Resumen Introducción a la Programación

Contenido

[¿Qué es una computadora? 3](#_Toc173369553)

[¿Qué es un algoritmo? 3](#_Toc173369554)

[¿Qué es un programa? 3](#_Toc173369555)

[Especificación 3](#_Toc173369556)

[Diseño 4](#_Toc173369557)

[Implementación 4](#_Toc173369558)

[Contratos 4](#_Toc173369559)

[Sobre - especificación 5](#_Toc173369560)

[Sub – especificación 5](#_Toc173369561)

[Lenguajes de programación 5](#_Toc173369562)

[Paradigmas de la programación 5](#_Toc173369563)

[Programación declarativa 5](#_Toc173369564)

[Paradigma imperativo 6](#_Toc173369565)

[Lógica proposicional 6](#_Toc173369566)

[Sintaxis 7](#_Toc173369567)

[Semántica clásica 7](#_Toc173369568)

[Tablas de verdad 7](#_Toc173369569)

[Semántica trivaluada 8](#_Toc173369570)

[Tautologías, contradicciones y contingencias 8](#_Toc173369571)

[Equivalencia entre formulas 8](#_Toc173369572)

[Relación de fuerza 8](#_Toc173369573)

[Tipos de Datos 9](#_Toc173369574)

[Tipo Enteros 9](#_Toc173369575)

[Tipo Reales 9](#_Toc173369576)

[Tipo Bool 9](#_Toc173369577)

[Tipo Char (caracteres) 10](#_Toc173369578)

[Tipo enumerados 10](#_Toc173369579)

[Tipo Upla o Tupla 10](#_Toc173369580)

[Secuencias 11](#_Toc173369581)

[Programación Funcional (Haskell) 12](#_Toc173369582)

[Expresión 12](#_Toc173369583)

[Ecuaciones orientadas 12](#_Toc173369584)

[Transparencia referencial 13](#_Toc173369585)

[Modo de ejecución en Haskell 13](#_Toc173369586)

[Haskell: tipos de datos, operaciones y funciones 13](#_Toc173369587)

[Programación Imperativa (Python) 14](#_Toc173369588)

[Modo de ejecución 14](#_Toc173369589)

[Estados 15](#_Toc173369590)

[Variables 15](#_Toc173369591)

[Ciclos 16](#_Toc173369592)

[Datos en Python 17](#_Toc173369593)

[Testing 19](#_Toc173369594)

[Niveles y conceptos del testing 19](#_Toc173369595)

[Criterios para seleccionar datos de prueba 20](#_Toc173369596)

[Método de partición de categorías 21](#_Toc173369597)

[Control Flow Graph 22](#_Toc173369598)

[Criterios de adecuación 22](#_Toc173369599)

# ****¿Qué es una computadora?****

Es una máquina electrónica que procesa datos automáticamente de acuerdo con un programa almacenado en memoria.

# ****¿Qué es un algoritmo?****

Es la descripción de los pasos precisos para resolver un problema a partir de datos de entrada adecuados.

Nos indica como resolver el problema. Por ejemplo, la receta de algo.

# ¿Qué es un programa?

Es la descripción de un algoritmo en lenguaje de programación.

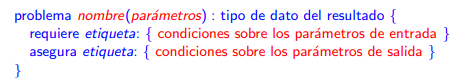
Indica como se resuelve el problema en lenguaje de programación.

# Especificación

Es la descripción del problema a resolver, generalmente en leguaje formal.

Nos dice que problema tenemos y las propiedades de los datos de entrada y salida.

Cada especificación tiene la siguiente estructura general:



* Nombre: es el nombre que le damos al problema y que será resuelto por una función con el mismo nombre.
* Parámetros: son los datos de entrada separados por comas y donde se indica el tipo de dato seguido de dos puntos. Ejemplo de parámetros
  + Problema nombre (n:Int, s:seq(Int)): Int{

…

}

* Requieres: describen todas las condiciones y posibles valores de los datos de entrada.
* Aseguras: describen todas las condiciones y posibles valores de los datos de salida.
* Etiquetas: nombres declarativos opcionales de las condiciones de los requiere y asegura.

