

Séance #3 - Interfaces graphiques

Thibault Raffailac

Ingénieur pédagogique (PhD, Interaction Homme-Machine)

thibault.raffailac@ec-lyon.fr

Plan du cours

I. Définitions et historique

1. Qu'est-ce qu'une interface graphique ?
2. Pourquoi concevoir une interface graphique ?
3. Historique des innovations
4. Interfaces de bureau
5. Interfaces Web
6. Interfaces tactiles

2. Fonctionnement technique

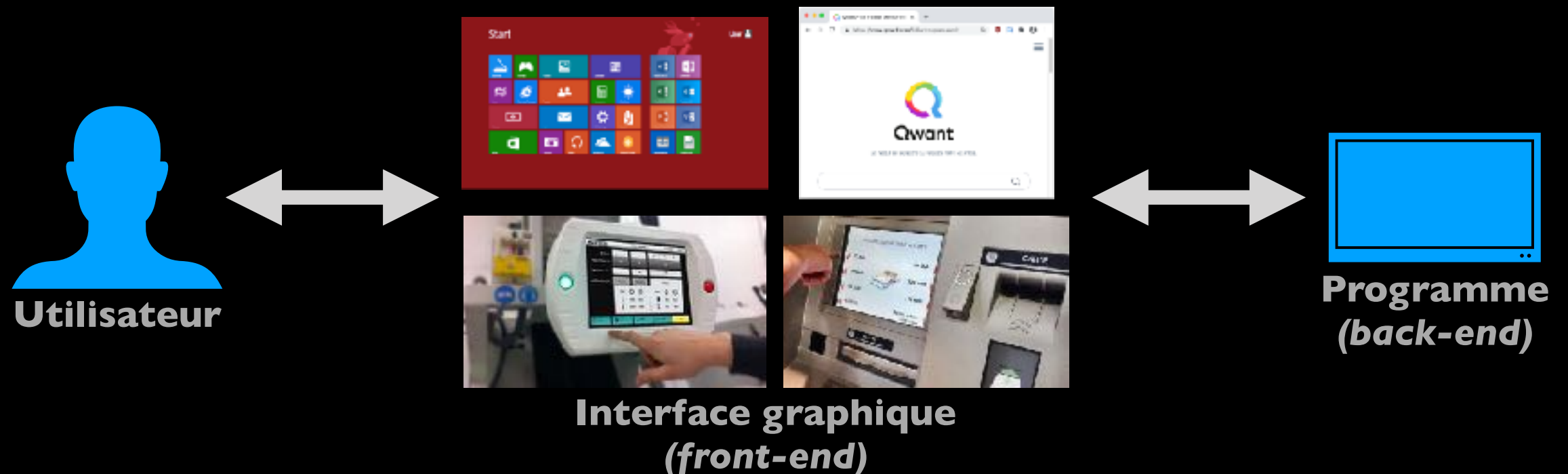
3. Ingénierie d'une interface avec objets

4. Bases de design visuel

Qu'est-ce qu'une interface graphique ?

Une *interface* est l'ensemble des dispositifs matériels et logiciels qui permettent à une personne de commander, contrôler, superviser un système informatique.

L'*interface graphique* est une interface dotée d'un écran et souvent d'un dispositif de pointage, avec laquelle une personne interagit en agissant sur des éléments graphiques dessinés à l'écran.



Pourquoi concevoir une interface graphique ?

Avant les interfaces graphiques, on utilisait les ordinateurs avec des *interfaces en ligne de commande* (et avant, avec cartes perforées...).

- On donne un ordre, la machine l'exécute, et renvoie un résultat

Problème : Si on ne sait pas comment utiliser la machine ?

- *Read The Fucking Manual!*

```
mar@marina:~$ cd /usr/local/src/app-shell/bash $ sudo /etc/crontab/crontab status
Password:
* status started
mar@marina:~$ cd /usr/local/src/app-shell/bash $ ping -q -c 1 en.wikipedia.org
PING en.wikipedia.org (91.191.174.2) 30004 bytes of data:

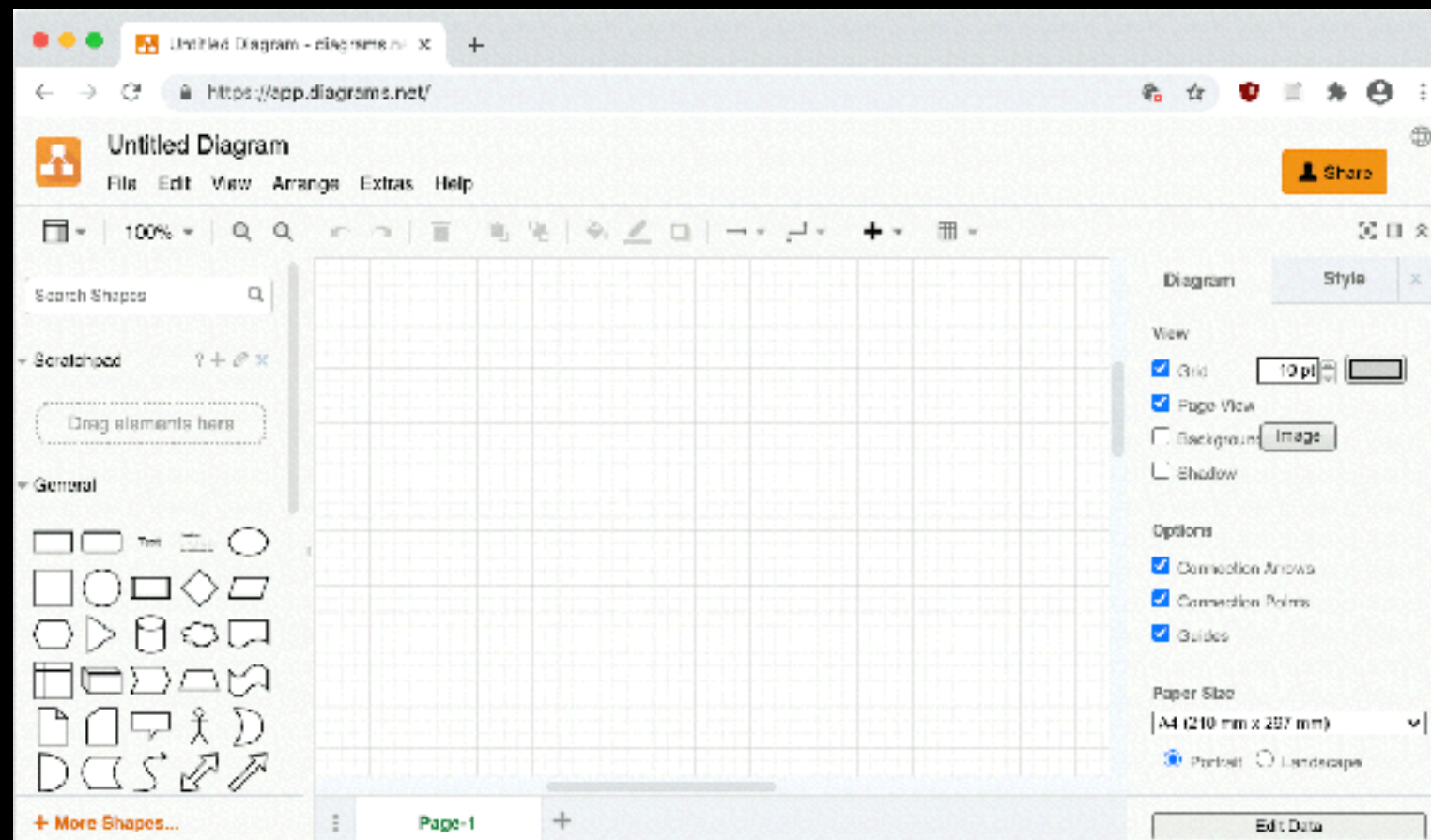
--- en.wikipedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 0.022/0.022/0.020/0.000 ms
mar@marina:~$ cd /usr/local/src/app-shell/bash $ grep -r /dev/sda /etc/fstab --exclude=
/dev/sda1 /boot
/dev/sda2 none
/dev/sda3 /

mar@marina:~$ cd /usr/local/src/app-shell/bash $ date
Sat Aug 3 02:42:24 HST 2009
mar@marina:~$ cd /usr/local/src/app-shell/bash $ lsmod
Module                  Size  Used by
rmdis_wlan              28424  0
rmdis_host              8796  1 rmdis_wlan
cdc_ether               5642  1 rmdis_host
usb_lsm                 18888  3 rmdis_wlan,rmdis_host,cdc_ether
ccrport_pci             30424  0
qlnx                   2888128 22
parport_lsm             39848  1 parport_lsm
iTCO_wdt                10272  0
iTCO_i801               9600  0
mar@marina:~$ cd /usr/local/src/app-shell/bash $
```

