6. Localización, depuración y resolución de problemas

Introducción a Python

Contenidos

Contenidos

1. Introducción: ¿Qué es?

Proceso que involucra:

- **Localización:**
  - Identificar la fuente del problema
- **Depuración:**
  - Analizar el flujo de ejecución y variables
- **Resolución:**
  - Implementar correcciones de forma controlada

Objetivo Final

- Prevenir que el problema ocurra de nuevo mediante una **solución robusta y documentada**

1. Introducción: Importancia

- Garantizar el **funcionamiento correcto** del software
- Mejorar la **calidad** y la **mantenibilidad**
- Reducir **costos** a largo y corto plazo
- Gran auge de los asistentes de código

1. Introducción: Importancia

Caso Mariner I (1962):

- Un bug en el software causó que el cohete se desviara de su camino
- Solo 4 mins 53 segs funcionando
- Para evitar problemas lo explotaron.

1. Introducción: Importancia

Soviet gas pipeline explosion (1982):

- Guerra Fría, Unión soviética quería conseguir material de EEUU.
- La CIA plantó un bug en un ordenador canadiense.
- La mayor explosión no nuclear.

1. Introducción: Importancia

Incidente Therac-25 (1985-1987):

- Fallo por una condición de carrera en el SO.
- Un rayo de alta radiación fuera de precisión.
- 4 muertos y 3 heridos por exceso de radiación.

1. Introducción: Los bugs son comunes

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature
- Programar no es estresante

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature
- Programar no es estresante

Carlos, 24 años

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature
- Programar no es estresante
- Hay un poco de código en tu error

Carlos, 24 años

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature
- Programar no es estresante
- Hay un poco de código en tu error
- ¿Por qué no compila?

Carlos, 24 años

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature
- Programar no es estresante
- Hay un poco de código en tu error
- ¿Por qué no compila?
- ¿Por qué compila?

Carlos, 24 años

1. Introducción: Los bugs son comunes

Por algo son una gran fuente de memes:

- No es un Bug, es una Feature
- Programar no es estresante
- Hay un poco de código en tu error
- ¿Por qué no compila?
- ¿Por qué compila?
- Error en línea 42 (No hay nada)

Carlos, 24 años

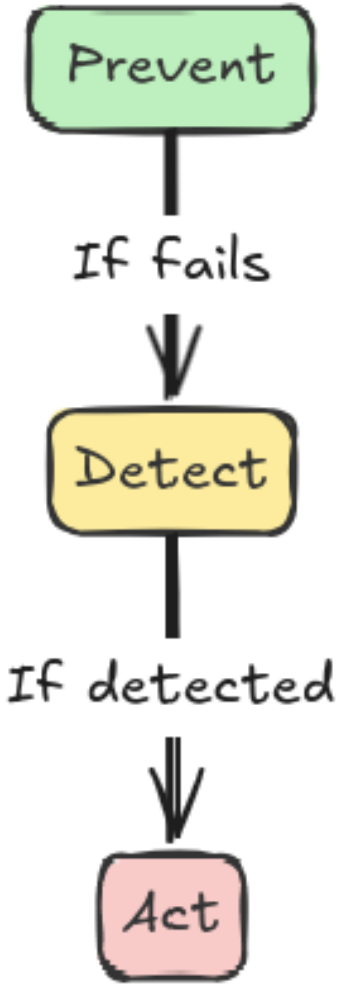
1. Introducción: Los bugs son comunes

Debuggear es como un juego clásico de misterio, solo que eres detective, víctima y verdugo al mismo tiempo.

1. Introducción

Un sistema que garantiza la seguridad previene, detecta y actúa.

- **Prevenir:** evitar errores en el código
- **Detectar:** identificar rápidamente los errores cuando ocurren
- **Actuar:** solucionar errores eficazmente



1. Introducción: Los bugs son comunes

“Program testing can be used to show the presence of bugs, but never to show their absence!”

