

Principales fonctions

1. Mathématiques

Opérations classiques

- $a + b$, $a - b$, $a * b$ opérations classiques
- a / b division « réelle » (renvoie un nombre flottant)
- $a // b$ quotient de la division euclidienne (renvoie un entier)
- $a \% b$ reste de la division euclidienne, appelé a modulo b
- $\text{abs}(x)$ valeur absolue
- $x ** n$ puissance x^n
- $4.56\text{e}12$ pour 4.56×10^{12}

Module « math »

L'usage d'autres fonctions mathématiques nécessite le recours au module `math` qui s'appelle par la commande :

```
from math import *
```

- $\text{sqrt}(x)$ racine carrée \sqrt{x}
- $\cos(x)$, $\sin(x)$, $\tan(x)$ fonctions trigonométriques $\cos x$, $\sin x$, $\tan x$ en radians
- π valeur approchée de $\pi = 3.14159265\dots$
- $\text{floor}(x)$ entier juste en-dessous de x
- $\text{ceil}(x)$ entier juste au-dessus de x
- $\text{gcd}(a,b)$ pgcd de a et de b

Module « random »

Le module `random` génère des nombres de façon pseudo-aléatoire. Il s'appelle par la commande :

```
from random import *
```

- $\text{random}()$ à chaque appel, renvoie un nombre flottant x au hasard vérifiant $0 \leq x < 1$.
- $\text{randint}(a,b)$ à chaque appel, renvoie un nombre entier n au hasard vérifiant $a \leq n \leq b$.
- $\text{choice}(liste)$ à chaque appel, tire au hasard un élément de la liste.
- $liste.\text{shuffle}()$ mélange la liste (la liste est modifiée).

Écriture binaire

- $\text{bin}(n)$ renvoie l'écriture binaire de l'entier n sous la forme d'une chaîne. Exemple : $\text{bin}(17)$ renvoie `'0b10001'`.
- Pour écrire directement un nombre en écriture binaire, il suffit d'écrire le nombre en commençant par `0b` (sans guillemets). Par exemple `0b11011` vaut 27.

2. Booléens

Un booléen est une donnée qui prend soit la valeur `True` (« vrai »), soit la valeur `False` (« faux »).

Comparaisons

Les tests de comparaison suivants renvoient un booléen.

- `a == b` test d'égalité
- `a < b` test inférieur strict
- `a <= b` test inférieur large
- `a > b` ou `a >= b` test supérieur
- `a != b` test de non égalité

Ne pas confondre « `a = b` » (affectation) et « `a == b` » (test d'égalité).

Opérations sur les booléens

- `P and Q` « et » logique
- `P or Q` « ou » logique
- `not P` négation

3. Chaînes de caractères I

Chaînes

- `"A"` ou `'A'` un caractère
- `"Python"` ou `'Python'` une chaîne de caractères
- `len(chaine)` la longueur de la chaîne. Exemple : `len("Python")` renvoie 6.
- `chaine1 + chaine2` concaténation.
Exemple : `"J aime bien" + "Python"` renvoie `"J aime bienPython"`.
- `chaine[i]` renvoie le *i*-ème caractère de `chaine` (la numérotation commence à 0).
Exemple avec `chaine = "Python"`, `chaine[1]` vaut `"y"`. Voir le tableau ci-dessous.

Lettre	P	y	t	h	o	n
Rang	0	1	2	3	4	5

Conversion nombre/chaîne

- **Chaîne.** `str(nombre)` convertit un nombre (entier ou flottant) en une chaîne. Exemples : `str(7)` renvoie la chaîne `"7"` ; `str(1.234)` renvoie la chaîne `"1.234"`.
- **Entier.** `int(chaine)` renvoie l'entier correspondant à la chaîne. Exemple `int("45")` renvoie l'entier 45.
- **Nombre flottant.** `float(chaine)` renvoie le nombre flottant correspondant à la chaîne. Exemple `float("3.14")` renvoie le nombre 3.14.

Sous-chaînes

- `chaine[i:j]` renvoie la sous-chaîne des caractères de rang *i* à *j* – 1 de `chaine`.
Exemple : avec `chaine = "Ceci est une chaine"`, `chaine[2:6]` renvoie `"ci e"`.
- `chaine[i:]` renvoie les caractères de rang *i* jusqu'à la fin de `chaine`.
Exemple : `chaine[5:]` renvoie `"est une chaine"`.
- `chaine[:j]` renvoie les caractères du début jusqu'au rang *j* – 1 de `chaine`. Exemple : `chaine[:4]` renvoie `"Ceci"`.

Mise en forme

La méthode `format()` permet de mettre en forme du texte ou des nombres. Cette fonction renvoie une chaîne de caractères.

