Índice

[1 Git 3](#_Toc425428933)

[INSTALACIÓN 3](#_Toc425428934)

[CREAR REPOSITORIO 3](#_Toc425428935)

[TRABAJANDO CON GIT 3](#_Toc425428936)

[COMANDOS DE GIT 4](#_Toc425428937)

[RESOLVER CONFLITOS EN GIT: 5](#_Toc425428938)

[CASO 1 5](#_Toc425428939)

[Fichero local modificado incorrectamente 5](#_Toc425428940)

[CASO 2 6](#_Toc425428941)

[REPOSITORIO DE NEO: 6](#_Toc425428942)

[PASOS EN NEO CON GIT 6](#_Toc425428943)

[RAMAS EN GIT (branches) 6](#_Toc425428944)

[CONFIGURAR PATH 7](#_Toc425428945)

[Variables de entorno 7](#_Toc425428946)

[2-JADE 8](#_Toc425428947)

[JADE EN SUBLIME 8](#_Toc425428948)

[ETIQUETAS BÁSICAS EN JADE 8](#_Toc425428949)

[EXTEND: 9](#_Toc425428950)

[JADE BUCLES 9](#_Toc425428951)

[Each 10](#_Toc425428952)

[While 10](#_Toc425428953)

[Arrays 10](#_Toc425428954)

[Interpolación 11](#_Toc425428955)

[3-JAVASCRIPT 11](#_Toc425428956)

[Funciones 11](#_Toc425428957)

[OPERADORES 11](#_Toc425428958)

[NEGACIÓN: 12](#_Toc425428959)

[AND 12](#_Toc425428960)

[OR 13](#_Toc425428961)

[ARRAYS 13](#_Toc425428962)

[Relacionales 13](#_Toc425428963)

[ESTRUCTURAS DE CONTROL DE FLUJO 14](#_Toc425428964)

[FUNCIONES 14](#_Toc425428965)

[BUCLES FOR IF: 15](#_Toc425428966)

[4-SASS 16](#_Toc425428967)

[Variables 16](#_Toc425428968)

[VARIABLES REGLA DEFAULT 16](#_Toc425428969)

[MIXINS 16](#_Toc425428970)

[EXTEND 17](#_Toc425428971)

[CONTENT 17](#_Toc425428972)

[Placeholder selectors 17](#_Toc425428973)

[INTERPOLACIÓN 18](#_Toc425428974)

[FUNCIONES EN SASS 18](#_Toc425428975)

[MEDIA QUERIES 19](#_Toc425428976)

[COMPILAR EN CONSOLA: 19](#_Toc425428977)

[ORGANIZACIÓN 20](#_Toc425428978)

[5-COMPASS 21](#_Toc425428979)

[6-ESTRUCTURA DE NEO 21](#_Toc425428980)

[7-GRID 22](#_Toc425428981)

# 1 Git

## INSTALACIÓN

1. INSTALAR EL MYGIT, DESCARGAR De <http://msysgit.github.io/>
2. A LA HORA DE INSTALAR, SELECCIONAR “**RunGitfromthe Windows CommandPrompt”**. Y en la opción “Configuring the line ending conversions”, dejar la opción “**Checkout Windows-style,**commit Unix-style line endings” (seleccionada por defecto)
3. Para comprobar si va, abrir consola (cmd en Inicio) escribir git –version
4. Si no va puede ser cosa del Path de Windows que no lleva a git (ahí acaba mi conocimiento del tema ver <http://guides.beanstalkapp.com/version-control/git-on-windows.html> o <http://blog.countableset.ch/2012/06/07/adding-git-to-windows-7-path/>
5. Git GUI y GIT Bash son las formas fácil y friki de usar GIT. No hace falta abrirlas, mejor usar la consola de Windows para trabajar con GIT.
6. Crear cuenta de GITHUB https://github.com/
   1. La mía es username guiomarv/nayade1984
7. Configurar GIT con la consola
   1. Gitconfig -- global user.name “nombre de usuario”
   2. Gitconfig -- global user.mail “email al que iran los cambios de git”

## CREAR REPOSITORIO

1. Ir a la carpeta del reposition (en local) en la consola y ejecutar **git init** (crea repositorio en local)
2. Ir a github.com a nuestra cuenta y crear ahí el repositorio web. Este repositorio estará vacío por supuesto.
3. Luego en la consola, le daremos el origin del repositorio local indicándolo así:

