Universidad Continental

CONSTRUCCIÓN DE SOFTWARE

Guía de Laboratorio

Guía de Trabajo
(Construcción de Software)

Material publicado con fines de estudio.
Código: (ASUC00947)
Huancayo, 2023
De esta edición

Huancayo-Perú
Teléfono: (51 64) 481-430 anexo 7361
Correo electrónico: recursosucvirtual@continental.edu.pe
http://www.continental.edu.pe/

Cuidado de edición Fondo Editorial

Diseño y diagramación Fondo Editorial

Contenido

Presentación

Esta guía ha sido diseñada como un recurso fundamental para acompañar a los estudiantes en el desarrollo de proyectos integradores utilizando Python. En un mundo donde la tecnología evoluciona rápidamente, es esencial contar con herramientas prácticas que permitan aplicar conceptos teóricos en entornos reales. Esta guía no solo proporciona instrucciones claras y estructuradas, sino que también fomenta el trabajo colaborativo, el pensamiento crítico y la resolución de problemas, habilidades indispensables para cualquier profesional del desarrollo de software.

A lo largo de esta guía, se abordan temas clave como el diseño de interfaces gráficas, la implementación de lógica de negocio, la configuración de entornos de despliegue con Docker y la presentación efectiva de proyectos. Además, se incluyen actividades prácticas que permiten a los estudiantes aplicar estos conceptos en proyectos reales, como una aplicación de conversión de monedas o un sistema de gestión de tareas. Cada actividad está diseñada para ser escalable, permitiendo a los estudiantes adaptarla a sus niveles de experiencia y explorar nuevas tecnologías.

El objetivo principal de esta asignatura es que los estudiantes desarrollen competencias integrales en el diseño, desarrollo y despliegue de aplicaciones de software. A través de las unidades, se busca que los estudiantes dominen el uso de frameworks y herramientas modernas, comprendan principios de usabilidad y portabilidad, y apliquen metodologías ágiles en sus proyectos. Al finalizar, los estudiantes estarán preparados para crear soluciones tecnológicas que sean funcionales, accesibles y adaptables a diferentes entornos.

Para aprovechar al máximo esta guía, se recomienda a los estudiantes trabajar de manera colaborativa, asignando roles claros dentro de sus equipos y manteniendo una comunicación constante. Es importante seguir cada paso de las actividades con atención y documentar todo el proceso, ya que esto facilitará la presentación final del proyecto. Además, se anima a los estudiantes a explorar recursos adicionales, experimentar con nuevas tecnologías y buscar retroalimentación continua para mejorar sus habilidades técnicas y blandas. La práctica constante y la disposición para aprender son claves para alcanzar el éxito en esta asignatura.

Daniel Gamarra Moreno

Primera **Unidad**

Construcción sólida desde el inicio

Semana 1: Sesión 2

Desarrollo rápido de aplicaciones sencillas

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 1

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como PyCharm, VS Code o Jupyter Notebook) para desarrollar el prototipo funcional.
- Crear un repositorio en GitHub donde se subirá el código del proyecto y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen su planificación, el prototipo desarrollado y cómo colaboraron como equipo.

I. Propósito

Al finalizar la sesión, cada estudiante desarrolla aplicaciones sencillas aplicando prácticas ágiles y utilizando herramientas modernas de desarrollo, en el contexto de un entorno configurado y un flujo de trabajo ágil.

II. Descripción de la actividad por realizar

1. Planificación del Proyecto (15 minutos)

   Objetivo: Definir un proyecto pequeño que pueda ser desarrollado en el tiempo asignado.

   Pasos:

   - Cada equipo debe elegir una idea simple para su proyecto. Por ejemplo:
   - Una calculadora básica.
   - Un generador de contraseñas aleatorias.
   - Un programa que simule un juego de adivinanzas.
   - Un convertidor de unidades (por ejemplo, de Celsius a Fahrenheit).
   - El equipo debe dividir el proyecto en tareas pequeñas y asignar responsabilidades a cada miembro.
   - Crear un diagrama o lista de tareas en papel o en una herramienta digital (como Trello o Notion) para visualizar el flujo de trabajo.
   - Documentar la planificación en un archivo README.md que luego será subido al repositorio de GitHub.

2. Desarrollo del Prototipo Funcional (30 minutos)

Objetivo: Implementar un prototipo funcional del proyecto utilizando Python.

