

Séance #3 - Interfaces graphiques

Thibault Raffailac

Ingénieur pédagogique (PhD, Interaction Homme-Machine)

thibault.raffailac@ec-lyon.fr

Plan du cours

1. Définitions et historique
2. Fonctionnement technique
3. Ingénierie d'une interface graphique
4. Bases de design visuel

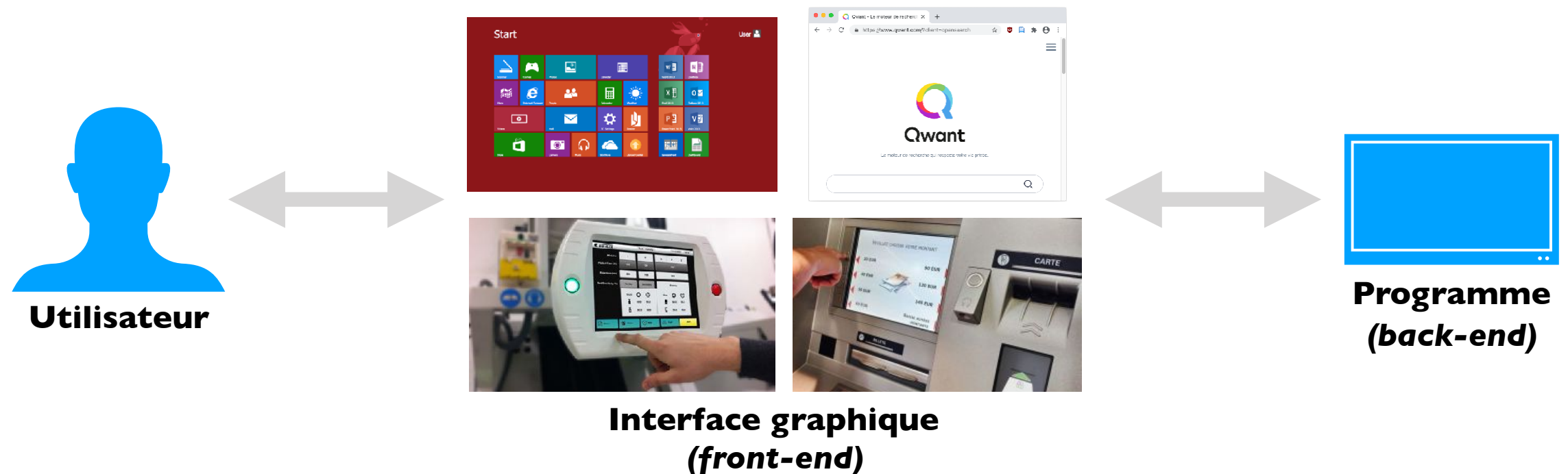
Plan du cours

- 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 graphique
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 des 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!*

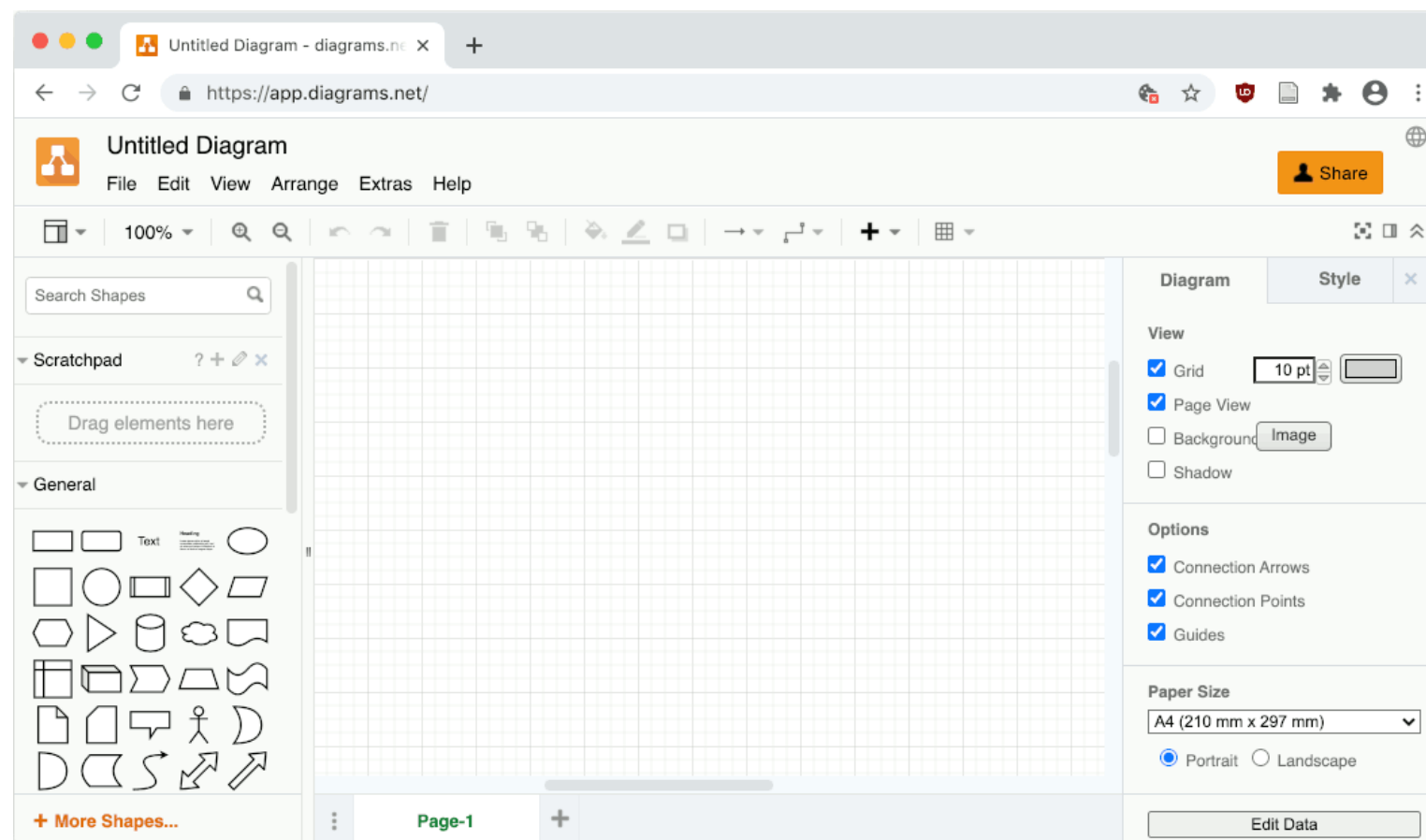
```
mars@marsmain /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
mars@marsmain /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esams.wikimedia.org (91.198.174.2) 56(84) bytes of data.

--- rr.esams.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.820/49.820/49.820/0.000 ms
mars@marsmain /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=3
/dev/sda1      /boot
/dev/sda2      none
/dev/sda3      /
mars@marsmain /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
mars@marsmain /usr/portage/app-shells/bash $ lsmod
Module          Size  Used by
rndis_wlan      23424  0
rndis_host      8696   1 rndis_wlan
cdc_ether        5672   1 rndis_host
usbnet          18688   3 rndis_wlan,rndis_host,cdc_ether
parport_pc      38424  0
fglrx           2388128 20
parport         39648   1 parport_pc
iTCO_wdt        12272   0
i2c_i801         9380    0
mars@marsmain /usr/portage/app-shells/bash $
```

