IFT-1004 - Introduction à la programmation

Module 7 : Les interfaces graphiques

Honoré Hounwanou, ing.

Département d'informatique et de génie logiciel
Université Laval

Table des matières

La programmation événementielle

Objectif

Apprendre à utiliser et prototyper rapidement une application graphique en Python. Pour cela, nous allons voir les bases de la programmation événementielle, de l'ergonomie et nous allons explorer les possibilités offertes par Python.

La programmation événementielle

Jusqu'à présent, nous avons traité des programmes qui, séquentiellement :

- lisent ou produisent des entrées ;
- effectuent des traitements ;
- et produisent des sorties.

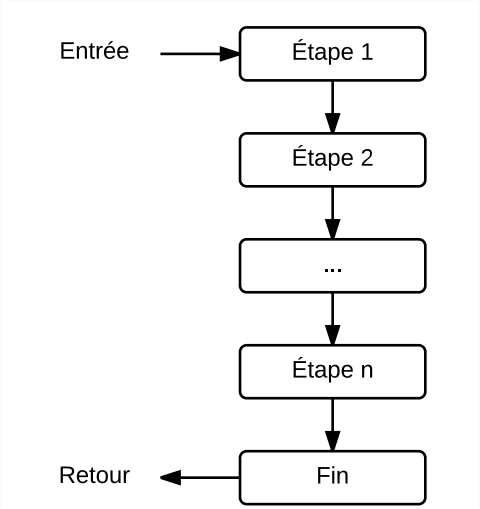
En programmation événementielle, on répond à des événements :

- clics et déplacements de souris ;
- touches de clavier ;
- redimensionnement de fenêtre ;
- etc.

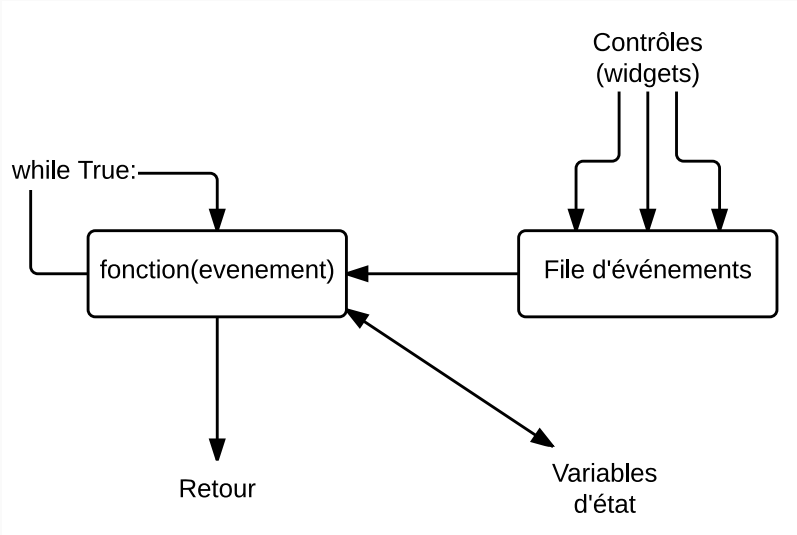
Le déroulement du programme est entièrement déterminé par la séquence des événements.

La programmation événementielle

Programmation séquentielle :

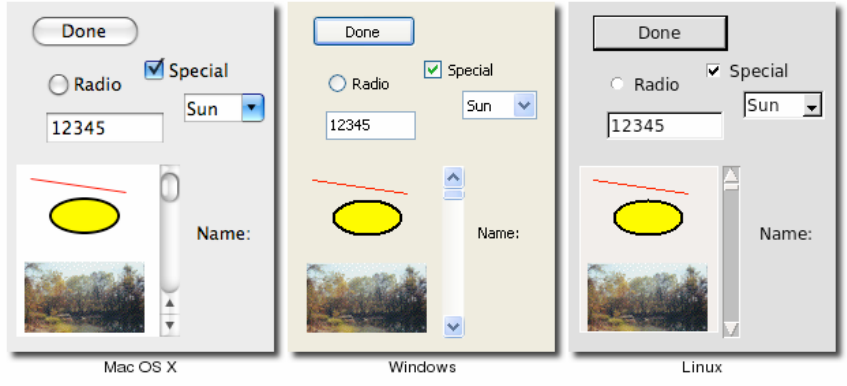


Programmation événementielle :



