Programación e Informática

Maestría en Inteligencia Artificial (MIA)
UDeSA 2025

Formulario de bienvenida

Docentes

Matías López y Rosenfeld

Daniela Villani

¿Y ustedes?

Les invitamos a presentarse, entre otras cosas pueden mencionar:

- formación de grado
- experiencia profesional (si tienen) en el tema o en otros ámbitos
- lenguajes de programación que manejan (o prefieren)
- qué buscan en la maestría (y en la materia)

Objetivo de la materia

Establecer las bases de programación e informática sobre las cuales se desarrollarán los próximos cursos de la MIA.

Para hoy tenemos

- Presentación de la materia
- Intro a Python
- Repositorio (GIT)

Dinámica del curso

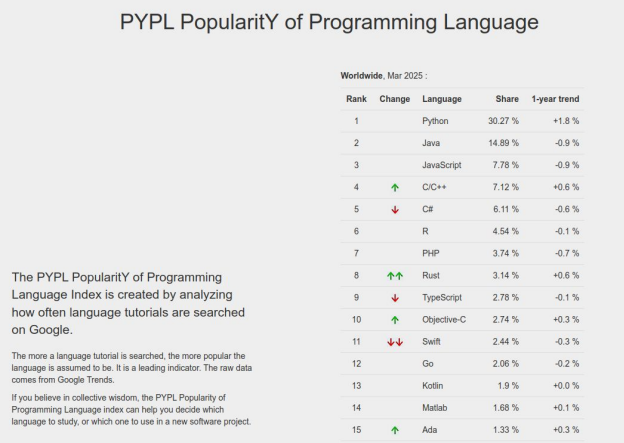
- Clases semanales (los jueves) apuntando a tener una parte teórica y luego ejercitación
- Entregas semanales de ejercicios
- Examen a final del curso (a confirmar)

¿Por qué usar Python?

- Open Source
- Cross-platform
- Tiene un fuerte desarrollo en una gran variedad de áreas:
  - Desarrollo de aplicaciones con interfaz gráfica
  - Ciencia de Datos
  - Simulación numérica
  - Desarrollo para Web e Internet

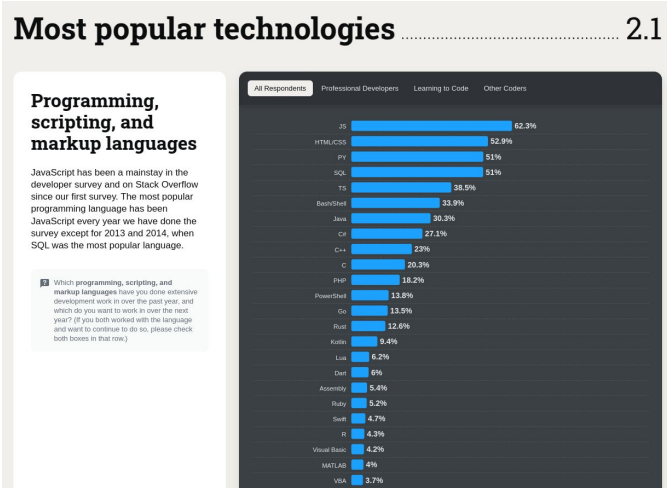
Es un lenguaje que se usa cada día más.

Aprender Python es una inversión segura.

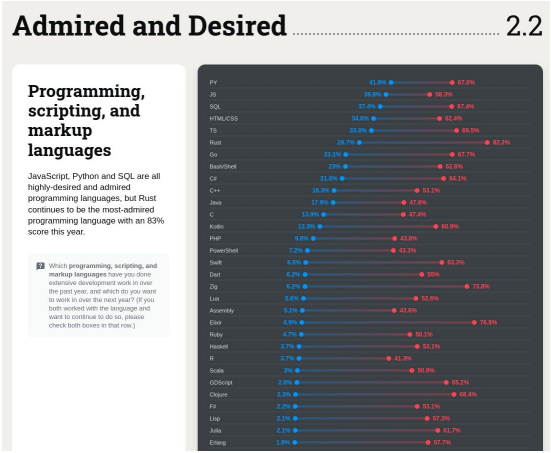


https://pypl.github.io/PYPL.html

https://www.tiobe.com/tiobe-index/



https://survey.stackoverflow.co/2024/technology

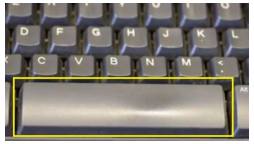


https://survey.stackoverflow.co/2024/technology

¿Qué cosas dispone?