Pasos:

- Abrir el IDE seleccionado y comenzar a escribir el código del proyecto.
- Utilizar funciones, bucles y estructuras condicionales según sea necesario.
- Asegurarse de que el código sea legible y bien organizado. Se recomienda seguir las mejores prácticas de programación, como nombrar variables de manera descriptiva y comentar el código cuando sea necesario.
- Realizar pruebas continuas para verificar que el prototipo funcione correctamente.
- Subir el código al repositorio de GitHub en intervalos regulares para mantener un historial de cambios.

3. Fase 3: Presentación y Entrega (15 minutos)

Objetivo: Compartir los resultados del proyecto con el resto de la clase.

Pasos:

- Cada equipo debe preparar una breve presentación (máximo 5 minutos) que incluya:
- La descripción del proyecto y su propósito.
- La planificación realizada y cómo se distribuyeron las tareas.
- Una demostración del prototipo funcional.
- Reflexiones sobre el proceso de desarrollo y las lecciones aprendidas.
- Subir el código final y la documentación completa al repositorio de GitHub.
- Compartir el enlace del repositorio al docente para su evaluación.

Semana 2: Sesión 2

Uso práctico de inteligencia artificial generativa en generación de código.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 1

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como PyCharm, VS Code o Jupyter Notebook) para desarrollar la funcionalidad específica de su aplicación.
- Integrar una herramienta de inteligencia artificial generativa (como GitHub Copilot, ChatGPT u otra herramienta disponible) en el flujo de trabajo para generar y ajustar código automáticamente.
- Crear un repositorio en GitHub donde se subirá el código del proyecto y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen cómo integraron la herramienta de IA, la funcionalidad desarrollada y los desafíos encontrados.

I. Propósito

Al finalizar la sesión, cada estudiante integra herramientas de inteligencia artificial generativa para generar y ajustar código automáticamente, en el contexto de un proyecto de desarrollo de software

II. Descripción de la actividad por realizar

1. Los equipos desarrollan un programa que genera contraseñas seguras basadas en criterios específicos (longitud, uso de caracteres especiales, números, etc.).
2. Utilizan una herramienta de IA generativa para sugerir funciones de generación de contraseñas o para optimizar el código.
3. Implementan validaciones para asegurarse de que las contraseñas cumplan con los criterios establecidos.
4. Suben el código a GitHub y documentan el proceso en un archivo README.md.
5. Entrega: Repositorio de GitHub con el código funcional y documentación.

Semana 3: Sesión 2

Manejo de excepciones para robustez en aplicaciones.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 1

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como PyCharm, VS Code o Jupyter Notebook) para trabajar en el código proporcionado.
- Implementar manejo de excepciones en el código existente y personalizar excepciones para mejorar la robustez de la aplicación.
- Crear un repositorio en GitHub donde se subirá el código modificado y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen cómo implementaron el manejo de excepciones, las pruebas unitarias realizadas y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante implementa manejo de excepciones para garantizar la robustez de una aplicación, en el contexto de un código existente y pruebas unitarias validadas.

II. Descripción de la actividad por realizar

1. Implementar manejo de excepciones para capturar errores comunes como:
    a. División por cero (ZeroDivisionError).
    b. Entradas no válidas (ValueError).
2. Crear al menos una excepción personalizada para manejar un caso específico del programa. Por ejemplo:

```
class ValorNegativoError(Exception):
    def __init__(self, mensaje="El valor no puede ser negativo"):
        self.mensaje = mensaje
        super().__init__(self.mensaje)
```

3. Asegurarse de que el programa maneje correctamente las excepciones y proporcione mensajes claros al usuario en caso de error.

4. Subir el código modificado al repositorio de GitHub en intervalos regulares para mantener un historial de cambios.
5. Escribir pruebas unitarias utilizando el módulo unittest de Python para verificar que las excepciones se manejan correctamente.

```
import unittest

class TestCalculadora(unittest.TestCase):
    def test_division_por_cero(self):
        with self.assertRaises(ZeroDivisionError):
            dividir(10, 0)

    def test_valor_negativo(self):
        with self.assertRaises(ValorNegativoError):
            validar_valor(-5)
```

6. Ejecutar las pruebas unitarias y asegurarse de que todas pasen correctamente.
7. Documentar los resultados de las pruebas en un archivo README.md.

