Learning Objectives: String Basics

- **Identify the three properties of strings**
- **Recognize that strings are immutable**
- **Determine if a string is present in another string**
- **Print a string from the start index to end index**
- **Utilize escape characters to add special characters to a string**

String Properties

String Length

We have already seen strings in the “Fundamentals”section. We are going to dig a little deeper with this data type. All strings have the following characteristics:

1. **Characters** - Strings are made up of characters between quotation marks (previously covered in the “Fundamentals” section).
2. **Length** - Each string has a length (total number of characters).
3. **Index** - Each character in a string has a position, called an index.

To calculate the length of a string, use the `length()` method. This method will return an integer that is the sum of all of the characters between the quotation marks.

```
String myString = "Hello";
int lenString = myString.length();
System.out.println(lenString);
```

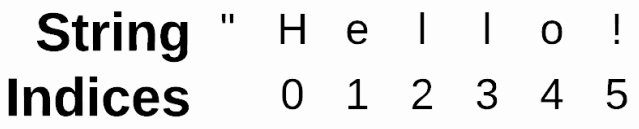
challenge

What happens if you:

- Change `myString` to `"Hello world!"`?
- Change `myString` to `""`?
- Change `myString` to `"-1"`?

String Index

Previously in the **arrays** module, we learned that an array has elements that reside in each of its position or **index**. A string too has indices that correspond to the position where each of its character resides. Like array indices, string indices also start at 0.

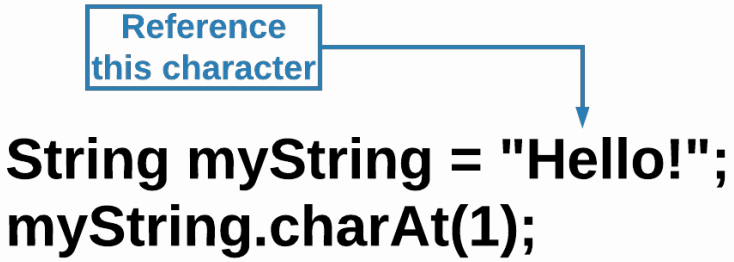


.guides/img/StringIndex

▼ **Strings & Quotation Marks**

Quotation marks are required to declare the value of a string. However, quotation marks are not a part of the string itself. That is why quotation marks are not counted with the `length()` function and why they do not have an index.

To reference a character, use the string name followed by the `charAt()` method. Within parentheses `()`, provide the index number of the character you want the system to return.



.guides/img/StringCharAt

```
String myString = "Hello!";
char character = myString.charAt(1);
System.out.println(character);
```

challenge

What happens if you:

- Change `char character` to `myString.charAt(myString.length())`?
- Change `char character` to `myString.charAt(myString.length()-1)`?
- Change `char character` to `myString.charAt(-1)`?

Immutability

Immutability

You now know how to reference each character of a string. What do you think the code below will do?

```
String myString = "House";
myString.charAt(0) = "M";
System.out.println(myString);
```

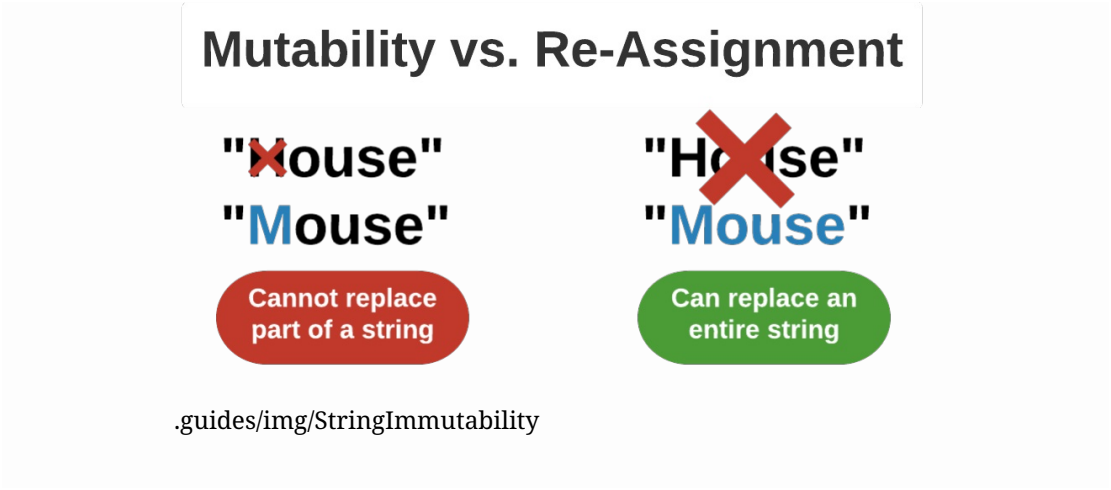
If you thought the code above would print `Mouse`, that would be a logical guess. However, you see an error. Unlike arrays where the characters can be manipulated, strings are immutable. That means you cannot change their value.

