UTN | FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

**Asignatura: Ingeniería de Software**

Trabajo Práctico N° 6

# "Implementación de User Stories"

Curso: **4K4**


Grupo 3:

- Antonellini Juan Manuel, 63029
- Paz Fessia Facundo, 78579
- Demasi Pablo Sebastián, 76726
- Valle Matías 70869,
- Caminos Julieta, 61123

Docentes:

- Covaro, Laura Ines
- Boiero Rovera, Gerardo Javier
- Crespo, María Mickaela


**Fecha de Presentación**: 26/04/2022

# Índice

# Enunciado

| Unidad | Nro. 3: Gestión del Software como producto |
|---|---|
| Consigna | Implementar una User Story determinada usando un lenguaje de programación elegido por el grupo respetando un documento de reglas de estilo. |
| Objetivo | Que el estudiante comprenda la implementación de una User Story como una porción transversal de funcionalidad que requiere la colaboración de un equipo multidisciplinario. |
| Propósito | Familiarizarse con los conceptos de requerimientos ágiles y en particular con User Stories en conjunto con la aplicación de las actividades de SCM correspondientes. |
| Entradas | Conceptos teóricos sobre el tema desarrollados en clase.<br>Definición completa de las User Stories correspondientes al Trabajo Práctico 2 "Requerimientos Ágiles – User Stories y Estimaciones" |
| Salida | Implementación de la User Story correspondiente en un programa ejecutable<br>Documento de estilo de código |
| Instrucciones | • Seleccionar una User Story a implementar de entre las siguientes opciones:<br>  ○ Realizar Pedido a Comercio adherido (grupos pares)<br>  ○ Realizar un Pedido de "lo que sea" (grupos impares)<br>• Seleccionar el conjunto de tecnologías para implementar la funcionalidad elegida.<br>• Buscar y seleccionar un documento de buenas prácticas y/o reglas de estilo de código para el lenguaje de programación a utilizar.<br>• Implementar la US siguiendo las reglas de estilo determinadas. |

## Reglas de Estilo de Código

Fuente: https://www.w3schools.com/js/js_conventions.asp

# JavaScript Coding Conventions

Coding conventions are **style guidelines for programming**. They typically cover:

- Naming and declaration rules for variables and functions.
- Rules for the use of white space, indentation, and comments.
- Programming practices and principles

Coding conventions **secure quality**:

- Improves code readability
- Make code maintenance easier

Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.

## Variable Names

At W3schools we use **camelCase** for identifier names (variables and functions).

All names start with a **letter**.

At the bottom of this page, you will find a wider discussion about naming rules.

```
firstName = "John";
lastName = "Doe";

price = 19.90;
tax = 0.20;

fullPrice = price + (price * tax);
```

## Spaces Around Operators

Always put spaces around operators ( = + - * / ), and after commas:

### Examples:

```
let x = y + z;
```

```
const myArray = ["Volvo", "Saab", "Fiat"];
```

## Code Indentation

Always use 2 spaces for indentation of code blocks:

### Functions:

```
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
```

Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently.

## Statement Rules

General rules for simple statements:

- Always end a simple statement with a semicolon.

### Examples:

```
const cars = ["Volvo", "Saab", "Fiat"];

const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

General rules for complex (compound) statements:

- Put the opening bracket at the end of the first line.
- Use one space before the opening bracket.
- Put the closing bracket on a new line, without leading spaces.
- Do not end a complex statement with a semicolon.

### Functions:

```
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
```

### Loops:

```
for (let i = 0; i < 5; i++) {
  x += i;
}
```

### Conditionals:

```
if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

## Object Rules

General rules for object definitions:

- Place the opening bracket on the same line as the object name.
- Use colon plus one space between each property and its value.
- Use quotes around string values, not around numeric values.
- Do not add a comma after the last property-value pair.
- Place the closing bracket on a new line, without leading spaces.
- Always end an object definition with a semicolon.

### Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Short objects can be written compressed, on one line, using spaces only between properties, like this:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

## Line Length < 80

For readability, avoid 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 or a comma.

### Example

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

# Naming Conventions

Always use the same naming convention for all your code. For example:

- Variable and function names written as **camelCase**
- Global variables written in **UPPERCASE** (We don't, but it's quite common)
- Constants (like PI) written in **UPPERCASE**

Should you use **hyp-hens**, **camelCase**, or **under_scores** in variable names?

This is a question programmers often discuss. The answer depends on who you ask:

**Hyphens in HTML and CSS:**

HTML5 attributes can start with data- (data-quantity, data-price).

CSS uses hyphens in property-names (font-size).

Hyphens can be mistaken as subtraction attempts. Hyphens are not allowed in JavaScript names.
**Underscores:**

Many programmers prefer to use underscores (date_of_birth), especially in SQL databases.

Underscores are often used in PHP documentation.

**PascalCase:**

PascalCase is often preferred by C programmers.

**camelCase:**

camelCase is used by JavaScript itself, by jQuery, and other JavaScript libraries.

