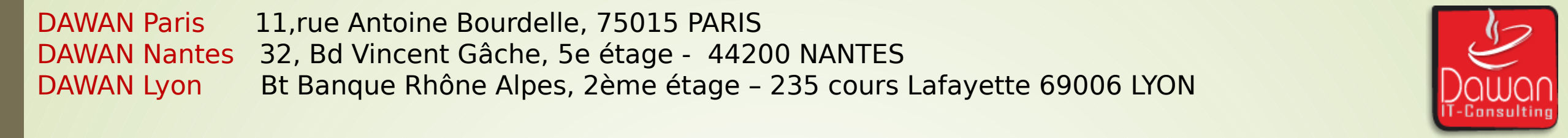
DAWAN Paris 11,rue Antoine Bourdelle, 75015 PARIS
DAWAN Nantes 32, Bd Vincent Gâche, 5e étage - 44200 NANTES
DAWAN Lyon Bt Banque Rhône Alpes, 2ème étage – 235 cours Lafayette 69006 LYON



Formation Python: Interfaces graphiques et API REST

Plus d'info sur http://www.dawan.fr ou 0810.001.917

Formateur: Matthieu LAMAMRA

Python : TKinter

- Présentation

  - Bibliothèque graphique intégrée de base à Python
  - Basée sur **Tk** (librairie c)
  - L'avantage est la **portabilité**
  - **https://docs.python.org/3.6/library/tkinter.html**

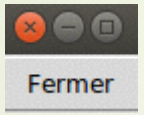
```
from tkinter import *

fenetre = Tk()
label = Label(fenetre, text="Hello World")
label.pack()
fenetre.mainloop()
```

