ISe
Ingeniería de Servicios

Universidad Rey Juan Carlos

Servicios Móviles y Ubicuos- Transparencias

Índice

Servicios Móviles y Ubicuos

Presentación de la asignatura

Departamento de Ciencias de la Educación, Lenguaje, Cultura y Artes, Ciencias Histórica-Jurídicas y Humanísticas y Lenguas Modernas

www.kybele.es

- **Fco. Javier Pérez**
  - **Despacho:** 2012A Ed. Ampliación de Rectorado, Móstoles.
  - **Contacto:** francisco.perez@urjc.es
  - **Tutorías:** a petición del alumno.

Objetivos de la Asignatura

Competencias a adquirir:

- **Conocer y aplicar fundamentos** de las tecnologías de la información y las comunicaciones al ámbito de los servicios móviles y ubicuos.
- Capacidad para comprender el **proceso de análisis y desarrollo** de sistemas de servicios móviles y ubicuos.
- Capacidad para adquirir conocimientos básicos sobre **estructura y funcionamiento** para la implantación de sistemas de servicios móviles y ubicuos.
- Capacidad para analizar y comprender los **principios básicos** de las redes móviles y fijas y las aplicaciones basadas en tecnologías de red para la implantación de sistemas basados en servicios.

Objetivos de la Asignatura

Algunos resultados de aprendizaje:

- Conocimiento de los **fundamentos** en los que se basa la computación móvil y ubicua.
- Análisis y utilización de las **diferentes plataformas** de desarrollo y ejecución de **servicios móviles**.
- Desarrollo de aplicaciones y sistemas basados en **computación ubicua**.

Programa de la asignatura

Teoría

- Tema 1: Desarrollo de Servicios Móviles.
- Tema 2: Computación Ubicua.

Práctica

- Tema 3: Aplicaciones Híbridas.
- Tema 4: Aplicaciones Nativas en Android.

Evaluación continua

- **Participación en foro de discusión** (10%).
- **Trabajo de investigación** (20% - Nota mínima: 5).
- **Proyecto final** (30% - Nota mínima: 5 – en cada parte):
  - Desarrollo de una aplicación híbrida (15%).
  - Desarrollo de una aplicación nativa Android (15%).
- **Examen final** (40% - Nota mínima: 4):
  - 2/3 contenidos de la asignatura, 1/3 trabajos de investigación.

Bibliografía

- Learning Mobile App Development: A Hands-on Huide to Building Apps With iOS and Android. Jakob Iversen, Michael Eierman. Addison-Wesley, 2014.

- Professional Mobile Application Development. Jeff McWherter, Scott Gowell. Wrox, 2012.

Tema 1: Desarrollo de Servicios Móviles

Servicios Móviles y Ubicuos

- A **mobile service** is a service offered by means of a **mobile device**
  - (In the context of this course)
- Mobile device: a portable computing device
- Basic features:
  - Small enough to hold and operate in the hand
  - Operating system capable of running mobile apps
  - Network access for data communications
  - Local non-removable data storage

Mobile device types

- Types
  - Mobile phones / smartphones
  - Tablets
  - Wearable computers
  - Portable media players
  - Digital cameras (photo / video)
  - Personal navigation devices
  - Personal digital assistants (PDAs)
  - …
- Even though laptops can be moved they are too big to be carried all the time

Mobile devices key features

- Computing capabilities
- Built-in sensors
- Different communication capabilities
  - Near Field Communication (NFC)
  - Bluetooth
  - WiFi
  - Cellular (3G, 4G…)

- Reaching costumers
  - Users usually have their devices within reach
    - Repetition: they can check for updated content
    - Boredom: they can use the service while passing time
    - Urgency: they can use the service if they have an urge
  - Mobile devices usage keeps growing
    - http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/
- Innovation
  - Mobile services can make use of the devices capabilities to provide new and better services

Mobile operating systems

- A mobile operating system (OS) is an operating system for mobile devices
- On top of the job of any computer operating system it controls features specific for mobile
  - Phone
  - Touchscreen
  - GPS
  - Accelerometer
  - Camera
  - …

Mobile OS market share

- Main mobile OSs for smartphones and tablets currently (2017):
  - Android
  - iOS (Apple)
  - Windows Phone
  - Windows 10 (tablets)
- Some stats:
  - Sales
    - https://en.wikipedia.org/wiki/Mobile_operating_system#Market_share
    - http://www.idc.com/promo/smartphone-market-share/os
  - Web traffic
    - https://deviceatlas.com/device-data/explorer/webusage-by-country/traffic/no-tablet/country/es/type/os_name

How to provide a mobile service

- Two main ways
  - Web
  - App
- Web
  - Like any other web, adapted for mobile use
  - Accessible using a browser in the mobile device
- Mobile app
  - Software application designed to run in a mobile device
  - Specific for an operating system

Web for mobile

- Mobile devices usually have web browsers and can access any web content
- For a web site / web application to be usable in a mobile device its user interface needs to be adapted to the device screen size and other mobile features
- Most web sites and applications are also accesible for desktop use
- Two ways of adapting a web for mobile use:
  - Dedicated version of the website for mobile
  - Responsive web design

Browser detection

- To know if a device is mobile and send the user the proper version of the website
- Based on the value ‘user-agent’ header in HTTP request
  - Also available in navigator.userAgent in JavaScript
- Tricky and fragile
  - Each browser can send any value they decide to

https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent

Responsive web design

- An approach to web design aimed at allowing webpages to be displayed in response to some features of the user's browser
  - Usually screen size
- Main technical resource: CSS media queries
  - Sections with CSS rules to be applied when certain criteria are met
- Some design concepts:
  - Mobile first: design first for mobile and then add more features for bigger screens
    - Progressive enhancement
  - Opposite: graceful degradation

CSS media queries

- Expressions that are evaluated to a boolean value
  - If the result is true the rules related to the media query are applied
- Two ways to express media queries
  - In a link to a stylesheet

```
<link rel="stylesheet" media="(max-width: 800px)" href="example.css" />
```

  - Inside a CSS

```
@media (max-width: 600px) {
  .facet_sidebar {
    display: none;
  }
}
```

Mobile apps

- Mobile apps are specific for an operating system
  - To provide the same service to different platforms a distribution of the app needs to be developed for each of them
- They are usually distributed through the app stores
  - Android
    - Google Play
    - Amazon Appstore
  - iOS
    - App Store
  - Windows Phone
    - Windows Phone Store

Web advantages

- No need for developing a version of the application for each platform
- Web development (easier to find developers)
  - http://stackoverflow.com/research/developer-survey-2016
- Cheaper development
  - Development team
  - Hardware

App advantages

- Users can access apps more easily once installed
- Faster and less data transfer consumption
- Available even if there is no network connection
- Apps can use mobile-specific features (camera, accelerometer, notifications, data storage…)
  - HTML5 allows web to use some of them

App development: native

- A native application is an application program developed for use on a particular platform or device
  - Since they are specific they can interact with the OS
- Native apps are developed using the languages specific for each platform
  - Android
    - Java
  - iOS
    - Objective-C
    - Swift
  - Windows Phone
    - .NET

Native development: implications

- Since development is specific for each platform different versions of the app need to be developed to offer it to different platforms
- Development cost is high
  - Team: developers for each platform needed
  - Hardware: iOS apps can only be developed in Mac
  - Different devices are needed to test
  - Development licences
- All device or platform features can be used

Third-party solutions

- Some solutions to avoid the extra cost of developing different versions of an app for each platform
  - Native
    - Compiled: code in a single language compiled to native binaries for each platform
    - Interpreted: a virtual machine interprets the code at runtime
    - In some cases only a subset of native features are available
  - Hybrid
    - Based on HTML, CSS and JavaScript
    - Displayed in a WebView element
    - Plugins to provide native device functionality

