The Kyoto College of Graduate Studies for Informatics

kcg.edu

コンピュータプログラミング概論

第12回　eラーニング資料

安　平勲

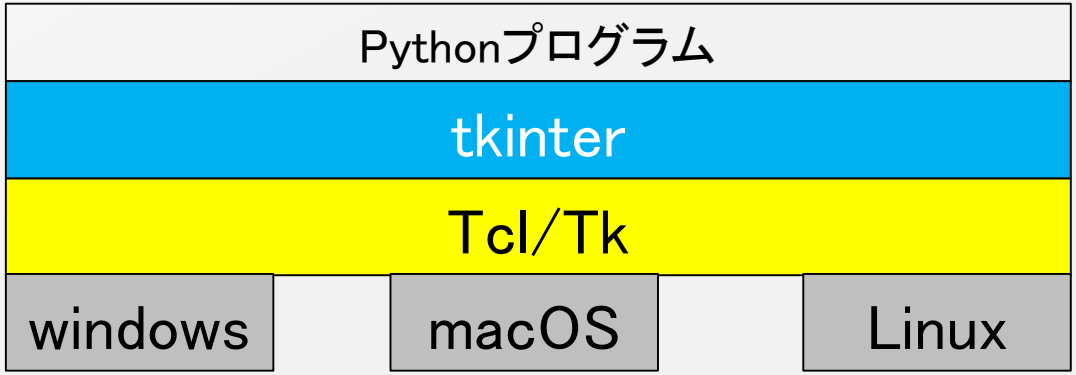
h_an@kcg.ac.jp

GUI（Graphical User Interface）

- ユーザ操作を伴うアプリケーション開発ではGUIは不可欠
- 一般にGUIアプリケーションを作るには，ボタン・テキスト入力などの基本的なGUI構成要素（**ウイジェット**）のライブラリが必要になる
- PythonにはGUIのライブラリが豊富にあるが，定番の**tkinter**を紹介する　公式ドキュメントはここ（詳しくないが・・）

tkinterとは

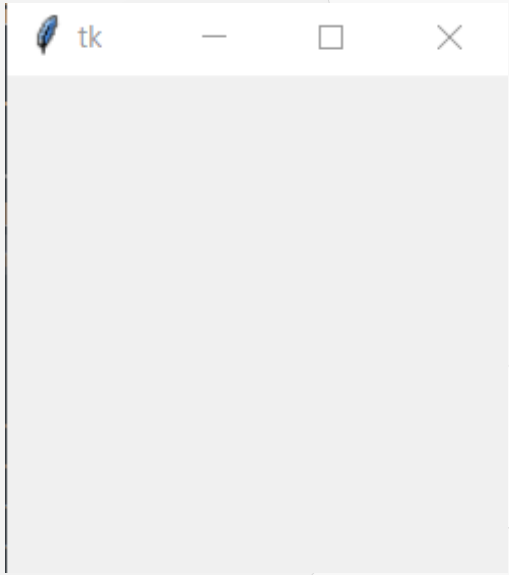
- tkinterはTcl/Tkとのインタフェースを提供するPythonの標準モジュール
- Tcl/Tkは異なるOSのGUI仕様の差異を吸収する共通フレームワーク（Tclが言語でTkがツールキット　※Python専用ではない）



- turtleもtkinterを使ってGUIを実現している

tkinterでまず**窓**を出す

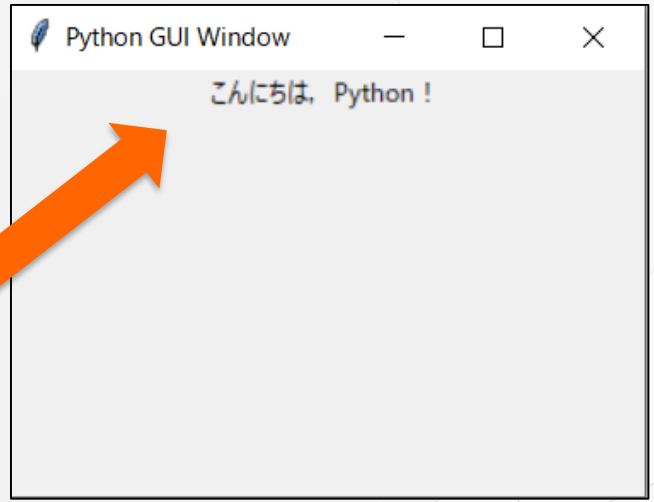
```
import tkinter as tk
root = tk.Tk()
    #ウイジェットの並びを
    #この間に記述する
root.mainloop()
```