*git remote add origin* <https://github.com/guiomarv/junior.git>

1. Y luego subimos todos los archives que tenemos creados el local con un push (después de hacer un commit claro)

git push -u origin master

1. Si el repositorio ya está en git y lo queremos bajar a local, hay queclonarlo:
   1. Vamos en la consola donde queremos que se almacene el repositorio (se creará allí una carpeta)
   2. Git clone https://github.com/guiomarv/junior.git

## TRABAJANDO CON GIT

Comprobamos el estado del repositorio con *git status* (estando Dentro de la carpeta del repositorio con la consola)

Si modificamos algo en el directorio local hay que añadirlo al index (Stage) una vez que esté bien, para poder incluirlo posteriormente en un commit. El add indica que hay cambios pues git no está active como sí lo hacía subersion con el comando watch.

Una vez que modifiquemos algo se puede comprobar con *git status.*

Working directory Staged Area HEAD Repositorio

(commits)

### COMANDOS DE GIT

#### Git status

Muestra los cambios realizados en nuestro repositorio local. No se recogen los cambios realizados en el repositorio remote, solo lo que hayamos hecho nosotros. Éstos cambios pasarían a estar en el directorio de trabajo. Para eliminarlos, haríamos *git reset CARPETA o Archivo*

#### Git pull

Incorpora los cambios del servidor remoto a la rama actual. Un git pull es un *git fetch* más un *git merge FETCH\_HEAD.* Eso significa que queda el commit anterior en la rama local (cosa que no ocurriría con *git fetch* + *git rebase* donde se borraría nuestro commit local )Usar mejor *git fetch* y luego *git rebase.*

#### Git commit

Guarda la información de los cambios nueva previamente añadidos con *git add* junto con un comentario del usuario. **No** los sube al Repositorio Común. Cada commit queda documentado en el index de git pudiendo navegarse por él usando *git checkout*.

#### Git fetch

Descarga los últimos cambios subidos al repositorio. Hacer mejor *git fetch* que git pull, porque así permite coger los últimos cambios sin añadir todo y **siguendo en el mismo commit que anteriormente.** Después de *git fetch* haríamos un *git rebase,* que lo que hace es eliminar el commit anterior para limpiar la línea de trabajo.

#### Git rebase

Después de descargar los últimos cambios, hace un merge con los archivos nuevos y hace desaparecer el commit anterior limpiando la línea de commits innecesarios. Se genera un commit final que incluye los cambios locales con los remotos.

REBASE MERGE

Repo remoto commits locales Repo remoto commits locales

Hace que el commit anterior desaparezca se mantiene el commit anterior

#### Git merge

Une los cambios propios con los del repositiorio remoto, **y no borra el último commit en local**.

Se genera un commit final que incluye los cambios locales con los remotos.

#### Git Reset

El comando reset mueve el branch actual a otra posición, y opcionalmente actualiza el stage y el directorio de trabajo. Además se usa para copiar archivos desde los commits a stage sin tocar el directorio de trabajo.

Para machacar todos los cambios locales con lo que está en remote, usaremos el comando

*Git reset –hard@{u}*

#### Git add

Añadir archivos modificados en local para comitearlos más tarde. Existe la opción *git add . –A* que añade TODOS los cambios hechos en local, pero major ir seleccionando para evitar errors. Para añadir todo lo de una carpeta poner git add *–all CARPETA*

#### Git log

Permite ver el historial de commits realizados. La primera línea es un hash único que identifica al commit, seguida del autor, la fecha en que se hizo y el comentario que acompañó al commit.git log nos mostrará todas las entradas del log. Para ver sólo un número determinado sólo tienes que añadirle el parámetro -NUMERO, donde NUMERO es el número de entradas que quieres ver, por ejemplo: *git log -4*

### RESOLVER CONFLITOS EN GIT:

Git indica cúales el archivo que da el conflicto y sobre escribe en las líneas que dan conflicto la version de cada usuario o rama. Hay que ir a las líneas en concreto que dan conflicto, escoger una version y añadirla para luego hacer comit y push normalmente.

