Photographie numérique & Python

Fiche d'activité – Parcours en 4 séances

Quand tu prends une photo, ce n'est pas seulement l'objectif qui travaille : des **algorithmes** calculent l'image, la corrigent, l'améliorent… et permettent aussi de la **truquer**. Dans ce parcours, tu vas écrire toi-même de petits programmes en Python pour comprendre comment on fabrique et transforme une image numérique.

Séance 1 – La tortue	Donner des instructions à un robot dessinateur, répéter avec une boucle.
Séance 2 – Les fonctions	Donner un nom à un morceau de programme, utiliser un paramètre.
Séance 3 – Pixels et couleurs	Comprendre l'écran et le capteur, créer une image pixel par pixel.
Séance 4 – Trucages	Niveaux de gris, négatif, miroir, incrustation sur fond vert, contours.

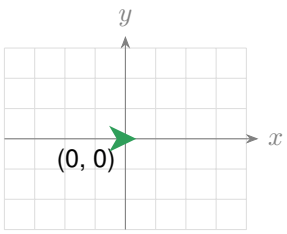
Comment travailler ? Les programmes se tapent dans la page « Photographie numérique & Python » du site **webclasse.fr**. Les numéros d'étapes de cette fiche sont les mêmes que sur la page. Lis la consigne, tape ton programme, clique sur **Exécuter**, observe la fenêtre d'affichage puis clique sur **Vérifier**. Sur cette fiche, tu complètes les réponses et tu recopies les programmes qui fonctionnent : elle te servira de résumé.

RÈGLE D'OR

Les programmes donnés en exemple ne se copient pas : **tape-les toi-même**. C'est en tapant qu'on retient, et qu'on apprend à repérer ses erreurs. Une majuscule, une parenthèse ou un décalage en trop, et Python refuse : lis alors le message d'erreur, il indique la ligne et une piste.

Les instructions de la tortue

`from turtle import *`	première ligne obligatoire
`forward(d)`	avance de d pas
`left(a)`	tourne de a degrés vers la gauche
`right(a)`	tourne de a degrés vers la droite
`up() / down()`	lève / baisse le crayon
`color("red")`	couleur du trait (nom anglais)



La tortue part du centre (0, 0), tournée vers la droite.

Séance 1 – La tortue : instructions et boucles

Objectifs : écrire ses premières instructions Python, comprendre et utiliser la boucle `for`.

Étape 1.2 — Premier programme

Exemple

```
from turtle import *

forward(120)
left(90)
color("red")
forward(80)
```

Tape ce programme, exécute-le, puis complète le tableau.

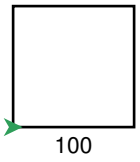
Instruction	Ce qu'elle fait
`from turtle import *`	
`forward(120)`	
`left(90)`	
`color("red")`	

Étape 1.3 — Le carré

Exercice

Écris un programme qui trace un carré de 100 pas de côté, **sans boucle**.

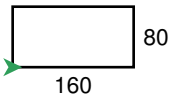
Mon programme



Étape 1.4 — Le rectangle

Exercice

Trace un rectangle de 160 pas de large et 80 pas de haut, toujours sans boucle.
Quels nombres changent par rapport au carré ? .....



ÉTAPE 1.5 – LA BOUCLE FOR

Pour **répéter** des instructions, on utilise une boucle. Exemple : un escalier de 3 marches.

```
for i in range(3):
    forward(40)
    left(90)
    forward(40)
    right(90)
```

- `for i in range(3):` veut dire « répète 3 fois » ; les deux-points à la fin sont obligatoires ;
- les lignes **décalées de 4 espaces** forment le **bloc répété** (une marche) ;
- `i` est un compteur : `range(n)` donne 0, 1, ..., $n-1$ et `range(a, b)` donne $a, \ldots, b-1$.

a. Combien de fois le bloc de `for i in range(6):` est-il répété ? .....

b. Quelles valeurs prend `i` avec `range(2, 5)` ? .....

c. Dans le programme ci-contre, combien de fois la tortue tourne-t-elle ? .....

```
for i in range(3):
    forward(50)
left(90)
```

Étape 1.6 — L'escalier

Exemple

Tape le programme de l'escalier. Change ensuite `3` en `6` : combien de marches obtiens-tu ? .....

