Introduction à la programmation 1

3 – Logique Booléenne, accès fichier

Julien Deantoni

Contenue de l'UE (non contractuel)

1) Introduction, notion de variables, de type de données simple et structuré
2) Variables, séquence d'instructions en mémoire, Notion d'algorithme, de boucles et de branchements conditionnels
3) **Logique Booléenne et comparaison, lecture/écriture dans un fichier**
4) Écrire des fonctions afin de rendre le code plus lisible et réutilisable
5) Contrôle continu intermédiaire
6) Introduction au web, structure d'une page en web en HTML
7) Modifier le style d'une structure HTML en CSS
8) Résumé
9) Contrôle continu final

Fiche mémotechnique

- https://webusers.i3s.unice.fr/~deantoni/teaching_resources/DS4H/IP1/memoPython/

Créer une variable

- NomVariable **:** type **=** type**(** valeurInitiale **)**
  - `i1: int = int(0)`
  - `i2: int = int('123')`
  - `s1: str = str('zaza')`
  - `s2: str = str(123)`
  - `liste1: list[int] = list[int]( [1,2,3] )`
  - `liste2: list[int] = list[int]([42,18])`

    `type des éléments de la liste`
  - `liste2: list[str] = list[str](['toto','titi'])`
  - `i3: int = i1 + i2`

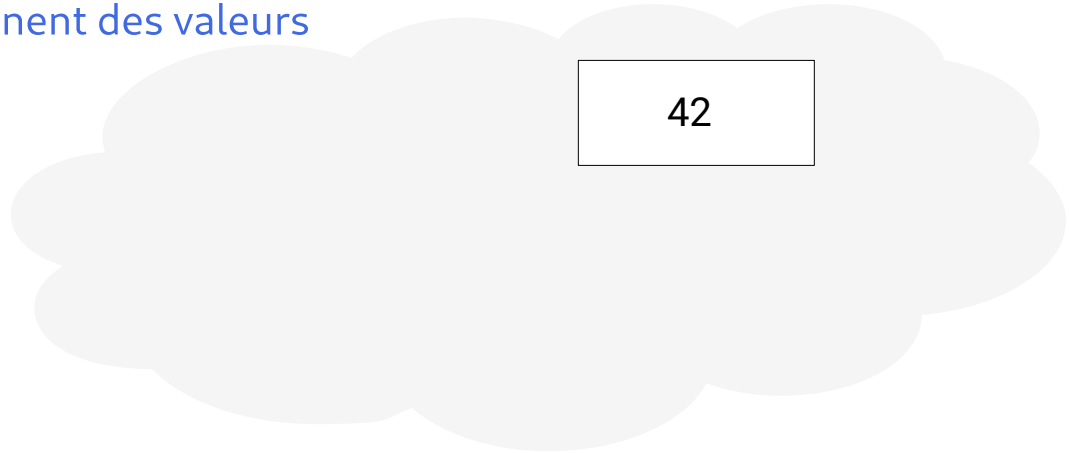
Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les *immutables*.
- Immutable : int, str, float, bool
    - Le contenu des cases mémoires associé aux variables ne peut pas être modifié !
- Mutable : list, Point2D
    - Le contenu des cases mémoires associé aux variables peut être modifié

Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1 i: int = int(42)
2
3 i = i + 1
```

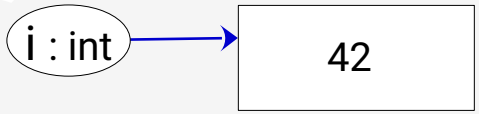


- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1 i: int = int(42)
2
3 i = i + 1
```

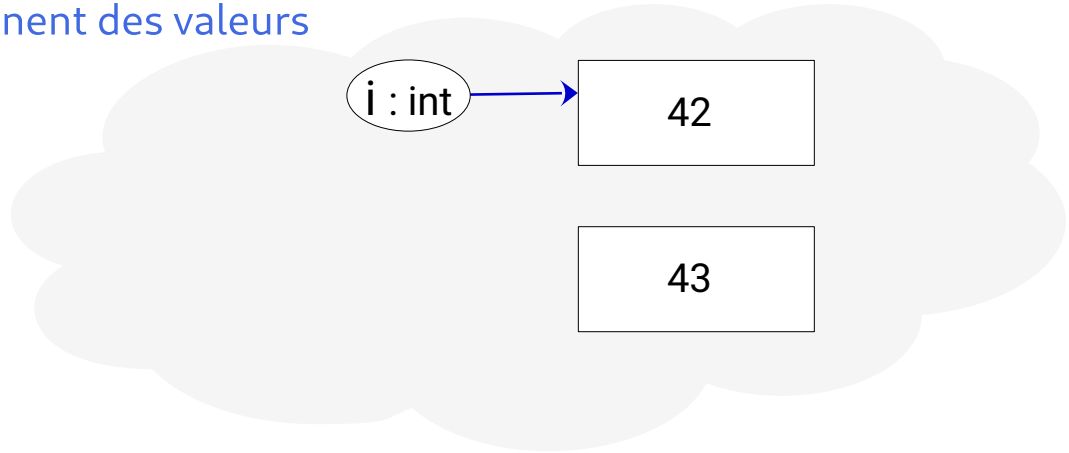


- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1  i: int = int(42)
2
3  i = i + 1
```

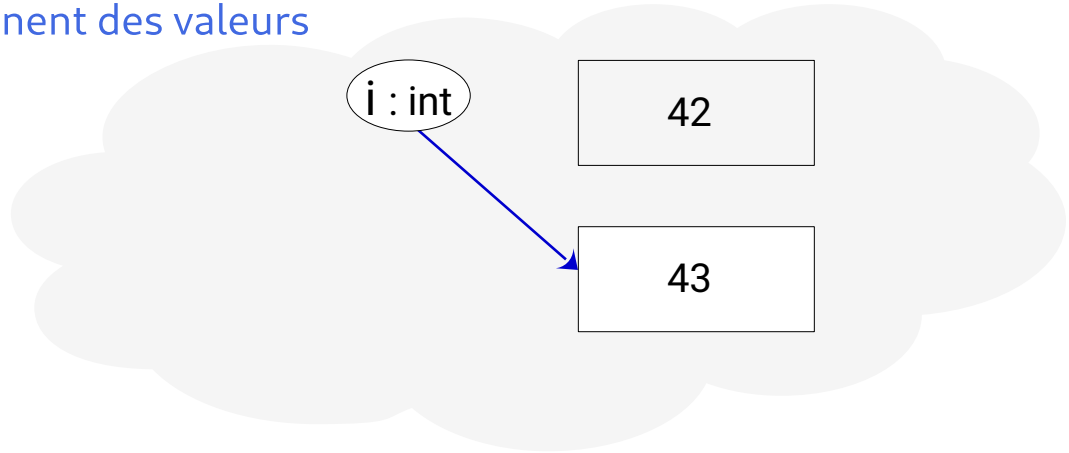


- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1  i: int = int(42)
2
3  i = i + 1
```



- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

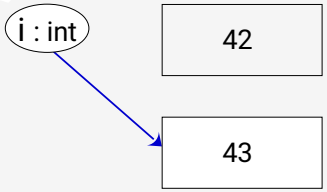
Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.

- Immutable : int, str, float, bool

  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

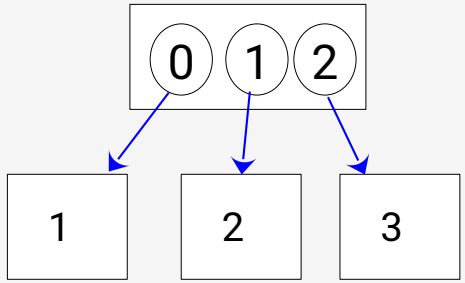
```
1 i: int = int(42)
2
3 i = i + 1
```



- Mutable : list, Point2D, etc

  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```



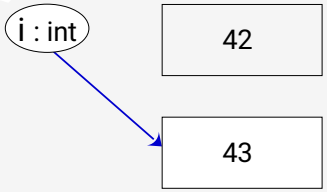
Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.