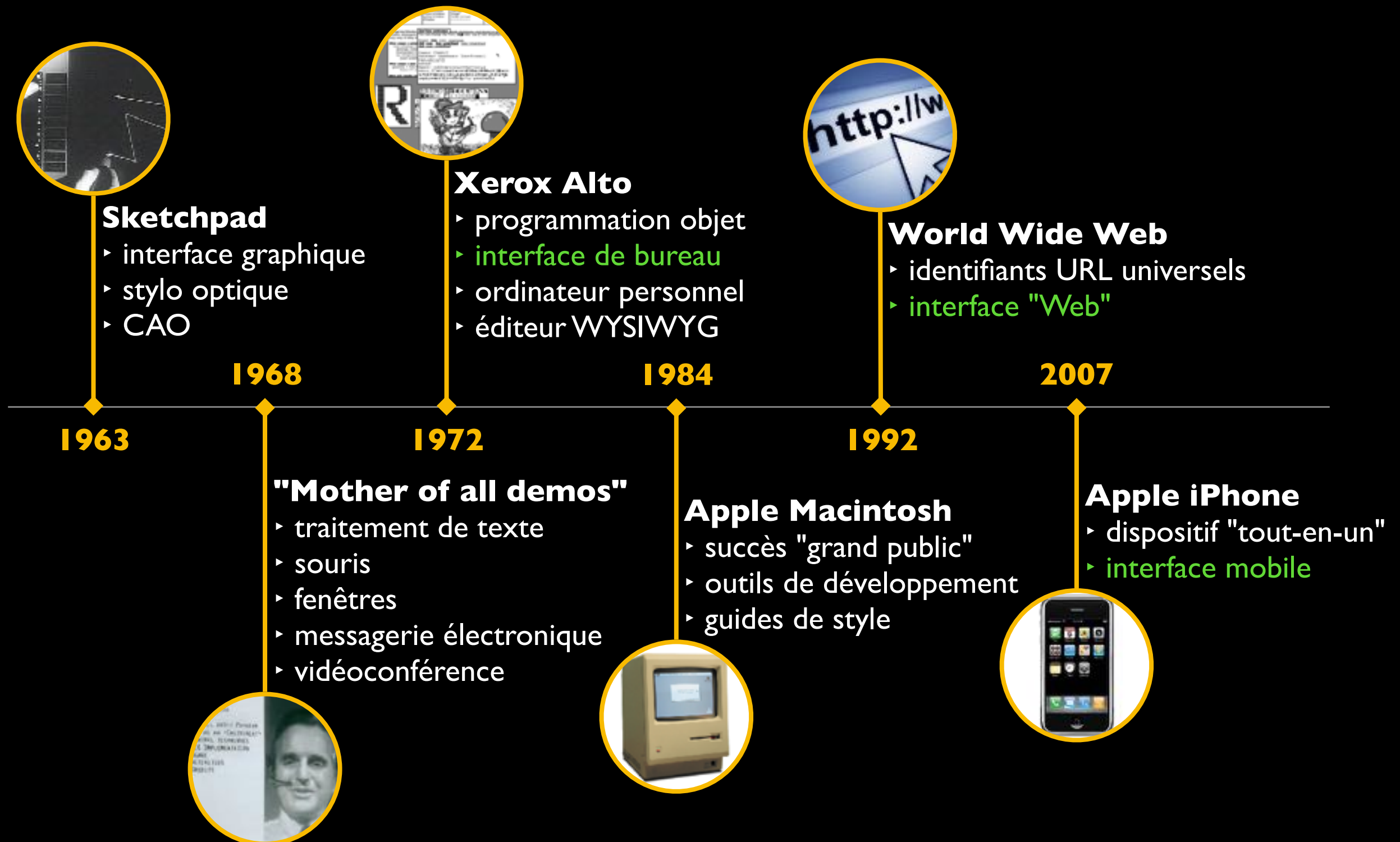
Pourquoi concevoir une interface graphique ?

Solution : afficher toutes les options à l'écran (icônes) ou faciliter leur recherche dans des sous-catégories (menus), manipuler les éléments visuels directement à l'écran (pointeurs)

- Réduction de la difficulté d'apprentissage d'une application



Historique des innovations



Historique des innovations

Aujourd'hui, trois types d'interfaces cohabitent :

- interfaces de bureau (*desktop*)
- interfaces Web accessibles par un navigateur Web
- interfaces mobiles sur smartphones, tablettes et écrans tactiles

Les interfaces de jeux vidéo ou Réalité Virtuelle sont trop variées et en évolution pour être clairement définies aujourd'hui.

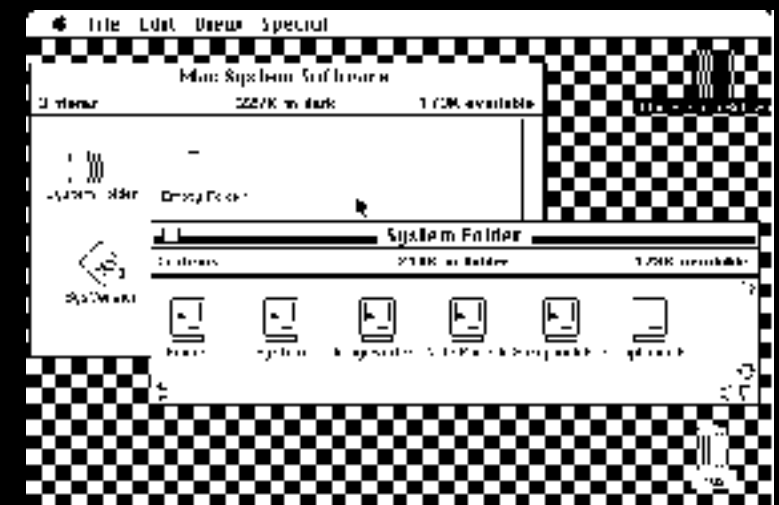
Interfaces de bureau

Conçues pour faciliter l'adoption par des usagers familiers d'un bureau.

- On y retrouve les dossiers et fichiers, le glisser-déposer (*drag&drop*), la possibilité d'éparpiller les documents ouverts sur le bureau (fenêtres), ainsi que de nombreux accessoires (corbeille, bloc-notes, calculatrice, horloge, ...).

Elles se sont progressivement enrichies d'éléments hors bureau.

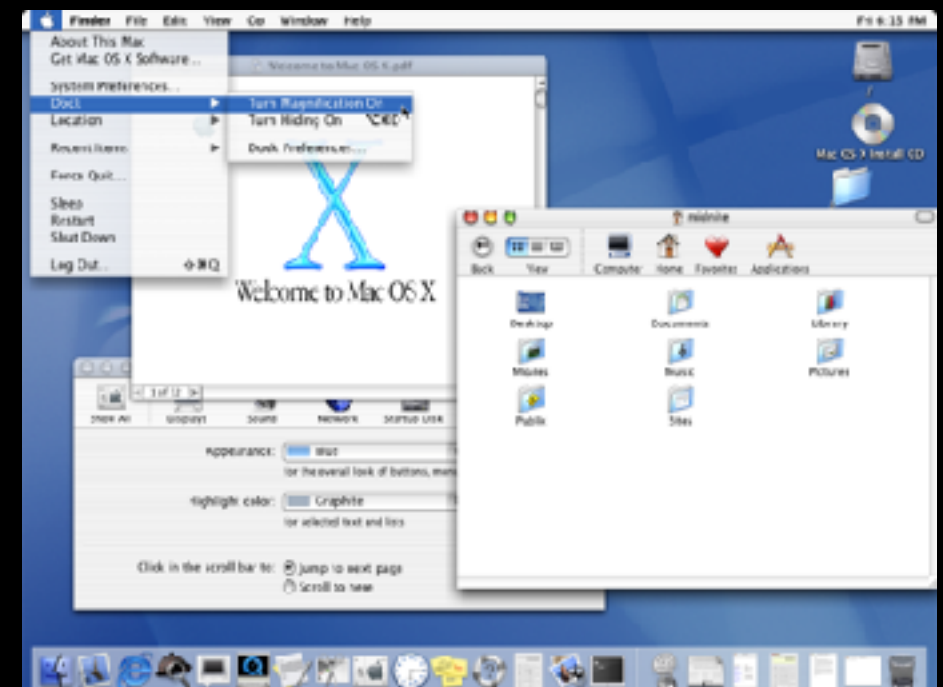
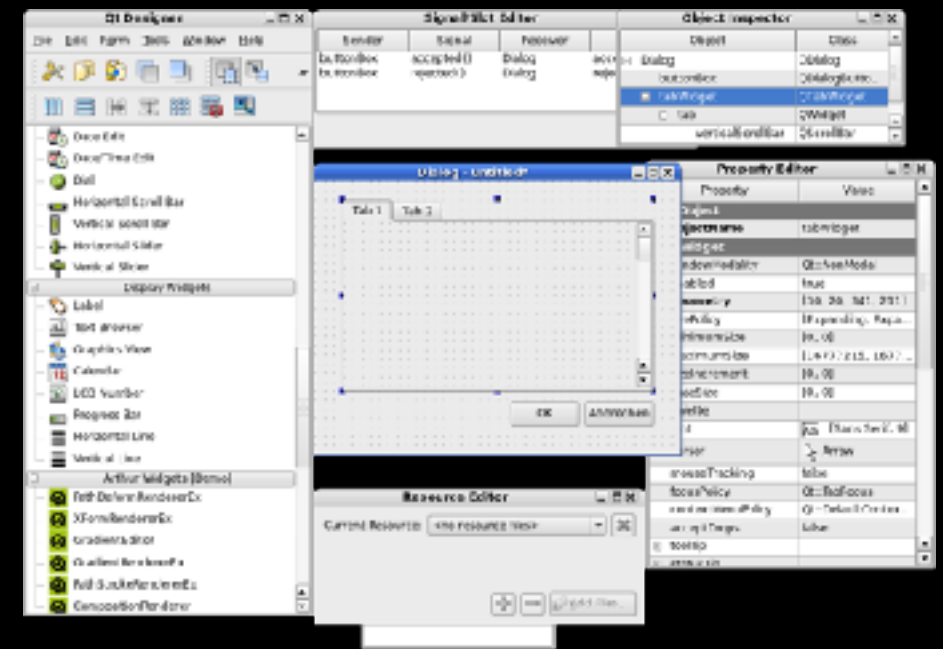
- Barres de menu, fonds d'écran, boutons, barres de défilement, retour en arrière (*undo*), ...



