Laboratorio 1

February 16, 2024

Laboratorio #1 - (Git & Github)

Por: Homero Castañón

Carné: 24008059

1 1. ¿Qué es git?

Git es un sistema de control de versiones distribuido y de código abierto desarrollado por Linus Torvalds en 2005. Está diseñado para manejar proyectos de desarrollo de software de cualquier tamaño de manera rápida y eficiente. Es conocido por su capacidad para admitir ramificaciones y etiquetado, áreas de preparación eficientes y múltiples flujos de trabajo. Almacena todo en una base de datos por el valor hash de su contenido, lo que lo hace rápido, seguro y escalable. Su arquitectura distribuida permite que cada copia de trabajo sea un repositorio completo, facilitando la colaboración entre los miembros del equipo. Es ampliamente utilizado en la industria del software y es una habilidad valiosa para los desarrolladores.

2 2. ¿Qué es Github?

GitHub es una plataforma en línea que facilita la colaboración y el control de versiones para desarrolladores de codigo y software. Funciona como un servicio de alojamiento en la nube para repositorios Git, permitiendo a individuos y equipos gestionar su código y realizar un seguimiento de los cambios en él. Fundada en 2008 y adquirida por Microsoft en 2018 por 7.500 millones de dólares, GitHub es el servidor de código fuente más grande del mundo.

Los desarrolladores pueden almacenar, organizar y compartir código en GitHub, lo que les permite trabajar en proyectos de software de manera colaborativa. Es especialmente útil para proyectos complejos y de gran escala, ya que facilita el seguimiento de los cambios a lo largo del tiempo y la colaboración entre diferentes personas.

Además, GitHub ofrece integración con una variedad de herramientas y servicios de terceros, lo que mejora la eficiencia en el desarrollo de software. También sirve como una plataforma de redes sociales para desarrolladores, donde pueden conectarse, colaborar y mostrar su trabajo. En resumen, GitHub es una herramienta esencial para desarrolladores de software que buscan trabajar de manera colaborativa y gestionar proyectos de código abierto de manera efectiva.

3 3. MagicCells

Los “MagicCells” son comandos especiales que permiten a los usuarios realizar diversas tareas y personalizaciones dentro de Jupyter Notebooks.

Por ejemplo, la función mágica de celda representa el contenido de una celda de código como HTML y la función mide el tiempo de ejecución del código de una celda.

Existen dos tipos de *MagicCells*:

A. Cell Magic Functions B. Line Magic Functions

3.1 3.1 Cell Magic Functions

Cell Magic Fuction Son comandos especiales que permiten al usuario modificar explícitamente el comportamiento de una celda de código. Las funciones de Cell Magic tienen el prefijo ‘%%’ seguido del nombre del comando. Las funciones de Cell Magic sirven para diversas tareas y personalizaciones.

3.1.1 3.1.1 Ventajas de las funciones *MagicCell*

- Comportamiento especializado y características dentro de Jupyter Notebook
- Personalización
- Proporciona documentación rica al representar contenido en varios formatos como Markdown y HTML.
- Simplifique tareas comunes como cronometrar la ejecución de código, creación de perfiles, visualización de datos, etc.

3.1.2 3.1.2 Limitaciones de las funciones *MagicCell*

- Dependencias de diferentes núcleos o entornos.
- La ambigüedad ocurre si se usan demasiadas magias celulares y puede causar conflictos potenciales.
- Es difícil recordar la sintaxis de diferentes funciones mágicas celulares.

Función mágica	Descripción
%%intento	Ejecuta celdas con bash en un subproceso.
%%captura	Ejecuta la celda, capturando las llamadas display() enriquecidas de stdout, stderr y IPython.
%%html	Representar la celda como un bloque de HTML.
%%javascript	Ejecuta el bloque de celdas de código Javascript.
%%látex	Renderizar la celda como un bloque de LaTeX.
%%reducción	Representar la celda como bloque de texto Markdown.
%%perla	Ejecuta celdas con perl en un subproceso.
%%pipi	Ejecutar celdas con pypy en un subproceso.
%%pitón	Ejecutar celdas con Python en un subproceso.
%%python2	Ejecuta celdas con python2 en un subproceso.
%%python3	Ejecutar celdas con python3 en un subproceso.
%%rubí	Ejecuta celdas con Ruby en un subproceso.

Función mágica	Descripción
%%guion	Ejecuta una celda mediante un comando de shell.
%%sh	Ejecuta celdas con sh en un subproceso.
%%svg	Representar la celda como un literal SVG.
%%escribir archivo	Escribe el contenido de la celda en un archivo.

