Izmir,
Pescara
2026

Software de Simulador de Código Abierto (Open-Source) iVEBSIM+

LENGUAJE DE PROGRAMACIÓN

SOFTWARE DE SIMULADOR

COMPLEMENTO DE INTELIGENCIA ARTIFICIAL (ARTIFICIAL INTELLIGENCE - AI)
USO DEL SIMULADOR

EJERCICIOS PRÁCTICOS

KA220VET Erasmus+ Project

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

Funded by
the European Union

"Financiado por la Comisión Europea en el marco del Programa Erasmus+. El contenido aquí refleja las opiniones de los autores, y la Comisión Europea y la Agencia Nacional Turca no pueden ser consideradas responsables de estas opiniones."

Funded by the European Union

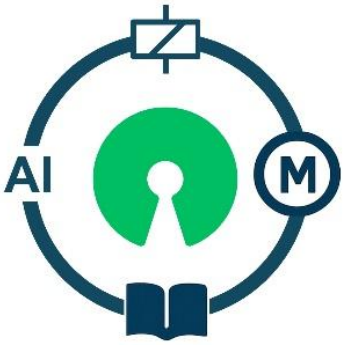
ÍNDICE (INDEX)

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by
the European Union

Funded by
the European Union

Funded by
the European Union



PRÓLOGO (FOREWORD)

Los sistemas educativos se están transformando debido al impacto de la transformación digital; especialmente en el ámbito de la formación profesional y técnica, la necesidad de entornos de aprendizaje innovadores, flexibles y accesibles aumenta cada día.

Este proyecto, titulado “Developing Vocational Education with an AI-Supported Open Source Simulator” (Desarrollo de la Formación Profesional con un Simulador de Código Abierto con soporte de Inteligencia Artificial - AI) y realizado dentro de las asociaciones de cooperación Erasmus+ KA220-VET, busca ofrecer una respuesta sostenible e innovadora a esta necesidad.

Este estudio multinacional, coordinado por Turquía con la colaboración de Polonia, España, Italia, Portugal y Rumanía, tiene como objetivo fortalecer la formación profesional mediante herramientas digitales y hacer que los procesos de enseñanza sean más interactivos y basados en la práctica.

El resultado principal del proyecto es el simulador iVEBSIM+, un software de código abierto (open-source) con soporte de Inteligencia Artificial (AI) diseñado para el campo de la electrónica industrial y el control. En este sentido, el proyecto no solo produce material educativo; también busca crear un ecosistema de aprendizaje digital que sea sostenible, mejorable y fácil de compartir.

Este libro ha sido diseñado como uno de los resultados del proyecto. En la primera parte, se presenta información teórica y práctica sobre Python, que es el lenguaje de programación básico utilizado para desarrollar el software del simulador. Temas como los fundamentos de la programación, las estructuras de variables y los mecanismos de decisión y bucles (loops) se explican de forma clara y orientada a la práctica para que los estudiantes de formación profesional puedan entenderlos fácilmente.

En las siguientes secciones, se tratan en detalle los siguientes temas:

- La base teórica del software del simulador.
- La creación de la interfaz gráfica de usuario (GUI - Graphical User Interface).
- La integración de componentes electrónicos industriales en el entorno digital.
- El desarrollo y uso del complemento de Inteligencia Artificial (AI).
- Ejercicios de aplicación preparados para su uso en la formación profesional.
- Procesos de aplicación piloto realizados con estudiantes.

El proceso del proyecto se organiza según la finalización de una etapa técnica específica en cada movilidad. Todas las fases, desde la creación de los fundamentos de Python hasta el desarrollo del núcleo del simulador, han sido realizadas mediante la cooperación internacional. Esto ha permitido mejorar la capacidad técnica y compartir las mejores prácticas entre los países participantes.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Una de las características más importantes de este estudio es que el simulador desarrollado es de **código abierto (open-source)**. Todos los códigos fuente del Simulador iVEBSIM+ se comparten con el público a través del sitio web del proyecto, permitiendo que educadores, estudiantes y desarrolladores puedan contribuir. Este enfoque se basa en la transparencia, la accesibilidad y la producción colectiva, apoyando la igualdad digital en la formación profesional. Además, explicar detalladamente las etapas de desarrollo del software motiva a los alumnos a convertirse en desarrolladores en lugar de ser solo usuarios.

El aprendizaje basado en simulación es fundamental en la formación profesional, ya que ofrece un área de experimentación segura, reduce costes y permite repetir las aplicaciones las veces que sea necesario. El soporte de la **Inteligencia Artificial (AI)** hace que el proceso de aprendizaje sea más eficiente a través de comentarios individualizados, análisis de errores y guía inteligente. Este libro y el software que lo acompaña integran estos dos enfoques para ayudar a adquirir las competencias digitales necesarias hoy en día.

Esperamos que este trabajo sirva de guía para profesores, estudiantes, desarrolladores e investigadores. Nuestro mayor deseo es que este estudio, nacido del conocimiento de una asociación internacional, se difunda siguiendo los principios de la ciencia abierta y los recursos educativos abiertos.

Agradecemos a todos los socios del proyecto y a los educadores por su esfuerzo.

İTO Vakfı Süleyman Taştekin MTAL

Zespol Szkol Nr1 im. prof. B. Krupinski ego

I. I. S. A. VOLTA

Liceul Tehnologic "Anghel Saligny"

Instituto Nossa Senhora Da Encarnacao Cooperativa De Ensino E Cultura CRL

Asocicion de profesores Tecnicos y de Ensenanza Secundar ia de Automocion de Galicia

"Financiado por la Comisión Europea en el marco del Programa Erasmus+. El contenido aquí refleja las opiniones de los autores, y la Comisión Europea y la Agencia Nacional Turca no pueden ser consideradas responsables de estas opiniones."

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

Funded by the European Union

LENGUAJE DE PROGRAMACIÓN PYTHON

1. LA IMPORTANCIA DEL LENGUAJE DE PROGRAMACIÓN PYTHON

Python es uno de los lenguajes de programación más utilizados en la actualidad. Fue desarrollado por Guido van Rossum y lanzado en 1991. Es un lenguaje de alto nivel con un amplio soporte de librerías donde puedes desarrollar aplicaciones en casi cualquier campo.

¿Por qué Python?

- Es fácil de aprender porque tiene una syntax (sintaxis) simple.
- Es similar al idioma inglés.
- Utiliza nuevas líneas para completar un comando en lugar de puntos y comas o paréntesis.
- Utiliza la indentation (sangría o espacio al principio de la línea) para definir el alcance de los loops (bucles), funciones y clases.
- Puede ejecutarse en todas las plataformas como Windows, Linux y Mac.

Áreas de Uso:

- Programación de sistemas.
- Programación de interfaces de usuario (GUI).
- Desarrollo web.
- Programación de redes.
- Desarrollo de videojuegos.
- Ciencia de datos, Machine Learning (aprendizaje automático).
- Análisis de datos.
- Inteligencia Artificial (AI).
- Scripting y herramientas.

- Figure 1. Librerías de Python

2. INSTALACIÓN DEL LENGUAJE DE PROGRAMACIÓN PYTHON

Para escribir código con el lenguaje de programación Python, primero debes instalar un editor o un entorno de desarrollo integrado (**IDE**). Sin embargo, antes de eso, el lenguaje Python debe estar instalado en tu sistema operativo.

2.1. Python instalación

Pasos para la instalación en Windows:

1. Visita el sitio oficial www.python.org.
2. En la sección "Downloads", selecciona la versión más reciente compatible con tu sistema operativo.
3. Una vez descargado el archivo .exe, ejecútalo para iniciar la instalación.
4. **IMPORTANTE:** En la primera pantalla, asegúrate de marcar la casilla **"Add Python to PATH"**. Esto es fundamental para ejecutar Python desde la línea de comandos.
5. Haz clic en "Install Now". Esto instalará automáticamente **pip** (el gestor de paquetes) y la documentación.

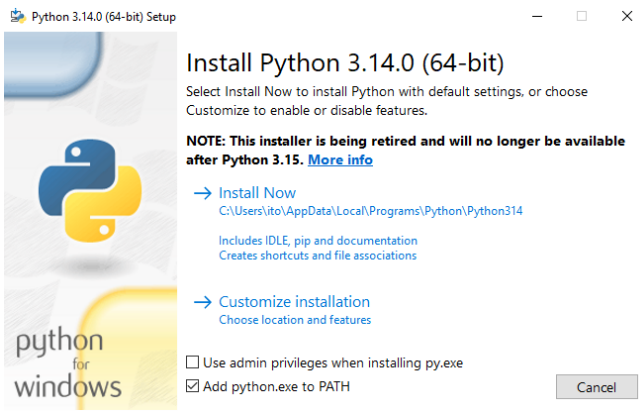


Figure 2. Instalación en Windows

Para macOS:

- Descarga el instalador con la extensión **.pkg** desde el mismo sitio web oficial.
- Sigue los pasos de instalación estándar del sistema.
- Python se instalará en la carpeta **Applications** (Aplicaciones) → **Python x.x**.

Para Linux:

- Abre la **terminal** y ejecuta los siguientes comandos en orden:
- sudo apt update
- sudo apt install python3 python3-pip

2.2. Verificación de la Instalación

Para comprobar si Python se ha instalado correctamente:

1. Abre el **Command Prompt** (Símbolo del sistema) o la **terminal**.
2. Escribe el comando python --version y pulsa Enter.
3. Si la instalación fue exitosa, verás la versión instalada (ej. Python 3.12.x).

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

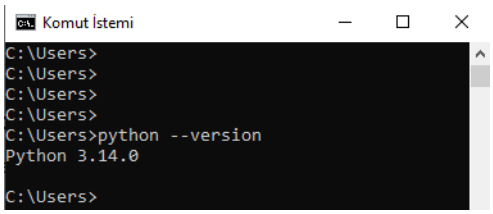


Figure 3. Versión del Instalador de Python

2.3. Verificación del Gestor de Paquetes (pip)

Puedes verificar el gestor de paquetes escribiendo el siguiente comando en el **Command Prompt** (CMD) o terminal:

pip –version

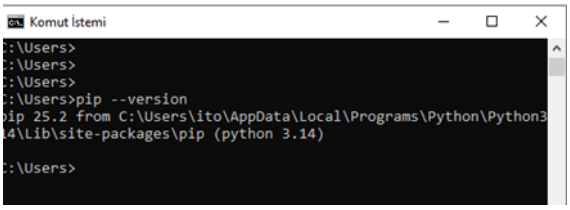


Figure 4. Versión del Instalador de Python.

Si ves el número de versión de Python como en la Figura 3, se ha instalado correctamente. Si no lo ves, refresca los pasos de instalación. **pip** (Python Package Installer) es una herramienta utilizada para la gestión de librerías y la instalación de paquetes en Python. Te permite instalar fácilmente las librerías que necesitas para tus proyectos.

2.4. Ejecución del primer programa Python

Python no requiere un editor para escribir código de comandos. El código puede ejecutarse directamente usando el Command Prompt (Símbolo del sistema). La Table 1 proporciona un ejemplo de un code snippet (fragmento de código) que imprime "Hello" en la pantalla. Sin embargo, a medida que el código del programa se vuelve más extenso y el uso de librerías aumenta, este método puede no ser el ideal para los programadores.

Una vez completada la instalación de Python, puedes ejecutar tu primer programa utilizando **el IDLE (Integrated Development and Learning Environment)** que viene con Python o el Command Prompt.

Ejecución con el Command Prompt:

1. Abre el Command Prompt (CMD).
2. Escribe python y presiona Enter. Esto iniciará el Python interactive shell (consola interactiva de Python).
3. Escribe el siguiente código y presiona Enter: print("Hello World")
4. Verás la salida "Hello World" en la pantalla.

Ejecución como un archivo de script:

1. Abre un editor de texto simple (como Notepad).
2. Escribe el siguiente código: print("Welcome to iVEBSIM+")
3. Guarda el archivo con el nombre hello.py.
4. Abre el Command Prompt, dirígete a la carpeta donde guardaste el archivo y escribe: python hello.py

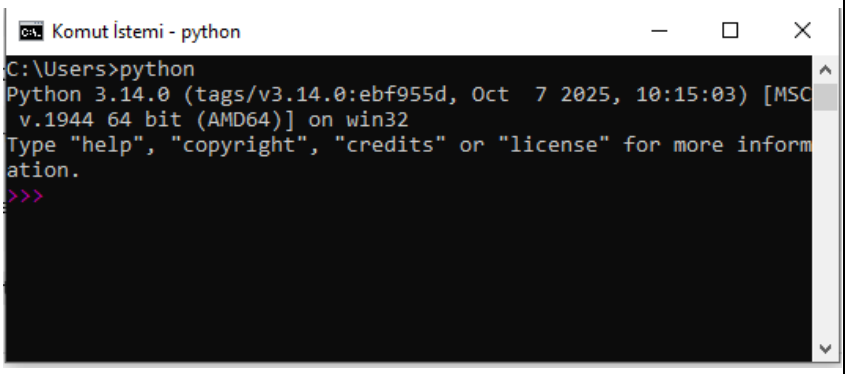
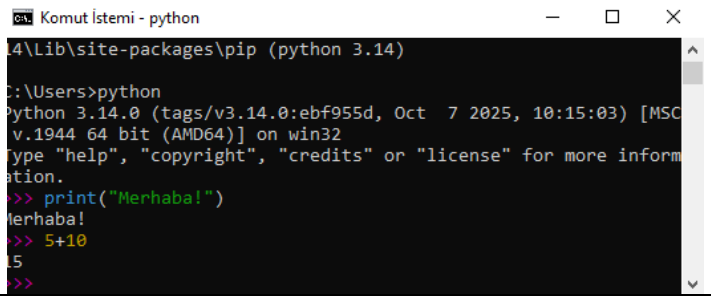
Códigos:	La salida de tu pantalla se verá así::
Pyhton	
Códigos:	La salida de tu pantalla se verá así:
print("Merhaba!") 5+10	

Table 1. Escritura de Programas Usando el Command Prompt

3. USO DE UN EDITOR

Puedes escribir código **Python** en un editor de texto simple y guardarlo con la extensión **.py**. Sin embargo, usar un **code editor** (editor de código) profesional hace que el proceso de desarrollo sea mucho más fácil y eficiente. En la **Figure 5**, se creó un nuevo archivo llamado **mycode.py**.

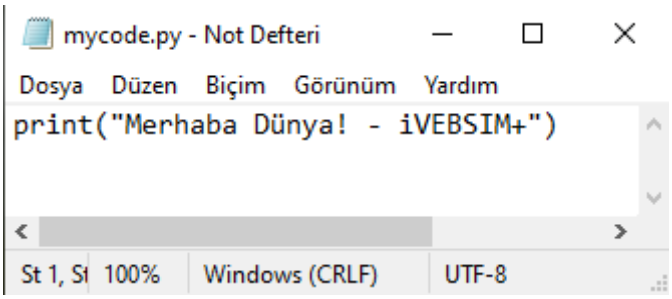


Figure 5. Uso del editor de texto

Cuando escribes el comando python mycode.py en el **command prompt** (símbolo del sistema), el programa se ejecutará y aparecerá en tu consola como se muestra a continuación.

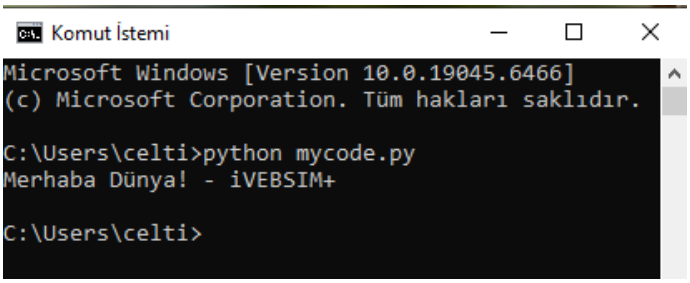


Figure 6. Salida del programa en consola

También podemos usar editores para ver nuestro código de forma más organizada y escribirlo más fácilmente. **Visual Studio Code**, **PyCharm** ve **Sublime Text** son algunos ejemplos.