- Bibliotecas (`import pandas`)
- Gestor de paquetes (`pip3 install pandas`)
- Paquetes que agrupan muchas herramientas de ML (https://scikit-learn.org/)
- Paquetes para graficar/generar reportes
- Paquetes para interactuar con bases de datos y otras fuentes

¿Qué tenemos que saber si nunca usamos Python?

El código está determinado por la indentación (espacios en blanco), no es lo mismo espacio () que tabulación (

).

No se usa ; al final de la línea.

No se usa { } para ordenar los bloques de código, sino espacios y :.

Es un lenguaje interpretado, no se compila.

Es tipado dinámicamente.

C++

```cpp
#include <iostream>

using namespace std;

int main() {
    int secret_number, num_of_guesses, user_guess;

    secret_number = 42;

    num_of_guesses = 1;

    // Read in user's guess as an integer
    cout << "Please enter a number: ";
    cin >> user_guess;

    while ((user_guess != secret_number) && (num_of_guesses < 5)) {
        cout << "Incorrect. " << "Please enter a number: ";
        cin >> user_guess;
        num_of_guesses++;
    }

    if (user_guess == secret_number) {
        cout << "Correct" << endl;
    } else {
        cout << "Incorrect. Game over." << endl;
    }

    return 0;
}
```

Python

```python
secret_number = 42

num_of_guesses = 1

# Read in user's guess as an integer
user_guess = int(input("Please enter a number: "))

while user_guess != secret_number and num_of_guesses < 5:
    print("Incorrect. ")
    user_guess = int(input("Please enter a number: "))
    num_of_guesses = num_of_guesses + 1

if user_guess == secret_number:
    print("Correct")
else:
    print("Incorrect. Game over.")
```

IDE - Entorno de desarrollo integrado

- ¿Cuáles conocen?
- ¿Qué ventajas tienen?
- ¿Cuál es la recomendación?
- ¿Jupyter Notebook o IDE?

GIT

Final.py

Final_2.py

Final_final.py

GIT

Sistema de control de versiones

Nos permite registrar los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo.

- Facilita el seguimiento de los cambios, integrar modificaciones y revertirlos para volver a versiones previas.
- Por eso, propicia el trabajo colaborativo, facilita compartir código y modificarlo, además está diseñado para tratar de evitar conflictos y, en caso de accidentes, brinda herramientas para poder resolverlos.
- Git es un sistema de control de versiones **distribuido** y de código abierto. Fue diseñado con énfasis en la performance (permite manejar proyectos muy grandes), la seguridad y la flexibilidad. Además, provee un amplio conjunto de comandos que permiten realizar operaciones de alto y bajo nivel.

Un poco de jerga

(y sus correspondientes anglicismos verbales)

- Repositorio (repo)
  Es el lugar donde almacenamos el código, los archivos y el historial de revisiones de cada archivo.
- Branch
  Un branch es una ramificación del flujo de trabajo principal (Main).
- Commit
  Es una "foto" del estado del progreso en un punto determinado.
- Conflicto
  Existen modificaciones diferentes en un mismo archivo que requieren establecer algún criterio para elegir cuál prevalece.
- Clonar, pushear, mergear... ¡Ampliaremos!

GitHub

GitHub es una plataforma que nos permite administrar, compartir y almacenar nuestros repositorios de Git.

Es la que vamos a utilizar en la materia, pero existen otras alternativas similares.

Instalación

Instalar Git en nuestra computadora

- **En Linux:**

  `apt-get install git`

  (Para otras distribuciones ingresar aquí)

- **En Windows:**

  Git for Windows gitforwindows.org

- **En Mac OS X:**

  Usando Homebrew: **brew install git**.

  git-scm.com/downloads/mac

Crear un usuario en GitHub **github.com**

Configuraciones iniciales

Cada modificación va asociada a nuestro usuario y mail. Para establecerlos:

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

La opción `--global` hace que los establezcamos por única vez. En caso de querer utilizar excepcionalmente otras credenciales en algún proyecto, podemos ejecutar el comando de configuración sin incluirla.

Configuraciones iniciales

Para poder trabajar con repositorios Git que estén en Internet (GitHub, Bitbucket, GitLab, etc.), podemos configurar una clave SSH que nos identifique con el servidor que estemos usando.

