

Grado en Ingeniería del Software  
Doble Grado en Matemática Computacional e Ingeniería del Software  
Doble Grado en Física Computacional e Ingeniería de Software

# Verificación de Software



## Práctica Análisis de Software con Sonarqube

Alonso Álvarez García  
Rafael Socas Gutiérrez

Curso 2023/24



## Datos de los alumnos

#	Nombre y apellidos	Curso
1	Pedro A. Morales Nieto	4B
2		
3		
4		
5		

## Instrucciones

- Completa la práctica en este mismo Power Point rellenando las páginas en blanco o incluyendo más páginas si necesitas más espacio para los pantallazos y las explicaciones.
- Una vez completado el Power Point, guárdalo en formato pdf. **A la plataforma BB sube el pdf resultante.**
- Sube a BB también el fichero **main\_final.py** con el código ya depurado y asegurándote que funciona correctamente la aplicación.
- Rellene el nombre/apellidos y el curso de los participantes del grupo.
- **IMPORTANTE:** Recordad que en un contexto profesional importa mucho la forma, además del contenido. No se trata únicamente de hacer bien el trabajo, hay que saber transmitirlo adecuadamente. Es decir, cuidad la presentación de resultados. Además, siempre que sea posible, haremos una miniexposición en clase. **Esta parte supone el 20% de la nota.**
- **Fecha máxima de entrega: lunes 19 febrero 23:59.**

2  
Puntos

## Contexto

### Conceptos / Análisis estático

#### ANÁLISIS ESTÁTICO

Es el proceso de evaluar un componente o sistema sin ejecutarlo, basándose en su forma, estructura, contenido o documentación.

#### ANÁLISIS ESTÁTICO DE CÓDIGO

Es el análisis del código fuente, llevado a cabo sin ejecutar el software.

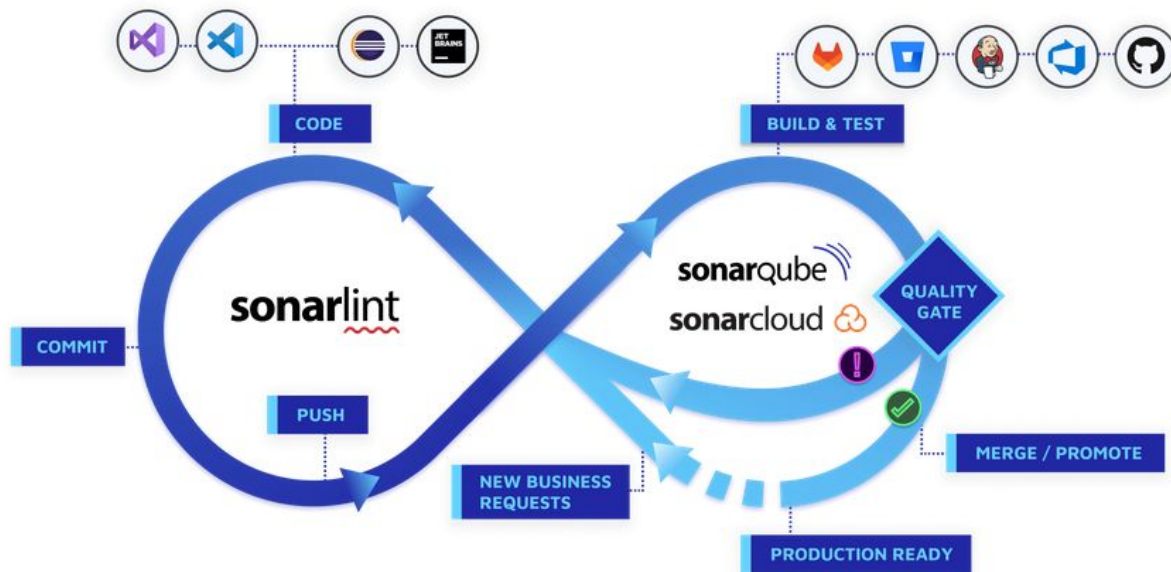
#### ¿POR QUÉ ANÁLISIS ESTÁTICO?

- Se hace en una etapa temprana del proceso de desarrollo → defectos más baratos de corregir.
- Permite una homogeneidad en el código escrito por distintos desarrolladores para el mismo proyecto.
- Facilita seguir buenas prácticas de calidad en el código.
- Permite detectar patrones incorrectos, duplicaciones, vulnerabilidades... sin ejecutar el código.

