Strings in JavaScript

Jerry Cain
CS 106AX
October 10, 2022

Using Methods in the `String` Class

- JavaScript defines many useful methods to operate on strings. Before trying to use those methods individually, it is important to understand how those methods work at a more general level.
- Because strings are objects, JavaScript uses the receiver syntax to invoke string methods. So, if `str` is a string, you invoke the `name` method using `str.name(arguments)`.
- None of the methods in JavaScript’s `String` class change the value of the string used as the receiver. These methods return a new string on which the desired changes have been performed.
- Classes that prohibit clients from changing an object’s state are said to be immutable. Immutable types have many advantages and play an important role in programming.

Selecting Characters from a String

- Conceptually, a string is an ordered collection of characters.
- In JavaScript, the character positions in a string are identified by an `index` that begins at 0 and extends up to one less than the length of the string. For example, the characters in the string “hello, world” are arranged like this:

```
  0 1 2 3 4 5 6
h e l l o , w o r l d
```

- You can obtain the number of characters by checking the `length` property, as in `str.length`.
- You can select an individual character by calling `charAt(k)`, where `k` is the index of the desired character. The expression `str.charAt(0);` returns the one-character string “h” that appears at index 0.

Concatenation

- One of the most useful operations available for strings is concatenation, which consists of combining two strings end to end with no intervening characters.
- As you know from earlier in the quarter, concatenation is built into JavaScript in the form of the `+` operator.
- It is also important to recall that JavaScript interprets the `+` operator as concatenation if one or both operands are strings. If both operands are numbers, the `+` operator signifies addition.

Extracting Substrings

- The `substring` method makes it possible to extract a piece of a larger string by providing index numbers that determine the extent of the substring.
- The general form of the `substring` call is

```
str.substring(p1, p2);
```

where `p1` is the first index position in the desired substring and `p2` is the index position immediately following the last position in the substring.
- As an example, if you wanted to select the substring “ell” from a string variable `str` containing “hello, world” you would make the following call:

```
str.substring(1, 4);
```
Comparing Strings

- JavaScript allows you to call the standard relational operators to compare the values of two strings in a natural way. For example, if \( s_1 \) and \( s_2 \) are strings, the expression \( s_1 === s_2 \) is \( true \) if the strings \( s_1 \) and \( s_2 \) contain the same characters.

- String comparisons involving the operators \( < \), \( <= \), \( > \), and \( >= \) are implemented in a fashion like traditional alphabetic ordering: if the first characters match, the comparison operator checks the second characters, and so on.

- Characters are compared numerically using their Unicode values. For example, \( \text{"cat"} > \text{"CAT"} \) because the character code for \( \text{"c"} \) (99) is greater than the code for \( \text{"C"} \) (67). This style of comparison is called lexicographic ordering.

- We'll revisit character encodings and Unicode next week.

Searching in a String

- The \( \text{indexOf} \) method takes a string and returns the index within the receiver at which the first instance of that string begins. If the string is not found, \( \text{indexOf} \) returns \( -1 \). For example, if \( \text{str} \) contains the string \( \text{"hello, world"} \):

  - \( \text{str.indexOf("h"}) \) returns \( 0 \)
  - \( \text{str.indexOf("o")} \) returns \( 4 \)
  - \( \text{str.indexOf("ell")} \) returns \( 1 \)
  - \( \text{str.indexOf("x")} \) returns \( -1 \)

- The \( \text{indexOf} \) method takes an optional second argument that indicates the starting position for the search. Thus:

  - \( \text{str.indexOf("o", 5)} \) returns \( 8 \)

- The \( \text{lastIndexOf} \) method works similarly except that it searches backward from the end of the receiving string.

Other Methods in the String Class

- \( \text{String.fromCharCode(code)} \) returns the one-character string whose Unicode value is \( code \).
- \( \text{charCodeAt(index)} \) returns the Unicode value of the character at the specified index.
- \( \text{startsWith(prefix)} \) returns \( true \) if this string starts with \( prefix \).
- \( \text{endsWith(suffix)} \) returns \( true \) if this string ends with \( suffix \).
- \( \text{trim()} \) returns a copy of this string with leading and trailing spaces removed.
- \( \text{toLowerCase()} \) returns a copy of this string converted to lower case.
- \( \text{toUpperCase()} \) returns a copy of this string converted to upper case.

Simple String Idioms

When you work with strings, there are two idiomatic patterns that are particularly important:

1. Iterating through the characters in a string.

   ```javascript
   let result = "";
   for (let i = 0; i < str.length; i++) {
     let ch = str.charAt(i);
     ... code to process each character in turn ...
     result += ch;
   }
   ```

2. Growing a new string character by character.

   ```javascript
   let result = "";
   for (whatever limits are appropriate to the application) {
     let ch = ...;  // code to determine the next character to be added ...
     result += ch;
   }
   ```

Reversing a String

Let’s review some String methods before continuing:

- \( \text{"AEIOUaeiou".length} \) 10
- \( \text{"ABCDERG".charAt(6)} \) 1
- \( \text{"Harry Potter".indexOf("a")} \) 1
- \( \text{"Harry Potter".indexOf("a", 6)} \) -1
- \( \text{"Harry Potter".lastIndexOf("rr")} \) 2
- \( \text{"bumfuzzle".substring(3, 7)} \) "fuzz"
- \( \text{"cabotage".substring(1, 1)} \) "c"
- \( \text{"agelast".substring(3)} \) "last"
Generating Acronyms

- An acronym is a word formed by taking the first letter of each word in a sequence, as in "North American Free Trade Agreement" → "NAFTA"
- "not in my back yard" → "nimby"
- "self-contained underwater breathing apparatus" → "scuba"

The text describes and implements two versions of a function acronym() that generates an acronym for str:
- The first version searches for spaces in the string and includes them as markers to separate words.
- The second version looks at every character and keeps track of whether the algorithm is scanning a word formed by sequential letters. This version correctly handles strings that have leading, trailing, or multiple spaces.