- **Texte**

`Test` `Test` `Test`
 — `'{:10}'.format('Test')` alignement à gauche (sur 10 caractères)
 — `'{:>10}'.format('Test')` alignement à droite
 — `'{:~10}'.format('Test')` centré

- **Entier**

`456` `456` `000456`
 — `'{:d}'.format(456)` entier
 — `'{:6d}'.format(456)` alignement à droite (sur 6 caractères)
 — `'{:06d}'.format(456)` ajout de zéros non significatifs (sur 6 caractères)

- **Nombre flottant**

`3.141593` `3.14159265` `3.1416` `003.1416`
 — `'{:f}'.format(3.141592653589793)` nombre flottant
 — `'{: .8f}'.format(3.141592653589793)` 8 chiffres après la virgule
 — `'{:8.4f}'.format(3.141592653589793)` sur 8 caractères avec 4 chiffres après la virgule
 — `'{:08.4f}'.format(3.141592653589793)` ajout de zéros non significatifs

4. Chaînes de caractères II

Encodage

- `chr(n)` renvoie le caractère associé au numéro de code ASCII/unicode *n*. Exemple : `chr(65)` renvoie "A" ; `chr(97)` renvoie "a".
- `ord(c)` renvoie le numéro de code ASCII/unicode associé au caractère *c*. Exemple : `ord("A")` renvoie 65 ; `ord("a")` renvoie 97.

Le début de la table des codes ASCII/unicode est donné ci-dessous.

33	!	43	+	53	5	63	?	73	I	83	S	93]	103	g	113	q	123	{
34	"	44	,	54	6	64	@	74	J	84	T	94	^	104	h	114	r	124	
35	#	45	-	55	7	65	A	75	K	85	U	95	_	105	i	115	s	125	}
36	\$	46	.	56	8	66	B	76	L	86	V	96	'	106	j	116	t	126	~
37	%	47	/	57	9	67	C	77	M	87	W	97	a	107	k	117	u	127	-
38	&	48	0	58	:	68	D	78	N	88	X	98	b	108	l	118	v		
39	'	49	1	59	;	69	E	79	O	89	Y	99	c	109	m	119	w		
40	(50	2	60	<	70	F	80	P	90	Z	100	d	110	n	120	x		
41)	51	3	61	=	71	G	81	Q	91	[101	e	111	o	121	y		
42	*	52	4	62	>	72	H	82	R	92	\	102	f	112	p	122	z		

Majuscules/minuscules

- `chaine.upper()` renvoie une chaîne en majuscules.
- `chaine.lower()` renvoie une chaîne en minuscules.

Chercher/remplacer

- `sous_chaine in chaine` renvoie « vrai » ou « faux » si `sous_chaine` apparaît dans `chaine`.
Exemple : `"PAS" in "ETRE OU NE PAS ETRE"` vaut `True`.
- `chaine.find(sous_chaine)` renvoie le rang auquel la sous-chaîne a été trouvée (et -1 sinon).
Exemple : avec `chaine = "ABCDE"`, `chaine.find("CD")` renvoie 2.
- `chaine.replace(sous_chaine,nouv_sous_chaine)` remplace chaque occurrence de la sous-chaîne par la nouvelle sous-chaîne.
Exemple : avec `chaine = "ABCDE"`, `chaine.replace("CD", "XY")` renvoie `"ABXYE"`.

Séparer/regrouper

- `chaine.split(separateur)` sépare la chaîne en une liste de sous-chaînes (par défaut le séparateur est l'espace).
Exemples :
— `"Etre ou ne pas etre.".split()` renvoie `['Etre', 'ou', 'ne', 'pas', 'etre.']`
— `"12.5;17.5;18".split(";")` renvoie `['12.5', '17.5', '18']`
- `separateur.join(liste)` regroupe les sous-chaînes en une seule chaîne en ajoutant le séparateur entre chaque.
Exemples :
— `" ".join(["Etre", "ou", "ne", "pas", "etre."])` renvoie `'Etreounepasetre.'` Il manque les espaces.
— `" ".join(["Etre", "ou", "ne", "pas", "etre."])` renvoie `'Etre ou ne pas etre.'`
C'est mieux lorsque le séparateur est une espace.
— `"--".join(["Etre", "ou", "ne", "pas", "etre."])` renvoie `'Etre--ou--ne--pas--etre.'`

5. Listes I

Construction d'une liste

Exemples :

- `liste1 = [5,4,3,2,1]` une liste de 5 entiers.
- `liste2 = ["Vendredi", "Samedi", "Dimanche"]` une liste de 3 chaînes.
- `liste3 = []` la liste vide.
- `list(range(n))` liste des entiers de 0 à $n-1$.
- `list(range(a,b))` liste des entiers de a à $b-1$.
- `list(range(a,b,saut))` liste des entiers de a à $b-1$, avec un pas donné par l'entier `saut`.