Semana 4: Sesión 2

Principios de código limpio.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos

Docente: ..... Unidad: 1

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como PyCharm, VS Code o Jupyter Notebook) para trabajar en el código proporcionado.
- Refactorizar el código existente aplicando las convenciones de estilo PEP 8 y técnicas de refactorización aprendidas (como renombrar variables, eliminar código duplicado, modularizar funciones, etc.).
- Usar herramientas de análisis estático como pylint o flake8 para evaluar la calidad del código antes y después de la refactorización.
- Crear un repositorio en GitHub donde se subirá el código refactorizado y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen cómo refactorizaron el código, los cambios realizados y los resultados obtenidos con las herramientas de análisis estático.

I. Propósito

Al finalizar la sesión, cada estudiante aplica principios de código limpio y refactorización para mejorar la mantenibilidad y legibilidad del software, en el contexto de un proyecto existente y usando herramientas de análisis estático.

II. Descripción de la actividad por realizar

1. Código por refactorizar:

```
def f(a):
    s = 0
    for i in a:
        s += i
    return s

def g(a):
    t = 0
    for i in a:
        t += i
    return t / len(a)

nums = [10, 20, 30, 40, 50]
print("Suma:", f(nums))
print("Promedio:", g(nums))
```

2. Abrir el IDE seleccionado y comenzar a refactorizar el código base.
3. Aplicar las siguientes técnicas de refactorización:
    a. Renombrar variables y funciones para que sean más descriptivas.
    b. Eliminar código duplicado extrayendo funciones reutilizables.
    c. Modularizar el código dividiéndolo en funciones más pequeñas y específicas.
    d. Asegurarse de que el código cumpla con las convenciones de estilo PEP 8 (indentación, longitud de líneas, espaciado, etc.).
4. Usar herramientas de análisis estático como pylint o flake8 para evaluar la calidad del código antes y después de la refactorización.
5. Subir el código modificado al repositorio de GitHub en intervalos regulares para mantener un historial de cambios.

Segunda **Unidad**

Colaboración eficiente en entornos distribuidos

Semana 5: Sesión 2

Control de versiones efectivo con Git y GitHub.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 2

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Seguir un taller guiado para instalar Git, crear un repositorio local, realizar commits básicos y clonar un repositorio desde GitHub.
- Crear un repositorio en GitHub donde se subirá el código y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los comandos utilizados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante utiliza Git y GitHub para gestionar un repositorio de código, realizando commits, clonando repositorios y manejando ramas básicas, en un entorno local y remoto.

II. Descripción de la actividad por realizar

Parte 1:

1. Descargar e instalar Git desde git-scm.com .
2. Configurar el nombre de usuario y correo electrónico globalmente:

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

3. Verificar la instalación ejecutando:

```
git --version
```

4. Crear una carpeta para el proyecto y navegar hasta ella:

```
mkdir mi-proyecto
cd mi-proyecto
```

5. Inicializar un repositorio Git:

```
git init
```

6. Crear un archivo **README.md** con contenido básico:

```
echo "# Mi Primer Repositorio" > README.md
```

7. Añadir el archivo al área de preparación (staging area):

```
git add README.md
```

8. Realizar el primer commit:

```
git commit -m "Primer commit: Agregado README.md"
```

9. Verificar el estado del repositorio:

```
git status
```

Parte 2:

1. Crear un repositorio vacío en GitHub (por ejemplo, llamado taller-git).
2. Copiar la URL del repositorio (HTTPS o SSH).
3. Clonar el repositorio en tu máquina local:

```
git clone https://github.com/tu-usuario/taller-git.git
```

4. Navegar al directorio clonado:

```
cd taller-git
```

5. Crear un archivo hello.py con el siguiente contenido:

```
print("Hola, Git y GitHub!")
```

6. Añadir el archivo al área de preparación y realizar un commit:

```
git add hello.py
git commit -m "Agregado archivo hello.py"
```

7. Subir los cambios al repositorio remoto:

```
git push origin main
```

Semana 6: Sesión 2

Control de versiones efectivo con Git y GitHub con ramas.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 2

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Resolver conflictos de fusión, usar git stash para manejar cambios temporales y aplicar git rebase para mantener una historia de commits limpia.
- Configurar un archivo .gitignore para excluir archivos innecesarios del repositorio.
- Crear un repositorio en GitHub donde se subirá el código y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los comandos utilizados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante aplica técnicas avanzadas de Git, como la resolución de conflictos, el uso de git stash y git rebase, y colabora en proyectos mediante GitHub, siguiendo buenas prácticas de mensajes de commit y exclusión de archivos innecesarios.