3.1. Instalación de Visual Studio Code

Visual Studio Code (**VS Code**) es un **code editor** (editor de código) gratuito, potente y popular que funciona en **Windows**, **macOS** y **Linux**. Para instalarlo, sigue estos pasos:

1. Ve al sitio web oficial: https://code.visualstudio.com/.
2. Haz clic en el botón de descarga para tu sistema operativo.
3. Ejecuta el archivo de instalación descargado.
4. Sigue las instrucciones del asistente de instalación (**installation wizard**). Se recomienda seleccionar la opción "**Add to PATH**" durante la configuración.

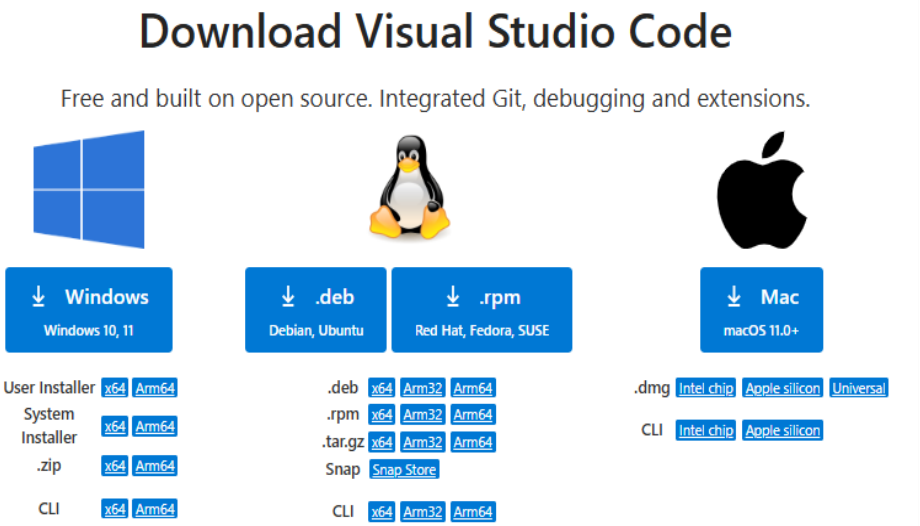


Figure 7. Página de descarga de VS Code

Una vez que la instalación se haya completado, puedes iniciar **Visual Studio Code** desde tu escritorio o el menú de inicio.

3.2. Instalación de la extensión de Python para Visual Studio Code

Para usar funciones de **Python** como **code completion** (completado de código), **debugging** (depuración de errores) y **unit testing** (pruebas unitarias) en **Visual Studio Code**, debes instalar la **extension** de Python desarrollada por **Microsoft**. Sigue estos pasos:

1. Haz clic en el icono de **Extensions** ( ) en la **Activity Bar** (barra de actividad) situada al costado de **VS Code**.
2. Escribe "**Python**" en la barra de búsqueda.
3. Busca la **extension** llamada "**Python**" proporcionada por **Microsoft** y haz clic en el botón **Install**.

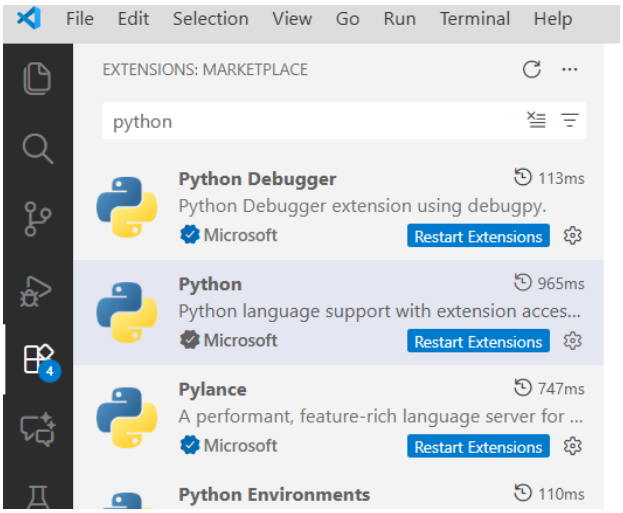


Figure 8. Instalación de Python en VS Code

3.3. Ejecución del primer código

Después de instalar la **extension**, puedes crear y ejecutar tu primer archivo de **Python**:

1. Selecciona **File > New File** (Archivo > Nuevo archivo) o presiona **Ctrl+N**.
2. Guarda el archivo con una extensión **.py** (por ejemplo, **main.py**).
3. Escribe el siguiente código en el archivo:

Python print("Hello World!")

4. Ejecuta el código haciendo clic en el botón **Run** (icono de reproducción/play icon) en la esquina superior derecha o presionando **F5**.

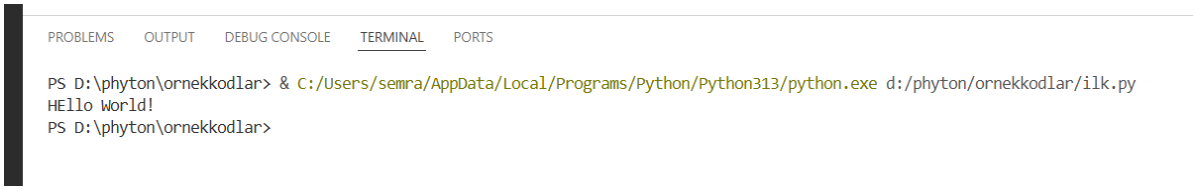


Figure 9. Salida de pantalla

3.4. Kit de herramientas de Inteligencia Artificial para Visual Studio Code

El **AI Toolkit** es una **extension** (extensión) que permite a los desarrolladores integrar y probar aplicaciones inteligentes utilizando modelos de **Artificial Intelligence (AI)** (Inteligencia Artificial). Proporciona acceso a varios modelos y simplifica el proceso de integración dentro del editor. **AI Toolkit** ofrece una integración fluida con modelos de **AI** populares como OpenAI, Anthropic, Google y GitHub.

3.4.1. Instalación

Para instalar el **AI Toolkit** en **Visual Studio Code**, sigue los pasos a continuación:

1. Abre el menú de **Extensions** en **VS Code**.
2. Escribe "**AI Toolkit**" en la barra de búsqueda.
3. Localiza la **extension** y haz clic en el botón **Install**.

Una vez que la instalación se haya completado, el icono de **AI Toolkit** aparecerá en la **Activity Bar** (barra de actividad) en el lado izquierdo del editor.

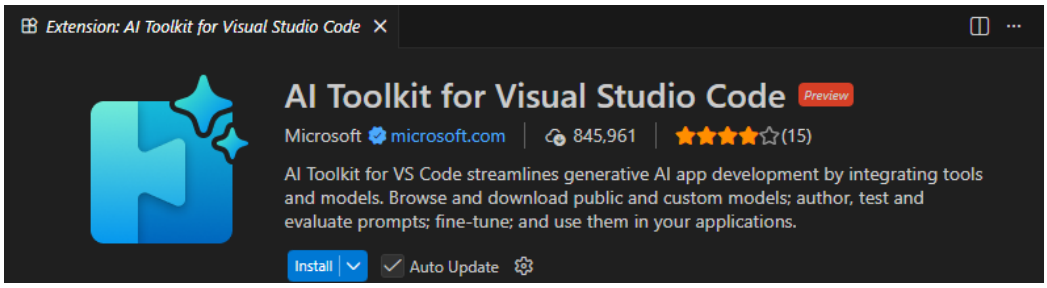


Figure 10. Instalación del AI Toolkit

La **Figure 11** muestra la estructura general del **VS Code Editor**. Cuando se abre el menú **Auto** en la esquina inferior derecha, se muestran los **Artificial Intelligence (AI)** models (modelos de Inteligencia Artificial) en el editor, y los modelos de **AI** se gestionan desde este menú. Cuando se selecciona la opción **Auto**, todos los modelos se activan y se utilizan.

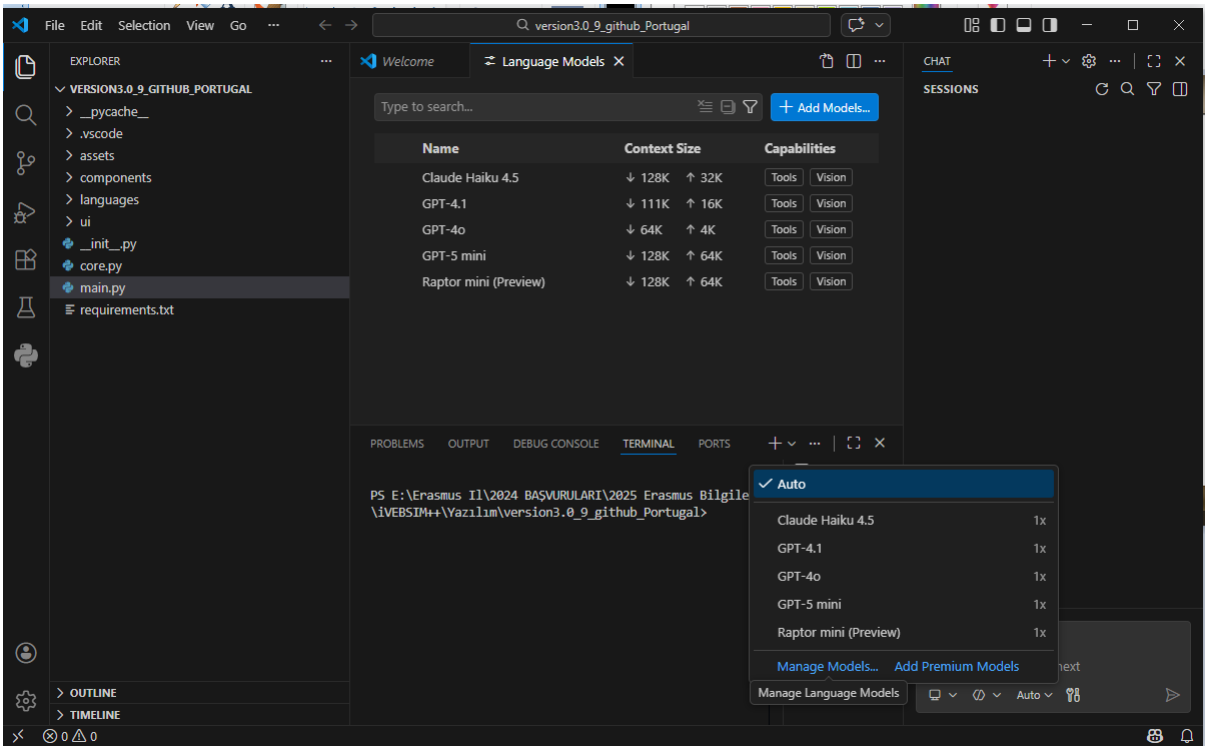


Figure 11. Gestión del soporte de AI en VS Code

3.4.2. Funcionamiento

Una vez finalizada la instalación, puedes iniciar el **AI Toolkit** y comenzar a usarlo. El **toolkit** ofrece las siguientes ventajas:

- **Análisis de errores:** Los resultados de los errores cometidos se pueden monitorear instantáneamente, y el proceso se puede reintentar volviendo al principio.
- **Recopilación de datos:** Se pueden recopilar datos detallados sobre cómo funciona el sistema en casos extremos forzando los límites del mismo.

El **AI Toolkit** permite a los desarrolladores probar sus modelos en un entorno local e integrarlos fácilmente en sus proyectos.

Como se muestra en la **Figure 12**, se solicitó a la **AI** (Inteligencia Artificial) desarrollar un programa para sumar números del 1 al 10 usando el menú de la parte inferior derecha. Posteriormente, se le indicó a la **AI** que escribiera este código en **Python** en el **Editor**. La **AI** completó las tareas abriendo un archivo llamado **sum_example.py**, guardando el código, ejecutándolo e imprimiendo la salida (**output**). Este tipo de integración de **AI** se utilizó frecuentemente en este proyecto.

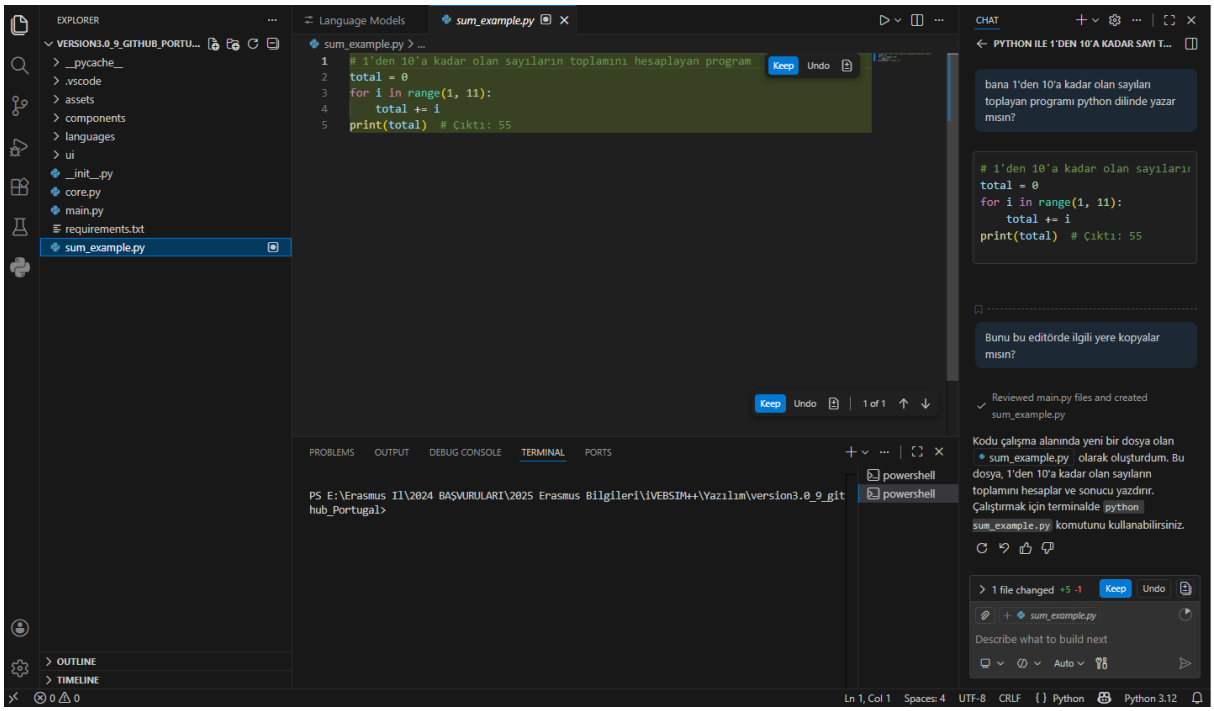


Figure 12. Uso de la Inteligencia Artificial en VS Code

4. VARIABLES DE PYTHON Y REGLAS PARA VARIABLES

En **Python**, no existe un comando específico para declarar una **variable**. Una **variable** se crea en el momento en que le asignas un valor por primera vez. Las **variables** son contenedores para almacenar valores de datos.

Reglas para los Nombres de Variables:

- El nombre de una **variable** debe comenzar con una letra o el carácter de subrayado (_).
- El nombre de una **variable** no puede comenzar con un número.
- El nombre de una **variable** solo puede contener caracteres alfanuméricos y subrayados (A-z, 0-9 y _).
- Los nombres de las **variables** distinguen entre mayúsculas y minúsculas (por ejemplo: age, Age y AGE son tres variables diferentes).
- Las palabras clave de **Python** (tales como if, for, while, print) no pueden usarse como nombres de variables.

Ejemplos de Asignación de Variables:

```
user_name = "Mert"

exam_score = 85

_status = True
```

En el ejemplo anterior, user_name almacena un valor de tipo **string** (cadena de texto), exam_score almacena un valor **integer** (entero) y _status almacena un valor **boolean** (booleano). **Python** determina automáticamente el tipo de dato basándose en el valor asignado.

4.1. Operadores de Python

Los operadores se utilizan para realizar operaciones sobre **variables** y valores. **Python** divide los operadores en varios grupos: aritméticos, de asignación, de comparación y operadores lógicos.

4.1.1. Operadores aritméticos

Los operadores aritméticos se utilizan con valores numéricos para realizar operaciones matemáticas comunes.

