



# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 10 – TKINTER II

# O QUE IREMOS APRENDER

- 01** MÓDULO TTK
- 02** WIDGETS
- 03** TIPOS DE DADOS
- 04** MÃOS NO CÓDIGO

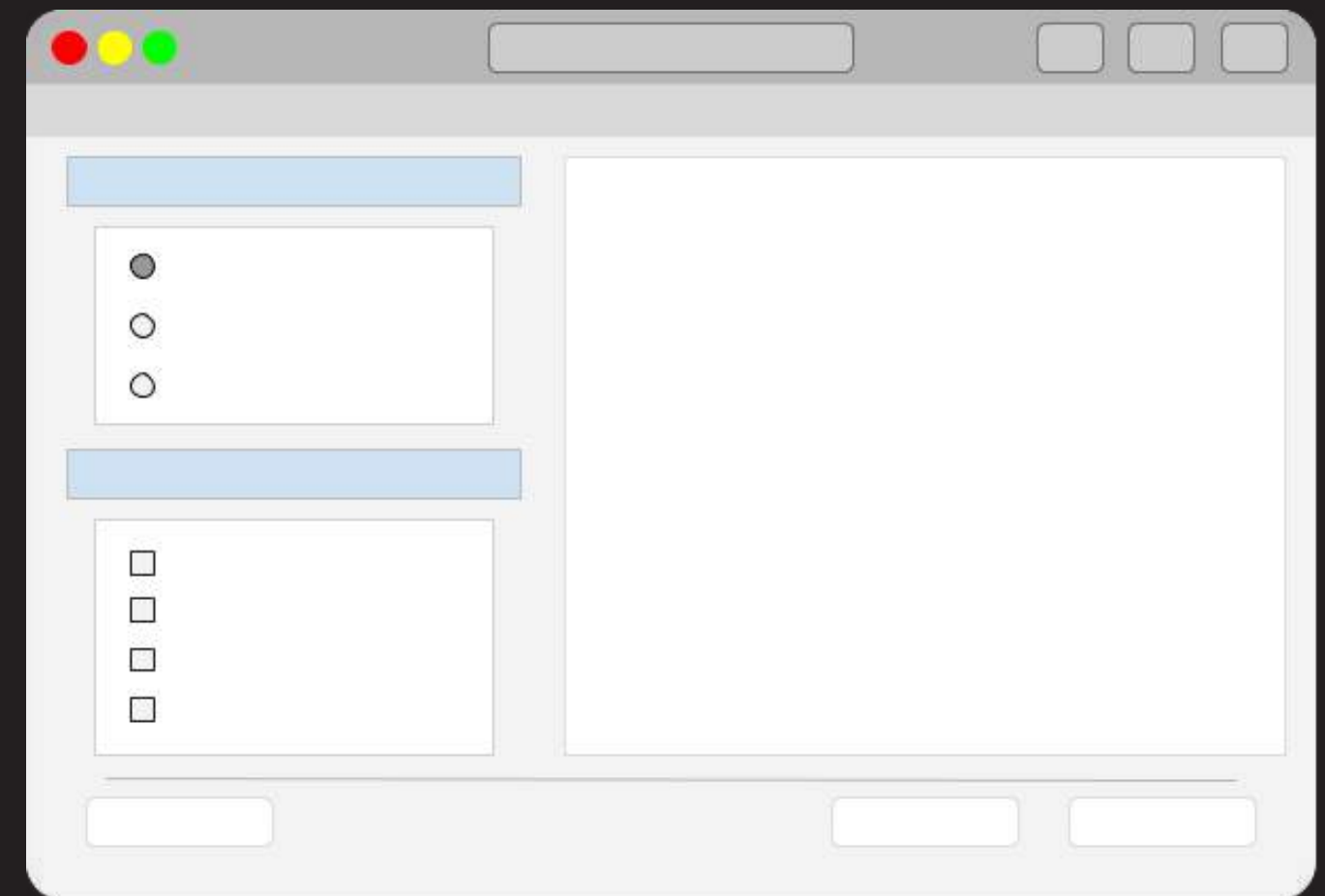
# Tkinter II

Na aula passada, exploramos os conceitos fundamentais de Tkinter, incluindo a criação de interfaces gráficas (GUI), a compreensão de widgets e métodos de posicionamento. Nesta aula, daremos continuidade ao nosso estudo aprofundando-nos no **módulo ttk (themed Tkinter)**. Uma extensão do Tkinter que fornece uma variedade de widgets mais avançados e com um estilo mais moderno e uniforme, tornando as aplicações mais atraentes visualmente.



