TASK

String Handling

Visit our website

Introduction

WELCOME TO THE STRING HANDLING TASK!

One of the most crucial concepts to grasp when it comes to programming is string handling. For the upcoming tasks you will need to be very comfortable with string handling, so it is important to refresh and consolidate your knowledge. In this task, you will briefly recap some key points and then learn to create more advanced programs with strings which use more functions and programming techniques.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

INDEXING STRINGS

You can think of the string 'Hello world!' as a list and each character in the string as an item with a corresponding index.

```
'   H   e   l   l   o       w   o   r   l   d   !   '
    0   1   2   3   4   5   6   7   8   9   10  11
```

The space and the exclamation point are included in the character count, so 'Hello world!' is 12 characters long, from 'H' at index 0 to '!' at index 11.

```
const myString = "Hello, world!";

console.log(myString[0]); // H
console.log(myString[1]); // e
console.log(myString[2]); // l
console.log(myString[3]); // l
console.log(myString[4]); // o
```

Remember that if you specify an index, you'll get the character at that position in the string. You can also use **myString.slice(x, y)** to slice strings by specifying a range from one index to another, the starting index is included and the ending index is not.

Note: slicing a string does not modify the original string. You can capture a slice from one variable in a separate variable. Try typing the following into the interactive shell:

```
const myString = "Hello, world!";
slicedString = myString.slice(0, 5);

console.log(slicedString); // Hello
```

By slicing and storing the resulting substring in another variable, you can have both the whole string and the substring handy for quick, easy access.

STRING METHODS

Once you understand strings and their indexing, the next step is to master using some of the common string methods. These are built-in modules of code that perform certain operations on strings. These methods are useful as they save time since there is no need to write the code over and over again to perform certain operations. The most common of string methods are (where ***s*** is the variable that contains the string we are working with):

- **s.toLowerCase()** and **s.toUpperCase()** — convert a string to either uppercase or lowercase.
- **s.trim()** — removes any whitespaces from the beginning or end of a string.
- **s.indexOf('text')** — searches for a specific text and returns its position in the string you are searching. If the string isn't found, -1 is returned.
- **s.replace('oldText', 'newText')** — replaces the first occurrence of 'oldText' with 'newText'.
- **s.split('word')** — breaks down a string into a list of smaller pieces. The string is separated based on what is called a *delimiter*. This is a string or char value that is passed to the method. If no value is given it will automatically split the string using whitespace as the delimiter and create a list of the characters.
- To add an item to the end of your list, you use the ***append()*** method. For example, ***list.push(item)*** adds the single item within the brackets to the end of a list. You will learn more about lists later on in this course.
- **string_list.join("")** — takes a list of strings or characters and joins them to create one string. You can specify what character, if any, you would like to join the list elements with. For example, *["apples", "bananas", "carrots"].join("@")* would output: *"apples@bananas@carrots"*.

Examine **example.js** to learn how each of these methods can be used.

ESCAPE CHARACTER

Javascript uses the backslash (\) as an escape character. The backslash (\) is used as a marker character to tell the compiler/interpreter that the next character has some special meaning. The backslash, together with certain other characters, is known as an escape sequence.

Some useful escape sequences are listed below:

- **\n** - Newline

- **\t** - Tab

The escape character can also be used if you need to include quotation marks within a string. You can put a backslash (\) in front of a quotation mark so that it doesn't terminate the string. Put a backslash in front of another backslash to include a backslash in a string.

STRING BUILDING

String building (or formatting) is the process of dynamically changing the value of strings based on specific variables. This is a very common practice in string manipulation.

In JavaScript, there are two ways to format a string:

- Using backticks
- Appending strings

To illustrate string formatting, we will start with the simplest example: using backticks.

```
let costPrice = 23.45;
console.log(`Cost of item: ${costPrice}. Thank you for shopping!`);
```

Note the use of ` as opposed to ‘ or “. This is a special kind of character that allows automatic string formatting. This is done with using **${variable_name}**. In the background, JavaScript will read the expression within the curly braces, convert that to a string, and insert it into the string.

