Izmir,
Pescara
2026

Open-Source Simulator Software iVEBSIM+

PROGRAMMING LANGUAGE

SIMULATOR SOFTWARE

ARTIFICIAL INTELLIGENCE ADD-ON

USING THE SIMULATOR

PRACTICE EXERCISES

KA220VET Erasmus+ Project

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

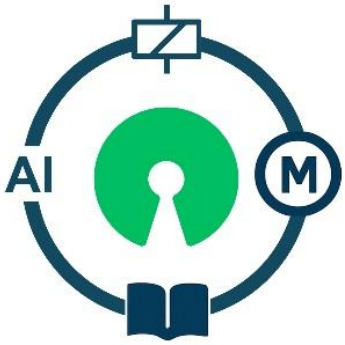
Funded by
the European Union

"Supported by the European Commission under the Erasmus+ Program. The content here reflects the views of the authors, and the European Commission and the Turkish National Agency cannot be held responsible for these views."

Funded by
the European Union

INDEX

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union



FOREWORD

Educational systems are being reshaped by the impact of digital transformation; especially in the field of vocational and technical education, the need for innovative, flexible, and accessible learning environments is increasing every day.

This project, titled “Developing Vocational Education with an AI-Supported Open Source Simulator” and carried out within the scope of Erasmus+ KA220-VET cooperation partnerships, aims to produce a sustainable and innovative response to this need.

This multi-national study, carried out under the coordination of Türkiye with the partnership of Poland, Spain, Italy, Portugal, and Romania, aims to strengthen vocational education with digital tools and make teaching processes more interactive and application-based.

The main output of the project, the iVEBSIM+ simulator, is an entirely open-source, AI-supported software designed for the field of industrial electronics and control. In this respect, the project does not only produce educational material; it also aims to create a shareable, improvable, and sustainable digital learning ecosystem.

This book you are holding has been designed as one of the outputs of the project. In the first part of the book, theoretical and practical information regarding Python, which is used as the basic programming language in the development of the simulator software, is presented. Topics such as programming fundamentals, variable structures, and decision and loop mechanisms are handled with a clarity that vocational education students can understand and with an application-oriented approach.

In the following sections, the following topics are discussed in detail:

- The theoretical background of the simulator software
- Creating the graphical user interface
- Integration of industrial electronic components into the digital environment
- Development and use of the artificial intelligence add-on
- Application exercises prepared for use in vocational education
- Pilot application processes carried out with students

The project process is structured based on the completion of a specific technical stage in each mobility. All stages, from creating Python fundamentals to developing the simulator core, from expanding the industrial component library to artificial intelligence integration and preparing pedagogical exercises, have been carried out within international cooperation.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Thus, both technical capacity has been improved and the sharing of best practices between countries has been ensured.

One of the most important features of this study is that the developed simulator is **open source**. All source codes of the iVEBSIM+ Simulator are shared with the public through the project website, offering a structure open to the contributions of educators, students, and developers. This approach is based on the principles of transparency, accessibility, and collective production, and it supports digital equality in vocational education. Furthermore, sharing the development stages of the software in detail within the book encourages learners to become developers rather than just users.

Simulation-based learning in vocational education is of great importance as it provides a safe experimental area, reduces costs, and allows for repeatable application opportunities. **Artificial Intelligence** support makes the learning process more efficient and interactive through possibilities such as individualized feedback, error analysis, and smart guidance. This book and the accompanying software aim to contribute to the acquisition of digital competencies required by the age by integrating these two approaches.

This work has been prepared with the hope that it will be a guide for teachers working in vocational and technical education institutions, students studying in the field, software developers, and researchers. Our greatest wish is for this study, which emerged from the knowledge and experience of an international partnership, to spread in line with the understanding of open science and open educational resources.

This project is supported by the European Union, and the views expressed in the content belong to their authors.

We would like to thank all project partners who contributed and the educators who put in the effort.

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

"Supported by the European Commission under the Erasmus+ Program. The content here reflects the views of the authors, and the European Commission and the Turkish National Agency cannot be held responsible for these views."

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

Funded by the European Union

PYTHON PROGRAMMING LANGUAGE

1. THE IMPORTANCE OF PYTHON PROGRAMMING LANGUAGE

Python is one of the most widely used programming languages today. It was developed by Guido van Rossum and released in 1991. It is a high-level language with wide library support where you can develop applications in almost every field.

Why Python?

- It is easy to learn because it has a simple syntax.
- It is similar to the English language.
- It uses new lines to complete a command instead of semicolons and parentheses.
- It uses indentation (space at the beginning of the line) to define the scope of loops, functions, and classes.
- It can run on all platforms such as Windows, Linux, and Mac.

Usage Areas:

- System programming
- User interface programming
- Web development
- Network programming
- Game development
- Data science, machine learning
- Data analysis
- Artificial Intelligence (AI)
- Scripting and tools

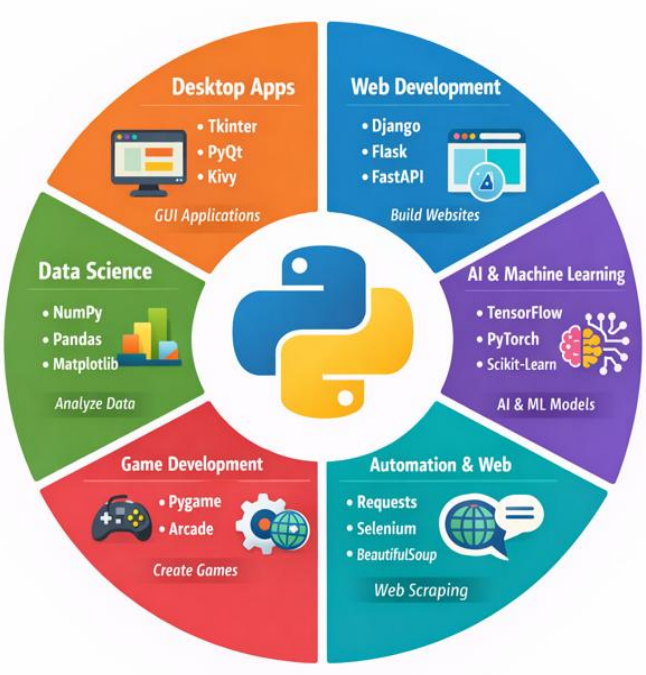


Figure 1. Python Libraries

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

2. INSTALLING THE PYTHON PROGRAMMING LANGUAGE

To write code with the Python programming language, you first need to install an editor on your computer.

2.1. Python Installation

For Windows:

1. Go to the official Python website at https://www.python.org/.
2. Click on the **Downloads** menu.
3. Download the latest version of Python (e.g., Python 3.14.0).
4. Run the downloaded installation file. **Important:** Select the **"Add python.exe to PATH"** checkbox before clicking the "Install Now" button. This step is necessary to run Python from the command line.
5. When the installation is complete and you receive the "Setup was successful" message, click the "Close" button.

Figure 2. Installation Window

For macOS:

- Download the installer with the .pkg extension from the same website and follow the standard installation steps.
- Python is installed in the Applications → Python x.x folder.

For Linux:

- Open the terminal and run the following commands in order:
- sudo apt update
- sudo apt install python3 python3-pip

2.2. Verification of the Installation

After the installation is complete, you can verify it by opening the Command Prompt (CMD) in Windows and typing the following command:

python --version

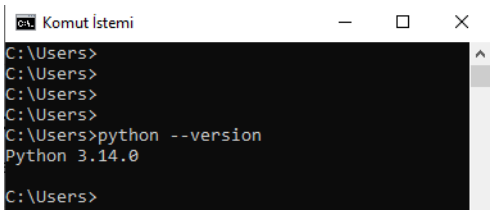


Figure 3. Installation Version

If you see the Python version number as in Figure 3, it has been installed correctly. If you don't see it, refresh the installation steps.

2.3. Checking the Package Manager

You can verify the package manager by typing the following command in the Command Prompt (CMD) or terminal: pip –version

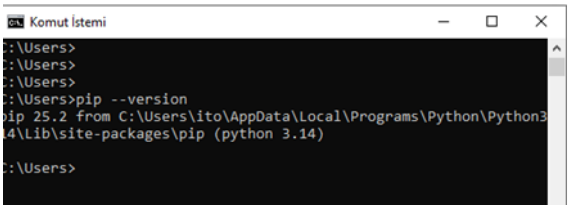


Figure 4. Package Manager Version

pip (Python Package Installer) is a tool used for library management and package installation in Python. It allows you to easily install the libraries you need for your projects.

2.4. Running the First Python Program

Python does not require an editor to write command code. Code can be executed directly using the Command Prompt. Table 1 provides an example of a code snippet that prints "Hello" to the screen. However, as program code gets bigger and library usage becomes more, this method may not be ideal for programmers

After the Python installation is complete, you can run your first program using the **IDLE (Integrated Development and Learning Environment)** that comes with Python or the Command Prompt.

