

Introducción a la programación

Práctica 7: Funciones sobre listas (tipos complejos)

1.3 Suma Total

```
problema sumaTotal (in s:seq< $\mathbb{Z}$ >) :  $\mathbb{Z}$  {  
  requiere: { True }  
  asegura: { res es la suma de todos los elementos de s }  
}
```

Nota: no utilizar la función `sum()` nativa

1.3 Suma Total

```
problema sumaTotal (in s:seq< $\mathbb{Z}$ >) :  $\mathbb{Z}$  {  
  requiere: { True }  
  asegura: { res es la suma de todos los elementos de s }  
}
```

Nota: no utilizar la función `sum()` nativa

Pista: En este ejercicio estaremos usando una variable que acumula el resultado y luego lo devuelve.

¿Qué es Debugging y para qué sirve?

1. Podemos ir paso a paso analizando los valores de las variables durante la ejecución
2. Sirve para poder realizar seguimiento del código
3. Podemos avanzar paso a paso o saltar al siguiente breakpoint
4. Podemos terminar la ejecución por la mitad o bien continuar hasta el final
5. Con VSCode podemos agregar breakpoints durante el momento de debugging, o eliminarlos
6. Se pueden agregar breakpoints con condiciones lógicas, por ejemplo: `valor_actual == 7`

Agregar un breakpoint (punto de detención) en el código

```
1
2  def suma_total(s:[int])-> int:
3      total:int = 0
4      indice_actual:int = 0
5      longitud:int = len(s)
6
7      while (indice_actual < longitud):
8          valor_actual:int = s[indice_actual]
9          total = total + valor_actual
10         indice_actual += 1
11
12     return total
13
```




Figura: Agregamos un breakpoint en la línea 7 del código

Ejecutar con Debug

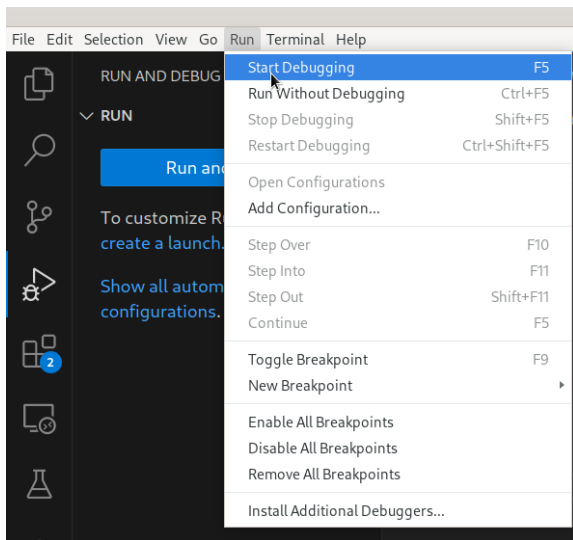
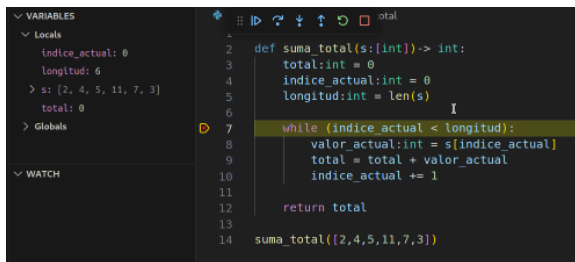


Figura: Ejecutamos el código con la opción Debug

Usamos los controles de la IDE para desplazarnos



The image shows a Python IDE interface. On the left, there is a 'VARIABLES' panel with a 'Locals' section containing the following variables and values:

- `indice_actual: 0`
- `longitud: 6`
- `s: [2, 4, 5, 11, 7, 3]`
- `total: 0`

Below this is a 'WATCH' section which is currently empty. The main editor on the right displays a Python function `suma_total` and its call. The function is as follows:

```
1 def suma_total(s:[int])-> int:
2     total:int = 0
3     indice_actual:int = 0
4     longitud:int = len(s)
5     I
6
7     while (indice_actual < longitud):
8         valor_actual:int = s[indice_actual]
9         total = total + valor_actual
10        indice_actual += 1
11
12    return total
13
14 suma_total([2,4,5,11,7,3])
```

The line number 7, corresponding to the start of the `while` loop, is highlighted in yellow. Above the code editor, a toolbar contains standard debugger icons: a play button, a step-through button, a step-over button, a step-back button, a step-forward button, a refresh button, and a close button. A small 'total' label is visible next to the close button.

