When using the `===` operator, equal strings are not equal, because the `===` operator expects equality in both type and value.

Example

```
var x = "John";
var y = new String("John");

// (x === y) is false because x and y have different types (string and object)
```

```
<!DOCTYPE html>
<html>
<body>

<h2>Never Create Strings as objects.</h2>
<p>Strings and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = "John";             // x is a string
var y = new String("John");  // y is an object
document.getElementById("demo").innerHTML = (x===y);
</script>

</body>
</html>
```

Never Create Strings as objects.

Strings and objects cannot be safely compared.

false

Or even worse. Objects cannot be compared:

Example

```
var x = new String("John");
var y = new String("John");

// (x == y) is false because x and y are different objects
```

Try it Yourself »

Example

```
var x = new String("John");
var y = new String("John");

// (x === y) is false because x and y are different objects
```

Finding a String in a String

The `indexOf()` method returns the index of (the position of) the `first` occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method returns the position of the first occurrence of a
specified text:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The indexOf() method returns the position of the first occurrence of a specified text:

7

The `lastIndexOf()` method returns the index of the **last** occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Try it Yourself »

Both `indexOf()`, and `lastIndexOf()` return -1 if the text is not found.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The lastIndexOf() method returns the position of the last occurrence of a
specified text:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The lastIndexOf() method returns the position of the last occurrence of a specified text:

21

Both methods accept a second parameter as the starting position for the search:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate", 15);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method accepts a second parameter as the starting position for
the search:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate",15);
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The indexOf() method accepts a second parameter as the starting position for the search:

21

The `lastIndexOf()` methods searches backwards, meaning: if the second parameter is `15`, the search starts at position 15, counting from the end, and searches to the beginning of the string.

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate", 15);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The lastIndexOf() method accepts a second parameter as the starting position
for the search.</p>
<p>Remember that the lastIndexOf() method searches backwards, so position 15
means 15 from the end.</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate", 15);
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The lastIndexOf() method accepts a second parameter as the starting position for the search.

Remember that the lastIndexOf() method searches backwards, so position 15 means 15 from the end.

7

Searching for a String in a String

The `search()` method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The search() method returns the position of the first occurrence of a
specified text in a string:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The search() method returns the position of the first occurrence of a specified text in a string:

7

The two methods, `indexOf()` and `search()`, are **equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are **NOT** equal. These are the differences:

- The `search()` method cannot take a second start position argument.
- The `indexOf()` method cannot take powerful search values (regular expressions).

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`
- `substr(start, length)`

The slice() Method

`slice()` extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The result of res will be:

```
Banana
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12, -6);
```

The result of res will be:

```
Banana
```

If you omit the second parameter, the method will slice out the rest of the string:

Example

```
var res = str.slice(7);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(7);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana, Kiwi

or, counting from the end:

Example

```
var res = str.slice(-12);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12)
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana, Kiwi

The substring() Method

`substring()` is similar to `slice()`.

The difference is that `substring()` cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

The result of *res* will be:

```
Banana
```

If you omit the second parameter, `substring()` will slice out the rest of the string.

The substr() Method

`substr()` is similar to `slice()`.

The difference is that the second parameter specifies the **length** of the extracted part.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6);
```

The result of res will be:

```
Banana
```

If you omit the second parameter, `substr()` will slice out the rest of the string.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7);
```

The result of res will be:

```
Banana, Kiwi
```

If the first parameter is negative, the position counts from the end of the string.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(-4);
```

The result of res will be:

```
Kiwi
```

Replacing String Content

The `replace()` method replaces a specified value with another value in a string:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Try it Yourself »

The `replace()` method does not change the string it is called on. It returns a new string.

By default, the `replace()` function replaces **only the first** match:

Example

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

By default, the `replace()` function is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("MICROSOFT", "W3Schools");
```

To replace case insensitive, use a **regular expression** with an `/i` flag (insensitive):