# O que é o Módulo TTK

O módulo `ttk` (Themed Tkinter) é um aprimoramento do Tkinter. Ele fornece uma série de widgets de interface gráfica adicionais com uma aparência mais moderna e consistente em diferentes plataformas. A principal vantagem do `ttk` é a disponibilidade de widgets temáticos. Esses widgets têm uma aparência nativa mais elegante e sofisticada, adaptando-se ao estilo visual do sistema operacional em uso.



# TTK Widgets

Esse código faz com que vários tkinter.ttk widgets substituam automaticamente os widgets Tk.



```
1  # Para começar a usar Ttk, importe seu módulo:
2  from tkinter import ttk
3
4  # Para substituir os widgets básicos do Tk, a importação deve seguir a importação do Tk:
5  from tkinter import *
6  from tkinter.ttk import *
```

# Widgets

Ttk vem com 18 widgets, doze dos quais já existiam no tkinter: Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale, Scrollbar, e Spinbox.

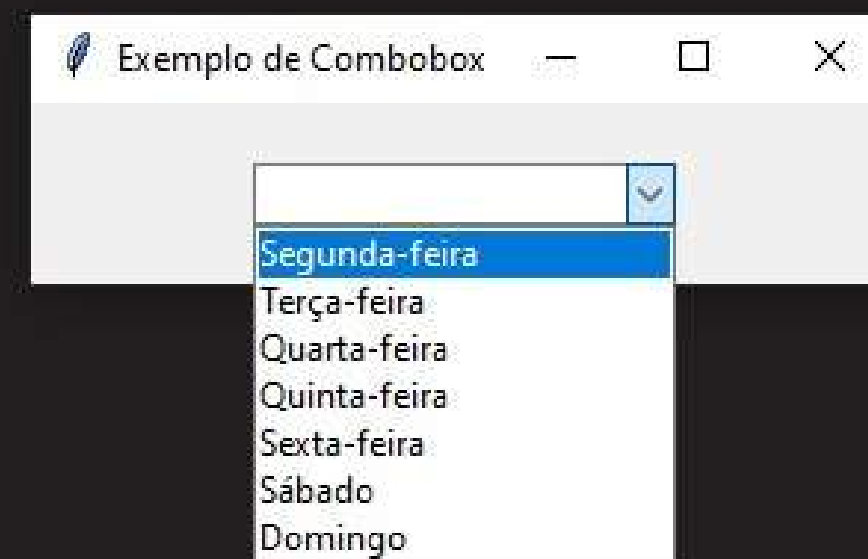
Os outros seis são novos:

- combobox(Combobox)
- separador(Separator)
- notebook(Notebook)
- barra de progresso(Progressbar)
- sizegrip(Sizegrip)
- visualização em cascata e em árvore(Treeview)

# Widgets

O `ttk.Combobox` combina uma caixa de entrada com uma lista suspensa (dropdown) de opções. É útil para seleção de itens em uma lista predefinida. Os usuários podem inserir texto na caixa ou escolher uma opção da lista suspensa

```
1 # Importando Tkinter e Ttk
2 from tkinter import *
3 from tkinter import ttk
4
5 # Criando Janela
6 janela = Tk()
7 janela.title("Exemplo de Combobox")
8
9 # Dias da semana
10 dias_semana = ["Segunda-feira", "Terça-feira", "Quarta-feira",
11               "Quinta-feira", "Sexta-feira", "Sábado", "Domingo"]
12
13 # Criando o Combobox
14 combobox = ttk.Combobox(janela, values=dias_semana)
15 combobox.pack(padx=75, pady=20)
16
17 janela.mainloop()
```





# Widgets

`ttk.Progressbar` é usada para exibir o progresso de uma tarefa. Ela pode ser usada para indicar o andamento de operações, como carregamento de dados ou conclusão de processos. A barra de progresso pode ser horizontal ou vertical.

```
1 import tkinter as tk
2 from tkinter import ttk
3
4 janela = tk.Tk()
5
6 barra_progresso = ttk.Progressbar(janela, orient="horizontal", length=200, mode="determinate")
7 barra_progresso.pack()
8 barra_progresso.start(50) #Inicia a barra de progresso
9
10 janela.mainloop()
```