# Diseño

Fase de diseño de una solución verificable respecto a la especificación.

Nos dice como resolvemos el problema. Hay dos estrategias comunes para encararlo:

* Top Down: dado un problema complejo descomponerlo en problemas más sencillos de resolver.
* Bottom Up: dado varios problemas sencillos componerlos hasta obtener la solución al problema original.

# Implementación

Implementación de un programa que debería ser solución a nuestro problema, a su vez este es verificable respecto a su especificación y diseño.

# Contratos

Una especificación es un contrato entre el programador de una función y el usuario de esa función.

Desde este punto de vista los requiere son llamados precondiciones ya que especifica lo que requiere la función y los asegura son llamados postcondiciones ya que indican lo que la función asegura que se va a cumplir después de llamarla.

## Sobre - especificación

Consiste en dar una postcondición más restrictiva o una precondición más laxa. Limita los posibles algoritmos para resolver el problema.

## Sub – especificación

Consiste en dar un pre – condición más restrictiva o una post – condición más débil. Permite soluciones no deseadas.

# Lenguajes de programación

Son el conjunto de instrucciones a través del cual los humanos interactúan con las computadoras y permiten escribir programas.

Existen 3 tipos de grandes grupos en los que se clasifican los lenguajes de programación:

* Lenguaje maquina: código binario, cadenas de ceros y unos directamente inteligible por la máquina.
* Lenguaje de bajo nivel: ejercen un control directo sobre el hardware de la máquina.
* Lenguaje de alto nivel: diseñados para que las personas puedan escribir y entender más fácilmente los programas.

Para estos se utilizan un editor de texto y un compilador o interprete (traducen el lenguaje de alto nivel a lenguaje maquina), o IDEs que son la combinación de ambos.

# Paradigmas de la programación

Existen dos grandes grupos a la hora de programar.

## Programación declarativa

Este paradigma se basa en declarar el resultado deseado y no como obtenerlo.

Tenemos dos paradigmas comunes:

* Paradigma lógico: los programas se construyen en base a expresiones lógicas.
* Paradigma funcional: basado en el modelo matemático de composición funcional.

## Paradigma imperativo

Describe la programación como una secuencia de instrucciones o comandos que cambian el estado de un programa.

A si vez existen dos grandes grupos de paradigmas:

* Paradigma estructurado: los programas se dividen en bloques (procedimientos y funciones) que pueden o no comunicarse entre sí.
* Paradigma orientado a objetos: se basa en la idea de encapsular estado y comportamiento en objetos, los cuales son entidades que se comunican entre sí por medio de mensajes.

Resumiendo, tenemos:

* Programación declarativa
  + Paradigma lógico
  + Paradigma funcional
* Programación imperativa
  + Paradigma estructurado
  + Paradigma orientado a objetos
* Multiparadigma: lenguajes que soportan más de un paradigma:
  + JAVA, PYTHON, .NET, PHP…

# Lógica proposicional

Es la lógica que habla sobre proposiciones.

Cada proposición tiene un valor de verdad, verdadero o falso.

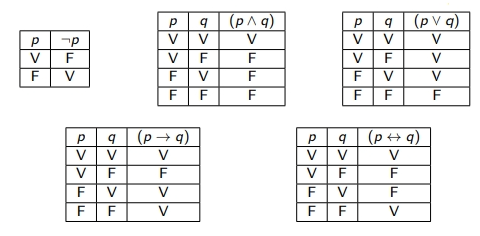
## Sintaxis

* Símbolos: True , False , ¬ , ∧ , ∨ , → , ↔ , ( , )
* Variables (infinitas): a, b, c, …
* Formulas
  + True y False son formulas.
  + Cualquier variable es una formula.
  + Si A es formula, ¬A también lo es
  + Si A­­1,A2 , …, An son fórmulas entonces (A1 ∧ A2 ∧ · · · ∧ An) es una fórmula y (A1 ∨ A2 ∨ · · · ∨ An) también.
  + Si A y B son formulas entonces (A→B) es formula y (A↔ B) también.

