# Clase 1: Presentación, herramientas.

*6/06/2020 - ICOMDS-OF - Agustín Bernardo*

El día de hoy motivaremos los contenidos del curso, los explicaremos y mencionaremos brevemente. Además, discutiremos e instalaremos las herramientas que necesitamos para llevarlo a cabo.

---

**¿Por qué un curso de programación?**

Hace unos años, un egresado de la EIS llamado Juampi Maspóns dió un curso optativo de Programación en PIC (microcontroladores). Eso influenció notablemente la carrera de muchos de los que lo hicimos. Pecando de romántico, nos enamoró un poco la lógica con la que las cosas se solucionaban. El impacto fue notable. En mi caso, hice la carrera de física en el Instituo Balseiro y me volqué al Desarrollo de software para la industria médica. Diez mil veces me vi diciendo "esto se soluciona con un par de líneas de código".

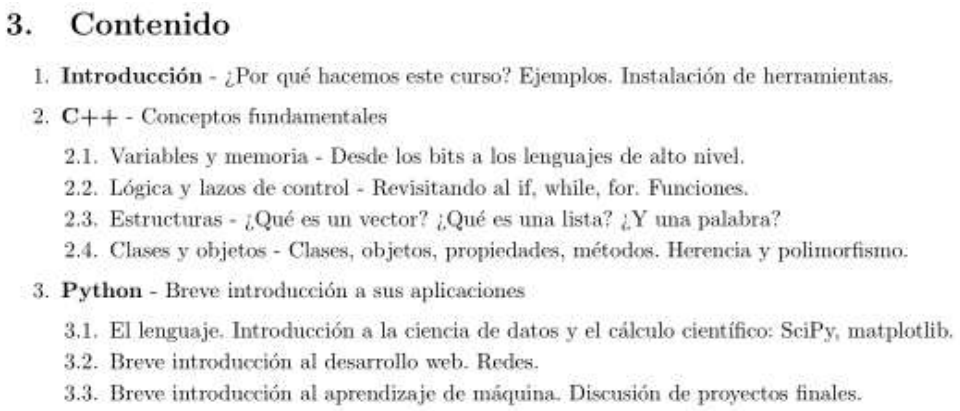
**¿Cómo se seleccionaron los contenidos del curso?**

Dentro de mi *juvenil* experiencia, seleccioné el camino más lógico, rápido y directo para tratar algunos de los tópicos más calientes a día de hoy: IA, WebDev, Data Science. Naturalmente, hay que empezar por los fundamentos. **El curso va a ir al caño. Están avisados.**

Estoy a su disposición, en todos los casos, para las preguntas que les puedan surgir.

**¿Y cuáles son estos contenidos que mencionás?**

Bueno, como les mandé en el PDF, dejé puesto en la invitación, vamos a tener ocho clases, las cuales son:



3. Contenido

1. **Introducción** - ¿Por qué hacemos este curso? Ejemplos. Instalación de herramientas.
2. **C++** - Conceptos fundamentales
   - 2.1. Variables y memoria - Desde los bits a los lenguajes de alto nivel.
   - 2.2. Lógica y lazos de control - Revisitando al if, while, for. Funciones.
   - 2.3. Estructuras - ¿Qué es un vector? ¿Qué es una lista? ¿Y una palabra?
   - 2.4. Clases y objetos - Clases, objetos, propiedades, métodos. Herencia y polimorfismo.
3. **Python** - Breve introducción a sus aplicaciones
   - 3.1. El lenguaje. Introducción a la ciencia de datos y el cálculo científico: SciPy, matplotlib.
   - 3.2. Breve introducción al desarrollo web. Redes.
   - 3.3. Breve introducción al aprendizaje de máquina. Discusión de proyectos finales.

Sí, copié y pegué la imagen. Alto ratonazo.

Ahora sí, a los bifes.