# Widgets

O `ttk.Separator` exibe uma barra separadora horizontal ou vertical. Uma segunda abordagem para agrupar widgets em uma exibição é colocar uma régua fina horizontal ou vertical entre grupos de widgets. obs: orient pode ser `HORIZONTAL` ou `VERTICAL`

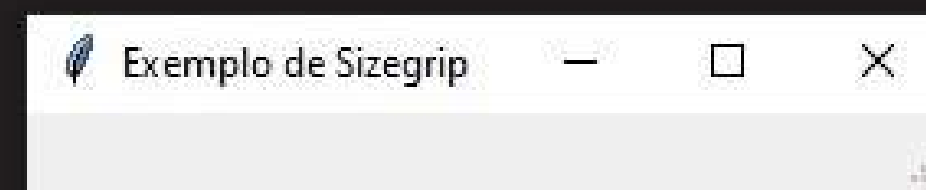
```
1 # Importando Tkinter e Ttk
2 from tkinter import *
3 from tkinter import ttk
4
5 # Criando Janela
6 janela = Tk()
7 janela.title("Exemplo de Separator")
8
9 # Criando label 1
10 label1 = ttk.Label(janela, text="Texto 1")
11 label1.pack(pady=10)
12
13 # Criando separador
14 separador = ttk.Separator(janela, orient=HORIZONTAL)
15 separador.pack(padx=50, ipadx=30)
16
17 # Criando label 2
18 label2 = ttk.Label(janela, text="Texto 2")
19 label2.pack(pady=10)
20
21 janela.mainloop()
```



# Widgets

O `ttk.Sizegrip` (também conhecido como caixa de crescimento) permite ao usuário redimensionar a janela pressionando e arrastando a alça. O `Sizegrip` é útil quando você deseja permitir que os usuários redimensionem manualmente uma janela ou um quadro em seu aplicativo.

```
1 # Importando Tkinter e Ttk
2 from tkinter import *
3 from tkinter import ttk
4
5 # Criando Janela
6 janela = Tk()
7 janela.title("Exemplo de Sizegrip")
8
9 # Configurando a janela para ser redimensionável
10 janela.resizable(True, True) # Por padrão a janela é redimensionável
11
12 # Criando um widget Sizegrip
13 sizegrip = ttk.Sizegrip(janela)
14 sizegrip.pack(anchor=SE, padx=3, pady=3, expand=True)
15
16 janela.mainloop()
```