Example

```
str = "Please visit Microsoft!";
var n = str.replace(/MICROSOFT/i, "W3Schools");
```

To replace all matches, use a **regular expression** with a `/g` flag (global match):

Example

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace(/Microsoft/g, "W3Schools");
```

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
```

A string is converted to lower case with `toLowerCase()`:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

The concat() Method

`concat()` joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

The `concat()` method can be used instead of the plus operator. These two lines do the same:

Example

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

String.trim()

The `trim()` method removes whitespace from both sides of a string:

Example

```
var str = "       Hello World!        ";
alert(str.trim());
```

Extracting String Characters

There are 3 methods for extracting string characters:

- `charAt(position)`
- `charCodeAt(position)`
- Property access [ ]

The charAt() Method

The `charAt()` method returns the character at a specified index (position) in a string:

Example

```
var str = "HELLO WORLD";
str.charAt(0);            // returns H
```

The charCodeAt() Method

The `charCodeAt()` method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example

```
var str = "HELLO WORLD";

str.charCodeAt(0);          // returns 72
```

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

```
var str = "HELLO WORLD";
str[0];                    // returns H
```

Converting a String to an Array

A string can be converted to an array with the `split()` method:

Example

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

```html
<!DOCTYPE html>
<html>
<body>

<p>Click "Try it" to display the first array element, after a string split.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var str = "a,b,c,d,e,f";
  var arr = str.split(",");
  document.getElementById("demo").innerHTML = arr[0];
}
</script>

</body>
</html>
```

Click "Try it" to display the first array element, after a string split.

Try it

a

```html
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var str = "Hello";
var arr = str.split("");
var text = "";
var i;
for (i = 0; i < arr.length; i++) {
  text += arr[i] + "<br>"
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Write a JavaScript function to check whether a string is blank or not.

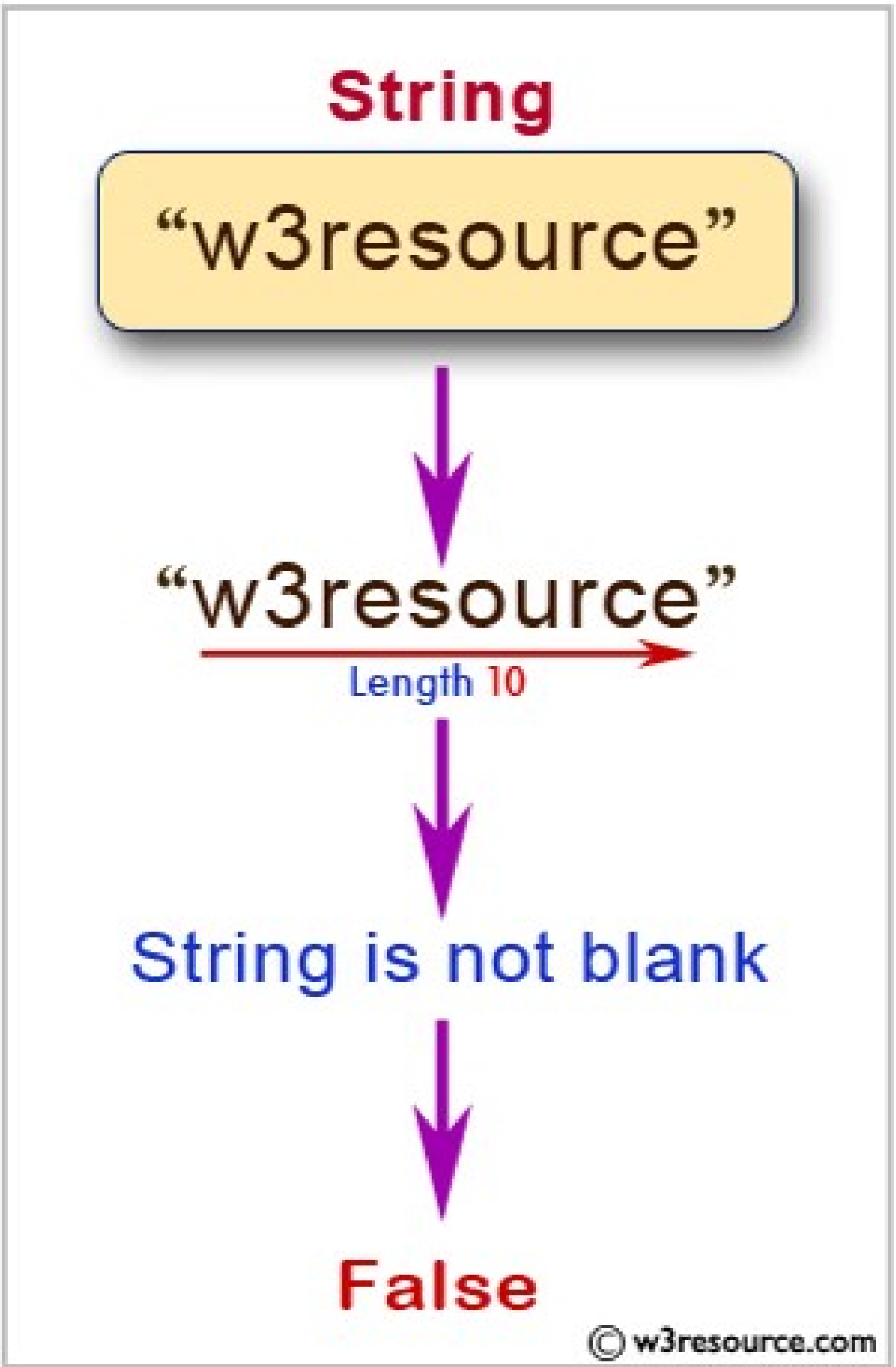
Test Data :

