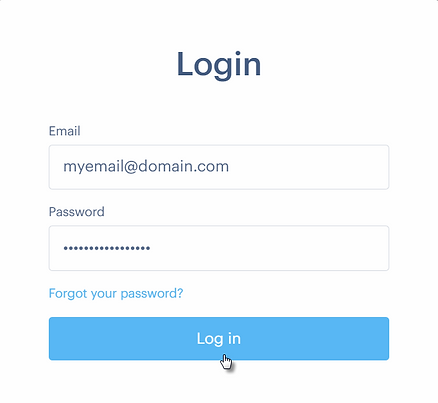
**Cómo hacer casos de prueba**

Aprende a escribir tus casos de prueba de la manera más efectiva.



Un caso de prueba es un conjunto de acciones que se ejecutan para verificar una característica o funcionalidad particular de una aplicación de software.



Si se toma esta imagen como ejemplo se pueden observar distintos casos de prueba a simple vista.

Uno de los casos de prueba consiste en:

***Pasos***

1. Insertar un email válido

2. Insertar una contraseña válida

3. Hacer click en el botón de login

***Resultados esperados***

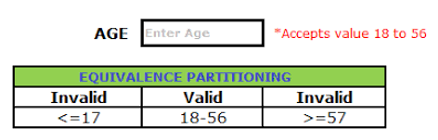
1. Que el usuario sea redirigido a la página principal de la aplicación.

Con el concepto de *"caso de prueba"* bien aprendido es tiempo de conocer cuál debe ser el contenido a tomar en cuenta para poder concluir que un caso de prueba está completo.

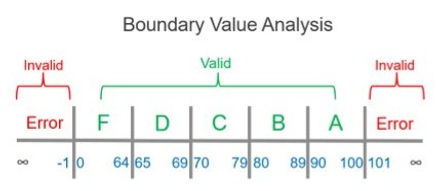
*¿Qué debe contener un caso de prueba?*

* **Identificador:** Puede ser numérico o alfanumérico (la mayoría de herramientas lo generan solo).
* **Nombre del caso de prueba:** Debe ser conciso. Se debe utilizar una nomenclatura que esté definida, pero si no existe una, lo recomendable es incluir el nombre del módulo al que corresponde el caso de prueba.
* **Descripción:** La descripción debe decir qué se va a probar, en algunos casos, en esta sección se incluye el ambiente de pruebas, la data y las pre-condiciones o asunciones.
* **Pasos:** Son las acciones que se deben realizar para obtener los resultados.
* **Resultados esperados:** es lo que le indica al tester cual debería ser la experiencia luego de ejecutar los pasos y así determinar si el test falló o pasó.
* **Estado y resultados actuales:** No necesariamente van dentro del caso de prueba. Muchas herramientas incluyen estas informaciones asociadas al caso de prueba, o como un récord aparte.
* **Los casos de prueba deben ser simples:** Se deben crear casos de prueba que sean lo más simple posible ya que otra persona que no sea el autor puede ejecutarlos.​ Use un lenguaje asertivo como *“ir a la página de inicio”*, *“ingresar datos”*, *“hacer clic en esto”*, etc. Esto facilita la comprensión de los pasos de prueba y hace que la ejecución sea más rápida.
* **El título debe ser fuerte:** La manera correcta de comenzar con el título de un caso de prueba es con un verbo en infinitivo. Los verbos en infinitivo denotan un mandato.
* **Tome al usuario final en cuenta:** El objetivo final es crear casos de prueba que cumplan con los requisitos del cliente y que sean fáciles de usar. Un tester debe crear casos de prueba tomando en cuenta la perspectiva del usuario final.
* **Evite la repetición de casos de prueba​:** Si se necesita un caso de prueba para ejecutar algún otro, llame al caso de prueba por su id. Inclúyalo en la columna de pre-condiciones o donde corresponda dependiendo la herramienta.
* **No asuma:** No asuma la funcionalidad y las características de la aplicación mientras prepara el caso de prueba. Mire a los documentos de especificación y si tiene alguna duda, pregunte.
* **Asegura la mayor cobertura posible:** Asegúrese de escribir casos de prueba para todos los requisitos especificados. ​Algo que se puede utilizar es una [matriz de trazabilidad](https://obsbusiness.school/es/blog-project-management/herramientas-esenciales/cual-es-la-utilidad-de-la-matriz-de-trazabilidad) para garantizar que se prueben todos los casos de pruebas asociados. La mayoría de herramientas permiten vincular los casos de prueba entre si. ​
* **Autonomía:** El caso de prueba debe generar los mismos resultados cada vez, sin importar quién lo pruebe.
* **Implementa técnicas de prueba:** No es posible verificar todas las condiciones posibles de una aplicación, pero las técnicas de prueba ayudan a seleccionar los casos de prueba con la máxima posibilidad de encontrar un defecto.​