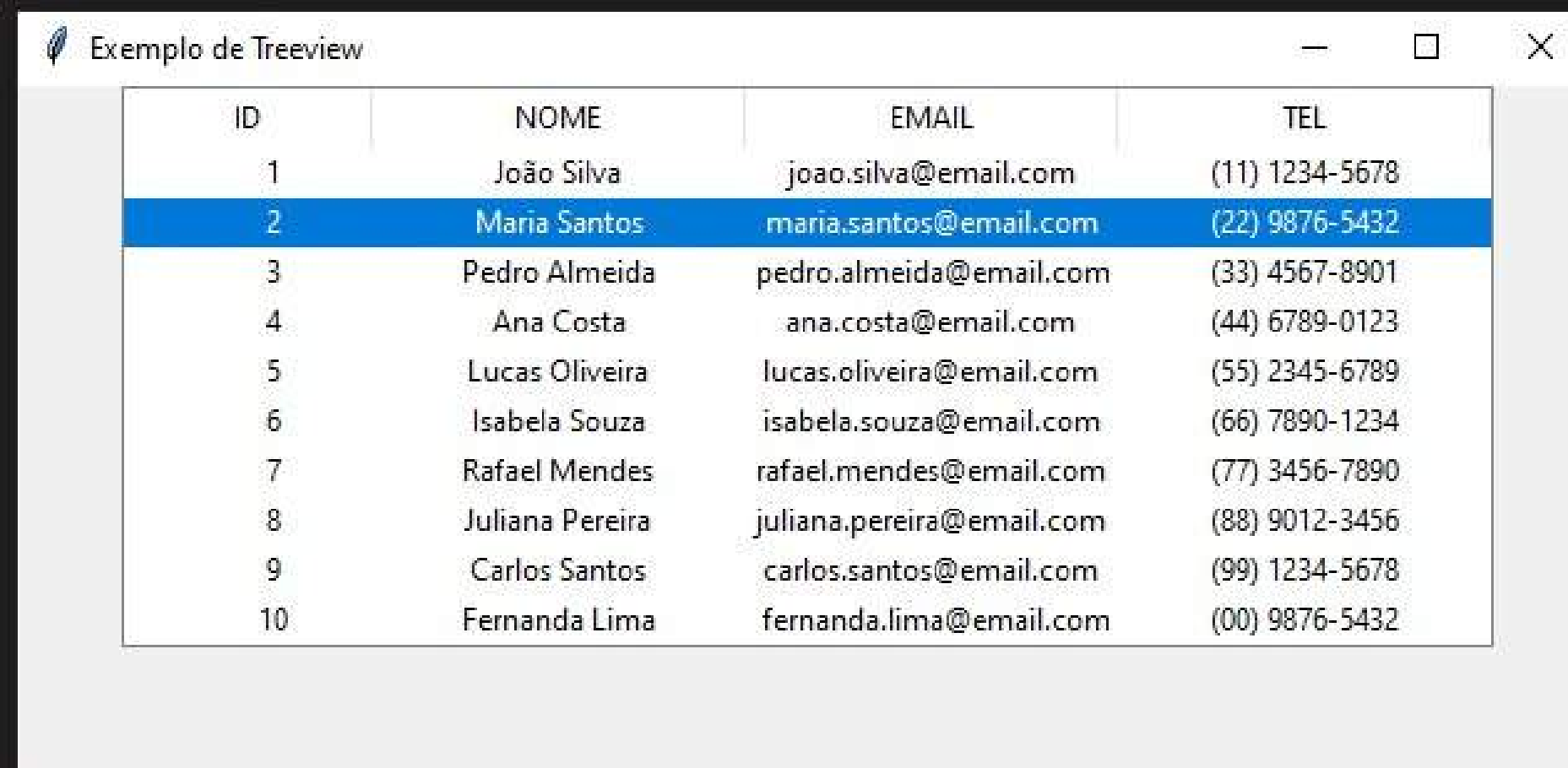


# Widgets

A **visualização em árvore (Treeview)** é um widget de visualização em árvore que exibe uma hierarquia de itens e permite que os usuários naveguem por ela. Como a maioria dos widgets Tk, ele oferece flexibilidade incrível para que possa ser personalizado para atender a uma ampla variedade de situações.

A exibição em árvore pode exibir uma ou mais informações adicionais sobre cada item. Eles são mostrados como colunas à direita da exibição da árvore principal.

# Widgets



ID	NOME	EMAIL	TEL
1	João Silva	joao.silva@email.com	(11) 1234-5678
2	Maria Santos	maria.santos@email.com	(22) 9876-5432
3	Pedro Almeida	pedro.almeida@email.com	(33) 4567-8901
4	Ana Costa	ana.costa@email.com	(44) 6789-0123
5	Lucas Oliveira	lucas.oliveira@email.com	(55) 2345-6789
6	Isabela Souza	isabela.souza@email.com	(66) 7890-1234
7	Rafael Mendes	rafael.mendes@email.com	(77) 3456-7890
8	Juliana Pereira	juliana.pereira@email.com	(88) 9012-3456
9	Carlos Santos	carlos.santos@email.com	(99) 1234-5678
10	Fernanda Lima	fernanda.lima@email.com	(00) 9876-5432

DADOS FICTÍCIOS. TODA E QUALQUER SEMELHANÇA COM DADOS REAIS É COINCIDÊNCIA

obs: Cada linha da Treeview é um item raiz. Está no nível superior da hierarquia do Treeview.

# Widgets

Podemos especificar a lista de colunas usando a `columns` opção de configuração do widget `treeview`, seja ao criar o widget pela primeira vez:



```
1 # tree = ttk.Treeview(janela, columns=("nome", "email", "telefone")) # Outra forma de definir as colunas
2
3 # Definindo as colunas
4 tree["columns"] = ("nome", "email", "telefone")
```

# Widgets

Cada coluna é referenciada por um nome simbólico que atribuímos no `columns`, ou seja, `'nome'`, `'email'` e `'telefone'` são nomes simbólicos para se referir a coluna.