- tkinterモジュールの**Tkクラス**で，**ルートウインドウ**（最上位の窓）**オブジェクト**を作る　　※オブジェクト名をrootに設定
- *オブジェクト名*.**mainloop**メソッドで入力イベントの待ち受けに入る

ラベル・ウイジェットを窓に追加する

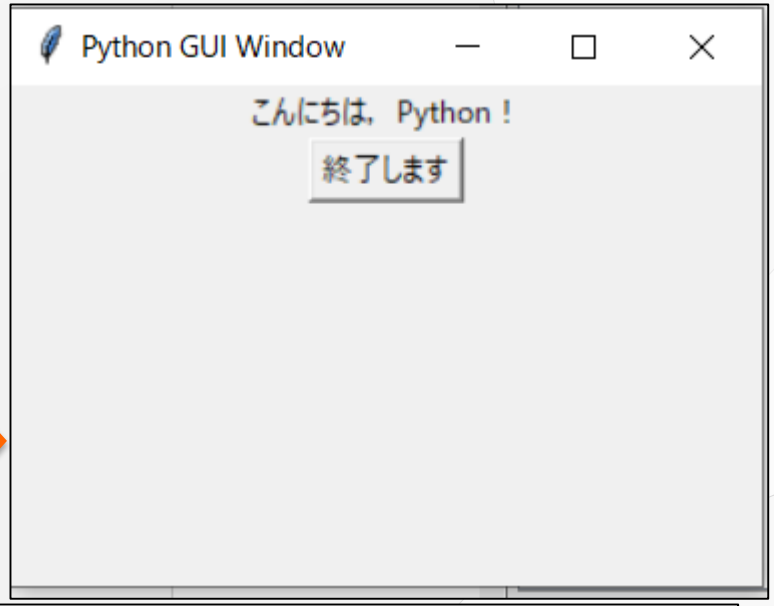
```
import tkinter as tk

#窓を開く
root = tk.Tk()

#窓にタイトル。そしてサイズ調整
root.title("Python GUI Window")
root.geometry("300x200")

#ラベルを追加
tk.Label(root, text="こんにちは，Python！").pack()
```



- tkinterモジュールの**Labelクラス**から**ラベルオブジェクト**を作る
  ※第１引数にはルートオブジェクト（root）を指定
  ※第２引数にはラベルに表示したい文字列を指定
- 作成したラベルオブジェクトを.pack()メソッドでルートウインドウ（root）内に配置する

ボタン・ウイジェットを追加する

```
#窓にタイトル。そしてサイズ調整
root.title("Python GUI Window")
root.geometry("300x200")

#ラベルを追加
tk.Label(root, text="こんにちは，Python！").pack()

#ボタンを追加
tk.Button(root,text="終了します",
          command=root.destroy).pack()
```



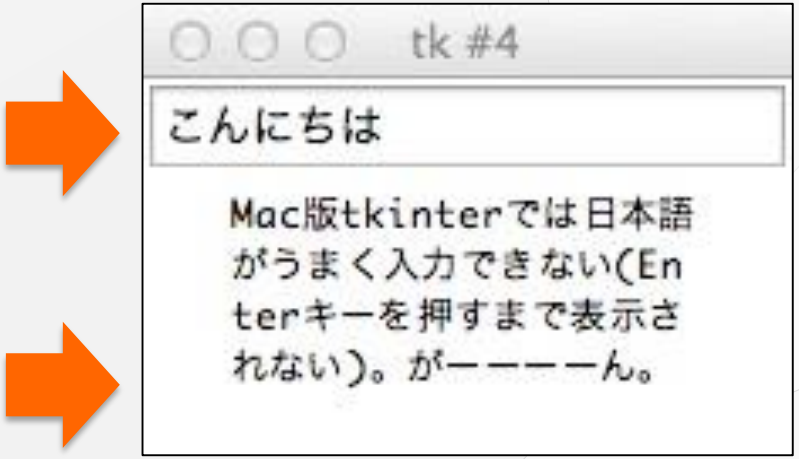
- Buttonクラスからボタンオブジェクトを作り，packで配置する
  - 『command=』はボタンが押された時（イベント）の処理を記述
  - 『.destroy』は窓rootを閉じるメソッド名　　※（）は書かない