A. Tabla de funciones

B. Ejemplos

[74]:
```
%%time
for i in range(1000000):
    pass
```

```
CPU times: user 77.8 ms, sys: 3.42 ms, total: 81.2 ms
Wall time: 81.7 ms
```

[88]:
```
%%writefile my_script.py
print("Hello, world!")
```

```
Overwriting my_script.py
```

[86]:
```
%%markdown
# Encabezado de Markdown
* Punto para describir elementos
- _HOLA_
```

4 Encabezado de Markdown

- Punto para describir elementos
- *HOLA*

[94]:
```
%%html
<font size=10 color='hotpink'>Hola Profesor, es un gusto verle aquí</font>
```

```
<IPython.core.display.HTML object>
```

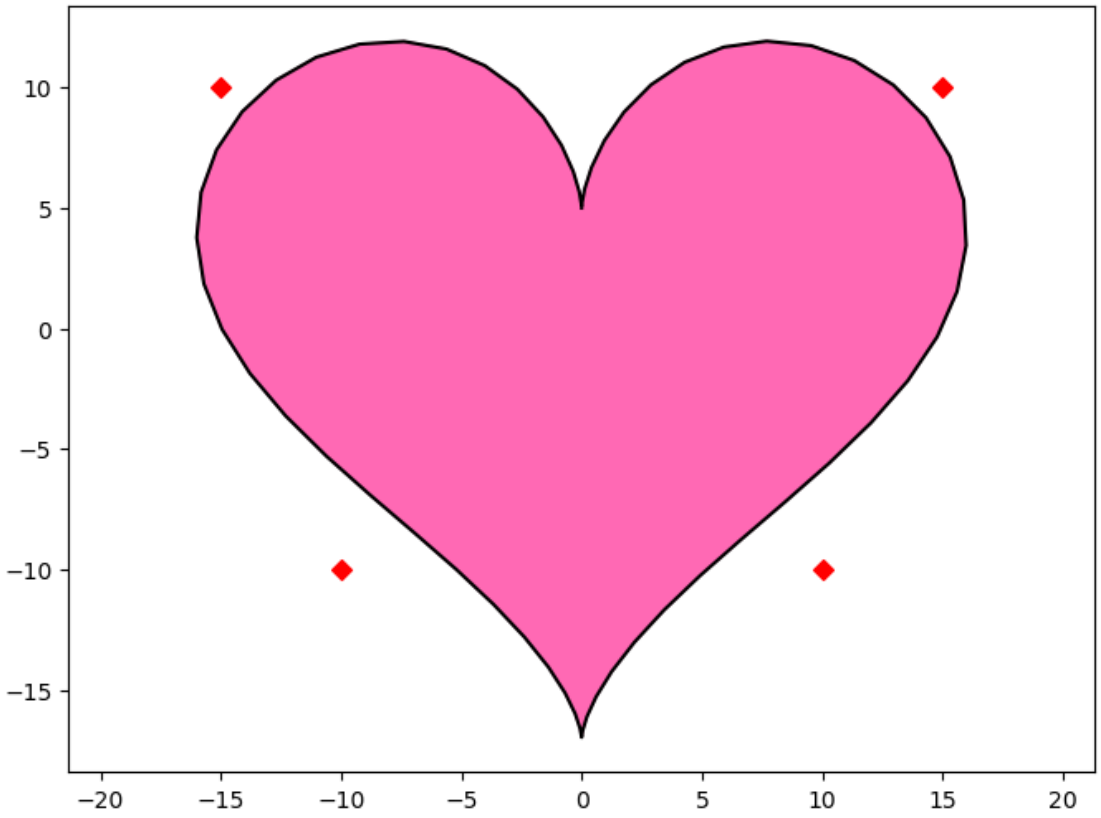
[96]:
```
%%latex
$e^{i\pi} + 1 = 0$
```

$e^{i\pi} + 1 = 0$

[ ]:

[ ]:

[113]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Definir funciones para las coordenadas x e y del corazón
def xhrt(t):
    return 16 * np.sin(t) ** 3

def yhrt(t):
    return 13 * np.cos(t) - 5 * np.cos(2 * t) - 2 * np.cos(3 * t) - np.cos(4 *␣
 ↪t)

# Crear datos para trazar el corazón
t = np.arange(0, 2 * np.pi, 0.1)
x = xhrt(t)
y = yhrt(t)

# Crear la figura y los ejes
plt.figure(figsize=(8, 6))
plt.axis('equal')

# Dibujar el corazón (línea)
plt.plot(x, y, color='black')

# Rellenar el corazón (polígono)
plt.fill(x, y, color='hotpink')

# Dibujar los puntos en la parte superior del corazón
points_x = [10, -10, -15, 15]
points_y = [-10, -10, 10, 10]
plt.scatter(points_x, points_y, marker='D', color='red', zorder=5)

# Mostrar el gráfico
plt.show()
```


10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20