### CASO 1

### Fichero local modificado incorrectamente

* añadido al Staged (con add)
  + GIT RESET: Devuelve el Stage area al estado del ultimo commit(copia del HEAD archivos al StageArea)
  + GIT CHECHKOUT: Copia del Stage area archivos a la carpeta local
* Sin añadir al Staged (en local )
  + GIT CHECHKOUT: Copia del Stage area archivos a la carpeta local

Git checkout ruta/al/archivo/delquese/descartancambios.scss

O con Git checkout origin master: directamente se copiará del repositorio los archivos y machacará los cambios locales.

#### Guardar todos los cambios de una carpeta:

Usar el comando git add –all Carpeta-que-sea

### CASO 2

*The file will have its original line endings in your working directory. warning: LF will be replaced by CRLF in Gemfile.*

In Unix systems the end of a line is represented with a line feed (LF). In windows a line is represented with a carriage return (CR) and a line feed (LF) thus (CRLF). when you get code from git that was uploaded from a unix system they will only have a LF. It's nothing to worry about.

## REPOSITORIO DE NEO:

Ahora mismo mi repositorio está en: C:\DatosSinRespaldo\gvaldes\neo

Para verlo, hay que compilarlo, para ello abrir la consola, ir a este repositorio y escribir **gulp** (instrucción de compass para compilar)

En el navegador ir a <http://localhost:1337/index.html>

Intalar gulp:

* Si no está instalado, lo pedirá, para ello escribir **“**‘npm install -g gulp’ para instar gulp en el ordeandor globalmente. (instalar en C)
* Después habrá que instalar gulp en el diretorio local . Ir al directorio local y **escribir npm install gulp.**
* Habrá que instalar gulp por cada repositorio creado.
* Npm installl instalará todos los paquetes que necesita un repositorio (cuando el repositorio ya tenga los paquetes de configuración creados)

## PASOS EN NEO CON GIT

Para ver la página hay que compilarla para lo cual hacer git gulp con la consola de git.

1)GIT STATUS: ver los cambios guardados. (cambios que hemos hecho en local)

2)GIT ADD: añadimos los cambios, indicamos a git qué se ha camgbiado para poder subirlos después.

3)GIT COMMIT: sube los cambios al repositorio local. Sólo se sube lo que se haya añadido con git add.

4)GIT PUSH: (Sólo cuando estemos seguros de que lo que tenemos en nuestro repositorio funciona y no va a romper nada) Subimos los cambios al repositior Remoto.

## RAMAS EN GIT (branches)

Se pueden ir creando ramas que son copias paralelas del directorio principal para ir haciendo pruebas. Las ramasno se crean dentro del directorio principal, están en la misma carpeta que los commits. La rama principal (tronco) se llamará por defecto master. El repositorio remote se llama como alias **origin**.

* Crear rama: -**git branch NuevaRama**
* Trabajar en la nuevarama – **git checkout NuevaRama**
* Subir la rama al Directorio remoto: **git push origin NuevaRama**
* Git branch Nos muestra las ramas que hay actualmente del repositorio

Se cambia de rama con **git checkout NuevaRama o git checkout master**

**Los archivos en la carpeta local se actualizarán en consecuencia a la rama que hayamos escogido para trabajar.** El resto del proceso de trabajo será igual menos que habrá que tener en cuenta en qué rama estamos trabajando.

Las diferencias entre una rama y la master las muestra con **git diff master NuevaRama**

**Git merge branch:** Cuando hayamos acabado de desarrollar lo que sea en la nueva Rama podemos actualizar esos cambios con los de la rama principal. Si se han tocado las mismas líneas habrá un conflicto.

## CONFIGURAR PATH

Si no reconoce el comando “node” o “git” o “jade” (tanto el sublime como la consola) puede ser que no esté correctamente configurado el path de las variables de entorno.

### Variables de entorno

Las variables de entorno forman un conjunto de valores dinámicos que normalmente afectan el comportamiento de los procesos en una computadora.