Abrimos la terminal y ejecutamos `ssh-keygen` . Le damos Enter a todo

En la terminal, ejecutamos cd ~/.ssh , y luego abrimos con cat el archivo .pub.

Copiamos todo lo que aparezca.

En GitHub, vamos a Settings, SSH Keys. Pegamos lo que copiamos en el campo Key, y le ponemos un Título y agregamos la key.

Obtener un repositorio

Para obtener una copia local de un repositorio existente en algún servidor

utilizamos el comando `git clone` *`URL`*

```
git clone https://github.com/Daniverovi/Practica-Git.git
```

Git permite usar distintos protocolos de transferencia.

El ejemplo anterior usa el protocolo `https://`, pero también se puede utilizar `git://` o `usuario@servidor:ruta/del/repositorio.git` que utiliza el protocolo de transferencia SSH.

En ese caso el comando a ejecutar para clonar el repositorio sería:
`git clone git@github.com:Daniverovi/Practica-Git.git`

¡Vamos a intentarlo!

Git en acción: Add

Una vez que tenemos cambios hechos, tenemos que marcarlos como preparados antes de confirmarlos. Retomando la jerga de Git, decimos que pasamos los cambios a `staged`.

Primero, creamos/modificamos localmente el archivo en cuestión.

Luego, ejecutamos `git add [nombre del archivo]`.

Adentro del repositorio que clonaron recién, crear un archivo y marcarlo como *staged* usando el comando `git add`

Git en acción: Commit

Cuando ya tenemos ciertos cambios en *staged*, podemos confirmarlos ejecutando **`git commit -m [mensaje]`**.

El [mensaje] es una breve descripción de los cambios que acabamos de confirmar. Es una buena práctica incluirlos y que brinden información precisa sobre lo que se ha modificado.

Confirmar los cambios que pasaron a *staged* en el paso anterior.

Git en acción: Status

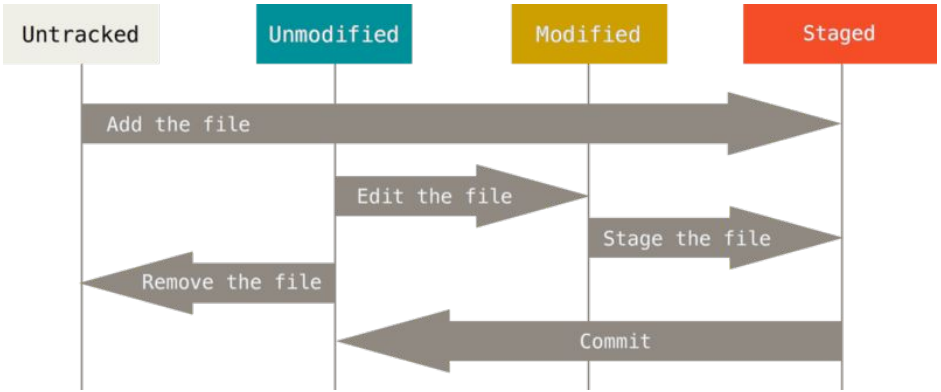
¿Cuál es el estado de mi proyecto?

Las modificaciones que realizamos pueden estar diferentes estados.

El comando `git status` nos brinda un poco más de información al respecto.

Los estados en GIT

- **Confirmado (commited)**: significa que los datos están almacenados de manera segura en tu base de datos local.
- **Modificado (modified)**: significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- **Preparado (staged)**: significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.
- **Sin seguimiento (untracked)**: significa que el archivo nunca fue agregado al repositorio.



Flujo de trabajo en Git

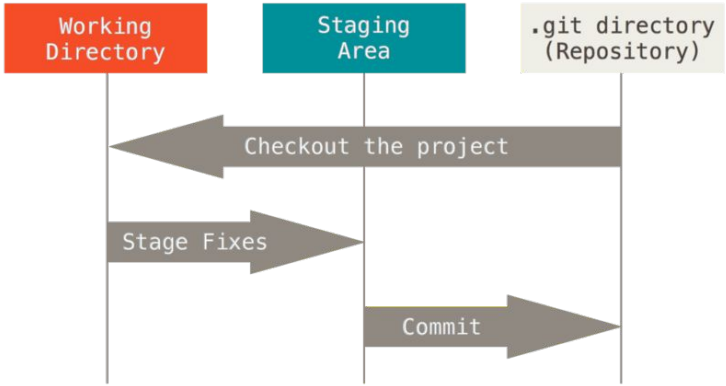
El **directorio de Git (.git directory (Repository)** es donde se almacenan los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia al clonar un repositorio desde otra computadora.

