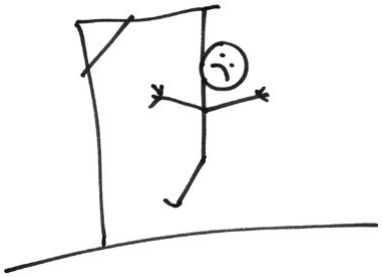


Chapitre 8

Le pendu

8.1. Nouveaux thèmes abordés dans ce chapitre

- code ASCII
- `tkinter : canvas, pack, place, create_text, delete`

8.2. Règles du jeu

Voir § 7.4.

8.3. Code du programme

pendu-graphique.py

```
# Le jeu du pendu

from tkinter import *
from random import choice

fichier = open("liste_mots.txt", "r")
liste_mots = fichier.readlines()    # met tous les mots du fichier dans une liste
fichier.close()


def lettre_dans_mot(lettre) :
    global partie_en_cours, mot_partiel, mot_choisi, nb_echecs, image_pendu
    if partie_en_cours :
        nouveau_mot_partiel = ""
        lettre_dans_mot = False
        i=0
        while i<len(mot_choisi):
            if mot_choisi[i]==lettre:
                nouveau_mot_partiel = nouveau_mot_partiel + lettre
                lettre_dans_mot = True
            else:
                nouveau_mot_partiel = nouveau_mot_partiel + mot_partiel[i]
            i+=1
        mot_partiel = nouveau_mot_partiel
        afficher_mot(mot_partiel)
        if not lettre_dans_mot :        # lettre fausse. Changer le dessin.
            nb_echecs += 1
            nomFichier = "pendu_"+str(nb_echecs)+".gif"
            photo=PhotoImage(file=nomFichier)
            image_pendu.config(image=photo)
            image_pendu.image=photo
```

```
            if nb_echecs == 7:  # trop d'erreurs. Fini.
                partie_en_cours = False
                afficher_mot(mot_choisi)
        elif mot_partiel == mot_choisi:  # le mot a été trouvé !
            partie_en_cours = False


def afficher_mot(mot):
    global lettres
    mot_large = ""
    i=0
    while i<len(mot):  # ajoute un espace entre les lettres
        mot_large = mot_large + mot[i] + " "
        i+=1
    canevas.delete(lettres)
    lettres = canevas.create_text(320,60,text=mot_large,fill='black',font='Courrier 30')


def init_jeu():
    global mot_choisi, mot_partiel, image_pendu, lettres, nb_echecs
    global partie_en_cours, liste_mots
    nb_echecs = 0
    partie_en_cours = True
    mot_choisi = choice(liste_mots).rstrip()
    mot_choisi = mot_choisi.upper()
    mot_partiel = "-" * len(mot_choisi)
    canevas.delete(lettres)
    afficher_mot(mot_partiel)
    photo=PhotoImage(file="pendu_0.gif")
    image_pendu.config(image=photo)
    image_pendu.image=photo


# création du widget principal

fenetre = Tk()
fenetre.title("Le jeu du pendu")

canevas = Canvas(fenetre, bg='white', height=500, width=620)
canevas.pack(side=BOTTOM)

bouton = [0]*26
for i in range(26):
    bouton[i] = Button(fenetre,text=chr(i+65),command=lambda x=i+65:lettre_dans_mot(chr(x)))
    bouton[i].pack(side=LEFT)

bouton2 = Button(fenetre,text='Quitter',command=fenetre.quit)
bouton2.pack(side=RIGHT)
bouton1 = Button(fenetre,text='Recommencer',command=init_jeu)
bouton1.pack(side=RIGHT)

photo=PhotoImage(file="pendu_0.gif")
image_pendu = Label(canevas, image=photo, border=0)
image_pendu.place(x=120, y=140)
lettres = canevas.create_text(320,60,text="",fill='black',font='Courrier 30')

init_jeu()

fenetre.mainloop()
fenetre.destroy()
```

8.4. Analyse du programme

Le programme est divisé en cinq parties.