## Semántica clásica

* True vale V.
* False vale F.
* ¬ vale “no”, se llama negación.
* ∧, se interpreta como “y”, se llama conjunción.
* ∨, se interpreta como “o” (no exclusivo), se llama disyunción.
* → se interpreta como “si... entonces”, se llama implicación.
* ↔ se interpreta como “si y solo si”, se llama doble implicación o equivalencia.

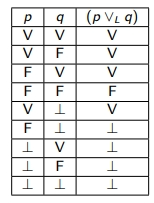
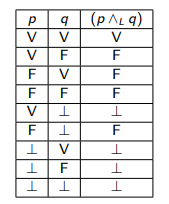
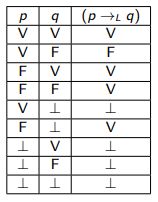
## Tablas de verdad



Se dice que toda expresión está bien definida si todas las proposiciones valen V o F.

## Semántica trivaluada

Surge de la necesidad de razonar problemas con indefiniciones.

## Tautologías, contradicciones y contingencias

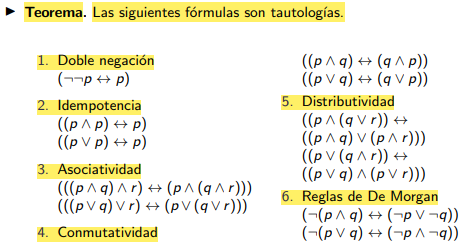
Una tautología es una fórmula que siempre da True.

Una contradicción es una fórmula que siempre da False.

Una contingencia es una fórmula que no es tautología ni contradicción (dependiendo de sus valores puede dar V o F).

## Equivalencia entre formulas

Sean A y B formulas, A y B son equivalentes (A≡B) si y solo si, (A↔B) es tautología.



## Relación de fuerza

Se dice que A es mas fuerte que B cuando (A→B) es tautología. También se dice que A fuerza a B o que B es más débil que A.

# Tipos de Datos

Son un conjunto de valores provistos por una serie de operaciones que involucran a esos valores.

## Tipo Enteros

Son el conjunto de los números enteros.

Poseen las siguientes operaciones:

* Suma: a + b
* Resta: a – b
* Valor absoluto: abs(a)
* Multiplicación: a\*b
* División entera: a div b.
* Resto: a mod b == Ra(b)
* Potencia: pot(a, b) == a^b
* División: a/b (da un valor real)

Admiten fórmulas que comparan términos tipo enteros como <, >, etc.

## Tipo Reales

Son el conjunto de los números reales. Admiten las mismas operaciones y fórmulas que los tipos enteros salvo por lo siguiente:

* No admite ni div ni mod
* Se agregan:
  + Logaritmo: logb(a)
  + Funciones trigonométricas

## Tipo Bool

Son el conjunto de valor de verdad (True o False).

Admiten las operaciones con lógica bi-valuada.

## Tipo Char (caracteres)

Son el conjunto de letras, dígitos y símbolos.

Se Admiten las siguientes funciones:

* Funcion ord: numera los caracteres con la siguiente propiedad, ord(‘a’) + 1 = ord(b).
* Funcion char: devuelve el carácter según la numeración de ord tal que char(ord(‘a’)) = ‘a’

## Tipo enumerados

Contienen una cantidad finita de elementos. Su estructura es la siguiente:

* Enum Nombre {Constantes}

Por convención las constantes se escriben en mayúsculas, por ejemplo:

* Enum dia {LUN, MAR, MIER, JUEV, VIE, SAB, DOM}

La funcion ord se traslada dando la posición del elemento comenzando en 0. Por ejemplo ord(LUN) = 0, ord(JUEV)=3

## Tipo Upla o Tupla

Son una estructura de datos que ordena elementos de distintos tipos, su estructura es la siguiente:

* Sea el nombre de la tupla y los distintos tipos de datos que la componen, notación en la especificación. un ejemplo de especificación.
* Para obtener el n – ésimo valor de la tupla la notación es la siguiente:
  + , por ejemplo, si , y

## Secuencias

Contienen varios elementos del mismo tipo.