Running with the Command Prompt:

1. Open the Command Prompt (CMD).
2. Type python and press Enter. This will start the Python interactive shell.
3. Type the following code and press Enter: print("Hello World")
4. You will see the output "Hello World" on the screen.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Running as a Script File:

1. Open a simple text editor (like Notepad).
2. Write the following code: print("Welcome to iVEBSIM+")
3. Save the file with the name hello.py.
4. Open the Command Prompt, navigate to the folder where you saved the file, and type: python hello.py

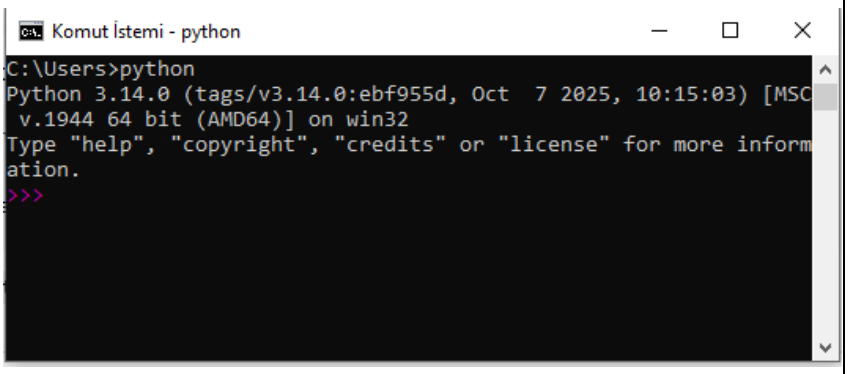
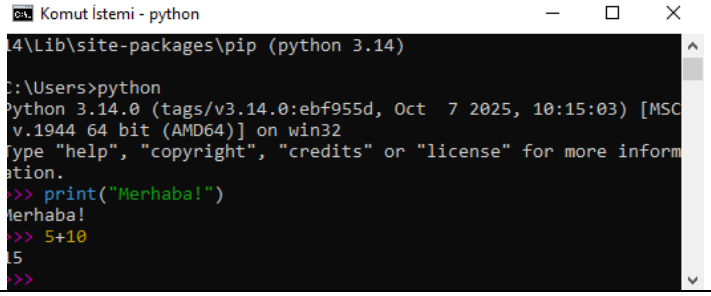
Codes:	Your screen output will look like this:
Pyhton	
Codes:	Your screen output will look like this:
print("Merhaba!") 5+10	

Table 1. Writing Programs Using the Command Prompt

3. USING AN EDITOR

You can write Python code in a simple text editor and save it with a .py extension. However, using a professional code editor makes the development process much easier and more efficient. In Figure 5, a new file named mycode.py was created.

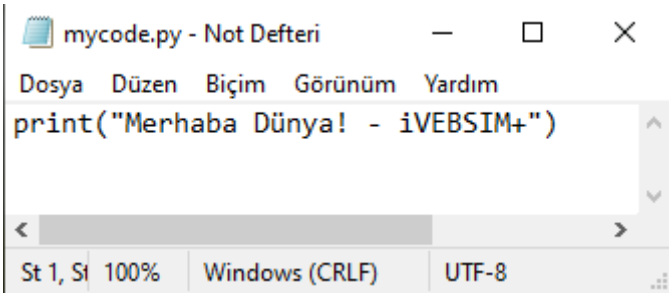


Figure 5. Using the Text Editor

When you type the command `python mycode.py` in the command prompt, the program will run and appear in your console as shown below.

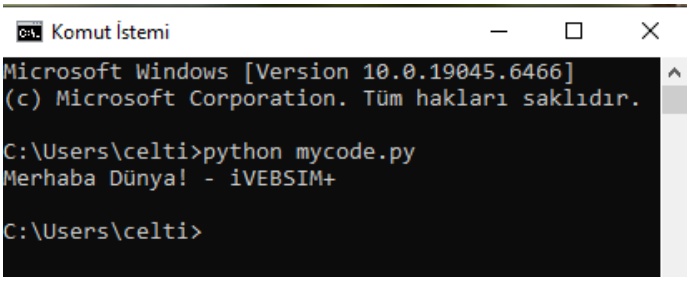


Figure 6. Console Program Output

We can also use editors to see our code more organized and write it more easily. Visual Studio Code, PhyCharm, and Sublime Text are some examples.

3.1. Visual Studio Code Installation

Visual Studio Code (VS Code) is a free, powerful, and popular code editor that works on Windows, macOS, and Linux. To install it, follow these steps:

1. Go to the official website: https://code.visualstudio.com/.
2. Click the download button for your operating system.
3. Run the downloaded installation file.
4. Follow the instructions on the installation wizard. It is recommended to select the option "Add to PATH" during the setup.

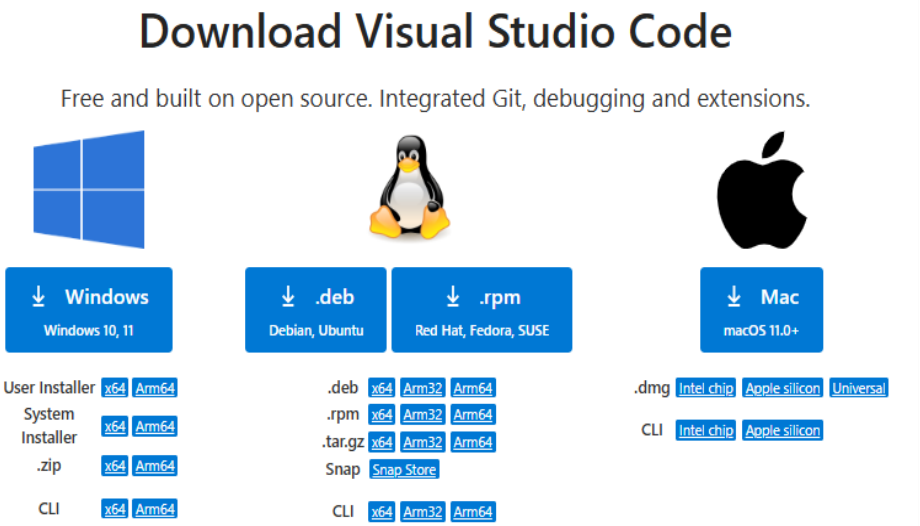


Figure 7. VS Code Download Page

Once the installation is complete, you can launch **Visual Studio Code** from your desktop or start menu.

3.2. Installing the Visual Studio Code Python Extension

To use Python features like code completion, **debugging** (hata ayıklama), and unit testing in **Visual Studio Code**, you must install the Python extension developed by Microsoft. Follow these steps:

1. Click on the **Extensions** icon ( ) in the Activity Bar on the side of **VS Code**.
2. Type "Python" in the search bar.
3. Find the extension named "Python" provided by Microsoft and click the **Install** button.

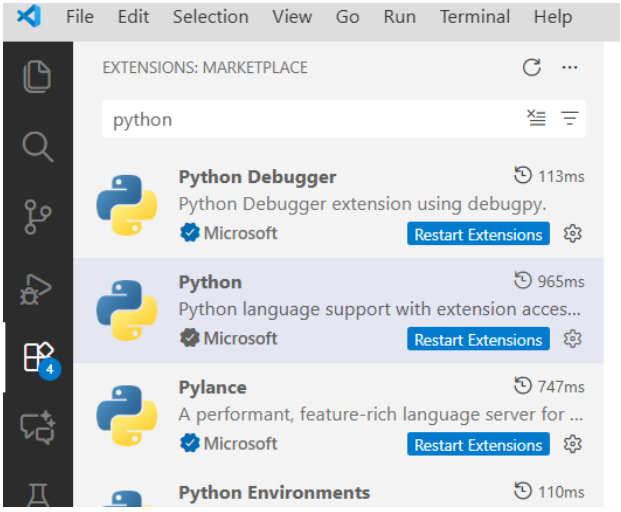


Figure 8. VS Code Python Installation

3.3. Running the First Code

After installing the extension, you can create and run your first Python file:

1. Select File > New File or press Ctrl+N.
2. Save the file with a .py extension (e.g., main.py).

3. Write the following code into the file:

   Python

   print("Hello World!")

4. Run the code by clicking the Run button (play icon) in the top right corner or by pressing F5.

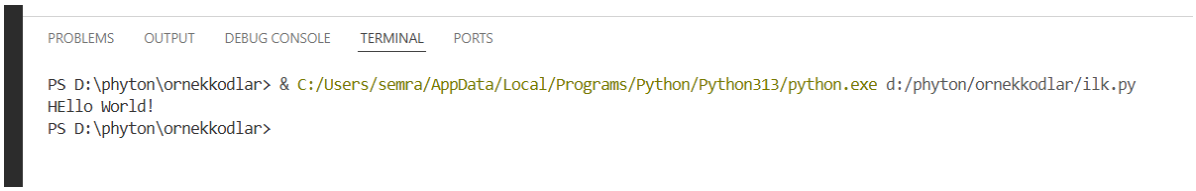


Figure 9. Screen Output

3.4. Visual Studio Code Artificial Intelligence Toolkit

The AI Toolkit is an extension that allows developers to integrate and test smart applications using Artificial Intelligence (AI) models. It provides access to various models and simplifies the integration process within the editor. AI Toolkit offers seamless integration with popular AI models such as OpenAI, Anthropic, Google, and GitHub.

3.4.1. Installation

To install the **AI Toolkit** in **Visual Studio Code**, follow the steps below:

1. Open the **Extensions** menu in **VS Code**.
2. Type "AI Toolkit" in the search bar.
3. Locate the extension and click the **Install** button.

