UNIVERSITÉ
CÔTE D'AZUR

Programmation impérative en Python

Cours 4. Graphismes, objets et variables globales

Olivier Baldellon

Courriel : prénom.nom@univ-cotedazur.fr

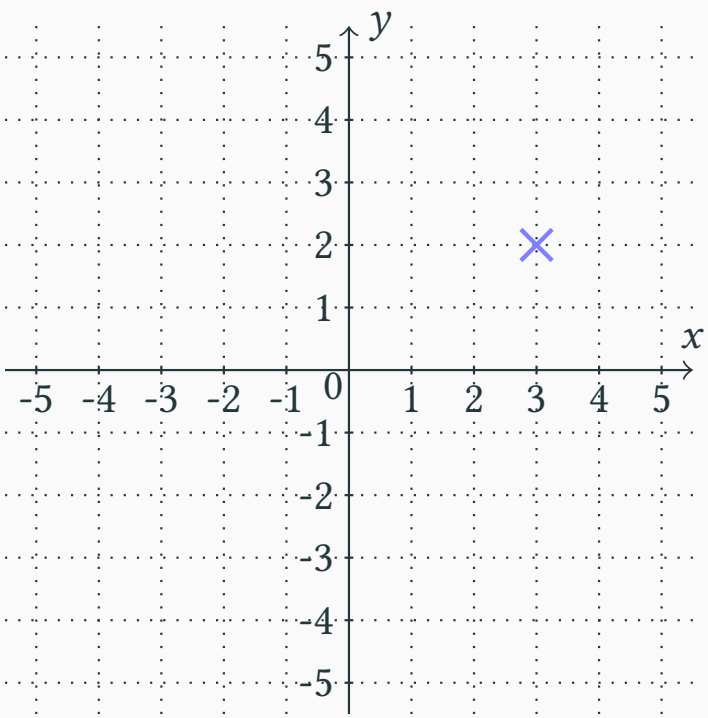
Page professionnelle : https://upinfo.univ-cotedazur.fr/~obaldellon/

Licence 1 — Faculté des sciences et ingénierie de Nice — Université Côte d'Azur

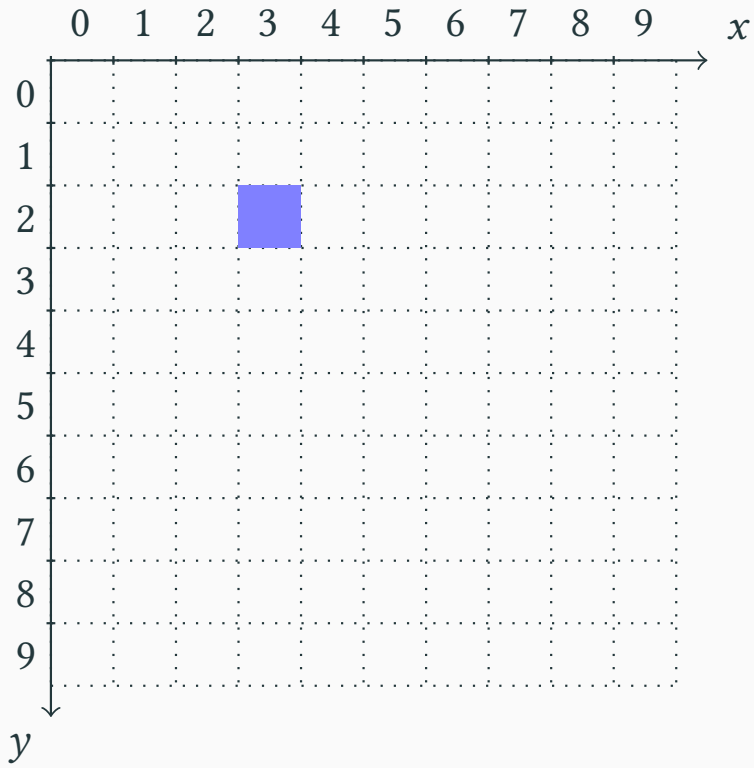
Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

- En informatique, comme en mathématiques, on utilise des coordonnées.
- Il y a cependant trois différences pratiques.
  - L'axes des ordonnées pointe vers le bas.
  - L'origine est en haut à gauche et non au centre.
  - On travaille sur des pixels et non des points (exemple avec le point (3,2)).



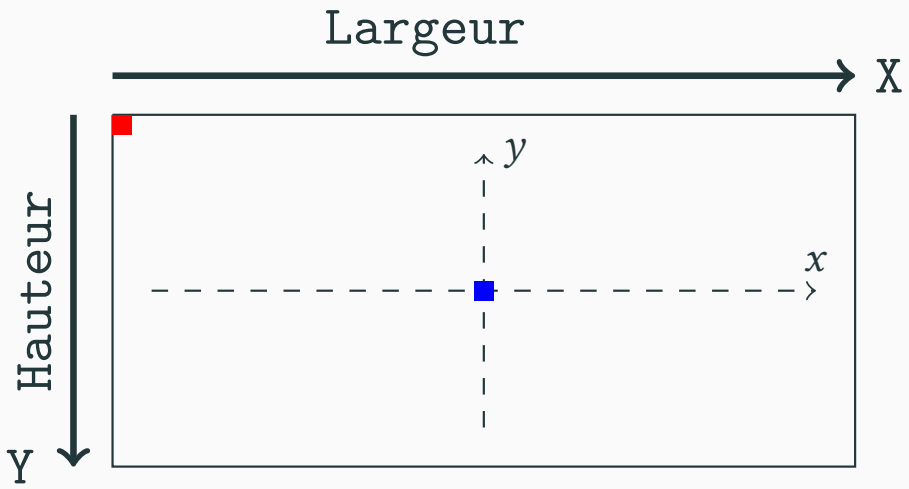
Mathématiques



Informatique

Écrire une fonction de traduction

- La traduction est possible mais souvent inutile.
- Notons (x, y) les coordonnées mathématiques et `(X,Y)` les coordonnées informatiques.



- Quand $(x, y) = (0, 0)$, `(X,Y)=(Largeur//2, Hauteur//2)`
- Quand `(X,Y)=(0,0)` $(x, y) =$ `(-Largeur//2,Hauteur//2)`

- On a ainsi `X` $= x +$ `Largeur//2` et `Y` $= -y +$ `Hauteur//2`

SCRIPT

```
def coordonnées(x,y,Largeur,Hauteur):
    return (x+Largeur//2, -y-Hauteur//2)
```

Les courbes dans le plan sont généralement définies par une équation.

- Cercle : ensemble des points du plan dont la distance à un centre (x_0, y_0) est égale à un rayon R.

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

- Une droite peut s'exprimer par une équation de la forme.

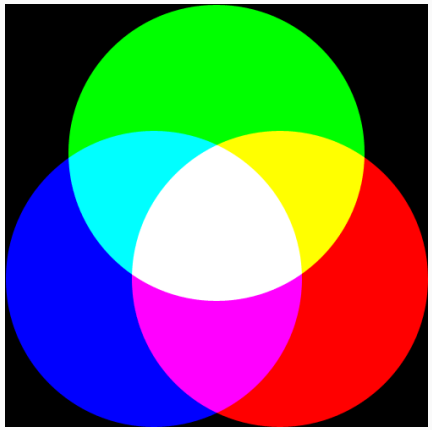
$$ax + by + c = 0$$

- Ces équations ne sont pas utilisables en l'état.
  - il est très rare qu'un pixel aux coordonnées entières satisfasse ces équations
  - Nous verrons aujourd'hui et en TD comment tracer un cercle et un disque.
  - Nous verrons en TP comment tracer un segment de droite.

Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

- En informatique, toutes les couleurs sont codées comme combinaison de trois couleurs : le rouge (`R`), le vert (`G`) et le bleu (`B`).



- Chaque couleur va de 0 à 255.
  - `(255,0,0)` : rouge
  - `(0,255,0)` : vert
  - `(0,0,255)` : bleu
- On obtient les autres couleurs à partir des précédentes.
  - `(255,255,0)` `(R+G)` donne le jaune
  - `(255,0,255)` `(R+B)` donne le magenta
  - `(0,255,255)` `(B+G)` donne le cyan
- Les trois couleurs ensembles donnent le blanc `(255,255,255)`.
- L'absence de couleur donne le noir `(0,0,0)`.

Les tuples, un type composé

- Vous avez noté la présence de *tuples* (*n*-uplets en bon français).
  - Les coordonnées sont définies comme un couple `(x,y)` : un 2-uplet
  - Les couleurs comme un triplet (ou 3-uplet) `(r,g,b)`.

