

Introduction à l'informatique

2 - Algorithmes

Julien Deantoni

Intervenants

- Sarah Antier (Sarah.ANTIER@univ-cotedazur.fr),
- Julien Deantoni (Julien.DEANTONI@univ-cotedazur.fr),
- Franck Guingne (Franck.GUINGNE@univ-cotedazur.fr),
- Stéphane Jeannin (Stephane.JEANNIN@univ-cotedazur.fr)

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
8:00-10:00		TD G1 Elec/PPPE M13 (→ 10/10) puis M37 Jeannin	TP G5 Oui-Si PV 316-317 Antier - Deantoni		
10:15-12:15	TD G2 MIASH2 M13 Jeannin		TP G2 MIASH2 PV 316-317 Antier - Deantoni	TD G4 LAS-IN M12 Jeannin	TD G3 MIASH 3 M35 Deantoni
13:15-15:15		TD G5 Oui-Si M12 Deantoni	TP G1 Elec/PPPE PV 316-317 Jeannin - Guingne		TP G4 LAS-IN PV 302 Deantoni
15:30-17:30		TP G3 MIASH3 PV 316-317 Deantoni - Jeannin	CM Amphi Physique 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

Opérations sur les variables

- On peut réaliser les opérations sur les variables telles que définis par le type. Par exemple en Python

NomVar : type = initialization value

```
i : int = int(42)      # déclaration de la variable i, initialisée à 42
j : int = 2           # déclaration de la variable j, initialisée à 2
k : int = i+j         # déclaration de la variable i, initialisée à 44
                     # appel de l'opération addition avec pour paramètres i et j
l : int = k//2        # déclaration de la variable l, initialisée à 22
                     # appel de l'opération division entière
                     # avec pour paramètres k et 2

s1 : str = str("Hello")
s2 : str = str(" ")
s3 : str = "world"
greeting : str = s1+s2+s3 # déclaration de la variable greeting, initialisée à Hello World
                        # 2 appels de l'opération concaténation avec pour
                        # paramètres s1 et s2 puis le résultat de s1+s2 et s3
```

Opérations sur les variables

- On peut réaliser les opérations sur les variables telles que définis par le type. Par exemple en Python

NomVar : type = initialization value

```
1 i : int = int(42)           # déclaration de la variable i, initialisée à 42
2 j : int = 2                 # déclaration de la variable j, initialisée à 2
3 k : int = i+j               # déclaration de la variable i, initialisée à 44
4                             # appel de l'opération addition avec pour paramètres i et j
5 l : int = k//2              # déclaration de la variable l, initialisée à 22
6                             # appel de l'opération division entière avec k et 2
7 s1 : str = str("Hello")
8 s2 : str = str(" ")
9 s3 : str = "world"
10 greeting : str = s1+s2+s3  # déclaration de la variable greeting, initialisée à Hello World
                              # 2 appels de l'opération concaténation avec pour
                              # paramètres s1 et s2 puis le résultat de s1+s2 et s3
```

Pas d'exécution et valeurs dans la mémoire

	Step #	Line #
1	i: int = int(42)	1
2	j: int = 2	2
3	k: int = i+j	3
4		4
5	s1: str = str("Hello")	5
6	s2: str = str("!")	6
7	s3: str = s1+s2	7

Pas d'exécution et valeurs dans la mémoire

		Step #	Line #	i
1	i: int = int(42)	1	1	42
2	j: int = 2	2	2	
3	k: int = i+j	3	3	
4			4	
5	s1: str = str("Hello")	4	5	
6	s2: str = str("!")	5	6	
7	s3: str = s1+s2	6	7	

Pas d'exécution et valeurs dans la mémoire

	Step #	Line #	i	j
1	1	1	42	<u>undefined</u>
2	2	2	42	2
3	3	3		
4		4		
5	4	5		
6	5	6		
7	6	7		

Pas d'exécution et valeurs dans la mémoire

		Step #	Line #	i	j	k
1	i: int = int(42)	1	1	42	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	2	42	2	<u>undefined</u>
3	k: int = i+j	3	3	42	2	44
4			4			
5	s1: str = str("Hello")	4	5			
6	s2: str = str("!")	5	6			
7	s3: str = s1+s2	6	7			

Pas d'exécution et valeurs dans la mémoire

	Step #	Line #	i	j	k
1	i: int = int(42)	1	42	undefined	undefined
2	j: int = 2	2	42	2	undefined
3	k: int = i+j	3	42	2	44
4		4	42	2	44
5	s1: str = str("Hello")	4			
6	s2: str = str("!")	5			
7	s3: str = s1+s2	6			

Pas d'exécution et valeurs dans la mémoire

	Step #	Line #	i	j	k	s1
1	i: int = int(42)	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	42	2	<u>undefined</u>	<u>undefined</u>
3	k: int = i+j	3	42	2	44	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>
5	s1: str = str("Hello")	5	42	2	44	'Hello'
6	s2: str = str("!")	6				
7	s3: str = s1+s2	7				

Pas d'exécution et valeurs dans la mémoire

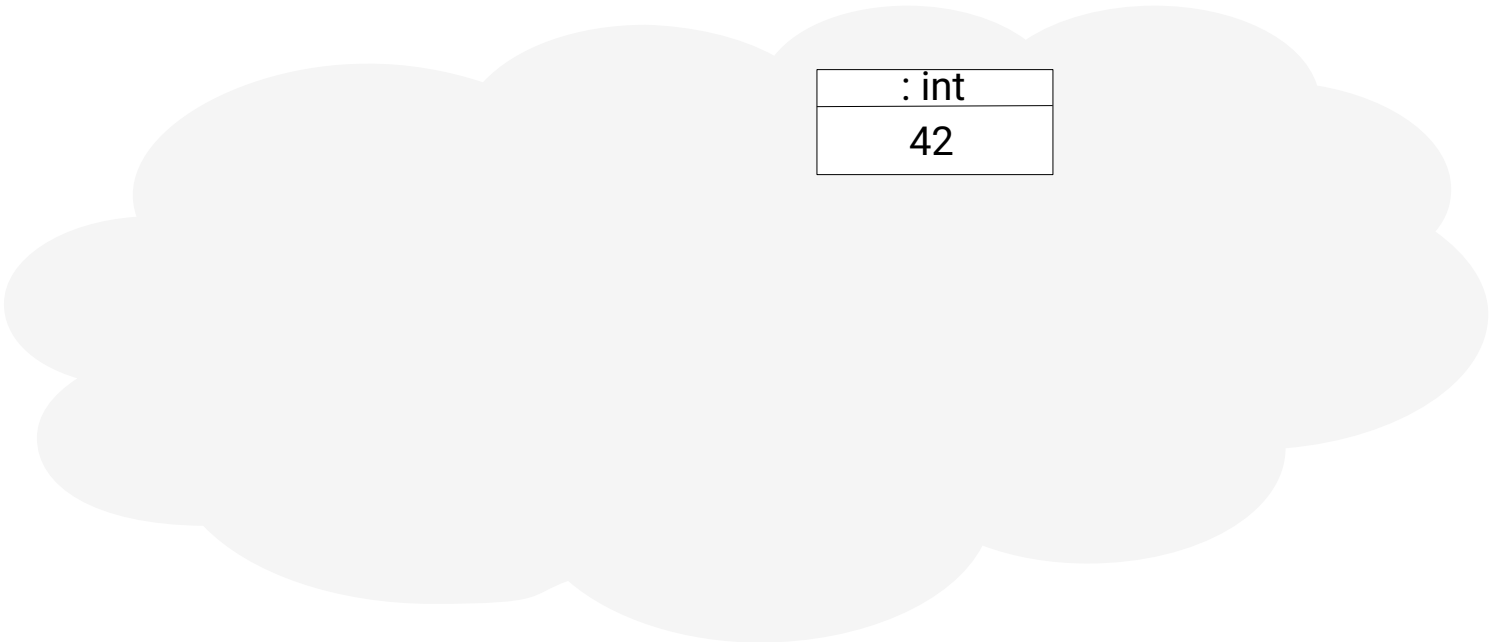
	Step #	Line #	i	j	k	s1	s2
1	1	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	3	3	42	2	44	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>
5	4	5	42	2	44	'Hello'	<u>undefined</u>
6	5	6	42	2	44	'Hello'	'!'
7	6	7					

Pas d'exécution et valeurs dans la mémoire

	Step #	Line #	i	j	k	s1	s2	s3
1	1	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	3	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	4	5	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	5	6	42	2	44	'Hello'	'!'	<u>undefined</u>
7	6	7	42	2	44	'Hello'	'!'	'Hello !'

Pas d'exécution et valeurs dans la mémoire

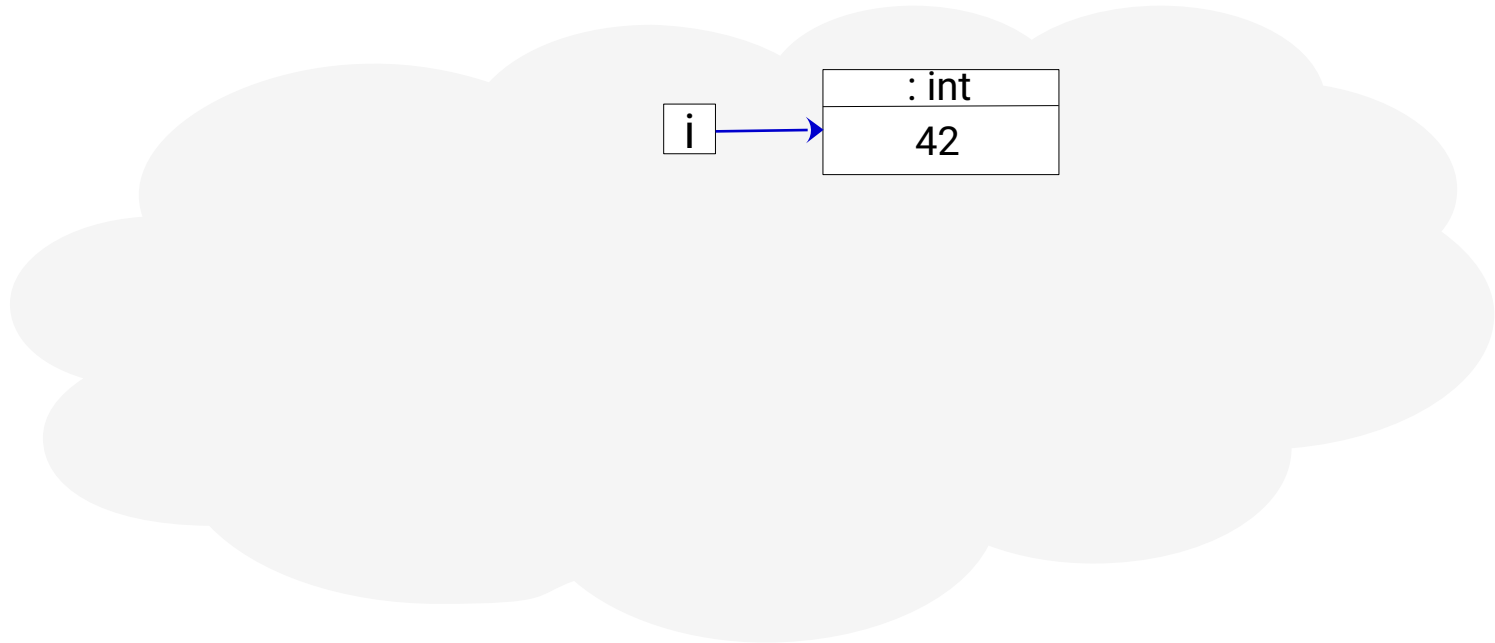
	Step #	Line #	i	j	k	s1	s2	s3
1	i: int = int(42)	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	k: int = i+j	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	s1: str = str("Hello")	4	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	s2: str = str("!")	5	42	2	44	'Hello'	'!'	<u>undefined</u>
7	s3: str = s1+s2	6	42	2	44	'Hello'	'!'	'Hello !'



Représentation du contenu de la mémoire non exact mais suffisante pour comprendre les notions qui vont suivre

Pas d'exécution et valeurs dans la mémoire

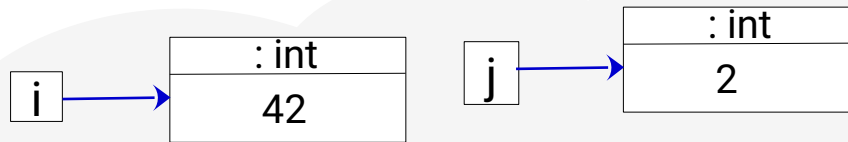
	Step #	Line #	i	j	k	s1	s2	s3
1	<code>i: int = int(42)</code>	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	<code>j: int = 2</code>	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	<code>k: int = i+j</code>	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	<code>s1: str = str("Hello")</code>	5	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	<code>s2: str = str("!")</code>	6	42	2	44	'Hello'	'!'	<u>undefined</u>
7	<code>s3: str = s1+s2</code>	7	42	2	44	'Hello'	'!'	'Hello !'



Représentation du contenu de la mémoire non exact mais suffisante pour comprendre les notions qui vont suivre

Pas d'exécution et valeurs dans la mémoire

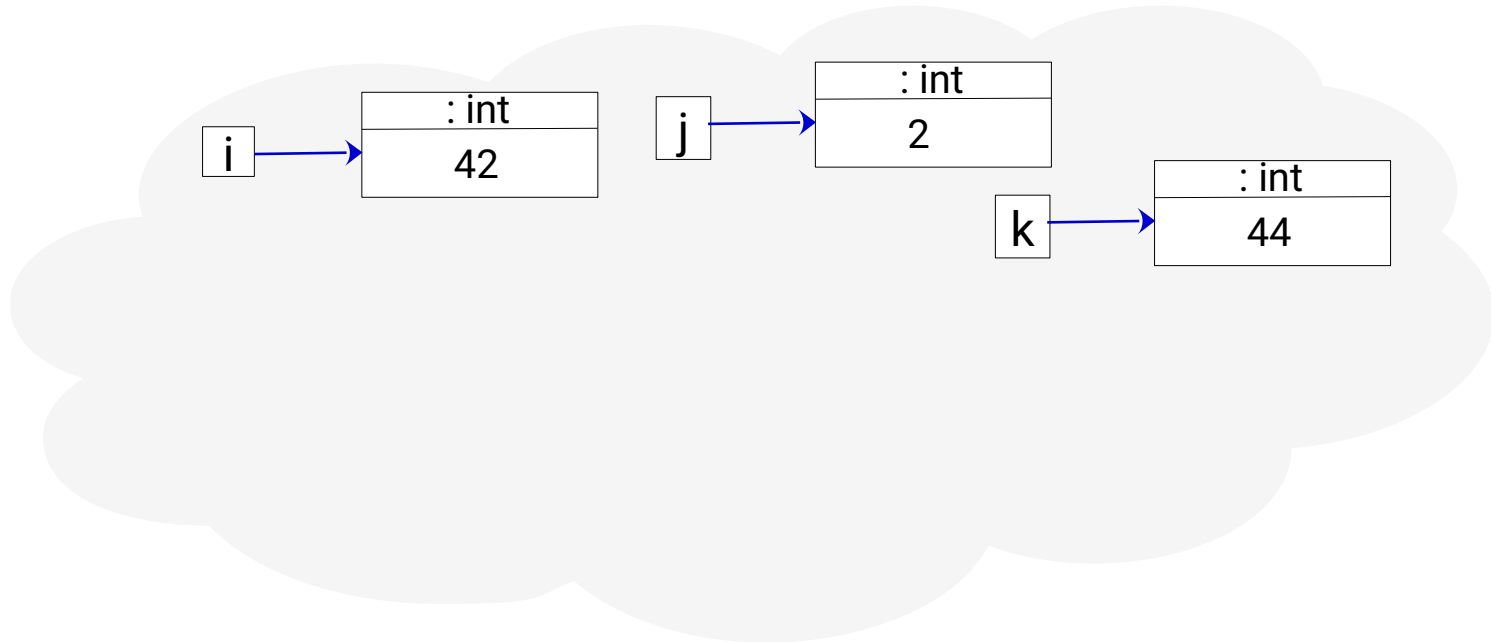
	Step #	Line #	i	j	k	s1	s2	s3
1	i: int = int(42)	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	k: int = i+j	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	s1: str = str("Hello")	4	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	s2: str = str("!")	5	42	2	44	'Hello'	'!'	<u>undefined</u>
7	s3: str = s1+s2	6	42	2	44	'Hello'	'!'	'Hello !'



Représentation du contenu de la mémoire non exact mais suffisante pour comprendre les notions qui vont suivre

Pas d'exécution et valeurs dans la mémoire

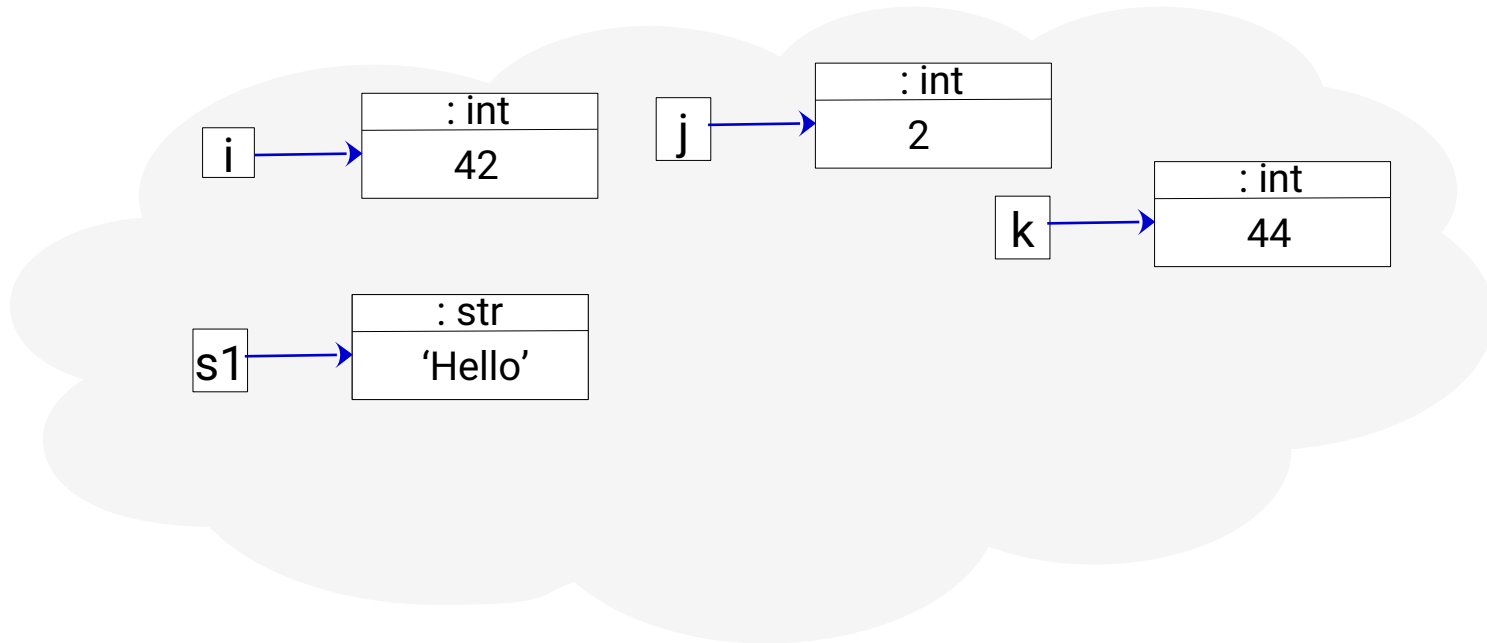
	Step #	Line #	i	j	k	s1	s2	s3
1	i: int = int(42)	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	k: int = i+j	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	s1: str = str("Hello")	4	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	s2: str = str("!")	5	42	2	44	'Hello'	'!'	<u>undefined</u>
7	s3: str = s1+s2	6	42	2	44	'Hello'	'!'	'Hello !'



