Izmir,
Pescarar
2026

Software Simulador de Código Aberto

iVEBSIM+

LINGUAGEM DE PROGRAMAÇÃO

SOFTWARE DE SIMULAÇÃO

MÓDULO DE INTELIGÊNCIA ARTIFICIAL

UTILIZAR O SIMULADOR

EXERCÍCIOS PRÁTICOS

Projeto Erasmus+ KA220VET

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

Avrupa Birliği tarafından finanse edilmektedir

"Apoiado pela Comissão Europeia no âmbito do Programa Erasmus+. O conteúdo aqui presente reflete as opiniões dos autores, e a Comissão Europeia e a Agência Nacional Turca não podem ser responsabilizadas por essas opiniões."

Cofinanciado pelo Programa Erasmus+ da União Europeia

ÍNDICE

PREFÁCIO

Os sistemas educativos estão a ser reformulados pelo impacto da transformação digital; especialmente no campo da educação profissional e técnica, a necessidade de ambientes de aprendizagem inovadores, flexíveis e acessíveis aumenta a cada dia.

Este projeto, intitulado “Desenvolver a Educação Profissional com um Simulador de Código Aberto com Suporte de IA” e realizado no âmbito das parcerias de cooperação Erasmus+ KA220-VET, visa produzir uma resposta sustentável e inovadora a esta necessidade.

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

O principal resultado do projeto, o simulador iVEBSIM+, é um software totalmente de código aberto, com suporte de IA, concebido para o campo da eletrónica industrial e controlo. Neste sentido, o projeto não se limita a produzir material educativo; também visa criar um ecossistema de aprendizagem digital partilhável, melhorável e sustentável.

O livro que tem nas mãos foi concebido como um dos resultados do projeto. Na primeira parte do livro, são apresentadas informações teóricas e práticas sobre Python, que é utilizada como linguagem de programação base no desenvolvimento do software de simulação. Temas como fundamentos de programação, estruturas de variáveis e mecanismos de decisão e repetição são abordados com uma clareza que os alunos de educação profissional conseguem compreender e com uma abordagem orientada para a prática.

Nas secções seguintes, os seguintes tópicos são abordados em detalhe:

- O enquadramento teórico do software de simulação
- Criação da interface gráfica do utilizador
- Integração de componentes eletrónicos industriais no ambiente digital
- Desenvolvimento e utilização do módulo de inteligência artificial
- Exercícios práticos preparados para uso na educação profissional
- Processos de aplicação piloto realizados com alunos

O processo do projeto está estruturado com base na conclusão de uma etapa técnica específica em cada mobilidade. Todas as etapas, desde a criação dos fundamentos em Python até ao desenvolvimento do núcleo do simulador, desde a expansão da biblioteca de componentes industriais até à integração da inteligência artificial e à preparação de exercícios pedagógicos, foram realizadas no âmbito da cooperação internacional.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Cofinanciado pelo Programa Erasmus+ da União Europeia

Desta forma, tanto a capacidade técnica foi melhorada como a partilha de boas práticas entre países foi assegurada.

Uma das características mais importantes deste estudo é que o simulador desenvolvido é **de código aberto**. Todos os códigos fonte do Simulador iVEBSIM+ são partilhados com o público através do sítio web do projeto, oferecendo uma estrutura aberta às contribuições de educadores, alunos e programadores. Esta abordagem baseia-se nos princípios da transparência, acessibilidade e produção coletiva, e apoia a igualdade digital na educação profissional. Além disso, partilhar as etapas de desenvolvimento do software em detalhe no livro encoraja os aprendentes a tornarem-se programadores em vez de meros utilizadores.

A aprendizagem baseada em simulação na educação profissional é de grande importância, pois proporciona uma área de experimentação segura, reduz custos e permite oportunidades de aplicação repetíveis. **Inteligência Artificial** o suporte torna o processo de aprendizagem mais eficiente e interativo através de possibilidades como feedback individualizado, análise de erros e orientação inteligente. Este livro e o software que o acompanha visam contribuir para a aquisição das competências digitais exigidas pela era atual, integrando estas duas abordagens.

Este trabalho foi preparado com a esperança de que sirva de guia para professores que trabalham em instituições de educação profissional e técnica, alunos que estudam na área, programadores de software e investigadores. O nosso maior desejo é que este estudo, que emergiu do conhecimento e experiência de uma parceria internacional, se difunda em linha com a compreensão da ciência aberta e dos recursos educativos abertos.

Este projeto é apoiado pela União Europeia, e as opiniões expressas no conteúdo pertencem aos seus autores.

Gostaríamos de agradecer a todos os parceiros do projeto que contribuíram e aos educadores que se empenharam.

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

"Apoiado pela Comissão Europeia no âmbito do Programa Erasmus+. O conteúdo aqui presente reflete as opiniões dos autores, e a Comissão Europeia e a Agência Nacional Turca não podem ser responsabilizadas por essas opiniões."

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

Cofinanciado pelo Programa Erasmus+ da União Europeia

LINGUAGEM DE PROGRAMAÇÃO PYTHON

1. A IMPORTÂNCIA DA LINGUAGEM DE PROGRAMAÇÃO

Python é uma das linguagens de programação mais amplamente utilizadas atualmente. Foi desenvolvida por Guido van Rossum e lançada em 1991. É uma linguagem de alto nível com amplo suporte de bibliotecas, onde é possível desenvolver aplicações em quase todas as áreas.

Porquê Python?

- É fácil de aprender porque tem uma sintaxe simples.
- É semelhante à língua inglesa.
- Utiliza novas linhas para completar um comando em vez de ponto e vírgula e parênteses.
- Utiliza indentação principal (espaço no início da linha) para definir o âmbito de ciclos, funções e classes.
- Pode ser executado em todas as plataformas como Windows, Linux e Mac.

Áreas de Utilização:

- Programação de sistemas
- Programação de interfaces de utilizador
- Desenvolvimento web
- Programação de redes
- Desenvolvimento de jogos
- Ciência de dados, aprendizagem automática
- Análise de dados
- Inteligência Artificial principal (AI)
- Scripts e ferramentas

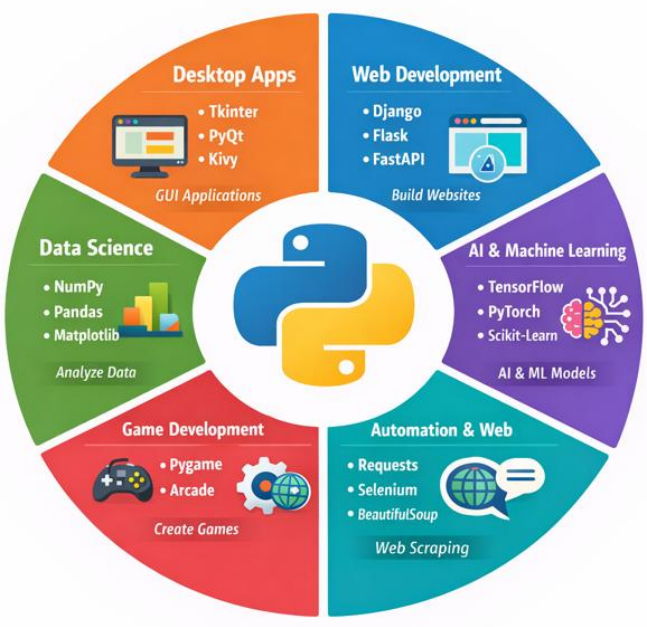


Figura 1. Bibliotecas Python

2. INSTALAÇÃO DA LINGUAGEM DE PROGRAMAÇÃO PYTHON

Para escrever código com a linguagem de programação Python, é necessário primeiro instalar um editor no computador.

2.1 Instalação do Python

Para Windows:

1. Aceda ao sítio oficial do Python em https://www.python.org/.
2. Clique no ícone **Downloads** no menu.
3. Transfira a versão mais recente do Python principal (por exemplo, Python 3.14.0).
4. Execute o ficheiro de instalação transferido. **Importante:** Selecione a **"Add python.exe ao PATH"** caixa de verificação antes de clicar no "Instalar Agora" botão. Este passo é necessário para executar o Python a partir da linha de comandos.
5. Quando a instalação estiver concluída e receber a mensagem "Configuração concluída com êxito", clique no botão "Fechar".

Figure 2. Instalação no Windows

Para macOS:

- Transfira o instalador com extensão .pkg do mesmo sítio e siga os passos de instalação padrão.
- Python é instalado nas Aplicações → Python x.x pasta.

Para Linux:

- Abra o terminal e execute os seguintes comandos por ordem:
- sudo apt update
- sudo apt install python3 python3-pip

2.2 Verificação da Instalação

Após a conclusão da instalação, pode verificá-la abrindo a Linha de Comandos principal (CMD) no Windows e escrevendo o seguinte comando:

python –version

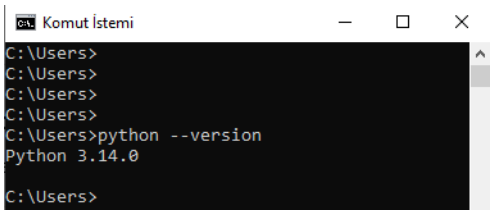


Figura 3. Versão da Instalação

Se vir o número da versão do Python, tal como na Figura 3, significa que foi instalado corretamente. Se não o vir, repita os passos de instalação.

2.3. Verificar o Gestor de Pacotes

Pode verificar o gestor de pacotes escrevendo o seguinte comando na Linha de Comandos principal (CMD) ou terminal: pip --version

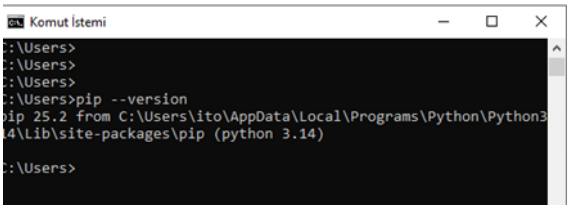


Figura 4. Versão do Gestor de Pacotes

pip (Instalador de Pacotes Python) é uma ferramenta utilizada para gestão de bibliotecas e instalação de pacotes em Python. Permite instalar facilmente as bibliotecas necessárias para os seus projetos.

2.4. Executar o Primeiro Programa Python

Python não requer um editor para escrever código de comandos. O código pode ser executado diretamente usando a Linha de Comandos. A Tabela 1 fornece um exemplo de um fragmento de código que imprime "Olá" no ecrã. No entanto, à medida que o código do programa aumenta e o uso de bibliotecas se torna maior, este método pode não ser ideal para os programadores.

Após a conclusão da instalação do Python, pode executar o seu primeiro programa usando o **IDLE (Ambiente Integrado de Desenvolvimento e Aprendizagem)** que acompanha o Python ou a Linha de Comandos.