---

# Herramientas

## Lenguajes

Tenemos visto que queremos programar. Para decirle a la máquina *qué* hacer, tenemos que escribírselo. Lo detalles de qué es lo que pasa cuando escribimos y corremos un programa los vamos a ver la clase que viene. Hoy vamos a ver alguna que otra clasificación rápida.

#### Compilado vs Interpretado

- El lenguaje compilado es transformado a código de máquina por el compilador, y luego el sistema operativo corre directamente las instrucciones. Ejemplos: C/C++/Golang. En general son más rápidos
- El lenguaje interpretado se corre línea a línea a través de un interprete, que a su vez va pasando las instrucciones una por una al procesador. Naturalmente, este proceso es más lento, pero suelen ser más cómodos del punto de vista del desarrollador. Además, muchas veces hay tanto músculo en el procesador que esperar 10 $\mu$s o 10 ms es lo mismo. Ejemplos: Python/JS.

#### Tipado estático o tipado dinámico

- Los lenguajes de tipado estático son aquellos donde uno define explícitamente el tipo de las variables a utilizar. Repito - esto se va a ver en profundidad el sábado que viene. C/C++/TypeScript/Etc.
- Los de tipado dinámico se rigen en un principio muy interesante: Si hace cuack, nada y tiene pico, es un pato. Python, JavaScript.

#### Paradigma de programación

- Procedural: Se sigue un procedimiento rígido.
- Funcional: Se define el problema y los mecanismos del lenguaje lo solucionan.
- Orientado a objetos: Es muy similar al procedural, pero uno se enfoca en modelar adecuadamente el problema que tiene que resolver generando entidades llamadas "objetos" que pertencen a "clases", y tienen propiedades y métodos. Como imaginan, porque este item es el más largo, es al que más bola le vamos a dar.

Nosotros vamos a trabajar con **C++** y **Python** en este curso. Tal vez, si llegamos, veamos algo de HTML/CSS/JS al final, en la parte de WebDev.

```
    Ahora, instalemos dos cosas: Anaconda y MinGW-x64, para poder trabajar
con estos dos lenguajes.
```

https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86_64.exe (https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86_64.exe) (déjenlol bajando, pesa como 500MB)

https://sourceforge.net/projects/mingw-w64/files/latest/download (https://sourceforge.net/projects/mingw-w64/files/latest/download) (este es importante!)

## Editores de texto, entornos de desarrollo

Para *programar* hay que **escribir** código. ¿En qué vamos a escribir código? ¿En Word? No.

Escribir código, aunque utilice las mismas letras que el idioma, es una actividad muy distinta a escribir lenguaje natural.

Por este motivo, existen herramientas específicas para dicho fin. Vamos a diferenciar dos clases, y a nombrar la herramienta que finalmente vamos a usar:

```
    Editores de texto
```

Un editor de texto es un software que le permite al usuario reconocer lo que escribe en el teclado, y traducirlo a un archivo que puede ser luego guardado en el disco duro. ¡Eh, el bloc de notas! ¡Word!. Sí, sí, son editores de texto - un **buen** editor de texto, además, te simplifica la vida con un millón de funciones de edición especializadas. Desde el viejo y confiable *control ce, control ve* a cosas realmente poderosas, como Code Refactoring, snippets, autocompletes con machine learning y la mar en coche. Algunos (esta lista no es exhaustiva) son:

- Sublime (El rey, hasta que llegó VSCode)
- Kate
- Vim (Este es zarpado, pero la curva de aprendizaje es jodida. Sólo para pros!)
- Atom