テキスト入力欄・ウイジェットを追加する

- テキスト入力欄
  - １行のみ：

    tk.**Entry**(root, width=20).pack()
  - 複数行：

    tk.**Text**(root, width=20,
    height=5).pack()



ウイジェットの配置：Geometry Manager

- *ウイジェットオブジェクト*.**pack**(*引数*)メソッドのキーワード引数により配置場所をコントロールできる
  - **side** = "left" | "right" | "top" | "bottom"
    - ルートウインドウのどの辺に寄せて並べるか
  - **anchor** = “n” | “e” | “s” | “w” # 北|西|南|東
    - **side**と直交する方向でどちらに寄せるか

- packの代わりに.**grid()**メソッドでは格子状に配置できる
  - **grid**(row = 0, column = 1)

↑１行目・２列目の格子に配置する

イベント処理

- Buttonウイジェットでは，ユーザのボタンclick（=イベント）に対する処理をcommand引数で追加する

  tk.Button(root, **command** = *関数名*)

例題では”command=root.destroy”

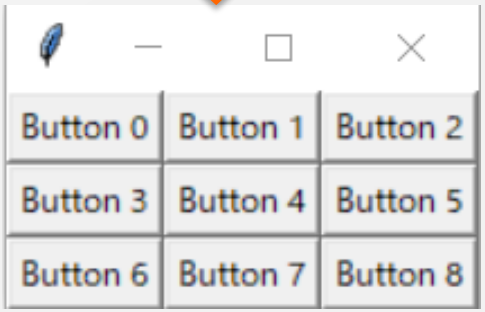
※引数無しの関数しか呼べない！　⇒（）が書けない

☞ 引数を渡したい時は，lambda式（無名関数）を使う

command = **lambda** : *関数名*(*値1*, *値2*, ...)

イベント処理とgrid()メソッド

```
for x in range(9):
    #    command引数にlambda式を使い，引数のあるprint関数を書く
    button = tk.Button(root, text = f"Button {x}",
                       command = lambda : print("Buttonが押された"))
    #    gridメソッドでボタンを格子状に配置
    button.grid(row = x // 3, column = x % 3 )
```



各ボタンを押すと
”Buttonが押された“
をコンソールに表示

コンソールからウインドウへ（出力１）

- print( )　=>
  可変の文字列の**Label**ウイジェットを窓に表示

  # **StringVar**クラスで可変文字列オブジェクトを作成
  buff = tk.StringVar()

  #*textvariable*引数に**buff**を設定した**Label**ウイジェットを作成・配置
  tk.Label(root, **textvariable** = **buff**).pack()

  #可変文字列オブジェクト**buff**に**set()**メソッドで可変の文字列を設定
  buff.set(*“可変文字列”***)**

コンソールからウインドウへ（出力その２）

- print()　=> 窓にLabelオブジェクトの文字列を表示
- ***Label****オブジェクト*.**config**(text=“文字列”)メソッドで，表示中の文字列を変更できる

