2.4.-Depurar programas

2.4. Depurar (Debug) un programa

1. ¿Qué es hacer debug o depurar?

En ocasiones, cuando realizamos un programa, el resultado obtenido no es el esperado, es decir, hemos cometido algún error de cálculo, el cual hace que el programa funcione pero no lo haga correctamente. En estas ocasiones es muy importante depurar el programa, encontrar el origen del error.

En la depuración lo que hacemos es ejecutar el programa **paso a paso**, así podemos ver las instrucciones que se están ejecutando, además de poder ver los valores que van tomando las variables. Esto nos sirve de ayuda para ver las variables, los valores que van tomando, las instrucciones que se están ejecutando y así poder ver dónde hemos cometido un error.

Para depurar, podemos usar las herramientas que viene en los entornos de desarrollo integrados/editores, como Visual Code, o usar herramientas que vienen con el interprete de Python, como Pdb

A continuación, veremos un ejemplo del uso de Visual Code

2. Ejemplo de uso de Visual Code

En Visual Code, el editor de texto que vamos a utilizar, para poder depurar un programa en Python se necesita instalar el plugin correspondiente. Para ello, vamos a la opción **Extensions** y buscamos **Python**. Nos aparecerá una lista de plugins, y seleccionamos el que se llama **Python**.

Una vez instalado este plugin, vamos a la opción **Debug**, que se encuentra en la columna izquierda de Visual Code, así podemos empezar a depurar nuestro programa.

Si el programa a depurar es muy grande, tal vez no nos interese realizar la traza desde el principio, por lo que podemos indicar en qué instrucción empezar la depuración. De esta forma creamos un **breakpoint** o un punto de ruptura, con el que conseguimos que cuando la ejecución llegue a esa instrucción, se pare y empiece a hacer la depuración.

Un breakpoint (o punto de ruptura o punto de parada) es el mecanismo que nos va a permitir detener el flujo de ejecución de un programa en una instrucción en concreto.

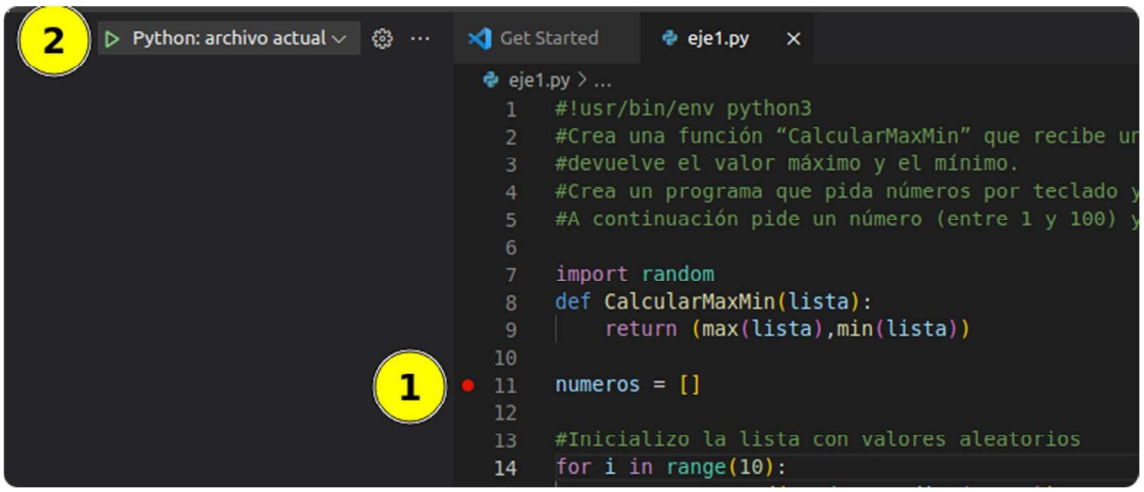
En este ejemplo vamos a depurar este programa:

```
    #!usr/bin/env python3
    #Crea una función "CalcularMaxMin" que recibe una lista con
valores numéricos y
    #devuelve el valor máximo y el mínimo. Crea un programa que
complete una lista de numeros
    #aleatorios (entre 1 y 100) y muestre el máximo y el mínimo,
utilizando la función anterior.
```