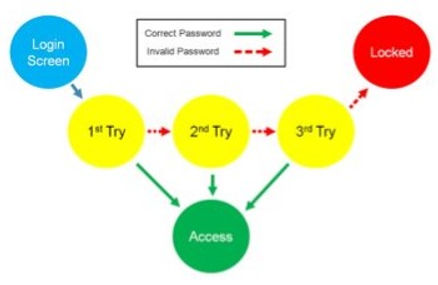
*Partición de equivalencia (EP):* esta técnica divide el rango en partes o grupos iguales que tienden a tener el mismo comportamiento.​



*Análisis de valor de límite (BVA)*: como su nombre lo indica, es la técnica que define la prueba de límites para un rango específico de valores.​



*Técnica de transición de estado:* este método se utiliza cuando el comportamiento del software cambia de un estado a otro después de una acción particular.



# Técnicas de prueba de caja negra

Las *técnicas de prueba* son mecanismos que los testers utilizan con el objetivo de identificar los casos de prueba con mayor probabilidad de encontrar defectos y obtener la mayor cobertura posible en cuanto a las pruebas de algún sistema. Cuando hablamos de técnicas de *caja negra*, esto significa que al poner en práctica estas técnicas, solo nos concentraremos en los datos de entrada y los resultados obtenidos en la prueba, sin ver la estructura interna de la funcionalidad.

¿Por qué utilizar técnicas de pruebas de caja negra?

1. Ayudan a determinar los casos de prueba negativos de una manera más eficaz.
2. Hacen más visible los casos de prueba con la mayor probabilidad de tener defectos.
3. Ahorra el diseño y la ejecución de pruebas redundantes.
4. Ayudan a medir el porcentaje de cobertura de las pruebas.

Veamos algunas de estas técnicas de prueba de caja negra y como se manifiestan los beneficios listados arriba.

***Partición de equivalencia (equivalence partitioning):*** en esta técnica se dividen los datos de entrada en particiones o clases de equivalencia, los datos que producen el mismo resultado son agrupados en una misma clase, es decir, se espera que todos los miembros de una partición sean procesados de la misma manera. Algunas consideraciones a tener en cuenta con esta técnica:

* Cada dato de entrada de debe pertenecer a una sola *partición* o *clase de equivalencia*.
* Los *valores válidos* son los que se supone que el sistema debe aceptar. Las particiones que contienen los valores válidos son llamadas *"partición de equivalencia válida".*
* Similar al punto anterior, los *valores* que son rechazados por un sistema son llamados *valores no válidos* y las particiones que los contienen se conocen como *"partición de equivalencia no válida".*
* Cada una de las *particiones de equivalencia no válidas* son un caso de prueba diferente y deben probarse de manera independiente, no deben combinarse como un solo caso de prueba solo por el simple hecho de que son *inválidas*.