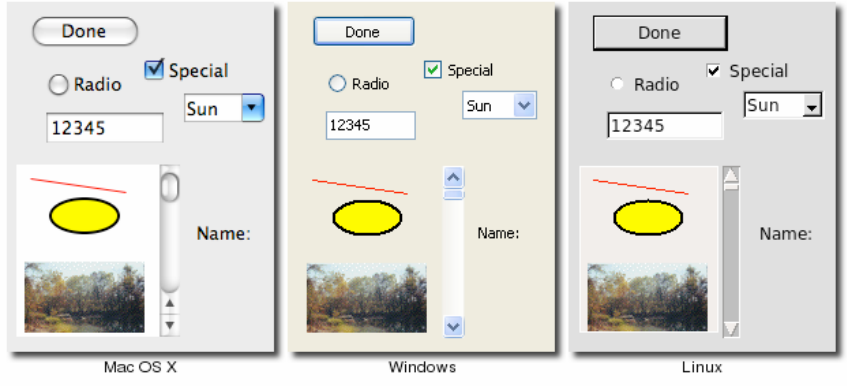
Contrôles - widgets

Un *widget* (*window gadget*) est un élément d'interface permettant une interaction avec l'utilisateur.



Contrôles - widgets

Sur cette image, on a (de haut en bas et de gauche à droite) : `Button`, `CheckButton`, `RadioButton`, `Combobox`, `Entry`, `Canvas`, `Scrollbar`, `Label`.



Les widgets changent d'apparence selon le système d'exploitation et la configuration de l'utilisateur.

Les événements

L'utilisateur interagit avec un widget :

- par exemple, il clique sur un bouton
- le bouton appelle une fonction spécifiée par le programmeur
- cette fonction s'appelle un «*callback*»

Leur traitement se fait généralement de manière automatique :

- le système accumule les événements dans une file d'attente
- les événements sont transmis aux widgets
- chaque widget appelle la fonction *callback* définie par le programmeur

Les interfaces

Une *interface* est définie par une hiérarchie de widgets à l'intérieur d'une fenêtre.

Des mécanismes permettent de positionner les widgets les uns par rapport aux autres

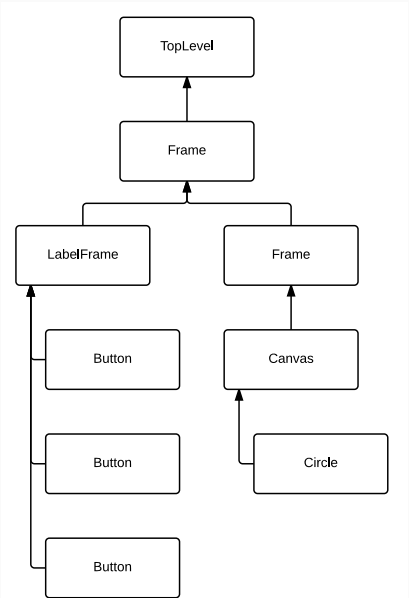
- les widgets ont habituellement une dimension « naturelle »
- ces mécanismes permettent de spécifier l'agencement des widgets dans la fenêtre
- et comment cet agencement doit changer lorsqu'on change la dimension de la fenêtre

