Pascal ORTIZ

Écart de jours

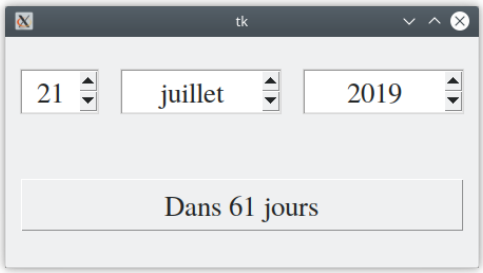
Version du 31 octobre 2019

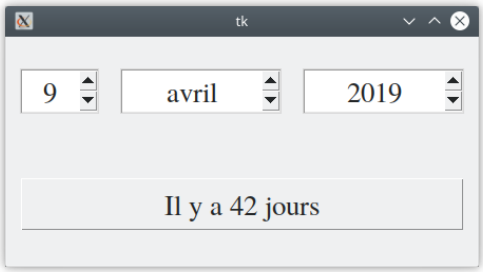
Table des matières

Présentation de l'activité 2
Placement des widgets 3
Remplissage des spinbox 4
La gestion du temps en Python 6
Initialisation des spinbox 8
Implémentation de l'écart de dates 12

Présentation de l'activité

Nous allons écrire une application graphique qui reçoit une date et affiche dynamiquement le nombre de jours qui la séparent de la date d'aujourd'hui :





La date est choisie avec des spinbox. *Dynamiquement* signifie ici que le nombre de jours s'obtient **dès qu**'on modifie un élément de la date parmi les trois possibles. Une application moins dynamique aurait fourni le nombre de jours en cliquant séparément sur un bouton.

Le nombre de jours s'affiche sur le bouton en bas. Ce même bouton sert à réinitiliser la date choisie à la date du jour (*aujourd'hui*). L'affichage de l'écart est adapté au cas où on est le même jour ou un des deux jours qui suivent ou qui précèdent. Dans les autres cas, l'affichage précise si la date est antérieure (par exemple *Il y 42 jours*) ou postérieure (par exemple, *Dans 42 jours*).

Ce type d'application se rencontre habituellement dans des pages Web, sous des formes variées :

— avec des entrées et un calendrier incorporé, sur le site timeanddate

— avec des entrées et des combobox sur le site ephemeride ou encore sur le site calculator.

Comme dans beaucoup d'activités d'interface graphique, il y a trois parties à réaliser :

— l'interface graphique : création et placement des widgets avec les options appropriées

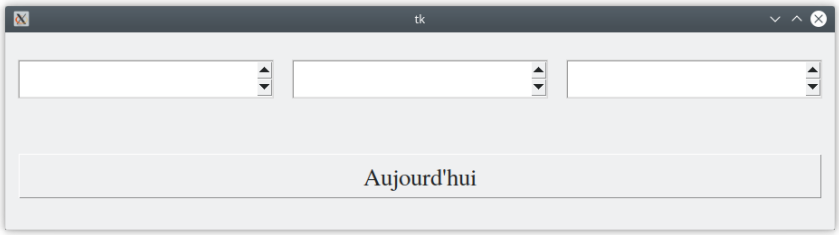
— la partie logique/calcul/modèle de l'application

— la connexion de la partie calcul et de la partie graphique.

Enfin, dans ce tutoriel, par commodité, je dirai *une spinbox* à l'anglaise (on devrait dire *sélection rotative*) même au pluriel.

Placement des widgets

On va créer et placer les widgets mais il n'auront aucun contenu :



Il y a 4 widgets à créer :

— 3 spinbox

— 1 bouton.

Je vais les organiser dans la fenêtre avec la méthode `grid` en 2 lignes et 3 colonnes. La première ligne contient les trois spinbox (une par colonne) et la 2e ligne contient le bouton, étalé sur 3 colonnes.