Once the installation is complete, the **AI Toolkit** icon will appear in the Activity Bar on the left side of the editor.

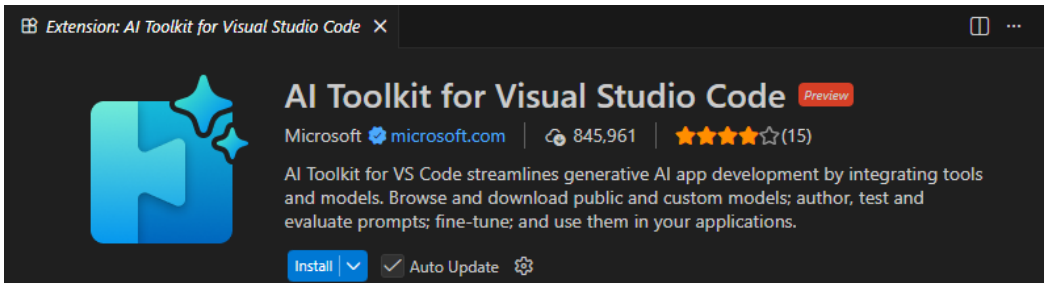


Figure 10. AI Toolkit Installation

Figure 11 shows the general structure of the VS Code Editor. When the Auto menu in the lower right corner is opened, the Artificial Intelligence Models in the editor are displayed, and the AI models are managed from this menu. When the Auto option is selected, all models are activated and used.

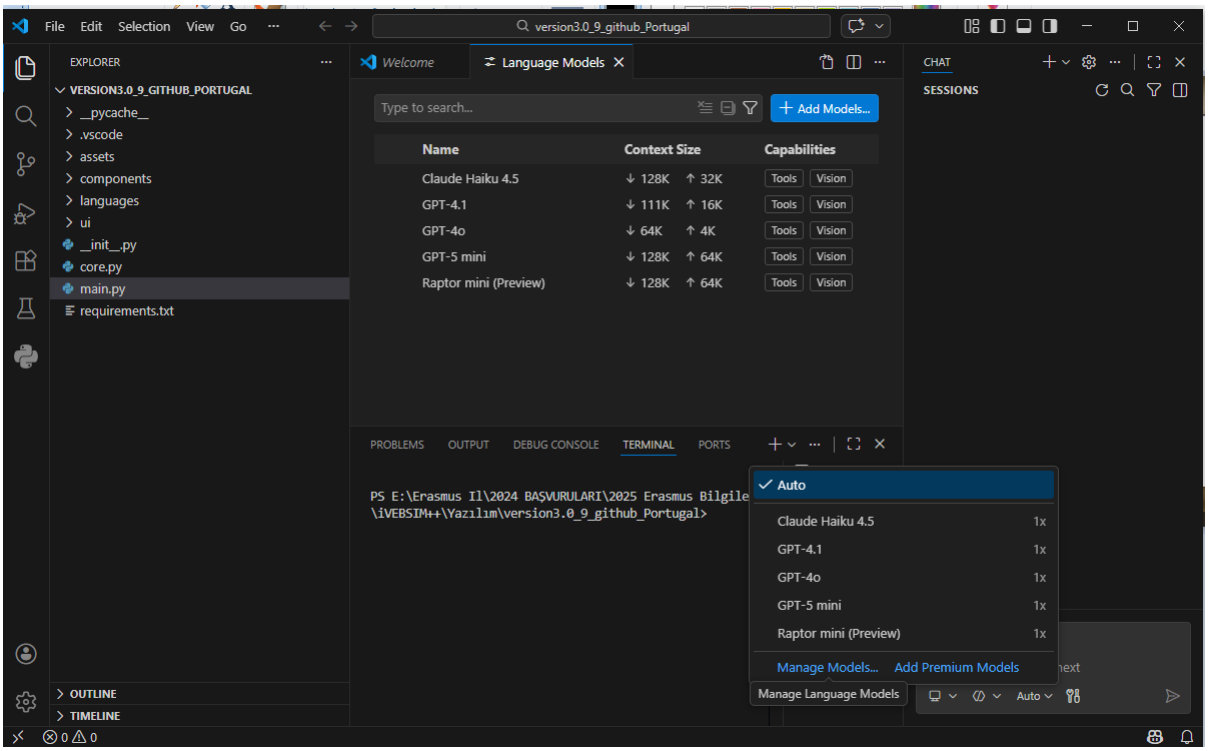


Figure 11. VS Code AI Support Management

3.4.2. Operation

After the installation is finished, you can launch the **AI Toolkit** and start using it. The toolkit offers the following advantages:

- **Error Analysis:** The results of errors made can be monitored instantly, and the process can be retried by going back to the beginning.
- **Data Collection:** Detailed data can be collected on how the system performs in extreme cases by pushing the limits of the system.

The **AI Toolkit** allows developers to test their models in a local environment and easily integrate them into their projects.

As shown in Figure 12, the AI was requested to develop a program to sum numbers from 1 to 10 using the menu at the bottom right. Subsequently, the AI was instructed to write this code in Python in the Editor. The AI completed the tasks by opening a file named **sum_example.py**, saving the code, running it, and printing the output. This type of AI integration was frequently utilized in this Project

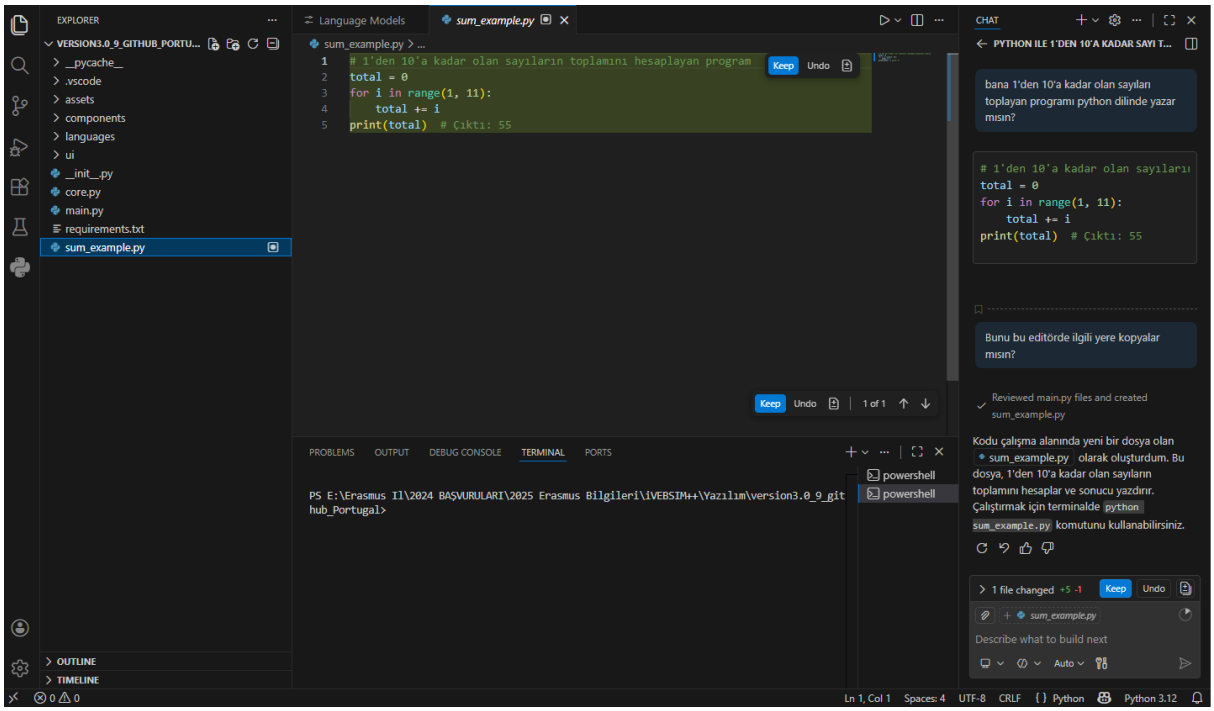


Figure 12. VS Code Artificial Intelligence Usage

4. PYTHON VARIABLES AND VARIABLE RULES

In Python, there is no specific command to declare a variable. A variable is created the moment you first assign a value to it. Variables are containers for storing data values.

Rules for Variable Names:

- A variable name must start with a letter or the underscore character (_).
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable names are case-sensitive (age, Age, and AGE are three different variables).
- Python keywords (such as if, for, while, print) cannot be used as variable names.

Example Variable Assignments:

```
Python
user_name = "Mert"
exam_score = 85
_status = True
```

In the example above, user_name stores a string value, exam_score stores an integer value, and _status stores a boolean value. Python automatically determines the data type based on the value assigned.

4.1. Python Operators

Operators are used to perform operations on variables and values. Python divides operators into several groups: arithmetic, assignment, comparison, and logical operators.

4.1.1. Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations.

Operator	Definition	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (Remainder after division)	a % b
**	Exponentiation (Power)	a ** b
//	Floor Division (Division without remainder)	a // b

Example Codes:	**Ekran Çıktısı**
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. Assignment Operators

Assignment operators are used to assign values to variables. They can also be used to perform a mathematical operation and assign the result to the same variable simultaneously.