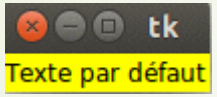
Python : TKinter

- widgets

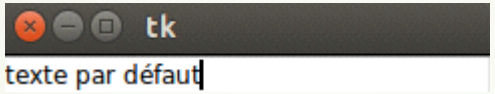
Button



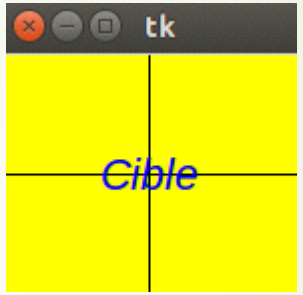
Label



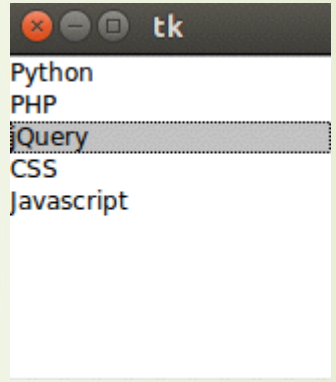
Entry



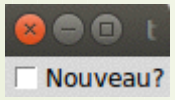
Canvas



Listbox

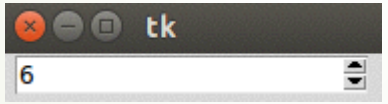


Checkbutton

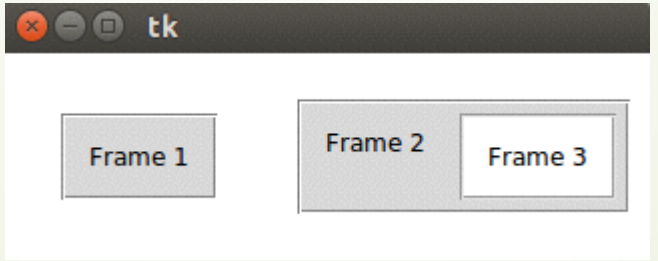


Scale

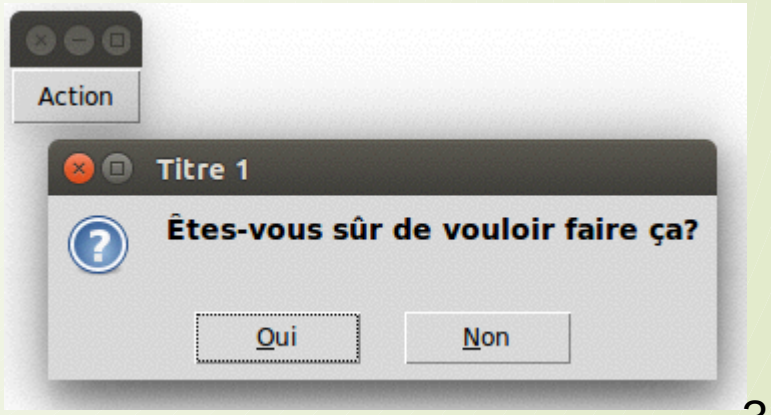
Spinbox



Frame



Messagebox



Python : TKinter

- Implémenter un widget

- Créer l'objet
- **Le premier paramètre est l'élément conteneur** (la fenetre ou autre conteneur)
- Les autres paramètres sont nommés (optionnels ou pas en fonction de l'élément)

text	texte du widget	pady	marge interne en y
foreground(fg)	couleur du texte	width	largeur en taille de police
activeforeground	idem quand selectionné	height	hauteur en taille de police
background(bg)	couleur de fond		
activebackground	idem quand selectionné		
padx	marge interne en x		

Python : TKinter

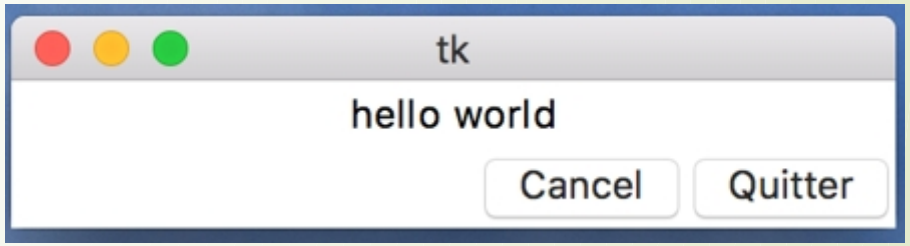
- Placer les composants : la méthode pack()

- Accepte un paramètre **side** : [« **TOP** » (défaut), « **BOTTOM** », « **LEFT** », « **RIGHT** »]
- Divise le conteneur en deux zones et place le widget du coté déclaré par side

```
from tkinter import *

fen = Tk()
fen.minsize(300, 40)
label = Label(fen, text='hello world')
label.pack()

button_quit = Button(fen, text='Quitter')
button_quit.pack(side=RIGHT)
button_cancel = Button(fen, text='Cancel')
button_cancel.pack(side=RIGHT)
fen.mainloop()
```



Python : TKinter

- Placer les composants : la méthode grid()

- Accepte les paramètres **column** et **row**
- **row=0, column=0** correspondent à **en haut à gauche**
- On choisit l'ordre d'affichage par l'ordre d'écriture des boucles
- **Incompatible avec pack() !!!**

```
from tkinter import *

fenetre = Tk()
value = 1
for i in range(3):
    for j in range(3):
        Button(fenetre, text=value).grid(column=i, row=j)
        value += 1

fenetre.mainloop()
```

Python : TKinter

- Interactions : les boutons

➢ l'attribut **command** accepte des methodes internes à Tk ou des fonctions **« custom »**

```
from tkinter import *

fenetre = Tk()

def log_text():
  print("hello world")

Button(fenetre, text='Log', command=log_text).pack()

bouton=Button(fenetre, text="Fermer", command=fenetre.quit)
bouton.pack()

fenetre.mainloop()
```

Python : TKinter

- Interactions : les entrées

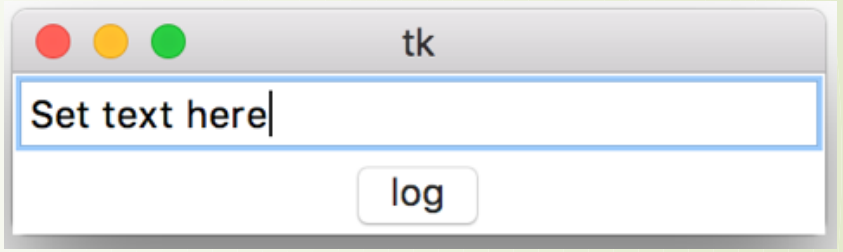
  - On utilise une classe Tk ad hoc pour intéragir avec la valeur saisie

```
from tkinter import *

fenetre = Tk()
entry_value = StringVar()
entry_value.set('Set text here')
Entry(fenetre, textvariable=entry_value, width=30).pack(side=TOP)

def log_value():
  print(entry_value.get())

Button(fenetre, text='log', command=log_value).pack()
fenetre.mainloop()
```