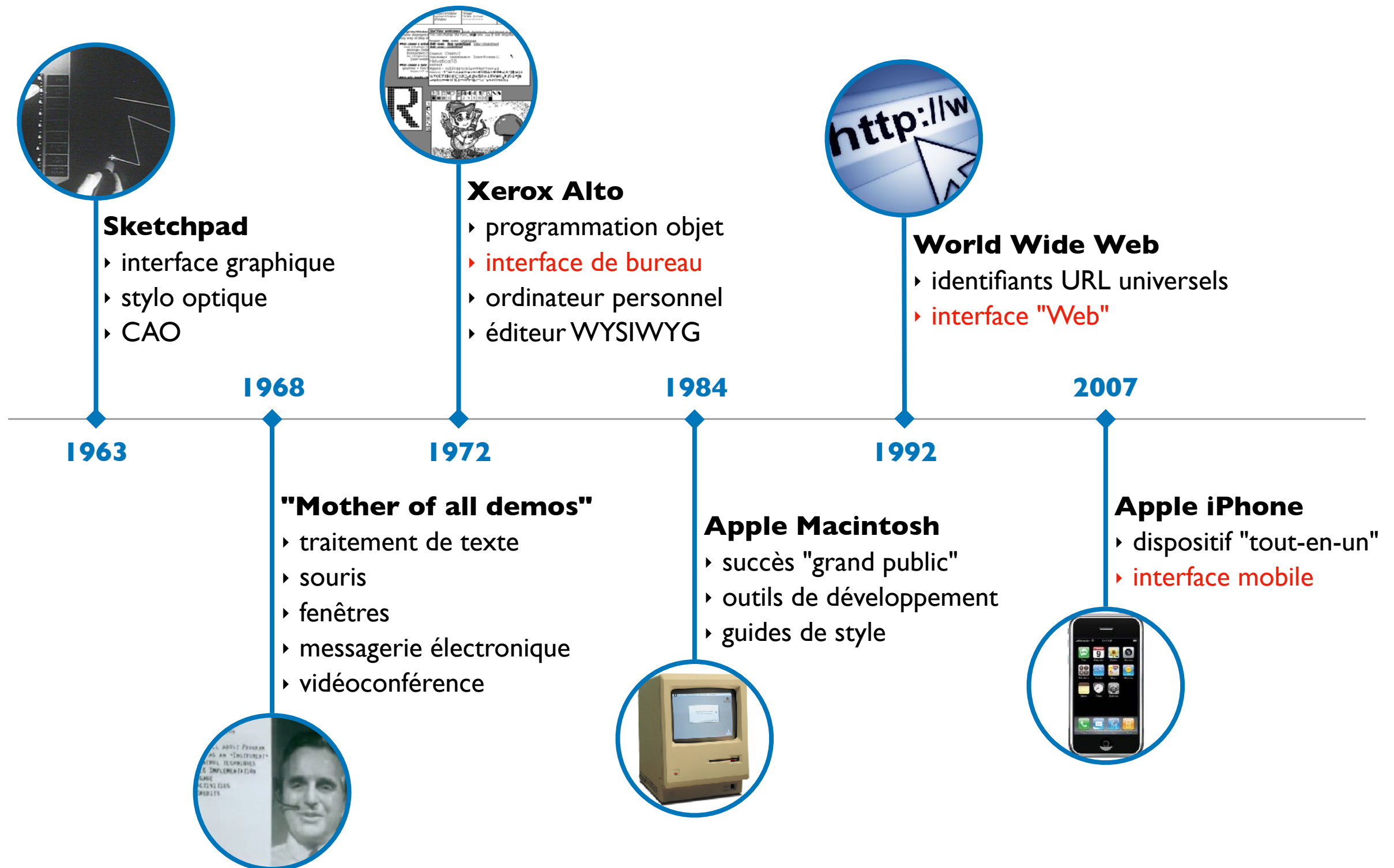
Pourquoi concevoir une interface graphique ?

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.

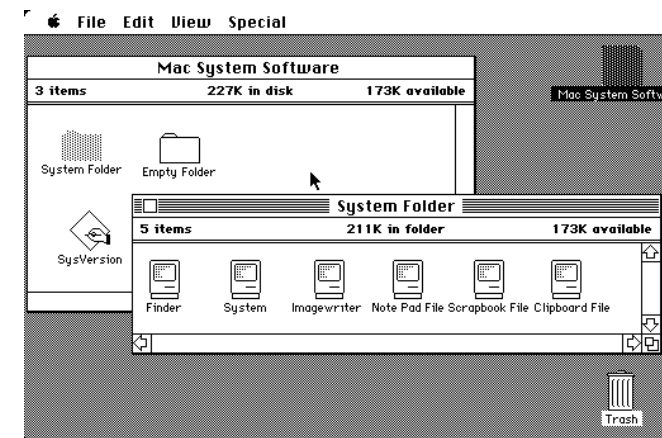
Interfaces de bureau

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.