O nome da coluna que será apresentado na Treeview será inserido no `text` durante a definição do cabeçalho da coluna feito através do `tree.heading("nome", text="NOME")`

obs: Por padrão, a primeira coluna é referida por `#0`, as demais são as colunas adicionais (`'nome'`, `'email'`, `'telefone'`).

# Widgets

O método `.column()` é usado para configurar as propriedades de uma coluna específica.

```
tree.column("email", *)
```

Este método receberá o identificador da coluna (nome simbólico) e `*` que referencia as propriedades:

- `width`: largura
- `minwidth`: largura mínima
- `anchor`: alinhamento



# Widgets

O método `.insert()` é usado para inserir um novo item no Treeview. A forma padrão para a sua construção é:  
`tree.insert("", END, text=" ", values=dados)`  
onde:

- ("string vazia"): Indica que o novo item será um item raiz(Pai), ou seja, estará no nível superior da hierarquia do Treeview.
- **END**: Indica que o novo item será adicionado como o último filho do Treeview, independentemente do item pai

# Widgets

**text:** Recebe uma string que será apresentada na coluna #0.

**values:** Recebe uma tupla com os dados que serão inseridos em cada coluna adicional de forma posicional.

```
1 # Importando Tkinter e Ttk
2 from tkinter import *
3 from tkinter import ttk
4
5 # Criando Janela
6 janela = Tk()
7 janela.title("Exemplo de Treeview")
8
9 # Criando o Treeview
10 tree = ttk.Treeview(janela)
11
12 # Definindo as colunas
13 tree["columns"] = ("nome", "email", "telefone")
14
15 # Formatando as colunas
16 tree.column("#0", width=100, minwidth=100, anchor=CENTER)
17 tree.column("nome", width=150, minwidth=150, anchor=CENTER)
18 tree.column("email", width=150, minwidth=150, anchor=CENTER)
19 tree.column("telefone", width=150, minwidth=150, anchor=CENTER)
20 tree.column("telefone", width=150, minwidth=150, anchor=CENTER)
```