Interfaces de bureau

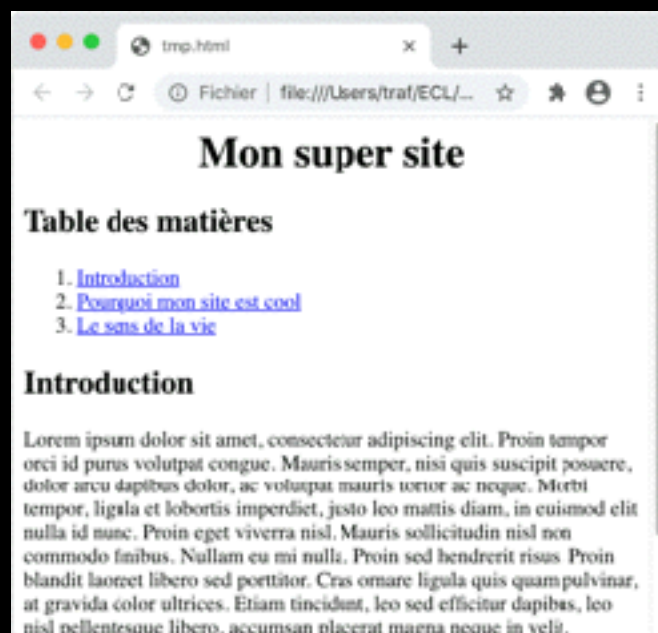
Une *interface de bureau* a accès à une fenêtre d'écran, une souris et un clavier. Elle peut dessiner librement, mais utilise généralement les éléments communs aux interfaces de bureau (boutons, menus, ...) pour faciliter le développement et l'adoption par les utilisateurs.

Exemples d'outils (informels, si vous souhaitez approfondir) : MFC/WPF (Windows), Cocoa (macOS), X11/Motif (Linux), Qt (multiplateformes)

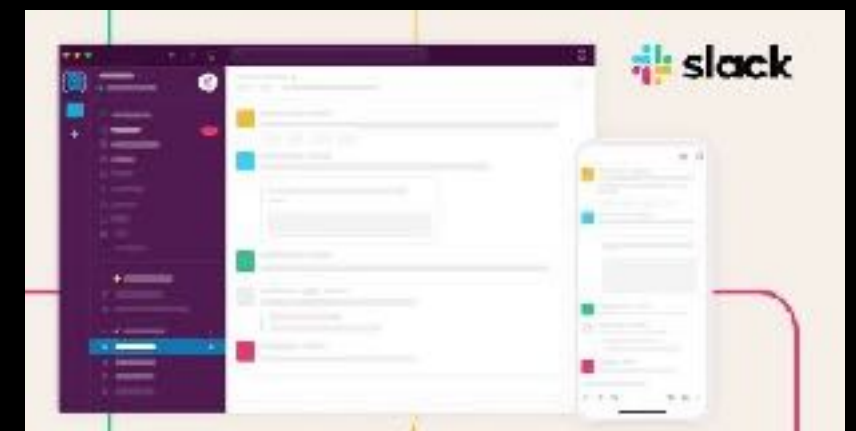


Interfaces Web

Conçues pour représenter des documents textuels accessibles par le Web, et pouvant se référencer mutuellement (liens *hypertexte*). Elles ont évolué pour représenter des formulaires (ex. sondages), puis tous types d'interfaces aujourd'hui.



source: leagueoflegends.fandom.com



source: slack.com

Interfaces Web

Une *interface Web* est un ensemble de pages de largeurs fixes et infiniment longues, dans lesquelles les éléments visuels s'insèrent par défaut les uns en dessous des autres (comme des paragraphes). Les pages se référencent mutuellement par liens hypertexte.

Un navigateur Web (ex. Firefox) est nécessaire pour accéder à l'interface Web depuis un environnement de bureau ou un smartphone.

Le développement Web repose aujourd'hui principalement sur les langages HTML, CSS et JavaScript (voir INF-tc3 et cours d'options pour approfondir).

Interfaces mobiles

Conçues pour le contexte particulier des smartphones :

- écran plus petit, horizontal ou vertical
- pointeur imprécis (problème du *fat finger*)
- attention limitée des utilisateurs (perturbation environnementale)



Interfaces mobiles

Une *interface mobile* a accès à un écran de petite taille horizontal ou vertical, un pointeur avec pression (doigt), et de nombreux capteurs (caméra, radio, NFC, accélération, orientation, ...).

Il y a souvent une seule application active à l'écran à la fois, les autres étant mises en arrière-plan (écran réduit). On réduit les décorations visuelles à l'écran (attention limitée) et on évite les cases à cocher et petits boutons (*fat finger*).

Exemples d'outils : Android Studio (Android), XCode (iPhone)

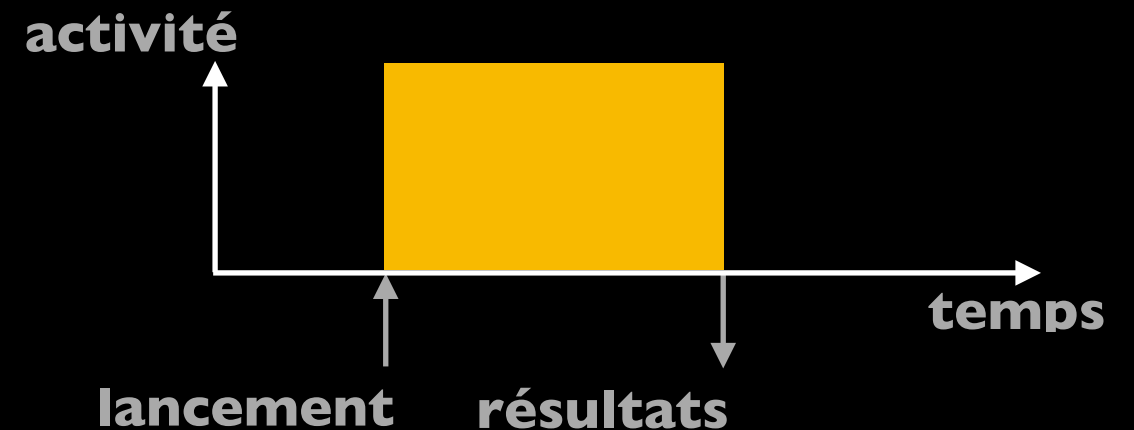
Plan du cours

1. Définitions et historique
2. Fonctionnement technique
 1. Système algorithmique vs. interactif
 2. Boucle de traitement
 3. Lecture des entrées
 4. Distribution d'évènements
 5. Exécution des commandes
 6. Rendu des sorties
3. Ingénierie d'une interface avec objets
4. Bases de design visuel

Système algorithmique vs. interactif

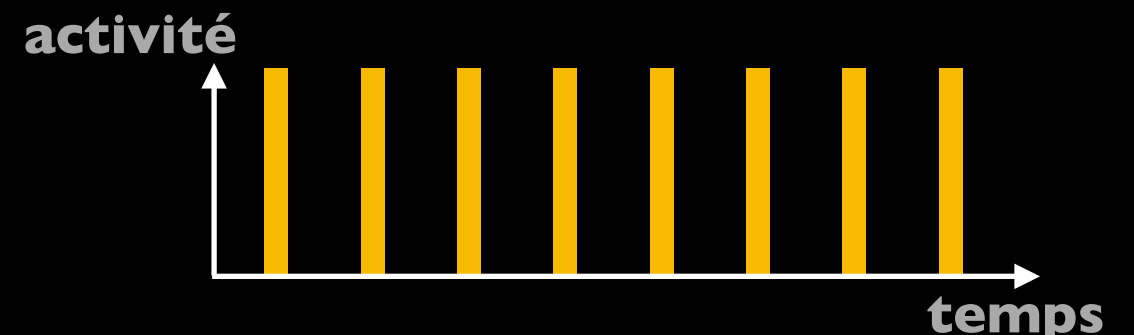
Système algorithmique :

