Chapitre 8

La photographie numérique

Objectifs du chapitre

Thème : La photographie numérique.
Distinguer les photosites du capteur et les pixels de l'image, calculer la taille d'une image, coder une couleur en RVB, retrouver les métadonnées EXIF, transformer une image par programme en agissant sur les composantes de ses pixels et identifier les algorithmes qui construisent l'image finale.
Organisation : programme officiel, cours, méthodes, ateliers Python, exercices corrigés, QCM et fiche de synthèse.

Programme officiel (BO)

Thème « La photographie numérique » – Contenus et capacités attendues

— **Photosites, pixels, résolution (du capteur, de l'image), profondeur de couleur** : Distinguer les photosites du capteur et les pixels de l'image en comparant les résolutions du capteur et de l'image selon les réglages de l'appareil.

— **Métadonnées EXIF** : Retrouver les métadonnées d'une photographie.

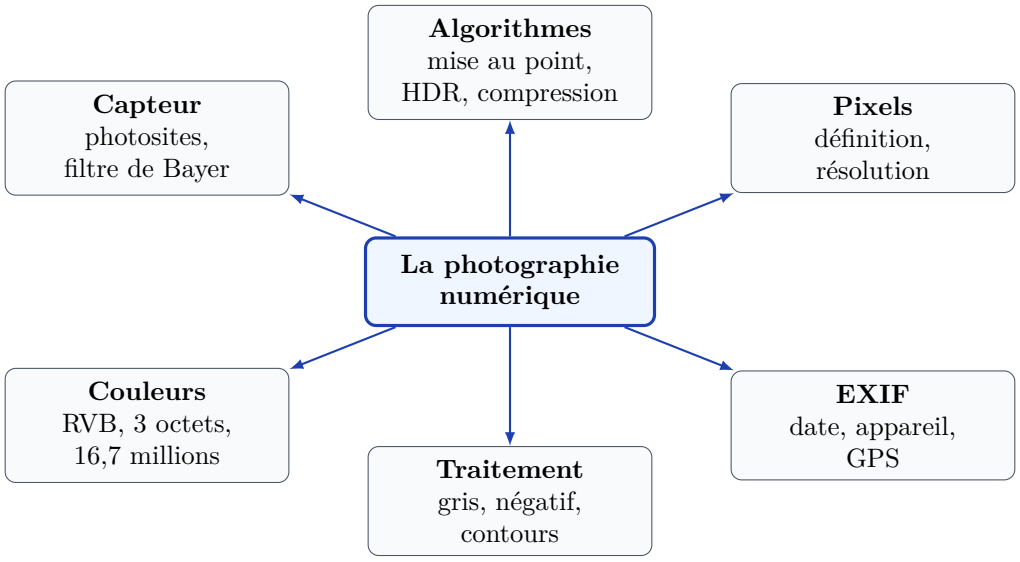
— **Traitement d'image** : Traiter par programme une image pour la transformer en agissant sur les trois composantes de ses pixels.

— **Rôle des algorithmes dans les appareils photo numériques** : Expliciter des algorithmes associés à la prise de vue. Identifier les étapes de la construction de l'image finale.

Source : BO spécial n°1 du 22 janvier 2019, programme de Sciences numériques et technologie de Seconde générale et technologique.

8.1 Carte mentale du chapitre

Carte mentale

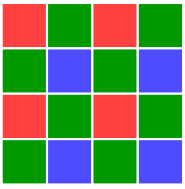


8.2 Du capteur à l'image : photosites et pixels

La photographie numérique a remplacé la pellicule en quelques années (premier appareil numérique en 1975 chez Kodak, premiers téléphones photo vers 2000). Aujourd'hui, la quasi-totalité des photos sont prises avec un smartphone : plus de mille milliards de photos par an dans le monde. Leur coût de copie est presque nul et elles se diffusent instantanément par Internet.

Définition – Photosites et pixels

- Le **capteur** d'un appareil photo est une puce couverte de millions de cellules sensibles à la lumière : les **photosites**. Chaque photosite mesure la quantité de lumière qu'il reçoit.
- Devant le capteur, un **filtre de Bayer** fait que chaque photosite ne reçoit qu'une couleur : rouge, vert ou bleu.
- L'image finale est un tableau de **pixels** (*picture elements*) : chaque pixel a une couleur complète, calculée par un algorithme à partir de plusieurs photosites.



