JavaScript
**JavaScript**

**JavaScript** is one of the most popular programming languages in the world. It is supported by every major web browser, and makes web applications such as Gmail and Google Maps possible.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Language</th>
<th>Type</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Python</td>
<td>📚</td>
<td>100.0</td>
</tr>
<tr>
<td>2</td>
<td>Java</td>
<td>📚</td>
<td>95.4</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>📚</td>
<td>94.7</td>
</tr>
<tr>
<td>4</td>
<td>C++</td>
<td>📚</td>
<td>92.4</td>
</tr>
<tr>
<td>5</td>
<td>JavaScript</td>
<td>📚</td>
<td>88.1</td>
</tr>
<tr>
<td>6</td>
<td>C#</td>
<td>📚</td>
<td>82.4</td>
</tr>
<tr>
<td>7</td>
<td>R</td>
<td>📚</td>
<td>81.7</td>
</tr>
<tr>
<td>8</td>
<td>Go</td>
<td>📚</td>
<td>77.7</td>
</tr>
<tr>
<td>9</td>
<td>HTML</td>
<td>📚</td>
<td>75.4</td>
</tr>
<tr>
<td>10</td>
<td>Swift</td>
<td>📚</td>
<td>70.4</td>
</tr>
</tbody>
</table>

 Ranking of the Top 10 Programming Languages from the IEEE Spectrum website, 2021
JavaScript is not limited to web browsers. There are software tools that run it independently of browsers.

- One of the more popular tools that executes JavaScript outside of a web browser is **Node.js**, a JavaScript runtime environment that enables developers to write command-line tools.
- **Node.js** is also used for server-side scripting: running short programs on web servers to produce dynamic web page content before the page is sent to the user’s web browser.

JavaScript is executed by an interpreter, although modern JavaScript interpreters compile the JavaScript code at execution time into another format that can be executed quickly. This is called *just-in-time (JIT)* compilation.
Variables

In C++ and Java, you must declare the name and type of variables before you use them. Like Python, you don’t need to do this in JavaScript. However, you may declare the name of a variable using the `var` keyword.

```javascript
var total
result = 2000

Unlike C++ and Java, JavaScript uses *dynamic typing*, which means the type of a variable is determined at run time. So the variable `total` can contain an `int` at one point of a program and a `str` later.

```javascript
total = 1000
total = "This is the total:"
```
A constant is a variable whose value cannot be changed. In JavaScript, a constant is declared using the `const` keyword. By convention, constants are named with an identifier using only uppercase letters:

```javascript
const PI = 3.14159;
const TAX_RATE = 0.05;
```
## Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| **string** | Group of characters delimited with single or double quotes | name = "Hermione";  
quote = ‘She asked, "What time is it?"’; |
| **number** | Numbers with or without decimal places | highScore = 300;  
pi = 3.14; |
| **Boolean** | true or false | sunny = true;  
rainy = false; |
| **array** | Similar to a Python list | teams =  
["Longhorns","Cowboys","Texans"] |
| **object** | Key-value pairs, similar to a dictionary | movie = {title:"Frozen",  
rating:"G"}; |
| **undefined** | Variable without a value | var status; |
| **null** | The “not a value” value | carType = null; |
There are two ways to specify comments in JavaScript:

// This is a single-line comment.  
// It works like the hashtag in Python.

/* This is a multi-line comment.  It works like  
the triple-quote in Python.  Everything after the  
slash-star is part of the comment until you reach  
the matching star-slash at the end.  */
Semicolons

JavaScript statements *may or may not* be terminated by a semicolon.

// correct
x = 5
y = 10
z = x + y

// also correct
x = 5;
y = 10;
z = x + y;

/* but you can also use semicolons to separate two statements on the same line */
x = 5;  y = 10
z = x + y
Input and Output

A JavaScript program may ask the user to input a string using the `prompt()` function. This works a lot like the Python `input()` function.

```javascript
let testScore = prompt("Enter your test score: ")
```

document.write() and document.writeln() print strings directly into the document. writeln() adds a newline character at the end.