Représentation du contenu de la mémoire non exact mais suffisante pour comprendre les notions qui vont suivre

Pas d'exécution et valeurs dans la mémoire

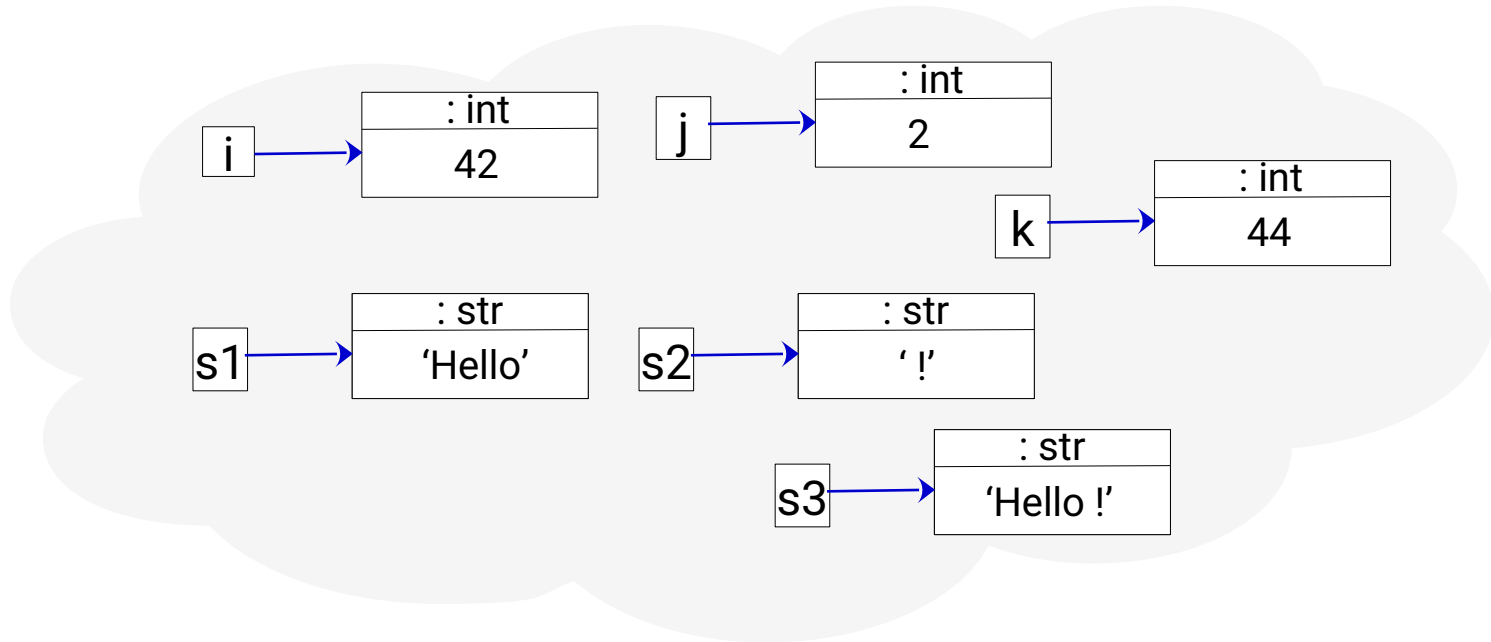
	Step #	Line #	i	j	k	s1	s2	s3
1	i: int = int(42)	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	k: int = i+j	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	s1: str = str("Hello")	4	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	s2: str = str("!")	5	42	2	44	'Hello'	'!'	<u>undefined</u>
7	s3: str = s1+s2	6	42	2	44	'Hello'	'!'	'Hello !'



Représentation du contenu de la mémoire non exact mais suffisante pour comprendre les notions qui vont suivre

Pas d'exécution et valeurs dans la mémoire

	Step #	Line #	i	j	k	s1	s2	s3
1	i: int = int(42)	1	42	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
2	j: int = 2	2	42	2	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
3	k: int = i+j	3	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
4		4	42	2	44	<u>undefined</u>	<u>undefined</u>	<u>undefined</u>
5	s1: str = str("Hello")	4	42	2	44	'Hello'	<u>undefined</u>	<u>undefined</u>
6	s2: str = str("!")	5	42	2	44	'Hello'	'!'	<u>undefined</u>
7	s3: str = s1+s2	6	42	2	44	'Hello'	'!'	'Hello !'



Représentation du contenu de la mémoire non exact mais suffisante pour comprendre les notions qui vont suivre

Structuration de l'information

Opération sur les variables (2/3)

- On peut réaliser les opérations sur les variables telles que définis par le type. Par exemple en Python

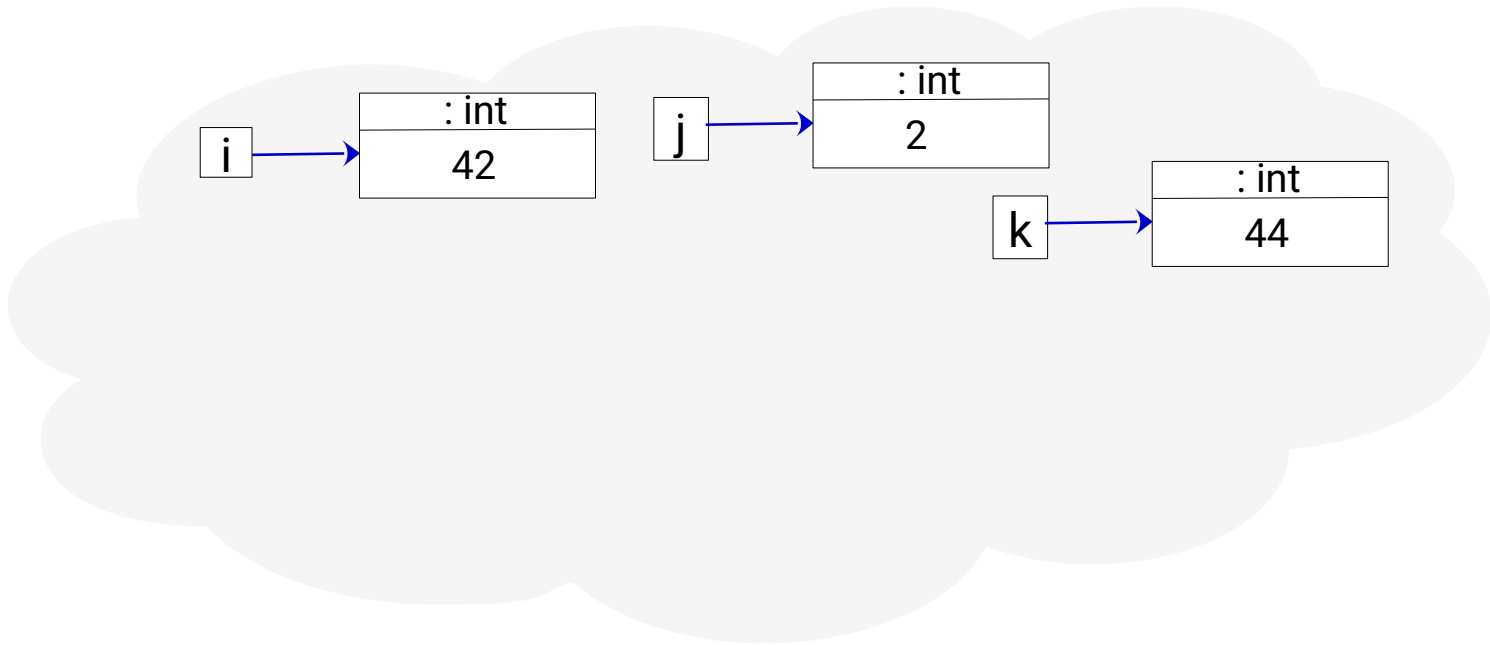
```
i : int = int(42)    #déclaration de la variable i, initialisée à 42
j : int = 2         #déclaration de la variable j, initialisée à 2
k : int = i+j       #déclaration de la variable i, initialisée à 44
                    # appel de l'opération addition avec pour paramètres i et j
del j               #suppression de la variable j de la mémoire

k = j+1             → erreur
```

*Les variables sont stockées dans la RAM. Même si on le fait peu, on peut à tout moment enlever une variable de la mémoire en utilisant **del***

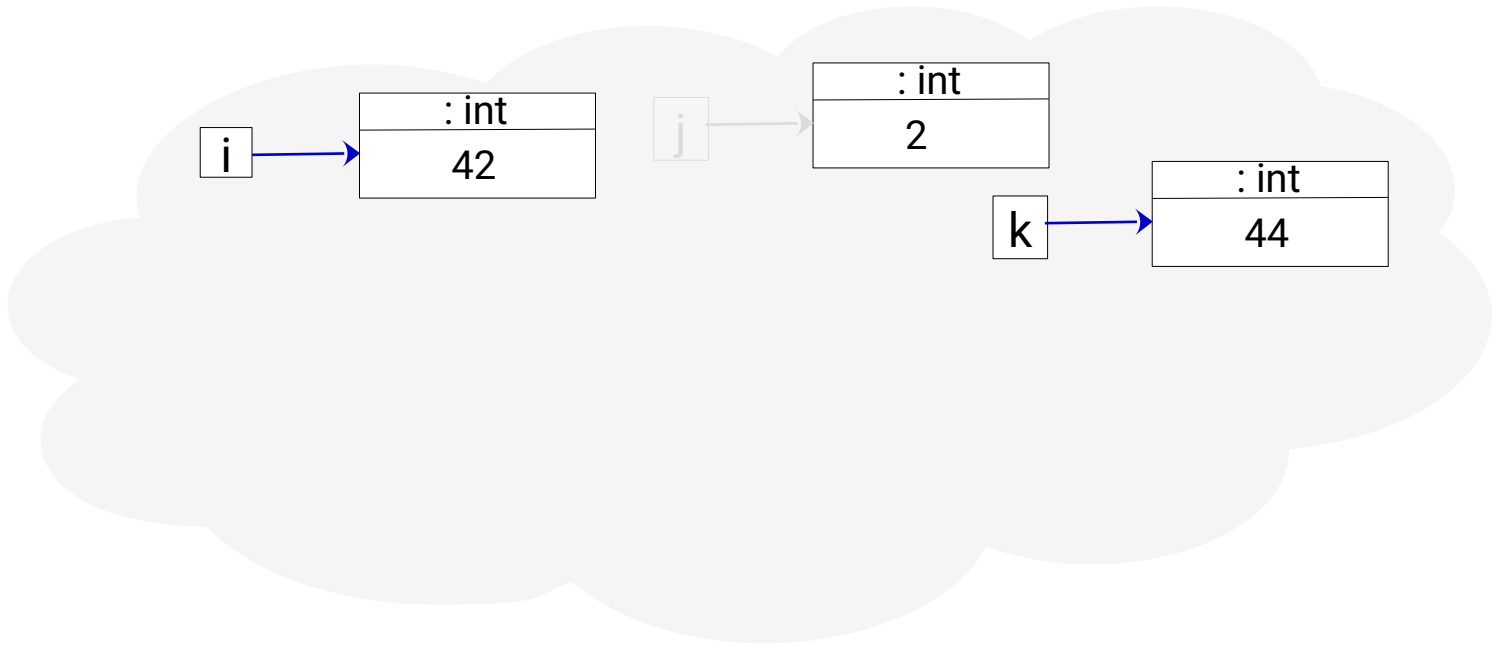
Pas d'exécution et valeurs dans la mémoire

	<u>Step #</u>	<u>Line #</u>	<u>i</u>	<u>j</u>	<u>k</u>
1 <code>i : int = int(42)</code>	1	1	42	<u>undefined</u>	<u>undefined</u>
2 <code>j : int = 2</code>	2	2	42	2	<u>undefined</u>
3 <code>k : int = i+j</code>	3	3	42	2	44
4		4	42	2	44
5 <code>del j</code>	4	5	42	<u>undefined</u>	44
6		6	42	<u>undefined</u>	44
7 <code>k = j+1</code>	5	7	42	<u>undefined</u>	44



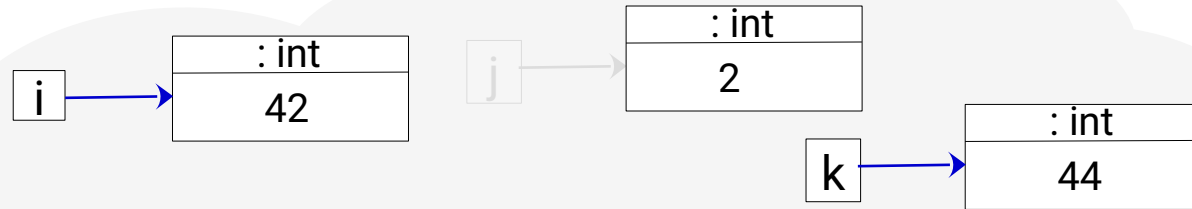
Pas d'exécution et valeurs dans la mémoire

	<u>Step #</u>	<u>Line #</u>	<u>i</u>	<u>j</u>	<u>k</u>
1 <code>i : int = int(42)</code>	1	1	42	<u>undefined</u>	<u>undefined</u>
2 <code>j : int = 2</code>	2	2	42	2	<u>undefined</u>
3 <code>k : int = i+j</code>	3	3	42	2	44
4	4	4	42	2	44
5 del j	4	5	42	<u>undefined</u>	44
6	6	6	42	<u>undefined</u>	44
7 <code>k = j+1</code>	5	7	42	<u>undefined</u>	44



Pas d'exécution et valeurs dans la mémoire

	<u>Step #</u>	<u>Line #</u>	<u>i</u>	<u>j</u>	<u>k</u>
1 <code>i : int = int(42)</code>	1	1	42	<u>undefined</u>	<u>undefined</u>
2 <code>j : int = 2</code>	2	2	42	2	<u>undefined</u>
3 <code>k : int = i+j</code>	3	3	42	2	44
4	4	4	42	2	44
5 <code>del j</code>	4	5	42	<u>undefined</u>	44
6	6	6	42	<u>undefined</u>	44
7 <code>k = j+1</code>	5	7	42	<u>undefined</u>	44



```

-----
NameError
Cell In [3], line 7
    3 k: int = i+j
    5 del j
----> 7 k = j + 1

```

Traceback (most recent call last)

NameError: name 'j' is not defined

Pas d'exécution et valeurs dans la mémoire

		Step #	Line #	anInt	resultatDiv	reste
1	anInt: int = 36	1	1	36	undefined	undefined
2	reste: int = anInt % 2	2	2	36	undefined	0
3			3	36	undefined	0
4	resultatDiv: int = anInt // 2	3	4	36	18	0
5	reste = resultatDiv % 2	4	5	36	18	0
6			6	36	18	0
7	resultatDiv = resultatDiv // 2	5	7	36	9	0
8	reste = resultatDiv % 2	6	8	36	9	1
9			9	36	9	1
10	resultatDiv = resultatDiv // 2	7	10	36	4	1
11	reste = resultatDiv % 2	8	11	36	4	0
12			12	36	4	0
13	resultatDiv = resultatDiv // 2	9	13	36	2	0
14	reste = resultatDiv % 2	10	14	36	2	0
15			15	36	2	0
16	resultatDiv = resultatDiv // 2	11	16	36	1	0
17	reste = resultatDiv % 2	12	17	36	1	1

Structuration de l'information

Pas d'exécution et valeurs en mémoire

- État de la mémoire vive

		Step #	Line #	anInt	resultatDiv	resteDiv
1	anInt: int = 36	1	1	36	undefined	undefined
2	reste: int = anInt % 2	2	2	36	undefined	0
3			3	36	undefined	0
4	resultatDiv: int = anInt // 2	3	4	36	18	0
5	reste = resultatDiv % 2	4	5	36	18	0
6			6	36	18	0
7	resultatDiv = resultatDiv // 2	5	7	36	9	0
8	reste = resultatDiv % 2	6	8	36	9	1
9			9	36	9	1
10	resultatDiv = resultatDiv // 2	7	10	36	4	1
11	reste = resultatDiv % 2	8	11	36	4	0
12			12	36	4	0
13	resultatDiv = resultatDiv // 2	9	13	36	2	0
14	reste = resultatDiv % 2	10	14	36	2	0
15			15	36	2	0
16	resultatDiv = resultatDiv // 2	11	16	36	1	0
17	reste = resultatDiv % 2	12	17	36	1	1
		13		undefined	undefined	undefined

⇒ Comprendre comment évoluent les valeurs de variables en mémoire pendant l'exécution permettra de donner l'ordre des instructions nécessaire pour réaliser le traitement que l'on souhaite

Structuration de l'information

Opération sur les variables (3/3)

- On peut réaliser les opérations sur les variables telles que définis par le type. Par exemple en Python

Pour information, la liste de toutes les opérations possibles est consultable ici :
https://www.w3schools.com/python/python_operators.asp

```
i: int = int(42)    # déclaration de la variable i, initialisée à 42
j: int = int(2)    # déclaration de la variable j, initialisée à 2
j = i - j          # affectation de la valeur 15 à la variable j

b1: bool = bool( i < j )    # comparaison entre deux entier. Renvoie vrai ou faux
b2: bool = bool( i == j )
b3: bool = bool(not b1)

f : float = float(j / i) # approximation de la valeur de j / i stockée dans un float
                    # attention ne jamais comparer 2 float avec une égalité (==)

j = i**2           # i puissance 2 (au carré)
f2 : float = j**(1/2) # j puissance 1/2 (racine de j)
```

Structuration de l'information

Liste

Afin de stocker plusieurs éléments dans une seule variable, on utilise un type *Collection*. Il existe différentes collections mais nous ne verrons dans ce cours que le type prédéfini «*list*». En python, une liste est :

- Ordonnée et indexable. Les éléments sont rentrés et stockés dans un ordre particulier et on peut facilement récupérer l'élément à une position particulière.

```
mesAmis: list[str] = ['tata', 'titi', 'toto'] # une variable de type list
ami0: str = mesAmis[0] # une variable de type string initialisée avec l'élément à l'indice 0, i.e., 'zaza'
titi: str = mesAmis[1] # attention titi est le nom de la variable, et 'titi' est son contenu
```

- Modifiable. Il est possible de modifier, d'ajouter ou d'enlever des éléments.

```
mesAmis[0] = 'zaza' # ['zaza', 'titi', 'toto']
mesAmis = mesAmis + ['zozo'] # ['zaza', 'titi', 'toto', 'zozo']
del mesAmis[1] # ['zaza', 'toto', 'zozo']
mesAmis = mesAmis + 'zozo' # TypeError: can only concatenate list (not "str") to list
```

