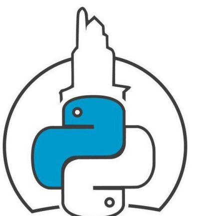
Introducción a los **depuradores**

Python Coruña, 2023/03/10

Diego Moreda Rodríguez



Depurador: definición académica

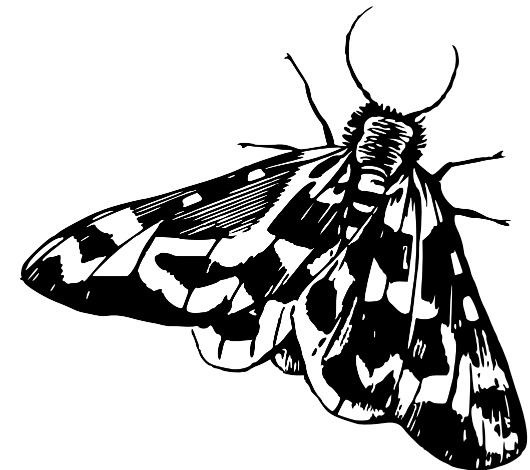
debugger

/ˌdiːˈbʌɡə/

noun.

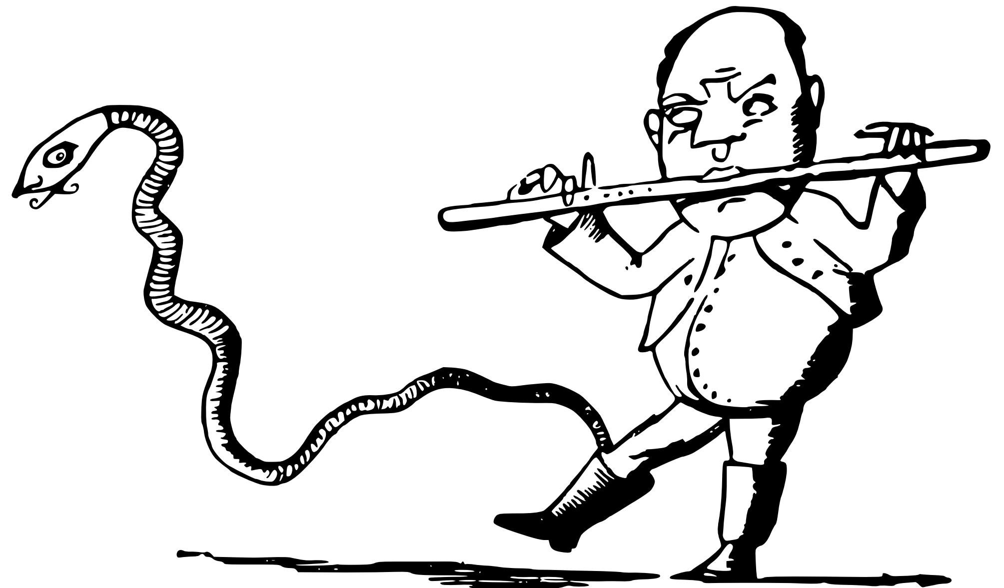
a computer program that assists in the detection and correction of errors in other computer programs.