Ejemplo:

Imaginemos que tenemos un sistema que **solo permite el acceso a personas de 18 a 60 años de edad** (para fines de este ejemplo vamos a obviar la existencia de números negativos) y el sistema le pide al usuario que introduzca su edad, si el usuario tiene *menos de 18* años o *más de 60* no permite el acceso y lanza un mensaje dependiendo de la edad, a los menores de 18 les dice: *"No puede acceder, su edad es* ***menor*** *a la requerida"* y a los mayores de 60: *"No puede acceder, su edad es* ***mayor*** *a la requerida".* Veamos las particiones de equivalencia:

| ***​Partición no válida*** | ***​Partición válida*** | ***​Partición no válida*** |
| --- | --- | --- |
| <=17 | ​18-60 | ​>=61 |

Ya que el sistema debe comportarse de la misma manera para todos los menores de 18 años, todas esas edades se agrupan en una misma partición y así igual con los otros dos grupos. Para lograr un 100% de cobertura los casos de prueba deben cubrir cada una de las particiones identificadas, utilizando como mínimo un valor de cada partición para las pruebas. Esto quiere decir que utilizando solo tres valores (uno de cada partición) ya probamos todos los casos. Aquí vemos dos de los beneficios mencionados arriba:

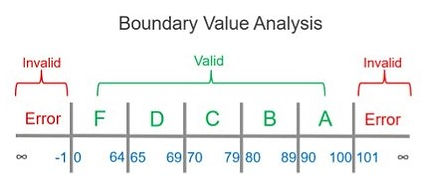
1. Nos ahorrarnos la ejecución de pruebas redundantes, por ejemplo, sería lo mismo probar con 18 que probar con 25, o probar con 1 que probar con 17.
2. Podemos medir el porcentaje de cobertura fácilmente, por ejemplo, si probamos con 15, 20 y 62 ya tenemos el 100% de cobertura.

***Análisis de valor de límite (Boundary Value Analysis):*** Esta técnica es una extensión de la partición de equivalencia, solo se utiliza para valores numéricos y cuando los valores se encuentran ordenados. Una partición contiene un valor inicial y un valor final (un número mínimo y uno máximo) y son denominados como valores límite.

A diferencia de la partición de equivalencia, que toma como mínimo un número dentro del rango de la partición como dato de entrada, en esta técnica obligatoriamente deben tomarse como mínimo los dos valores límite y opcionalmente, un valor entre los límites. Veamos un ejemplo:

Para este sistema de calificaciones cada letra representa un rango (F=0-64, D=65-69, C=70-79, B=80-89, A=90-100), los valores válidos van de 0 a 100, los valores menores que 0 representan una partición inválida, al igual que los valores mayores a 100.

Para obtener una cobertura de 100% el usuario debe introducir al menos todos los valores que se encuentran en azul en la imagen siguiente:



Esta técnica surge como respuesta a un caso muy común: es muy probable encontrar defectos en los valores límites, siguiendo el ejemplo dado, puede que se haya cometido un error al programar uno de los rangos de calificaciones y que 65 se registe como F en el sistema.

Aquí nuevamente vemos algunos de los beneficios que se mencionan más arriba:

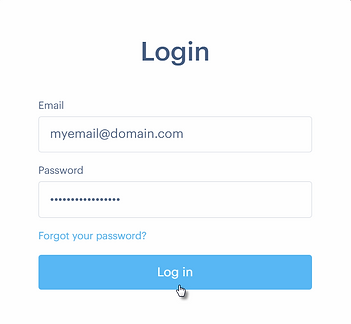
1. Ayudan a determinar los escenarios negativos de una manera más eficaz: vemos claramente cuales son las particiones inválidas.
2. Hacen más visible los casos de prueba con la mayor probabilidad de tener defectos: en los límites se tiende a encontrar más defectos que en el medio de un rango.

***Tabla de decisión (Decision table):*** Es una técnica utilizada para probar el comportamiento de un sistema utilizando diferentes combinaciones de entrada. Este es un enfoque donde las diferentes combinaciones de entrada y los resultados se muestran en forma tabular.