- Peut contenir plusieurs fois le même élément

Mutable versus Immutable data types

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

		Step #	Line #	i
1	i: int = int(42)	1	1	42
2			2	42
3	i = i + 1	2	3	43

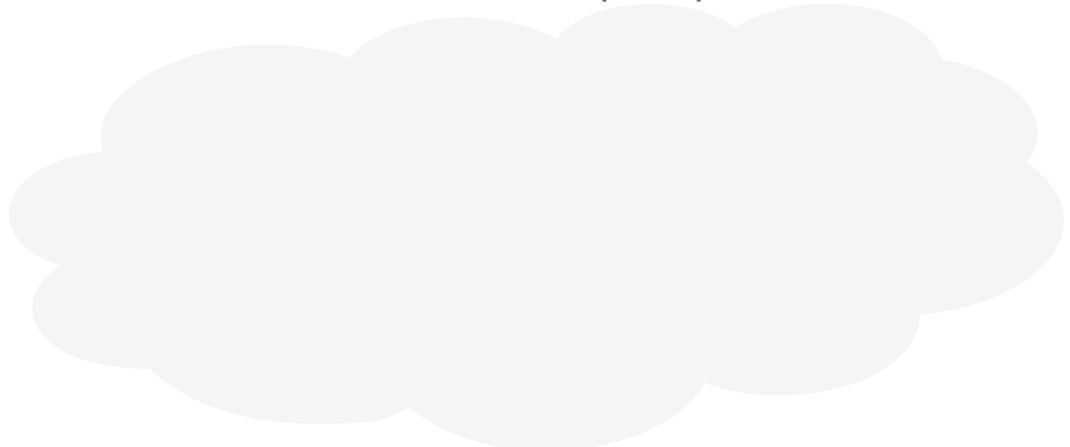


- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

Mutable versus Immutable data types

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

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

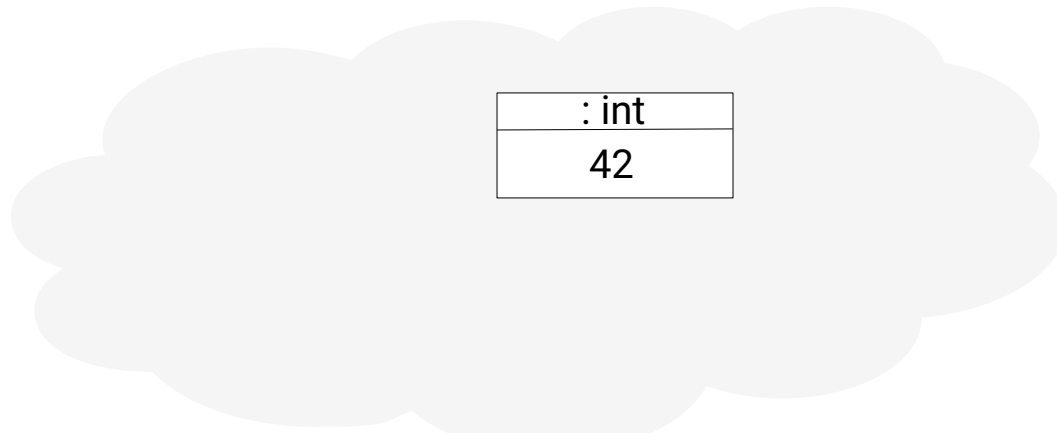


- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

Mutable versus Immutable data types

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

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



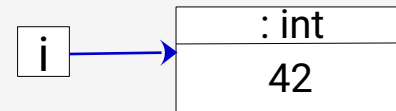
: int
42

- 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 *immuables*.
- Immutable : int, str, float, bool
 - Le contenu des cases mémoires associé aux variables ne peut pas être modifié !

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

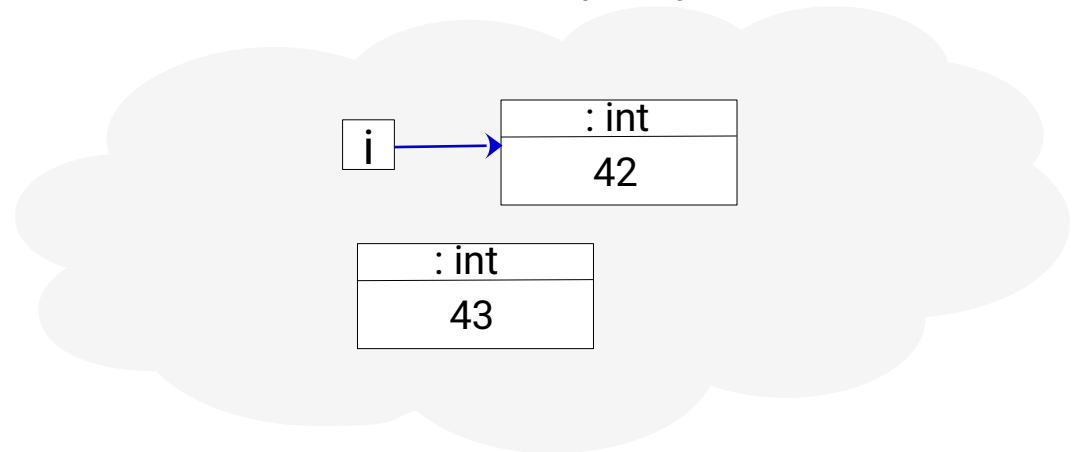


- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

Mutable versus Immutable data types

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

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

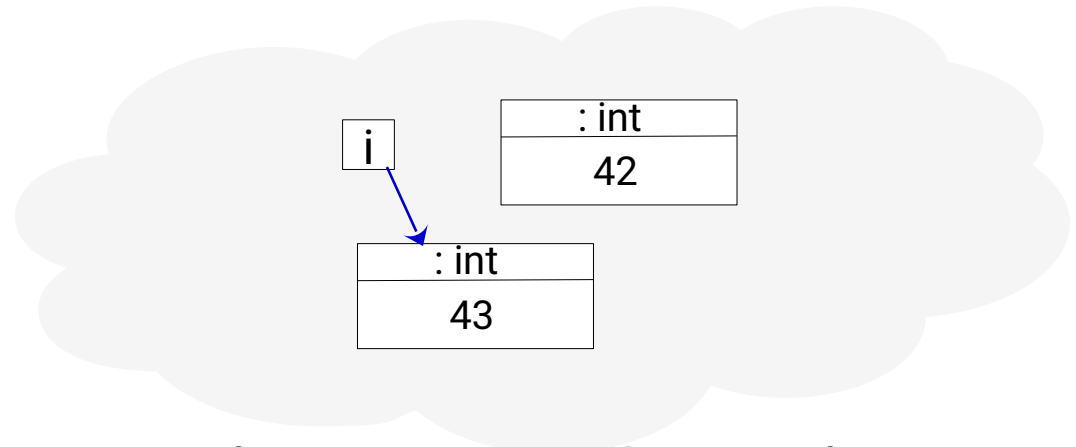


- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

Mutable versus Immutable data types

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

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

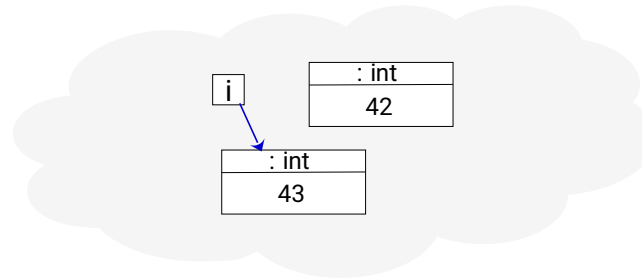


- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

Mutable versus Immutable data types

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

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



- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

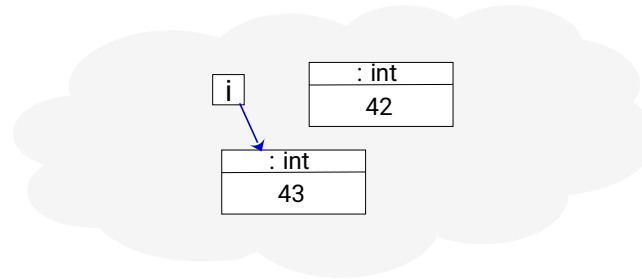
```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```



Mutable versus Immutable data types

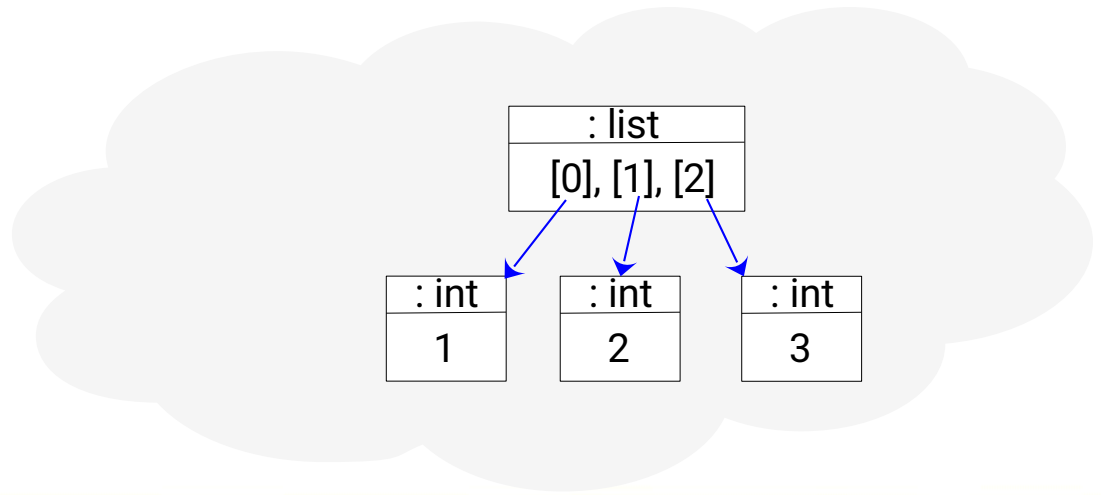
- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les *immuables*.
- Immutable : int, str, float, bool
 - Le contenu des cases mémoires associé aux variables ne peut pas être modifié !

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



- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

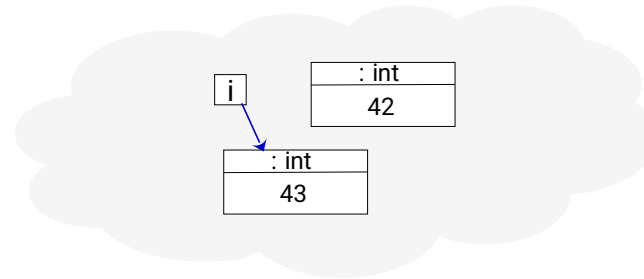
```
1 l: list[int] = list([1,2,3])
2
3 l[0] = l[0] + 1
```



Mutable versus Immutable data types

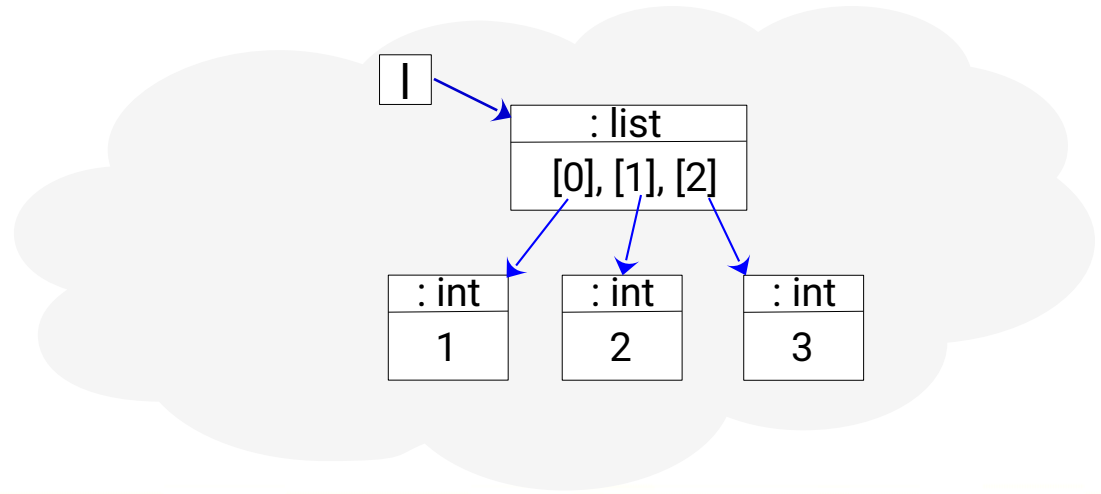
- Python, comme tout langage informatique définit deux sortes différentes de type de données : les *mutables* et les *immuables*.
- Immutable : int, str, float, bool
 - Le contenu des cases mémoires associé aux variables ne peut pas être modifié !

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



- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

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

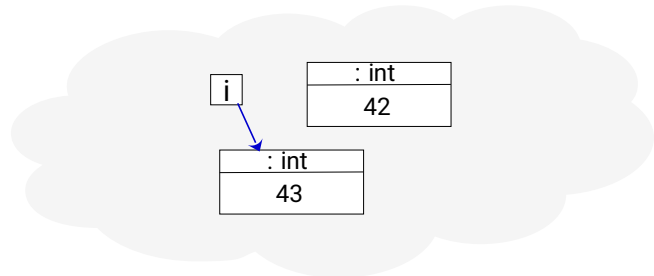


Mutable versus Immutable data types

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

```

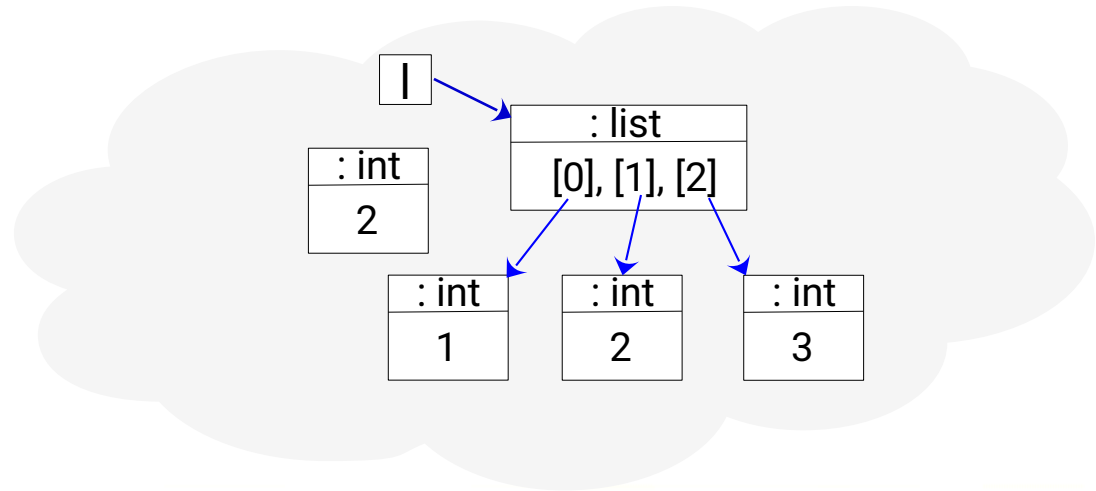
1 i: int = int(42)
2
3 i = i + 1
    
```



- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

```

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

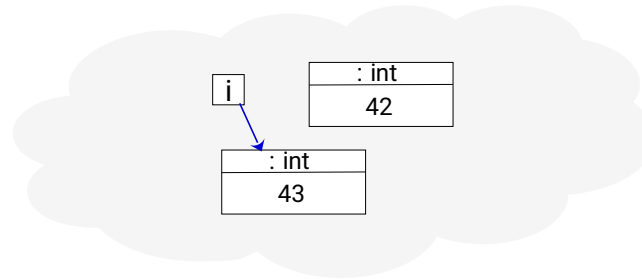


Mutable versus Immutable data types

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

```

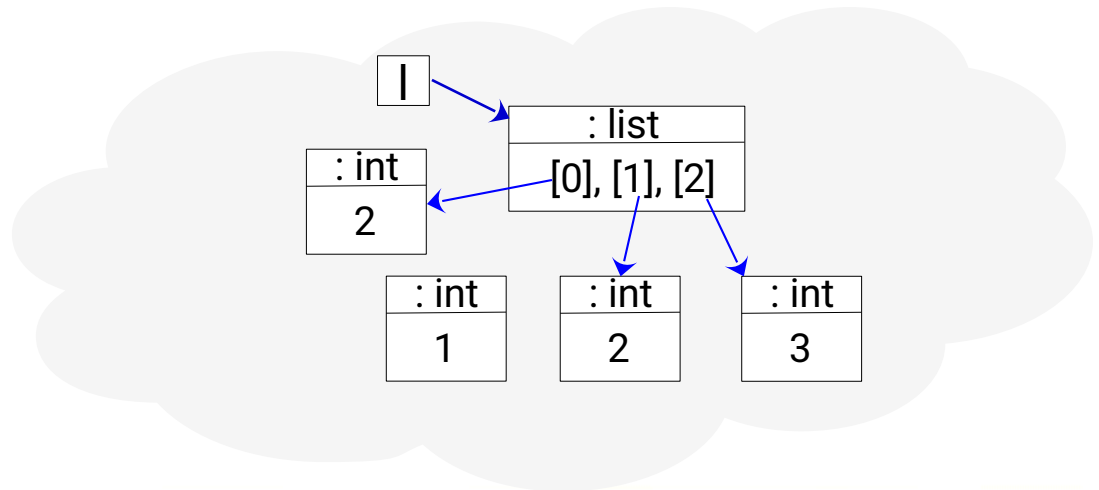
1 i: int = int(42)
2
3 i = i + 1
    
```



- Mutable : list, Point2D
 - Le contenu des cases mémoires associé aux variables peut être modifié

```

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



Mutable versus Immutable data types

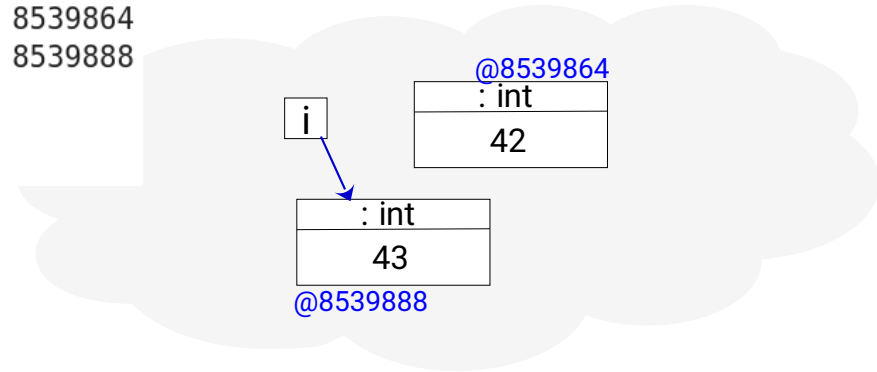
id(var) à la rescousse

- 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 !