SHELL
```
>>> bleu=(0,0,255)
>>> type(bleu)
<class 'tuple'>
```

- On peut facilement récupérer les composantes d'un tuple.

SHELL
```
>>> (R,G,B)=couleur_mystère
>>> G
123
```

- Permet d'échanger deux variables (sans utiliser de variable auxiliaire) :

SHELL
```
>>> (x,y)=(2,3) # Je définis x et y en une instruction
>>> (x,y)=(y,x) # Je fais l'échange
>>> x
3
>>> y
2
```

Mélangeons les couleurs

- En règle générale, on ne peut pas ajouter des couleurs.
  - Chaque composante doit être infèrieure à 255 : ~~(355,300,455)~~
  - D'ailleurs le + ne fait pas l'addition !

SHELL
```
>>> (255,100,200) + (100, 200, 255)
(255, 100, 200, 100, 200, 255)
```

- Écrivons un programme qui fait la moyenne entre deux couleurs.

SCRIPT
```
def mélange(c1,c2):
    (r1,g1,b1) = c1
    (r2,g2,b2) = c2
    r3=(r1+r2)//2
    g3=(g1+g2)//2
    b3=(b1+b2)//2
    return (r3,g3,b3)
```

SHELL
```
>>> rouge=(255,0,0)
>>> jaune=(255,255,0)
>>> orange=mélange(rouge,jaune)
>>> orange
(255, 127, 0)
```

- Écrire un programme qui mélange deux couleurs selon une proportion p
  - `mélange_p(0.75,c1,c2)` = $0,75 \times$ `c1` + $0,25 \times$ `c2`
  - `mélange_p(p,c1,c2)` = $p \times$ `c1` + $(1-p) \times$ `c2`
  - Remarque : on doit avoir :

```
mélange_p(0.5,c1,c2) == mélange(c1,c2)
```

- Le gris est un mélange entre blanc et noir.
  - `(10,10,10)` gris très foncé
  - `(200,200,200)` gris très clair

- Pour avoir des couleurs plus claires ou plus foncées :
  - Bleu foncé : mélange de bleu et de noir `(0,0,100)`
  - Jaune clair : mélange de jaune et blanc `(255,255,100)`
  - Marron : orange foncé

Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

- On appelle variable locale
  - une variable définie dans le corps d'une fonction.
  - une variable définie comme argument d'une fonction

- Les noms n'ont pas d'importance.

SCRIPT
```
def boucle(n):
    j=0
    while j<n:
        print(j,end='')
        j=j+1
    print('')
```

$\Leftrightarrow$

SCRIPT
```
def boucle(fin):
    k=0
    while k<fin:
        print(k,end='')
        k=k+1
    print('')
```

- Une variable locale n'est utilisable que localement.

SHELL
```
>>> j=17
>>> boucle(3)
012
>>> j
17
```

Variables globales

- Une variable en dehors de toute fonction est globale. Pour la modifier dans une fonction, il faut la déclarer explicitement.

SCRIPT

```
def boucle(n):
    global j # <- Je peux modifier un j défini ailleurs
    print('j =',j,end=' : ')
    j=0
    while j<n:
        print(j,end='')
        j=j+1
```

- Contrairement aux variables locales, les noms sont importants.

SHELL

```
>>> j=17
>>> boucle(3) # On modifie j
j = 17 : 012
>>> j
3
```

- Variable globale : à éviter sauf cas exceptionnels.
  - paramètre global accessible de partout (ok)
  - paramètre modifié par toutes les fonctions : non !
  - Par défaut, les variables sont locales

- On peut définir une fonction dans une fonction

SCRIPT
```
def max3(a,b,c):
    def max2(x,y):
        if x>y: return x
        else: return y
    return max2(max2(a,b),c)
```

- Mais la fonction locale ne peut pas être utilisée ailleurs.

SHELL
```
>>> max3(17,-5,133)
133
>>> max2(3,4)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError:  name 'max2' is not defined
```

- Difficile à lire. Privilégiez des fonctions locales sur 1 ou 2 lignes.

SCRIPT
```
def cercle(t):
    def X(t): return 100*cos(t/10)
    def Y(t): return 100*sin(t/10)
    return (X(t),Y(t))
```

Il y a deux types de programme.

- Les fonctions sont des programmes qui renvoient (`return`) une valeur et ne modifient pas l'état.
  - semblables aux fonctions mathématiques
  - utilisées lorsqu'on doit faire un calcul

- Les procédures sont des programmes qui ne renvoient aucune valeur mais modifient l'état du système. L'état peut être :
  - L'affichage sur l'écran (`print`) ;
  - la sauvegarde d'un fichier sur le disque ;
  - la modification d'une fenêtre graphique.
  - la modification de variables globales

- On peut modifier l'état et renvoyer une valeur en même temps.
  - À éviter sans bonnes raisons.
  - Il est bien de séparer les parties qui font un calcul ou un algorithme…
  - … des parties du code qui interagissent (`input`, `print`, etc.)

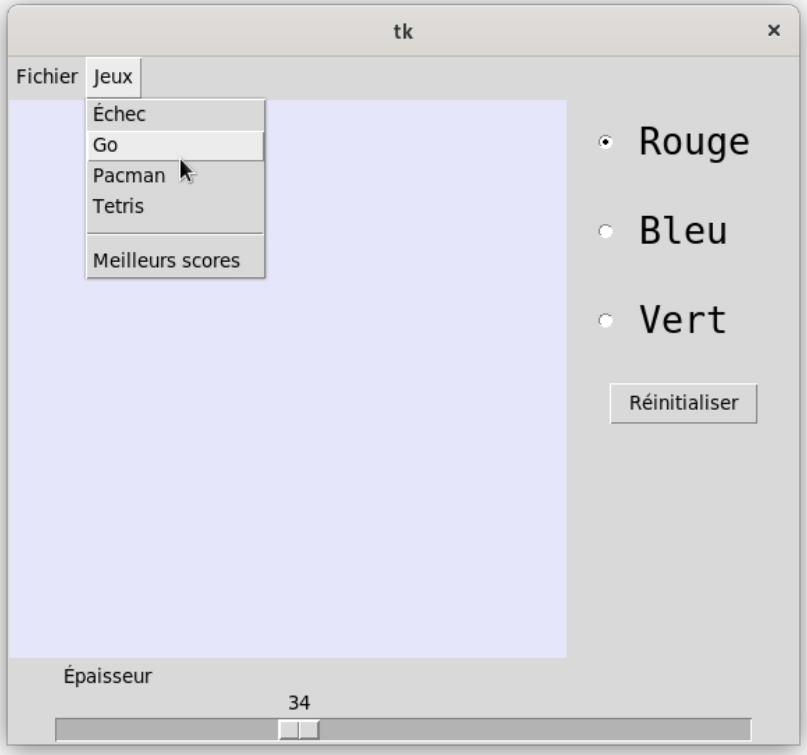
Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

Tk est une bibliothèque graphique disponible sous de nombreux langages

- Sous Python, c'est le module tkinter
- Disponible sur GNU/Linux, Windows et Mac

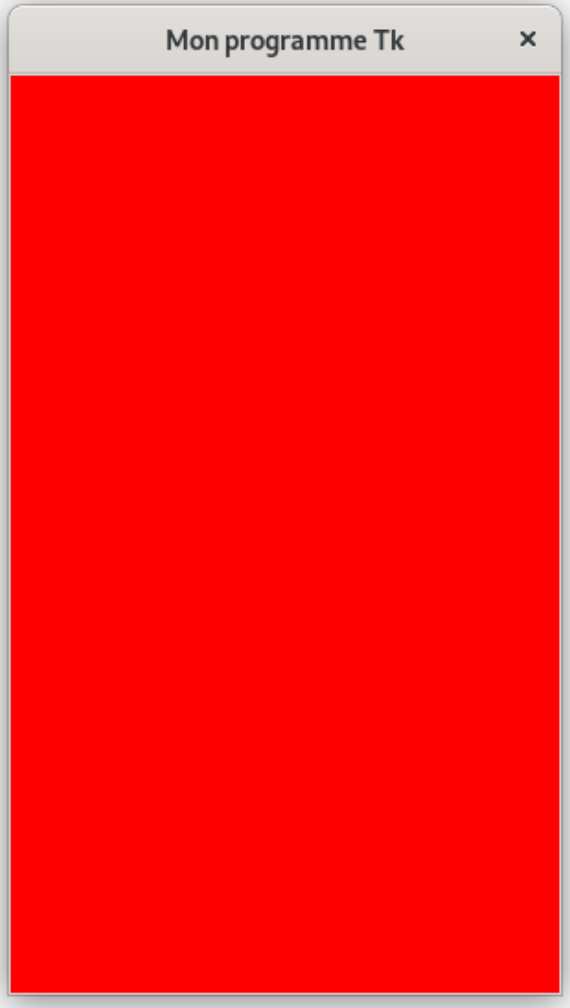
- Dessin
  - C'est l'objectif de cette semaine
  - Créer et sauvegarder des images
- Animation
  - Séquence d'images
  - Plus tard dans le semestre
- Interface graphique complète
  - Gestion des évènement
  - Gestion du clavier et de la souris.
  - Jeux
  - Menu, boutons, cases à cocher
  - Plus tard dans le semestre