Filtre de Bayer : chaque photosite ne mesure qu'une couleur (50 % de verts, 25 % de rouges, 25 % de bleus). Le *dématriçage* calcule les deux couleurs manquantes de chaque pixel à partir des photosites voisins.

Définition – Définition et résolution

- La **définition** d'une image est son nombre de pixels : largeur × hauteur. Une image de 4 000 × 3 000 pixels compte 12 000 000 pixels, soit **12 mégapixels** (Mpx).
- La **résolution du capteur** est son nombre de photosites. La **résolution de l'image** dépend des réglages : on peut enregistrer une image de 12 Mpx avec un capteur de 48 Mpx (4 photosites regroupés pour un pixel, moins de bruit la nuit) ou réduire la taille pour gagner de la place.
- En impression, on parle aussi de résolution en **ppp** (pixels par pouce) : environ 300 ppp pour une bonne qualité.

8.3 Coder la couleur d'un pixel

Définition – Le codage RVB

Chaque pixel est codé par trois nombres : l'intensité du **R**ouge, du **V**ert et du **B**leu (RGB en anglais), chacun entre **0 et 255**. On l'écrit `(R, V, B)`.

— `(0, 0, 0)` est le noir, `(255, 255, 255)` le blanc ; quand les trois composantes sont égales, on obtient un gris.

— `(255, 0, 0)` est le rouge pur ; `(255, 255, 0)` le jaune (synthèse additive : rouge + vert = jaune).

— Chaque composante est codée sur 1 octet (8 bits, $2^8 = 256$ valeurs) : un pixel occupe 3 octets, soit une **profondeur de couleur** de 24 bits, et $256^3 \approx$ **16,7 millions** de couleurs possibles.

Animation interactive « Mélanger les couleurs » : à manipuler sur https://utspe.com/snt/chapitre-8.

Méthode – Calculer la taille d'une image

Taille non compressée = nombre de pixels × 3 octets. Pour 12 Mpx : 12 000 000 × 3 = 36 000 000 octets, soit **36 Mo**. Pour gagner de la place, l'image est **compressée** : au format **JPEG**, elle ne pèse plus que 3 à 5 Mo, au prix d'une légère perte de qualité ; le format PNG compresse sans perte ; le format RAW garde les données brutes du capteur.

```
largeur = 4000
hauteur = 3000
pixels = largeur * hauteur
taille = pixels * 3                  # 3 octets par pixel
print(pixels / 1000000, "megapixels")
print(taille / 1000000, "Mo sans compression")
```

8.4 Les métadonnées EXIF

Définition – EXIF

Les métadonnées **EXIF** sont enregistrées automatiquement par l'appareil dans le fichier de la photo : date et heure, marque et modèle de l'appareil, réglages (ouverture, durée d'exposition, sensibilité ISO, flash), définition, orientation, et souvent **coordonnées GPS**.

Métadonnée	Exemple	Ce qu'elle révèle
Date et heure	`2024:06:21 21:14:03`	quand la photo a été prise
Appareil	`Google Pixel 8`	le modèle de téléphone
Exposition	`1/120 s, f/1.7, ISO 400`	les réglages calculés par l'appareil
Dimensions	`4080 × 3072`	la définition (12,5 Mpx)
GPS	`45.1885 N, 5.7245 E`	le lieu exact de la prise de vue

Méthode – Retrouver et effacer les métadonnées

Sur ordinateur : propriétés du fichier (onglet Détails) ou un visualiseur d'images ; sur téléphone : bouton « i » de la galerie. Avant de publier une photo prise chez soi, on peut supprimer la position (option « supprimer la localisation » lors du partage) ou désactiver la géolocalisation de l'appareil photo.

8.5 Traiter une image par programme

Une image est un **tableau de pixels** : on peut la transformer en appliquant le même calcul à chaque pixel. Voici les filtres les plus classiques.

