

# 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

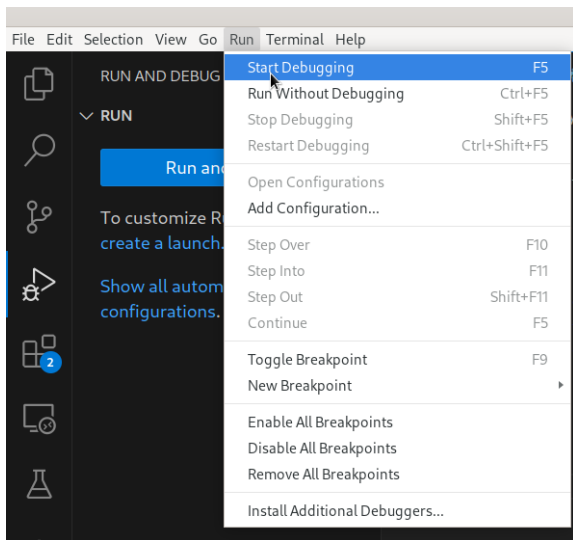
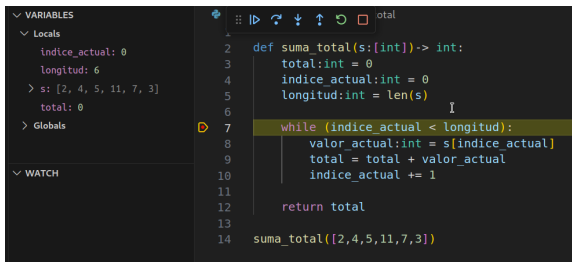


Figura: Ejecutamos el código con la opción Debug

# 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 area on the right displays a Python function `suma_total` with the following code:

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

The code is being executed, and the debugger is currently paused at line 7, which is highlighted in green. The cursor is positioned at the end of the line `while (indice_actual < longitud):`.

**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  | <b>0</b> | 65  | <b>A</b> | 97  | <b>a</b> |
| 49  | <b>1</b> | 66  | <b>B</b> | 98  | <b>b</b> |
| 50  | <b>2</b> | 67  | <b>C</b> | 99  | <b>c</b> |
| 51  | <b>3</b> | 68  | <b>D</b> | 100 | <b>d</b> |
| 52  | <b>4</b> | 69  | <b>E</b> | 101 | <b>e</b> |
| ... | ...      | ... | ...      | ... | ...      |
| 57  | <b>9</b> | 90  | <b>Z</b> | 122 | <b>z</b> |

Cuadro: parte de la codificación ASCII

**Nota:** se puede ver la tabla completa en [elcodigoascii.com.ar](http://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  $\leq$  '9') and (caracter  $\geq$  '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  $\leq$  '9') and (caracter  $\geq$  '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”

| <b>i</b> | <b>contrasena[i]</b> | <b>Algoritmo 1</b>     | <b>Algoritmo 2</b>     |
|----------|----------------------|------------------------|------------------------|
| 0        | c                    | vale_condicion = False | vale_condicion = False |
|          |                      |                        |                        |



## Ejercicio 1.7: comparativa “tiene al menos un número”

**contrasena:** `str` = “cl4v3”

| <b>i</b> | <b>contrasena[i]</b> | <b>Algoritmo 1</b>     | <b>Algoritmo 2</b>     |
|----------|----------------------|------------------------|------------------------|
| 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 =<br>$i < \text{len}(\text{contrasena}) =$<br>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