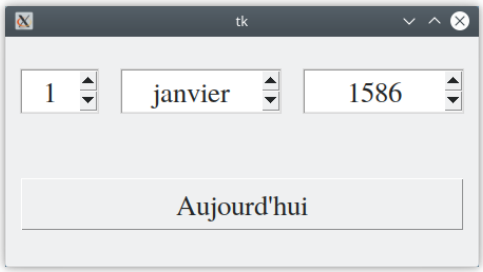
Rappelons qu'une spinbox est un widget mixte constitué d'une entrée et deux curseurs fléchés *haut* et *bas*. Plutôt que de créer 3 spinbox séparément, je vais les créer dans une boucle `for` dans laquelle on donnera toutes les options communes. Voici le code :

```
from tkinter import *

root = Tk()

PADX = 10
PADY = 30
FONT = "Times 20"

spins = []
for j in range(3):
    spin = Spinbox(
        root,
        justify=CENTER,
        state="readonly",
        readonlybackground="white",
        font=FONT,
        textvariable=StringVar())
    spin.grid(row=0, column=j, padx=PADX, pady=PADY)

btn = Button(root, text="Aujourd'hui", font=FONT)
btn.grid(
    row=1, column=0, columnspan=3, sticky=E + W, padx=PADX, pady=PADY)

root.mainloop()
```

— Ligne 10 : la boucle de création et placement des trois spinbox.
— Ligne 11 : création de chaque spinbox. Chaque spinbox sera placée dans une liste, cf. ligne 9, à usage ultérieur. La spinbox des jours sera `spins[0]`, celle des mois sera `spins[1]` et celle des annnées `spins[2]`.
— Ligne 15 : une spinbox contient une entrée, par défaut modifiable en écriture. Mais vu qu'on peut choisir avec les flèches la valeur de l'entrée, l'intérêt du caractère modifibale ne me paraît pas établi. Sans compter qu'il faut gérer les erreurs de saisies de l'utilisateur. Donc je préfère bloquer en écriture l'entrée avec l'option `readonly`.
— Ligne 17 : le passage en statut `readonly` affecte l'apparence du fond de l'entrée. J'en ai donc changé la couleur.
— Ligne 20 : création du bouton.
— ligne 18 : l'appel à la méthode `grid` de placement de la spinbo courante. `row=0` désigne le 1re ligne. Chaque widget est placé dans sa colonne dans la grille
— Ligne 22 : il y a trois colonnes. L'option `columnspan=3` coinjointement avec l'option `sticky=E + W` étale le bouton sur trois colonnes. Cette largeur assez importante sera utile pour l'affichage du texte à indiquer ultérieurement.
— lignes 5-6 : on place un padding fixe autour de chaque widget (lignes 18 et 22) pour aérer l'interface.

Remplissage des spinbox

On a placé les spinbox et on a choisi les options communes. On va maintenant préciser les options spécifiques. On obtiendra l'interface suivante :



Les spinbox contiennent

— les numéros de jours,
— les noms de mois
— les années.

Il faut donc les placer.

On va devoir compléter les options déjà écrites. Il faut donc savoir comment modifier les options d'un widget. J'utiliserai pour cela la méthode `configure` des widgets.

Pour les noms de mois, on va d'abord écrire une liste des mois :

```
MOIS = [
    "janvier", "février", "mars", "avril", "mai", "juin", "juillet",
    "août", "septembre", "octobre", "novembre", "décembre"
]
```

Cette liste sera ensuite donnée à l'option `values` de la spinbox. Cela veut dire que les noms de mois vont s'afficher et qu'on pourra les faire défiler avec les curseurs.

D'autre part, on va élargir pour mettre des espaces autour de chaque nom, ce qui se fera avec l'option `width`. Enfin, on souhaiterait qu'en défilant avec le curseur bas jusqu'à *décembre* on puisse passer à janvier sans revenir en arrière (défilement circulaire). C'est possible avec l'option `wrap=True`.