Fuente: Oxford Dictionary



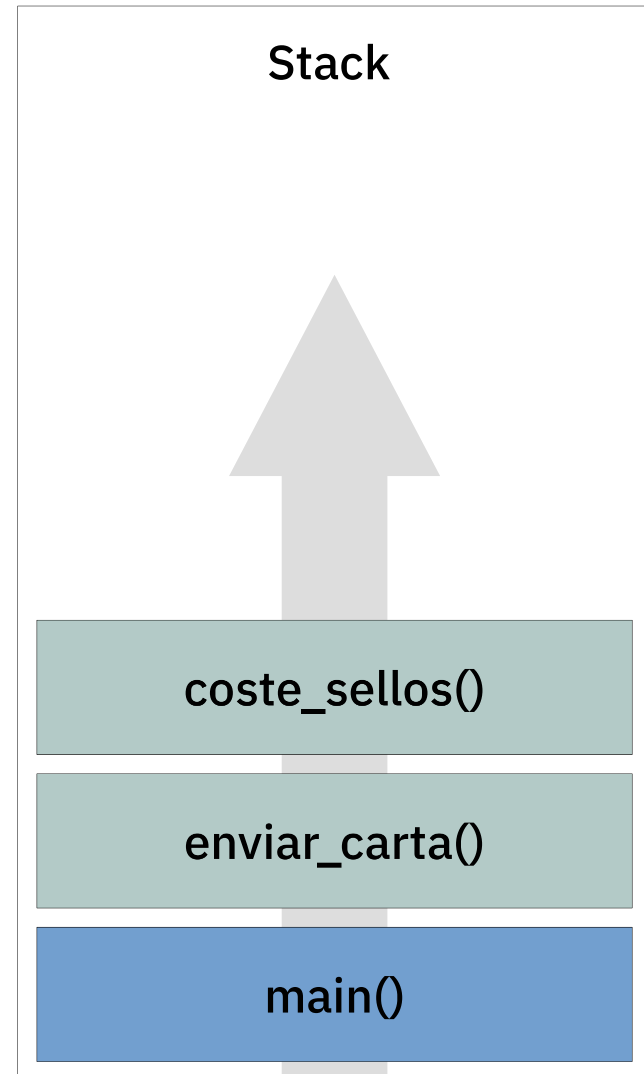
Depurador: definición romántica

1. **Controlar** el paso del tiempo
2. **Navegar** por tus abstracciones (y las de otros)
3. **Observar** qué está ocurriendo con detalle
4. Manipular, modificar, y probar teorías e ideas de forma **interactiva**

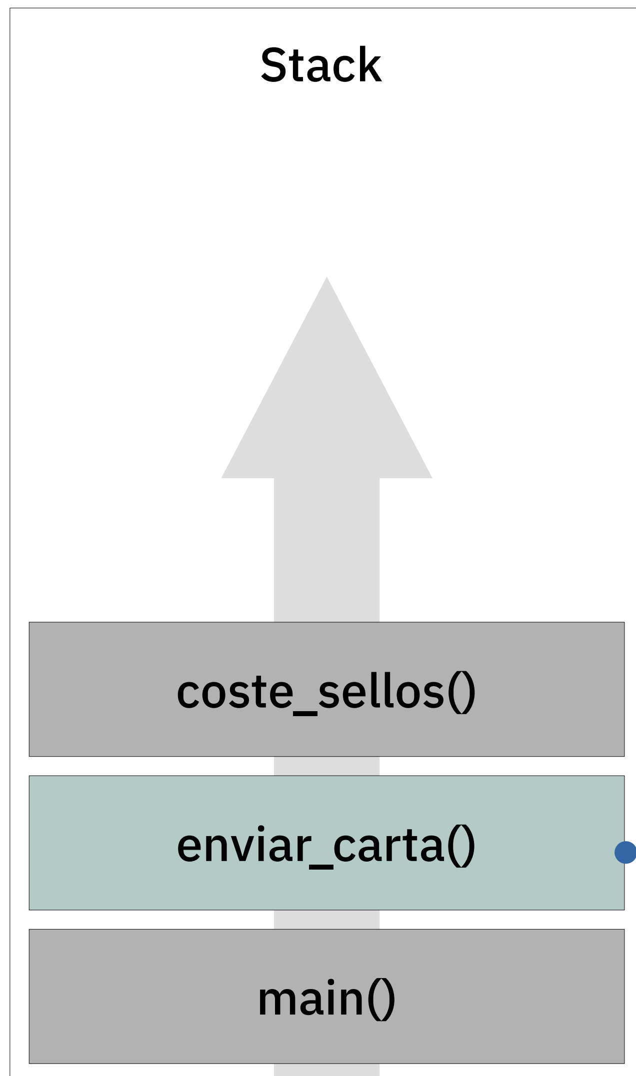


Un poco de teoría: call stack

```
   ...
01 def enviar_carta(persona, texto):
02     contenido = chatgpt.mejora(texto)
03     direccion = persona.direccion()
04 
05     if coste_sellos(direccion) < 0.5:
06         correos.envia(
07             persona,
08             contenido)
09 
10 def coste_sellos(direccion):
11     print(”calculando importe”)
12     importe = importes[direccion.pais]
13 
14     return importe*1.21
15 ...
16 
17 enviar_carta(juan, “archivo.txt”)
```