Some native third-party solutions

- Xamarin
  - .NET (C#)
- Appcelerator Titanium
  - JavaScript
- React Native
  - Based on React (JavaScript library for UI)
- Unity
  - It is a game development platform, but can be used for developing apps

Hybrid solutions

- Apache Cordova is the most popular hybrid application development framework
  - Native container that wraps the app
  - App built with HTML, CSS and JavaScript like any web
  - JavaScript API to access device functions
    - Many plugins available for different functions
    - Custom plugins can be developed
  - Different distributions based on Cordova
    - PhoneGap
    - Ionic
    - …

Hybrid vs Native vs Web

- Cost of development, maintenance & time to market
  - Web < Hybrid < Native
- User experience
  - Native > Hybrid > Web
- Device features access
  - Native > Hybrid > Web
- Other things to consider
  - Expertise of available development team
  - Reusable code from web to hybrid

Hybrid vs Native vs Web

- Exercise: read and discuss some articles on pros and cons of each option and when to choose each type
  - http://www.mrc-productivity.com/blog/2016/06/the-mobile-app-comparison-chart-hybrid-vs-native-vs-mobile-web/
  - http://howwedostartups.com/articles/Native-vs-Hybrid
  - https://www.ymedialabs.com/hybrid-vs-native-mobile-apps-the-answer-is-clear/
  - https://www.weebpal.com/blog/hybrid-vs-native-mobile-app-develpment-when-go-hybrid

App development issues

- Design
  - Life cycle
  - Screen size and orientation
  - Connectivity
  - Battery
  - Hardware
- Testing
- Publishing

App life cycle

- Mobile operating systems are not true multitasking systems
  - Only one app can be active at a time
  - When an app is interrupted is put in background
  - The OS can kill an app in the background if available memory is low or if it remains too long
- To handle these state changes there is a life cycle for apps
  - Code can be executed on state changes
    - Usually to store and retrieve data
  - Different in Android and iOS

Screen size and orientation

- Mobile devices have small screens
  - UIs need to be designed carefully
  - Scroll is possible, but should be used carefully
    - Especially horizontal scroll
- Mobile devices have different screen resolutions
  - App devlopment should take this in consideration
- Users can change screen orientation
  - This also has to be taken into account
  - Sometimes orientation can be locked so that the app only works in one

- Mobile devices can lose connectivity or have very low speeds
- Apps should be designed to take this possibility into account
  - Warn users when connectivity is lost and the app cannot perform some actions
  - Download and upload data asynchronously
    - Users can do other things in the meantime
  - Cache data to be uploaded when connection is not available

Battery

- Mobile devices are not always connected to a power source
  - It is important to design apps to use only the energy they really need
- Main power drains
  - Screen
    - Make sure the code is efficient
  - Sensors (GPS, camera, communication…)
    - The app should only turn them on when they are needed
    - Turn off sensors once the task is completed
    - Also to be handled in app life cycle

- Availability
  - Not all mobile devices have the same hardware components
  - Some components can be unavailable in certain situations
    - For example, GPS indoors
- Delays
  - Some hardware components need time to initialize
- Accuracy
  - Components' accuracy can be different in different devices
  - In some cases accuracy improves with time (GPS)

- Testing apps is expensive
  - Emulators and simulators are not enough
    - Some errors only happen in actual devices
  - New releases of operating systems can introduce some differences
    - New testing needs to be done for every new release
  - In Android there are many different manufacturers, and some features can work in a slightly different way
    - iOS devices are more standardized, but still there are different clases (iPad, iPhone, iPod Touch) and OS versions

App distribution (1/3)

- The most common way to distribute an app is through the stores (Google Play / Apple Store)
- To publish an app in the stores some items need to be provided
  - Icon
  - Screenshots
  - Description
  - Category
  - Price (the store gets a part)
  - Application package
    - APK in Android
    - IPA in iOS

App distribution (2/3)

- The stores have some requirements and guidelines that need to be followed for the app to be accepted
  - Apple reviews apps before publishing
    - Publishing takes longer
    - Apps can be rejected (not unusual)
  - Google publishes apps but can remove them later if they find out that it violates the rules (less usual)
- There is a fee required to be able to publish
- Users rate apps in the stores
  - Important to have good reviews from the beginning!

App distribution (3/3)

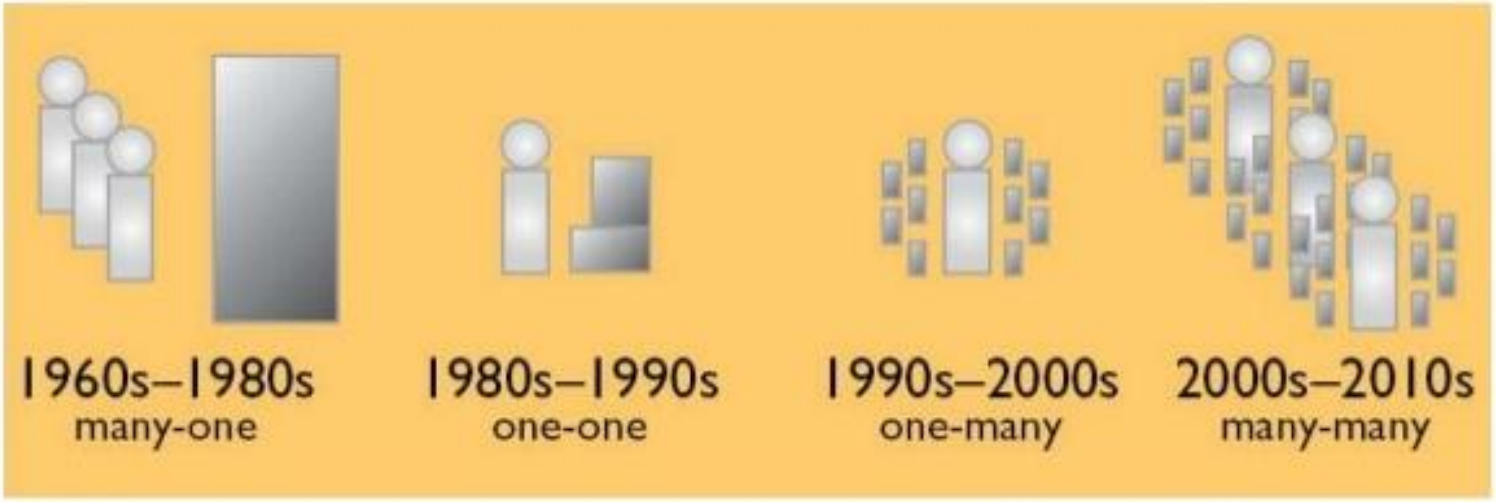
- Apps can also be distributed to individuals in an organization for internal use
  - In Android it can be done easily sending the APK
    - The device needs to be set up to accept apps from unknown sources
  - In iOS an additional license is required
    - iOS Enterprise Developer license (299$ per year)
  - Useful for apps that support a service, instead of providing the service itself

Tema 2: Computación ubicua

Servicios Móviles y Ubicuos

Three waves of computing

- Mainframe computing (60's – 80's)
  - Many users – one computer
- Desktop computing (80's – 90's)
  - One user – one computer
- Ubiquitous computing (since 90's)
  - One user – many computers



What is ubiquitous computing (ubicomp)

- Ubiquitous means everywhere
- Term coined by Mark Weiser (Xerox PARC) in 1988 in “The Computer for the 21st Century”
  - Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user
  - “The Computer for the 21st Century”

https://www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf

Some quotes by Weiser

- The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.
- Computers are approachable only through complex jargon that has nothing to do with the tasks for which people actually use computers.
- Machines that fit the human environment, instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods.

