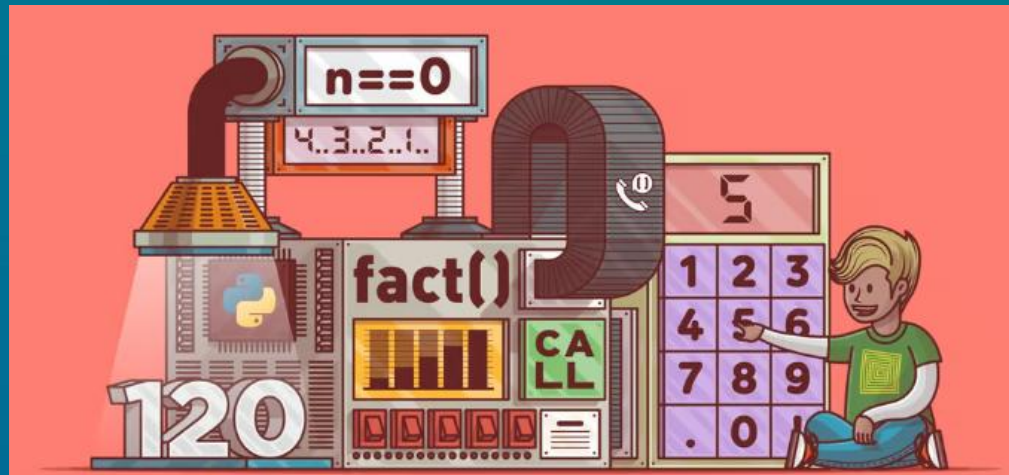


# Funciones recursivas



# Las muñecas rusas (matrioshkas)

Las matrioshkas son un conjunto de muñecas que se colocan una dentro de la otra. Cada muñeca se abre para revelar otra más pequeña en su interior, y así sucesivamente, hasta llegar a la más pequeña que no se abre.

A partir de esta analogía, podemos analizar el concepto de **recursividad**.



# Recursividad

Cuando un proceso se define en términos de sí mismo hablamos de recursividad. Básicamente un problema podrá ser resuelto de forma recursiva si la solución se puede expresar en términos de si misma. Para poder obtener esta solución, deberá resolverse el mismo problema sobre un conjunto de datos de entrada menor.

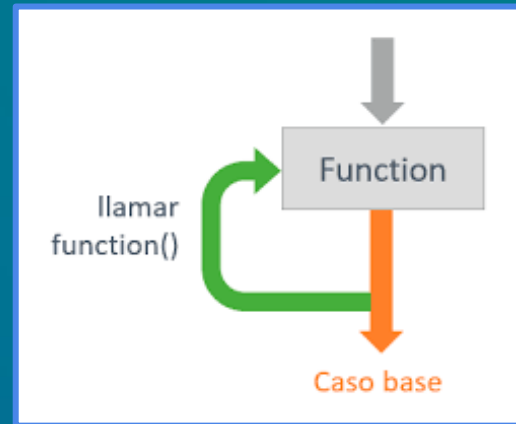
¿En qué trabajás? Estoy intentando arreglar los problemas que creé cuando intentaba arreglar los problemas que creé cuando intentaba arreglar los problemas que creé.

**Y así nació la recursividad.**



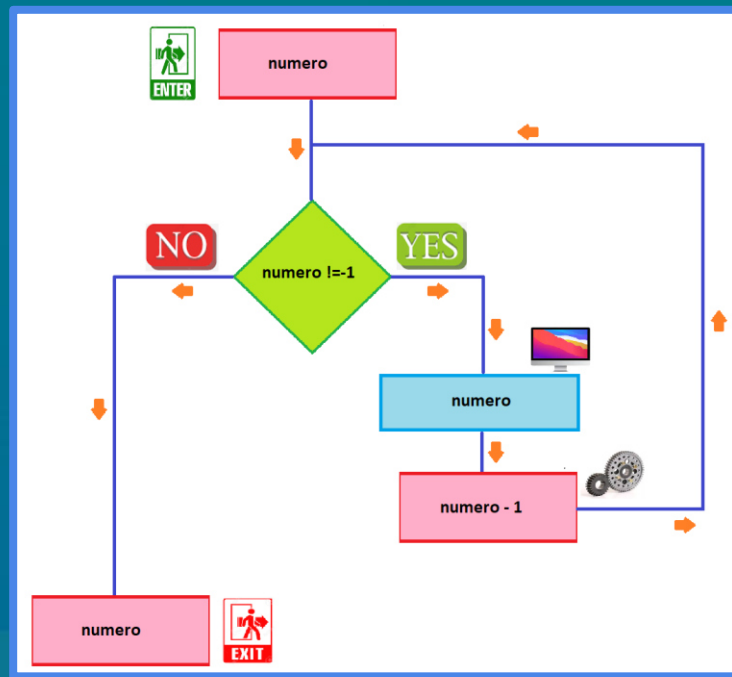
# Funciones recursivas

La recursividad aplicada al mundo de la programación nos permite resolver tareas en donde las mismas pueden ser divididas en **sub tareas cuya funcionalidad es la misma**. Una función recursiva es aquella que está definida en función de sí misma, por lo que se llama repetidamente a sí misma **hasta llegar a un punto de salida**.