Example Codes	Screen Output
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

Operator	Example	Description
=	a = 2	The value 2 is assigned to variable a.
+=	a += 2	Adds 2 to variable a and assigns the result back to a. (Equivalent to a = a + 2)
-=	a -= 2	Subtracts 2 from variable a and assigns the result back to a. (Equivalent to a = a - 2)
*=	a *= 2	Multiplies variable a by 2 and assigns the result back to a. (Equivalent to a = a * 2)
/=	a /= 2	Divides variable a by 2 and assigns the result back to a. (Equivalent to a = a / 2)
%=	a %= 2	Takes the modulus of variable a by 2 and assigns the result back to a. (Equivalent to a = a % 2)
**=	a **= 2	Raises variable a to the power of 2 and assigns the result back to a. (Equivalent to a = a ** 2)
//=	a //= 2	Performs floor division on variable a by 2 and assigns the result back to a. (Equivalent to a = a // 2)

4.1.3. Comparison Operators

Comparison operators are used when you want to compare the values of two variables and perform an action based on the result.

Operator	Definition	Example
==	Equal to	a == b
!=	Not equal to	a != b
<	Less than	a < b
>	Greater than	a > b
<=	Less than or equal to	a <= b
>=	Greater than or equal to	a >= b

Example Codes	Screen Output
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. Logical Operators

Logical operators are used to decide the program flow by checking the values of two or more variables.

Operator	Example	Description
and	a < 3 and b >= 5	Returns **True** if all conditions are true. In this example, if variable a is less than 3 and variable b is equal to or greater than 5, it returns **True**.
or	a < 3 or b > 4	Returns **True** if at least one of the conditions is true. In this example, it is enough for a to be less than 3 or b to be greater than 4 to return **True**.
not	not(a < 3)	Used to reverse the result (if it is **True**, it becomes **False**; if it is **False**, it becomes **True**).

Example Codes	Screen Output
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. Python Data Types

In Python, there are generally data types such as **string** (textual), **numbers** (numeric), **boolean**, **list**, **tuple**, **dictionary**, and **set**. The variable types we will use most at the beginning stage are listed below.

4.2.1. String (Textual) Data Type

String data types are used to store textual expressions. Values are written inside single quotes (' ') or double quotes (" "). The characters can be letters (t, c), numbers (1, 9, 2, 3), or special symbols (&,/).

name=”Mert”
surname=”Yılmaz”

4.2.2. Numeric Data Type

Used to store numeric values. There are three main numeric types:

- **int (Integer):** Used for whole numbers (e.g., 5, -10, 100).
- **float (Floating Point):** Used for decimal numbers (e.g., 3.14, -0.5).
- **complex:** Used for complex numbers (e.g., 3+5j).

Example:

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

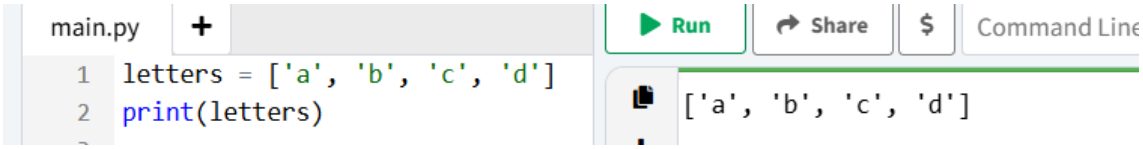
4.2.3. Python Lists

Lists are used to store multiple items in a single variable. Lists are defined using square brackets []. They are ordered, changeable, and allow duplicate values. You can store different data types such as int, float, and string in a single list. Lists are used to store multiple data in an ordered and modifiable structure.

Examples:

```
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]
```

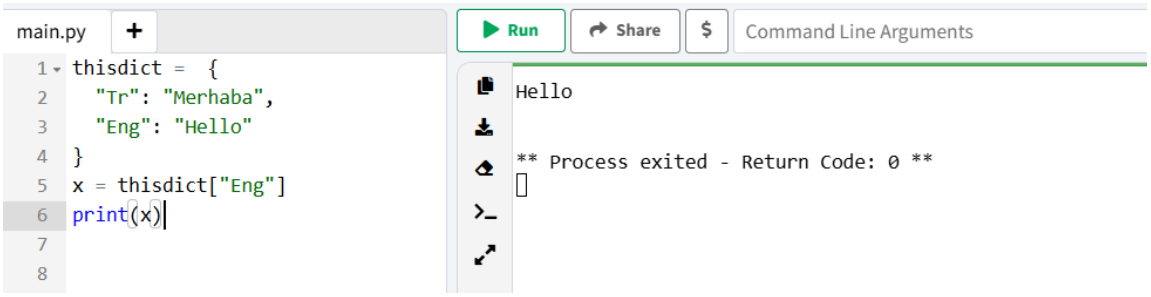


4.2.4. Dictionary

Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered (as of Python 3.7), changeable, and does not allow duplicates. Dictionaries are written with curly brackets {}.

Examples:

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



5. DECISION STRUCTURES

Decision structures allow the program to perform different actions based on whether a certain condition is met.

5.1. Decision Structure – IF..ELIF

The if statement is used to execute a block of code only if a specified condition is true. If the first condition is false, the elif (else if) statement allows us to check for multiple other conditions.

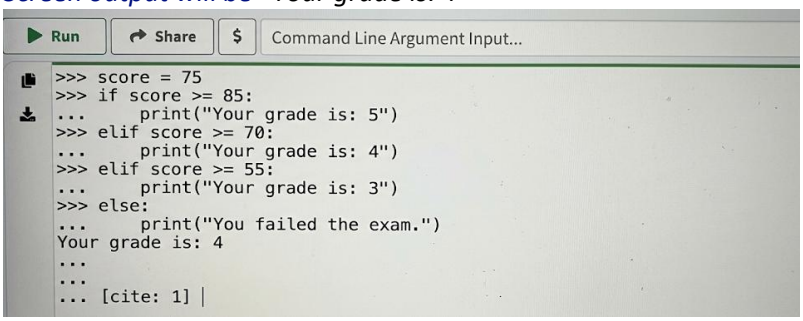
Syntax:

```
if condition:
   # Code to execute if condition is true
elif another_condition:
   # Code to execute if the first condition is false
   # and this condition is true
else:
   # Code to execute if none of the above conditions are true
```

Example Code:

```
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*

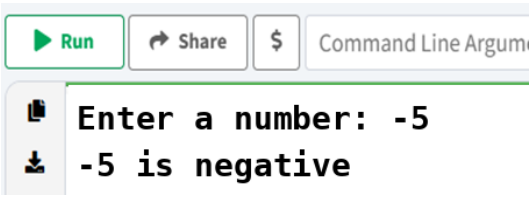


Example Code:

```
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")
```

The program asks for a number to be entered from the keyboard.

For example, if it is assumed that the value -5 is entered into the variable 'a', it will start checking in the if block.

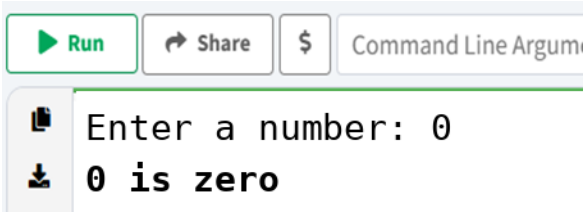


- Is the value of variable A greater than 0? Since -5 is not greater than 0, we proceed to the other condition expression.
- A < 0, is the value of variable A less than 0? Yes, since the condition is true, print("..") command will execute. It will print “-5 number is negative” ,

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

For example, let's say the value 0 is entered into the variable 'a'.

- Is the value of variable A greater than 0? Since 0 is not greater than 0, we proceed to the other condition.
- A<0, Is the value of variable A less than 0? No, 0<0 is not true. Therefore, since all conditions are not met, it passes to the else block. <0 ,



- print("..") command will run. It will print "0 is zero" to the screen.

6. LOOP STRUCTURES

Loop structures are used to repeat a specific block of code multiple times. Instead of writing the same code over and over, we use loops to perform repetitive tasks efficiently. Loops help make our code simpler, more understandable, and shorter. We use For, While, and Do While loop structures.

6.1. FOR Loop

The for loop is used to iterate over a sequence (such as a list, a tuple, a dictionary, a set, or a string). It is generally used when the number of repetitions is known beforehand.

Syntax:

```
for variable in sequence:
    # Code to be executed
```

Using the range() Function with FOR:

To loop through a set of code a specific number of times, we can use the range() function. The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Syntax

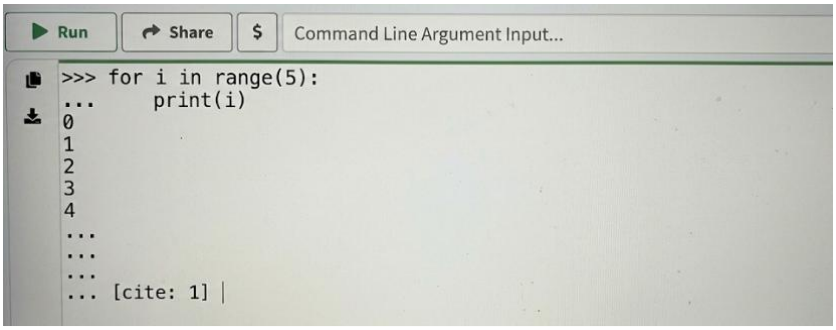
The range() function can take up to three arguments: range(start, stop, step)

Parameters

- **start (Optional):** The starting value of the sequence. If omitted, it defaults to **0**.
- **stop (Required):** The number at which the sequence stops. Note that the sequence **does not include** this number (it stops at stop - 1).
- **step (Optional):** The increment (or decrement) between each number in the sequence. If omitted, it defaults to **1**.

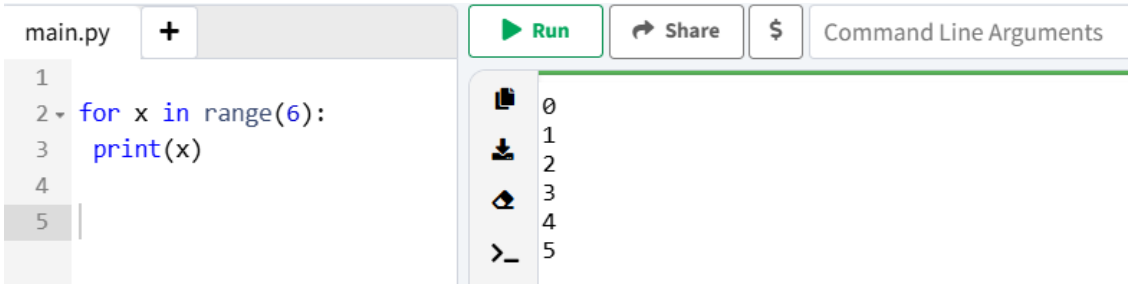
Example 1:

```
for i in range(5):
    print(i)
```



Note: The code above will print numbers from 0 to 4. The number 5 is not included.

Example 2: The following code displays the numbers from 1 to 6 on the screen.



Example 3: In the code below, the values are printed in increments of 100 from 50 to 350.

If we run it step by step:

• The value 50 is assigned to the counter variable x, and the value of the x variable is checked. Is x <= 350? Is x less than 350? Yes, our condition is met. And it enters the loop. It combines the string "Number =" with the value of the x variable and prints "Number = 50" to the screen. It returns to the beginning of the loop.

• The value of the counter variable x is increased by 100. The value of the x variable becomes 150. Is x <= 350? i.e., is 150 < 350? Yes, our condition is met. It enters the loop. The print command runs again, combining the value of the x variable with the constant text expression and printing "Number = 150" to the screen. It returns to the beginning of the loop. • The value of the counter variable x is increased by 100. The value of the x variable becomes 250. Is x < 350? i.e., is 250 < 350? Yes, our condition is met. It enters the loop. The `print again` command runs, combining the value of the `x`

variable with the constant text expression, printing `Number = 250` to the screen. It then returns to the beginning of the loop.

• It increments the value of the `x` counter variable by 100. The value of the `x` variable becomes 350. Is x < 350? That is, is 350 < 350? No, our condition is not met because it is equal. We exit the loop. Since there are no commands to execute outside the loop, our program terminates. The screen will look like the image above.

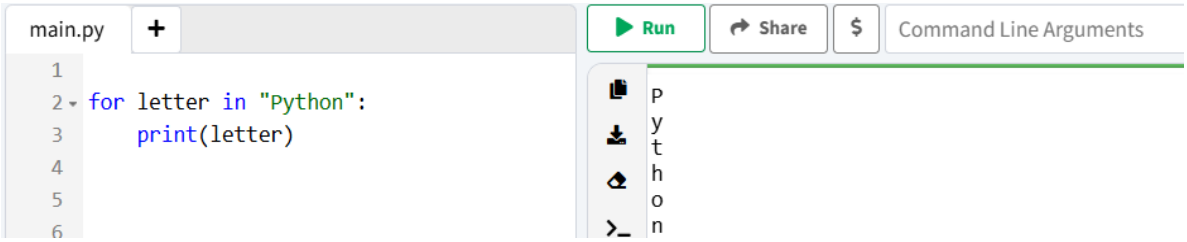
Thus, we execute the commands inside the for loop 3 times until our condition is met. In conclusion, the number of iterations of the for loop varies depending on the starting value and the condition. The commands inside the loop continue to execute until the condition is met.

Code	Output	Description
range(5)	0, 1, 2, 3, 4	Starts at 0, stops before 5.
range(2, 6)	2, 3, 4, 5	Starts at 2, stops before 6.
range(0, 10, 2)	0, 2, 4, 6, 8	Starts at 0, increments by 2.
range(5, 0, -1)	5, 4, 3, 2, 1	Counts down from 5 to 1.

Using the "In" Command with the For Loop

The **"in"** command in a for loop is used to check if a value exists within a sequence (like a list, string, or range). It acts as a bridge that allows the loop to pick each element from the sequence one by one and assign it to the loop variable.

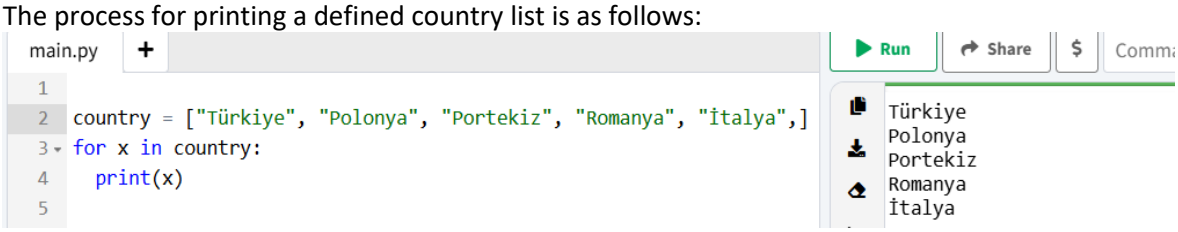
Usage with a String (Text): A for loop can also iterate through each character in a string.



Output: P, y, t, h, o, n (each on a new line)

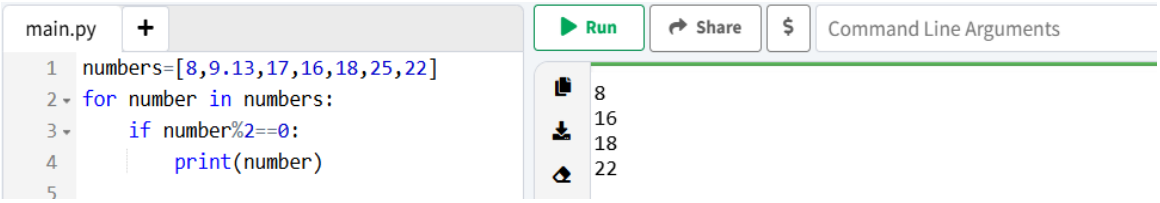
Usage with a List: When used with a list, the **"in"** command ensures that the loop variable takes the value of each item in the list sequentially.

Example with a List:
The process for printing a defined country list is as follows:



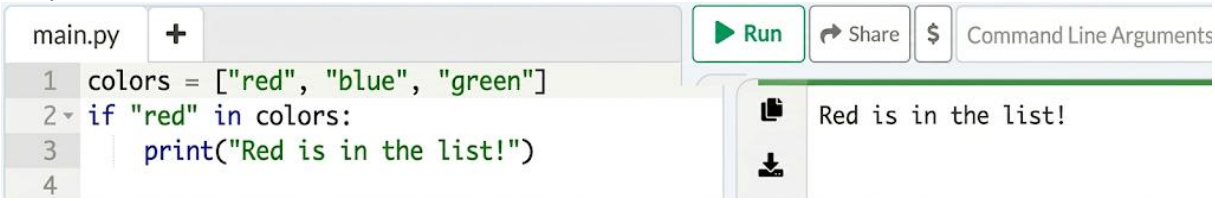
TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Example with numbers: The system prints even numbers from a given number sequence to the screen.



Else in FOR Loop: The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

Membership Check: Beyond loops, the "in" keyword can also be used in if statements to check for the presence of an element



6.2. WHILE Loop

The while loop is used to execute a set of statements as long as a condition is true. Unlike the for loop, the while loop is generally preferred when the number of repetitions is not predetermined. The loop exits as soon as the condition is no longer met, and the program continues executing from where the loop ended.

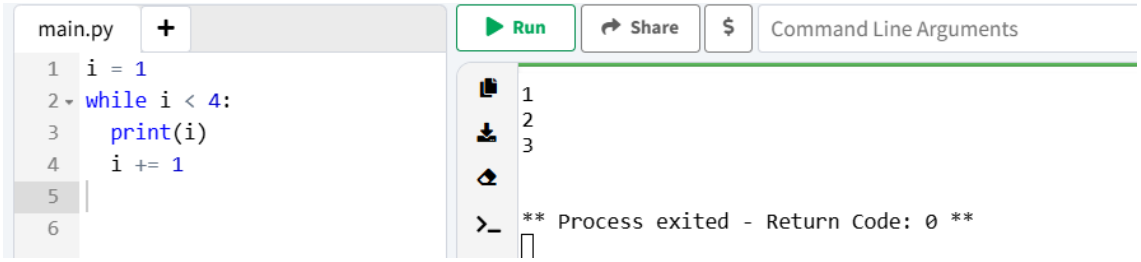
Syntax:

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

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

Example

The code below prints the numbers from 1 to 4 on the screen.



WHILE LOOP STEP-BY-STEP (i < 4)

1. **Initialization:** The variable i starts at **1**.
2. **Iteration 1:** Since 1 < 4 is **True**, it prints **1** and i becomes **2**.
3. **Iteration 2:** Since 2 < 4 is **True**, it prints **2** and i becomes **3**.
4. **Iteration 3:** Since 3 < 4 is **True**, it prints **3** and i becomes **4**.
5. **Termination:** Now i is **4**. Since 4 < 4 is **False**, the loop stops.

Summary of Output: 1, 2, 3

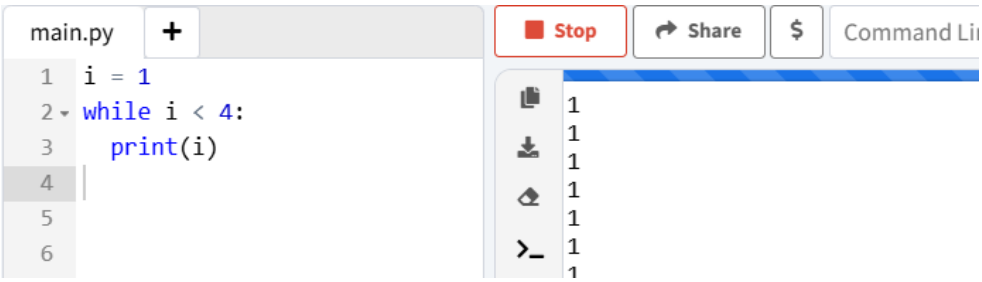
If we increase the condition value here, starting with i=1:

• For i<=10, the commands inside the loop will run 10 times,

• For i<=100, the commands inside the loop will run 100 times.

• For i<5, the commands inside the loop will run 4 times. (Because there is no <= expression here, our loop will not run for i=5.)

Note: In the example below, the loop continues as long as. The i += 1 statement is missing; and the value of i is not incremented, the condition will always remain true, resulting in an "**Infinite Loop**". Stop is pressed.



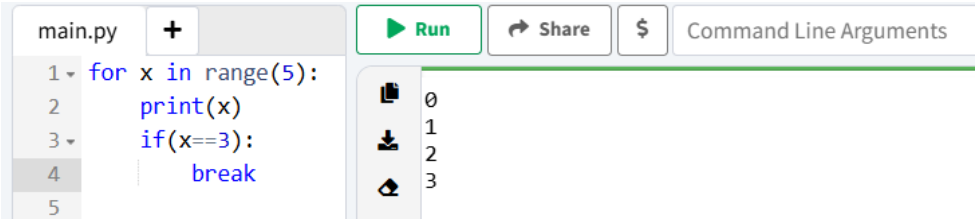
6.3. The Break Statement

With the break statement, we can stop the loop even if the while condition is still true. This is particularly useful when you want to exit the loop when a specific secondary condition occurs.

Example Code:

Without Break, Output: 1, 2, 3,4 (The loop terminates when i reaches 4).

With Break, Output: 1, 2, 3 (The loop terminates when i reaches 3).



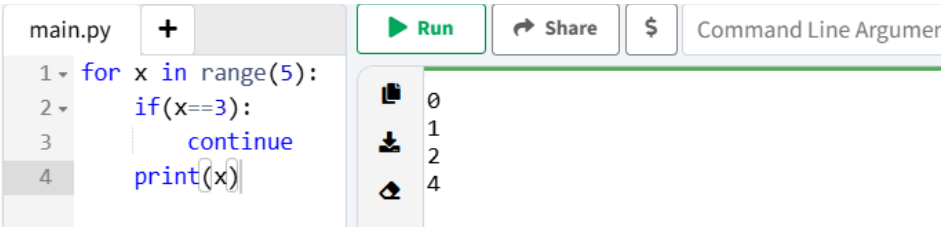
6.4. The Continue Statement

With the continue statement, we can stop the current iteration of the loop and skip to the next cycle. Unlike break, it does not terminate the entire loop.

Example Code:

In the code below, the value of x is checked, and if it is equal to 3, the `continue` command is executed, returning to the beginning of the loop. Looking at the output, you can see that the value 3 is not printed to the screen for this reason.

Output: 1, 2, 4, 5, 6 (The number 3 is skipped).



TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by
the European Union

CREATION OF SIMULATOR SOFTWARE

7. SIMULATOR SOFTWARES

Simulator softwares are computer programs that provide a realistic experience to the user by imitating a physical system, process, or device from the real world in a digital environment. These softwares allow us to foresee how a system will react to specific inputs by using complex mathematical models and algorithms. They are generally used for education and engineering purposes; thus, scenarios that are high-cost, dangerous, or difficult to access can be experienced repeatedly in a safe and controlled virtual space.

Basic Benefits of Simulator Softwares

- **Security:** It allows working in risk-free environments for situations that carry vital risks in real life (high voltage, high pressure, etc.).
- **Cost Savings:** It prevents the wear and tear of real equipment and eliminates operational expenses such as energy.
- **Error Analysis:** The results of mistakes made can be monitored instantly, and the process can be restarted and tried again.
- **Data Collection:** It provides detailed data on how the system performs in extreme situations by pushing its limits.

7.1. Industrial Control Simulator Softwares

Industrial electronic control simulation systems are softwares that allow complex automation processes and electrical circuits to be designed and tested in a digital environment without the need for physical installation. It has many advantages as stated below;

- **Security:** It reduces the risk of electric shock, explosion, or material damage that may occur in real panels to zero. It allows you to experiment without fear of making mistakes.
- **Cost Savings:** It allows you to set up complex circuits without physically purchasing expensive motors, switches, and contactors.
- **Learning Speed:** It significantly speeds up the learning process because it instantly shows how theoretical knowledge results in practice.
- **Low System Requirements:** It can run smoothly without lagging even on very old computers.
- **Logic Development:** These programs minimize error margins. It is the best starting point to establish the "control logic" before moving on to PLC (Programmable Logic Controller) programming.

In addition to the advantages, technical personnel should know that;

- **Program Update Issue:** The software is quite old and does not fully cover the developments in modern automation systems (modern PLCs, servo drives, etc.).

- **Limited Number of Elements:** Its library may be insufficient for very professional projects.
- **Visual Simplicity:** It may not offer as detailed dynamic visuals as its more advanced competitors (such as ©TIA Portal etc.).
- **Does Not Cover Physical Errors:** It cannot simulate real-life physical failures such as cable looseness, oxidation, or magnetic interference.

In light of all this, while creating a simulator software—which we will call **iVEBSIM+** from now on;

- **Wide Library:** Includes classic control elements such as contactors, time relays, buttons, asynchronous motors, and various sensors.
- **Dynamic Simulation:** After setting up the circuit, you can see how the system works in real-time by pressing the "Play" button.
- **Error Check:** The software warns you when a wrong connection is made or a short circuit occurs.
- **Multi-Language Support:** Since it is local software, it offers a completely Turkish interface. It supports various languages.
- **Drag-and-Drop Logic:** Circuit design is made with a visual interface instead of complex coding languages.

Features	**iVEBSIM+**	**Professional Software (TIA Portal, etc.)**
Purpose of Use:	Training and Basic Logic	Industrial Application
Difficulty Level:	Very Easy	Difficult / Requires Expertise
Cost	Free /Accessible	Very High
Hardware Requirements:	Low	High

Table 1. Industrial Software Comparison

8. TKINTER – VISUAL PROGRAMMING

8.1 What is Tkinter?

Graphical User Interface (GUI) libraries belonging to that programming language are used when creating an interface in a software.

Tkinter is the standard graphical user interface of the Python Programming Language. It is built on the Tcl/Tk toolkit and is automatically installed with the Python Programming Language.

It is small in size and fast. Besides platform independence (Windows, macOS, Linux), most importantly, it is easy to learn and use. **Source Code:** Lib/tkinter/ init .py

8.1.1. Architecture of Tkinter

Tcl/Tk is not a single library; it consists of several different modules, each with separate functionality and its own official documentation. Binary versions of Python also offer a plug-in module alongside them.

Tcl

Tcl is a dynamic interpretive programming language, just like Python. Although it can be used on its own as a general-purpose programming language, it is most commonly embedded into C applications as a script engine or as an interface to the Tk toolkit. Unlike Python, Tcl's execution model is designed around cooperative multitasking, and Tkinter bridges this difference.

Tk

Tk is a Tcl package implemented in C that adds special commands to create and manipulate GUI widgets (arayüz bileşenleri).

Ttk

Themed Tk (Ttk) is a newer Tk widget family that provides a much better appearance on different platforms compared to many classic Tk widgets. Ttk has been distributed as part of Tk since Tk version 8.5. Python bindings are provided in a separate module, tkinter.ttk. Internally, Tk and Ttk use the facilities of the underlying operating system; namely Xlib on Unix/X11, Cocoa on macOS, and GDI on Windows.

8.1.2. Tkinter Basic Components - Widgets

A widget (arayüz bileşeni) can basically be called a component. They perform a specific task. A Tkinter user interface consists of individual widgets. Each widget is represented as a Python object instantiated from classes such as ttk.Frame, ttk.Label, and ttk.Button.

Basic Widgets:

- **Frame:** A frame for grouping other widgets.
- **Label:** A label for displaying text or images.
- **Button:** A clickable button.
- **Entry:** A single-line text entry.

- **Text:** A multi-line text area.
- **Canvas:** An area for drawing and graphics.
- **Menu:** Menu bars.
- **Scrollbar:** Scroll bars.
- **Checkbutton:** Checkboxes.
- **Radiobutton:** Selection buttons.
- **Listbox:** List boxes.
- **Scale:** Sliders.
- **Spinbox:** Number selectors.

8.2. Window Creation

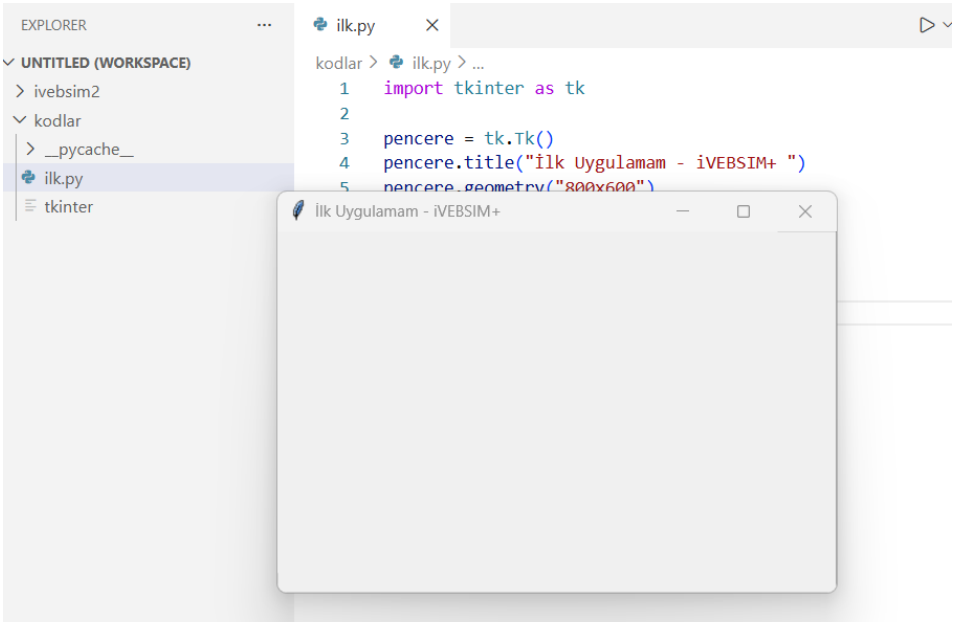
In **Tkinter**, you follow these steps to create a window:

- **Import Tkinter**.
- Create the main window object (**tk.Tk()**).
- Set window properties (title, size, position).
- Add **widgets** (**buttons**, **labels**, **entry** fields).
- Bind events (such as clicking or pressing a key).
- Start the main **loop** (**mainloop()**).

Sample Application