Mon premier programme

SCRIPT

```
import tkinter as tk

# On crée une fenêtre
root = tk.Tk()
root.title("Mon programme Tk")

# On crée un canvas (zone de dessin)
Hauteur=700
Largeur=300

# Pour raccourcir la ligne suivante
(H,L,c)=(Hauteur,Largeur,'red')
Dessin=tk.Canvas(root,height=H,width=L,bg=c)
Dessin.pack()
```

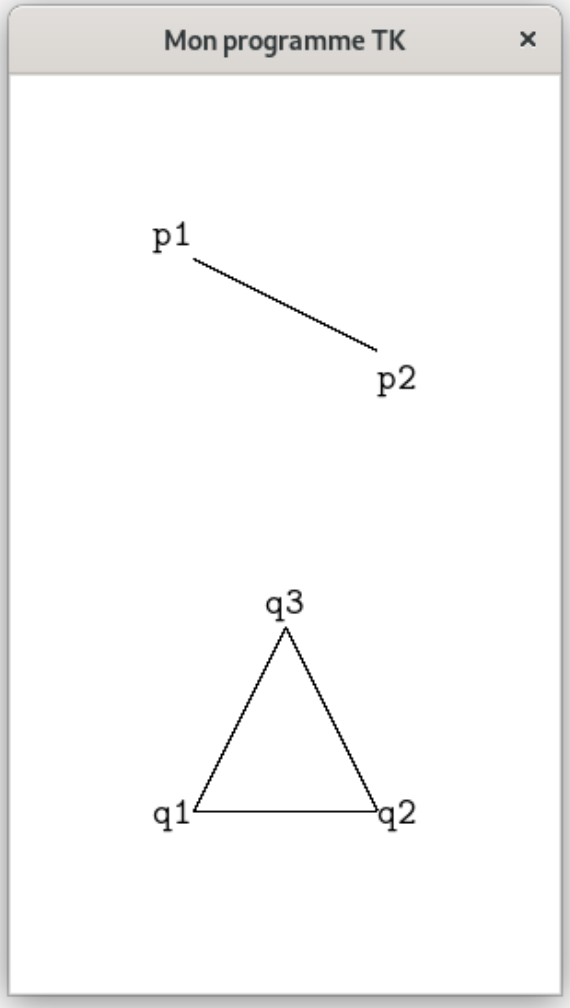


- Il faudra bien retenir le nom du *canvas* (ici `Dessin`)
  - Ce nom sera nécessaire pour appeler des fonctions
  - On l'utilisera comme variable globale.
  - `Hauteur` et `Largeur` sont aussi globales
- Votre script doit se terminer par `root.mainloop()`

- À la suite du code précédent

SCRIPT

```
p1=(100,100)
p2=(200,150)

Dessin.create_line(p1,p2)

q1 = (100,400)
q2 = (200,400)
q3 = (150,300)

Dessin.create_line(q1,q2)
Dessin.create_line(q2,q3)
Dessin.create_line(q3,q1)
```



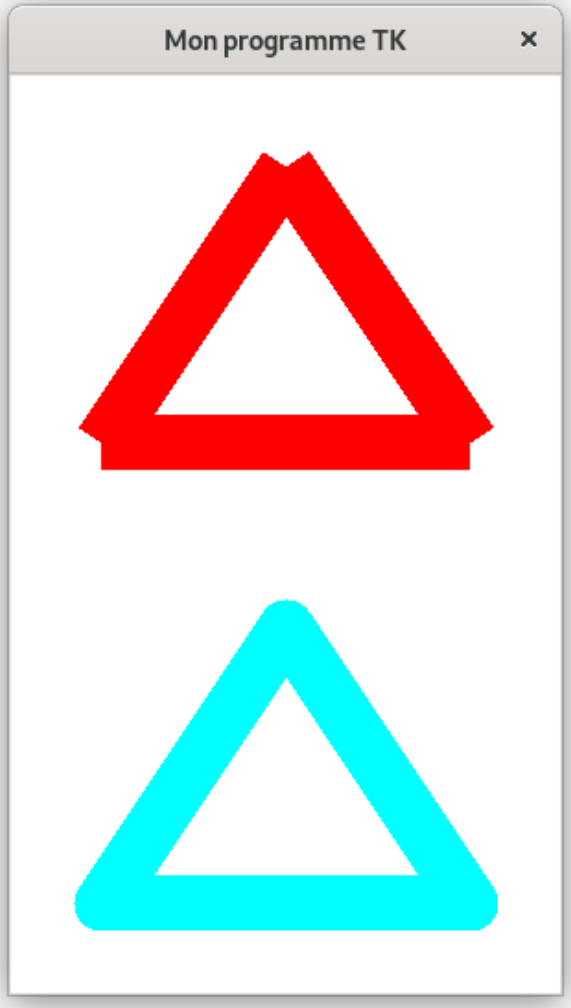
- Les noms de point ne s'affichent normalement pas !
- Je les ai rajoutés pour vous aider.

Tracer des segments moins moches

- On peut ajouter des options
  - pour la couleur `fill=...`
  - pour l'épaisseur `width=...`
  - pour la forme des bords `cap=...`

SCRIPT

```
p1 = ( 50,200)
p2 = (250,200)
p3 = (150,50)

Dessin.create_line(p1,p2,width=30,fill='red')
Dessin.create_line(p2,p3,width=30,fill='red')
Dessin.create_line(p3,p1,width=30,fill='red')

q1 = ( 50,450)
q2 = (250,450)
q3 = (150,300)

w=30
f='cyan'
c='round'
Dessin.create_line(p,q,width=w,fill=f,cap=c)
Dessin.create_line(q,r,width=w,fill=f,cap=c)
Dessin.create_line(r,p,width=w,fill=f,cap=c)
```

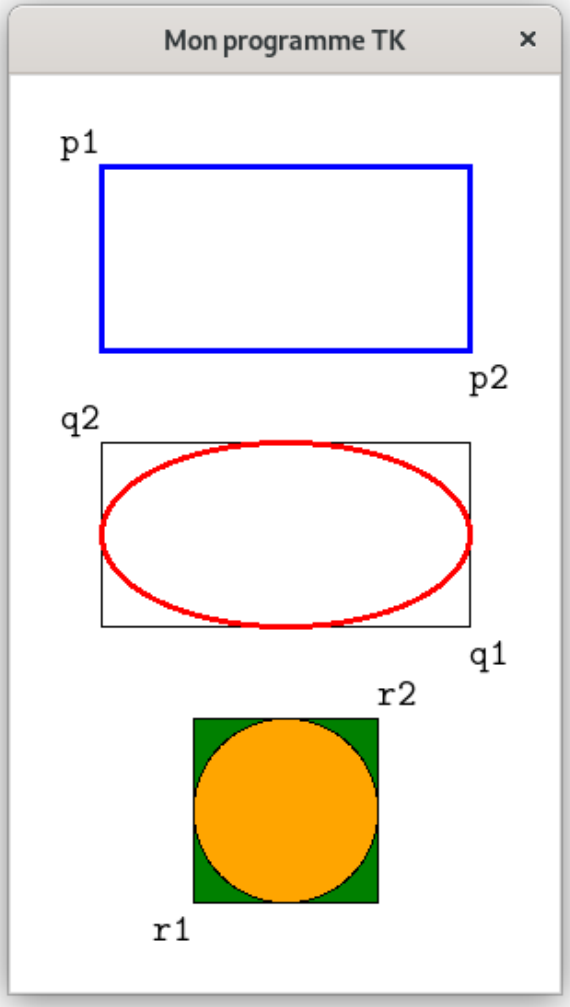


Tracer des ellipses et des rectangles

- On trace un rectangle à partir de 2 points diagonales
- On trace une ellipse à partir du rectangle la contenant

SCRIPT

```
p1 = ( 50,50)
p2 = (250,150)
b='blue'
Dessin.create_rectangle(p1,p2,width=3,outline=b)

q1 = (250,300)
q2 = ( 50,200)
Dessin.create_rectangle(q1,q2)
Dessin.create_oval(q1,q2,width=3,outline='red')

r1 = (100,450)
r2 = (200,350)
# Carré
Dessin.create_rectangle(r1,r2,fill='green')
# Cercle
Dessin.create_oval(r1,r2,fill='orange')
```