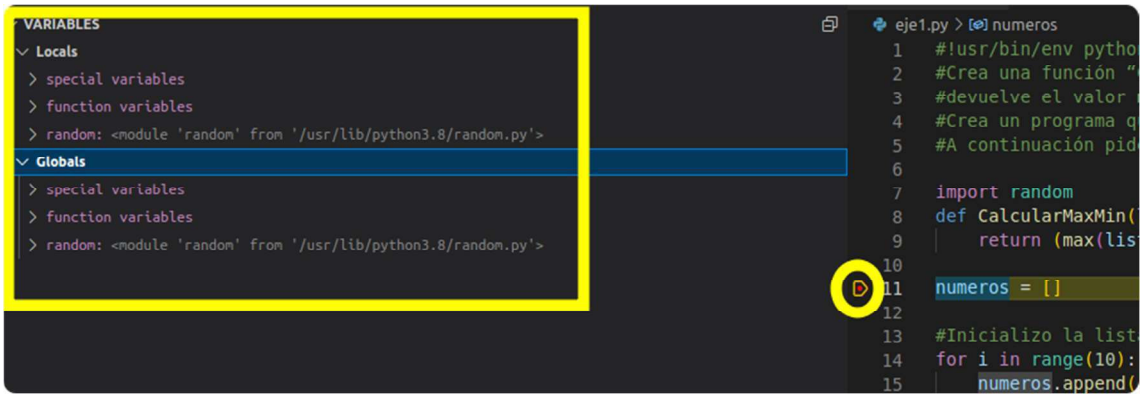
```
    #Por último, pide un número (entre 1 y 100) y el programa debe
decir si está en la lista anterior.
    import random
    def CalcularMaxMin(lista):
        return (max(lista),min(lista))


    numeros = []
    #Inicializo la lista con valores aleatorios
    for i in range(0..10):
    numeros.append(random.randint(1,1000))
    vmax,vmin = CalcularMaxMin (numeros)
    print("El valor máximo es ",vmax)
    print("El valor mínimo es ",vmin)

    numero = int(input("Dime un número del 1 al 100:"))
    while numero > 0 or numero < 100:
    print("El número debe estar entre 1 y 100")
    numero = int(input("Dime un número del 1 al 100:"))

    if numero in numero:
    print("El número está en la lista")
    elsa:
    print("El número no está en la lista")
```

Nota: El programa no es correcto. Corrige los errores antes de hacerlo funcionar. Una vez funcione, tendrás que depurarlo para que funcione de acuerdo a lo que se pide.

Marcamos un breakpoint en la línea número 11 y pulsamos el botón **Start debugging** para comenzar la depuración.