```javascript
document.writeln("The test score is" + testScore)
```

`console.log` prints a string to the web console. The console is kind of like the Python shell: it’s a command-line interpreter that is bundled with your browser that allows you to issue commands to your browser.

```javascript
console.log("The value printed was" + testScore)
```
Arithmetic Operators

+   addition
-   subtraction
*   multiplication
/   division
%   modulus (remainder)
**  exponentiation
++  incrementation (add 1)
--  decrementation (subtract 1)

+=  -=  *=  /=  %=  compound assignment operators
String Concatenation

The string concatenation operator “+” works a little differently in JavaScript as compared to Python.

- If both operands are numbers, + performs addition.
  \[ 2 + 3 \rightarrow 5 \]

- If both operands are strings, + performs string concatenation.
  \[ "2" + "3" \rightarrow "23" \]

- If one operand is a number and one is a string, the number is automatically converted to a string, and then the two strings are concatenated together.
  \[ "2" + 3 \rightarrow "23" \]

This last rule only applies to "+", not to any other arithmetic operators.

\[ "2" * 3 \rightarrow 6 \]
Data Conversion

The following JavaScript functions are useful for data conversion:

- `parseInt()`: convert a string into an integer.
  \[
  \text{parseInt("32") } \rightarrow 32
  \]

- `parseFloat()`: convert a string into a float.
  \[
  \text{parseFloat("98.6") } \rightarrow 98.6
  \]

- `String()`: convert a number into a string.
  \[
  \text{String(3.14159) } \rightarrow "3.14159"
  \]

If `parseInt()` or `parseFloat()` are given a non-number to parse, they return the value `NaN`. This is a JavaScript value that means “not a number”. For example,

\[
\text{parseInt("three") } \rightarrow \text{NaN}
\]

The JavaScript function `isNaN()` returns `true` if the argument is not a number, or `false` otherwise.
Conditionals

The *if-elif-else* structure in Python looks like this in JavaScript:

```javascript
if (condition1) {
    // block of code
} else if (condition2) {
    // block of code
} else if (condition3) {
    // block of code
    ...
} else {
    // block of code
}
```

- There is no keyword *elif* like there is in Python. Instead, JavaScript simply uses *else if*.
- Like Java and C++, curly braces `{ }` are used to delimit blocks of code, instead of a colon and indentation.
**Relational and Logical Operators**

==    equal
!=    not equal
===   identity
!==   non-identity
<     less than
<=    less than or equal to
>     greater than
>=    greater than or equal to

&&    | |    !    logical operators and, or, not
Conditional Operator (or Ternary Operator)

The conditional operator from Python looks and works the same way in JavaScript. The assignment statement:

\[
\text{value} = \text{condition} \ ? \ \text{expression1} : \ \text{expression2}
\]

means the same thing as

```javascript
if (condition) {
    value = expression1
} else {
    value = expression2
}
```
Switch

The switch statement is an alternative to writing multiple else-if statements.

```javascript
switch (expression) {
    case value1:
        // statements executed when expr === value1
        break
    case value2:
        // statements executed when expr === value2
        break
    . . .
    default:
        // statements executed when no cases match
}
```
This switch statement is equivalent to the following if-else structure:

```javascript
if (expression === value1) {
    // statements executed when expr === value1
}
else if (expression === value2) {
    // statements executed when expr === value2
    ...
}
else {
    // statements executed when no cases match
}
```
while and do-while Loops

A while loop looks and works just like you’d expect (remember curly braces):

```java
while (condition) {
    // body
}
```

A do-while loop tests the condition at the bottom. That means the body will always execute at least one time:

```java
do {
    // body
} while (condition)
```
for loops in JavaScript follow the C++ and Java style, and do not look like the ones you learned in Python:

```javascript
for (initialization; condition; final expression) {
    // body
}
```

initialization: a statement to execute before the loop starts (for example, to initialize a loop counter).

condition: a condition that tells you when to continue with the next iteration of the loop. (If true, execute the body of the loop.)

final expression: a statement to execute at the bottom of the loop (for example, to increment a loop counter).
You define functions in JavaScript using the `function` keyword (don’t forget the curly braces):