```
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
```

When we run the codes, we see our Form page as follows.



First Application Example - Detailed Review

Now, let's examine our first **Tkinter** application step by step:

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

    Imports the **Tkinter** module with the alias "**tk**".

2. **Main Window:**

    pencere = tk.Tk()

- tk.Tk() creates the main window.
- This window is the main container for all other **widgets**.
- Each application can have only one Tk() object.

3. **Window Properties:**

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

- title(): The text that appears in the title bar.
- geometry(): Sets the window size in pixels.

4. **Main Loop:**

pencere.mainloop()

- Makes the window visible.
- Listens for mouse and keyboard events.
- Keeps the application running.
- It is an infinite **loop**, but it ends when the window is closed.

8.3. Adding Widgets and Binding Events

Widgets allow adding form elements such as **Frame**, **Label**, **Button**, **Entry**, **Text**, **Canvas**, **Menu**, **Scrollbar**, **Checkbutton**, and **Radiobutton** as mentioned in 8.1.2.

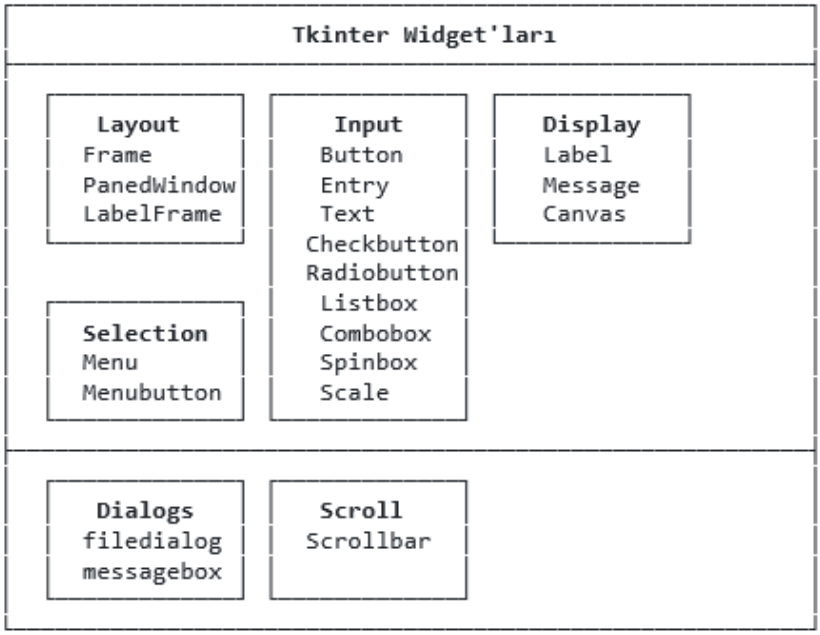


Figure 13. Tkinter's Widgets

8.3.1 Adding Labels

Used to add text to the screen. The Label Method from the **Tkinter** library is used to add labels to the form.

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

- text: The text to be displayed.
- font: Font type, size, and style.
- bg: Background color (**background**).
- fg: Foreground color (**foreground**).

8.3.2 Adding Buttons

Creates a clickable button. The Button Method from the **Tkinter** library is used.

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

- text: The text on the button.

- command: The function to be called during a click event.
- state: "normal", "active", "disabled" states.

8.3.3 Setting Layouts

pack() places objects into the window. You can determine how they will be positioned horizontally and vertically on the form.

```
etiket.pack(pady=20)

buton.pack(pady=10)
```

- pady: Vertical spacing (pixels).
- padx: Horizontal spacing.
- side: Which side (**TOP**, **BOTTOM**, **LEFT**, **RIGHT**).
- fill: Filling empty space (**X**, **Y**, **BOTH**).
- expand: Expanding in empty space.

Sample Application

```
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()
```

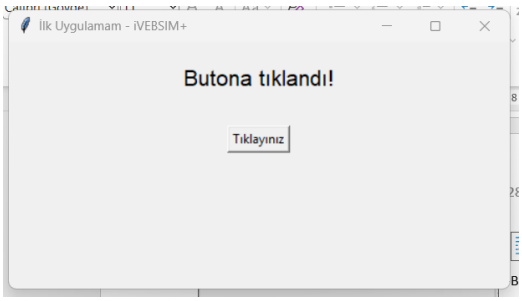
This code performs the following operations:

- Loads the **Tkinter** library.
- Creates and sets up the main window.

- Creates a label displaying the text "Merhaba Tkinter!".
- Defines the click function.
- Creates the "Tıklayınız" button.
- Places **widgets** into the window.
- Displays the window and listens for events.

Executing the Code

When the button is clicked, the form appears as shown in the image on the side.



8.3.4 Adding Entries

It is used to receive single-line text input from the user.

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

Sample Application

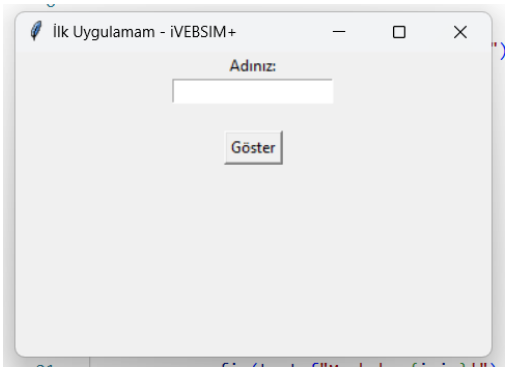
```
import tkinter as tk

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()
```

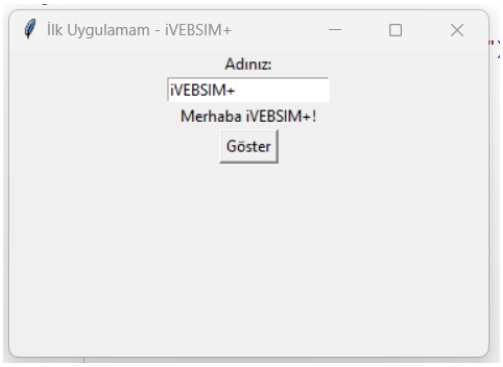
This code performs the following operations:

- Loads the Tkinter library.
- Creates and sets up the main window.
- Creates a Label (etiket) displaying the text "Adınız".
- Creates an Entry (giriş kutusu) for entering the name.
- Creates a Label to display the result.
- Defines the click function.
- Creates the "Göster" Button.
- Displays the window and listens for events.

Running the Codes



When the button is clicked, we see the form as shown in the side image.



8.3.5 Adding Radiobutton

It is used for the user to select only one option from multiple choices.