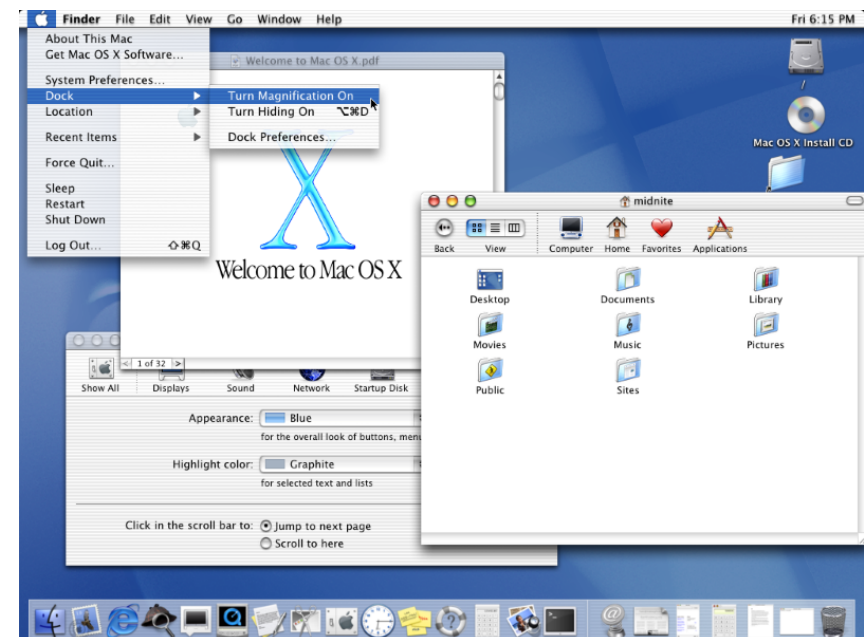
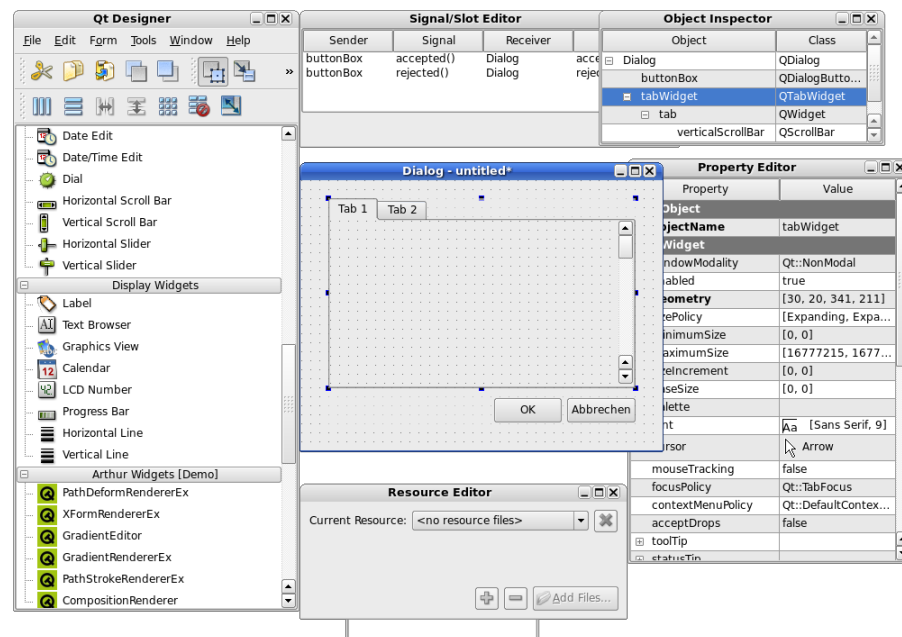
- Barres de menu, fonds d'écran, boutons, barres de défilement, retour en arrière (*undo*), ...



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)

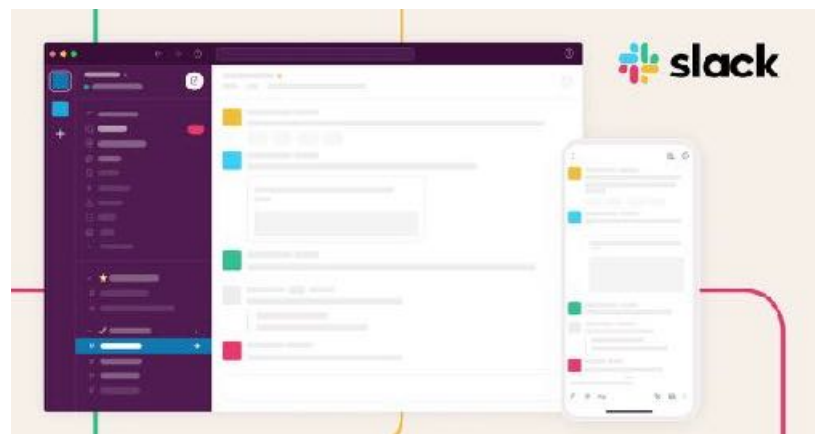


Interfaces Web

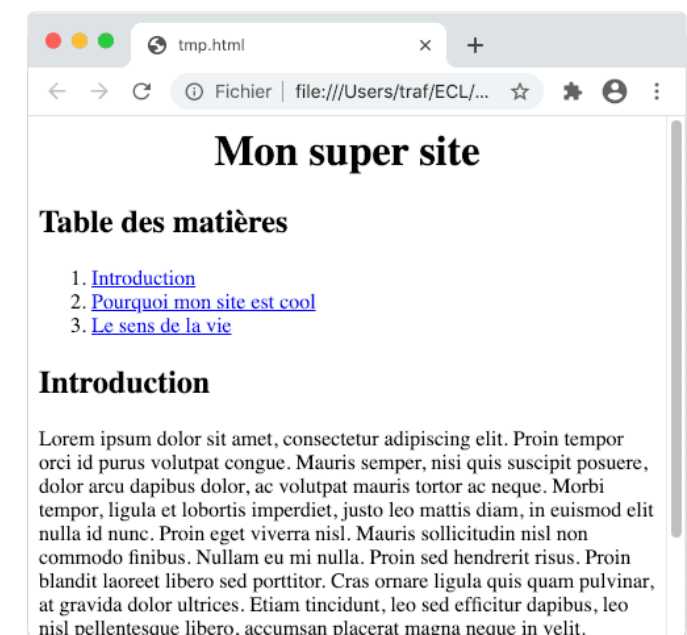
Conçues pour représenter des documents textuels accessibles par le Web, et pouvant se référencer mutuellement (liens *hypertexte*). Elles ont évolué pour représenter des formulaires (ex. questionnaires), puis tous types d'interfaces aujourd'hui.



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 intégrer l'interface Web dans 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 (accélération, orientation, température, luminosité, ...).

Il y a souvent une seule application active à l'écran à la fois, les autres étant mises en sommeil à l'arrière-plan. 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)

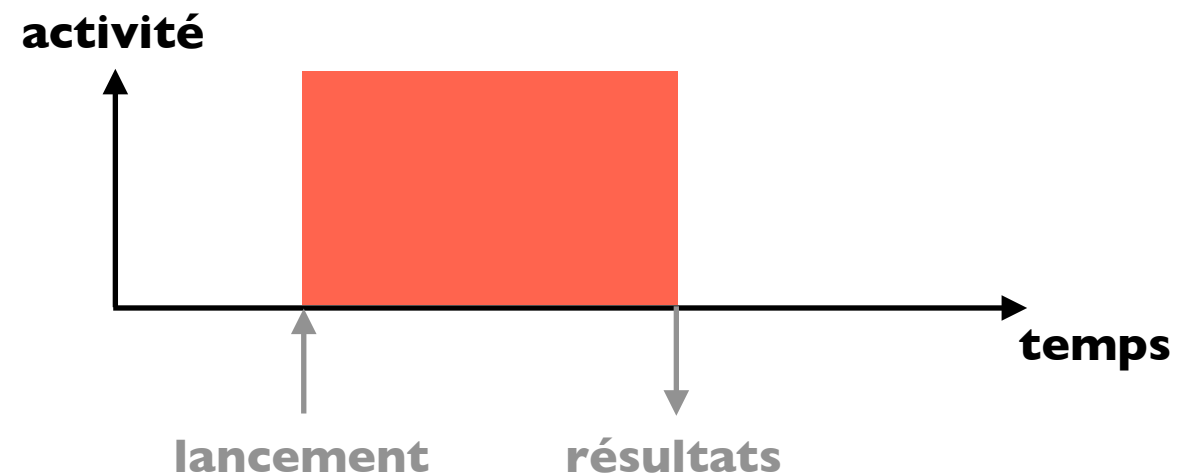
Plan du cours

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 graphique
4. Bases de design visuel