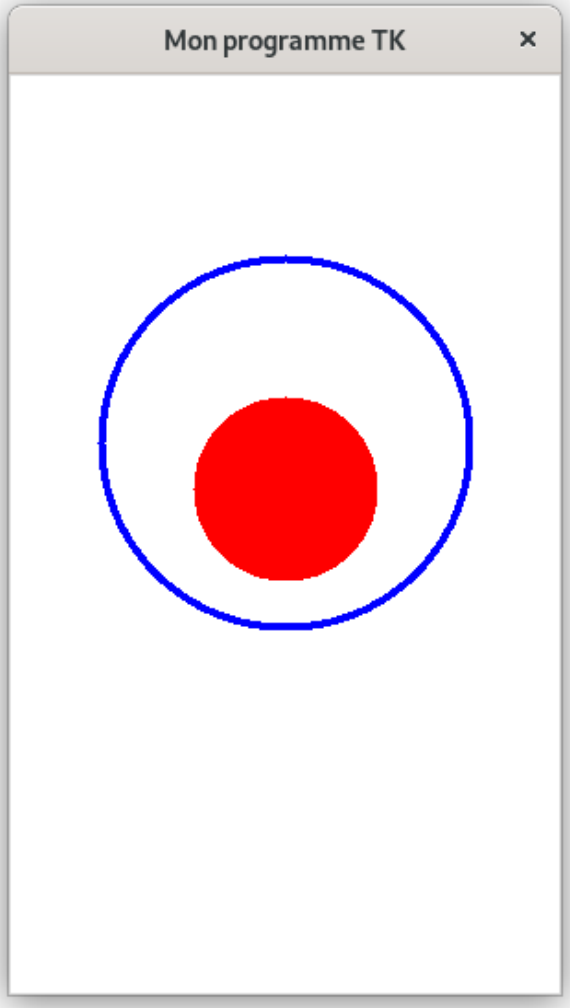
- On peut choisir la couleur pour :
  - tracer : option `outline=...`
  - remplir : option `fill=...`
- Les noms de point ne s'affichent normalement pas !

Écrire sa propre fonction de tracer de cercle

- On souhaite une fonction plus simple à utiliser :
  - pour les cercles
  - pour les disques

SCRIPT

```
def cercle(x,y,rayon,épaisseur,couleur):
    p1 = (x-rayon,y-rayon)
    p2 = (x+rayon,y+rayon)
    (w,c) = (épaisseur,couleur)
    Dessin.create_oval(p1,p2,width=w,outline=c)

def disque(x,y,rayon,couleur):
    p1 = (x-rayon,y-rayon)
    p2 = (x+rayon,y+rayon)
    Dessin.create_oval(p1,p2,width=0,fill=couleur)

épaisseur=4
cercle(150,200,100,épaisseur,'blue')
disque(150,225,50,'red')
```



- Rappel : un octet (0…255) correspond à deux chiffres hexadécimaux.

Chiffre hexa.	0	1	2	…	9	A	B	C	D	E	F
nombre (base 10)	0	1	2	…	9	10	11	12	13	14	15

- Conversion :
  - `216//16` donne `D` (13) et `216%16` donne `8`
  - 216 correspond à `D8`.

- Le calcul dans l'autre sens est tout aussi rapide :
  - `D8` $= 13 \times 16 + 8 = 216$

- Une couleur sous Tk se note `'#RRGGBB'` où `RR`, `GG` et `BB` sont les écritures hexadécimales des trois composantes.

- De nombreuses couleurs sont cependant prédéfinies sous Tk sous forme de chaîne de caractère. (`'red'`, `'green'`, `'orange'`, etc)

- À quoi correspond la couleur suivante : `'#00FF00'` ?

```shell
>>> 15*16 + 15 # FF
255
```

  - R=0
  - G=255
  - B=0
  - C'est du vert !

- Comment coder le orange (255,126,0) en hexadécimale ?

```shell
>>> 126//16
7
>>> 126 % 16
14
```

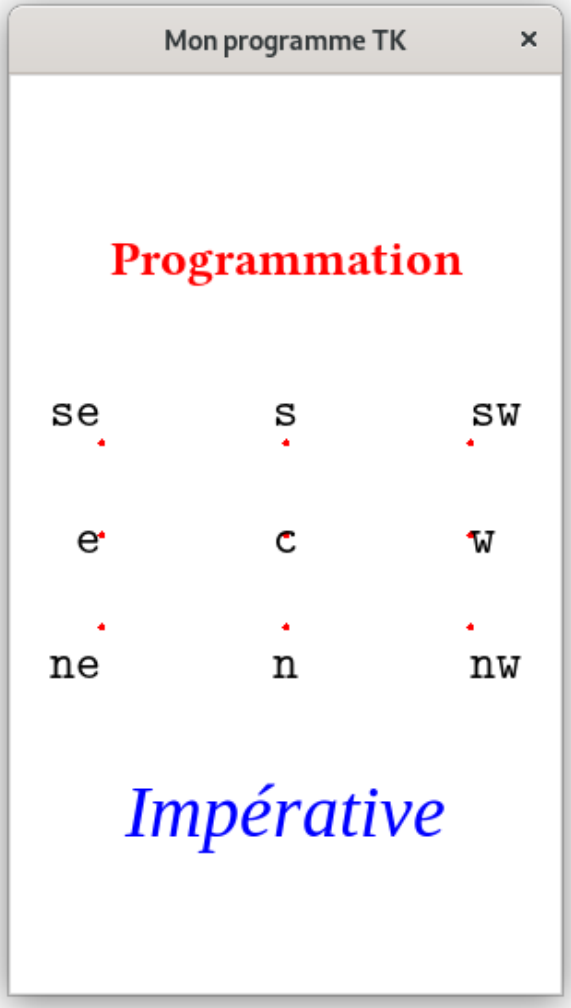
  - `'#FF7E00'`

Afficher un texte

- Pour afficher du texte il y a de nombreuses options
  - le texte (`text=...`)
  - la police (`font=...`)
  - ancrage (`anchor=...`)
  - la couleur (`fill=...`)

SCRIPT

```
#  À lire calmement chez soi.

def texte(x,y,a):
    f ="LatinModernMono 20"; p=(x,y); rayon=2
    p1 = (x-rayon,y-rayon); p2 = (x+rayon,y+rayon)
    Dessin.create_oval(p1,p2,width=0,fill='red')
    # La ligne importante
    Dessin.create_text(p,text=a,anchor=a,font=f)

texte(50,200,"se")
# [...] on ne va pas tous les mettre
texte(250,300,"nw")

def texte2(x,y,t,f,c):
    # Par défaut anchor='c'
    Dessin.create_text((x,y),text=t,font=f,fill=c)

f1="LinuxLibertine 20 bold" # bold = « Gras»
f2="LiberationSerif 20 italic"
texte2(150,100,"Programmation",f1,'red')
texte2(150,400,"Impérative",f2,'blue')
```



- Durant les cours, TD, TP et contrôle nous n'utiliserons que 4 primitives.
- `Dessin.create_line(p,q)`
  - option `fill=` pour la couleur
- `Dessin.create_rectangle(p,q)`
  - option `fill=` pour le couleur de remplissage
  - option `outline=` pour le couleur du bord
- Les fonctions `disque` et `cercle` que vous devez définir

SCRIPT

```
def cercle(x,y,rayon,épaisseur,couleur):
    p = (x-rayon,y-rayon)
    q = (x+rayon,y+rayon)
    Dessin.create_oval(p,q,width=épaisseur,outline=couleur)

def disque(x,y,rayon,couleur):
    p = (x-rayon,y-rayon)
    q = (x+rayon,y+rayon)
    Dessin.create_oval(p,q,width=0,fill=couleur)
```

- Votre code doit terminer par `root.mainloop()` !!!