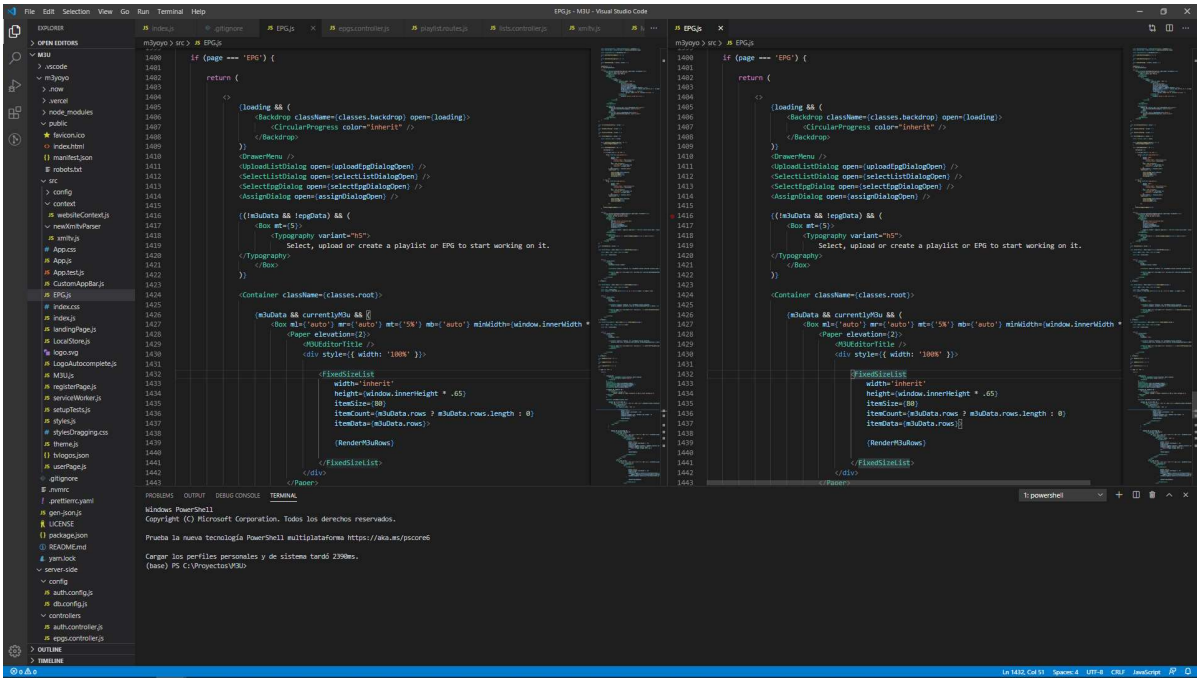
```
Entornos de desarrollo (IDE's)
```

Los entornos de desarrollo son, valga la redundancia, entornos que tienen adentro un editor de texto y además herramientas específicas para correr código. Muchas veces están vinculados a un único lenguaje. Ellos son/pueden ser:

- CodeBlocks: C/C++
- Eclipse: Java
- Spyder: Python

```
VS Code
```

Visual Studio Code es lo que efectivamente vamos a usar. Yo creo que hoy en día, toda persona que se dedique a hacer código lo usa. En esencia, es un editor de texto, pero a diferencia de la mayoría de los editores de texto viene con un montón de subfeatures que hacen que la experiencia del usuario (UX) sea increíblemente buena. Tiene la posibilidad de instalarle plugins (nosotros vamos a usar vaaaaaarios plugins) para transformarlo en un entorno de desarollo multipropósito. Además, podés abrir la consola adentro del editor.



```
Sí, se ve flama. A continuación, vamos a instalar VS Code. Sí, ese cód
igo es mío. ¡Tiene colorcitos!
```

## Consola

¿Alguna vez vieron algún hacker moviendo un mouse? Yo tampoco. ¿Por qué? Porque es más rápido escribir que mover la mano. Creo. Capaz que por creer eso soy malardo en el counter, quién sabe.

La *interfaz de usuario* es lo que nos permite interactuar con la computadora. Hay de dos tipos: gráficas (GUI, Graphical User Interface), o por línea de comandos (CLI, command-line interface). A la hora de laburar, es *muchísimo* mejor usar una línea de comandos.