Systeme algorithmique vs. interactif

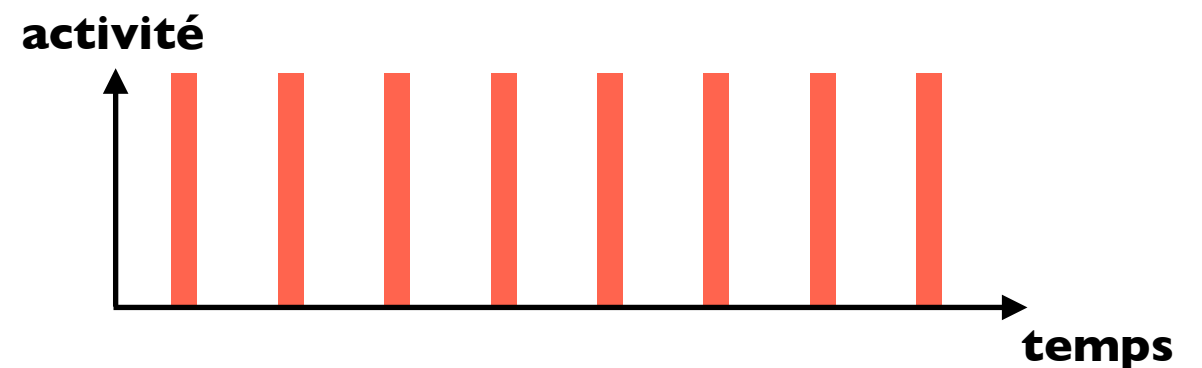
Systeme algorithmique :

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



Systeme interactif :

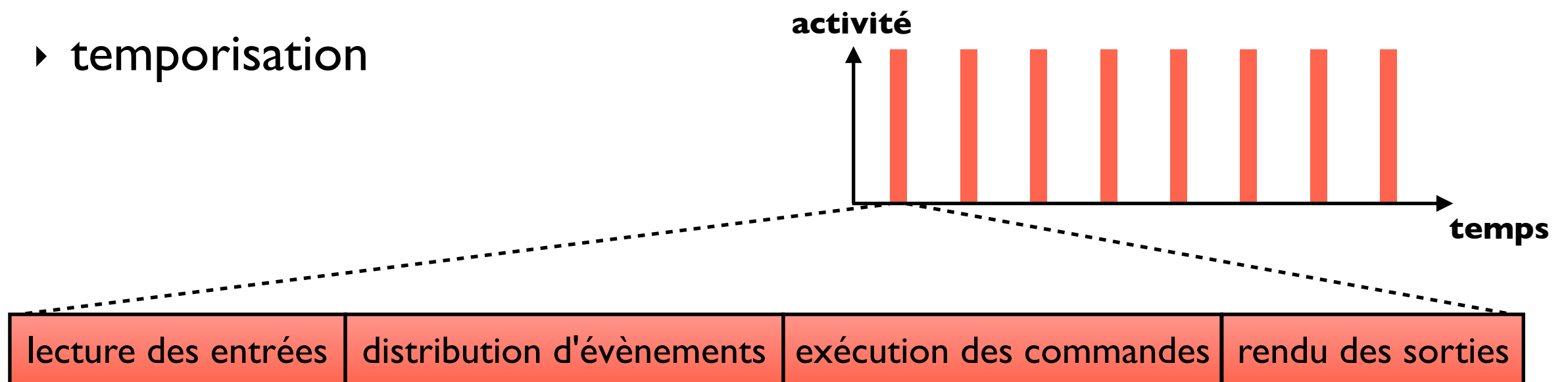
- produit des résultats en réaction à des événements extérieurs
- ne termine jamais (sauf lorsqu'on l'éteint)



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



Lecture des entrées



Pilotes de périphériques

conversion des signaux du périphérique en actions du système (ex. bouton I3 = "Volume Up")



Système d'exploitation

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



Outils de développement

détection 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-clic)

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*, qu'on illustre avec 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 en % vers 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, ...)

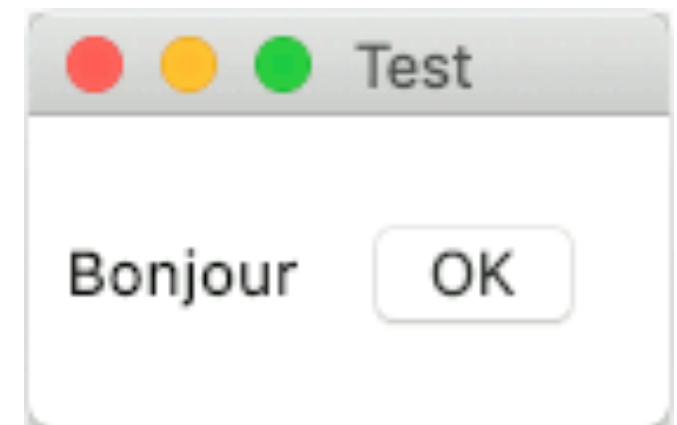
Plan du cours

1. Définitions et historique
2. Fonctionnement technique
3. Ingénierie d'une interface graphique
 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
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.

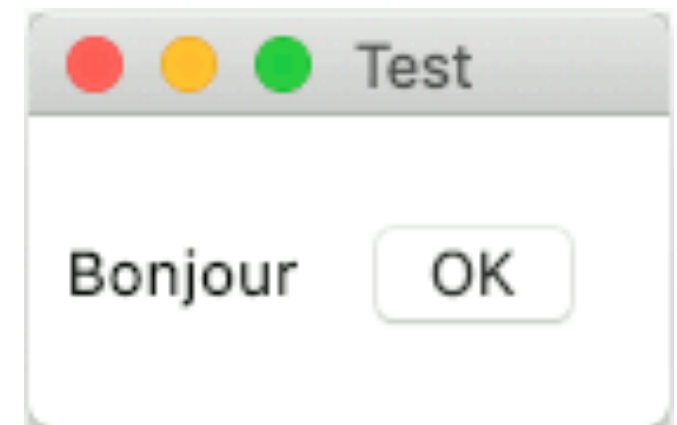
Soit l'interface suivante (un label et un bouton)



Pourquoi utilise-t-on des objets ?