What is not ubiquitous computing

- Computers that can be carried to any place (mobile)
  - They are not part of the background, they are the focus
  - But they can be part of a ubiquitous system
- Virtual reality
  - It creates another world, instead of becoming part of the real world

Enabling technologies

- Processing
  - Cheaper, faster, smaller, more energy efficient
- Storage
  - Big, fast and small in size.
- Networking
  - Global, local, ad-hoc, low-power, high bandwidth, low latencies
- Sensors
  - Types, speed, accuracy, price and robustness.
- Displays and user interfaces
  - Projection, flexible materials, low power, new interaction ways
- Actuators
  - Computer controlled

Examples of applications

- Building and home automation
  - http://www.essence-grp.com/smart-living
- Healthcare
  - http://www.essence-grp.com/smart-care
- Transportation
- Monitoring
  - Environment, manufacturing, stockage…
- All daily activities
  - https://youtu.be/6Cf7IL_eZ38

General features

- There are three general features that are shared across a wide variety of ubicomp applications
  - the ability to provide **transparent interfaces**
  - the ability to automatically adapt the behavior of a program based on knowledge of the **context** of its use
  - the ability to automate the **capture of live experiences** for later recall
- These features are not unique to ubiquitous systems

Transparent interaction

- Ubicomp vision: “pervasive computation without intrusion”
- Remove the physical barrier between user and computational device
- Keyboard and mouse are still the most commonly used interfaces
- Need:
  - Flexible interfaces
  - Varied interfaces that can provide similar functionality

Context awareness

- Context – information about the environment in which the application operates and reacts accordingly.
- LOCATION and TIME are simple examples of context
- Context-aware application is one which:
  - can **capture** the context
  - assign **meaning** to it
  - change **behavior** accordingly
- Applications with context awareness allow rapid personalization of their services

Automated capture

- Capture everyday experiences and make the records available for later use
- Constraints:
  - Multiple streams of information
  - Their time synchronization
  - Their correlation and integration
- Automated tools that support capture, integration and future access of info
  - We can view many of the interactive experiences of our lives as generators of rich multimedia content

Similar concepts

- Pervasive Computing
- Ambient Intelligence
- “Everyware”
- Internet of Things (IoT)
  - “Things that think”
  - Smart dust

- An environment in which:
  - People interact with embedded (and mostly invisible) computers
  - Devices are aware of their surroundings and peers
  - Devices are able to provide services or use services from peers effectively
- Subsumes:
  - Mobile computing: many of these devices are mobile
  - Distributed systems: many devices interact

- Ambient Intelligence (AmI) refers to electronic environments that are sensitive and responsive to the presence of people
  - Embedded: many networked devices are integrated into the environment
  - Context aware: these devices can recognize users and their situational context
  - Personalized: they can be tailored to users' needs
  - Adaptive: they can change in response to users actions
  - Anticipatory: they can anticipate users' desires without conscious mediation

- The Internet of Things (IoT) is the internetworking of physical devices, vehicles, buildings, and other items
  - These “things” can communicate and exchange data
  - They can perform actions based on it
    - They often act with no direct input of the user

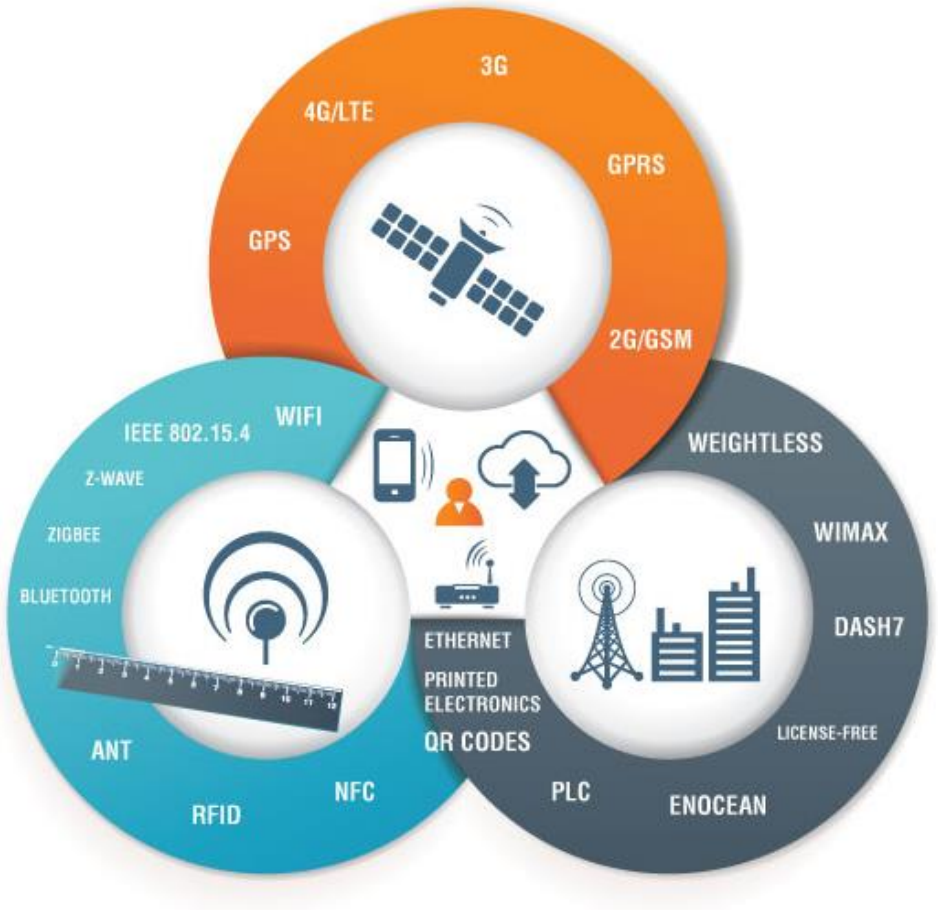
https://youtu.be/QSIPNhOiMoE

- Smart dust: tiny microelectromechanical systems (MEMS) such as sensors, robots, or other devices

Related areas

- Ubiquitous computing makes use of many technologies and techniques from other areas
  - Communications
  - Distributed computing
  - Sensor networks
  - Mobile computing
  - Wearable computing
  - Human-computer interaction
  - Data management
  - Artificial intelligence
  - Augmented reality

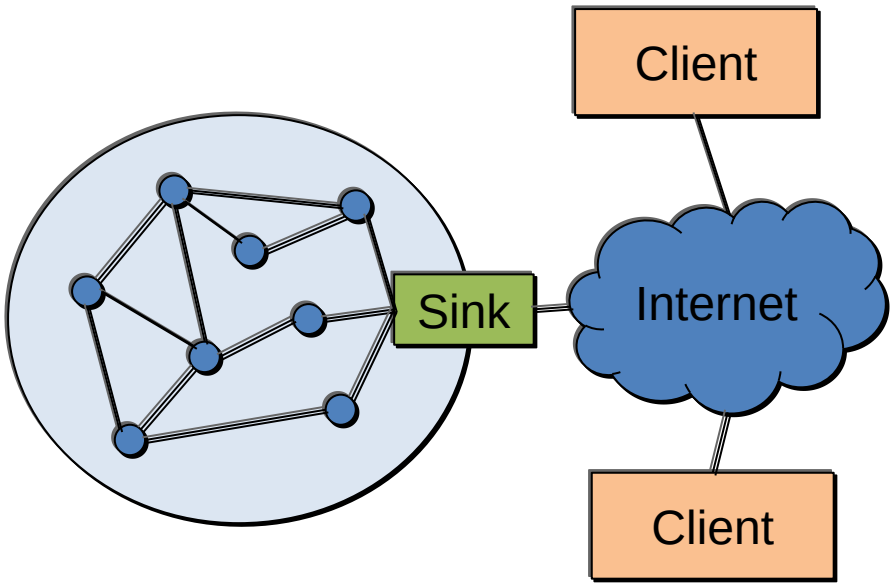
- Components in a ubiquitous system communicate
- Different needs depending on required:
  - Bandwidth
  - Range
  - Energy consumption
  - Latency