II. Descripción de la actividad por realizar

1. Crear un nuevo con un archivo en blanco llamado main.py.
2. Inicia el repositorio y configura las variables globales
3. Crear dos ramas (rama-a y rama-b) desde la rama principal (main).

4. En rama-a, modificar el archivo main.py para agregar una función simple:

```
def saludar():
    print("Hola desde Rama A!")
```

5. En rama-b, modificar el mismo archivo main.py para agregar una función diferente:

```
def despedirse():
    print("Adiós desde Rama B!")
```

6. Intentar fusionar ambas ramas en main:

```
git checkout main
git merge rama-a
git merge rama-b
```

7. Resolver el conflicto manualmente editando el archivo main.py para incluir ambos cambios:

```
def saludar():
    print("Hola desde Rama A!")

def despedirse():
    print("Adiós desde Rama B!")
```

8. Finalizar la fusión y realizar un commit:

```
git add main.py
git commit -m "Resuelto conflicto de fusión entre rama-a y rama-b"
```

9. Hacer cambios en el archivo main.py (por ejemplo, agregar un comentario temporal).
10. Usar git stash para guardar los cambios sin realizar un commit:

```
git stash push -m "Cambios temporales"
```

11. Verificar que los cambios no están en el área de trabajo:

```
git status
```

12. Aplicar los cambios guardados cuando sea necesario:

```
git stash apply
```

13. Crear una nueva rama (rama-c) desde main y realizar varios commits pequeños:

```
git checkout -b rama-c
echo "Línea 1" >> archivo.txt
git add archivo.txt
git commit -m "Primer cambio en rama-c"

echo "Línea 2" >> archivo.txt
git add archivo.txt
git commit -m "Segundo cambio en rama-c"
```

14. Reorganizar los commits de rama-c sobre main usando git rebase:

```
git checkout main
git rebase rama-c
```

15. Observar cómo los commits de rama-c ahora forman parte de la historia de main.

16. Crear un archivo .gitignore en la raíz del proyecto.
17. Agregar patrones para excluir archivos temporales, carpetas específicas o archivos generados automáticamente:

```
# Ignorar archivos temporales
*.tmp

# Ignorar archivos de Python compilados
__pycache__/
*.pyc
```

18. Verificar que los archivos especificados en .gitignore no son rastreados por Git:

```
git status
```

Semana 7: Sesión 2

Configuración de un repositorio con GitFlow.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 2

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Implementar GitFlow en un proyecto colaborativo, creando y manejando ramas específicas como feature, release y hotfix.
- Resolver conflictos que surjan durante la fusión de ramas.
- Crear un repositorio en GitHub donde se subirá el código y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los comandos utilizados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante implementa el flujo de trabajo GitFlow en un proyecto colaborativo, manejando ramas como feature, release y hotfix, y resolviendo conflictos en un entorno de desarrollo organizado.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a implementar el flujo de trabajo GitFlow en un proyecto colaborativo utilizando Python. GitFlow es una metodología que organiza el desarrollo de software mediante el uso de ramas específicas para diferentes propósitos, como la implementación de nuevas funcionalidades (feature), la preparación de versiones (release) y la corrección de errores críticos (hotfix). Durante la actividad, los equipos simularán un ciclo de desarrollo completo siguiendo el flujo GitFlow.

1. Parte 1: Desarrollo de Funcionalidades (feature): Cada equipo trabajará

en la creación de nuevas funcionalidades para un programa base en Python. Por ejemplo, ¡un programa que inicialmente solo imprime "Bienvenido al proyecto GitFlow!" será ampliado con funciones adicionales, como la capacidad de saludar al usuario o despedirse. Cada funcionalidad será desarrollada en una rama feature separada.

2. Parte 2: Preparación de Versiones y Corrección de Errores (release y hotfix): Una vez que las funcionalidades estén listas, los equipos prepararán una versión estable del proyecto utilizando una rama release. Posteriormente, simularán un error crítico en la versión publicada y lo corregirán utilizando una rama hotfix. Durante este proceso, resolverán cualquier conflicto que surja durante la fusión de ramas.

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre cómo GitFlow facilita la organización y colaboración en proyectos de software.