Python : TKinter

- Interactions : les checkboxes

➢ On utilise une classe Tk ad hoc pour intéragir avec la valeur « cochée »

```
from tkinter import *

fenetre = Tk()
checkValue = IntVar()
bouton = Checkbutton(fenetre, text='un label', variable=checkValue,
onvalue=5, offvalue=0)
bouton.pack()

def get_value():
  if checkValue.get():
    print("value is {}".format(checkValue.get()))
  else:
    print('not Checked')

Button(fenetre, text='log', command=get_value).pack()
fenetre.mainloop()
```

Python : TKinter

- Interactions : les listes de sélection

  - On utilise une classe Tk ad hoc pour intéragir avec la liste des options sélectionnables

```
fenetre = Tk()

choices = Variable(fenetre, ('Java', 'Python', 'Swift'))
listbox = Listbox(fenetre, listvariable=choices, selectmode='single')
listbox.insert(END, 'Ruby')
listbox.insert(0, 'Pascal')
listbox.insert(1, 'Scala')
listbox.pack()

for index in listbox.curselection():
  print(choices.get()[index])

fenetre.mainloop()
```

Python : TKinter

- Macro composants

- Pour les programmes conséquents, créer ses propres composants
- Regrouper les widgets et la logique dans des classes héritant de **Frame**
- Implanter la logique et permettre la communication via des méthodes dédiées

Python : TKinter

- Macro composants : Exemple de classe

```
class UserInfo(Frame):
    def __init__(self, master=None, cnf={}, **kw):
        super().__init__(self, master, cnf, **kw)

        Label(self, text='nom').grid(column=0, row=0)
        Label(self, text='prenom').grid(column=0, row=1)

        self.last_name_value = StringVar()
        self.first_name_value = StringVar()

        Entry(self, textvariable=self.last_name_value, width=30)\
        .grid(column=1, row=0)
        Entry(self, textvariable=self.first_name_value, width=30) \
        .grid(column=1, row=1)

    def set_defaut_values(self):
        self.first_name_value.set('John')
        self.last_name_value.set('Doe')
```

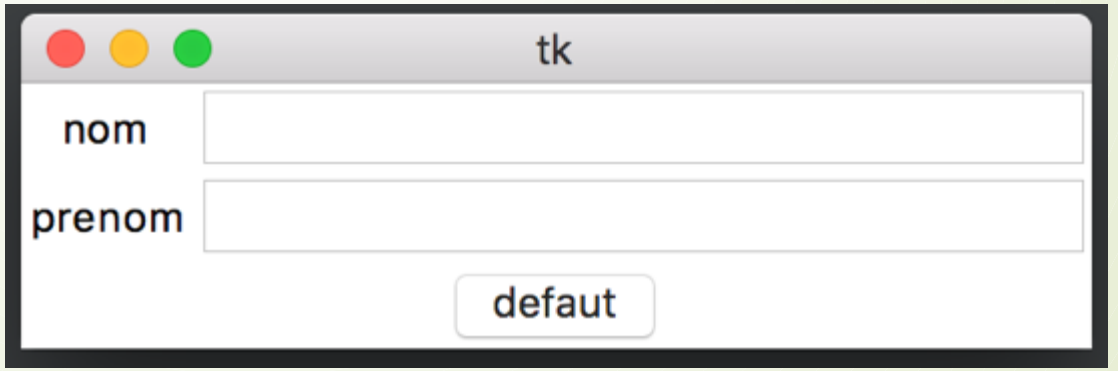
Python : TKinter

- Macro composants : Instanciation

```
from tkinter import *

fenetre = Tk()
fenetre.title('Demo')
user_frame = UserInfo(fenetre)
user_frame.pack()

Button(fenetre, text='defaut', command=user_frame.set_defaut_values).pack()
root.mainloop()
```



Python : Les logs

- Enjeux

- Suivi de l'exécution d'un programme, sans l'interrompre
- Pour un programme en CLI, **print()** est à privilégier
- Le module standard **logging** permet un paramétrage fin de la journalisation
- Il met à disposition plusieurs niveaux d'alerte croissants, et un seuil d'écriture configurable :
  - **DEBUG :** information verbeuse à finalité de diagnostic
  - **INFO :** information de **franchissement d'étape** dans l'exécution
  - **WARNING :** quelquechose de **non bloquant** ne s'est pas déroulé comme prévu
  - **ERROR :** message d'**erreur bloquante**, dans un bloc except par exemple
  - **CRITICAL :** idem pour une erreur **bloquante et préjudiciable** à l'activité