## Contexto

Sonarqube: Herramienta *open source* para realizar análisis estático de código. Es de las más utilizadas a nivel mundial

### Developing with Sonar



supports dozens of popular languages, development frameworks and IaC platforms



<https://docs.sonarsource.com/sonarqube/latest/user-guide/concepts/>

## Contexto

### Resultados / Issues



**BUG:** Un error que romperá tu código y necesita ser arreglado inmediatamente.



**VULNERABILITY:** parte del código que hace que el software sea vulnerable a ataques y que debe ser arreglado inmediatamente.



**SECURITY HOTSPOT:** parte del código vulnerable a ataques pero que, al no comprometer la seguridad de todo el software, no necesita un arreglo inmediato.



**CODE SMELL:** no es un error, pero hace que el código sea confuso y difícil de mantener.



## Contexto

### Resultados / Coverage and Duplications



0.0%

Coverage on 334 Lines to cover

-

Unit Tests



2.0%

Duplications on 1.2k Lines

2

Duplicated Blocks

**COBERTURA:** parte del código que no está “cubierta” (es decir, que no está siendo probada) por test unitarios. Sonarqube no detecta esto, pero es capaz de leer los reportes de herramientas que sí lo hacen, como Jacoco en Java o Istanbul en Javascript.

**DUPLICACIONES:** número de bloques de código o archivos duplicados. Tener duplicaciones dificulta la mantenibilidad y lectura del código.

## Objetivos

- Ejecutar un análisis estático de código para detectar *bugs*, vulnerabilidades, *code smell*, etc.
- Analizaremos un código Python, aunque las ideas y métodos desarrollados aplican a cualquier lenguaje.
- Nos apoyaremos en la archiconocida herramienta de análisis estático [sonarqube](#).
- Depurar un código Python para hacerlo más robusto y en nuestro caso que funcione.





## Tarea 1: preparación del entorno (1/3)

- Nos apoyaremos en el entorno de desarrollo (IDE) Visual Studio Code y en el intérprete Python 3. Para instalarlo se seguirá el siguiente tutorial <https://code.visualstudio.com/docs/python/python-tutorial>.
- Crear una carpeta con donde se guardarán el código a testear. El fichero inicial a incluir en esa carpeta se aporta junto con este enunciado.
- En el IDE, abrir la carpeta y crear un entorno virtual (Ctrl+Shift+P) de tipo Venv (ver tutorial anterior).
- Instalar los siguientes paquetes en Python para que pueda funcionar la aplicación bajo análisis una vez depurada  
    > `pip install matplotlib numpy tk`

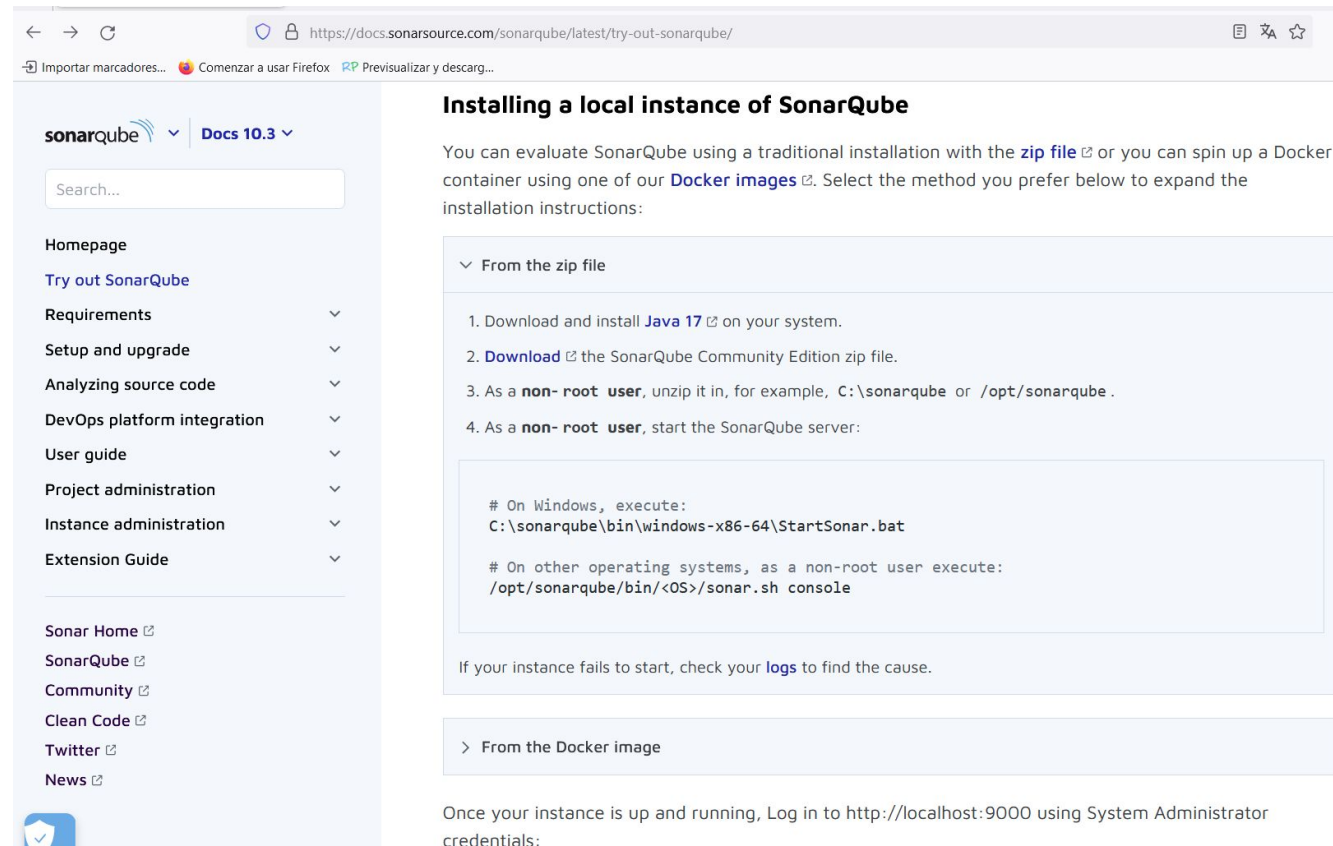
## Tarea 1: preparación del entorno (2/3)