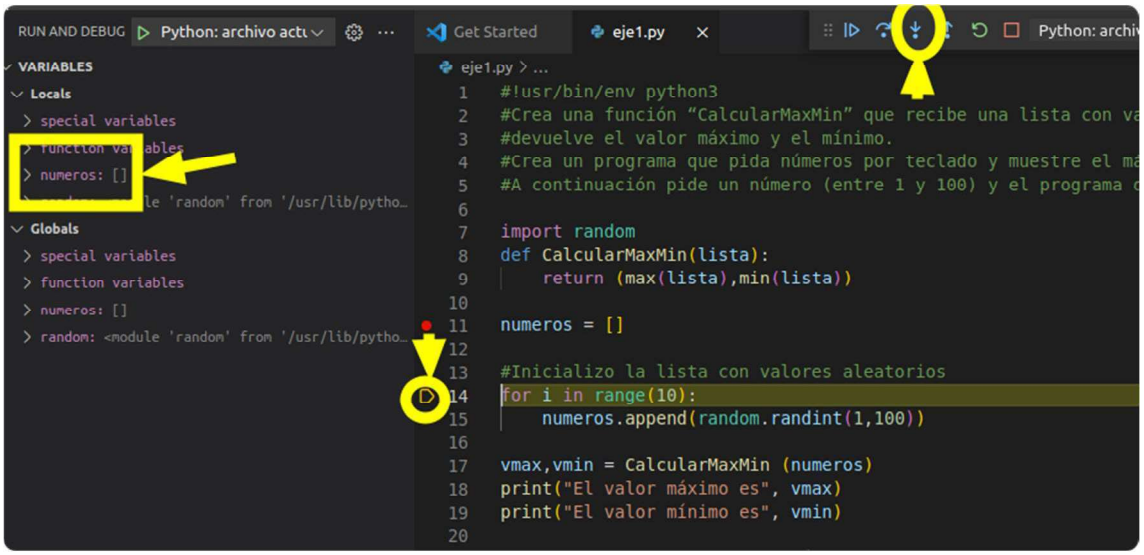


Podemos ver que el programa resalta en color la próxima instrucción que se va a ejecutar. Al mismo tiempo observamos que en la parte superior izquierda nos muestra los valores de las variables con las que estamos trabajando. (En el apartado **Variables: locals y globals**)

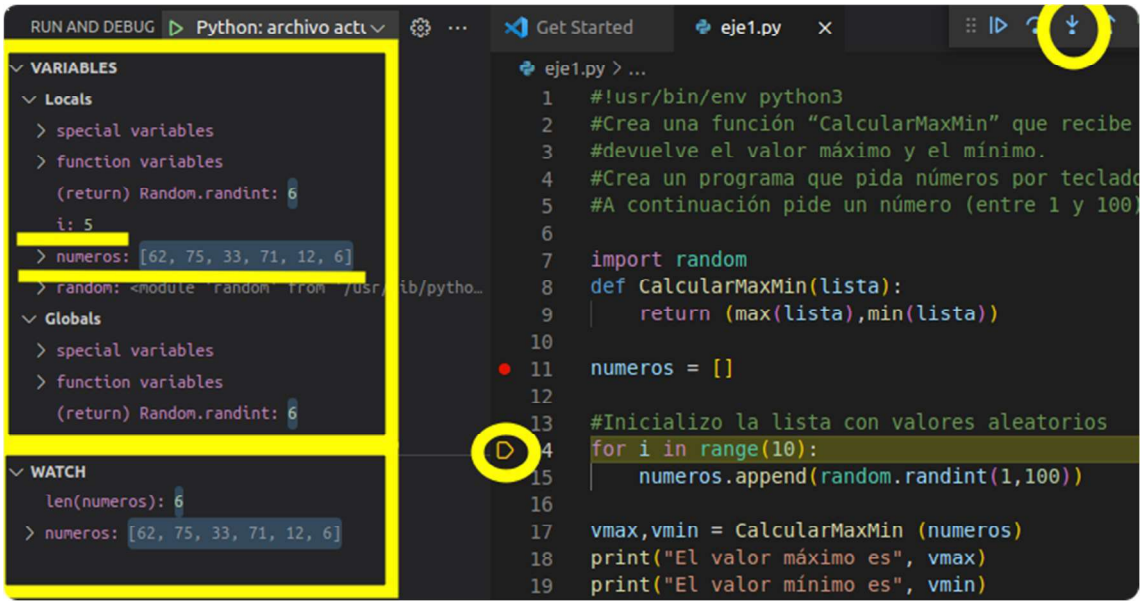


A continuación, ejecutamos la siguiente instrucción, pulsando la opción **Step Into** o la tecla **F11**, observamos que se ha creado la variable `numeros` se ha creado y es una lista vacía.

