EPFL

ICC

Programmation

SIE/GCG, **Cours 8**

15 avril 2024

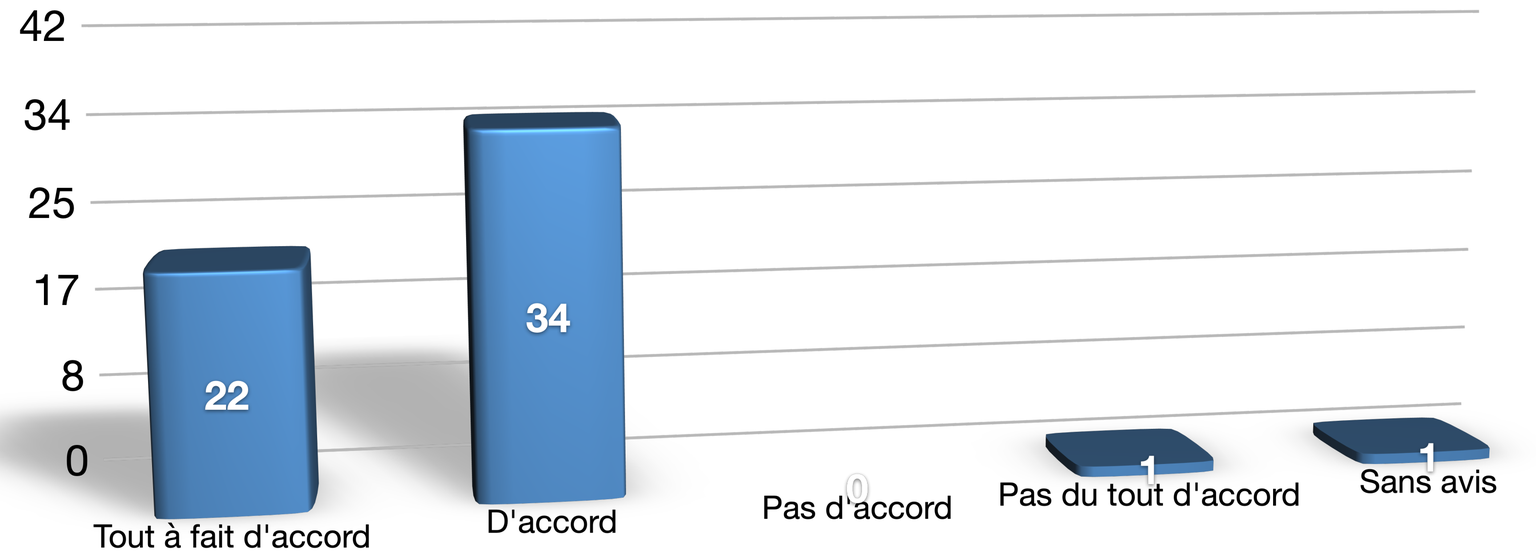
Jean-Philippe Pellet

Previously, on Programmation...

- Types de base en Python: int, float, str, bool
- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Conditions pour exécuter du code selon la valeur d'une expression booléenne
- Boucles pour exécuter du code plusieurs fois:
  - Boucle `while <condition>: ...`
  - Boucle `for i in range(...): ...`
- Déclaration de fonctions avec type de retour et paramètres:
  - `def calculate_area(r: float) -> float: return ...`
- Utilisation de listes
- Utilisation de sets
- Utilisation de dictionnaires
- Déclaration de classes: `@dataclass class Rectangle: …`
- Midterm! Résultats en cours de semaine prochaine

Évaluation du cours

«Le déroulement du cours permet ma formation et un climat de classe approprié.»



Pourcentages cumulés: **38%** **97%** **97%**

Participation: 58/143, dont 27 avec remarque

Merci pour votre feedback!

C'est un plaisir pour Olivier et moi de donner ce cours dans de telles conditions!