— Edsger W. Dijkstra

1. Introducción: Tipos de errores

- **Errores de Sintaxis:** fallos de escritura en el código.
- **Errores de Ejecución:** problemas durante la ejecución.
- **Errores Lógicos:** resultados inesperados

Contenidos

Contenidos

2. Uso de print: Concepto

- **Concepto:** Usar `print` para observar valores de variables y el flujo del código
- **Objetivo:** Identificar errores rápidamente
- **Aplicación:** Ideal para depurar pasos sencillos y código en desarrollo temprano

2. Uso de print: Buenas prácticas

Ubicación Estratégica:

- Imprimir antes y después de puntos clave del flujo
- Colocar `print` en funciones para verificar llamadas y retornos

Formateo de Strings:

- `f-strings y str.format`
- Claridad en mensajes

2. Uso de print: Concepto

Ventajas:

- Fácil de usar y sin herramientas adicionales
- Eficaz para depurar en casos simples
- Ideal para flujo al prototipar

Desventajas:

- Poco eficaz en código complejo
- Saturación de mensajes en el código (Limpiar).
- No ofrece contexto detallado como un depurador o `logging`

Contenidos

Contenidos

3. Archivos de registro: Qué es

Definición: Proceso de registrar eventos y mensajes sobre el funcionamiento de una aplicación.

Está en la librería estándar de Python

Este ayuda a:

- Diagnosticar problemas en código.
- Monitorear el comportamiento en producción.
- Auditar y analizar incidentes.

3. Archivos de registro: Mejor que el print

- Permite persistir los mensajes de registro fácilmente.
- No es necesaria la limpieza de los mensajes (tipos de logging)
- Muchas opciones de personalización.
- Más fácil de estructurar.

3. Archivos de registro: Niveles de logging

Nivel	Cuando es usado
`DEBUG`	Información detallada, típicamente de interés sólo durante el diagnóstico de problemas.
`INFO`	Confirmación de que las cosas están funcionando como se esperaba.
`WARNING`	Un indicio de que algo inesperado sucedió, o indicativo de algún problema en el futuro cercano (por ejemplo, «espacio de disco bajo»). El software sigue funcionando como se esperaba.
`ERROR`	Debido a un problema más grave, el software no ha sido capaz de realizar alguna función.
`CRITICAL`	Un grave error, que indica que el programa en sí mismo puede ser incapaz de seguir funcionando.

Fuente: https://docs.python.org/es/3/howto/logging.html

3. Archivos de registro: Niveles de logging

En resumen:

- **DEBUG**: Mensajes detallados, para desarrollo.
- **INFO**: Mensajes informativos, ejecución normal.
- **WARNING**: Advertencias de posibles problemas.
- **ERROR**: Errores que afectan la ejecución.
- **CRITICAL**: Errores graves, requiere atención inmediata.

3. Archivos de registro: Niveles de logging

Ejemplos:

- **DEBUG**: El valor de x es 42
- **INFO**: La base de datos de virus ha sido actualizada
- **WARNING**: El espacio en disco está bajo. Quedan menos de 20% de espacio.
- **ERROR**: Error en la conexión a la base de datos. Intentando reconectar.
- **CRITICAL**: Fallo crítico en el sistema: todos los servidores están caídos.

3. Archivos de registro: Niveles de logging

Elementos de logging:

- Logger
- Handlers
- Filtros

3. Archivos de registro: Loggers

- Componente central que gestiona la creación, configuración y envío de mensajes de log.
- Una buena práctica es crear uno por módulo.
- Otra buena práctica es establecer un formato personalizado de log.