Hay múltiples terminales. Está PowerShell, CMD, Unix Terminal, etcétera. Nosotros vamos a usar Powershell, que viene por defecto con el VSCode y anda bárbaro. Sí, repito: ¡está adentro de VSCode! Sale andando de cheto, no hace falta modificar nada.

## Control de versiones

Cuando uno trabaja en un archivo de word, siempre va guardando lo que va haciendo. Los últimos cambios son los mejores. Cuando uno escribe código, las cosas no son tan así. Va cambiando, va mutando, va probando cosas. Además, hay proyectos con millones de líneas de código (¡El kernel de Linux!). Por eso, no sólo se guarda el código haciendo "Ctrl-S". Se utiliza un control de versiones más adecuado. El más famoso de ellos es **Git**.

**Git** nos permite guardar *cada versión* del archivo en el que estamos trabajando. Y no sólo eso: te permite ir viendo qué cambios se hicieron, quién los hizo, *por qué se hicieron* y demás. ¿Cómo funciona?

Saque, ponga, meta, modifique. Lo vemos todo, lo sabemos todo.

### Commit

Un *commit* o nodo es una foto de una versión del programa. Digo una foto porque es algo estático en el tiempo, que queda guardado y se puede mirar cuando sea. Tiene información de:

- Parent Commit
- Autor
- Fecha
- Código (ID)
- Mensaje
- Contenido

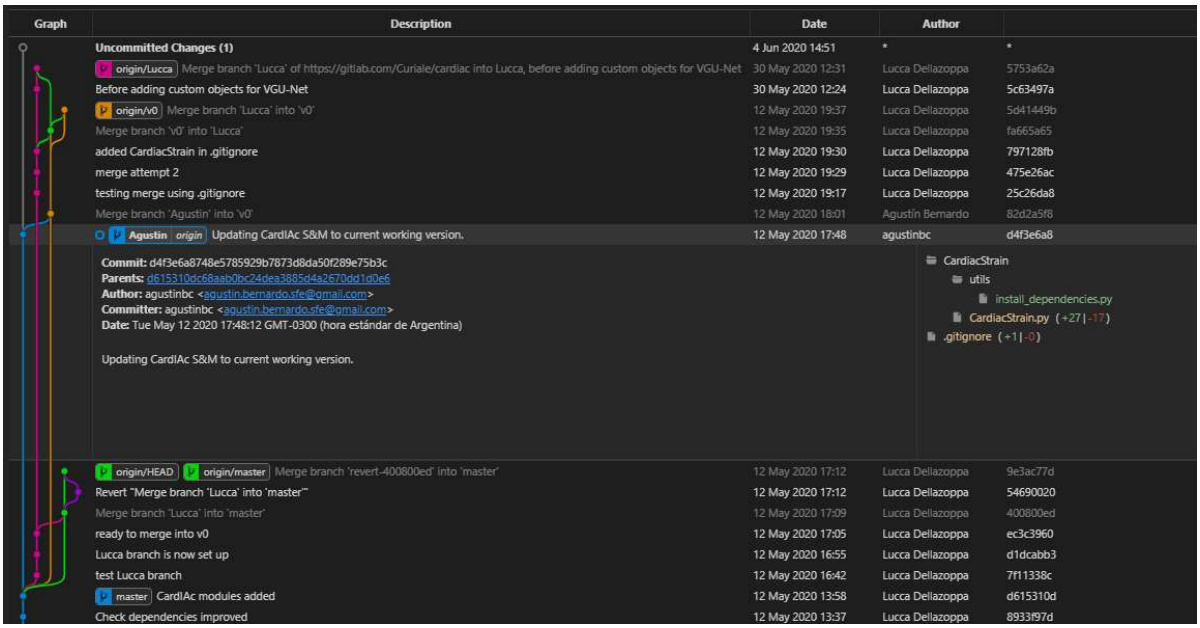
Naturalmente, en el contenido está lo que queremos visualizar. "ID" es un identificador. Y Parent Commit es el commit anterior. Así, cada commit sabe quién es su padre, de alguna forma.