* En las filas se colocan las condiciones y los resultados, donde las condiciones se ponen en la parte superior y los resultados en la parte inferior.
* Cada columna corresponde a una regla la cual es definida por las combinaciones de entrada.
* Los valores a colocarse son generalmente booleanos como V (verdadero) y F (Falso). Si hay un valor que no importa para obtener cierto resultado se puede marcar con un guion "-".

Ejemplo:

Vamos a crear una tabla de desición para una página de inicio de sesión.



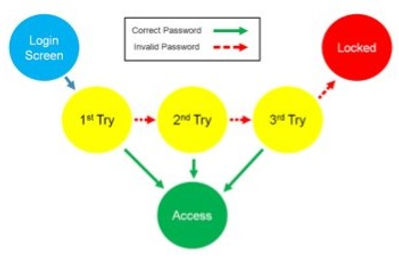
Las condiciones son simples, si se proporciona un usuario y una contraseña válida, el usuario puede iniciar sesión exitosamente, de lo contrario aparecerán mensajes de error. Para la siguiente tabla **E=Error**, **S=Inicio de sesión exitoso**.

| **Condiciones** | **​Regla 1** | **Regla 2** | **Regla 3** | **Regla 4** |
| --- | --- | --- | --- | --- |
| Usuario válido (V/F) | ​F | V | F | V |
| Contraseña válida (V/F) | F | F | V | V |
| **Resultados** | **​E** | **E** | **E** | **S** |

Ya con esta tabla fuimos capaces de generar 4 casos de pruebas para probar una página de inicio de sesión.

***Técnica de Transición de Estado (State Transition Testing):*** En esta técnica las condiciones de entrada provocan cambios de estado. Es ideal para cuando el sistema bajo prueba dependen de eventos o valores pasados y para cuando se debe probar un sistema que consta de una secuencia de eventos. Ejemplo:

Utilizando el mismo de inicio de sesión de la técnica anterior, supongamos que si una persona introduce una contraseña incorrecta tres veces seguidas el usuario se bloquea. Veamos cómo se representa esto en un diagrama de transición de estado.



La cobertura para esta técnica se mide, habitualmente, como el número de estados o transiciones identificados **probados**, dividido por el número total de estados o transiciones identificados en el **sistema**,y es normalmente expresado como un porcentaje.

# Técnicas de prueba de caja blanca

Las técnicas de prueba son mecanismos que los testers utilizan con el objetivo de identificar los casos de prueba con mayor probabilidad de encontrar defectos y obtener la mayor cobertura posible en cuanto a las pruebas de algún sistema.

Las técnicas de pruebas de caja blanca son aquellas que se basan en el conocimiento interno del código de un programa o sistema. Estas técnicas son muy útiles para encontrar errores en el código y garantizar que no exista [código muerto](https://es.wikipedia.org/wiki/C%C3%B3digo_muerto#:~:text=En%20programaci%C3%B3n%2C%20se%20conoce%20como,algo%20que%20jam%C3%A1s%20se%20utiliza) o algún aspecto del sistema sin probar.

A continuación, veamos algunas de las técnicas de pruebas de caja blanca más utilizadas.

#### 

#### Técnica de cobertura de sentencia

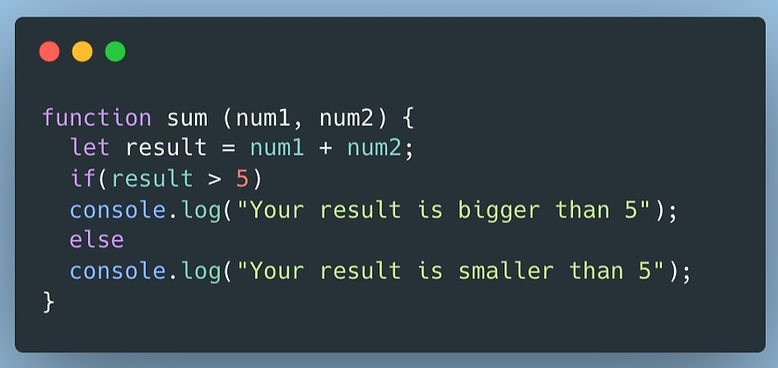
La cobertura de sentencia es una técnica cuyo objetivo principal es asegurarse de que todas las sentencias de código del programa han sido ejecutadas al menos una vez durante las pruebas.

