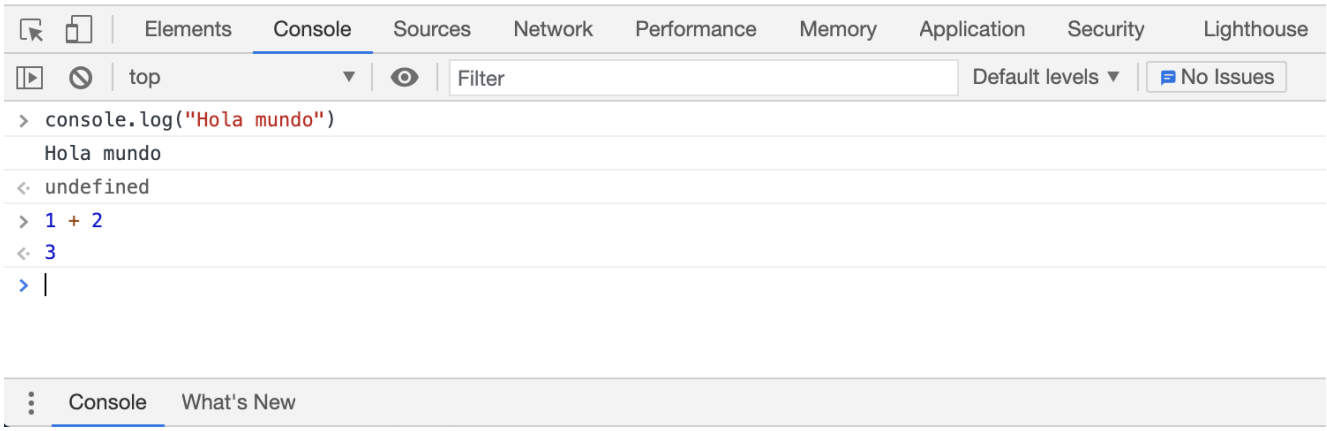
Javascript

> El siguiente texto es una guia teorica-pracitca sobre javascript escrita para la materia de desarrollo de aplicaciones en la nube de la carrera de ingeniería en sistemas de información - UTN FRSF

Intro

Javascript es un lenguaje de programación que ejecuta en todos los browsers.

Por ejemplo, en Chrome podemos utilizar las herramientas de desarrolador para abrir una consola presionando Option + ⌘ + J (macOS), o Shift + CTRL + J (Windows/Linux)



Dentro la consola de Chrome ingresamos expresiones de js como nuestro input `>` que luego se ejecutan y su resultado se imprime como output `<` en la terminal misma.

> TIP: La consola de Chrome es util para realizar pruebas mientras estamos aprendiendo ya que nos permite ver resultados inmediatos y sus herramientas de autocompletado e inspeccion son potentes.

Para poder escribir nuestro código y luego poder ejecutarlo vamos a necesitar un poco de ayuda de NodeJS, un entorno de ejecución de javascript, el cual está basado en el motor que utiliza Chrome.

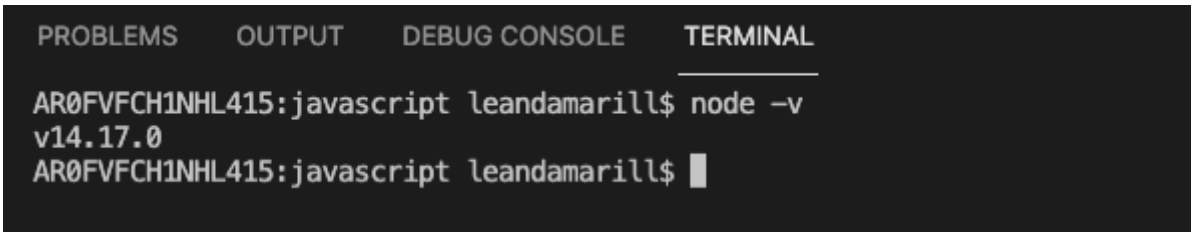
Utilizando Node es que podemos ejecutar javascript en todos lados, incluso en servidores.