- Nous n'avons vu qu'une partie des possibilités offertes par Tk
- Premier réflexe : regarder la documentation officielle.
  - Problème : il n'y en a pas !
- Il existe un excellent site sur lequel je me suis beaucoup basé.
  - il a de plus l'avantage d'être francophone !
  - écrit sous forme de tutoriel
  - documentation du canvas Tkinter (cours d'aujourd'hui) :

`http://pascal.ortiz.free.fr/contents/tkinter/tkinter/le_canevas`

- Le site décrit Tkinter en général :

`http://pascal.ortiz.free.fr/contents/tkinter/tkinter/`

- Au format pdf :

`http://pascal.ortiz.free.fr/contents/tkinter/tkinter/tkinter.pdf`

- Une sauvegarde sur mon site (légale car sous license CC-BY)

`https://upinfo.univ-cotedazur.fr/~obaldellon/L1/py/tkinter.pdf`

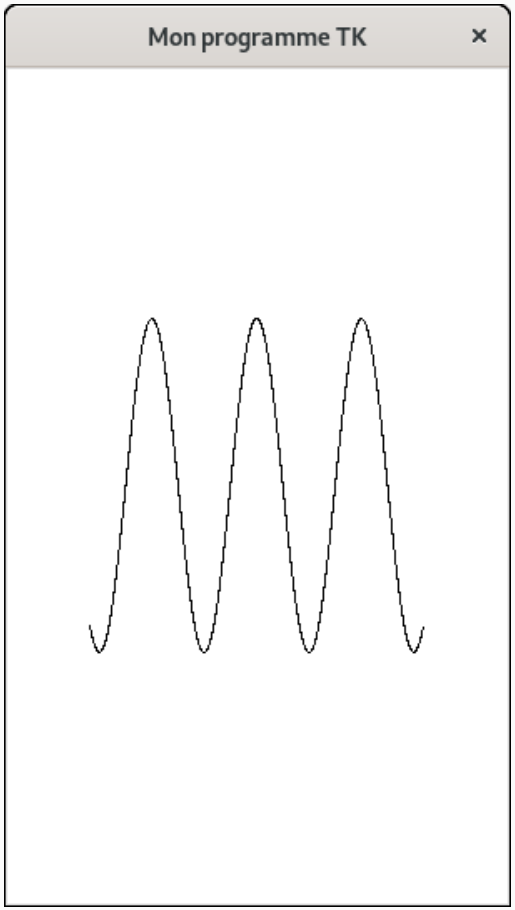
Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

- On fait varier `x` de `xmin` à `xmax` par pas de `dx`.
- Et on relie les points de coordonnée `(x,f(x))`
- Le point suivant `(x,f(x))` étant `(x+dx,f(x+dx))`

SCRIPT

```
def ligne(p,q):
    p1 = coordonnées(p)
    q1 = coordonnées(q)
    Dessin.create_line(p1,q1)

def trace_fonction(f, xmin, xmax, dx):
    (x,y) = (xmin,f(xmin))
    while x < xmax:
        p = (x+dx,f(x+dx))
        ligne((x,y),p)
        (x,y) = p
```



- Par exemple, on peut tracer une sinusoïde.

SCRIPT

```
def sinusoïde(x):
    return 100 * cos(x/10)
# On lance le script
trace_fonction(sinusoïde,-200,200,1.0)
```

SCRIPT

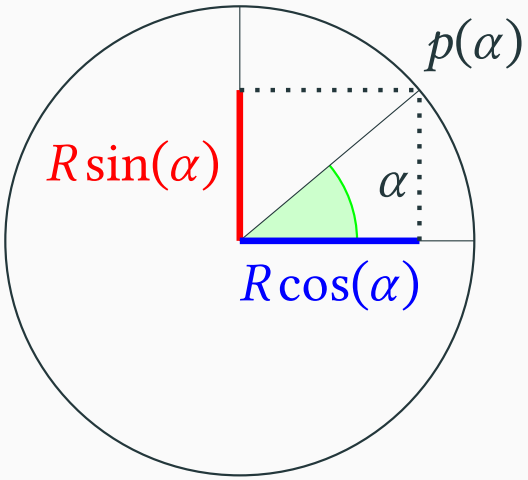
```
from math import * # cos, sin, pi
import tkinter as tk
root = tk.Tk() # On crée la  fenêtre et le canvas (zone de dessin)
root.title("Mon programme Tk")
(Larg,Haut)=(300,500) # Variables globales
Dessin=tk.Canvas(root,height=Haut,width=Larg,bg='white')
Dessin.pack()

def coordonnées(p):
    (x,y)=p
    return (x+Larg//2, -y+Haut//2)

def ligne(p,q):
    p1 = coordonnées(p)
    q1 = coordonnées(q)
    Dessin.create_line(p1,q1)

def trace_fonction(f, xmin, xmax, dx):
    (x,y) = (xmin,f(xmin))
    while x < xmax:
        p = (x+dx,f(x+dx))
        ligne((x,y),p)
        (x,y) = p

def sinusoïde(x): return 100*cos(x/10)

trace_fonction(sinusoïde,-100,100,1.0)
```

- Pour tracer une courbe on exprime généralement y en fonction de x
- On peut aussi exprimer x et y en fonction du temps t : $p(t) = (x(t), y(t))$
- Lorsque t varie, $p(t)$ décrit une courbe : une *courbe paramétrique*.
- Pour tracer une telle courbe, il suffit de relier les points $p(t)$ avec $p(t+dt)$.
- On a besoin de t_{min} (le début), de t_{max} (la fin) et de dt (le pas).

SCRIPT

```
def trace_paramétrique(tmin,tmax,fx,fy,dt):
    """ Trace la courbe d'équation x=fx(t) y=fy(t)
    avec tmin <= t < tmax et un pas dt
    """
    def p(t): return (fx(t),fy(t))
    t=tmin
    while t<tmax:
        ligne(p(t),p(t+dt))
        t=t+dt
```

- Utilisons notre fonction pour tracer un cercle !

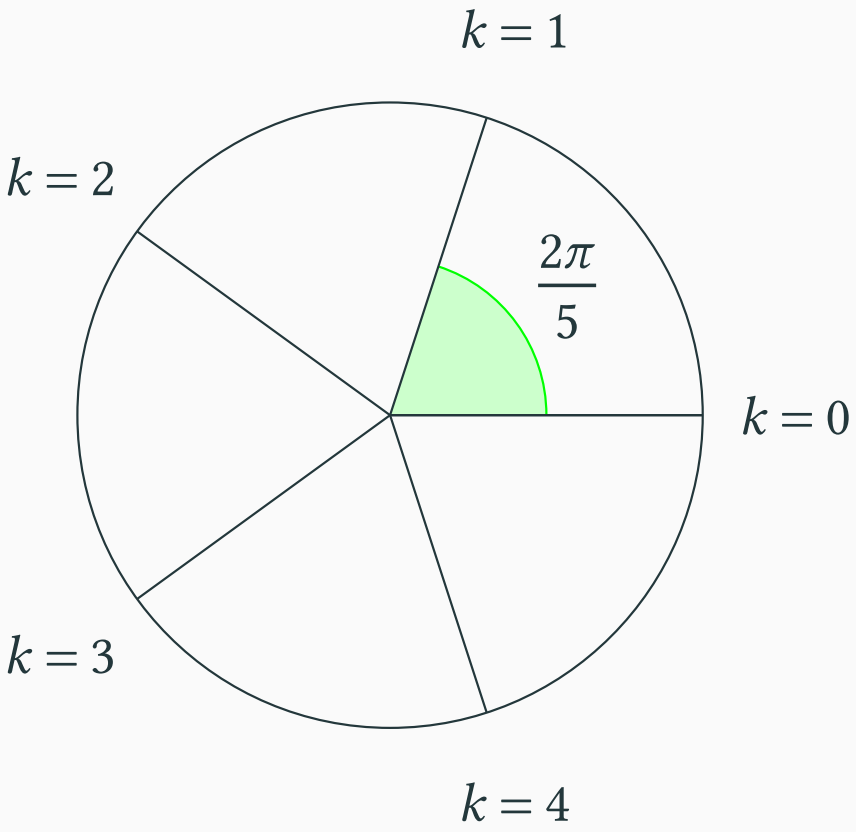


$$\begin{cases} x(t) = x_0 + R\cos(t), \\ y(t) = y_0 + R\sin(t). \end{cases}$$

SCRIPT

```
def tracer_cercle(centre,rayon):
    (x0,y0) = centre
    def fx(t): return x0+rayon*cos(t)
    def fy(t): return y0+rayon*sin(t)
    trace_paramétrique(0, 2*pi, fx, fy, 0.01)
    # pi est défini dans la bibliothèque math
```

- Un polygone régulier est juste un cercle avec une mauvaise résolution !
- Pour un polygone à n côtés, le pas est $\frac{2\pi}{n}$