Filtre	Calcul sur un pixel (R, V, B)
Niveaux de gris	on remplace les trois composantes par leur moyenne `g = (R + V + B) // 3` : `(g, g, g)`
Négatif	chaque composante est remplacée par son complément à 255 : `(255 - R, 255 - V, 255 - B)`
Un seul canal	on garde une composante et on met les autres à 0, par exemple `(R, 0, 0)`
Luminosité	on ajoute la même valeur aux trois composantes, sans dépasser 255
Contours	on compare la luminosité d'un pixel à celle de ses voisins : si l'écart dépasse un **seuil**, le pixel est un contour (noir), sinon il devient blanc

Animation interactive « Appliquer des filtres à une image » : à manipuler sur https://utspe.com/snt/chapitre-8.

Méthode – Parcourir les pixels en Python

On représente une image par une liste de lignes, chaque ligne étant une liste de pixels `(R, V, B)`. Deux boucles imbriquées parcourent tous les pixels :

```
image = [[(255, 0, 0), (0, 255, 0)],
         [(0, 0, 255), (200, 200, 200)]]

negatif = []
for ligne in image:
    nouvelle = []
    for (r, v, b) in ligne:
        nouvelle.append((255 - r, 255 - v, 255 - b))
    negatif.append(nouvelle)
print(negatif)
```

8.6 Les algorithmes des appareils photo

Propriété – Les étapes de la construction de l'image

1. **Avant la prise de vue** : mise au point automatique (l'algorithme cherche le réglage qui donne le plus de contraste), mesure de la lumière et choix de l'exposition, détection des visages ;
2. **acquisition** : les photosites mesurent la lumière (souvent plusieurs images en rafale) ;
3. **dématriçage** : calcul des trois couleurs de chaque pixel à partir du filtre de Bayer ;
4. **balance des blancs** : correction pour qu'un objet blanc paraisse blanc, quelle que soit la lumière (soleil, lampe) ;
5. **réduction du bruit**, accentuation des contours, correction des déformations de l'objectif ;
6. **fusion** de plusieurs images : HDR (détails dans les zones sombres et claires), mode nuit, mode portrait (flou d'arrière-plan calculé) ;
7. **compression** (JPEG ou HEIF) et ajout des métadonnées EXIF.

On parle de **photographie computationnelle** : sur un smartphone, l'image finale est autant calculée par des algorithmes (et de l'intelligence artificielle) que captée par l'optique. Une photo n'est donc jamais une copie exacte de la réalité.

Attention

Les retouches et les images générées par intelligence artificielle peuvent tromper : avant de partager une photo « choc », on vérifie sa source, et on peut rechercher l'image d'origine avec une recherche d'image inversée.

8.7 Ateliers Python

Les ateliers se font sur ordinateur, dans un éditeur Python ou directement sur https://utspe.com/snt/chapitre-8. Pour chacun : recopier le programme de départ, suivre les étapes dans l'ordre et exécuter le programme après chaque modification.

Atelier 1 – Définition et taille d'une image — calculs, fonctions, f-strings

Objectif : Calculer la définition et la taille d'une image selon les réglages de l'appareil, et mesurer l'effet de la compression.

1. Exécuter le programme : il calcule la définition d'une photo de 4 000 × 3 000 pixels.
2. Écrire une fonction `taille_mo(largeur, hauteur)` qui renvoie la taille non compressée en Mo (3 octets par pixel, 1 Mo = 1 000 000 octets).
3. Afficher la taille pour 4 000 × 3 000, 8 000 × 6 000 (capteur de 48 Mpx en pleine définition) et 1 920 × 1 080 (image HD).
4. Le fichier JPEG de la première photo pèse 4,2 Mo. Calculer le taux de compression (taille non compressée ÷ taille JPEG).

Programme de départ :

```
largeur = 4000
hauteur = 3000
pixels = largeur * hauteur
print(f"Definition : {pixels} pixels, soit {pixels / 1000000} Mpx")

# a completer
```

Coup de pouce : `def taille_mo(largeur, hauteur): return largeur * hauteur * 3 / 1000000`. Taux : `taille_mo(4000, 3000) / 4.2`.