Python : Les logs

- Module logging : les fonctions et la configuration

- 5 fonctions permettent d'émettre des messages reprenant les noms des niveaux d'alerte :
  - **debug**(*msg*), **info**(*msg*), **warning**(*msg*), **error**(*msg*) et **critical**(*msg*)
- **basicConfig**(*path*, *level=logging.WARNING*, […]) configure le fichier de logs et le seuil d'écriture

```
import logging
logging.basicConfig(filename='program.log',level=logging.INFO)
logging.debug("This message won't go to the log file")
logging.info('This one will')
logging.warning('So does that one')
```

Python : Les logs

- logging et modules importés
  - Si basicConfig s'exécute en début de programme, les modules importés peuvent utiliser logging

```python
import logging
import mylib

def main():
    logging.basicConfig(filename='program.log', level=logging.INFO)
    logging.info('Started')
    mylib.do_something()
    logging.info('Finished')

if __name__ == '__main__':
    main()
```

```python
import logging

def do_something():
    logging.info('Doing something')
```

Python : Les logs

- Formatage des messages

  - Par l'injection de variables

```
logging.warning('%s of the %s', 'Beware', 'Dawg!')
```

  - Par l'injection de paramètres depuis la configuration : **%(levelname)s, %(asctime)s, ...**

```
logging.basicConfig(format='%(asctime)s %(message)s', datefmt='%Y/%m/%d %I:%M')

2019/08/08 08:48 Beware of the Dawg!
```

Python : requêtes HTTP

- Principe

- **HTTP: Hyper Text Transfer Protocol – protocole historique de communication sur le web**

- Basé sur des requêtes sur des **URLS**, avec échange d'entêtes et téléchargement d'une réponse

- Des gammes de code réponse dont les plus connus sont :

  - **200 :** la requête à été traitée
  - **301 :** idem mais l'url a été redirigée
  - **403 :** accès refusé
  - **404 :** page non trouvée
  - **500 :** erreur serveur

- 4 modes principaux d'acquisition, dit **« verbes HTTP »** assurant une fonction sémantique

  - **GET :** Lecture - utilisation de paramètres « querystring »?param1=[]&param2=[] dans l'URL
  - **POST :** Création - ajout de données (formulaires) dans la requête pour créer une ressource
  - **PUT :** Modification - idem mais pour mettre à jour une ressource
  - **DELETE :** suppression d'une ressource

Python : requêtes HTTP

- Le module requests

  - Disponible via **pip install requests**

  - Requêtes basiques :

```
import requests

r = requests.get('https://api.github.com/events')
r = requests.post('https://httpbin.org/post', data = {'key':'value'})
r = requests.put('https://httpbin.org/put', data = {'key':'value'})
r = requests.delete('https://httpbin.org/delete')
```

  - Requête GET avec paramètres querystring :

```
payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.get('https://httpbin.org/get', params=payload)
```

Python : requêtes HTTP

- Le module requests : la réponse

➢ Tester le code réponse

```
r.status_code == requests.codes.ok  #200
```

➢ Corps de réponse et paramètres :

```
r.text          # réponse en mode texte
r.content       # réponse en mode bytes ( pour les images par exemple)
r.json()        # réponse json
r.encoding      # encodage de la réponse
```

Python : requêtes HTTP

- Le module requests : les entêtes

➢ Entêtes de la requête

```
headers = {'user-agent': 'my-app/0.0.1'}
r = requests.get(url, headers=headers)
```

➢ Entêtes de la réponse :

```
r.headers
>>> { 'content-encoding': 'gzip', 'transfer-encoding': 'chunked',
    'connection': 'close', 'server': 'nginx/1.0.4', 'x-runtime': '148ms',
    'etag': '"e1ca502697e5c9317743dc078f67693f"',
    'content-type': 'application/json' }
```

Python : requêtes HTTP

- Le module requests : Upload d'un fichier

➢ Requête POST multipart :

```
files = {'file': open('example.xls', 'rb')}
r = requests.post(url, files=files)
```