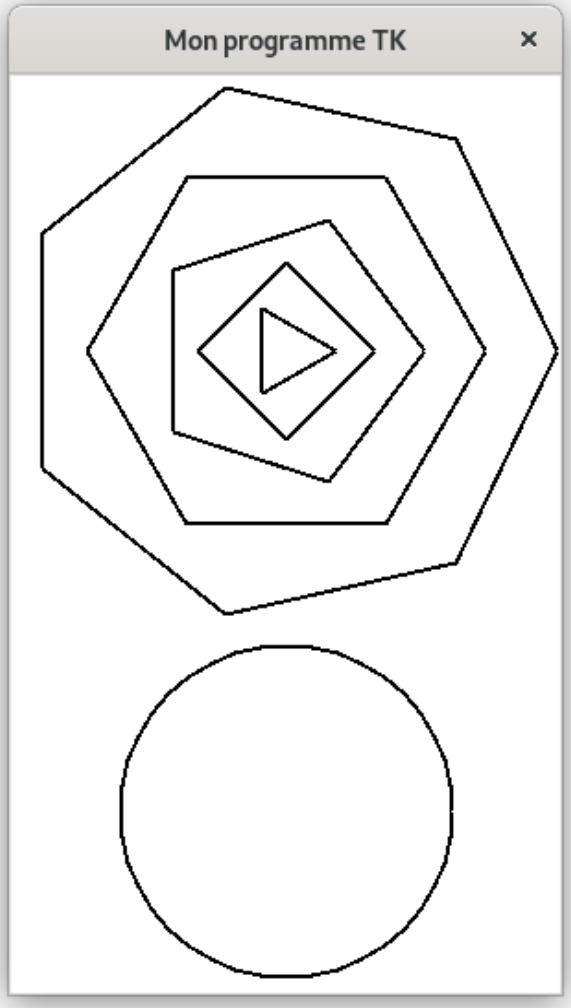


$$\begin{cases} x(k) = x_0 + r\cos\left(\frac{2k\pi}{n}\right), \\ y(k) = y_0 + r\sin\left(\frac{2k\pi}{n}\right). \end{cases}$$

- Mise en pratique !

SCRIPT

```
def point(x0,y0,R,k,n):
    x = x0 + R * cos(2*k*pi/n)
    y = y0 + R * sin(2*k*pi/n)
    return (x,y)

def polygone(x0,y0,R,n):
    def p(k): return point(x0,y0,R,k,n)
    for k in range(n):
        Dessin.create_line(p(k),p(k+1))

for i in range(3,8):
    polygone(150,200,3*i**2,i)

# cercle : polygone a beaucoup de côtés
polygone(150,450,100,40) #  beaucoup = 40
```



- À contrario, un cercle est un polygone à beaucoup de côtés.

Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

Rappel : les types en Python

- En Python, tous les éléments ont un type

SHELL
```
>>> type(4/2)
<class 'float'>
>>> type(4//2)
<class 'int'>
>>> type(max)
<class 'builtin_function_or_method'>
>>> type((1,2,3))
<class 'tuple'>
```

- Vraiment tous les éléments

SHELL
```
>>> root = tk.Tk()
>>> type(root)
<class 'tkinter.Tk'>
>>> type(print("Coucou"))
Coucou
<class 'NoneType'>
>>> type(type(3))
<class 'type'>
```

- En Python, les types sont appelés classes.
- Python est un langage objet.

Un premier exemple : les étudiants

- Créons notre propre type.
- Nous voulons créer un type `Étudiant`
- Chaque étudiant possède plusieurs caractéristiques
  - Nom
  - Prénom
  - numéro d'étudiant
  - etc.
- Nous voulons définir un certain nombre de fonction pour ce type.
  - Par exemple afficher le mail

- Le premier réflèxe serait d'utiliser des uplets.

SCRIPT
```
étudiant1 = ('Nicolas','Bourbaki')
étudiant2 = ('Albert','Mousquetaire')

def mail(étudiant):
    (prénom,nom)=étudiant
    return prénom + '.' + nom + '@etu.univ-cotedazur.fr'
```

SHELL
```
>>> mail(étudiant1)
'Nicolas.Bourbaki@etu.univ-cotedazur.fr'
```

- Mais si on souhaite ajouter des informations, il faut réécrire toutes les fonctions.

SCRIPT
```
étudiant1 = ('nb655957','Nicolas','Bourbaki')
étudiant2 = ('am314151','Albert','Mousquetaire')

def mail(étudiant):
    (id,prénom,nom)=étudiant # ligne modifiée
    return prénom + '.' + nom + '@etu.univ-cotedazur.fr'
```

- On va plutôt créer une classe `Étudiant`

SCRIPT

```
class Étudiant:

    def __init__(self,id,prénom,nom):
        print("Création d'un nouvel étudiant")
        self.id = id
        self.prénom = prénom
        self.nom = nom

# On n'est plus dans la classe
def mail(quelqu_un):
    p = quelqu_un.prénom
    n = quelqu_un.nom
    return p + '.' + n + '@etu.univ-cotedazur.fr'
```

- L'utilisation est plutôt simple.

SHELL

```
>>> nico = Étudiant('nb655957','Nicoca', 'Bourbaki')
Création d'un nouvel étudiant
>>> mail(nico)
'Nicoca.Bourbaki@etu.univ-cotedazur.fr'
>>> nico.prénom='Nicolas'
>>> mail(nico)
'Nicolas.Bourbaki@etu.univ-cotedazur.fr'
```

- Une classe est un type (ici `Étudiant`)
- les éléments d'une classe sont appelés objets
- Un objet contient des variables : les attributs.
  - Ici il y a deux attributs prénom et nom

SCRIPT
```
class Étudiant:
    def __init__(self,prénom,nom):
        self.prénom = prénom
        self.nom = nom

    def courriel(self):
        domaine = '@etu.univ-cotedazur.fr'
        return self.prénom + '.' + self.nom + domaine
```

- Lors de la définition de la classe, ils sont précédés de `self`.
- En dehors de la classe, ils sont précédés du nom de l'objet

SHELL
```
>>> nico=Étudiant('Nicolas','Bourbaki') # nico : objet
>>> nico.nom # nom : attribut de nico
'Bourbaki'
```

- On appelle méthode une fonction définie dans une classe.

SCRIPT
```
class Étudiant:
    def __init__(self,prénom,nom): # Première méthode
        print("Création d'un objet de la classe Étudiant")
        self.prénom = prénom ; self.nom = nom

    def courriel(self,domaine):  # Seconde méthode
        return self.prénom + '.' + self.nom + domaine
```

- `__init__` : méthode particulière appelée lors de la création de l'objet

SHELL
```
>>> nico=Étudiant('Nicolas','Bourbaki')
Création d'un objet de la classe Étudiant
```

- Lors de leur définition, le premier argument est toujours `self`
  - `self` représente l'objet
  - L'appel de méthode se fait après le nom de l'objet : `nico.courriel(…)`
  - Les méthodes sont appelées sans l'argument `self`

SHELL
```
>>> nico.courriel('@etu.univ-cotedazur.fr')
'Nicolas.Bourbaki@etu.univ-cotedazur.fr'
```

- Sur le même modèle créons une classe `Enseignant`

SCRIPT

```
class Étudiant:
    def __init__(self,id,prénom,nom):
        self.id = id
        self.prénom = prénom
        self.nom = nom

    def courriel(self):
        domaine = '@etu.univ-cotedazur.fr'
        return self.prénom + '.' + self.nom + domaine


class Enseignant:
    def __init__(self,id,prénom,nom):
        self.id = id
        self.prénom = prénom
        self.nom = nom

    def courriel(self):
        domaine = '@univ-cotedazur.fr'
        return self.prénom + '.' + self.nom + domaine
```

- Pourquoi utiliser une méthode plutôt qu'une fonction ?

SHELL
```
>>> nico = Étudiant('nb655957','Nicolas', 'Bourbaki')
>>> moi = Enseignant('obaldellon','Olivier','Baldellon')
>>> type(moi)
<class '__console__.Enseignant'>
>>> type(nico)
<class '__console__.Étudiant'>
```

- `nico` et `moi` ne sommes pas du même type.
- Et pourtant l'utilisateur utilise la même méthode.

SHELL
```
>>> print(moi.courriel()) ; print(nico.courriel())
Olivier.Baldellon@univ-cotedazur.fr
Nicolas.Bourbaki@etu.univ-cotedazur.fr
```

La même ? Pas tout à fait.

  - même nom
  - mais ce sont bien deux méthodes distinctes
  - permet le polymorphisme (code fonctionnant avec différents types)

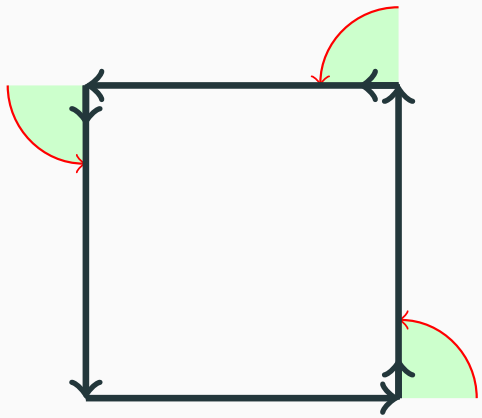
- Difficile de faire du polymorphisme avec des fonctions classiques.

