ESPE UNIVERSIDAD DE LAS FUERZAS ARMADAS INNOVACIÓN PARA LA EXCELENCIA	INFORME DE PRÁCTICA DE LABORATORIO

DEPARTAMENTO:	**Ciencias de la computación**	**CARRERA:**	**Ingeniería de Software**		
ASIGNATURA:	Pruebas de Software	**NIVEL:**	6to	**FECHA:**	10/05/2025
DOCENTE:	Luis Alberto Castillo Salinas	**PRÁCTICA N°:**	1	**CALIFICACIÓN:**	

Evaluación de software mediante SonarQube

Jairo Smith Bonilla Hidalgo

RESUMEN

Durante el laboratorio, se llevó a cabo la instalación y configuración de SonarQube mediante un contenedor Docker, junto con la integración del escáner SonarScanner en el sistema. Se analizaron distintos proyectos desarrollados en lenguajes como Java, Python, JavaScript (React) y C# (.NET), con el objetivo de identificar problemas de calidad en el código fuente. Se configuraron correctamente los archivos sonar-project.properties en cada caso, ajustando rutas y parámetros según el tipo de tecnología. Posteriormente, se realizó un proceso de refactorización para corregir los errores más críticos y vulnerabilidades señaladas por la herramienta. Finalmente, se volvieron a ejecutar los análisis con SonarScanner, comprobando una mejora en las métricas de calidad. Este proceso permitió la integración de herramientas de calidad, en el ciclo de desarrollo de software.

Palabras Claves: SonarQube, calidad de código, refactorización

1. INTRODUCCIÓN:

La calidad del software es un aspecto fundamental en el desarrollo de aplicaciones, ya que influye directamente en su funcionalidad, seguridad, mantenimiento y rendimiento. En este contexto, el presente laboratorio se enfoca en el uso de la herramienta SonarQube como medio para familiarizarse con los procesos de evaluación de calidad en proyectos de software. La práctica se desarrolla sobre proyectos escritos en distintos lenguajes de programación, tales como Java, Python, JavaScript (React) y C# con .NET, lo que permite un enfoque integral. A partir del análisis realizado con SonarQube, se analizará sobre conceptos clave como la mantenibilidad, eficiencia, fiabilidad y buenas prácticas de codificación, con el fin de establecer las bases para una programación más profesional y alineada con los estándares de calidad.

2. OBJETIVO(S):

2.1 El objetivo principal de la práctica es realizar un análisis de la calidad del software utilizando SonarQube, evaluando proyectos desarrollados en diferentes lenguajes de programación, como Java, Python, React y C# con .NET. Este análisis permite identificar aspectos clave como vulnerabilidades de seguridad, código duplicado, complejidad, eficiencia y otros factores que afectan la fiabilidad y mantenibilidad del código.

2.2 Se busca aplicar las mejores prácticas de programación a través de las recomendaciones proporcionadas por SonarQube, refactorizando el código según los estándares de calidad explicados en clase. Con ello, se pretende mejorar la seguridad, fiabilidad y eficiencia de los proyectos evaluados.

2.3 Además, se tiene como objetivo aplicar y comprender los conceptos de calidad de software, tales como funcionalidad, fiabilidad, eficiencia, mantenibilidad, seguridad y usabilidad, observando cómo cada uno de estos aspectos impacta en la calidad general del software a lo largo del proceso de análisis y refactorización.

3. MARCO TEÓRICO:

El análisis de calidad del software es un proceso fundamental dentro del ciclo de desarrollo, ya que permite detectar errores, vulnerabilidades, malas prácticas y otros aspectos que afectan la mantenibilidad y seguridad del código. Una

de las herramientas más reconocidas para este propósito es SonarQube, una plataforma de análisis estático de código que evalúa automáticamente diferentes métricas de calidad.

SonarQube permite realizar evaluaciones sobre aspectos como errores, vulnerabilidades, código duplicado, secciones sin pruebas, puntos críticos de seguridad y problemas de mantenibilidad. Soporta múltiples lenguajes de programación, entre ellos Java, Python, JavaScript y C#.

