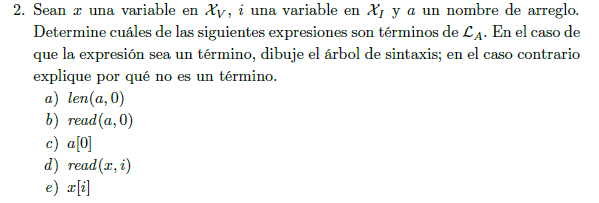
Camilo Andrés Quintero Rodríguez

1. Sección 6.6: 2,9,11,12,15,16,17,18,19
2. Sección 9.1: 1, 2
3. Sección 6.6: **2**,**9**,**11**,**12**,**15**,**16**,**17**,**18,19**







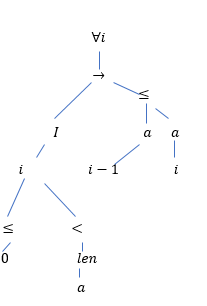


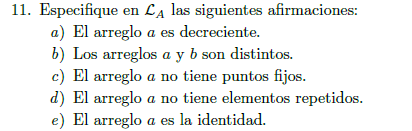
Texto

Descripción generada automáticamente

Gráfico, Gráfico radial

Descripción generada automáticamente

* 1. 



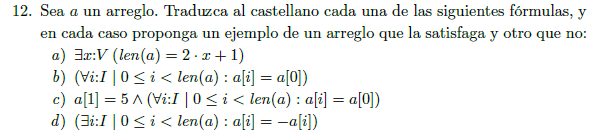


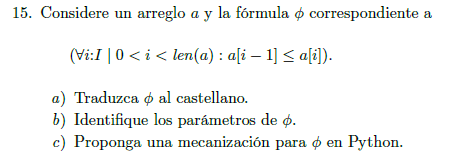




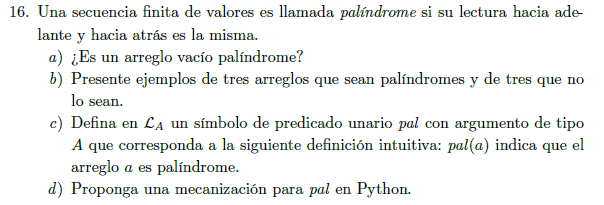








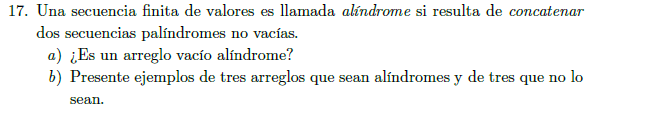






:

:



Texto

Descripción generada automáticamente

1. No, un arreglo vacío no se considera un alíndrome ya que no puede ser resultado de la concatenación de dos secuencias palíndromes no vacías. Para que una secuencia sea alíndrome, debe estar compuesta por al menos dos secuencias palíndromes no vacías concatenadas.
2. - [1, 2, 3, 2, 1, 1, 2, 2, 1] En este caso, la secuencia palíndrome "1, 2, 3, 2, 1" está concatenada con la secuencia palíndrome "1, 2, 2, 1" para formar el alíndrome.

* [5, 2, 7, 6, 2, 7, 5, 4, 2, 1,2,4] En este caso, la secuencia palíndrome "5, 2, 7, 6, 2, 7, 5” está concatenada con la secuencia palíndrome "4, 2, 1,2,4" para formar el alíndrome.
* [9, 8, 7, 8,9, 3, 5, 7, 5,3] En este caso, la secuencia palíndrome "9, 8, 7, 8,9" está concatenada con la secuencia palíndrome "3, 5, 7, 5,3" para formar el alíndrome.

Ejemplos de tres arreglos que no sean alíndromes:

* [1, 2, 3, 4, 5, 6, 7, 8, 9] Esta secuencia no es un alíndrome porque no se puede descomponer en dos secuencias palíndromes no vacías.
* [2, 3, 4, 2, 1, 5, 6, 4, 2] Esta secuencia no es un alíndrome porque no se puede descomponer en dos secuencias palíndromes no vacías.
* [1, 2, 3, 2, 4, 5, 6, 7, 8, 9] Esta secuencia no es un alíndrome porque aunque contiene una secuencia palíndrome "1, 2, 3, 2, 1", no puede ser descompuesta en dos secuencias palíndromes no vacías.

Texto

Descripción generada automáticamente

**Aserción:** Una aserción es una expresión booleana que se utiliza para verificar ciertas propiedades de un programa en un punto específico de su ejecución. La aserción se coloca antes o después de un fragmento de código y se utiliza para comprobar que se cumple una determinada condición en ese punto del programa. Si la aserción no se cumple, el programa puede generar un error o comportarse de manera inesperada.

**Ejemplo:** Supongamos que estamos desarrollando una función para calcular la raíz cuadrada de un número. Podemos agregar una aserción antes de la operación de la raíz cuadrada para verificar que el número de entrada sea mayor o igual a cero. Si la aserción no se cumple, podemos generar un error o lanzar una excepción para evitar que el programa continúe ejecutándose.

**Precondición:** Una precondición es una condición que se debe cumplir antes de que se ejecute una función o método. La precondición se utiliza para especificar las condiciones necesarias para que la función o método se ejecute correctamente. Si la precondición no se cumple, la función puede generar un error o lanzar una excepción.

**Ejemplo:** Supongamos que estamos desarrollando una función para dividir dos números. Podemos agregar una precondición para verificar que el divisor sea diferente de cero. Si la precondición no se cumple, podemos generar un error o lanzar una excepción para evitar la división por cero.

**Poscondición:** Una poscondición es una condición que se debe cumplir después de que se ejecute una función o método. La poscondición se utiliza para especificar las condiciones que se deben cumplir después de que la función o método se ejecute correctamente. Si la poscondición no se cumple, la función puede generar un error o comportarse de manera inesperada.

**Ejemplo:** Supongamos que estamos desarrollando una función para ordenar una lista de números. Podemos agregar una poscondición para verificar que la lista esté ordenada después de que se ejecute la función. Si la poscondición no se cumple, podemos generar un error o lanzar una excepción para indicar que la lista no está ordenada correctamente.

**Invariante:** Un invariante es una propiedad que se mantiene constante durante la ejecución de un programa. El invariante se utiliza para verificar que ciertas propiedades se mantengan verdaderas en todo momento. Los invariantes son especialmente útiles para verificar la corrección de estructuras de datos y algoritmos.

**Ejemplo:** Supongamos que estamos desarrollando una clase para una pila de números. Podemos agregar un invariante para verificar que el tamaño de la pila sea mayor o igual a cero y menor o igual al tamaño máximo de la pila. Si el invariante no se cumple, podemos generar un error o lanzar una excepción para indicar que la pila no está en un estado válido.

1. Sección 9.1: 1, 2