Le code correspondant sera :

```
spin_month = spins[1]
spin_month.configure(values=MOIS, width=11, wrap=True)
```

Pour les jours, il sont numérotés de 1 à 31. On pourrait placer l'option `values` de la spinbox des jours sur la liste 1, 2, ..., 31. Il y a une autre possibilité que je vais utiliser : renseigner l'option `from_` à 1 et l'option `to` à 31. D'où le code :

```
spin_day = spins[0]
spin_day.configure(from_=1, to=31, wrap=True, width=4)
```

Concernant les années, a priori elles ne sont pas plafonnées mais ce n'est pas très réaliste. On va faire commencer les années en 1586, première année civile du calendrier grégorien. Et on va assurer le fonctionnement de l'application jusqu'au prochain millénaire. D'où le code suivant :

```
spin_year = spins[2]
spin_year.configure(from_=1586, to=3000, width=10, wrap=True)
```

On pourrait permettre un défilement « infini » en appliquant la technique `register` signalée dans le widget spinbox. Mais ça compliquerait le code final pour un bénéfice limité.

On obtient le code suivant :

```
from tkinter import *
from datetime import date, timedelta

MOIS = [
    "janvier", "février", "mars", "avril", "mai", "juin", "juillet",
    "août", "septembre", "octobre", "novembre", "décembre"
]

root = Tk()

PADX = 10
PADY = 30
FONT = "Times 20"

spins = []
tkvars = []
for j in range(3):
```

```
    tkvar = StringVar()
    tkvars.append(tkvar)
    spin = Spinbox(
        root,
        justify=CENTER,
        state="readonly",
        readonlybackground="white",
        font=FONT,
        textvariable=tkvar)
    spin.grid(row=0, column=j, padx=PADX, pady=PADY)
    spins.append(spin)

# les jours

spin_day = spins[0]

spin_day.configure(from_=1, to=31, wrap=True, width=4)

# les mois

spin_month = spins[1]

spin_month.configure(values=MOIS, width=10, wrap=True)

# les années

spin_year = spins[2]

spin_year.configure(from_=1586, to=3000, width=10, wrap=True)

btn = Button(root, text="Aujourd'hui", font=FONT)
btn.grid(
    row=1, column=0, columnspan=3, sticky=E + W, padx=PADX, pady=PADY)
root.mainloop()
```

La gestion du temps en Python

Pour poursuivre notre application, il va falloir travailler avec des données temporelles : des dates du calendrier, la date d'aujourd'hui et des écarts de date. Pour cela, on va utiliser le module standard `datetime` qui permet de gérer les dates (et, d'ailleurs, aussi les heures). Typiquement, il fournit les outils pour connaître, par exemple :

— la date ou l'heure courantes,
— le jour de la semaine d'une date donnée,
— l'écart entre deux dates ou deux heures données,
— etc.

`date` est une classe fournie par le module `datetime` et qui permet de construire des dates, par exemple :

```
from datetime import date

ma_date = date(2038, 1, 19)

print(ma_date)
```

```
2038-01-19
```

— `ma_date` représente ici la date du 19 janvier 2038.

Un appel de la forme `date(a, m, j)` construit un objet de type `date` représentant le jour `j` du mois `m` et de l'année `a` où $1 \leq j \leq 31$ et $1 \leq m \leq 12$. Les dates considérées sont des dates d'un calendrier grégorien éternel.

Attributs usuels d'une date

Un objet `ma_date` de type `date` possède des attributs permettant d'accéder au jour du mois, au mois ou à l'année de `ma_date` :

```
from datetime import date

ma_date = date(2038, 1, 19)

print(ma_date.year)
print(ma_date.month)
print(ma_date.day)
```

```
2038
1
19
```

— `ma_date` représente le 19 janvier de l'an 2038.