Feedback complet

- Cours de programmation super, le prof motivé nous motive justement à apprendre! Par contre pour la théorie, il n'est PAS NORMAL que nous soyons dans un amphi à 300 places pour plus de 500 inscrits !!! Ce n'est pas digne d'une des meilleurs université européenne !! Ce cours serait également plus logique en anglais, car le code s'écrit en anglais et non pas en francais
- Cours intéressants, bien enseignés. ICC-T parfois pas très clair sur ce qui sera demandé à l'examen.
- Cours théorique pratique parfait, cours théorique très intéressant mais trop lent dans l’ensemble .
- Explications claires. Pour informatique programmation, la matiere pourrait etre survolee un peu plus rapidement mais je ne trouve pas ca derangant, mais je peux ressentir un desinteret par certains eleves a certains moments du cours, ce qui derange le climat de la classe, mais sinon cours clair et exemplification des concepts claire egalement.
- ICC-P: très bon cours, bien expliqué, les exemples sont pertinents et on a toutes les informations nécessaire pour réussir les exercices. ICC-T : bon cours, intéressant avec de bons exemples. Peut-être un peu lent certaines fois
- Il me semble un peu inutile et confondant qu'il y a qu'un cours en français et pas en anglais, car il faut souvent apprendre deux mots techniques pour les mêmes choses (anglais et français) et enfin la programmation pratique souvent se fait qu'en anglais.
- J'apprécie énormément ce cours, que je trouve très constructif. Les notions sont très bien présentées en cours et le support de cours aussi. Je souligne le mooc en ICC-T que je trouve très agréable et synthétique. Pour la partie théorique, je trouve que les séries sont en bonne cohérence avec le cours et apprécie énormément que les assistants restent jusqu'à 18h, ce temps supplémentaire est souvent nécessaire, merci. Les assistants sont également très agréables et expliquent merveilleusement bien. Ils sont à l'écoute et prennent le temps de tout bien expliquer, même des notions de cours. Quant à la partie programmation, personnellement j'aime beaucoup les séries, je les trouve très intéressantes. Cependant, il est vrai qu'elles sont parfois difficiles à terminer en 2h et les assistants nous font bien comprendre lorsque leurs deux heures d'assistanat sont terminées. Aussi, certains assistants nous disent souvent quoi écrire au lieu de nous expliquer la vraie explication derrière les lignes de code lorsqu'on est perdus. Je trouve que certains assistants ne sont pas de bonne volonté à tout reprendre certaines notions depuis le début alors qu'on est clairement perdus. Certains d'entre nous n'ont jamais fait de programmation avant, écrire en python n'est pas inné. Le cours avance plutôt vite, et je pense qu'on pourrait mieux assimiler toutes les notions et notations si celles ci étaient répétées sur deux séries d'exercices, plutôt que de constamment apprendre plein de nouvelles choses. Je me pose également des questions sur le format du midterm et de l'examen pour la partie programmation. Je trouve que les séries sont souvent assez compliquées car beaucoup de nouvelles notions pas présentées en cours y sont introduites.
- j'apprécie que les cours soient en ligne. pour le cours de théorie il est difficile de réviser le cours parce qu'on sais pas trop ce qu'on doit savoir pour les examens
- Je trouve que le cours est bien fait. Mais est que les exercices en ICC-T représentent bien la difficulté de l’examen que l’on aura ?
- Le cours de ICC-T avec Olivier Lévêque sont moyen mais ce n'est pas la faute du professeur qui fait des cours interactifs et qui ont une bonne structure. Effectivement, c'est un avis totalement subjectif qui me pousse à ne pas beaucoup apprécier ce cours. Néanmoins, plus je viens en cours, plus le professeur arrive à m'intéresser sur des petits détails lorsqu'il parle de problèmes dont les scientifiques sont actuellement en train d'essayer de résoudre. Les résumés et rappels lors du cours sont super utiles. Ensuite le cours ICC-P avec Jean-Philippe Pellet me passionne énormément alors que je m'attendais à l'inverse. Le professeur a une manière incroyable d'enseigner. En effet, au début du semestre j'ai été beaucoup rassuré lorsqu'il nous a dit qu'il reprendrait des bases et qu'il nous enseignerait comme si nous avions jamais fait de programmation avant. Cela est rassurant car on voit souvent la programmation comme un groupe fermé qu'il est difficile d'entrer. Ensuite, les cours sont super bien organisés en "pas à pas" avec des exemples claires et captivants. La séance d'exercices est un plaisir car on y est bien préparé. Aussi je trouve que les résumés au début et à la fin de chaque cours sont très utiles et le titre " 'Previously' on programmation" est très drole.