Instalar el entorno **Sonarqube** siguiendo el siguiente tutorial

Importante:

- Instalar **Java 17**
- Reiniciar el portátil tras la instalación
- Incluir en la variable de entorno de Windows PATH la ruta de los /bin de sonarqube. P.e ejemplo si se ha descargado en C:\ la ruta es **C:\sonarqube\bin**

<https://docs.sonarsource.com/sonarqube/latest/try-out-sonarqube/>



The screenshot shows the SonarQube documentation page for installing a local instance. The page title is "Installing a local instance of SonarQube". It provides instructions for evaluating SonarQube using a traditional installation with a zip file or a Docker container. The "From the zip file" section lists four steps: 1. Download and install Java 17 on your system. 2. Download the SonarQube Community Edition zip file. 3. As a non-root user, unzip it in, for example, C:\sonarqube or /opt/sonarqube. 4. As a non-root user, start the SonarQube server. It includes a code block for the commands to run on Windows and other operating systems. The "From the Docker image" section is partially visible at the bottom.

**Installing a local instance of SonarQube**

You can evaluate SonarQube using a traditional installation with the [zip file](#) or you can spin up a Docker container using one of our [Docker images](#). Select the method you prefer below to expand the installation instructions:

**From the zip file**

1. Download and install [Java 17](#) on your system.
2. [Download](#) the SonarQube Community Edition zip file.
3. As a **non-root user**, unzip it in, for example, C:\sonarqube or /opt/sonarqube.
4. As a **non-root user**, start the SonarQube server:

```
# On Windows, execute:
C:\sonarqube\bin\windows-x86-64\StartSonar.bat

# On other operating systems, as a non-root user execute:
/opt/sonarqube/bin/<OS>/sonar.sh console
```

If your instance fails to start, check your [logs](#) to find the cause.

**From the Docker image**

Once your instance is up and running, Log in to <http://localhost:9000> using System Administrator credentials:

## Tarea 1: preparación del entorno (3/3)

1  
Punto

Instalar el entorno **Sonarqube** siguiendo el siguiente tutorial

<https://docs.sonarsource.com/sonarqube/10.3/analyzing-source-code/scanners/sonarscanner/>

10.3 | Analyzing source code | Scanners | SonarScanner

### SonarScanner CLI

By <a href="#">SonarSource</a>	<a href="#">GNU LGPL 3</a>	<a href="#">Issue Tracker</a>	<a href="#">Show more</a>
<b>5.0.1</b>			2023-08-04
Bug fix to the JRE binaries for Linux			
<a href="#">Linux 64-bit</a>	<a href="#">Windows 64-bit</a>	<a href="#">Mac OS X 64-bit</a>	<a href="#">Docker</a> <a href="#">Any (Requires a pre-installed JVM)</a> <a href="#">Release notes</a>

The SonarScanner CLI is the scanner to use when there is no specific scanner for your build system.

