Tu calificación: 100 %

Tu calificación más reciente: **100 %** • Tu calificación más alta: **100 %** • Para aprobar necesitas al menos un 80 %. Guardamos tu puntaje más alto.

Próximo artículo

Estás viendo una versión traducida automáticamente de esta evaluación
Puedes volver a ver el contenido en su idioma original si lo prefieres. No perderás el progreso que hayas conseguido si cambias el idioma. Mostrar la versión en Inglés

1. Cuando un usuario informa de que una "aplicación no funciona", ¿cuál es una pregunta de seguimiento adecuada para recabar más información sobre el problema? 1 / 1 punto

- ¿Está enchufado el servidor?
- ¿Por qué necesita la aplicación?
- ¿Tiene un número de ticket de asistencia?
- (•) ¿Qué debería ocurrir al abrir la aplicación?

Correcto
¡Impresionante! Preguntar al usuario cuál debería ser el resultado esperado le ayudará a reunir más información para comprender y aislar el problema.

2. La función find_item utiliza la búsqueda binaria para localizar recursivamente un elemento en la lista, devolviendo True si se encuentra, False en caso contrario. Algo falta en esta función. ¿Puede detectar qué es y solucionarlo? Añada líneas de depuración donde proceda, para ayudar a localizar el problema. 1 / 1 punto

```
def find_item(list, item):
  #Returns True if the item is in the list, False if not.
  if len(list) == 0:
    return False ## OK

  #Is the item in the center of the list?
  middle = len(list)//2 ## OK
  if list[middle] == item:
    return True ## OK

  #Is the item in the first half of the list?
  ## if item < list[middle]: ## Incorrect
  if item in list[:middle]:
  #Call the function with the first half of the list
    return find_item(list[:middle], item) ## OK
  else:
    #Call the function with the second half of the list
    return find_item(list[middle+1:], item) ## OK

  return False

#Do not edit below this line - This code helps check your work!
list_of_names = ["Parker", "Drew", "Cameron", "Logan", "Alex", "Chris", "Terry", "Jamie", "Jordan", "Taylor"]

print(find_item(list_of_names, "Alex")) ## True
print(find_item(list_of_names, "Andrew")) ## False
print(find_item(list_of_names, "Drew")) ## True
print(find_item(list_of_names, "Jared")) ## False
```

Ejecutar
Restablecer

Correcto
¡Bien hecho, usted! Has ordenado el código y has encontrado la pieza que faltaba, ¡así se hace!

3. La función `binary_search` devuelve la posición de la llave en la lista si se encuentra, o -1 si no se encuentra. Usted quiere asegurarse de que funciona correctamente, por lo que necesita colocar líneas de depuración que le permitan saber cada vez que la lista se corta por la mitad, si está a la izquierda o a la derecha. No querrá imprimir nada cuando se localice la tecla. 1 / 1 punto

Por ejemplo, el comando `binary_search([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 3)` realiza estos pasos:

1. Determina que la clave, 3, está en la mitad izquierda de la lista e imprime "Comprobando el lado izquierdo".
2. A continuación, determina que 3 se encuentra en la mitad derecha de la nueva lista e imprime "Comprobando el lado derecho".
3. Por último, devuelve el valor 2, que es la posición de la llave en la lista.

Añada comandos al código para imprimir "Comprobando el lado izquierdo" o "Comprobando el lado derecho" en los lugares adecuados.

```
def binary_search(list, key):
    #Returns the position of key in the list if found, -1 otherwise.

    #List must be sorted:
    list.sort()
    left = 0
    right = len(list) - 1

    while left <= right:
        middle = (left + right) // 2

        if list[middle] == key:
            return middle
        if list[middle] > key:
            print("Checking the left side")
            right = middle - 1
        if list[middle] < key:
            print("Checking the right side")
            left = middle + 1
    return -1

print(binary_search([10, 2, 9, 6, 7, 1, 5, 3, 4, 8], 1))
"""Should print 2 debug lines and the return value:
Checking the left side
Checking the left side
0
"""

print(binary_search([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 5))
"""Should print no debug lines, as it's located immediately:
4
"""

print(binary_search([10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 7))
"""Should print 3 debug lines and the return value:
Checking the right side
Checking the left side
Checking the right side
6
"""
```

Ejecutar
Restablecer

Correcto
¡Buen trabajo! Vea lo útil que es la depuración para mostrar cómo el proceso está funcionando.

4. Cuando intentamos encontrar un error en un archivo de registro o salida a pantalla, ¿qué comando podemos utilizar para revisar, digamos, las 10 primeras líneas? 1 / 1 punto

- wc
- cola
- (•) head
- bisecar

Correcto
¡Impresionante! El comando head imprimirá las primeras líneas de un archivo, 10 líneas por defecto.

5. La función best_search compara las funciones linear_search y binary_search, para localizar una clave en la lista, y devuelve cuántos pasos necesitó cada método, y cuál es el mejor para esa situación. No es necesario ordenar la lista, ya que la función binary_search la ordena antes de proceder (y utiliza un paso para hacerlo). En este caso, las funciones linear_search y binary_search devuelven ambas el número de pasos que se tardó en localizar la clave o en determinar que no está en la lista. Si el número de pasos es el mismo para ambos métodos (incluido el paso extra para ordenar en binary_search), entonces el resultado es un empate. Rellene los espacios en blanco para que esto funcione. 1 / 1 punto

```
def linear_search(list, key):
    #Returns the number of steps to determine if key is in the list

    #Initialize the counter of steps
    steps=0
    for i, item in enumerate(list):
        steps += 1
        if item == key:
            break
    return steps

def binary_search(list, key):
    #Returns the number of steps to determine if key is in the list

    #List must be sorted:
    list.sort()

    #The Sort was 1 step, so initialize the counter of steps to 1
    steps=1

    left = 0
    right = len(list) - 1
    while left <= right:
        steps += 1
        middle = (left + right) // 2

        if list[middle] == key:
            break
        if list[middle] > key:
            right = middle - 1
        if list[middle] < key:
            left = middle + 1
    return steps

def best_search(list, key):
    steps_linear = linear_search(list, key)
    steps_binary = binary_search(list, key)
    results = "Linear: " + str(steps_linear) + " steps, "
    results += "Binary: " + str(steps_binary) + " steps. "
    if (steps_linear < steps_binary):
```

Ejecutar
Restablecer

Correcto
¡Bien hecho! Te estás volviendo bueno trabajando con los diferentes ¡métodos de búsqueda!