- Le cours de programmation (ICC-P) se passe très bien, le professeur est dynamique et a vraiment commencé son cours à zéro. Nous avons bien vu toutes les bases de python, ce qui rend les séries faisables. ICC-T est beaucoup plus difficile, il est dur de suivre le cours dû au monde dans l'auditoire. Les attendus de ce cours restent aussi très flou, le pseudo-code étant beaucoup plus souple que python. Il est donc difficile d'écrire des algorithmes en pseudo-code et de comprendre ce qui est vraiment interdit. Mais le professeur reste investi, et la mise à disposition de son MOOC est un vrai plus.
- Les 2 cours sont bien organisés et les explications des professeurs sont claires. Toutefois en ICCP, n'ayant jamais fait de programmation avant, il m'est difficile de réaliser les séries depuis les deux dernières semaines. J'espère que ce lundi, nous aurons plus de précision quant au midterm prochain.
- Les cours et séries d'exos sont bien pensés avec une progression et des explications clairs
- les cours sont bons même si en programmation, la façon de programmer en déclarant toujours les types en python me parait inutile et ne facilite pas la compréhension, je suggère au moins de la rendre facultative à l'examen. Pour la partie théorique, les exercices facultatifs sont amusants continuer :)
- Les cours vont vite en ICC T et le prof part trop du principe qu'on connait déjà certaines choses et donc n'explique pas tout clairement ce qui fait que c'est facile de décrocher, et de ne plus y arriver. Pour ICC-P, ce serait sympa de nous expliquer d'avantage l'évaluation du cours. On nous parle de mini projet et mais sans savoir ce qu'on va devoir faire, si on sera seul ou non.. Il serait bien aussi de nous parler de l'examen écrit, à quoi bon faire les séries sur notre machine alors que l'examen sera écrit ?? De plus, l'utilisation d'un formulaire pour l'examen d'ICC est possible selon les anciens BA2, en quoi consistera-t-il ?
- Les deux matières sont très bien organisées et expliquées, les enseignants expliquent de manière constructive le cours. Les slides sont suffisantes et claires. Pour la partie théorique, les séries sont bien adaptées au cours et les assistants arrivent très bien à nous expliquer les concepts. Cependant pour la partie pratiqué je trouve que les séries ne sont pas forcément en rapport avec le cours et que certaines notions nécessaires à connaître pour faire la série ne sont pas encore vu en cours. Les corrections de ces séries ne sont également pas assez détaillées.
- Malgré le fait que je n'aime pas forcément la programmation, je pense quand même M. Lévêque et M.Pellet sont des très bons professeurs. Ils expliquent très bien et semblent pendre du plaisir à enseigner ce qui se ressent grandement dans l'ambiance général de l'amphi. Les exemples de cours sont très utils notamment en ICC théorique. Je trouve en revanche que parfois les cours de ICC pratique n'ont parfois pas trop de rapport avec ce qui est vu par la suite dans les series, on doit parfois utiliser des fonctions pas encore vues (l.append par exemple deux semaines avant l'avoir vu en cours) . Par ailleurs cela complique donc la comprehension de la correction fourni. Il serait très utile d'avoir un explication pour les parties de codes plus délicates plutôt que juste le code écrit. A part ca je trouve les cours assez intéressants.
- Partie programmation: résumés en début de cours souvent un peu longs comparés au temps de cours
- Pour ICC P : n’ayant aucune base en Python, je trouve les exercices souvent compliqués, il est rare que je réussisse à les effectuer par moi-même… Cependant le format du cours est appréciable.
- prof programmation super
- Programmation : le prof est top, il explique bien. le ratio 1h de cours et 2h d'exos est bien mais je trouve quand même que les séries sont longues, surtout si on a jamais fait de programmation avant, c'est compliqué de tout assimilé en si peu de temps et devoir tout gérer. j'aurais aimé avoir 1 heure de plus en ex avec des assistants. Théorie : le prof est bien, il réexplique les concepts plusieurs fois pour qu'on comprenne bien et je trouve ça top. les séries d'ex sont biens.
- Programmation: cours très interessant, bien donné. Cependant, les séries sont trop dures comparées au cours surtout qu'on a uniquement 45 minutes de cours par 2h d'exercices. Les assistants ne sont pas souvent de bonne volonté et présents pour nous aider. Ils ont l'air agacés dès qu'on les appelle et ne veulent pas pendre le temps de bien nous expliquer (à part 2/3 exceptions). Théorie: cours bien donné, c'est incroyable de nous laisser l'opportunité d'avoir plus de temps d'exercices avec les assistants.
- RAS, partie programmation comme théorique sont bien enseignées, les séries en adéquation avec le cours et les corrections bien explicitées. Un apport de précisions sur les modalités d'examen, et notamment sur les questions-types et format de ce dernier serait vraiment super !
- The programming part is very interesting in the course and the exercises have a good amount of topics out of the course and new concepts to develop while programming.
- tres bien
- très bon cours en prog, théorie cool mais difficile de voir à quoi ressemblera l'examen
- Très satisfaite des deux cours, le rythme est idéal.