La date d'aujourd'hui

La date du jour courant (ce que j'appelle *aujourd'hui*) est obtenue avec la méthode `today` de la classe `date` :

```
from datetime import date

aujourdhui = date.today()
print(aujourdhui)
```

```
2019-05-21
```

Comme `date.today` renvoie un objet de type date, cet objet possède les attributs déjà signalés, comme `year` :

```
from datetime import date

aujourdhui = date.today()
print(aujourdhui)

print(aujourdhui.year)
```

```
2019-05-21
2019
```

Le calcul de l'écart de date

Le module standard `datetime` peut automatiquement indiquer l'écart entre deux dates :

```
1 from datetime import date
2
3 ma_date = date(2038, 1, 19)
4 apres = date(2038, 3, 2)
5 ecart = (apres-ma_date).days
6 ecart_bis = (ma_date-apres).days
7
8 print("Écart :", ecart)
9 print("Écart :", ecart_bis)
```

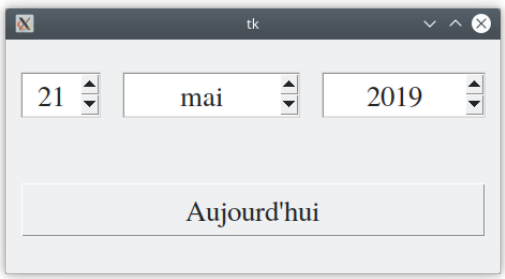
```
10 Écart : 42
11 Écart : -42
```

— Ligne 6 : on calcule un écart de date en faisant une différence de dates (avec l'opérateur -). Cet écart est un objet qui possède un attribut `days` qui indique l'écart en jours.

— Lignes 9 et 11 : l'écart peut être négatif si on demande la différence entre une date et une date postérieure.

Initialisation des spinbox

Quand l'interface de l'application s'ouvre, la date visible est celle du jour (aujourd'hui).



Il faudra donc l'initialiser. Cela sera fait automatiquement par le module `datetime` comme expliqué ci-dessus. Donc quelque part dans le code, on va trouver ceci (code partiel) :

```
from datetime import date
aujourdhui = date.today()

d = aujourdhui.day
m = aujourdhui.month
y = aujourdhui.year
```

Pour initialiser les spinbox, il faut remplir l'entrée de chaque widget. Cette entrée peut être pilotée par l'option `textvariable` initialisée par une variable de contrôle Tkinter (une spinbox, à la différence d'un widget `Entry`, ne contient pas d'option `text`).

On va donc changer notre code d'initialisation des spinbox et y ajouter l'option `textvariable` :

```
1  spins = []
2  tkvars = []
3  for j in range(3):
4      tkvar = StringVar()
5      tkvars.append(tkvar)
6      spin = Spinbox(
7          root,
8          justify=CENTER,
9          state="readonly",
10         readonlybackground="white",
11         font=FONT,
12         textvariable=tkvar)
13     spin.grid(row=0, column=j, padx=PADX, pady=PADY)
14     spins.append(spin)
```

— Ligne 2 : la liste des variables de contrôle.

— Ligne 4 : création d'une variable de contrôle pour chaque widget et (ligne 5) placement dans la liste.

— Ligne 12 : chaque variable de contrôle devient option `textvariable` de chaque widget.

La mise-à-jour dans l'interface de la date du jour doit être faite à l'initilisation de l'application mais aussi chaque fois qu'on va cliquer sur le bouton. Donc, on va directement écrire une fonction qui se chargera des deux tâches. Le code (non complet) correspondant est le suivant :

```
1  MOIS = [
2      "janvier", "février", "mars", "avril", "mai", "juin", "juillet",
3      "août", "septembre", "octobre", "novembre", "décembre"
4  ]
5
6
7  def reset():
8      aujourdhui = date.today()
9
10     d = aujourdhui.day
11     m = aujourdhui.month
12     y = aujourdhui.year
13
14     var_day = tkvars[0]
15     var_day.set(d)
16
17     var_month = tkvars[1]
18     var_month.set(MOIS[m - 1])
19
20     var_year = tkvars[2]
21     var_year.set(y)
```

```
22
23      btn["text"] = "Aujourd'hui"
```

— Lignes 10-12 : calcul des éléments de la date du jour

— Ligne 14 : récupération de la variable de contrôle du texte de la spinbox des jours

— Ligne 15 : mise à jour du texte avec la méthode `set` de StringVar.