Distributed computing

- A **distributed system** is a software system in which components located on networked computers communicate and coordinate their actions by passing messages
  - The components interact with each other in order to achieve a common goal
- Some significant characteristics of distributed systems are:
  - Concurrency of components
  - Lack of a global clock
  - Independent failure of components
- Components usually communicate by means of a **middleware**

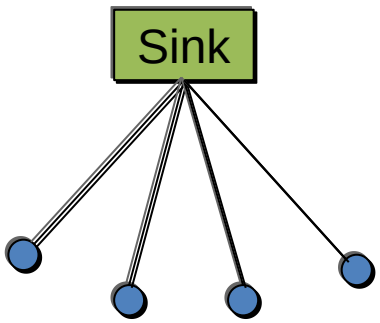
- Wireless Sensor Networks (WSNs) are networks of nodes that cooperatively sense and control the environment
  - Sometimes called Wireless sensor and actuator networks (WSANs)
- WSNs usually include
  - Sensor/actuator nodes
  - Wireless network
  - Gateways (sinks)



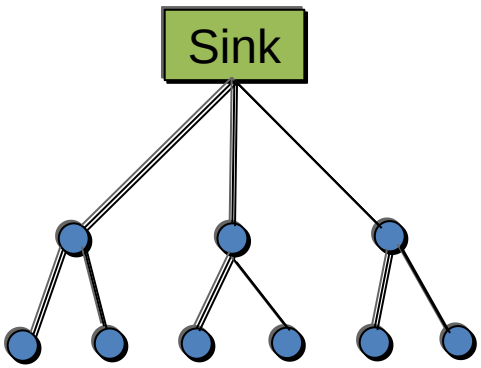
- A sensor node in a WSN has:
  - Microprocessor
    - Cheap, low-energy consumption
    - Examples: Arduino (microcontroller), TinyOS (operating system for low-power wireless devices)
  - Sensors and/or actuators
  - Communications capabilites
    - Low-energy consumption, usually short range
    - Examples: ZigBee (based on IEEE 802.15.4), Bluetooth low energy (BLE or Bluetooth Smart), Z-Wave, WiFi, WiMAX
  - Power supply
    - Energy harvesting: energy obtained from an external source (solar, kinetic, wind, thermal…)

Sensor networks: topologies

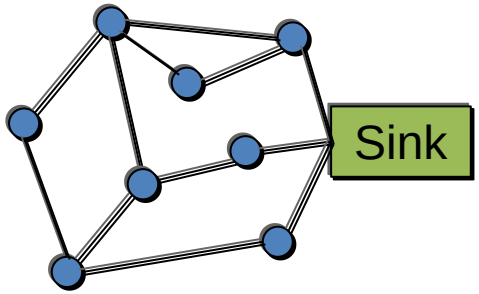
- Topology is the arrangement of a network, including its nodes and connecting lines
  - If some nodes are not connected directly to a gateway (multihop networks) routing protocols are needed
  - In some case nodes need to discovery their neighbours and create the network
  - In some case nodes can move, and the network needs to be reconfigured



Star



Tree



Mesh

Wearable computing

- Wearable computers, also known as body-borne computers or wearable are miniature electronic devices that are worn by the bearer under, with or on top of clothing
- This class of wearable technology has been developed for general or special purpose information technologies and media development
- Wearable computers are especially useful for applications that require more complex computational support than just hardware coded logics
- Features:
  - There is a constant interaction between computer and user
  - The ability to multi-task

- HCI involves the study, planning, design and uses of the interaction between people (users) and computers
  - Often regarded as the intersection of computer science, behavioral sciences, design and several other fields of study
- Ubiquitous computing demands different ways of interaction with the systems
  - Touchscreens
  - Voice recognition
  - …

Artificial Intelligence

- Artificial Intelligence (AI) is the simulation of human intelligence processes by machines, especially computer systems
- It includes:
  - Machine learning
  - Reasoning
  - Self-correcting
- Ubiquitous systems need AI for:
  - Perception (voice and image recognition)
  - Decision making
  - Adaptation

Augmented Reality

- Augmented Reality (AR) is a live, copy, view of a physical, real-world environment whose elements are augmented by computer-generated sensory input such as sound, video, graphics or GPS data
  - As a result, the technology functions by enhancing one's current perception of reality.
- With the help of advanced AR technology (e.g. adding computer vision and object recognition) the information about the real world of the user becomes interactive and digitally manipulable

Consequences of ubiquitous computing

- Advantages
  - Life made easier
  - Provides convenient access of relevant information
  - Enables cost reduction
  - Innovation in provision of services
- Issues
  - Privacy
  - Security

Universidad
Rey Juan Carlos

Tema 3: Desarrollo de Aplicaciones Híbridas

Servicios Móviles y Ubicuos

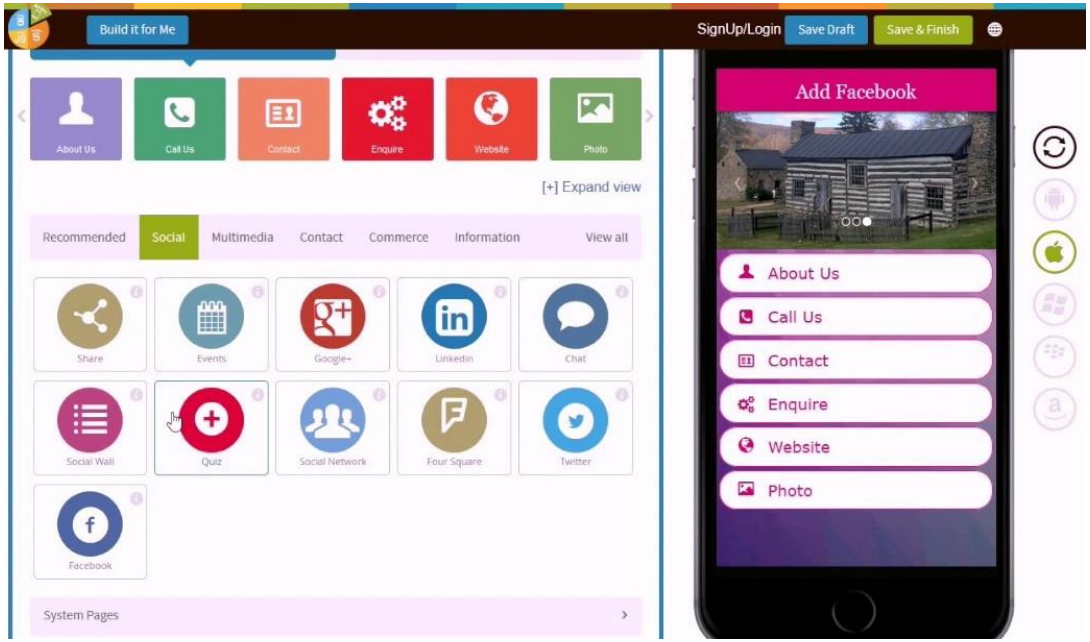
Web / Híbridas / Nativas

1. Herramientas de Desarrollo

Basadas en plantillas

appypie

appsbuilder



Basadas en plantillas. Ventajas e inconvenientes:

- En muchos casos, no es necesario saber programar.
- Facilitan en gran medida el desarrollo, publicación y mantenimiento.
- Herramientas comerciales.
- Personalización limitada.
- Funcionalidad limitada.