Feedback, résumé *(seul. ICC-P)*

Bien	Pas bien	Commentaire/action
Enthousiasme, plaisir à enseigner	*Trop lent (début du cours)*	*Je vais aller plus vite*
Commencé de zéro	*Séries un peu longues, couvrent plus que dans le cours*	*Je veillerai à raccourcir un peu*
Démos de code	*Corrections pas claires*	*Corrigés vidéo*
Progression appropriée	*Assistants pas toujours très dispo*	*Surpris… Je fais passer le message*
Bon rythme	*Evaluation pas claire*	*On clarifie encore tout à l'heure*

Cours de cette semaine

Manipulation d'images
Introduction au miniprojet

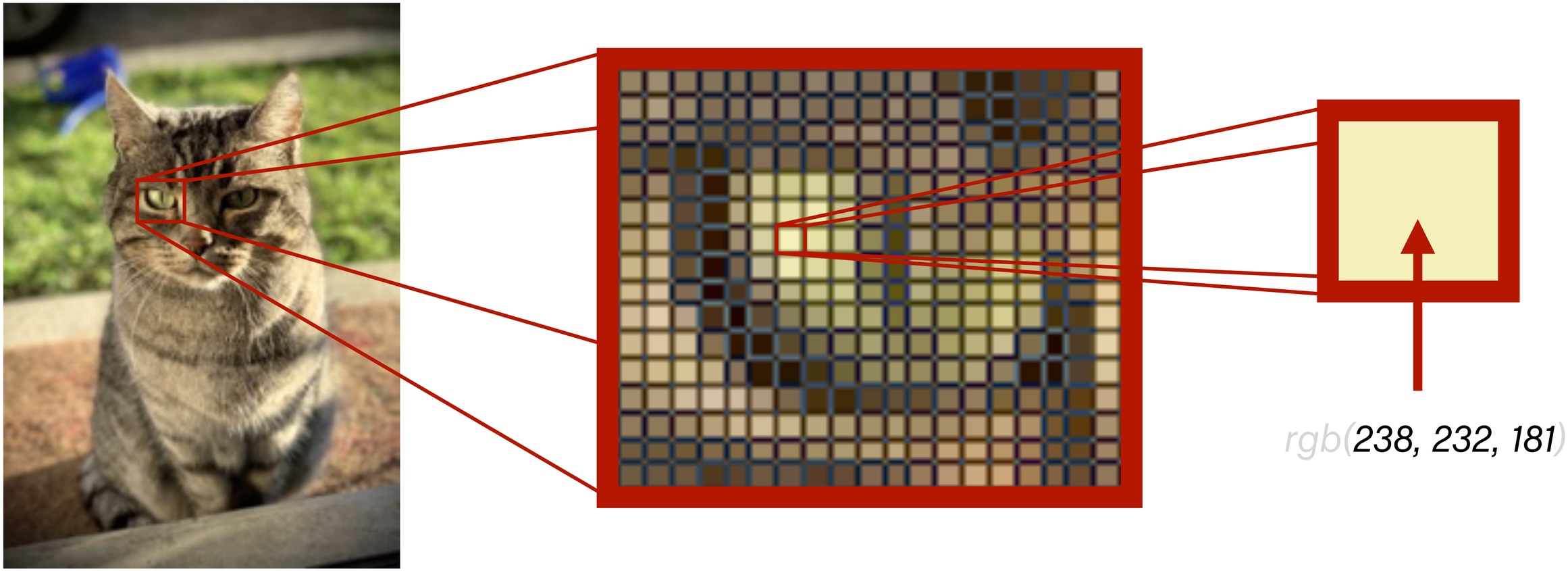
Cours de cette semaine

Manipulation d'images

Introduction au miniprojet

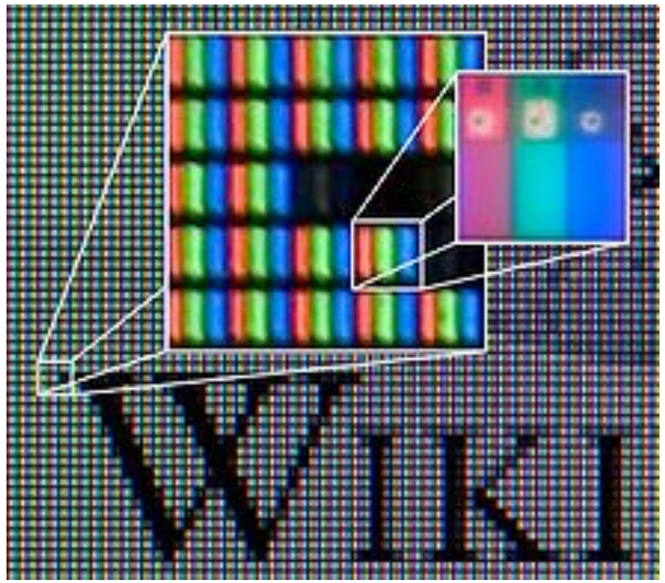
Représentation des images et couleurs

- Images matricielles (JPEG, PNG, …), images vectorielles (SVG, …)
- Image matricielle = grille de pixels (*picture element*)



Stockage d'un pixel

- En couleur: 3 composantes — rouge, vert, bleu
  - Chaque composante va de 0 à 255, donc stockage sur 8 bits
  - 24 bits par pixel, 16'777'216 couleurs possibles (2^{24})



- En niveaux de gris, une seule composante
  - 8 bits par pixel

Manipulation d'images en Python

- Python n'est pas spécialement efficace pour traiter de grandes quantités de données avec les types de base (liste, etc.)
  - On paie la flexibilité du langage en mémoire et en temps d'exécution
  - Il nous faut une structure de données qui fournisse des listes (multidimensionnelles) homogènes
- numpy — Numerical Python
  - Fournit des listes rapides
  - Dim. 0: scalaire; dim. 1: vecteur; dim. 2: matrice; 3+: tenseur…
    - ➡ Ou: 2 dimensions: images en niveaux de gris; 3 dimensions: images RGG
- Bibliothèque Pillow, décode et encode les images et les manipule avec numpy
- On utilisera des fonctions auxiliaires qui elles-mêmes utiliseront Pillow et numpy