La ejecución de los análisis se realiza a través de SonarScanner, un componente que envía los resultados del escaneo al servidor de SonarQube. Cada proyecto debe contar con un archivo de configuración llamado sonar-project.properties, el cual define parámetros esenciales como el nombre del proyecto, su clave, el origen del código fuente y la ubicación de los binarios en el caso de lenguajes compilados como Java o .NET. Integrar herramientas como SonarQube en el proceso de desarrollo contribuye a mantener un código más limpio, confiable y fácil de mantener, lo que reduce costos a largo plazo y mejora la calidad del producto final.

4. DESCRIPCIÓN DEL PROCEDIMIENTO:

Para llevar a cabo el análisis del código mediante SonarQube, se siguieron los siguientes pasos:

1. Instalación de SonarQube:

- Se descargó e instaló SonarQube en un contenedor Docker, configurando el puerto de acceso a la interfaz web en el puerto 9000.

- También se configuró SonarScanner, herramienta que se utiliza para realizar el análisis del código y enviar los resultados al servidor SonarQube.

2. Configuración de los Proyectos:

- Se creó un archivo de configuración sonar-project.properties en el directorio raíz de cada proyecto. Este archivo contiene información como el nombre del proyecto, la clave, la versión y las rutas del código fuente y los binarios, dependiendo del tipo de lenguaje utilizado.

- En el caso de proyectos en Java, se configuraron las rutas de los archivos .java y .class, mientras que, para proyectos en lenguajes como JavaScript, Python y C#, se especificaron las rutas adecuadas para cada uno.

3. Generación de Tokens:

- Se generó un token de autenticación en la interfaz de SonarQube, el cual se incluyó en el archivo de configuración para permitir la comunicación entre el SonarScanner y el servidor de SonarQube.

4. Ejecución del Análisis:

- Se ejecutó el comando sonar-scanner desde la terminal para iniciar el análisis del código. Este comando escaneó los proyectos, detectando errores, vulnerabilidades, duplicaciones, y otras métricas clave relacionadas con la calidad del código.

5. Revisión de Resultados:

- Una vez completado el análisis, se accedió a la interfaz web de SonarQube para revisar los informes generados. Estos informes presentaron las métricas de fiabilidad, seguridad, mantenibilidad, entre otras, permitiendo identificar las áreas que necesitaban refactorización o mayor atención.

6. Refactorización del Código:

- A partir de los resultados obtenidos, se procedió a refactorizar el código para corregir los problemas más críticos, como vulnerabilidades de seguridad, bugs en la lógica y código duplicado.

7. Reevaluación del Proyecto:

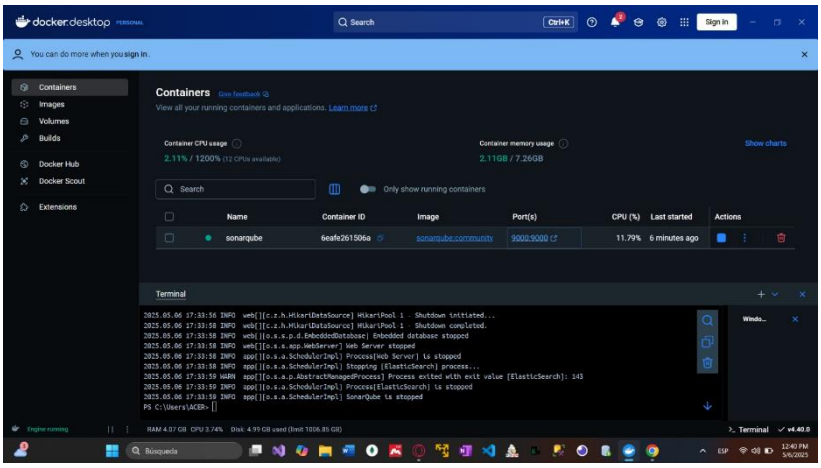
- Posteriormente, se volvió a ejecutar el análisis para verificar que las correcciones habían reducido la cantidad de problemas detectados, mejorando la calidad del código.