Executar com a Linha de Comandos:

1. Abra a Linha de Comandos (CMD).
2. Escreva python e prima Enter. Isto iniciará o ambiente interativo do Python.
3. Escreva o seguinte código e prima Enter: print("Olá Mundo")
4. Verá a saída "Olá Mundo" no ecrã.

Executar como Ficheiro de Script:

1. Abra um editor de texto simples principal (como o Bloco de Notas).
2. Escreva o seguinte código: print("Bem-vindo ao iVEBSIM+")
3. Guarde o ficheiro com o nome hello.py.
4. Abra a Linha de Comandos, navegue até à pasta onde guardou o ficheiro e escreva: python hello.py

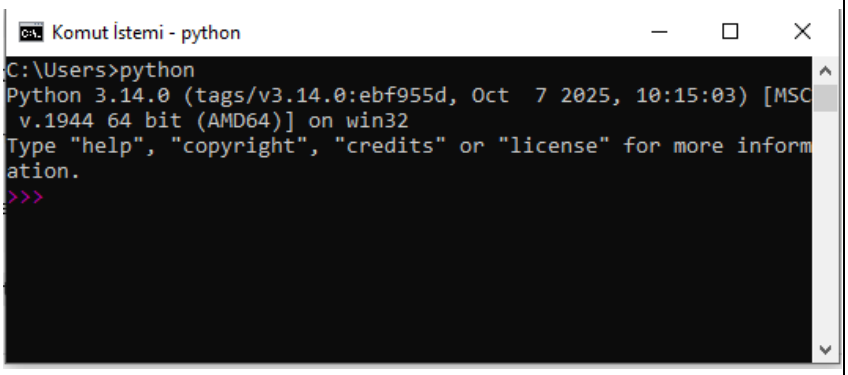
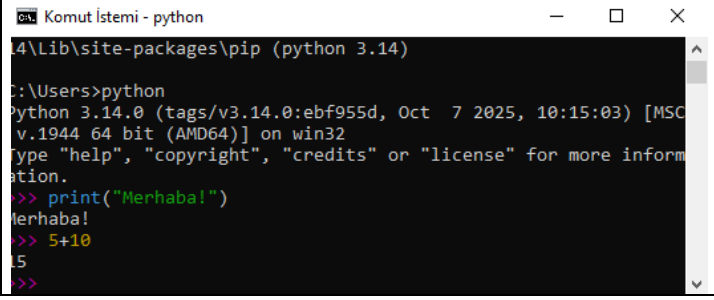
Códigos:	A saída no ecrã terá este aspeto:
Pyhton	
Códigos:	A saída no ecrã terá este aspeto:
print("Merhaba!") 5+10	

Tabela 1. Escrever Programas Usando a Linha de Comandos

3. UTILIZAR UM EDITOR

Pode escrever código Python num editor de texto simples e guardá-lo com a extensão .py. No entanto, utilizar um editor de código profissional torna o processo de desenvolvimento muito mais fácil e eficiente. Na Figura 5, foi criado um novo ficheiro chamado mycode.py

.

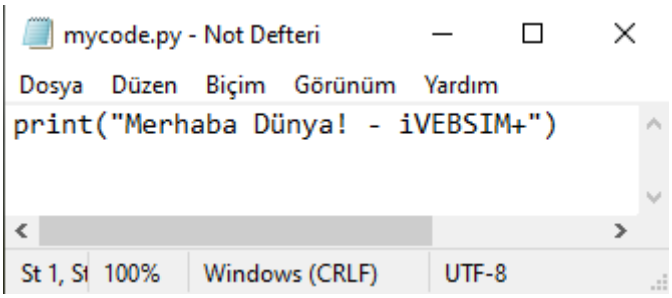


Figura 5. Utilizar o Editor de Texto.

Quando escrever o comando 'python mycode.py' na linha de comandos, o programa será executado e aparecerá na consola conforme indicado abaixo.

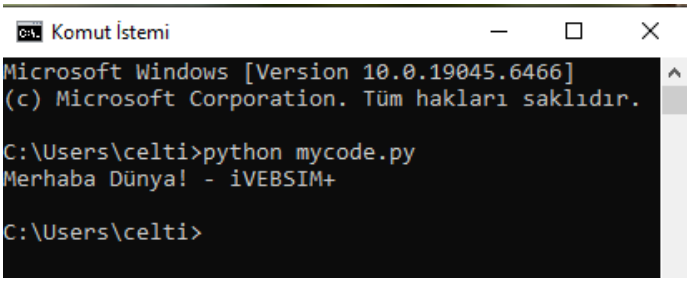


Figura 6. Saída do Programa na Consola

Também podemos usar editores para ver o nosso código mais organizado e escrevê-lo com mais facilidade. O Visual Studio Código, o PyCharm e o Sublime Text são alguns exemplos.

3.1 Instalação do Visual Studio Código

O Visual Studio Código principal (VS Código) é um editor de código gratuito, poderoso e popular que funciona em Windows, macOS e Linux. Para instalá-lo, siga estes passos:

1. Aceda ao sítio oficial: https://code.visualstudio.com/.
2. Clique no botão de transferência relativa ao seu sistema operativo.
3. Execute o ficheiro de instalação transferido.
4. Siga as instruções do assistente de instalação. Recomenda-se selecionar a opção "Add to PATH" durante a configuração.

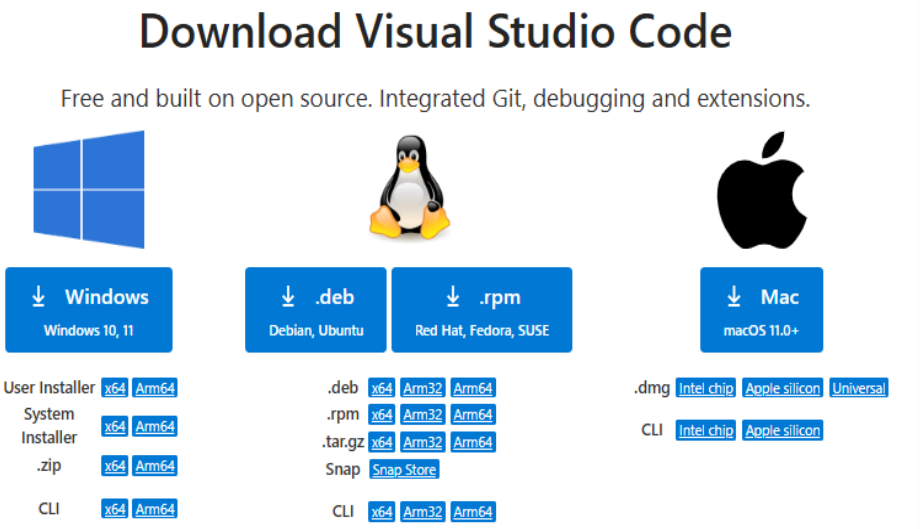


Figura 7. Página de Transferência do VS Código

Após a conclusão da instalação, pode iniciar o **Visual Studio Código** a partir do ambiente de trabalho ou do menu iniciar.

3.2 Instalar a Extensão Python do Visual Studio Código

Para utilizar funcionalidades Python como a conclusão de código, **depuração** e unit testing no **Visual Studio Code**, deve instalar a extensão Python desenvolvida pela Microsoft. Siga estes passos:

1. Clique no ícone **menu** na Barra de Atividades do lado do **VS Code**.
2. Escreva "Python" na barra de pesquisa.
3. Localize a extensão denominada "Python" fornecida pela Microsoft e clique no botão **Instalar**.

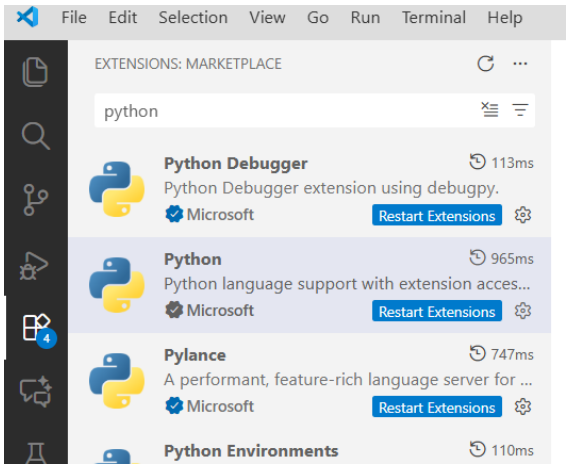


Figure 8. VS Código Python Instalaration.

3.3 Executar o Primeiro Código

Após instalar a extensão, pode criar e executar o seu primeiro ficheiro Python:

1. Selecione File > New File ou prima Ctrl+N.
2. Guarde o ficheiro com extensão .py principal (por exemplo, main.py).

3. Escreva o seguinte código no ficheiro:
4. print("Hello World!")
1. Execute o código clicando no botão **Run** (ícone de reprodução) no canto superior direito ou premindo F5.

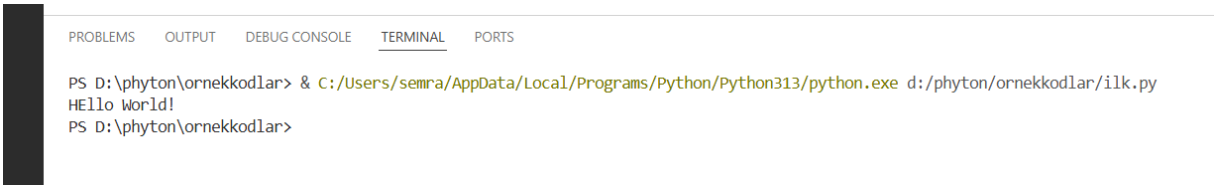


Figura 9. Saída no Ecrã

3.4 Visual Studio Code Inteligência Artificial Toolkit

O AI Toolkit é uma extensão que permite aos programadores integrar e testar aplicações inteligentes utilizando modelos de Inteligência Artificial (IA). Proporciona acesso a vários modelos e simplifica o processo de integração no editor. O AI Toolkit oferece uma integração perfeita com modelos de IA populares, tais como OpenAI, Anthropic, Google e GitHub..

3.4.1. Instalação

Para instalar o **AI Toolkit** no **Visual Studio Code**, siga os passos abaixo:

1. Abra o **Extensions** menu no **VS Code**.
2. Type "AI Toolkit" no the search bar.
3. Localize a extensão e clique no botão **Instalar**.

Após a conclusão da instalação, o ícone do **AI Toolkit** icon will appear no the Activity Bar on the left side of the editor.