Operador	Definición	Ejemplo
+	Suma (Addition)	a + b
-	Resta (Subtraction)	a - b
*	Multiplicación (Multiplication)	a * b
/	División (Division)	a / b
%	Módulo (Modulus) - Resto de la división	a % b
**	Exponenciación (Exponentiation) - Potencia	a ** b
//	División entera (Floor Division) - División sin resto	a // b

Ejemplos de códigos:	**Salida de pantalla**
a = 10 b = 5 print(a) print(b)	10 5
result = a + b print("Result:", result)	Result: 15
result = a - b print("Result:", result)	Result: 5
sonuc = a * b print("Result:", result)	Result: 50
result = a / b print("Result:", result)	Result: 2.0
result = a % b print("Result:", result)	Result: 0
result = a ** b print("Result:", result)	Result: 100000

result = a // b
print("Result:", result)
	Result: 2

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

4.1.2. Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las **variables**. También pueden utilizarse para realizar una operación matemática y asignar el resultado a la misma **variable** de forma simultánea

Ejemplos de códigos	Salida de pantalla
x = 10 print(x)	10
x = 10 x += 5 print(x)	15
x = 10 x -= 3 print(x)	7
x = 4 x *= 3 print(x)	12
x = 10 x /= 4 print(x)	2.5
x = 10 x //= 4 print(x)	2
x = 10 x %= 3 print(x)	1
x = 2 x **= 3 print(x)	8

Operador	Ejemplo	Descripción
=	a = 2	Se asigna el valor 2 a la variable a.
+=	a += 2	Suma 2 a la variable a y asigna el resultado de nuevo a a. (Equivalente a a = a + 2)
-=	a -= 2	Resta 2 de la variable a y asigna el resultado de nuevo a a. (Equivalente a a = a - 2)
*=	a *= 2	Multiplica la variable a por 2 y asigna el resultado de nuevo a a. (Equivalente a a = a * 2)
/=	a /= 2	Divide la variable a por 2 y asigna el resultado de nuevo a a. (Equivalente a a = a / 2)
%=	a %= 2	Calcula el módulo de la variable a entre 2 y asigna el resultado de nuevo a a. (Equivalente a a = a % 2)
**=	a **= 2	Eleva la variable a a la potencia de 2 y asigna el resultado de nuevo a a. (Equivalente a a = a ** 2)
//=	a //= 2	Realiza una división entera en la variable a entre 2 y asigna el resultado de nuevo a a. (Equivalente a a = a // 2)

4.1.3. Operadores de comparación

Los operadores de comparación se utilizan cuando se desea comparar los valores de dos **variables** y realizar una acción basada en el resultado.

Operador	Ejemplo	Descripción
==	• **Igual a** (Equal to)	a == b
!=	• **No igual a** (Not equal to)	a != b
<	• **Menor que** (Less than)	a < b
>	• **Mayor que** (Greater than)	a > b
<=	• **Menor o igual que** (Less than or equal to)	a <= b
>=	• **Mayor o igual que** (Greater than or equal to)	a >= b

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Ejemplos de códigos	Salida de pantalla
a = 10 b = 10 print(a == b)	True
a = 10 b = 5 print(a != b)	True
a = 7 b = 10 print(a > b)	False
a = 5 b = 8 print(a < b)	True
a = 10 b = 10 print(a >= b)	True
a = 12 b = 10 print(a <= b)	False

4.1.4. Operadores lógicos

Los operadores lógicos se utilizan para decidir el flujo del programa al comprobar los valores de dos o más **variables**.

Operador	Ejemplo	Descripción
and	a < 3 and b >= 5	Devuelve **True** si todas las condiciones son verdaderas. En este ejemplo, si la **variable a** es menor que 3 y la **variable b** es igual o mayor que 5, devuelve **True**.
or	a < 3 or b > 4	Devuelve **True** si al menos una de las condiciones es verdadera. En este ejemplo, basta con que la **variable a** sea menor que 3 o la **variable b** sea mayor que 4 para devolver **True**.
not	not(a < 3)	Se utiliza para **invertir el resultado** (si es **True**, se convierte en **False**; si es **False**, se convierte en **True**).

Ejemplos de códigos	Salida de pantalla
a = 10 b = 5 print(a<3 and b>=5)	False
a = 10 b = 5 print(a<3 and b>=5)	True
durum = True print(not durum)	False

4.2. Tipos de datos de Python

En **Python**, generalmente existen tipos de datos como **string** (textual), **numbers** (numérico), **boolean**, **list**, **tuple**, **dictionary** y **set**. Los tipos de variables que más utilizaremos en la etapa inicial se enumeran a continuación.

4.2.1. Tipo de dato String (Textual)

Los tipos de datos **string** se utilizan para almacenar expresiones textuales. Los valores se escriben dentro de comillas simples (' ') o comillas dobles (" "). Los caracteres pueden ser letras (t, c), números (1, 9, 2, 3) o símbolos especiales (&, /).

```
name=”Mert”
surname=”Yılmaz”
```

4.2.2. Tipo de dato numérico

Se utiliza para almacenar valores numéricos. Existen tres tipos numéricos principales:

- **int** (Integer): Se usa para números enteros (p. ej., 5, -10, 100).
- **float** (Floating Point): Se usa para números decimales (p. ej., 3.14, -0.5).
- **complex**: Se usa para números complejos (p. ej., 3+5j).

Ejemplo:

```
age = 17                # int
pi_value = 3.14         # float
```

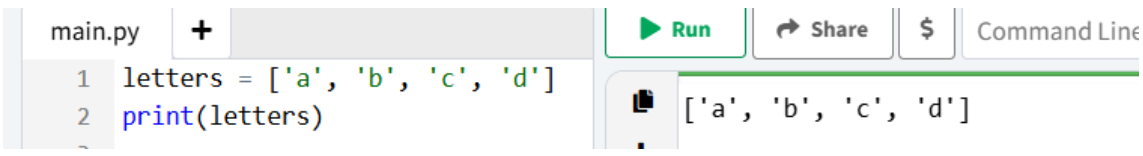
4.2.3. Listas de Python (Lists)

Las **lists** se utilizan para almacenar múltiples elementos en una sola **variable**. Se definen utilizando corchetes []. Son ordenadas, modificables y permiten valores duplicados. Puedes almacenar diferentes tipos de datos como **int**, **float** y **string** en una misma lista.

Las **lists** se utilizan para almacenar múltiples datos en una estructura ordenada y que se puede modificar.

Ejemplos:

```
fruits = ["apple", "banana", "cherry"]
print(fruits)


sayı=[10,20,30,40,50,60,70]
sehir=["İzmir", "İstanbul", "Ankara", "Bolu"]
first_list=[“Ankara”, 312, 0.6]
```



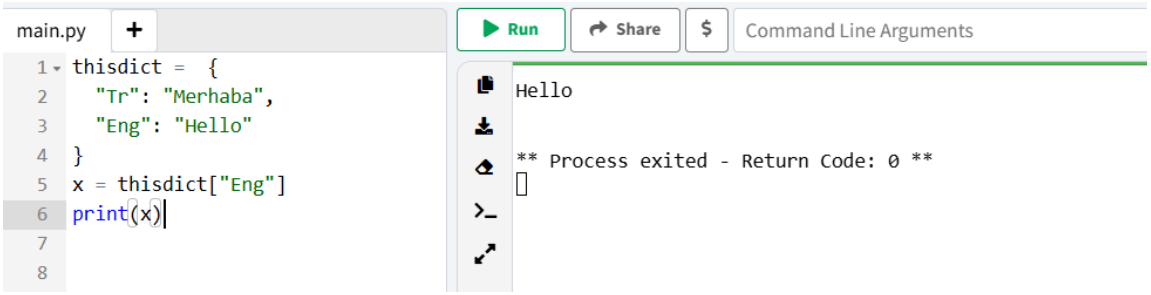
4.2.4. Diccionario (Dictionary)

Los **dictionaries** se utilizan para almacenar valores de datos en pares **key:value** (clave:valor). Un **dictionary** es una colección que es ordenada (a partir de **Python 3.7**), modificable y no permite duplicados.

Los **dictionaries** se escriben con llaves {}.

Ejemplos:

```
student = {
  "name": "Mert",
  "school_number": 154,
  "grade": 11
}
print(student)
print(student["name"])                    # Output: Mert
```



5. ESTRUCTURAS DE DECISIÓN

Las **decision structures** (estructuras de decisión) permiten que el programa realice diferentes acciones dependiendo de si se cumple o no una determinada condición.

5.1. Estructura de decisión – IF..ELIF

La sentencia **if** se utiliza para ejecutar un bloque de código solo si una condición específica es verdadera (**true**). Si la primera condición es falsa, la sentencia **elif** (abreviatura de *else if*) nos permite comprobar múltiples condiciones adicionales.

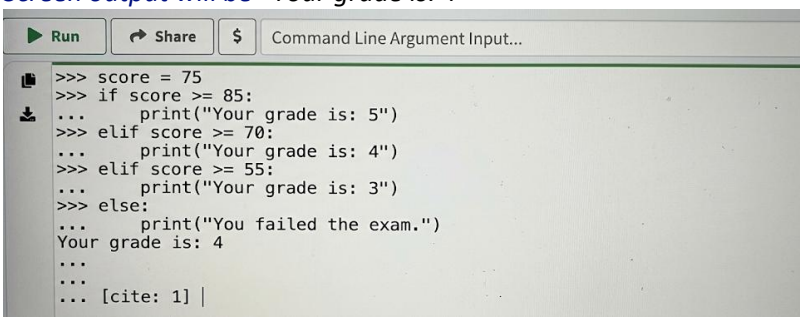
Sintaxis:

```
if condicion:
    # Código que se ejecuta si la condición es verdadera
elif otra_condicion:
    # Código que se ejecuta si la primera condición es falsa
    # y esta condición es verdadera
else:
    # Código que se ejecuta si ninguna de las condiciones anteriores es verdadera
```

Ejemplo de código:

```
score = 75

if score >= 85:
   print("Your grade is: 5")

elif score >= 70:
   print("Your grade is: 4")

elif score >= 55:
   print("Your grade is: 3")

else:
   print("You failed the exam.")
```

Screen output will be *Your grade is: 4*



Ejemplo de código:

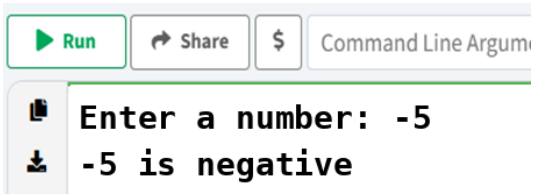
```
a=int(input("input a number"))
if(a>0):
    print(str(a)+"  is positive")
elif a < 0:
    print (str(a)+"  is negative")
else:
    print (str(a)+" is zero")
```

El programa solicita que se ingrese un número desde el teclado.

Por ejemplo, si suponemos que se ingresa el valor **-5** en la **variable 'a'**, el programa comenzará a realizar las comprobaciones en el bloque **if**:

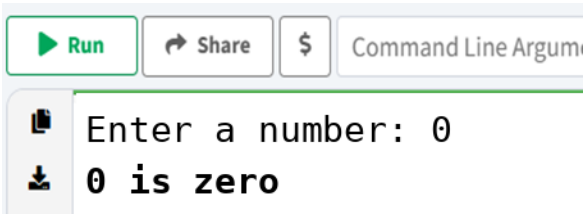
- **¿Es el valor de la variable 'a' mayor que 0?** Dado que -5 no es mayor que 0, el programa pasa a la siguiente expresión de condición.

- **a < 0, ¿es el valor de la variable 'a' menor que 0?** Sí; como la condición es verdadera (**True**), se ejecutará el comando **print()**. En pantalla se mostrará: *"El número -5 es negativo"*.



Por ejemplo, supongamos que se ingresa el valor **0** en la **variable 'a'**:

- **¿Es el valor de la variable 'a' mayor que 0?** Como 0 no es mayor que 0, el programa pasa a la siguiente condición.
- **a < 0, ¿es el valor de la variable 'a' menor que 0?** No, la expresión 0 < 0 no es verdadera. Por lo tanto, dado que no se cumple ninguna de las condiciones anteriores, el programa pasa al bloque **else**.
- Se ejecutará el comando **print()** y se mostrará en pantalla: **"0 es cero"**.



6. ESTRUCTURAS DE BUCLES (LOOPS)

Las estructuras de bucles se utilizan para repetir un bloque de código específico varias veces. En lugar de escribir el mismo código una y otra vez, usamos **loops** para realizar tareas repetitivas de manera eficiente. Los bucles ayudan a que nuestro código sea más simple, comprensible y corto. Utilizamos las estructuras de bucle **For**, **While** y **Do While**.

6.1. Bucle FOR

El bucle **for** se utiliza para iterar sobre una secuencia (como una **list**, una **tuple**, un **dictionary**, un **set** o un **string**). Generalmente se usa cuando el número de repeticiones se conoce de antemano.

Sintaxis:

Python

```
for variable in secuencia:
    # Código a ejecutar
```

Uso de la función range() con FOR:

Para recorrer un bloque de código un número específico de veces, podemos usar la función range(). Esta función devuelve una secuencia de números que comienza en 0 por defecto, se incrementa de 1 en 1 (por defecto) y termina en un número específico.

Sintaxis: La función range() puede recibir hasta tres argumentos: range(start, stop, step)

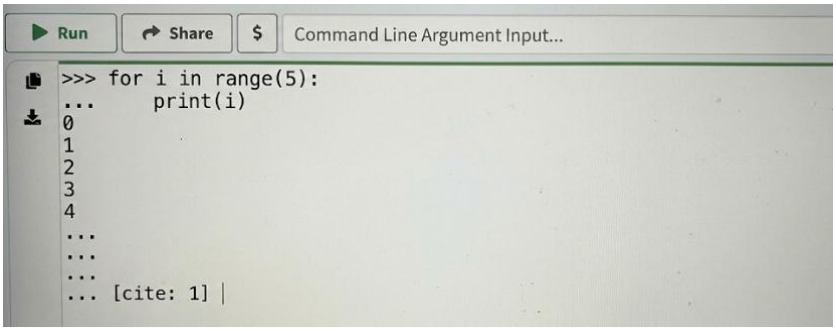
Parámetros:

- **start** (Opcional): El valor inicial de la secuencia. Si se omite, el valor por defecto es 0.
- **stop** (Requerido): El número en el que se detiene la secuencia. Ten en cuenta que la secuencia **no incluye este número** (se detiene en stop - 1).
- **step** (Opcional): El incremento (o decremento) entre cada número de la secuencia. Si se omite, el valor por defecto es 1.

Ejemplo 1:

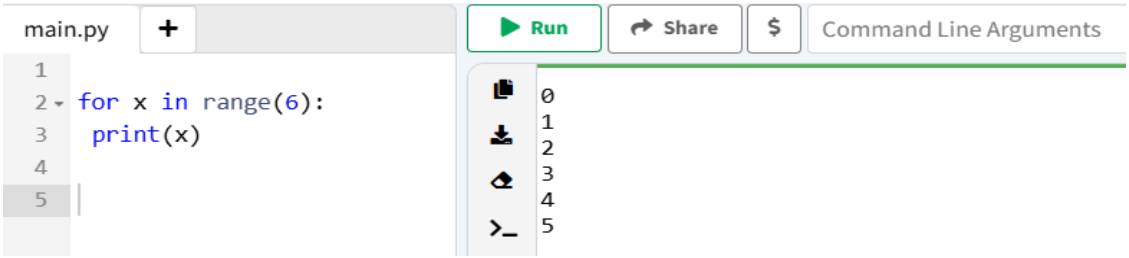
for i in range(5):

print(i)



Nota: *El código anterior imprimirá los números del* ***0 al 4****. El número 5* ***no está incluido****.*

Ejemplo 2: El siguiente código muestra los números del **1 al 6** en la pantalla.



Ejemplo 3: En el siguiente código, los valores se imprimen en incrementos de 100, desde el 50 hasta el 350.