Corrigé rédigé

```
def taille_mo(largeur, hauteur):
    return largeur * hauteur * 3 / 1000000

for (l, h) in [(4000, 3000), (8000, 6000), (1920, 1080)]:
    print(f"{l} x {h} : {l * h / 1000000} Mpx, {taille_mo(l, h):.1f} Mo")

taux = taille_mo(4000, 3000) / 4.2
print(f"Le JPEG est environ {taux:.1f} fois plus leger")
```

Atelier 2 – Transformer un pixel — tuples, fonctions, return

Objectif : Écrire les fonctions qui transforment un pixel (R, V, B) : niveaux de gris, négatif et luminosité.

1. Un pixel est un tuple `(r, v, b)`. Exécuter le programme pour décomposer un pixel orange.
2. Compléter `gris(pixel)` : elle renvoie `(g, g, g)` avec `g` la moyenne entière des trois composantes.
3. Compléter `negatif(pixel)`.

4. Écrire `eclaircir(pixel, n)` qui ajoute `n` à chaque composante sans dépasser 255 (utiliser `min`).

Programme de départ :

```
def gris(pixel):
    r, v, b = pixel
    return pixel          # a completer

def negatif(pixel):
    r, v, b = pixel
    return pixel          # a completer

orange = (255, 165, 0)
r, v, b = orange
print("rouge", r, "vert", v, "bleu", b)
print("gris :", gris(orange))
print("negatif :", negatif(orange))
```

Coup de pouce : `g = (r + v + b) // 3` puis `return (g, g, g)`. Négatif : `return (255 - r, 255 - v, 255 - b)`. Éclaircir : `min(255, r + n)` pour chaque composante.

Corrigé rédigé

```
def gris(pixel):
    r, v, b = pixel
    g = (r + v + b) // 3
    return (g, g, g)

def negatif(pixel):
    r, v, b = pixel
    return (255 - r, 255 - v, 255 - b)

def eclaircir(pixel, n):
    r, v, b = pixel
    return (min(255, r + n), min(255, v + n), min(255, b + n))

orange = (255, 165, 0)
print("gris :", gris(orange))
print("negatif :", negatif(orange))
print("eclairci :", eclaircir(orange, 60))
```

Atelier 3 – Afficher une image en caractères

listes de listes, boucles imbriquées

Objectif : Construire une petite image (un paysage) pixel par pixel, puis l'afficher avec des caractères plus ou moins « foncés ».

1. Exécuter : le programme fabrique une image de 32 × 12 pixels et l'affiche (chaque caractère représente un pixel).
2. Repérer le soleil, le ciel, l'herbe et le chemin. Changer la couleur du ciel en `(150, 200, 255)` : que se passe-t-il ?
3. Ajouter une maison : un rectangle de pixels `(120, 40, 30)` pour `12 <= x <= 17` et `5 <= y <= 7` (attention à l'ordre des conditions).
4. Afficher la couleur du pixel situé en `x = 24`, `y = 3` : `image[3][24]`. Pourquoi écrit-on la ligne `y` en premier ?

Programme de départ :

```
largeur, hauteur = 32, 12
```

```
image = []
for y in range(hauteur):
    ligne = []
    for x in range(largeur):
        if (x - 24) ** 2 + 4 * (y - 3) ** 2 < 24:
            ligne.append((255, 235, 120))     # soleil
        elif y >= 8 and 6 <= x <= 9:
            ligne.append((150, 120, 90))      # chemin
        elif y >= 8:
            ligne.append((30, 110, 40))       # herbe
        else:
            ligne.append((40, 70, 160))       # ciel
    image.append(ligne)

def afficher(image):
    nuances = "@%#*+=-:. "        # du plus sombre au plus clair
    for ligne in image:
        texte = ""
        for pixel in ligne:
            r, v, b = pixel
            gris = (r + v + b) // 3
            texte = texte + nuances[gris * len(nuances) // 256]
        print(texte)

afficher(image)
```

Coup de pouce : Ajouter un `elif 12 <= x <= 17 and 5 <= y <= 7:` juste après le test du soleil, avant ceux du chemin et de l'herbe. `image` est une liste de lignes : `image[y]` est la ligne numéro `y`, et `image[y][x]` le pixel de la colonne `x`.