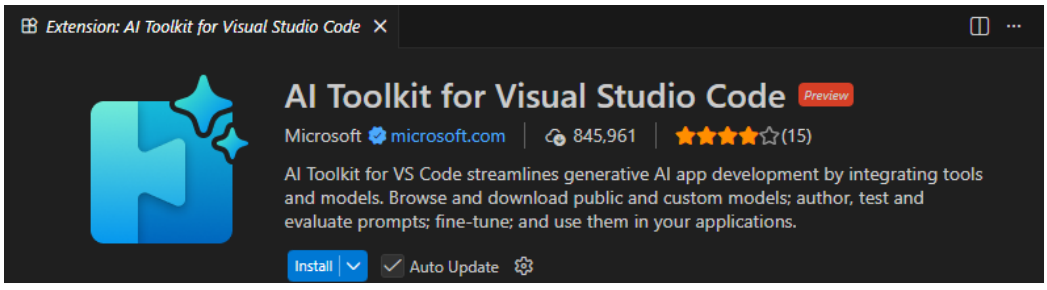


Figure 10. Instalação do AI Toolkit

A Figura 11 mostra a estrutura geral do VS Code Editor. Quando se abre o menu Auto no canto inferior direito, são apresentados os modelos de inteligência artificial no editor e a gestão desses modelos é efetuada a partir deste menu. Quando se seleciona a opção Auto, todos os modelos são ativados e utilizados.

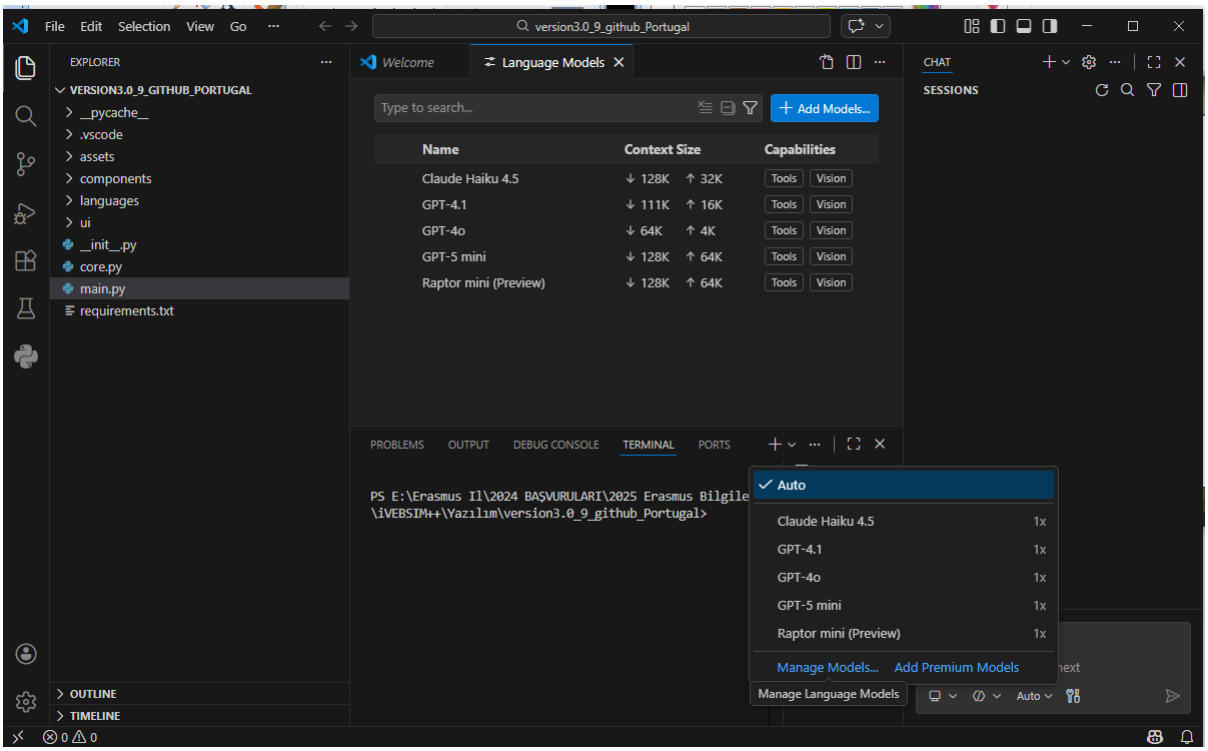


Figura 11. Gestão do Suporte de IA no VS Code.

3.4.2. Funcionamento

Após a conclusão da instalação, pode iniciar o **AI Toolkit** e começar a utilizá-lo. O kit de ferramentas oferece as seguintes vantagens:

- **Análise de Erros:** Os resultados dos erros cometidos podem ser monitorizados instantaneamente, e o processo pode ser repetido regressando ao início.
- **Recolha de Dados:** É possível recolher dados detalhados sobre o desempenho do sistema em condições normais, levando o sistema ao limite.

O **AI Toolkit** permite aos programadores testar os seus modelos num ambiente local e integrá-los facilmente nos seus projetos.

Conforme mostrado na Figura 12, foi solicitado à IA que desenvolvesse um programa para somar os números de 1 a 10, utilizando o menu no canto inferior direito. Posteriormente, a IA recebeu instruções para escrever este código em Python no Editor. A IA completou as tarefas abrindo um ficheiro chamado sum_example.py, guardando o código, executando-o e imprimindo o resultado. Este tipo de integração de IA foi frequentemente utilizado neste projeto.

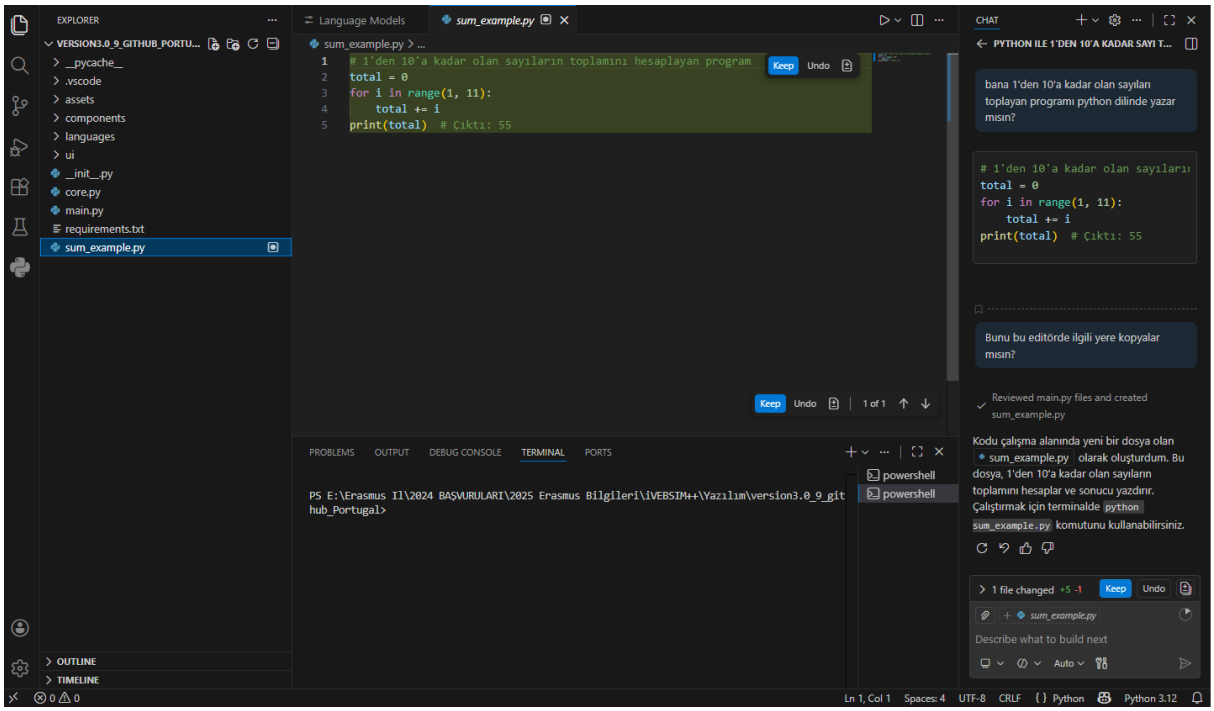


Figure 12. VS Código Inteligência Artificial Usage.

4. VARIÁVEIS PYTHON E REGRAS DE VARIÁVEIS

Em Python, não existe um comando específico para declarar uma variável. Uma variável é criada no momento em que se atribui um valor pela primeira vez. As variáveis são contentores para armazenar valores de dados.

Regras para Nomes de Variáveis:

- O nome de uma variável deve começar por uma letra ou pelo caractere de sublinhado (_).
- Um nome de variável não pode começar com um número.
- A variable name can only contain alpha-numeric characters and underscores principal (A-z, 0-9 e _).
- Os nomes de variáveis são sensíveis a maiúsculas e minúsculas principal (age, Age e AGE são três variáveis diferentes).
- Palavras-chave Python principal (como if, for, while, print) não podem ser usadas como nomes de variáveis.

Exemplos de Atribuições de Variáveis:

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

No exemplo acima, user_name armazena um valor de tipo string, exam_score armazena um valor inteiro e _status armazena um valor booleano. O Python determina automaticamente o tipo de dados com base no valor atribuído.

4.1. Operadores Python

Os operadores são utilizados para realizar operações em variáveis e valores. Python divide os operadores em vários grupos: aritméticos, de atribuição, de comparação e lógicos.

4.1.1. Operadores Aritméticos

Os operadores aritméticos são utilizados com valores numéricos para realizar operações matemáticas comuns.

Operador	Definição	Exemplo
+	Adição	a + b
-	Subtração	a - b
*	Multiplicação	a * b
/	Divisão	a / b
%	Módulo (Resto da divisão)	a % b
**	Exponenciação (Potência)	a ** b
//	Floor Divisão (Divisão sem resto)	a // b

Exemplo Códigos:	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

4.1.2. Operadores de atribuição

Os operadores de atribuição são utilizados para atribuir valores a variáveis. Também podem ser utilizados para realizar uma operação matemática e atribuir o resultado à mesma variável simultaneamente.

Exemplo Códigos	Saída no Ecrã
x = 10 print(x)	10
x = 10 x += 5 print(x)	15
x = 10 x -= 3 print(x)	7
x = 4 x *= 3 print(x)	12
x = 10 x /= 4 print(x)	2.5
x = 10 x //= 4 print(x)	2
x = 10 x %= 3 print(x)	1
x = 2 x **= 3 print(x)	8

Operador	Exemplo	Descrição
=	a = 2	É atribuído o valor 2 à variável a.
+=	a += 2	Adiciona 2 à variável a e atribui o resultado novamente a a. (Equivalente a a = a + 2)
-=	a -= 2	Subtrai 2 da variável a e atribui o resultado de volta a a. (Equivalente a a = a - 2)
*=	a *= 2	Multiplica a variável a por 2 e atribui o resultado novamente a a. (Equivalente a a = a * 2)
/=	a /= 2	Divide a variável a por 2 e atribui o resultado novamente a a. (Equivalente a a = a / 2)
%=	a %= 2	Calcula o módulo da variável a dividindo-a por 2 e atribui o resultado à variável a. (Equivalente a a = a % 2)
**=	a **= 2	Eleva a variável a à potência de 2 e atribui o resultado novamente à variável a. (Equivalente a a = a ** 2)
//=	a //= 2	Realiza a divisão por inteiro da variável a por 2 e atribui o resultado de volta à variável a. (Equivalente a a = a // 2)

4.1.3. Comparison Operadors

Os operadores de comparação são utilizados quando se pretende comparar os valores de duas variáveis e realizar uma ação com base no resultado.