Para mostrar el contenido de la variable de entorno PATH (que mantiene la [ruta](http://es.wikipedia.org/wiki/Ruta_(inform%C3%A1tica)) de búsqueda de programas en el sistema) debe escribirse en la línea de comandos

**echo %PATH%**

Configurar PATH

:Ir a Equipo/botón derecho/ propiedades del sistema/Configuración avanzada del sistema/Variables de entorno. Hay que añadir donde esté instalado el git, el npm (de node js para que monte jade) y el Node.js, separando cada ruta por **;**

Por ejemplo, para jade es : C:\Users\gvaldes\AppData\Roaming\npm

Para git C:\Program Files (x86)\Git\bin;

Asi que será PATH

RUTA: C:\Program Files\nodejs; C:\Program Files (x86)\Git\bin;C:\Users\gvaldes\AppData\Roaming\npm

Luego hay que reiniciar el equipo

# 2-JADE

## JADE EN SUBLIME

Para instalar jade hay que tener i nstalado node.js que es el programa que con la función nmpinstallal instalará Jade.

Instalar jade en el ordenador, abrir la consolay :npminstall jade --global

(seinstaladesde jade.js, hay quetener jade.js instalado antes)

Bajarse el **Packcage control** (<https://packagecontrol.io/installation>)

1. en sublime, ir a preferencia/browsepackage/ buscar en la ruta y subir un nivel, en la carpeta “Installed packages” copiar lo que noshemosbajado y reiniciar el sublime.
2. Luegoir a preferencias/package control/installpackage/y seleccionar jade. Volver a reiniciar
3. En sublime tenemos que configurar para que se monte el jade. Ir a tools/buildsystem/new buildsystem y copiar el código: configurandolasrutas de lascarpetas antes)

{

"cmd": ["cmd", "/c", "jade", "$file", "--out", "${file\_path/myfolder/./}"],

"working\_dir": "${file\_path/myfolder/./",

"selector": "source.jade"

}

(ponerrutaabsoluta) ej

{

"cmd": ["cmd", "/c", "jade", "$file", "--out", "${C:/MAMP/htdocs/./}"],

"working\_dir": "${C:/MAMP/htdocs/./",

"selector": "source.jade"

}

1. Ponertodos los archivos .jade en la ruta antes marcada, y darles a ctr+B para que se contruyan. Es possible que haya que reiniciar el sublime.
2. Para editarlos están en : C:\Users\gvaldes\AppData\Roaming\Sublime Text 2\Packages\User

## ETIQUETAS BÁSICAS EN JADE

Links en jade: **a(href="#") ACEPTAR**

Imágenes en jade: **img(src="images/gata.png")**

Párrafo: p

Varias líneas:

p

| Text can be included in a number of

| different ways.

p.

This way is shortest if you need big

blocks of text spanning multiple

lines.

Enlaces :

a(href="#") galería

## EXTEND:

Extend hace que una plantilla se pueda “extender” en otra:

//- layout.jade

doctype html

html

head

block title

title Default title

body

block content

en otra plantilla:

//- index.jade

extends ./layout.jade

block title

title Article Title

block content

h1 My Article

## JADE BUCLES

En jade se pueden hacer de dos tipos: each y while

### Each

Usados para listados, por ejemplo.

No es necesario ir pintando cada li con su nombre, basta con usar un bucle each y asociar varios valores :

ul.listado

each val in ["principal", 2, 3, 4, 5]

li= val

Si van anidados, entonces hay que pintar el anidamiento y luego darle el valor al elemento que contendrá el título. Por ejemplo, para la clásica lista de enlaces y spans:

ul

each val in ["a", "b", "c", "d", "e"]

li

a(href="#")

span= val

### While

Se puede usar para crear un bucle que se repetirá hasta que se cumpla la condición.

Por ejemplo:

- var n = 0

ul

while n < 4

li= n++

La variable se tiene que declarar a la altura del ul, es decir fuera.

### Arrays

Se pueden definir arrays igual que en js. Hay un parámetro llamado **index** que nos dá el número de elementos del array (empezando en 0) Puede servir para asignar diferentes valores dentro de cada elemento del bucle each. (ver ejemplo)

Para usarlo con un bucle each, hay que poner:

each val**, index in** ["home", "varios", "galería", "fotos", "videogalería"]

Ejemplo:

ul

each val, **index** in ["home", "varios", "galería", "fotos", "videogalería"]

li

- var links= ["1", "2", "3", "4", "5"]

- var resultado= links**[index]**

a(href="#{resultado}")

span= val

usamos el valor de index para recorrer el segundo array que nos dá el valor del href de los enlaces que van en cada li.