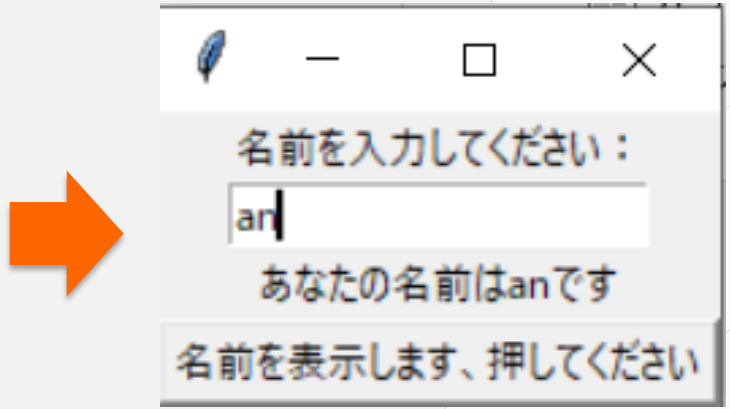
#**Label**オブジェクトだけを作成
*Label**オブジェクト名*＝**tk.Label**(root)
configメソッドでLabelオブジェクトの可変の文字列を変更
*Label**オブジェクト名*.**config**(text=f“可変文字列”)
#**Label**オブジェクトを配置
*Label**オブジェクト名*.**pack**()

コンソールからウインドウへ（入力）

- **input() =>**
  Entry（テキスト入力欄）ウイジェットへの入力値を **.get()** メソッドで取得

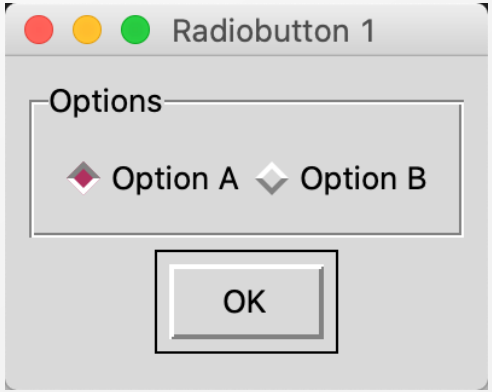
コンソールからウインドウへ（入力）

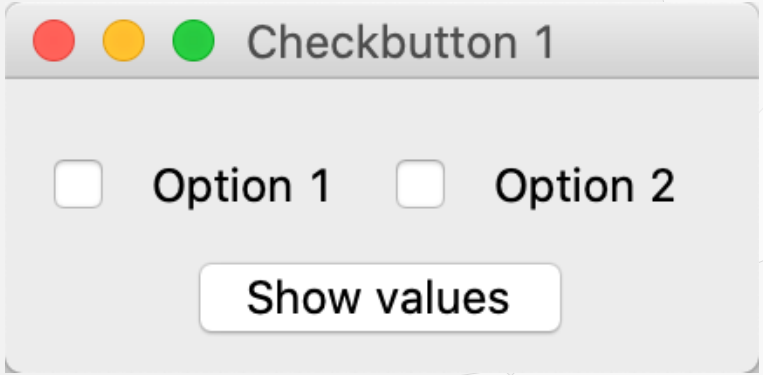
```
tk.Label(root, text = "名前を入力してください:").pack()
#入力欄を用意
entrywords = tk.Entry(root, width=20)
entrywords.pack()

#可変の文字列オブジェクトを用意
buff = tk.StringVar()
#表示Labelを作成・配置。可変の文字列オブジェクトを設定
tk.Label(root, textvariable = buff).pack()

def callname():
    #入力textを取得
    your_name=entrywords.get()
    #可変の文字列オブジェクトに入力textを設定
    buff.set(f"あなたの名前は{your_name}です")

tk.Button(root,text="名前を表示します，押してください",
          command=callname).pack()
```