- lit des entrées, fait des calculs, produit des résultats
- termine lorsque les résultats sont obtenus



Système interactif :

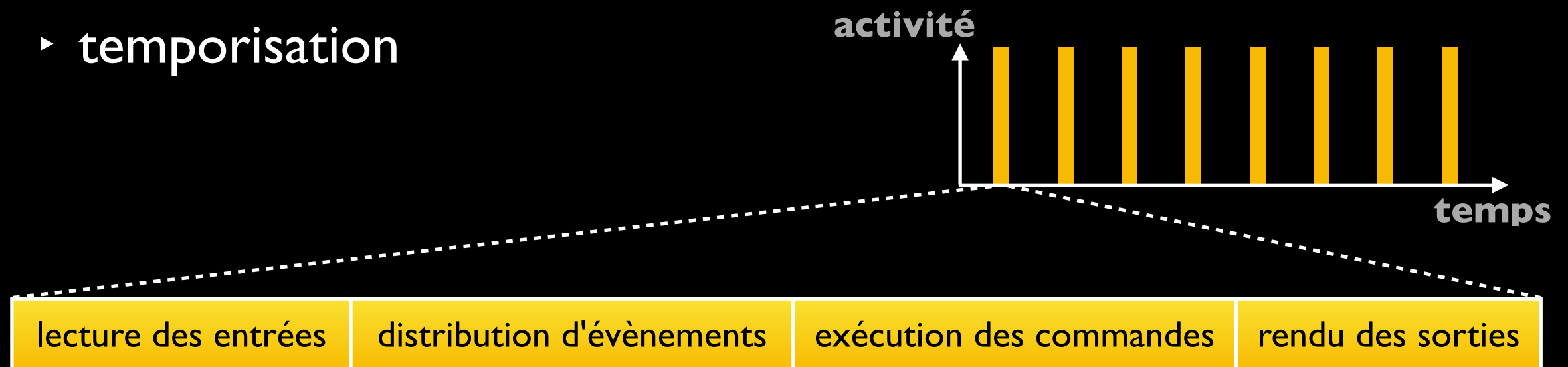
- produit des résultats en réaction à des événements extérieurs
- ne termine jamais (sauf lorsqu'on l'éteint)



Boucle de traitement

Chaque système interactif est une boucle infinie (généralement à 60Hz, fréquence de l'écran) sur une série d'étapes :

- lecture des entrées (ex. coordonnées de la souris)
- distribution d'évènements aux éléments de l'interface
- exécution des commandes (ex. sauvegarder le document)
- rendu des sorties (ex. commandes à la carte graphique)
- temporisation



Lecture des entrées



Pilotes de périphériques

conversion des signaux du périphérique en actions du système (ex. bouton 13 = "Volume Up")

Système d'exploitation

filtrage des informations pour conserver ce que l'application *peut* voir (ex. clics hors fenêtre)

Outils de développement

détectent des actions de plus haut niveau (ex. raccourcis clavier, double clic, *swipe*)

Distribution d'évènements

Lors de la lecture des entrées, chaque action qu'une personne a réalisé s'accompagne d'un ensemble de données. Par exemple un clic de souris est caractérisé par :

- le numéro du bouton cliqué
- la position du curseur à l'écran
- l'horodatage précis du moment du clic
- le numéro de la souris qui a généré le clic (si plusieurs souris)
- les touches du clavier pressées au moment du clic (ex. CTRL)

Distribution d'évènements

Dans une interface graphique, on parle d'évènement lorsqu'on utilise une structure de données pour stocker les données relatives à une action, afin de faciliter leur manipulation dans le programme :

- on les stocke dans des files d'évènements (*event queue*) pour les lire par paquets à intervalles réguliers
- une fonction qui s'exécute en réaction à un évènement (*event handler*) reçoit cet objet en argument

```
def cliquer(event):  
    x = event.x  
    y = event.y  
    print("Vous avez cliqué en {},{}".format(x, y))
```

Exécution des commandes

Imaginons qu'on dispose d'une fonction pour enregistrer le document :

```
def enregistrer_document(nom_fichier):  
    # ... code d'enregistrement ...  
    print("Document enregistré dans le fichier", nom_fichier)
```

- Pour enregistrer manuellement (depuis le code), on exécute simplement `enregistrer_document("fichier.txt")`

Problème : Comment déclencher cette fonction en réaction à une action de l'utilisateur ?

Exécution des commandes

Solution : On passe une fonction (*callback*) au système, qui se charge de l'exécuter chaque fois qu'une action est détectée.

C'est l'*inversion de contrôle*, illustrée par le principe d'Hollywood : « Ne nous appelez pas, c'est nous qui vous appellerons ».

Exemple avec Tkinter (utilisé en TD) :

```
def clic_sur_enregistrer(event):  
    enregistrer_document("fichier.txt")
```

```
bouton_enregistrement.bind("<Button-1>", clic_sur_enregistrer)
```

↑
**2) sur le bouton
représenté par cet objet**

↑
1) chaque clic

↑
3) appellera cette fonction

Rendu des sorties



Pilotes de périphériques

conversion des instructions du système en signaux aux périphériques (ex. volume de % à Volts)

Système d'exploitation

intégration des sorties avec celles des autres applications et du système (ex. mixage audio)

Outils de développement

fonctions facilitant le rendu de sorties complexes (ex. ombres sous les boutons, flou gaussien, ...)

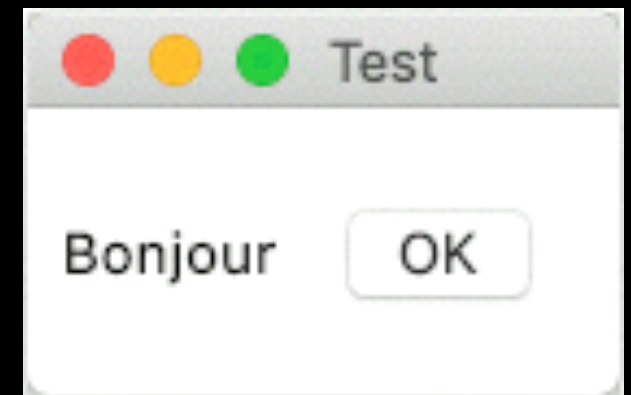
Plan du cours

1. Définitions et historique
2. Fonctionnement technique
3. Ingénierie d'une interface avec objets
 1. Pourquoi utilise-t-on des objets ?
 2. Widgets et arbre de scène
 3. Gestionnaires de positionnement
 4. Séparation de la structure et du code
 5. Séparation de la structure et de l'apparence
 6. Mise en pratique
4. Bases de design visuel

Pourquoi utilise-t-on des objets ?

Jusqu'à présent on a décrit le fonctionnement global à toutes les applications interactives, maintenant on se concentre sur les applications implémentant une interface graphique avec des objets.

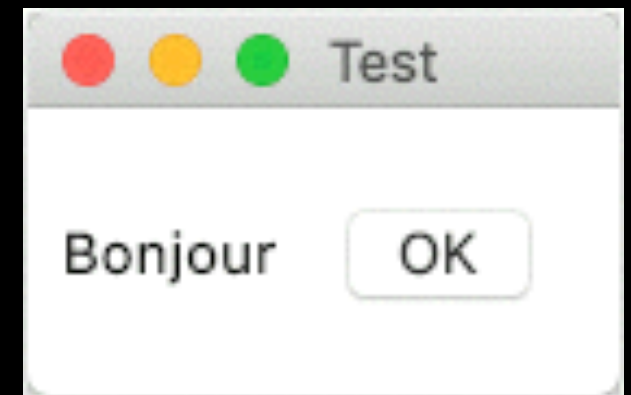
Soit l'interface suivante (un label et un bouton)



Pourquoi utilise-t-on des objets ?