5. ANÁLISIS DE RESULTADOS:

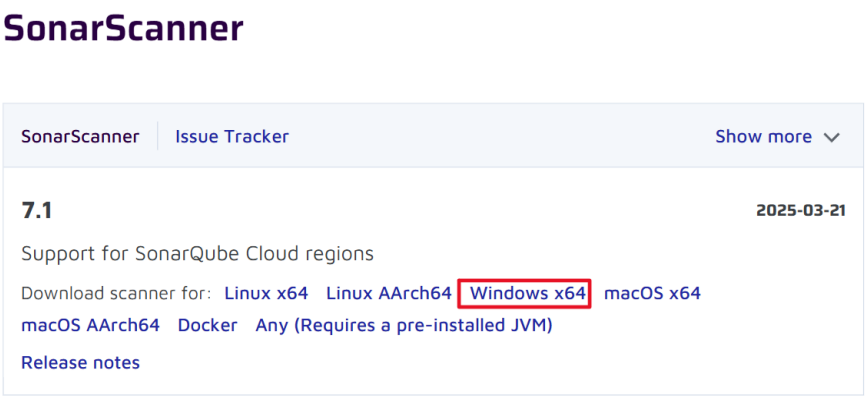
Proyecto	Métrica	Antes de la Refactorización	Después de la Refactorización
C# - alquiler-vehiculos	**Security**	E (1)	A (0)
	Reliability	C (24)	C (24)
	Maintainability	A (24)	A (24)
	Hotspots Reviewed	0.0%	0.0%
	Coverage	0.0%	0.0%
	Duplications	0.0%	0.0%
React - consumo-api-react	**Security**	A (0)	A (0)
	Reliability	C (77)	B (16)
	Maintainability	A (116)	A (25)
	Hotspots Reviewed	0.0%	A (—)
	Coverage	0.0%	0.0%
	Duplications	13.9%	7.8%
Python - project-python	**Security**	E (1)	A (0)
	Reliability	D (26)	D (26)
	Maintainability	E (246)	E (246)
	Hotspots Reviewed	0.0%	0.0%
	Coverage	0.0%	0.0%
	Duplications	0.0%	0.0%
Java - sistema-ventas	**Security**	A (0)	A (0)
	Reliability	C (2)	C (2)
	Maintainability	A (488)	A (488)
	Hotspots Reviewed	A (—)	A (—)
	Coverage	0.0%	0.0%
	Duplications	12.7%	12.7%

6. GRÁFICOS O FOTOGRAFÍAS:

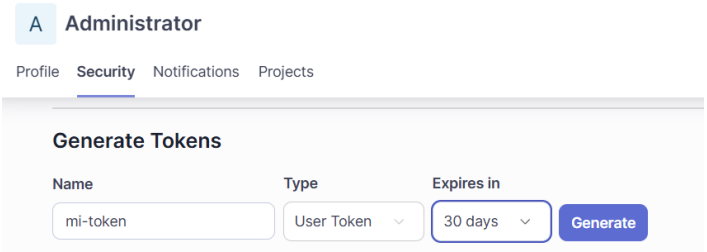
- Se descargó e instaló SonarQube en un contenedor Docker en el puerto 9000 con el comando: docker run --name sonarqube -p 9000:9000 sonarqube:community



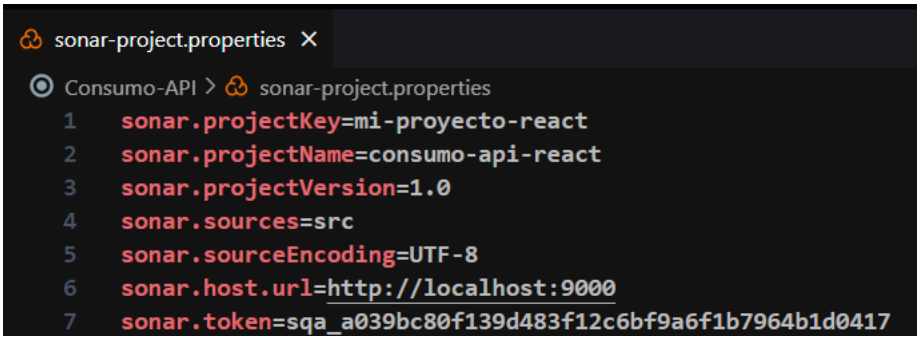
- Descarga y configuración de SonarScanner



- Generación del token en la interfaz web de SonarQube.