Tout d'abord, on charge les mots lus dans un fichier externe dans la liste `liste_mots`, comme dans la version non graphique (voir chapitre 7).

```
fichier = open("liste_mots.txt", "r")
liste_mots = fichier.readlines()    # met tous les mots du fichier dans une liste
fichier.close()
```

La procédure `lettre_dans_mot()` se charge de vérifier que la lettre proposée figure dans le mot choisi. Si c'est le cas, la lettre sera placé au bon endroit dans le mot qui sera affiché.

```
def lettre_dans_mot(lettre) :
    global partie_en_cours, mot_partiel, mot_choisi, nb_echecs, image_pendu
    if partie_en_cours :
        nouveau_mot_partiel = ""
        lettre_dans_mot = False
        i=0
        while i<len(mot_choisi):
            if mot_choisi[i]==lettre:
                nouveau_mot_partiel = nouveau_mot_partiel + lettre
                lettre_dans_mot = True
            else:
                nouveau_mot_partiel = nouveau_mot_partiel + mot_partiel[i]
            i+=1
        mot_partiel = nouveau_mot_partiel
        afficher_mot(mot_partiel)
```

Dans le cas contraire, le dessin du pendu sera continué. En fait, l'image du pendu sera remplacée par une autre plus complète.

```
        if not lettre_dans_mot :        # lettre fausse. Changer le dessin.
            nb_echecs += 1
            nomFichier = "pendu_"+str(nb_echecs)+".gif"
            photo=PhotoImage(file=nomFichier)
            image_pendu.config(image=photo)
            image_pendu.image=photo
```

Si le dernier dessin est affiché, la partie s'arrête et le mot complet et affiché.

```
            if nb_echecs == 7:  # trop d'erreurs. Fini.
                partie_en_cours = False
                afficher_mot(mot_choisi)
        elif mot_partiel == mot_choisi:  # le mot a été trouvé !
            partie_en_cours = False
```

Comme son nom l'indique, la procédure `afficher_mot()` écrit sur l'écran un mot partiel composé de tirets et des lettres qui ont été trouvées.

```
def afficher_mot(mot):
    global lettres
    mot_large = ""
    i=0
    while i<len(mot):  # ajoute un espace entre les lettres
        mot_large = mot_large + mot[i] + " "
        i+=1
    canevas.delete(lettres)
    lettres = canevas.create_text(320,60,text=mot_large,fill='black',font='Courrier 30')
```

Nous reviendrons aux paragraphes 8.4.1 et 8.4.2 sur la dernière ligne de cette procédure. Nous verrons ce que signifie le `chr` et le `lambda`.

Remarquons que l'ancien mot partiel, qui a comme nom de variable `lettres`, est effacé par l'instruction `canevas.delete(lettres)`, avant que le nouveau soit réécrit au même endroit par l'instruction `canevas.create_text()`.

La procédure `init_jeu()` initialise toutes les variables globales avant de (re)commencer une partie, tire au hasard un nouveau mot dans la liste et écrit des tirets correspondant au nombre de lettres, et enfin affiche le premier dessin du pendu, qui se nomme `pendu_0.gif`.

```
def init_jeu():
    global mot_choisi, mot_partiel, image_pendu, lettres, nb_echecs
    global partie_en_cours, liste_mots
    nb_echecs = 0
    partie_en_cours = True
    mot_choisi = choice(liste_mots).rstrip()
```

```
    mot_choisi = mot_choisi.upper()
    mot_partiel = "-" * len(mot_choisi)
    afficher_mot(mot_partiel)
    photo=PhotoImage(file="pendu_0.gif")
    image_pendu.config(image=photo)
    image_pendu.image=photo
```

Dans la dernière partie, on met en place les différentes composantes de l'interface graphique.