### Interpolación

Podemos sustituir un texto o un número por una variable dentro del html, o de los parámetros del mismo (como el href de un enlace)

- **var name**="Joan"

p holaa **#{name}**

y del ejemplo anterior:

**a(href="#{resultado}")**

# 3-JAVASCRIPT

Funciones

Las funciones.

Variables: las variables son parámetros a los quepodemosdarvaloresdiferentes.

Si una function trabaja con variables, hasta queéstas no tengan un valor asignado(true) no harán nada. El valor se puede asignar tanto en el jscomo en el html encerrado en un <script>.Ver index1.html.

1. Primero lee las variables, si no tienen valor, no hace nada
2. Realiza la función

Con la palabra clave: var. Porejemplo, var x = 42. Estasintaxis se puedeusarparadeclarar variables locales y globales.

Simplemente asignándole un valor. Porejemplo, x = 42**. Esto siempre declara una variable global**

Las variables pueden tener un valor o no. Si el valor no está definido, será undefined.

El valor undefined se comporta como false cuando es usado en un contexto lógico [boolean].

Las variables pueden ser de tres tipos: numeros, strings (cadenas de texto encerrado entre comillas o de tipo boleano (true o false)

OPERADORES:

* **++**incrementa valores numéricos en +1
* **--** disminuye valores numéricos en -1

El resultado de ejecutar el script anterior es el mismoquecuando se utiliza el operador ++numero, por lo quepuedeparecerqueesequivalenteindicar el operador ++ delante o detrásdelidentificador de la variable. Sin embargo, el siguiente ejemplomuestrasusdiferencias:

var numero1 = 5;

var numero2 = 2;

numero3 = numero1++ + numero2;

// numero3 = 7, numero1 = 6

var numero1 = 5;

var numero2 = 2;

numero3 = ++numero1 + numero2;

// numero3 = 8, numero1 = 6

**Si el operador ++ se indicacomoprefijo del identificador de la variable, su valor se incrementa antes de realizar cualquier otra operación. Si el operador ++ se indicacomosufijo del identificador de la variable, su valor se incrementadespués de ejecutar la sentencia en la queaparece.**

Portanto, en la instrucción numero3 = numero1++ + numero2;, el valor de numero1 se incrementadespués de realizar la operación (primero se suma y numero3 vale 7, después se incrementa el valor de numero1 y vale 6). Sin embargo, en la instrucción numero3 = ++numero1 + numero2;, en primer lugar se incrementa el valor de numero1 y después se realiza la suma (primero se incrementa numero1 y vale 6, después se realiza la suma y numero3 vale 8).

## NEGACIÓN:

Operadorqueconvierteun valor en su contrario. Sólo funciona con valores boleanos false, true

Se escribe **!variable**

Imaginemos

Var visible=true;

Var !visible=false;

### AND

La operaciónlógica AND obtienesuresultadocombinando dos valoresbooleanos. El operador se indicamediante el símbolo&& y **suresultadosolamentees true si los dos operandos son true**: (LOS COMBINA, NO LOS SUMA)

Sólo se haráverdadsitenemoslas dos variables.

var valor1 = true;

var valor2 = false;

resultado = valor1 && valor2; // resultado = false

valor1 = true;

valor2 = true;

resultado = valor1 && valor2; // resultado = true

Cuando no son valoresboleanos hay queconvertirlos. En cualquiercaso:

Si la variable contieneunnúmero, se transforma en false si vale 0 y en true paracualquierotronúmero (positivo o negativo, decimal o entero).

Si la variable contieneunacadena de texto, se transforma en false si la cadenaesvacía ("") y en true en cualquierotrocaso.