Step into: Ejecuta una sentencia y en el caso de ser la llamada a una función, entra dentro de esta para depurarla paso a paso.

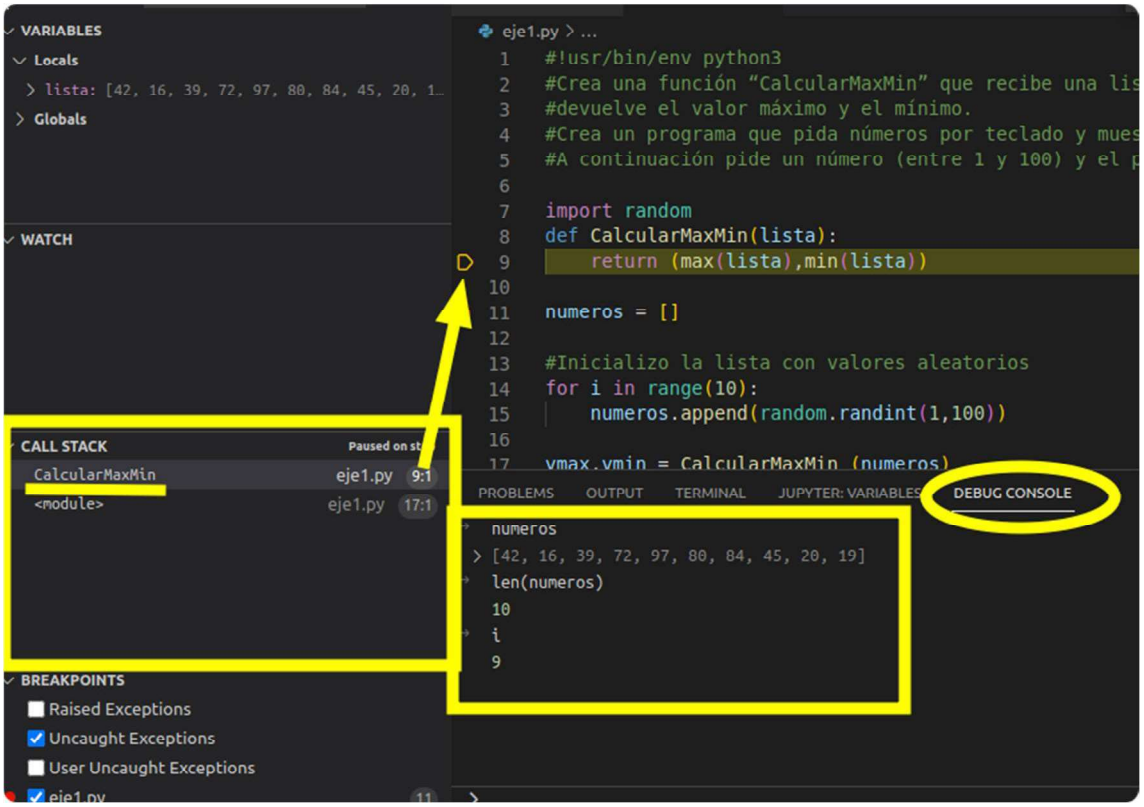


Seguimos ejecutando la depuración con la tecla **F11** y vemos que el programa entra en un bucle y se van añadiendo números aleatorios a la lista. En el apartado de **Variables** vemos que la variable `numeros` va cambiando, y también su longitud, lo que podemos también en el apartado **Watch**, si añadimos una expresión, por ejemplo: `len(numeros)` o solo `numeros`



Como podemos comprobar es una forma muy sencilla de ver qué instrucciones se están ejecutando, además de poder ver en cualquier momento los valores de las variables.

En cualquier momento podemos ir a la **consola de depuración**, seleccionando en **Debug Console**. Esta consola se abre en la parte inferior de la pantalla, y en la que podemos ejecutar instrucciones Python en el entorno de nuestro programa, es decir, con el valor de las variables y tal como en ese momento está ejecutado.