Démo

Création d'images en niveaux de gris

Module *miniprojectutils.py* à télécharger depuis Moodle; fournit les fonctions faciles d'accès

```
from miniprojectutils import *

img = new_image_grey(10, 10)
print(img)
save_image(img, "black.png")
```

Nouvelle image en niveaux de gris de 10 × 10 pixels

Affiche les valeurs brutes: une matrice de 10 × 10 zéros = 100 pixels noirs

Enregistre ces données au format PNG (sans pertes). Pour le miniprojet: plutôt .jpg (avec pertes, mais fichiers plus petits)

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

```
for row in range(10):
    for col in range(10):
        img[row, col] = (row + 1) * 20
```

Équivalent! Le ":" sélectionne toute la ligne!

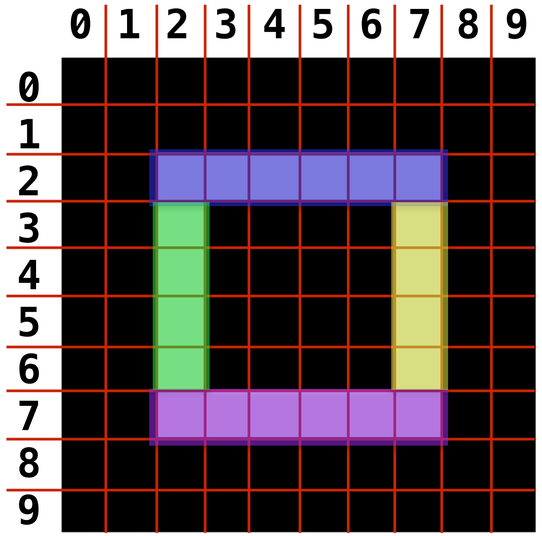
```
for row in range(10):
    img[row, :] = (row + 1) * 20
```

```
print(img)
save_image(img, "gradient.png")
```

```
[[ 20  20  20  20  20  20  20  20  20  20]
 [ 40  40  40  40  40  40  40  40  40  40]
 [ 60  60  60  60  60  60  60  60  60  60]
 [ 80  80  80  80  80  80  80  80  80  80]
 [100 100 100 100 100 100 100 100 100 100]
 [120 120 120 120 120 120 120 120 120 120]
 [140 140 140 140 140 140 140 140 140 140]
 [160 160 160 160 160 160 160 160 160 160]
 [180 180 180 180 180 180 180 180 180 180]
 [200 200 200 200 200 200 200 200 200 200]]
```

Dessiner des formes

Tâche: générer l'image ci-contre ⇒




```
img = new_image_grey(10, 10)

for col in range(2, 8):
    img[2, col] = 255

for row in range(3, 7):
    img[row, 2] = 255
    img[row, 7] = 255

for col in range(2, 8):
    img[7, col] = 255

print(img)
save_image(img, "box.png")
```

Écriture des zones blanches pixel par pixel

Ligne par ligne

```
img[2, 2:8] = 255
img[3:7, 2] = 255
img[3:7, 7] = 255
img[7, 2:8] = 255
```

```
img[[2, 7], 2:8] = 255
img[3:7, [2, 7]] = 255
```

«Zone par zone»

Des images couleur

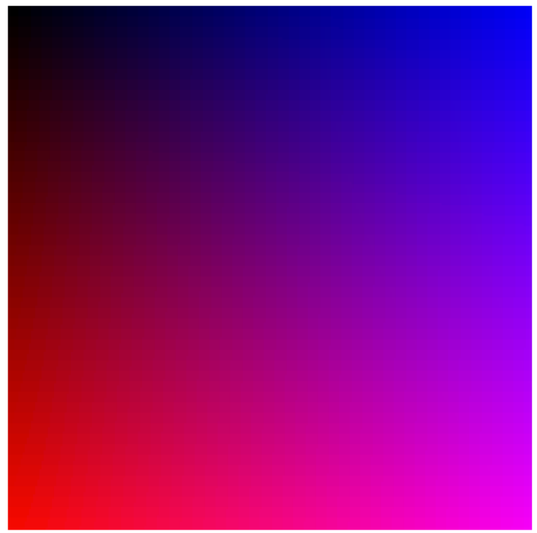
Image RGB: chaque pixel est maintenant non plus un nombre, mais 3, pour les 3 composantes RGB

```
img = new_image_rgb(250, 250)

print(img[125, 125])

for row in range(250):
    for col in range(250):
        img[row, col] = [row, 0, col]

save_image(img, "output.png")
```

Le pixel (125, 125) est noir, représenté ici par la liste de 3 éléments: `[0 0 0]`

Pour modifier un pixel donné, on ne lui affecte pas une seule valeur, mais de nouveau, une liste de 3 composantes

Que génère ce code?