Yes, but...

The code below works just fine. Isn't this an example of changing the value of a string?

```
String myString = "House";
myString = "Mouse";
System.out.println(myString);
```

In Java, strings cannot be manipulated, thus they are considered immutable. However, strings like other data types can be re-assigned. The first example on this page is about mutability. That is, changing just a part of a whole. The second example is about the assignment operator. Re-assignment replaces the entire value of a variable with a new value. So, you can replace an entire string (re-assignment), but you cannot change part of a string (immutability).



.guides/img/StringImmutability

Contains

The contains() Method

The `contains()` method tells you if a character or a string is present in another string. `contains()` returns a boolean value, either `true` if the string is present or `false` if it is not.

```
String myString = "The brown dog jumps over the lazy fox.";

System.out.println(myString.contains("dog"));
```

challenge

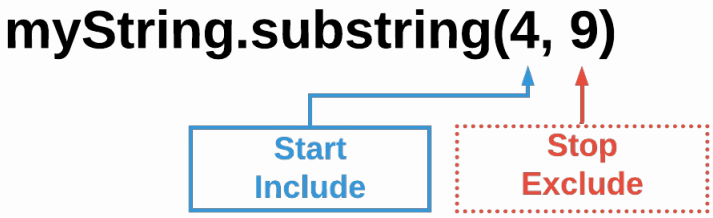
What happens if you:

- Change the `print` statement to be `System.out.println(myString.contains("cat"));`?
- Change the `print` statement to be `System.out.println(myString.contains("Dog"));`?
- Change the `print` statement to be `System.out.println(myString.contains(" "));`?
- Change the `print` statement to be `System.out.println(myString.contains(myString));`?

Substring

The substring() Method

The `substring()` method returns a portion of the string. Within parentheses `()`, provide the index at which you want the string to start followed by a comma followed by the index at which you want the string to stop. **Note** that the *start* index is **inclusive** but the *stop* index is **not inclusive**. The `substring()` method does not modify the original string. Instead, it returns a partial copy of the original string.



.guides/img/StringSubstring

```
String myString = "The brown dog jumps over the lazy fox.";
String mySlice = myString.substring(4, 9);

System.out.println(mySlice);
```

challenge

What happens if you:

- Change `mySlice` to be `myString.substring(1, 2)`?
- Change `mySlice` to be `myString.substring(0, myString.length())`?
- Change `mySlice` to be `myString.substring(1, 1)`?
- Change `mySlice` to be `myString.substring(2)`?

▼ **`substring()` default stop index**

If only one index number is present, such as `myString.substring(2)`, Java will force the start index to that index number and the end index number will be the index of the last character of the string + 1. Essentially, `myString.substring(2)` will become `myString.substring(2, myString.length())` which is also the same as `myString.substring(2, 38)`.

Escape Characters

Escape Characters

An escape character is a character that has a different interpretation than what you see in a string. Escape characters always start with a backslash (`\`). The most common escape character is the newline character (`\n`) which causes Java to print on the next line.

```
String myString = "Hello\nworld";
System.out.println(myString);
```

Escape Character	Description	Example
\\	Prints a backslash	`System.out.println("\\")`
\'	Prints a single quote	`System.out.println("\'")`
\"	Prints a double quote	`System.out.println("\"")`
\t	Prints a tab (spacing)	`System.out.println("Hello\tworld")`

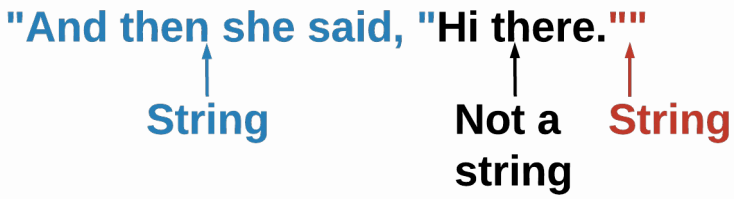
challenge

What happens if you:

- Use `\n\n` instead of `\n` in the original code?
- Replace `\n\n` in your current code with `\t`?
- Replace `\t` in your current code with `\"`?

Quotes Inside Quotes

Imagine that you have this small bit of dialog, `And then she said, "Hi there."` and want to store it as a string. Typing `"And then she said, "Hi there.""` would cause an error.



.guides/img/StringQuotes

When you use a `"` to start a string, Java looks for the next `"` to end it. To avoid syntax errors, you can use a double quote to start your string, single quotes for the inner quote, and end the string with a double quote.

```
String myString = "And then she said, 'Hi there.'";
System.out.println(myString);
```

challenge

What happens if you:

- Replace single quotes (`'`) with double quotes (`"`) and double quotes(`"`) with single quotes (`'`) in the original code?
- Change all quotes to double quotes (`"`)?
- Use the escape character `\"` for the inner quotation marks that surround `Hi there.` (e.g. `"And then she said, \"Hi there.\""`)?