その他のウイジェット

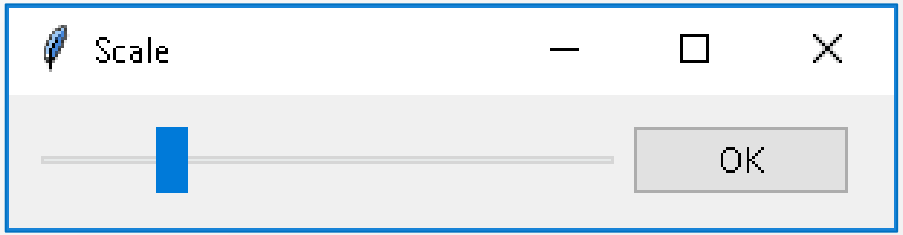
- ボタン系
  - RadioButton, CheckButton

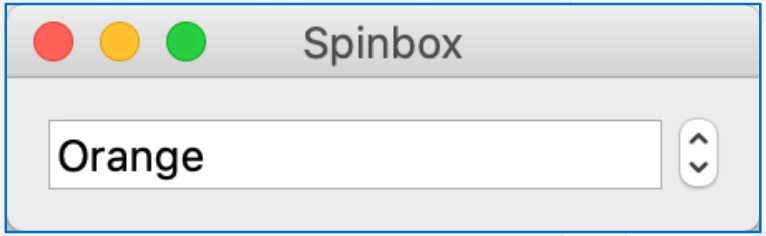




その他のウイジェット

- 値の選択系
  - Scale, Spinbox, Menu





- 描画系
  - Canvas, PhotoImage など

イベント処理（その２）

- Button以外のウイジェットでイベント処理を実現するには，対象ウイジェットに**bind**メソッドでイベント名と呼び出される関数の組を追加する
- Labelウイジェットによるイベント処理の例（次頁）

```
lb = tk.Label( root, text="押してね" )
lb.bind( "<ButtonPress-1>", 関数名 or lambda式 )
```

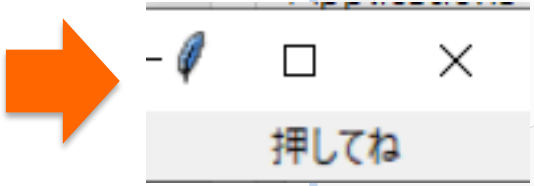
「マウスの左ボタンが押された」イベント名

＊：イベント名のリスト（英語）：
http://effbot.org/tkinterbook/tkinter-events-and-bindings.html

- 関数には，eventオブジェクトを受け取る仮引数が１つ必要

イベント処理（その２）

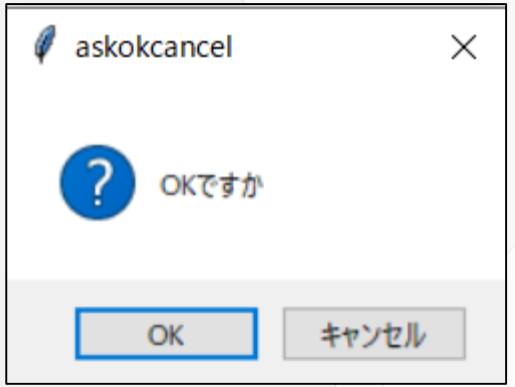
```
def b1Press(event):   #仮引数（イベント）を必ず指定
    #マウスをクリックした時に実行する処理
    lb.config(text = "Button1 press")    #configメソッドでLabelのtextを変更

def b1Release(event): ):   #仮引数（イベント）を必ず指定
    #マウスボタン1 が離された時に実行する処理
    lb.config(text = “Button1 release”)    ")    #configメソッドでLabelのtextを変更

lb = tk.Label(root, text = "押してね")

# Label ウイジェットに二つのイベント処理を設定
lb.bind(“<ButtonPress-1>”, b1Press) #マウスをクリックすると関数を呼ぶ
lb.bind(“<ButtonRelease-1>”, b1Release) ) #クリックをやめると関数を呼ぶ
lb.pack()
```



ダイアログ

- メッセージ表示系ダイアログ
  - tkinter.messagebox モジュールをimportして使う

```
import tkinter.messagebox as mb
answer = mb.askokcancel("タイトル", "メッセージ")
# OK・CANCELが戻り値
```



  - その他のメッセージボックス

```
# イエス・ノーが戻り値
answer = mb.askyesno ("タイトル", "メッセージ")

# インフォメーション
mb.showinfo ("タイトル", "メッセージ")
```

ダイアログ

- ファイル選択系ダイアログ
  - tkinter.filedialogモジュールをimportして使う

```
import tkinter.filedialog as fd
filename = fd.askopenfilename()
```

✓ GUIでファイルを読込む
✓ 開きたいファイルや保存したいファイルを指定できる**オープン・ダイアログ**・ウインドウが使える