### OR

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es true si alguno de los dos operandos es true:

Ejercicio index 3.html

### ARRAYS

Se pueden agrupar todas las variables relacionadas en unacolección de variables o array. El ejemplo anterior se puederehacer de la siguiente forma:

vardias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

Ahora, una única variable llamada diasalmacenatodos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres[ y ] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

Varnombre\_array = [valor1, valor2, ..., valorN];

Una vez definido un array, es muy sencillo acceder a cada uno de suselementos. Cada elemento se accede indicando suposición dentro del array. La únicacomplicación, queesresponsable de muchoserrorescuando se empieza a programar, es**quelasposiciones de los elementosempiezan a contarse en el 0 y no en el 1:**

VardiaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"

varotroDia = dias[5]; // otroDia = "Sábado"

(ver index2.html)

### Relacionales

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

var numero1 = 3;

var numero2 = 5;

resultado = numero1 > numero2; // resultado = false

resultado = numero1 < numero2; // resultado = true

numero1 = 5;

numero2 = 5;

resultado = numero1 >= numero2; // resultado = true

resultado = numero1 <= numero2; // resultado = true

resultado = numero1 == numero2; // resultado = true

resultado = numero1 != numero2; // resultado = false

**Se debe tener especial cuidado con el operador de igualdad (==),** ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts**. El operador == se utiliza para comparar** el valor de dos variables, por lo que es muy diferente **del operador =, que se utiliza para asignar** un valor a una variable:

## ESTRUCTURAS DE CONTROL DE FLUJO

La estructura la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

if(condicion) {...}

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.

Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. Ejemplo: EJERCICIO 4 Y 3 y 5

## FUNCIONES

INICIAR FUNCIONES:

* Con **JQuery** (librería de Javascript):

// jQuery

$(document).ready( function () {

SomeFunction();

}

En total:

<scriptsrc="js/jquery-1.11.0.min.js"type="text/javascript"></script>

<scripttype="text/javascript">

$(document).ready(function () {

yourFunction();

});

functionyourFunction(){

//some code

}

</script>

* Con **Javascript** Puro
  + **window.onload = Nombrefunction();**

Se ejecutará en cuanto se carge la página (window on load) (index6b.html)

* + También podemos poner un **script al final del body** y se ejecutará en cuanto carge el body:(index6html.)

(function() {

// your page initialization code here

// the DOM will be available here

}

La function se ejecutará sola.

## BUCLES FOR IF:

for(inicializacion; condicion; actualizacion) {}

La idea del funcionamiento de un bucle for es la siguiente: *"mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición"*.

* La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
* La "condición" es el único elemento que decide si continua o se detiene la repetición.
* La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Bucle :

var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {

alert(mensaje);

}

La parte de la inicialización del bucle consiste en:

var i = 0;

Se crea la variable i y se le asigna el valor de 0.

Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es:

i< 5

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple.

En este caso, mientras la variable i valga menos de 5 el bucle se ejecuta indefinidamente.

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente.

Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle:

i++

En este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el for.

Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

# 4-SASS

## Variables

uso de variables para almacenar valores Para ello, utiliza cualquier palabra como nombre de la variable, añádele el símbolo $ por delante y establece su valor como si fuera una propiedad CSS normal.

Si por ejemplo defines una variable de la siguiente manera:

$width: 5em;

Ahora ya puedes utilizar la variable llamada $width como valor de cualquier propiedad CSS:

#main {width: $width;}

Una limitación importante de las variables es que sólo están disponibles dentro del contexto donde se han definido. Esto significa que si defines la variable dentro de una regla anidada, sólo estará disponible para esas reglas anidadas. Si quieres poder utilizar una variable como valor de cualquier propiedad de la hoja de estilos, defínela fuera de cualquier selector.

## VARIABLES REGLA DEFAULT

Variable Defaults: !default

Al lado del valor de la variable, si la variable ya tenía asignado un valor, no se reasignará, pero si no lo tenía, se le dará ése.

$content: "First content";

$content: "Second content?" !default;

$new\_content: "First time reference" !default;

## MIXINS

Los Mixins te permiten reutilizar estilos sin tener que copiarlos y pegarlos cada vez que quieres utilizarlos. También pueden ser utilizados como funciones.

Mixin de estilos:

@Mixin botones {

color: white;

width: 30px;

border-radius: 10px;

}

Mixin como función (pasa variables)

@Mixin botones ($background-color) {

$background-color: background-color;

color: white;

width: 30px;

border-radius: 10px;

}

• Como regla general está bien escribir primero el @include y luego poner las propiedades singulares (como background: black;) para poder sobreescribir alguna de las que nos trae el Mixin si fuera necesario.

• Asegúrate que el @Mixin está declarado antes de usar los @include.