```

1 i: int = int(42)
2 print('id de i ligne 2:', id(i))
3 i = i + 1
4 print('id de i ligne 4:', id(i))
    
```

id de i ligne 2: 8539864
 id de i ligne 4: 8539888



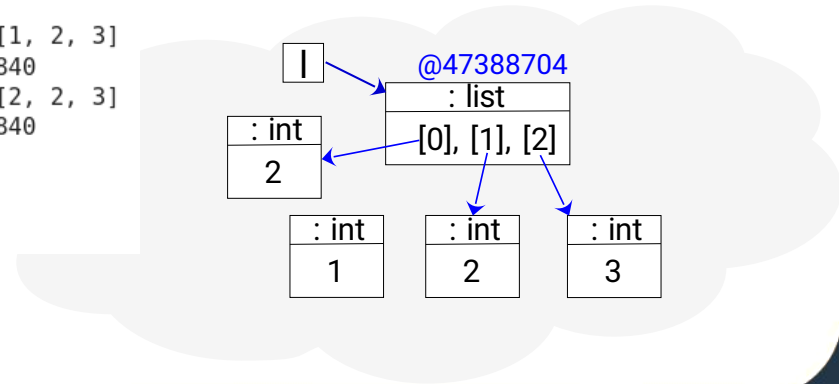
- Mutable : list, Point2D

- Le contenu des cases mémoires associé aux variables peut être modifié

```

1 l: list[int] = [1,2,3]
2 print('valeurs de l ligne 2', l)
3 print('id de l ligne 2', id(l))
4 l[0] = l[0] + 1
5 print('valeurs de l ligne 4', l)
6 print('id de l ligne 4', id(l))
    
```

valeurs de l ligne 2 [1, 2, 3]
 id de l ligne 2 53295840
 valeurs de l ligne 4 [2, 2, 3]
 id de l ligne 4 53295840



Mutable versus Immutable data types

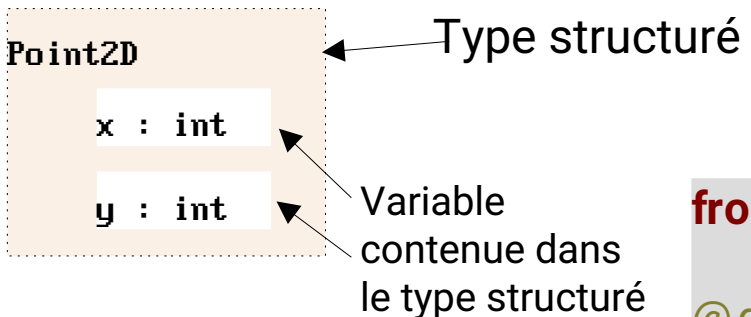
id(var) à la rescousse

- 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



```

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

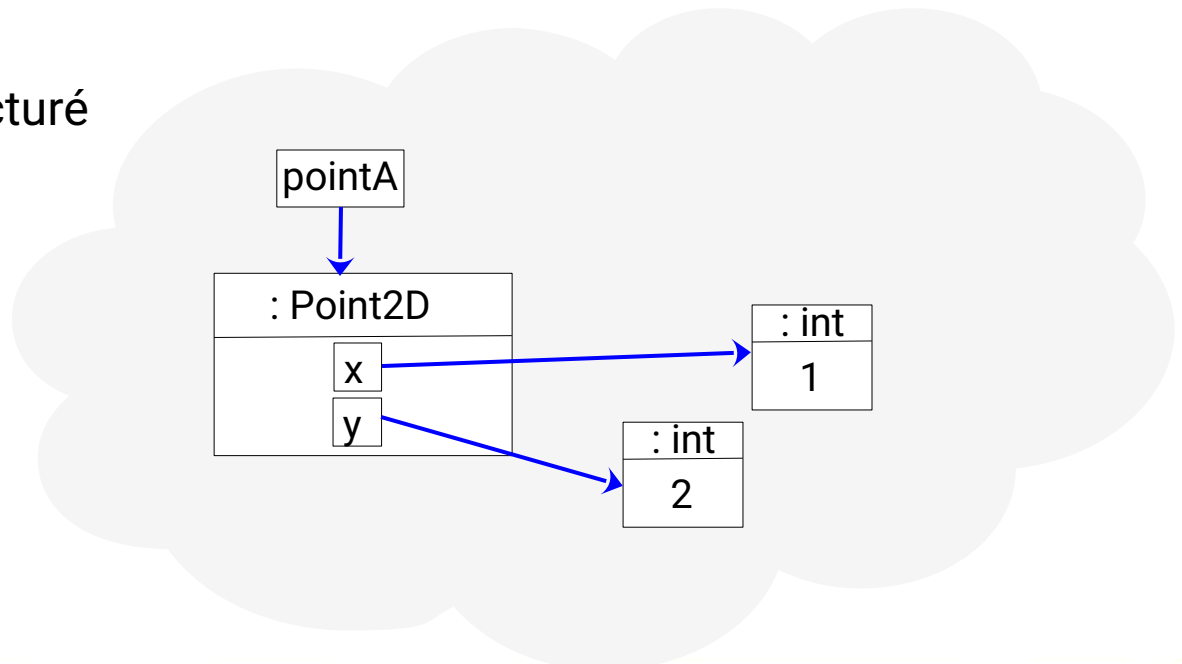
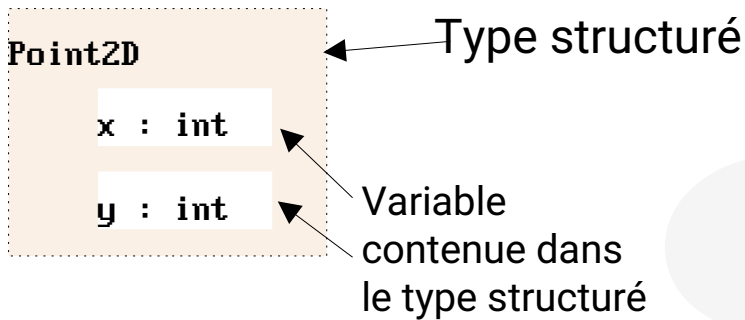
id(var) à la rescousse

- 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



Mutable versus Immutable data types

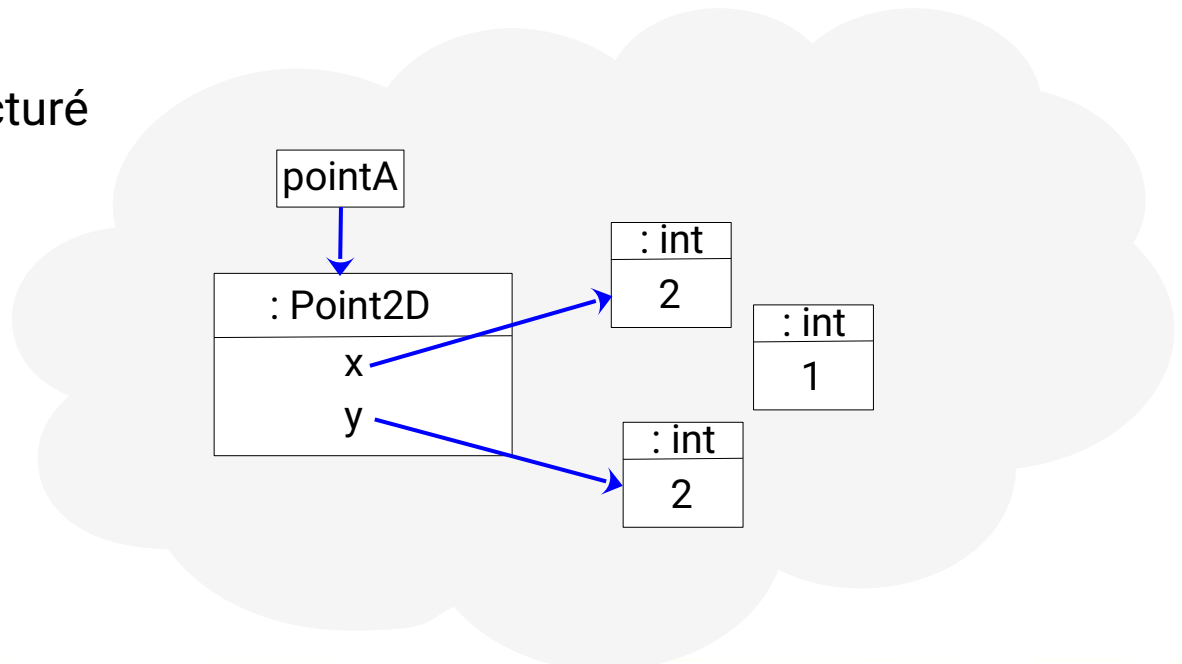
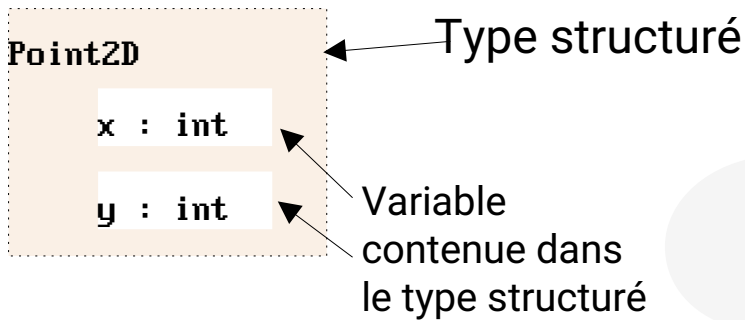
id(var) à la rescousse

- 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



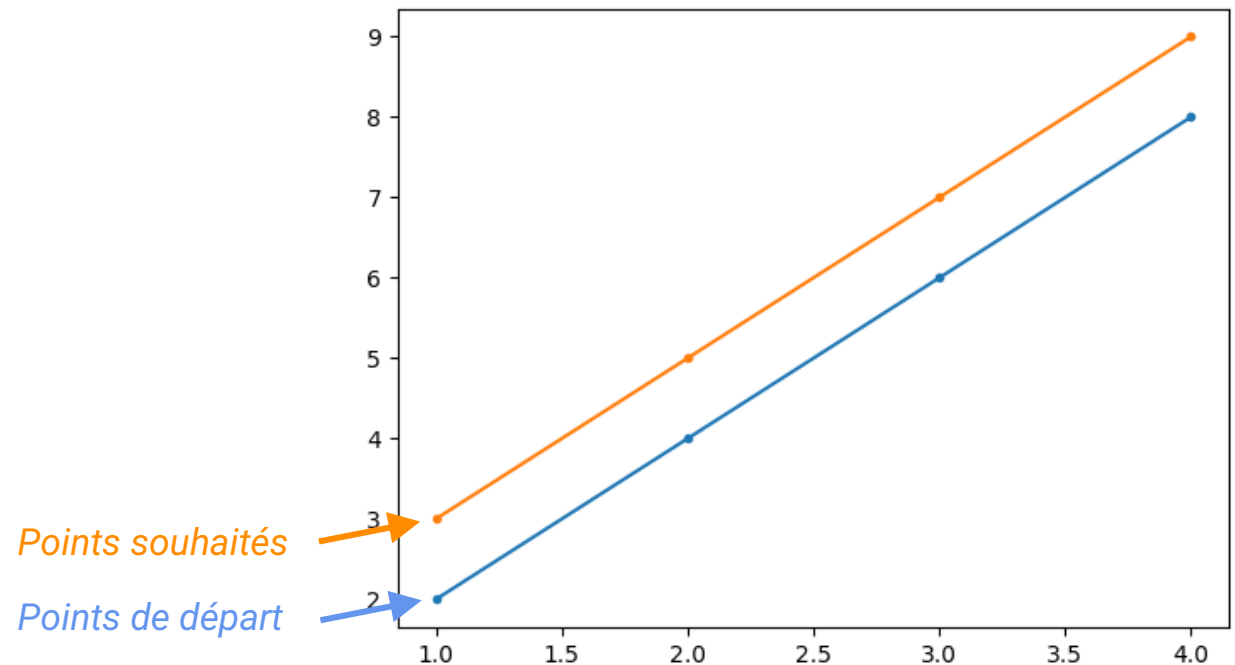
Premiers algorithmes

Premiers algorithmes

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

Premiers algorithmes

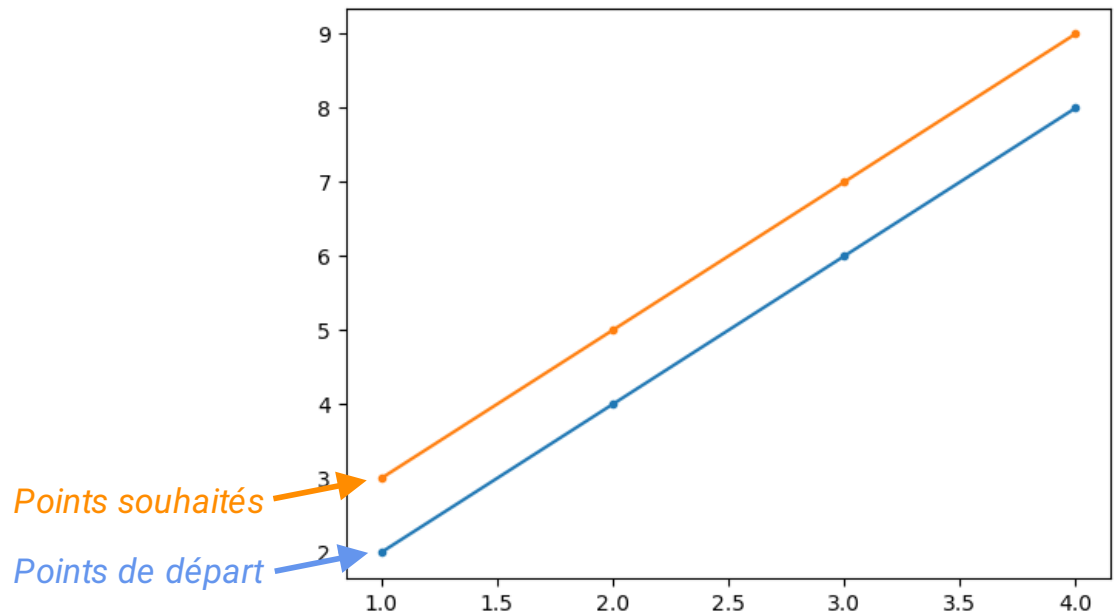
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



Premiers algorithmes


- On a un ensemble de 4 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



Premiers algorithmes


- On a un ensemble de 4 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
2
3
4
5
6
7

Séquence d'instructions



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

Premiers algorithmes

- On a un ensemble de 4 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

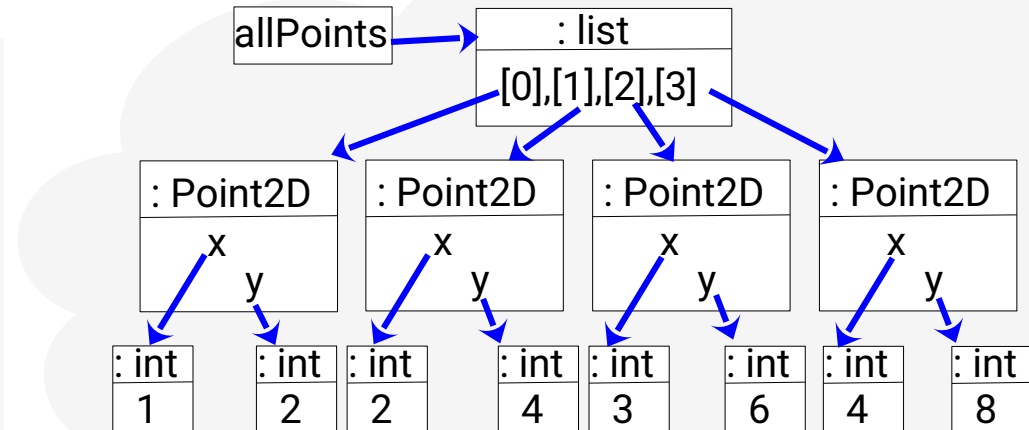


[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]

0
1
2
3

1
2
3
4
5
6
7

Séquence d'instructions



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

0
1
2
3

Premiers algorithmes

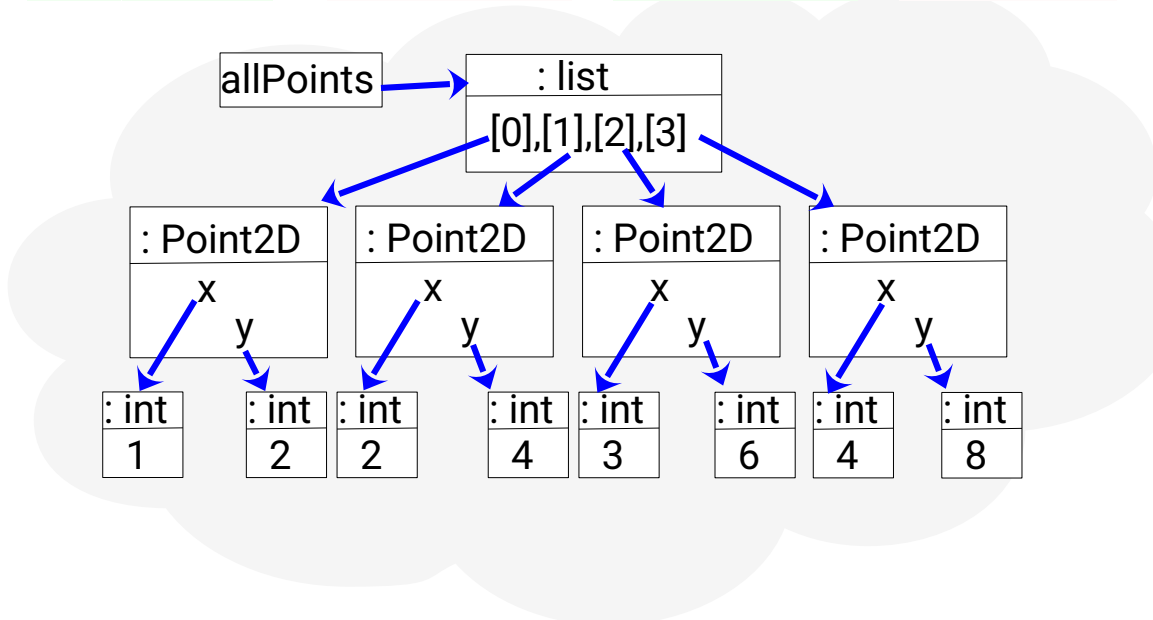
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



Premiers algorithmes

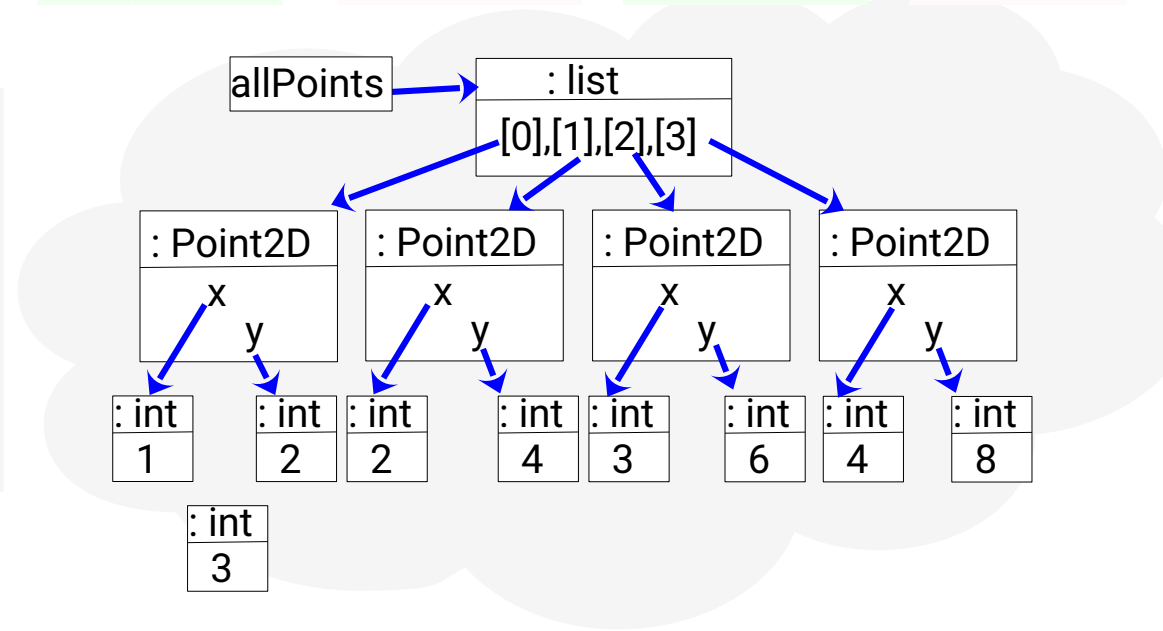
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