Para instalar node basta con dirigirnos a https://nodejs.org/es/ para descargar e instalar la ultima version `LTS`. Otra alternativa es utilizar `nvm` (node version manager), para cuando necesitamos administrar multiples versiones, el cual se encuentra disponible para linux/mac y windows

Para comprobar si se instalo correctamente abrimos alguna terminal y ejecutamos

```
$ node -v
```

Deberiamos ver una respuesta indicando el numero de version de node indicandonos que todo salio bien



Con node instalado solo nos falta un editor de texto, vscode es una buena opción pero tambien existen otros.
Ahora solo nos falta abrir nuestro editor de texto de preferencia y estamos listo para empezar a escribir javascript.

0 - Hola mundo

Creando un archivo de extensión `.js` podemos empezar a escribir código, lo que escribamos aquí se ejecutara de arriba hacia abajo hasta finalizar.

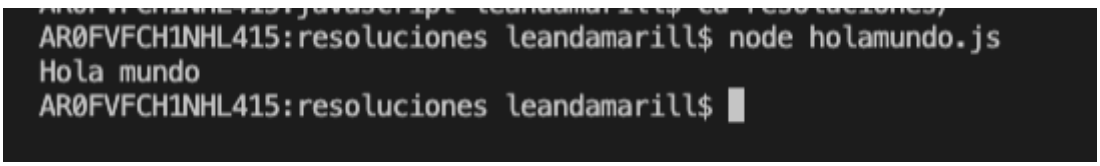
Al escribir

```
console.log("Hola mundo");
```

Suponiendo que nuestro archivo se llama `holamundo.js` podemos abrir una terminal en el mismo directorio que el archivo y ejecutar

```
node holamundo.js
```

Y observamos



Sintaxis basica de javascript

> Esta sección es un pequeño resumen de sintaxis moderna de javascript (ES6), no es necesario revisar pero la idea es que resulte familiar a lo ya conocido por el alumno o que sirva de punto de partida para explorar

Declarando variables

Existen 3 formas de hacerlo

```
var foo = "asd"; // Variable mutable de scope de funcion o global (No recomendado)
let foo = "asd"; // Variable mutable de scope de bloque
const foo = "asd"; // Variable inmutable de scope de bloque
```

> Para ver la diferencia de porque es recomendable `let` sobre `var` pueden leer este link de let vs var

En javascript no declaramos el tipo de dato al momento de crear variables, de hecho aunque no es para nada recomendable, el tipo de dato de la variable puede modificarse en tiempo de ejecucion.

El siguiente codigo es javascript valido y nos muestra los tipos de dato mas comunes

```
let miVariable;
console.log("Type: ", typeof miVariable); // Type: undefined

miVariable = "asd";
console.log("Type: ", typeof miVariable); // Type: string

miVariable = 100;
console.log("Type: ", typeof miVariable); // Type: number

miVariable = 0.212312;
console.log("Type: ", typeof miVariable); // Type: number (no hay distincion de enteros)

miVariable = true;
console.log("Type: ", typeof miVariable); // Type: boolean

// Muchas cosas en javascript son 'object'
miVariable = \[1, 2, 3\]; // Arrays
console.log("Type: ", typeof miVariable); // Type: object

miVariable = {
    id: 1,
    nombre: "juan",
```

```
    apellido: "carlos",
    habilitado: true
}; // Objetos
console.log("Type: ", typeof miVariable); // Type: object

miVariable = null; // Null
console.log("Type: ", typeof miVariable); // Type: object

// Tambien podemos declarar variables definidas

let miVariableDefinida = miVariable;
const miConstanteDefinida = miVariable;
```

Bloques de control

Javascript tambien soporta los bloques de control similares a los que ya conocemos de otros lenguajes

- witch
- while
- for
- if...else

Comparando variables

```
// Operador '==' - Igualdad 'de valor' (no recomendable)
if (100 == "100") {
  console.log("Iguales (?) :thonk:");
}

// Operador '==' - Igualdad de valor y de tipo
if (100 === "100") {
  console.log("Esto no se ejecuta");
}
```

Valores Truty & Falsy de variables

Los valores en javascript se consideran verdaderos o falsos dependiendo de su contenido al momento de ser evaluado en un 'contexto booleano', esto es conveniente para checkeos.

truthy	falsy

truthy	falsy
true	false
'false' (el string false)	0
'0' (el string 0)	'' (string vacio)
() (funcion vacia)	null
[] (array vacio)	undefined
{} (objeto vacio)	NaN (Not a number)
Los demas valores	

Algunos ejemplos

```
let nombre = undefined;

// Usamos !! para negar dos veces e imprimir el valor booleano de la
variable
// tambien podriamos haber escrito
//
// if(nombre){
//   console.log("true")
// }
//   console.log("false")
// }

console.log(!!nombre); // false

nombre = null;
console.log(!!nombre); // false

nombre = "";
console.log(!!nombre); // false

nombre = "Ricardo";
console.log(!!nombre); // true
```

El objeto global 'process' de Node

Node nos proporciona el objeto global process para obtener información e interactuar con el proceso actual.

