ICC: Programmation

GC/MX, Cours 9, 15 novembre 2019

Jean-Philippe Pellet

Previously, on Programmation…

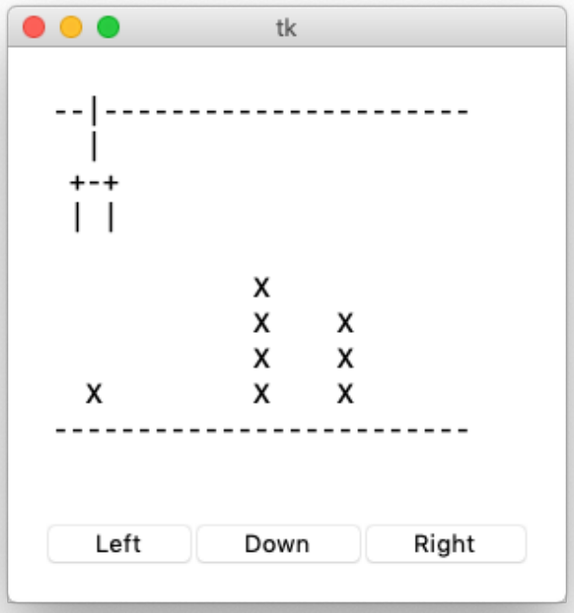
- Types de base en Python: int, float, str, bool
- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Conditions pour exécuter du code selon la valeur d'une expression booléenne: `if <condition>: ... else: …` et ses variantes
- Boucles pour exécuter du code plusieurs fois:
- Déclaration de fonctions avec type de retour et paramètres
- Utilisation de listes
- Utilisation de sets
- Utilisation de dictionnaires
- Utilisation de tuples, par exemple pour retourner plusieurs valeurs
- Déclaration de classes simples
- *Midterm!*

Évaluation du cours

- Donnez une note au cours sur IS-Academia
  - Dites ce qui vous a plu, ce qui vous a déplu; ce qui vous convient, ce qui ne vous convient pas
- Feedback anonyme
  - Mais soyez constructifs: tout le monde en profitera
- Discussion des résultats dès que je les obtiens
- Vos commentaires — indiquez svp. pour quelle partie vous faites un commentaire:
  - **théorie** → partie d'Olivier Lévêque
  - **python** → partie programmation

Miniprojet

- Comme les exercices: pas noté
- Chaque semaine, de nouveaux exercices qui se basent sur les résultats de la semaine précédente
  - Mais corrigé disponible chaque semaine si vous vous perdez
- But: développement d'une petite application simulant une grue programmable, à la Cargo-Bot
- Cette semaine: création d'une interface graphique de base



Démo CargoBot

Cargo 101
Transporter
Go Left
Color Sort

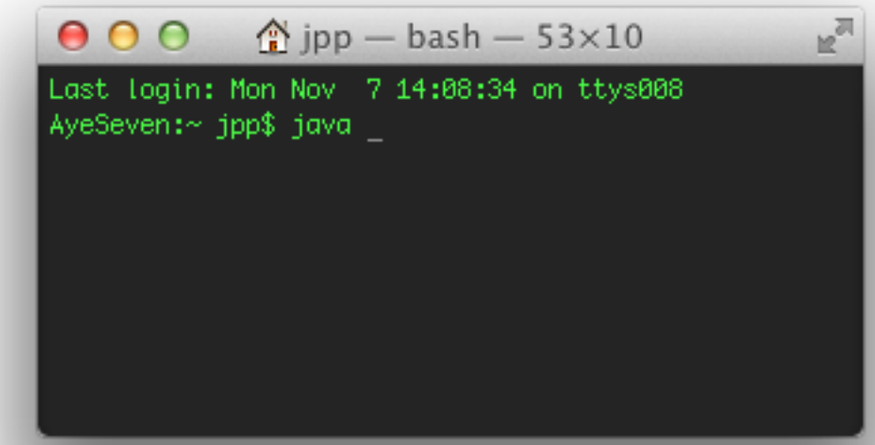
Miniprojet (suite)

- Utilisation des concepts vus au premier semestre
- Exercices plus graphiques, plus «sympathiques»
  - Mais concepts plus spécialisés
- Le cours continue de voir certains concepts généraux de Python
  - En parallèle, on voit certains concepts liés spécifiquement au miniprojet
  - Le tout sera matière d'examen

Cours de cette semaine

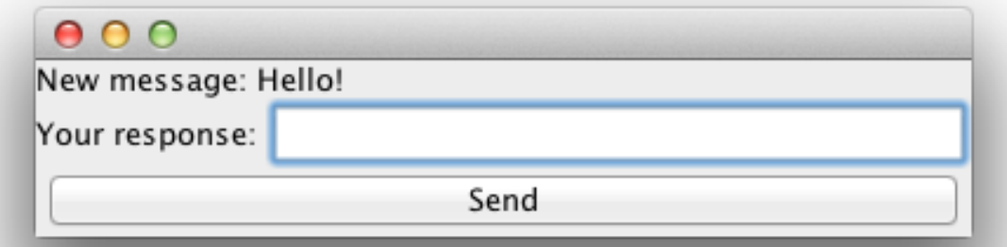
Introduction à Tkinter

Ligne de commande vs. GUI



Ligne de commande

Le programme lit et écrit des lignes de texte sur un terminal ou une console



Graphical User Interface

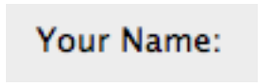
Le programme affiche des fenêtres et répond aux interactions de l'utilisateur

Programmation des GUI

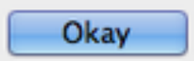
- Au début (1970-1980), beaucoup plus compliqué que la ligne de commande
  - Complexité du clavier, de la souris, du dessin à l'écran
  - Interaction possible avec n'importe quel élément graphique à n'importe quel moment
  - Puissance limitée des machines
- Avec Python: approche facile, orientée objet
  - Tkinter: une librairie de Python pour gérer les GUI, basée sur Tk

Tkinter *(Tee-Kay-inter)*

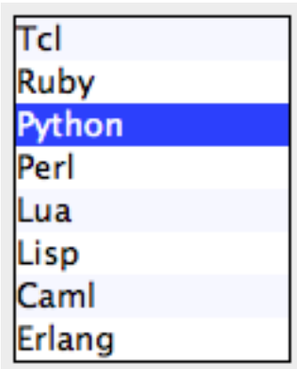
Chaque élément graphique est représenté par un objet



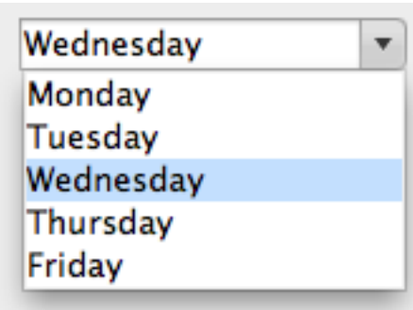
Label



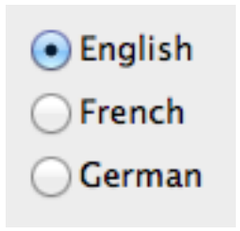
Button



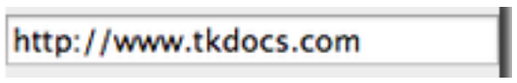
Listbox



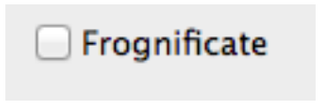
Combobox



Radiobutton



Entry



Checkbutton

Progressbar

Scrollbar

Les éléments Tkinter

- Tous ces éléments sont configurés et sont placés dans une *fenêtre racine représentant l'application*, une instance retournée par l'appel Tk()
- On définit aussi comment le programme réagit en cas d'interaction avec ces objets (clic, touche, etc.)

Démo

Créer une nouvelle fenêtre Tkinter

```
import tkinter as tk
from tkinter import Tk, Label

root = Tk()

screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

width = 500
height = 100
left = (screen_width - width) // 2
top = (screen_height - height) // 3

root.geometry(f"{width}x{height}+{left}+{top}")

label = Label(root, text="Welcome to Tkinter")
label.grid()
label.pack()

root.mainloop()
```

On importe tous les *widgets* qu'on va utiliser

On crée de l'objet racine, obligatoire; fournit aussi une fenêtre de base

On centre la fenêtre sur l'écran *(optionnel, bien sûr)*

On ajoute chaque widget en précisant son «parent» et une liste de propriétés

On utilise un *geometry manager* pour gérer la position du widget (soit l'un [grid], soit l'autre [pack]; pas les deux)

On lance la boucle principale

Boucle principale dans les GUI

```
root.mainloop()
```

- Boucle qui attend les événements et les dispatche au(x) widget(s) responsable(s) de les traiter
- Pseudocode:

— ***répéter*** *tant qu'il y a une fenêtre ouverte:*

➡ ***attendre*** *le prochain événement (clavier, souris, etc.)*

➡ ***déterminer*** *le widget qui doit traiter cet événement (position, focus, etc.)*

➡ ***donner l'information*** *au widget (appel de méthode ou autre)*

Geometry strings

```
root.geometry(f"{width}x{height}+{left}+{top}")
```

- Chaque fenêtre a: largeur, hauteur, décalage X (left), décalage Y (top) par rapport à l'écran
- Origine: coin supérieur gauche
- Pas possible de faire `root.width = 400`
- Mais appel d'une méthode `geometry(s)`
- `s` est un "*geometry string*" et a la forme "*w*x*h*±*x*±*y*"

Positionnement des éléments

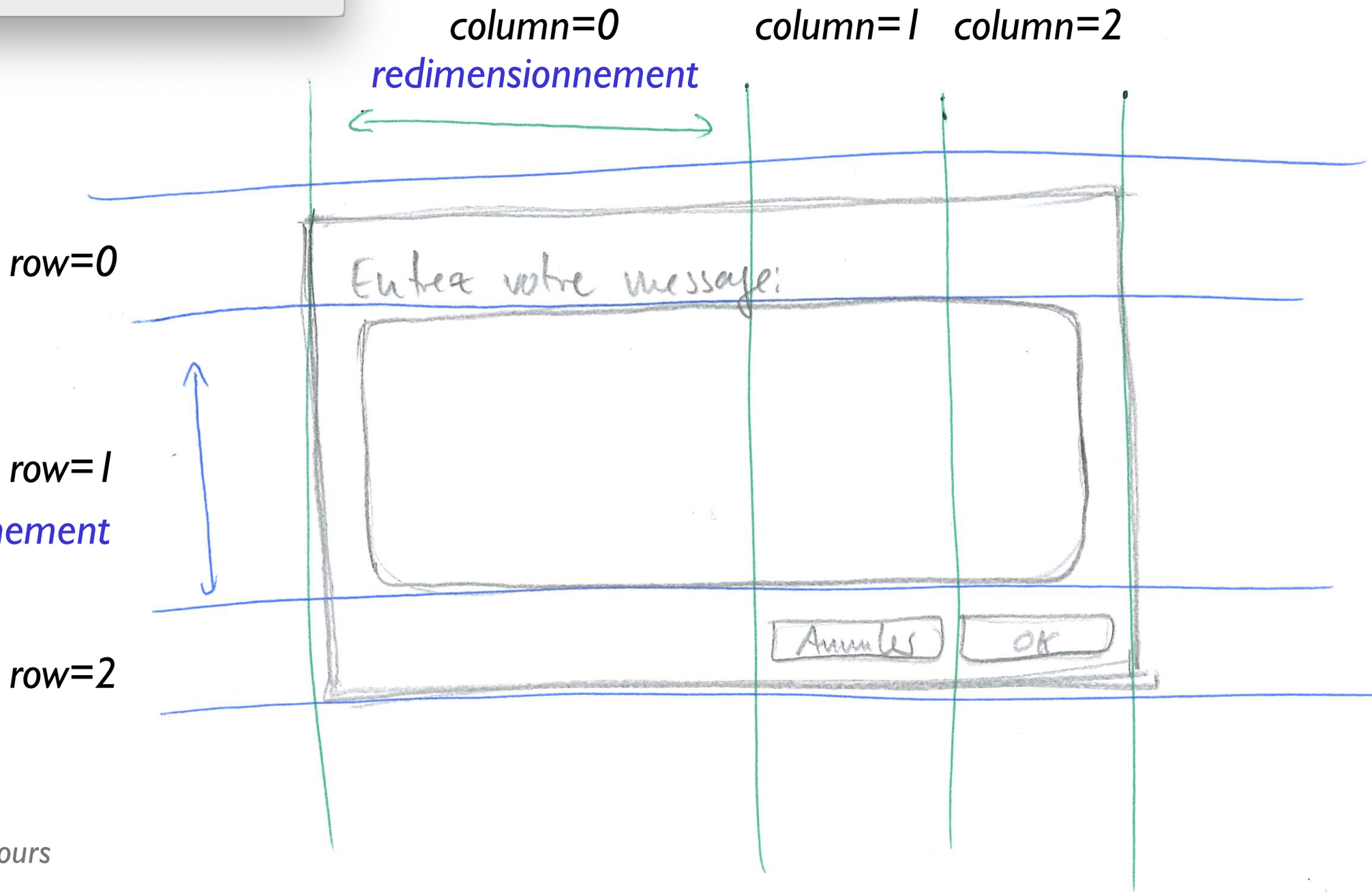
- Chaque élément doit aussi avoir: largeur, hauteur, décalage X, décalage Y
- Redimensionnement et/ou déplacement avec la fenêtre? Distribution de l'espace? Pénible à faire à la main
- *Geometry managers*
  - Après avoir créé chaque widget, on indique comment il se positionne *par rapport à son parent*
  - Approche recommandée: méthodes *pack()* ou *grid(…)*
    - ➡ Selon exemple plus loin

Frames et layouts

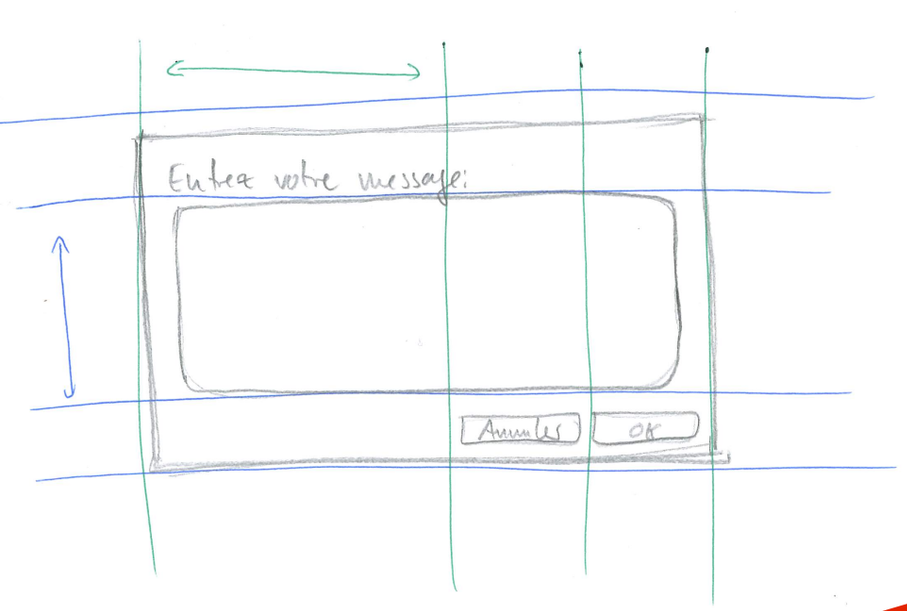
- Chaque fenêtre (dont la fenêtre principale retournée par Tk()) a un geometry manager
- Il doit être le même pour tous ses "enfants"
- Des fois, sous-structure nécessaire (sous-grilles)
- Les frames sont invisibles, mais permettent de regrouper des éléments (ou de faire des bordures/cadres)
- Une frame peut être donné comme parent pour d'autres sous-widgets

— … qui peuvent être arrangés selon une sous-grille

Exemple plus complet



Exemple plus complet




```
root = Tk()
root.geometry("500x180")
root.configure(bg="#ECECEC")

frame = Frame(root, borderwidth=10)
frame.pack(fill=BOTH, expand=True)

for i in range(3):
    frame.columnconfigure(i, pad=10)
for i in range(3):
    frame.rowconfigure(i, pad=10)

frame.rowconfigure(1, weight=1)
frame.columnconfigure(0, weight=1)

label = Label(frame, text="Entrez votre message:")
label.grid(columnspan=3, sticky=W)

text_field = Text(frame)
text_field.grid(row=1, columnspan=3, sticky=NSEW)

cancel_button = Button(frame, text="Annuler")
cancel_button.grid(column=1, row=2)

ok_button = Button(frame, text="OK", default=ACTIVE)
ok_button.grid(column=2, row=2)
```

Autour de tous les widgets, il y a une marge de 10

On centre frame dans root

Toutes les rangées et colonnes ont une marge de 10

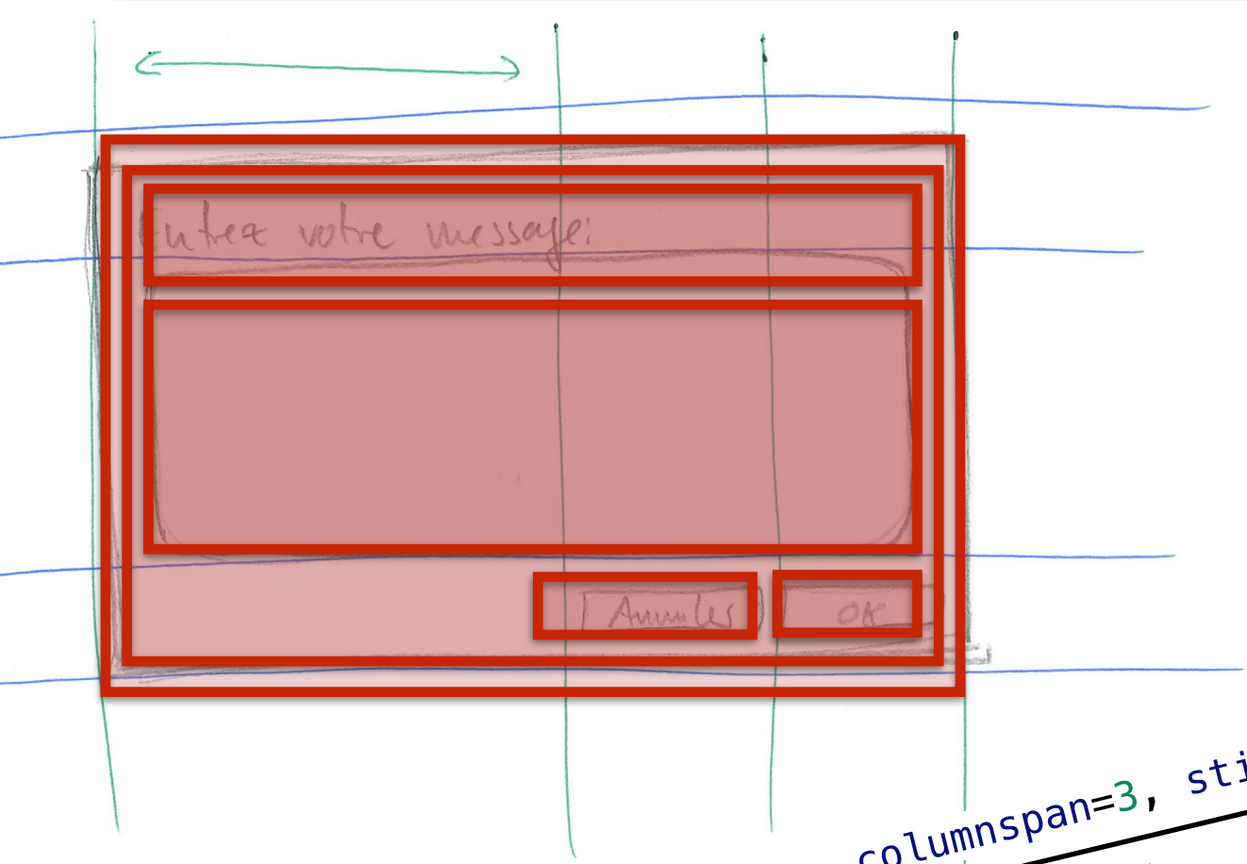
Row 1 sera redimensionnée; Column 0 sera redimensionnée

«Je fais 3 colonnes de large, je suis aligné à l'ouest (W; gauche)»

«Je suis en row=1, je "colle" à tous mes côtés (NSEW)»

«Je suis en row=2 et column=2, je suis le bouton par défaut»

Exemple vu graphiquement



Positionnement: recettes

- Pour remplir tout le parent (sans grille) et être redimensionné:
  - `widget.pack(fill=BOTH, expand=True)`
- Pour configurer les colonnes/lignes d'un parent pour le redimensionnement:
  - `parent.columnconfigure(column_index, weight=1)`
    `parent.rowconfigure(row_index, weight=1)`
- Pour se mettre dans la colonne `c` et la rangée `r` du parent:
  - `widget.grid(column=c, row=r, sticky=NSEW)`
  - `NSEW`: indique le positionnement/redimensionnement à l'intérieur de la cellule
    - ➡ N: aligné en haut; S: aligné en bas; NS: redimensionné verticalement
    - ➡ W: aligné à gauche; E: aligné à droite; EW: redimensionné horizontalement
    - ➡ Combinaisons possibles
  - Paramètres `columnspan` et `rowspan` pour occuper plusieurs lignes ou colonnes

Réagir à un événement

- Deux mécanismes

  — Certains widgets (comme Button) appellent directement une fonction indiquable lors de l'initialisation

```
def button_clicked() -> None:
    print("clicked!")

button = Button(root, text="Click Me", command=button_clicked)
```

Juste le nom de la fonction, sans appel avec ()

  — «Abonnement» à des événements d'un type donné

```
def space_was_typed(e: Event) -> None:
    print("typed a space")

root.bind("<Key-space>", space_was_typed)
```

Les fonctions appelée par un événement doivent accepter un paramètre de type Event

Documentation Tkinter

- Référence Tkinter 8.5:
  https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html
- Tutoriel (plusieurs langages):
  https://tkdocs.com/tutorial/index.html
- Exemples pour grid():
  http://effbot.org/tkinterbook/grid.htm
- Liste des événements:
  http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

Résumé Cours 9

- Python fournit des bibliothèques pour la programmation GUI. Notre exemple: *Tkinter*
- Chaque élément à l'écran est un objet: `Label`, `Button`, `Entry`, etc.
- Chaque widget est positionné via un geometry manager
- Pour réagir aux interactions avec les éléments, on s'abonne à des événements et on prépare des fonctions pour y réagir
  - Clic, clavier, déplacement souris, redimensionnement, etc.
  - Les fonctions sont passée sans appel avec ()

Annonce

Vendredi prochain: demi-salle d'exercices uniquement en BC07-08 (journée des gymnasiens). Pour les MX: prenez votre machine si vous le pouvez!