# Pongámonos a prueba

El siguiente diagrama de flujo muestra una cuenta regresiva desde un número ingresado hasta el 0. Planteemos una solución desde un punto de vista convencional (con una estructura repetitiva) y desde un punto de vista recursivo



# Factorial de un número

El factorial de un número entero positivo se define como el producto de todos los números enteros positivos desde la unidad hasta el número definido.

Matemáticamente definimos al factorial:

$$n! = n * (n - 1)!$$

Si  $n \geq 0$



$$5! = 5 * 4 * 3 * 2 * 1 = 5 * 4! = 120$$

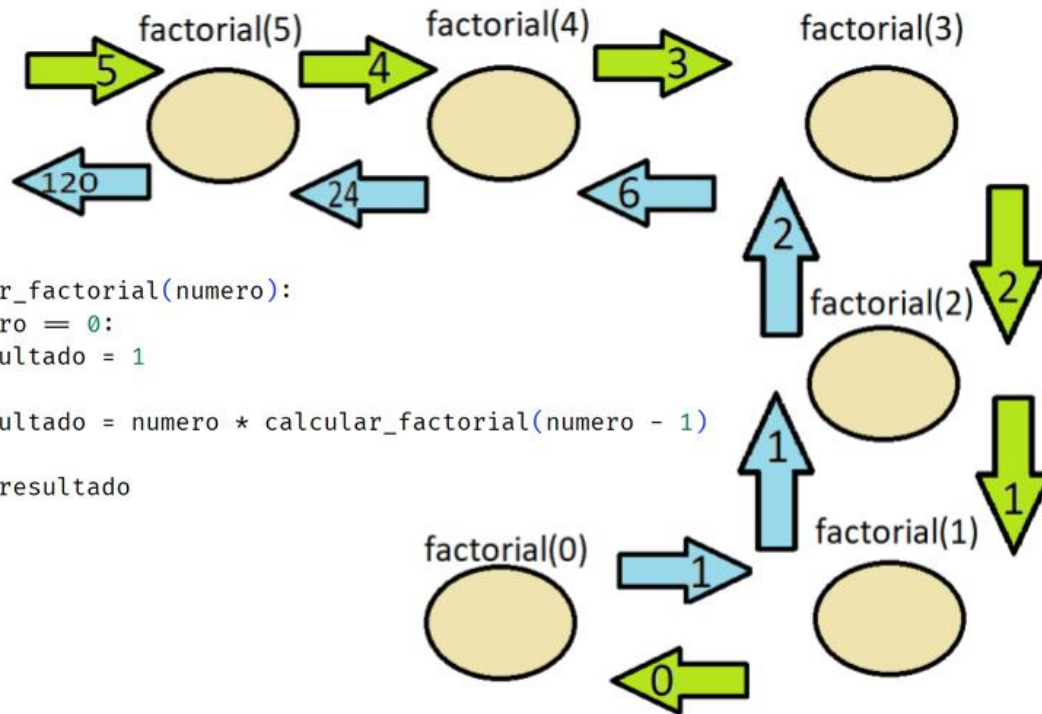
$$4! = 4 * 3 * 2 * 1 = 4 * 3! = 24$$

$$3! = 3 * 2 * 1 = 3 * 2! = 6$$

$$2! = 2 * 1 = 2 * 1! = 2$$

$$1! = 1$$

$$0! = 1$$

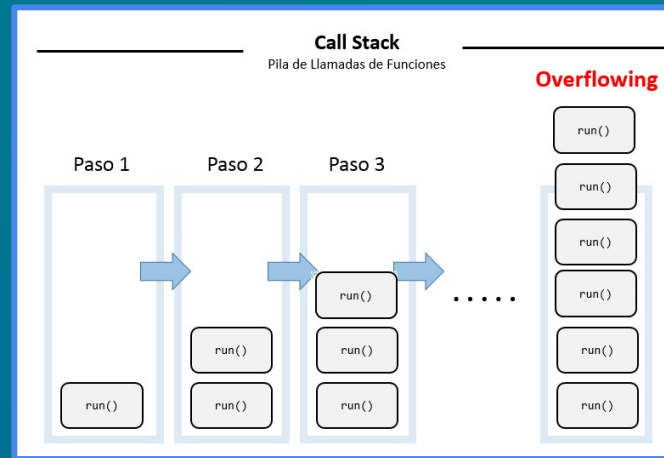


```
def calcular_factorial(numero):  
    if numero == 0:  
        resultado = 1  
    else:  
        resultado = numero * calcular_factorial(numero - 1)  
    return resultado
```