Se notan donde es el tipo de dato.

Ejemplos:

* es una secuencia de tipo Entero.
* secuencia vacía.

A continuación, explico las funciones sobre secuencias.

* Longitud: , representa la longitud de una secuencia.
  + Notación: .
* Indexación: , es el elemento de la i-ésima posición de .
  + Notación:
* Pertenece: , da True si pertenece a .
  + Notación:
* Igualdad: , dos secuencias son iguales si y solo si tienen la misma cantidad elementos y en cada posición de ambas coinciden sus elementos.
  + Notación:
* Cabeza: , es el primer elemento de la secuencia .
  + Notación:
* Cola: , es la secuencia resultante de eliminar el primer elemento de .
  + Notación:
* Agregar cabeza: , agrega el elemento al principio de la lista .
  + Notación: .
* Concatenación: , es la secuencia de los elementos de seguido de los de .
  + Notación: .
* Subsecuencia: , son los elementos de entre las posiciones inclusive y exclusive.
  + Notación:
* Cambiar una posición: , cambia el valor de la lista en la posición por .
  + Notación:

# Programación Funcional (Haskell)

Un programa en lenguaje funcional es un conjunto de ecuaciones orientadas que definen un conjunto de funciones.

La ejecución de un programa en este caso corresponde a la evaluación de una expresión.

## Expresión

Una expresión es un valor que pertenece a un tipo de dato y no se puede usar como si fuese otro, por ello se dice que Haskell es fuertemente tipado. Las expresiones pueden ser:

* Atómicas: no se pueden reducir, por ejemplo, un valor.
* Compuestas: combinan expresiones atómicas con operaciones como por ejemplo 1+3, 1==2.

## Ecuaciones orientadas

A la hora de crear una función primero definimos su signatura (dominio y codominio), luego definimos la ecuación orientada que expresa la forma de declarar la función de manera que el lado izquierdo representa la expresión a definir y del derecho su definición, ejemplo:

## Transparencia referencial

Es la propiedad que garantiza que el valor de una expresión depende exclusivamente de sus subexpresiones. Por lo tanto, cada expresión representa el mismo valor en cualquier parte del programa.

Una definición alternativa dice: Hay transparencia referencial cuando al realizar una operación con los mismos valores siempre da el mismo resultado.

## Modo de ejecución en Haskell

La ejecución de una función se realiza por mecanismo de reducción y con un orden de evaluación normal o lazy (perezoso).

El mecanismo de reducción consiste en sustituir las subexpresiones del lado izquierdo de la ecuación orientada por sus resultados.

El orden de evaluación lazy consiste en que primero se evalúa la función y luego sus argumentos.

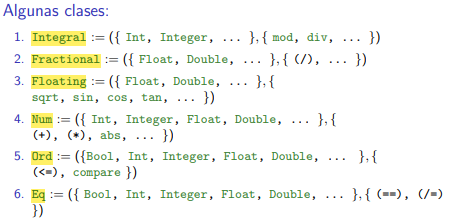
Las expresiones para las cuales Haskell no encuentra un resultado se dice que están indefinidas. Hay dos tipos de funciones en base a esto:

* Totales: está definida para todos los valores de entrada.
* Parciales: no está definida para ciertos tipos de parámetros.

## Haskell: tipos de datos, operaciones y funciones

En Haskell existen los tipos de datos nombrados con sus respectivas funciones con la diferencia de que también se pueden usar las clases de tipos.

Las clases de tipos son un conjunto de tipos de datos a los que se les puede aplicar un conjunto de funciones.