• Las propiedades ya definidas con un valor ( ejemplo ($background-color:black) van al final de la declaración de todas las propiedades del mixin o nos dará un error

* Los mixins pueden llevar un @extend si queremos reescribir código css cuando hagamos el include.
* Los **mixin SIEMPRE devuelven css, nunca otro tipo de valores como sí hacen las funciones.**

## EXTEND

Se escribe @extend

http://www.emenia.es/extend-en-sass/

Permite heredar los estilos de una clase a otra. Dentro de las reglas @media hay que volver a declarar los extends porque si están fuera de ellas, no ser reconocerán.

## CONTENT

Dentro de un mixin, @content permite añadir contenido css al invocar el mixin. Se declara entre llaves y no entre paréntesis.

// Mixin for big font properties

@mixin font-big($font-big-weight:normal) {

**@content;**

font: {

family: $\_font1;

size: $\_font-extra-big;

weight: $font-big-weight;

}

}

.block-font-big{

@include font-big **{font-size:$font-big-weight;color:#ddd;}**

}

Placeholder selectors

(Selectores comodín)

Los “Placeholder selectors” se escriben con el símbolo de porcentaje: % y pueden extenderse, pero nunca pueden ser un selector en sí mismos.

Solo sirven para pasar reglas extend, no para poner estilos.

/\* style.scss \*/

%contenido-base {

background: #eaeaea;

padding: 10px;

}

.contenido {

@extend %contenido-base;

}

.lateral {

@extend %contenido-base;

background: #ddd;

}

.home .contenido {

border-bottom: 5px solid black;

}

## INTERPOLACIÓN

Ahorra un valor por medio de #{}

$name: foo;

$attr: border;

p.#{$name} {

#{$attr}-color: blue;

}

## FUNCIONES EN SASS

Las funciones se diferencian de los mixins en que devuelven un parámetro.

Las funciones trabajan con un parámetro (como si fuese la x de una función matemática) que es lo que se le pasa entre paréntesis. El parámetro es lo que usará la función para trabajar y transformarlo. Las funciones generarán un valor, que hay que recoger con **return.**

Luego de definir la función, tendríamos que invocarla. Se invoca también poniendo entre paréntesis un parámetro, que en este caso será el parámetro que se esté usando. (no tiene porqué coincidir con el parámetro )

@function obtener-colores($i){

**$background-color:null;// primero se declara la variable vacía porque aún no tiene valor.**

@if $reference-color == ocean {

$background-color: $\_colour-grill1 \* ($i / 10 + 0.5);

} @else if $reference-color == forest {

$background-color: $\_colour-grill2 \* ($i / 10 + 0.5);

} @else {

$background-color: $\_colour-grill3 \* ($i / 10 + 0.5);

}

**@return $background-color; //para retornar el valor**

}

La función se llama “obtener colores”, usa el parámetro i que es el que le sirve para calcular el color y devuelve el parámetro background-color

@for $i from 1 through $for-elements {

.course#{$i} {

width: 100% / $for-elements;

**background-color: obtener-colores($i);**

}

}

En el bucle for se usa el parámetro i. Invocamos la función para que nos dé el valor del color de fondo. Le pasamos entre paréntesis el parámetro que está usando el bucle. (si no coincide con el valor que usa la función, no importa, el caso es que coincida con el valor que usa el bucle for)

//WHILE DIRECTIVE

$element: 1;

$max-elements: 6;

$virtual-cols: 0;

@for $i from 1 through $max-elements {

$virtual-cols: $virtual-cols + $i;

}

@while $element <= $max-elements {

.level#{$element} {

$col-width: (100 / $virtual-cols) \* $element;

@if unitless($col-width) {

//@error "#{$col-width} May to be in percent"; //ERROR

//@warn "Assuming #{$col-width} to be in percent"; //WARN

$col-width: 1% \* $col-width;

}

width: $col-width;

**background-color: obtener-colores($element);**

$element: $element + 1;

}

}

En este caso, el bucle while usa el parámetro $element, así que es el parámetro que le pasamos a la función “obtener-colores”

## MEDIA QUERIES

## COMPILAR EN CONSOLA:

Comprobar que sass esté instalado (sass –v) Para tener instalado sass hay que tener instalado RUBY Y con Ruby instalar sass.