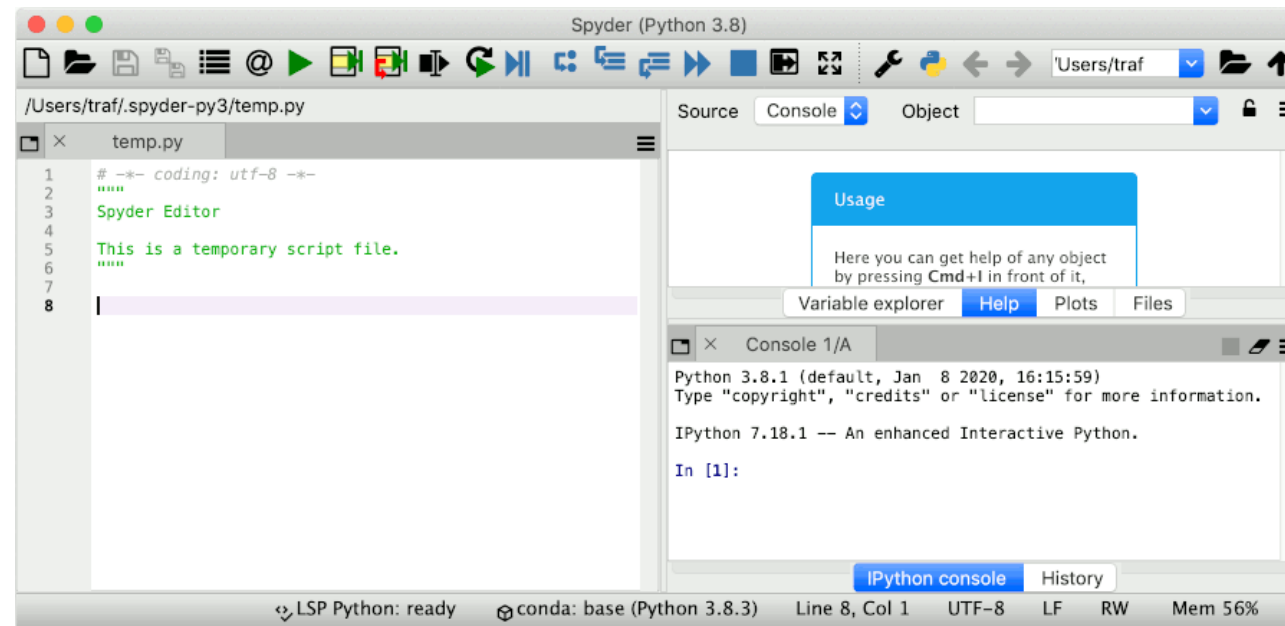
Programmer cette interface "à la main" (sans objets) est assez simple :

- 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 !

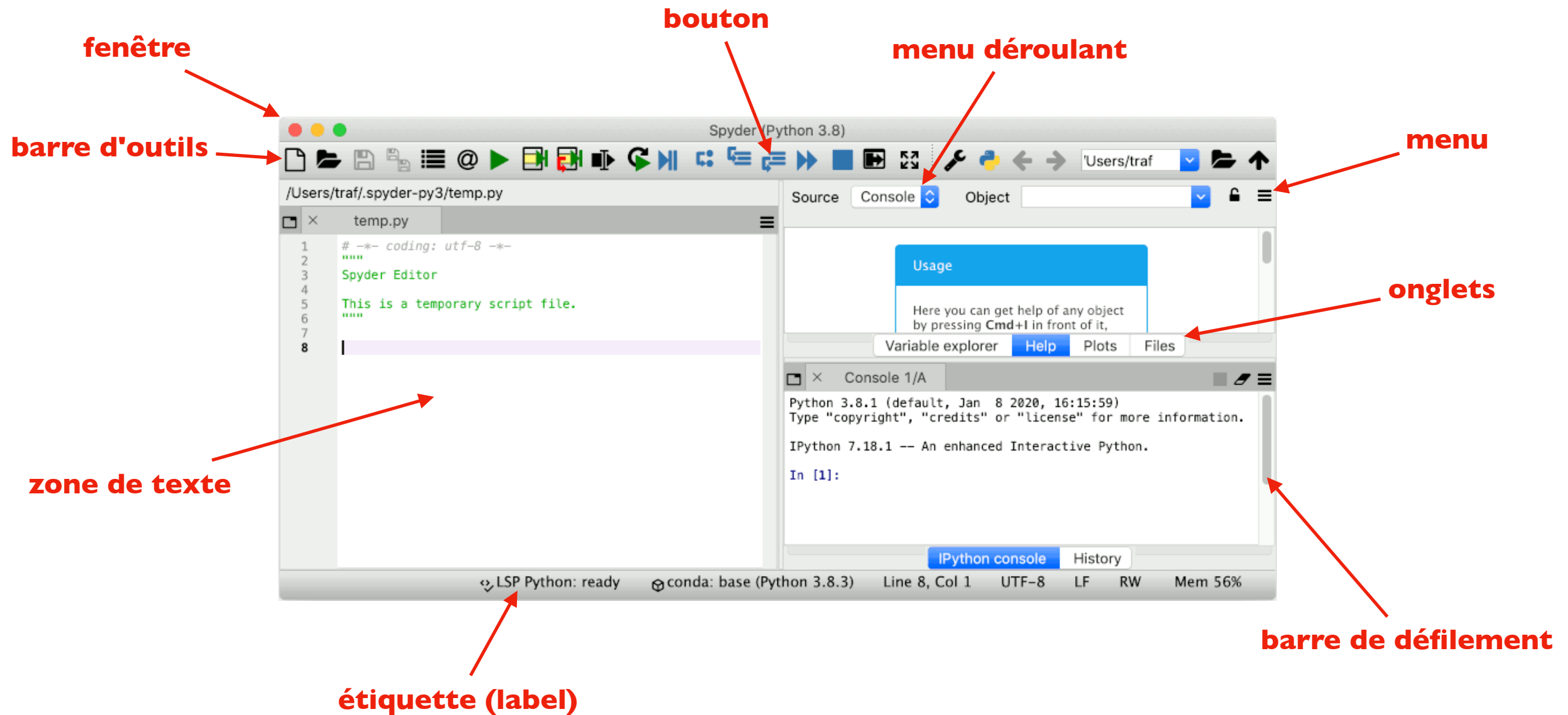
Pourquoi utilise-t-on des objets ?