Gracias a esto podemos introducir el concepto de polimorfismo, esto es una función que puede aplicarse sobre distintos tipos de datos.

Pattern matching, es definir la función en varias líneas ya que cada definición es tomada como una conjunción.

Para funciones que requieran una sucesión de operaciones se utiliza la recursión, pensamos la función desde el último paso hasta su caso base.

[Funciones básicas de Haskell (us.es)](https://www.cs.us.es/~jalonso/cursos/i1m/doc/Funciones_basicas.html#Conj) link donde encontrar operaciones y funciones.

# Programación Imperativa (Python)

Un programa en Python es una colección de tipos y funciones.

A diferencia de Haskell aquí se utilizan variables y no argumentos, la iteración en vez de recursión, pero sí es fuertemente tipado (una variable no puede usarse como si fuera de otro tipo a no ser que se haga una conversión).

## Modo de ejecución

Su evaluación consiste en ejecutar una por una las instrucciones del bloque. El orden es de arriba hacia abajo.

## Estados

Se llama estado de un programa al valor de todas las variables en un punto de ejecución.

Desde este punto de vista la ejecución se ve como una sucesión de estados.

La asignación es la instrucción que transforma estados.

## Variables

Python es de tipado dinámico por lo que una variable puede tomar varios valores de distintos tipos.

Cada variable tiene asignado un espacio de memoria y esto permite explicar dos conceptos a la hora de evaluar funciones.

* Pasaje por Valor (o Copia): la información de la variable se almacena en una dirección de memoria diferente al recibirla en la función, y, por lo tanto, los cambios efectuados no afectan a la variable original.
* Pasaje por Referencia: se utiliza la misma dirección de memoria de la variable original por lo que sí es modificado por la función.

Conceptualmente existen tres tipos de parámetros:

* Entrada (in): su valor no se modifica al salir de la función.
* Salida (out): su valor es modificado al salir de la función, pero no importa ni debería ser leído.
* Entrada y salida (inout): su valor es modificado al salir de la función, y su valor si debería ser tenido en cuenta por la función.

Las variables tienen un distinto tipo de alcance, que es ámbito o espacio donde es reconocida. Las variables declaradas dentro de un bloque se llaman locales y se destruyen luego de ejecutar el bloque. Las que no se encuentran en ningún bloque son llamadas globales y cualquier bloque puede acceder a ellas y modificarla.

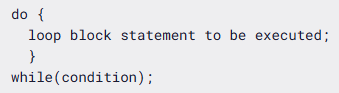
Además, hay dos tipos más:

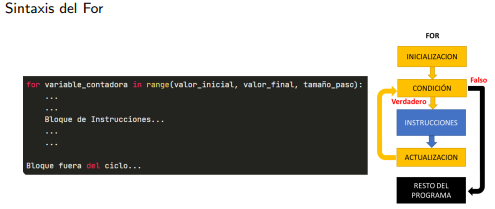
* No local o enclosed: aparecen declaradas en una función que residen dentro de otra función.
* Integrado o built-in: declaraciones propias de Python.

## Ciclos

Como ya había mencionado no utilizamos la recursión sino la iteración que consiste en la repetición, cada iteración es una repetición. Las estructuras de control encargadas de repetir un bloque de código mientras se cumpla una condición son:

* While: la estructura de código se repite mientras la condición se cumpla.
* Do while: como la estructura anterior pero primero se declara el bloque.



* For: el bloque se repite hasta que la variable contadora llegue a su valor final. Esta variable también puede recorrer listas.

La instrucción break permite romper la ejecución de un ciclo.

## Datos en Python

Se tienen los ya mencionados con sus respectivas funciones con la diferencia de que no existe el tipo Char, existe el tipo String (str). Estos son llamados primitivos.

Las listas, tuplas, pilas, etc. Son considerados como compuestos. Estos permiten introducir el siguiente concepto.