Les interfaces

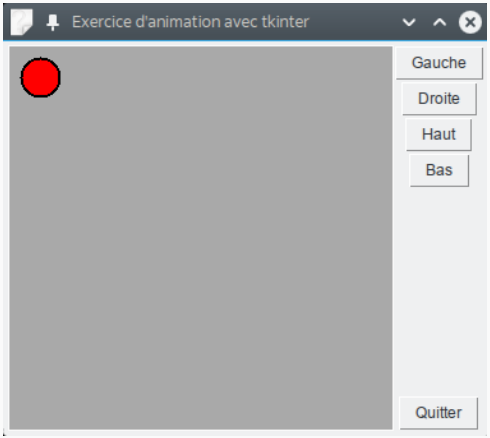
Le `TopLevel` correspond à une fenêtre de l'écran.

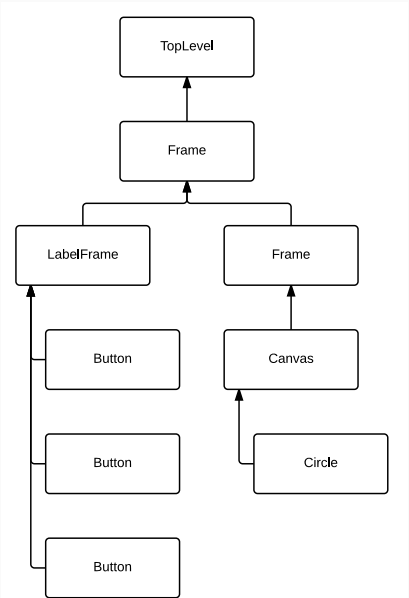
Un `Frame` est une région rectangulaire qui contient plusieurs autres composants.

Les *feuilles* de la hiérarchie sont des contrôles.



Les interfaces





Les interfaces graphiques en Python : `tkinter`

Le module `tkinter`

Ce module est une interface Python sur la librairie *Tcl/Tk*

- `tkinter` signifie «Tk interface »
- Tcl signifie « Tool command language » (se prononce « tickle »)
- Tk veut dire « Toolkit » (boite à outils)

Il permet de faire des interfaces graphiques en Python sur toutes les plateformes

- le programme fonctionne sur **toutes les plateformes**
- mais son apparence dépend habituellement de celle-ci

La documentation de `tkinter`

www.freenetpages.co.uk/hp/alan.gauld/tutgui.htm

usingpython.com/using-tkinter/

- tutoriels assez simples (le deuxième est moins verbeux)
- bon point de départ

www.tkdocs.com/tutorial/index.html

- tutoriel assez complet sur Tcl/Tk 8.5 avec des exemples dans plusieurs langages sont Python/tkinter
- (choisir Python dans le menu à droite)

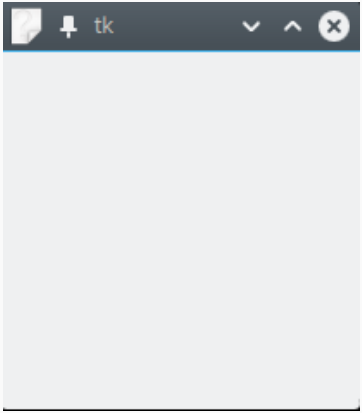
infohost.nmt.edu/tcc/help/pubs/tkinter/

- manuel de référence sur `tkinter`

Hello World !

```
from tkinter import Tk

root = Tk()
root.mainloop()
```



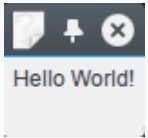
Hello World !

```
from tkinter import Tk, Label

root = Tk()

etiq = Label(root, text="Hello World!")
etiq.grid()

root.mainloop()
```



La création (l'instanciation) d'un objet de type `Tk` provoque la création d'une fenêtre.

Un widget de type `Label` crée une chaîne de caractères. Rien n'est affiché tant que le widget n'est pas *positionné* à l'aide de la méthode `grid`.

text="Hello World" ?