Si ejecutamos el código paso a paso:

- **Se asigna el valor 50 a la variable contador x** y se comprueba su valor. ¿Es x <= 350? ¿Es x menor que 350? Sí, se cumple nuestra condición y el programa entra en el bucle. Combina el **string** "Number =" con el valor de la variable x e imprime **"Number = 50"** en la pantalla. Luego, regresa al inicio del bucle.

- **El valor de la variable contador x aumenta en 100.** El valor de la variable x pasa a ser 150. ¿Es 150 < 350? Sí, se cumple nuestra condición. Entra en el bucle. El comando **print** se

ejecuta de nuevo, combinando el valor de la variable x con la expresión de texto constante e imprimiendo **"Number = 150"** en la pantalla. Regresa al inicio del bucle.

- **El valor de la variable contador x aumenta en 100.** El valor de la variable x pasa a ser 250. ¿Es 250 < 350? Sí, se cumple nuestra condición. Entra en el bucle. Se ejecuta el comando **print** nuevamente, combinando el valor de la variable x con la expresión de texto constante e imprimiendo **"Number = 250"** en la pantalla. Luego, regresa al inicio del bucle.
- **Se incrementa el valor de la variable contador x en 100.** El valor de la variable x pasa a ser 350. ¿Es x < 350? Es decir, ¿es 350 < 350? **No**, nuestra condición no se cumple porque el valor es igual al límite. Salimos del bucle. Como no hay más comandos que ejecutar fuera del bucle, nuestro programa finaliza. La pantalla se verá como en la imagen de arriba.

De este modo, ejecutamos los comandos dentro del bucle **for** 3 veces hasta alcanzar el límite. En conclusión, el número de **iteraciones** del bucle **for** varía dependiendo del valor inicial y la condición. Los comandos dentro del bucle continúan ejecutándose hasta que se alcanza la condición de parada.

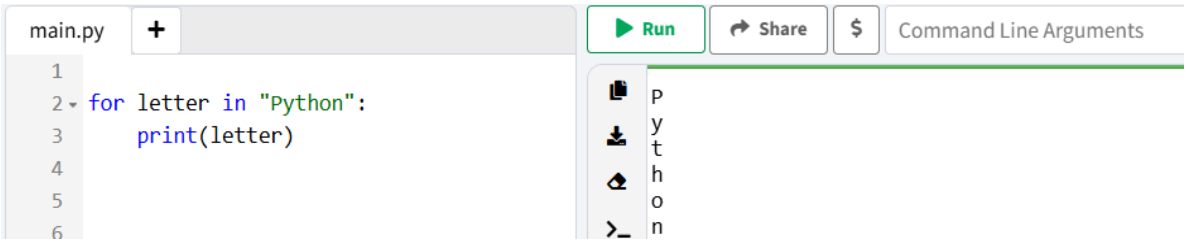
Código	Salida	Descripción
range(5)	0, 1, 2, 3, 4	Comienza en 0, se detiene antes de 5.
range(2, 6)	2, 3, 4, 5	Comienza en 2, se detiene antes de 6.
range(0, 10, 2)	0, 2, 4, 6, 8	Comienza en 0, incrementa de 2 en 2.
range(5, 0, -1)	5, 4, 3, 2, 1	Cuenta hacia atrás desde 5 hasta 1.

Uso del comando "in" con el bucle For

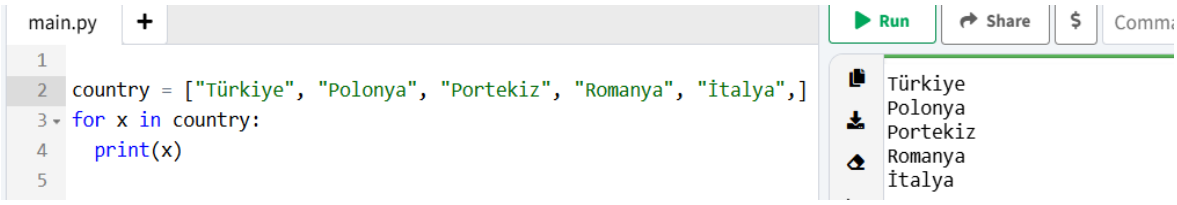
El comando **"in"** en un bucle **for** se utiliza para comprobar si un valor existe dentro de una secuencia (como una **list**, un **string** o un **range**). Actúa como un puente que permite al bucle tomar cada elemento de la secuencia uno por uno y asignarlo a la **variable del bucle**.

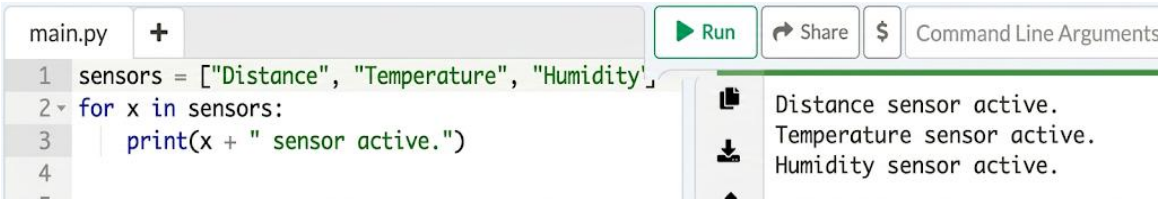
Uso con un String (Texto):

Un bucle **for** también puede iterar a través de cada carácter en un **string**.

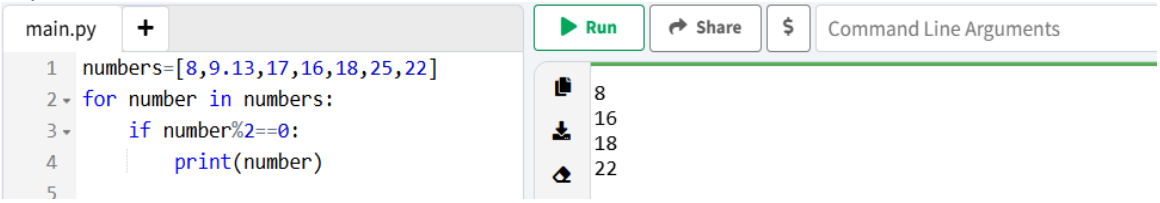


Salida: P, y, t, h, o, n (cada uno en una línea nueva).
Uso con una lista (List):
Cuando se utiliza con una **list**, el comando **"in"** asegura que la variable del bucle tome el valor de cada elemento de la lista de forma secuencial.
Ejemplo con una lista: El proceso para imprimir una lista de países definida es el siguiente:

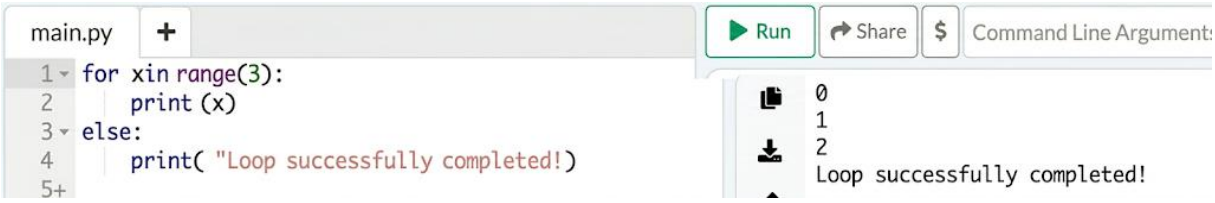




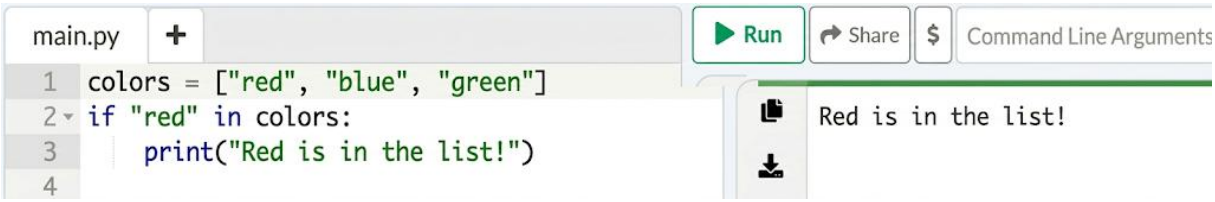
Ejemplo con números: El sistema imprime los números pares de una secuencia de números dada en la pantalla.



Else en el bucle FOR La palabra clave **else** en un bucle **for** especifica un bloque de código que se ejecutará cuando el bucle haya terminado por completo.



Comprobación de pertenencia (Membership Check) Más allá de los bucles, la palabra clave **"in"** también puede utilizarse en sentencias **if** para comprobar la presencia de un elemento dentro de una secuencia (como una lista o una cadena de texto).



6.2. Bucle WHILE

El bucle **while** se utiliza para ejecutar un conjunto de instrucciones mientras una condición sea verdadera (**true**). A diferencia del bucle **for**, el bucle **while** se prefiere generalmente cuando el número de repeticiones **no está predeterminado**.

El bucle finaliza en cuanto la condición deja de cumplirse, y el programa continúa su ejecución desde el punto donde terminó el bucle.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Sintaxis:

```
while condition:
  # Code to be executed
```

Ejemplo

El siguiente código imprime los números del **1 al 4** en la pantalla.

PASO A PASO DEL BUCLE WHILE (i < 4)

- **Inicialización:** La variable **i** comienza en 1.
- **Iteración 1:** Como 1 < 4 es **True**, se imprime **1** y el valor de **i** pasa a ser 2.
- **Iteración 2:** Como 2 < 4 es **True**, se imprime **2** y el valor de **i** pasa a ser 3.
- **Iteración 3:** Como 3 < 4 es **True**, se imprime **3** y el valor de **i** pasa a ser 4.
- **Finalización (Termination):** Ahora **i** vale 4. Como la condición 4 < 4 es **False**, el bucle se detiene.

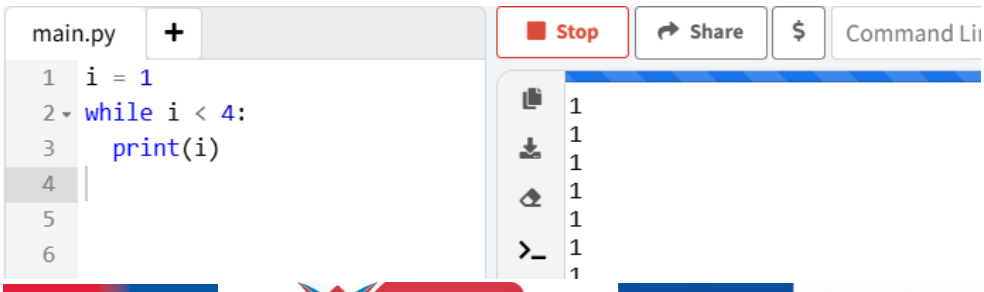
Resumen de la salida: 1, 2, 3.

Variaciones de la condición

Si aumentamos el valor del límite (empezando siempre con i = 1):

- Para **i <= 10** Los comandos se ejecutarán **10 veces**.
- Para **i <= 100** Los comandos se ejecutarán **100 veces**.
- Para **i < 5** Los comandos se ejecutarán **4 veces**. (Al no usar el símbolo <=, el bucle no se ejecuta cuando i llega a 5).

Nota sobre el "Bucle Infinito": > Si en el código falta la instrucción i += 1, el valor de **i** nunca aumentará. Por lo tanto, la condición siempre será verdadera (**True**), creando un **"Infinite Loop"**. En este caso, el programa no se detendrá solo y deberás presionar el botón de **"Stop"** (Detener) en tu editor de código.



TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

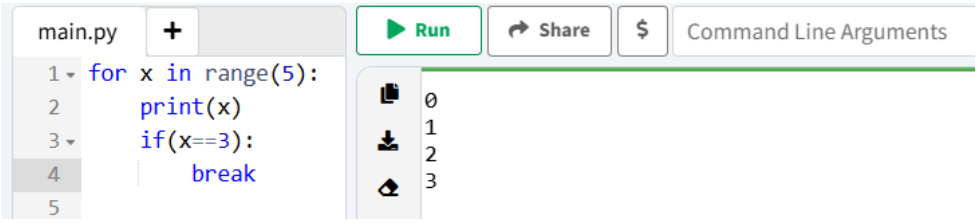
6.3. La sentencia Break

Con la sentencia **break**, podemos detener el bucle incluso si la condición del **while** sigue siendo verdadera. Esto es particularmente útil cuando deseas salir del bucle cuando ocurre una condición secundaria específica.

Ejemplo de código:

Sin Break, Salida: 1, 2, 3, 4 (El bucle termina cuando i llega a 4).

Con Break, Salida: 1, 2, 3 (El bucle termina cuando i llega a 3).



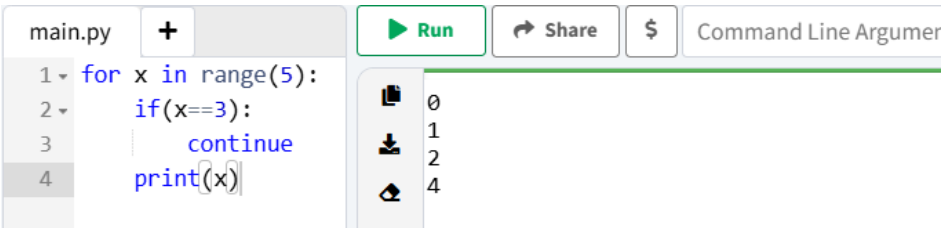
6.4. La sentencia Continue

Con la sentencia **continue**, podemos detener la iteración actual del bucle y saltar al siguiente ciclo. A diferencia de **break**, no termina el bucle completo.

Código de ejemplo:

En el código siguiente, se comprueba el valor de **x**, y si es igual a 3, se ejecuta el comando continue, regresando al principio del bucle. Al observar la salida, puedes ver que el valor 3 no se imprime en la pantalla por esta razón.

Salida: 1, 2, 4, 5, 6 (Se omite el número 3).



TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by
the European Union

CREACIÓN DE SOFTWARE DE SIMULACIÓN

7. SOFTWARES DE SIMULACIÓN

Los softwares de simulación son programas informáticos que proporcionan una experiencia realista al usuario al imitar un sistema físico, proceso o dispositivo del mundo real en un entorno digital. Estos softwares nos permiten prever cómo reaccionará un sistema a entradas específicas mediante el uso de complejos modelos matemáticos y algoritmos. Generalmente se utilizan con fines educativos y de ingeniería; por lo tanto, los escenarios que son de alto costo, peligrosos o de difícil acceso se pueden experimentar repetidamente en un espacio virtual seguro y controlado. Beneficios Básicos de los Softwares de Simulación

Beneficios Básicos de los Softwares de Simulación

- **Seguridad:** Permite trabajar en entornos libres de riesgo para situaciones que conllevan riesgos vitales en la vida real (alto voltaje, alta presión, etc.)
- **Ahorro de Costos:** Evita el desgaste del equipo real y elimina los gastos operativos como la energía.
- **Análisis de Errores:** Los resultados de los errores cometidos se pueden monitorear instantáneamente, y el proceso se puede reiniciar y probar de nuevo.
- **Recolección de Datos:** Proporciona datos detallados sobre cómo funciona el sistema en situaciones extremas al llevarlo al límite.

7.1. Softwares de Simulación de Control Industrial

Los sistemas de simulación de control electrónico industrial son softwares que permiten diseñar y probar procesos de automatización complejos y circuitos eléctricos en un entorno digital sin necesidad de una instalación física. Tiene muchas ventajas como se indica a continuación;

- **Seguridad**: Reduce a cero el riesgo de descarga eléctrica, explosión o daños materiales que puedan ocurrir en los paneles reales. Permite experimentar sin miedo a cometer errores.
- **Ahorro de Costos**: Permite configurar circuitos complejos sin comprar físicamente costosos motores, interruptores y contactores.
- **Velocidad de Aprendizaje:** Acelera significativamente el proceso de aprendizaje porque muestra instantáneamente cómo los conocimientos teóricos dan resultado en la práctica.
- **Bajos Requisitos del Sistema:** Puede ejecutarse sin problemas y sin retrasos (lagging) incluso en computadoras muy antiguas.
- **Desarrollo de la Lógica:** Estos programas minimizan los márgenes de error. Es el mejor punto de partida para establecer la "lógica de control" antes de pasar a la programación de PLC (Programmable Logic Controller / Controlador Lógico Programable).