Accéder à un élément

- `liste[i]` renvoie l'élément de la liste de rang i . Attention, le rang commence à 0.
Exemple : `liste = ["A", "B", "C", "D", "E", "F"]` alors `liste[2]` renvoie `"C"`.

Lettre	"A"	"B"	"C"	"D"	"E"	"F"
Rang	0	1	2	3	4	5

- `liste[-1]` renvoie le dernier élément, `liste[-2]` renvoie l'avant-dernier élément...
- `liste.pop()` supprime le dernier élément de la liste et le renvoie (c'est l'opération « dépiler »).

Ajouter un élément (ou plusieurs)

- `liste.append(element)` ajoute l'élément à la fin de la liste. Exemple : si `liste = [5,6,7,8]` alors `liste.append(9)` rajoute 9 à la liste, `liste` vaut `[5,6,7,8,9]`.
- `nouv_liste = liste + [element]` fournit une nouvelle liste avec un élément en plus à la fin.
Exemple : `[1,2,3,4] + [5]` vaut `[1,2,3,4,5]`.

- `[element] + liste` renvoie une liste où l'élément est ajouté au début. Exemple : `[5] + [1,2,3,4]` vaut `[5,1,2,3,4]`.
- `liste1 + liste2` concatène les deux listes. Exemple : avec `liste1 = [4,5,6]` et `liste2 = [7,8,9]` alors `liste1 + liste2` vaut `[4,5,6,7,8,9]`.

Exemple de construction. Voici comment construire la liste qui contient les premiers carrés :

```
liste_carres = []          # On part d'une liste vide
for i in range(10):
    liste_carres.append(i**2)  # On ajoute un carré
```

À la fin `liste_carres` vaut :

`[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

Parcourir une liste

- `len(liste)` renvoie la longueur de la liste. Exemple : `len([5,4,3,2,1])` renvoie 5.
- Parcourir simplement une liste (et ici afficher chaque élément) :

```
for element in liste:
    print(element)
```

- Parcourir une liste à l'aide du rang.

```
n = len(liste)
for i in range(n):
    print(i, liste[i])
```

6. Listes II

Mathématiques

- `max(liste)` renvoie le plus grand élément. Exemple : `max([10,16,13,14])` renvoie 16.
- `min(liste)` renvoie le plus petit élément. Exemple : `min([10,16,13,14])` renvoie 10.
- `sum(liste)` renvoie la somme de tous les éléments. Exemple : `sum([10,16,13,14])` renvoie 53.

Trancher des listes

- `liste[a:b]` renvoie la sous-liste des éléments du rang a au rang $b - 1$.
- `liste[a:]` renvoie la liste des éléments du rang a jusqu'à la fin.
- `liste[:b]` renvoie la liste des éléments du début jusqu'au rang $b - 1$.

Lettre	"A"	"B"	"C"	"D"	"E"	"F"	"G"
Rang	0	1	2	3	4	5	6

Par exemple si `liste = ["A","B","C","D","E","F","G"]` alors :

- `liste[1:4]` renvoie `["B","C","D"]`.
- `liste[:2]` c'est comme `liste[0:2]` et renvoie `["A","B"]`.
- `liste[4:]` renvoie `["E","F","G"]`. C'est la même chose que `liste[4:n]` où $n = \text{len}(\text{liste})$.

Trouver le rang d'un élément

- `liste.index(element)` renvoie la première position à laquelle l'élément a été trouvé. Exemple : avec `liste = [12, 30, 5, 9, 5, 21]`, `liste.index(5)` renvoie 2.

- Si on souhaite juste savoir si un élément appartient à une liste, alors l'instruction :

`element in liste`

renvoie True ou False. Exemple : avec `liste = [12, 30, 5, 9, 5, 21]`, «`9 in liste`» est vrai, alors que «`8 in liste`» est faux.

Ordonner

- `sorted(liste)` renvoie la liste ordonnée des éléments.
Exemple : `sorted([13,11,7,4,6,8,12,6])` renvoie la liste `[4,6,6,7,8,11,12,13]`.
- `liste.sort()` ne renvoie rien mais par contre la liste `liste` est maintenant ordonnée.

Inverser une liste

Voici trois méthodes :

- `liste.reverse()` modifie la liste sur place ;
- `list(reversed(liste))` renvoie une nouvelle liste ;
- `liste[::-1]` renvoie une nouvelle liste.

Supprimer un élément

Trois méthodes.

- `liste.remove(element)` supprime la première occurrence trouvée.
Exemple : `liste = [2,5,3,8,5]`, la commande `liste.remove(5)` modifie la liste qui maintenant vaut `[2,3,8,5]` (le premier 5 a disparu).
- `del liste[i]` supprime l'élément de rang *i* (la liste est modifiée).
- `element = liste.pop()` supprime le dernier élément de la liste et le renvoie. C'est l'opération « dépiler ».

