be passed a *String*. The function will use *window.alert* to display one of the following based on the data in the parameter:

- *It's not a number.*.
- *It's a number.*
- *It's an integer*.

You will need to use *if* statements in this function. Call the function from inside a loop in the *window.onload* function. The loop will contain a *window.prompt* statement that will ask the user to enter something. What they enter is then passed to the function you just wrote. Now you can test your functions interactively.

Test thoroughly and verify the results. Be sure to enter some data that has leading, trailing, or embedded *whitespace* characters.