Figura: Podemos ver las variables con sus valores al momento del break y usar los controles para movernos

Usamos los controles de la IDE para desplazarnos



F5 Continuar hasta el siguiente breakpoint (o si no hay más hasta el final)

F10 Siguiente paso saltando ingresar a la función que se esté evaluando en esta línea

F11 Siguiente paso ingresando a la función que se esté evaluando en esa línea

Shift+F11 Salir de la evaluación de la función a la que se ingresó

Ctrl + Shift + F5 Reiniciar el debug desde el principio

Shift + F5 Detener el debugging

Ejercicio 1.1: Pertenece

```
problema pertenece ( in s:seq< $\mathbb{Z}$ >, in e:  $\mathbb{Z}$ ) : Bool {  
  requiere: { True }  
  asegura: { (res = true)  $\leftrightarrow$  e  $\in$  s }  
}
```

Implementar al menos de 3 formas distintas éste problema.

¿Si la especificáramos e implementáramos con tipos genéricos, se podría usar esta misma función para buscar un caracter dentro de un string?

Ejercicio 1

7. Analizar la fortaleza de una contraseña. El parámetro de entrada será un *string* con la contraseña, y la salida otro *string* con tres posibles valores: VERDE, AMARILLA y ROJA.

Ejercicio 1.7: fortaleza de una contraseña

► **VERDE** si:

- a) longitud es mayor o igual a 8 caracteres
- b) tiene al menos una minúscula
- c) tiene al menos una mayúscula
- d) tiene al menos un número (0...9)

Ejercicio 1.7: fortaleza de una contraseña

► **VERDE** si:

- a) longitud es mayor o igual a 8 caracteres
- b) tiene al menos una minúscula
- c) tiene al menos una mayúscula
- d) tiene al menos un número (0...9)

► **ROJA** si:

- a) longitud es menor a 5 caracteres

Ejercicio 1.7: fortaleza de una contraseña

► **VERDE** si:

- a) longitud es mayor o igual a 8 caracteres
- b) tiene al menos una minúscula
- c) tiene al menos una mayúscula
- d) tiene al menos un número (0...9)

► **ROJA** si:

- a) longitud es menor a 5 caracteres

► **AMARILLA** en caso contrario

Ejercicio 1.7: el código ASCII

- ▶ Asocia un número a cada caracter

Ejercicio 1.7: el código ASCII

- ▶ Asocia un número a cada caracter
- ▶ Determina un orden entre los caracteres

Ejercicio 1.7: el código ASCII

- ▶ Asocia un número a cada caracter
- ▶ Determina un orden entre los caracteres

47	/	64	@	96	'
48	0	65	A	97	a
49	1	66	B	98	b
50	2	67	C	99	c
51	3	68	D	100	d
52	4	69	E	101	e
...
57	9	90	Z	122	z

Cuadro: parte de la codificación ASCII

Nota: se puede ver la tabla completa en elcodigoascii.com.ar

Ejercicio 1.7: el código ASCII

- ▶ Python permite realizar comparaciones entre caracteres basadas en el orden dado por el código ASCII

Ejercicio 1.7: el código ASCII

- ▶ Python permite realizar comparaciones entre caracteres basadas en el orden dado por el código ASCII
- ▶ **Probar en Python imprimir las siguientes expresiones bool:**
 - ▶ `print('a' < 'b')`
 - ▶ `print('A' > 'Z')`
 - ▶ `print('0' < '5')`
 - ▶ `print('@' < 'E')`

Ejercicio 1.7: Cómo establecer condiciones sobre los caracteres

- ▶ ¿Cómo puedo saber si un **caracter** es un número?