- Immutable : int, str, float, bool

  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

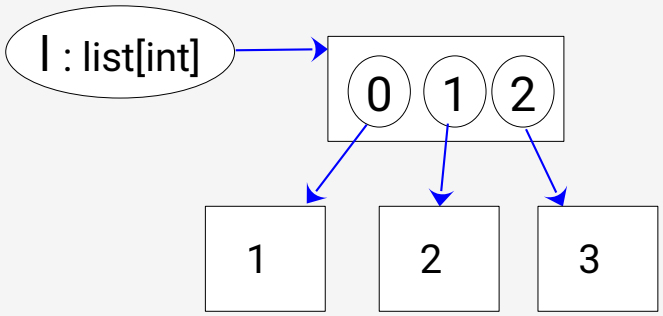
```
1 i: int = int(42)
2
3 i = i + 1
```



- Mutable : list, Point2D, etc

  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```



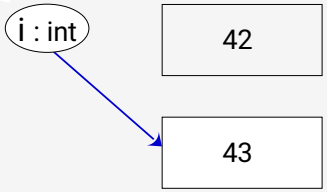
Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.

- Immutable : int, str, float, bool

  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1 i: int = int(42)
2
3 i = i + 1
```

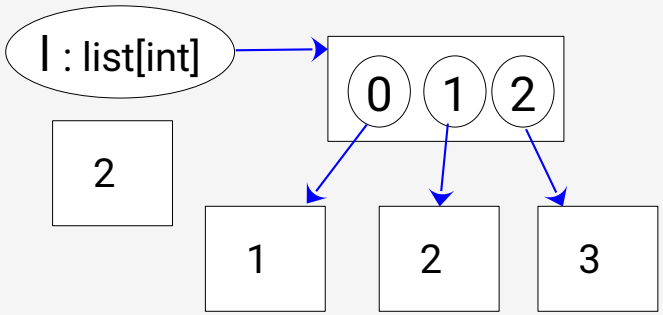


- Mutable : list, Point2D, etc

  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```

La valeur accessible via l'étiquette '0' de 'l'



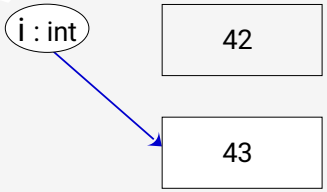
Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.

- Immutable : int, str, float, bool

  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1 i: int = int(42)
2
3 i = i + 1
```

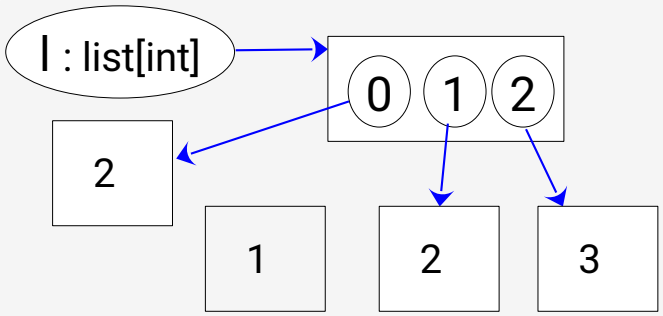


- Mutable : list, Point2D, etc

  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```

L'étiquette '0' de 'l'



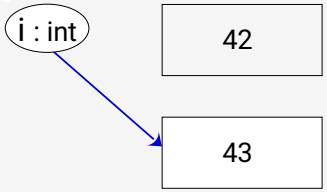
Mutable versus Immutable data types

- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les immutables.

- Immutable : int, str, float, bool

  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

```
1 i: int = int(42)
2
3 i = i + 1
```

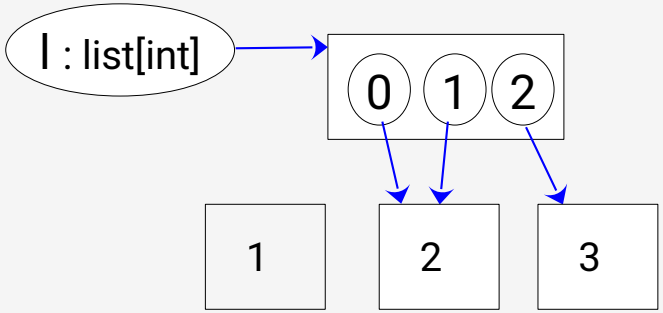


- Mutable : list, Point2D, etc

  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```

L'étiquette '0' de 'l'



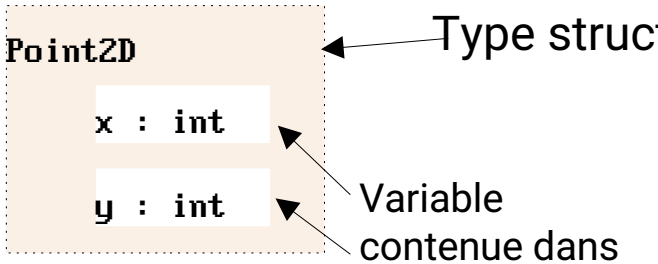
Mutable versus Immutable data types

id(var) à la rescousse

- La fonction `id(var)` donne l'adresse de la variable `var` dans la mémoire
- Immutable : int, str, float, bool, etc → Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs
- Mutable : list, **Point2D**, etc → Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

```
1 pointA: Point2D = Point2D(1,2)
2 print('id de pointA ligne 2:', id(pointA))
3 pointA.x = pointA.x+1
4 print('id de pointA ligne 4:', id(pointA))
```

```
id de pointA ligne 2: 47721368
id de pointA ligne 4: 47721368
```




```
from dataclasses import dataclass

@dataclass
class Point2D:
    x : int
    y : int
```

Annotation pour se simplifier la vie

Mot clé `class` suivi du **nom du type** et de ' **:**'

Ensemble de variables internes typées et ordonnées

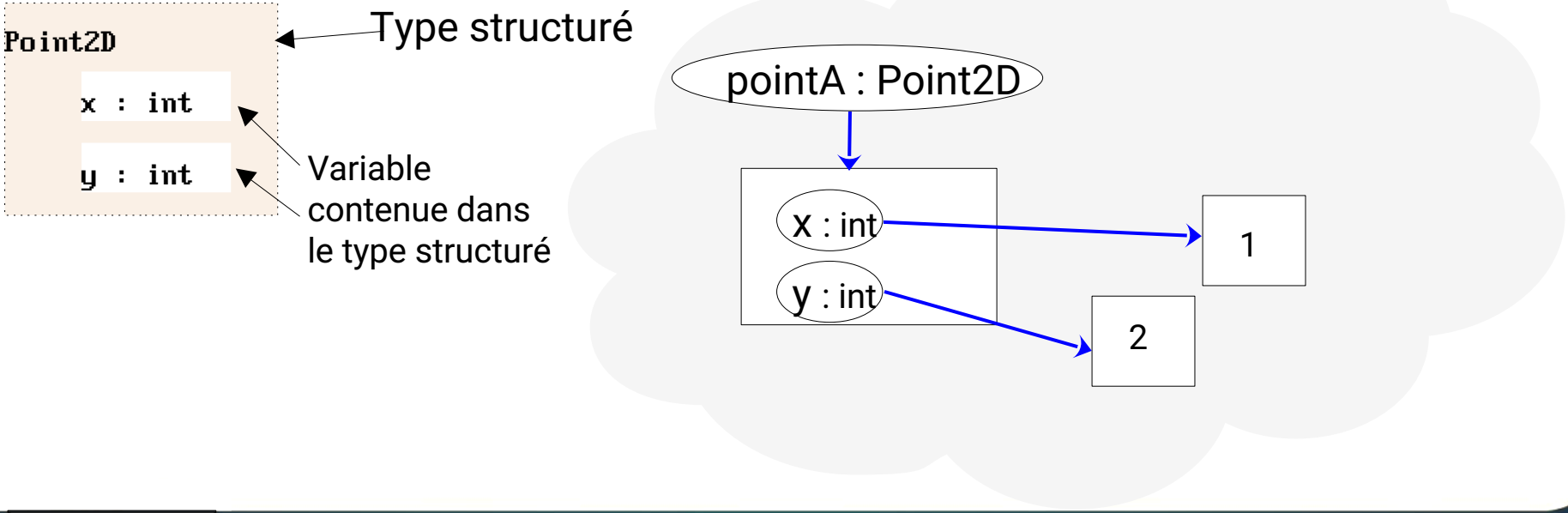
Mutable versus Immutable data types

id(var) à la rescousse

- La fonction `id(var)` donne l'adresse de la variable `var` dans la mémoire
- Immutable : int, str, float, bool
  - Le contenu des cases mémoires associé aux variables ne peut pas être modifié !
- Mutable : list, **Point2D**
  - Le contenu des cases mémoires associé aux variables peut être modifié

```
1  pointA: Point2D = Point2D(1,2)
2  print('id de pointA ligne 2:', id(pointA))
3  pointA.x = pointA.x+1
4  print('id de pointA ligne 4:', id(pointA))
```

```
id de pointA ligne 2: 47721368
id de pointA ligne 4: 47721368
```