Étape 1.7 — Le carré avec une boucle

Exercice

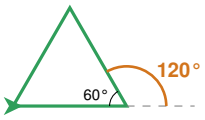
Réécris le carré de l'étape 1.3 avec une boucle `for`. Ton programme ne doit contenir qu'un seul `forward`.

Mon programme

Étape 1.8 — Le triangle

Exercice

Trace un triangle équilatéral de 100 pas de côté avec une boucle. Attention : la tortue ne tourne pas de 60° mais de **120°** à chaque sommet (voir la figure).



Mon programme

Étape 1.9 & 1.10 — Polygones réguliers

Exercice

Quand la tortue fait le tour complet d'un polygone, elle tourne en tout de **360°**. Complète le tableau.

Polygone	**Côtés**	**Calcul**	**Angle**	**Boucle et virage**
Triangle	3	$360 \div 3$	120°	`for i in range(3): ... left(120)`
Carré				
Hexagone				
Octogone				
Polygone à 10 côtés				

Étape 1.11 — L'étoile

Défi

Trace une étoile à 5 branches de 200 pas. À chaque pointe, la tortue tourne **vers la droite** (instruction `right`). Pour tracer l'étoile, elle fait **deux tours complets**, soit 720° en tout.



Angle de chaque virage : .....

Mon programme

JE RETIENS – SÉANCE 1

- ☐ Je sais déplacer la tortue avec `forward`, `left` et `right`.
- ☐ Je sais écrire une boucle `for i in range(n):` avec les deux-points et le décalage.
- ☐ Je sais calculer l'angle d'un polygone régulier : $360 \div n$.

Mon carnet d'erreurs : une erreur que j'ai faite, et comment je l'ai corrigée

Séance 2 – Les fonctions

Objectifs : définir une fonction avec `def`, l'appeler, utiliser un paramètre.

Étape 2.1 — Te souviens-tu ?

Rappels

a. Que fait l'instruction `left(90)` ? .....

b. Quel caractère manque à la fin de la ligne `for i in range(5)` ? .....

c. Quelle figure le programme ci-contre trace-t-il ?

.....

```
for i in range(3):
    forward(100)
    left(120)
```

Étape 2.2 — Définir une fonction

Exemple

```
from turtle import *

def motif():
    forward(30)
    left(90)
    forward(30)
    right(90)
    forward(30)
    right(90)
    forward(30)
    left(90)

motif()
```

Une **fonction** est un morceau de programme qui porte un **nom**. On la **définit** avec `def`, puis on l'**appelle** par son nom.

1. Sans le taper, dessine ce que va tracer la tortue (un carreau = 30 pas) :



2. Tape le programme **sans sa dernière ligne** `motif()` et exécute-le. Que trace la tortue ?

.....

3. Ajoute la dernière ligne et exécute. Ton dessin était-il juste ?

.....

Étape 2.3 — La ligne brisée

Exercice

```
def lignebrise():
    for i in range(.......):
        ..................

lignebrise()
```

La fonction `lignebrise()` doit répéter 6 fois le motif. Complète-la.



ÉTAPE 2.4 – LES PARAMÈTRES

```
def carre(x):
    for i in range(4):
        forward(x)
        left(90)

carre(50)
carre(100)
```

`x` est un **paramètre** : une « case » qui reçoit une valeur à chaque appel. L'appel `carre(50)` exécute la fonction avec $x = 50$, l'appel `carre(100)` avec $x = 100$.

Avec l'appel `carre(30)`, de combien la tortue avance-t-elle à chaque côté ? .....