Sommaire

- Partie I. Géométrie cartésienne
- Partie II. Couleurs et tuples
- Partie III. Variables locales/globales
- Partie IV. Dessiner avec Tk
- Partie V. Algorithmes de tracer de figure
- Partie VI. Créer son propre type
- Partie VII. Exemples
- Partie VIII. Table des matières

Une tortue géomètre

- Pour l'instant, on n'utilise que des coordonnées absolues.
  - Pratique et efficace
  - Mais nécessite de calculer les coordonnées à l'avance.
- On veux se déplacer de manière relative :
  - Avancer, Reculer
  - Tourner à gauche ou à droite
- Correspond historiquement à la bibliothèque Turtle.
  - Avance
  - Tourne de 90°
  - Avance
  - Tourne de 90°
  - Avance
  - Tourne de 90°
  - Avance



SCRIPT

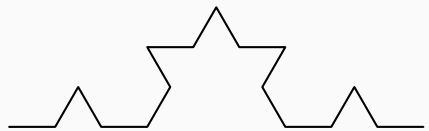
```
from math import cos, sin, pi

class Crayon:

    def __init__(self,canvas):
        self.x = 0 ; self.y = 0 ; self.a = 0 # angle
        self.couleur='black'
        self.cv = canvas

    def déplace(self,p):
        (self.x,self.y) = p

    def avance(self,r):
        p = (self.x,self.y)
        self.x = self.x + r*cos(self.a)
        self.y = self.y + r*sin(self.a)
        q = (self.x,self.y)
        self.cv.create_line(p,q,fill=self.couleur)

    def tourne(self,alpha):
        # On convertit alpha des degrés aux radians
        # On prend l'opposé, pour garder le même sens
        #    qu'en math (sens trigo et non horaire).
        self.a = self.a  - 2*pi*alpha/360
```

La suite (K_n) des courbes de Koch de base T est construite de proche en proche (par récurrence).

- K_0 est un segment de longueur T.

- K_1 est obtenue par chirurgie à partir de K_0.

- K_2 est obtenue par la même chirurgie appliquée à chaque côté de K_1.



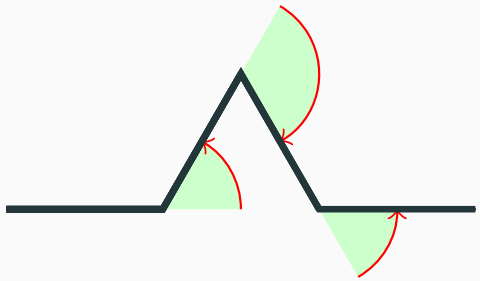
- K_3 est obtenue par la même chirurgie appliquée à chaque côté de K_2.

- À chaque étape, tous les segments ont la même longueur.

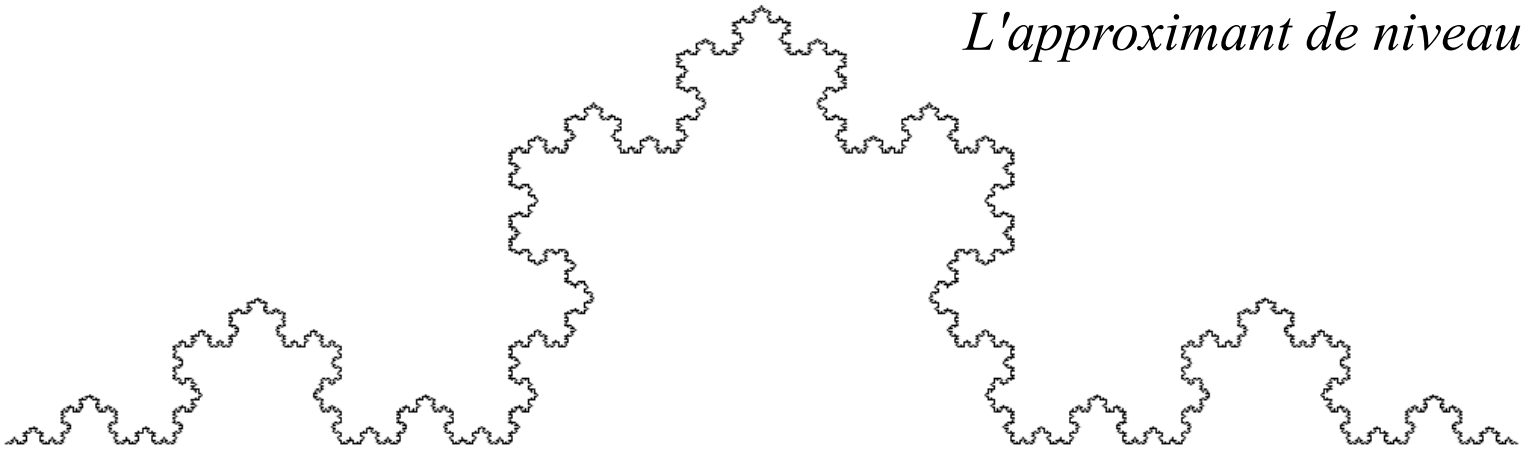
- Mathématiquement, la courbe K_n s'obtient comme assemblage de quatre courbes K_{n-1}. On peut donc la dessiner par récurrence sur n.

SCRIPT

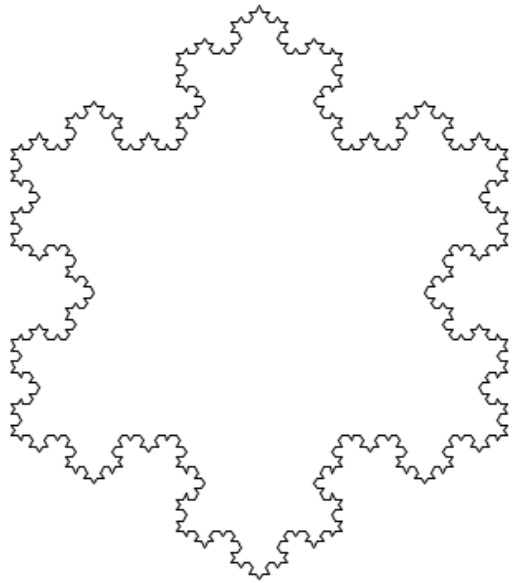
```
def koch(n, T):
    """ approximant de la courbe
    de Koch de niveau n et de
    base T """
    if n == 0:
        crayon.avance(T)
    else:
        koch(n - 1, T / 3)
        crayon.tourne(60)
        koch(n - 1, T / 3)
        crayon.tourne(-120)
        koch(n - 1, T / 3)
        crayon.tourne(60)
        koch(n - 1, T / 3)
```



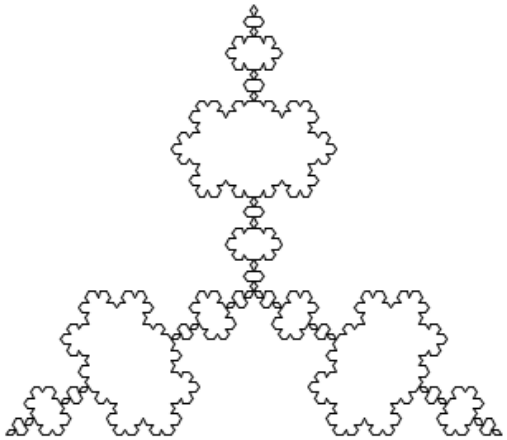
- La courbe de Koch K est la limite des courbes K_n quand $n \to \infty$.
- Découverte en 1904 par le mathématicien suédois Helge von Koch. La courbe K est fractale, continue, mais sans tangente en aucun point.