Operador	Definição	Exemplo
==	Igual a	a == b
!=	Diferente de	a != b
<	Menor que	a < b
>	Maior que	a > b
<=	Menor que ou igual	a <= b
>=	Maior que ou igual	a >= b

Exemplo Códigos	Saída no Ecrã
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 Operadors

Os operadores lógicos são utilizados para decidir o fluxo do programa verificando os valores de duas ou mais variáveis.

Operador	Exemplo	Descrição
and	a < 3 and b >= 5	Devolve **True** se todas as condições forem verdadeiras. Neste exemplo, se a variável a for menor que 3 e a variável b for igual ou maior que 5, retorna **True**.
or	a < 3 or b > 4	Devolve **True** se pelo menos uma das condições for verdadeira. Neste exemplo, basta que a seja menor que 3 ou que b seja maior que 4 para que seja devolvido **True**.
not	not(a < 3)	Utilizado para inverter o resultado principal (se for **True**, passa a ser **False**; se for **False**, passa a ser **True**).

Exemplo Códigos	Saída no Ecrã
a = 10 b = 5 print(a<3 and b>=5)	False
a = 10 b = 5 print(a<3 and b>=5)	True
durum = True print(not durum)	False

4.2. Tipos de Dados Python

Em Python, existem geralmente tipos de dados como **string** (alfanumérico), **números** principal (numérico), **booleano**, **lista**, **tuplo**, **dicionário** e **conjunto**. Os tipos de variáveis que mais utilizaremos na fase inicial estão listados de seguida.

4.2.1. Tipo de Dados String (alfanumérico)

Os tipos de dados de cadeia de caracteres são utilizados para armazenar expressões textuais. Os valores são escritos entre aspas simples (‘ ’) ou aspas duplas (“ ”). Os caracteres podem ser letras (t, c), números (1, 9, 2, 3) ou símbolos especiais (&, /).

```
name=“Mert”
surname=’Yılmaz’
```

4.2.2. Tipo de Dados Numérico

Utilizado para armazenar valores numéricos. Existem três tipos numéricos principais:

- **int (Inteiro):** Utilizado para números inteiros (e.g., 5, -10, 100).
- **float (Ponto Flutuante):** Utilizado para números decimais (e.g., 3.14, -0.5).
- **complex:** Utilizado para números complexos (e.g., 3+5j).

Exemplo:

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

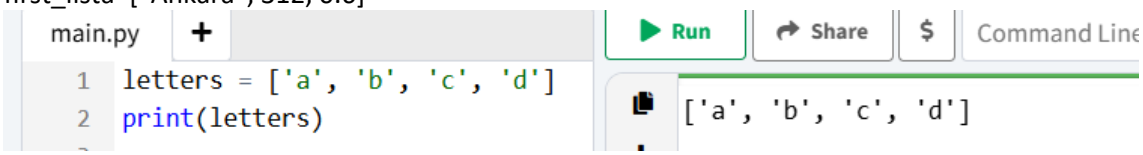
4.2.3. Listas em Python

As listas são utilizadas para armazenar vários itens numa única variável. As listas são definidas utilizando parênteses retos [ ]. São ordenadas, modificáveis e permitem valores duplicados. É possível armazenar diferentes tipos de dados, tais como int, float e string, numa única lista. As listas são utilizadas para armazenar vários dados numa estrutura ordenada e modificável.

Exemplos:

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

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

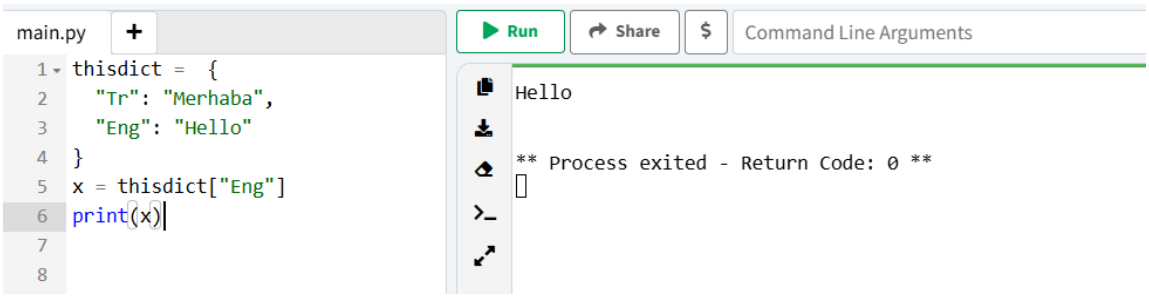


4.2.4. Dicionário

Os dicionários são utilizados para armazenar valores de dados na forma de pares chave:valor. Um dicionário é uma coleção ordenada (a partir do Python 3.7), mutável e que não permite duplicados. Os dicionários são escritos entre chaves { }.

Exemplos:

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



5. ESTRUTURAS DE DECISÃO

As estruturas de decisão permitem ao programa realizar diferentes ações com base no cumprimento de uma determinada condição.

5.1. Decision Structure – IF..ELIF

A instrução «if» é utilizada para executar um bloco de código apenas se uma determinada condição for verdadeira. Se a primeira condição for falsa, a instrução «elif» (ou «else if») permite-nos verificar várias outras condições.

Sintaxe:

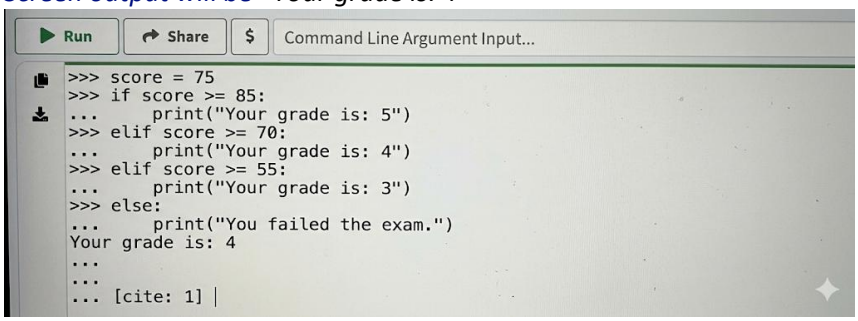
```
if condition:
    # Código to execute if condition is true
elif another_condition:
    # Código to execute if the first condition is false
    # and this condition is true
else:
    # Código to execute if none of the above conditions are true
```

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Cofinanciado pelo Programa Erasmus+ da União Europeia

Exemplo Código:

```
score = 75

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

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

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

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

Screen output will be *Your grade is: 4*

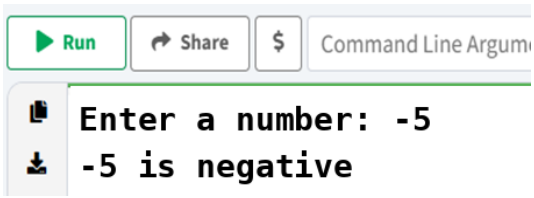


Exemplo Código:

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

O programa solicita que seja introduzido um número através do teclado.

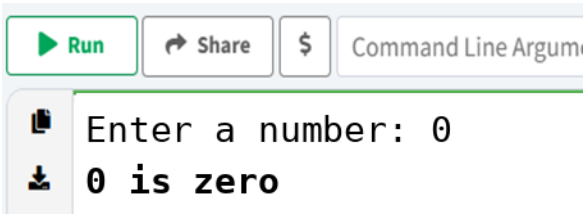
Por exemplo, se assumirmos que o valor -5 é introduzido na variável «a», o programa começará a verificar o bloco «if».



- O valor da variável A é maior que 0? Como -5 não é maior que 0, passamos para a outra expressão de condição.
- • A < 0: o valor da variável A é inferior a 0? Sim, uma vez que a condição é verdadeira, o comando print("..") será executado. Irá imprimir "-5, o número é negativo",

Por exemplo, digamos que o valor 0 é atribuído à variável «a».

- O valor da variável A é maior que 0? Como 0 não é maior que 0, passamos para a outra condição.
- A<0, O valor da variável A é menor que 0? Não, 0<0 não é verdadeiro. Portanto, como nenhuma das condições é cumprida, passa para o bloco else. <0,



- print("..") O comando será executado. Irá exibir «0 é zero» no ecrã.

6. ESTRUTURAS DE CICLO

As estruturas de ciclo são utilizadas para repetir um bloco específico de código várias vezes. Em vez de escrever o mesmo código repetidamente, utilizamos ciclos para realizar tarefas repetitivas de forma eficiente. Os ciclos ajudam a tornar o nosso código mais simples, mais compreensível e mais curto. Utilizamos as estruturas de ciclo For, While e Do While.

6.1. Ciclo FOR

O for de ciclo é utilizado para percorrer um objeto de sequência (como uma lista, um tuplo, um dicionário, um conjunto ou uma cadeia de caracteres). É geralmente utilizado quando o número de repetições é conhecido de antemão.

Sintaxe

for variable in sequence:

```
  # Código to be executed
```

Utilização da função range() com o FOR:

Para percorrer um conjunto de código um número específico de vezes, podemos utilizar a função range(). A função range() devolve uma sequência de números, começando por 0 por predefinição, incrementando-se em 1 (por predefinição) e terminando num número especificado.

Sintaxe

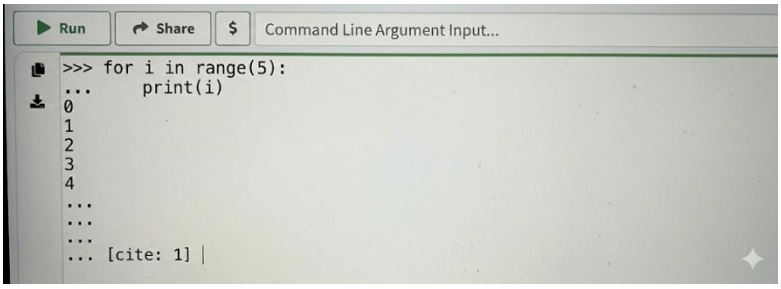
A função range() pode aceitar até três argumentos: range(início, fim, passo)

Parâmetros

- **início (Opcional):** O valor inicial da sequência. Se omitido, o valor padrão é 0.
- **fim (Obrigatório):** O número no qual a sequência termina. Note que a sequência não inclui este número (termina em fim - 1).
- **passo (Opcional):** O incremento (ou decremento) entre cada número da sequência. Se omitido, o valor padrão é 1.

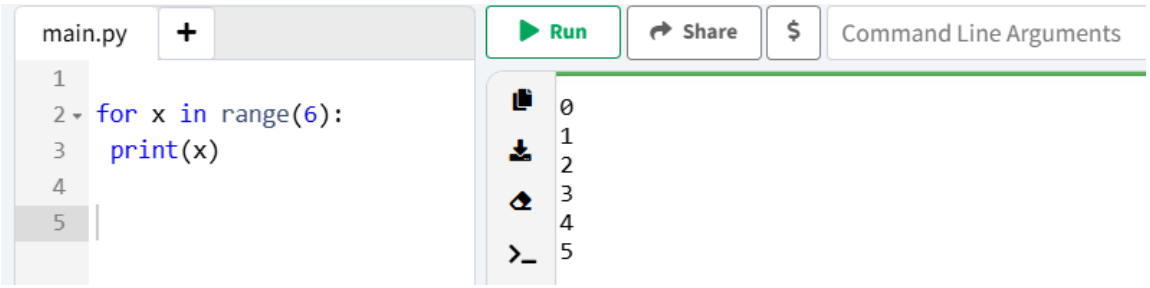
Exemplo 1:

```
for i no range(5):
 print(i)
```



Nota: O código acima irá imprimir os números de 0 a 4. O número 5 não está incluído.

Exemplo 2: O código seguinte apresenta os números de 1 a 6 no ecrã.



Exemplo 3: No código abaixo, os valores são apresentados em incrementos de 100, de 50 a 350.

Se executarmos passo a passo:

• É atribuído o valor 50 à variável contador x e verifica-se o valor da variável x. Será que x <= 350? Será que x é menor que 350? Sim, a nossa condição é satisfeita. E entra no ciclo. Combina a cadeia de caracteres “Number =” com o valor da variável x e apresenta “Number = 50” no ecrã. Regressa ao início do ciclo.
• O valor da variável contador x é aumentado em 100. O valor da variável x passa a ser 150. É x <= 350? Ou seja, é 150 < 350? Sim, a nossa condição é satisfeita. Entra no ciclo. O comando print é executado novamente, combinando o valor da variável x com a expressão de texto constante e exibindo “Number = 150” no ecrã. Ele retorna ao início do ciclo.
• O valor da variável contador x é aumentado em 100. O valor da variável x passa a ser 250. É x < 350? Ou seja, é 250 < 350? Sim, a nossa condição é satisfeita. Entra no ciclo. O comando `print again` é executado, combinando o valor da variável `x` com a expressão de texto constante, imprimindo `Number = 250` no ecrã. Em seguida, regressa ao início do ciclo.

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Cofinanciado pelo Programa Erasmus+ da União Europeia

• Incrementa o valor da variável contadora `x` em 100. O valor da variável `x` passa a ser 350. Será que x < 350? Ou seja, será que 350 < 350? Não, a nossa condição não é satisfeita porque é igual. Saímos do ciclo. Como não há comandos para executar fora do ciclo, o nosso programa termina. O ecrã ficará como na imagem acima.

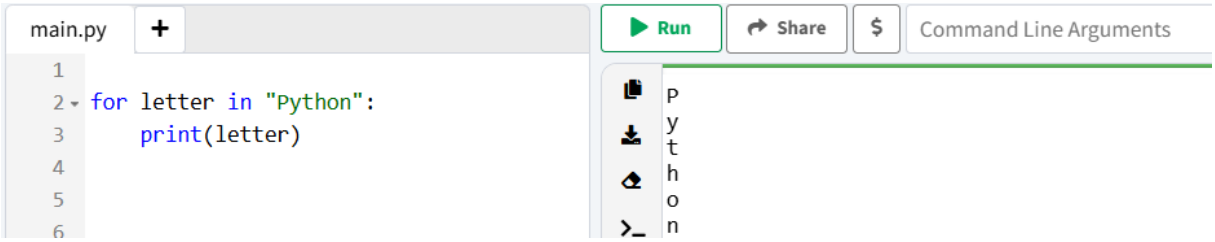
Assim, executamos os comandos dentro do ciclo for 3 vezes até que a nossa condição seja satisfeita. Portanto, o número de iterações do ciclo for varia dependendo do valor inicial e da condição. Os comandos dentro do ciclo continuam a ser executados até que a condição seja satisfeita.

Código	Saída	Descrição
range(5)	0, 1, 2, 3, 4	Começa em 0, para antes de 5.
range(2, 6)	2, 3, 4, 5	Começa em 2, para antes de 6.
range(0, 10, 2)	0, 2, 4, 6, 8	Começa em 0, incrementa de 2 em 2.
range(5, 0, -1)	5, 4, 3, 2, 1	Conta decrescentemente de 5 a 1.

Utilizar o Comando "In" com o Ciclo For

O comando «**in**» é utilizado para verificar se um valor existe numa sequência (como uma lista, uma cadeia de caracteres ou um intervalo). Funciona como uma ponte que permite ao ciclo selecionar cada elemento da sequência, um a um, e atribuí-lo à variável do ciclo.

Utilização com uma cadeia de caracteres (texto): Um ciclo «for» também pode percorrer cada caractere de uma cadeia de caracteres.

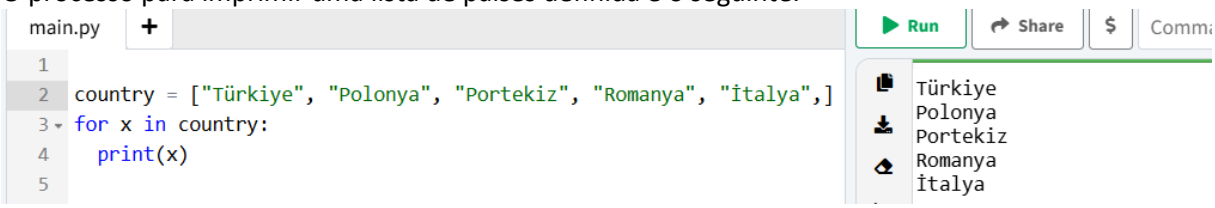


Saída: P, y, t, h, o, n (cada um numa nova linha)

Utilização com uma Lista: Quando utilizado com uma lista, o comando «in» garante que a variável «ciclo» assuma sequencialmente o valor de cada elemento da lista.

Exemplo com a List:

O processo para imprimir uma lista de países definida é o seguinte:



TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY
Cofinanciado pelo Programa Erasmus+ da União Europeia

```
main.py
sensors = ["Distance", "Temperature", "Humidity"]
for x in sensors:
    print(x + " sensor active.")
```

```
Distance sensor active.
Temperature sensor active.
Humidity sensor active.
```

Exemplo com números: O sistema apresenta na tela os números pares de uma determinada sequência numérica.

```
main.py
numbers=[8,9.13,17,16,18,25,22]
for number in numbers:
    if number%2==0:
        print(number)
```

```
8
16
18
22
```

Bloco «else» no ciclo «for»: A palavra-chave «else» num ciclo «for» especifica um bloco de código a ser executado quando o ciclo terminar.

```
main.py
for xin range(3):
    print (x)
else:
    print( "Loop successfully completed!)
```

```
0
1
2
Loop successfully completed!
```

Verificação de pertença: Além dos ciclos, a palavra-chave «in» também pode ser utilizada em instruções «if» para verificar a presença de um elemento.

```
main.py
colors = ["red", "blue", "green"]
if "red" in colors:
    print("Red is in the list!")
```