```javascript
function funcName(arg1,arg2,...) {
    // body
}
```

Example:

```javascript
function average(a,b,c) {
    return (a+b+c) / 3
}
```

```javascript
avg = average(213,184,192)
document.writeln("avg = " + avg)
```
## Array Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(value)</td>
<td>Add a value to the end of the array</td>
<td>nums = [2, 4, 6]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nums.push(8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--&gt; nums becomes [2, 4, 6, 8]</td>
</tr>
<tr>
<td>pop()</td>
<td>Remove last element and return it</td>
<td>nums = [2, 4, 6]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x = nums.pop()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--&gt; returns 6, nums becomes [2, 4]</td>
</tr>
<tr>
<td>unshift(value)</td>
<td>Add a value to the beginning of the array</td>
<td>nums = [2, 4, 6]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nums.unshift(0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--&gt; nums becomes [0, 2, 4, 6]</td>
</tr>
<tr>
<td>shift()</td>
<td>Remove first element and return it</td>
<td>nums = [2, 4, 6]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x = nums.shift()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--&gt; returns 2, nums becomes [4, 6]</td>
</tr>
</tbody>
</table>
### Array Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| splice(start, count, values) | Add or remove elements from anywhere in the array and return deleted elements, if any | nums = [2, 4, 6, 8, 10]  
nums.splice(3)  
--> deletes all elements from index 3 to the end: nums becomes [2, 4, 6]  
nums.splice(0, 2)  
--> deletes 2 elements starting at index 0: nums becomes [6]  
nums.splice(0, 0, 3, 5)  
--> adds 3 and 5 starting at index 0: nums becomes [3, 5, 6]  
nums.splice(2, 0, 7, 9, 11)  
--> adds 7, 9, and 11 starting at index 2: nums becomes [3, 5, 7, 9, 11, 6] |
## Array Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| `indexOf(item)`         | returns the index of the first time `item` appears in the array, starting at `start` | `nums = [0,2,4,2,0]`  
  `nums.indexOf(2)`  
  `nums.indexOf(2,2)`  
  `nums.indexOf(2) --> 1`  
  `nums.indexOf(2,2) --> 3` |
| `indexOf(item, start)` |                                                               |                                              |
| `lastIndexOf(item)`     | returns the index of the last time `item` appears in the array, starting at `start` | `nums = [0,2,4,2,0]`  
  `nums.lastIndexOf(2)`  
  `nums.lastIndexOf(2,2)`  
  `nums.lastIndexOf(2) --> 3`  
  `nums.lastIndexOf(2,2) --> 1` |
| `lastIndexOf(item, start)` |                                                              |                                              |
| `sort()`                | sort an array in ascending order by treating each element as a string using Unicode values | `nums = [200,30,1000,4]`  
  `nums.sort()`  
  `nums.sort() --> [1000,200,30,4]` |
| `sort(funcName)`        | `funcName` is the name of a function that tells sort how to order two items | `function compare(x,y) { return x > y`  
  `nums.sort(compare)`. |
## String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| charAt(index)            | returns the index of the first time item appears in the array, starting at start | nums = [0,2,4,2,0]  
nums.indexOf(2)  --&gt; 1  
nums.indexOf(2,2)  --&gt; 3 |
| lastIndexOf(item)        | returns the index of the last time item appears in the array, starting at start | nums = [0,2,4,2,0]  
nums.lastIndexOf(2)  --&gt; 3  
nums.lastIndexOf(2,2)  --&gt; 1 |
| lastIndexOf(item, start) |                                                                             |                                                                         |
| sort()                   | sort an array in ascending order by treating each element as a string using Unicode values | nums = [200,30,1000,4]  
nums.sort()  --&gt; [1000,200,30,4] |
| sort(funcName)           | funcName is the name of a function that tells sort how to order two items   | function compare(x,y) {  
    return x &gt; y  
}  
nums.sort(compare) |