```
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 Adding Checkbutton

It is used for the user to select one or more options from the given choices.

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

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

8.3.7 Adding Scrollbar

It is used to add a scroll bar to the form screen.

```
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 Adding Menu

It is used to add a top menu bar.

```
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 Binding Events

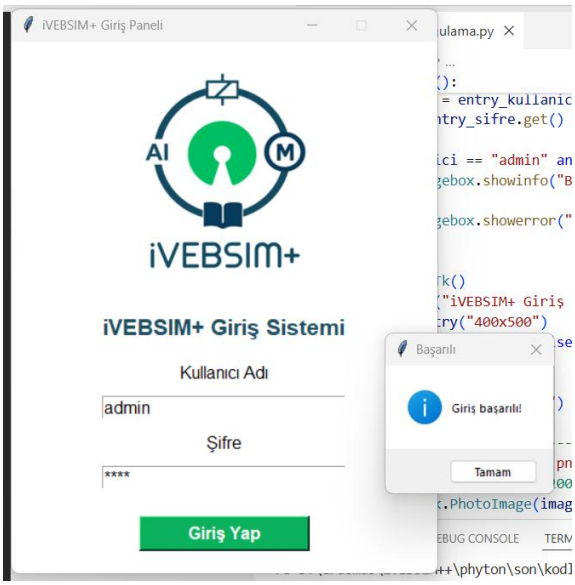
In **widgets**, we can define the codes we want to run during click events inside a function.

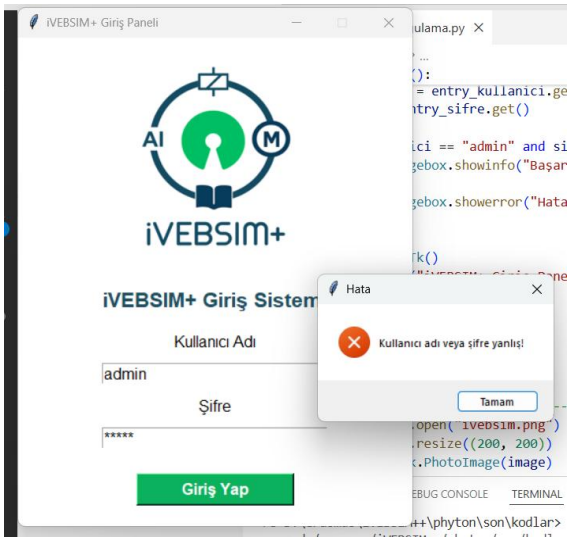
```
 def goster():
    isim = giris.get()
    sonuc.config(text=f"Merhaba {isim}!")
buton = tk.Button(pencere, text="Göster", command=goster)
```

Sample Codes:

```
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

9. CREATING THE SIMULATOR INTERFACE

Simülatör arayüzü çalışmaya başladığında önce bir karşılama ekranı gelmektedir. Bu ekranda bu programa destek veren kurumlar yazmaktadır. Bu pencere fare ile kliklendiğinde programın arayüzü ekrana gelmektedir. Bizim yazılımımızda karşılama ekranı main.py ve program ana penceresi ui.py olarak isimlendirilmiştir.

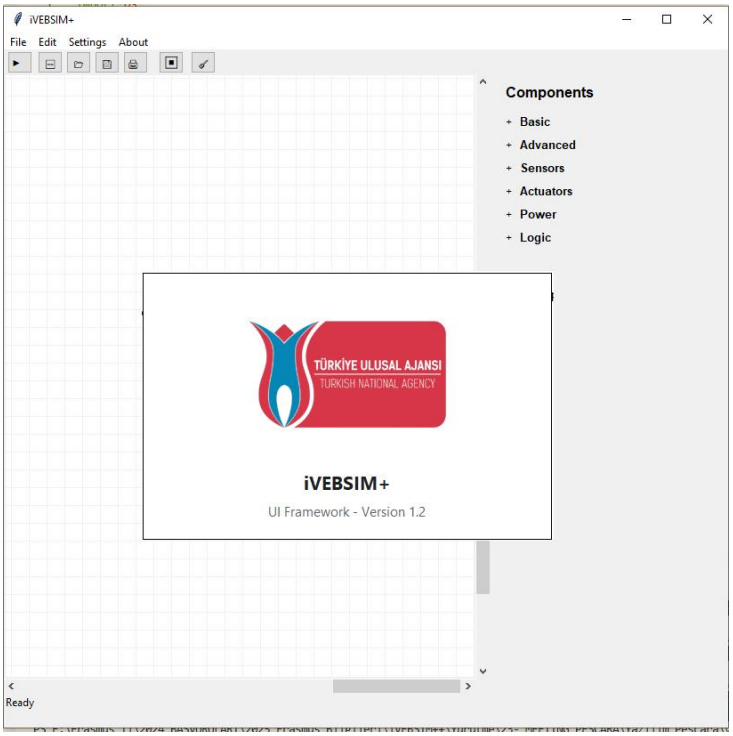


Figure 14 – Software View

9.1 Creating Splash

To launch the application and display a Welcome Screen (splash) to the user, follow these steps:

The Python Tkinter library is added.

```
import os

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

A top splash screen is created on the main TK window, which is given as root.

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

The window size appears to be fixed.

```
# 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)
```

The window content is created.

```
# 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(
```

The splash size is recalculated and centered.

```
# 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}")
```

Figure 15 – Splash View

The splash is triggered by a click or key press, activating “proceed”. Proceed closes the welcome screen and recalls the main screen.

```
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)
```

The splash is called and the event loop is initiated.

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


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

9.2 Creating the Interface Infrastructure

The `ui.py` file defines the `App` class, which provides the graphical user interface for the software. The software includes menus, a toolbar, a drawing area in the center (`Canvas`), and right/left side panels (component accordion). To create the application interface, in addition to the previously mentioned points, the following steps are followed:

A screen window is created within 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)
```

The menu bar (File / Edit / Settings / About) is created with `tk.Menu`. Each menu command is linked to a method (`_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)
```

The toolbar is created using ‘ttk.Button’. Buttons such as Save, Start, Stop, Clear, etc. are added.

```
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)
```

The main area is configured with `ttk.PanedWindow` and partitioned into two panels using `PanedWindow`. The left side is set as `canvas_frame`, containing the drawing area (`Canvas`), and the right side as `side_panel`, containing the component accordion.

The canvas area is created with `scrollregion`, and scrollbars are added horizontally and vertically. When the canvas area is resized, the grid will be updated using `configure`. Also, when using coordinates on the canvas, the `canvasx` and `canvasy` methods are used. This allows for conversion from the screen to the 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()))
```

The side panel features accordion-style categories (`basic`, `advanced`, `sensors`, `actuators`, `power`, `logic`) – created as headers and bodies using `_add_accordion_section`. A footer logo is also added at the bottom. Accordion functions are controlled using `def _add_accordion_section`. On/off status is controlled using an “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))
```

Grid and View functions are implemented as follows.

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

Menu Commands and Helper Methods are defined separately.

* File operations (placeholders): `_new_file`, `_open_file`, `_save_file`, `_save_as_file`

* Edit operations (placeholders): `_undo`, `_redo`, `_cut`, `_copy`, `_paste`. –

* Settings: `_show_preferences`, `_change_language`, Grid toggle checkbutton.

* About Us: `_show_about` → displays an information window about the application.

```
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 Multilingualism

The software is built on multilingualism.

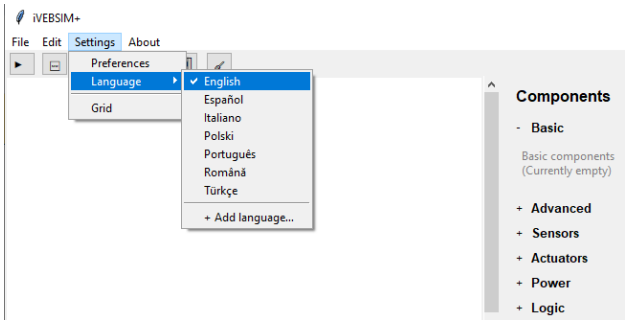


Figure 16 – Multilingualism

The Python i18n library is used to enable multilingualism in the program. When the application is started (in the __init__ method), the default language code is set to "en" (English).

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

Language files are stored in the languages folder in JSON format (e.g., en.json, tr.json, es.json, etc.). Each JSON file has a structure such as "code" and "name" required fields (language code and display name). The config.json file stores the selected language: {"selected_language": "en"}. This is so that the application remembers the last selected language when it is reopened.

Figure 17 – json files

The Translation Load Function (_load_translations) takes a language code as a parameter (e.g., "tr") and opens the languages/{code}.json file to load the 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
```

The Settings menu has a "Language" submenu. The `_populate_language_menu()` function scans all JSON files in the languages folder. It retrieves the "code" and "name" from each file. When the user selects a language, `_on_language_selected()` is called, and the lang_code is updated.

```
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()
```

After selecting a language, the UI is completely rebuilt. Changing the language requires destroying and recreating all widgets. Text in Tkinter doesn't change dynamically. Therefore, the menu bar is removed and the entire UI is rebuilt from scratch.

```
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()
```

10. ADDING ELEMENTS AND CREATING THE SIMULATION

NEXT ROMANIA……

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

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))
```

NOTES:

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union

NOTES:

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Funded by the European Union