```
Red is in the list!
```

6.2. Ciclo WHILE

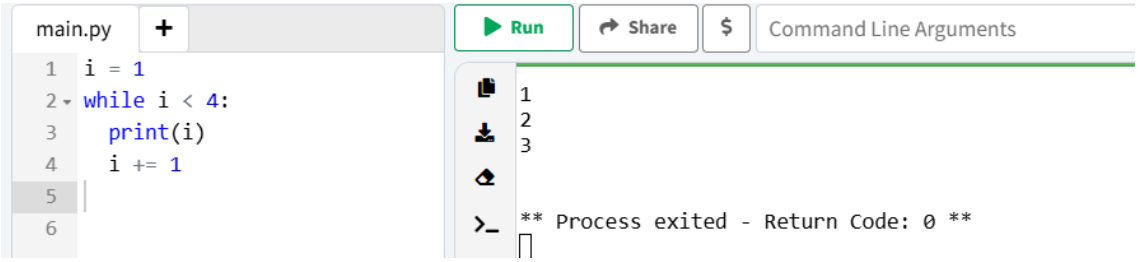
O ciclo while é utilizado para executar um conjunto de instruções enquanto uma condição for verdadeira. Ao contrário do ciclo for, o ciclo while é geralmente preferido quando o número de repetições não está predeterminado. O ciclo termina assim que a condição deixar de ser satisfeita e o programa continua a ser executado a partir do ponto em que o ciclo terminou.

Sintaxe:

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

Exemplo

O código abaixo exibe os números de 1 a 4 no ecrã.



CICLO WHILE PASSO A PASSO (i < 4)

1. **Inicialização:** O variable i starts at **1**.
2. **Iteração 1:** Como 1 < 4 é **True**, imprime **1** e i torna-se **2**.
3. **Iteração 2:** Como 2 < 4 é **True**, imprime **2** e i torna-se **3**.
4. **Iteração 3:** Como 3 < 4 é **True**, imprime **3** e i torna-se **4**.
5. **Terminação:** Agora i é **4**. Como 4 < 4 é **False**, o ciclo para.

Resumo da Saída: 1, 2, 3

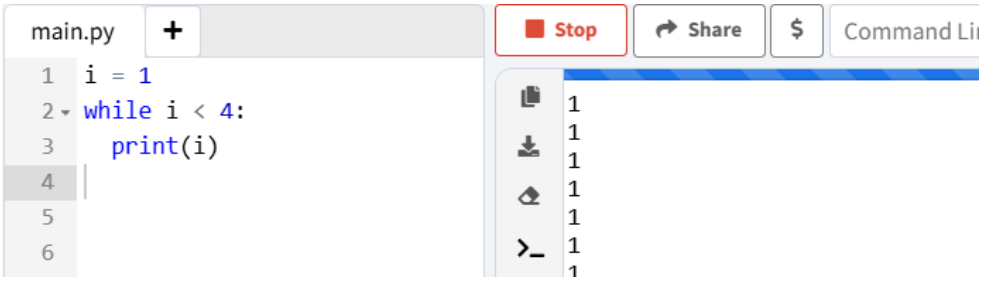
Se aumentarmos o valor da condição aqui, começando com i=1:

• Para i<=10, os comandos dentro do ciclo serão executados 10 vezes,

• Para i<=100, os comandos dentro do ciclo serão executados 100 vezes.

• Para i<5, os comandos dentro do ciclo serão executados 4 vezes. principal (Porque não existe aqui a expressão <=, o nosso ciclo não será executado para i=5.)

Nota: No exemplo abaixo, o ciclo continua enquanto a instrução «i += 1» não for incluída; e, como o valor de i não é incrementado, a condição permanecerá sempre verdadeira, resultando num «ciclo infinito». Prima Stop para parar.



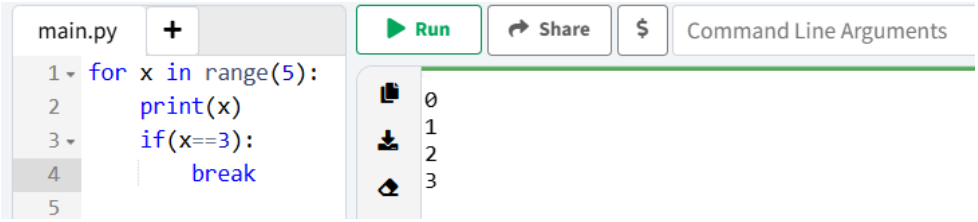
6.2.1. O Break Statement

Com a instrução break, podemos parar o ciclo mesmo que a condição while ainda seja verdadeira. Isto é particularmente útil quando se pretende sair do ciclo quando ocorre uma condição secundária específica.

Exemplo Código:

Without Break, Saída: 1, 2, 3,4 principal (O ciclo terminates when i reaches 4).

With Break, Saída: 1, 2, 3 principal (O ciclo terminates when i reaches 3).



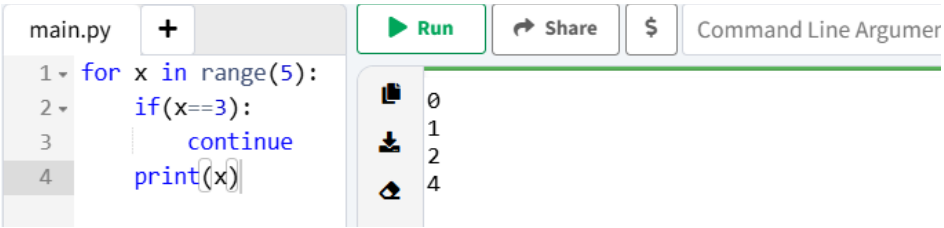
6.2.2. O Continue Statement

Com a instrução continue, podemos parar a iteração atual do ciclo e avançar para o próximo ciclo. Ao contrário do break, não termina o ciclo inteiro.

Exemplo Código:

No código abaixo, verifica-se o valor de x e, se for igual a 3, é executado o comando `continue`, voltando ao início do ciclo. Observando a saída, pode-se ver que o valor 3 não é apresentado no ecrã por este motivo.

Saída: 1, 2, 4, 5, 6 (O número 3 é ignorado).



CREATION OF SOFTWARE DE SIMULAÇÃO

7. SOFTWARES DE SIMULAÇÃO

Os softwares de simulação são programas informáticos que proporcionam uma experiência realista ao utilizador, imitando um sistema físico, um processo ou um dispositivo do mundo real num ambiente digital. Estes softwares permitem-nos prever como um sistema reagirá a estímulos específicos, recorrendo a modelos matemáticos e algoritmos complexos. São geralmente utilizados para fins educativos e de engenharia; assim, cenários que envolvem custos elevados, são perigosos ou de difícil acesso podem ser experimentados repetidamente num espaço virtual seguro e controlado.

Benefícios Básicos dos Softwares de Simulação

- **Segurança:** Permite trabalhar em ambientes sem riscos em situações que envolvem riscos vitais na vida real (alta tensão, alta pressão, etc.).
- **Poupança de Custos:** Previne o desgaste de equipamentos reais e elimina despesas operacionais como o combustível.
- **Análise de erros:** É possível monitorizar instantaneamente os resultados dos erros cometidos e reiniciar o processo para tentar novamente.
- **Recolha de dados:** Fornece dados detalhados sobre o desempenho do sistema em situações extremas, levando-o ao limite.

7.1. Softwares de Simulação de Controlo Industrial

Os sistemas de simulação de controlo eletrónico industrial são softwares que permitem conceber e testar processos de automação complexos e circuitos elétricos num ambiente digital, sem necessidade de instalação física. Apresentam inúmeras vantagens, conforme indicado abaixo:

- **Segurança:** Reduz a zero o risco de choque elétrico, explosão ou danos materiais que possam ocorrer em painéis reais. Permite experimentar sem receio de cometer erros.
- **Poupança de custos:** Permite montar circuitos complexos sem a necessidade de adquirir fisicamente motores, interruptores e contactores dispendiosos.
- **Velocidade de aprendizagem:** Acelera significativamente o processo de aprendizagem, pois mostra instantaneamente como o conhecimento teórico se traduz na prática.
- **Requisitos Baixos do Sistema:** Pode funcionar sem problemas mesmo em computadores muito antigos.
- **Desenvolvimento Lógico:** Estes programas minimizam as margens de erro. É o melhor ponto de partida para estabelecer a "lógica de controlo" antes de avançar para a programação PLC principal (Controlador Lógico Programável).

Para além das vantagens, o pessoal técnico deve saber que;

- **Problema de atualização do programa:** O software é bastante antigo e não abrange totalmente os principais

desenvolvimentosdos sistemas de automação modernos (PLCs modernos, servoacionamentos, etc.).

- **Número Limitado de Elementos:** A sua biblioteca pode ser insuficiente para projetos muito profissionais.
- **Simplicidade Visual:** A sua interface continua um pouco «retro» para os padrões atuais. Pode não oferecer visuais dinâmicos tão detalhados como os seus principais concorrentes mais avançados (tais como o Cade Simu).
- **Não Cobre Erros Físicos:** Não consegue simular falhas físicas da vida real como folgas em cabos, oxidação ou interferência magnética.

Tendo isto em conta, ao criar um software de simulação — ao qual passaremos a chamar iVEBSIM+ a partir de agora:

- **Ampla biblioteca:** Inclui elementos de controlo clássicos, tais como contactores, relés temporizados, botões, motores assíncronos e vários sensores.
- **Simulação Dinâmica:** Após montar o circuito, pode ver como o sistema funciona em tempo real premindo o botão «Play».
- **Verificação de Erros:** O software avisa-o quando é feita uma ligação errada ou ocorre um curto-circuito.
- **Suporte Multi-Idioma:** Como é um software local, oferece uma interface completamente em turco.
- **Lógica de Arrastar e Largar:** O design de circuitos é feito com uma interface visual em vez de linguagens de codificação complexas.

Características	**iVEBSIM+**	**Software Profissional (TIA Portal, etc.)**
Finalidade de Uso:	Formação e lógica básica	Aplicação Industrial
Nível de Dificuldade:	Muito Fácil	Difícil / Requer Experiência
Custo	Gratuito / Acessível	Muito Alto

Tabela 1. Comparação de Software Industrial

CRIAÇÃO DA INTERFACE DO SIMULADOR

8.1 O que é o Tkinter?

As principais bibliotecas de interface gráfica do utilizador (GUI) próprias dessa linguagem de programação são utilizadas na criação de uma interface para um software.

O Tkinter é a interface gráfica de utilizador padrão da Linguagem de Programação Python. É construído sobre o toolkit Tcl/Tk e é instalado automaticamente com a Linguagem de Programação Python.

É pequeno, leve e rápido. Além de ser independente da plataforma (Windows, macOS, Linux), o mais importante é que é fácil de aprender e utilizar. **Código Fonte:** Lib/tkinter/__init__.py

8.1.1. Arquitetura

Tcl/Tk não é uma única biblioteca; consiste em vários módulos diferentes, cada um com funcionalidade separada e a sua própria documentação oficial. As versões binárias do Python também oferecem um módulo plug-in junto com eles.

Tcl

O Tcl é uma linguagem de programação interpretativa dinâmica, tal como o Python. Embora possa ser utilizado de forma independente como linguagem de programação de uso geral, é mais comum ser incorporado em aplicações C como motor de scripts ou como interface para o kit de ferramentas Tk. A biblioteca Tcl possui uma interface C para criar e gerir uma ou mais instâncias do interpretador Tcl, executar comandos e scripts Tcl nessas instâncias e adicionar comandos personalizados implementados em Tcl ou C. Cada interpretador possui uma fila de eventos e existem funcionalidades disponíveis para enviar eventos para ela e processá-los. Ao contrário do Python, o modelo de execução do Tcl foi concebido em torno da multitarefa cooperativa e o Tkinter faz a ponte entre esta diferença.

Tk

O Tk é um pacote Tcl implementado em C que adiciona comandos especiais para criar e manipular os principais componentes da interface gráfica (GUI). Cada objeto Tk contém a sua própria instância do interpretador Tcl, com o Tk carregado nela. Os componentes do Tk são altamente personalizáveis, mas isso tem como contrapartida uma aparência antiquada. O Tk utiliza a fila de eventos do Tcl para criar e tratar eventos da interface gráfica.

Ttk

O Tk Themed (Ttk) é uma família mais recente de widgets Tk que oferece uma aparência muito superior em diferentes plataformas, em comparação com muitos widgets Tk clássicos. O Ttk é distribuído como parte do Tk desde a versão 8.5. As ligações Python são fornecidas num módulo separado, o tkinter.ttk. Internamente, o Tk e o Ttk utilizam os recursos do sistema operativo subjacente; nomeadamente o Xlib no Unix/X11, o Cocoa no macOS e o GDI no Windows.

8.1.2. Widgets Básicos do Tkinter

Um widget (componente de interface) pode, basicamente, ser considerado um componente. Estes executam uma tarefa específica. Uma interface de utilizador Tkinter é composta por widgets individuais. Cada widget é representado como um objeto Python instanciado a partir de classes como ttk.Frame, ttk.Label e ttk.Button.

Widgets Básicos:

- **Frame:** Uma moldura para agrupar outros widgets.
- **Label:** Uma etiqueta para exibir texto ou imagens.
- **Botão:** A clickable.
- **Entry:** Uma entrada de texto de linha única.
- **Text:** Uma área de texto multilinha.
- **Canvas:** Uma área para desenho e gráficos.
- **Menu:** Barras de menu.
- **Scrollbar:** Barras de deslocamento.
- **Checkbutton:** Caixas de verificação.
- **Radiobutton:** Botões de seleção.
- **Listbox:** Caixas de lista.
- **Scale:** Deslizadores.
- **Spinbox:** Seletores de números.

2.2. Criação de Janelas

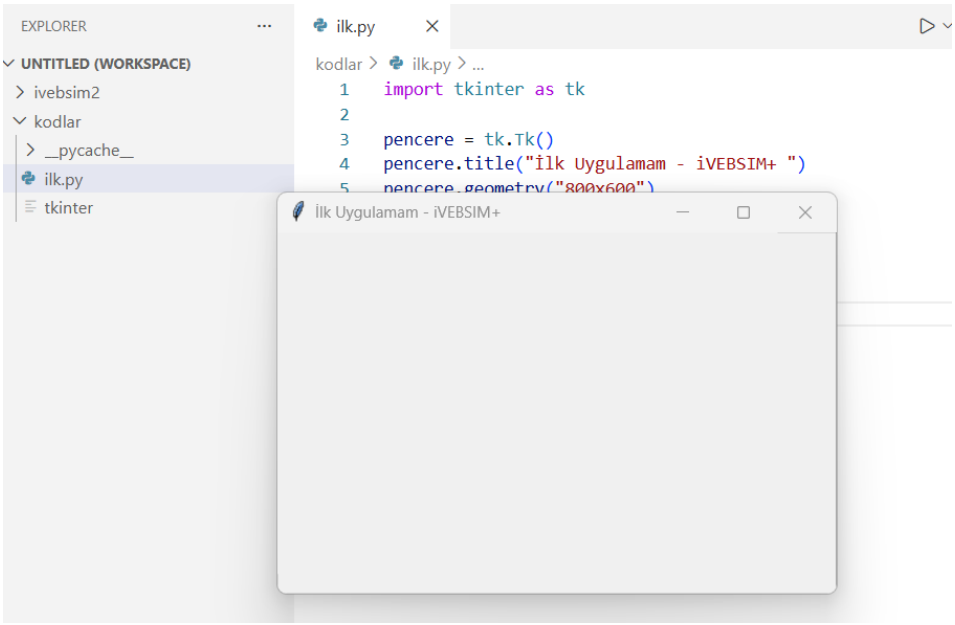
No **Tkinter**, siga estes passos para criar uma janela:

- **Importar Tkinter**.
- Criar o objeto da janela principal principal (**tk.Tk()**).
- Definir as propriedades da janela principal (título, tamanho, posição).
- Adicionar **widgets** principal (**botões**, **etiquetas**, **entrada** campos).
- Bind eventos principal (such as clicking or pressing a key).
- Iniciar o **ciclo** principal (**mainciclo()**).

Aplicação de Exemplo