  Además de las ventajas, el personal técnico debe saber que;

- **Problema de Actualización del Programa:** El software es bastante antiguo y no cubre por

completo los avances en los sistemas de automatización modernos (PLCs modernos, servo drives, etc.).

- **Número Limitado de Elementos:** Su librería (*library*) puede ser insuficiente para proyectos muy profesionales.
- **Simplicidad Visual:** Puede que no ofrezca elementos visuales dinámicos tan detallados como sus competidores más avanzados (como ©TIA Portal, etc.).
- **No Cubre Errores Físicos:** No puede simular fallas físicas de la vida real como cables sueltos, oxidación o interferencia magnética.

A la luz de todo esto, al crear un software de simulación —al que llamaremos iVEBSIM+ (*Artificial Intelligence Supported Open Source Simulator* / Simulador de Código Abierto Soportado por Inteligencia Artificial) de ahora en adelante—;

- **Librería Amplia:** Incluye elementos de control clásicos como contactores, relés de tiempo (*time relays*), botones, motores asíncronos y varios sensores.
- **Simulación Dinámica:** Después de configurar el circuito, se puede ver cómo funciona el sistema en tiempo real presionando el botón "Play".
- **Verificación de Errores:** El software le advierte cuando se realiza una conexión incorrecta o ocurre un cortocircuito.
- **Soporte Multi-Idioma:** Al ser un software local, ofrece una interfaz completamente en turco. Soporta varios idiomas.
- **Lógica de Arrastrar y Soltar (*Drag-and-Drop*):** El diseño del circuito se realiza con una interfaz visual en lugar de lenguajes de programación complejos.

Características	**iVEBSIM+**	**Software Profesional (TIA Portal, etc.)**
Propósito de Uso:	Entrenamiento y Lógica Básica	Aplicación Industrial
Nivel de Dificultad:	Muy Fácil	Difícil / Requiere Experiencia Técnica
Costo	Gratis / Accesible	Muy Alto
Requisitos de Hardware:	Bajos	Altos

Tabla 1. Comparación de Software Industrial

8. TKINTER – PROGRAMACIÓN VISUAL

8.1 ¿Qué es Tkinter?

Las librerías de Interfaz Gráfica de Usuario (GUI - *Graphical User Interface*) pertenecientes a un lenguaje de programación se utilizan al crear una interfaz en un software.

Tkinter es la interfaz gráfica de usuario estándar del Lenguaje de Programación Python. Está construido sobre el conjunto de herramientas Tcl/Tk y se instala automáticamente junto con el Lenguaje de Programación Python.

Es de tamaño pequeño y rápido. Además de su independencia de plataforma (Windows, macOS, Linux), lo más importante es que es fácil de aprender y usar. Código Fuente: Lib/tkinter/__init__.py

8.1.1. Arquitectura de Tkinter

Tcl/Tk no es una sola librería; consta de varios módulos diferentes, cada uno con una funcionalidad separada y su propia documentación oficial. Las versiones binarias de Python también ofrecen un módulo de complemento junto a ellas.

Tcl

Tcl es un lenguaje de programación interpretado dinámico, al igual que Python. Aunque se puede utilizar por sí solo como un lenguaje de programación de propósito general, lo más común es que se integre en aplicaciones de C como un motor de scripts o como una interfaz para el conjunto de herramientas Tk. A diferencia de Python, el modelo de ejecución de Tcl está diseñado en torno al multitarea cooperativo, y Tkinter cierra esta diferencia.

Tk

Tk es un paquete de Tcl implementado en C que añade comandos especiales para crear y manipular widgets de GUI (*arayüz bileşenleri* / componentes de interfaz).

Ttk

Themed Tk (Ttk) es una familia de widgets de Tk más nueva que proporciona una apariencia mucho mejor en diferentes plataformas en comparación con muchos widgets clásicos de Tk. Ttk se ha distribuido como parte de Tk desde la versión 8.5 de Tk. Los enlaces (*bindings*) de Python se proporcionan en un módulo separado, tkinter.ttk. Internamente, Tk y Ttk utilizan los recursos del sistema operativo subyacente; es decir, Xlib en Unix/X11, Cocoa en macOS y GDI en Windows.

8.1.2. Componentes Básicos de Tkinter - Widgets Un widget (*arayüz bileşeni*) se puede llamar básicamente un componente. Realizan una tarea específica. Una interfaz de usuario de Tkinter consta de widgets individuales. Cada widget se representa como un objeto de Python instanciado a partir de clases como ttk.Frame, ttk.Label y ttk.Button.

Widgets Básicos:

- **Frame:** Un marco para agrupar otros widgets.

- **Label:** Una etiqueta para mostrar texto o imágenes.
- **Button:** Un botón en el que se puede hacer clic.
- **Entry:** Una entrada de texto de una sola línea.
- **Text:** Un área de texto de varias líneas.
- **Canvas:** Un área para dibujos y gráficos.
- **Menu:** Barras de menú.
- **Scrollbar:** Barras de desplazamiento.
- **Checkbutton:** Casillas de verificación..
- **Radiobutton:** Botones de selección única.
- **Listbox:** Cuadros de lista.
- **Scale:** Deslizadores (*sliders*).
- **Spinbox:** Selectores numéricos.

8.2. Creación de Ventanas

En Tkinter, se siguen estos pasos para crear una ventana:

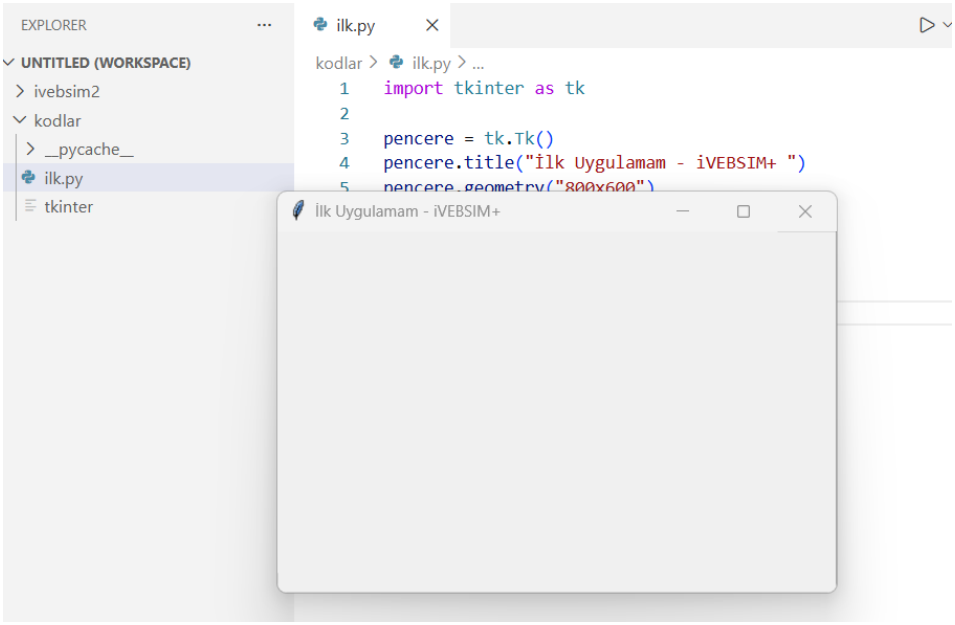
Importar Tkinter

- Crear el objeto de la ventana principal (tk.Tk()).
- Configurar las propiedades de la ventana (título, tamaño, posición).
- Añadir widgets (botones, etiquetas, campos de entrada).
- Vincular eventos (como hacer clic o presionar una tecla).
- Iniciar el bucle principal (mainloop()).

Aplicación de Muestra

```
1  import tkinter as tk
2
3  pencere = tk.Tk()
4  pencere.title("İlk Uygulamam - iVEBSIM+ ")
5  pencere.geometry("800x600")
6  pencere.mainloop()
7
8
9
```

Cuando ejecutamos los códigos, vemos nuestra página de Formulario (*Form page*) de la siguiente manera.



Ejemplo de la Primera Aplicación - Revisión Detallada

Ahora, examinemos nuestra primera aplicación de Tkinter paso a paso:

1. **Importación:**
    - import tkinter as tk

    Importa el módulo Tkinter con el alias "**tk**".

2. **Ventana Principal:**

    pencere = tk.Tk()

- tk.Tk() crea la ventana principal.
- Esta ventana es el contenedor principal para todos los demás **widgets.**
- Cada aplicación puede tener solo un objeto Tk().

3. **Propiedades de la Ventana:**

pencere.title("İlk Uygulamam - iVEBSIM+")
pencere.geometry("800x600")

- title(): El texto que aparece en la barra de título.
- geometry(): Establece el tamaño de la ventana en píxeles.

4. **Bucle Principal:**

pencere.mainloop()

- Hace que la ventana sea visible.
- Escucha los eventos del mouse y del teclado.
- Mantiene la aplicación en ejecución.
- Es un bucle infinito, pero termina cuando se cierra la ventana.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

8.3. Añadir Widgets y Vincular Eventos

Los widgets permiten añadir elementos de formulario tales como **Frame, Label, Button, Entry, Text, Canvas, Menu, Scrollbar, Checkbutton y Radiobutton**, como se mencionó en la sección 8.1.2.

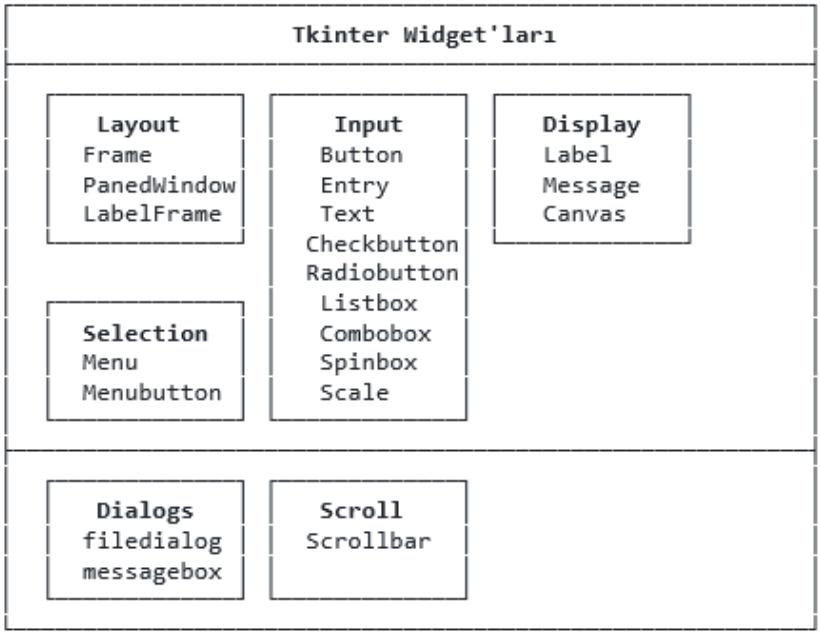


Figura 13. Widgets de Tkinter

8.3.1 Añadir Etiquetas (Adding Labels)

Se utiliza para añadir texto a la pantalla. Se utiliza el Método Label de la librería Tkinter para añadir etiquetas al formulario (*form*).

etiket = tk.Label(pencere, text="Merhaba Tkinter!", font=("Arial", 16))

- text: The text to be displayed. El texto que se va a mostrar.
- font: Font type, size, and style. Tipo de fuente, tamaño y estilo.
- bg: Color de fondo (***background***).
- fg: Color del primer plano (***foreground***).

8.3.2 Añadir Botones (Adding Buttons)

Crea un botón en el que se puede hacer clic. Se utiliza el Método Button de la librería Tkinter.

buton = tk.Button(pencere, text="Bana Tıkla", command=selam_ver)

- text: El texto que aparece en el botón.
- command: La función que se llamará durante un evento de clic.
- state: Estados "normal", "active" (activo) o "disabled" (deshabilitado).

8.3.3 Configurar Diseños (Setting Layouts)

pack() coloca objetos dentro de la ventana. Puede determinar cómo se posicionarán horizontal y

verticalmente en el formulario (*form*).

etiket.pack(pady=20)

buton.pack(pady=10)

- pady: Espaciado vertical (píxeles).
- padx: Espaciado horizontal.
- side: De qué lado (**TOP**, **BOTTOM**, **LEFT**, **RIGHT**).
- fill: Llenado de espacio vacío (**X**, **Y**, **BOTH**).
- expand: Expansión en el espacio vacío.

Ejemplo de solicitud

```
import tkinter as tk
# Create main window
pencere = tk.Tk()
pencere.title("İlk Uygulamam - iVEBSIM+")
pencere.geometry("800x600")


# Create label (display text)

etiket = tk.Label(pencere, text="Merhaba Tkinter!", font=("Arial", 16))
etiket.pack(pady=20)


# Click function

def selam_ver():
   etiket.config(text="Butona tıklandı!")

buton = tk.Button(pencere, text="Tıklayınız", command=selam_ver)
buton.pack(pady=10)

# Start main loop
pencere.mainloop()
```

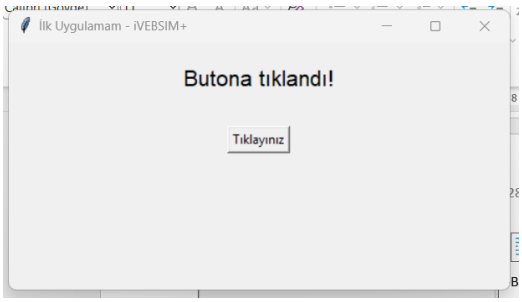
Este código realiza las siguientes operaciones:

- Carga la librería Tkinter.
- Crea y configura la ventana principal.
- Crea una etiqueta (label) que muestra el texto "Merhaba Tkinter!".
- Define la función de clic (*click function*).
- Crea el botón "Tıklayınız".
- Coloca los widgets (componentes de interfaz) dentro de la ventana.

- Muestra la ventana y escucha los eventos.

Ejecutando el código

Al hacer clic en el botón, aparece el formulario tal como se muestra en la imagen lateral.



8.3.4 Agregar entradas

Se utiliza para recibir texto de una sola línea introducido por el usuario.

```
kutu = tk.Entry(pencere)
kutu.pack()
```

Ejemplo de solicitud

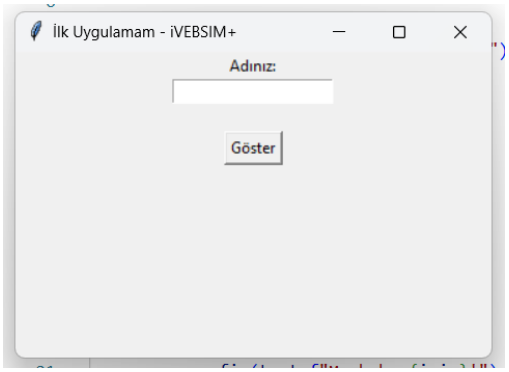
```
import tkinter as tk

pencere = tk.Tk()
pencere.title("İlk Uygulamam - iVEBSIM+")
pencere.geometry("800x600")

# Etiket
etiket = tk.Label(pencere, text="Adınız:")
etiket.pack()

# Giriş kutusu
giris = tk.Entry(pencere)
giris.pack()

# Sonucu gösterecek etiket
sonuc = tk.Label(pencere, text="")
sonuc.pack()

def goster():
    isim = giris.get()
    sonuc.config(text=f"Merhaba {isim}!")

buton = tk.Button(pencere, text="Göster", command=goster)
buton.pack()
```

```
pencere.mainloop()
```

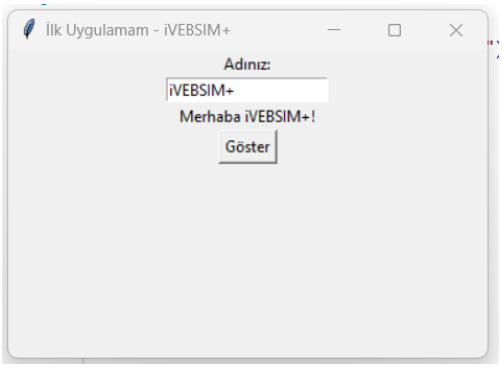
Este código realiza las siguientes operaciones:

- Carga la biblioteca Tkinter.
- Crea y configura la ventana principal.
- Crea una etiqueta (Etiqueta) que muestra el texto "Nombre".
- Crea un campo de entrada (Campo de entrada) para ingresar el nombre.
- Crea una etiqueta para mostrar el resultado.
- Define la función de clic.
- Crea el botón "Mostrar".
- Muestra la ventana y escucha los eventos.
- Ejecución del código

Ejecutando los códigos



Al hacer clic en el botón, vemos el formulario tal como se muestra en la imagen lateral.