Premiers algorithmes

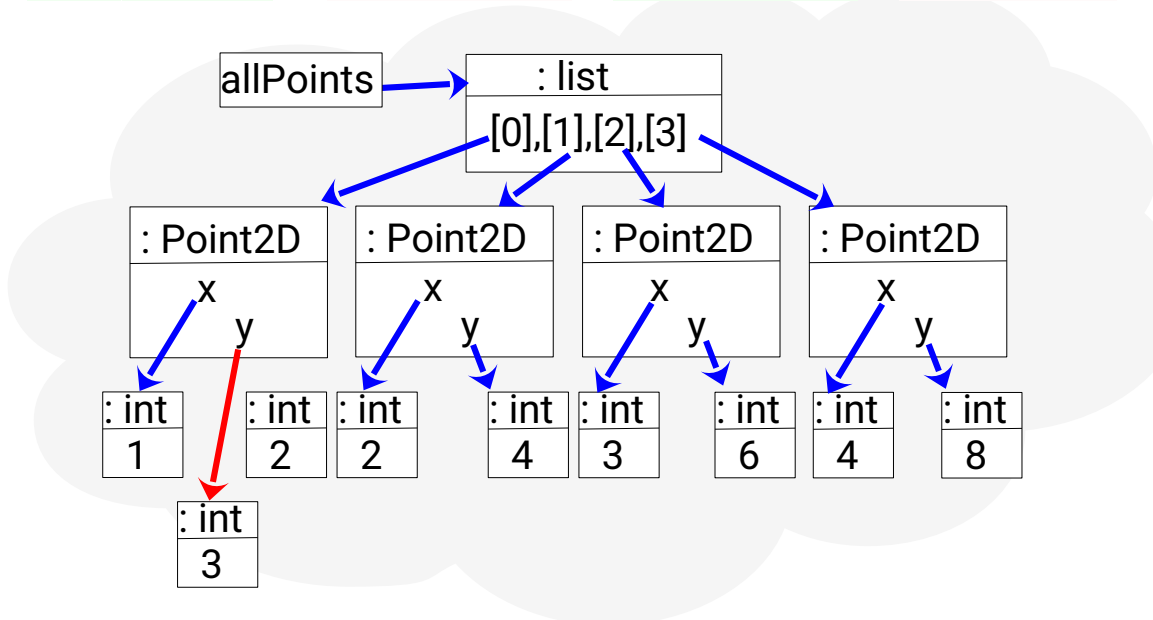
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



Premiers algorithmes

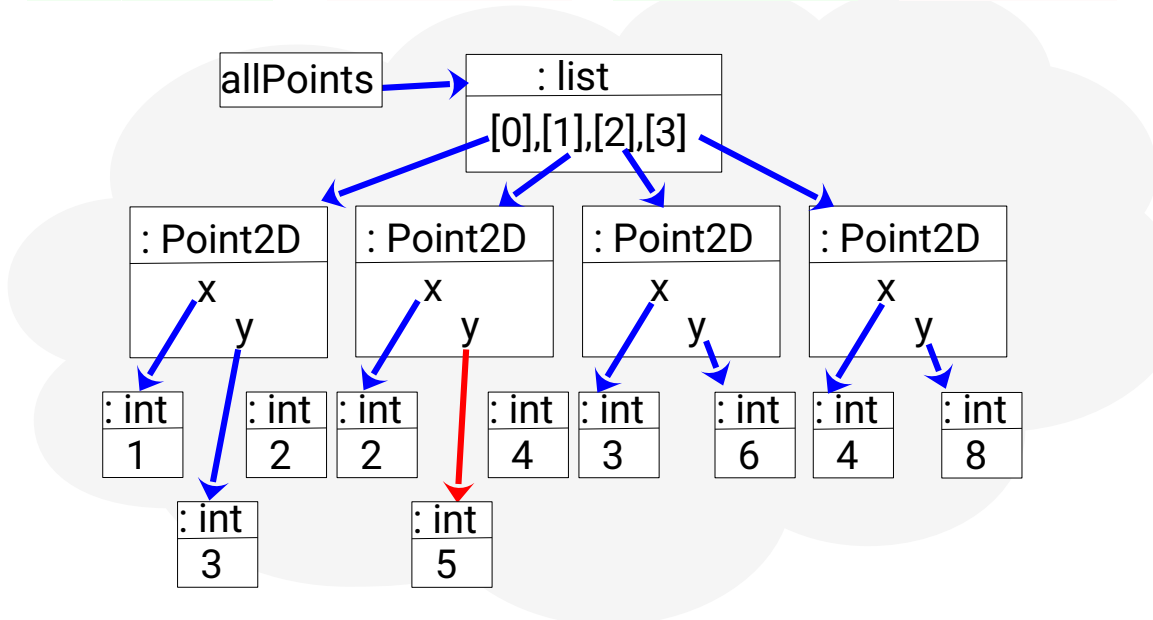
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



Premiers algorithmes

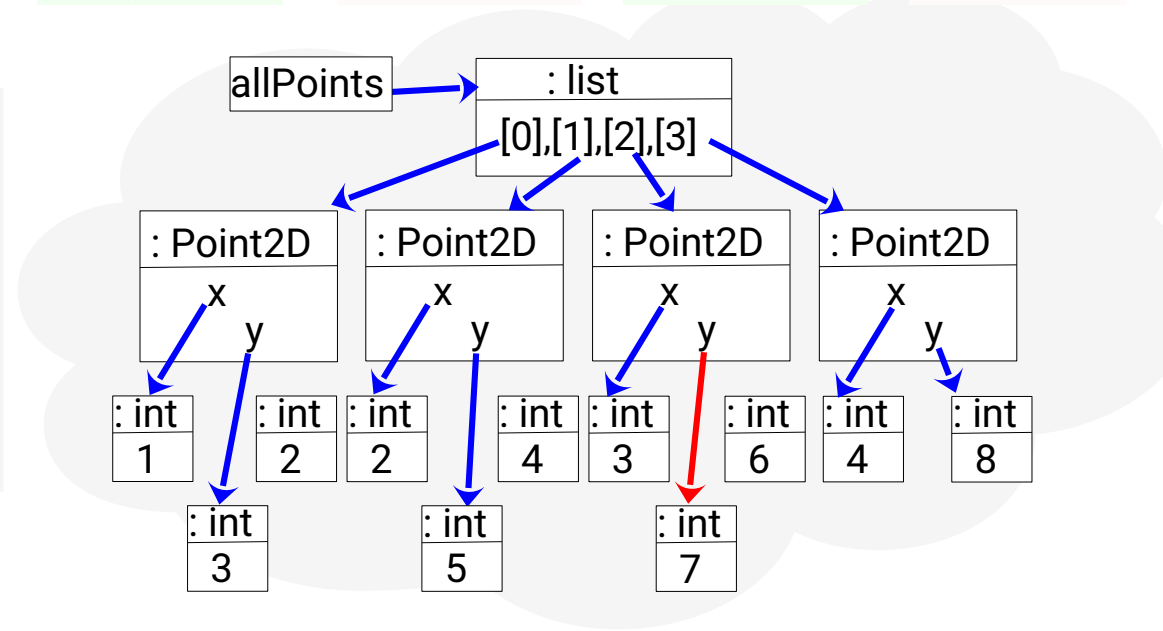
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



Premiers algorithmes

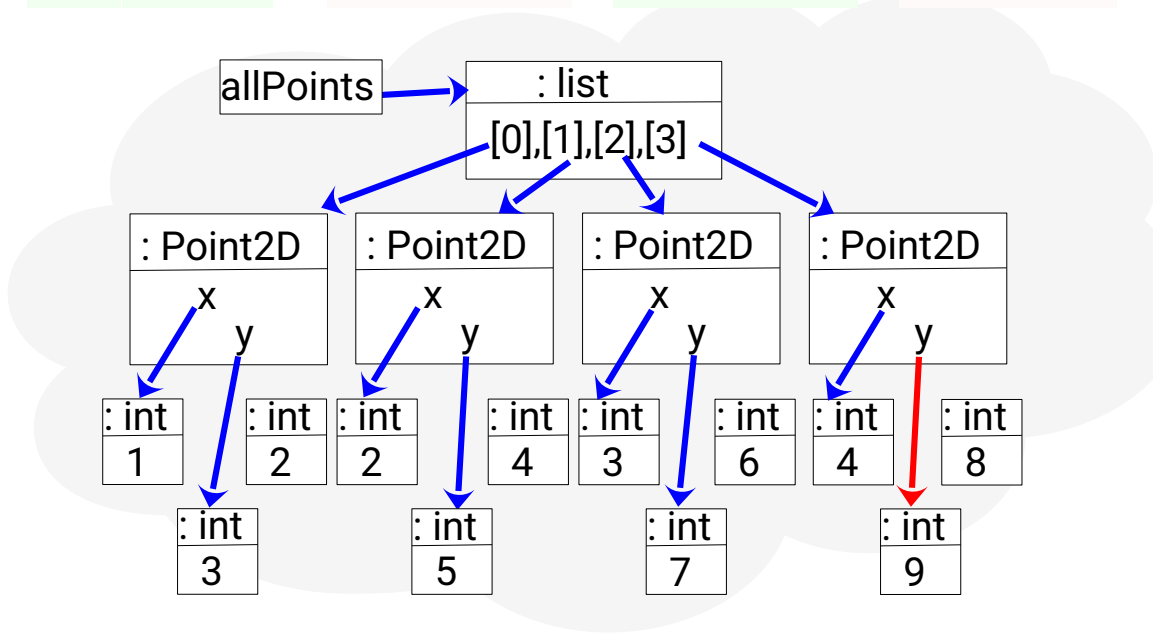
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



Premiers algorithmes

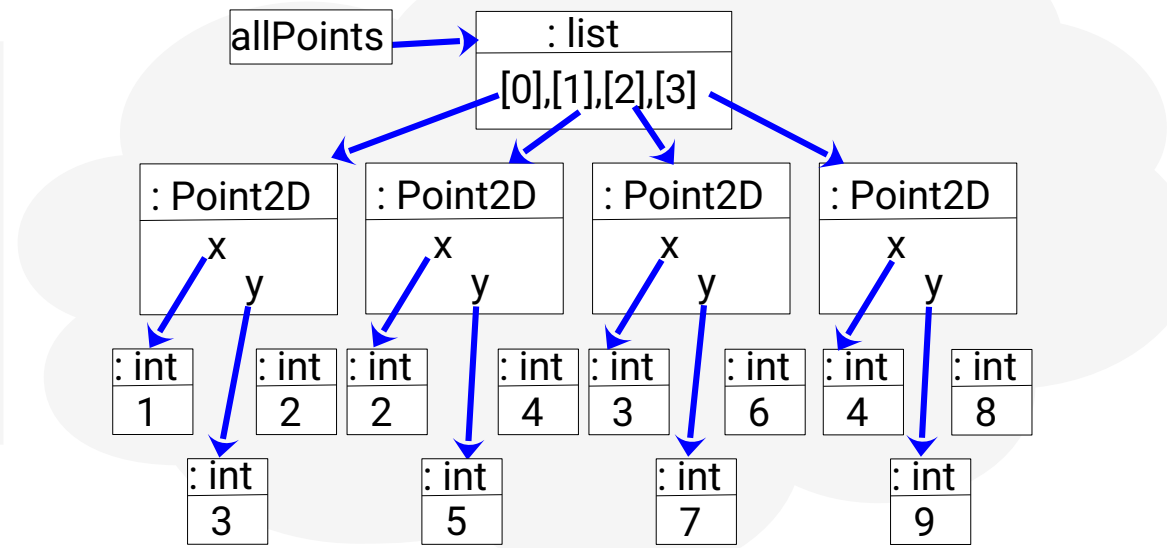
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste



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

```

1 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```



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

Premiers algorithmes

- On a un ensemble de 4 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 allPoints[0].y = allPoints[0].y + 1
2
3 allPoints[1].y = allPoints[1].y + 1
4
5 allPoints[2].y = allPoints[2].y + 1
6
7 allPoints[3].y = allPoints[3].y + 1
    
```

Step #	Line #	<code>allPoints</code>
1	1	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
	2	
2	3	[(x=1,y=3),(x=2,y=5),(x=3,y=6),(x=4,y=8)]
	4	
3	5	[(x=1,y=2),(x=2,y=5),(x=3,y=7),(x=4,y=8)]
	6	
4	7	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=9)]



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

Premiers algorithmes

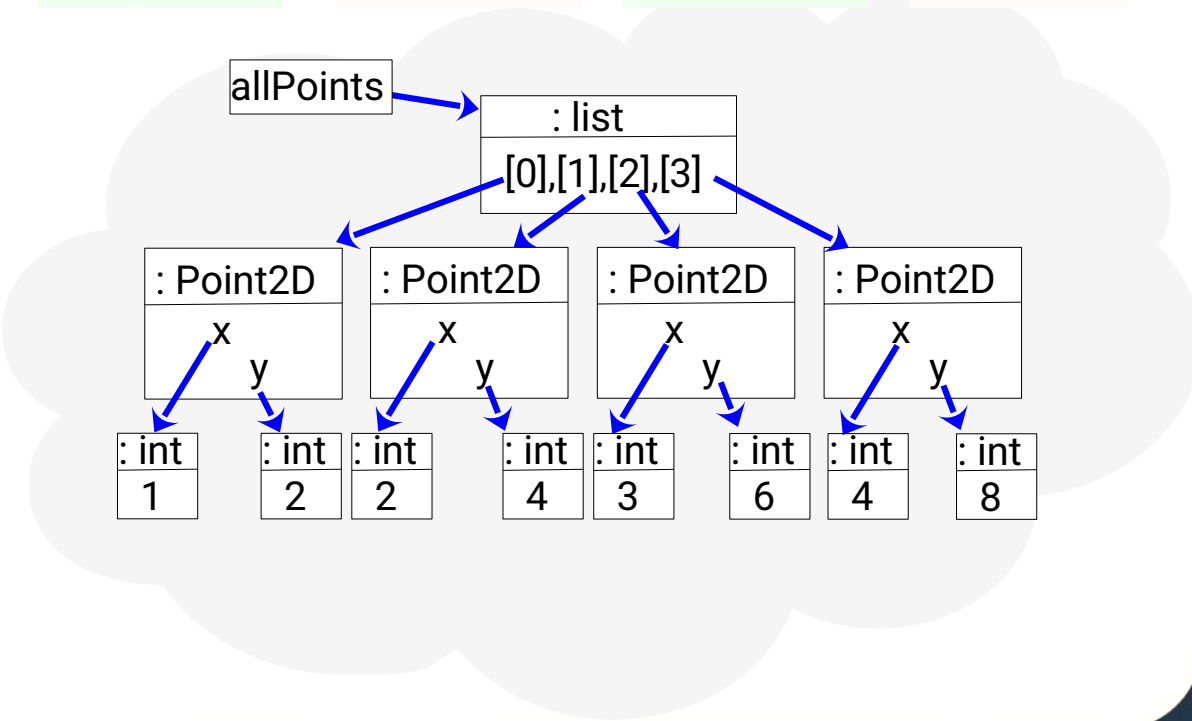
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- On désire appliquer une translation de 1 selon l'axe des ordonnées à tous les points de la liste