— Idem pour les spinbox de mois (lignes 17-18) et d'années (lignes 20-21)

— Ligne 18 : la numérotation des mois dans `MOIS` commence à 0 tandis que le module `date` numérote à partir de 1, d'où l'écart `m - 1`.

— Ligne 23 : le texte du bouton indiquera `Aujourd'hui` quand on cliquera dessus.

Voici le code complet :

```
from tkinter import *
from datetime import date

MOIS = [
    "janvier", "février", "mars", "avril", "mai", "juin",
    "juillet", "août", "septembre", "octobre", "novembre", "décembre"
]

PADX = 10
PADY = 30
FONT = "Times 20"


def go():
    root = Tk()
    build_and_place(root)
    reset()

    root.mainloop()


def build_and_place(root):
    global tkvars, btn

    spins = []
    tkvars = []
    for j in range(3):
        tkvar = StringVar()
        tkvars.append(tkvar)
        spin = Spinbox(
            root,
            justify=CENTER,
            state="readonly",
            readonlybackground="white",
            font=FONT,
            textvariable=tkvar)
```

```
        spin.grid(row=0, column=j, padx=PADX, pady=PADY)
        spins.append(spin)

    # les jours

    spin_day = spins[0]

    spin_day.configure(from_=1, to=31, wrap=True, width=4)

    # les mois

    spin_month = spins[1]

    spin_month.configure(values=MOIS, width=11, wrap=True)

    # les années

    spin_year = spins[2]

    spin_year.configure(from_=1586, to=3000, width=10, wrap=True)

    btn = Button(root, text="Aujourd'hui", font=FONT, command=reset)
    btn.grid(
        row=1,
        column=0,
        columnspan=3,
        sticky=E + W,
        padx=PADX,
        pady=PADY)


def reset():
    aujourdhui = date.today()

    d = aujourdhui.day
    m = aujourdhui.month
    y = aujourdhui.year
    var_day = tkvars[0]
    var_day.set(d)
    var_month = tkvars[1]
    var_month.set(MOIS[m - 1])
    var_year = tkvars[2]
    var_year.set(y)

    btn["text"] = "Aujourd'hui"


go()
```

Tout le code (sauf l'appel en dernière ligne) a été placé dans des fonctions. Cela contraint à déclarer quelques variables en `global` mais le code placé ainsi écrit reste **beaucoup plus clair et intelligible** que du code placé en **vrac**. Cela permet de savoir plus facilement qui fait quoi et à quel moment.

Comme leur nom l'indique, la fonction `go` lance le programme et les initialisations, la fonction `build_and_place` génère les widgets. la fonction `reset` initialise ou réinitialise la date visible.

La fonction `reset` ne peut admettre de paramètres car elle sera ultérieurement une fonction callback du bouton de l'interface. Les variables utilisées par `reset` sont placées dans le scope global par la fonction `build_and_place`.

Implémentation de l'écart de dates

Pour calculer l'écart entre aujourd'hui et la date indiquée par l'interface, le programme doit d'abord récupérer la date entrée par l'utilisateur. Il suffit pour cela de lire le contenu de chacune des trois entrées des spinbox. Ce contenu étant placé dans une StringVar, il va se lire avec la méthode `get` de la StringVar. Ainsi, pour une date telle que le 19 janvier 2038, on peut récupérer 3 chaînes comme indiqué ci-dessous :

```
dd = tkvars[0].get()
mm = tkvars[1].get()
yy = tkvars[2].get()

print(repr(dd), repr(mm), repr(yy))
```

```
'19' 'janvier' '2038'
```

Ci-dessus, il a été utilisé la fonction `repr` pour que le type des objets soit bien apparent (type chaîne).