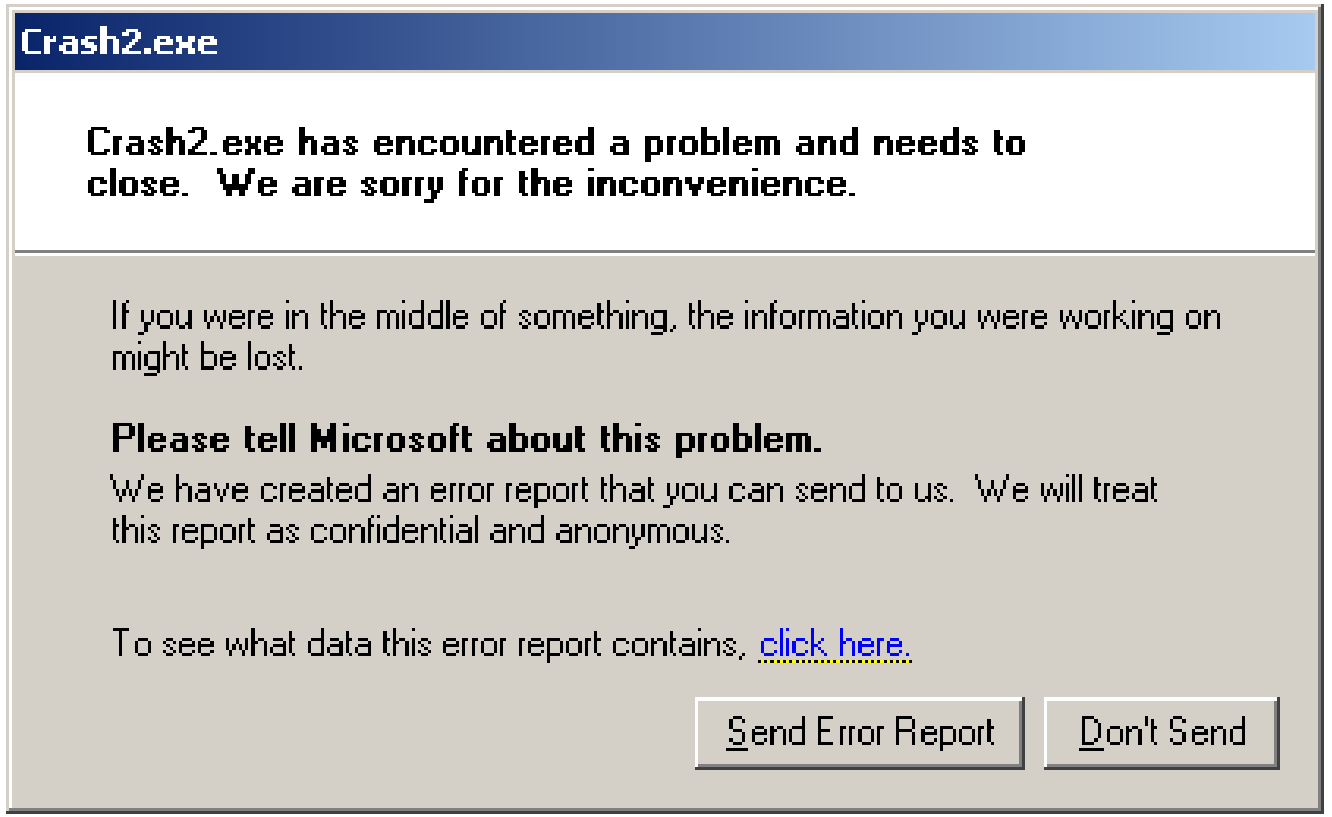
3. Archivos de registro: Handlers

Un handler (manejador) define dónde y cómo se envían los mensajes de log.

¿Por qué usar handlers?

- Mostrar ciertos mensajes en la consola.
- Guardar todos los mensajes en un archivo de log.
- Enviar mensajes críticos por correo o a un servidor de monitoreo.
- Dar un formato a los distintos logs.

3. Archivos de registro: Handlers



3. Archivos de registro: Filtros

Un filtro decide si un mensaje de log debe ser procesado, por quién y bajo qué condiciones.

¿Por qué usar filtros?

- Registrar mensajes solo de ciertos módulos.
- Incluir solo ciertos mensajes que contengan una palabra clave.
- Excluir mensajes que no cumplan ciertas condiciones.

Contenidos

Contenidos

4. Uso del debugger de Python

¿Qué es pdb?