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

0
1
2
3

```

1 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



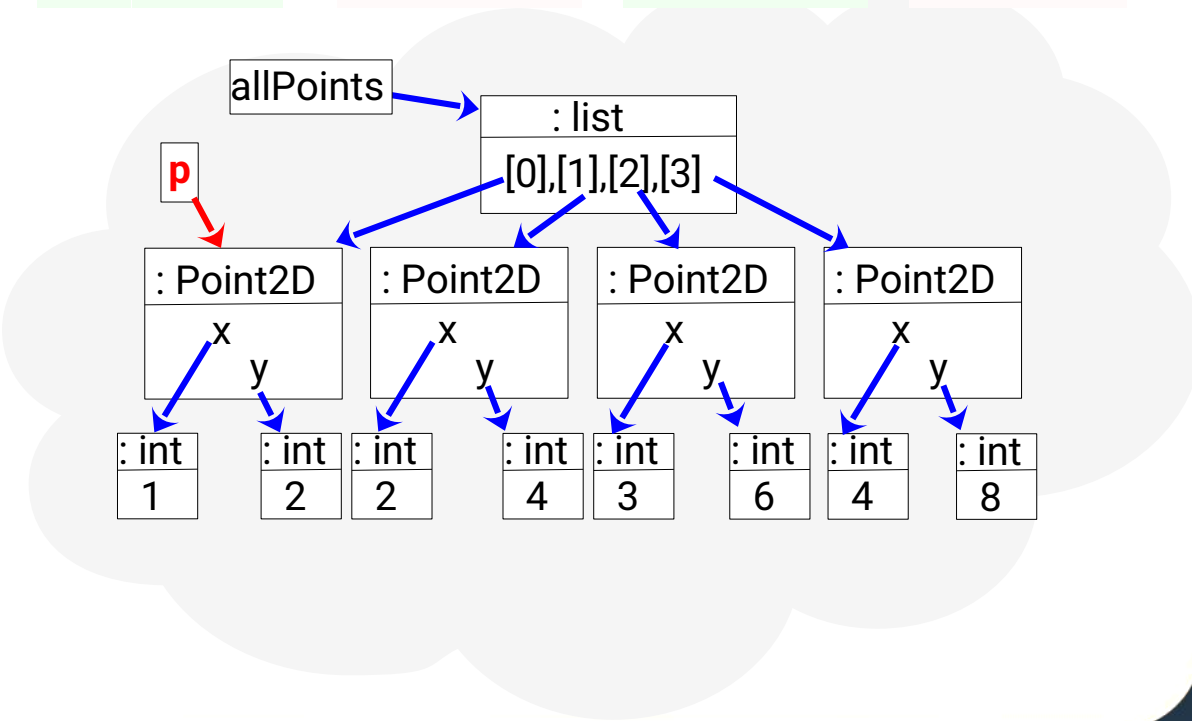
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



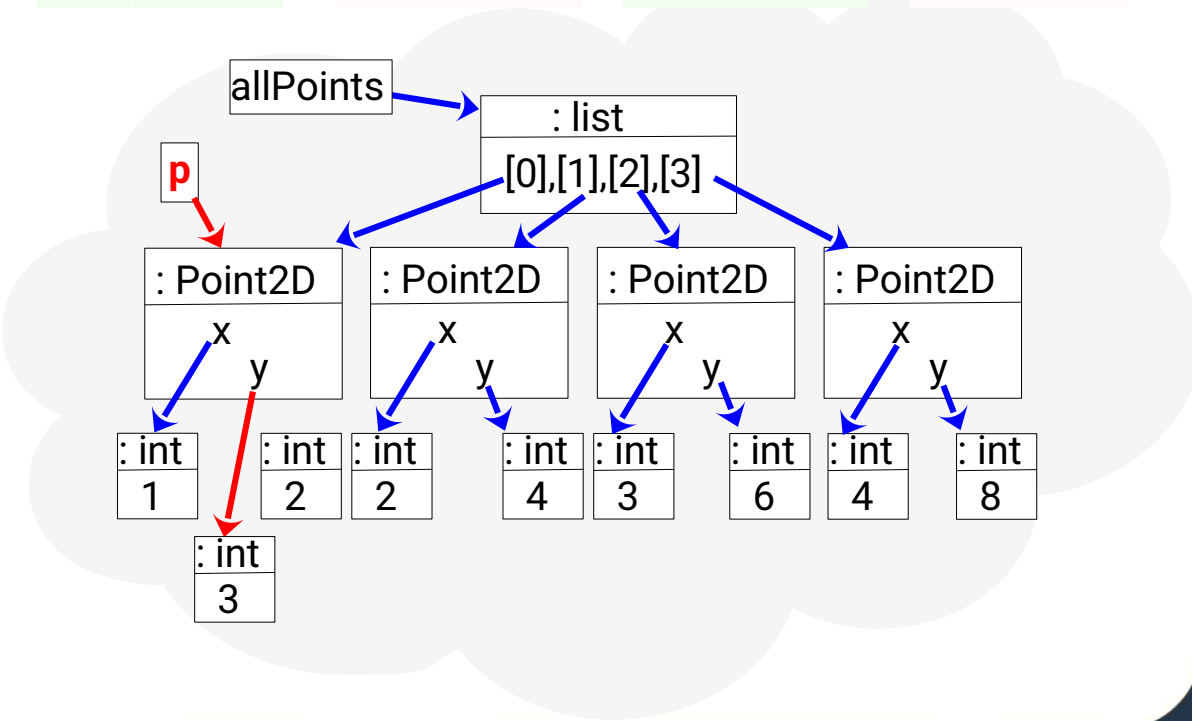
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



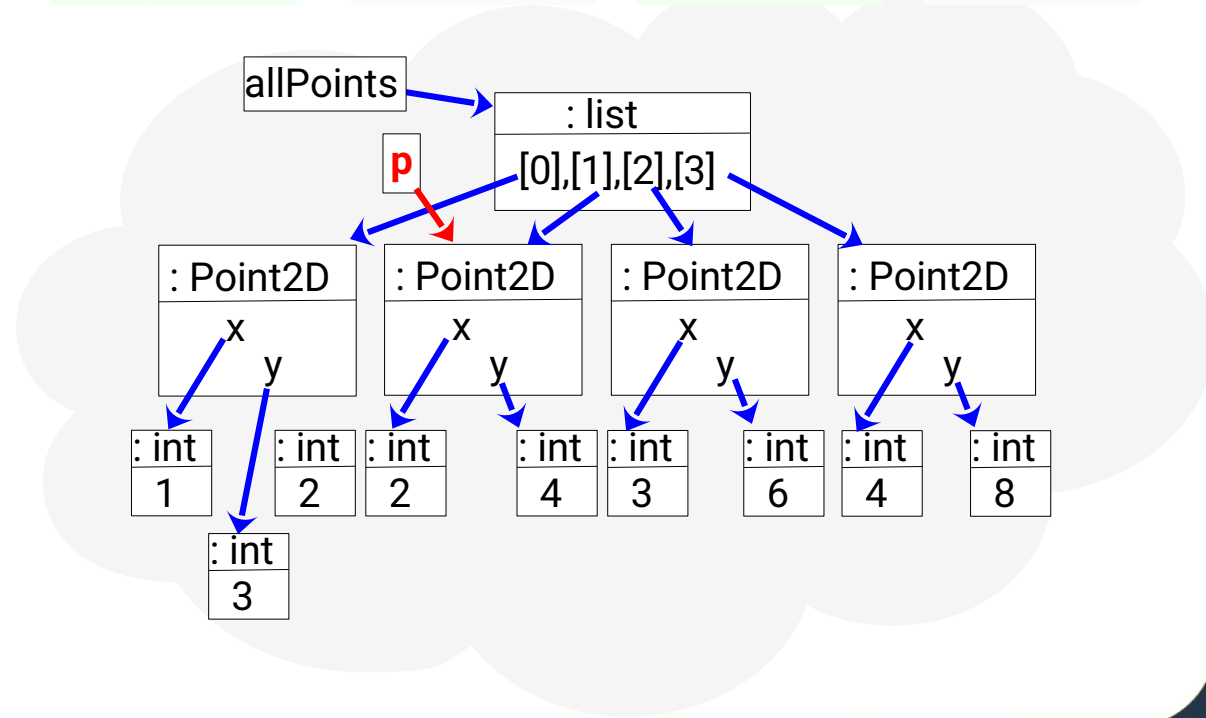
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



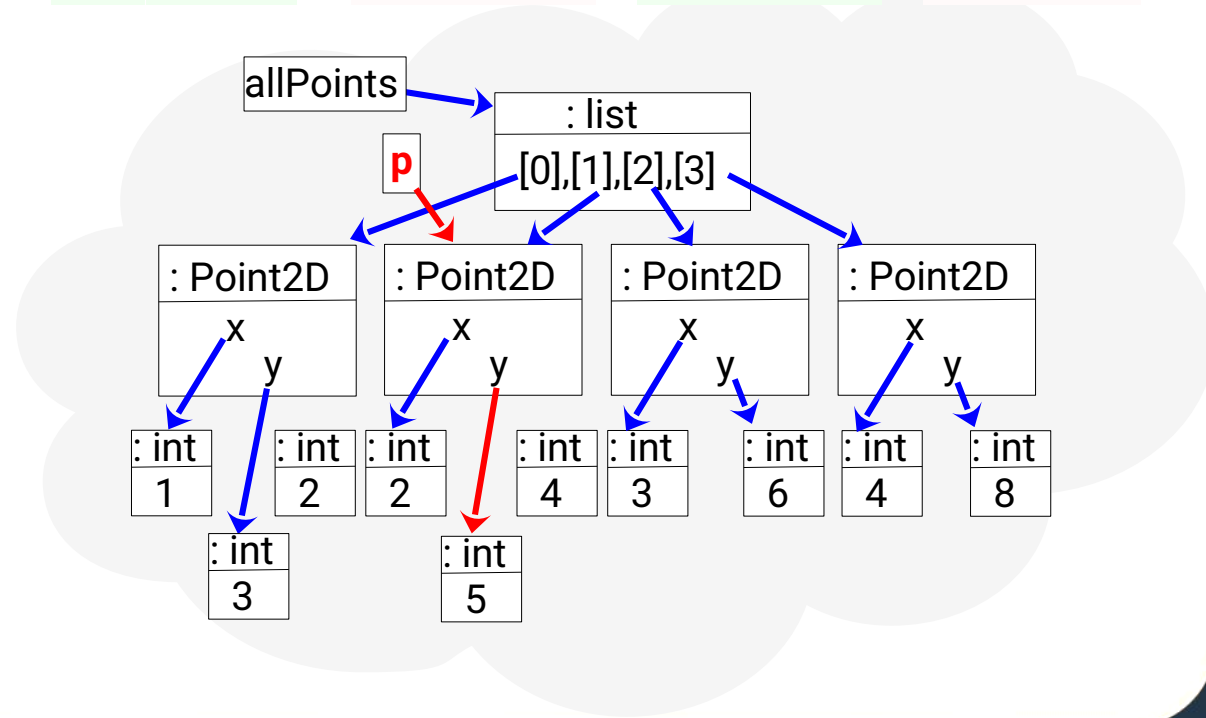
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



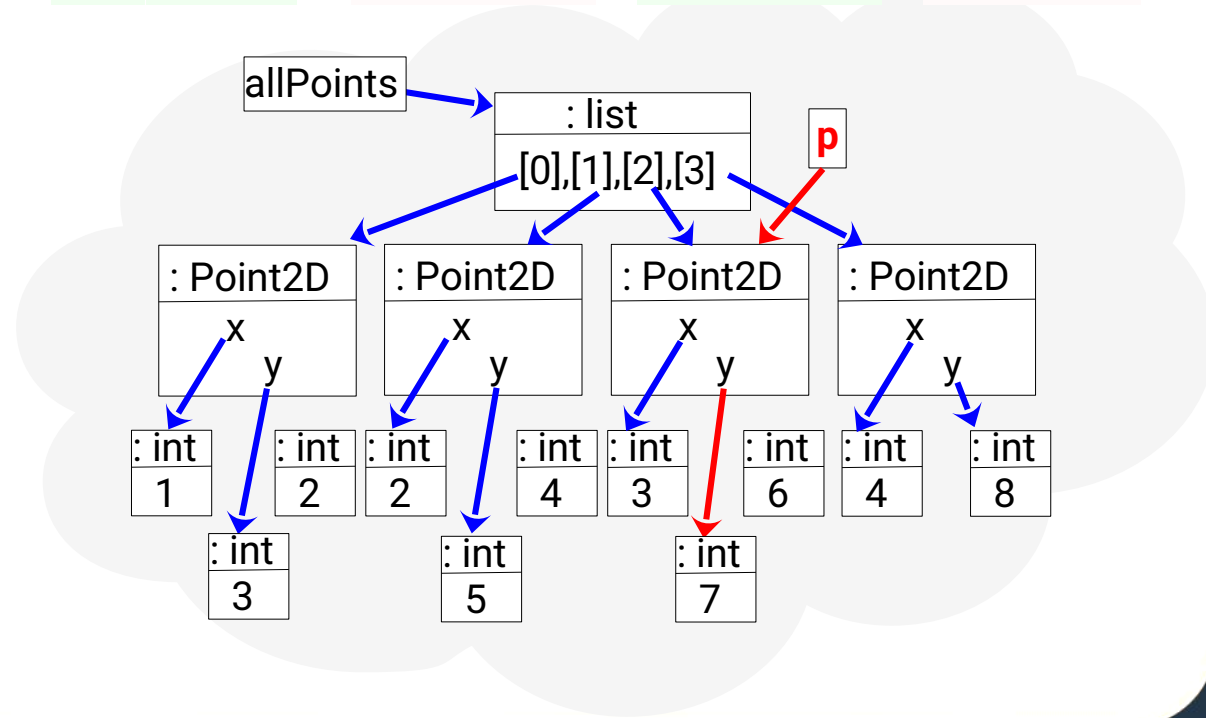
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



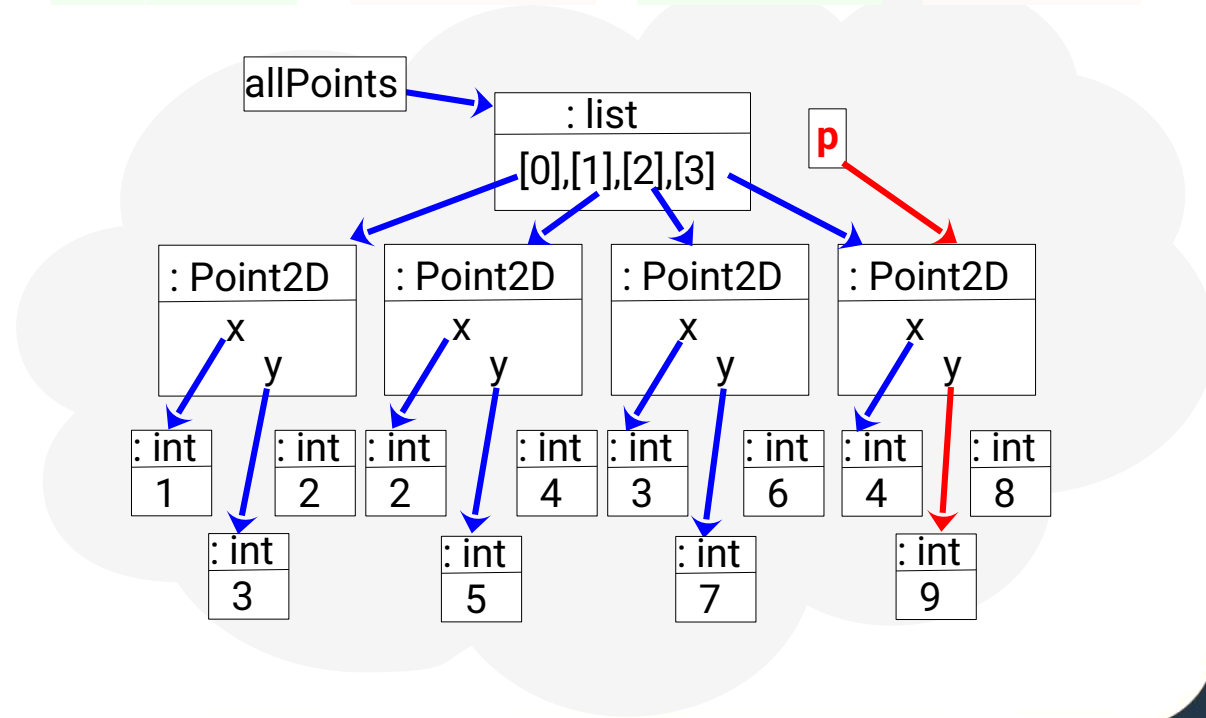
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



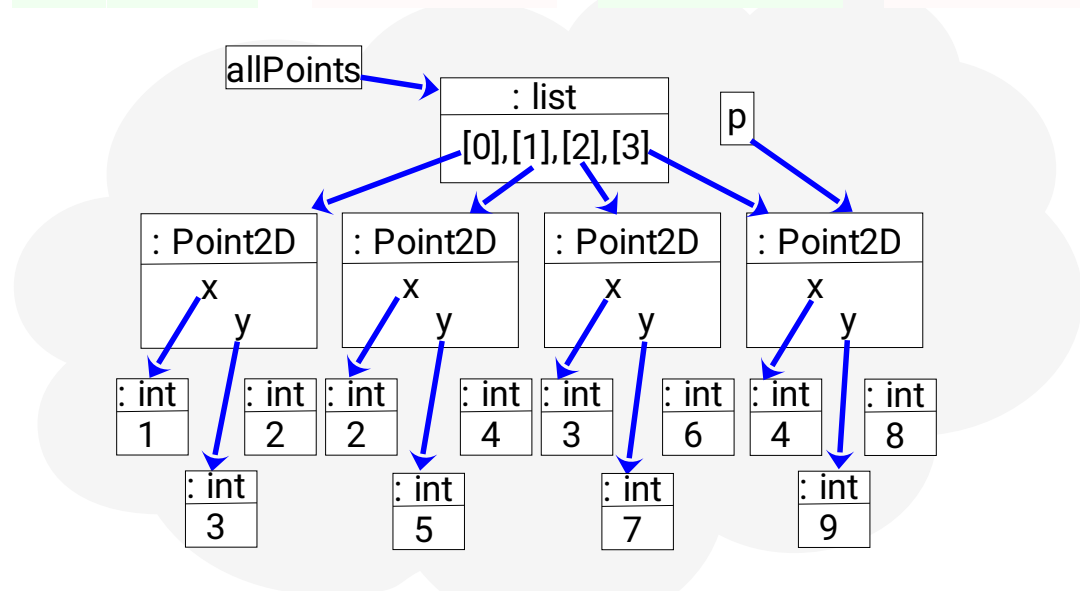
Premiers algorithmes

- On a un ensemble de 4 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 p: Point2D = allPoints[0]
2 p.y = p.y + 1
3
4 p = allPoints[1]
5 p.y = p.y + 1
6
7 p = allPoints[2]
8 p.y = p.y + 1
9
10 p = allPoints[3]
11 p.y = p.y + 1
    
```



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

Premiers algorithmes

- On a un ensemble de 4 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

	Step #	Line #	p	allPoints
1	1	1	(x=1, y=2)	[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
2	2	2	(x=1, y=3)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
3		3	(x=1, y=3)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
4	3	4	(x=2, y=4)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
5	4	5	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
6		6	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
7	5	7	(x=3, y=6)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
8	6	8	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
9		9	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
10	7	10	(x=4, y=8)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
11	8	11	(x=4, y=9)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]

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

0
1
2
3

Premiers algorithmes

- On a un ensemble de 4 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

```

indice: int = 0

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
    
```

```

p: Point2D = allPoints[0]
p.y = p.y + 1

p = allPoints[1]
p.y = p.y + 1

p = allPoints[2]
p.y = p.y + 1

p = allPoints[3]
p.y = p.y + 1
    
```



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

0 1 2 3

Premiers algorithmes

- On a un ensemble de 4 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

```

indice: int = 0

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
    
```

Séquences d'instructions exactement identiques

⇒ l'ordinateur sait répéter la même chose plein de fois

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

0 1 2 3

Premiers algorithmes

- On a un ensemble de 4 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

```

indice: int = 0

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
    
```

Séquences d'instructions exactement identiques

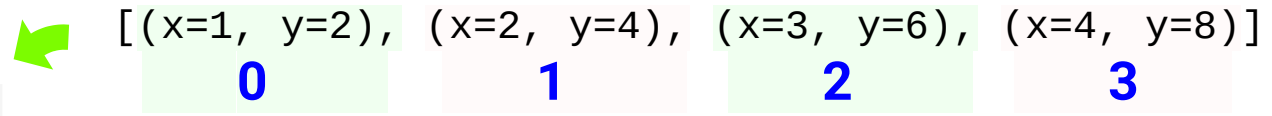
⇒ Tant que `indice` est inférieur ou égal à 3, on exécute la séquence

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

0 1 2 3

Premiers algorithmes

- On a un ensemble de 4 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



```

indice: int = 0

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

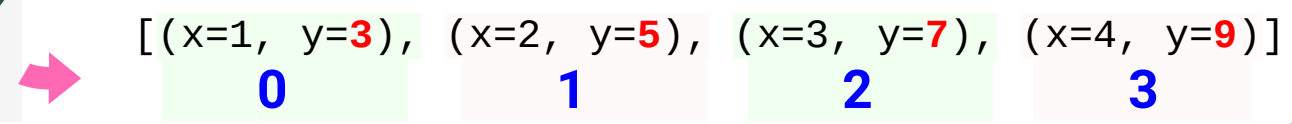
p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
    
```

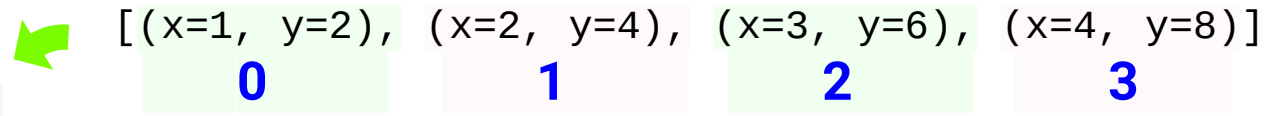
Séquences d'instructions exactement identiques

⇒ Tant que `indice` est inférieur à 4, on exécute la séquence



Premiers algorithmes

- On a un ensemble de 4 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



```

indice: int = 0

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

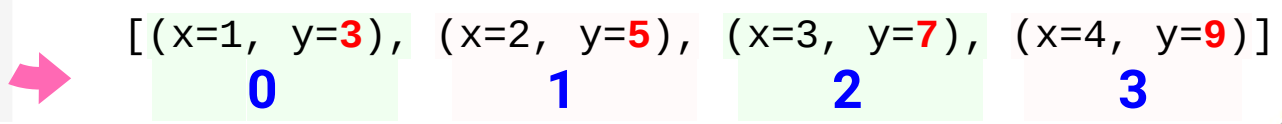
p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
    
```

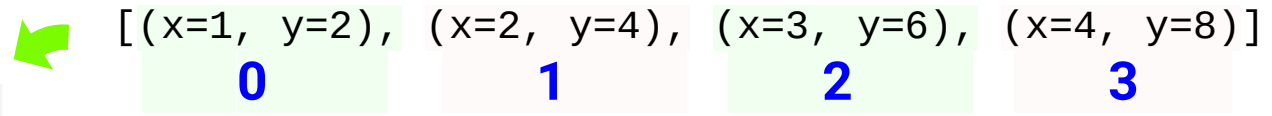
Séquences d'instructions exactement identiques