Now that we have an idea of what string formatting is, let's take a look at the other method: appending strings.

```
let item = "Shoes";
let costPrice = 123.45;

console.log("The " + item + " costs R" + costPrice);
```

This form of formatting strings is not considered the neatest form of string formatting. Note that this uses the ‘+’ operator, which you normally see with numbers, rather than strings. In JavaScript, this operator is a bit smarter and recognises that, if strings are used for the ‘+’ operator, it must just append the strings. If it notices that you are using

the '+' operator with a string and a number, it will convert that number to a string, and append it.

String **concatenation**, while not the neatest, syntactically speaking, is still very valuable in JavaScript. This is for those situations where you don't yet know how long your string is going to be, and want to base it on the length of, for example, an array.

```
let items = [
    "Shoes",
    "Shirts",
    "Socks"
];

let prices = [
    123.45,
    100.00,
    10.00
];

let inventoryDisp = "";
for(let i = 0; i < 3; i++) {
    let item = items[i];
    let price = prices[i];
    inventoryDisp = inventoryDisp + `Item ${i}: ${item} ${price}\n`;
}
console.log(inventoryDisp);
```

Over here, we see that we just concatenate a new item for each element in the lists. This way, you can make the print longer by just adding more items to the list. There is one small problem in this code: there will always be a **"\n"** character at the end of the print. This is not very neat and just looks unprofessional. We could fix this by placing the **"\n"** character at the beginning of the string, but this leaves us with a leading **"\n"** instead, which doesn't help us either.

We can be even smarter here by using some of the string methods we have learnt earlier. Let's take a look at something a bit more difficult:

```
let itemsCSV = "Shoes, Shirts, Socks";
let prices = [
```

```
    123.45,
    100.00,
    10.00
];

let items = itemsCSV.split(', ');

let invDispItems = [];

for (let i = 0; i < items.length; i++){
    invDispItems.push(`Item ${i}: ${items[i]} ${prices[i]}`);
}
console.log(invDispItems.join('\n'));
```

In this code, we have one extra problem: our items are arranged as a single CSV string, rather than a **list** of strings. We can easily fix this with the **string.split()** method, and this gives us our array of strings as normal. We create a new array to store each line of string to be displayed. Each line is added using the **push()** method. To convert the list of strings to a single string, you simply call the **join()** method, and join using a **"\n"** character in order to ensure that each item in the list appears on a new line. This way, there is no leading or trailing **"\n"**.

A note from our coding mentor

Jared

Hey again, have you ever wondered where a string variable gets its name from? According to an **article on StackOverflow**:

"The 1971 OED (p. 3097) quotes an 1891 Century Dictionary on a source in the Milwaukee Sentinel of 11 Jan. 1898 (section 3, p. 1) to the effect that this is a compositor's term. Printers would paste up the text that they had generated in a long strip of characters."

Instructions

First, read **example.js**. Open it using Visual Studio Code.

- **example.js** should help you understand some simple JavaScript. Every task will have example code to help you get started. Make sure you read all of **example.js** and try your best to understand.

- You may run **example.js** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with JavaScript.

Compulsory Task 1

Follow these steps:

- Open up **alternative.js**. This program uses a set of test strings and calls a method **alternativeString** on each string.

- This method inputs a string and outputs a modified string with each alternate character as an uppercase character and each other alternate character a lowercase character.

- For example, the string “Hello World” would become “HeLlO WoRlD”

- Fill in the code for the **alternativeString** method. Do not modify any other aspect of the script.

Compulsory Task 2

Follow these steps:

- Open up **separation.js**. This program uses a set of test strings and calls a method **separationString** on each string.

- This method should take a string as input, and output a new string with each word on a new line.

- Fill in the code for **separationString**. Do not modify any other aspect of the script.

Compulsory Task 3

Follow these steps:

- Open up **disappear.js**. This program uses a set of test strings and calls a method **disappearString** on each string.

- This method should take as input two strings: myString and toErase. It will look at each letter in toErase, and remove the first instance of that letter from myString.

- For example:
    - the quick brown fox jumps over the lazy dog.

After stripping **a**,**e**,**i**,**o**,**u** becomes:

    - th qck brwn fox jumps over the lzy dog.

Optional Bonus Task

Follow these steps:

- Open up **optional_task.js**. This program uses a set of test strings and calls a method **isPalindrome** on each string.

- This method should determine whether a string is a palindrome.

- A palindrome is a string which reads the same backwards and forward. Some examples of palindromes are: racecar, dad, level and noon.

- Fill in the **isPalindrome** method.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.