# Widgets

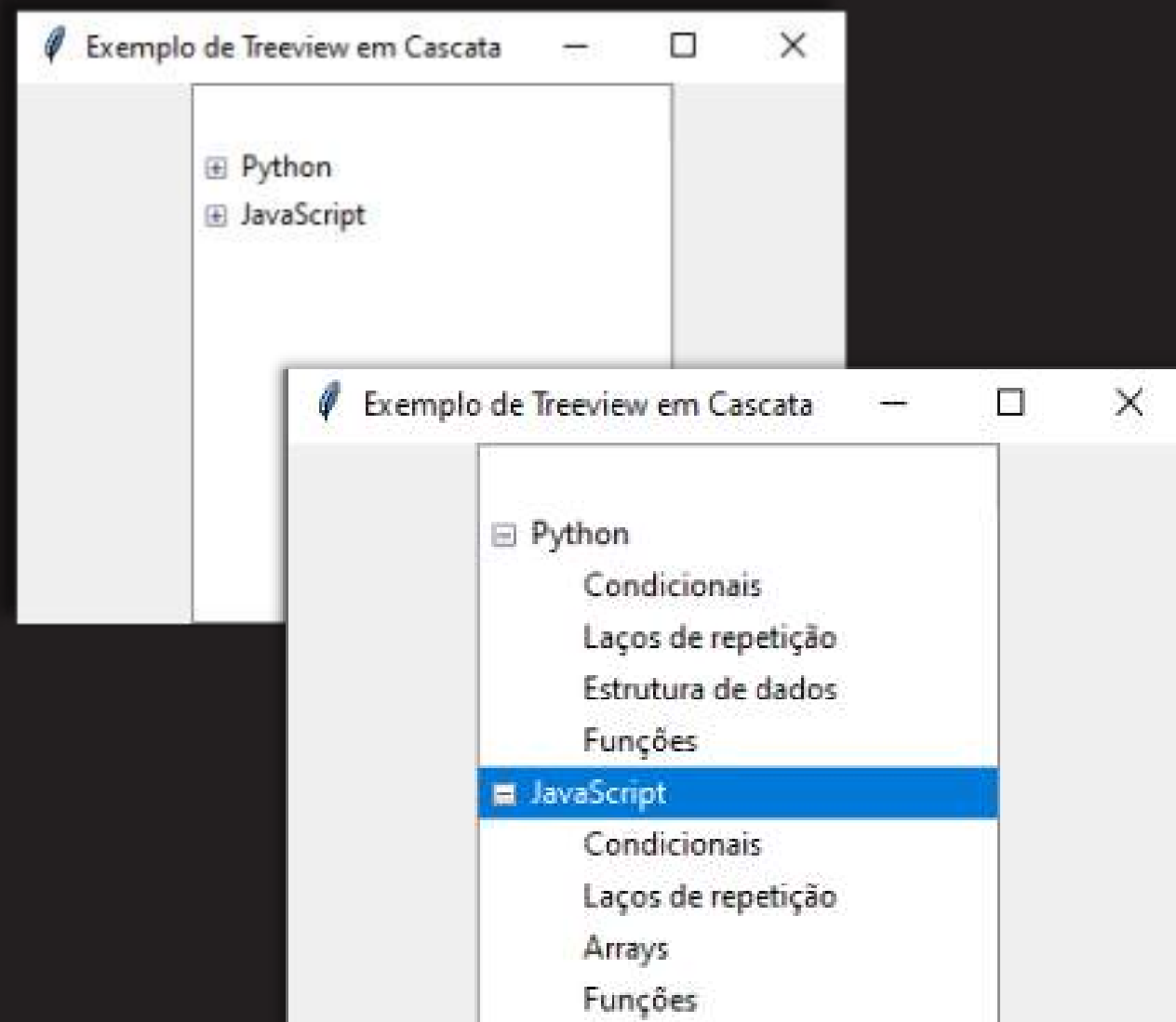
```
1 # Definindo os cabeçalhos das colunas
2 tree.heading("#0", text="ID")
3 tree.heading("nome", text="NOME")
4 tree.heading("email", text="EMAIL")
5 tree.heading("telefone", text="TEL")
6
7
8 # Adicionando itens ao Treeview
9 tree.insert("", END, text="1", values=("João Silva", "joao.silva@email.com", "(11) 1234-5678"))
10 tree.insert("", END, text="2", values=("Maria Santos", "maria.santos@email.com", "(22) 9876-5432"))
11 tree.insert("", END, text="3", values=("Pedro Almeida", "pedro.almeida@email.com", "(33) 4567-8901"))
12 tree.insert("", END, text="4", values=("Ana Costa", "ana.costa@email.com", "(44) 6789-0123"))
13 tree.insert("", END, text="5", values=("Lucas Oliveira", "lucas.oliveira@email.com", "(55) 2345-6789"))
14 tree.insert("", END, text="6", values=("Isabela Souza", "isabela.souza@email.com", "(66) 7890-1234"))
15 tree.insert("", END, text="7", values=("Rafael Mendes", "rafael.mendes@email.com", "(77) 3456-7890"))
16 tree.insert("", END, text="8", values=("Juliana Pereira", "juliana.pereira@email.com", "(88) 9012-3456"))
17 tree.insert("", END, text="9", values=("Carlos Santos", "carlos.santos@email.com", "(99) 1234-5678"))
18 tree.insert("", END, text="10", values=("Fernanda Lima", "fernanda.lima@email.com", "(00) 9876-5432"))
19
20 # Exibindo o Treeview
21 tree.pack()
22
23 janela.mainloop()
```

# Widgets

A **visualização em cascata (Treeview)** é um widget de visualização em árvore que exibe uma hierarquia de itens e permite que os usuários naveguem por ela. Diferentemente da visualização em árvore, a visualização em cascata não possui colunas. Os itens são alocados de forma hierárquica.

Utilizamos o iid para criar um identificador para o item raiz/pai, que durante a adição dos itens filhos, será identificado substituindo a string vazia que indicava um item raiz.

# Widgets



```
1 # Adicionando o item raiz/pai
2 tree.insert("", END, iid="py", text="Python")
3
4 # Adicionando o item filho
5 tree.insert("py", END, text="Condicionais")
6 tree.insert("py", END, text="Laços de repetição")
```



# Tipos de Dados

No tkinter, existem vários tipos de dados que podem ser usados para diferentes propósitos ao construir interfaces gráficas. Alguns dos principais tipos de dados no tkinter são:

**StringVar:** É uma variável especial do tipo string usada para armazenar valores de texto. Pode ser associada a widgets como Entry, Label, Button, entre outros

# Tipos de Dados

**IntVar, DoubleVar:** São variáveis especiais para armazenar valores numéricos inteiros e de ponto flutuante, respectivamente. Podem ser usadas com widgets como Spinbox, Scale, Checkbutton, entre outros.

**BooleanVar:** É uma variável especial para armazenar valores booleanos (True ou False). É comumente usada com o widget Checkbutton para capturar estados de seleção/deseleção.

**Listbox:** É um widget que pode exibir uma lista de itens. Os itens em uma Listbox são geralmente strings, mas também é possível inserir outros tipos de dados.



# Tipos de Dados



```
1 # StringVar
2 nome_var = StringVar()
3 nome_entry = ttk.Entry(janela, textvariable=nome_var)
4 nome_entry.pack()
```



```
1 # BooleanVar
2 aceito_var = BooleanVar()
3 aceito_checkbutton = ttk.Checkbutton(janela, text="Aceito os termos", variable=aceito_var)
4 aceito_checkbutton.pack()
```

# ATIVIDADE PRÁTICA 1

Crie uma aplicação que execute uma tarefa demorada e utilize uma barra de progresso (`ttk.Progressbar`) para mostrar o andamento da tarefa. Simule a conclusão da tarefa com um botão "Concluir Tarefa".

# ATIVIDADE PRÁTICA 2

Calculadora: Atualize a calculadora para que todas as operações sejam visualizadas na interface gráfica. Utilize o Entry ou Label para a visualização das operações e resultados.

# ATIVIDADE PRÁTICA 3

Cadastro de Alunos: Modifique o programa para que ao cadastrar um aluno, os dados sejam exibidos em uma árvore de exibição (Treeview).

# DESAFIO PRÁTICO

## Aplicativo de gerenciamento de tarefas

Crie um aplicativo de gerenciamento de tarefas que permita aos usuários adicionar, priorizar e categorizar tarefas. Use um `ttk.Combobox` para permitir a seleção de prioridades e categoria

# Material Complementar

- Exploração: Não tenha medo de explorar e testar diferentes códigos. A experimentação é uma grande aliada da aprendizagem.
- Perguntas: Faça perguntas, seja curioso! Entender o "porquê" das coisas ajuda a consolidar o conhecimento.
- Revisão: Revise o que aprendeu, tente explicar para si mesmo ou para outras pessoas. Ensinar é uma ótima forma de aprender.

Prática: A prática leva à perfeição. Quanto mais

- exercícios fizer, mais fácil será lembrar e entender os conceitos.





# SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 11 DE  
PYTHON. POO I

The logo consists of the letters 'IN' in a white, bold, sans-serif font, centered within a solid red square.

**INFINITY SCHOOL**  
VISUAL ART CREATIVE CENTER



# POO (Programação Orientada a Objetos)

Quem desenvolve softwares sabe o quanto o trabalho pode ser desafiador. Afinal de contas, programar é trazer soluções para os problemas dos clientes utilizando algoritmos em uma linguagem de programação. Portanto, a programação orientada a objetos trouxe agilidade no desenvolvimento de software, reduzindo o tempo para identificar e corrigir erros de programação. Isso revitalizou o paradigma da programação como um todo. E desse assunto que iremos tratar nesta aula.

```
function decorate(event) {  
  event = event || window.event;  
  var target = event.target || event.srcElement;  
  if (target && (target.getAttribute('action') || target  
    ga(function (tracker) {  
      var linkerParam = tracker.get('linkerParam');  
      document.cookie = '_shopify_ga=' + linkerParam + '  
    }));  
  listener(window, 'load', function(){  
    for (i=0; i < document.forms.length; i++) {  
      if (document.forms[i].getAttribute('action')  
        document.forms[i].submit, decorate);  
    }  
  })  
}
```



# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 10 – TKINTER II