Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Mecatrónica **Dr.-Ing. Carlos Adrián Salazar García**
MT-7003 Microprocesadores y microcontroladores **Verano 2025-2026**
Valor: 5%

Tarea 1. Estudiantes: César Arce Vanegas, Cristian Villalobos Mata

GitHub, Pytest y Flake 8

Preguntas Teóricas (20pts, 2pts c/u)

1) ¿Explique la principal utilidad de Git como herramienta de desarrollo de código?

De acuerdo con GitHub (s.f.), en un sistema de control de versiones distribuido, los desarrolladores tienen una copia del proyecto y del historial de este, y, contrario a los sistemas de control de versiones centralizados, estos DVCS no requieren de constante conexión a un repositorio central. Git es el más popular de estos DVCS y se utiliza tanto en el desarrollo de software de código abierto como comercial.

Permite que los desarrolladores puedan ver en un solo lugar toda la línea de tiempo de los cambios, decisiones y el paso evolutivo de un proyecto, facilitando la comprensión y colaboración en este. Además, permite que se pueda trabajar en cualquier momento ya que se mantiene la integridad del código fuente, y de forma segura, a través del uso de “branches” o ramas.

Referencias.
GitHub. (s. f.). *About Git*. Recuperado el 22 de diciembre de 2025, de https://docs.github.com/en/get-started/using-git/about-git

2) Explique la diferencia entre Git y GitHub

De acuerdo con la pregunta anterior, sabemos que Git es un sistema de control de versiones distribuido, el cual permite que los desarrolladores tengan en un solo lugar acceso a historial, cambios, entre otros de un proyecto, mientras que GitHub es una plataforma que aloja repositorios Git y se encarga de proporcionar a los desarrolladores varias herramientas para entregar mejor código mediante comandos, issues, pull requests, revisión de código, entre otras.

De acuerdo con GitHub (s.f.) esta plataforma integra directamente la colaboración durante el proceso de desarrollo, organizando el trabajo en repositorios donde cada desarrollador pueda definir requisitos o lineamientos y se puedan establecer expectativas para los diversos miembros del equipo, para posteriormente, crear ramas, para las actualizaciones, commits para conservar los cambios realizados, pull requests para debatir y comentar las modificaciones y para luego juntar todos los pasos realizados.

Referencias.
GitHub. (s. f.). *About Git*. Recuperado el 22 de diciembre de 2025, de https://docs.github.com/en/get-started/using-git/about-git

3) ¿Qué es un branch?

De acuerdo con Git-SCM (s.f.), crear una rama significa que se hace una separación de la línea principal de desarrollo con el fin de seguir trabajando sin afectar dicha línea principal. Git maneja las ramas de una forma muy ligera, provocando que las operaciones de ramificación sean casi instantáneas, y a su vez, facilitando que los cambios de una rama a otra sean casi igual de rápidos. También, Git fomenta flujos de trabajo donde frecuentemente se dé la fusión y creación de ramas.

Referencias.

Git-SCM. (s. f.). *Git Branching – Branches in a Nutshell*. Recuperado el 22 de diciembre de 2025, de https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

4) En el contexto de GitHub. ¿Qué es un Pull Request?

De acuerdo con GitHub (s. f.), los pull requests se refieren a proposiciones para unir cambios de código en un proyecto, es el principal método de colaboración que ofrece la plataforma GitHub, ya que permite la discusión y revisión de los cambios propuestos antes de incorporarlos en el proyecto que se trabaja.

Referencias.

GitHub. (s. f.). *About pull requests*. Recuperado el 22 de diciembre de 2025, de https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

5) ¿Qué es un commit?

De acuerdo con GitHub (s.f.), un commit se asemeja a guardar un archivo que ha sido editado, registrando los cambios realizados en los archivos que componen la rama de trabajo. A cada uno de estos commits, se les asigna un valor único que sirve como identificador, llamado **SHA** o **hash**, que identifica:

- Los cambios específicos realizados al proyecto trabajado.
- El momento donde se realizaron dichos cambios .
- La persona o desarrollador responsable de hacer estos cambios.

Además, cuando se realiza dicha operación, es necesario incluir un mensaje que describa los cambios realizados.

Referencias.

GitHub. (s. f.). *About commits*. Recuperado el 22 de diciembre de 2025, de https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits

6) Describa lo que sucede al ejecutar la siguiente operación: “git rebase main”.