- archivo de configuración sonar-project.properties en el directorio raíz de cada proyecto

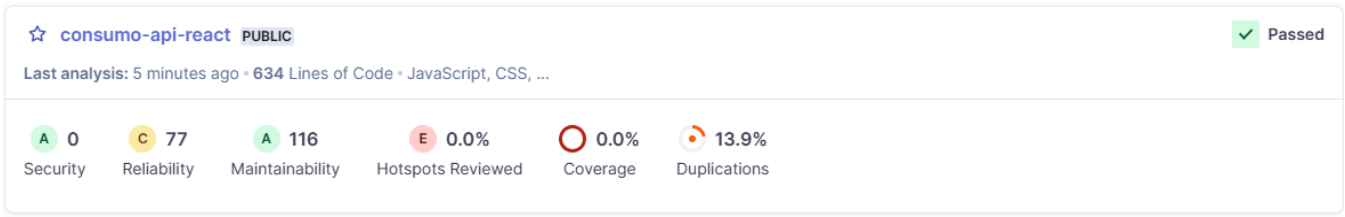


- Ubicación del archivo en la raíz del código fuente de cada proyecto y la ejecución del análisis con sonar-scanner.

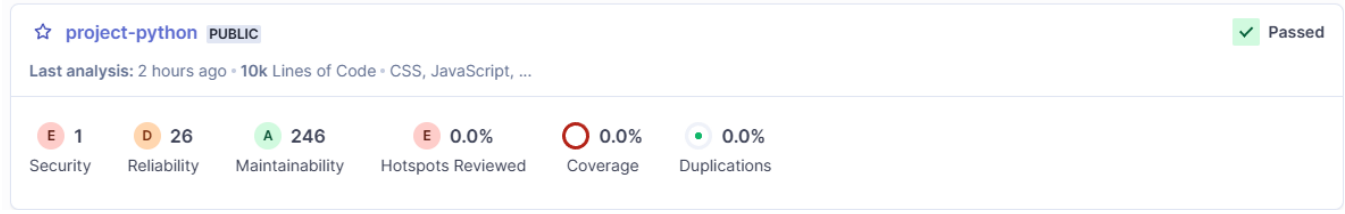
```
PS C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\Pruebas\
proyecto_pruebas\Consumo-API> sonar-scanner
```

- **Revisión de Resultados antes de la refactorización:**

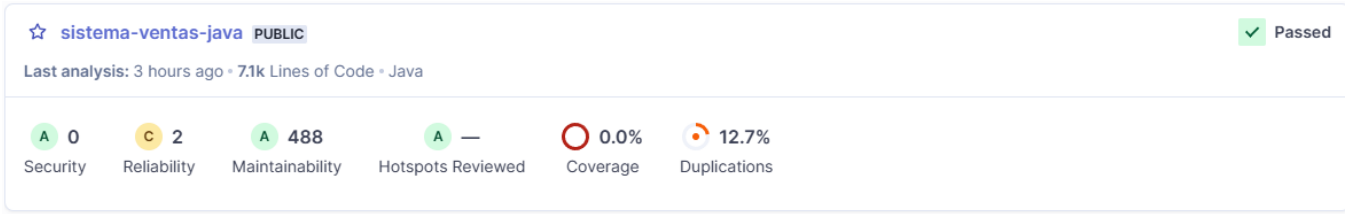
React:



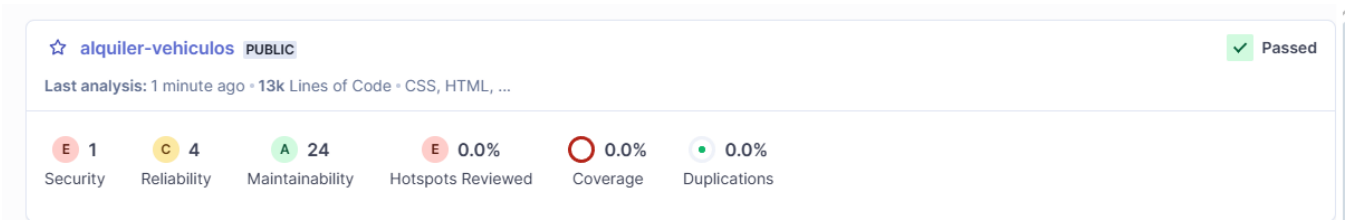
Python:



Java:

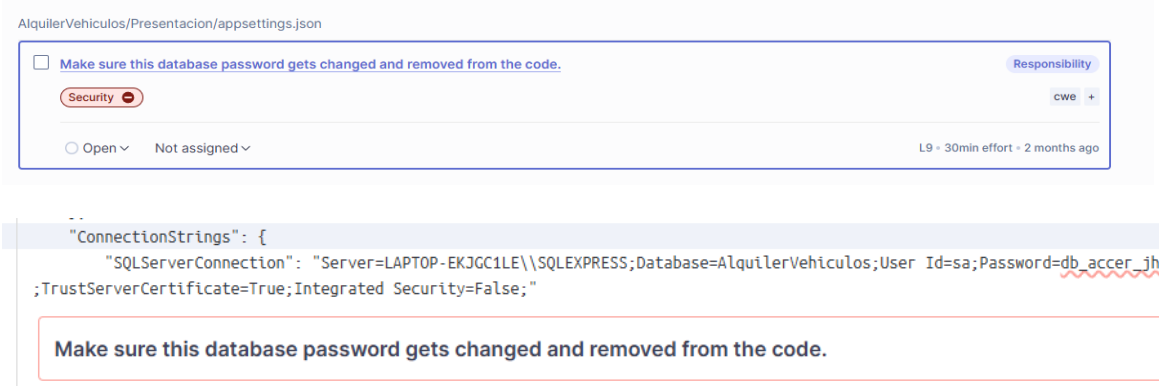


C# con .NET

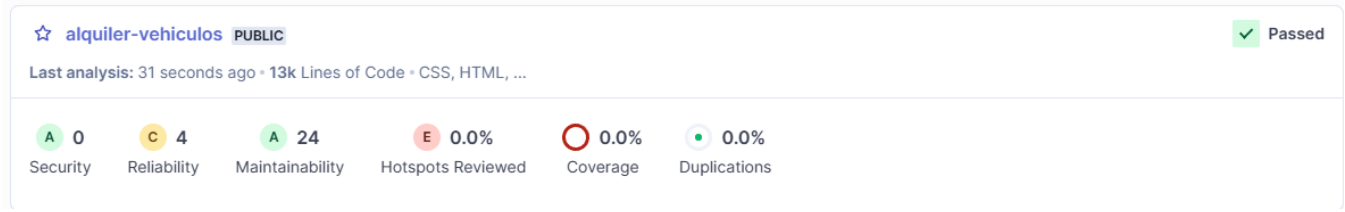


- **Refactorización del Código y Reevaluación del Proyecto:**
- C# con .NET:

El principal problema principal en el proyecto C# con .NET es de seguridad, ya que la contraseña de la base de datos está escrita directamente en el código. SonarQube recomienda extraerla y almacenarla de forma segura.

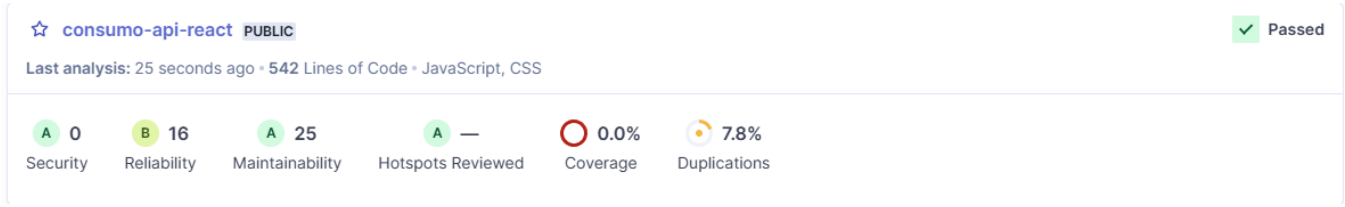


Siguiendo estas recomendaciones, entonces el resultado en c# con .NET después de la refactorización fue:



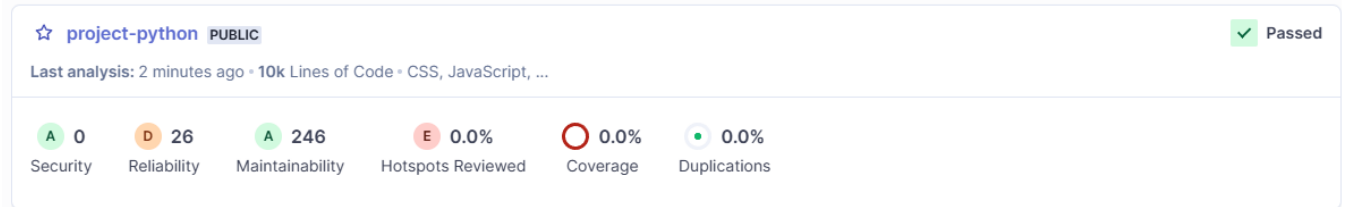
- **React:**

En el proyecto de React que consume la API de TMDB, se observó una mejora significativa en la fiabilidad, pasando de una calificación C a B, como resultado de aplicar las recomendaciones por SonarQube. Además, el porcentaje de líneas duplicadas disminuyó de 13.9 % a 7.8 %**,** y el número total de líneas de código se redujo de 634 a 542. Estas mejoras reflejan un código más limpio, mantenible y con menor redundancia estructural.



- **Python:**

El principal problema principal detectado en el proyecto de Python fue de seguridad, debido a que la clave secreta de Django utilizada en producción estaba expuesta directamente en el código. Esto representa una vulnerabilidad crítica, ya que dicha clave debería mantenerse confidencial y gestionarse de forma segura fuera del repositorio. Como solución, se procedió a revocar la clave del código fuente, tal como recomienda SonarQube, entonces se obtuvo el siguiente resultado:



7. DISCUSIÓN:

A lo largo del laboratorio, se aplicaron los conceptos vistos en clase sobre los componentes de la calidad del software, tales como la fiabilidad, eficiencia, mantenibilidad, seguridad y usabilidad. Mediante el uso de SonarQube se realizó un análisis estático del código en distintos lenguajes de programación, lo cual permitió identificar vulnerabilidades, código duplicado, malas prácticas y elementos que afectaban el rendimiento y la mantenibilidad. Tras realizar las refactorizaciones sugeridas por la herramienta, se observaron mejoras concretas en indicadores clave como la seguridad y la fiabilidad.

Por ejemplo, en el proyecto de C# con .NET, se eliminó una mala práctica relacionada con el almacenamiento inseguro de contraseñas en el código, mejorando así la seguridad. En el proyecto de React, se logró reducir el porcentaje de duplicación de código de un 13.9 % a un 7.8 %, lo que favorece la eficiencia y la mantenibilidad.
En el caso del proyecto en Python, se identificó y corrigió la exposición de la clave secreta de Django, una vulnerabilidad crítica desde el punto de vista de la seguridad. Gracias a estas mejoras, se pudo comprobar de forma práctica cómo el análisis automático y la aplicación de buenas prácticas realmente contribuyen a mejorar la calidad del software

8. CONCLUSIONES:

- A lo largo del laboratorio, se logró realizar un análisis de la calidad del software en proyectos desarrollados en Java, Python, React y C# con .NET, utilizando SonarQube. Esto permitió identificar vulnerabilidades de seguridad, código duplicado, complejidad innecesaria, y otros factores que afectan la fiabilidad y mantenibilidad del código. Al aplicar las recomendaciones proporcionadas por la herramienta, se observó una mejora en la calidad general del software, contribuyendo al cumplimiento de los estándares de calidad aprendidos en clase.

- La práctica permitió aplicar y reforzar los conceptos clave sobre los componentes de la calidad del software, como funcionalidad, fiabilidad, eficiencia, mantenibilidad, seguridad y usabilidad. A través de la refactorización de los proyectos según las sugerencias de SonarQube, se mejoraron aspectos críticos como la seguridad y la fiabilidad, lo que demuestra la importancia de las buenas prácticas de programación en el desarrollo de software. Además, se evidenció cómo el análisis automatizado ayuda a mejorar la calidad del código, asegurando un mejor rendimiento y una mayor facilidad de mantenimiento a largo plazo.

9. BIBLIOGRAFÍA:

SonarScanner. (s/f). Sonarsource.com. Recuperado el 11 de mayo de 2025, de https://docs.sonarsource.com/sonarqube-server/9.6/analyzing-source-code/scanners/sonarscanner/

10. ANEXOS

- https://github.com/JairoBonilla2004/Proyecto-React
- https://github.com/JairoBonilla2004/Sistema_Ventas_Java/
- https://github.com/JairoBonilla2004/Alquiler_Vehiculos
- https://github.com/JairoBonilla2004/Proyecto-con-python