Tipos de datos abstractos (TDA), son modelos que definen tipos de datos, y son abstractos porque quien los utiliza no necesita conocer los detalles de su representación. Un ejemplo es el tipo lista.

Ahora procedo a explicar los tipos de dato array (arreglo), lista, pila y cola.

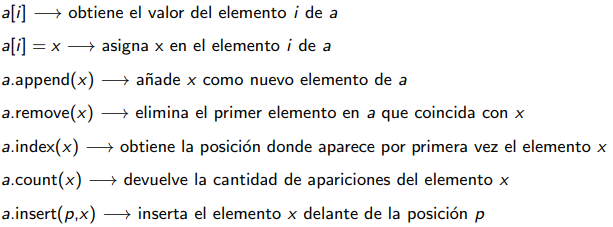
Los arreglos son secuencias de elementos del mismo tipo con longitud finita, contrario a las listas (son un tipo de array) que no son finitas y pueden contener elementos de distintos tipos. Arreglos y listas comparten funciones, pero se declaran de manera distintas.

La declaración de un array es la siguiente:



La declaración de una lista es

Las funciones que comparten son las siguientes:



Pila es una lista de elementos de tipo LIFO (Last In - First Out) donde el último elemento ingresado es el primero en salir.

* Declaración:
* Operaciones:
  + : apila/ingresa el elemento .
  + , toma un elemento de la pila (el último de la lista).
  + , devuelve si la pila está vacía.

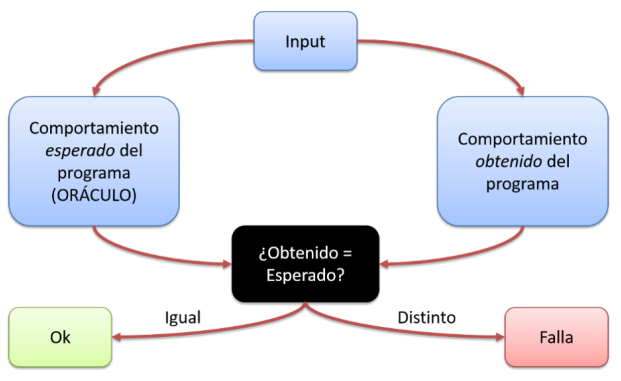
Cola es una lista de elementos de tipo FIFO (Firs In – First Out) donde el primer elemento en entrar es el primero en salir.

* Declaración:
* Operaciones:
  + : ingresa el elemento a la cola.
  + , toma un elemento de la cola (el primero de la lista).
  + , devuelve si la cola está vacía.

# Testing

Es el proceso de ejecutar un producto para verificar que satisface los requerimientos (especificación) e identificar las diferencias entre el comportamiento real y el esperado. Su objetivo es identificar defectos en el software.

Una forma de entender sencillamente el testing es la comparación entre el comportamiento esperado y el obtenido del programa. Si son iguales entonces el test pasó correctamente, pero en caso contrario se lo considera una falla.



## Niveles y conceptos del testing

Según la cantidad de código que abarca el testing se lo considero como:

* Test de sistema, comprende todo el sistema y constituye el test de aceptación.
* Test de integración, orientado al correcto funcionamiento de partes del sistema de forma aislada, ya que, si es así, implica que también lo es en conjunto.
* Test de unidad, cuando se realiza a una pequeña parte del código.

Al programa a testear lo llamaremos “programa bajo test”.

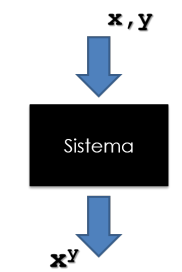
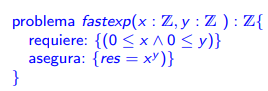
La asignación concreta a parámetros de entrada es llamada “test input” (o dato de prueba).

El programa que ejecuta el programa bajo test y verifica si cumple la condición de aceptación es llamado “test case” (o caso de test/prueba). Y “test suite” es el conjunto de los casos de prueba.