Semana 8: Sesión 2

Gestión y automatización en la construcción del software.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 2

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Configurar un pipeline básico en GitHub Actions que incluya:
    - Ejecución automática de pruebas unitarias.
    - Simulación de un proceso de entrega continua (CD).
- Crear un repositorio en GitHub donde se subirá el código, las pruebas unitarias y el archivo de configuración del pipeline.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los comandos utilizados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante configura un pipeline de integración y entrega continua (CI/CD) utilizando GitHub Actions, automatizando procesos de compilación, pruebas y despliegue, garantizando entregas iterativas y consistentes en un proyecto de software.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a configurar un pipeline básico de CI/CD utilizando GitHub Actions . El objetivo es automatizar procesos clave como la ejecución de pruebas unitarias y la simulación de un despliegue continuo. Para ello, los equipos trabajarán en un proyecto simple en Python, implementarán pruebas unitarias con el módulo unittest y configurarán un archivo .github/workflows/ci-cd.yml para definir el pipeline.

1. El ejercicio se divide en tres partes principales:
    a. Desarrollo del Código Base: Crear un programa simple en Python que realice operaciones matemáticas básicas (suma, resta, multiplicación, división).
    b. Implementación de Pruebas Unitarias: Escribir pruebas unitarias para validar el correcto funcionamiento del programa.
    c. Configuración del Pipeline en GitHub Actions: Crear un archivo de configuración que ejecute automáticamente las pruebas unitarias cuando se realice un push al repositorio y simule un proceso de entrega continua.

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre cómo la automatización de pruebas y despliegues mejora la calidad y consistencia del software.

Tercera **Unidad**

Pruebas automatizadas para iteraciones exitosas

Semana 9: Sesión 2

Introducción a pruebas unitarias y su integración en ciclos ágiles.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 3

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Crear pruebas unitarias básicas utilizando el módulo unittest de Python para validar el correcto funcionamiento de un programa que resuelve ecuaciones de segundo grado.
- Crear un repositorio en GitHub donde se subirá el código del programa, las pruebas unitarias y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los casos de prueba implementados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante configura un pipeline de integración y entrega continua (CI/CD) utilizando GitHub Actions, automatizando procesos de compilación, pruebas y despliegue, garantizando entregas iterativas y consistentes en un proyecto de software.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a crear pruebas unitarias básicas utilizando el módulo unittest de Python. Las pruebas unitarias son fundamentales para garantizar que el código funcione correctamente y cumpla con los requisitos esperados. Los equipos trabajarán en un programa simple en Python que resuelva ecuaciones de segundo grado de la forma ax^2+bx+c=0. Luego, implementarán pruebas unitarias para validar el comportamiento del programa en diferentes

escenarios, incluyendo casos normales (soluciones reales), casos sin solución real (discriminante negativa) y casos extremos (coeficientes nulos).

El ejercicio se divide en tres partes principales:

1. Desarrollo del Código Base: Crear un programa simple en Python que resuelva ecuaciones de segundo grado.
2. Implementación de Pruebas Unitarias: Escribir pruebas unitarias para validar el correcto funcionamiento del programa.
3. Ejecución y Validación: Ejecutar las pruebas unitarias localmente y verificar que todas pasen correctamente.

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre la importancia de las pruebas unitarias en el desarrollo de software.

Semana 10: Sesión 2

Práctica de Desarrollo Guiado por Pruebas (TDD).

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 3

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Implementar el ciclo de TDD (Red, Green, Refactor) para desarrollar una función que calcule los factores primos de un número natural.
- Crear pruebas unitarias utilizando el módulo unittest de Python para validar el correcto funcionamiento de la función.
- Crear un repositorio en GitHub donde se subirá el código del programa, las pruebas unitarias y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los casos de prueba implementados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante aplica el ciclo de Desarrollo Guiado por Pruebas (TDD) (Red, Green, Refactor), escribiendo pruebas significativas y refactorizando código, en el contexto de un ejercicio práctico simple.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a aplicar el ciclo de Desarrollo Guiado por Pruebas (TDD) para desarrollar una función que calcule los factores primos de un número natural. El ciclo TDD consta de tres fases:

1. Red: Escribir una prueba que inicialmente falla porque la funcionalidad aún no está implementada.
2. Green: Implementar el código mínimo necesario para que la prueba pase.