Frameworks de desarrollo

Frameworks de desarrollo. Ventajas e inconvenientes:

- Gratuitas (excepto servicios de pago).
- Gran personalización tanto de interfaz como de funcionalidad.
- Necesidad de conocer el framework y las tecnologías relacionadas.

Herramientas mixtas

Universidad
Rey Juan Carlos

2. Introducción a JavaScript

ECMAScript:

- Especificación para crear un lenguaje de scripting.
- Los navegadores utilizan ECMAScript para interpretar código JavaScript.
- ECMAScript establece las reglas que un lenguaje de scripting debe seguir para ser considerado ECMAScript.

https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

JavaScript:

- Lenguaje de scripting que sigue las especificaciones del ECMAScript.
- JavaScript es el lenguaje, mientras que ECMAScript es la especificación.
- Usado principalmente del lado cliente (frontend).
- Lenguaje interpretado.
- Orientado a objetos.

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Introduction#%C2%BFqu%C3%A9_es_javascript

Lenguajes relacionados:

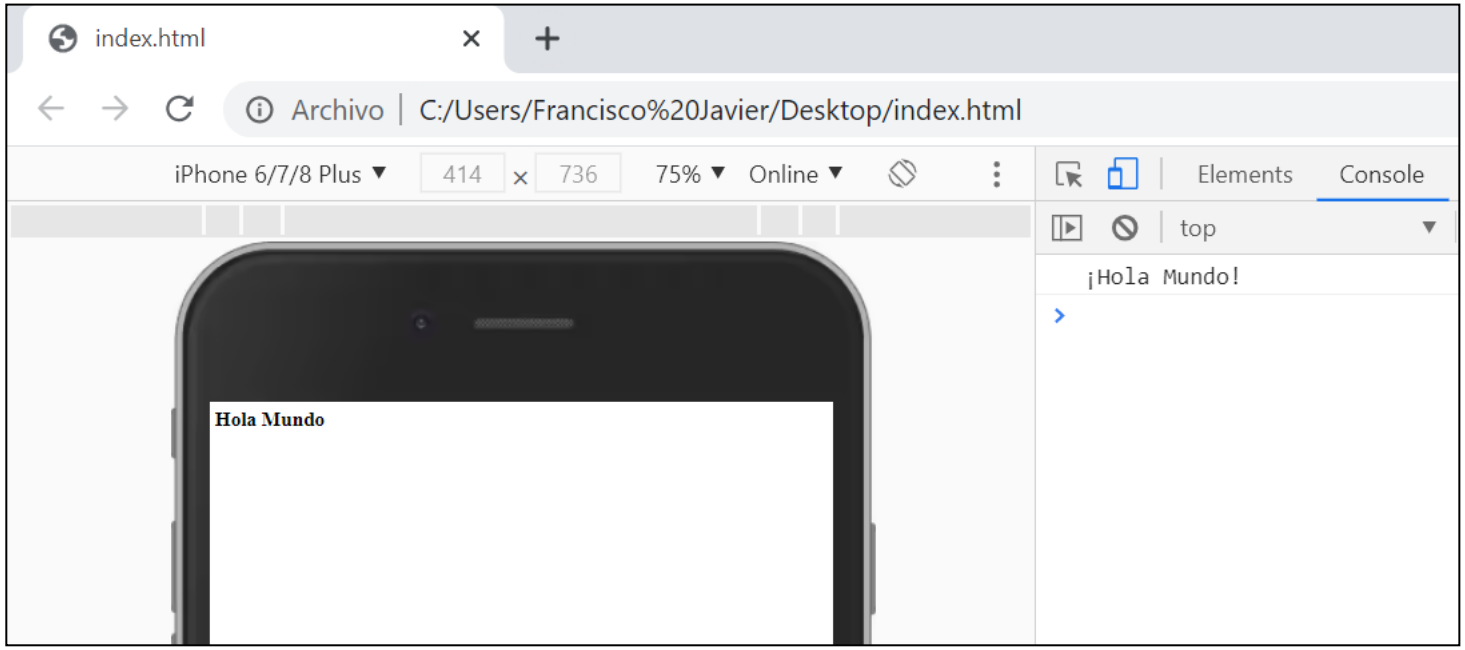
- HTML. Lenguaje de marcado. Define el contenido del documento.
- CSS. Lenguaje de diseño gráfico. Define la presentación del documento.

Ventajas y desventajas de JavaScript:

- Sintaxis sencilla.
- Mucha documentación.
- Ejecución rápida.
- Versátil.
- Multiplataforma.
- Seguridad.
- Ideado para desarrollos pequeños.

Entornos de Desarrollo

Ejecución y depuración



Google Chrome. Abrir herramientas de desarrollador: Ctrl + Mayús + I

Universidad
Rey Juan Carlos

3. Fundamentos de JavaScript

Tres formas de incluir scripts en HTML:

```html
<html>

<head>
    <title>Primera App</title>
</head>

<body>

    <h1>Hola Mundo</h1>

    <script type="text/javascript">
        // Esto es un comentario de una línea.
        /*
        Esto es
        un comentario
        de varias líneas.
        */
        alert("¡Hola Mundo!");
    </script>

</body>

</html>
```

index.html

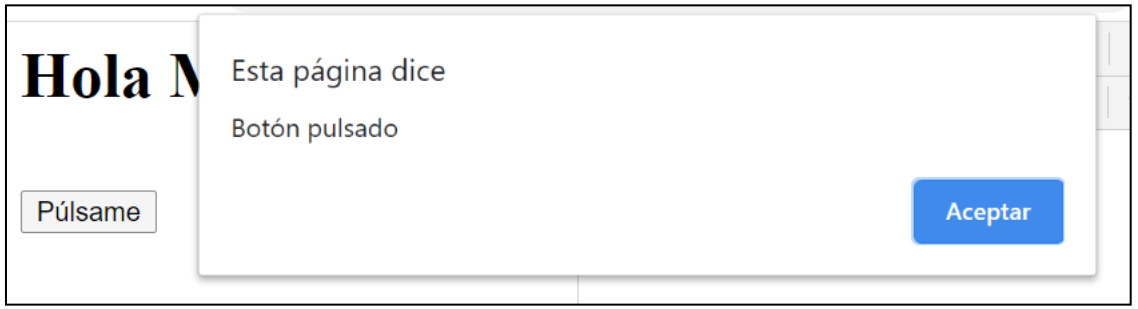
```html
<html>

<head>
    <title>Segunda App</title>
</head>

<body>

    <h1>Hola Mundo</h1>

    <script src="./miScript.js"></script>

</body>

</html>
```

miScript.js

```
alert("Hola Mundo!");
```

La tercera forma consiste en añadir eventos a elementos HTML.

Ejercicio: incluir un botón que al pulsarlo, la página nos muestre una alerta.