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 :

Bonjour

Label

OK

Button



choix

Checkbutton

zone de texte

Entry

condiments



mayo
ketchup

OptionMenu

☐ option 1

☐ option 2

Radiobutton

35

Scale

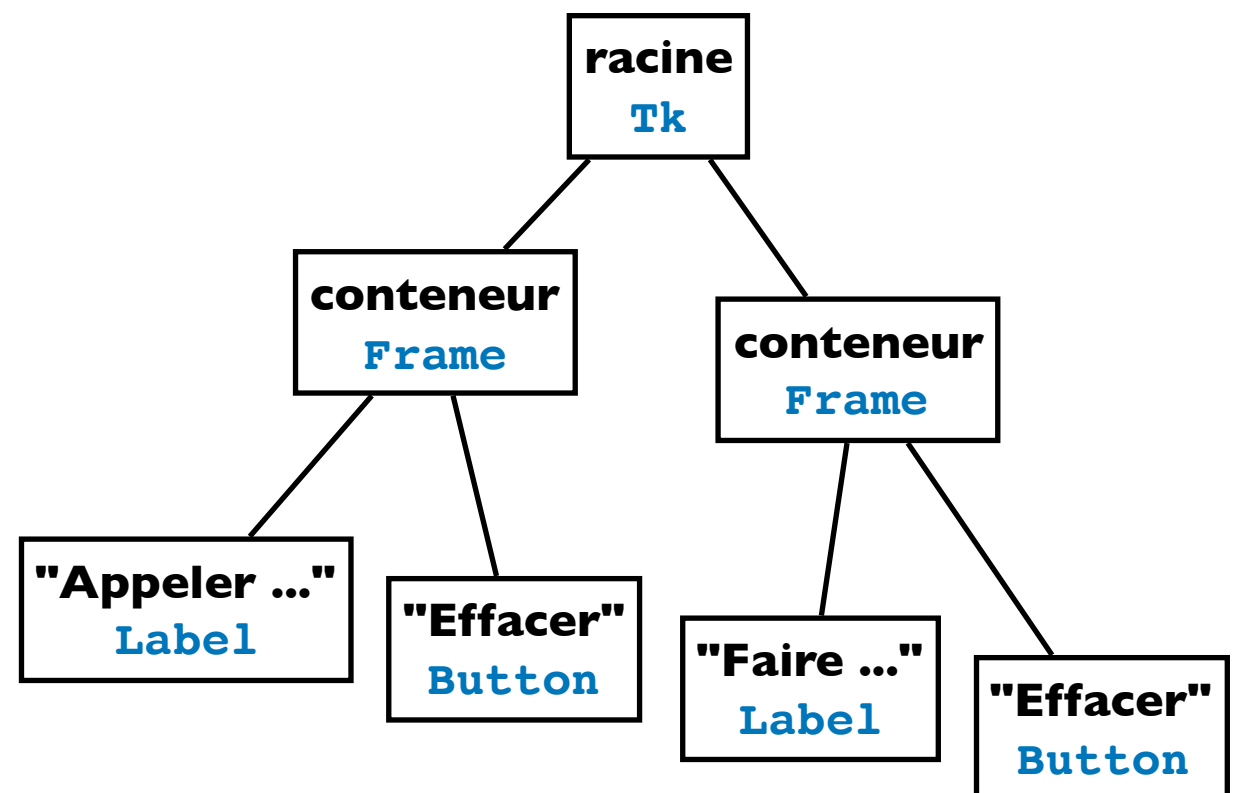


Scrollbar

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.