### Branch

Un *Branch* es un conjunto de commits alineados. Significa "Rama" en inglés. Y tiene sentido, porque después de todo, cada rama se tiene nodos que se van conectando. Pero ojo: también se mezclan, y puede ponerse complicado.

**Repositorio (tree)**

Un arbol es un conjunto de ramas. Más allá de eso, es una estructura abstracta de datos. En sí, todo el arbol contiene las fotos de cada una de las versiones que van ocurriendo, con su información de auto, mensaje, etcétera.




Claramente esto permite trabajar de a varios sin romper un programa (a veces se rompe igual), tener una historia de quién hizo qué, cuándo, y por qué. Les puedo poner la firma de que es fundamental que sepan que esto existe. Miren lo que es el de linux:

**Github**

Para poder trabajar con Git necesitamos un servidor de Git. Obvio, podríamos tener uno en nuestra PC, pero sería al recontra pedo. Todo está en la nube hoy. Háganse una cuenta de GitHub:

www.github.com (http://www.github.com)

Y de ahí ya van a poder crear sus propios repositorios!

Necesito que instalen Git (no pesa nada!): https://git-scm.com/download/win (https://git-scm.com/download/win)

---

# Ejercicio: Hello World++

El día de hoy sólo va a tener un ejercicio, de alguna forma.

Lo que vamos a hacer es *clonar* el repositorio de las clases (donde voy a subir absolutamente todo, incluido este documento), *crear* un repositorio propio de ustedes donde guarden lo que subí/voy a ir subiendo, y puedan tener las soluciones a sus propios ejercicios.

Necesito que tengan instalados:

- MinGW-w64
- VS Code
- Git

Vamos al link del repositorio del curso:

https://github.com/agustinbc/icomds-of (https://github.com/agustinbc/icomds-of)

Vamos a "Clone or download", copiamos este link: https://github.com/agustinbc/ICOMDS-OF.git (https://github.com/agustinbc/ICOMDS-OF.git)