8.3.5 Agregar botón de opción

Se utiliza para que el usuario seleccione solo una opción entre varias.

```
secenek = tk.StringVar()
r1 = tk.Radiobutton(pencere, text="Erkek", variable=secenek, value="Erkek")
r2 = tk.Radiobutton(pencere, text="Kadın", variable=secenek, value="Kadın")
r1.pack()
r2.pack()
```

8.3.6 Agregar botón de verificación

Se utiliza para que el usuario seleccione una o más opciones de entre las opciones dadas.

```
secim = tk.IntVar()
check = tk.Checkbutton(pencere, text="Python seviyorum", variable=secim)
check.pack()
```

8.3.7 Agregar barra de desplazamiento

Se utiliza para añadir una barra de desplazamiento a la pantalla del formulario.

```
scroll = tk.Scrollbar(pencere)
scroll.pack(side="right", fill="y")
```

```
metin = tk.Text(pencere, yscrollcommand=scroll.set)
metin.pack()
scroll.config(command=metin.yview)
```

8.3.8 Agregar menú

Se utiliza para agregar una barra de menú superior.

```
menu = tk.Menu(pencere)
pencere.config(menu=menu)
dosya = tk.Menu(menu)
menu.add_cascade(label="Dosya", menu=dosya)
dosya.add_command(label="Çıkış", command=pencere.quit)
```

8.3.9 Vinculación de eventos

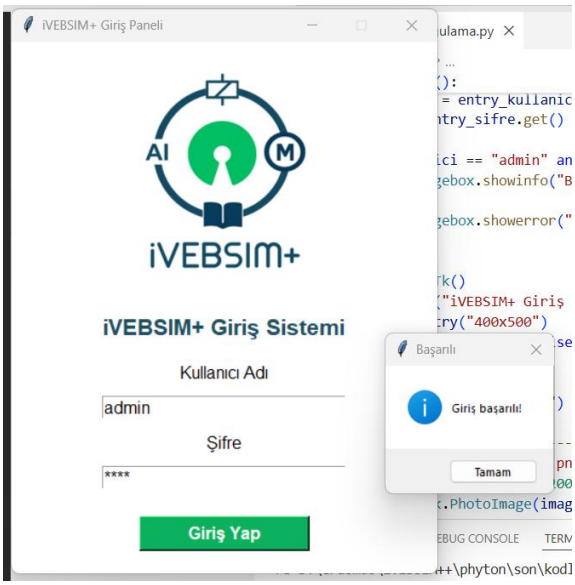
En los widgets, podemos definir el código que queremos ejecutar durante los eventos de clic dentro de una función.

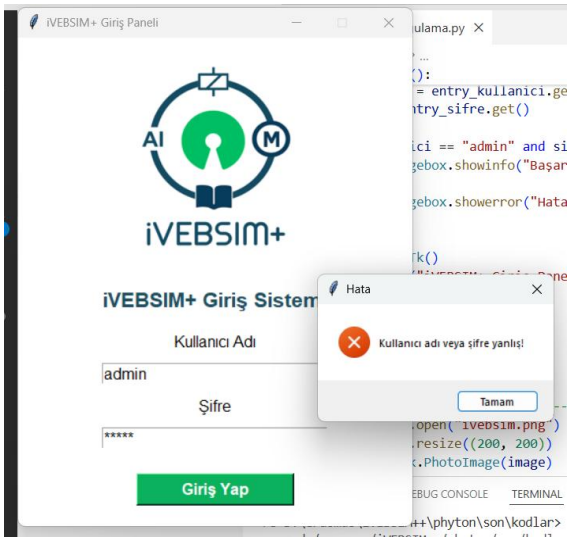
```
 def goster():
    isim = giris.get()
    sonuc.config(text=f"Merhaba {isim}!")
buton = tk.Button(pencere, text="Göster", command=goster)
```

Códigos de ejemplo:

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk   # Pillow kütüphanesi gerekli

# Giriş fonksiyonu
def giris_yap():
    kullanici = entry_kullanici.get()
    sifre = entry_sifre.get()

    if kullanici == "admin" and sifre == "1234":
        messagebox.showinfo("Başarılı", "Giriş başarılı!")
    else:
        messagebox.showerror("Hata", "Kullanıcı adı veya şifre yanlış!")
```

```
# Ana pencere
pencere = tk.Tk()
pencere.title("iVEBSIM+ Giriş Paneli")
pencere.geometry("400x500")
pencere.resizable(False, False)
# Arka plan rengi
pencere.configure(bg="white")

# ---------- LOGO ----------
image = Image.open("ivebsim.png")
image = image.resize((200, 200))  # Logo boyutu
logo = ImageTk.PhotoImage(image)

logo_label = tk.Label(pencere, image=logo,
bg="white")
logo_label.pack(pady=20)

# ---------- BAŞLIK ----------
baslik = tk.Label(pencere, text="iVEBSIM+ Giriş Sistemi",
                  font=("Arial", 16, "bold"),
                  bg="white",
                  fg="#1f4e5f")
baslik.pack(pady=10)

# ---------- KULLANICI ADI ----------
label_kullanici = tk.Label(pencere, text="Kullanıcı Adı",
                           font=("Arial", 12),
                           bg="white")
label_kullanici.pack(pady=5)
entry_kullanici = tk.Entry(pencere, font=("Arial", 12), width=25)
entry_kullanici.pack(pady=5)

# ---------- ŞİFRE ----------
label_sifre = tk.Label(pencere, text="Şifre",
                       font=("Arial", 12),
                       bg="white")
label_sifre.pack(pady=5)
entry_sifre = tk.Entry(pencere, font=("Arial", 12),
width=25, show="*")
entry_sifre.pack(pady=5)

# ---------- BUTON ----------
giris_buton = tk.Button(pencere,
                        text="Giriş Yap",
                        font=("Arial", 12, "bold"),
                        bg="#0bb15d",
                        fg="white",
                        width=15,
                        command=giris_yap)
giris_buton.pack(pady=20)
pencere.mainloop()
```





TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

8. CREACIÓN DE LA INTERFAZ DEL SIMULADOR

Al iniciarse la interfaz del simulador, aparece primero una pantalla de bienvenida. Esta pantalla muestra las organizaciones que apoyan el programa. Al hacer clic en esta ventana con el ratón, aparece la interfaz del programa. En nuestro software, la pantalla de bienvenida se llama main.py y la ventana principal del programa se llama ui.py.

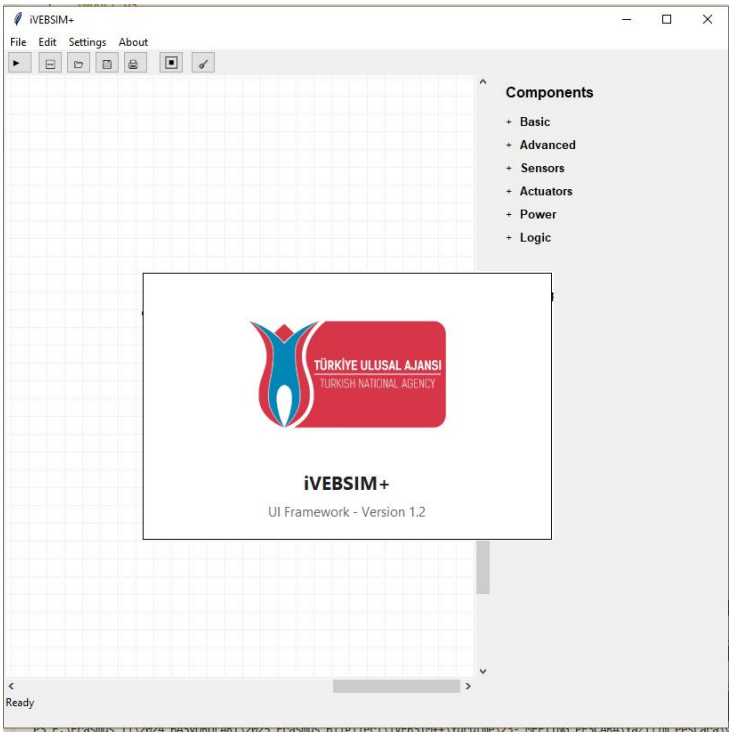


Figura 14 – Vista del software

9.1 Creación de la Pantalla de Bienvenida (Creating Splash)

Para iniciar la aplicación y mostrar una Pantalla de Bienvenida (*splash screen*) al usuario, siga estos pasos:

Se añade la librería Tkinter de Python.

```
import os

import sys
import tkinter as tk
from ui import App
```

Se crea una pantalla de bienvenida superior (*top splash screen*) en la ventana principal de TK, la cual se define como raíz (*root*).

```
def show_splash(root):
    splash = tk.Toplevel(root)
    splash.overrideredirect(True)#açıklma
```

El tamaño de la ventana parece estar fijo.

```
# Initial size
    width, height = 460, 300
    splash.update_idletasks()
    sw = splash.winfo_screenwidth()
    sh = splash.winfo_screenheight()
    x = (sw - width) // 2
    y = (sh - height) // 2
    splash.geometry(f"{width}x{height}+{x}+{y}")

    container = tk.Frame(splash, bg="#ffffff", bd=1, relief="solid")
    container.pack(fill="both", expand=True)
```

Se crea el contenido de la ventana.

```
# Logo image
    logo_path = resource_path("sembol", "logoarkaplan.png")
    logo_img = None
    try:
        if os.path.exists(logo_path):
            im = Image.open(logo_path)
            # Fit into splash width with some margin
            target_w = 280
            ratio = target_w / max(1, im.width)
            target_h = int(im.height * ratio)
            im = im.resize((target_w, target_h))
            logo_img = ImageTk.PhotoImage(im)
    except Exception:
        logo_img = None
    if logo_img is not None:
        logo = tk.Label(container, image=logo_img, bg="#ffffff")
        logo.image = logo_img
        logo.pack(pady=(18, 8))
    else:
        logo = tk.Label(container, text="⚡", font=("Segoe UI", 56),
bg="#ffffff", fg="#1a73e8")
        logo.pack(pady=(22, 6))

    title = tk.Label(container, text="iVEBSIM+", font=("Segoe UI", 16,
"bold"), bg="#ffffff", fg="#202124")
    title.pack()

    subtitle = tk.Label(container, text="UI Framework - Version 1.2",
font=("Segoe UI", 11), bg="#ffffff", fg="#5f6368")
    subtitle.pack(pady=(2, 12))
```

```
    about = tk.Label(
```

El tamaño de la pantalla de bienvenida (*splash*) se vuelve a calcular y se centra.

```
# Recenter after layout so size is accurate
    splash.update_idletasks()
    width = splash.winfo_width()
    height = splash.winfo_height()
    sw = splash.winfo_screenwidth()
    sh = splash.winfo_screenheight()
    x = (sw - width) // 2
    y = (sh - height) // 2
    splash.geometry(f"{width}x{height}+{x}+{y}")
```

Figura 15 – Vista de la Pantalla de Bienvenida (Splash View)

La pantalla de bienvenida (*splash*) se activa mediante un clic o al presionar una tecla, lo que ejecuta la función de continuar (*"proceed"*). Esta acción cierra la pantalla de bienvenida y vuelve a llamar a la pantalla principal.

```
def proceed(event=None):
        try:
            splash.destroy()
        except Exception:
            pass
        root.deiconify()
        root.geometry("1100x780")
        App(root)
        root.focus_force()

    # Close on any click or key
    for widget in (splash, container, logo, title, subtitle, about):
        widget.bind("<Button-1>", proceed)
```

```
        widget.bind("<Key>", proceed)
```

La pantalla de bienvenida (*splash*) es llamada y se inicia el bucle de eventos (*event loop*).

```
def run():
    root = tk.Tk()
    root.withdraw()
    show_splash(root)
    root.mainloop()


if __name__ == "__main__":
    run(
```

9.2 Creación de la Infraestructura de la Interfaz

El archivo ui.py define la clase App, la cual proporciona la interfaz gráfica de usuario para el software. El software incluye menús, una barra de herramientas, un área de dibujo en el centro (Canvas) y paneles laterales derecho e izquierdo (acordeón de componentes). Para crear la interfaz de la aplicación, además de los puntos mencionados anteriormente, se siguen estos pasos:

Se crea una ventana de pantalla dentro de *build.ui*.

```
def _build_ui(self):
        self.master.title("iVEBSIM+")
        self.grid(row=0, column=0, sticky="nsew")
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0, weight=1)
```

La barra de menú (File / Edit / Settings / About) se crea con tk.Menu. Cada comando de menú está vinculado a un método (_new_file, _save_file, etc.).

```
# Menu bar
        self.menubar = tk.Menu(self.master)

        # File menu
        self.menu_file = tk.Menu(self.menubar, tearoff=0)
        self.menu_file.add_command(label=self._t("new"),
command=self._new_file)
        self.menu_file.add_command(label=self._t("open"),
command=self._open_file)
```

La barra de herramientas (*toolbar*) se crea utilizando ttk.Button. Se añaden botones como Guardar (*Save*), Iniciar (*Start*), Detener (*Stop*), Limpiar (*Clear*), etc.

```
self.toolbar = ttk.Frame(self)
        self.toolbar.grid(row=0, column=0, sticky="ew")
```

```
        self.btn_start = ttk.Button(self.toolbar, text="▶", width=3,
command=self.start_sim)
        self.btn_start.pack(side="left", padx=4)
        # Icon-only file action buttons next to Start
        self.btn_new = ttk.Button(self.toolbar, text="🆕", width=3,
command=self.clear)
        self.btn_clear.pack(side="left", padx=4)
```

El área principal se configura con ttk.PanedWindow y se divide en dos paneles utilizando PanedWindow. El lado izquierdo se establece como canvas_frame, el cual contiene el área de dibujo (Canvas), y el lado derecho como side_panel, el cual contiene el acordeón de componentes.

El área del lienzo (*canvas*) se crea con scrollregion, y se añaden barras de desplazamiento (*scrollbars*) horizontal y verticalmente. Cuando se cambia el tamaño del área del lienzo, la cuadrícula (*grid*) se actualizará utilizando configure. Además, al utilizar coordenadas en el lienzo, se utilizan los métodos canvasx y canvasy. Esto permite la conversión desde la pantalla hacia el lienzo (*canvas*).

```
 self.main = ttk.PanedWindow(self, orient=tk.HORIZONTAL)
        self.main.grid(row=1, column=0, sticky="nsew")
        self.rowconfigure(1, weight=1)
        self.columnconfigure(0, weight=1)

        # Canvas area
        self.canvas_frame = ttk.Frame(self.main)
        self.main.add(self.canvas_frame, weight=3)
        self.canvas_frame.rowconfigure(0, weight=1)
        self.canvas_frame.columnconfigure(0, weight=1)
        self.canvas = tk.Canvas(self.canvas_frame, bg="#ffffff",
scrollregion=(0, 0, 2000, 2000))
        self.canvas.grid(row=0, column=0, sticky="nsew")
        self.h_scroll = ttk.Scrollbar(self.canvas_frame,          # Redraw
grid on resize/scroll when enabled
        self.canvas.bind("<Configure>", lambda e:
(self._grid_enabled.get() and self._draw_grid()))
        self.canvas.bind_all("<MouseWheel>", lambda e:
(self._grid_enabled.get() and self._draw_grid()))
```

El panel lateral (*side panel*) presenta categorías con estilo de acordeón (basic, advanced, sensors, actuators, power, logic), las cuales se crean como encabezados y cuerpos utilizando _add_accordion_section. También se añade un logotipo de pie de página en la parte inferior. Las funciones del acordeón se controlan utilizando def _add_accordion_section. El estado de activado/desactivado (encendido/apagado) se controla mediante un indicador (*"Indicator"*).