Corrigé rédigé

```
largeur, hauteur = 32, 12
image = []
for y in range(hauteur):
    ligne = []
    for x in range(largeur):
        if (x - 24) ** 2 + 4 * (y - 3) ** 2 < 24:
            ligne.append((255, 235, 120))     # soleil
        elif 12 <= x <= 17 and 5 <= y <= 7:
            ligne.append((120, 40, 30))       # maison
        elif y >= 8 and 6 <= x <= 9:
            ligne.append((150, 120, 90))      # chemin
        elif y >= 8:
            ligne.append((30, 110, 40))       # herbe
        else:
            ligne.append((40, 70, 160))       # ciel
    image.append(ligne)

def afficher(image):
    nuances = "@%#*+=-:. "        # du plus sombre au plus clair
    for ligne in image:
        texte = ""
        for pixel in ligne:
            r, v, b = pixel
            gris = (r + v + b) // 3
            texte = texte + nuances[gris * len(nuances) // 256]
```

```
        print(texte)

afficher(image)
print("Pixel (24, 3) :", image[3][24])
```

Atelier 4 – Appliquer un filtre à toute l'image

fonctions, boucles imbriquées

Objectif : Écrire une fonction qui applique le négatif à tous les pixels de l'image, puis une autre qui ne garde que la composante rouge.

1. Exécuter : le programme affiche l'image de l'atelier 3.
2. Compléter `appliquer_negatif(image)` : elle renvoie une **nouvelle** image où chaque pixel est remplacé par son négatif.
3. Afficher le négatif : les zones claires deviennent sombres.
4. Écrire de même `appliquer_rouge(image)` qui ne garde que la composante rouge, et observer le résultat.

Programme de départ :

```
largeur, hauteur = 32, 12
image = []
for y in range(hauteur):
    ligne = []
    for x in range(largeur):
        if (x - 24) ** 2 + 4 * (y - 3) ** 2 < 24:
            ligne.append((255, 235, 120))     # soleil
        elif y >= 8 and 6 <= x <= 9:
            ligne.append((150, 120, 90))      # chemin
        elif y >= 8:
            ligne.append((30, 110, 40))       # herbe
        else:
            ligne.append((40, 70, 160))       # ciel
    image.append(ligne)

def afficher(image):
    nuances = "@%#*+=-:. "        # du plus sombre au plus clair
    for ligne in image:
        texte = ""
        for pixel in ligne:
            r, v, b = pixel
            gris = (r + v + b) // 3
            texte = texte + nuances[gris * len(nuances) // 256]
        print(texte)

def appliquer_negatif(image):
    resultat = []
    for ligne in image:
        nouvelle = []
        # a completer
        resultat.append(nouvelle)
    return resultat

afficher(image)
print()
afficher(appliquer_negatif(image))
```

Coup de pouce : Dans la boucle sur les lignes : `for (r, v, b) in ligne: nouvelle.append((255 - r, 255 - v, 255 - b))`.

Corrigé rédigé

```python
largeur, hauteur = 32, 12
image = []
for y in range(hauteur):
    ligne = []
    for x in range(largeur):
        if (x - 24) ** 2 + 4 * (y - 3) ** 2 < 24:
            ligne.append((255, 235, 120))     # soleil
        elif y >= 8 and 6 <= x <= 9:
            ligne.append((150, 120, 90))      # chemin
        elif y >= 8:
            ligne.append((30, 110, 40))       # herbe
        else:
            ligne.append((40, 70, 160))       # ciel
    image.append(ligne)

def afficher(image):
    nuances = "@%#*+=-:. "         # du plus sombre au plus clair
    for ligne in image:
        texte = ""
        for pixel in ligne:
            r, v, b = pixel
            gris = (r + v + b) // 3
            texte = texte + nuances[gris * len(nuances) // 256]
        print(texte)

def appliquer_negatif(image):
    resultat = []
    for ligne in image:
        nouvelle = []
        for (r, v, b) in ligne:
            nouvelle.append((255 - r, 255 - v, 255 - b))
        resultat.append(nouvelle)
    return resultat

def appliquer_rouge(image):
    resultat = []
    for ligne in image:
        nouvelle = []
        for (r, v, b) in ligne:
            nouvelle.append((r, 0, 0))
        resultat.append(nouvelle)
    return resultat

afficher(image)
print()
afficher(appliquer_negatif(image))
print()
afficher(appliquer_rouge(image))
```

Atelier 5 – Détecter les contours

indices, abs(), seuil

Objectif : Programmer l'algorithme d'extraction de contours : un pixel est un contour s'il est très différent de son voisin de droite ou de son voisin du dessous.