On peut passer des arguments à une fonction en nommant le paramètre auquel nous voulons passer l'argument. Étant donné la fonction suivante :

```
def fonction(param_1, param_2, param_3):
    print("param_1 =", param_1)
    print("param_2 =", param_2)
    print("param_3 =", param_3)
```

Il est possible de l'appeler en nommant un ou plusieurs paramètres. Nous pouvons également changer l'ordre des arguments.

```
>>> fonction(1, param_3=2, param_2=3)
param_1 = 1
param_2 = 3
param_3 = 2
```

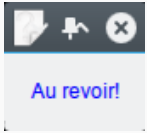
Hello World !

```
from tkinter import Tk, Label

root = Tk()

etiq = Label(root, text="Hello World!")
etiq.grid(padx=10, pady=10)
etiq["text"] = "Au revoir!"
etiq["foreground"] = "blue"

root.mainloop()
```



Hello World !

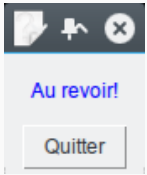
```
from tkinter import Tk, Label, Button

root = Tk()

etiq = Label(root, text="Hello World!")
etiq.grid(padx=10, pady=10)
etiq["text"] = "Au revoir!"
etiq["foreground"] = "blue"

bouton = Button(root, text="Quitter",
                command=root.quit)
bouton.grid()

root.mainloop()
```



La méthode `grid()`

La méthode `grid()` d'un widget permet de positionner celui-ci à l'intérieur du rectangle de son parent.

- par défaut, le widget est centré ;
- chaque widget possède une dimension « naturelle » ;
- la méthode grid peut recevoir des arguments nommés qui auront une influence sur le résultat.

La méthode `grid()`

Nom du paramètre	Utilité
`column`	Le numéro de colonne du widget (à partir de 0)
`columnspan`	Le nombre de colonnes occupées (1 par défaut)
`padx`	Espace additionnel horizontal (en pixels) ajouté autour du widget
`pady`	Espace additionnel vertical (en pixels) ajouté autour du widget
`row`	Le numéro de ligne du widget (à partir de 0)
`rowspan`	Le nombre de lignes occupées (1 par défaut)
`sticky`	La façon de distribuer l'espace blanc à l'intérieur de la cellule

Le paramètre `sticky`

Le fonctionnement de l'attribut `sticky` est le suivant :

- par défaut, le widget est centré ;
- nous pouvons utiliser les constantes `NW`, `N`, `NE`, `W`, `E`, `SW`, `S` ou `SE` pour positionner le widget selon les points cardinaux ;
- on peut aussi utiliser `N+S` pour étirer verticalement le widget ;
- ou `E+W` pour l'étirer horizontalement

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/layout-mgt.html#grid

grid: exemple

```
from tkinter import Tk, Frame, Button

root = Tk()
root.title("grid()")

cadre = Frame(root)
cadre.grid(padx=10, pady=10)

for i in range(3):
    for j in range(3):
        t = "({},{})".format(i, j)
        b = Button(cadre, text=t, padx=1, pady=3)
        b.grid(row=i, column=j, padx=5, pady=7)

root.mainloop()
```

Propriétés des widgets

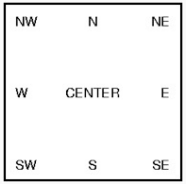
Chaque widget possède un ensemble d'attributs qui déterminent son apparence :

- les attributs sont stockés dans un dictionnaire ;
- ils possèdent une valeur par défaut ;
- on peut définir les valeurs de ces attributs au moment de la création du widget à l'aide d'**arguments nommés** ;
- ou par la suite en utilisant l'opérateur `[]`.

Le `Label`

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/label.html

- `anchor = NW | N | NE | W | E | SW | S | SE`
- `background = couleur`
- `foreground = couleur`
- `justify = LEFT | CENTER | RIGHT`
- `padx = n (pixels)`
- `pady = n (pixels)`
- `text = chaîne`
- `height = n (lignes)`
- `width = n (caractères)`
- etc.