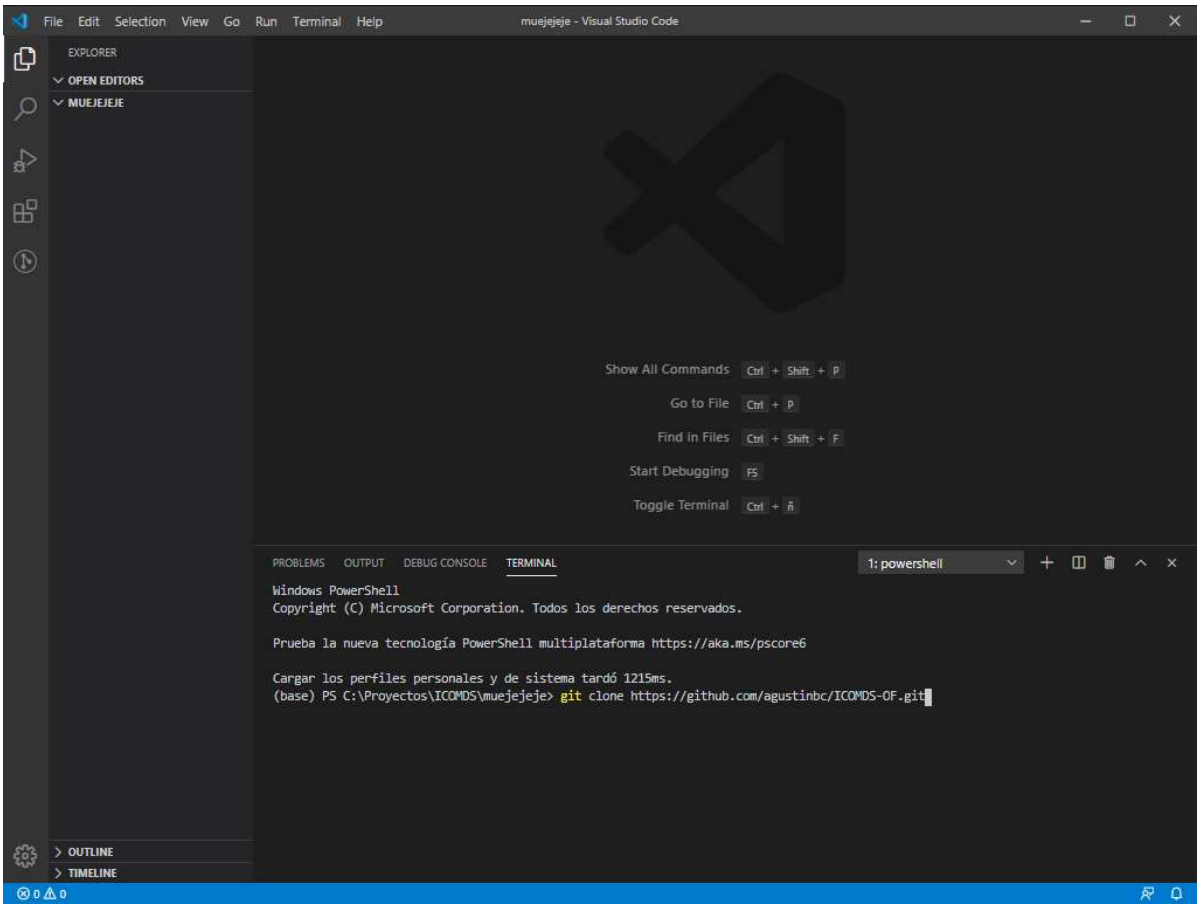
Abrimos el VSCode en una nueva carpeta (con "open folder", ctrl+k -> ctrl+o), vacía, con el nombre que nos parezca adecuado para el curso.

Abrimos la consola (ctrl+shift+ñ) y escribimos:

```
» git config --global user.name «nombre de usuario»
» git config --global user.email «mial»
» git config --global user.password «pass»
```

para iniciar sesión. Después:

```
» git clone «link»
```




Boom, tenemos el repositorio *clonado*. Así de sencillo, me robaron el código.

Ahora lo vamos a subir a su propio repositorio. Vayan a github.com con su cuenta, y creen un nuevo repositorio. Luego, en *clone or download* copien su link. Vuelvan a la consola de VSCode y escriban:

```
» cd ICOMDS-OF
» git remote set-url origin «yourLinkRepo»
» git remote add repositorioAgustin «link»
```

Listo. Ahora, cuando hagan un push, van a levantar la información a su repositorio. Pueden hacer:

```
» git remote set-url origin «yourLinkRepo»
» git push #la info a su repo
» git pull repositorioAgustin master #actualizan las cosas con mi propio r
epositorio
```

A continuación, vamos a hacer un ejemplito muy básico de C++ (¿hello world?) y lo vamos a subir a nuestro propio repositorio.

Dentro de la carpeta, creamons una nueva carpeta con nombre "Resoluciones propias". Ahí creamos un archivo llamado "helloworld.cpp".

Les escribimos el código:

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Y lo tenemos que compilar usando minGW. Para eso, sencillamente, vamos a la consola y hacemos:

```
» cd './Resoluciones Propias'
» g++ helloworld.cpp -o helloworld.exe
» .\helloworld.exe
```

Y bueno, la consola nos dice "Hola mundo!". Ahora tenemos que subir este archivo a nuestro propio git. Para eso, hacemos:

```
» git add .
» git commit -m "Mi primer commit!"
» git push
```

¡Y listo! Idealmente debería estar andando, y se debería subir a su propio repositorio lo que hicieron.

In [ ]:
```
1
```