Liste par compréhension

- Partons d'une liste, par exemple `maliste = [1,2,3,4,5,6,7,6,5,4,3,2,1]`.
- `liste_doubles = [2*x for x in maliste]` renvoie une liste qui contient les doubles des éléments de la liste `maliste`. C'est donc la liste `[2,4,6,8,...]`.
- `liste_carres = [x**2 for x in maliste]` renvoie la liste des carrés de éléments de la liste `maliste`. C'est donc la liste `[1,4,9,16,...]`.
- `liste_partielle = [x for x in maliste if x > 2]` extrait la liste composée des seuls éléments strictement supérieurs à 2. C'est donc la liste `[3,4,5,6,7,6,5,4,3]`.

Liste de listes

Exemple :

`tableau = [[2,14,5], [3,5,7], [15,19,4], [8,6,5]]`

correspond au tableau :

		indice <i>j</i>		
		→		
		<i>j</i> = 0	<i>j</i> = 1	<i>j</i> = 2
indice <i>i</i>	<i>i</i> = 0	2	14	5
	<i>i</i> = 1	3	5	7
	<i>i</i> = 2	15	19	4
	<i>i</i> = 3	8	6	5

Alors `tableau[i]` renvoie la sous-liste de rang *i*, et `tableau[i][j]` renvoie l'élément situé dans la sous-liste de rang *i*, au rang *j* de cette sous-liste. Par exemple :

- `tableau[0]` renvoie la sous-liste `[2,14,5]`.
- `tableau[1]` renvoie la sous-liste `[3,5,7]`.
- `tableau[0][0]` renvoie l'entier 2.
- `tableau[0][1]` renvoie l'entier 14.
- `tableau[2][1]` renvoie l'entier 19.

Un tableau de n lignes et p colonnes.

- `tableau = [[0 for j in range(p)] for i in range(n)]` initialise un tableau et le remplit de 0.
- `tableau[i][j] = 1` modifie une valeur du tableau (celle à l'emplacement (i,j)).

7. Entrée/sortie

Affichage

- `print(chaine1, chaine2, chaine3, ...)` affiche des chaînes ou des objets. Exemple : `print("Valeur =", 14)` affiche `Valeur = 14`. Exemple : `print("Ligne 1 \n Ligne 2")` affiche sur deux lignes.
- **Séparateur.** `print(..., sep="...")` change le séparateur (par défaut le séparateur est le caractère espace). Exemple : `print("Bob", 17, 13, 16, sep="; ")` affiche `Bob; 17; 13; 16`.
- **Fin de ligne.** `print(..., end="...")` change le caractère placé à la fin (par défaut c'est le saut de ligne `\n`). Exemple `print(17, end="")` puis `print(89)` affiche 1789 sur une seule ligne.

Entrée clavier

`input()` met le programme en pause et attend de l'utilisateur un message au clavier (qu'il termine en appuyant sur la touche « Entrée »). Le message est une chaîne de caractères.

Voici un petit programme qui demande le prénom et l'âge de l'utilisateur et affiche un message du style « Bonjour Kevin » puis « Tu es mineur/majeur » selon l'âge.

```
prenom = input("Comment t'appelles-tu ? ")
print("Bonjour", prenom)
```

```
age_chaine = input("Quel âge as-tu ? ")
age = int(age_chaine)
```

```
if age >= 18:
    print("Tu es majeur !")
else:
    print("Tu es mineur !")
```

8. Fichiers

Commande

- `fic = open("mon_fichier.txt", "r")` ouverture en lecture ("`r`" = *read*).
- `fic = open("mon_fichier.txt", "w")` ouverture en écriture ("`w`" = *write*). Le fichier est créé s'il n'existe pas, s'il existait le contenu précédent est d'abord effacé.
- `fic = open("mon_fichier.txt", "a")` ouverture en écriture, les données seront écrites à la fin des données actuelles ("`a`" = *append*).
- `fic.write("une ligne")` écriture dans le fichier.
- `fic.read()` lit tout le fichier (voir plus bas pour autre méthode).
- `fic.readlines()` lit toutes les lignes (voir plus bas pour autre méthode).

- `fic.close()` fermeture du fichier.