Importante:

- Descargar sonarqube-scanner
- Incluir en la variable de entorno de Windows PATH la ruta de los /bin de sonarqube-scanner. P.e ejemplo si se ha descargado en C:\ la ruta es **C:\sonarqube-scanner\bin**

Finalmente, abrir el navegador WEB y <http://localhost:9000> para entrar en sonarqube

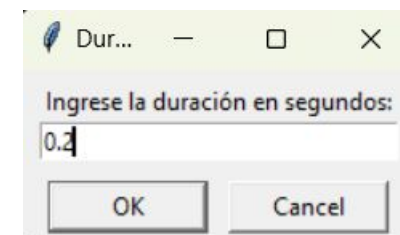
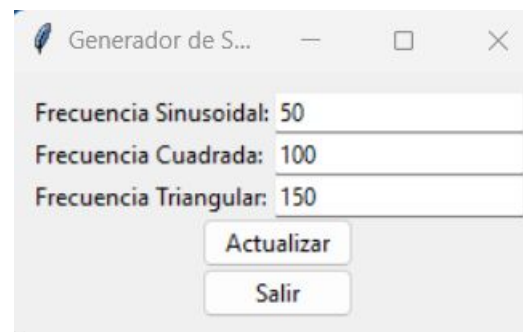
```
Windows PowerShell
INFO: Sensor HTML [web]
INFO: Sensor HTML [web] (done) | time=1ms
INFO: Sensor TextAndSecretsSensor [text]
INFO: 1 source file to be analyzed
INFO: 1/1 source file has been analyzed
INFO: Sensor TextAndSecretsSensor [text] (done) | time=290ms
INFO: Sensor VB.NET Project Type Information [vbnet]
INFO: Sensor VB.NET Project Type Information [vbnet] (done) | time=1ms
INFO: Sensor VB.NET Analysis Log [vbnet]
INFO: Sensor VB.NET Analysis Log [vbnet] (done) | time=50ms
INFO: Sensor VB.NET Properties [vbnet]
INFO: Sensor VB.NET Properties [vbnet] (done) | time=0ms
INFO: Sensor IaC Docker Sensor [iac]
INFO: 0 source files to be analyzed
INFO: 0/0 source files have been analyzed
INFO: Sensor IaC Docker Sensor [iac] (done) | time=77ms
INFO: ----- Run sensors on project
INFO: Sensor Analysis Warnings import [csharp]
INFO: Sensor Analysis Warnings import [csharp] (done) | time=0ms
INFO: Sensor Zero Coverage Sensor
INFO: Sensor Zero Coverage Sensor (done) | time=4ms
INFO: SCM Publisher SCM provider for this project is: git
INFO: SCM Publisher 1 source file to be analyzed
INFO: SCM Publisher 1/1 source file have been analyzed (done) | time=262ms
INFO: CPD Executor Calculating CPD for 1 file
INFO: CPD Executor CPD calculation finished (done) | time=4ms
INFO: Analysis report generated in 54ms, dir size=206.6 kB
INFO: Analysis report compressed in 12ms, zip size=29.9 kB
INFO: Analysis report uploaded in 28ms
INFO: ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=verificacion
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=1da3a5b7-70ff-4937-8025-d4c1243d1a47
INFO: Analysis total time: 7.079 s
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
INFO: Total time: 10.238s
INFO: Final Memory: 29M/108M
INFO: -----
PS C:\Universidad\Verficacion\3-Practicas\verificacion\P1-Sonarqube\CodigoP1>
```

## Código a analizar y depurar

```
main_inicial.py 9+ x
main_inicial.py > ...
1 import tkinter as tk
2 from tkinter import ttk
3 from tkinter import simpledialog
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
7
8 from flask import request, render_template_string
9
10 fm=4000 # frecuencia muestro 4000 Hz
11 fm=fm
12
13 tm=120 # tiempo de análisis
14 tm=tm
15 tm=2*tm
16
17
18 def generar_senial(frecuencia, tipo, duracion):
19     tiempo == np.linspace(0, duracion, int(1000 * duracion), endpoint=False)
20
21     if tipo == 'sinusoidal':
22         return tiempo, np.sin(2 * np.pi * frecuencia * tiempo)
```

Se entrega un código Python a depurar `main_inicial.py`

Se trata de una aplicación con entorno gráfico que solicita la frecuencia de una señal senoidal, cuadrada y triangular. Posteriormente el tiempo que se quiere representar. Una vez introducidos los datos, se presentan las tres señales temporales y su respectivas transformadas de Fourier.