La fenêtre

La première ligne crée la fenêtre, la seconde lui donne un titre. Notez que l'on ne donne pas les dimensions de la fenêtre : elle est « élastique » et s'adaptera automatiquement au contenu.

```
fenetre = Tk()
fenetre.title("Le jeu du pendu")
```

Le canevas

Un canevas est une surface réservée aux éléments graphiques. On indique dans quelle fenêtre il devra se trouver, sa couleur de fond (blanc), et ses dimensions (500 pixels de haut et 620 de large). La seconde ligne indique que ce canevas sera placé en bas de la fenêtre. La fonction `pack` sera expliquée plus en détail au § 8.4.3.

```
canevas = Canvas(fenetre, bg='white', height=500, width=620)
canevas.pack(side=BOTTOM)
```

Les boutons

Les 26 boutons lettres seront placés dans la fenêtre, au-dessus du canevas (puisqu'on a dit que le canevas occupe le bas de la fenêtre) et tassés à gauche (la fonction `chr` sera expliquée au § 8.4.1),

```
bouton = [0]*26
for i in range(26):
    bouton[i] = Button(fenetre,text=chr(i+65),command=lambda x=i+65:lettre_dans_mot(chr(x)))
    bouton[i].pack(side=LEFT)
```

un bouton *Quitter*, tassé à droite

```
bouton2 = Button(fenetre,text='Quitter',command=fenetre.quit)
bouton2.pack(side=RIGHT)
```

et un bouton *Recommencer*, tassé à droite

```
bouton1 = Button(fenetre,text='Recommencer',command=init_jeu)
bouton1.pack(side=RIGHT)
```

Notez que le bouton le plus à droite (*Quitter*) doit être écrit en premier.

Le dessin du pendu

Le dessin du pendu, qui est un label, est placé dans la canevas aux coordonnées (120,140). La fonction `place` sera expliquée plus en détail au § 8.4.4.

```
photo=PhotoImage(file="pendu_0.gif")
image_pendu = Label(canevas, image=photo, border=0)
image_pendu.place(x=120, y=140)
```

Le mot à découvrir

Le mot à découvrir est lui aussi placé dans le canevas, aux coordonnées (320, 60). Il sera écrit en noir, avec la police « Courrier 30 ».

```
lettres = canevas.create_text(320,60,text="",fill='black',font='Courrier 30')
```

Remarquez bien que le dessin du pendu et le mot à découvrir sont placés dans le canevas,

contrairement aux boutons.

8.4.1. Le code ASCII

On prononce phonétiquement « aski »

Le code ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) est une norme d'encodage informatique des caractères alphanumériques de l'alphabet latin. La norme **ASCII** établit une correspondance entre une représentation numérique des caractères de l'alphabet latin ainsi que les symboles et les signes. Par exemple, le caractère « A » est associé à 65 et « a » à 97. Les nombres du code ASCII vont de 0 à 127.

Le codage **ASCII (voir tableau ci-dessous)** est souvent complété par des correspondances supplémentaires afin de permettre l'encodage informatique d'autres caractères, comme les caractères accentués par exemple. Cette norme s'appelle **ISO-8859** et se décline par exemple en **ISO-8859-1** lorsqu'elle étend l'**ASCII** avec les caractères accentués d'Europe occidentale.

La colonne jaune est le code ASCII en base 16, aussi appelée base hexadécimale. Elle ne nous sert à rien pour l'instant.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

En Python, la fonction qui donne de code ASCII d'un caractère s'appelle `ord`. Ainsi,

```
ord('A')
```

donnera comme résultat 65.

Inversement, pour écrire le caractère correspondant à un code ASCII, on utilisera la fonction `chr` :

```
chr(65)
```

donnera comme résultat 'A'. On comprend maintenant mieux notre ligne mystérieuse :

```
bouton[i] = Button(fenetre,text=chr(i+65),command=lambda x=i+65:lettre_dans_mot(chr(x)))
```

Sur le bouton 0 sera écrit la lettre 'A' (`chr(65)`), sur le bouton 1 la lettre 'B' (`chr(66)`), etc.

Exercice 8.1

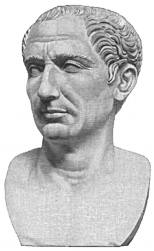
Écrivez une fonction qui construit un dictionnaire ayant comme clé un symbole (lettre, chiffres, etc.) et comme valeur son code ASCII.