⇒ Tant que `indice < 4`, on exécute la séquence



Premiers algorithmes

- On a un ensemble de 4 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



```

indice: int = 0

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

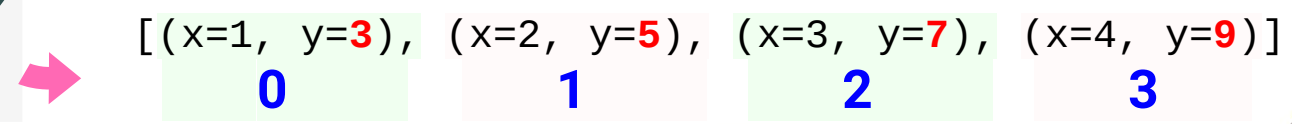
p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
    
```

Séquences d'instructions exactement identiques

⇒ `while (indice < 4):`
on exécute la séquence



Premiers algorithmes

- On a un ensemble de 4 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

indice: int = 0

p: Point2D = allPoints[indice]
 p.y = p.y + 1
 indice = indice + 1

p: Point2D = allPoints[indice]
 p.y = p.y + 1
 indice = indice + 1

p: Point2D = allPoints[indice]
 p.y = p.y + 1
 indice = indice + 1

p: Point2D = allPoints[indice]
 p.y = p.y + 1
 indice = indice + 1

Séquences d'instructions exactement identiques

⇒ while (indice < 4):

```

p: Point2D = allPoints[indice]
p.y = p.y + 1
indice = indice + 1
```


→ [(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 4 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 < 4):
3     p: Point2D = allPoints[indice]
4     p.y = p.y + 1
5     indice = indice + 1
```


 [(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 4 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 < 4):
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	<u>undefined</u>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
2	2	0	<u>undefined</u>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
3	3	0	<code>(x=1, y=2)</code>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
4	4	0	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
5	5	1	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>

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

`[(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 4 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

	Step #	Line #	i	p	allPoints
1	1	1	0	undefined	[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
2	2	2	0	undefined	[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
3	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	4	0	(x=1, y=3)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
5	5	5	1	(x=1, y=3)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
6	6	2	1	(x=1, y=3)	[(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 est le code à exécuter **tant que la condition** est vraie

Premiers algorithmes

La boucle While

- On a un ensemble de 4 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 < 4):
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

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

Premiers algorithmes


La boucle While

- On a un ensemble de 4 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 < 4):
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

Step #	Line #	i	p	allPoints
1	1	0	<u>undefined</u>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
2	2	0	<u>undefined</u>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
3	3	0	<code>(x=1, y=2)</code>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
4	4	0	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
5	5	1	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
6	2	1	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
7	3	1	<code>(x=2, y=4)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
8	4	1	<code>(x=2, y=5)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>
9	5	2	<code>(x=2, y=5)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>
10	2	2	<code>(x=2, y=5)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>

Premiers algorithmes


La boucle While

- On a un ensemble de 4 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)]

```

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')
```


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

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	(x=1, y=3)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
7	3	1	(x=2, y=4)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
8	4	1	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
9	5	2	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
10	2	2	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
11	3	2	(x=3, y=6)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
12	4	2	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
13	5	3	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
14	2	3	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]

Premiers algorithmes


La boucle While

- On a un ensemble de 4 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 < 4):
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

Step #	Line #	i	p	allPoints
1	1	0	<u>undefined</u>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
2	2	0	<u>undefined</u>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
3	3	0	<code>(x=1, y=2)</code>	<code>[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
4	4	0	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
5	5	1	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
6	2	1	<code>(x=1, y=3)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
7	3	1	<code>(x=2, y=4)</code>	<code>[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]</code>
8	4	1	<code>(x=2, y=5)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>
9	5	2	<code>(x=2, y=5)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>
10	2	2	<code>(x=2, y=5)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>
11	3	2	<code>(x=3, y=6)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]</code>
12	4	2	<code>(x=3, y=7)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]</code>
13	5	3	<code>(x=3, y=7)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]</code>
14	2	3	<code>(x=3, y=7)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]</code>
15	3	3	<code>(x=4, y=8)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]</code>
16	4	3	<code>(x=4, y=9)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]</code>
17	5	4	<code>(x=4, y=9)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]</code>
18	2	4	<code>(x=4, y=9)</code>	<code>[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]</code>

Premiers algorithmes

La boucle While

- On a un ensemble de 4 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 < 4):
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


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

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	(x=1, y=3)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
7	3	1	(x=2, y=4)	[(x=1, y=3), (x=2, y=4), (x=3, y=6), (x=4, y=8)]
8	4	1	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
9	5	2	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
10	2	2	(x=2, y=5)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
11	3	2	(x=3, y=6)	[(x=1, y=3), (x=2, y=5), (x=3, y=6), (x=4, y=8)]
12	4	2	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
13	5	3	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
14	2	3	(x=3, y=7)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
15	3	3	(x=4, y=8)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=8)]
16	4	3	(x=4, y=9)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]
17	5	4	(x=4, y=9)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]
18	2	4	(x=4, y=9)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]
19	6	4	(x=4, y=9)	[(x=1, y=3), (x=2, y=5), (x=3, y=7), (x=4, y=9)]

Premiers algorithmes


La boucle While

- On a un ensemble de 4 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 < 4):
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 ~~4~~ 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

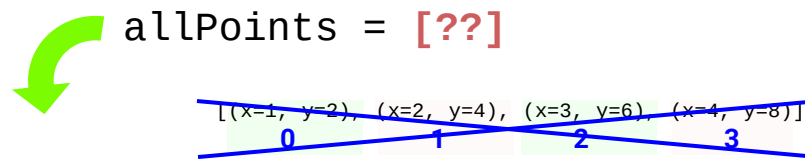
 `allPoints = [??]`
~~`[(x=1, y=2), (x=2, y=4), (x=3, y=6), (x=4, y=8)]`~~
~~`0 1 2 3`~~

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

Premiers algorithmes

La boucle While

- On a un ensemble de ~~4~~ 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 < 4):
3     p: Point2D = allPoints[indice]
4     p.y = p.y + 1
5     indice = indice + 1
6 print('traitement fini')
```

⇒ le problème a changé et devient plus générique puisque **l'on ne sait pas a priori** combien de fois il faut répéter le bloc de code représentant le traitement
→ on va donc inspecter la variable `allPoints` et modifier **la condition de répétition** en conséquence.

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

 `allPoints = [??]`

```
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')
```

⇒ le problème a changé et devient plus générique puisque **l'on ne sait pas a priori** combien de fois il faut répéter le bloc de code représentant le traitement
→ on va donc inspecter la variable `allPoints` et modifier **la condition de répétition** en conséquence.

Ici la fonction `len(var)` est définie si `var` est de type `list` et elle renvoie le nombre d'élément dans la liste `var`

<https://www.google.com/search?q=longueur+list+python>

- 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**

- 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**
- La condition porte habituellement sur une ou plusieurs variables qui sont modifiées dans le bloc de code à répéter pour éviter d'avoir une boucle infinie

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


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**
- La condition porte habituellement sur une ou plusieurs variables qui sont modifiées dans le bloc de code à répéter pour éviter d'avoir une boucle infinie

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

⇒ Boucle infinie car `(indice < 4)` sera toujours vrai
La ligne 6 ne sera jamais exécutée

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**
- La condition porte habituellement sur une ou plusieurs variables qui sont modifiées dans le bloc de code à répéter pour éviter d'avoir une boucle infinie

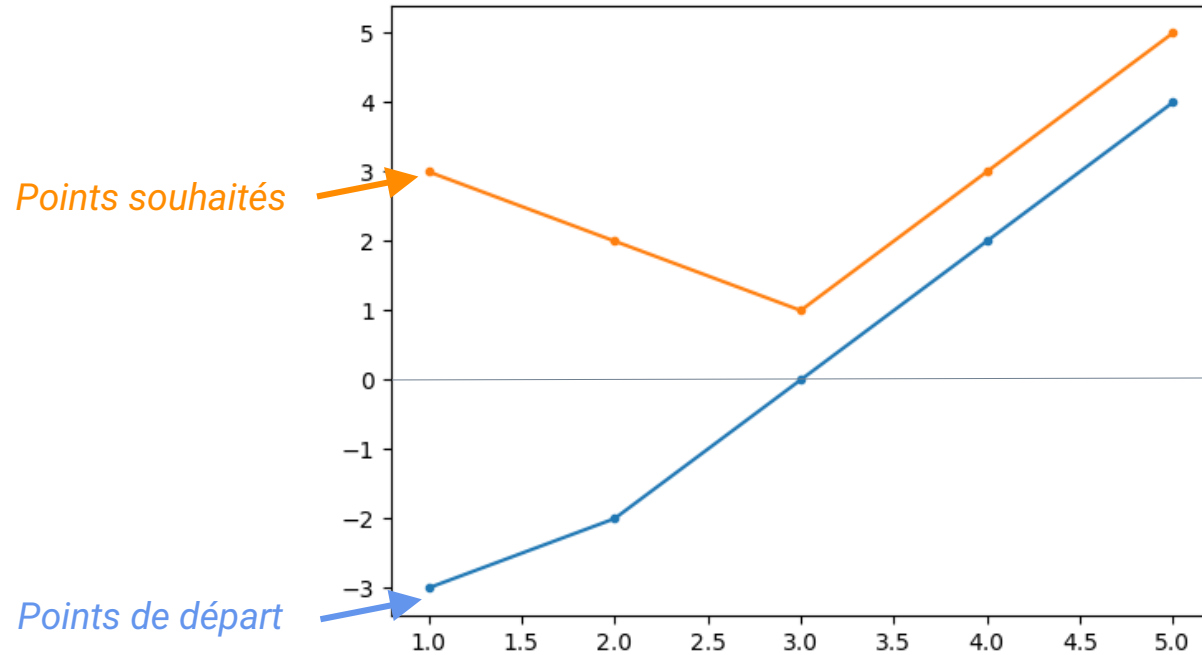
```
1 indice: int = 0
2 while (indice < 4):
3     p: Point2D = allPoints[indice]
4     p.y = p.y + 1
5     indice = indice + 1
6 print('traitement fini')
```

⇒ Boucle finie car `(indice < 4)` finira par être faux

Premiers algorithmes

Le branchement conditionnel *If*

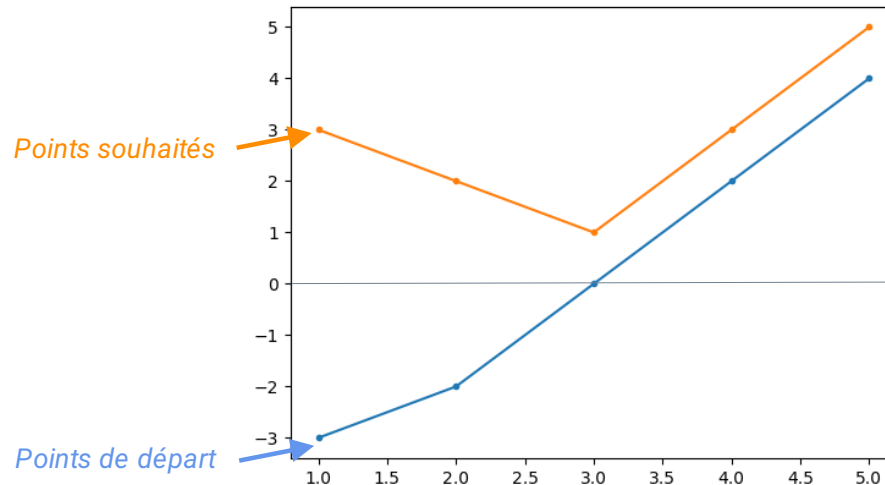
- On a un ensemble de 4 points dans une liste nommée `allPoints`.
- Si l'ordonnée d'un point est négative, on désire la remplacer par sa valeur absolue
- Si l'ordonnée d'un point n'est pas négative, alors on lui ajoute 1



Premiers algorithmes

Le branchement conditionnel If

- On a un ensemble de 4 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



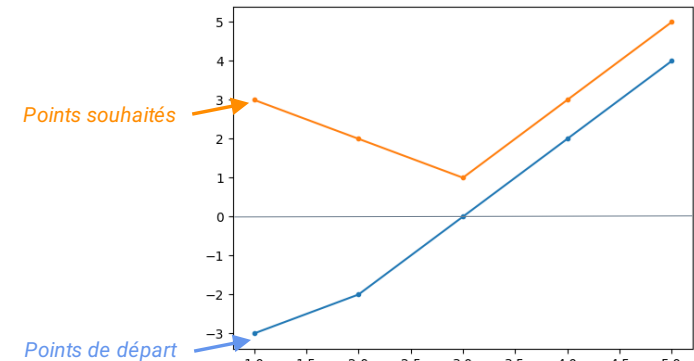
- 1) On parcourt toute la liste comme précédemment
- 2) On ajoute du code qui représente le traitement conditionnel
 - 1) On remarque que les points 2 et 3 ci dessus sont exclusifs !