Como `process` es un objeto global solo basta referenciarlo en nuestro programa para poder utilizarlo, sin embargo nuestro editor de texto a veces puede no reconocerlo. Si quisieramos ayudar el autocompletado del editor podríamos escribir:

```
const process = require('process');
```

Utilizando process recuperar los argumentos de consola accediendo a proces.argv

> Ejercicios
> Si querés resolver los ejercicios pensados para esta sección podes dirijirte a la hoja de ejercicios-00

---

1 - Funciones y asincronía en javscript

// TODO
//sintaxis
// first class citizen
//Non-blocking I/O by default - Event driven.
//

2 - Manipulando objetos y arrays

Objetos

// TODO

Operaciones sobre arrays

// TODO

Desestructuracion

// TODO

3 - Potenciando node

NPM

Hasta ahora solamente hemos usado las herramientas provistas por NodeJS pero nos estamos perdiendo de una de las ventajas mas grandes de javascript, su extensa comunidad y la gran cantidad de librerias de terceros a nuestra disposición.

Para ayudarnos a encontrar y administrar estas dependencias NodeJS, por defecto, incluye un sistema de gestión de paques llamado NPM (Node Package Manager)

Solo teniendo node instalado, desde una terminal, podemos acceder a el

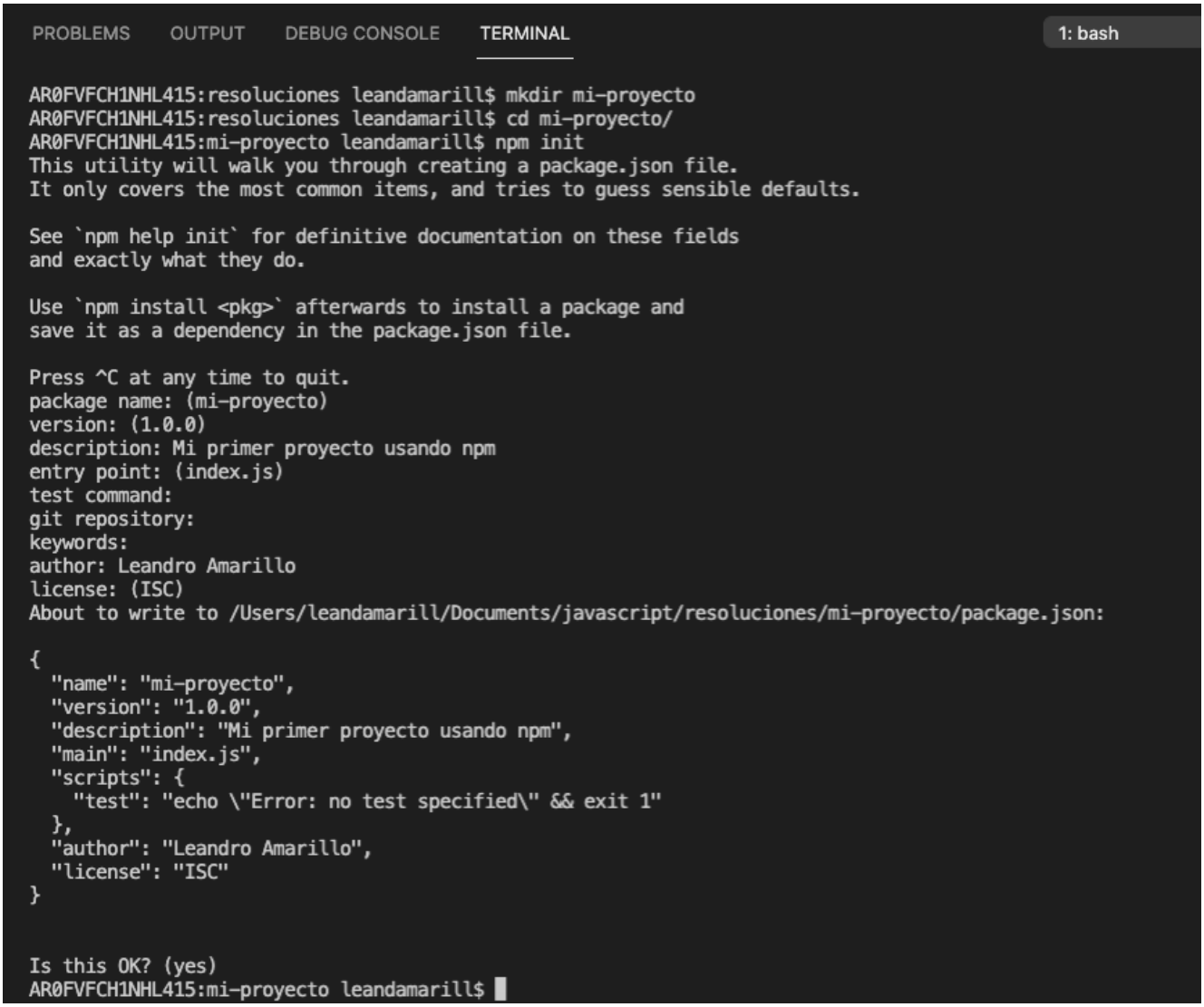
```
$ npm -v
```

Todos nuestros ejercicios implicaban crear un archivo con extensión `.js` y ejecutarlo mediante el comando `node archivo.js`. Si bien puede ser suficiente para scripts sencillos, para desarrollar aplicaciones complejas que necesitan de librerías externas en versiones especificas empezamos a tener problemas.

Utilizando npm podemos generar una estructura de proyecto para nuestra aplicación mediante el comando `npm init`

```
$ mkdir mi-proyecto && cd mi-proyecto // Creamos una carpeta y navegamos
dentro
$ npm init
```

Al ejecutar el comando `init` la utilizadad de npm nos guiará paso a paso en la creación de un archivo `package.json`.



En el queda descrito el nombre del proyecto, la versión y otras caracteristicas de las cuales, sin duda la mas importante, es el registro de las dependencias necesarias para ejecutar nuestro código.

Agregando librerias de terceros