Exercice 8.2

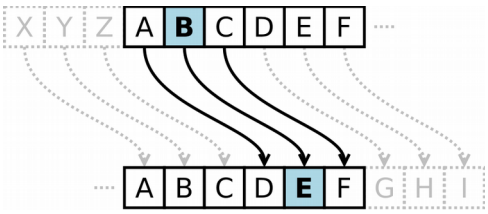
La fonction upper() permet d'écrire une phrase avec uniquement des majuscules. Programmez cette fonction, que vous nommerez majuscules(texte), après avoir remarqué que chaque lettre majuscule est éloignée de 32 rangs de sa correspondante minuscule. Les autres caractères (espace, caractères de ponctuation, ...) resteront inchangés. On supposera que le texte est en anglais, et donc non accentué.

Exercice 8.3 : le chiffre de César

Le *chiffre de César* est une méthode de chiffrement consistant à décaler les lettres de l'alphabet de quelques crans vers la gauche. Par exemple, si on décale les lettres de 3 rangs (comme le faisait Jules César), on obtient la table de chiffrement suivante :



Cette méthode a été utilisée par **Jules César** dans ses correspondances secrètes (ce qui explique le nom « chiffre de César »).



En utilisant cette table, le message *Ave Caesar morituri te salutant* devient DYH FDHVDU PRULWXUL WH VDOXWDQW.

Écrivez un programme Python qui implémente le chiffre de César. Utilisez pour cela les fonctions `ord()` et `chr()`. On doit pouvoir choisir le décalage des lettres entre 1 et 25. L'entrée sera un stexte quelconque sans accents et sans ponctuation ; par contre, le texte d'entrée sera composé de majuscules et de minuscules. Le texte de sortie sera écrit tout en majuscules. Les espaces du texte d'entrée se retrouveront aux mêmes endroits dans le texte chiffré.

Faites en sorte que l'on puisse crypter, mais aussi décrypter un message.

Exercice 8.4 : chiffre de Vigenère

Le *chiffre de Vigenère* est une amélioration décisive du chiffre de César. Sa force réside dans l'utilisation non pas d'un, mais de 26 alphabets décalés pour chiffrer un message. Ce chiffre utilise une clé qui définit le décalage pour chaque lettre du message (A : décalage de 0 cran, B : 1 cran, C : 2 crans, ..., Z : 25 crans).

Blaise de Vigenère
(1523-1596)

Exemple : chiffrons le texte « CHIFFRE DE VIGENERE » avec la clef « BACHELIER » (cette clé est éventuellement répétée plusieurs fois pour être aussi longue que le texte clair).

Clair	c	h	i	f	f	r	e	d	e	v	i	g	e	n	e	r	e
Clé	B	A	C	H	E	L	I	E	R	B	A	C	H	E	L	I	E
Décalage	1	0	2	7	4	11	8	4	17	1	0	2	7	4	11	8	4
Chiffré	D	H	K	M	J	C	M	H	V	W	I	I	L	R	P	Z	I

Modifiez le programme de l'exercice 8.3 pour crypter et décrypter des messages avec le chiffre de Vigenère.

D'autres exercices de cryptologie ?

Si le sujet vous intéresse, vous pourrez trouver d'autres exercices sur le site *Ars Cryptographica* :

www.apprendre-en-ligne.net/crypto/python/

Nous avons déjà vu les fonctions lambda au § 6.4

8.4.2. Les fonctions lambda

Par défaut, le paramètre `command` d'un `Button` nécessite une fonction sans argument (voir programme du chapitre 6). Vous pouvez cependant utiliser les fonctions `lambda` pour passer des arguments supplémentaires, ce qui peut être utile si vous avez plusieurs `Button` qui pointent sur la même fonction.