Fijaros cómo pueden alternarse comillas en JavaScript

Posible solución:

index.html

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Ejercicios SMU</title>
5   </head>
6   <body>
7
8       <h1>Hola mundo</h1>
9
10      <button onclick="alert('Botón pulsado');">Pulsar</button>
11
12  </body>
13  </html>
```

Escribir por consola:

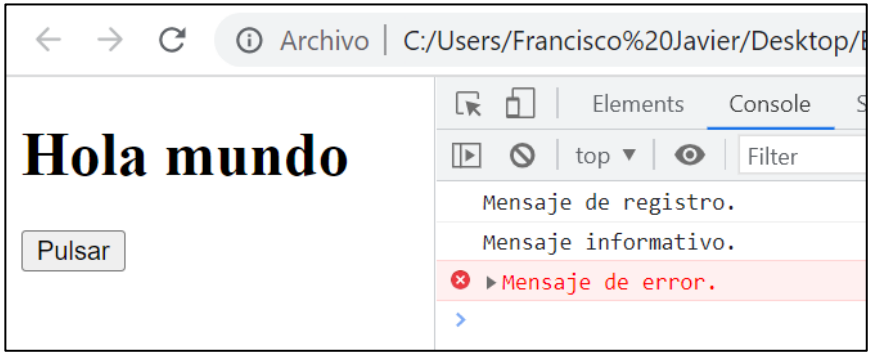
- console.log("mensaje de registro");
- console.info("mensaje informativo");
- console.error("mensaje de error");
- ...

Ejercicio: Al pulsar el botón del ejercicio anterior, hacer que se muestre por consola un mensaje de log, otro de información y otro de error.

Posible solución:

index.html

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Ejercicios SMU</title>
5   </head>
6   <body>
7
8       <h1>Hola mundo</h1>
9
10      <button onclick="console.log('Mensaje de registro.');
            console.info('Mensaje informativo.');console.error('
            Mensaje de error.');">Pulsar</button>
11
12  </body>
13  </html>
```

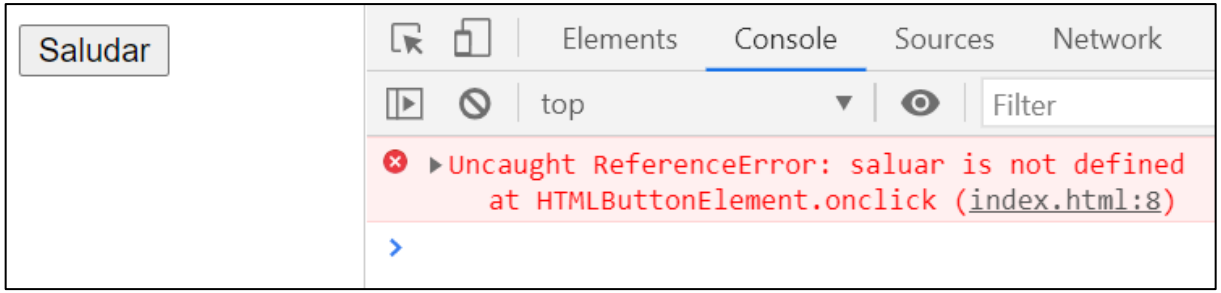


Funciones (sintaxis básica):

```html
<!DOCTYPE html>
<html>
<head>
    <title>Prueba</title>
</head>
<body>

    <button type="button" id="miBoton" onclick="saludar();">
        Saludar
    </button>

    <script type="text/javascript">

        function saludar() {
            alert("Hola Mundo");
            alert("segunda alerta");
        }

    </script>

</body>
</html>
```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Prueba</title>
</head>
<body>

    <button type="button" id="miBoton" onclick="saludar();">
        Saludar
    </button>

    <script type="text/javascript" src="./funciones.js"></script>

</body>
</html>
```

```
function saludar() {
    alert("Hola Mundo");
    alert("segunda alerta");
}
```



Ejercicio: Modificar la última tarea para realizarla mediante una función llamada *mostrarMensajes* contenida en un fichero js externo.

Posible solución:

index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejercicios SMU</title>
</head>
<body>

    <h1>Hola mundo</h1>

    <button onclick="mostrarMensajes();">Pulsar</button>

    <script src="./funciones.js"></script>

</body>
</html>
```

funciones.js

```js

function mostrarMensajes() {
    console.log('Mensaje de registro.');
    console.info('Mensaje informativo.');
    console.error('Mensaje de error.');
}
```

A partir de ahora, los eventos llamarán a funciones donde incluiremos el código.

Estas funciones estarán en ficheros externos que importaremos al final del *body* del HTML.

El objeto *document* representa el documento sobre el que estamos trabajando. Entre otras cosas, nos permite escribir contenido HTML:

- document.write("<h1>Hola Mundo</h1>");

Ejercicio: Al pulsar el botón del ejercicio anterior, hacer que cambie el contenido HTML del documento y muestre, en negrita y cursiva, el texto ***botón pulsado***.

Posible solución:

index.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Ejercicios SMU</title>
5  </head>
6  <body>
7
8      <h1>Hola mundo</h1>
9
10     <button onclick="cambiarContenidoHTML();">Pulsar</button>
11
12     <script src="./funciones.js"></script>
13
14 </body>
15 </html>
```

funciones.js

```
1
2  function cambiarContenidoHTML() {
3      document.write("<b><i>botón pulsado</i></b>");
4  }
5
6
```

Con el objeto *document*, también podemos recuperar cualquier elemento HTML…

- document.getElementById(“mi_ID”)

…y sustituir su contenido si se trata de un contenedor *div* o párrafo (*p*):

- document.getElementById("mi_ID").innerHTML = "Hola Mundo";

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Fundamentos</title>
</head>
<body>
  <h2>¿Qué podemos hacer con JavaScript?</h2>
  <p id="parrafo">JavaScript puede cambiar contenido HTML.</p>
  <button type="button"
      onclick="document.getElementById('parrafo').innerHTML = 'Hola Mundo,  desde JavaScript!! '">Pulsamé!</button>
  </body>
</html>
```

Ejercicio: Al pulsar el botón del ejercicio anterior, inyectar en un contenedor *div* (localizado justo antes del botón), una imagen pequeña.

Posible solución:

index.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Ejercicios SMU</title>
5  </head>
6  <body>
7
8      <h1>Hola mundo</h1>
9
10     <div id="contenedor_imagen"></div>
11
12     <button onclick="insertarImagen();">Pulsar</button>
13
14     <script src="./funciones.js"></script>
15
16 </body>
17 </html>
```

funciones.js

```
1
2  function insertarImagen() {
3      document.getElementById("contenedor_imagen").innerHTML = "<img src='./urjc_logo.png' width='200px'>";
4  }
5
6
```

…modificar cualquiera de sus propiedades:

- document.getElementById(“mi_ID”).src = “miImagen.png”

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Fundamentos</title>
</head>
<body>
  <h2>¿Qué podemos hacer con JavaScript?</h2>
  <p id="parrafo">JavaScript puede cambiar atributos HTML.</p>
  <button onclick="document.getElementById('imagenLuz').src='LuzEncendida.gif'">Encender luz !</button>
  <img id="imagenLuz" src="LuzApagada.gif" style="width:100px">
  <button onclick="document.getElementById('imagenLuz').src='LuzApagada.gif'">Apagar luz !</button>
</body>
</html>
```

Ejercicio: Añadir un nuevo botón al ejercicio anterior, que al pulsarlo cambie la imagen.

Posible solución:

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejercicios SMU</title>
</head>
<body>

    <h1>Hola mundo</h1>

    <div id="contenedor_imagen"></div>

    <button onclick="insertarImagen();">Pulsar</button>
    <button onclick="cambiarImagen();">Cambiar imagen</button>

    <script src="./funciones.js"></script>

</body>
</html>
```

funciones.js

```
function insertarImagen() {
    document.getElementById("contenedor_imagen").innerHTML = "<img id='logo_urjc' src='./urjc_logo.png' width='200px'>";
}

function cambiarImagen() {
    document.getElementById("logo_urjc").src = "./urjc_logo_2.png";
}
```

…modificar sus propiedades css:

- document.getElementById(“mi_ID”).style.fontSize = “35px”

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Fundamentos</title>
</head>
<body>
  <h2>¿Qué podemos hacer con JavaScript?</h2>
  <p id="parrafo">JavaScript puede cambiar estilos HTML(CSS).</p>
  <button type="button" onclick="document.getElementById('parrafo').style.fontSize='35px'">Pulsamé !!</button>
</body>
</html>
```

Ejercicio: crear un documento HTML con dos contenedores *div*.