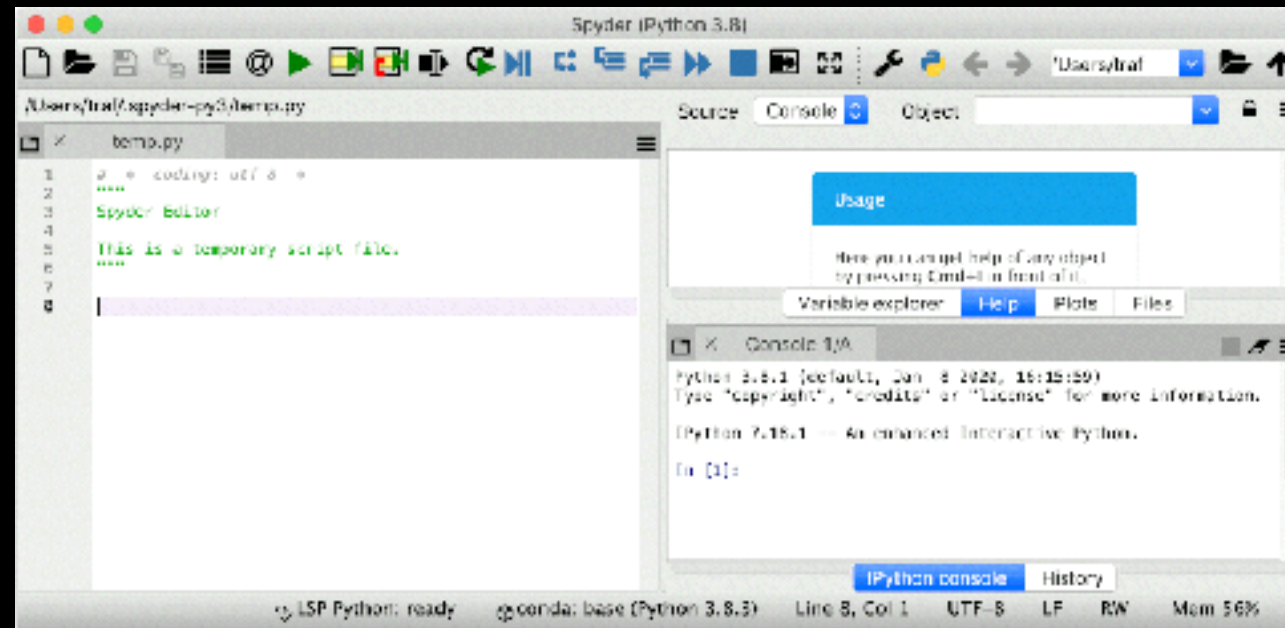
Programmer cette interface "à la main" (sans objets) est assez simple :

- à chaque itération de la boucle de traitement, on dessine "Bonjour", "OK" et un rectangle à bords arrondis autour (la barre de titre est gérée par le système d'exploitation)
- chaque fois qu'un mouvement de la souris est détecté, on vérifie si elle entre/sort du bouton et on change sa couleur si c'est le cas (survol)
- chaque fois qu'un clic de souris est détecté, on vérifie si elle est à l'intérieur du bouton et on déclenche une commande si c'est le cas



Pourquoi utilise-t-on des objets ?

Maintenant :



Problèmes :

- Énormément de boutons et d'éléments à gérer
- Beaucoup de code redondant (ex. dessin de chaque bouton)
- Pourtant chaque bouton a des caractéristiques uniques (ex. icône, commande à exécuter, position à l'écran, ...)

Il faut un moyen de réutiliser le code et les attributs redondants !

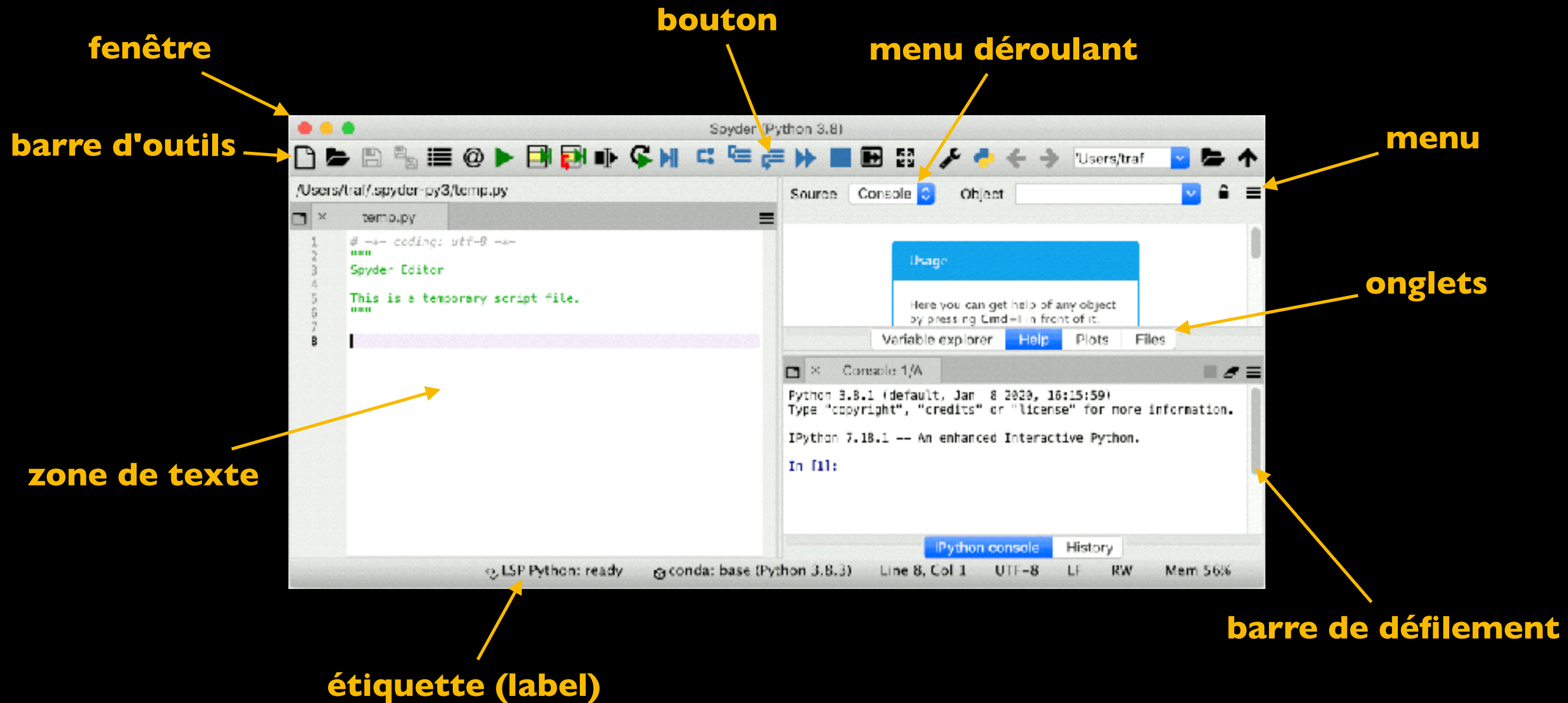
Pourquoi utilise-t-on des objets ?

Solution : On utilise des objets pour les éléments de l'interface !

Notes :

- La programmation par objets est apparue dans les années 60 pour modéliser des systèmes physiques
- Le développement des interfaces graphiques dans les années 70 a fortement contribué au succès des objets
- Aujourd'hui la quasi totalité des interfaces graphiques sont programmées avec des objets (en alternative voir ImGui)

Widgets et arbre de scène



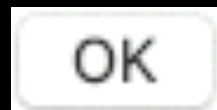
Widgets et arbre de scène

Les *widgets* (*window gadgets*) sont des éléments réutilisables d'une interface graphique, que les programmeurs assemblent pour former une interface complète.

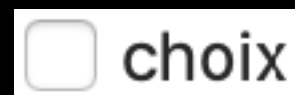
Exemples avec les classes de Tkinter :

A rectangular button-like widget with the text "Bonjour" in a black sans-serif font.

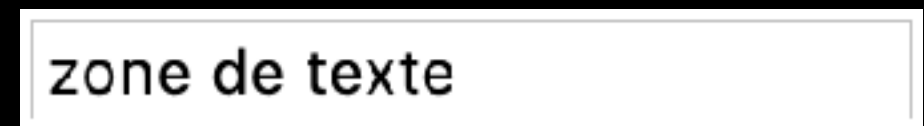
Label

A rectangular button widget with the text "OK" in a black sans-serif font.

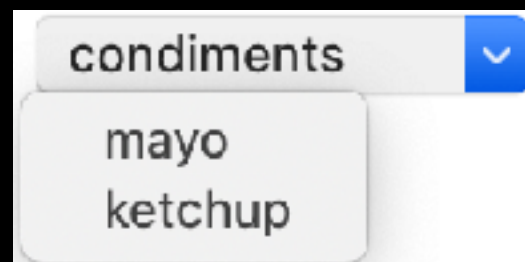
Button

A widget consisting of a small square checkbox followed by the text "choix" in a black sans-serif font.

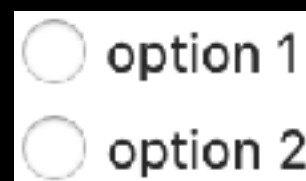
Checkbutton

A rectangular text entry field with a thin border and the text "zone de texte" inside.

