Trabajo Práctico:   
Diseño de Casos de Prueba

# Objetivo y evaluación

Analizar una aplicación y diseñar casos de prueba que sean:

* **Efectivos**: Capaces de encontrar problemas en la implementación. Implica Cobertura de pruebas
* **Eficientes**: Logren encontrar problemas con la menor cantidad de pruebas posibles.

Además consideraremos

* Código y documentación clara y completa
* Facilidad para ejecutar y revisar el código

**Fecha entrega: 28 de abril 2025**

# Formato de entrega

## Lenguaje

Los casos de prueba deben entregarse automatizados junto con una implementación de la aplicación.

Puedes desarrollarlo en JavaScript, Python, Ruby. Consulta con el docente para otros lenguajes

## Repo

1. **Repositorio GitHub**: Crea un repositorio privado que contenga tu proyecto.
2. **Estructura del repositorio**:
   * Código fuente de la aplicación
   * Código de los casos de prueba
   * Archivo README.md con instrucciones claras (ver detalles abajo)
   * Archivo de configuración para GitHub Codespaces (devcontainer.json)
3. **README del proyecto**: Debe incluir:
   * Descripción breve de la aplicación implementada
   * Resultado del [Análisis](#_heading=h.b85uvc17g75a)
   * [Casos de prueba](#_heading=h.i9d19t84asn9) diseñados
   * Instrucciones para ejecutar las pruebas
   * Resultados esperados
4. **Ejecución simplificada**: Configura tu proyecto para que pueda ejecutarse con un solo comando (por ejemplo, npm test, pytest, rspec etc.)
5. **Acceso al repositorio**:
   * Invita al docente (usuario: *jgabardini*) como colaborador
   * Asegúrate de que el repositorio esté configurado para funcionar con GitHub Codespaces

## **Notas importantes**

* No se aceptarán entregas por otros medios que no sean el repositorio de GitHub
* El proyecto debe ser ejecutable sin necesidad de instalaciones adicionales
* Si encuentras algún problema con la configuración, consulta con anticipación

# Aplicación: Elephant Carpaccio Light

(Basado en Elephant Carpaccio de Alistair Cockburn)

La aplicación es la versión inicial, minimalista (\*), de un sistema de facturación para USA.

La aplicación se utiliza desde línea de comando. Por ejemplo

https://elephant-server.onrender.com/api/invoice?quantity=10&price=100&state=CA[[1]](#footnote-0)

Como se ve en el ejemplo, acepta 3 datos ingresados por el usuario:

* Cuántos ítems comprados (en el ejemplo, 10)
* Costo unitario (en el ejemplo, 100 usd)
* Código de 2 letras para el estado (en el ejemplo CA)

Salida:

* Subtotal de la compra (cantidad de ítems por el precio unitario)
* Descuento a partir del subtotal (porcentaje y valor)
* Impuestos (porcentaje y valor), sobre el subtotal con descuento aplicado.
* Total: Subtotal menos descuento más impuestos

| **Subtotal (\*)** | **% Descuento (\*)** |  | **Estado** | **Impuesto** |
| --- | --- | --- | --- | --- |
| $1.000 | 3% | UT | 6,85% |
| $5.000 | 5% | NV | 8,00% |
| $7.000 | 7% | TX | 6,25% |
| $10.000 | 10% | AL | 4,00% |
| $50.000 | 15% | CA | 8,25% |

(\*) Por ser inicial, minimalista:

* No se implementan descuentos, priorizamos cumplimiento legal / impositivo.
* Solo implementamos en algunos estados, porque son los que tienen mayor volumen operado (CA, TX, AL, NV, UT).

# Análisis

1. Identifica las variables de entrada, internas y salida. Y los valores aceptables para cada una de ellas

| E/I/S | Variable | Valores aceptables |
| --- | --- | --- |
| E | Cantidad de items | Naturales>=1 |
| E | Precio unitario | Reales positivos |
| E | Codigo del estado | UT, NV, TX, Al, CA |
| S | Total a pagar | Real positivo |

1. Elige una variable de cada tipo (entrada, interna, salida). y analiza las clases de equivalencia.

| Variable | Clase equivalencia | Válido | Valor | Observación / Qué se prueba |
| --- | --- | --- | --- | --- |
| Precio\_Unitario | Item | si | 51.072 | Que se inicia correctamente con un precio valido |
| cantidad | Item | Si | 2 | Que se inicia correctamente con una cantidad entera válida |
| Cantidad | Item | No | -1 | testInicializacionIncorrecta testea que se lance la excepción si se inicializa con un valor incorrecto |
| Codigo | StateCode | No | “JU” | en este test se verifica que solo se setee el valor correcto, y si este es correcto pone la tasa automáticamente |
| Codigo | StateCode | Si | “TX” | al crearse la instancia por default pone los valores como si estuviese en el estado “UT” con una tasa se 6,85%, al cambiarse por “TX” se cambia a 6,25% |
| items | Factura | Si |  | el testCorrectoCambioDeItems testea que se cambia la lista de manera correcta siempre y cuando no sea vacia |
| Items | Factura | si |  | testCorrectoAgregadoDeItemAFactura mira si se agrega correctamente a la lista de item, se agrego primero los 2 pares de borcegos 46 y luego las zapatillas 38 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Casos de prueba

| Cantidad | Precio  unitario | Estado | Resultados | Qué se prueba |
| --- | --- | --- | --- | --- |
| 2 | 51,072 | Valido | 102,144 | Que se calcula correctamente el valor del item |
| 1 | 17,336 | Valido | 17,336 | Que se inicializa correctamente otro ítem para agregarlo a la factura |
| -1 | 51,072 | Invalido | 51,072 | que se ignora correctamente un valor fuera del rango de la variable, en este caso |
| 1 | -17,336 | invalido | excepcion | se debería tirar una excepción por parametro invalido ya que no hay valores negativos |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Comentarios adicionales:

se armaron test que se setea correctamente el valor nuevo en la instancia de item, si es nulo o blanco, se debería lanzar una excepción, al igual que cuando se ingresan valores fuera del rango

se asocia, para simplificar, el impuesto a la factura y no a cada item, se habia pensado inicialmente asociarlo a cada item pero se desecho esta idea.

1. La aplicación puede tardar hasta un minuto para la primera respuesta. Está en un plan gratuito de Render. [↑](#footnote-ref-0)