Ejercicio 1.7: Cómo establecer condiciones sobre los caracteres

- ▶ ¿Cómo puedo saber si un **caracter** es un número?
- ▶ `es_un_numero = (caracter ≤ '9') and (caracter ≥ '0')`
- ▶ Podemos usar `es_un_numero` como una **condicion** para analizar cuando vamos recorriendo la contraseña.

Ejercicio 1.7: Cómo establecer condiciones sobre los caracteres

- ▶ **¿Cómo puedo saber si un `caracter` es un número?**
- ▶ `es_un_numero = (caracter ≤ '9') and (caracter ≥ '0')`
- ▶ Podemos usar `es_un_numero` como una **condicion** para analizar cuando vamos recorriendo la contraseña.
- ▶ El mismo análisis haremos para cualquier otra condición. A continuación veremos opciones de algoritmos para cualquier condición.

Ejercicio 1.7: dos formas de verificar “tiene al menos un”

```
i: int = 0
vale_condicion: bool = False
while i < len(contrasena):
    if condicion:
        vale_condicion: bool = True
    i += 1
```

Ejercicio 1.7: dos formas de verificar “tiene al menos un”

```
i: int = 0
vale_condicion: bool = False
while i < len(contrasena) and not condicion:
    i += 1
vale_condicion: bool = i < len(contrasena)
```

Ejercicio 1.7: comparativa “tiene al menos un número”

contrasena: `str` = “cl4v3”

i	contrasena[i]	Algoritmo 1	Algoritmo 2
0	c	vale_condicion = False	vale_condicion = False

Ejercicio 1.7: comparativa “tiene al menos un número”

contrasena: `str` = “cl4v3”

i	contrasena[i]	Algoritmo 1	Algoritmo 2
0	c	vale_condicion = False	vale_condicion = False
1	l	vale_condicion = False	vale_condicion = False

Ejercicio 1.7: comparativa “tiene al menos un número”

contrasena: `str` = “cl4v3”

i	contrasena[i]	Algoritmo 1	Algoritmo 2
0	c	vale_condicion = False	vale_condicion = False
1	l	vale_condicion = False	vale_condicion = False
2	4	vale_condicion = True	-

Ejercicio 1.7: comparativa “tiene al menos un número”

contrasena: `str` = “cl4v3”

i	contrasena[i]	Algoritmo 1	Algoritmo 2
0	c	vale_condicion = False	vale_condicion = False
1	l	vale_condicion = False	vale_condicion = False
2	4	vale_condicion = True	-
3	v	vale_condicion = True	-

Ejercicio 1.7: comparativa “tiene al menos un número”

contrasena: `str` = “cl4v3”

i	contrasena[i]	Algoritmo 1	Algoritmo 2
0	c	vale_condicion = False	vale_condicion = False
1	l	vale_condicion = False	vale_condicion = False
2	4	vale_condicion = True	-
3	v	vale_condicion = True	-
4	3	vale_condicion = True	-

Ejercicio 1.7: comparativa “tiene al menos un número”

contrasena: `str` = “cl4v3”

i	contrasena[i]	Algoritmo 1	Algoritmo 2
0	c	vale_condicion = False	vale_condicion = False
1	l	vale_condicion = False	vale_condicion = False
2	4	vale_condicion = True	-
3	v	vale_condicion = True	-
4	3	vale_condicion = True	-
5	-	vale_condicion = True	vale_condicion = $i < \text{len}(\text{contrasena}) =$ True

Cuadro: seguimiento paso a paso de ambos algoritmos

Ejercicio 2.1

Dada una lista de números, en las posiciones pares borra el valor original y coloca un cero. Esta función modifica el parámetro ingresado, es decir, la lista es un parámetro de tipo inout.

5.1 Pertenece a Cada Uno

```
problema perteneceACadaUno (in s:seq<seq< $\mathbb{Z}$ >>, in e: $\mathbb{Z}$ , out  
res: seq<Bool>) {  
  requiere: { True }  
  asegura: { |res| = |s| }  
  asegura: { la posición i de res tiene el valor de  
             pertenece(s[i],e) }  
}
```

Nota: Reutilizar la función `pertenece()` implementada previamente para listas