Como ya mencionamos, existe una gran cantidad de librerias de terceros listas para ser utilizadas en nuestro código, por ejemplo, podemos ver algunas de ellas en el repositorio awasome-javascript que se dedica a recolectar y catalogarlas.

Para agregar una libreria, desde una terminal en el directorio donde se encuentra el `package.json`, debemos ejecutar

```
npm i nombre-de-libreria
```

Luego de agregar una librería al proyecto el `package.json` habrá agregado una entrada dentro de la key `dependencies`.

Algunas librerias no son necesarias para ejecutar la aplicación pero si para facilitar el desarrollo (ej: framework de testing unitario) y para esto npm tambien nos deja

instarlarlas utilizando

```
npm i nombre-de-libreria -D
```

Estas dependencias tambien se agregan al `package.json` pero dentro de la entrada `devDependencies`

El directorio node-modules

Si nos descargamos un proyecto que utiliza npm, por ejemplo desde github, gracias a este `package.json` simplemente tenemos que ejecutar `npm i` para descargar las dependencias requeridas. Estas dependencias se guardan dentro de un directorio llamado `node-modules`, si es la primera vez que se descargan tambien se generá un archivo denominado `package-lock.json` el cual indica las versiones de los paquetes que se descargaron y si este esta versionado permite que todos los que descarguen el proyecto utilicen las mismas versiones de las dependecias.

Importando módulos en nuestro código

NodeJS utiliza la sintaxis `require('nombre-del-modulo')` para cargar nuestras dependencias, esta función devuelve un objeto que contiene las funcionalidades que el modulo exporta es por esto que si, por ejemplo, estamos importando el modugo `fs` (file system) incluido en nodejs se suele hacer de la siguiente manera

```
const fs = require('fs');
```

La misma sintaxis aplica a las librerias de terceros, por ejemplo para utilizar la librería dayjs que nos facilita el manejo de fechas primero ejecutamos `npm i dayjs` para agregar y descargar la dependecia y luego para utilizarla solo hacemos

```
const dayjs = require('dayjs');
```

4 - API usando NodeJS y Express

Express

//TODO