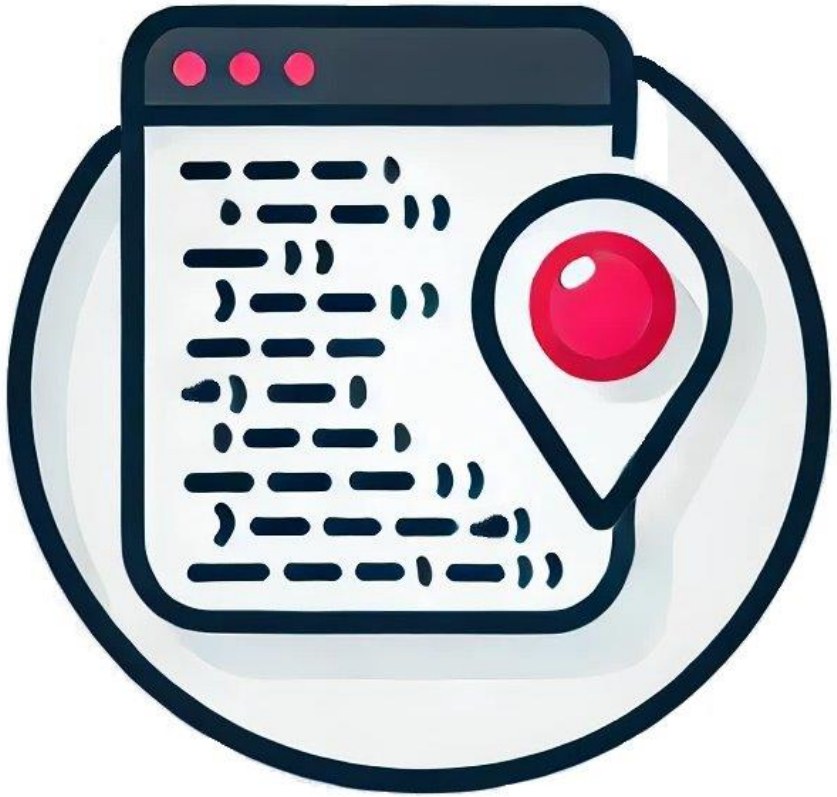
- Depurador interactivo de Python.
- Análisis del flujo de ejecución, examinar variables, y localizar errores.

¿Por qué usar pdb?

- Permite detener el código en puntos específicos.
- Facilita la inspección del estado de la aplicación en tiempo real.

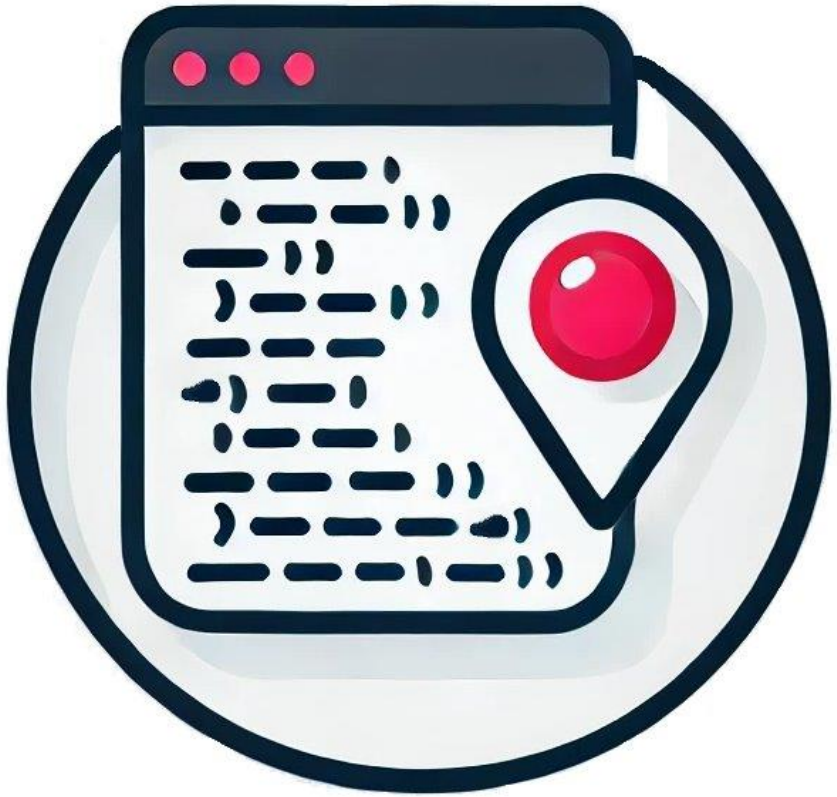
Uso:

- Librería estándar; no es necesario instalarlo.
- Ideal para depurar código sin necesidad de una interfaz gráfica.