Cuando hablamos de una sentencia de código nos referimos a cualquier instrucción que el programa puede ejecutar, como declaraciones de variable, asignaciones, llamadas de función, bucles y condicionales.

La prueba de cobertura de sentencia es importante porque ayuda a identificar las sentencias que no han sido ejecutadas. Las sentencias no ejecutadas pueden ser un signo de código muerto o de secciones de código que no se utilizan. También puede ser un signo de que algunas funcionalidades del programa no se han probado adecuadamente.

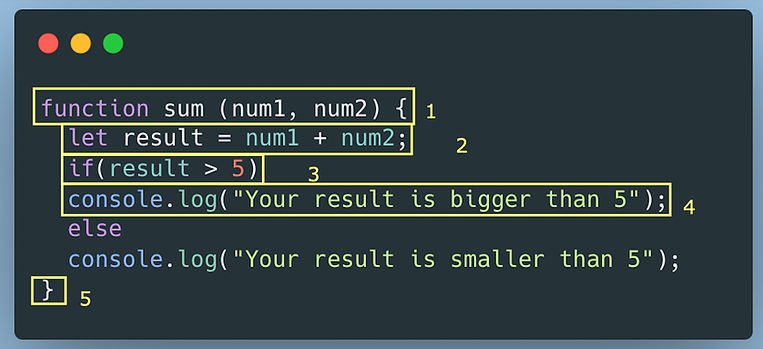
La cobertura se mide como el número de sentencias ejecutadas entre el número total de sentencias ejecutables en el sistema y generalmente se expresa como un porcentaje. Veamos un ejemplo:

Dado el siguiente bloque de código, vamos a calcular cuantos casos de prueba se necesitan para alcanzar el 100% de cobertura.



Lo primero que debemos hacer es contar las sentencias, en este caso, vamos a decir que cada línea de código es una sentencia. Entonces para este ejemplo tenemos 7 sentencias.

Si num1 = 4 y num2 = 6 entonces la variable result será igual a 10 y por tanto entraremos en la primera condición.



En este caso solo 5 sentencias han sido ejecutadas utilizando el primer caso de prueba, lo que significa que la cobertura es igual a 5/7 = 0.71, lo que equivale a un **71% de cobertura.**

Si queremos alcanzar un 100% de cobertura necesitamos un 2do caso de prueba, este caso de prueba podría ser num1 = 3 y num2 = 0.

Este 2do caso de prueba ejecutado por sí solo también representa un 71% de cobertura, ya que también equivale a la ejecución de 5 sentencias, sin embargo, si combinamos el 1er caso de prueba con el 2do, fácilmente alcanzamos un 100% de cobertura ya que las 7 sentencias habrán sido ejecutadas.

En conclusión, para este ejemplo se necesitan **2 casos de prueba** para alcanzar el **100% de cobertura de sentencias**.

#### Técnica de cobertura de decisión

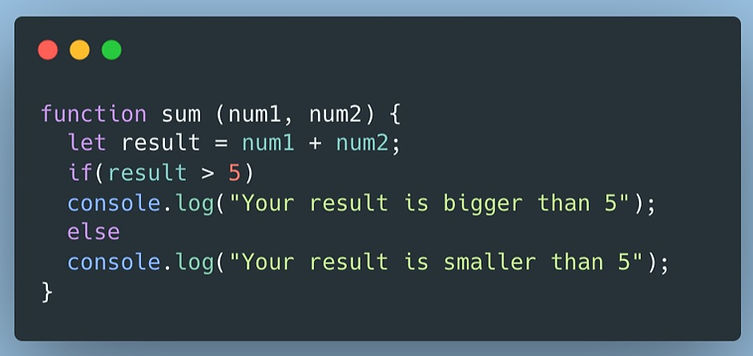
La técnica de cobertura de decisión se utiliza para medir la cantidad de decisiones en una estructura de control que se han probado. El objetivo principal de esta técnica es asegurarse de que todas las posibles opciones en una estructura de control han sido probadas.