Premiers algorithmes

Premiers algorithmes

- Un algorithme est une séquence d'instructions dont l'exécution résout un problème particulier posé à l'avance. Le problème a résoudre peut être généraliste ou spécifique

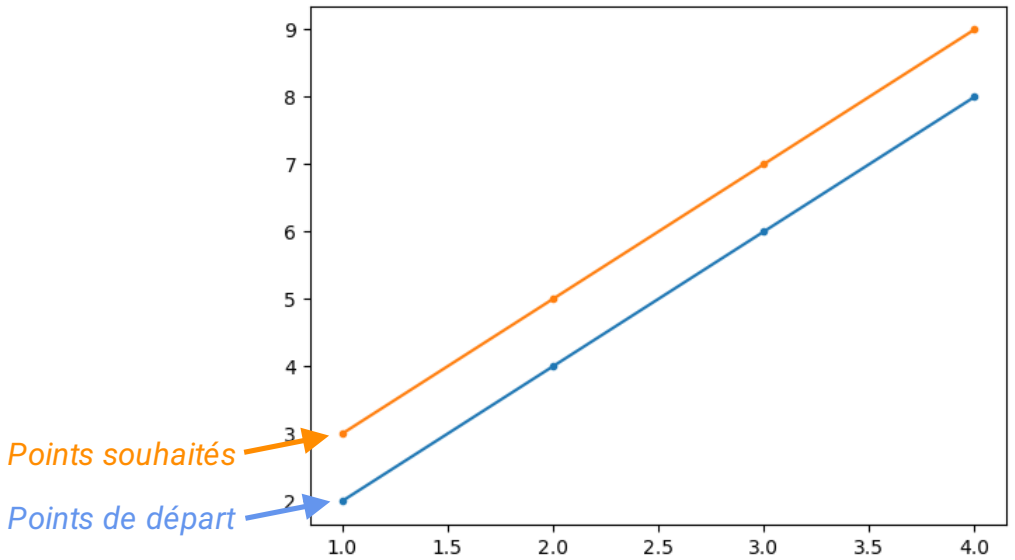
Premiers algorithmes

- On a un ensemble de points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste

```
1
2
3
4
5
6
7
```

Séquence d'instructions

```
[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
   0            1           2           3
```




```
[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]
   0            1           2           3
```

Premiers algorithmes

La boucle While

- On a un ensemble de points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste

```
[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
   0            1           2           3
```

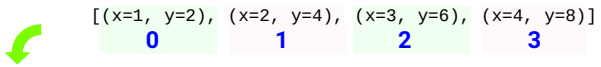
```
1 indice: int = 0
2 while (indice < len(allPoints)):
3     p: Point2D = allPoints[indice]
4     p.y = p.y + 1
5     indice = indice + 1
6 print('traitement fini')
```

```
[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]
   0            1           2           3
```

Premiers algorithmes

La boucle While

- On a un ensemble de points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste




```
1 indice: int = 0
2 while (indice < len(allPoints)):
3     p: Point2D = allPoints[indice]
4     p.y = p.y + 1
5     indice = indice + 1
6 print('traitement fini')
```

Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
6	2	1	undefined	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]

Le ***bloc de code*** identifié par le retrait du bord de la page (indentation) est le code à exécuter **tant que** **la condition** est vraie

Premiers algorithmes

La boucle While

- Une boucle (*loop* en anglais) vous permet de répéter des blocs d'instructions selon vos besoins.
- Il existe différentes manières de définir des boucles en python
- La boucle tant que (*while* en anglais) est définie par un bloc de code à répéter ainsi qu'une expression booléenne appelée condition

```
1 indice: int = 0
2 while (indice < 4):
3     p: Point2D = allPoints[indice]
4     p.y = p.y + 1
5     indice = indice + 1
6 print('traitement fini')
```

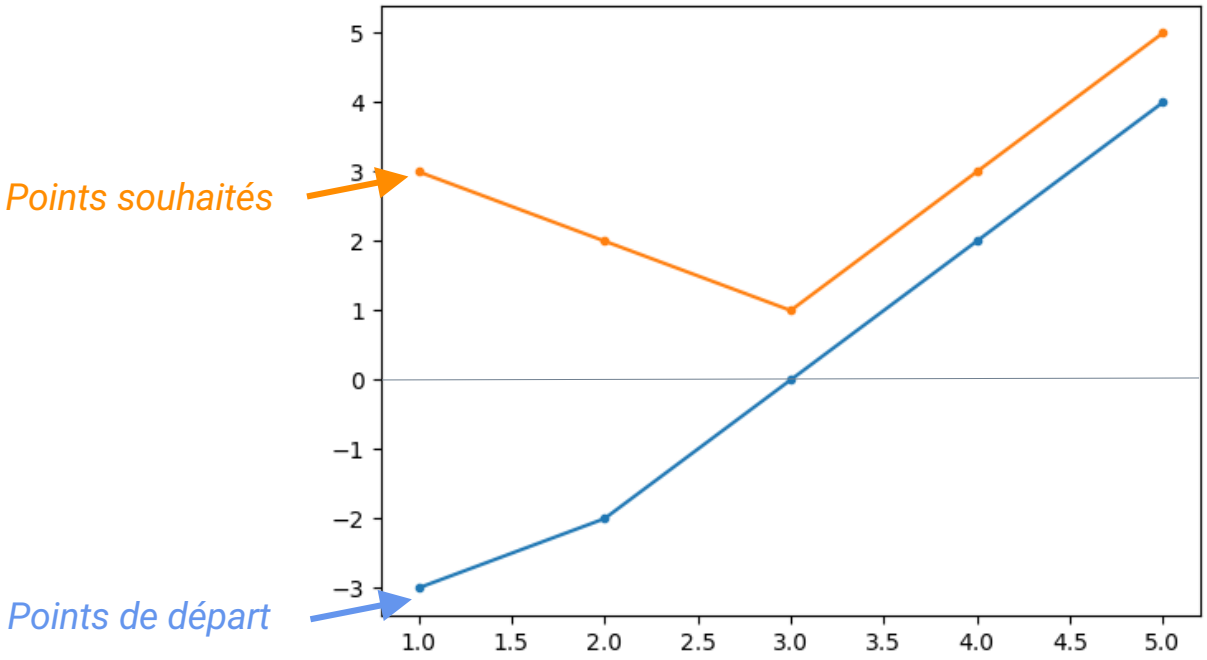
Le ***bloc de code*** identifié par le retrait du bord de la page est le code à exécuter **tant que la condition** est vraie

Lorsque **la condition** est fausse, on va directement à **la ligne qui suit** le bloc de code répété

Premiers algorithmes

Le branchement conditionnel If

- On a un ensemble de points dans une liste nommée `allPoints`.
- Si l'ordonnée d'un point est négative, on désire la remplacer par sa valeur absolue
- Si l'ordonnée d'un point n'est pas négative, alors on lui ajoute 1

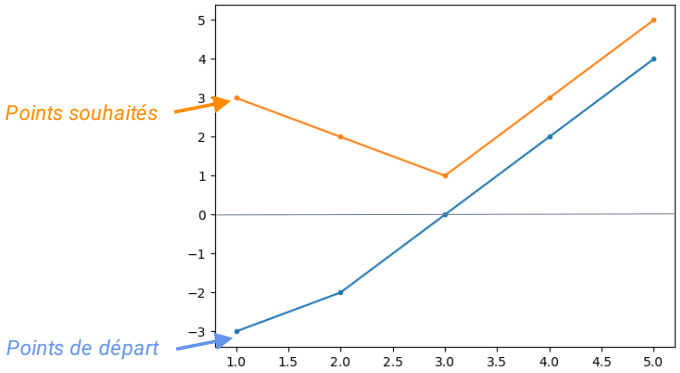


Premiers algorithmes

Le branchement conditionnel If

- On a un ensemble de points dans une liste nommée `allPoints`.
- Si l'ordonnée d'un point est négative, on désire la remplacer par sa valeur absolue
- Si l'ordonnée d'un point n'est pas négative, alors on lui ajoute 1

```
1  indice: int = 0
2  while (indice < len(allPoints)):
3      p: Point2D = allPoints[indice]
4      if (p.y < 0):
5          p.y = -p.y
6      else:
7          p.y = p.y + 1
8      indice = indice + 1
9  print('traitement fini')
```



Si la condition est vraie, alors le bloc de code qui suit la condition est exécuté, **sinon** le bloc de code qui suit le mot clef else est exécuté.

Rappel : les blocs de code sont définis en fonction de l'indentation (du retrait par rapport au bord du fichier)

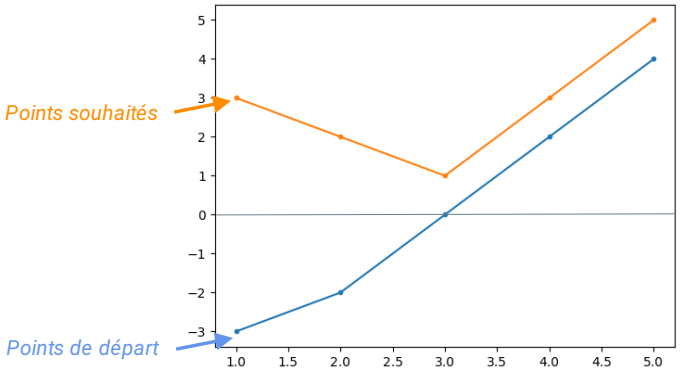
Premiers algorithmes

Le branchement conditionnel If

- On a un ensemble de points dans une liste nommée `allPoints`.
- Si l'ordonnée d'un point est négative, on désire la remplacer par sa valeur absolue
- Si l'ordonnée d'un point n'est pas négative, alors on lui ajoute 1

```
1  indice: int = 0
2  while (indice < len(allPoints)):
3      p: Point2D = allPoints[indice]
4      if (p.y < 0):
5          p.y = -p.y
6      else:
7          p.y = p.y + 1
8      indice = indice + 1
9  print('traitement fini')
```

optionnel



Si la condition est vraie, alors le bloc de code qui suit la condition est exécuté, **sinon** le bloc de code qui suit le mot clef else est exécuté.

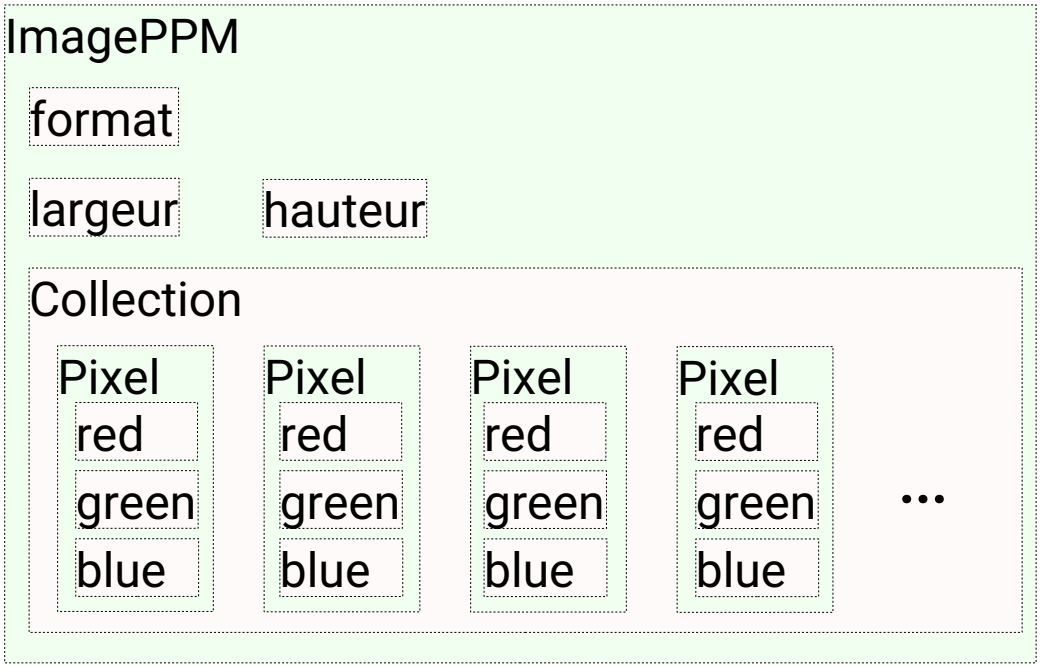
optionnel

Rappel : les blocs de code sont définis en fonction de l'indentation (du retrait par rapport au bord du fichier)

Encodage des images. Un exemple simple

Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
  - une image numérique brute est un ensemble de pixel où chaque pixel a une couleur particulière, encodée par exemple en RGB. De plus la largeur et la hauteur est donnée.



Dans le langage Python

```python
@dataclass
class Pixel:
    r: int
    g: int
    b: int

@dataclass
class ImagePPM:
    format: str
    width:int
    height:int
    pixels:list[Pixel]
```

Manipulation des images

- Soit une variable de type `ImagePPM` nommée `img`. On veut définir un algorithme qui modifie la composante rouge de chaque pixel d'une image en la mettant à 0

Dans le langage Python

```
@dataclass
class Pixel:
    r: int
    g: int
    b: int

@dataclass
class ImagePPM:
    format: str
    width:int
    height:int
    pixels:list[Pixel]
```

```
1  index: int = int(0)
2
3  while (index < len(img.pixels)):
4      p: Pixel = img.pixels[index]
5      p.r = 0
6      index = index + 1
```

Ici on a uniquement modifié la représentation de l'image en mémoire. De plus on a considéré que l'image était déjà en mémoire. Il faut donc

1) lire le fichier sur le disque dur et *peupler* la variable en mémoire vive; et
2) il faut écrire les informations contenues dans la variable dans un fichier sur le disque dur.

Le format PPM P3

- Nous allons utiliser le format PPM et plus particulièrement son sous format P3 pour les images couleurs. L'avantage est que les images sont enregistrées dans un fichier texte sur le disque dur.
- Exemple, une image de 2 pixels sur 2 pixels :

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2 2 255
0 0 0
255 255 255
255 255 255
0 0 0
```

MiniDamier.ppm

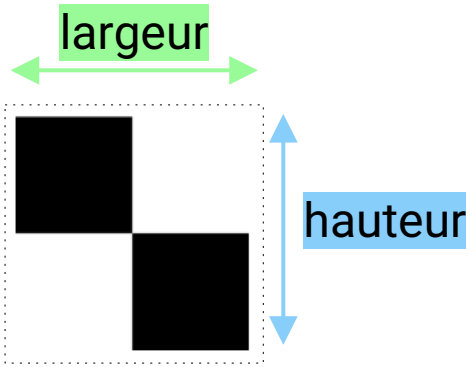
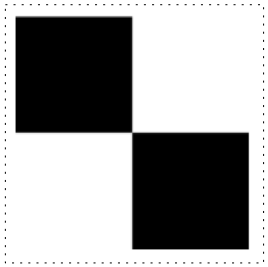


Image miniDamier

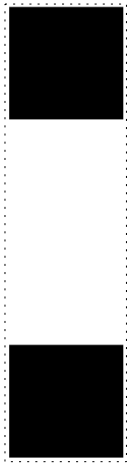
Le format PPM P3

- Nous allons utiliser le format PPM et plus particulièrement son sous format P3 pour les images couleurs. L'avantage est que les images sont enregistrées dans un fichier texte sur le disque dur.
- Exemple, une image de 2 pixels sur 2 pixels :

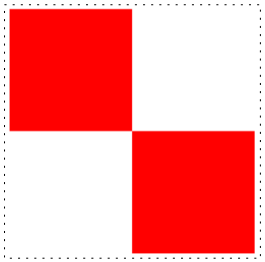
```
P3
2 2 255
0 0 0
255 255 255
255 255 255
0 0 0
```



```
P3
1 4 255
0 0 0
255 255 255
255 255 255
0 0 0
```



```
P3
2 2 255
255 0 0
255 255 255
255 255 255
255 0 0
```



```
P3
4 1 255
0 0 0
255 255 255
255 255 255
0 0 0
```

Le format PPM P3

- Nous allons utiliser le format PPM et plus particulièrement son sous format P3 pour les images couleurs. L'avantage est que les images sont enregistrées dans un fichier texte sur le disque dur.

- Exemple, une image de 2 pixels sur 2 pixels :

Le format PPM ne spécifie pas si l'on doit séparer les valeurs par des espaces ou des retours à la ligne.

⇒ Dans un 1er temps nous considérerons que les valeurs sont toutes séparées par des retours à la ligne

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2
2
255
0
0
0
255
255
255
255
255
255
0
0
0
```

MiniDamier.ppm

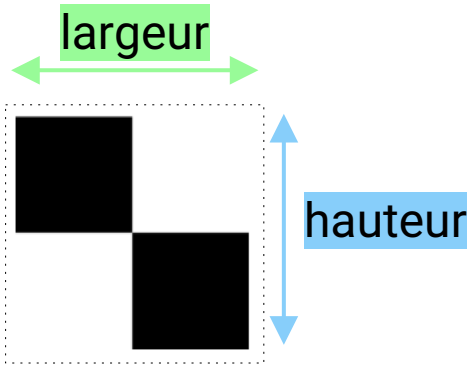


Image miniDamier

Lire dans un fichier en Python

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*

2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères

3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire

4) Fermer le *wrapper* de fichier

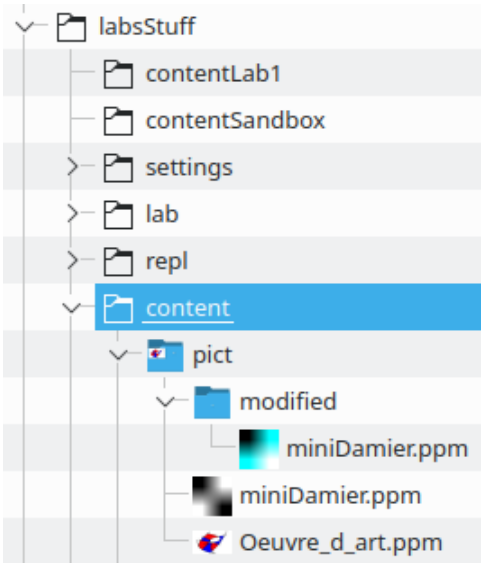
Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```

Chemin vers le fichier à ouvrir



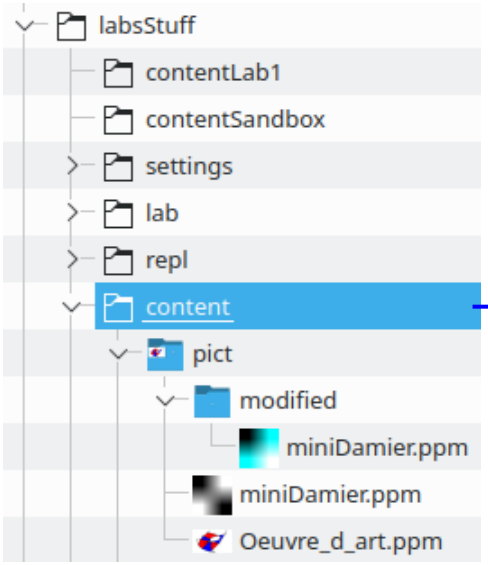
Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```

Le chemin commence dans le dossier qui contient le fichier contenant ce code
⇒ on parle de chemin relatif



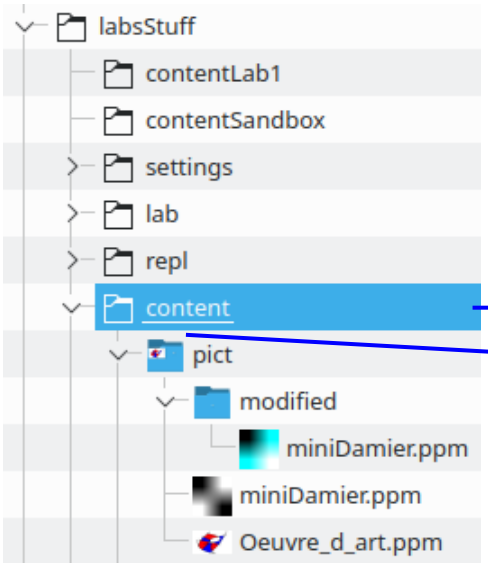
Dossier courant. Nommé .

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```



Dossier courant. Nommé .

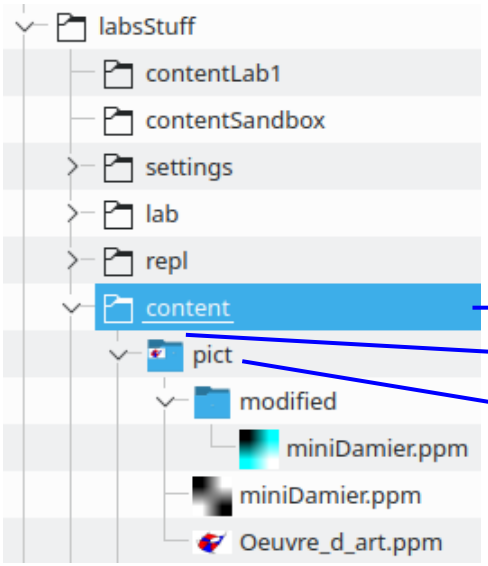
Naviguer dans le dossier avec le séparateur /

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```



Dossier courant. Nommé .

Naviguer dans le dossier avec le séparateur /

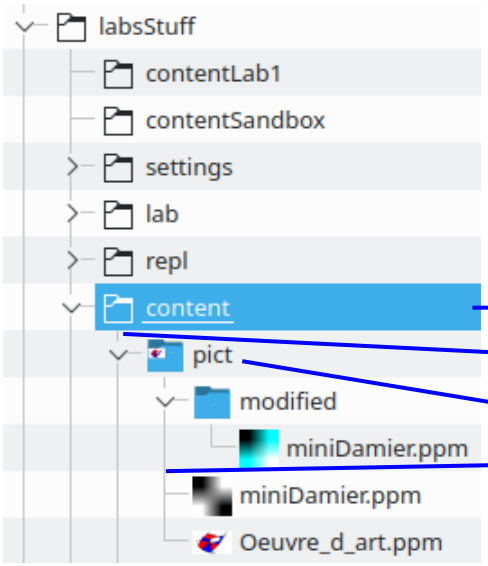
Utiliser le nom du dossier dans lequel aller, ici pict

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```



Dossier courant. Nommé .

Naviguer dans le dossier avec le séparateur /

Utiliser le nom du dossier dans lequel aller, ici pict

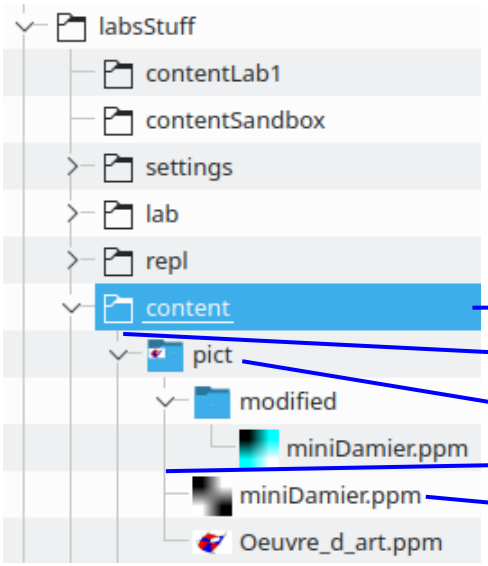
Naviguer dans le dossier avec le séparateur /

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```



Dossier courant. Nommé .

Naviguer dans le dossier avec le séparateur /

Utiliser le nom du dossier dans lequel aller, ici pict

Naviguer dans le dossier avec le séparateur /

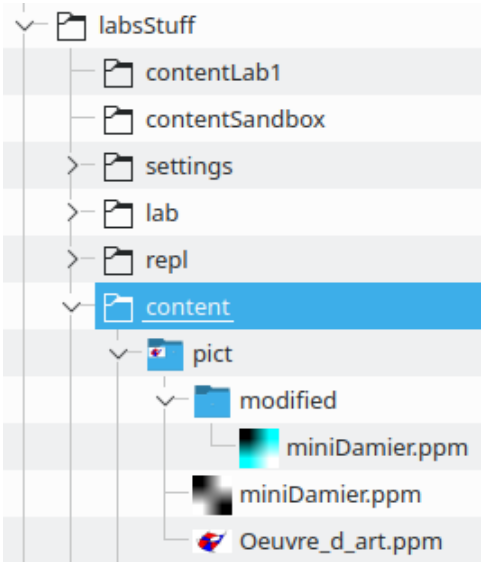
On choisit le fichier dans le dossier atteint, ici miniDamier.ppm

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```

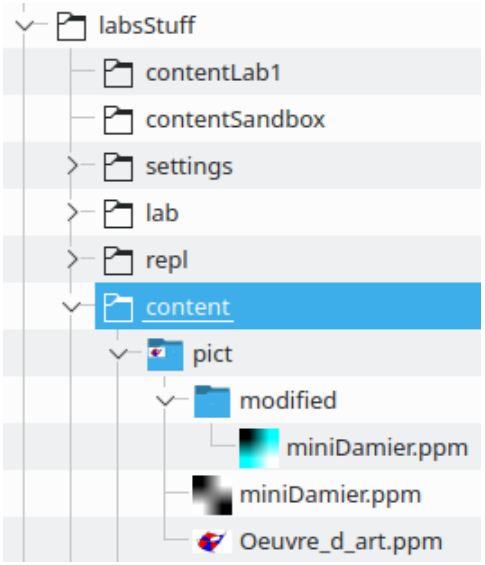


Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```



`'./pict/Oeuvre_d_art.ppm'`

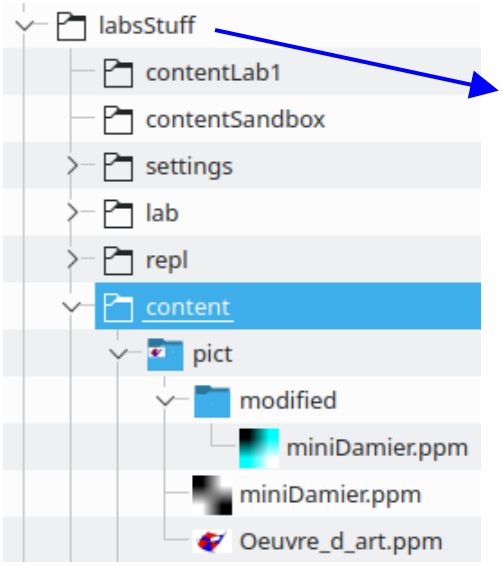
`'./pict/modified/miniDamier.ppm'`

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```



Dossier parent. Nommé `..`

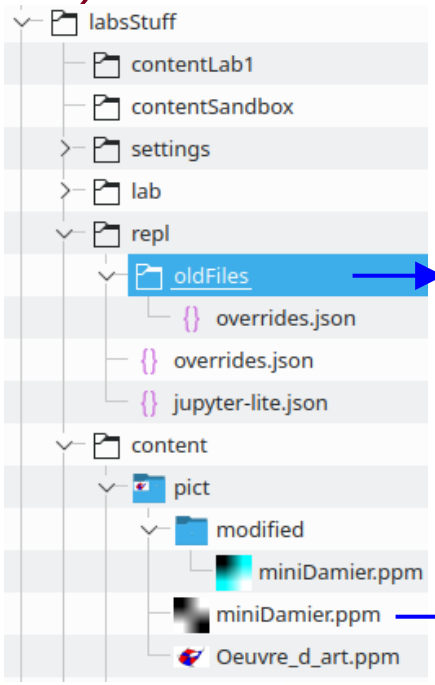
`'../lab/anotherFile.txt'`

On commence dans le dossier 'du dessus' de celui qui contient le fichier contenant ce code

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier




```
fichier: TextIOWrapper = open(
    ' ???????'
    , 'r')
```

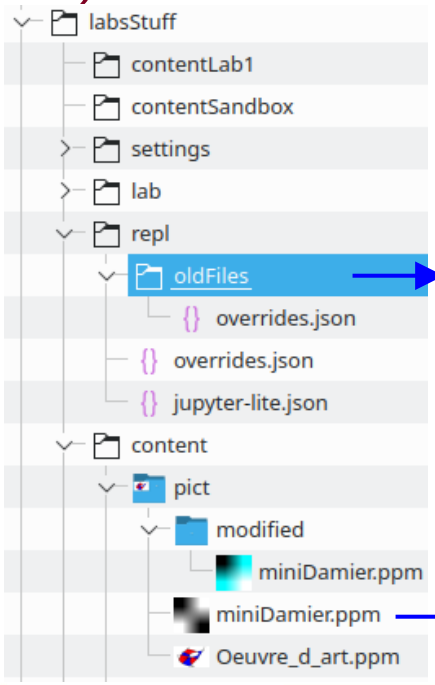
Dossier courant

Comment ouvrir ce miniDamier.ppm ?

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier




```
fichier: TextIOWrapper = open(
            ' ../../content/pict/miniDamier.ppm'
            , 'r')
```

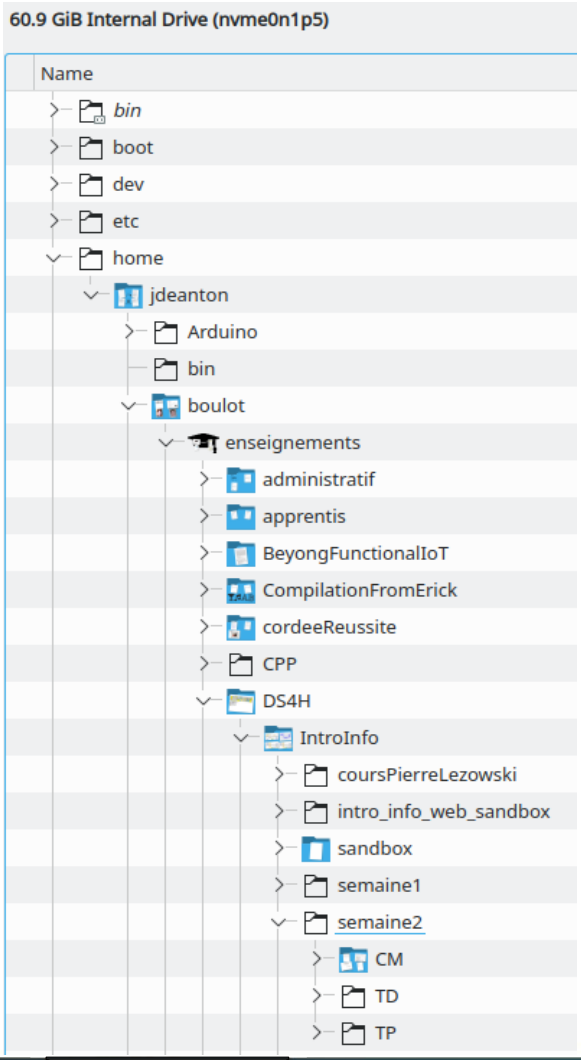
Dossier courant

Comment ouvrir ce miniDamier.ppm ?

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*



Si l'on désire définir un chemin commençant à la *racine d'un disque dur*, alors

- sous linux et Mac on commence le chemin par **/**

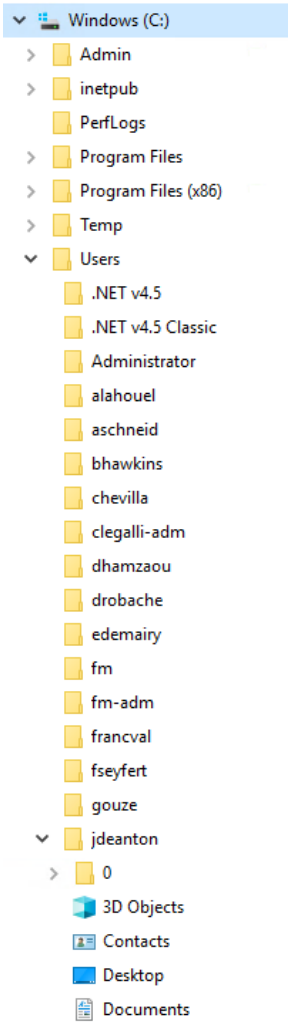
'/home/jdeanton/boulot/enseignements/DS4H/IntroInfo/semaine2/TP/pict/miniDamier.ppm'

⇒ on parle de chemin **absolu**

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*



Si l'on désire définir un chemin commençant à la *racine d'un disque dur*, alors

- sous linux et Mac on commence le chemin par **/**

'/home/jdeanton/boulot/enseignements/DS4H/IntroInfo/semaine2/TP/pict/miniDamier.ppm'

- sous Windows on commence par la lettre représentant le disque suivi de **:**

'C:\Users\jdeanton\Documents\boulot\enseignements\DS4H\IntroInfo\semaine2\TP\pict\miniDamier.ppm'

Sous windows **/** est toujours remplacé par ****

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*

```
'./dossier1/fichier1.txt'
'../documents/maths/correctionTP1.doc'
```
Chemins relatifs Linux ou Mac

```
'.\Pictures\Vacances\2022\futil.jpg'
```
Chemin relatif Windows

```
'/home/nomUtilisateur/temp/todolist.txt'
```
Chemin absolu Linux ou Mac

```
'C:\Users\nomUtilisateur\boulot\aTrier\notesCoursIIW.txt'
```
Chemin absolu Windows

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
```

On ouvre en lecture (**r**ead)
et en texte (pas en binaire)

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
contenu = fichier.readlines()
```

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire pict sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')

contenu: list[str] = fichier.readlines()
```

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire pict sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')

contenu: list[str] = fichier.readlines()

ligne1: str = contenu[0]
# and more
```

Lire dans un fichier en Python

Naviguer dans l'arborescence des fichiers :

1) Ouvrir en lecture le fichier *miniDamier.ppm* se trouvant dans le répertoire pict sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste de contenu pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')

contenu: list[str] = fichier.readlines()

ligne1: str = contenu[0]
# and more

fichier.close()
```

Lire une image PPM dans un fichier en Python

1) Ouvrir en lecture le fichier miniDamier.ppm se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
file: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
content: list[str] = file.readlines()

format: str = content[0]
width: str = content[1]
height: str = content[2]
index: int = int( ?? )
while(index < len(content)) :
        #récupérer les valeurs
        #créer les pixels et les stocker
        #augmenter index

#Créer une variable de type image
file.close()
```

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2
2
255
0
0
0
255
255
255
255
255
255
0
0
0
```

Lire une image PPM dans un fichier en Python

1) Ouvrir en lecture le fichier miniDamier.ppm se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
file: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
content: list[str] = file.readlines()

format: str = content[0]
width: str = content[1]
height: str = content[2]
index: int = int(4)
while(index < len(content)) :
        #récupérer les valeurs
        #créer les pixels et les stocker
        #augmenter index

#Créer une variable de type image
file.close()
```

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2
2
255
0
0
0
255
255
255
255
255
255
0
0
0
```

Lire une image PPM dans un fichier en Python

1) Ouvrir en lecture le fichier miniDamier.ppm se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Récupérer toutes les lignes du fichier dans une liste de chaines de caractères
3) Manipuler la liste pour récupérer les informations et créer les variables utiles en mémoire
4) Fermer le *wrapper* de fichier

```
file: TextIOWrapper = open('./pict/miniDamier.ppm', 'r')
content: list[str] = file.readlines()

format: str = content[0]
width: str = content[1]
height: str = content[2]
index: int = int(4)
while(index < len(content)) :
        #récupérer les valeurs
        #créer les pixels et les stocker
        #augmenter index

#Créer une variable de type image
file.close()
```

```
@dataclass
class ImagePPM:
    format: str
    width:int
    height:int
    pixels:list[Pixel]
```

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2
2
255
0
0
0
255
255
255
255
255
255
0
0
0
```

Écrire dans un fichier en Python

1) Ouvrir **en écriture** le fichier m*iniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*

2) Écrire une chaine de caractère dans le fichier et recommencer si nécessaire

3) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'w')
```

On ouvre en écriture (**w**rite)
et en texte (pas en binaire)

Écrire dans un fichier en Python

1) Ouvrir **en écriture** le fichier m*iniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*

2) Écrire une chaine de caractère dans le fichier et recommencer si nécessaire

3) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'w')

fichier.write('chaine a mettre dans le fichier\n')
# continuer à écrire si nécessaire
```

Écrire dans un fichier en Python

1) Ouvrir **en écriture** le fichier m*iniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*

2) Écrire une chaine de caractère dans le fichier et recommencer si nécessaire

3) Fermer le *wrapper* de fichier

```
fichier: TextIOWrapper = open('./pict/miniDamier.ppm', 'w')

fichier.write('chaine a mettre dans le fichier\n')
# continuer à écrire si nécessaire

fichier.close()
```

Écrire dans un fichier en Python

1) Ouvrir en écriture le fichier m*iniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Écrire une chaine de caractère dans le fichier et recommencer si nécessaire
3) Fermer le *wrapper* de fichier

Soit une variable de type `ImagePPM` nommée `img`

```
file: TextIOWrapper = open('./pict/miniDamier.ppm', 'w')
file.write('P3\n')
file.write(str(img.width)+'\n')
file.write(str(img.height)+'\n')
file.write('255\n')

index: int = int(0)
while(index < len(img.pixels)):
        #récupérer les valeurs de pixel
        #écrire les valeurs dans le fichier
        #augmenter index

file.close()
```

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2
2
255
0
0
0
255
255
255
255
255
255
0
0
0
```

Écrire dans un fichier en Python

1) Ouvrir en écriture le fichier m*iniDamier.ppm* se trouvant dans le répertoire *pict* sur le disque et récupérer un *wrapper*
2) Écrire une chaine de caractère dans le fichier et recommencer si nécessaire
3) Fermer le *wrapper* de fichier

Soit une variable de type `ImagePPM` nommée `img`

```
file: TextIOWrapper = open('./pict/miniDamier.ppm', 'w')
file.write('P3\n')
file.write(str(img.width)+'\n')
file.write(str(img.height)+'\n')
file.write('255\n')

index: int = int(0)
while(index < len(img.pixels)):
        #récupérer les valeurs de pixel
        #écrire les valeurs dans le fichier
        #augmenter index

fichier.close()
```

R, G, B d'un pixel noir

R, G, B d'un pixel blanc

R, G, B d'un pixel blanc

R, G, B d'un pixel noir

```
P3
2
2
255
0
0
0
255
255
255
255
255
255
0
0
0
```

⇒ vous aurez l'occasion de tester tout cela au TD numéro 3

Algèbre de Boole

Algèbre de Boole ?

L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les expressions à deux valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

WIKIPÉDIA
L'encyclopédie libre

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

Algèbre de Boole ?

L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Elle a de nombreuses applications en informatique et dans la conception des circuits électroniques.

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

> La logique, […] est, dans une première approche, l'étude des règles formelles que doit respecter toute argumentation correcte.
>
> L'encyclopédie libre

Algèbre de Boole ?

L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les expressions à deux valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

WIKIPÉDIA
L'encyclopédie libre

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

Algèbre de Boole ?

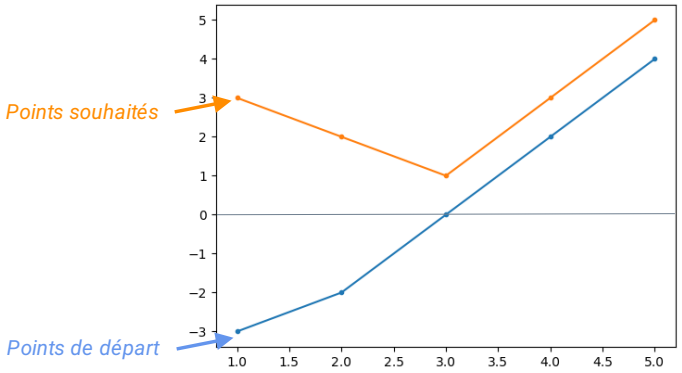
L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les expressions à deux valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

WIKIPÉDIA
L'encyclopédie libre

- C'est l'algèbre sous-jacente aux prédicats utilisés dans les boucles et les branchements conditionnels

```
1  indice: int = 0
2  while (indice < len(allPoints)):
3      p: Point2D = allPoints[indice]
4      if (p.y < 0):
5          p.y = -p.y
6      else:
7          p.y = p.y + 1
8      indice = indice + 1
9  print('traitement fini')
```



Algèbre de Boole ?

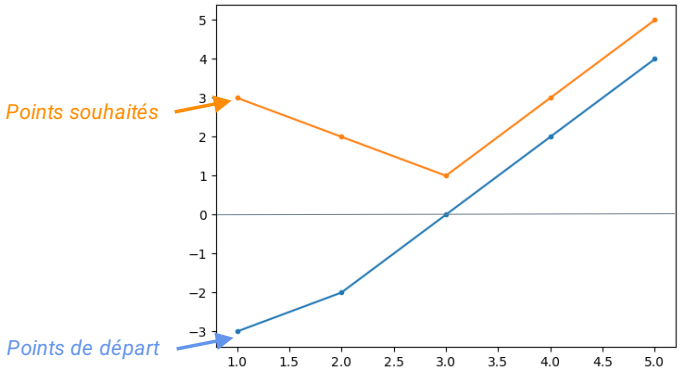
L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les expressions à deux valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

WIKIPÉDIA
L'encyclopédie libre

- C'est l'algèbre sous-jacente aux prédicats utilisés dans les boucles et les branchements conditionnels

```
indice: int = 0
while (indice < len(allPoints)):
    p: Point2D = allPoints[indice]
    if (p.y < 0 and p.x > 0):
        p.y = -p.y
    else:
        p.y = p.y + 1
    indice = indice + 1
print('traitement fini')
```



Algèbre de Boole ?

L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les expressions à deux valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

WIKIPÉDIA
L'encyclopédie libre

- C'est l'algèbre sous-jacente aux prédicats utilisés dans les boucles et les branchements conditionnels
- Pour faire des raisonnements rigoureux sur des objets mathématiques
- Pour mécaniser voire automatiser les raisonnements.
- Pour faire des raisonnements rigoureux en général !

Algèbre de Boole ?

L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques, ce qui permet d'utiliser des techniques algébriques pour traiter les expressions à deux valeurs du calcul des propositions. Elle fut lancée en 1854 par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

Elle fut utilisée la première fois pour les circuits de commutation téléphonique par Claude Shannon.

WIKIPÉDIA
L'encyclopédie libre

L'histoire suivante est extraite de l'ouvrage « Le livre qui rend fou » de Raymond Smullyan.

Un ~~prisonnier~~ étudiant doit choisir entre deux cellules, chaque cellule pouvant contenir une princesse ou un tigre. S'il tombe sur le tigre il est dévoré, s'il délivre la princesse il est gracié. Sur chaque cellule, Il y a une inscription qui peut être vraie ou fausse. Le roi dit une phrase qui est vraie.

Calcul propositionnel

- théorie logique qui définit les lois formelles du raisonnement
- Consiste en l'évaluation d'expressions résultant en une valeur parmi {faux, vrai} (notées aussi {F, V} ou {0, 1})
- Une **proposition atomique** est une affirmation susceptible d'être **vraie** ou **fausse**. Exemples :
  - il pleut
  - 2022 est une année bissextile
  - i > 10
- On peut former une proposition en utilisant des ***opérateurs logiques***, *par exemple*
  - Il pleut ***et*** il y a du soleil
  - i > 10 **ou** i < 0
  - x est multiple de 4 **implique que** x est pair

Opérateur logique unaire

- Les opérateurs sont définis par des **tables de vérité**

a	$\neg a$
F	V
V	F

- `¬a` est la négation de `a`
- `F` est la **contradiction** (aussi notée $\bot$) : toujours faux
- `V` est la **tautologie** (aussi notée $\top$) : toujours vrai.

Opérateurs logiques binaires

- Les opérateurs sont définis par des **tables de vérité**

a	b	$a \wedge b$	$a \vee b$
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

⚠️le ou (□) est inclusif

Ce n'est pas forcément le même sens qu'en français « courant ».

En logique, *fromage ou dessert* est vraie si

- je prends du fromage et pas de dessert
- je prends du dessert et pas de fromage
- je prends du dessert et du fromage

Opérateurs logiques binaires

Quelques propriétés

- a = ¬¬a
- a ∨ ¬a = V
- a ∧ ¬a = F
- a ∨ V = V
- a ∨ F = a
- a ∧ V = a
- a ∧ F = F

Deux formules logiques f et g sont équivalentes si et seulement si elles ont la même table de vérité. On note alors f = g.

Opérateurs logiques binaires

Quelques propriétés

- a = ¬¬a

a	$\neg a$	$\neg\neg a$
F	V	F
V	F	V

- a ∨ ¬a = V
- a ∧ ¬a = F
- a ∨ V = V
- a ∨ F = a
- a ∧ V = a
- a ∧ F = F

Deux formules logiques `f` et `g` sont équivalentes si et seulement si elles ont la même table de vérité. On note alors `f = g`.

Opérateurs logiques binaires

Quelques propriétés

- a = ¬¬a

a	$\neg a$	$\neg\neg a$
F	V	F
V	F	V

- a ∨ ¬a = V
- a ∧ ¬a = F

a	$\neg a$	$a \wedge \neg a$
F	V	F
V	F	F

- a ∨ V = V
- a ∨ F = a
- a ∧ V = a
- a ∧ F = F

Deux formules logiques `f` et `g` sont équivalentes si et seulement si elles ont la même table de vérité. On note alors `f = g`.

Opérateurs logiques binaires

Quelques propriétés

- a = ¬¬a

a	$\neg a$	$\neg\neg a$
F	V	F
V	F	V

- a ∨ ¬a = V
- a ∧ ¬a = F

a	$\neg a$	$a \wedge \neg a$
F	V	F
V	F	F

- a ∨ V = V
- a ∨ F = a
- a ∧ V = a
- a ∧ F = F

Deux formules logiques `f` et `g` sont équivalentes si et seulement si elles ont la même table de vérité. On note alors `f = g`.

¬(a ∧ ¬b) = ¬a ∨ b ???

Opérateurs logiques binaires

Quelques propriétés

- a = ¬¬a
- a ∨ ¬a = V
- a ∧ ¬a = F
- a ∨ V = V
- a ∨ F = a
- a ∧ V = a
- a ∧ F = F

a	$\neg a$	$\neg\neg a$
F	V	F
V	F	V

a	$\neg a$	$a \wedge \neg a$
F	V	F
V	F	F

Deux formules logiques f et g sont équivalentes si et seulement si elles ont la même table de vérité. On note alors f = g.

¬(a ∧ ¬b) = ¬a ∨ b

a	b	$\neg b$	$a \wedge \neg b$	$\neg(a \wedge \neg b)$	$\neg a$	$\neg a \vee b$
F	F	V	F	V	V	V
F	V	F	F	V	V	V
V	F	V	V	F	F	F
V	V	F	F	V	F	V

Construire une table de vérité

- Écrire sur la première ligne les propositions atomiques et les formules, une colonne pour chaque.
- Déterminer le nombre de lignes : 2^n où n nombre de propositions atomiques.
- Exemple : si $n=3$, alors $2^n=8$ lignes.
- Compléter les lignes avec toutes les combinaisons possibles.
  - Technique : Compter jusque 2^n en binaire pour n'oublier aucune ligne. ($0=F$ et $1=V$)
- Remplir les colonnes de droite, éventuellement en découpant les formules compliquées.

Construire une table de vérité et sauver un étudiant

- Écrire sur la première ligne les propositions atomiques et les formules, une colonne pour chaque.
- Déterminer le nombre de lignes : 2^n où n nombre de propositions atomiques.
- Exemple : si $n=3$, alors $2^n=8$ lignes.
- Compléter les lignes avec toutes les combinaisons possibles.
- Technique : Compter jusque 2^n en binaire pour n'oublier aucune ligne. ($0=F$ et $1=V$)
- Remplir les colonnes de droite, éventuellement en découpant les formules compliquées.

Un ~~prisonnier~~ étudiant doit choisir entre deux cellules, chaque cellule pouvant contenir une princesse ou un tigre. S'il tombe sur le tigre il est dévoré, s'il délivre la princesse il est gracié. Sur chaque cellule, Il y a une inscription qui peut être vraie ou fausse. Le roi dit une phrase qui est vraie.

Construire une table de vérité et sauver un étudiant

Un ~~prisonnier~~ étudiant doit choisir entre deux cellules, chaque cellule pouvant contenir une princesse ou un tigre. S'il tombe sur le tigre il est dévoré, s'il délivre la princesse il est gracié. Sur chaque cellule, Il y a une inscription qui peut être vraie ou fausse. Le roi dit une phrase qui est vraie.

Cellule1Princesse	Cellule2Princesse	Affiche1	Affiche2	PhraseRoi
F (tigre)	F (tigre)	F	F	F
F (tigre)	V (princesse)	F	V	V
V (princesse)	F (tigre)	V	V	F
V (princesse)	V (princesse)	F	F	F

Logique dans le code python

- Mêmes opérateurs, notés un peu différemment

opérateur	notation classique	python
et	$\wedge$	`and`
ou	$\vee$	`or`
non	$\neg$	`not`

- Comparaisons

comparaison	notation classique	python
strictement supérieur	$>$	`>`
supérieur ou égal	$\geqslant$	`>=`
strictement inférieur	$<$	`<`
inférieur ou égal	$\leqslant$	`<=`
égal	$=$	`==`
différent	$\neq$	`!=`

⚠️ en Python le simple $=$ est une affectation et se lit de droite à gauche !