Couleurs :
Encodage RGB en hexadécimal :
- `'#rgb'` (4 bits)
- `'#rrggbb'` (8 bits)
- `'#rrrrggggbbbb'` (12 bits)

Couleurs prédéfinies :
- `'white'`, `'black'`
- `'red'`, `'green'`, `'blue'`
- `'cyan'`, `'yellow'`, `'magenta`

Le Button

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/button.html

```
b = Button(parent, command=fonction, option=valeur, ...)
```

- anchor = NW | N | NE | W | E | SW | S | SE
- background = couleur
- foreground = couleur
- justify = LEFT | CENTER | RIGHT
- padx = n (pixels)
- pady = n (pixels)
- text = chaîne
- height = n (lignes)
- width = n (caractères)

Le Frame

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/web/frame.html

```
w = Frame(parent, option=valeur, ...)
```

- `background = couleur`
- `borderwidth = n (pixels)`
- `padx = n (pixels)`
- `pady = n (pixels)`
- `relief = FLAT | RAISED | SUNKEN | GROOVE | RIDGE`
- etc.

Le Entry

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/web/entry.html Permet à l'utilisateur d'éditer une ligne de texte dans un rectangle.

```
w = Entry(parent, option=valeur, ...)
```

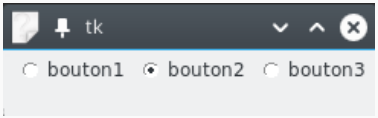
- background = couleur
- borderwidth = n (pixels)
- foreground = couleur
- font = police d'écriture (voir cette page)
- justify = LEFT | RIGHT | CENTER
- relief = FLAT | RAISED | SUNKEN | GROOVE | RIDGE
- show = '*' (pour les mots de passe)
- state = DISABLED | NORMAL | ACTIVE
- width = n (pixels)

Le Radiobutton

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/radiobutton.html

On utilise ce type de widget en groupe.

- À un instant donné, un seul bouton radio peut être actif dans le groupe ;
- Tous les boutons d'un même groupe sont associés à un même objet de type `IntVar` ou `StringVar`.

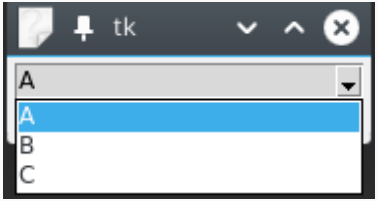


Un exemple d'utilisation du `Radiobutton` est disponible sur le site Web du cours.

Le Combobox

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/web/ttk-Combobox.html

Un Combobox permet à l'utilisateur de choisir une valeur parmi un ensemble de valeurs prédéfinies.



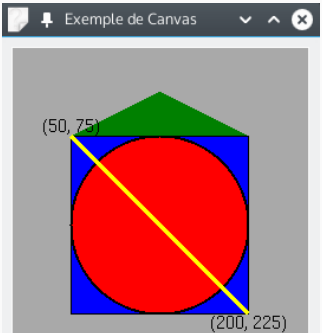
Un exemple d'utilisation du Combobox est disponible sur le site Web du cours.

Le Canvas

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/web/canvas.html

Un **Canvas** est une région rectangulaire dans laquelle on peut dessiner différentes primitives graphiques : arcs de cercle, ellipses, lignes, rectangles et polygones, images et texte.

L'origine du canvas est en haut à gauche. Le point $(0, 0)$ est donc situé en haut à gauche.



Et les autres...