Translating Pig Latin to English

Section 7.4 works through the design and implementation of a program to convert a sentence from English to Pig Latin. In this dialect, the Pig Latin version of a word is formed by applying the following rules:

1. If the word begins with a consonant, the wordToPigLatin() function moves the initial consonant string to the end of the word and then adds the suffix ay, as follows:

   - scrum → amscru

2. If the word begins with a vowel, wordToPigLatin() generates the Pig Latin version simply by adding the suffix ay, like this:

   - apple → appleway

3. If the word contains no vowels at all, wordToPigLatin() returns the original word unchanged.

Pseudocode for the Pig Latin Program

```plaintext
function wordToPigLatin(word)
    Initialize variables start and result.
    if (word contains vowels)
        else (word contains consonants)
            Append "ay" to the end of the word.
            Return the result.
    else (word contains no vowels)
        Return the original word unchanged.
```

```
function wordToPigLatin(result)
    Declare a variable called=result to hold the growing string.
    for (each character position in word) 
        if (the current character is a vowel)
            Append the translated word to result variable.
        else
            Append the character to the result variable.
    Append the translated word to the result variable.
    Return the result variable.
```

```
function findVowels(astr)
    Declare a variable called=result to hold the growing string.
    for (each character position in astr) 
        if (the current character is a vowel)
            Append the character to the result variable.
        else
            Append the character to the result variable.
    Append the translated word to the result variable.
    Return the result variable.
```

Translating Pig Latin to English

"stout plunder lover"

- inWord is true if and only if we’re in a word, and start is the index of the first character of the word we’re currently in (or -1 if we’re not in a word).
- inWord is now true and start is set equal to 0. We set the value of i to start at the same time inWord is transitioning from false to true, so we can remember where the current word of interest begins.
- This is an interesting transition, since the current word we’re in is just now ending. We can isolate the word by calling astr.substring(start, i), where astr is assumed to be the entire sentence or fragment to be translated.
- If we’re transitioning from a true to false case, we move that "stout" word. And now, astr.substring(start, i) produces "stout".
- If the word contains no vowels at all, wordToPigLatin() returns the original word unchanged.

```plaintext
function findVowels(astr)
    Declare a variable called=result to hold the growing string.
    for (each character position in astr) 
        if (the current character is a vowel)
            Append the character to the result variable.
        else
            Append the character to the result variable.
    Append the translated word to the result variable.
    Return the result variable.
```

```
function wordToPigLatin(word)
    Initialize variables start and result.
    if (word contains vowels)
        else (word contains consonants)
            Append "ay" to the end of the word.
            Return the result.
    else (word contains no vowels)
        Return the original word unchanged.
```

```
function wordToPigLatin(result)
    Declare a variable called=result to hold the growing string.
    for (each character position in word) 
        if (the current character is a vowel)
            Append the translated word to result variable.
        else
            Append the character to the result variable.
    Append the translated word to the result variable.
    Return the result variable.
```

```
function findVowels(astr)
    Declare a variable called=result to hold the growing string.
    for (each character position in astr) 
        if (the current character is a vowel)
            Append the character to the result variable.
        else
            Append the character to the result variable.
    Append the translated word to the result variable.
    Return the result variable.
```
Simulating the PigLatin Program

```javascript
function toPigLatin(word) {
  let head = word.substring(0);,
  let tail = word.substring(1);
  if (head == "a") return head + word;
  return head + tail + "ay";
}

let result = toPigLatin("this is pig latin");
```

The GLabel Class

You can display a string in the graphics window using the GLabel class, as illustrated by the following function that displays the string "hello, world" on the graphics window:

```javascript
function HelloWorld() {
  let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
  let label = GLabel("hello, world", 100, 75);
  label.setFont("36px Helvetica");
  label.setColor("Red");
  gw.add(label);
}
```

Operations on the GLabel Class

Function to create a GLabel

```javascript
GLabel(text, x, y)
```

Creates a label containing the specified text that begins at the point (x, y).

Methods specific to the GLabel class

```javascript
label.setFont(font)
```

Sets the font used to display the label as specified by the font string.

The font is specified as a CSS fragment, the details of which are described in the JavaScript textbook, pp. 129-131.

Examples of legal font strings:

- "italic 36px Helvetica"
- "24px 'Times New Roman'"
- "bold 14px 'Courier', 'Courier New', Monaco"
- "oblique bold 44px 'Lucida Blackletter', serif"

The Geometry of the GLabel Class

- The GLabel class relies on a set of geometrical concepts that are derived from classical typesetting:
  - The baseline is the imaginary line on which the characters rest.
  - The origin is the point on the baseline at which the label begins.
  - The height of the font is the distance between successive baselines.
  - The ascent is the distance characters rise above the baseline.
  - The descent is the distance characters drop below the baseline.
- You can use the getHeight, getAscent, and getDescent methods to determine the corresponding property of the font.
- You can use the getWidth method to determine the width of the entire label, which depends on both the font and the text.

The End