1. La fonction `lum(pixel)` renvoie la luminosité d'un pixel (moyenne des composantes). Exécuter le programme.

2. Compléter : pour chaque pixel, calculer l'écart `abs(lum(p) - lum(droite)) + abs(lum(p) - lum(dessous))`.
3. Si l'écart est supérieur au seuil, afficher `#`, sinon un espace.
4. Tester avec un seuil de 20 puis de 100 : quels contours disparaissent ?

Programme de départ :

```
largeur, hauteur = 32, 12
image = []
for y in range(hauteur):
    ligne = []
    for x in range(largeur):
        if (x - 24) ** 2 + 4 * (y - 3) ** 2 < 24:
            ligne.append((255, 235, 120))     # soleil
        elif y >= 8 and 6 <= x <= 9:
            ligne.append((150, 120, 90))      # chemin
        elif y >= 8:
            ligne.append((30, 110, 40))       # herbe
        else:
            ligne.append((40, 70, 160))       # ciel
    image.append(ligne)

def lum(pixel):
    r, v, b = pixel
    return (r + v + b) // 3

seuil = 40
for y in range(hauteur - 1):
    texte = ""
    for x in range(largeur - 1):
        p = image[y][x]
        droite = image[y][x + 1]
        dessous = image[y + 1][x]
        texte = texte + "."      # a completer
    print(texte)
```

Coup de pouce : `ecart = abs(lum(p) - lum(droite)) + abs(lum(p) - lum(dessous))`, puis `if ecart > seuil: texte = texte + "#"` sinon `texte = texte + " "`. On s'arrête à `largeur - 1` et `hauteur - 1` car le dernier pixel n'a pas de voisin à droite ou en dessous.

Corrigé rédigé

```
largeur, hauteur = 32, 12
image = []
for y in range(hauteur):
    ligne = []
    for x in range(largeur):
        if (x - 24) ** 2 + 4 * (y - 3) ** 2 < 24:
            ligne.append((255, 235, 120))     # soleil
        elif y >= 8 and 6 <= x <= 9:
            ligne.append((150, 120, 90))      # chemin
        elif y >= 8:
            ligne.append((30, 110, 40))       # herbe
        else:
            ligne.append((40, 70, 160))       # ciel
    image.append(ligne)

def lum(pixel):
```

```
    r, v, b = pixel
    return (r + v + b) // 3

def contours(seuil):
    for y in range(hauteur - 1):
        texte = ""
        for x in range(largeur - 1):
            p = image[y][x]
            droite = image[y][x + 1]
            dessous = image[y + 1][x]
            ecart = abs(lum(p) - lum(droite)) + abs(lum(p) - lum(dessous))
            if ecart > seuil:
                texte = texte + "#"
            else:
                texte = texte + " "
        print(texte)

contours(40)
print("-" * 31)
contours(100)
```

8.8 Exercices corrigés

Exercice

Exercice 1 – Photosites et pixels. Un smartphone a un capteur de 48 millions de photosites. Par défaut, il enregistre des photos de 4 000 × 3 000 pixels.
1. Quelle est la définition des photos en mégapixels ? Combien de photosites sont utilisés pour un pixel ?
2. Dans le mode « 48 Mpx », quelle est la définition de l'image si le rapport largeur/hauteur reste 4/3 ?
3. Pourquoi le mode par défaut donne-t-il de meilleures photos la nuit ?

Corrigé rédigé

1. 4 000 × 3 000 = 12 000 000 pixels = 12 Mpx ; 48 ÷ 12 = 4 photosites par pixel.
2. 8 000 × 6 000 = 48 000 000 pixels.
3. En regroupant 4 photosites, chaque pixel reçoit plus de lumière : l'image est moins bruitée quand il fait sombre.

Exercice

Exercice 2 – Couleurs RVB. 1. Quelle couleur correspond à `(0, 0, 255)` ? À `(255, 255, 0)` ? À `(100, 100, 100)` ?
2. Donner le code d'un blanc, d'un noir, d'un cyan.
3. Combien de gris différents peut-on coder ? Combien de couleurs au total ?