- `LabelFrame` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/labelframe.html)
- `ListBox` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/listbox.html)
- `Menu` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/menu.html)
- `MenuButton` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/menubutton.html)
- `Scale` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/scale.html)
- `Scrollbar` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/scrollbar.html)
- `Spinbox` (http://infohost.nmt.edu/tcc/help/pubs/tkinter/spinbox.html)

Écouter les événements : la méthode `bind()`

http ://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

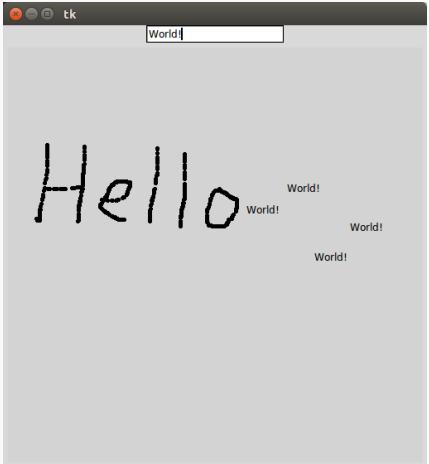
Lorsque l'utilisateur interagit avec les widgets d'une fenêtre, des **événements** sont déclenchés dans `tkinter`. Nous pouvons **lier** ces événements à des fonctions ou méthodes à l'aide de la méthode `bind()`, disponible pour tous les widgets.

Par exemple, nous pourrions lier une touche de clavier spécifique dans une boîte d'entrée `Entry`. Lorsque l'utilisateur aura mis le *focus* sur ce widget et aura pressé sur cette touche, `tkinter` appellera la fonction spécifiée.

Le nom lié à chaque événement est prédéfini, et peut être trouvé à l'adresse ci-haut.

`bind()`: Exemple

Vous trouverez sur le site Web du cours un exemple d'utilisation de la méthode `bind()`, qui lie les événements de clics et de déplacements de la souris au dessin dans un `Canvas`. Un exemple de résultat est le suivant :



Fenêtres de dialogue

Il existe différentes fenêtres de dialogue pré-construites :

- pour signifier les erreurs, avertissements ;
- pour informer l'utilisateur ;
- pour poser une question : `messagebox`
  - ouverte (texte) ;
  - avec réponse par oui ou non.
- pour choisir un nom de fichier où lire/sauvegarder : `filedialog`
- pour choisir une couleur : `colorchooser`

Le module `messagebox`

http://infohost.nmt.edu/tcc/help/pubs/tkinter/dialogs.html#tkMessageBox

Les fonctions disponibles sont `askokcancel`, `askquestion`, `askretrycancel`, `askyesno`, `showerror`, `showinfo` et `showwarning`.

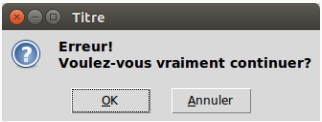
Les options sont :

- `default` : permet de spécifier lequel des boutons sera celui par défaut ; les valeurs possibles sont `CANCEL`, `IGNORE`, `OK`, `NO`, `RETRY` ou `YES`;
- `icon` permet de spécifier l'icône à utiliser. Les valeurs possibles sont `ERROR`, `INFO`, `QUESTION` ou `WARNING`;
- `title` : le titre de la fenêtre ;
- `parent` permet de spécifier la fenêtre au dessus de laquelle la fenêtre de dialogue sera affichée.

Le module `messagebox` : Exemple

```
>>> from tkinter import messagebox
>>> messagebox.showerror("Titre", "Message d'erreur!")
'ok'
```


```
>>> from tkinter import messagebox
>>> messagebox.askokcancel("Titre",
        "Erreur!\nVoulez-vous vraiment continuer?")
True
```



Le module `filedialog`

http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/tkFileDialog.html

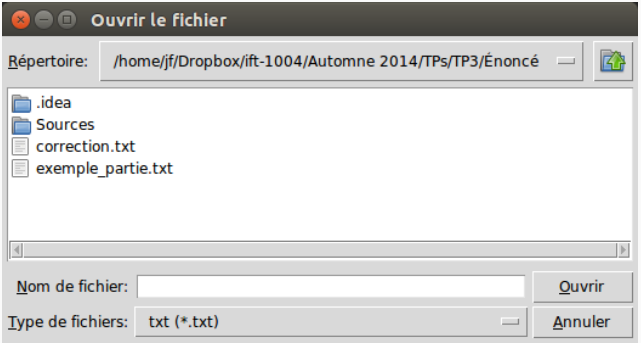
Les fonctions disponibles sont `askopenfilename` et `asksaveasfilename`.