Un poco de teoría: call stack, 2



En cada “frame”:

- nombre de la función
- argumentos
- referencia al fichero y su línea
- *locals*: variables en scope

```
def enviar_carta(persona, texto):
    contenido = chatgpt.mejora(texto)
    direccion = persona.direccion()

    if coste_sellos(direccion) < 0.5:
```

Depurador: definición algo más técnica

1. *Controlar el paso del tiempo*

- **“breakpoints”** - puntos de interés en los que detener la ejecución
  - pueden ser condicionales, temporales, ...
- control fino sobre la **ejecución**
  - siguiente línea, retorno de función, dentro de la función, siguiente breakpoint ...

Depurador: definición algo más técnica, 2

2. *Navegar por tus abstracciones (y las de otros)*

- saltar entre **frames** del stack

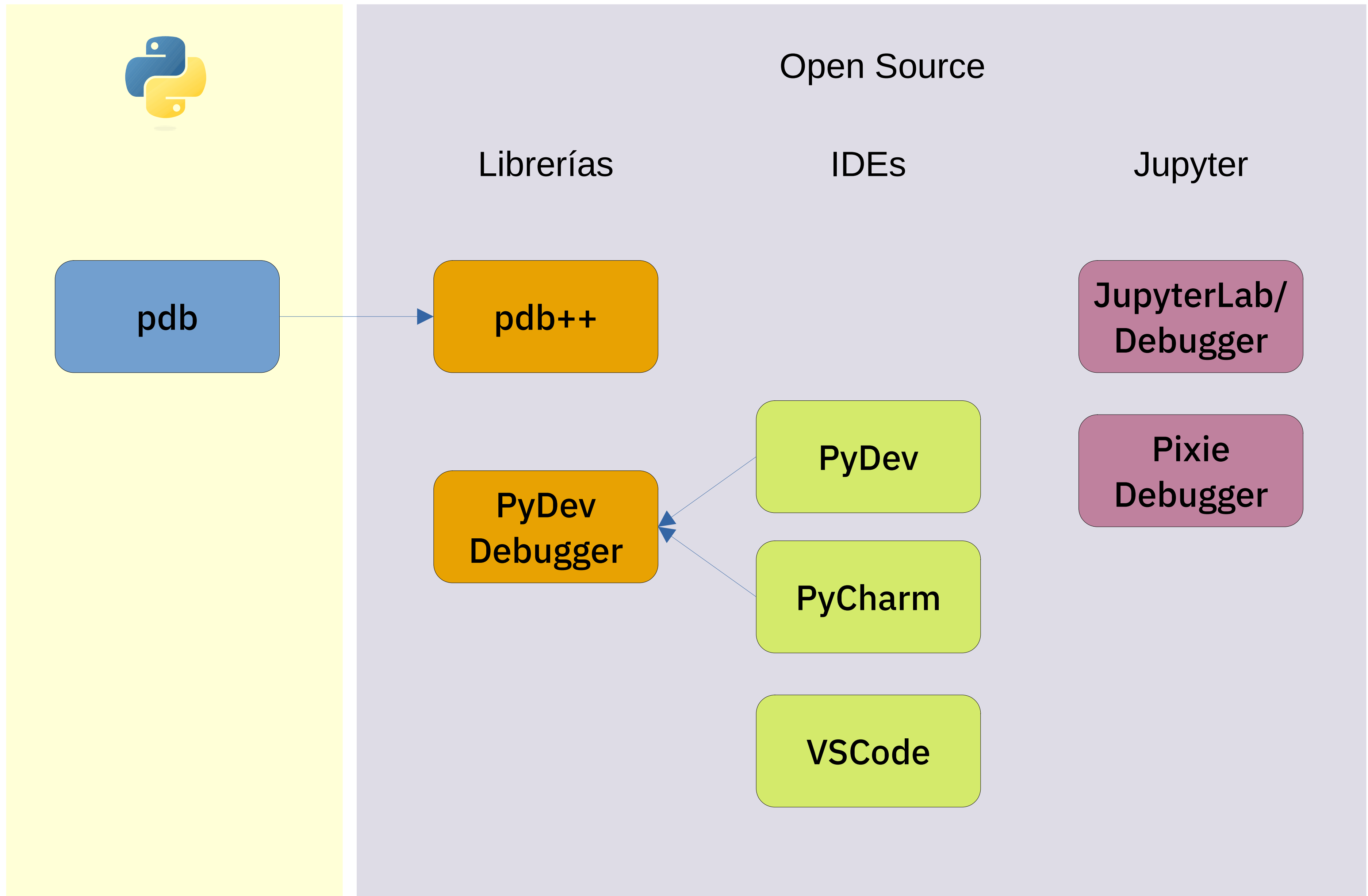
3. *Observar qué está ocurriendo con detalle*

- toda la información, y **expresiones** personalizadas
- uso de breakpoints de forma **dinámica**

4. *Manipular, modificar, y probar teorías e ideas de forma interactiva*

- **intérprete** de Python en el contexto que quieras!

Mapa (incompleto) de depuradores



pdb: “batteries included”

+demo!

- Incluido en la librería standard
- Establecer un breakpoint:

```
import pdb;
pdb.set_trace()
```

- Ejecutar un script bajo pdb:

```
python3 -m pdb mi_script.py
```

- Interactivo, a través de diferentes comandos ... *y un tanto árido*

https://docs.python.org/3/library/pdb.html

Python Debugger Cheatsheet

Getting started

import pdb;pdb.set_trace()	start pdb from within a script
python -m pdb <file.py>	start pdb from the commandline

Basics

h(elp)	print available commands
h(elp) *command*	print help about *command*
q(quit)	quit debugger

Examine

p(rint) *expr*	print the value of *expr*
pp *expr*	pretty-print the value of *expr*
w(here)	print current position (including stack trace)
l(ist)	list 11 lines of code around the current line
l(ist) *first, last*	list from *first* to *last* line number
a(rgs)	print the args of the current function

Miscellaneous

!*stmt*	treat *stmt* as a Python statement instead of a pdb command
alias *map stmt*	map Python statement as a map command
alias *map* <*arg1 ...*> *stmt*	pass arguments to Python statement. *stmt* includes %1, %2, ... literals.

Save pdb commands to local <./.pdbrc> file for repetitive access.

Movement

<ENTER>	repeat the last command
n(ext)	execute the current statement (step over)
s(tep)	execute and step into function
r(eturn)	continue execution until the current function returns
c(ontinue)	continue execution until a breakpoint is encountered
u(p)	move one level up in the stack trace
d(own)	move one level down in the stack trace
until	continue execution until the end of a loop or until the set line
j(ump)	set the next line that will be executed (local frame only)

Breakpoints

b(reak)	show all breakpoints with its *number*
b(reak) *lineno*	set a breakpoint at *lineno*
b(reak) *lineno, cond*	stop at breakpoint *lineno* if Python condition *cond* holds, e.g. i==42
b(reak) *file:lineno*	set a breakpoint in *file* at *lineno*
b(reak) *func*	set a breakpoint at the first line of a *func*
tbreak *lineno*	set a temporary breakpoint at *lineno*, i.e. is removed when first hit
disable *number*	disable breakpoint *number*
enable *number*	enable breakpoint *number*
clear *number*	delete breakpoint *number*

Author: Florian Preinstorfer (nblock@archlinux.us) — version 1.2 — license cc-by-nc-sa 3.0
See `https://github.com/nblock/pdb-cheatsheet` for more information.

IDEs y Jupyter

+demo!

- Entre muchas otras utilidades, contienen un **depurador** incoporporado
- Soporte para debugger en **Jupyter** (reciente)
- Merece la pena familiarizarse y conocerlos:
  - más **visuales** y potentes
  - mantener/corregir lleva mucho más tiempo que desarrollar
  - permite aprender más sobre otras librerías

El depurador es sólo el principio

- Una herramienta más en tu arsenal - y una muy útil
- Otras técnicas y conceptos relacionados:
  - Profiling / tracing
  - Instrospección
  - Monkey-patching
- ***Know your tools!***