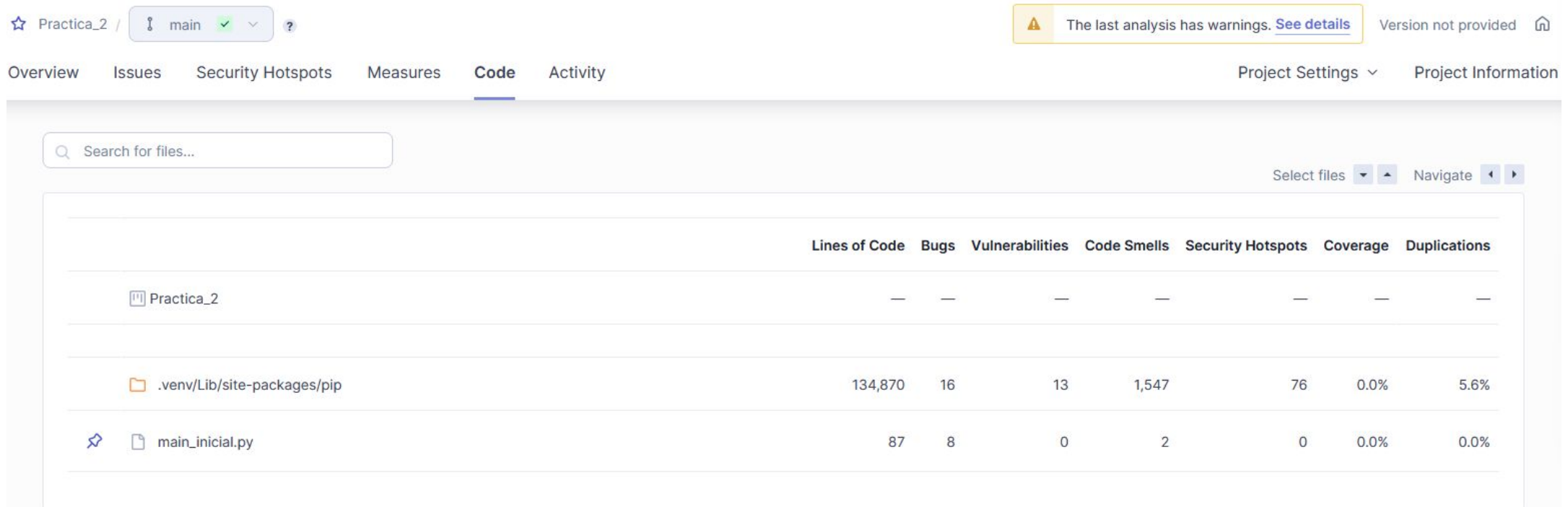


Este código, tras incluir los datos, **falla**. Además, **contiene bugs y code smells** que se tiene que depurar en esta práctica.

## Tarea 2: arranque el Sonarqube, haga el primer análisis y muestre los resultados

1  
Punto

Debería salir algo similar a la siguiente imagen. Aunque lo ideal es analizar todos los errores, en esta práctica **SOLO** debe centrarse en fichero `main_inial.py` que tiene 8 bugs y 2 code smells



☆ Practica\_2 / ⓘ main ✓ ?

The last analysis has warnings. [See details](#) Version not provided

Overview Issues Security Hotspots Measures **Code** Activity Project Settings Project Information

Search for files...

Select files Navigate

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
Practica_2	—	—	—	—	—	—	—
.venv/Lib/site-packages/pip	134,870	16	13	1,547	76	0.0%	5.6%
main_inial.py	87	8	0	2	0	0.0%	0.0%



← → ↺ ⓘ localhost:9000/code?id=verificacion ☆ 📁 ⬇ 🗑 🌑

sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More 🔍

🔍 ? A

☆ verificacion / ⓘ main ✓ ▾ ?

Overview Issues Security Hotspots Measures **Code** Activity

Project Settings ▾ Project Information

🔍 Search for files...

