Ejercicios en Clase

Ejercicio 1: Mejora de Nombres de Variables y Funciones

n1: numProductos
r: resultadoFinal
calcTotal(): cantidadTotal()
tempData: datosTemporales
x: coordenadaX
y: coordenadaY
procesar(): procesarDatosCliente()
res(): obtenerResultado()

Ejercicio 2: Nomenclatura y Formatos

	snake_case	***CamelCase***	***Kebab-case***
Total ventas	total_ventas	totalVentas	total-ventas
Calcular precio descuento	calcular_precio_descuento	calcularPrecioDescuento	calcular-precio-descuento
costoEnvio	costo_envio	costoEnvio	costo-envio

Ejercicio 3: Investigación de Roles de Programación

1. Backend Developer

Responsabilidades Principales:

- Diseñar, construir y mantener la lógica del lado del servidor.
- Crear y gestionar bases de datos.
- Asegurar la seguridad, escalabilidad y eficiencia del sistema.
- Integrar servicios de terceros y manejar autenticación/autorización.
- Escribir pruebas automatizadas para garantizar la estabilidad del código.

Contribución al Ciclo de Desarrollo:

- Participa en las fases de desarrollo y mantenimiento.
- Proporciona la base funcional que el frontend utiliza.
- Trabaja junto a frontend developers y DevOps para asegurar la entrega fluida del producto.

2. Data Scientist

Responsabilidades Principales:

- Analizar grandes volúmenes de datos para descubrir patrones y tendencias.
- Desarrollar modelos predictivos o de machine learning.
- Preparar y limpiar conjuntos de datos.
- Visualizar datos y comunicar hallazgos a stakeholders.
- Colaborar con ingenieros para integrar modelos en aplicaciones.

Contribución al Ciclo de Desarrollo:

- Apoya en las etapas de análisis de requerimientos y mejora del producto.
- Ayuda a tomar decisiones basadas en datos.
- En entornos con productos basados en inteligencia artificial, sus modelos son una parte clave del sistema.

3. QA Engineer

Responsabilidades Principales:

- Diseñar, ejecutar y automatizar pruebas para validar la funcionalidad del software.
- Identificar, reportar y hacer seguimiento de errores o inconsistencias.
- Verificar que el software cumpla con los requisitos técnicos y de negocio.
- Trabajar con desarrolladores para prevenir errores antes de que lleguen a producción.
- Documentar casos de prueba y escenarios de uso.

Contribución al Ciclo de Desarrollo:

- Participa en las etapas de testing e integración.
- Garantiza la calidad del software antes del lanzamiento.
- Previene errores costosos mediante pruebas tempranas y frecuentes.

4. Frontend Developer

Responsabilidades Principales:

- Diseñar e implementar la interfaz de usuario (UI) que interactúa con el usuario final.
- Traducir maquetas/diseños (por ejemplo, de Figma) a código funcional usando HTML, CSS y JavaScript (o frameworks como React, Angular, Vue).
- Consumir APIs del backend y presentar los datos de forma intuitiva y accesible.
- Asegurar la accesibilidad, usabilidad y compatibilidad en distintos navegadores/dispositivos.
- Optimizar el rendimiento de carga y la experiencia de usuario.

Contribución al Ciclo de Desarrollo:

- Trabaja activamente en la fase de desarrollo e iteración de interfaces.
- Colabora con diseñadores UX/UI y backend developers.
- Aporta al valor del producto mediante una experiencia de usuario atractiva, fluida y funcional.

Ejercicio 4: Pseudocódigo para Calcular el Área de una Figura

//Creo una funcion donde le paso como parametro las variables de base y de altura
funcion calcularAreaTriangulo(base, altura):
//Creo una variable donde calculo el area del triangulo mediante su respectiva formula
    areaTriangulo = base * altura / 2
//Devuelvo el valor del area del triangulo
    retornar areaTriangulo
//Muestro un mensaje donde se ve el calculo del area del triangulo segun dos numeros dados
mostrarAreaTriangulo(“El area del triangulo es: ” + calcularAreaTriangulo(2, 4)

Ejercicio 5: Mejorar Fragmento de Código

```
suma_total = 0
para cada numero en lista_de_valores:
   suma_total += numero
si suma_total > 100
   mostrar("Total alto")
si_no
   mostrar("Total bajo")
```

Ejercicios en Casa

Tarea 1: Renombrar Variables usando Diferentes Nomenclaturas

	snake_case	*CamelCase*	*Kebab-case*
precio-total	precio_total	precioTotal	precio-total
costo-envio-extra	costo_envio_extra	costoEnvioExtra	costo-envio-extra
impuesto-añadido	impuesto_añadido	impuestoAñadido	impuesto-añadido

Tarea 2: Investigación Profunda de Roles en Programación

DevOps Engineer (Ingeniero DevOps)

Responsabilidades Principales:

- **Automatización del ciclo de vida del software:** Crear pipelines de integración continua y entrega continua (CI/CD) para facilitar la construcción, prueba y despliegue automático del código.
- **Gestión de infraestructura:** Configurar y mantener servidores, redes, contenedores (Docker) y servicios en la nube (AWS, Azure, GCP).
- **Monitoreo y observabilidad:** Implementar herramientas que permitan vigilar el rendimiento y la estabilidad del sistema en tiempo real (ej. Prometheus, Grafana).
- **Gestión de la configuración y aprovisionamiento:** Automatizar el entorno de producción usando herramientas como Ansible, Terraform o Chef.
- **Colaboración entre equipos:** Actuar como puente entre desarrollo y operaciones para mejorar la eficiencia del proceso y la calidad del producto final.
- **Seguridad en el despliegue:** Asegurar que los entornos sean seguros y que los despliegues no expongan vulnerabilidades.