Subscripting dans numpy

- Dans `img[x, y]`, `x` et `y` peuvent être:
  - Une coordonnée sous forme de nombre entier: `0` `10` `i` *etc.*
  - Une plage: `0:10` `5:` `10:` `:` *etc.*
  - Une liste de coordonnées: `[0, 2, 8]` *etc.*
- La lecture d'un pixel via `img[x, y]` donne:
  - Un nombre entier si l'image est en niveaux de gris
  - Une liste de trois nombres si l'image est en RGB
- Dans l'écriture d'un pixel via `img[x, y] = p`, `p` doit être:
  - Un nombre entier si l'image est en niveaux de gris
  - Une liste de trois nombres si l'image est en RGB

Cours de cette semaine

Manipulation d'images

Introduction au miniprojet

Miniprojet: administratif

- **10%** de la note (midterm: 15%, examen final: 75%)
  - Examen final: plus de questions de ICC-T (⅔) que ICC-P (⅓)
- Possible de travailler par deux ou tout·e seul·e
  - Groupe à indiquer via lien Moodle qui sera rendu disponible avant la soumission
- Charge horaire estimée: 10 heures
  - 4 heures cette semaine, puis 3 × 2 heures
  - Attention, c'est une **estimation**!

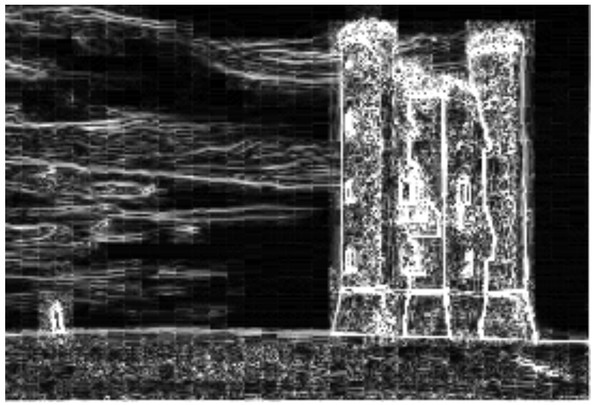
Miniprojet

Selon une idée et projet original de Jamila Sam et Barbara Jobstmann

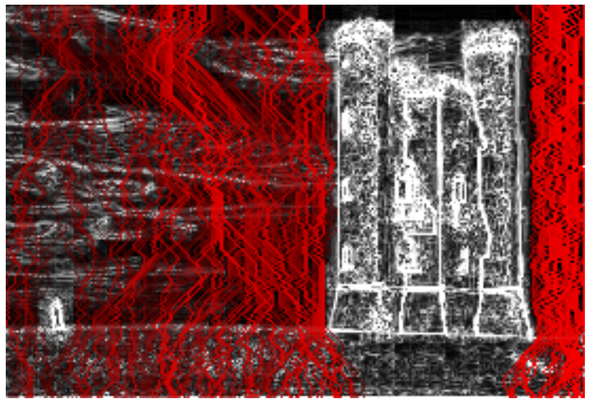
Original

Conversion en niveaux de gris et lissage

Détection des pixels qui sont porteurs de plus ou moins d'information par rapport à leurs voisins avec un filtre Sobel



Détection de tracés de haut en bas qui passent par des pixels de basse information



Suppression de ces tracés dans l'image originale

Résultat

Miniprojet

Simplification du code de base du miniprojet qui est sur Moodle

```python
def seam_carving(img_path: str, num_cols: int) -> None:
    name, ext = split_name_ext(img_path)
    folder = name + os.path.sep
    os.makedirs(folder, exist_ok=True)

    img = load_image(img_path)

    img_grey = to_grayscale(img)
    save_image(img_grey, folder + "grey" + ext)

    img_grey = smoothen(img_grey)
    save_image(img_grey, folder + "smooth" + ext)

    img_grey = sobel(img_grey)
    save_image(img_grey, folder + "sobel" + ext)
    img_grey = np.uint8(img_grey)

    for i in range(num_cols):
        seam = find_seam(img_grey)

        img_highlight = highlight_seam(img, seam)
        save_image(img_highlight, folder + f"highlight_{i}" + ext)
        img_grey_highlight = highlight_seam(img_grey, seam)
        save_image(img_grey_highlight, folder + f"highlight_{i}_grey" + ext)

        img = remove_seam(img, seam)
        save_image(img, folder + f"step_{i}" + ext)
        img_grey = remove_seam(img_grey, seam)
```

Création d'un dossier pour stocker le résultat

Chargement de l'image

Conversion en niveaux de gris et sauvegarde intermédiaire

Lissage de l'image (+ sauvegarde)