Écrire des lignes dans un fichier

```
fic = open("mon_fichier.txt", "w")

fic.write("Bonjour le monde\n")

ligne = "Coucou\n"
fic.write(ligne)

fic.close()
```

Lire les lignes d'un fichier

```
fic = open("mon_fichier.txt", "r")

for ligne in fic:
    print(ligne)

fic.close()
```

Lire un fichier (méthode officielle)

```
with open("mon_fichier.txt", "r") as fic:
    for ligne in fic:
        print(ligne)
```

9. Tortue

Le module `turtle` s'appelle par la commande :

```
from turtle import *
```

Principales commandes

- `forward(longueur)` avance de longueur pas
- `backward(longueur)` recule
- `right(angle)` tourne vers la droite selon l'angle donné en degrés
- `left(angle)` tourne vers la gauche
- `setheading(direction)` s'oriente dans une direction (0 = droite, 90 = haut, -90 = bas, 180 = gauche)
- `goto(x,y)` se déplace jusqu'au point (x,y)
- `setx(newx)` change la valeur de l'abscisse (déplacement horizontal)
- `sety(newy)` change la valeur de l'ordonnée (déplacement vertical)
- `down()` abaisse le stylo
- `up()` relève le stylo
- `width(epaisseur)` change l'épaisseur du trait
- `color(couleur)` change la couleur du trait : "red", "green", "blue", "orange", "purple",...
- `position()` renvoie la position (x,y) de la tortue
- `heading()` renvoie la direction angle vers laquelle pointe la tortue

- `towards(x,y)` renvoie l'angle entre l'horizontale et le segment commençant à la tortue et finissant au point (x,y)
- `speed("fastest")` vitesse maximale de déplacement
- `exitonclick()` termine le programme dès que l'on clique

Plusieurs tortues

Voici un exemple de programme avec deux tortues.

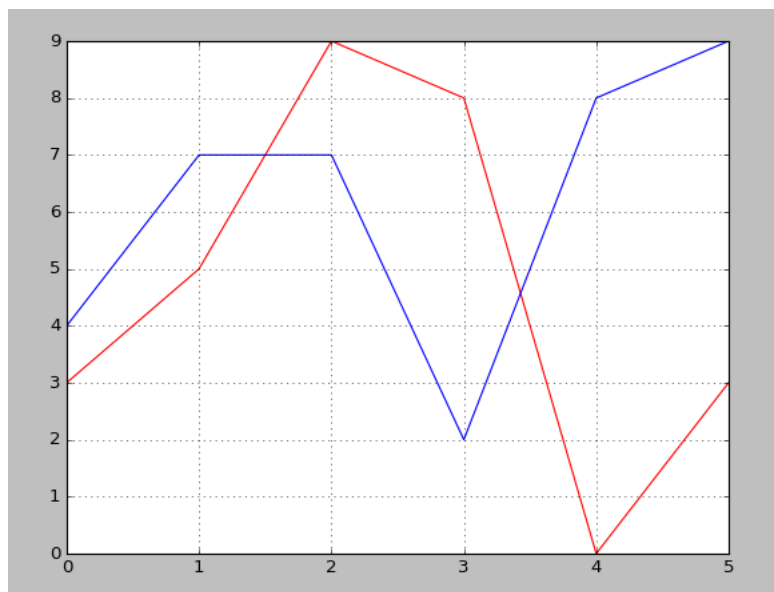
```
tortue1 = Turtle()    # Avec un 'T' majuscule !
tortue2 = Turtle()

tortue1.color('red')
tortue2.color('blue')

tortue1.forward(100)
tortue2.left(90)
tortue2.forward(100)
```

10. Matplotlib

Avec le module `matplotlib` il est très facile de tracer une liste. Voici un exemple.



```
import matplotlib.pyplot as plt
```

```
liste1 = [3,5,9,8,0,3]
liste2 = [4,7,7,2,8,9]
```

```
plt.plot(liste1,color="red")
plt.plot(liste2,color="blue")
plt.grid()
plt.show()
```

Principales fonctions.

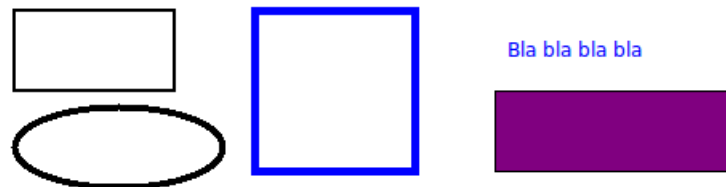
- `plt.plot(liste)` trace les points d'une liste (sous la forme (i, ℓ_i)) et les joint.

- `plt.plot(listex,listey)` trace les points d'une liste (sous la forme (x_i, y_i) où x_i parcourt la première liste et y_i la seconde).
- `plt.scatter(x,y,color='red',s=100)` affiche un point en (x, y) (d'une grosseur s).
- `plt.grid()` trace une grille.
- `plt.show()` affiche tout.
- `plt.close()` termine le tracé.
- `plt.xlim(xmin,xmax)` définit l'intervalle des x .
- `plt.ylim(ymin,ymax)` définit l'intervalle des y .
- `plt.axis('equal')` impose un repère orthonormé.