Corrigé rédigé

1. Bleu ; jaune (rouge + vert) ; un gris foncé (trois composantes égales).
2. Blanc `(255, 255, 255)`, noir `(0, 0, 0)`, cyan `(0, 255, 255)`.
3. 256 gris (de `(0, 0, 0)` à `(255, 255, 255)`) ; $256^3 = 16\,777\,216$ couleurs.

```python
print(256 ** 3, "couleurs")
```

Exercice

Exercice 3 – Taille d'une photo. Une photo de 6 000 × 4 000 pixels est codée en RVB (3 octets par pixel).
1. Calculer sa taille non compressée en Mo.
2. Une carte mémoire de 64 Go peut-elle contenir 2 000 photos non compressées ? Et en JPEG, à 8 Mo par photo ?

Corrigé rédigé

1. 6 000 × 4 000 × 3 = 72 000 000 octets = **72 Mo**.
2. 2 000 × 72 Mo = 144 000 Mo = 144 Go > 64 Go : non. En JPEG : 2 000 × 8 Mo = 16 Go : oui, il reste de la place.

```python
taille = 6000 * 4000 * 3 / 1000000
print(taille, "Mo")
print(2000 * taille / 1000, "Go non compresse")
print(2000 * 8 / 1000, "Go en JPEG")
```

Exercice

Exercice 4 – Calculer des filtres à la main. Pour le pixel `(200, 90, 40)`, calculer : son niveau de gris (moyenne entière) ; son négatif ; sa version éclaircie de 80 ; sa version « rouge seul ».

Corrigé rédigé

Gris : (200 + 90 + 40) // 3 = 110, donc `(110, 110, 110)`. Négatif : `(55, 165, 215)`. Éclairci de 80 : `(255, 170, 120)` (280 est ramené à 255). Rouge seul : `(200, 0, 0)`.

Exercice

Exercice 5 – Lire les métadonnées EXIF. Une photo partagée sur un forum contient : `Date 2024:06:21 21:14:03` ; `Modèle Google Pixel 8` ; `1/120 s, f/1.7, ISO 400` ; `4080 × 3072` ; `GPS 45.1885 N, 5.7245 E`.
1. Quelle est la définition de la photo en Mpx ? Combien pèserait-elle sans compression ?
2. Quelles informations personnelles cette photo révèle-t-elle ?
3. Que faut-il faire avant de partager une photo ?

Corrigé rédigé

1. 4 080 × 3 072 ≈ 12,5 Mpx ; sans compression : environ 37,6 Mo.
2. Le lieu précis (près de Grenoble), la date et l'heure, le modèle de téléphone : si la photo est prise chez soi, elle révèle l'adresse.
3. Supprimer les données de localisation (option de partage ou outil de suppression des métadonnées), ou désactiver la géolocalisation de l'appareil photo.

```python
print(4080 * 3072 / 1000000, "Mpx")
print(4080 * 3072 * 3 / 1000000, "Mo")
```

Exercice

Exercice 6 – Contours à la main. On considère une ligne de pixels en niveaux de gris : `[20, 22, 25, 200, 205, 210, 60, 58]`. Un pixel est un contour si l'écart avec son voisin de droite dépasse 50.

1. Quels pixels sont des contours ?
2. Écrire un programme qui affiche `#` pour un contour et `.` sinon.

Corrigé rédigé

1. Écarts : 2, 3, 175, 5, 5, 150, 2. Les pixels n° 3 (valeur 25) et n° 6 (valeur 210) sont des contours : ce sont les bords de la zone claire.

```
ligne = [20, 22, 25, 200, 205, 210, 60, 58]
texte = ""
for i in range(len(ligne) - 1):
    if abs(ligne[i] - ligne[i + 1]) > 50:
        texte = texte + "#"
    else:
        texte = texte + "."
print(texte)
```

8.9 QCM – Teste tes connaissances

Pour chaque question, une seule réponse est correcte.