3. Refactor: Mejorar el código sin cambiar su comportamiento, asegurando que las pruebas sigan pasando.

Los equipos trabajarán en un programa simple en Python que calcule los factores primos de un número natural. Luego, implementarán pruebas unitarias para validar el correcto funcionamiento del programa en diferentes escenarios, incluyendo casos felices (números con factores primos) y casos infelices (números sin factores primos o entradas inválidas).

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre cómo el ciclo TDD mejora la calidad del software.

Semana 11: Sesión 2

Resolución de problemas con Katas TDD.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 3

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Resolver el problema de calcular el promedio de una lista de números utilizando el ciclo de TDD (Red, Green, Refactor). El proceso debe descomponerse en casos específicos: lista vacía, lista con un elemento, lista con dos elementos y lista con más de dos elementos.
- Crear pruebas unitarias utilizando el módulo unittest de Python para validar el correcto funcionamiento de la solución.
- Crear un repositorio en GitHub donde se subirá el código del programa, las pruebas unitarias y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los casos de prueba implementados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante resuelve problemas de programación utilizando Katas TDD, aplicando el ciclo de TDD y mejorando continuamente su solución, en un entorno colaborativo y de retroalimentación.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a resolver un problema de programación utilizando el ciclo de Desarrollo Guiado por Pruebas (TDD) . El objetivo es calcular el promedio de una lista de números, descomponiendo el problema en casos específicos:

1. Lista vacía (ningún elemento).
2. Lista con un solo elemento.

3. Lista con dos elementos.
4. Lista con más de dos elementos.
5. El ciclo TDD consta de tres fases:
    a. Red: Escribir una prueba que inicialmente falla porque la funcionalidad aún no está implementada.
    b. Green: Implementar el código mínimo necesario para que la prueba pase.
    c. Refactor: Mejorar el código sin cambiar su comportamiento, asegurando que las pruebas sigan pasando.

Los equipos trabajarán en un programa simple en Python que calcule el promedio de una lista de números. Luego, implementarán pruebas unitarias para validar el correcto funcionamiento del programa en diferentes escenarios, incluyendo casos felices (listas válidas) y casos infelices (listas vacías).

Semana 12: Sesión 2

Implementación ágil de mapeo Objeto-Relacional (ORM).

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 3

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Implementar un modelo de datos para una aplicación To-Do List utilizando un ORM como SQLAlchemy o Django ORM. El modelo debe incluir las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar tareas con los siguientes atributos:
    - Descripción de la tarea (cadena de texto).
    - Estado de la tarea (Por hacer, Haciendo, Realizado).
- Crear pruebas unitarias utilizando el módulo unittest de Python para validar el correcto funcionamiento de las operaciones CRUD.
- Crear un repositorio en GitHub donde se subirá el código del programa, las pruebas unitarias y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los casos de prueba implementados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante implementa un mapeo Objeto-Relacional (ORM) en un proyecto, creando modelos y realizando operaciones CRUD, en el contexto de un proyecto ágil con pruebas unitarias.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a implementar un modelo de datos para una aplicación To-Do List utilizando un ORM como SQLAlchemy

o Django ORM. El objetivo es crear un sistema que permita gestionar tareas con los siguientes atributos:

1. Descripción de la tarea: Una cadena de texto que describe la tarea.
2. Estado de la tarea: Un campo que indique si la tarea está "Por hacer", "Haciendo" o "Realizada".

Los equipos trabajarán en un programa simple en Python que implemente las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar las tareas. Además, implementarán pruebas unitarias para validar el correcto funcionamiento de las operaciones CRUD en diferentes escenarios.

Cuarta **Unidad**

Implementación y despliegue ágil

Semana 13: Sesión 2

Diseño de interfaces gráficas centradas en el usuario.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 4

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, diseñador de interfaz y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Implementar una aplicación de conversión de tipo de cambio utilizando una interfaz gráfica de usuario (GUI) con las siguientes características:
    - Unidades monetarias disponibles: PEN (Perú), EUR (euro) y USD (dólar).
    - Entrada para ingresar la cantidad a convertir.
    - Botones para seleccionar las monedas de origen y destino.
    - Resultado visible en tiempo real tras la conversión.
- Crear un repositorio en GitHub donde se subirá el código del programa, los wireframes/mockups diseñados y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los principios de usabilidad aplicados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante diseña interfaces gráficas centradas en el usuario, aplicando principios de usabilidad y accesibilidad, y utilizando herramientas de prototipado para crear wireframes y mockups.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a diseñar e implementar una aplicación de conversión de tipo de cambio con una interfaz gráfica de usuario (GUI). La aplicación permitirá convertir entre tres monedas: PEN (soles peruanos), EUR (euros) y USD (dólares estadounidenses). El objetivo es aplicar

principios de usabilidad y accesibilidad para garantizar que la interfaz sea intuitiva y funcional.