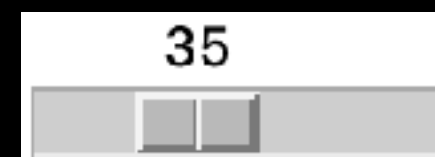
Entry

A widget showing a dropdown menu. The top part is a button labeled "condiments" with a blue downward arrow. Below it, a list of options is visible: "mayo" and "ketchup".

OptionMenu

A widget showing two radio button options. Each option consists of a small circle followed by the text "option 1" and "option 2" respectively.

Radiobutton

A horizontal slider widget. It has a rectangular track with a vertical slider knob in the middle. The number "35" is displayed above the track.

Scale

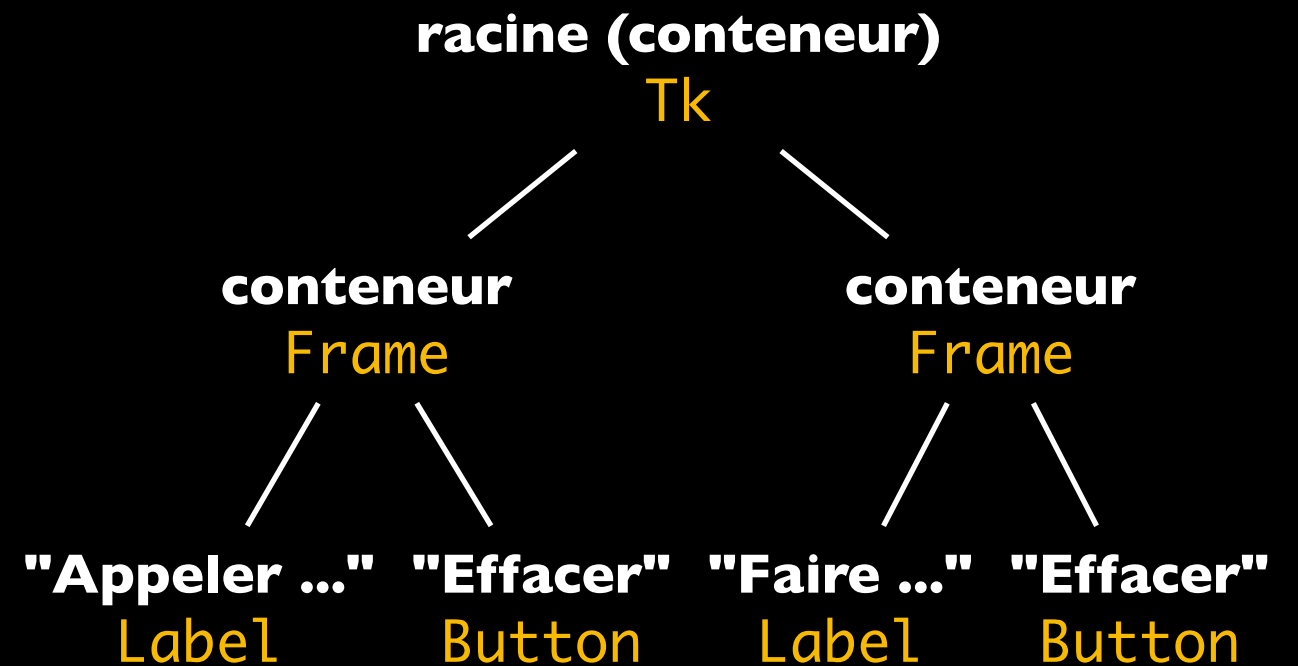
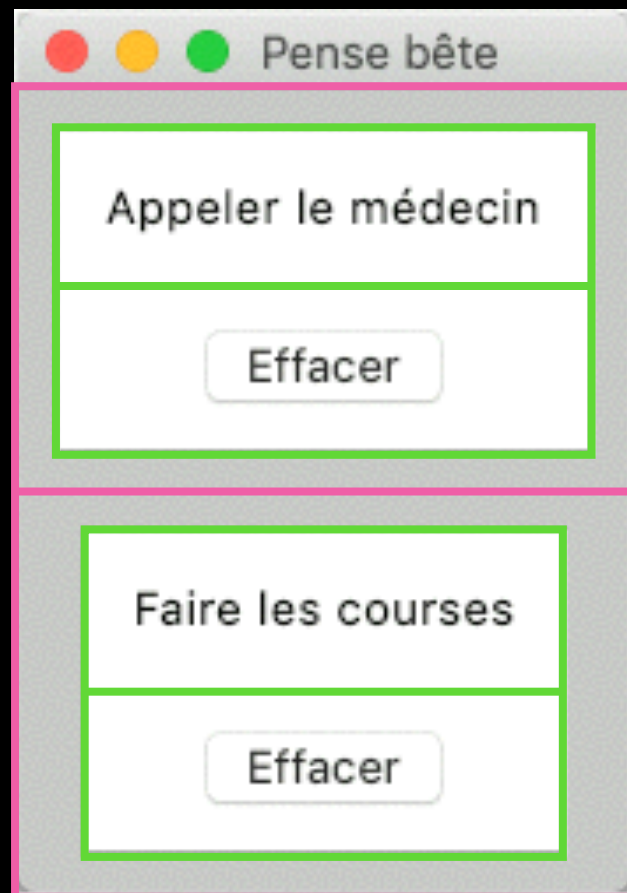
A vertical scrollbar widget, consisting of a vertical track and a small rectangular slider knob.

Scrollbar

Widgets et arbre de scène

Pour créer une interface, on assemble des widgets dans une structure hiérarchique : l'*arbre de scène*. Les noeuds intermédiaires (conteneurs), sont également des widgets.

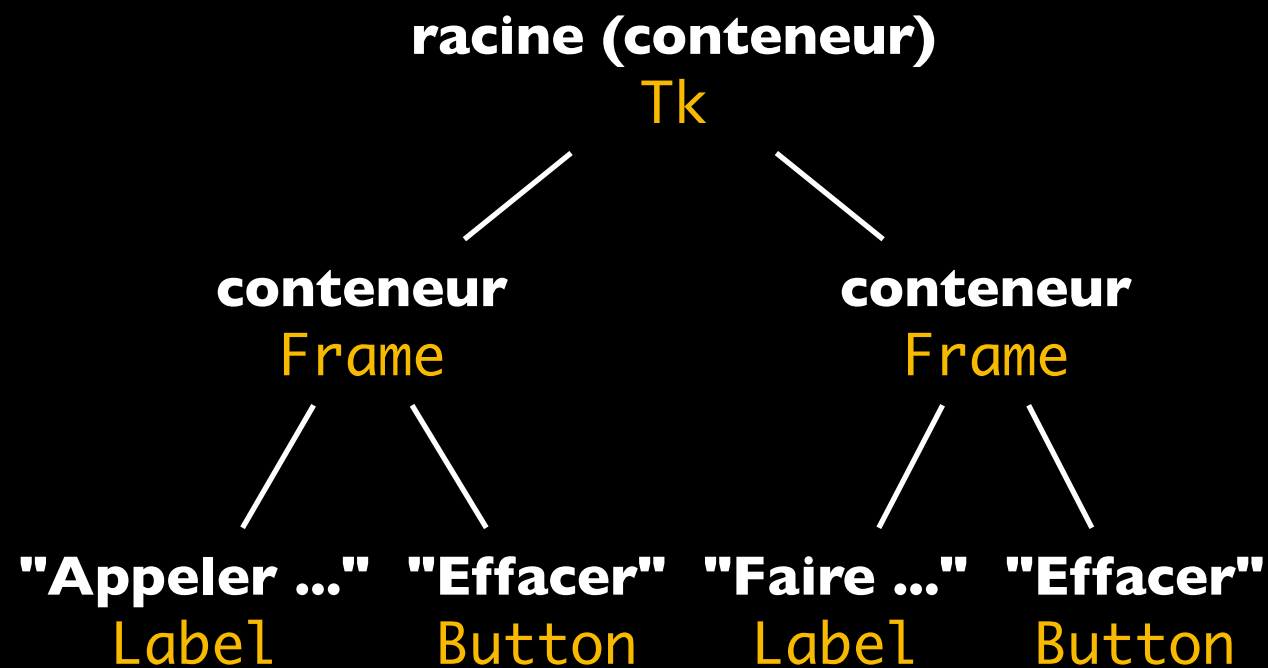
- Chaque parent englobe visuellement ses enfants.



Widgets et arbre de scène

Pour créer une interface, on assemble des widgets dans une structure hiérarchique : l'*arbre de scène*. Les noeuds intermédiaires (conteneurs), sont également des widgets.

- Chaque parent englobe visuellement ses enfants.



```
racine = Tk()
conteneur1 = Frame(racine, ...)
conteneur2 = Frame(racine, ...)
appeler = Label(conteneur1,
    text="Appeler le médecin")
effacer1 = Button(conteneur1,
    text="Effacer")
faire = Label(conteneur2,
    text="Faire les courses")
effacer2 = Button(conteneur2,
    text="Effacer")
```

Gestionnaires de positionnement

Lorsqu'un conteneur est le parent de plusieurs éléments, il y a différentes manières de les arranger visuellement entre eux.