Application du filtre Sobel (+ sauvegarde)

Boucle qui se répète autant de fois qu'on veut enlever de colonnes

On repère un *seam* — *semaine prochaine*

On enregistre des images intermédiaires où on montre où se trouve le *seam* détecté

On supprime le *seam* de l'image (et aussi de l'image en gris) et on enregistre le résultat, puis on recommence la boucle si nécessaire

Miniprojet: votre travail

```python
def rgb_to_grey(r: int, g: int, b: int) -> int:
    """Convert an RGB color to a greyscale value."""
    return ...  # TODO
```

Convertit un pixel avec ses 3 composantes RGB en un seul niveau de gris (selon formule de la donnée)

```python
def to_grayscale(img: Image) -> Image:
    """Convert the given image to grayscale."""
    print("   Converting to grayscale...")
    return ...  # TODO
```

Convertit une image RGB entière en une image de même taille en niveau de gris

```python
def clamp_index(index: int, length: int) -> int:
    """Return the index, clamped to the range [0, length-1]."""
    return ...  # TODO
```

Retourne un index pas plus petit que 0 et pas plus grand que *length* – 1

```python
def apply_kernel(img_grey: Image, kernel: Kernel) -> Image:
    """Apply a kernel to an image."""
    return ...  # TODO
```

Applique un *kernel* à une image (détails dans la donnée)

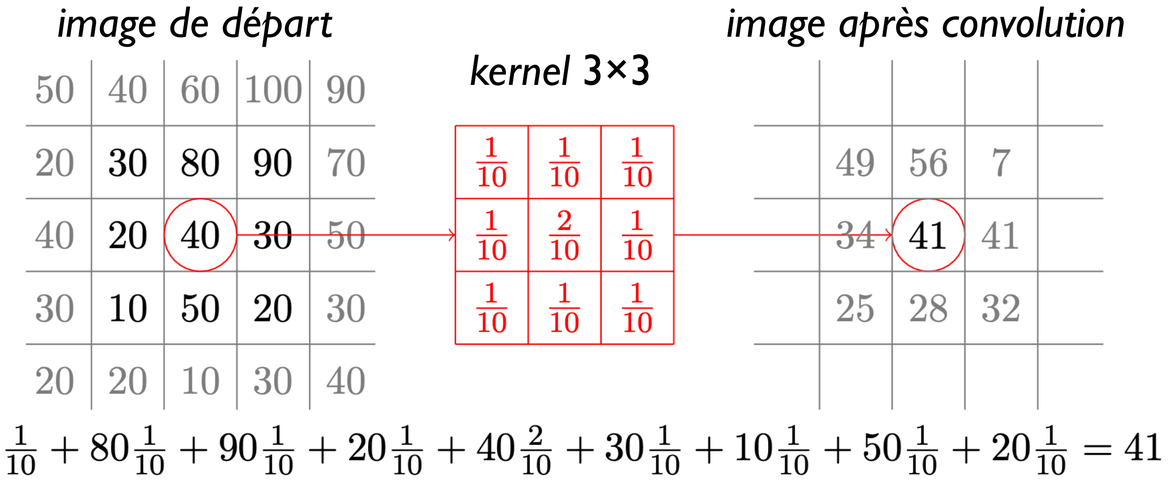
```python
def find_seam(img_grey: Image) -> Seam:
    """Find the seam with the lowest energy."""
    print("   Finding seam...")
    ...
```

Cherche un *seam* dans une image (dont on part du principe qu'elle déjà passée par le lissage et le Sobel)

intro théorique la semaine prochaine; sera subdivisé en 4 fonctions

Convolution

- **`smoothen`** and **`sobel`** ne sont pas à implémenter directement…
  - Car ils sont des cas particulier de **`apply_kernel`**
  - Ceci implémente une opération de **convolution** sur l'image
- La convolution d'une image crée une nouvelle image où chaque pixel est le résultat d'une somme pondérée des pixels voisins
  - La façon de faire cette pondération est déterminée par la **matrice de convolution**, ou **kernel**
  - Exemple du lissage: il s'agit de faire une sorte de «moyenne 2D» des pixels voisins



Que faire dans les bords?

- Pour calculer chaque nouveau pixel, on a besoin des pixels environnants
- Que faire des les bords?
  - On utilise la valeur du pixel le plus proche

?	?	?	
?	40	30	50
?	50	20	30
	10	30	40

40	40	30	
40	40	30	50
50	50	20	30
	10	30	40

Pour calculer le nouveau pixel en position (0, 0),
on simule des pixels de bord supplémentaires lorsqu'on applique le kernel

Kernels

Smoothen

$$\begin{bmatrix} \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{2}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \end{bmatrix}$$

Moyenne des pixels environnants, avec un poids légèrement plus haut pour le pixel central

⇓

Original

Sobel X

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

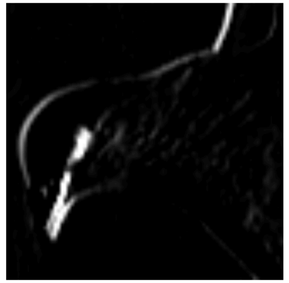
Sobel Y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

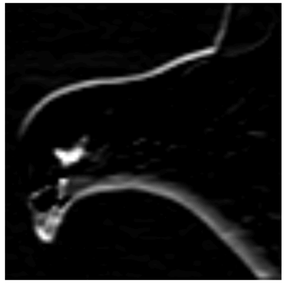
Calcule en quelque sorte une approximation du gradient de l'image, sur l'axe X et sur l'axe Y.