Habilidades Técnicas Necesarias:

- **Lenguajes de scripting:** Bash, Python, YAML (para automatizaciones y configuración).
- **Herramientas CI/CD:** Jenkins, GitLab CI, GitHub Actions.
- **Contenedores y orquestación:** Docker, Kubernetes.
- **Cloud computing:** Experiencia con plataformas como AWS, Azure o Google Cloud.

- **Infraestructura como código:** Terraform, CloudFormation.
- **Sistemas operativos:** Administración avanzada de sistemas Linux.
- **Monitoreo y logging:** ELK Stack (Elasticsearch, Logstash, Kibana), Datadog, Prometheus.

Tarea 3: Algoritmo para Gestionar una Lista de Tareas

```
lista_de_tareas = []

función añadir_tarea(tarea):
   lista_de_tareas.agregar({tarea, falso})

función completar_tarea(indice):
   lista_de_tareas[indice][1] = verdadero

función eliminar_tarea(indice):
   eliminar lista_de_tareas[indice]

función mostrar_tareas():
   para cada i en rango(0, longitud(lista_de_tareas)):
      tarea = lista_de_tareas[i][0]
      estado = lista_de_tareas[i][1]
      si estado:
         mostrar(i + ". " + tarea + " [Completada]")
      si_no:
         mostrar(i + ". " + tarea + " [Pendiente]")
```

Tarea 4: Documentación sobre Buenas Prácticas

La programación no es solo escribir código que funcione, sino escribir código **claro, mantenible y comprensible** para otros desarrolladores (y para uno mismo en el futuro). A continuación, se presentan tres prácticas fundamentales que mejoran la calidad del software: el uso de nomenclatura descriptiva, la modularidad y los comentarios adecuados.

1. Nomenclatura Descriptiva

¿Por qué es importante?

Usar nombres claros y descriptivos para variables, funciones y clases permite que el código sea **autoexplicativo**. Esto reduce la necesidad de comentarios innecesarios y facilita que otros desarrolladores comprendan rápidamente qué hace cada parte del código.

2. Modularidad

¿Qué es?

La modularidad consiste en **dividir el código en funciones o módulos pequeños**, cada uno con una

responsabilidad clara. Esto mejora la organización, facilita las pruebas y permite la reutilización del código.

Ventajas:

- El código es más fácil de entender y depurar.
- Se puede reutilizar una función en distintos lugares.
- Fomenta la colaboración en equipos (cada miembro trabaja en módulos distintos).

3. Uso Adecuado de Comentarios

¿Cuál es el objetivo?

Los comentarios deben **explicar el "por qué"** de una decisión de programación, no el "cómo", que ya debe deducirse del código bien escrito. Los comentarios mal utilizados (o excesivos) pueden volverse obsoletos y confundir más que ayudar.

Buenas prácticas:

- Usa comentarios para explicar lógica compleja o decisiones no obvias.
- Evita comentarios obvios o redundantes.
- Usa comentarios para marcar TODOs o advertencias (`# TODO: mejorar rendimiento`).

Tarea 5: Identificación de Problemas en Pseudocódigo

1.- Nombres de variables poco descriptivos (a,b): No se entiende que representan ambas variables y esto dificulta la lectura del código. Cambiaria las variables a: precioUnitario y precioTotal.
precio_unitario = 10
precio_total = precio_unitario * 2

2.- Falta de contexto del codigo: El código no explica por qué se realiza esta operación. Por lo que, agregaria un comentario que describa la intención general.
#Calcular el precio total de 2 productos

3.- Impresion de una variable incorrecta o irrelevante: Se imprime "a", pero no el resultado calculado "b" que es el dato relevante. Por mi parte, imprimiría el valor "b"
print(precio_total)

4.- No hay modularización ni encapsulamiento: Todo el codigo esta en un solo bloque y esto impide su reutilización. Para solucionar este problema, encapsularia la lógica en una funcion
función calcular_precio_total(cantidad, precio_unitario):
  retornar cantidad * precio_unitario

5.- Falta de interacción o entrada de datos: El valor esta hardcodeado y no se puede reutilizar con diferentes valores. Yo permitiria la entrada de datos.

```
cantidad = 2
precio_unitario = 10
precio_total = calcular_precio_total(cantidad, precio_unitario)
print(precio_total)
```