El **directorio de trabajo (Working Directory)** es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.

La **Staging Area** se trata de un archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice ("index").

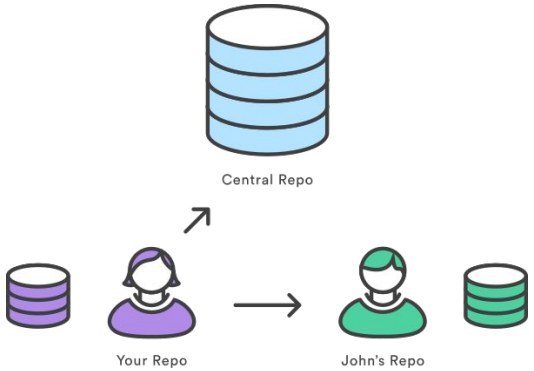
1. Modificás una serie de archivos en tu **directorio de trabajo.**
2. Preparás los archivos, agregándolos a la **Staging Area**
3. Confirmás los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu **directorio de Git**.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (**committed**). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (**staged**). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (**modified**).



Repositorios remotos

El trabajo colaborativo requiere trabajar sobre repositorios remotos, que son aquellas copias de nuestros proyectos a las que accedemos por Internet. Puede haber varios, cada uno de los cuales puede ser de solo lectura o de lectura/escritura, según los permisos que tengamos.



Repositorios remotos

Para integrar nuestro trabajo junto con el de otros colaboradores en el mismo proyecto, debemos gestionar esos repositorios remotos. Mandando (`push`) y recibiendo (`pull`) datos cuando se desee compartir cambios.

Para **enviar** los cambios desde nuestro repositorio local a algún repositorio remoto, debemos ejecutar: `git push [remoto] [branch]`.

En la medida que no definamos nuevas ramas o branches, vamos a utilizarlo como `git push origin master`

Para **obtener** cambios desde un repositorio remoto a nuestro repositorio

local, ejecutamos: `git pull [remoto] [branch]`.

Y para utilizarlo, por el momento, `git pull origin master`

Conflictos

¿Qué pasa si dos personas editan, al mismo tiempo, la misma línea de código del mismo proyecto en las copias de sus repositorios locales y luego confirman los cambios?

¿Qué ocurre cuando quieran compartir sus cambios?

¿Qué va a hacer Git ante este problema? Quejarse.

Una de las dos personas va a tener que integrar las modificaciones de la otra.

Para eso, va a decidir cómo quiere que quede el archivo, va a ejecutar el comando add y luego va a commitear los cambios, en lo posible con un mensaje que diga algo referido al Merge o integración.

Git init

Este comando crea un repositorio local vacío.

Para utilizarlo, nos paramos en el directorio que queremos convertir en un repositorio.

Ejecutamos `git init`

Esto crea un subdirectorio `.git` que tiene todos los archivos necesarios de Git.

Git remote

Este comando se utiliza para ver los repositorios remotos asociados

Ejecutamos `git remote -v`.

Para agregar un repositorio remoto ejecutamos

`git remote add [nombre que queramos] [URL]`.

Ejercicio (de a dos personas)

1. Crear un repositorio local vacío desde alguno de sus equipos.
2. Crear un repositorio nuevo en GitHub, y darle permiso a su compañero/a para hacer push.
3. Asociar el repositorio remoto recién creado.
4. Crear un archivo README con contenidos distintos en la primera línea.
5. **Integrante 1:** Hacer push de los cambios al repositorio remoto.
6. **Integrante 2:** intentar hacer push de los cambios al repositorio remoto.
7. **Integrante 2:** Bajarse los cambios del repositorio remoto.
8. **Integrante 2:** Resolver los conflictos.
9. **Integrante 2:** Añadir y confirmar el archivo que tenía conflicto.
10. **Integrante 2:** pushear estos nuevos cambios.
11. **Integrante 1:** bajarse los nuevos cambios.

Recursos

Pro Git, de Scott Chacon y Ben Straub. Es uno de los libros más consultados sobre qué es Git, cómo funciona internamente, y cómo usarlo.

Hoja de referencia rápida de GitHub y Documentación del sitio

Hay **muchos** tutoriales y recursos en línea, de todas formas lo más recomendable para dominar esta herramienta es la práctica.

¡Muchas gracias!