Un gradient de 0 veut dire que les pixels ne changent pas trop le long d'un axe, donc peu d'information. Un haut gradient indique des changements d'intensité des pixels, donc beaucoup d'information.

⇓



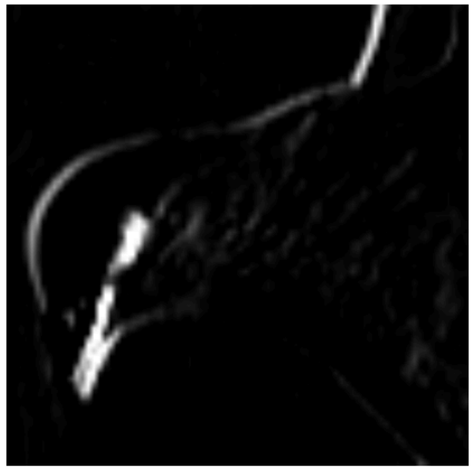
⇓



Sobel, suite

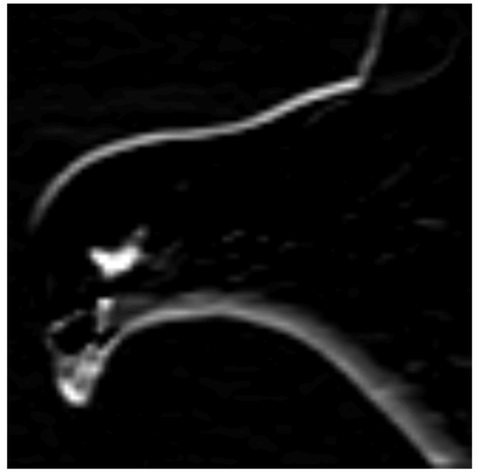
- Comment combiner cette mesure du changement horizontal (Sobel X) et du changement vertical (Sobel Y)?

Composante X
du gradient



x

Composante Y
du gradient



y

⇒

***Norme** du gradient*

$\sqrt{x^2 + y^2}$

(à calculer bien sûr pour chaque pixel)

Implémentation de smoothen

```
def smoothen(img_grey: Image) -> Image:
    """Smooth the image using a 3x3 kernel."""
    print("   Smoothening image...")
    kernel_smooth = np.array([
        [1, 1, 1],
        [1, 2, 1],
        [1, 1, 1],
    ]) / 10

    return apply_kernel(img_grey, kernel_smooth)
```

Définit le kernel de lissage. On utilise un np.array plutôt que des listes «normales» parce que c'est plus efficace

… et parce qu'on peut d'un coup diviser tous les éléments par 10 comme ceci!

Ensuite, le résultat du lissage est «simplement» le résultat de la convolution de l'image en paramètre avec le kernel de lissage

Implémentation de sobel

```
def sobel(img_grey: Image) -> Image:
    """Apply the Sobel filter to the image."""
    print("   Sobel...")
    kernel_sobel_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1],
    ])
    sobel_x = apply_kernel(img_grey, kernel_sobel_x)
    kernel_sobel_y = np.array([
        [-1, -2, -1],
        [ 0,  0,  0],
        [ 1,  2,  1],
    ])
    sobel_y = apply_kernel(img_grey, kernel_sobel_y)
    result = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)
    return result
```

On fait la convolution avec Sobel X pour calculer la composante horizontale

On fait la convolution avec Sobel Y pour calculer la composante verticale

On prend la racine de la somme des carrés (opération faite ici d'un coup sur tous les éléments!)

Cours de la semaine prochaine

Algorithme pour trouver le seam de moindre énergie

Résumé Cours 9

- Les images en Python sont (pour notre cas) représentés par structures bidimensionnelles (niveaux de gris) ou tridimensionnelles de numpy
- On peut lire et écrire des pixels individuels, qui sont soit un nombre unique entre 0 et 255 (niveaux de gris), soit trois nombres (RGB)
- Une convolution sur une image génère une nouvelle image selon une matrice de convolution ou kernel
- En lissant une image en niveaux de gris et en prenant la norme du gradient estimé avec les kernels Sobel X et Sobel Y, on obtient une image donnant une bonne idée de l'information générale véhiculée dans chaque pixel