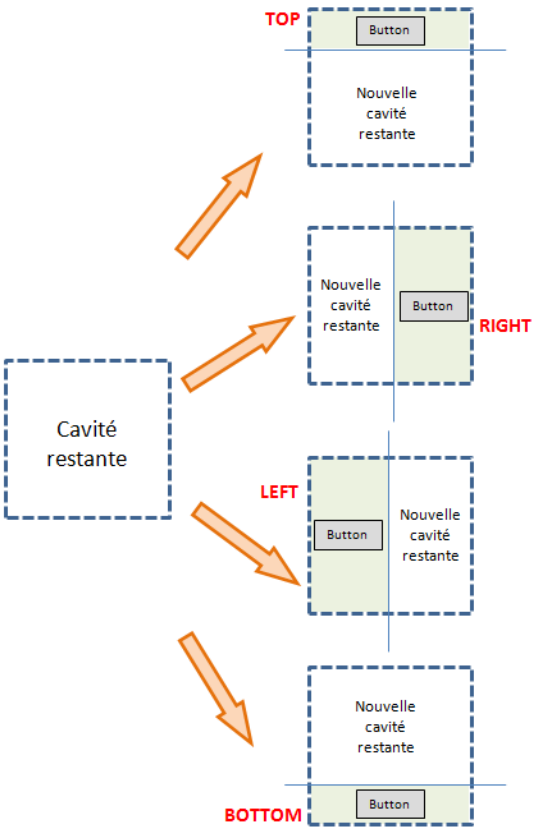
8.4.3. L'instruction `pack` de `tkinter`

Le module `tkinter` contient trois fonctions permettant de positionner les éléments d'une interface graphique : `grid` (que nous avons déjà vue au chapitre 4), `pack` et `place`. (voir § 8.4.4).

Le terme widget désigne toute entité susceptible d'être placée dans une fenêtre (un bouton, une image, etc)

Lorsqu'on fait appel à `pack()`, `tkinter` réduit la fenêtre pour épouser au mieux les éléments qu'elle contient. Chaque ajout d'un widget se fera de l'une des quatre manières ci-dessous, selon le côté choisi (`LEFT`, `RIGHT`, `TOP` ou `BOTTOM`).

C'est Calogero Gigante (www.gigante.be) qui a inventé ce terme de « cavité » et proposé le schéma ci-contre.



Nous utiliserons le terme de « cavité » pour désigner l'espace qui sera prochainement occupé par un widget.

On divise la cavité restante en deux, pour créer une nouvelle cavité restante, à laquelle on appliquera le même schéma lors de l'ajout d'un nouveau widget.

Dans notre programme, on a d'abord placé un canevas en bas de la fenêtre. La cavité restante est donc la partie supérieure de la fenêtre. Ensuite, on a ajouté l'un après l'autre les 26 boutons lettres à gauche ; la cavité restante était donc la partie supérieure droite de la fenêtre. Puis on a ajouté à droite le bouton *Quitter*. La cavité restante se trouve donc toujours dans la partie supérieure de la fenêtre, coincée entre le bouton *Z* et le bouton *Quitter*. C'est donc là que sera ajouté le dernier bouton *Recommencer*. À noter qu'il reste une cavité entre *Z* et *Recommencer*.

8.4.4. L'instruction `place` de `tkinter`

La commande `place` permet de placer un widget à la position (*x, y*). Pour être précis, c'est le coin supérieur gauche du widget qui est positionné en (*x, y*).

```
image_pendu.place(x=120, y=140)
```

Attention ! Les coordonnées (0, 0) se trouvent en haut à gauche du canevas (et non en bas à gauche comme on pourrait s'y attendre).

Il est possible que l'image sorte du canevas : cela ne provoquera pas d'erreur, mais votre image sera tronquée.

Exercice 8.5

Modifiez le code du § 8.3.

Quand un bouton lettre aura été pressé, la lettre y figurant disparaîtra. Cela permettra de savoir quelles lettres ont déjà été proposées.

Ajoutez aussi un message de félicitation qui apparaîtra quand le mot aura été trouvé.

Exercice 8.6

Créez une interface graphique pour le jeu de l'exercice 7.12.

8.5. Ce que vous avez appris dans ce chapitre

- Les caractères et autres symboles sont codés grâce au code ASCII.
- Un canevas est une surface réservée aux éléments graphiques.
- Il existe trois manières de positionner les éléments d'une interface graphique : `grid`, `pack` et `place`.
- Une chaîne de caractères peut être transformée en un élément graphique avec la fonction `create_text()`.