11. Tkinter

11.1. Graphiques

Pour afficher ceci :



le code est :

```
# Module tkinter
from tkinter import *

# Fenêtre tkinter
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)

# Un rectangle
canvas.create_rectangle(50,50,150,100,width=2)

# Un rectangle à gros bords bleus
canvas.create_rectangle(200,50,300,150,width=5,outline="blue")

# Un rectangle rempli de violet
canvas.create_rectangle(350,100,500,150,fill="purple")

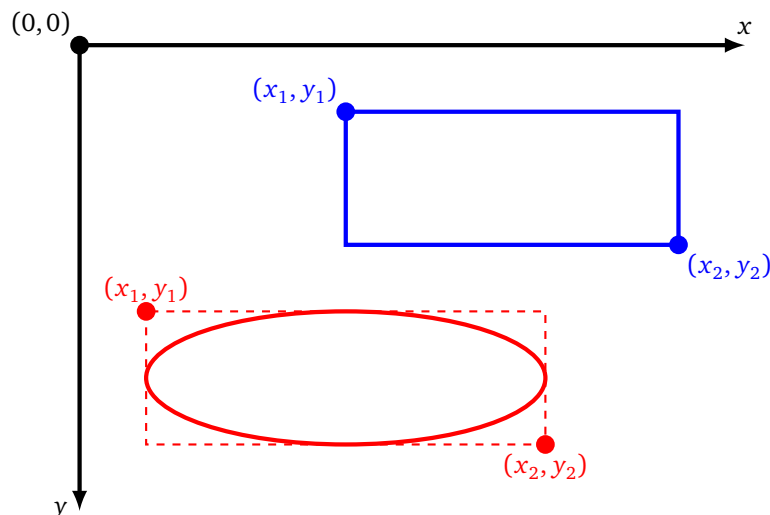
# Un ovale
canvas.create_oval(50,110,180,160,width=4)

# Du texte
canvas.create_text(400,75,text="Bla bla bla bla",fill="blue")
```

```
# Ouverture de la fenêtre
root.mainloop()
```

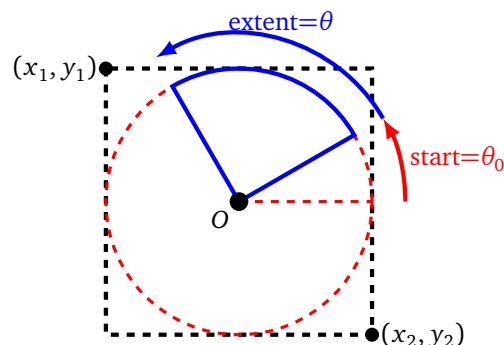
Quelques explications :

- Le module `tkinter` nous permet de définir des variables `root` et `canvas` qui définissent une fenêtre graphique (ici de largeur 800 et de hauteur 600 pixels). On décrit ensuite tout ce que l'on veut ajouter dans la fenêtre. Et enfin, la fenêtre est affichée par la commande `root.mainloop()` (tout à la fin).
- Attention ! Le repère graphique de la fenêtre a son axe des ordonnées dirigé vers le bas. L'origine $(0,0)$ est le coin en haut à gauche (voir la figure ci-dessous).
- Commande pour tracer un rectangle : `create_rectangle(x1,y1,x2,y2)` ; il suffit de préciser les coordonnées (x_1, y_1) et (x_2, y_2) de deux sommets opposés. L'option `width` ajuste l'épaisseur du trait, `outline` définit la couleur de ce trait et `fill` définit la couleur de remplissage.
- Une ellipse est tracée par la commande `create_oval(x1,y1,x2,y2)`, où (x_1, y_1) , (x_2, y_2) sont les coordonnées de deux sommets opposés d'un rectangle encadrant l'ellipse voulue (voir la figure). On obtient un cercle lorsque le rectangle correspondant est un carré.
- Le texte est affiché par la commande `canvas.create_text()` en précisant les coordonnées (x, y) du point à partir duquel on souhaite afficher le texte.



Portion de cercle. La fonction `create_arc()` n'est pas très intuitive. Il faut penser que l'on dessine un cercle, en précisant les coordonnées de deux sommets opposés d'un carré qui l'entoure, puis en précisant l'angle de début et l'angle du secteur (en degrés).

```
canvas.create_arc(x1,y1,x2,y2,start=debut_angle,extent=mon_angle)
```

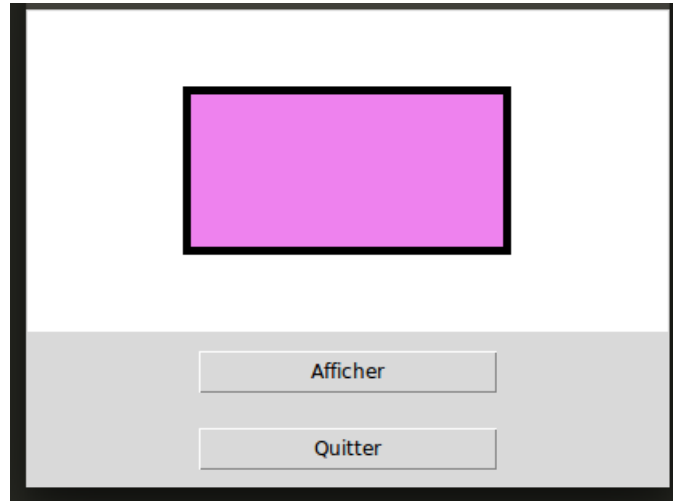


L'option `style=PIESLICE` affiche un secteur au lieu d'un arc.