## Pero ... ¿qué pasa en memoria?

- La gestión de memoria en las funciones recursivas se realiza mediante la pila de llamadas.
- Cada vez que se llama a una función recursiva, se agrega una nueva instancia de esa función a la pila de llamadas.
- A medida que la función se llama recursivamente se acumulan mas y mas instancias en la pila.
- Cuando se alcanza la condición de salida, las funciones comienzan a desapilarse, en caso de que dicha condición no se cumpla y se llegue a cierto límite (llamado el límite de recursión), python lanzará un error de desbordamiento de pila (**RecursionError**).



# Ventajas



- **Simplicidad conceptual:** algunos problemas se pueden expresar de manera más clara y concisa.
- **Solución elegante para problemas recursivos:** hay problemas que naturalmente se prestan a una solución recursiva, como árboles o la división y conquista.
- **Facilidad de mantenimiento:** en algunos casos, el código recursivo puede ser más fácil de entender y mantener que su equivalente repetitivo.

# Desventajas



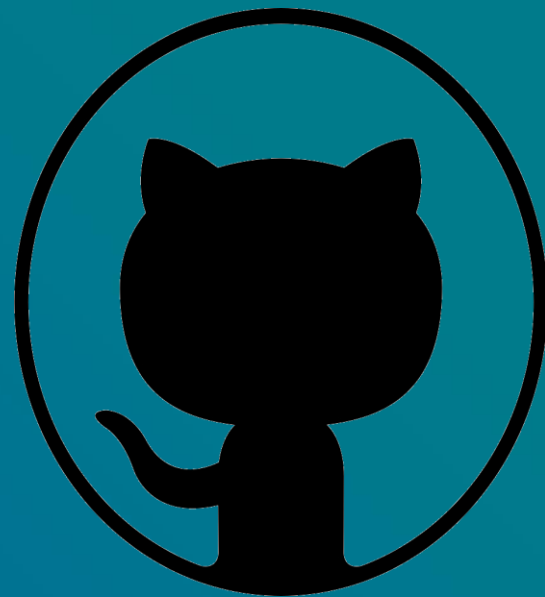
- **Consumo de memoria:** cada llamada recursiva agrega una nueva instancia de la función a la pila de llamadas, esto puede consumir una cantidad significativa de memoria, especialmente para problemas con profundidad recursiva.
- **Eficiencia:** en general las funciones recursivas pueden ser menos eficientes que sus equivalentes iterativos debido al costo adicional de gestionar la pila de llamadas y la posibilidad de realizar cálculos redundantes.
- **Límite de recursión:** python tiene un límite en la profundidad de recursión, por lo que las funciones recursivas pueden no ser adecuadas para problemas con grandes profundidades recursivas.

# Control de versiones



# ¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo de software que permite a los desarrolladores alojar proyectos en repositorios remotos y colaborar con otros a través de un control de versiones basado en Git.