Le *gestionnaire de positionnement* (*layout manager*) désigne l'algorithme chargé de positionner un ensemble de widgets au sein d'un conteneur.

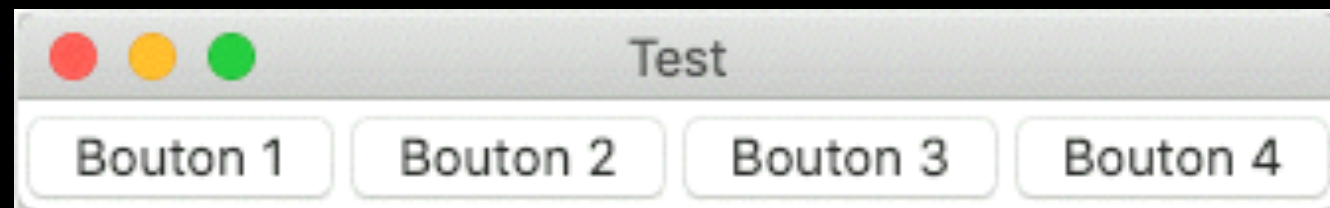
On distingue généralement trois types de positionnement :

- le long d'une direction
- par grille
- absolu

Gestionnaires de positionnement

Le positionnement le long d'une direction (`pack` dans Tkinter) consiste à empiler les éléments les uns à la suite des autres dans une direction donnée (ex. de haut en bas, de gauche à droite).

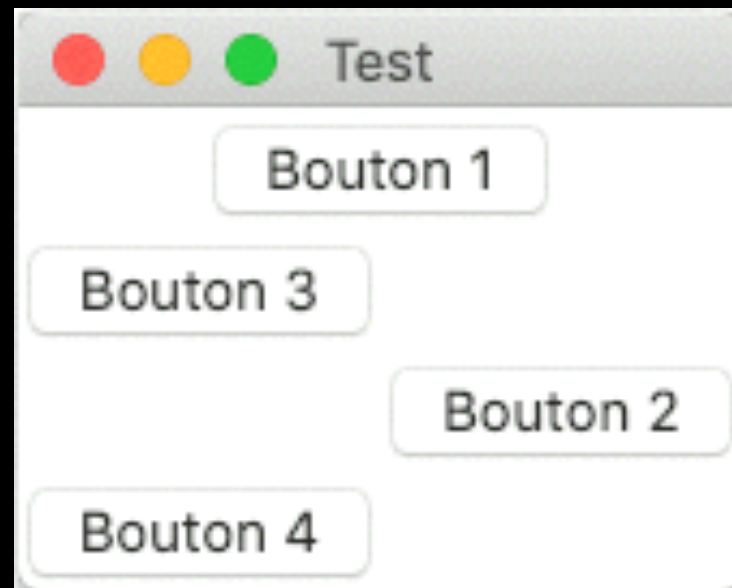
Lorsque les éléments dépassent la taille du conteneur, ils sont ignorés (dans le cas de Tkinter) ou peuvent être placés sur une nouvelle ligne/colonne (dans les interfaces Web).



Gestionnaires de positionnement

Le positionnement sur une grille (*grid* dans Tkinter) consiste à diviser l'espace du conteneur en un nombre de lignes et colonnes (comme un tableur Excel), et y placer les widgets.

Il est généralement possible de positionner un widget à cheval sur plusieurs cellules.



Gestionnaires de positionnement

Le positionnement absolu (`place` dans Tkinter) consiste simplement à passer manuellement les coordonnées auxquelles positionner chaque widget.



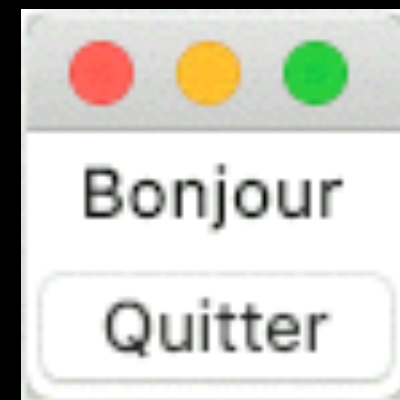
Séparation de la structure et du code

Lorsqu'on crée une première interface avec un arbre de widgets, elle est initialement "inerte" : les actions des utilisateurs sur les boutons, menus ou cases à cocher n'ont aucun effet.

Il reste encore à spécifier les fonctions qui seront exécutées pour chaque action sur chaque widget.

```
# arbre de scène
racine = Tk()
texte = Label(racine, text="Bonjour")
texte.pack()
bouton = Button(racine, text="Quitter")
bouton.pack()
```

```
# commandes de l'interface
bouton.config(command=racine.destroy)
```



Séparation de la structure et du code

On sépare ainsi la *structure* et le *code* d'une interface.

- Il est plus facile de modifier la structure sans changer le code (ex. remplacer une case à cocher par un bouton).
- On peut concevoir très vite un prototype visuel d'interface, pour communiquer sur un futur projet ou recueillir des retours rapides.
- Plusieurs personnes peuvent travailler en parallèle sur une même interface.

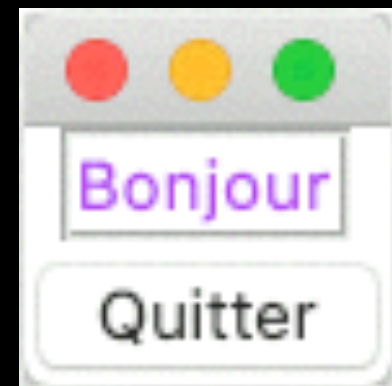
De nombreux outils de développement utilisent même des langages différents pour ces deux aspects (ex. HTML/JavaScript dans les interfaces Web).

Séparation de la structure et de l'apparence

Lorsqu'on crée une première interface avec un arbre de widgets, les éléments ont une apparence visuelle "par défaut" qu'on peut modifier, soit dans leurs constructeurs, soit avec des méthodes.

```
# arbre de scène
racine = Tk()
texte = Label(racine, text="Bonjour")
texte.pack()
bouton = Button(racine, text="Quitter")
bouton.pack()
```

```
# apparence de l'interface
texte.configure(fg="purple", borderwidth=2, relief=GR00VE)
```



Séparation de la structure et de l'apparence

De la même manière qu'avec le code, on sépare ici la *structure* et l'*apparence* de l'interface.





- On peut prototyper en deux temps, d'abord en définissant une structure globale (ex. barre d'outils, barre d'état, document central), puis en peaufinant les éléments individuels.
- Possibilité de confier l'apparence d'une interface à un(e) designer.

La séparation de l'apparence est moins répandue que celle du code (ex. peu pratiquée dans Tkinter), mais certains outils de développement fournissent un langage dédié pour l'apparence (principalement CSS avec les interfaces Web).