11.2. Boutons

Il est plus ergonomique d'afficher des fenêtres dans lesquelles les actions sont exécutées en cliquant sur des boutons.

Voici un petit programme qui affiche une fenêtre avec deux boutons. Le premier bouton change la couleur du rectangle, le second termine le programme.



Le code est :

```
from tkinter import *
from random import *

root = Tk()
canvas = Canvas(root, width=400, height=200, background="white")
canvas.pack(fill="both", expand=True)

def action_bouton():
    canvas.delete("all")          # Efface tout
    couleurs = ["red", "orange", "yellow", "green", "cyan", "blue", "violet", "purple"]
    coul = choice(couleurs)       # Couleur au hasard
    canvas.create_rectangle(100, 50, 300, 150, width=5, fill=coul)
    return

bouton_couleur = Button(root, text="Afficher", width=20, command=action_bouton)
bouton_couleur.pack()

bouton_quitter = Button(root, text="Quitter", width=20, command=root.quit)
bouton_quitter.pack()

root.mainloop()
```

Quelques explications :

- On crée un bouton par la commande `Button`. L'option `text` personnalise le texte qui s'affiche sur le bouton. On ajoute le bouton créé à la fenêtre par la méthode `pack`.
- Le plus important est l'action associée au bouton ! C'est l'option `command` qui reçoit le nom de la fonction à exécuter lorsque le bouton est cliqué. Pour notre exemple `command=action_bouton` associe au clic sur le bouton un changement de couleur.

- Attention ! il faut donner le nom de la fonction sans parenthèses : `commande=ma_fonction` et pas `command=ma_fonction()`.
- Pour associer au bouton « Quitter » la fermeture du programme, l'argument est `command=root.quit`.
- La commande `canvas.delete("all")` efface tous les dessins de notre fenêtre graphique.

11.3. Texte

Voici comment afficher du texte avec Python et le module des fenêtres graphiques tkinter.

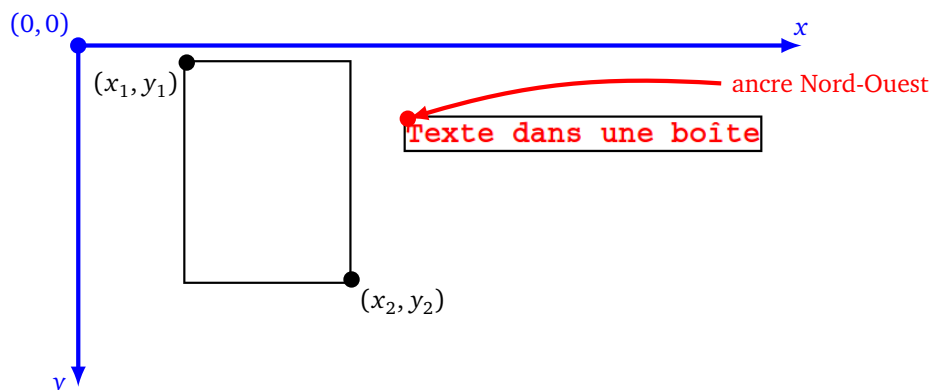
Du texte avec Python !

Le code est :

```
from tkinter import *
from tkinter.font import Font
# Fenêtre tkinter
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)
# Fonte
mafonte = Font(family="Times", size=20)
# Le texte
canvas.create_text(100,100, text="Du texte avec Python !",
anchor=NW, font=mafonte, fill="blue")
# Ouverture de la fenêtre
root.mainloop()
```

Quelques explications :

- Le texte est affiché par la fonction `canvas.create_text()`. Il faut préciser les coordonnées (x, y) du point à partir duquel on souhaite afficher le texte.
- L'option `text` permet de passer la chaîne de caractères à afficher.
- L'option `anchor` permet de préciser le point d'ancrage du texte, `anchor=NW` signifie que la zone de texte est ancrée au point Nord-Ouest (NW) (voir la figure ci-dessous).
- L'option `fill` permet de préciser la couleur du texte.
- L'option `font` permet de définir la fonte (c'est-à-dire le style et la taille des caractères). Voici des exemples de fontes, à toi de les tester :
 - `Font(family="Times", size=20)`
 - `Font(family="Courier", size=16, weight="bold")` en **gras**
 - `Font(family="Helvetica", size=16, slant="italic")` en *italique*