Do not start names with a $ sign. It will put you in conflict with many JavaScript library names.

# Loading JavaScript in HTML

Use simple syntax for loading external scripts (the type attribute is not necessary):

```
<script src="myscript.js"></script>
```

# Accessing HTML Elements

A consequence of using "untidy" HTML styles, might result in JavaScript errors.

These two JavaScript statements will produce different results:

```
const obj = getElementById("Demo")
```

```
const obj = getElementById("demo")
```

If possible, use the same naming convention (as JavaScript) in HTML.

## File Extensions

HTML files should have a **.html** extension (**.htm** is allowed).

CSS files should have a **.css** extension.

JavaScript files should have a **.js** extension.

## Use Lower Case File Names

Most web servers (Apache, Unix) are case sensitive about file names:

london.jpg cannot be accessed as London.jpg.

Other web servers (Microsoft, IIS) are not case sensitive:

london.jpg can be accessed as London.jpg or london.jpg.

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive, to a case sensitive server, even small errors can break your web site.

To avoid these problems, always use lowercase file names (if possible).

## Performance

Coding conventions are not used by computers. Most rules have little impact on the execution of programs.

Indentation and extra spaces are not significant in small scripts.

For code in development, readability should be preferred. Larger production scripts should be minified.

# HTML Style Guide

Source: https://www.w3schools.com/html/html5_syntax.asp

A consistent, clean, and tidy HTML code makes it easier for others to read and understand your code.

Here are some guidelines and tips for creating good HTML code.

---

## Always Declare Document Type

Always declare the document type as the first line in your document.

The correct document type for HTML is:

```
<!DOCTYPE html>
```

---

## Use Lowercase Element Names

HTML allows mixing uppercase and lowercase letters in element names.

However, we recommend using lowercase element names, because:

- Mixing uppercase and lowercase names looks bad
- Developers normally use lowercase names
- Lowercase looks cleaner
- Lowercase is easier to write

### Good:

```
<body>
<p>This is a paragraph.</p>
</body>
```

### Bad:

```
<BODY>
<P>This is a paragraph.</P>
</BODY>
```

---

---

## Close All HTML Elements

In HTML, you do not have to close all elements (for example the <p> element).

However, we strongly recommend closing all HTML elements, like this:

**Good:**

```
<section>
  <p>This is a paragraph.</p>
  <p>This is a paragraph.</p>
</section>
```

**Bad:**

```
<section>
  <p>This is a paragraph.
  <p>This is a paragraph.
</section>
```

---

## Use Lowercase Attribute Names

HTML allows mixing uppercase and lowercase letters in attribute names.

However, we recommend using lowercase attribute names, because:

- Mixing uppercase and lowercase names looks bad
- Developers normally use lowercase names
- Lowercase look cleaner
- Lowercase are easier to write

**Good:**

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

**Bad:**

```
<a HREF="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

---

## Always Quote Attribute Values

HTML allows attribute values without quotes.

However, we recommend quoting attribute values, because:

- Developers normally quote attribute values
- Quoted values are easier to read
- You MUST use quotes if the value contains spaces

**Good:**

```
<table class="striped">
```

Bad:

```
<table class=striped>
```

Very bad:

This will not work, because the value contains spaces:

```
<table class=table striped>
```

---

## Always Specify alt, width, and height for Images

Always specify the alt attribute for images. This attribute is important if the image for some reason cannot be displayed.

Also, always define the width and height of images. This reduces flickering, because the browser can reserve space for the image before loading.

Good:

```
<img src="html5.gif" alt="HTML5" style="width:128px;height:128px">
```

Bad:

```
<img src="html5.gif">
```

---

## Spaces and Equal Signs

HTML allows spaces around equal signs. But space-less is easier to read and groups entities better together.

Good:

```
<link rel="stylesheet" href="styles.css">
```

Bad:

```
<link rel = "stylesheet" href = "styles.css">
```

---

## Avoid Long Code Lines

When using an HTML editor, it is NOT convenient to scroll right and left to read the HTML code.

Try to avoid too long code lines.

---

# Blank Lines and Indentation

Do not add blank lines, spaces, or indentations without a reason.

For readability, add blank lines to separate large or logical code blocks.

For readability, add two spaces of indentation. Do not use the tab key.

### Good:

```
<body>

<h1>Famous Cities</h1>

<h2>Tokyo</h2>
<p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area, and the most populous metropolitan area in the world.</p>

<h2>London</h2>
<p>London is the capital city of England. It is the most populous city in the United Kingdom.</p>

<h2>Paris</h2>
<p>Paris is the capital of France. The Paris area is one of the largest population centers in Europe.</p>

</body>
```

### Bad:

```
<body>
<h1>Famous Cities</h1>
<h2>Tokyo</h2><p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area, and the most populous metropolitan area in the world.</p>
<h2>London</h2><p>London is the capital city of England. It is the most populous city in the United Kingdom.</p>
<h2>Paris</h2><p>Paris is the capital of France. The Paris area is one of the largest population centers in Europe.</p>
</body>
```

### Good Table Example:

```
<table>
  <tr>
      <th>Name</th>
      <th>Description</th>
  </tr>
  <tr>
      <td>A</td>
```

```
    <td>Description of A</td>
  </tr>
  <tr>
    <td>B</td>
    <td>Description of B</td>
  </tr>
</table>
```

### Good List Example:

```
<ul>
  <li>London</li>
  <li>Paris</li>
  <li>Tokyo</li>
</ul>
```

## Never Skip the <title> Element

The <title> element is required in HTML.

The contents of a page title is very important for search engine optimization (SEO)! The page title is used by search engine algorithms to decide the order when listing pages in search results.

The <title> element:

- defines a title in the browser toolbar
- provides a title for the page when it is added to favorites
- displays a title for the page in search-engine results

So, try to make the title as accurate and meaningful as possible:

```
<title>HTML Style Guide and Coding Conventions</title>
```

---

## Omitting <html> and <body>?

An HTML page will validate without the <html> and <body> tags:

### Example

```
<!DOCTYPE html>
<head>
  <title>Page Title</title>
</head>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>
```

However, we strongly recommend to always add the <html> and <body> tags!

Omitting <body> can produce errors in older browsers.

Omitting <html> and <body> can also crash DOM and XML software.

---

## Omitting <head>?

The HTML <head> tag can also be omitted.

Browsers will add all elements before <body>, to a default <head> element.

**Example**

```
<!DOCTYPE html>
<html>
<title>Page Title</title>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

However, we recommend using the <head> tag.

## Close Empty HTML Elements?

In HTML, it is optional to close empty elements.

**Allowed:**

```
<meta charset="utf-8">
```

**Also Allowed:**

```
<meta charset="utf-8" />
```

If you expect XML/XHTML software to access your page, keep the closing slash (/), because it is required in XML and XHTML.

---

## Add the lang Attribute

You should always include the lang attribute inside the <html> tag, to declare the language of the Web page. This is meant to assist search engines and browsers.

### Example

```
<!DOCTYPE html>
<html lang="en-us">
<head>
  <title>Page Title</title>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

## Setting The Viewport

The viewport is the user's visible area of a web page. It varies with the device - it will be smaller on a mobile phone than on a computer screen.

You should include the following <meta> element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This gives the browser instructions on how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

## HTML Comments

Short comments should be written on one line, like this:

```
<!-- This is a comment -->
```

Comments that spans more than one line, should be written like this:

```
<!--
  This is a long comment example. This is a long comment example.
  This is a long comment example. This is a long comment example.
-->
```

Long comments are easier to observe if they are indented with two spaces.

---

## Using Style Sheets

Use simple syntax for linking to style sheets (the type attribute is not necessary):

```
<link rel="stylesheet" href="styles.css">
```

Short CSS rules can be written compressed, like this:

```
p.intro {font-family:Verdana;font-size:16em;}
```

Long CSS rules should be written over multiple lines:

```
body {
  background-color: lightgrey;
  font-family: "Arial Black", Helvetica, sans-serif;
  font-size: 16em;
  color: black;
}
```

- Place the opening bracket on the same line as the selector
- Use one space before the opening bracket
- Use two spaces of indentation
- Use semicolon after each property-value pair, including the last
- Only use quotes around values if the value contains spaces
- Place the closing bracket on a new line, without leading spaces

---

## Loading JavaScript in HTML

Use simple syntax for loading external scripts (the type attribute is not necessary):

```
<script src="myscript.js">
```

---

## Accessing HTML Elements with JavaScript

Using "untidy" HTML code can result in JavaScript errors.

These two JavaScript statements will produce different results:

### Example

getElementById("Demo").innerHTML = "Hello";

getElementById("demo").innerHTML = "Hello";

## Use Lower Case File Names

Some web servers (Apache, Unix) are case sensitive about file names: "london.jpg" cannot be accessed as "London.jpg".

Other web servers (Microsoft, IIS) are not case sensitive: "london.jpg" can be accessed as "London.jpg".

If you use a mix of uppercase and lowercase, you have to be aware of this.

If you move from a case-insensitive to a case-sensitive server, even small errors will break your web!

To avoid these problems, always use lowercase file names!

---

## File Extensions

HTML files should have a **.html** extension (**.htm** is allowed).

CSS files should have a **.css** extension.

JavaScript files should have a **.js** extension.

---

## Differences Between .htm and .html?

There is no difference between the .htm and .html file extensions!

Both will be treated as HTML by any web browser and web server.

---

## Default Filenames

When a URL does not specify a filename at the end (like "https://www.w3schools.com/"), the server just adds a default filename, such as "index.html", "index.htm", "default.html", or "default.htm".

If your server is configured only with "index.html" as the default filename, your file must be named "index.html", and not "default.html".

However, servers can be configured with more than one default filename; usually you can set up as many default filenames as you want.