```
        # Side panel with categorized components (accordion)
        self.side_panel = ttk.Frame(self.main, width=260)
        self.main.add(self.side_panel, weight=1)


        # Make accordion responsive to width changes
        self.side_canvas.bind(
            "<Configure>",

        # Build stacked, collapsible categories
        self._accordion_sections = []
        self._add_accordion_section(self._t("basic"), expanded=True)
        self._add_accordion_section(self._t("advanced"), expanded=False)

        # Footer logo image bottom-right
        footer = ttk.Frame(self.side_panel)
        footer.pack(side="bottom", fill="x", padx=10, pady=8)
        self._footer_logo_img = None
def _add_accordion_section(self, title, expanded=False):

        # Header
        header_frame = ttk.Frame(self.accordion)
        header_frame.pack(fill="x", expand=False, padx=10, pady=(6, 0))

        # Toggle indicator
        indicator = tk.StringVar(value="-" if expanded else "+")

        # Body
        body_frame = ttk.Frame(self.accordion)
        if expanded:
            body_frame.pack(after=header_frame, fill="x", expand=False,
padx=10, pady=(2, 6))

        # Empty state content
        empty = ttk.Label(body_frame,
text=self._t("components_empty").format(category=title),
foreground="gray")
        empty.pack(fill="x", expand=False, padx=6, pady=8)

        # Toggle logic
        def toggle():
            if body_frame.winfo_manager():
                body_frame.forget()
                indicator.set("+")
            else:
                body_frame.pack(after=header_frame, fill="x",
expand=False, padx=10, pady=(2, 6))
```

```
        self._accordion_sections.append((header_frame, body_frame,
indicator))
```

Las funciones de cuadrícula (*Grid*) y vista (*View*) se implementan de la siguiente manera.

```
    def _toggle_grid(self):
        if self._grid_enabled.get():
            self._draw_grid()
        else:
            self._clear_grid()
```

Los comandos de menú (*Menu Commands*) y métodos auxiliares (*Helper Methods*) se definen por separado.

* Operaciones de archivo (marcadores de posición / *placeholders*): `_new_file`, `_open_file`, `_save_file`, `_save_as_file`

* Operaciones de edición (marcadores de posición / *placeholders*): `_undo`, `_redo`, `_cut`, `_copy`, `_paste`. –

* Ajustes (*Settings*): _show_preferences, _change_language, casilla de verificación (*checkbutton*) para alternar la cuadrícula (*Grid*).

* Acerca de (*About Us*): _show_about → muestra una ventana de información sobre la aplicación.

```
def _new_file(self):
        self._status(self._t("new_file_clicked"))
        messagebox.showinfo(self._t("info"), self._t("new_file_info"))

    def _open_file(self):
        self._status(self._t("open_file_clicked"))
        messagebox.showinfo(self._t("info"), self._t("open_file_info"))

    def _save_file(self):
        self._status(self._t("save_file_clicked"))
        messagebox.showinfo(self._t("info"), self._t("save_file_info"))

    def _undo(self):
        self._status(self._t("undo_clicked"))
        messagebox.showinfo(self._t("info"), self._t("undo_info"))

    def _redo(self):
        self._status(self._t("redo_clicked"))
        messagebox.showinfo(self._t("info"), self._t("redo_info"))
```

```
    def _show_preferences(self):
        self._status(self._t("preferences_clicked"))
        messagebox.showinfo(self._t("preferences"),
self._t("preferences_info"))
```

9.3 Multilingüismo

El software está estructurado sobre la base del multilingüismo (*multilingualism*).

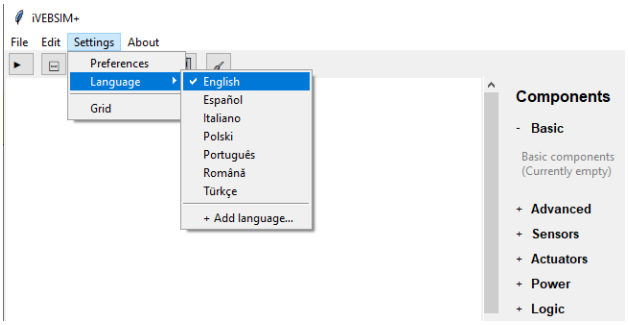


Figura 16 – Multilingüismo

Se utiliza la librería i18n de Python para habilitar el soporte multilingüe en el programa. Cuando se inicia la aplicación (en el método __init__), el código de idioma predeterminado se establece en "en" (inglés).

```
# i18n setup
        self.lang_code = "en"
        self.translations = self._load_translations(self.lang_code)
```

Los archivos de idioma se almacenan en la carpeta *languages* en formato JSON (por ejemplo, en.json, tr.json, es.json, etc.). Cada archivo JSON tiene una estructura con campos obligatorios como "code" (código de idioma) y "name" (nombre de visualización). El archivo config.json almacena el idioma seleccionado: {"selected_language": "en"}. Esto se hace para que la aplicación recuerde el último idioma seleccionado cuando se vuelva a abrir.

Figura 17 – json archivos

La Función de Carga de Traducción (_load_translations) toma un código de idioma como parámetro (por ejemplo, "tr") y abre el archivo languages/{code}.json para cargar el JSON.

```
def _load_translations(self, code: str) -> dict:
        path = os.path.join(self._languages_dir(), f"{code}.json")
        try:
            with open(path, "r", encoding="utf-8") as f:
                data = json.load(f)
                if isinstance(data, dict):
                    return data
        except Exception:
            pass
```

El menú de Ajustes (*Settings*) tiene un submenú llamado "Language". La función _populate_language_menu() escanea todos los archivos JSON en la carpeta *languages*. Extrae los campos "code" y "name" de cada archivo. Cuando el usuario selecciona un idioma, se llama a la función _on_language_selected() y se actualiza la variable lang_code.

```
def _populate_language_menu(self):
        try:
            self.menu_language.delete(0, "end")
        except Exception:
            return
def _on_language_selected(self):
        selected = self._language_var.get()
        if selected == getattr(self, "lang_code", None):
            return
        self.lang_code = selected
        self.translations = self._load_translations(self.lang_code)
        self._save_selected_language(self.lang_code)
        self._rebuild_ui()
```

Después de seleccionar un idioma, la interfaz gráfica de usuario (*UI*) se reconstruye por completo. Cambiar el idioma requiere destruir y volver a crear todos los *widgets*. El texto en *Tkinter* no cambia de forma dinámica. Por lo tanto, se elimina la barra de menú y toda la interfaz gráfica de usuario (*UI*) se reconstruye desde cero.

```
def _rebuild_ui(self):
        try:
            self.master.config(menu=None)
        except Exception:
            pass
        for child in self.winfo_children():
            try:
                child.destroy()
```

```
            except Exception:
                pass
        self._grid_lines = []
        self._accordion_sections = []
        self._build_ui()
```

9. AÑADIR ELEMENTOS Y CREAR LA SIMULACIÓN

PRÓXIMA RUMANIA……

CODES

MAIN.PY

```
import os
import sys
import tkinter as tk
from PIL import Image, ImageTk

from ui import App


def resource_path(*parts):
    """Get absolute path to resource, works for dev and for PyInstaller
onefile."""
    base_path = getattr(sys, "_MEIPASS",
os.path.dirname(os.path.abspath(__file__)))
    return os.path.join(base_path, *parts)


def show_splash(root):
    splash = tk.Toplevel(root)
    splash.overrideredirect(True)

    # Initial size
    width, height = 460, 300
    splash.update_idletasks()
    sw = splash.winfo_screenwidth()
    sh = splash.winfo_screenheight()
    x = (sw - width) // 2
    y = (sh - height) // 2
    splash.geometry(f"{width}x{height}+{x}+{y}")

    container = tk.Frame(splash, bg="#ffffff", bd=1, relief="solid")
    container.pack(fill="both", expand=True)

    # Logo image
    logo_path = resource_path("sembol", "logoarkaplan.png")
    logo_img = None
    try:
        if os.path.exists(logo_path):
            im = Image.open(logo_path)
            # Fit into splash width with some margin
            target_w = 280
            ratio = target_w / max(1, im.width)
            target_h = int(im.height * ratio)
            im = im.resize((target_w, target_h))
            logo_img = ImageTk.PhotoImage(im)
    except Exception:
```

```
        logo_img = None
    if logo_img is not None:
        logo = tk.Label(container, image=logo_img, bg="#ffffff")
        logo.image = logo_img
        logo.pack(pady=(18, 8))
    else:
        logo = tk.Label(container, text="⚡", font=("Segoe UI", 56),
bg="#ffffff", fg="#1a73e8")
        logo.pack(pady=(22, 6))

    title = tk.Label(container, text="iVEBSIM+", font=("Segoe UI", 16,
"bold"), bg="#ffffff", fg="#202124")
    title.pack()

    subtitle = tk.Label(container, text="UI Framework - Version 1.2",
font=("Segoe UI", 11), bg="#ffffff", fg="#5f6368")
    subtitle.pack(pady=(2, 12))

    about = tk.Label(
        container,
        text=(
            "Basit arayüz önizlemesi\n"
            "Bileşen kategorileri (şimdilik boş)\n"
            "Devam etmek için herhangi bir yere tıklayın"
        ),
        justify="center",
        font=("Segoe UI", 10),
        bg="#ffffff",
        fg="#5f6368",
    )
    about.pack(pady=(0, 16))

    # Recenter after layout so size is accurate
    splash.update_idletasks()
    width = splash.winfo_width()
    height = splash.winfo_height()
    sw = splash.winfo_screenwidth()
    sh = splash.winfo_screenheight()
    x = (sw - width) // 2
    y = (sh - height) // 2
    splash.geometry(f"{width}x{height}+{x}+{y}")

    def proceed(event=None):
        try:
            splash.destroy()
        except Exception:
            pass
        root.deiconify()
```

```
        root.geometry("1100x780")
        App(root)
        root.focus_force()

    # Close on any click or key
    for widget in (splash, container, logo, title, subtitle, about):
        widget.bind("<Button-1>", proceed)
        widget.bind("<Key>", proceed)

    # Make sure splash is above
    splash.attributes("-topmost", True)
    splash.after(50, lambda: splash.attributes("-topmost", False))


def run():
    root = tk.Tk()
    root.withdraw()
    show_splash(root)
    root.mainloop()