- Cada contenedor tendrá un título y un botón.
- Por defecto, el primer *div* estará visible y el segundo oculto.
- Al pulsar el botón del primer *div*, se ocultará este contenedor y mostrará el segundo.
- Al pulsar el botón del segundo *div*, se realizará el proceso inverso.

…o incluso recuperar el valor de un campo de un formulario:

- <input type="text" id="mi_ID" placeholder="Introduce tu nombre">
- … document.getElementById("mi_ID").value;

Pero antes…

Variables, constantes y operadores:

```
1   function miFuncion() {
2
3       var nombre = "Agustín";
4       nombre = nombre + " García";
5       var edad = 20;
6       edad = edad + 1;
7       var estudiante = true;
8       const idPersona = "12345678B";
9
10      var mensaje =    "ID Persona: " + idPersona +
11                       "\nNombre: " + nombre +
12                       "\nEdad: " + edad;
13
14      if (estudiante) {
15          mensaje = mensaje + "\nEs estudiante";
16      } else {
17          mensaje = mensaje + "\nNo es estudiante";
18      }
19
20      alert(mensaje);
21
22  }
```

```
1   function miFuncion() {
2
3       let nombre = "Agustín";
4       nombre = nombre + " García";
5       let edad = 20;
6       edad = edad + 1;
7       let estudiante = true;
8       const idPersona = "12345678B";
9
10      let mensaje =    "ID Persona: " + idPersona +
11                       "\nNombre: " + nombre +
12                       "\nEdad: " + edad;
13
14      if (estudiante) {
15          mensaje = mensaje + "\nEs estudiante";
16      } else {
17          mensaje = mensaje + "\nNo es estudiante";
18      }
19
20      alert(mensaje);
21
22  }
```

Variables, constantes y operadores:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Variables, constantes y operadores:

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Variables, constantes y operadores:

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Variables, constantes y operadores:

Operator	Description
&&	logical and
||	logical or
!	logical not

Parámetros de funciones:

```
1   function miFuncion() {
2
3       let operador1 = 2;
4       let operador2 = 3;
5       suma(operador1, operador2);
6   }
7
8   function suma(valor1, valor2) {
9       alert(valor1 + valor2);
10  }
```

```
1   function miFuncion() {
2       let operador1 = 2;
3       let operador2 = 1;
4       let resultado = suma(operador1, operador2);
5       alert(resultado);
6   }
7
8   function suma(valor1, valor2) {
9       return valor1 + valor2;
10  }
```

Ejercicio: crear un pequeño formulario HTML con los siguientes elementos:

- Un título.
- Un campo para introducir el nombre.
- Un contenedor *div* sin contenido.
- Un botón de enviar.
- Al pulsar el botón, aparecerá en el contenedor div el valor “Hola, ” y el nombre introducido en el campo de texto.

Ejercicio: modificar el ejercicio anterior para que al pulsar el botón, el nombre introducido se muestre como título *h1* en un contenedor div que se hará visible, ocultando el *div* donde está el formulario.

El contenedor donde se muestra el nombre, tendrá además un botón para regresar al formulario.

Ejercicio: crear un pequeño formulario HTML con los siguientes elementos:

- Un título.
- Un campo para introducir un número.
- Un botón de enviar.
- Al pulsar el botón, se mostrará una alerta con el valor del número introducido, incrementado en una unidad.

4. Material Adicional

Arrays: en JavaScript, los arrays son colecciones de elementos del mismo tipo o tipos diferentes.

Creación de un array:

```
let a = [‘palabra1’, ‘palabra2’, ‘palabra3’]; // Elementos del mismo tipo.
let b = [‘palabra1’, 23, true]; // Elementos de tipos distintos.
let c = []; // Array vacío.
let d = new Array() // Similar a la línea anterior.
```

Acceder a un elemento del array:

```
console.log(a[1]); // Imprime por consola palabra2.
```

Insertar elementos en la última posición:

a.push('palabra4');

Eliminar (y devolver) el elemento de la última posición:

a.pop();

Eliminar elementos intermedios:

a.splice(p, n) // p = posición del elemento, n = número de elementos a eliminar.

a.splice(0, 2) // Empezando en la posición 0, eliminar 2 elementos.

Insertar elementos intermedios:

a.splice(p, 0, valor) // En la posición p, eliminamos 0 elementos e insertamos valor.

// También nos sirve para sustituir elementos si modificamos el 0.

También podemos usar identificadores para las posiciones:

```
a['nombre'] = 'Pedro';
a['edad'] = 48;
console.log(a['nombre']);
console.log(a.nombre); // Funcionamiento igual que en la línea anterior.
```

Tres formas para recorrer arrays:

```
for (let i = 0; i < a.length; i++) {
    console.log(a[i]);
}

for (let elemento of a) {
    console.log(elemento);
}

a.forEach(function (elemento, indice) {
    console.log(elemento);
    console.log(indice);
});
```

Objetos: conjunto de cero o más parejas "nombre: valor".

```
let avion= {
        marca: "Boeing",
        modelo: "747",
        npasajeros: "450"
};
```

Acceso a sus elementos:

```
console.log(avion.modelo);
```

Comparar dos objetos completos:

```
if (JSON.stringify(objeto1) == JSON.stringify(objeto2)) {
        …
}
```

Local Storage:

Guardar datos en la memoria local del navegador:

```
localStorage.setItem('id', JSON.stringify(miObjeto));
```

Cargar datos de la memoria local del navegador:

```
let miObjeto = JSON.parse(localStorage.getItem('id');
```

Clases y objetos: Estructura de una clase.

```
class MiClase {

        // PROPIEDADES:
        propiedad1;
        propiedad2 = [];
        propiedad3 = 'mi Valor';

        // CONSTRUCTORES:
        constructor() {
                ...
        }

        // MÉTODOS:
        accion1(parametro1) {
                this.propiedad1 = parametro1;
        }

}
```

Clases y objetos:

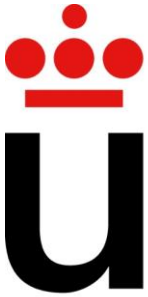
Para hacer referencia a una propiedad de la clase desde dentro de la misma, siempre utilizamos la palabra reservada *this*: this.propiedad1

Creación y uso de objetos:

```
let miObjeto = new MiClase();
miObjeto.propiedad2.push('hola mundo');
miObjeto.accion1(24);
```

ISe

Ingeniería de Servicios



Universidad Rey Juan Carlos

Servicios Móviles y Ubicuos

4.1 Aplicaciones nativas en Android: introducción

Recursos

- Entorno de desarrollo para Android
  - Android Studio
- SDK de Java
- SDK de Android
  - Se instala con Android Studio
- Teléfono o emulador para ejecutar la app
  - Android Studio permite crear y manejar emuladores
- Tutoriales y referencia
  - https://developer.android.com

Ejercicio: Hola Mundo

- Crear y ejecutar en un emulador el proyecto Hola Mundo
  - https://developer.android.com/training/basics/firstapp/creating-project.html
  - https://developer.android.com/training/basics/firstapp/running-app.html

Estructura de un proyecto

- Un proyecto Android consta de una serie de recursos
  - Clases de Java → código
  - Ficheros XML → configuración
  - Scripts de construcción
  - Otros (imágenes…)
- Se organiza en dos carpetas principales
  - *app* → la aplicación en sí
  - *Gradle Scripts* → scripts de construcción del proyecto

Estructura de un proyecto (2)

- La carpeta app tiene 3 carpetas:
  - *app/manifests* → contiene el manifiesto de la aplicación (*AndroidManifest.xml*)
    - El manifiesto describe las propiedades principales de la app y define sus componentes
  - *app/java* → contiene las clases de Java que componen la app organizadas en paquetes
  - *app/res* → contiene recursos de la aplicación
    - *app/res/values* → valores simples usados por la aplicación (strings, colores…)
    - *app/res/layout* → layout de las pantallas
    - …