Premiers algorithmes

Le branchement conditionnel *If*

- On a un ensemble de 4 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
5
6     indice = indice + 1
7
8     print('traitement fini')
```

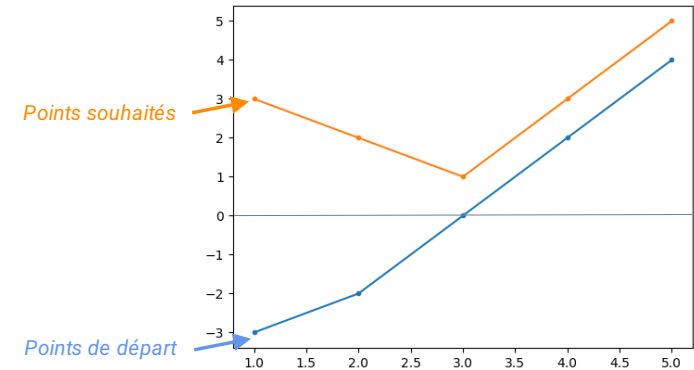


Premiers algorithmes

Le branchement conditionnel *If*

- On a un ensemble de 4 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 : en python, les blocs de code sont définis en fonction de l'indentation (du retrait par rapport au bord du fichier)

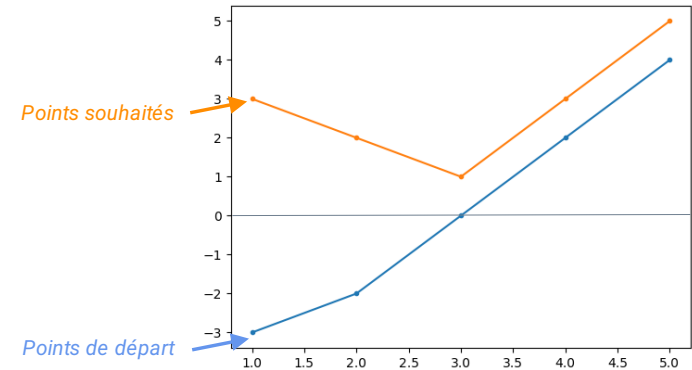
Premiers algorithmes

Le branchement conditionnel *If*

- On a un ensemble de 4 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 : en python, les blocs de code sont définis en fonction de l'indentation (du retrait par rapport au bord du fichier)

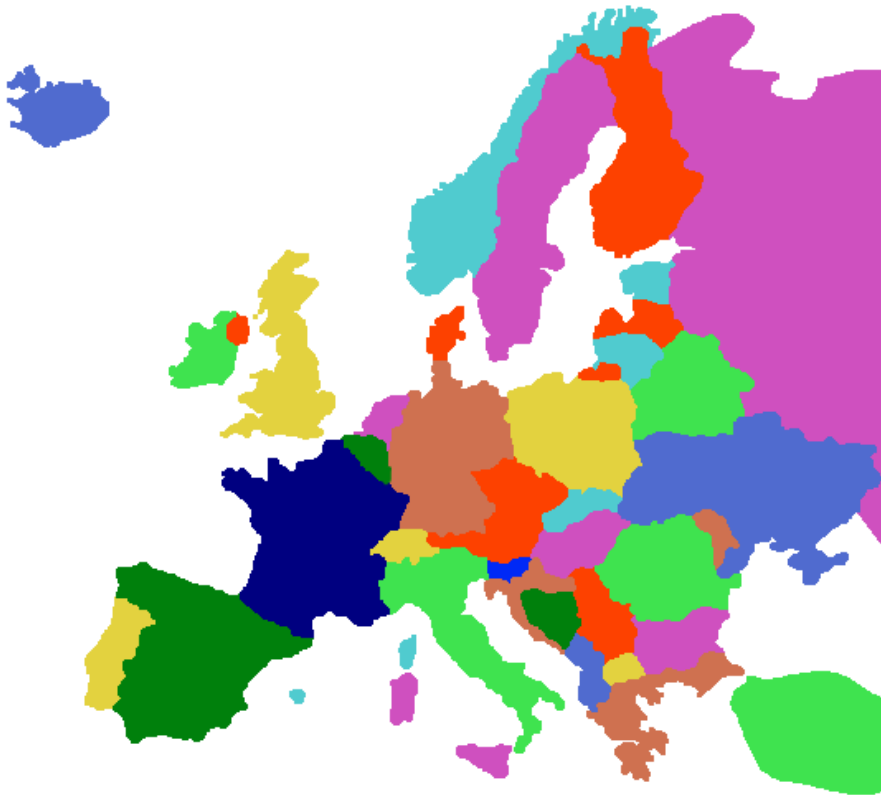
Algorithmes

- Les structures de code conditionnel et les boucles peuvent s'imbriquer de nombreuses manières, engendrant des comportements différents.
- Il est important de pratiquer afin de progresser. N'hésitez pas à jouer dans l'éditeur utilisé en TP et à étudier les valeurs des variables à chaque ligne. L'exécution ligne par ligne vous permet de comprendre quel est le flot d'exécution.
- Challengez vous les uns les autres. Posez vous des questions, entre aidez vous.
- Si vous avez des questions particulières et que personne autour de vous n'a la réponse, n'hésitez pas à me contacter moi ou les chargés de TD/TP.

Encodage des images. Un exemple simple

Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :



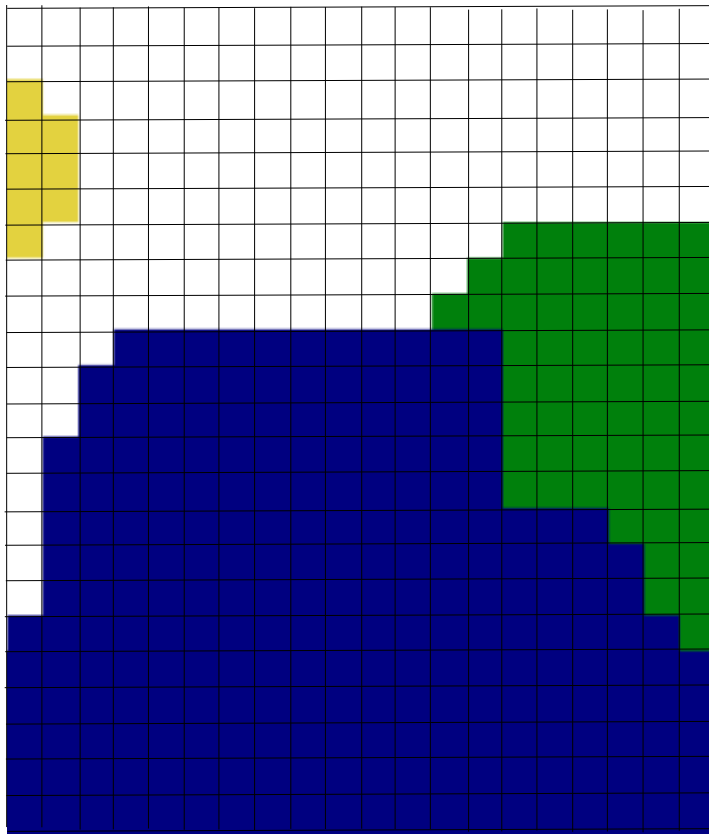
Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :



Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :



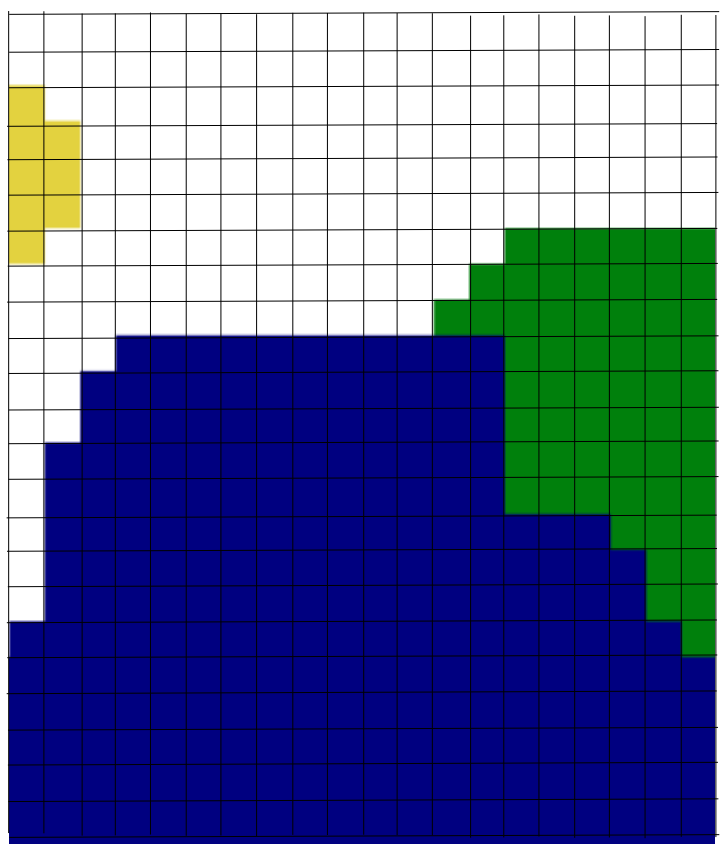
L'image est décomposée en *Pixels* (*Picture elements*).

Un pixel est la plus petite partie de l'image et n'est pas divisible

Combien de pixel dans une image aujourd'hui ?

Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :



L'image est décomposée en *Pixels* (*Picture elements*).

Un pixel est la plus petite partie de l'image et n'est pas divisible

Combien de pixel dans une image aujourd'hui ?

Double appareil photo premium
DOUBLE OBJECTIF DE 50 MP

FAIRPHONE 5 <https://shop.fairphone.com/fr/>

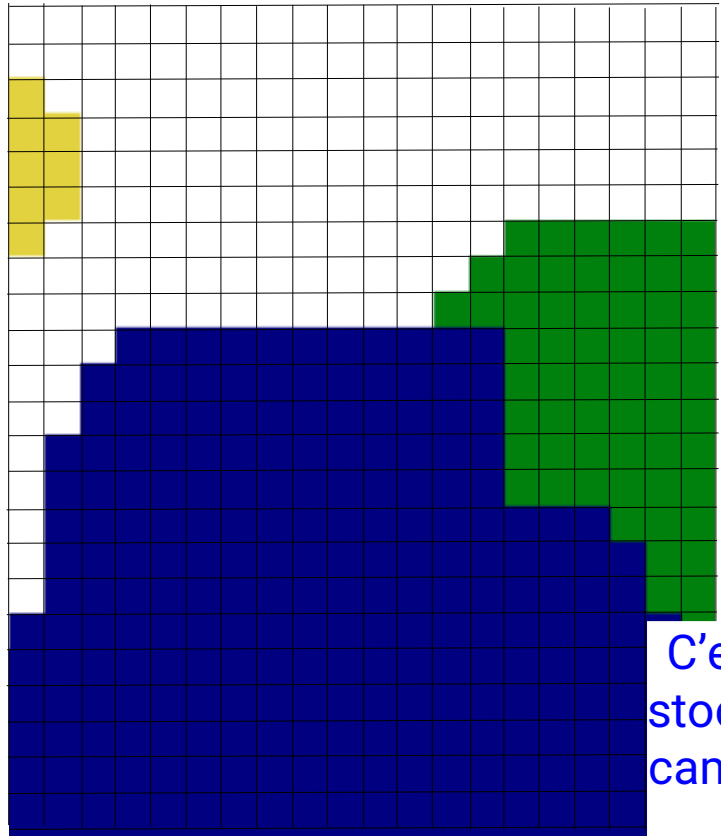
Nouvel appareil photo principal 48 Mpx.
 La photo au sommet de son art.



iphone 15

Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :



L'image est décomposée en *Pixels* (*Picture elements*).
 Un pixel est la plus petite partie de l'image et n'est pas divisible
Combien de pixel dans une image aujourd'hui ?



Nouvel appareil photo principal 48 Mpx.
 La photo au sommet de son art.

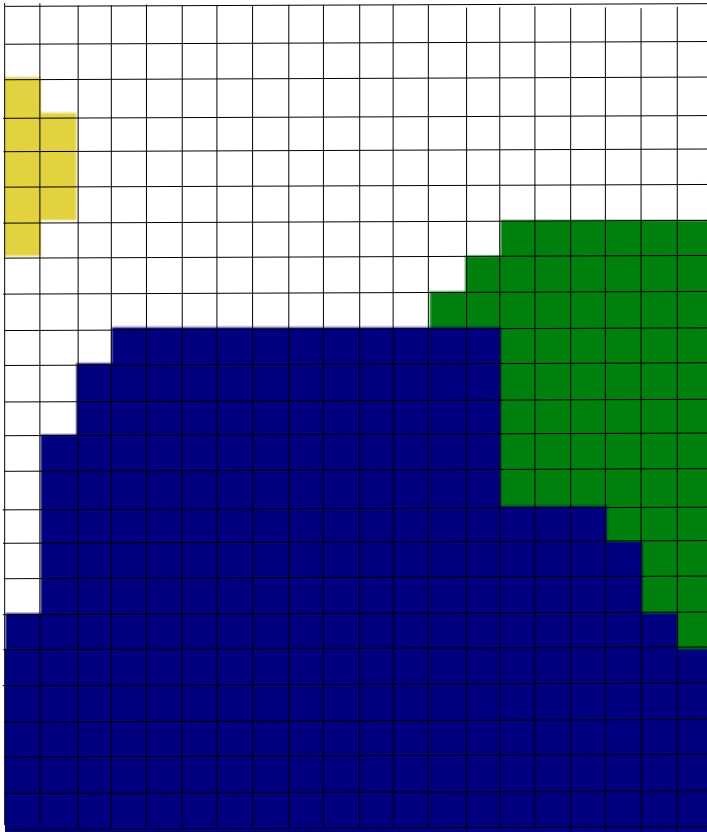


iphone 15

C'est le nombre d'informations stockées dans la mémoire par la caméra... Un truc d'informaticien en somme...

Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :



L'image est décomposée en *Pixels* (*Picture elements*).

Un pixel est la plus petite partie de l'image et n'est pas divisible

Combien de pixel dans une image aujourd'hui ?

DOUBLE OBJECTIF DE 50 MP

C'est le nombre d'informations stockées dans la mémoire par la caméra. Ce sont des **valeurs brutes** qui spécifient la couleur de chaque Pixel et qui seront ensuite manipulée (retouchée) par un programme pour permettre le rendu le meilleur possible sans pour autant prendre trop de place sur le disque dur.

FAIRPHONE 5 <https://shop.fairphone.com/fr/>

Encodage des images

- Quelle est la structure de données permettant de stocker les informations d'un image ?
- Exemple : une image numérique :

L'image est décomposée en *Pixels (Picture elements)*.

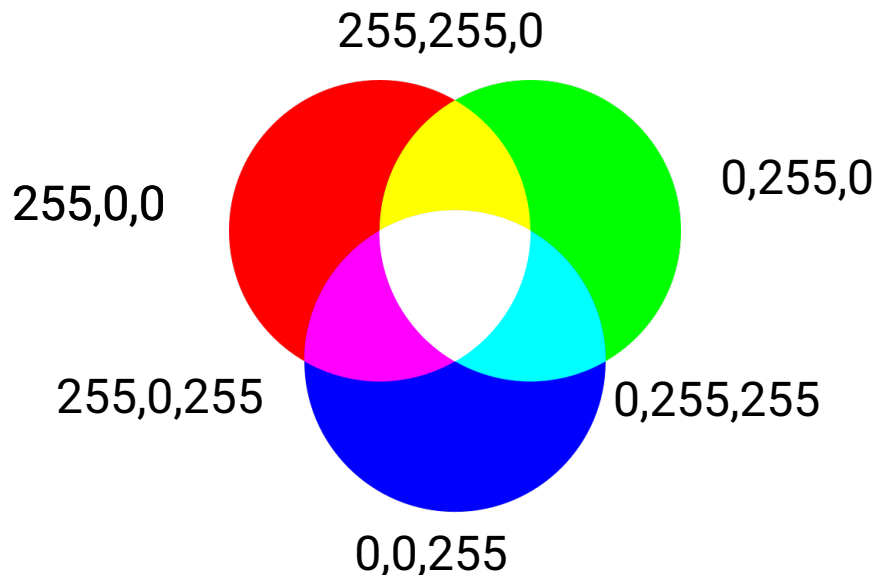
What is a RAW file?

A RAW file is the uncompressed and unprocessed image data captured by a digital camera or scanner's sensors. Shooting in RAW captures a high level of image detail, with large file sizes and lossless quality. The direct image data means you start with a high-quality image that can be edited, converted, and compressed in a non-destructive manner.

RAW files are a type of raster file format, but not actually images themselves. This means you need to import them into relevant software before you can edit or export them as a different raster image file, like a [JPEG](#). Many photographers who shoot in RAW manipulate the original data in software like Photoshop before compressing the RAW file into a different format for print or online.

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.
- Comment la couleur d'un pixel est-elle codée ?



Chaque Pixel a une couleur codée avec 3 valeurs selon la méthode dite additive :

- La quantité de rouge
- La quantité de vert
- La quantité de bleu

$Red, Green, Blue \in [0; 255]$

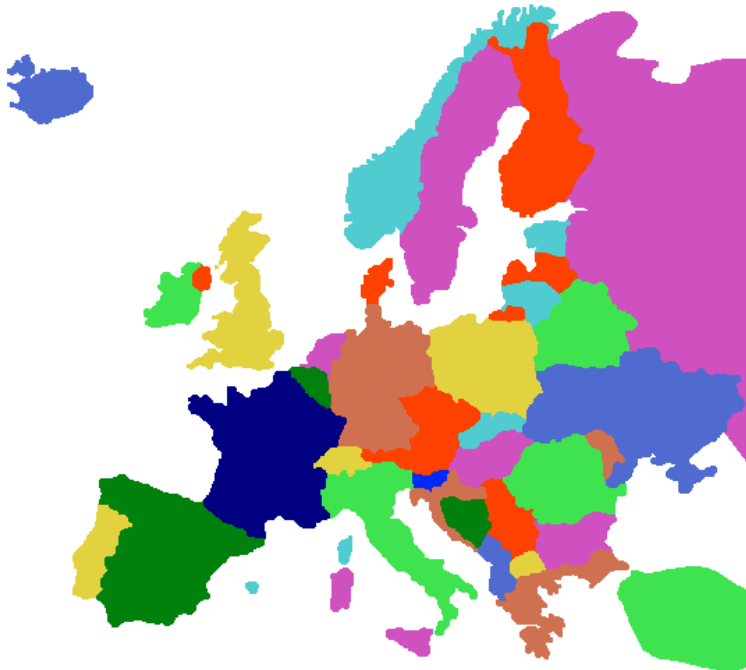
$R, G, B \in \mathbb{N}$

Il existe de nombreux outils pour déterminer la couleur résultante en fonction de ces 3 valeurs :

- <https://imagecolorpicker.com/>
- https://www.rapidtables.com/web/color/RGB_Color.html
- KColorChooser sur linux...

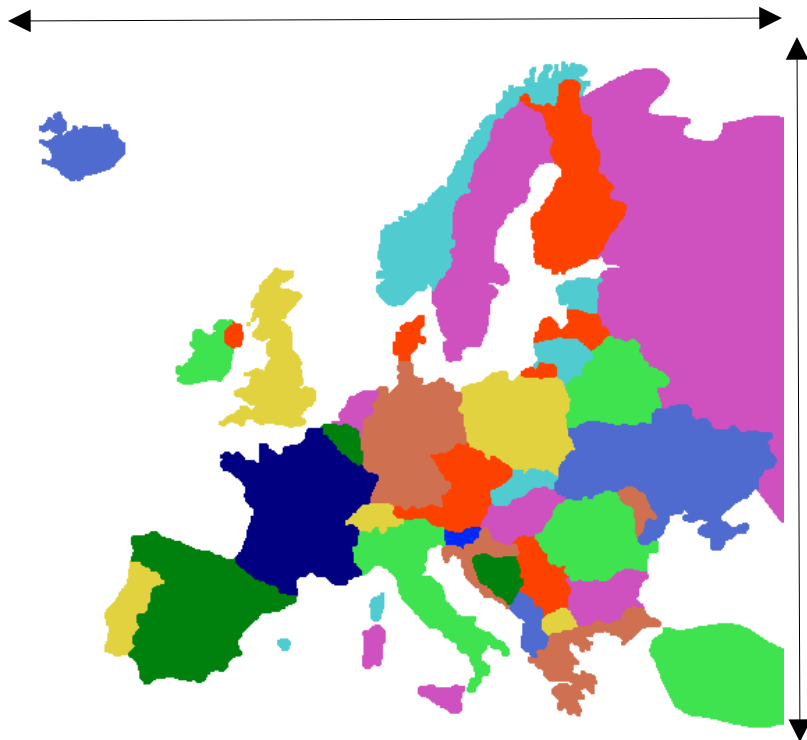
Encodage des images

- 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.



Encodage des images

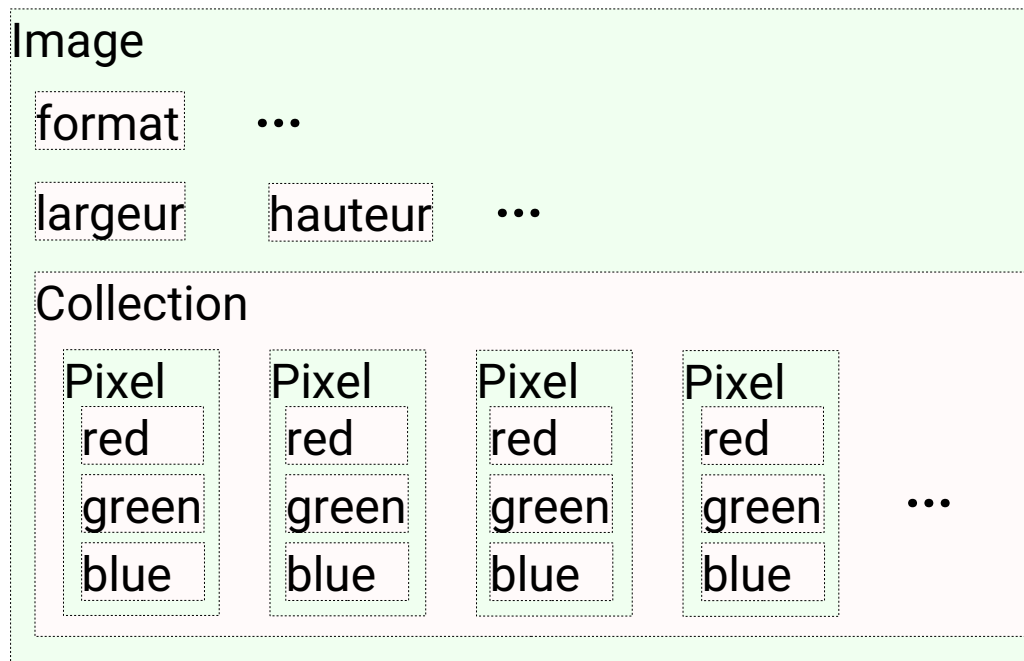
- Quelle est la structure de données permettant de stocker les informations d'un image ?
 - une image numérique brute est un ensemble de pixel où chaque pixel a une couleur particulière, encodée par exemple en RGB.



En nombre de pixels...

Encodage des images

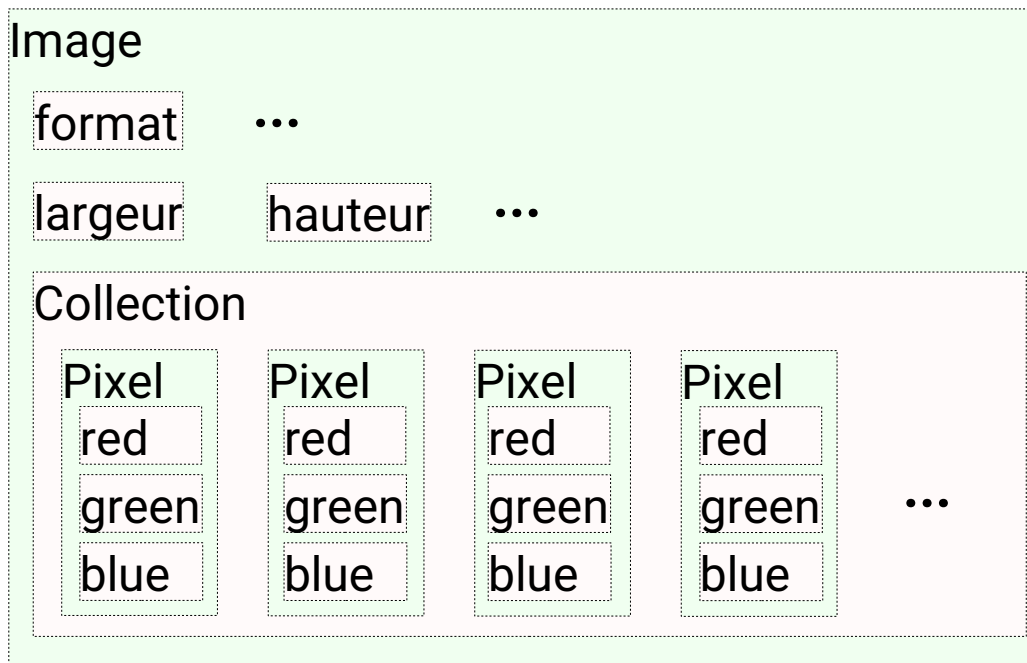
- Quelle est la structure de données permettant de stocker les informations d'un image ?
 - une image numérique brute est un ensemble de pixel où chaque pixel a une couleur particulière, encodée par exemple en RGB. De plus la largeur et la hauteur est donnée.



Premier jet

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

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

@dataclass
class Image:
    format:str
    length:int
    width:int
    pixels:list[Pixel]
```