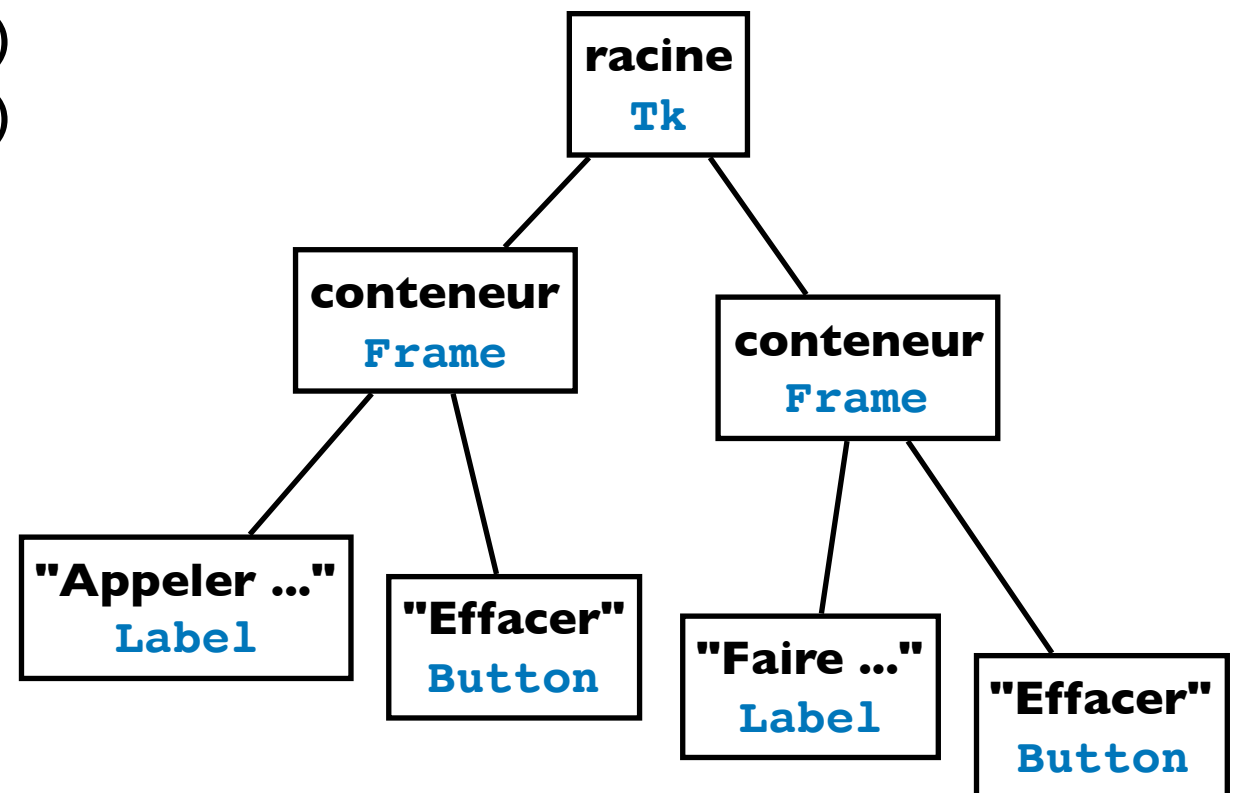


Widgets et arbre de scène

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")
```



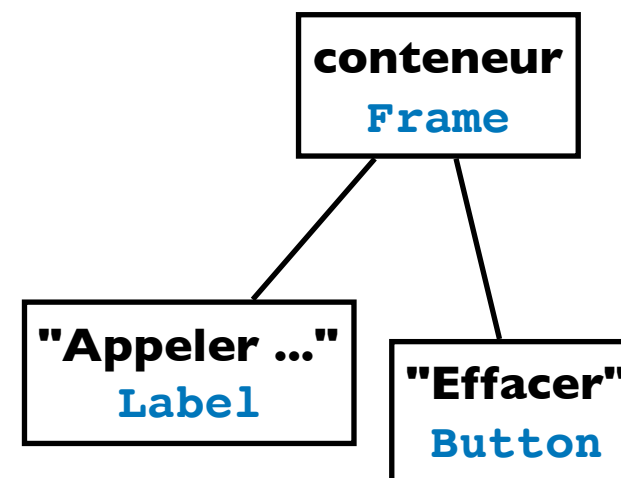
Gestionnaires de positionnement

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 le code 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).

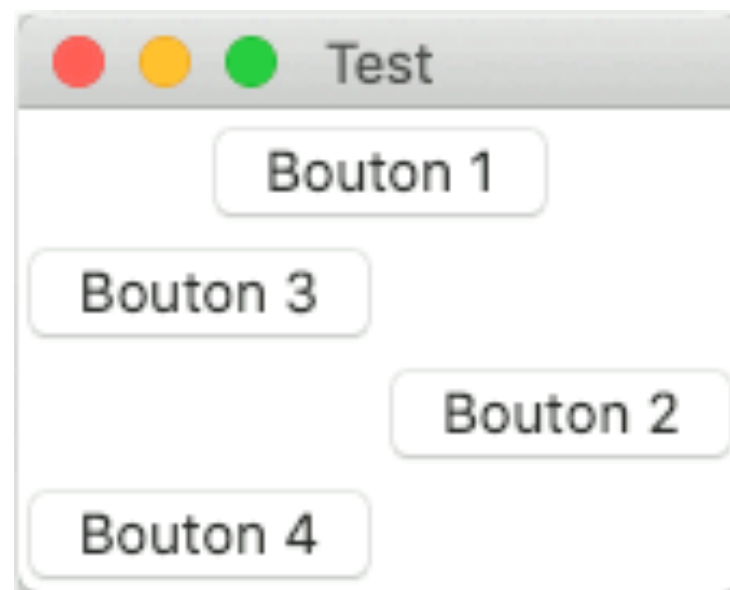
Lorsque les éléments dépassent la taille du conteneur, ils sont ignorés (dans le cas de Tkinter) ou peuvent être placés sur une nouvelle ligne/colonne (dans le cas de HTML).



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 souvent possible de faire occuper à un widget plusieurs cellules.



Gestionnaires de positionnement

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



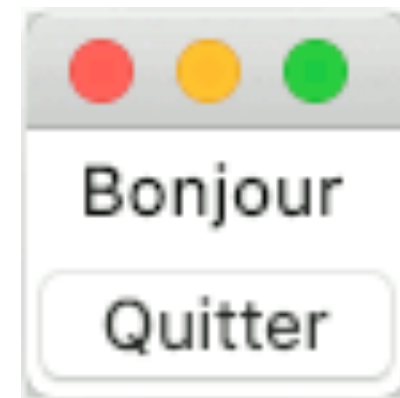
Séparation de la structure et du code

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).

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=GROOVE)
```



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).

Plan du cours

1. Définitions et historique
2. Fonctionnement technique
3. Ingénierie d'une interface graphique
4. Bases de design visuel
 1. De quoi (ne) parle-t-on (pas) ?
 2. Représentation des couleurs
 3. Cercle chromatique
 4. Principes de design C.R.A.P.
 5. 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.

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".

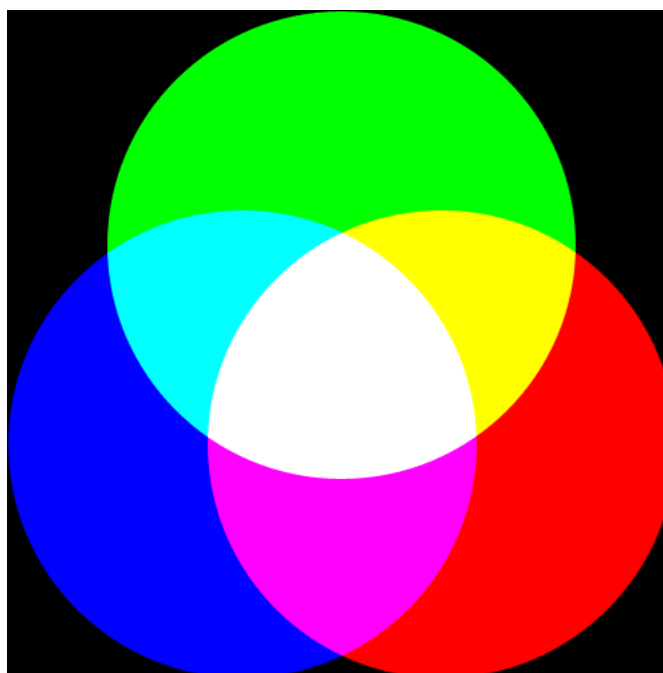
- Nous tâchons de passer en revue des éléments de base simples et utiles dans de nombreux contextes.

Représentation des couleurs

Il existe différentes manières de quantifier les couleurs, en fonction du contexte pour lequel on cherche à les reproduire.

Rouge Vert Bleu (RVB - *RGB*) est basé sur les cônes présents dans l'oeil humain, et adapté aux écrans qui utilisent ces mêmes couleurs fondamentales

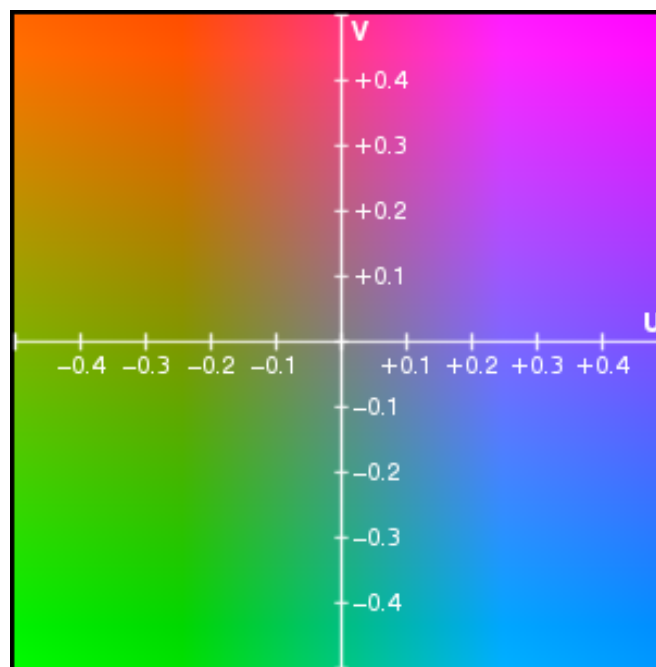
- $R, G, B \in [0, 255]^3$ (écrans dits "8bpp")



Représentation des couleurs

YUV ou YCbCr est basé sur la sensibilité accrue de l'oeil à l'intensité lumineuse (Y) par rapport aux couleurs (U/Cb chrominance bleue, V/Cr chrominance rouge), et largement utilisé en compression vidéo.