El ejercicio se divide en tres partes principales:

- Diseño de Wireframes y Mockups: Crear bocetos de la interfaz gráfica utilizando herramientas de prototipado como Figma, Balsamiq o papel y lápiz.
- Implementación de la Interfaz Gráfica: Desarrollar la GUI utilizando una biblioteca como tkinter o PyQT en Python.
- Pruebas de Usabilidad: Probar la interfaz con diferentes escenarios para asegurar que sea fácil de usar y accesible.

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre cómo los principios de usabilidad y accesibilidad mejoran la experiencia del usuario.

Semana 14: Sesión 2

Uso de Docker para despliegue y portabilidad ágil de aplicaciones.

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 4

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, revisor de código y documentador (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Configurar un contenedor Docker para desplegar una aplicación To-Do List simple en Python. Esto incluye:
    a. Crear un archivo Dockerfile para definir la imagen personalizada.
    b. Construir la imagen Docker.
    c. Ejecutar un contenedor basado en la imagen creada.
- Utilizar Docker Compose para simplificar el proceso de despliegue.
- Crear un repositorio en GitHub donde se subirá el código de la aplicación, el Dockerfile, el archivo docker-compose.yml y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los comandos utilizados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante configura y gestiona contenedores Docker, creando imágenes personalizadas y utilizando Docker Compose para facilitar el despliegue y garantizar la portabilidad de aplicaciones en diferentes entornos.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes aprenderán a configurar y gestionar contenedores Docker para desplegar una aplicación To-Do List simple en Python. El objetivo es crear una imagen personalizada utilizando un

Dockerfile, construir la imagen y ejecutar un contenedor basado en ella. Además, los estudiantes explorarán cómo Docker Compose puede simplificar el proceso de despliegue al permitir la gestión de múltiples servicios en un solo archivo de configuración.

El ejercicio se divide en tres partes principales:

1. Desarrollo de la Aplicación To-Do List: Crear una aplicación simple en Python que permita agregar y listar tareas.
2. Configuración del Contenedor Docker: Crear un Dockerfile para definir la imagen personalizada y construirla.
3. Despliegue con Docker Compose: Utilizar Docker Compose para simplificar el despliegue de la aplicación.

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre cómo Docker facilita la portabilidad y el despliegue de aplicaciones en diferentes entornos.

Semana 15: Sesión 2

Desarrollo del proyecto integrador

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 4

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe asignar roles dentro del grupo: líder de proyecto, desarrollador principal, diseñador de interfaz y revisor de código (los roles pueden rotar si el equipo lo considera necesario).
- Utilizar un IDE de su elección (como VS Code o PyCharm) para trabajar en el código proporcionado.
- Desarrollar una aplicación de conversión de monedas que incluya:
    - Una interfaz gráfica intuitiva utilizando tkinter o una biblioteca similar.
    - Lógica de negocio para realizar conversiones entre tres monedas: PEN (soles peruanos), USD (dólares estadounidenses) y EUR (euros).
    - Configuración de un contenedor Docker para garantizar la portabilidad de la aplicación.
- Crear un repositorio en GitHub donde se subirá el código del programa, el Dockerfile, el archivo docker-compose.yml (opcional) y la documentación correspondiente.
- Al finalizar la actividad, cada equipo debe hacer una breve presentación (máximo 5 minutos) donde expliquen los pasos seguidos, los principios aplicados y los resultados obtenidos.

I. Propósito

Al finalizar la sesión, cada estudiante desarrolla una aplicación integradora, combinando el diseño de interfaces gráficas con la lógica de negocio, y configurando un entorno de despliegue utilizando Docker para garantizar su portabilidad y adaptabilidad.

II. Descripción de la actividad por realizar

En esta actividad, los estudiantes trabajarán de forma colaborativa para desarrollar una aplicación de conversión de monedas como parte de su proyecto integrador. La aplicación permitirá permita convertir entre diferentes monedas utilizando una API de tipo de cambio. El objetivo es

combinar el diseño de interfaces gráficas con la lógica de negocio, además de configurar un entorno de despliegue utilizando Docker para garantizar la portabilidad y adaptabilidad de la aplicación.