11.4. Clic de souris

Voici un petit programme qui affiche une fenêtre graphique. Chaque fois que l'utilisateur clique (avec le bouton gauche de la souris) le programme affiche un petit carré (sur la fenêtre) et affiche « Clic à $x = \dots$, $y = \dots$ » (sur la console).

```
from tkinter import *

# Fenêtre
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Capture des clics de souris
def action_clic_souris(event):
    canvas.focus_set()
    x = event.x
    y = event.y
    canvas.create_rectangle(x,y,x+10,y+10,fill="red")
    print("Clic à x =",x,", y =",y)
    return

# Association clic/action
canvas.bind("<Button-1>", action_clic_souris)

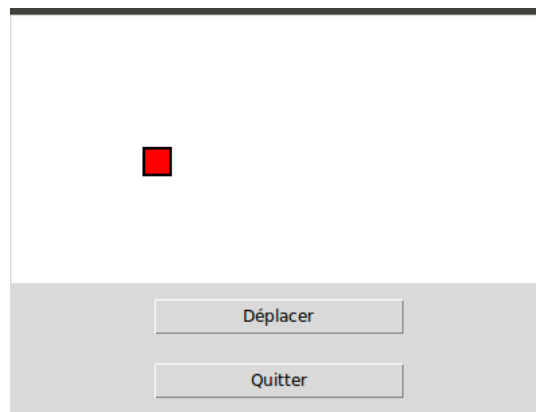
# Lancement
root.mainloop()
```

Voici quelques explications :

- La création de la fenêtre est habituelle. Le programme se termine par le lancement de la fenêtre avec la commande `mainloop()`.
- Le premier point clé est d'associer un clic de souris à une action, c'est ce que fait la ligne
`canvas.bind("<Button-1>", action_clic_souris)`
 Chaque fois que le bouton gauche de la souris est cliqué, Python exécute la fonction `action_clic_souris`. Note qu'il n'y a pas de parenthèses pour l'appel à la fonction.
- Second point clé : la fonction `action_clic_souris` récupère les coordonnées du clic et ici ensuite fait deux choses : elle affiche un petit rectangle à l'endroit du clic et affiche dans la fenêtre du terminal les coordonnées (x, y) .
- Les coordonnées x et y sont exprimées en pixels ; $(0, 0)$ désigne le coin en haut à gauche de la fenêtre (la zone délimitée par `canvas`).

11.5. Mouvement

Voici un programme qui fait se déplacer un petit carré en le faisant rebondir sur les bords de la fenêtre.



Voici les points principaux :

- Un objet `rect` est défini, c'est une variable globale, de même que ses coordonnées `x0`, `y0`.
- Cet objet est (un petit peu) déplacé par la fonction `deplacer()` qui décale le rectangle de `(dx, dy)`.
- Le point clé est que cette fonction sera exécutée une nouvelle fois après un court laps de temps. La commande :

```
canvas.after(50,deplacer)
```

demande une nouvelle exécution de la fonction `deplacer()` après un court délai (ici 50 millisecondes).

- La répétition de petits déplacements simule le mouvement.

```
from tkinter import *
```

```
Largeur = 400
```

```
Hauteur = 200
```

```
root = Tk()
```

```
canvas = Canvas(root, width=Largeur, height=Hauteur, background="white")
```

```
canvas.pack(fill="both", expand=True)
```

```
# Les coordonnées et la vitesse
```

```
x0, y0 = 100,100
```

```
dx = +5 # Vitesse horizontale
```

```
dy = +2 # Vitesse verticale
```

```
# Le rectangle à déplacer
```

```
rect = canvas.create_rectangle(x0,y0,x0+20,y0+20,width=2,fill="red")
```

```
# Fonction principale
```

```
def deplacer():
```

```
    global x0, y0, dx, dy
```

```
    x0 = x0 + dx # Nouvelle abscisse
```

```
    y0 = y0 + dy # Nouvelle ordonnée
```

```
    canvas.coords(rect,x0,y0,x0+20,y0+20) # Déplacement
```

```
    if x0 < 0 or x0 > Largeur:
```

```
        dx = -dx # Changement de sens horizontal
```

```
    if y0 < 0 or y0 > Hauteur:
```

```
        dy = -dy # Changement de sens vertical
```

```
    canvas.after(50,deplacer) # Appel après 50 millisecondes

    return

# Fonction pour le bouton
def action_deplacer():
    deplacer()
    return

# Boutons
bouton_couleur = Button(root,text="Déplacer", width=20, command=action_deplacer)
bouton_couleur.pack(pady=10)

bouton_quitter = Button(root,text="Quitter", width=20, command=root.quit)
bouton_quitter.pack(side=BOTTOM, pady=10)

root.mainloop()
```