Il va falloir ensuite convertir ce format en un format utilisable par la classe `date` du module `datetime`. On va donc écrire une fonction chargée de la conversion. Il suffit pour cela de changer la **chaîne** représentant le numéro de jour et l'année en les **entiers** correspondants (lignes 9-10 ci-dessous) et de calculer le numéro de mois à partir de son nom. Comme les noms de mois sont stockés dans l'ordre dans une liste MOIS, avec la méthode `index` d'une liste on peut récupérer ce numéro (ligne 11). D'où le code suivant :

```
1  from datetime import date
2
3  MOIS = [
4      "janvier", "février", "mars", "avril", "mai", "juin",
5      "juillet", "août", "septembre", "octobre", "novembre", "décembre"
6  ]
7
8  def date_from_string(str_year, str_month, str_day):
9      y = int(str_year)
10     d = int(str_day)
11     m = MOIS.index(str_month) + 1
12     return date(year=y, month=m, day=d)
13
14
```

```
dd = "19"
mm = "janvier"
yy = "2038"

print(date_from_string(yy, mm, dd))
```

```
2038-01-19
```

Concernant l'affichage dans l'application graphique, il faut traiter à part les cas que l'on va indiquer en toutes lettes, à savoir :

— aujourd'hui
— demain
— après-demain
— hier
— avant-hier.

Pour les autres écarts (à partir de 3 jours de la date du jour), par exemple *42 jours*, on écrira selon les cas :

— *Dans 42 jours*
— *Il y a 42 jours*

Enfin, il faut voir que rien n'a été prévu pour empêcher l'application d'afficher des dates invalides, comme le 31 avril. Il faut que ce cas soit géré. Si on fournit à la classe `date` de `datetime` une date invalide, par exemple `date(2038, 4, 31)`, il va lever une exception de type `ValueError`. Plutôt que d'empêcher ce cas de survenir, il est plus simple de gérer l'exception en affichant un message dans l'application pour que l'utilisateur rectifie sa date. D'où la fonction suivante qui va gérer dans l'application l'écart de dates :

```
def delta():
    yy = tkvars[2].get()
    mm = tkvars[1].get()
    dd = tkvars[0].get()

    try:

        date = date_from_string(yy, mm, dd)
        jours = (date - aujourdhui).days

        message = [
            "Aujourd'hui", "Demain", "Après-demain", "Avant-hier",
            "Hier"
        ]

        if abs(jours) < 3:
            btn["text"] = message[jours]
        elif jours > 2:
            btn["text"] = "Dans %s jours" % jours
        else:
            btn["text"] = "Il y a %s jours" % (-jours)
```

```
22
23      except ValueError:
24
25          btn["text"]="Date invalide"
```

— Lignes 2-4 : on récupère sous forme de chaînes le contenu des 3 spinbox.

— Ligne 8 : on convertit la date lue en une date au format `date` du module `datetime`

— Ligne 9 : comme indiqué plus haut, il se peut que la date indiquée par l'utilisateur soit invalide ce qui déclencherait une exception de type `ValueError`. Puisque l'instruction se trouve dans un bloc `try` (cf. ligne 6), elle est gérée si elle décleche l'erreur (cf. la ligne 23)

— Ligne 9 : `aujourdhui` est calculé ailleurs dans le programme et placé en `global`.

— Ligne 9 : on calcule le nombre de jours qui séparent la date indiquée et le jour courant. Noter que ce nombre peut être négatif ce qui indique que la date est antérieure.

— Lignes 16-21 : la suite du code dépend de cet écart. S'il est de moins de trois jours, un message particulier est affiché.