Por ejemplo podemos lanzar las ejecuciones de las sentencias `len(numeros)` o solo `numeros`.

Si en la consola de depuración escribimos la variable numeros, nos muestra el valor actual de la misma.

También podemos hacer instrucciones más complicada, por ejemplo, podemos comprobar si el número 1 está en la variable `numeros`, escribiendo `1 in numeros`.

Si continuamos ejecutando el programa, la instrucción for va a tener diez iteraciones, y cuando termine pasamos a la ejecución de una función en la línea 8.

Si estamos seguros que dicha función se ejecuta sin problemas y no necesitemos depurarla, podemos pulsar la opción **Step Over** o pulsar la tecla **F10**, y en ese caso se ejecuta la función pero sin entrar dentro de esta para depurar paso a paso.

Step over: Ejecuta una sentencia y en el caso de ser la llamada a una función, no entra dentro de esta para depurarla.

Si realmente queremos ver cómo se ejecuta esa función, pulsamos **F11** y la ejecución entraría en la función, calcularía el máximo y el mínimo de la lista y nos devolvería esos valores.

Por último, en el apartado **Calla Stack** podremos ver la pila de llamadas que se ha hecho en nuestro programa, pudiendo hacer un seguimiento de qué función ha llamado a qué otra función.

Esta práctica es muy recomendable para todo el que esté comenzando a programar, para localizar los posibles errores de programación que se hayan cometido, y además para aprender cómo se ejecutan las instrucciones dentro de nuestro programa.

Fuente

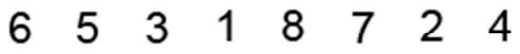
- Como depurar, Open Webinars
- Depurar con Pdb

* Práctica 2.4: Depurar programas

P2.4 - Ejercicio.

El algoritmo burbuja

El algoritmo burbuja te permite ordenar valores de un array. Funciona revisando cada elemento con el elemanto adyacente. Si ambos elementos no están ordenados, se procede a intercambiarlos, si por el contrario los elementos ya estaban ordenados se dejan tal como estaban. Este proceso sigue para cada elemento del arreglo hasta que quede completamente ordenado.



Ahora, vamos a entender cómo podemos programar este algoritmo.

Entendiendo el proceso

Vamos a ordenar la lista `a` de longitud `n=5`:

```
a = [8, 3, 1, 19, 14]
```

El algoritmo burbuja se compone de 2 bucles, uno dentro del otro. Llamaremos “bucle hijo” al que se encuentra dentro del otro bucle, es decir del “bucle padre”. Estos nombres son solo para que entiendas.

El “bucle padre” realizará el número de iteraciones necesarias para ordenar la lista (las iteraciones necesarias son n-1 veces) y el “bucle hijo” se encargará de comparar cada elemento con su adyacente y ordenarlos.

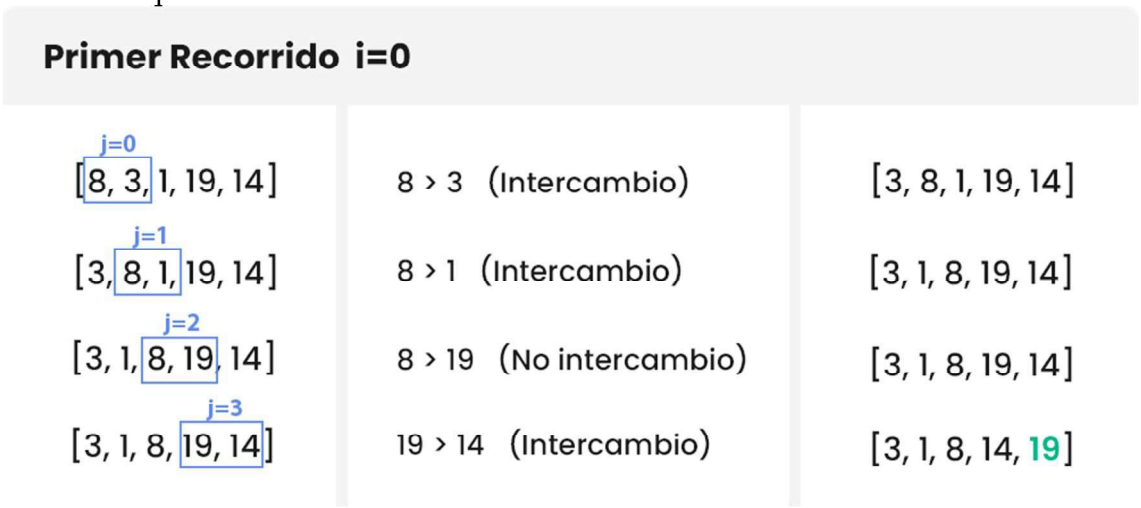
Si deseamos ordenar la lista `a`, el padre y el hijo comenzarán recorriendo n-1 veces (es decir, 4 veces), teniendo en cuenta, que cuando el padre realice una iteración, el número de iteraciones del hijo se irá reduciendo: comienza con n-1-0 iteraciones, luego n-1-1 iteraciones, luego n-1-2 iteraciones…

Un ejemplo gráfico

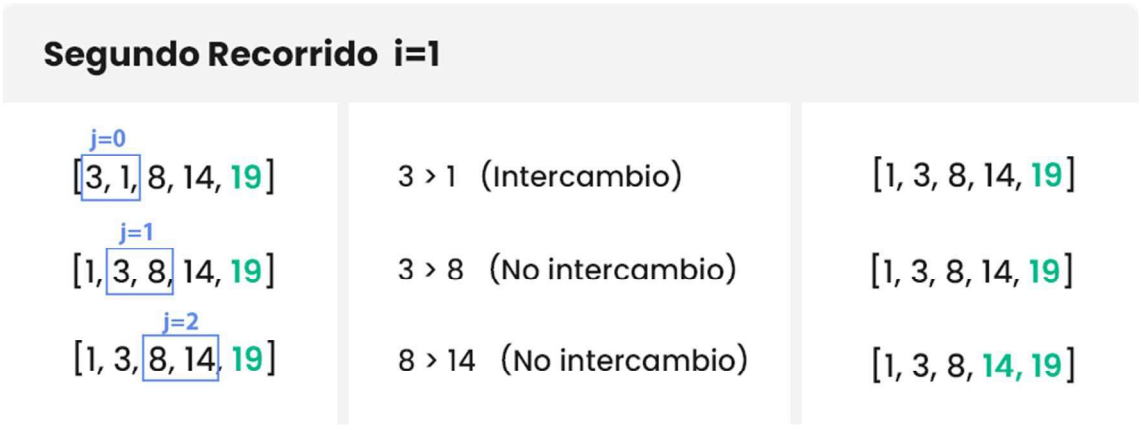
Parece muy complicado, ¿cierto? Analicemos esto gráficamente.

En los siguientes gráficos, el bucle padre tiene índice "i" y el bucle hijo tiene el índice "j", y recuerda, iniciaremos ambos bucles desde la posición 0 hasta el 3 (casi siempre los bucles inician desde el cero). Si te das cuenta 0, 1, 2 y 3 son igualmente los 4 recorridos que buscamos.

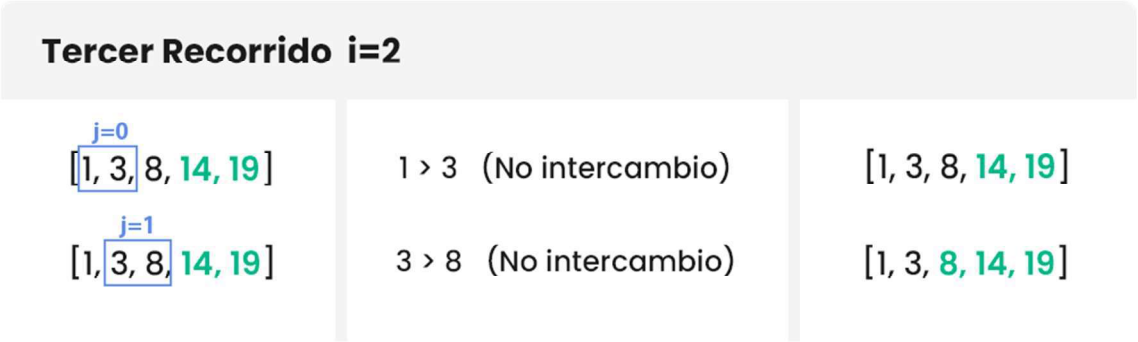
Pasemos al primer recorrido:



Primer recorrido del bucle padre `i=0`: el bucle hijo con índice `j` recorre desde 0 a 3 `(n-1)`. Como se puede apreciar, cada elemento es comparado con su adyacente. Si están ordenados correctamente se pasa a comparar con el siguiente elemento, y si no están ordenados se realiza un intercambio.

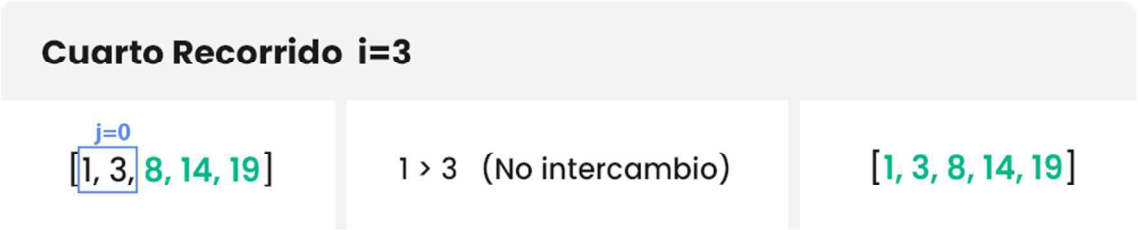


Segundo recorrido del bucle padre `i=1`: El bucle hijo con índice `j` reduce su rango, ahora va desde 0 a 2 `(n-1-1)` y ya no se evalúa el último elemento (el 19, de color verde) porque ya está ordenado. Además, en este recorrido se realiza solamente un intercambio y la lista queda completamente ordenada. Luego agregamos al 14 a la lista de elementos ordenados.



Tercer recorrido del bucle padre `i=2`: El bucle hijo con índice `j` sigue reduciendo su rango, con valores desde 0 a 1 `(n-1-2)`, porque los últimos elementos ya no se evalúan (porque están ordenados) y se van acumulando.

En este punto ya no existen intercambios, pero el algoritmo va a recorrer hasta i=n-1. No importa si la lista esta ordenada o no. En nuestro caso realizará un "Cuarto Recorrido", el cual es innecesario. Por este motivo existe una variación de este algoritmo que evita que se hagan recorridos extra una vez que la lista ya esta ordenada (en este artículo te enseñaré a implementar el original y la variación).



Cuarto recorrido del bucle padre `i=3`: El bucle hijo con índice `j` solo toma el valor de 0 `(n-1-3)`. Verifica que estén ordenados correctamente y el bucle padre llega al final de su recorrido.

Práctica

Ahora que ya sabemos cómo funciona el algoritmo de burbuja, pasemos a la práctica. Implementación en Python y utiliza el debugger para asegurarte que funciona adecuadamente y entiendes su funcionamiento.

Entrega: 1. El algoritmo implementado en src/main.py 2. Evidencia de haberlo debugueado: Capturas de pantallas con un (a) punto de ruptura establecido y el (b) valor que toman alguna variable mientras se está ejecutando. 3. Una evidencia de haberlo ejecutado y haber funcionado: Copia la salida por consola tras ejecutar tu programa.