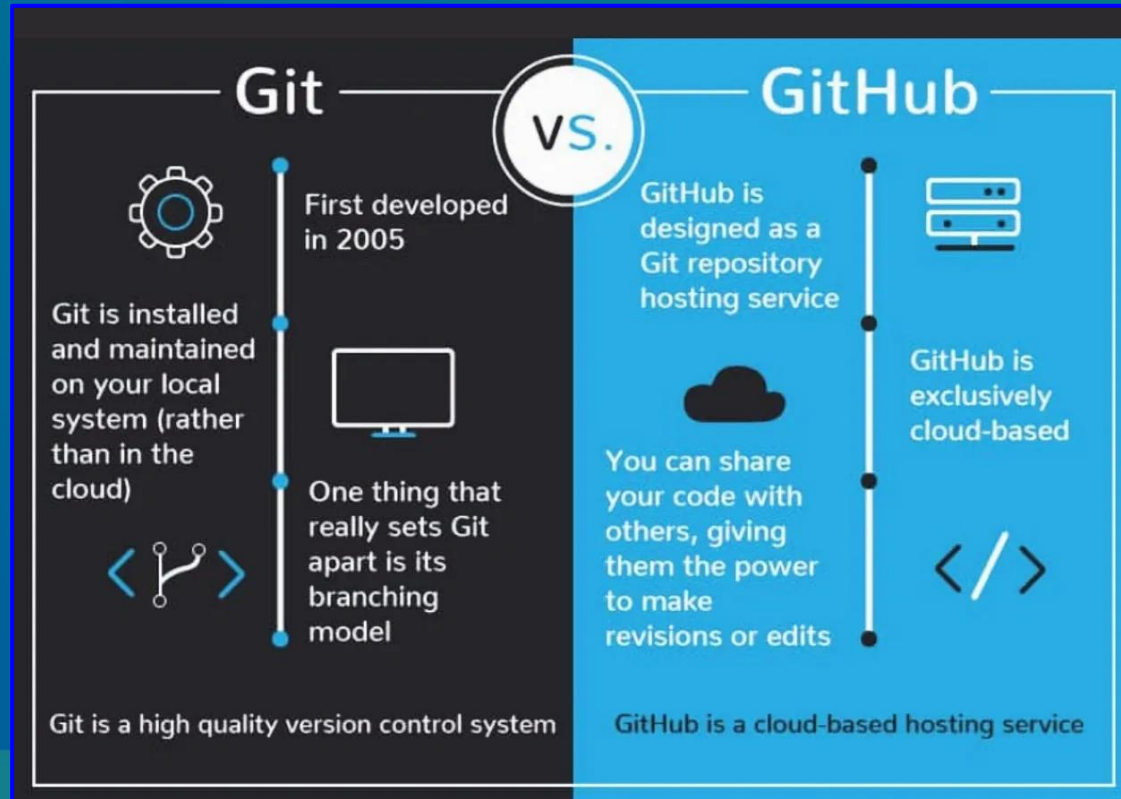




# ¿Qué es Git?

Git es un sistema de control de versiones open source que permite a los desarrolladores realizar un seguimiento de los cambios en el código fuente a lo largo del tiempo, facilitando la colaboración y la gestión de proyectos de software.







# Metodología de trabajo

1. Crear una cuenta en [www.github.com](https://www.github.com)
2. Descargar e instalar git desde <https://git-scm.com/downloads>
3. Configurar git:  
Abrir una terminal y configurar nombre y correo electrónico globalmente.

```
git config --global user.name "Tu Nombre"  
git config --global user.email "tuemail@example.com"
```

# Comandos git

La tecnología git es muy amplia. Existen infinidad de comandos para poder trabajar de manera local y remota con un repositorio. Muchos de esos comandos los aprenderán a lo largo de la carrera. Aquí va una lista de los comandos básicos que usaremos en esta cátedra.

# git init

Inicializa un nuevo repositorio de Git en el directorio actual, creando una carpeta .git.

**Nota:** para inicializar un repositorio previamente tenemos que crear una carpeta y abrir la misma desde línea de comandos.

```
Initialized empty Git repository in /ruta/a/tu/proyecto/.git/
```

# git clone

Me permite descargar y guardar una copia del repositorio remoto de github en nuestro repositorio local de git.

`git clone "url del repositorio remoto"`

# git status

Muestra el estado actual del repositorio, incluyendo los archivos modificados, agregados, y no rastreados (a modo de ejemplo crear un archivo ejemplo.txt dentro de nuestra carpeta donde ejecutamos el comando init).

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ejemplo.txt

nothing added to commit but untracked files present (use "git add" to track)
```

# git add .

Agrega todos los archivos modificados en el directorio actual al área de preparación (staging area).

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   ejemplo.txt
```

**Nota:** luego de ejecutar nuevamente el comando status, git nos informa que hay cambios que están listos para ser impactados (committed)

# git commit -m "mensaje"

Captura de Cambios: Git toma todos los cambios que han sido añadidos al área de preparación (staging area) mediante el comando git add y los guarda en el repositorio local como un nuevo commit. Un commit es un "punto de control" en el historial del proyecto, que registra el estado exacto de todos los archivos en ese momento y nos permite agregar un mensaje descriptivo del mismo.

```
git commit -m "Añadir ejemplo.txt al repositorio"
```

```
[master (root-commit) abc1234] Añadir ejemplo.txt al repositorio  
1 file changed, 1 insertion(+)  
create mode 100644 ejemplo.txt
```

# git commit

## Configuración variables globales

```
Usuario@GCX-PC MINGW64 ~/Desktop/Repositorio/9-9_111 (main)
$ git commit -m "Este es el primer commit"
Author identity unknown


*** Please tell me who you are.

Run

git config --global user.email "you@example.com"
git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Usuario@GCX-PC.(none)')
```





# git remote add origin <url del repositorio>

Asocia tu repositorio local con el repositorio remoto en GitHub.

```
git remote add origin https://github.com/tu_usuario/repo_prueba.git
```

**Nota:** verificar en github que el repositorio local se haya asociado al remoto.

# git push

Envía los commits locales al repositorio remoto, actualizando el proyecto compartido.

```
Enumerating objects: 3, done.  
Counting objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 217 bytes | 217.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/tu_usuario/repo_prueba.git  
* [new branch]      master -> master  
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