De acuerdo con Tran (2025) cuando se ejecuta este comando, se levantan temporalmente los commits únicos de la feature-branch, dejándolos de lado para luego, adelantar el punto de inicio de la rama hasta el commit más reciente hecho en el main, para luego aplicar todos los commits de la rama que fueron hechos a un lado haciendo que esta se asemeje a una línea recta donde el inicio de la rama se da después que se dieran los últimos cambios en el main.

Referencias.

Tran, J. (2025, 5 de junio). *Git Merge vs. Rebase: The Great Debate (And Which One YOU Should Use!)*. Recuperado el 22 de diciembre de 2025, de https://shiftasia.com/community/git-merge-vs-rebase-the-great-debate-and-which-one-you-should-use-2/

7) Explique que es un “merge conflict” y como lo resolvería.

De acuerdo con GitHub (2025) estos merge conflict o conflictos de fusión, se producen cuando distintos desarrolladores hacen cambios en diferentes en la misma línea del mismo archivo, o cuando alguno edita un archivo, y luego otro desarrollador lo elimina. Se deben resolver todos estos conflictos antes de que se pueda fusionar en GitHub un pull request. Con respecto a la resolución de estos conflictos de fusión, se deben editar manualmente los archivos en conflicto y seleccionar los cambios en la fusión final.

- Cuando el conflicto se da por cambios en las líneas en competencia o distintas personas se hacen cambios a una misma línea, en un mismo archivo en ramas distintas del Git, se puede resolver directamente en GitHub en el editor de conflictos.
- Con los demás tipos de conflictos de fusión, el cambio debe ser realizado de forma local en el repositorio para luego subirlo a la rama en GitHub.

Referencias.

GitHub. (2025). *About merge conflicts*. Recuperado el 22 de diciembre de 2025, de https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/about-merge-conflicts

8) ¿Qué es una Prueba Unitaria o Unittest en el contexto de desarrollo de software?

Dentro del desarrollo de software, las pruebas unitarias o unittest, son un instrumento que se utiliza para validar fragmentos del código fuente, asilando líneas del lenguaje codificado para saber si se está produciendo el correcto funcionamiento de una función, proceso o actividad específica.
De acuerdo con Tamushi (2022), estos teste se realizan en las etapas iniciales del desarrollo, ya que, si se realizan en etapas más avanzadas, se pueden generar errores. Además, clasifica estas pruebas en manuales y automatizados, como se observa en la siguiente tabla.

Tipo de prueba unitaria	**Descripción breve**	**Ejemplo / Ventaja**
Manual	Revisión hecha por el desarrollador.	Más lenta y propensa a errores humanos
Automatizada	Se ejecuta mediante scripts o frameworks (PyTest).	Repetible, y confiable para integración continua

Referencias.
Tamushi. (2022, 11 de julio). *¿Qué son las pruebas unitarias de software?* Recuperado el 22 de diciembre de 2025, de https://www.testingit.com.mx/blog/pruebas-unitarias-de-software

9) Bajo el contexto de pytest. ¿Cuál es la utilidad de un "assert"?

Es una función intrínseca de Python que verifica que algo se cumpla y se utiliza para aplicar y verificar que se ejecute correctamente un programa en las condiciones establecidas. Pytest cuenta la cantidad de errores de aserción de la prueba para decirle cuantos y cuales errores dejo pasar en la prueba realizada.

Referencias.

Pytest Development Team. (s. f.). *About pytest.raises()* (sección de "How to write and report assertions in tests"). Recuperado el 22 de diciembre de 2025, de https://docs.pytest.org/en/latest/how-to/assert.html#assertraises (docs.pytest.org)

10) Mencione y explique tres errores de formato detectables con Flake8

Código de error	Descripción del error	Explicación
E225	missing whitespace around operator	Los operadores deben de tener espacios a su alrededor, es decir un espacio antes y otro después de este.
E302	expected 2 blank lines, found 0	Anterior a un comando, se necesitan que existan dos líneas vacías, sin espacios ni algún otro carácter en ellas.
W505	doc line too long (82 > 79 characters)	Más caracteres de lo permitido en una sola línea de código.

Referencias.

Pycodestyle. (s. f.). *Error codes* (Introducción). Recuperado el 22 de diciembre de 2025, de https://pycodestyle.pycqa.org/en/latest/intro.html#error-codes