Q1. Les cellules du capteur sensibles à la lumière s'appellent…

a) des pixels
b) des photosites
c) des octets
d) des mégapixels

Q2. Une image de 3 000 × 2 000 pixels a une définition de…

a) 5 Mpx
b) 6 Mpx
c) 60 Mpx
d) 600 Mpx

Q3. Le filtre de Bayer contient…

a) autant de rouges, de verts et de bleus
b) deux fois plus de verts que de rouges ou de bleus
c) uniquement du vert
d) des filtres jaunes

Q4. Le pixel (255, 255, 255) est…

a) noir
b) blanc
c) gris
d) rouge

Q5. Combien d'octets faut-il pour coder un pixel en RVB ?

a) 1
b) 3
c) 8
d) 24

Q6. Que produit ce programme ?

```
p = (200, 100, 30)
r, v, b = p
print((r + v + b) // 3)
```

a) 110
b) 330
c) 110.0
d) (110, 110, 110)

Q7. Que produit ce programme ?

```
r, v, b = (10, 200, 255)
print((255 - r, 255 - v, 255 - b))
```

a) (10, 200, 255)
b) (245, 55, 0)
c) (0, 55, 245)
d) (255, 255, 255)

Q8. Les métadonnées EXIF peuvent contenir…

a) le prénom des personnes photographiées
b) les coordonnées GPS de la prise de vue
c) le mot de passe du téléphone
d) la liste des contacts

Q9. Le format JPEG…

a) compresse l'image avec une petite perte de qualité
b) garde les données brutes du capteur
c) agrandit l'image
d) supprime les couleurs

Q10. Dans un algorithme de contours, un pixel est un contour quand…

a) il est noir
b) sa luminosité est très différente de celle de ses voisins
c) il est au bord de l'image
d) il est rouge

Q11. La balance des blancs sert à…

a) corriger les couleurs selon l'éclairage
b) augmenter la définition
c) supprimer les métadonnées
d) faire la mise au point

Q12. La photographie computationnelle désigne…

a) les photos d'ordinateurs
b) les images construites en combinant plusieurs prises par des algorithmes
c) les photos imprimées
d) les captures d'écran

Corrigé rédigé

Réponses du QCM.
Q1. b) Les photosites sont sur le capteur ; les pixels sont dans l'image finale.
Q2. b) 3 000 × 2 000 = 6 000 000 pixels.
Q3. b) 50 % de verts, 25 % de rouges, 25 % de bleus : l'œil est plus sensible au vert.
Q4. b) Les trois composantes sont au maximum.
Q5. b) Un octet par composante, donc 3 octets (24 bits).
Q6. a) (200 + 100 + 30) // 3 = 330 // 3 = 110.
Q7. b) C'est le négatif : chaque composante est remplacée par 255 moins sa valeur.
Q8. b) Date, appareil, réglages, définition et souvent la position GPS.
Q9. a) Le JPEG divise la taille par 5 à 10 environ ; RAW garde les données brutes.
Q10. b) On compare l'écart de luminosité à un seuil.
Q11. a) Un objet blanc doit paraître blanc sous le soleil comme sous une lampe.
Q12. b) HDR, mode nuit, mode portrait : l'image finale est en grande partie calculée.

8.10 Fiche de synthèse

Fiche de synthèse

— **Capteur :** Millions de photosites ; le filtre de Bayer (50 % vert, 25 % rouge, 25 % bleu) ne laisse passer qu'une couleur par photosite ; le dématriçage calcule les pixels.

— **Définition :** Nombre de pixels = largeur × hauteur (4 000 × 3 000 = 12 Mpx). La résolution de l'image dépend des réglages, pas seulement du capteur.

— **Couleur :** Pixel (R, V, B), chaque composante de 0 à 255 (1 octet) : 3 octets par pixel, 16,7 millions de couleurs. Taille non compressée = pixels × 3 octets ; JPEG compresse.

— **EXIF :** Date, appareil, réglages, définition, GPS : à vérifier et à supprimer si nécessaire avant de partager.

— **Traitement :** Gris : (g, g, g) avec g la moyenne ; négatif : 255 − composante ; contours : écart de luminosité avec les voisins supérieur à un seuil.

— **Algorithmes :** Mise au point, exposition, dématriçage, balance des blancs, réduction du bruit, HDR, compression : l'image finale est calculée.