Recursos

- La carpeta *app/res* permite externalizar recursos de la aplicación, lo que tiene ventajas:
  - La gestión de los recursos se mantiene independiente del código
  - Puede haber alternativas de un mismo recurso y se utilizará la apropiado en cada caso
    - En función del tamaño y resolución de la pantalla
    - En función del idioma
    - …

https://developer.android.com/guide/topics/resources/providing-resources.html

Recursos (2)

- Para identificar los recursos necesitamos un identificador que viene dado por:
  - Tipo del recurso
  - Nombre del recurso
- Podemos acceder a un recurso:
  - Desde el código Java, mediante la clase R y el objeto que permite acceder a los recursos de la app (de clase Resources)
    - *R.tipo.nombre* nos da el identificador (ejemplo: *R.string.texto*)
    - getResources() nos da el objeto para acceder a los recursos, con diferentes funciones según su tipo

    Ejemplo: getResources().getString(R.string.texto);
  - Desde los ficheros de configuración XML
    - *@tipo/nombre* (ejemplo: *@string/texto*)

https://developer.android.com/guide/topics/resources/accessing-resources.html

Ejercicios

- Modificar los colores de la aplicación de ‘Hola Mundo’
- Crear un nuevo recurso de tipo string y hacer que se muestre este nuevo mensaje en lugar de ‘Hello World!’ como texto principal
- Crear un nuevo recurso de tipo string y mostrarlo este mensaje por la consola de ejecución utilizando la clase Log

  https://developer.android.com/studio/debug/am-logcat.html

Actividades

- Las actividades son los componentes principales de una app en Android
  - Clases que heredan de *android.app.Activity* o algunas de sus subclases
- Una actividad es algo que el usuario puede hacer
  - Típicamente una pantalla
- Cuando la actividad tiene un interfaz se puede asociar su layout con *setContentView*
  - El layout es un tipo de recurso

```
setContentView(R.layout.activity_main);
```

Ciclo de vida de una actividad

- Una actividad puede estar en diferentes estados:
  - Reanudada o en ejecución: en primer plano
  - Pausada: hay otra actividad en primer plano pero esta todavía es visible
  - Detenida
- La clase *Activity* proporciona callbacks que se ejecutan en los cambios de estado
  - Por ejemplo para almacenar información que haya que recuperar cuando la actividad vuelve a ejecutarse

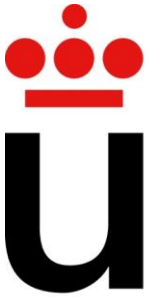
https://developer.android.com/guide/components/activities.html#Lifecycle

Ejercicio

- Implementar los callbacks de los distintos cambios de estado del ciclo de vida, y escribir un log para cada uno de ellos
- Observar que ocurre cuando:
  - Se gira la pantalla del emulador
  - Se pulsa el botón de volver
  - Se reanuda la actividad desde la lista de tareas
  - Se minimiza la aplicación
  - Se reanuda de nuevo

ISe

Ingeniería de Servicios



Universidad Rey Juan Carlos

Servicios Móviles y Ubicuos

4.2 Aplicaciones nativas en Android: interfaz de usuario

Interfaces de usuario

- Las interfaces de usuario en Android se componen fundamentalmente de dos tipos de elementos
  - Controles, que heredan de la clase *View*
    - Elementos individuales: texto, botones, imágenes…
  - Contenedores, que heredan de la clase *ViewGroup*
    - Sirven para agrupar controles y controlar su disposición en la pantalla
- Todo esto puede definirse en los XML de recursos de tipo *layout*

Diseños (layouts)

- El elemento principal de un diseño debe ser un contenedor (hereda de *ViewGroup*)
  - Dentro de él puede haber controles y otros contenedores
- Hay que tener en cuenta que los dispositivos móviles pueden tener diferentes tamaños, resoluciones y orientaciones
  - Es importante que la interfaz sea adecuada para todas las posibilidades
  - Es posible crear diferentes versiones de un diseño para que se muestre la que se adecúe a las características del dispositivo

Contenedores

- Hay diferentes tipos de contenedores que permiten organizar sus contenidos de distintas formas
  - *FrameLayout* → contiene un único elemento
  - *LinearLayout* → posiciona los elementos en columna (vertical) o en fila (horizontal)
  - *RelativeLayout* → cada elemento se posiciona en relación al contenedor o a otro elemento
  - *ConstraintLayout* → similar a *RelativeLayout* pero más complejo, se pueden establecer restricciones respecto al posicionamiento de cada elemento
  - …

Controles

- Los controles son objetos pertenecientes a diversas clases, todas ellas descendientes de *View*
- Con estos controles se pueden hacer varias cosas:
  - Responder a eventos (por ejemplo, cuando el usuario pulsa un botón)
  - Leer valores introducidos o seleccionados por el usuario
  - Modificar propiedades (por ejemplo, el texto que muestran)

Ejercicio

- Crear una actividad que muestre un formulario con varios controles que permitan introducir información sobre un usuario
  - Controles de texto: nombre, apellidos…
  - Controles de selección entre opciones: sexo…
  - Botones para resetear y enviar el formulario

https://developer.android.com/training/basics/firstapp/building-ui

Eventos

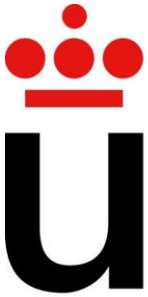
- Hay varias maneras de responder a los eventos que el usuario genera al interactuar con los controles
- La más sencilla es mediante el atributo *android:onClick* de la clase *View*
  - Se especifica el nombre del método que se llamará cuando ocurra el evento
  - Debe ser miembro de la clase de la actividad
  - Debe recibir un parámetro de clase View (el control que generó el evento)

Ejercicios

- Hacer que al pulsar el botón de enviar del formulario anterior se genere un log
- Generar un aviso (clase *Toast*) en lugar del log

https://developer.android.com/guide/topics/ui/notifiers/toasts.html

- Incluir en el aviso la información introducida en los otros controles del formulario
  - Para obtener el objeto correspondiente al control usar la función *findViewById(…)*
- Hacer que al pulsar el botón de resetear se borre el contenido de los controles



Universidad Rey Juan Carlos

Servicios Móviles y Ubicuos

4.3 Aplicaciones nativas en Android: intents y almacenamiento

Intents

- Una Intent es un objeto de acción que puede solicitar una acción de otro componente de la aplicación
- En particular se puede usar para solicitar el inicio de una actividad
  - Se puede pasar información para que la reciba la otra actividad
  - También se puede iniciar una actividad para que devuelva un resultado y trabajar después con él

https://developer.android.com/training/basics/firstapp/starting-activity.html

Almacenamiento

- Opciones principales para guardar datos:
  - Preferencias
    - De una actividad
    - Compartidas entre las actividades de una aplicación
  - Ficheros
    - Almacenamiento interno del teléfono
    - Almacenamiento externo (tarjeta SD o similar)
  - Bases de datos
    - Mediante SQLite, gestor de base de datos autónomo

https://developer.android.com/guide/topics/data/data-storage.html

Ejercicios

- Crear una aplicación con dos actividades:
  - Una con un cuadro de texto y un botón
  - Al pulsar el botón se lanzará la segunda actividad, que mostrará un texto que incluya el valor introducido en el cuadro de texto anterior
- En el formulario creado en el tema anterior:
  - Hacer que al pulsar un botón de salvar se almacenen los datos introducidos en las preferencias de la aplicación
  - Hacer que al cargar la actividad se inicialice el formulario con los datos guardados