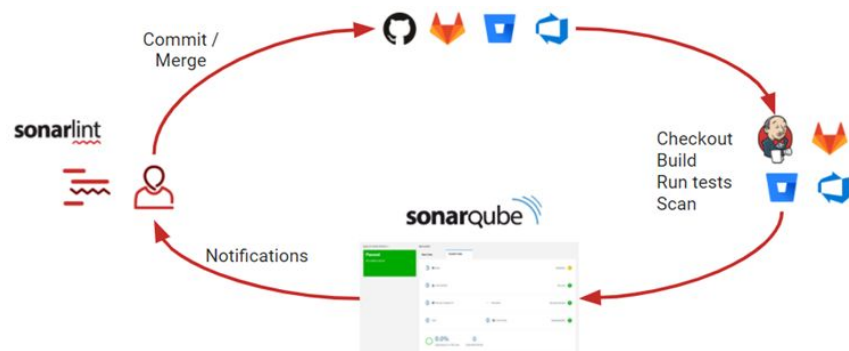
Select files ▾ ⬆ ⬇ Navigate ⬅ ➡

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
📁 verificacion	—	—	—	—	—	—	—
☆ 📄 main_inicial.py	87	8	0	2	0	0.0%	0.0%

## Tarea 3: fase de depuración

5  
Puntos

Genere un nuevo fichero `main_en_depuración.py` que sea igual que `main_inicial.py` y sobre este nuevo fichero (`main_en_depuración.py`) empiece a analizar los errores y a modificar el código Python para corregirlos. Itere las veces que sea necesario hasta que consiga eliminar los **8 bugs y 2 code smells** de `main_en_depuración.py`



Captura de pantalla de la interfaz de SonarQube mostrando los resultados de un análisis de código. La tabla de abajo muestra los datos de calidad para los archivos de código.

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
Practica_2	—	—	—	—	—	—	—
.venv\Lib\site-packages\pip	134,870	16	13	1,547	76	0.0%	5.6%
main_en_depuración.py	87	8	0	2	0	0.0%	91.1%
main_inicial.py	87	8	0	2	0	0.0%	91.1%

**8 bugs y 2 code smells**

Lo ideal es empezar por los **bugs** (son más críticos) y luego continuar por los **code smells**

# Práctica: Análisis de Software con Sonarqube



← → ↺ ⓘ localhost:9000/code?id=verificacion ☆ 📁 | 🖨️ 🔍

sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More 🔍

🔍 ? A

☆ verificacion / ⓘ main ✖️ ▼ ?

Overview Issues Security Hotspots Measures **Code** Activity Project Settings ▾ Project Information

Select files ▾ ▲ Navigate ◀ ▶

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
📁 verificacion	—	—	—	—	—	—	—
📄 main_en_depuracion.py	87	0	0	0	0	0.0%	45.2%
📄 main_inicial.py	87	8	0	2	0	0.0%	41.5%