Ir al archivo donde estén los scss con la consola y escribir:

sass –-watch estilos-master.scss

Cada cambio compilará un nuevo css

Font-face: se tiene que declarar al principio de la hoja, carga de font-family los archivos de fuente asi que tiene q ir al principio de la página. Para incluírla en unmixin se declarará el mixin fontface, la variable será font-family. La llamaremos una sola vez con un include.

**sass --watch directorio entrada (ejemplo scss): directorio salida (css)**

**sass –watch scss:css**

ORGANIZACIÓN:

* Cada unidad de código definida en un partial es un **MÓDULO.**
* Cada módulo en un partial: debemos separar nuestros módulos en partials, que SON ficheros con prefijo \_ destinados a ser importados
* **Un módulo no debe generar código por sí mismo**: es decir, un módulo solo contendrá mixins, funciones y variables Sass, de forma que únicamente importando dicho módulo no generaremos ningún código CSS de salida, salvo que invoquemos a algún mixin contenido en el módulo.
* **Cada módulo contendrá un mixin principal**: este mixin estará situado al principio del fichero y hará un uso estándar de las funciones y mixins que incorpora el módulo, generando un conjunto estándar de estilos del módulo.
* **Todas las variables con !default:** como explicamos en el capítulo 2, incorporando la directiva !default al final de las asignaciones en nuestras variables estaremos indicando que dicho valor únicamente se aplicará si dicha variable no se ha definido con anterioridad

En the sassway lo estructuran en tres partes:

1. Modules: donde se declaran funciones, mixins y variables. No da css de salida.
2. Partials : donde está el css puro.
3. Vendor: donde están los estilos css de los plu-gins que usemos que vengan por defecto.

**ATOMIC DESIGN WITH SASS: (**<http://www.smashingmagazine.com/2013/08/02/other-interface-atomic-design-sass/>)

(http://patternlab.io/)

Sistema de ordenación de SASS para encontrar el código rápidamente en función de su objetivo y relación con el resto del código.

* PANEL DE CONTROL: una hoja de estilos llena de imports de todos los demás partials
* UTILITIES: Donde estarán las utilidades, como hojas de reset, clearfix etc
* QUARKS: No contienen elementos, sólo sus estilos por defecto, **no llegan a formar nada.** Por ejemplo \_\_list.scss, \_\_paragraph.scss, \_\_tables.scss , \_\_links.scss, animaciones, paletas de colores etc.
* ÁTOMOS: Sus estilos no se adjuntan a un elemento de html determinado. Son elementos básicos que pueden ser combinados para hacer estructuras mayores. Son **estructuras básicas** que no se pueden romper. (ej botones, grid,media)
* MOLÉCULAS: Combinaciones de átomos y quarks. Grupos de elementos que funcionan estructuralmente como un todo (ej banners. Headig.group.scss., formulario de búsqueda )
* ORGANISMOS: Grupos de moléculas y átomos juntos para formar la sección de una interfaz.
* TEMPLATES: Organismos combinados para hacer objetos a nivel de página. Sirven para dar la estructura del contenido (no el contenido en sí)
* PÁGINAS: Son específicas instancias de las plantillas (templates) Son esencialmente para testear los estilos.

# 5-COMPASS

Instalar compass: ir a la carpeta del proyecto: y escribir en consola:

**Gem install compass**

Después crear con compass una carpeta:

**Compass create proyecto-compass**

Se creará una carpeta con el nombre que le hayamos definido que contendrá varias subcarpetas.

Si vamos a nuestra carpeta con el proyecto de compass y ponemos en la consola: **ls –a**,

se verán todos los contenidos de la carpeta.

Para compilar, usamos la consola con el comando: **compass watch**

# 6-ESTRUCTURA DE NEO

* ASSETS
  + Fonts
  + Guide
    - Data
    - Img
    - Scripts
      * Neo
      * vendor
    - stylesheets
  + Images
  + Loscanales
  + Scripts
    - Lib
    - App.js
  + Stylesheets
    - Mains.scss
    - Ie.scss
    - Main-guide
    - patterns
    - Átomos
    - Base
      * Base
      * Headings
      * Layout
      * Normailce
      * utilities
    - Globals
      * Mixins
      * Settings
    - Lib
    - Moléculas
    - Organismos
* MODALS

# 7-GRID

La rejilla de neo tiene 12 columnas con un máximo de 1200px y una separación de 20px.