— Lignes 11-14 : pour une meilleure lisibilité, les messages possibles sont placés dans une liste (d'une manière bien particulière, cf. explication ci-dessous).

— Ligne 17 : on utilise une astuce pour extraire le bon message. Le placement des messages dans la liste est fait en sorte que `message[jours]` soit le message à indiquer (rappel : en cas d'indice négatif, une liste est lue à l'envers en Python).

— Ligne 16 et ligne 20 : dans les autres cas, on affiche un message complet indiquant explicitement le nombre de jours.

— Lignes 17, 19, 21 et 25 : dans tous les cas, le message est indiqué sur le bouton d'où les réaffectations de `btn["text"]`.

Par ailleurs, il faudra que chaque spinbox réagisse à un clic en affichant le nombre de jours donc la fonction ci-dessus va être la fonction de l'option `command` de chaque spinbox. Il va falloir rajouter ces options. D'où le code (non complet) suivant :

```
1 def go():
2     root = Tk()
3     build_and_place(root)
4     reset()
5     for spin in spins:
6         spin["command"] = delta
7
8     root.mainloop()
```

— ligne 6 : option `command` de chaque widget.

Finalement voici le code complet :

```
from tkinter import *
from datetime import date

MOIS = [
    "janvier", "février", "mars", "avril", "mai", "juin", "juillet",
    "août", "septembre", "octobre", "novembre", "décembre"
]
```

```
PADX = 10
PADY = 30
FONT = "Times 20"


def go():
    root = Tk()
    build_and_place(root)
    reset()
    for spin in spins:
        spin["command"] = delta

    root.mainloop()


def build_and_place(root):
    global tkvars, btn, spins

    spins = []
    tkvars = []
    for j in range(3):
        tkvar = StringVar()
        tkvars.append(tkvar)
        spin = Spinbox(
            root,
            justify=CENTER,
            state="readonly",
            readonlybackground="white",
            font=FONT,
            textvariable=tkvar)
        spin.grid(row=0, column=j, padx=PADX, pady=PADY)
        spins.append(spin)

    # les jours

    spin_day = spins[0]

    spin_day.configure(from_=1, to=31, wrap=True, width=4)

    # les mois

    spin_month = spins[1]

    spin_month.configure(values=MOIS, width=10, wrap=True)

    # les années

    spin_year = spins[2]
```

```
    spin_year.configure(from_=1586, to=3000, width=10, wrap=True)

    btn = Button(root, text="Aujourd'hui", font=FONT, command=reset)
    btn.grid(
        row=1,
        column=0,
        columnspan=3,
        sticky=E + W,
        padx=PADX,
        pady=PADY)


def reset():
    global aujourdhui
    aujourdhui = date.today()

    d = aujourdhui.day
    m = aujourdhui.month
    y = aujourdhui.year
    var_day = tkvars[0]
    var_day.set(d)
    var_month = tkvars[1]
    var_month.set(MOIS[m - 1])
    var_year = tkvars[2]
    var_year.set(y)

    btn["text"] = "Aujourd'hui"


def date_from_string(str_year, str_month, str_day):
    y = int(str_year)
    d = int(str_day)
    m = MOIS.index(str_month) + 1
    return date(year=y, month=m, day=d)


def delta():
    yy = tkvars[2].get()
    mm = tkvars[1].get()
    dd = tkvars[0].get()

    try:

        date = date_from_string(yy, mm, dd)
        jours = (date - aujourdhui).days

        message = [
            "Aujourd'hui", "Demain", "Après-demain", "Avant-hier",
```

```
            "Hier"
        ]

        if abs(jours) < 3:
            btn["text"] = message[jours]
        elif jours > 2:
            btn["text"] = "Dans %s jours" % jours
        else:
            btn["text"] = "Il y a %s jours" % (-jours)

    except ValueError:

        btn["text"] = "Date invalide"


go()
```