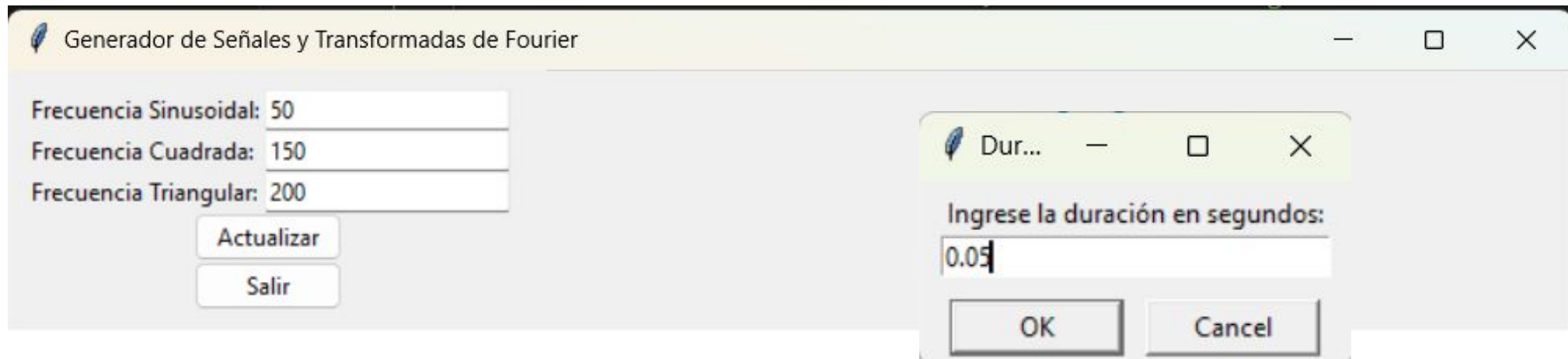
2 of 2 shown

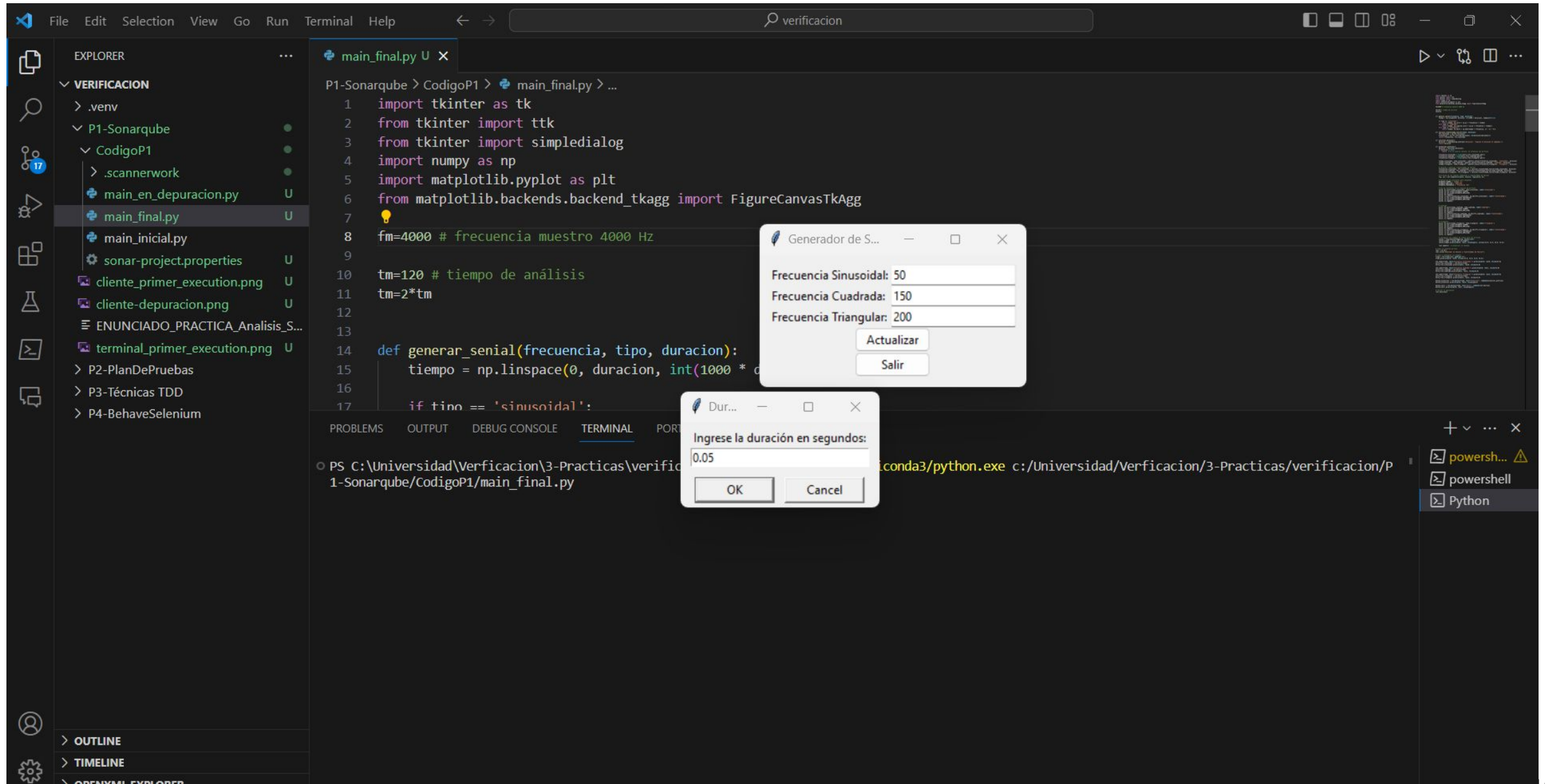
## Tarea 4: ejecución de la aplicación

1

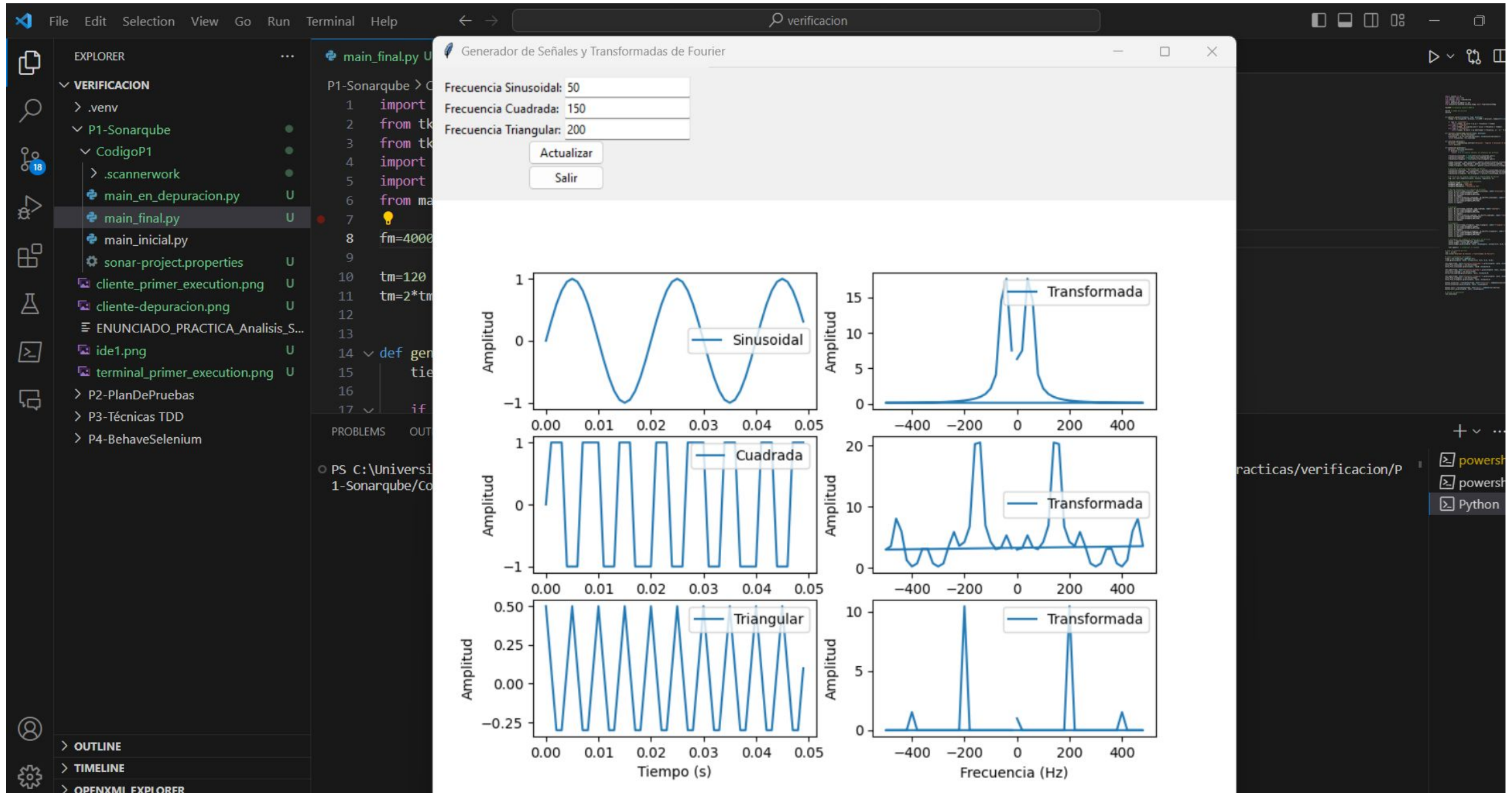
Punto

Una vez depurada la aplicación ([main\\_en\\_depuración.py](#)), guarde el fichero [main\\_en\\_depuración.py](#) como [main\\_final.py](#) y ejecútelo en el IDE de VS Code para validar su funcionamiento. Puede usar los siguientes parámetros (imágenes adjuntas). Presente evidencias de que la aplicación funciona correctamente.





The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays a project structure for 'VERIFICACION', including a sub-project 'P1-Sonarqube' with a file 'main\_final.py' marked with a 'U' (Unresolved) icon. The main editor window shows the code for 'main\_final.py', which includes imports for tkinter, numpy, and matplotlib, and a function 'generar\_senial'. Two dialog boxes are overlaid on the code: 'Generador de S...' (Signal Generator) with input fields for 'Frecuencia Sinusoidal: 50', 'Frecuencia Cuadrada: 150', and 'Frecuencia Triangular: 200', and buttons 'Actualizar' and 'Salir'; and 'Dur...' (Duration) with a text input 'Ingrese la duración en segundos:' containing '0.05' and buttons 'OK' and 'Cancel'. The bottom status bar shows the file path 'C:\Universidad\Verificacion\3-Practicas\verificacion\P1-Sonarqube\CodigoP1\main\_final.py'.







 Calle Playa de Liencres, 2 bis  
(entrada por calle Rozabella)  
Parque Europa Empresarial  
Edificio Madrid  
28290 Las Rozas, Madrid

 900 373 379  [info@u-tad.com](mailto:info@u-tad.com)

 [SOLICITA MÁS INFORMACIÓN](#)



CENTRO ADSCRITO A:



PROYECTO COFINANCIADO POR:

