# Notas de aula de programação
# Curso Livre de Python
## Aula 01 - Módulo Tkinter

Prof. Louis Augusto
louis.augusto@ifsc.edu.br


Instituto Federal de Santa Catarina
Campus São José

# Índice

# Sumário

# Tkinter GUI

Tkinter é um dos módulos python para construir uma interface gráfica. **GUI** significa *graphical user interface*, ou seja, interface gráfica de usuário.

Tkinter é de código aberto e é a GUI nativa do python (módulo `Tkinter.py`).

Vamos a um `Hello world` do tkinter:

```
import tkinter as tk
janela = tk.Tk()
strg = "Texto Qualquer"
strg2 = "Bem Simples"
strg=strg+"\n"+strg2
Label1 = tk.Label(text= strg)
Label1.pack()
janela.mainloop()
```




Do módulo tkinter importamos os métodos `Label` e `mainloop`, que são métodos que passam uma string para a tela e que mantém a janela aberta até que se aperte a janela de sair. Vamos estudar um pouco mais o que está acontecendo.

# Tkinter GUI

Tkinter é um dos módulos python para construir uma interface gráfica. **GUI** significa *graphical user interface*, ou seja, interface gráfica de usuário.

Tkinter é de código aberto e é a GUI nativa do python (módulo `Tkinter.py`).

Vamos a um `Hello world` do tkinter:

```
import tkinter as tk
janela = tk.Tk()
strg = "Texto Qualquer"
strg2 = "Bem Simples"
strg=strg+"\n"+strg2
Label1 = tk.Label(text= strg)
Label1.pack()
janela.mainloop()
```




Do módulo tkinter importamos os métodos `Label` e `mainloop`, que são métodos que passam uma string para a tela e que mantém a janela aberta até que se aperte a janela de sair. Vamos estudar um pouco mais o que está acontecendo.

# Tkinter GUI

Tkinter é um dos módulos python para construir uma interface gráfica. **GUI** significa *graphical user interface*, ou seja, interface gráfica de usuário.

Tkinter é de código aberto e é a GUI nativa do python (módulo `Tkinter.py`).

Vamos a um `Hello world` do tkinter:

```
import tkinter as tk
janela = tk.Tk()
strg = "Texto Qualquer"
strg2 = "Bem Simples"
strg=strg+"\n"+strg2
Label1 = tk.Label(text= strg)
Label1.pack()
janela.mainloop()
```




Do módulo tkinter importamos os métodos `Label` e `mainloop`, que são métodos que passam uma string para a tela e que mantém a janela aberta até que se aperte a janela de sair. Vamos estudar um pouco mais o que está acontecendo.

# Tkinter GUI

Tkinter é um dos módulos python para construir uma interface gráfica. **GUI** significa *graphical user interface*, ou seja, interface gráfica de usuário.

Tkinter é de código aberto e é a GUI nativa do python (módulo `Tkinter.py`).

Vamos a um `Hello world` do tkinter:

```
import tkinter as tk
janela = tk.Tk()
strg = "Texto Qualquer"
strg2 = "Bem Simples"
strg=strg+"\n"+strg2
Label1 = tk.Label(text= strg)
Label1.pack()
janela.mainloop()
```




Do módulo tkinter importamos os métodos `Label` e `mainloop`, que são métodos que passam uma string para a tela e que mantém a janela aberta até que se aperte a janela de sair. Vamos estudar um pouco mais o que está acontecendo.

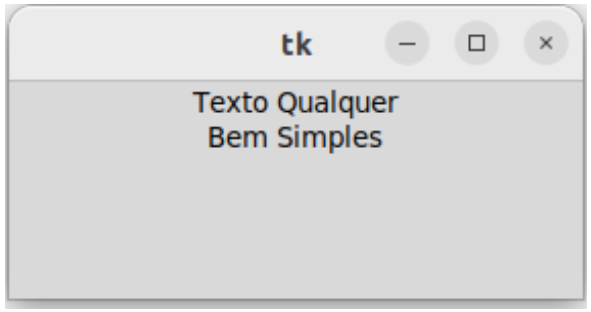
# Tkinter GUI

Tkinter é um dos módulos python para construir uma interface gráfica. **GUI** significa *graphical user interface*, ou seja, interface gráfica de usuário.

Tkinter é de código aberto e é a GUI nativa do python (módulo `Tkinter.py`).

Vamos a um `Hello world` do tkinter:

```
import tkinter as tk
janela = tk.Tk()
strg = "Texto Qualquer"
strg2 = "Bem Simples"
strg=strg+"\n"+strg2
Label1 = tk.Label(text= strg)
Label1.pack()
janela.mainloop()
```




Do módulo tkinter importamos os métodos `Label` e `mainloop`, que são métodos que passam uma string para a tela e que mantém a janela aberta até que se aperte a janela de sair. Vamos estudar um pouco mais o que está acontecendo.

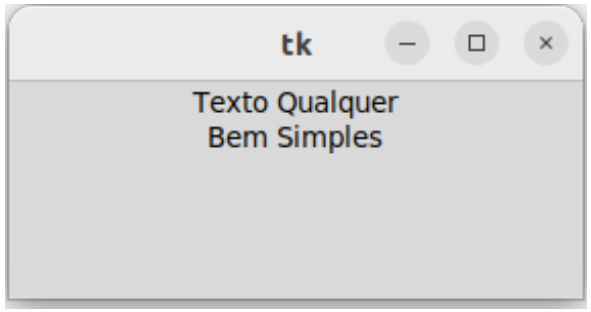
# Tkinter GUI

Tkinter é um dos módulos python para construir uma interface gráfica. **GUI** significa *graphical user interface*, ou seja, interface gráfica de usuário.

Tkinter é de código aberto e é a GUI nativa do python (módulo `Tkinter.py`).

Vamos a um `Hello world` do tkinter:

```
import tkinter as tk
janela = tk.Tk()
strg = "Texto Qualquer"
strg2 = "Bem Simples"
strg=strg+"\n"+strg2
Label1 = tk.Label(text= strg)
Label1.pack()
janela.mainloop()
```




Do módulo tkinter importamos os métodos `Label` e `mainloop`, que são métodos que passam uma string para a tela e que mantém a janela aberta até que se aperte a janela de sair. Vamos estudar um pouco mais o que está acontecendo.

# Tkinter GUI

Inicialmente você deve ter percebido que a tela do tkinter não apareceu no computador com o tamanho que está na imagem. Isto ocorreu porque não foi indicado um tamanho para a tela.

Podemos fazer isto com o comando:
`janela.geometry("100x100")` - em que `100x100` significa o tamanho em pixels do comprimento da base e da altura da janela.

A seguir foi gerado um texto para aparecer na tela. Chama-se widget a uma classe ou mesmo função no tkinter, que serve para abrigar alguma forma na tela, pode ser um rótulo, botão, menu etc. No nosso exemplo, `janela` é o widget raiz e `Label1` é um widget filho de `janela`.

Foi usado o widget Label para abrigar uma string. Todo widget precisa de um gerenciador de geometria para incluir na tela o widget, no caso foi usado o gerenciador `pack()`.

O script termina com o comando `janela.mainloop()` que é um laço infinito que se mantém ativo até que o botão de fechar o programa seja pressionado. [1]

[1] O botão de fechar fica no alto da janela, junto com o botão de maximizar e minimizar a tela do programa.

# Tkinter GUI

Inicialmente você deve ter percebido que a tela do tkinter não apareceu no computador com o tamanho que está na imagem. Isto ocorreu porque não foi indicado um tamanho para a tela.

Podemos fazer isto com o comando:
`janela.geometry("100x100")` - em que `100x100` significa o tamanho em pixels do comprimento da base e da altura da janela.

A seguir foi gerado um texto para aparecer na tela. Chama-se widget a uma classe ou mesmo função no tkinter, que serve para abrigar alguma forma na tela, pode ser um rótulo, botão, menu etc. No nosso exemplo, `janela` é o widget raiz e `Label1` é um widget filho de `janela`.

Foi usado o widget Label para abrigar uma string. Todo widget precisa de um gerenciador de geometria para incluir na tela o widget, no caso foi usado o gerenciador `pack()`.

O script termina com o comando `janela.mainloop()` que é um laço infinito que se mantém ativo até que o botão de fechar o programa seja pressionado. [1]

[1] O botão de fechar fica no alto da janela, junto com o botão de maximizar e minimizar a tela do programa.

# Tkinter GUI

Inicialmente você deve ter percebido que a tela do tkinter não apareceu no computador com o tamanho que está na imagem. Isto ocorreu porque não foi indicado um tamanho para a tela.

Podemos fazer isto com o comando:
`janela.geometry("100x100")` - em que `100x100` significa o tamanho em pixels do comprimento da base e da altura da janela.

A seguir foi gerado um texto para aparecer na tela. Chama-se widget a uma classe ou mesmo função no tkinter, que serve para abrigar alguma forma na tela, pode ser um rótulo, botão, menu etc. No nosso exemplo, `janela` é o widget raiz e `Label1` é um widget filho de `janela`.

Foi usado o widget Label para abrigar uma string. Todo widget precisa de um gerenciador de geometria para incluir na tela o widget, no caso foi usado o gerenciador `pack()`.

O script termina com o comando `janela.mainloop()` que é um laço infinito que se mantém ativo até que o botão de fechar o programa seja pressionado. [1]

[1] O botão de fechar fica no alto da janela, junto com o botão de maximizar e minimizar a tela do programa.

# Tkinter GUI

Inicialmente você deve ter percebido que a tela do tkinter não apareceu no computador com o tamanho que está na imagem. Isto ocorreu porque não foi indicado um tamanho para a tela.

Podemos fazer isto com o comando:
`janela.geometry("100x100")` - em que `100x100` significa o tamanho em pixels do comprimento da base e da altura da janela.

A seguir foi gerado um texto para aparecer na tela. Chama-se widget a uma classe ou mesmo função no tkinter, que serve para abrigar alguma forma na tela, pode ser um rótulo, botão, menu etc. No nosso exemplo, `janela` é o widget raiz e `Label1` é um widget filho de `janela`.

Foi usado o widget Label para abrigar uma string. Todo widget precisa de um gerenciador de geometria para incluir na tela o widget, no caso foi usado o gerenciador `pack()`.

O script termina com o comando `janela.mainloop()` que é um laço infinito que se mantém ativo até que o botão de fechar o programa seja pressionado. [1]

[1] O botão de fechar fica no alto da janela, junto com o botão de maximizar e minimizar a tela do programa.

# Tkinter GUI

Inicialmente você deve ter percebido que a tela do tkinter não apareceu no computador com o tamanho que está na imagem. Isto ocorreu porque não foi indicado um tamanho para a tela.

Podemos fazer isto com o comando:
`janela.geometry("100x100")` - em que `100x100` significa o tamanho em pixels do comprimento da base e da altura da janela.

A seguir foi gerado um texto para aparecer na tela. Chama-se widget a uma classe ou mesmo função no tkinter, que serve para abrigar alguma forma na tela, pode ser um rótulo, botão, menu etc. No nosso exemplo, `janela` é o widget raiz e `Label1` é um widget filho de `janela`.

Foi usado o widget Label para abrigar uma string. Todo widget precisa de um gerenciador de geometria para incluir na tela o widget, no caso foi usado o gerenciador `pack()`.

O script termina com o comando `janela.mainloop()` que é um laço infinito que se mantém ativo até que o botão de fechar o programa seja pressionado. [1]

[1] O botão de fechar fica no alto da janela, junto com o botão de maximizar e minimizar a tela do programa.

# Tkinter GUI com classe

Vamos repetir o programa anterior, mas utilizando a classe Tk():

```
import tkinter as tk
class JanelaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Tkinter com Classe")
        self.geometry("280x70")
        strg = "Texto Qualquer"
        strg2 = "Bem Simples"
        strg=strg+"\n"+strg2
        label = tk.Label(self, text=strg)
        label.pack()
if __name__ == "__main__":
    janela = JanelaPrincipal()
    janela.mainloop()
```

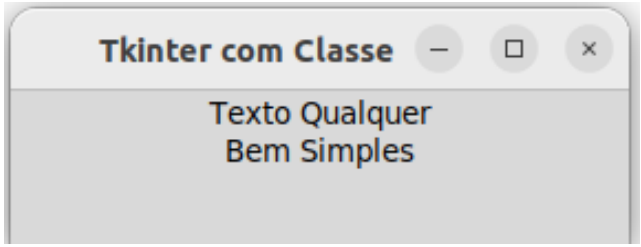



Temos o mesmo resultado do script, apenas adicionou-se um título ao programa e incluiu-se o tamanho da tela para ter um valor inicial visível para a tela de saída.

# Tkinter GUI com classe

Vamos repetir o programa anterior, mas utilizando a classe Tk():

```
import tkinter as tk
class JanelaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Tkinter com Classe")
        self.geometry("280x70")
        strg = "Texto Qualquer"
        strg2 = "Bem Simples"
        strg=strg+"\n"+strg2
        label = tk.Label(self, text=strg)
        label.pack()
if __name__ == "__main__":
    janela = JanelaPrincipal()
    janela.mainloop()
```

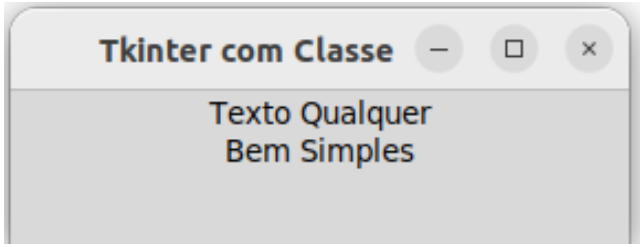



Temos o mesmo resultado do script, apenas adicionou-se um título ao programa e incluiu-se o tamanho da tela para ter um valor inicial visível para a tela de saída.

# Tkinter GUI com classe

Vamos repetir o programa anterior, mas utilizando a classe Tk():

```
import tkinter as tk
class JanelaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Tkinter com Classe")
        self.geometry("280x70")
        strg = "Texto Qualquer"
        strg2 = "Bem Simples"
        strg=strg+"\n"+strg2
        label = tk.Label(self, text=strg)
        label.pack()
if __name__ == "__main__":
    janela = JanelaPrincipal()
    janela.mainloop()
```




Temos o mesmo resultado do script, apenas adicionou-se um título ao programa e incluiu-se o tamanho da tela para ter um valor inicial visível para a tela de saída.

# Gerenciadores de geometria

Os gerenciadores de geometria sevem para distribuir os widgets na tela. Há três e cada um possui um objetivo e argumentos diferentes.

Temos três gerenciadores:

- pack() Busca utilizar o espaço livre dentro do widget pai para colocar o widget filho[2]. Permite que o widget filho varie de tamanho se o widget pai se alterar, deixando o mínimo de espaço vazio.
- grid() Transforma o widget pai numa malha contendo linhas e colunas, como uma matriz em Matemática.
- place() Os gerenciadores `pack()` e `grid()` calculam automaticamente onde cada widget é posicionado, já `place()` especifica a localização do widget filho no widget pai, sem alterar a posição quando o tamanho do widget pai sofrer variação.

Vamos estudar cada um dos gerenciadores.

[2] Lembre-se que no exemplo do código Hello World feito, o widget pai era a própria janela, e o widget filho era referenciado por Label1.

# Gerenciadores de geometria

Os gerenciadores de geometria sevem para distribuir os widgets na tela. Há três e cada um possui um objetivo e argumentos diferentes.

Temos três gerenciadores:

- pack() Busca utilizar o espaço livre dentro do widget pai para colocar o widget filho[2]. Permite que o widget filho varie de tamanho se o widget pai se alterar, deixando o mínimo de espaço vazio.
- grid() Transforma o widget pai numa malha contendo linhas e colunas, como uma matriz em Matemática.
- place() Os gerenciadores `pack()` e `grid()` calculam automaticamente onde cada widget é posicionado, já `place()` especifica a localização do widget filho no widget pai, sem alterar a posição quando o tamanho do widget pai sofrer variação.

Vamos estudar cada um dos gerenciadores.

[2]Lembre-se que no exemplo do código Hello World feito, o widget pai era a própria janela, e o widget filho era referenciado por Label1.

# Gerenciadores de geometria

Os gerenciadores de geometria sevem para distribuir os widgets na tela. Há três e cada um possui um objetivo e argumentos diferentes.

Temos três gerenciadores:

- pack() Busca utilizar o espaço livre dentro do widget pai para colocar o widget filho[2]. Permite que o widget filho varie de tamanho se o widget pai se alterar, deixando o mínimo de espaço vazio.
- grid() Transforma o widget pai numa malha contendo linhas e colunas, como uma matriz em Matemática.
- place() Os gerenciadores `pack()` e `grid()` calculam automaticamente onde cada widget é posicionado, já `place()` especifica a localização do widget filho no widget pai, sem alterar a posição quando o tamanho do widget pai sofrer variação.

Vamos estudar cada um dos gerenciadores.

[2]Lembre-se que no exemplo do código Hello World feito, o widget pai era a própria janela, e o widget filho era referenciado por Label1.

# Gerenciadores de geometria

Os gerenciadores de geometria sevem para distribuir os widgets na tela. Há três e cada um possui um objetivo e argumentos diferentes.

Temos três gerenciadores:

- pack() Busca utilizar o espaço livre dentro do widget pai para colocar o widget filho[2]. Permite que o widget filho varie de tamanho se o widget pai se alterar, deixando o mínimo de espaço vazio.
- grid() Transforma o widget pai numa malha contendo linhas e colunas, como uma matriz em Matemática.
- place() Os gerenciadores `pack()` e `grid()` calculam automaticamente onde cada widget é posicionado, já `place()` especifica a localização do widget filho no widget pai, sem alterar a posição quando o tamanho do widget pai sofrer variação.

Vamos estudar cada um dos gerenciadores.

[2]Lembre-se que no exemplo do código Hello World feito, o widget pai era a própria janela, e o widget filho era referenciado por Label1.

# Gerenciadores de geometria

Os gerenciadores de geometria sevem para distribuir os widgets na tela. Há três e cada um possui um objetivo e argumentos diferentes.

Temos três gerenciadores:

- pack() Busca utilizar o espaço livre dentro do widget pai para colocar o widget filho[2]. Permite que o widget filho varie de tamanho se o widget pai se alterar, deixando o mínimo de espaço vazio.
- grid() Transforma o widget pai numa malha contendo linhas e colunas, como uma matriz em Matemática.
- place() Os gerenciadores `pack()` e `grid()` calculam automaticamente onde cada widget é posicionado, já `place()` especifica a localização do widget filho no widget pai, sem alterar a posição quando o tamanho do widget pai sofrer variação.

Vamos estudar cada um dos gerenciadores.

[2]Lembre-se que no exemplo do código Hello World feito, o widget pai era a própria janela, e o widget filho era referenciado por Label1.

# Gerenciadores de geometria

Os gerenciadores de geometria sevem para distribuir os widgets na tela. Há três e cada um possui um objetivo e argumentos diferentes.

Temos três gerenciadores:

- pack() Busca utilizar o espaço livre dentro do widget pai para colocar o widget filho[2]. Permite que o widget filho varie de tamanho se o widget pai se alterar, deixando o mínimo de espaço vazio.
- grid() Transforma o widget pai numa malha contendo linhas e colunas, como uma matriz em Matemática.
- place() Os gerenciadores `pack()` e `grid()` calculam automaticamente onde cada widget é posicionado, já `place()` especifica a localização do widget filho no widget pai, sem alterar a posição quando o tamanho do widget pai sofrer variação.

Vamos estudar cada um dos gerenciadores.

[2]Lembre-se que no exemplo do código Hello World feito, o widget pai era a própria janela, e o widget filho era referenciado por Label1.

# Sumário

# Gerenciador pack()

Conforme dito anteriormente pack() busca usar o espaço livre dentro do widget pai, conforme o widget pai aumenta ou diminui.

O gerenciador de geometria pack() é controlado principalmente pelos seguintes arguementos com palavra-chave:

- side: posiciona o widget filho em um dos cantos, usar as variáveis tkinter LEFT, RIGHT, TOP, BOTTOM. Se o módulo tkinter for chamado como `import tkinter as tk` vamos inserir a informação como `side = tk.TOP`, por exemplo.
- fill: Faz com que o widget filho ocupe o espaço disponível. As opções são as constantes `X`, `Y` e `BOTH`, sendo `X` significando horizontal, `Y` vertical, e `BOTH` ambos, horizontal e vertical.
- expand: Adapta o tamanho do widget filho ao widget pai. É um argumento booleando, de forma que os valores são True ou False.
- padx e pady: gera espaço extra a partir do mínimo para dar melhor visualização ao interior do widget.
- anchor: serve para ancorar o widget em posições previstas.

# Gerenciador pack()

Conforme dito anteriormente pack() busca usar o espaço livre dentro do widget pai, conforme o widget pai aumenta ou diminui.

O gerenciador de geometria pack() é controlado principalmente pelos seguintes arguementos com palavra-chave:

- side: posiciona o widget filho em um dos cantos, usar as variáveis tkinter LEFT, RIGHT, TOP, BOTTOM. Se o módulo tkinter for chamado como `import tkinter as tk` vamos inserir a informação como `side = tk.TOP`, por exemplo.
- fill: Faz com que o widget filho ocupe o espaço disponível. As opções são as constantes `X`, `Y` e `BOTH`, sendo `X` significando horizontal, `Y` vertical, e `BOTH` ambos, horizontal e vertical.
- expand: Adapta o tamanho do widget filho ao widget pai. É um argumento booleando, de forma que os valores são True ou False.
- padx e pady: gera espaço extra a partir do mínimo para dar melhor visualização ao interior do widget.
- anchor: serve para ancorar o widget em posições previstas.

# Gerenciador pack()

Conforme dito anteriormente pack() busca usar o espaço livre dentro do widget pai, conforme o widget pai aumenta ou diminui.

O gerenciador de geometria pack() é controlado principalmente pelos seguintes arguementos com palavra-chave:

- side: posiciona o widget filho em um dos cantos, usar as variáveis tkinter LEFT, RIGHT, TOP, BOTTOM. Se o módulo tkinter for chamado como `import tkinter as tk` vamos inserir a informação como `side = tk.TOP`, por exemplo.
- fill: Faz com que o widget filho ocupe o espaço disponível. As opções são as constantes `X`, `Y` e `BOTH`, sendo `X` significando horizontal, `Y` vertical, e `BOTH` ambos, horizontal e vertical.
- expand: Adapta o tamanho do widget filho ao widget pai. É um argumento booleando, de forma que os valores são True ou False.
- padx e pady: gera espaço extra a partir do mínimo para dar melhor visualização ao interior do widget.
- anchor: serve para ancorar o widget em posições previstas.

# Gerenciador pack()

Conforme dito anteriormente pack() busca usar o espaço livre dentro do widget pai, conforme o widget pai aumenta ou diminui.

O gerenciador de geometria pack() é controlado principalmente pelos seguintes arguementos com palavra-chave:

- side: posiciona o widget filho em um dos cantos, usar as variáveis tkinter LEFT, RIGHT, TOP, BOTTOM. Se o módulo tkinter for chamado como `import tkinter as tk` vamos inserir a informação como `side = tk.TOP`, por exemplo.
- fill: Faz com que o widget filho ocupe o espaço disponível. As opções são as constantes `X`, `Y` e `BOTH`, sendo `X` significando horizontal, `Y` vertical, e `BOTH` ambos, horizontal e vertical.
- expand: Adapta o tamanho do widget filho ao widget pai. É um argumento booleando, de forma que os valores são True ou False.
- padx e pady: gera espaço extra a partir do mínimo para dar melhor visualização ao interior do widget.
- anchor: serve para ancorar o widget em posições previstas.

# Gerenciador pack()

Conforme dito anteriormente pack() busca usar o espaço livre dentro do widget pai, conforme o widget pai aumenta ou diminui.

O gerenciador de geometria pack() é controlado principalmente pelos seguintes arguementos com palavra-chave:

- side: posiciona o widget filho em um dos cantos, usar as variáveis tkinter LEFT, RIGHT, TOP, BOTTOM. Se o módulo tkinter for chamado como `import tkinter as tk` vamos inserir a informação como `side = tk.TOP`, por exemplo.
- fill: Faz com que o widget filho ocupe o espaço disponível. As opções são as constantes `X`, `Y` e `BOTH`, sendo `X` significando horizontal, `Y` vertical, e `BOTH` ambos, horizontal e vertical.
- expand: Adapta o tamanho do widget filho ao widget pai. É um argumento booleando, de forma que os valores são True ou False.
- padx e pady: gera espaço extra a partir do mínimo para dar melhor visualização ao interior do widget.
- anchor: serve para ancorar o widget em posições previstas.

# Gerenciador pack()

Conforme dito anteriormente pack() busca usar o espaço livre dentro do widget pai, conforme o widget pai aumenta ou diminui.

O gerenciador de geometria pack() é controlado principalmente pelos seguintes arguementos com palavra-chave:

- side: posiciona o widget filho em um dos cantos, usar as variáveis tkinter LEFT, RIGHT, TOP, BOTTOM. Se o módulo tkinter for chamado como `import tkinter as tk` vamos inserir a informação como `side = tk.TOP`, por exemplo.
- fill: Faz com que o widget filho ocupe o espaço disponível. As opções são as constantes `X`, `Y` e `BOTH`, sendo `X` significando horizontal, `Y` vertical, e `BOTH` ambos, horizontal e vertical.
- expand: Adapta o tamanho do widget filho ao widget pai. É um argumento booleando, de forma que os valores são True ou False.
- padx e pady: gera espaço extra a partir do mínimo para dar melhor visualização ao interior do widget.
- anchor: serve para ancorar o widget em posições previstas.

# Gerenciador pack()

Considere o código abaixo:

```
import tkinter as tk
class JanelaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Título")
        label = tk.Label(self, bd = 5,
            text="Texto qualquer",
            relief = 'groove')
        label.pack(fill=tk.BOTH,
            expand=True, padx=100, pady=50)
if __name__ == "__main__":
    janela = JanelaPrincipal()
    janela.mainloop()
```

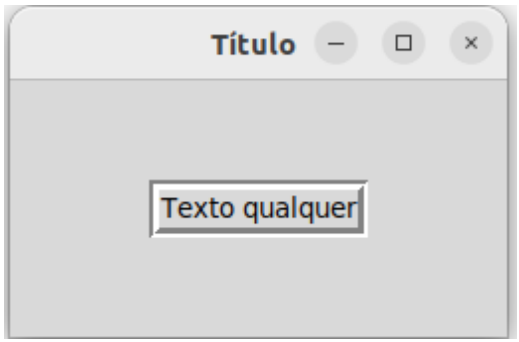



Observe que mesmo sem ter usado a linha `self.geometry("280x70")` o tkinter utilizou os valores inseridos em pack() para criar a janela com tamanho adequado.

# Gerenciador pack()

Considere o código abaixo:

```
import tkinter as tk
class JanelaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Título")
        label = tk.Label(self, bd = 5,
            text="Texto qualquer",
            relief = 'groove')
        label.pack(fill=tk.BOTH,
            expand=True, padx=100, pady=50)
if __name__ == "__main__":
    janela = JanelaPrincipal()
    janela.mainloop()
```

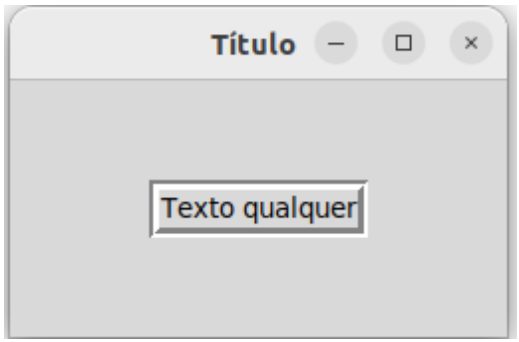



Observe que mesmo sem ter usado a linha `self.geometry("280x70")` o tkinter utilizou os valores inseridos em pack() para criar a janela com tamanho adequado.

# Gerenciador pack()

Considere o código abaixo:

```
import tkinter as tk
class JanelaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Título")
        label = tk.Label(self, bd = 5,
            text="Texto qualquer",
            relief = 'groove')
        label.pack(fill=tk.BOTH,
            expand=True, padx=100, pady=50)
if __name__ == "__main__":
    janela = JanelaPrincipal()
    janela.mainloop()
```




Observe que mesmo sem ter usado a linha `self.geometry("280x70")` o tkinter utilizou os valores inseridos em pack() para criar a janela com tamanho adequado.

# Argumento anchor

O argumento anchor serve para fixar o widget mesmo quando ocorrer mudança do tamanho da janela. Como entradas utiliza os pontos cardeais e colaterais da rosa dos ventos.

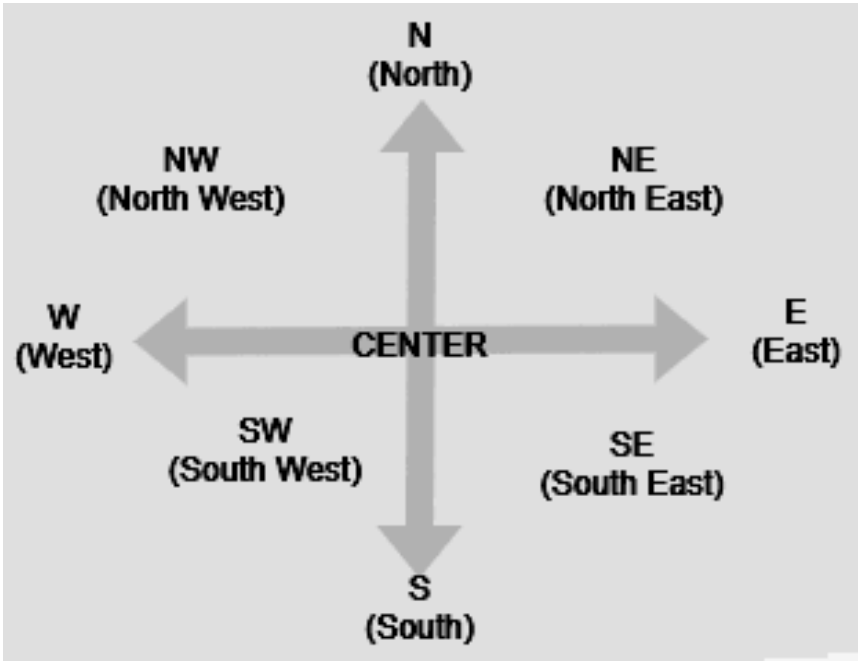



As linhas abaixo introduzem dois labels ancorados à direita e à esquerda:

```
Caixa1 = tk.Label(janela, text="Caixa 1", bg="green", fg="white")
Caixa1.pack(ipadx=20, ipady=20, anchor=tk.E, expand=True)
Caixa2 = tk.Label(janela, text="Caixa 2", bg="red", fg="white")
Caixa2.pack(ipadx=20, ipady=20, anchor=tk.W, expand=True)
```

# Argumento anchor

O argumento anchor serve para fixar o widget mesmo quando ocorrer mudança do tamanho da janela. Como entradas utiliza os pontos cardeais e colaterais da rosa dos ventos.




As linhas abaixo introduzem dois labels ancorados à direita e à esquerda:

```
Caixa1 = tk.Label(janela, text="Caixa 1", bg="green", fg="white")
Caixa1.pack(ipadx=20, ipady=20, anchor=tk.E, expand=True)
Caixa2 = tk.Label(janela, text="Caixa 2", bg="red", fg="white")
Caixa2.pack(ipadx=20, ipady=20, anchor=tk.W, expand=True)
```

# Gerenciador pack()

Para enxergar melhor como funciona cada argumento com palavra chave do pack(), considere no código descrito no slide 10 a linha

```
label.pack(fill=tk.BOTH, expand=True, padx=100, pady=50)
```

Alterando esta linha para os valores abaixo podem-se perceber as alterações.

```
label.pack(fill=tk.Y)
label.pack(fill=tk.Y, pady = 50)
label.pack(fill = tk.BOTH, expand = False)
label.pack(fill=tk.Y, padx = 70, pady = 50, expand = True)
```

# Gerenciador pack()

Para enxergar melhor como funciona cada argumento com palavra chave do pack(), considere no código descrito no slide 10 a linha

```
label.pack(fill=tk.BOTH, expand=True, padx=100, pady=50)
```

Alterando esta linha para os valores abaixo podem-se perceber as alterações.

```
label.pack(fill=tk.Y)
label.pack(fill=tk.Y, pady = 50)
label.pack(fill = tk.BOTH, expand = False)
label.pack(fill=tk.Y, padx = 70, pady = 50, expand = True)
```

# Sumário

# Gerenciador grid()

O gerenciador grid trata o widget pai como uma malha, contendo linhas e colunas, parecido com uma planilha.

O widget grid tem como mínimos argumentos `grid(row=0,column=0)`, e pode ter outras linhas e colunas, ambas iniciando-se por zero.

Para mesclar células usamos a palavra-chave `columnspan=2` (para mesclar 2 colunas) ou `rowspan=3`(para mesclar 3 linhas). A malha não tem linhas visíveis no widget pai.

Um comando completo seria:
`grid(row=1, column=0, columnspan=2)`, gerando uma célula que ocupa 2 posições na primeira linha.

Por padrão grid posiciona o widget centralizado na célula, mas existe o comando `sticky` que utiliza a rosa dos ventos, como o comando `archor` do gerenciador `pack()`.

Os comandos `pack()` e `grid()` não podem ser usados no mesmo widget pai. Em Frames diferentes podem ser utilizados.

# Gerenciador grid()

O gerenciador grid trata o widget pai como uma malha, contendo linhas e colunas, parecido com uma planilha.

O widget grid tem como mínimos argumentos `grid(row=0,column=0)`, e pode ter outras linhas e colunas, ambas iniciando-se por zero.

Para mesclar células usamos a palavra-chave `columnspan=2` (para mesclar 2 colunas) ou `rowspan=3`(para mesclar 3 linhas). A malha não tem linhas visíveis no widget pai.

Um comando completo seria:
`grid(row=1, column=0, columnspan=2)`, gerando uma célula que ocupa 2 posições na primeira linha.

Por padrão grid posiciona o widget centralizado na célula, mas existe o comando `sticky` que utiliza a rosa dos ventos, como o comando `archor` do gerenciador `pack()`.

Os comandos `pack()` e `grid()` não podem ser usados no mesmo widget pai. Em Frames diferentes podem ser utilizados.

# Gerenciador grid()

O gerenciador grid trata o widget pai como uma malha, contendo linhas e colunas, parecido com uma planilha.

O widget grid tem como mínimos argumentos `grid(row=0,column=0)`, e pode ter outras linhas e colunas, ambas iniciando-se por zero.

Para mesclar células usamos a palavra-chave `columnspan=2` (para mesclar 2 colunas) ou `rowspan=3`(para mesclar 3 linhas). A malha não tem linhas visíveis no widget pai.

Um comando completo seria:
`grid(row=1, column=0, columnspan=2)`, gerando uma célula que ocupa 2 posições na primeira linha.

Por padrão grid posiciona o widget centralizado na célula, mas existe o comando `sticky` que utiliza a rosa dos ventos, como o comando `archor` do gerenciador `pack()`.

Os comandos `pack()` e `grid()` não podem ser usados no mesmo widget pai. Em Frames diferentes podem ser utilizados.

# Gerenciador grid()

O gerenciador grid trata o widget pai como uma malha, contendo linhas e colunas, parecido com uma planilha.

O widget grid tem como mínimos argumentos `grid(row=0,column=0)`, e pode ter outras linhas e colunas, ambas iniciando-se por zero.

Para mesclar células usamos a palavra-chave `columnspan=2` (para mesclar 2 colunas) ou `rowspan=3`(para mesclar 3 linhas). A malha não tem linhas visíveis no widget pai.

Um comando completo seria:
`grid(row=1, column=0, columnspan=2)`, gerando uma célula que ocupa 2 posições na primeira linha.

Por padrão grid posiciona o widget centralizado na célula, mas existe o comando `sticky` que utiliza a rosa dos ventos, como o comando `archor` do gerenciador `pack()`.

Os comandos `pack()` e `grid()` não podem ser usados no mesmo widget pai. Em Frames diferentes podem ser utilizados.

# Gerenciador grid()

O gerenciador grid trata o widget pai como uma malha, contendo linhas e colunas, parecido com uma planilha.

O widget grid tem como mínimos argumentos `grid(row=0,column=0)`, e pode ter outras linhas e colunas, ambas iniciando-se por zero.

Para mesclar células usamos a palavra-chave `columnspan=2` (para mesclar 2 colunas) ou `rowspan=3`(para mesclar 3 linhas). A malha não tem linhas visíveis no widget pai.

Um comando completo seria:
`grid(row=1, column=0, columnspan=2)`, gerando uma célula que ocupa 2 posições na primeira linha.

Por padrão grid posiciona o widget centralizado na célula, mas existe o comando `sticky` que utiliza a rosa dos ventos, como o comando `archor` do gerenciador `pack()`.

Os comandos `pack()` e `grid()` não podem ser usados no mesmo widget pai. Em Frames diferentes podem ser utilizados.

# Gerenciador grid()

O gerenciador grid trata o widget pai como uma malha, contendo linhas e colunas, parecido com uma planilha.

O widget grid tem como mínimos argumentos `grid(row=0,column=0)`, e pode ter outras linhas e colunas, ambas iniciando-se por zero.

Para mesclar células usamos a palavra-chave `columnspan=2` (para mesclar 2 colunas) ou `rowspan=3`(para mesclar 3 linhas). A malha não tem linhas visíveis no widget pai.

Um comando completo seria:
`grid(row=1, column=0, columnspan=2)`, gerando uma célula que ocupa 2 posições na primeira linha.

Por padrão grid posiciona o widget centralizado na célula, mas existe o comando `sticky` que utiliza a rosa dos ventos, como o comando `archor` do gerenciador `pack()`.

Os comandos `pack()` e `grid()` não podem ser usados no mesmo widget pai. Em Frames diferentes podem ser utilizados.

# Exemplo com Frames

`Frame` é um widget container, que pode ser usado para arrumar posições de outros widgets. Um `Frame` é uma região retangular na tela que recebe outros widgets.

```
import tkinter as tk
JanelaPrincipal = tk.Tk()
JanelaPrincipal.geometry("450x450")
Frame1 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Red")
Frame1.grid(row=0, column=0)
Frame2 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Blue")
Frame2.grid(row=0, column=1)
Frame3 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Green")
Frame3.grid(row=1, column=0)
Frame4 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Brown")
Frame4.grid(row=1, column=1)
Frame5 = tk.Frame(JanelaPrincipal, width = 300, height = 150, bg = "Cyan")
Frame5.grid(row=2, column=0, columnspan=2)
Frame6 = tk.Frame(JanelaPrincipal, width = 150, height = 450, bg = "Yellow")
Frame6.grid(row = 0, rowspan= 3, column=2)
JanelaPrincipal.mainloop()
```

Gera a figura que pode ser vista no próximo slide.

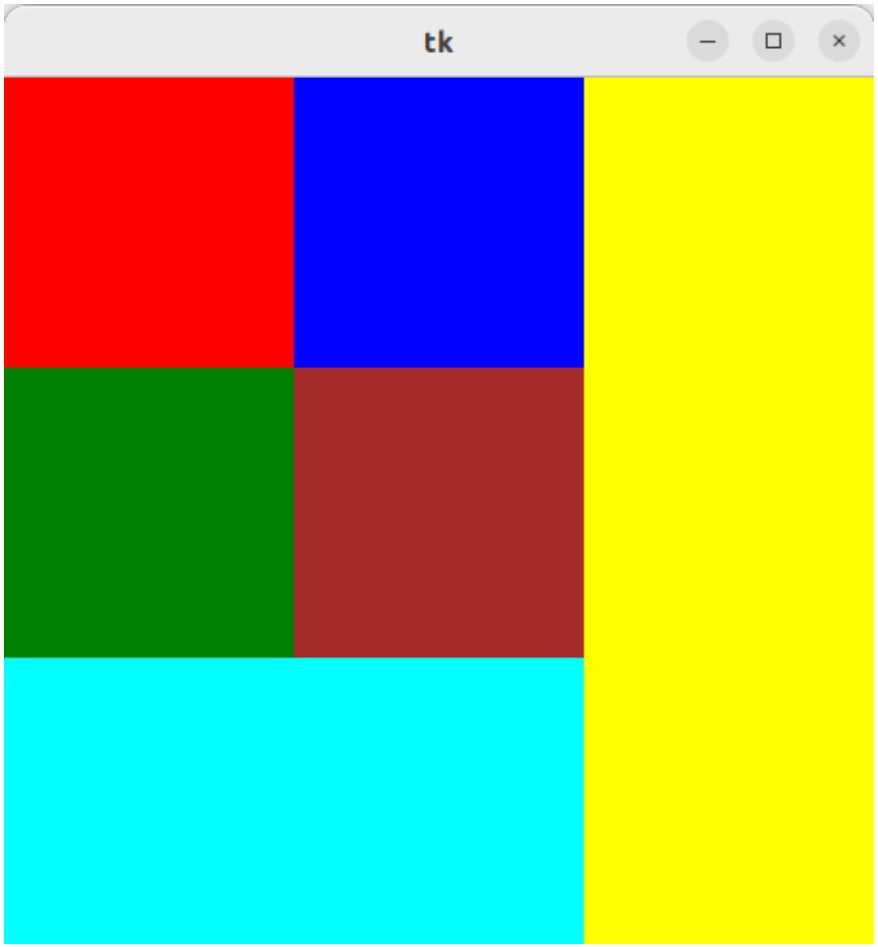
# Exemplo com Frames

`Frame` é um widget container, que pode ser usado para arrumar posições de outros widgets. Um `Frame` é uma região retangular na tela que recebe outros widgets.

```
import tkinter as tk
JanelaPrincipal = tk.Tk()
JanelaPrincipal.geometry("450x450")
Frame1 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Red")
Frame1.grid(row=0, column=0)
Frame2 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Blue")
Frame2.grid(row=0, column=1)
Frame3 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Green")
Frame3.grid(row=1, column=0)
Frame4 = tk.Frame(JanelaPrincipal, width = 150, height = 150, bg = "Brown")
Frame4.grid(row=1, column=1)
Frame5 = tk.Frame(JanelaPrincipal, width = 300, height = 150, bg = "Cyan")
Frame5.grid(row=2, column=0, columnspan=2)
Frame6 = tk.Frame(JanelaPrincipal, width = 150, height = 450, bg = "Yellow")
Frame6.grid(row = 0, rowspan= 3, column=2)
JanelaPrincipal.mainloop()
```

Gera a figura que pode ser vista no próximo slide.

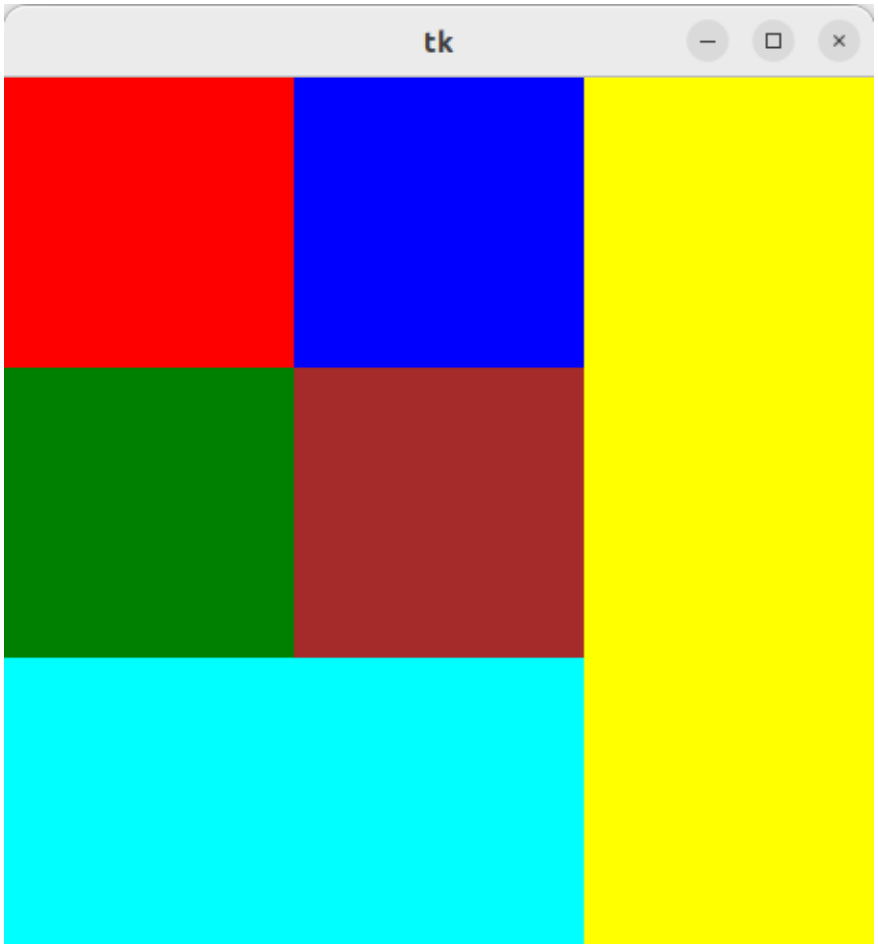
# Exemplo com Frames




Observe a divisão da Janela Principal em 6 partes, com 4 quadrados e dois retângulos que ocupam a área de dois quadrados.

Em cada um dos quadrados (Frames) pode-se colocar botões, labels, entries, ou outros widgets.

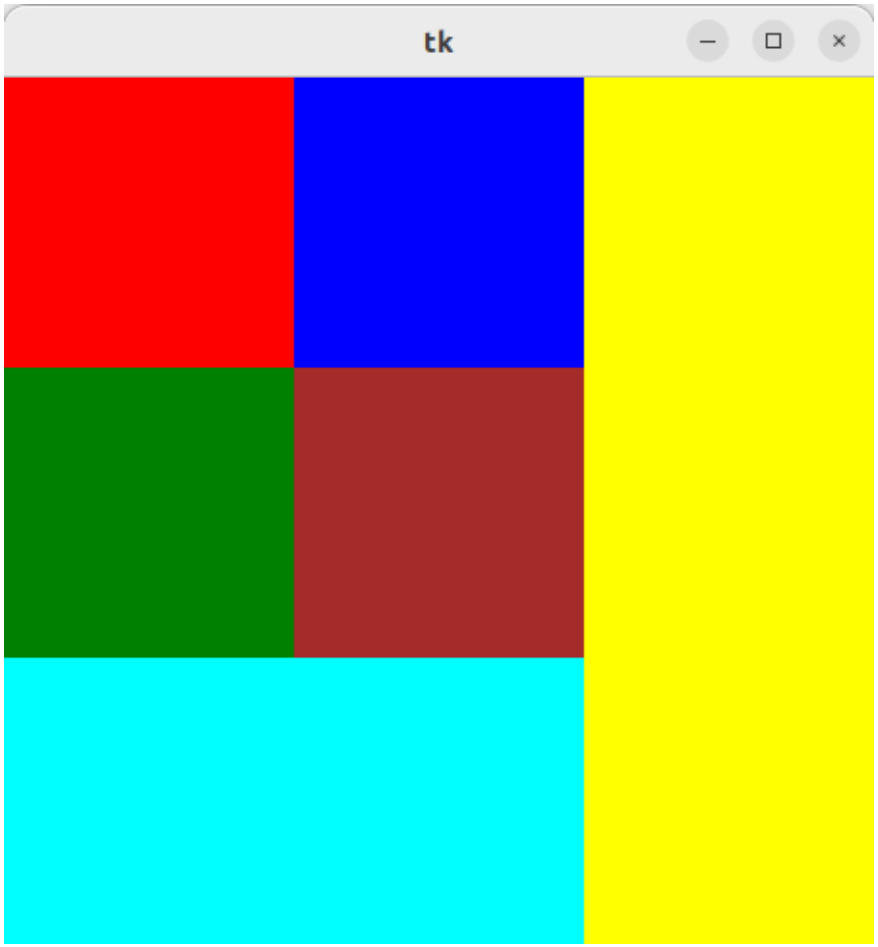
# Exemplo com Frames




Observe a divisão da Janela Principal em 6 partes, com 4 quadrados e dois retângulos que ocupam a área de dois quadrados.

Em cada um dos quadrados (Frames) pode-se colocar botões, labels, entries, ou outros widgets.

# Exemplo com Frames




Observe a divisão da Janela Principal em 6 partes, com 4 quadrados e dois retângulos que ocupam a área de dois quadrados.

Em cada um dos quadrados (Frames) pode-se colocar botões, labels, entries, ou outros widgets.

# Sumário

# Gerenciador place()

O gerenciador `place()` diferentemente de `pack()` e `grid()` não calcula onde o widget será localizado. Em `place()` é necessário localizar o widget, indicando as coordenadas em *x* e *y* em pixels.

O comando `NomeWidget.place(x=5, y=10)` coloca o widget na posição (5,10).

Para colocar no centro: `NomeWidget.place(relx=0.5,rely=0.5)` Para colocar 50% do comprimento e 25% da altura:
`NomeWidget.place(relwidth=0.5,relheight=0.25)`

A vantagem do uso de `place()` é que pode ser usado em conjunto com pack ou grid no mesmo widget pai.

# Gerenciador place()

O gerenciador `place()` diferentemente de `pack()` e `grid()` não calcula onde o widget será localizado. Em `place()` é necessário localizar o widget, indicando as coordenadas em *x* e *y* em pixels.

O comando `NomeWidget.place(x=5, y=10)` coloca o widget na posição (5,10).

Para colocar no centro: `NomeWidget.place(relx=0.5,rely=0.5)` Para colocar 50% do comprimento e 25% da altura: `NomeWidget.place(relwidth=0.5,relheight=0.25)`

A vantagem do uso de `place()` é que pode ser usado em conjunto com pack ou grid no mesmo widget pai.

# Gerenciador place()

O gerenciador `place()` diferentemente de `pack()` e `grid()` não calcula onde o widget será localizado. Em `place()` é necessário localizar o widget, indicando as coordenadas em *x* e *y* em pixels.

O comando `NomeWidget.place(x=5, y=10)` coloca o widget na posição (5,10).

Para colocar no centro: `NomeWidget.place(relx=0.5,rely=0.5)` Para colocar 50% do comprimento e 25% da altura:
`NomeWidget.place(relwidth=0.5,relheight=0.25)`

A vantagem do uso de `place()` é que pode ser usado em conjunto com pack ou grid no mesmo widget pai.

# Gerenciador place()

O gerenciador `place()` diferentemente de `pack()` e `grid()` não calcula onde o widget será localizado. Em `place()` é necessário localizar o widget, indicando as coordenadas em *x* e *y* em pixels.

O comando `NomeWidget.place(x=5, y=10)` coloca o widget na posição (5,10).

Para colocar no centro: `NomeWidget.place(relx=0.5,rely=0.5)` Para colocar 50% do comprimento e 25% da altura:
`NomeWidget.place(relwidth=0.5,relheight=0.25)`

A vantagem do uso de `place()` é que pode ser usado em conjunto com pack ou grid no mesmo widget pai.

# Sumário

# Entrada de dados pelo teclado