4. Uso del debugger de Python: conceptos

- **Breakpoint**: Punto de parada en el código que permite pausar la ejecución
- **Step**: Paso siguiente de ejecución entrando en función
- **Next**: Paso siguiente de ejecución sin entrar en función
- **Continue**: Ejecución hasta el siguiente breakpoint

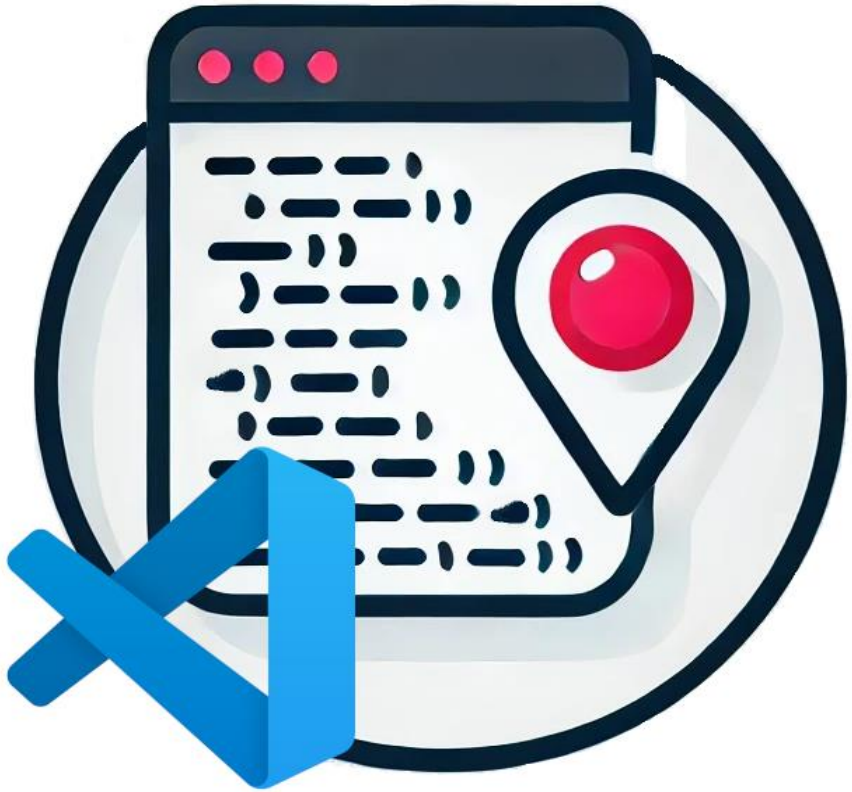


Contenidos

Contenidos

5. Debugger gráfico de Visual Studio Code

- **¡Lo mismo que pdb solo que es gráfico!**
- Mucho más fácil de usar
- ¿Por qué usar pdb entonces?
  - Cuando no hay acceso a interfaz gráfica.
  - Script en sitio remoto



Contenidos

Contenidos

6. Control de errores: ¿Qué es?

- Es la gestión de situaciones inesperadas.

- Manejo de problemas sin detenerse abruptamente.

Ejemplos:

- División por 0
- Fallo al abrir un archivo
- Error de conexión a una BD

6. Control de errores: Importancia

- Mejora la robustez y fiabilidad.
- Facilita la identificación de problemas.
- Ofrece un mejor entendimiento del código.
- Partes vulnerables vistosas.
- Reduce la interrupción del usuario.

6. Control de errores: Buenas prácticas

- Evitar excepciones genéricas
- Mensajes de error claros
- Usar `finally` para limpieza
- Evitar la anidación de `try-except`.

Contenidos

Contenidos

7. Tests unitarios: ¿Qué son?

- Verifica que una pequeña unidad de código.
- Probar un componente aislado.
- Suelen ser código de pruebas automatizados.
- Muy relacionado con el TDD

7. Tests unitarios: Importancia

- Detección temprana de errores.
- Facilitan el mantenimiento del código.
- Facilitan la mejora incremental.
- Documentan cómo debería comportarse el código.

7. Tests unitarios: Buenas prácticas

- Buenos nombres
- Casos normales y excepcionales
- Un test, un comportamiento
- Evitar dependencias entre tests
- Uso de setUp para recursos compartidos
- Limpiar recursos con tearDown

Contenidos

Contenidos

8. Tipado

- Se refiere al tipo de las variables (int, str, list,..)

- Estático (p.j. C++) vs dinámico (p.j. Python)

- Es buena práctica usar anotaciones y mypy para verificar el tipo.