1. Tecnologías sugeridas:
    - Backend: Python (FastAPI/Flask)
    - Frontend: Tkinter o HTML/CSS/JavaScript
    - API externa: ExchangeRate-API o similar
    - Pruebas: Pytest
2. Funcionalidades principales:
    - Convertir entre monedas seleccionadas por el usuario.
    - Mostrar tasas de cambio actualizadas.
    - Interfaz gráfica o web intuitiva.
    - Manejo de errores (por ejemplo, divisas no soportadas).

El ejercicio se divide en tres partes principales:

1. Diseño de la Interfaz Gráfica: Crear una interfaz gráfica intuitiva utilizando tkinter o PyQT o una biblioteca similar.
2. Implementación de la Lógica de Negocio: Desarrollar la lógica para realizar las conversiones de moneda.
3. Configuración del Entorno de Despliegue: Crear un Dockerfile y opcionalmente un archivo docker-compose.yml para desplegar la aplicación en un contenedor Docker.

Al finalizar, los equipos compartirán sus resultados y reflexionarán sobre cómo la combinación de interfaces gráficas, lógica de negocio y Docker mejora la calidad y portabilidad de la aplicación.

Semana 16: Sesión 2

Demostración del proyecto integrador

Sección: ..... Fecha: ...../...../..... Duración: 60 minutos Docente: ..... Unidad: 4

Nombres y apellidos: .....

Instrucciones

- Los estudiantes deben organizarse en equipos de 3 a 5 integrantes.
- Cada equipo debe preparar una presentación y demostración práctica de su proyecto integrador, asegurándose de que todos los miembros participen activamente.
- Utilizar un IDE de su elección (como VS Code o PyCharm) para mostrar el código del proyecto durante la demostración.
- Preparar una breve presentación (máximo 15 minutos) que incluya:
    a. Explicación del proceso de diseño y desarrollo.
    b. Demostración práctica del funcionamiento de la aplicación.
    c. Justificación de las decisiones técnicas tomadas.
- Crear un repositorio en GitHub donde se subirá el código completo del proyecto, documentación y cualquier recurso adicional necesario.
- Al finalizar la actividad, cada equipo debe compartir el enlace del repositorio con al docente y responder preguntas sobre el proyecto.

I. Propósito

Al finalizar la sesión, cada estudiante presenta y demuestra una aplicación integradora, explicando el proceso de diseño, desarrollo y despliegue, y defendiendo las decisiones técnicas tomadas para garantizar la usabilidad, portabilidad y adaptabilidad del proyecto.

III. Descripción de la actividad por realizar

En esta actividad, los estudiantes realizarán una demostración práctica de su proyecto integrador desarrollado previamente. El objetivo es presentar el proyecto de manera clara y concisa, destacando cómo se combinaron diferentes aspectos del desarrollo de software, como el diseño de interfaces gráficas, la lógica de negocio y la configuración de entornos de despliegue. Además, los estudiantes deberán justificar las decisiones técnicas tomadas para garantizar la usabilidad, portabilidad y adaptabilidad de la aplicación.

1. La demostración debe ser interactiva y centrarse en tres aspectos principales:
    a. Proceso de Diseño y Desarrollo: Explicar cómo se planificó y desarrolló el proyecto, incluyendo las herramientas y tecnologías utilizadas.
    b. Demostración Práctica: Mostrar el funcionamiento de la aplicación en tiempo real, destacando sus funcionalidades clave.
    c. Justificación de Decisiones Técnicas: Defender las decisiones tomadas para garantizar la calidad del proyecto, como la elección de tecnologías, patrones de diseño y configuración de Docker.

Al finalizar, los equipos recibirán retroalimentación del instructor y reflexionarán sobre cómo mejorar futuros proyectos.

Referencias

Chacon, S., & Straub, B. (2014). Pro Git (2da ed.). https://git-scm.com/book/es/v2.

GitHub. (s.f.). Documentación de GitHub . https://docs.github.com/es.

Python Software Foundation. (2023). Python 3.13.2 documentation. https://docs.python.org/3/?spm=5aebb161.2ef5001f.0.0.14b05171am3mcQ.