```
import tkinter as tk

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

Quando executamos os códigos, vemos a nossa página de formulário como se segue.



Primeiro exemplo de aplicação - Análise detalhada

Agora, vamos analisar a nossa primeira aplicação Tkinter passo a passo:

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

   Importa o **Tkinter** módulo com o alias "**tk**".

2. **Janela Principal:**

   pencere = tk.Tk()

- tk.Tk() cria a janela principal.
- Esta janela é o contentor principal para todos os outros **widgets**.
- Cada aplicação pode ter apenas um objeto Tk().

3. **Propriedades da Janela:**

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

- title(): O texto que aparece na barra de título.
- geometry(): Define o tamanho da janela para zero píxeis.

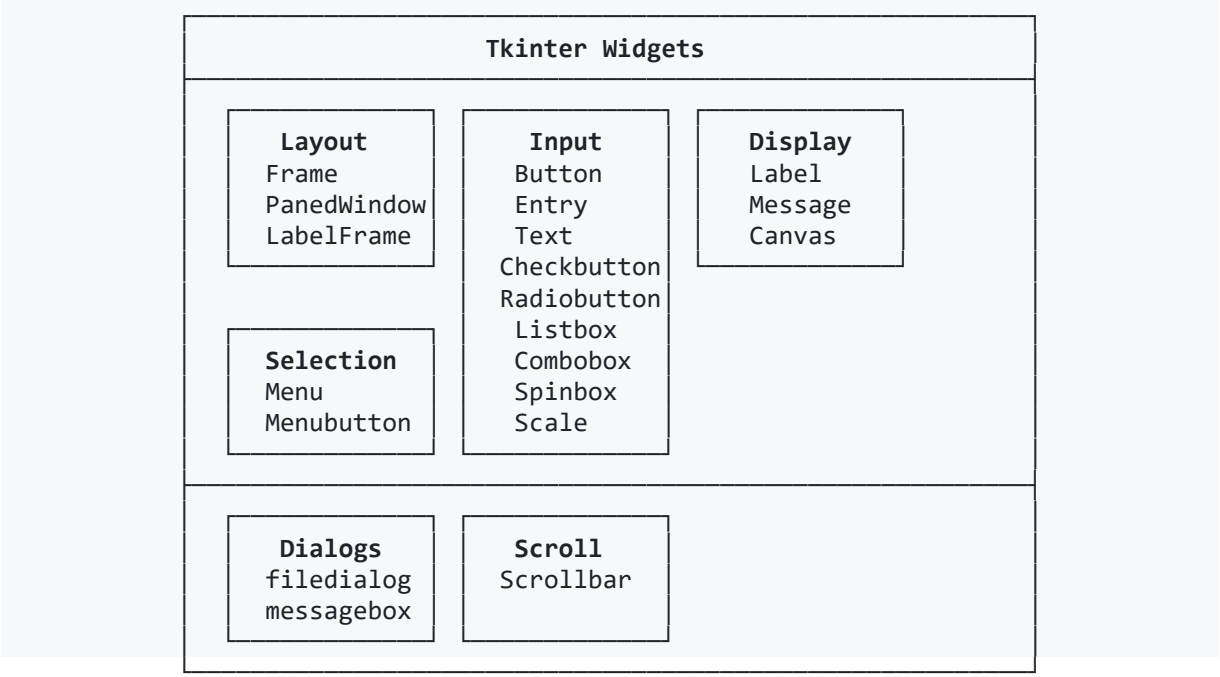
4. **Ciclo Principal:**

```
pencere.mainciclo()
```

- Torna a janela visível.
- Escuta eventos do rato e teclado.
- Mantém a aplicação em execução.
- É um **ciclo** infinito, mas termina quando a janela é fechada.

8.1.3. Adicionar Widgets e Vincular Eventos

Widgets permitem adicionar elementos de formulário como **Frame**, **Label**, **Button**, **Entry**, **Text**, **Canvas**, **Menu**, **Scrollbar**, **Checkbutton** e **Radiobutton** as mentioned no 2.1.2.



Adicionar Etiquetas

Utiliza-se para adicionar texto ao ecrã. O método `Label` da biblioteca Tkinter é utilizado para adicionar etiquetas ao formulário.

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

- text: O texto a ser exibido.
- font: Tipo, tamanho e estilo da fonte.
- bg: Cor de fundo (**background**).
- fg: Cor do primeiro plano (**foreground**).

8.3.2 Adicionar Botões

Cria um botão clicável. É utilizado o método O Button da biblioteca Tkinter.

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

- text: O texto do botão.
- command: A função a ser chamada durante o evento.
- state: estados "normal", "active", "disabled".

8.3.3 Definir Layouts

pack() coloca objetos na janela. Pode determinar como serão posicionados horizontal e verticalmente no formulário.

etiket.pack(pady=20)

buton.pack(pady=10)

- pady: Espaçamaneto vertival (pixels).
- padx: Espaçamento horizontal.
- side: Lado (**TOP**, **BOTTOM**, **LEFT**, **RIGHT**).
- fill: Preencher o espaço vazio (**X**, **Y**, **BOTH**).
- expand: Sem espaços em branco.

Aplicação de Exemplo

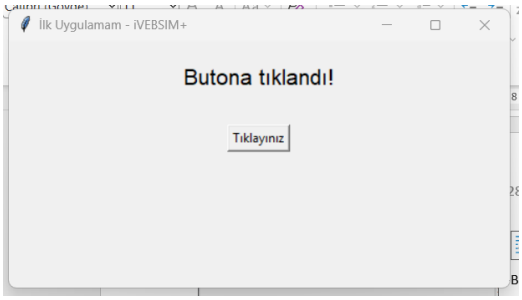
```
import tkinter as tk
# Create main window
pencere = tk.Tk()
pencere.title("İlk Uygulamam - iVEBSIM+")
pencere.geometry("800x600")

# 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 ciclo
pencere.mainciclo()
```

Este código realiza as seguintes operações:

- Carrega a **Tkinter** biblioteca.
- Cria e configura a janela principal.
- Cria uma etiqueta a exibir o texto "Olá Tkinter!".
- Define a função de clique.
- Cria o botão "Tıklayınız".
- Coloca **widgets** na janela.
- Displays the window and listaens for eventos.

Executar o Código

Quando se clica no botão, o formulário aparece como se pode ver na imagem ao lado.



8.3.4 Adicionar Entry

É utilizado para receber entrada de texto de linha única do utilizador.

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

Aplicação de Exemplo

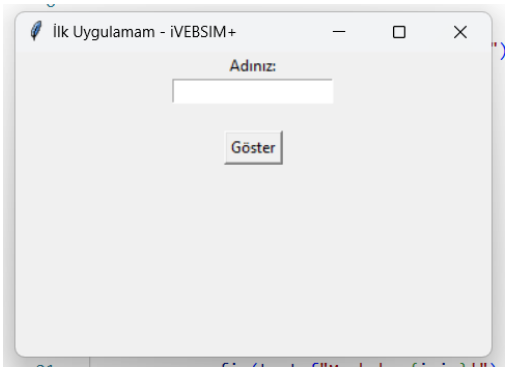
```
import tkinter as tk

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.mainciclo()
```

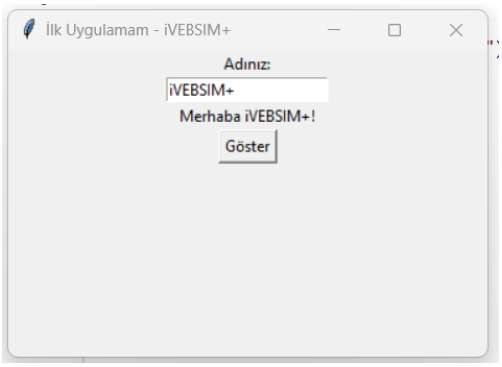
Este código realiza as seguintes operações:

- Carrega a Tkinter biblioteca.
- Cria e configura a janela principal.
- Cria um objeto Label (rótulo) que exibe o texto «O seu nome».
- Cria um objeto Entry (caixa de entrada) para introduzir o nome.
- Cria um Label para exibir o resultado.
- Define a função de clique.
- Cria o botão «Mostrar».
- Exibe a janela e aguarda eventos.

Executarning the Códigos



Quando se clica no botão, o formulário aparece como se pode ver na imagem ao lado.



8.3.5 Adicionar botão de Radio

É utilizado para o utilizador selecionar apenas uma opção de múltiplas escolhas.

```
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 Adicionar botão de verificação

É utilizado para o utilizador selecionar uma ou mais opções das escolhas fornecidas.

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

8.3.7 Adicionar Barra de Deslocamento

É utilizado para adicionar uma barra de deslocamento ao ecrã do formulário.

```
scroll = tk.Scrollbar(pencere)
scroll.pack(side="right", fill="y")
metin = tk.Text(pencere, yscrollcommand=scroll.conjunto)
metin.pack()
scroll.config(command=metin.yview)
```

8.3.8 Adicionar Menu

É utilizado para adicionar uma barra de menu superior.

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

8.3.9 Vincular Eventos

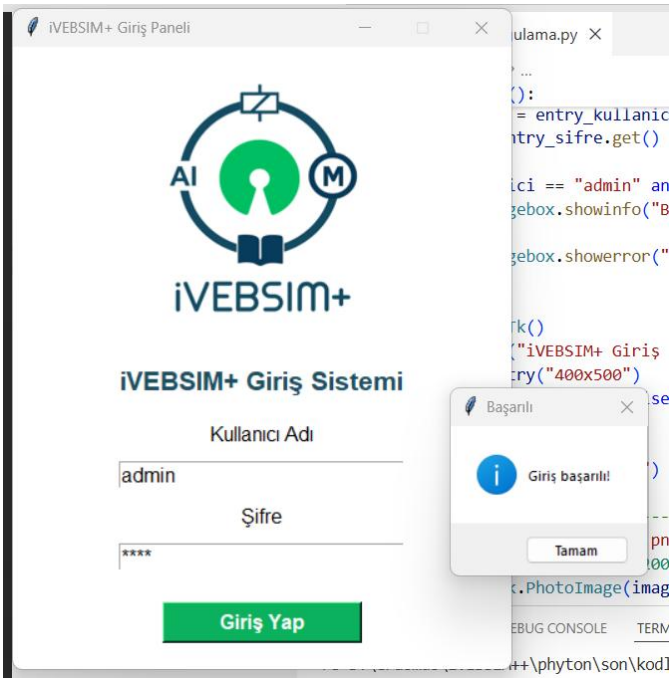
No **widgets**, podemos definir os códigos que queremos executar durante eventos de clique dentro de uma função.

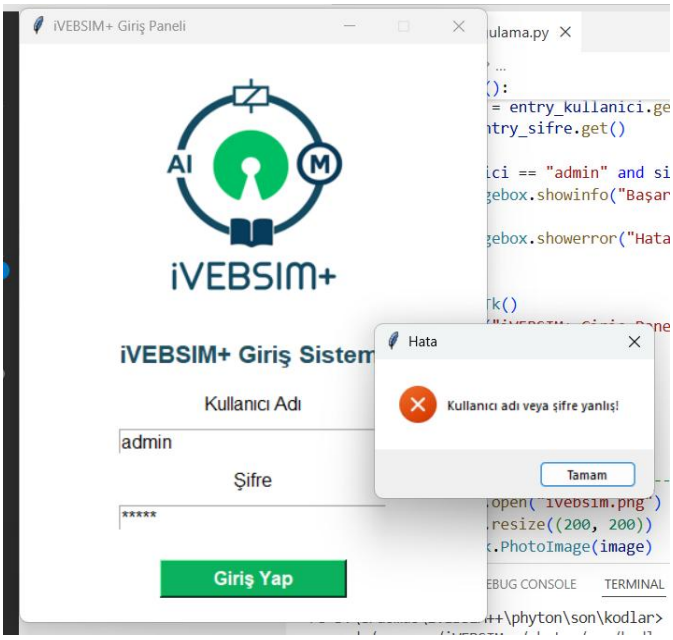
```
 def goster():
    isim = giris.get()
    sonuc.config(text=f"Merhaba {isim}!")
buton = tk.Button(pencere, text="Göster", command=goster)
```

Sample Códigos:

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

# Giriş fonksiyonu
def giris_yap():
    kullanici = entrada_kullanici.get()
    sifre = entrada_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)

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

entrada_sifre = tk.Entry(pencere, font=("Arial", 12),
width=25, show="*")
entrada_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.mainciclo()
```





9. CRIAÇÃO DA INTERFACE DO SIMULADOR

Quando a interface do simulador é iniciada, aparece primeiro um ecrã de boas-vindas. Neste ecrã, constam as instituições que apoiam este programa. Ao clicar com o rato nesta janela, a interface do programa é apresentada no ecrã. No nosso software, o ecrã de boas-vindas é denominado main.py e a janela principal do programa é denominada ui.py.

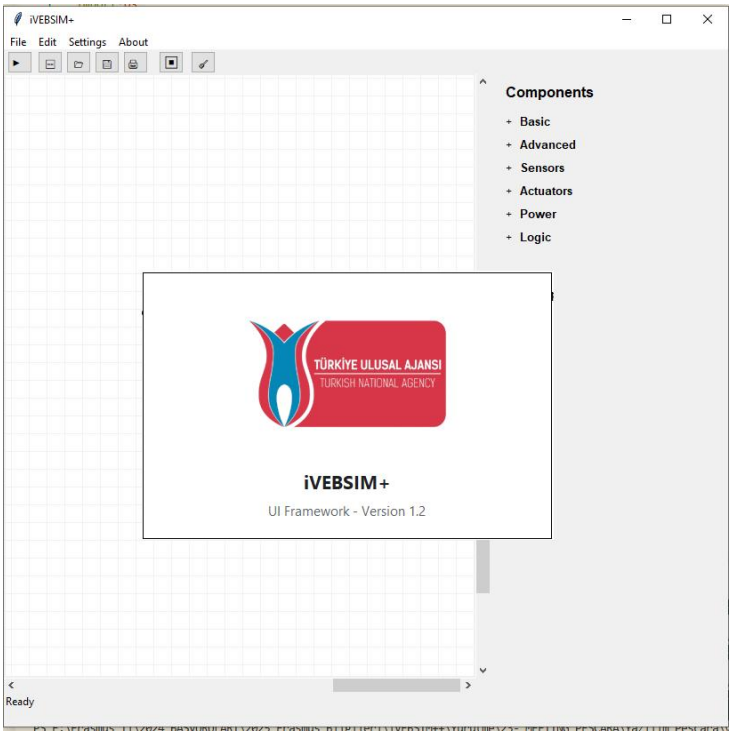


Figura 14 – Vista de software

9.1 Criar uma página inicial

Para iniciar a aplicação e apresentar um ecrã de boas-vindas (splash) ao utilizador, siga estes passos:

A biblioteca Python Tkinter foi adicionada.

```
import os

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

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

O tamanho da janela parece ser fixo.

```
# 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)
```

O conteúdo da janela foi criado.

```
# 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(
```

O tamanho do splash é recalculado e centralizado.

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

Figura 15 – Splash View

O ecrã inicial é acionado por um clique ou pressionando uma tecla, ativando a opção «Continuar». A opção «Continuar» fecha o ecrã de boas-vindas e volta ao ecrã principal.

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

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

O splash é chamado e o ciclo de eventos é iniciado.

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


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

9.2 Criação da infraestrutura da interface

O ficheiro `ui.py` define a classe `App`, que fornece a interface gráfica do utilizador para o software. O software inclui menus, uma barra de ferramentas, uma área de desenho no centro (`Canvas`) e painéis laterais à direita e à esquerda (componente em acordeão). Para criar a interface da aplicação, além dos pontos mencionados anteriormente, seguem-se os seguintes passos:

É criada uma janela de ecrã no ficheiro 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)
```

A barra de menus (Ficheiro / Editar / Configurações / Sobre) é criada com `tk.Menu`. Cada comando do menu está associado a um método (`_new_file`, `_save_file`, etc.).

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

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

A barra de ferramentas é criada utilizando «ttk.Button». São adicionados botões como «Guardar», «Iniciar», «Parar», «Limpar», etc.

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

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

A área principal é configurada com `ttk.PanedWindow` e dividida em dois painéis utilizando `PanedWindow`. O lado esquerdo é definido como `canvas_frame`, contendo a área de desenho (`Canvas`), e o lado direito como `side_panel`, contendo o componente acordeão.

A área da tela é criada com `scrollregion`, e são adicionadas barras de rolagem na horizontal e na vertical. Quando a área da tela é redimensionada, a grelha é atualizada utilizando `configure`. Além disso, ao utilizar coordenadas no ecrã, são utilizados os métodos `canvasx` e `canvasy`. Isto permite a conversão de ecrã para ecrã.

```
 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()))
```

O painel lateral apresenta categorias em estilo acordeão (`básico`, `avançado`, `sensores`, `atuadores`, `alimentação`, `lógica`) – criadas como cabeçalhos e corpos através da função `_add_accordion_section`. É também adicionado um logótipo no rodapé, na parte inferior. As funções do acordeão são controladas através da função `def _add_accordion_section`. O estado de ativo/inativo é controlado através de um «Indicador».

```
    # 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()
```

Os comandos do menu e os métodos auxiliares são definidos separadamente.

* Operações com ficheiros (marcadores de posição): `_new_file`, `_open_file`, `_save_file`, `_save_as_file`

* Operações de edição (placeholders): `_undo`, `_redo`, `_cut`, `_copy`, `_paste`. –

* Configurações: `_show_preferences`, `_change_language`, botão de seleção para ativar/desativar a grelha.

* Sobre nós: `_show_about` → exibe uma janela de informações sobre a aplicação.

```
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 Multilinguismo

O software foi concebido com base no multilinguismo.

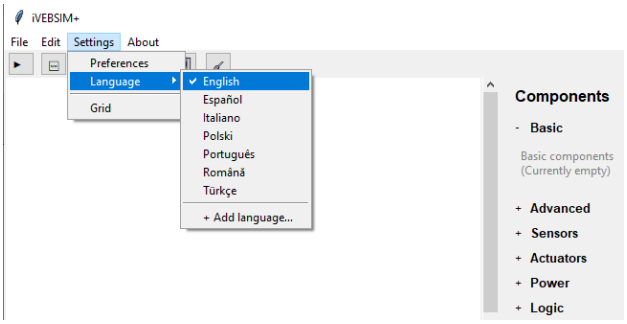


Figure 16 – Multilingualism

A biblioteca i18n do Python é utilizada para permitir o multilinguismo no programa. Quando a aplicação é iniciada (no método __init__), o código de idioma predefinido é definido como «en» (inglês).

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

Os ficheiros de idioma estão armazenados na pasta «languages» no formato JSON (por exemplo, en.json, tr.json, es.json, etc.). Cada ficheiro JSON tem uma estrutura com campos obrigatórios como «code» e «name» (código do idioma e nome de exibição). O ficheiro config.json armazena o idioma selecionado: {“selected_language”: “en”}. Isto permite que a aplicação se lembre do último idioma selecionado quando for reaberta.

Figure 17 – json files

A função de carregamento de traduções (_load_translations) recebe um código de idioma como parâmetro (por exemplo, «tr») e abre o ficheiro languages/{code}.json para carregar o 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
```

O menu «Definições» tem um submenu «Idioma». A função `_populate_language_menu()` analisa todos os ficheiros JSON na pasta «languages». Esta função recupera os campos «code» e «name» de cada ficheiro. Quando o utilizador seleciona um idioma, a função `_on_language_selected()` é chamada e o código do idioma é atualizado.

```
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()
```

Após selecionar um idioma, a interface do utilizador é totalmente reconstruída. A alteração do idioma requer a eliminação e a recriação de todos os componentes. O texto no Tkinter não se altera dinamicamente. Por conseguinte, a barra de menus é removida e toda a interface do utilizador é reconstruída do zero.

```
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. ADICIONAR ELEMENTOS E CRIAR A SIMULAÇÃO

NEXT ROMANIA……

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

Cofinanciado pelo
Programa Erasmus+
da União Europeia

NOTES: