

# Laboratorio de Programación

## Debugging en Python

Introducción a la Programación

Departamento de Computación, FCEyN, Universidad de  
Buenos Aires.

# Debugging

¿Qué es debuggear (en español, depurar)?

# Debugging

¿Qué es debuggear (en español, depurar)?

Realizar un seguimiento línea por línea de nuestro programa usando un debugger.

# Debugging

¿Qué es debuggear (en español, depurar)?

Realizar un seguimiento línea por línea de nuestro programa usando un debugger.

¿Y qué es un debugger?

# Debugging

## ¿Qué es debuggear (en español, depurar)?

Realizar un seguimiento línea por línea de nuestro programa usando un debugger.

## ¿Y qué es un debugger?

Es un programa que toma como entrada otro programa (un binario) y nos permite controlar el flujo de ejecución. En casi todos los IDEs es una herramienta que ya viene incorporada.

# ¿Hace falta?

## Alternativa 1: Imprimir por pantalla los resultados parciales.

A favor:

- ▶ Nos da una idea del recorrido del programa.

En contra:

- ▶ Proceso que consume mucho tiempo: agregar el código necesario, recompilar todo, correr el programa y analizar la salida. Todo eso cada vez que encontramos un bug.
- ▶ Ensuciamos el código.
- ▶ Al final hay que borrar todo lo que agregamos.

# ¿Hace falta?

## Alternativa 1: Imprimir por pantalla los resultados parciales.

A favor:

- ▶ Nos da una idea del recorrido del programa.

En contra:

- ▶ Proceso que consume mucho tiempo: agregar el código necesario, recompilar todo, correr el programa y analizar la salida. Todo eso cada vez que encontramos un bug.
- ▶ Ensuciamos el código.
- ▶ Al final hay que borrar todo lo que agregamos.

## Alternativa 2: Tratar de encontrar el error mirando el código.

A favor:

- ▶ Más rápido.

En contra:

- ▶ Para algoritmos no triviales, es difícil darse cuenta.

# Cómo lo usamos

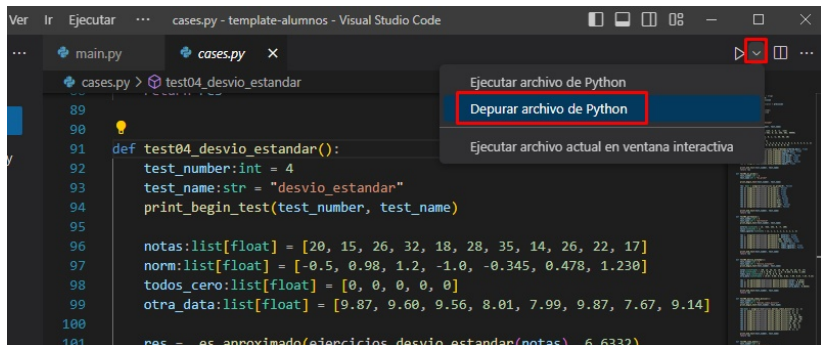
1. Compilamos en modo debug. Esto introduce en nuestro programa símbolos especiales para ser usados por el debugger. Esto es transparente para nosotros desde VS Code.
2. Corremos el programa con el debugger.

Conceptos fundamentales:

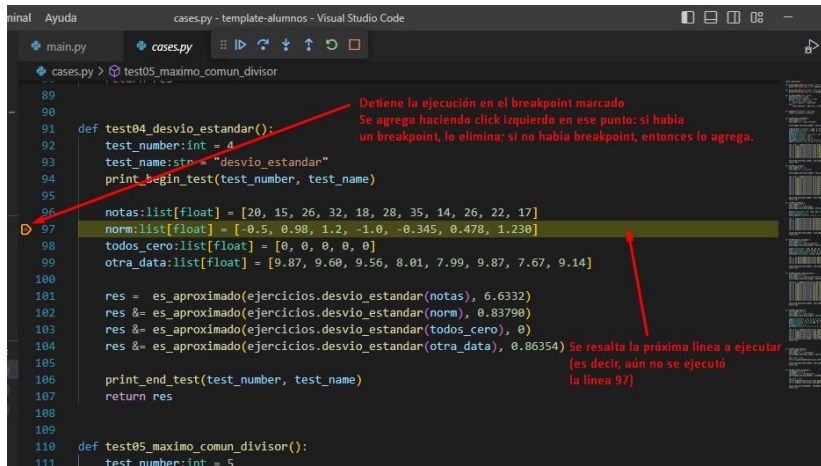
- ▶ *Breakpoint*: Suspender la ejecución del programa en una línea en particular.
- ▶ *Step Over*(F10): Ir hacia la siguiente línea. No importa si es un llamado a una función o algo más complejo, se ejecutan todas las instrucciones de esa línea.
- ▶ *Step into*(F11): Meterse dentro de una función.
- ▶ *Step out*(Shift+F11): Salir de una función.
- ▶ *Watchpoint o Inspección*: Hacer el seguimiento de una variable durante el transcurso de una función/programa.
- ▶ *Stacktrace/frames*: Ver en orden todas las funciones que fueron invocadas hasta el momento.



# Iniciar Debugger en VS Code



# Agregar Breakpoint



terminal Ayuda cases.py - template-alumnos - Visual Studio Code

main.py cases.py

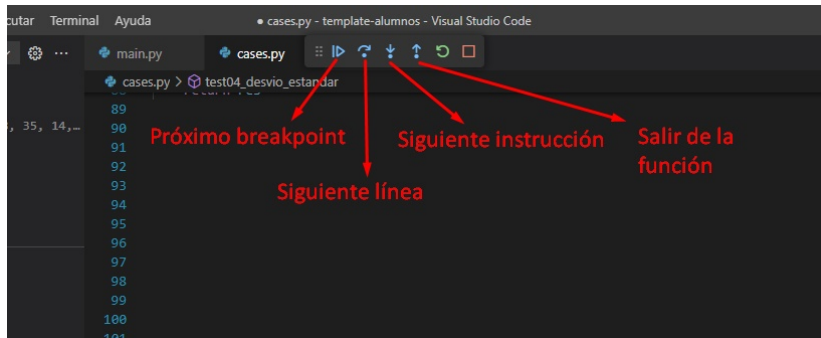
cases.py > test05\_maximo\_comun\_divisor

```
89
90
91 def test04_desvio_estandar():
92     test_number:int = 4
93     test_name:str = "desvio_estandar"
94     print_begin_test(test_number, test_name)
95
96     notas:list[float] = [20, 15, 26, 32, 18, 28, 35, 14, 26, 22, 17]
97     norm:list[float] = [-0.5, 0.98, 1.2, -1.0, -0.345, 0.478, 1.230]
98     todos_cero:list[float] = [0, 0, 0, 0, 0]
99     otra_data:list[float] = [9.87, 9.60, 9.56, 8.01, 7.99, 9.87, 7.67, 9.14]
100
101     res = es_aproximado(ejercicios.desvio_estandar(notas), 6.6332)
102     res &= es_aproximado(ejercicios.desvio_estandar(norm), 0.83790)
103     res &= es_aproximado(ejercicios.desvio_estandar(todos_cero), 0)
104     res &= es_aproximado(ejercicios.desvio_estandar(otra_data), 0.86354)
105
106     print_end_test(test_number, test_name)
107     return res
108
109
110 def test05_maximo_comun_divisor():
111     test_number:int = 5
```

Detiene la ejecución en el breakpoint marcado  
Se agrega haciendo click izquierdo en ese punto: si habia un breakpoint, lo elimina; si no habia breakpoint, entonces lo agrega.

Se resalta la próxima línea a ejecutar (es decir, aún no se ejecutó la línea 97)

# Funciones principales



# Inspeccionar Variables

**VARIABLES**

- Locals
  - > notas: [20, 15, 26, 32, 18, 28, 35, 14, ...]
  - test\_name: 'desvio\_estandar'
  - test\_number: 4
- Globals

**INSPECCIÓN**

- test\_number: 4
- len(notas): 11
- test\_number > 1: True

**PILA DE LLAMADAS**

En pausa en breakpoint
test04_desvio_estandar cases.py 97:1
main main.py 20:1
<module> main.py 43:1

**PUNTOS DE INTERRUPCIÓN**

- ☐ Raised Exceptions
- ☒ Uncaught Exceptions

**cases.py - template-alumnos - Visual Studio**

**cases.py**

```
89 Lista las variables existentes y sus valores  
90 hasta el punto de ejecución actual.  
91 def test04_desvio_estandar():  
92     test_number: int = 4  
93     test_name: str = "desvio_estandar"  
94     print_begin_test(test_number, test_name)  
95  
96     notas: list[float] = [20, 15, 26, 32, 18,  
97     norm: list[float] = [-0.5, 0.98, 1.2, -1.4  
98     todos_cero: list[float] = [0, 0, 0, 0, 0]  
99     otra_data: list[float] = [9.87, 9.60, 9.50  
100  
101     res = es_aproximado(ejercicios.desvio_e  
102     res &= es_aproximado(ejercicios.desvio_e  
103     res &= es_aproximado(ejercicios.desvio_e  
104     res &= es_aproximado(ejercicios.desvio_e  
105  
106     print_end_test(test_number, test_name)  
107     return res  
108  
109  
110 def test05_maximo_comun_divisor():  
111     test_number: int = 5  
112     test_name: str = "maximo_comun_divisor"  
113     print_begin_test(test_number, test_name)  
114  
115     res: bool = asegurar(ejercicios.maximo_co  
116     res &= asegurar(ejercicios.maximo_comun  
117     res &= asegurar(ejercicios.maximo_comun
```

**Permite agregar variables. También se puede ejecutar funciones sobre variables Por ejemplo len(notas)**