```
console.log(is_Blank(''));
console.log(is_Blank('abc'));
```

true

false

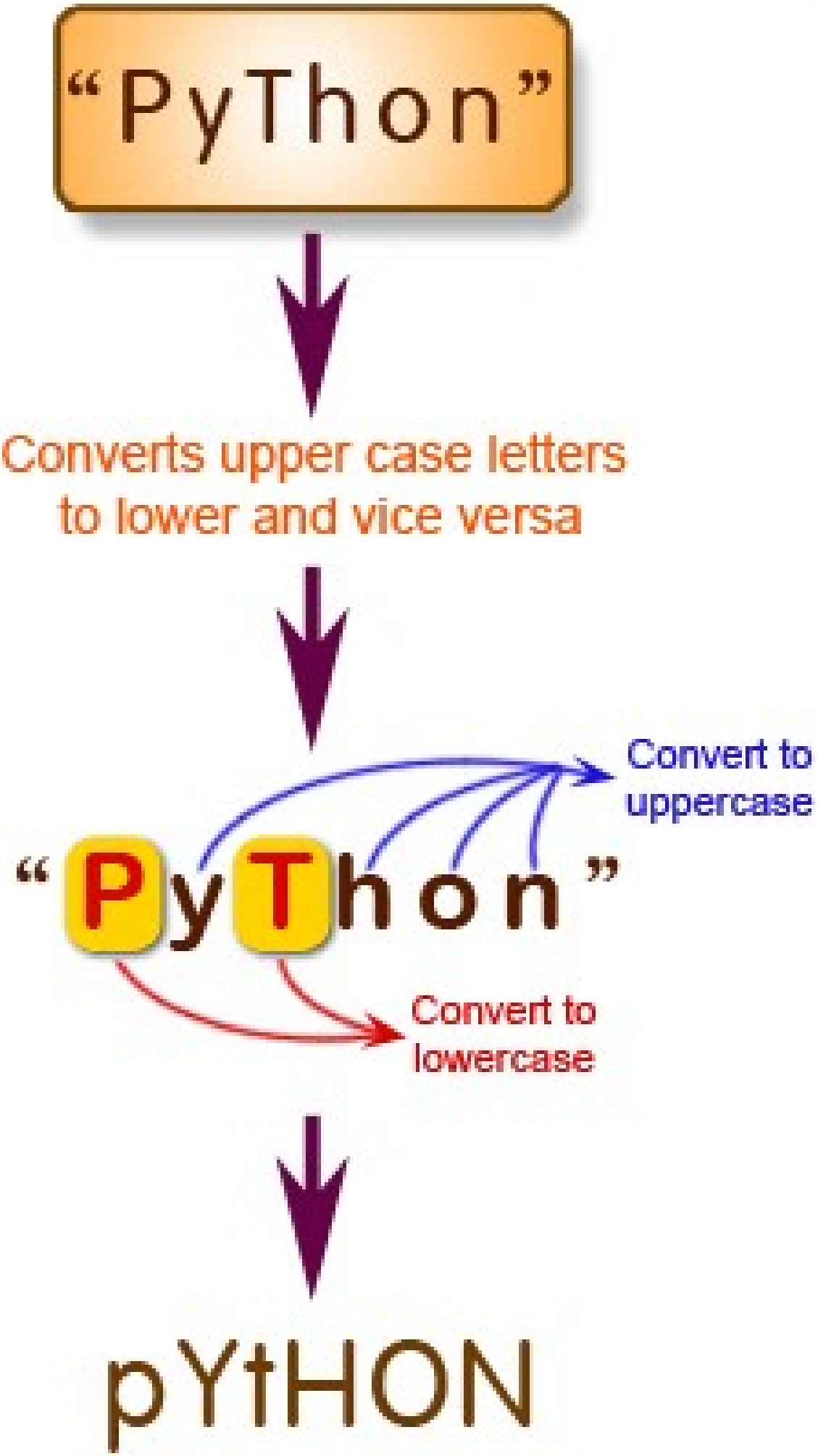
Pictorial Presentation:



JavaScript Code:

```
1  is_Blank =  function(input) {
2          if (input.length === 0)
3          return true;
4          else
5          return false;
6        }
7  console.log(is_Blank(''));
8  console.log(is_Blank('abc'));
```

Pictorial Presentation:


"PyThon"
Converts upper case letters
to lower and vice versa
Convert to
uppercase
"PyThon"
Convert to
lowercase
pYtHON

JavaScript Strings

A JavaScript string is zero or more characters written inside quotes.

Example

```
var x = "John Doe";
```

You can use single or double quotes:

Example

```
var carName1 = "Volvo XC60";  // Double quotes
var carName2 = 'Volvo XC60';  // Single quotes
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';
```

String Length

The length of a string is found with the built-in `length` property :

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Special Characters

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
var x = "We are the so-called "Vikings" from the north.";
```

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\ ) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (`\`) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence `\"` inserts a double quote in a string:

Example

```
var x = "We are the so-called \"Vikings\" from the north.";
```

The sequence `\'` inserts a single quote in a string:

Example

```
var x = 'It\'s alright.';
```

The sequence `\\` inserts a backslash in a string:

Example

```
var x = "The character \\ is called backslash.";
```

Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

Breaking Long Code Lines

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

A safer way to break up a string, is to use string addition:

Example

```
document.getElementById("demo").innerHTML = "Hello " +
"Dolly!";
```

You cannot break up a code line with a backslash:

Example

```
document.getElementById("demo").innerHTML = \
"Hello Dolly!";
```

Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

```
var firstName = "John";
```

But strings can also be defined as objects with the keyword `new`:

```
var firstName = new String("John");
```

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var x = "John";         // x is a string
var y = new String("John");  // y is an object

document.getElementById("demo").innerHTML =
typeof x + "<br>" + typeof y;
</script>

</body>
</html>
```

Don't create strings as objects. It slows down execution speed.
The `new` keyword complicates the code. This can produce some unexpected results:

When using the `==` operator, equal strings are equal:

Example

```
var x = "John";
var y = new String("John");

// (x == y) is true because x and y have equal values
```