Étape 2.5 — La fonction carre(x)

Exemple

Tape le programme de l'encadré en appelant `carre(20)`, `carre(50)` et `carre(100)`. Combien d'espaces faut-il devant `forward(x)` ? .....

Étape 2.6 — La fonction triangle(c)

Exercice

Écris une fonction `triangle(c)` qui trace un triangle équilatéral de côté `c`, puis appelle-la avec `triangle(60)` et `triangle(120)`.

Mon programme

Étape 2.7 — La rangée de carrés

Exercice

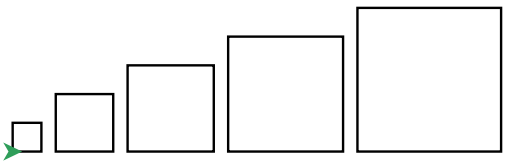
```
def carre(x):
    for i in range(4):
        forward(x)
        left(90)
    forward(x)

def rangee():
    for i in range(1, 6):
        carre(.....)
        up()
        forward(.....)
        down()

rangee()
```

Complète la fonction pour obtenir 5 carrés de côtés 20, 40, 60, 80 et 100, séparés par 10 pas. `up()` lève le crayon, `down()` le baisse.
Avec `range(1, 6)`, quelles valeurs prend `i` ?

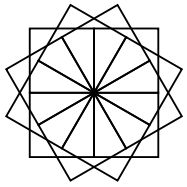
.....



Étape 2.8 — La rosace

Défi

Écris un programme qui définit `carre(x)` puis trace **12 carrés de 100 pas**, en tournant de 30° vers la gauche après chaque carré.
Pourquoi 12 et 30 ? .....



Mon programme

JE RETIENS – SÉANCE 2

- ☐ `def` **définit** une fonction : rien ne se passe tant qu'on ne l'**appelle** pas.
- ☐ Un **paramètre** reçoit une valeur différente à chaque appel.
- ☐ Une fonction peut en appeler une autre.

Mon carnet d'erreurs : une erreur que j'ai faite, et comment je l'ai corrigée

Séance 3 – Pixels et couleurs

Objectifs : comprendre l'écran et le capteur, coder une couleur en RVB, créer une image pixel par pixel.

Étape 3.2 — Une photo, c'est des pixels

Cours

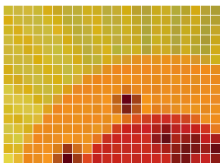
Une image numérique est un **quadrillage** de **pixels** : de petits carrés d'**une seule couleur**. La **définition** de l'image est son nombre de pixels : largeur × hauteur.

a. Une photo de 4 000 × 3 000 pixels contient ..... pixels, soit ..... mégapixels.

b. Une image de 1 920 × 1 080 pixels contient environ ..... pixels.



photo entière



zoom : les pixels

Étape 3.3 — Du capteur à l'image

Cours



photosites d'un capteur

Dans un appareil photo, la lumière arrive sur un **capteur** couvert de millions de **photosites**. Chacun ne mesure qu'**une seule couleur**. Ils sont groupés par 4 : deux **verts**, un **rouge**, un **bleu**. Un **algorithme** combine ces mesures pour calculer la couleur de chaque **pixel** de l'image.

La résolution du capteur se compte en ....., celle de l'image en .....

Étape 3.4 & 3.5 — L'écran : trois ampoules par pixel

Cours



un pixel d'écran vu à la loupe

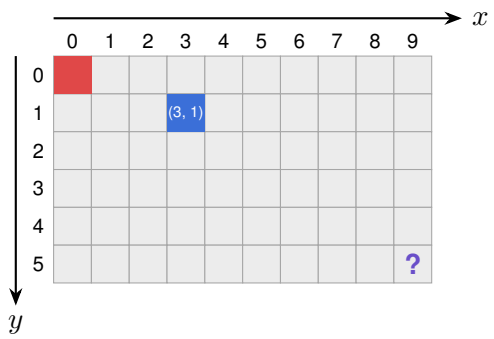
Chaque pixel d'un écran est fait de **trois minuscules ampoules** : une **rouge**, une **verte**, une **bleue**. Chacune brille de **0** (éteinte) à **255** (à fond). De loin, l'œil mélange leurs lumières et voit une seule couleur.

Le code `(r, g, b)` d'un pixel donne les **réglages des trois ampoules**. Chaque réglage est codé sur 8 bits : $2^8 = 256$ valeurs. Nombre de couleurs possibles : $256 \times 256 \times 256 =$ .....

Code	Ampoules allumées	Couleur vue de loin
`(0, 0, 0)`		
`(255, 0, 0)`		
`(0, 255, 0)`		
`(0, 0, 255)`		
`(255, 255, 0)`		
`(255, 255, 255)`		

Étape 3.6 — Le repère des pixels

Cours



une toute petite image de 10 × 6 pixels

Chaque pixel a deux coordonnées **(x, y)** numérotées à partir de **0** : x est la colonne, y la ligne. Le pixel (0, 0) est en haut à gauche et **y augmente vers le bas**.

a. Coordonnées de la case « ? » : .....

b. Colorie sur la figure le pixel (6, 2).

c. Dans une image de 300 × 200, coordonnées du pixel en bas à droite : .....

Étape 3.7 & 3.8 — Créer une image et allumer des pixels

Exemple

```
from PIL import Image

im = Image.new("RGB", (300, 200), "grey")
im.putpixel((10, 10), (0, 0, 0))
im.show()
```

`Image.new` crée une image grise de 300 pixels de large et 200 de haut ; `putpixel` colore **un** pixel ; `show` affiche l'image.
Attention aux **doubles parenthèses** : les coordonnées forment un groupe, la couleur un autre.

Étape 3.9 — Les quatre coins

Exercice

Crée une image jaune de 200 × 200, puis colore les coins : en haut à gauche bleu, en haut à droite rouge, en bas à gauche vert, en bas à droite blanc.

Coordonnées du coin en bas à droite : .....

Pourquoi pas (200, 200) ? .....

Mon programme

Étape 3.10 & 3.11 — Des segments

Exercice

a. Complète pour tracer un segment horizontal noir qui traverse toute l'image (300 × 200) à la hauteur $y = 100$.

```
for x in range(0, ..........):
    im.putpixel((x, ..........), (0, 0, 0))
```

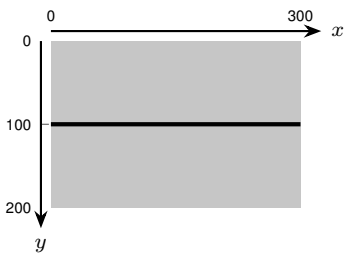


image attendue en a.

b. Écris la boucle qui trace un segment **vertical** rouge à $x = 150$, de haut en bas de l'image.

Mon programme

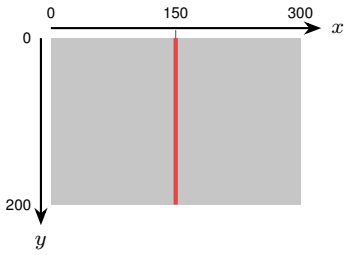


image attendue en b.

ÉTAPE 3.12 – DEUX BOUCLES IMBRIQUÉES

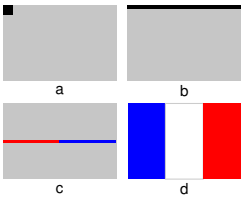
```
for x in range(0, 6):
    for y in range(0, 4):
        im.putpixel((x, y), (0, 0, 0))
```

Pour **chaque** valeur de x, la boucle sur y fait tout son travail : on colore l'image colonne par colonne. Ici, `putpixel` est exécuté ..... fois.

Étape 3.13 à 3.16 — Carré, bande, ligne bicolore, drapeau

Exercices

a. Colore en noir un carré de 20 × 20 pixels en haut à gauche : `range(0, .....)` pour x et pour y.
b. Trace une bande noire de 10 pixels d'épaisseur tout en haut de l'image.
c. Trace une ligne en $y = 100$: rouge pour x de 0 à 149, bleue pour x de 150 à 299.
d. Dessine le drapeau français (300 × 200) : bandes bleue, blanche, rouge de 100 pixels.



Mon programme

JE RETIENS – SÉANCE 3

- ☐ Chaque pixel d'écran = 3 ampoules ; le code `(r, g, b)` règle leur intensité de 0 à 255.
- ☐ Photosites (capteur) $\neq$ pixels (image).
- ☐ Le pixel (0, 0) est en haut à gauche ; dans une image $L \times H$, x va de 0 à $L - 1$.
- ☐ `im.putpixel((x, y), (r, g, b))` colore un pixel ; deux boucles imbriquées parcourent une zone.

Mon carnet d'erreurs : une erreur que j'ai faite, et comment je l'ai corrigée

Séance 4 – Retoucher et truquer des photos

Objectifs : lire les pixels d'une photo, programmer des filtres et un trucage par incrustation.

Étape 4.2 & 4.3 — Ouvrir une photo et lire un pixel

Exemple

```
from PIL import Image

im = Image.open("fleur.png")
largeur, hauteur = im.size
print("Largeur :", largeur)
print("Hauteur :", hauteur)
r, g, b = im.getpixel((100, 75))
print(r, g, b)
im.show()
```



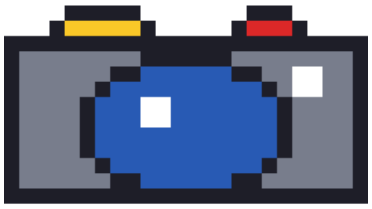
Largeur ..... Hauteur .....
Pixel (100, 75) : (.... , ... , ...)
Ce pixel est plutôt .....

Étape 4.4 — Compter les pixels blancs

Exemple

```
from PIL import Image

im = Image.open("appareil.png")
largeur, hauteur = im.size
compteur = 0
for x in range(0, largeur):
    for y in range(0, hauteur):
        r, g, b = im.getpixel((x, y))
        if r == 255 and g == 255 and b == 255:
            compteur = compteur + 1
print("Pixels blancs :", compteur)
```



24 × 16 pixels (agrandie)

Nombre de pixels blancs
.....

`if` exécute le bloc décalé seulement si la condition est vraie. Pour **comparer** on écrit == ; un seul = sert à **ranger** une valeur dans une variable.

Étape 4.5 & 4.6 — Le programme de base d'un filtre

Cours

```
from PIL import Image

im = Image.open("fleur.png")
largeur, hauteur = im.size
nouv = Image.new("RGB", (largeur, hauteur))

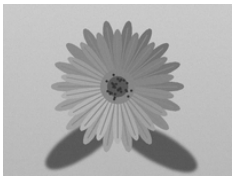
for x in range(0, largeur):
    for y in range(0, hauteur):
        r, g, b = im.getpixel((x, y))
        gris = (r + g + b) // 3
        nouv.putpixel((x, y), (gris, gris, gris))

im.show()
nouv.show()
```

1. ouvrir la photo, lire sa taille ;
2. créer une nouvelle image vide ;
3. parcourir tous les pixels ;
4. lire, calculer, écrire ;
5. afficher.

`//` est la **division entière**.
Combien vaut `17 // 3` ? .....
Gris du pixel (90, 120, 30) .....





Étape 4.7 à 4.10 — Les filtres

Exercices

Pour chaque filtre, écris ce qui remplace la ligne `nouv.putpixel((x, y), (gris, gris, gris))` du programme de base.

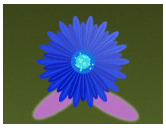


4.7 – Noir et blanc : si le gris est inférieur à 150, le pixel devient noir, sinon blanc.

.....

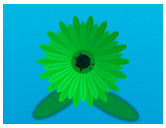
.....

.....



4.8 – Négatif : chaque composante c devient $255 - c$.
Négatif du jaune (255, 255, 0) : .....

.....



4.9 – Sans le rouge : le pixel (r, g, b) devient $(0, g, b)$.

.....

Pourquoi les pétales jaunes deviennent-ils verts ?

.....

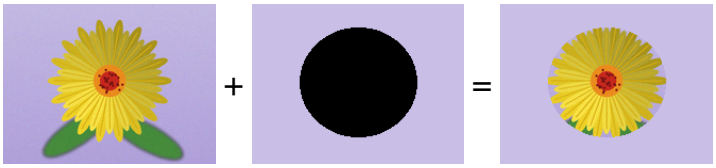


4.10 – Miroir : la photo mesure 200 pixels de large. Complète : le pixel de la colonne $x = 0$ va dans la colonne ....., celui de la colonne $x = 1$ dans la colonne ....., celui de la colonne $x = 199$ dans la colonne .....
Le pixel lu en (x, y) est donc écrit en $(\ldots\ldots, y)$.

.....

Étape 4.11 — Incrustation dans un masque

Exercice

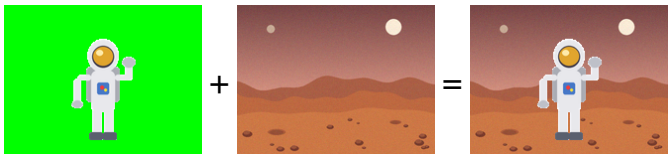


Si le pixel du masque est noir, on prend celui de la fleur ; sinon on garde celui du masque. Complète la condition.

```
        r1, g1, b1 = fleur.getpixel((x, y))
        r2, g2, b2 = masque.getpixel((x, y))
        if ..........................................:
            nouv.putpixel((x, y), (r1, g1, b1))
        else:
            nouv.putpixel((x, y), (r2, g2, b2))
```

Étape 4.12 — Le fond vert

Défi



Les pixels du fond vert valent exactement $(0, 255, 0)$. Écris les deux boucles qui réalisent l'incrustation.

Mon programme

Étape 4.13 — Détecter les contours

Défi

Sur un contour, la couleur change brusquement entre deux pixels voisins. On compare le niveau de gris g_1 du pixel (x, y) et celui g_2 de son voisin $(x + 1, y)$: si `abs(g1 - g2) > 20`, le pixel devient noir (sur une image blanche).
Pourquoi x doit-il s'arrêter à `largeur - 2` ?

.....



Étape 4.14 — Algorithmes et trucages

Bilan

Prise de vue mise au point, exposition → **Capteur** les photosites mesurent la lumière → **Développement** pixels RVB, balance des blancs, netteté → **Fusion d'images** HDR, panorama, réduction du bruit → **Fichier** compression JPEG, métadonnées EXIF

a. Cite deux traitements que tu as programmés et qu'un téléphone peut réaliser automatiquement.

.....

b. Le trucage du fond vert est-il visible à l'œil nu ? Quelles précautions prendre avant de croire ou de partager une photo ?

.....

.....

c. Pourquoi une photo publiée sur Internet risque-t-elle de ne jamais disparaître (droit à l'oubli) ?

.....

JE RETIENS – SÉANCE 4

- ☐ `r, g, b = im.getpixel((x, y))` lit un pixel.
- ☐ Un filtre parcourt tous les pixels avec deux boucles, calcule et écrit une nouvelle couleur.
- ☐ Gris : `(r + g + b) // 3` ; négatif : `(255 - r, 255 - g, 255 - b)`.
- ☐ Les algorithmes permettent d'améliorer une photo… et de la truquer.

Mon carnet d'erreurs : une erreur que j'ai faite, et comment je l'ai corrigée