Una estructura de control es cualquier sección de código que controle el flujo de ejecución del sistema. Esto puede incluir estructuras como condicionales (if-else), bucles (for, while) y switch statements. Cada una de estas estructuras de control pueden tener múltiples opciones o caminos, y la prueba de cobertura de decisión se utiliza para asegurarse de que cada una de estas opciones haya sido probada.

Una decisión se refiere a cualquier opción o camino que puede tomar la estructura de control, como un camino verdadero o falso en un condicional.

La cobertura es medida calculando el número de resultados de decisión ejecutados entre el número total de resultados de decisión que haya en el sistema y normalmente se expresa en porcentaje. Veamos un ejemplo:

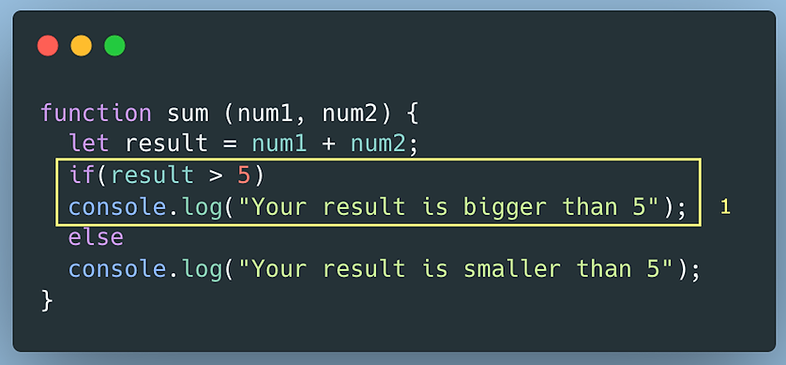
Utilizando el mismo ejemplo de arriba, vamos a calcular cuantos casos de prueba se necesitan para alcanzar un 100% de cobertura.



Lo primero que debemos hacer es contar las decisiones. En este caso, tenemos una estructura if-else con dos decisiones:

1. Si result > 5
2. Si result <= 5

Si num1 = 4 y num2 = 6 entonces la variable result será igual a 10 y por tanto entraremos en la primera condición.



Solo una de dos decisiones ha sido ejecutada, esto significa que la cobertura es de 1/2 = 0.5, lo que equivale a un **50% de cobertura**.

Para obtener un 100% necesitamos otro caso de prueba que nos ayude a poder ejecutar la segunda decisión. Al igual que en el ejemplo anterior este caso de prueba puede ser num1 = 3 y num2 = 0. De esta manera ambas decisiones serían ejecutadas y la cobertura sería de 100%.

En conclusión, para este ejemplo se necesitan **2 casos de prueba** para alcanzar el **100% de cobertura de decisión**.

Nota:

A pesar de que, para este ejemplo, en ambas técnicas se necesitaron dos casos de prueba para alcanzar el 100% de cobertura, no siempre se da la coincidencia de que se requiera la misma cantidad de casos de prueba para el mismo bloque de código aplicando ambas técnicas.

Es decir, se puede dar el caso de que para el mismo código para alcanzar un 100% cobertura utilizando la cobertura de sentencias se requieran 3 casos de prueba y que para la cobertura de decisión se requieran 4.

Si se alcanza el 100% de cobertura de decisiones, esto significa que también se alcanzó el 100% de cobertura de sentencias, pero alcanzar el 100% de cobertura de sentencias no asegura que se haya alcanzado el 100% de cobertura de decisiones.