if __name__ == "__main__":
    run()
```

UI.PY

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
from PIL import Image, ImageTk
import os
import sys
import json
import shutil
from core import SimulationCore, DEFAULT_WIDTH, DEFAULT_HEIGHT
class App(tk.Frame):
    def __init__(self, master, symbols_dir=None):
        super().__init__(master)
        self.master = master
        self.symbols_dir = symbols_dir  # Not used in this version
        self.core = SimulationCore()
        self.dragging_component_key = None
        self.ghost_item = None
        self.comp_id_to_canvas = {}
        self.comp_id_to_text = {}
        self.comp_id_to_ports = {}
        self.cable_id_to_lines = {}
        self._anim_tick = 0
        self._cable_wave = 0
        self._cable_wave_step = 2
        self._tick_interval_ms = 120

        # Cable editing
        self._dragging_cable = None
        self._dragging_segment = None
        self._drag_start_x = 0
        self._drag_start_y = 0

        # i18n setup
        self.lang_code = "en"
        self.translations = self._load_translations(self.lang_code)

        self._build_ui()
        self._status(self._t("ready"))

    # Resource helper
    def _resource_path(self, *parts) -> str:
        base_path = getattr(sys, "_MEIPASS",
os.path.dirname(os.path.abspath(__file__)))
        return os.path.join(base_path, *parts)

    # UI building
    def _build_ui(self):
```

```
        self.master.title("iVEBSIM+")
        self.grid(row=0, column=0, sticky="nsew")
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0, weight=1)

        # Menu bar
        self.menubar = tk.Menu(self.master)

        # File menu
        self.menu_file = tk.Menu(self.menubar, tearoff=0)
        self.menu_file.add_command(label=self._t("new"),
command=self._new_file)
        self.menu_file.add_command(label=self._t("open"),
command=self._open_file)
        self.menu_file.add_command(label=self._t("save"),
command=self._save_file)
        self.menu_file.add_command(label=self._t("save_as"),
command=self._save_as_file)
        self.menu_file.add_separator()
        self.menu_file.add_command(label=self._t("exit"),
command=self._exit_app)
        self.menubar.add_cascade(label=self._t("file"), menu=self.menu_file)

        # Edit menu
        self.menu_edit = tk.Menu(self.menubar, tearoff=0)
        self.menu_edit.add_command(label=self._t("undo"), command=self._undo)
        self.menu_edit.add_command(label=self._t("redo"), command=self._redo)
        self.menu_edit.add_separator()
        self.menu_edit.add_command(label=self._t("cut"), command=self._cut)
        self.menu_edit.add_command(label=self._t("copy"), command=self._copy)
        self.menu_edit.add_command(label=self._t("paste"),
command=self._paste)
        self.menubar.add_cascade(label=self._t("edit"), menu=self.menu_edit)

        # Settings menu
        self.menu_settings = tk.Menu(self.menubar, tearoff=0)
        self.menu_settings.add_command(label=self._t("preferences"),
command=self._show_preferences)
        # Language submenu
        self.menu_language = tk.Menu(self.menu_settings, tearoff=0)
        self._language_var = tk.StringVar(value=self.lang_code)
        self._populate_language_menu()
        self.menu_settings.add_cascade(label=self._t("language"),
menu=self.menu_language)
        self.menu_settings.add_separator()
        self._grid_enabled = tk.BooleanVar(value=False)
        self.menu_settings.add_checkbutton(label=self._t("grid"),
variable=self._grid_enabled, command=self._toggle_grid)
```

```
        self.menubar.add_cascade(label=self._t("settings"),
menu=self.menu_settings)

        # About menu
        self.menu_about = tk.Menu(self.menubar, tearoff=0)
        self.menu_about.add_command(label=self._t("about"),
command=self._show_about)
        self.menubar.add_cascade(label=self._t("about"), menu=self.menu_about)

        self.master.config(menu=self.menubar)

        self.toolbar = ttk.Frame(self)
        self.toolbar.grid(row=0, column=0, sticky="ew")

        self.btn_start = ttk.Button(self.toolbar, text="▶", width=3,
command=self.start_sim)
        self.btn_start.pack(side="left", padx=4)
        # Icon-only file action buttons next to Start
        self.btn_new = ttk.Button(self.toolbar, text="🆕", width=3,
command=self._new_file)
        self.btn_new.pack(side="left", padx=2)
        self.btn_open = ttk.Button(self.toolbar, text="📂", width=3,
command=self._open_file)
        self.btn_open.pack(side="left", padx=2)
        self.btn_save = ttk.Button(self.toolbar, text="💾", width=3,
command=self._save_file)
        self.btn_save.pack(side="left", padx=2)
        self.btn_print = ttk.Button(self.toolbar, text="🖨", width=3,
command=self._print_canvas)
        self.btn_print.pack(side="left", padx=(2, 8))
        self.btn_stop = ttk.Button(self.toolbar, text="⏹", width=3,
command=self.stop_sim)
        self.btn_stop.pack(side="left", padx=4)
        self.btn_clear = ttk.Button(self.toolbar, text="🧹", width=3,
command=self.clear)
        self.btn_clear.pack(side="left", padx=4)

        self.main = ttk.PanedWindow(self, orient=tk.HORIZONTAL)
        self.main.grid(row=1, column=0, sticky="nsew")
        self.rowconfigure(1, weight=1)
        self.columnconfigure(0, weight=1)

        # Canvas area
        self.canvas_frame = ttk.Frame(self.main)
        self.main.add(self.canvas_frame, weight=3)
        self.canvas_frame.rowconfigure(0, weight=1)
        self.canvas_frame.columnconfigure(0, weight=1)
```

```
        self.canvas = tk.Canvas(self.canvas_frame, bg="#ffffff",
scrollregion=(0, 0, 2000, 2000))
        self.canvas.grid(row=0, column=0, sticky="nsew")
        self.h_scroll = ttk.Scrollbar(self.canvas_frame, orient="horizontal",
command=self.canvas.xview)
        self.h_scroll.grid(row=1, column=0, sticky="ew")
        self.v_scroll = ttk.Scrollbar(self.canvas_frame, orient="vertical",
command=self.canvas.yview)
        self.v_scroll.grid(row=0, column=1, sticky="ns")
        self.canvas.configure(xscrollcommand=self.h_scroll.set,
yscrollcommand=self.v_scroll.set)
        self._grid_lines = []
        # Redraw grid on resize/scroll when enabled
        self.canvas.bind("<Configure>", lambda e: (self._grid_enabled.get()
and self._draw_grid()))
        self.canvas.bind_all("<MouseWheel>", lambda e:
(self._grid_enabled.get() and self._draw_grid()))

        # Side panel with categorized components (accordion)
        self.side_panel = ttk.Frame(self.main, width=260)
        self.main.add(self.side_panel, weight=1)

        # Components title
        components_label = ttk.Label(self.side_panel,
text=self._t("components"), font=("Arial", 12, "bold"))
        components_label.pack(pady=(10, 5), padx=10, anchor="w")

        # Scrollable container for accordion
        self.side_canvas = tk.Canvas(self.side_panel, highlightthickness=0)
        self.side_scrollbar = ttk.Scrollbar(self.side_panel,
orient="vertical", command=self.side_canvas.yview)
        self.side_canvas.configure(yscrollcommand=self.side_scrollbar.set)
        self.side_canvas.pack(side="left", fill="both", expand=True)
        self.side_scrollbar.pack(side="right", fill="y")

        self.accordion = ttk.Frame(self.side_canvas)
        self._accordion_window = self.side_canvas.create_window((0, 0),
window=self.accordion, anchor="nw")

        # Make accordion responsive to width changes
        self.side_canvas.bind(
            "<Configure>",
            lambda e: self.side_canvas.itemconfigure(self._accordion_window,
width=e.width)
        )
        self.accordion.bind(
            "<Configure>",
```

```
            lambda e:
self.side_canvas.configure(scrollregion=self.side_canvas.bbox("all"))
        )

        # Build stacked, collapsible categories
        self._accordion_sections = []
        self._add_accordion_section(self._t("basic"), expanded=True)
        self._add_accordion_section(self._t("advanced"), expanded=False)
        self._add_accordion_section(self._t("sensors"), expanded=False)
        self._add_accordion_section(self._t("actuators"), expanded=False)
        self._add_accordion_section(self._t("power"), expanded=False)
        self._add_accordion_section(self._t("logic"), expanded=False)

        # Footer logo image bottom-right
        footer = ttk.Frame(self.side_panel)
        footer.pack(side="bottom", fill="x", padx=10, pady=8)
        self._footer_logo_img = None
        try:
            logo_path = self._resource_path("sembol", "logoarkaplan.png")
            if os.path.exists(logo_path):
                im = Image.open(logo_path)
                # scale to side panel width approx
                target_w = 120
                ratio = target_w / max(1, im.width)
                target_h = int(im.height * ratio)
                im = im.resize((target_w, target_h))
                self._footer_logo_img = ImageTk.PhotoImage(im)
        except Exception:
            self._footer_logo_img = None
        if self._footer_logo_img is not None:
            footer_logo = ttk.Label(footer, image=self._footer_logo_img,
anchor="se")
        else:
            footer_logo = ttk.Label(footer, text="⚡", anchor="se",
font=("Segoe UI", 16))
        footer_logo.pack(anchor="se")

        # Bindings
        self.canvas.tag_bind("port", "<ButtonPress-1>", self._on_port_press)
        self.canvas.tag_bind("port", "<ButtonRelease-1>",
self._on_port_release)
        self.canvas.tag_bind("component", "<ButtonPress-1>",
self._on_component_press)
        self.canvas.tag_bind("component", "<B1-Motion>",
self._on_component_motion)
        self.canvas.tag_bind("component", "<ButtonRelease-1>",
self._on_component_release)
```

```
        self.canvas.tag_bind("component", "<Button-3>",
self._on_component_right_click)
        self.canvas.tag_bind("cable", "<ButtonPress-1>", self._on_cable_press)
        self.canvas.tag_bind("cable", "<B1-Motion>", self._on_cable_motion)
        self.canvas.tag_bind("cable", "<ButtonRelease-1>",
self._on_cable_release)
        self.canvas.tag_bind("cable", "<Button-3>",
self._on_cable_right_click)

        self.status = tk.StringVar(value="")
        ttk.Label(self, textvariable=self.status).grid(row=2, column=0,
sticky="ew")
        self.tip = ttk.Label(self, text="", foreground="#666")
        self.tip.grid(row=3, column=0, sticky="ew")


    def _create_component_section(self, parent_frame, category_name):
        """Create an empty component section for the given category"""
        # Create a scrollable frame for components
        canvas = tk.Canvas(parent_frame, height=300)
        scrollbar = ttk.Scrollbar(parent_frame, orient="vertical",
command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        # Empty placeholder for now
        empty_label = ttk.Label(
            scrollable_frame,
            text=self._t("components_empty").format(category=category_name),
            font=("Arial", 10),
            foreground="gray",
        )
        empty_label.pack(pady=20, padx=10)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

    def _add_accordion_section(self, title, expanded=False):
        """Create one collapsible accordion section with empty placeholder
list."""
        # Header
```

```
        header_frame = ttk.Frame(self.accordion)
        header_frame.pack(fill="x", expand=False, padx=10, pady=(6, 0))

        # Toggle indicator
        indicator = tk.StringVar(value="-" if expanded else "+")
        indicator_label = ttk.Label(header_frame, textvariable=indicator,
width=2)
        indicator_label.pack(side="left")

        title_label = ttk.Label(header_frame, text=title, font=("Arial", 10,
"bold"))
        title_label.pack(side="left", anchor="w")

        # Body
        body_frame = ttk.Frame(self.accordion)
        if expanded:
            body_frame.pack(after=header_frame, fill="x", expand=False,
padx=10, pady=(2, 6))

        # Empty state content
        empty = ttk.Label(body_frame,
text=self._t("components_empty").format(category=title), foreground="gray")
        empty.pack(fill="x", expand=False, padx=6, pady=8)

        # Toggle logic
        def toggle():
            if body_frame.winfo_manager():
                body_frame.forget()
                indicator.set("+")
            else:
                body_frame.pack(after=header_frame, fill="x", expand=False,
padx=10, pady=(2, 6))
                indicator.set("-")

        header_frame.bind("<Button-1>", lambda e: toggle())
        title_label.bind("<Button-1>", lambda e: toggle())
        indicator_label.bind("<Button-1>", lambda e: toggle())

        self._accordion_sections.append((header_frame, body_frame, indicator))

    def _show_about(self):
        message = self._t("about_message")
        messagebox.showinfo(self._t("about"), message)

    def _status(self, txt):
        self.status.set(txt)
        self.update_idletasks()
```

```
    # Simulation controls
    def start_sim(self):
        self.core.start()
        self._status(self._t("simulation_started"))

    def stop_sim(self):
        self.core.stop()
        self._status(self._t("simulation_stopped"))

    def clear(self):
        self.core.components.clear()
        self.core.connections.clear()
        self.canvas.delete("all")
        self._grid_lines.clear()
        if self._grid_enabled.get():
            self._draw_grid()
        self._status(self._t("canvas_cleared"))

    # Component interactions (placeholder)
    def _on_component_press(self, event):
        self._status(self._t("component_pressed"))

    def _on_component_motion(self, event):
        pass

    def _on_component_release(self, event):
        pass

    def _on_component_right_click(self, event):
        self._status("Component right-clicked")

    def _on_port_press(self, event):
        self._status(self._t("port_pressed"))

    def _on_port_release(self, event):
        self._status(self._t("port_released"))

    def _on_cable_press(self, event):
        self._status(self._t("cable_pressed"))

    def _on_cable_motion(self, event):
        pass

    def _on_cable_release(self, event):
        pass

    def _on_cable_right_click(self, event):
        self._status(self._t("cable_right_clicked"))
```

```
    # Menu command methods
    def _toggle_grid(self):
        if self._grid_enabled.get():
            self._draw_grid()
        else:
            self._clear_grid()

    def _clear_grid(self):
        for line_id in self._grid_lines:
            self.canvas.delete(line_id)
        self._grid_lines.clear()

    def _draw_grid(self, spacing: int = 20, color: str = "#eef3f8"):
        self._clear_grid()
        # Get current visible region
        x0 = int(self.canvas.canvasx(0))
        y0 = int(self.canvas.canvasy(0))
        x1 = int(self.canvas.canvasx(self.canvas.winfo_width()))
        y1 = int(self.canvas.canvasy(self.canvas.winfo_height()))
        # Snap start to spacing
        start_x = (x0 // spacing) * spacing
        start_y = (y0 // spacing) * spacing
        for x in range(start_x, x1 + 1, spacing):
            line = self.canvas.create_line(x, y0, x, y1, fill=color)
            self._grid_lines.append(line)
        for y in range(start_y, y1 + 1, spacing):
            line = self.canvas.create_line(x0, y, x1, y, fill=color)
            self._grid_lines.append(line)
        # Send grid to back
        for line_id in self._grid_lines:
            self.canvas.tag_lower(line_id)
    def _new_file(self):
        self._status(self._t("new_file_clicked"))
        messagebox.showinfo(self._t("info"), self._t("new_file_info"))

    def _open_file(self):
        self._status(self._t("open_file_clicked"))
        messagebox.showinfo(self._t("info"), self._t("open_file_info"))

    def _save_file(self):
        self._status(self._t("save_file_clicked"))
        messagebox.showinfo(self._t("info"), self._t("save_file_info"))

    def _save_as_file(self):
        self._status(self._t("save_as_clicked"))
        messagebox.showinfo(self._t("info"), self._t("save_as_info"))
```

```
    def _exit_app(self):
        self._status(self._t("exit_application"))
        self.master.quit()

    def _undo(self):
        self._status(self._t("undo_clicked"))
        messagebox.showinfo(self._t("info"), self._t("undo_info"))

    def _redo(self):
        self._status(self._t("redo_clicked"))
        messagebox.showinfo(self._t("info"), self._t("redo_info"))

    def _cut(self):
        self._status(self._t("cut_clicked"))
        messagebox.showinfo(self._t("info"), self._t("cut_info"))

    def _copy(self):
        self._status(self._t("copy_clicked"))
        messagebox.showinfo(self._t("info"), self._t("copy_info"))

    def _paste(self):
        self._status(self._t("paste_clicked"))
        messagebox.showinfo(self._t("info"), self._t("paste_info"))

    def _show_preferences(self):
        self._status(self._t("preferences_clicked"))
        messagebox.showinfo(self._t("preferences"),
self._t("preferences_info"))

    # i18n helpers
    def _languages_dir(self) -> str:
        return self._resource_path("languages")

    def _config_path(self) -> str:
        return os.path.join(self._languages_dir(), "config.json")

    def _ensure_default_languages(self) -> None:
        os.makedirs(self._languages_dir(), exist_ok=True)
        defaults = {
            # Only ensure files exist by copying from languages directory; do
not embed strings here
            # This method will just make sure base files exist by copying our
bundled JSONs if needed.
        }
        # We already added JSON files into the languages directory via the
project; nothing to generate here.
        # Ensure config exists
        if not os.path.exists(self._config_path()):
```

```
            try:
                with open(self._config_path(), "w", encoding="utf-8") as f:
                    json.dump({"selected_language": "tr"}, f,
ensure_ascii=False, indent=2)
            except Exception:
                pass

    def _load_saved_language(self) -> str:
        # Ensure defaults present on first run
        self._ensure_default_languages()
        try:
            with open(self._config_path(), "r", encoding="utf-8") as f:
                data = json.load(f)
                code = data.get("selected_language")
                if isinstance(code, str) and code.strip():
                    return code
        except Exception:
            pass
        return "tr"

    def _save_selected_language(self, code: str) -> None:
        try:
            os.makedirs(self._languages_dir(), exist_ok=True)
            with open(self._config_path(), "w", encoding="utf-8") as f:
                json.dump({"selected_language": code}, f, ensure_ascii=False,
indent=2)
        except Exception:
            pass

    def _load_translations(self, code: str) -> dict:
        path = os.path.join(self._languages_dir(), f"{code}.json")
        try:
            with open(path, "r", encoding="utf-8") as f:
                data = json.load(f)
                if isinstance(data, dict):
                    return data
        except Exception:
            pass
        # Fallback to Turkish file
        try:
            with open(os.path.join(self._languages_dir(), "tr.json"), "r",
encoding="utf-8") as f:
                data = json.load(f)
                if isinstance(data, dict):
                    return data
        except Exception:
            pass
        # Last resort: minimal inline fallback
```

```
        return {"code": "tr", "name": "Türkçe", "title": "Elektrik Devre
Simülasyonu"}

    def _t(self, key: str) -> str:
        return str(self.translations.get(key, key))

    def _list_available_languages(self):
        langs = []  # list of (code, name)
        try:
            os.makedirs(self._languages_dir(), exist_ok=True)
            for fname in os.listdir(self._languages_dir()):
                if not fname.lower().endswith(".json"):
                    continue
                fpath = os.path.join(self._languages_dir(), fname)
                try:
                    with open(fpath, "r", encoding="utf-8") as f:
                        data = json.load(f)
                    code = data.get("code") or os.path.splitext(fname)[0]
                    name = data.get("name") or code
                    if code != "config":
                        langs.append((code, name))
                except Exception:
                    continue
        except Exception:
            pass
        # Ensure unique by code
        seen = set()
        unique = []
        for code, name in langs:
            if code in seen:
                continue
            seen.add(code)
            unique.append((code, name))
        unique.sort(key=lambda x: x[1].lower())
        return unique

    def _populate_language_menu(self):
        try:
            self.menu_language.delete(0, "end")
        except Exception:
            return
        for code, name in self._list_available_languages():
            self.menu_language.add_radiobutton(
                label=name,
                variable=self._language_var,
                value=code,
                command=self._on_language_selected,
            )
```

```
        self.menu_language.add_separator()
        self.menu_language.add_command(label="+ Add language…",
command=self._import_language)

    def _on_language_selected(self):
        selected = self._language_var.get()
        if selected == getattr(self, "lang_code", None):
            return
        self.lang_code = selected
        self.translations = self._load_translations(self.lang_code)
        self._save_selected_language(self.lang_code)
        self._rebuild_ui()

    def _import_language(self):
        self._status(self._t("import_language_clicked"))
        src = filedialog.askopenfilename(
            title=self._t("select_language_json"),
            filetypes=[("JSON files", "*.json")],
        )
        if not src:
            return
        try:
            with open(src, "r", encoding="utf-8") as f:
                data = json.load(f)
            code = data.get("code")
            name = data.get("name")
            if not code or not isinstance(code, str):
                messagebox.showerror(self._t("error"),
self._t("invalid_language_file"))
                return
            os.makedirs(self._languages_dir(), exist_ok=True)
            dst = os.path.join(self._languages_dir(), f"{code}.json")
            shutil.copyfile(src, dst)
            self._populate_language_menu()
            messagebox.showinfo(self._t("info"),
self._t("language_imported").format(name=name or code))
        except Exception as e:
            messagebox.showerror(self._t("error"), str(e))

    def _rebuild_ui(self):
        try:
            self.master.config(menu=None)
        except Exception:
            pass
        for child in self.winfo_children():
            try:
                child.destroy()
            except Exception:
```

```
                pass
        self._grid_lines = []
        self._accordion_sections = []
        self._build_ui()

    def _change_language(self):
        self._status("Change language clicked")
        messagebox.showinfo("Dil", "Dil değiştirme işlemi")

    def _print_canvas(self):
        fname = filedialog.asksaveasfilename(defaultextension=".ps",
filetypes=[("PostScript", "*.ps")])
        if not fname:
            return
        try:
            self.canvas.postscript(file=fname, colormode='color')
            self._status(self._t("canvas_exported"))
        except Exception as e:
            messagebox.showerror(self._t("error"), str(e))
```

NOTAS:

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

NOTAS:

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union