Mise en pratique

tk

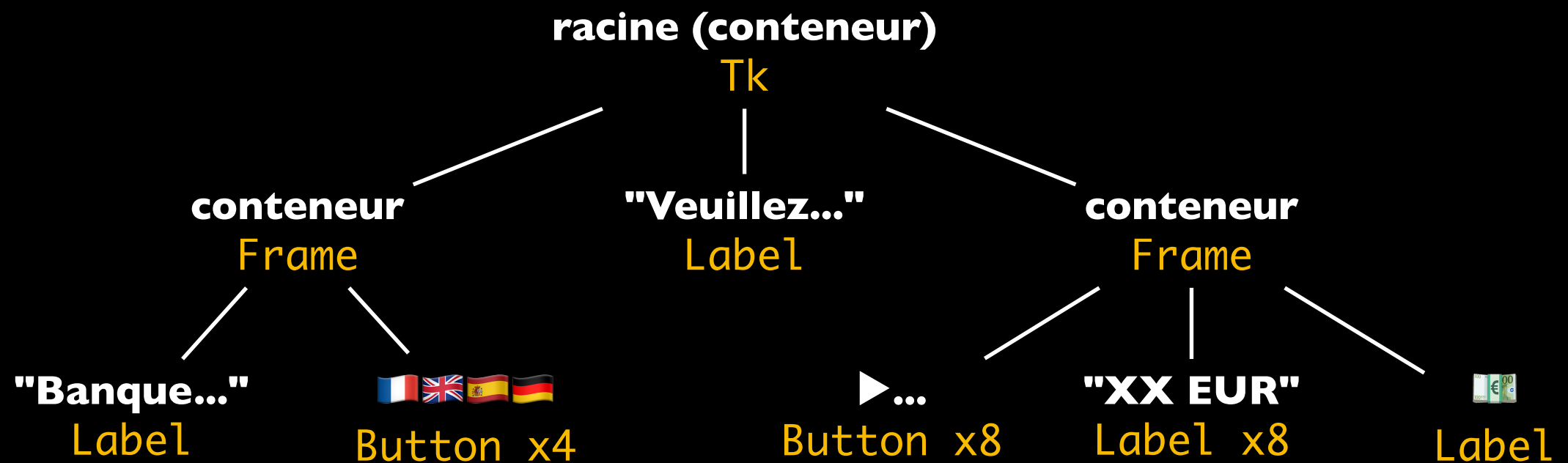
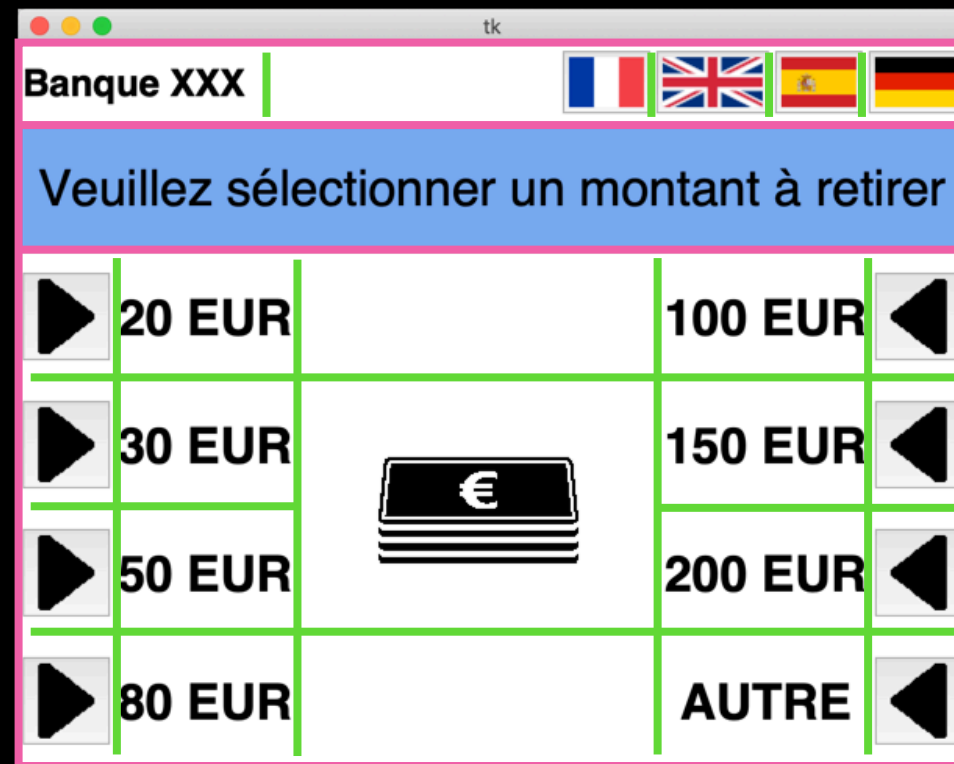
Banque XXX

Veuillez sélectionner un montant à retirer

	20 EUR		100 EUR	
	30 EUR		150 EUR	
	50 EUR		200 EUR	
	80 EUR		AUTRE	

Mise en pratique



Plan du cours

1. Définitions et historique
2. Fonctionnement technique
3. Ingénierie d'une interface avec objets
4. Bases de design visuel
 1. De quoi (ne) parle-t-on (pas) ?
 2. Principes de design C.R.A.P.
 3. Quelques exemples

De quoi (ne) parle-t-on (pas) ?

Le design est un métier : *UX designer* (interfaces graphiques), architecte d'intérieur, graphiste, designer de produit, de packaging, retail, etc.

Cette partie n'a pas pour prétention d'enseigner le design, mais de faciliter le dialogue avec des designers et comprendre leur travail.

On se limite au design visuel, sans notion d'expérience utilisateur (nécessite un cours à part).

Beaucoup de règles sont issues de l'expérience et du "bon sens", et seront frustrantes pour quiconque n'a pas "l'oeil".

- Nous tâchons de passer en revue des éléments de base simples et utiles dans de nombreux contextes.

Principes de design C.R.A.P.

Une vidéo bien réalisée par Ashley Holst :

<https://www.youtube.com/watch?v=JAQbxZAAS6k>

Exemple de poster



*Repas dansant
Costumé
Masqué Déguisé*



Tenue facultative (loups sur place 3 à 5 €)

Vendredi 8 mars 19h30
Hotel-Restaurant Le Râtelier
31530 Montaigut
(19km Nord-Ouest Toulouse)



Danses de Salon
Rock-Salsa-Bachata-Kizomba....
(un peu de danse en ligne et disco....)

Menu Unique 30€ (31€ nouveau adhérents)
Cocktail-Entrée-Plat-Dessert-café-vin
Inscription avant 15 février 2019 (places limitées)
(Régimes spéciaux-repas nous consulter)
Adhésion association (1€) obligatoire (bulletin inscription à remplir)

Association Dansons Tout Simplement (DTS)
contact@dansons-tout-simplement.com
Tel. DTS : 07 67 63 66 45 (SMS ou message)



Chèques et bulletins inscription à envoyer à:
Dansons Tout Simplement BP40071 - 31703 Blagnac

Exemple de poster

- ▶ Pas de mise en valeur des informations importantes (trop de polices et tailles de texte)
- ▶ Utilisation désordonnée de gras, italique, tailles et parenthèses pour mettre en valeur (pas de répétition)
- ▶ Couleurs bleu/rouge sur fond rouge peu lisibles (contraste avant/arrière)
- ▶ Alignement "cassé" par l'image de masque
- ▶ Proximité insuffisamment marquée



The poster is for a masquerade event titled 'Repas dansant Costumé Masqué Déguisé'. It features three images: a masquerade mask, a person in a pirate costume, and a person in a blue and black costume. The text is in various fonts and colors (blue, red, black) on a red background. The event is on Friday, March 8th at 19h30 at Hotel-Restaurant Le Râtelier in Montaigut. It includes details about the menu, registration, and contact information for the Association Dansons Tout Simplement (DTS).

*Repas dansant
Costumé
Masqué Déguisé*

Tenue facultative (loups sur place 3 à 5 €)

Vendredi 8 mars 19h30
Hotel-Restaurant Le Râtelier
31530 Montaigut
(19km Nord-Ouest Toulouse)

Danses de Salon

Rock-Salsa-Bachata-Kizomba....
(un peu de danse en ligne et disco....)

Menu Unique 30€ (31€ nouveau adhérents)
Cocktail-Entrée-Plat-Dessert-café-vin
Inscription avant 15 février 2019 (places limitées)
(Régimes spéciaux-repas nous consulter)
Adhésion association (1€) obligatoire (bulletin inscription à remplir)

Association Dansons Tout Simplement (DTS)
contact@dansons-tout-simplement.com
Tel. DTS : 07 67 63 66 45 (SMS ou message)

Chèques et bulletins inscription à envoyer à:
Dansons Tout Simplement BP40071 - 31703 Blagnac

Exemple de site Web (avant)



Exemple de site Web (après)

The screenshot displays the Bank of India website interface. At the top, a navigation bar includes links for 'About Us', 'Investor Corner', 'Interest Rate', 'Service Charges', 'Forex Card Rate', and 'Career'. A secondary bar on the right contains '+ - Overseas', 'RRBs', 'English', and a 'Search' icon. The main header features the 'Bank of India' logo with the tagline 'Relationship beyond banking', followed by a horizontal menu: 'Personal', 'Corporate', 'Rural', 'NRI', 'Online Services', 'MSME', 'Cards', and 'Offers'. A contact number '1800 220 229' and a hamburger menu icon are positioned on the right side of the header.

The main content area is divided into two sections. On the left, a large blue banner for 'DIGITAL APNAYEN' (15.08.2020 to 31.10.2020) features an image of a hand holding a smartphone displaying the bank's app. Below the banner, the text 'Digital Apnayen. Surakshit Rahan.' is displayed. On the right, a grid of service icons includes 'Mobile Banking', 'PoS', 'BHIM UPI', 'QR Code', 'Internet Banking', 'BHIM AADHAR', 'Debit Card', 'AEPS', and 'Credit Card'.

A vertical sidebar on the right side of the page contains a dropdown menu for 'Internet Banking' with sub-options for 'Personal' and 'Corporate'. Below this are several buttons: 'Interested in Our Products?', 'Locate us', 'Contact Us', 'Communication to BSE/NSE', 'Safe Banking', 'COVID Emergency Support Scheme', 'Covid-19 Emergency Credit', and 'ATM/POS Online Refund'. The 'BOI SEVA' logo is visible at the bottom of the sidebar.

Merci de votre attention !

Cette présentation s'inspire et reprend des éléments de :

- Programmation des interfaces graphiques, d'Anastasia Bezerianos
- Introduction à l'Interaction Homme-Machine, de Nicolas Roussel
- Des ordinateurs et des Hommes, de Stéphane Huot

Sauf indications contraires, toutes les images sont issues de Wikipédia ou produites par moi-même.