- Le vert est le plus fortement corrélé à Y (pour l'oeil) donc il n'a pas de canal distinct mais se déduit à partir de Y, U et V



Représentation des couleurs

Cyan Magenta Jaune Noir (CMJN - *CMYK*) est adapté à l'imprimerie qui utilise ces couleurs fondamentales

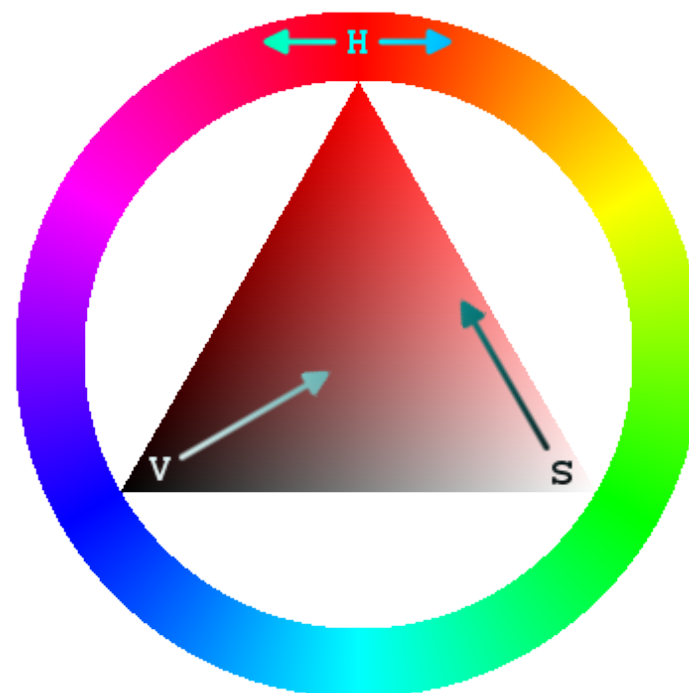
- Contrairement aux autres, les couleurs sont obtenues par soustraction (assombrissement depuis le blanc)



Représentation des couleurs

Teinte Saturation Valeur (TSV - *HSV*) est adapté au choix des couleurs pour les applications graphiques

- $T \in [0, 360]$
- $S \in [0, 1]$ est l'intensité de la couleur
- $V \in [0, 1]$ est la proportion de couleur par rapport au noir



Cercle chromatique

Le *cercle chromatique* est un outil facilitant la création d'assortiments de couleurs (ex. chartes graphiques).



Cercle chromatique

On se base sur les symétries autour du cercle pour opposer et associer les couleurs.



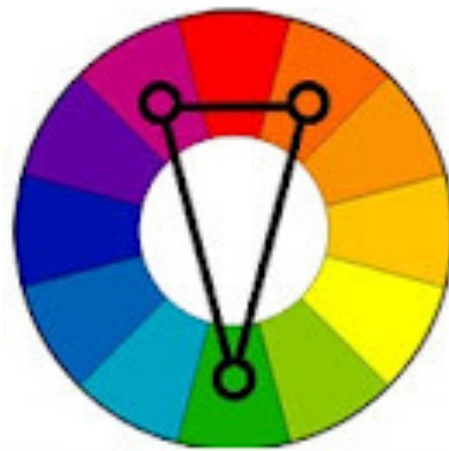
Complémentaires



Analogues



Triade



**Complémentaires
analogues**



**Double
complémentaires**



Carrées

Principes de design C.R.A.P.

Contraste

Accentuer la différence entre éléments d'importances différentes

Répétition

Répéter des principes visuels pour améliorer la lisibilité par régularité

Alignement

Aligner des éléments pour les associer et marquer leur sens de lecture

Proximité

Rapprocher visuellement les éléments apparentés

Principes de design C.R.A.P.

Contraste

Accentuer la différence entre éléments d'importances différentes

Répétition

Répéter des principes visuels pour améliorer la lisibilité par régularité

Alignement

Aligner des éléments pour les associer et marquer leur sens de lecture

Proximité

Rapprocher visuellement les éléments apparentés

Principes de design C.R.A.P.

Contraste

Accentuer la différence entre éléments d'importances différentes

Répétition

Répéter des principes visuels pour améliorer la lisibilité par régularité

Alignement

Aligner des éléments pour les associer et marquer leur sens de lecture

Proximité

Rapprocher visuellement les éléments apparentés

Principes de design C.R.A.P.

Contraste

Accentuer la différence entre éléments d'importances différentes

Répétition

Répéter des principes visuels pour améliorer la lisibilité par régularité

Alignement

Aligner des éléments pour les associer et marquer leur sens de lecture

Proximité

Rapprocher visuellement les éléments apparentés

Principes de design C.R.A.P.

Contraste

Accentuer la différence entre éléments d'importances différentes

Répétition

Répéter des principes visuels pour améliorer la lisibilité par régularité

Alignement

Aligner des éléments pour les associer et marquer leur sens de lecture

Proximité

Rapprocher visuellement les éléments apparentés

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 rouge/bleu OK
- ▶ Alignement "cassé" par l'image de masque
- ▶ Proximité insuffisamment marquée



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 dark navigation bar contains links: About Us, Investor Corner, Interest Rate, Service Charges, Forex Card Rate, Career, Overseas, RRBs, English, and a Search icon. Below this, the Bank of India logo and tagline "Relationship beyond banking" are on the left. A central banner for "DIGITAL APNAYEN" (15.08.2020 to 31.10.2020) features an image of a hand holding a smartphone. To the right of the banner, a grid of service icons includes Mobile Banking, PoS, BHIM UPI, QR Code, Internet Banking, BHIM AADHAR, Debit Card, AEPS, and Credit Card. A horizontal progress indicator is at the bottom of the banner. On the right side, a dark sidebar menu lists: Internet Banking (with a dropdown arrow), Personal, Corporate, Interested in Our Products?, Locate us (with a dropdown arrow), Contact Us (with a dropdown arrow), Communication to BSE/NSE, Safe Banking, COVID Emergency Support Scheme, Covid-19 Emergency Credit, and ATM/POS Online Refund. The BOI SEVA logo is positioned at the bottom of the sidebar.

Merci de votre attention !

Cette présentation est inspirée des présentations :

- 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.