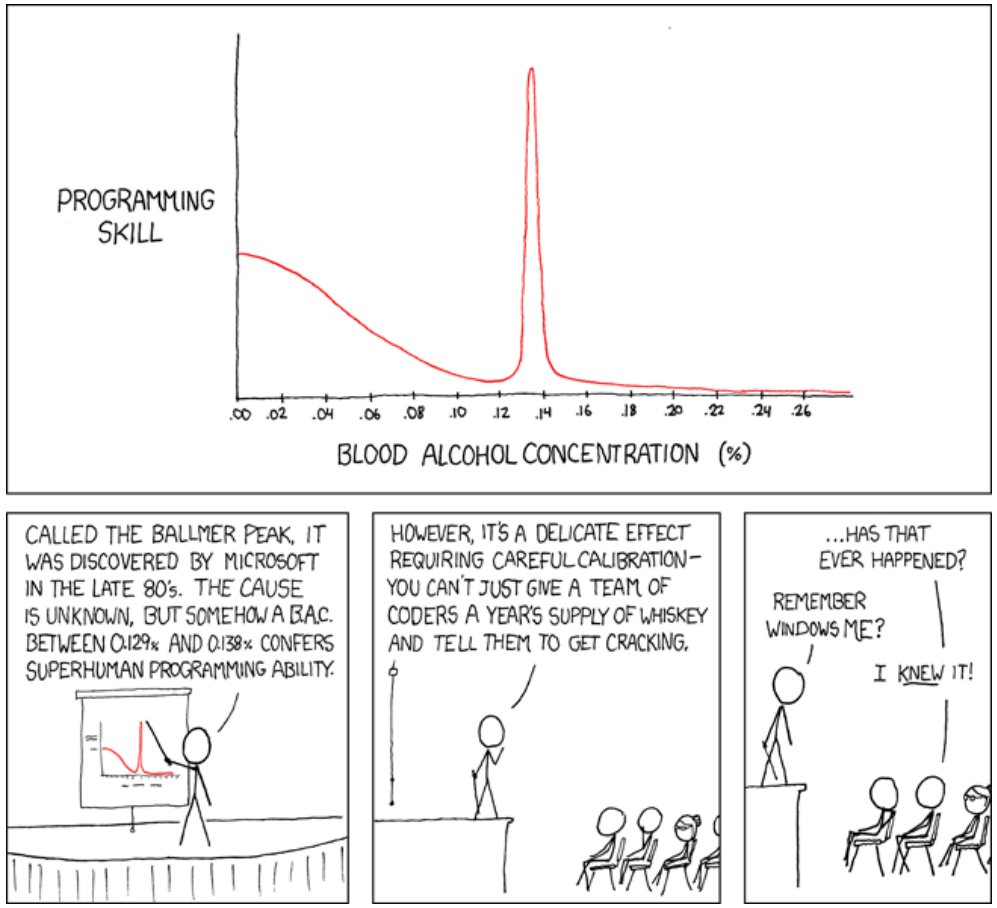
```
6
7       x = 5
8
```

```c
#include <stdio.h>

int main(){

    int x = 10;

    return 0;
}
```

Contenidos

Contenidos

9. Otros sistemas complementarios



PICO DE BALMER

PODRÍA SER ALGO MÁS QUE UNA BROMA GRACIOSA: https://arxiv.org/abs/2404.10002

9. Otros sistemas complementarios

¿Has probado a contárselo al pato?
Como si hablases con un bobo

Contenidos

Contenidos

10. Conclusiones

- La depuración es esencial para el desarrollo eficiente y confiable.
- Conocer las herramientas de depuración acelera el proceso.
- Comprender los errores comunes y sus patrones ayuda a prevenirlos.
- El enfoque estructurado en la localización de errores mejora la calidad del código.
- La documentación de los errores y sus soluciones contribuye al aprendizaje y a la colaboración.
- La prevención es tan importante como la detección.

Contenidos

Contenidos

11. Referencias

- Documentación Oficial de Python
- Logging en Python
- PDB (Python Debugger)
- Visual Studio Code y extensiones de Python
- Pruebas Unitarias con `unittest`
- Tipado Estático con `mypy`
- Recursos sobre CI/CD y GitHub Actions

6. Localización, depuración y resolución de problemas

Introducción a Python