```
### EJERCICIO 1 ###

def insert(T, elem):
  if T.root == None:
    T.root = TrieNode()
    T.root.children = []
  insertR(T.root, elem, 0)

def insertR(node, elem, index):
  # node.children es una lista de Python, que contiene TNodes.
  # La inserción comienza buscando desde la raíz, cuyo campo key está vacío,
  # por lo cual buscamos la letra elem[index] en la lista node.children del nodo raíz.
  foundNode = None
  for TNode in node.children:
    if TNode.key == elem[index]:
      foundNode = TNode
  # Si no existe, creamos un TNode nuevo que tenga esa letra como key,
  # y la insertamos en la lista node.children
  # Si esa es la última letra de la palabra, se marca como tal y finaliza la inserción.
  # Si no, se vuelve a llamar a la función recursivamente para insertar la próxima letra,
  # siguiendo desde el último TNode insertado.
  if foundNode == None:
    TNode = TrieNode()
    TNode.key = elem[index]
    TNode.parent = node
    TNode.children = []
    node.children.append(TNode)
    if len(elem)-1 == index:
      TNode.isEndOfWord = True
      return
    else:
      insertR(TNode, elem, index+1)
  # Si la letra elem[index] ya existe y es también la última letra a insertar,
  # se marca como tal y finaliza la inserción.
  elif len(elem)-1 == index:
    foundNode.isEndOfWord = True
    return
  # Por último, si la letra ya existe y todavía no se ha terminado la inserción, se llama
  # nuevamente a la función en forma recursiva continuando desde esa última letra (TNode) encontrada.
  else:
    insertR(foundNode, elem, index+1)
def search(T, elem):
  if T.root == None:
    return False
  index = 0
  return searchR(T.root, elem, index)

def searchR(node, elem, index):
  foundNode = None
  for TNode in node.children:
    if TNode.key == elem[index]:
      foundNode = TNode
  if foundNode == None:
    return False
  elif len(elem)-1 == index and foundNode.isEndOfWord:
    return foundNode
  else:
    return searchR(foundNode, elem, index+1)

##################
```

```
### EJERCICIO 2 ###

'''
Si en lugar de usar listas para almacenar los TNodes, usamos arreglos, podemos entonces intentar
acceder directamente a la letra que estamos buscando, sin necesidad de hacer un recorrido. Esto se
logra también realizando un mapeo de las letras del alfabeto considerado hacia un valor numérico,
que será luego la posición que ocupe siempre esa letra en un arreglo determinado, para los distintos
arreglos que vayan a usarse en cada nivel del Trie.

'''
###################
```

```
### EJERCICIO 3 ###

def delete(T, elem):
  foundNode = search(T, elem)
  if foundNode != None:
    # Caso en que la palabra a elminar está contenida dentro de otra
    if len(foundNode.children) != 0:
      foundNode.isEndOfWord = False
      return True
    else:
      # Caso en que la palabra contiene otras palabras en su interior,
      # o caso en que la palabra no comparte alguna parte con otra palabra,
      # y se elimina entonces completa hasta la raíz.
      foundNode.isEndOfWord = False
      while foundNode.isEndOfWord == False and foundNode.parent != None:
        foundNodeCopy = foundNode
        foundNode.parent.children.remove(foundNode)
        foundNode = foundNodeCopy.parent
      return True
  else:
    # Caso en que la palabra no existe en el Trie ingresado
    return False

###################
```

```
### EJERCICIO 4 ###

def printTrie(T):
  if T.root != None:
    listOfWords = []
    printTrieR(T.root.children, [], 0, listOfWords)
    print(listOfWords)
  else:
    print("El Trie está vacío")
    return

def printTrieR(TNodeChildren, prefix, n, containerList):
  for TNode in TNodeChildren:
    pre = []
    pre.append(TNode.key)
    TNodeC = TNode
    while len(TNodeC.children) == 1:
      TNodeC = TNodeC.children[0]
      pre.append(TNodeC.key)
    if len(TNodeC.children) > 1:
      printTrieR(TNodeC.children, prefix + pre, n, containerList)
    else:
      if n == 0 or len(prefix + pre) == n:
        containerList.append("".join(prefix + pre))
        #print("".join(prefix + pre))
      else:
        print("Palabra tiene prefijo, pero no longitud solicitada")
```

```python
def startsWith(T, p, n):
  i = 0
  node = T.root
  while i < len(p):
    j = i
    for TNode in node.children:
      if TNode.key == p[i]:
        node = TNode
        i += 1
    if i == j:
      print("Prefijo no encontrado en el Trie")
      return
  printTrieR(node.children, list(p), n, [])

####################
```

```python
### EJERCICIO 5 ###

def areEqual(T1, T2):
  T1List = []
  printTrieR(T1.root.children, [], 0, T1List)
  for word in T1List:
    if search(T2, word) == False:
      return False
  return True

####################
```

```python
### EJERCICIO 6 ###

def hasInvertedStrings(T):
  TList = []
  printTrieR(T.root.children, [], 0, TList)
  for word in TList:
    invertedWord = word[::-1]
    if search(T, invertedWord) != False:
      return True
  return False

####################
```

```python
### EJERCICIO 7 ###

def autoComplete(T, s):
  i = 0
  node = T.root
  while i < len(s):
    j = i
    for TNode in node.children:
      if TNode.key == s[i]:
        node = TNode
        i += 1
    if i == j:
      print("Prefijo no encontrado en el Trie")
      return
  autoCompLetters = ""
  while len(node.children) == 1:
    autoCompLetters += node.children[0].key
    node = node.children[0]
  print(autoCompLetters)

####################
```