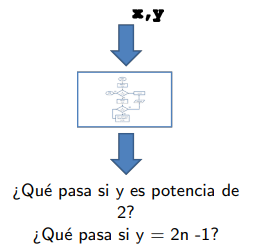
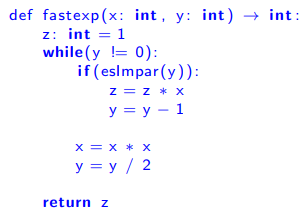
A la hora de plantear el testing intentaremos abarcar el mayor dominio de resultados y maximizar la probabilidad de encontrar errores, pero con la menor envergadura posible.

## Criterios para seleccionar datos de prueba

* De caja negra o funcionales, los datos de test se derivan a partir de la descripción del programa sin conocer su implantación.



* De caja blanca o estructurales, los datos de tests se derivan a partir de la estructura interna del programa.



Por lo general el criterio de caja blanca se utiliza como complemento del criterio de caja negra debido a lo difícil que es obtener el resultado esperado sin una especificación.

## Método de partición de categorías

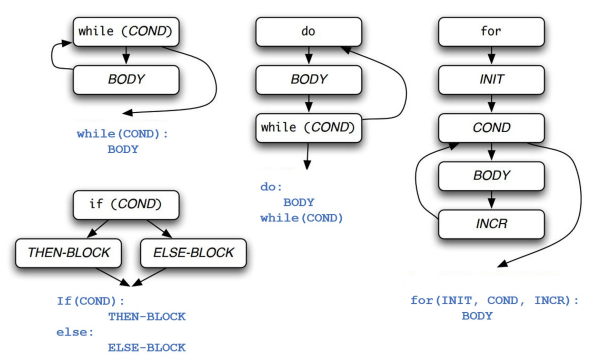
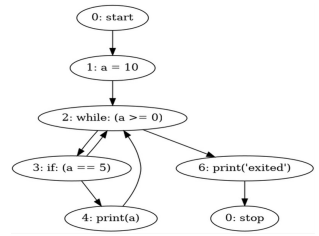
Es una técnica que permite generar casos de prueba de manera metódica.

Consiste en los siguientes pasos:

1. Descomponer la solución y listar todos los problemas (operaciones, funciones, etc., serán llamadas unidades funcionales) que queremos testear.
2. Elegir una unidad funcional, de ser posible por las que son utilizadas por otras.
3. Identificar factores, estos son parámetros o relaciones entre ellos que condicionen el comportamiento.
4. Determinar las características relevantes (categorías) de cada factor.
5. Determinar elecciones, estas son conjuntos de datos con comportamiento similar.
6. Clasificar las elecciones en
   1. Errores, resultados de ejecución incorrectos o indeterminados.
   2. Único
   3. Restricción, condición asociada a la elección.
7. Armar los casos de tests con las combinaciones entre las elecciones y las categorías, detallando el resultado esperado.
8. Volver al paso 2 y repetir.

## Control Flow Graph

Es la representación gráfica del programa se usa para definir criterios de adecuación para tests suites.



## Criterios de adecuación

Es un predicado que toma un valor de verdad para una tupla <programa, test suite>. Se basa en la idea de que todas las sentencias deben ser ejecutadas.

* Cada sentencia (nodo) debe ser ejecutado al menos una vez por algún test case.
* Todo arco (flechas) debe ser ejecutado al menos una vez por algún test case.
* Cada decisión (arco True o arco False) debe ser ejecutado al menos una vez por algún test case.
* Cada nodo de decisión debe ser evaluado en True o False al menos una vez.
  + , donde las condiciones básicas son formulas atómicas que componen una decisión.
* Todos los caminos deben ser transitados por al menos un test case.

En síntesis:

* Sentencias: cubrir todos los nodos.
* Arcos: cubrir todos los arcos.
* Decisiones: evaluar la guarda en True y False por cada if, while, for, etc.
* Condiciones básicas: cada componente de una guarda fue evaluado a True y False.
* Caminos: cubrir todos los caminos.