Les options sont :

- `defaultextension` : extension par défaut ajoutée au nom de fichier si l'utilisateur n'entre pas d'extension ;
- `filetypes` : liste des types de fichiers acceptés. Exemple : `[('PNG', '*.png'), ('JPG', '*,jpg')]` ;
- `title` : le titre de la fenêtre ;
- `parent` permet de spécifier la fenêtre au dessus de laquelle la fenêtre de dialogue sera affichée.

Le module `filedialog` : Exemple

```
>>> from tkinter import filedialog
>>> filedialog.askopenfilename(title='Ouvrir le fichier',
        filetypes=[('txt', '*.txt')]))
'exemple_partie.txt'
```



Le module `colorchooser`

http ://infohost.nmt.edu/tcc/help/pubs/tkinter/web/tkColorChooser.html

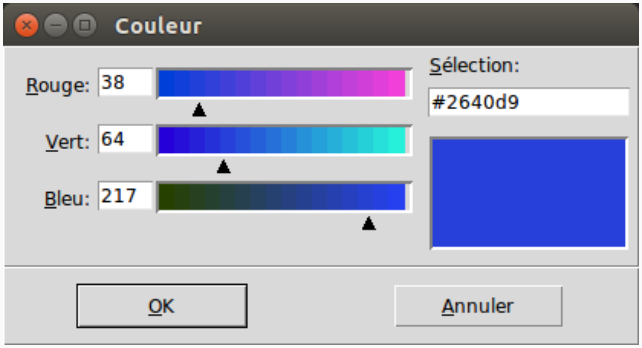
Fonctions disponible : `askcolor`.

Les options sont :

- `title` : le titre de la fenêtre ;
- `parent` permet de spécifier la fenêtre au dessus de laquelle la fenêtre de dialogue sera affichée.

Le module `colorchooser` : Exemple

```
>>> from tkinter import colorchooser
>>> filedialog.askcolor()
((38.1484375, 64.25, 217.84765625), '#2640d9')
```



Le design d’interfaces

Le design d'interfaces

Promouvoir la cohérence :

- homogénéité d'apparence et de fonctionnement ;
- exemple : même formulaire pour l'ajout et la modification.

Offrir des réactions informatives :

- Icônes pertinentes et zone de messages.
- « Popups » ?
  - le moins possible…
  - en cas d'erreurs graves ;
  - pour confirmer une suppression.

Le design d'interfaces

Un formulaire représente un scénario :

- avoir une séquence claire :
  - un début, un milieu et une fin
- les éléments de la séquence doivent être disposés de manière intelligente.

Limiter les choix :

- limiter les options disponibles ;
- permettre à l'utilisateur de choisir parmi des options valides plutôt que de permettre des valeurs invalides.

Le design d'interfaces

Ne pas surcharger une interface :

- une personne ne peut mémoriser qu'environ 7 éléments d'information à la fois ;
- faire plusieurs écrans.

Permettre une annulation aisée des actions :

- exemple : un bouton annuler qui fonctionne même s'il y a des erreurs de validation.

Réduire les manipulations avec la souris :

- Ajouter des raccourcis clavier.

Disposition

- En ordre logique d'utilisation ;
- Espaces vierges et encadrés :
  - regrouper des éléments ;
- Disposition verticale :
  - Du haut vers le bas ;
- Disposition horizontale :
  - De gauche à droite.

Lectures et travaux dirigés

Lectures et travaux dirigés

- Chapitres 8 et 13 de G. Swinnen
- Travaux dirigés : Acquisition et traitement de données par GUI.

Questions ?