L'approximant de niveau 6



Le flocon de Von Koch



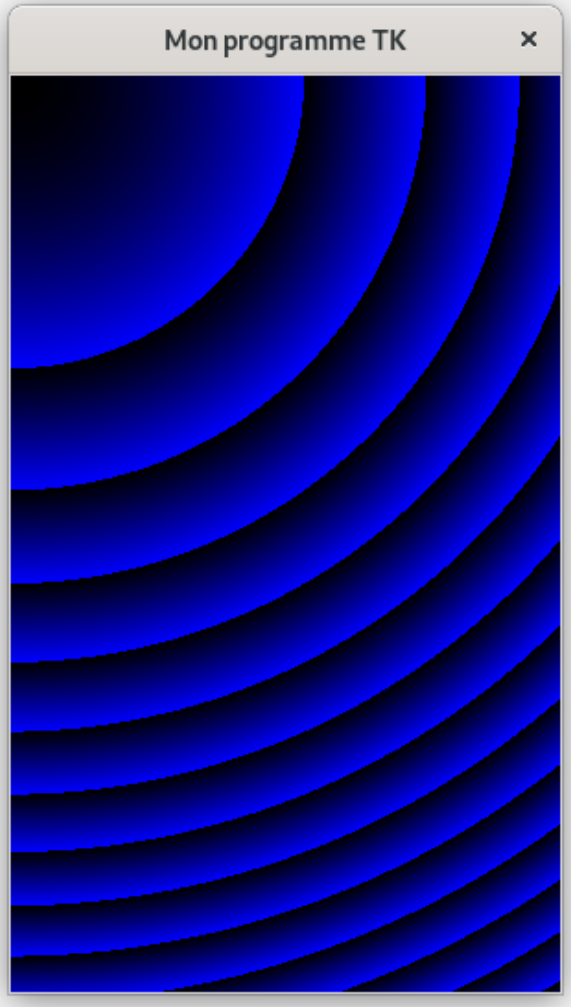
L'antiflocon

Exemple 1 : tracer de pixels

- On souhaite parfois tracer une figure pixel par pixel.
  - On peut facilement créer une fonction `pixel`.
  - Permet de faire de jolis dégradés !

SCRIPT

```
def couleur(r,g,b):
    d = '0123456789ABCDEF'
    rr = d[r//16]+d[r%16]
    gg = d[g//16]+d[g%16]
    bb = d[b//16]+d[b%16]
    return '#' + rr + gg + bb

def pixel(x,y,R,G,B):
    c = couleur(R,G,B)
    p = (x,y)
    # Un pixel est un petit rectangle de côté 1
    Dessin.create_rectangle(p,p,fill=c,outline='')

for i in range(0,300,1): # pour chaque colonne
    for j in range(0,500,1): # pour chaque ligne
        b = ((i**2+j**2)//100)%256
        # La clarté du bleu est proportionnelle
        # au carré de la distance à l'origine.
        # Et le tout modulo 256.
        pixel(i,j,0,0,b)
```

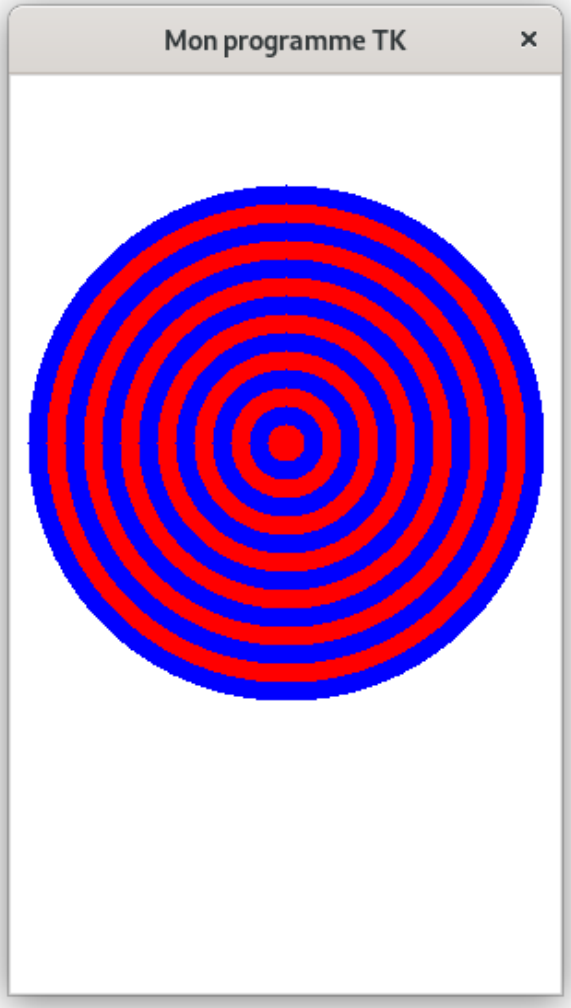


Exemple 2 : tracer une cible

- On commence à tracer le grand disque extérieur bleu ;
- on continue avec le grand disque rouge intérieur ;
- et on répète avec des disques de plus en plus petits.

SCRIPT

```
for i in range(13,0,-1):
    if i%2==1:
        couleur='blue'
    else:
        couleur='red'
    disque(150, 200, 10*i, couleur)
```

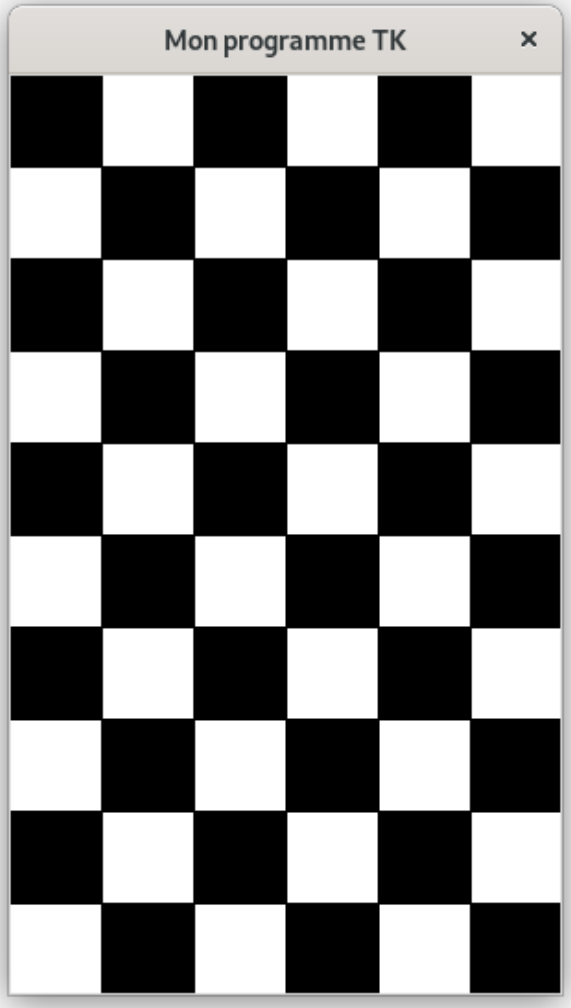


Exemple 3 : tracer un échiquier

- Deux systèmes de coordonnés :
  - `(x,y)` correspond aux pixels
  - `(i,j)` correspond aux carrés de 50 pixels de côté
- Le carré noir `(i,j)=(3,1)` : 4^{e} colonne, 2nd ligne
  - En posant `(x,y)=(3*c,1*c)`
  - `(x,y)` correspond au coin supérieur gauche
  - `(x+c,y+c)` correspond au coin inferieur droit
- Les cases noires sont celles vérifiant : `i+j est pair`

SCRIPT

```
c=50 # Côté d'une case en pixels

def case(i,j):
    (x,y) = (c*i,c*j)
    p=(x,y)
    q=(x+c,x+c)
    Dessin.create_rectangle(p,q,fill='black')

for i in range(0,Largeur//c):
    for j in range(0,Hauteur//c):
        if (i+j)%2==0:
            case(i,j)
```



Merci pour votre attention
Questions

?

Cours 4 — Graphismes, objets et variables globales

Partie I. Géométrie cartésienne

Les coordonnées en informatique

Écrire une fonction de traduction

Équations de droite et de cercle

Partie II. Couleurs et tuples

Le système RGB

Les tuples, un type composé

Mélangeons les couleurs

☕ Application

Partie III. Variables locales/globales

Variables locales

Variables globales

Fonctions locales

Procédures et fonctions

Partie IV. Dessiner avec Tk

Qu'est-ce donc?

Mon premier programme

Tracer des segments

Tracer des segments moins moches

Tracer des ellipses et des rectangles

Écrire sa propre fonction de tracer de cercle

Les couleurs sous Tk

☕ Exercices sur les Couleurs

Afficher un texte

Résumé

Pour aller plus loin

Partie V. Algorithmes de tracer de figure

Tracer une fonction

Tracer une fonction (complet)

Courbes paramétriques

Tracer des cercles

Tracer des polygones

Polygones et cercle

Partie VI. Créer son propre type

Rappel : les types en Python

Un premier exemple : les étudiants

Tentative avec des tuples

Créer un nouveau type

Classe : vocabulaire

Classes et attributs : vocabulaire

Au tour des enseignants!

Classes et méthodes

Partie VII. Exemples

Une tortue géomètre

La tortue et le Python

La courbe fractale de Koch

Fractale en Python

☕ Défis!

Exemple 1 : tracer de pixels

Exemple 2 : tracer une cible

Exemple 3 : tracer un échiquier

Partie VIII. Table des matières