Introduction à la programmation 1

4 – Résumé et fonction

Julien Deantoni

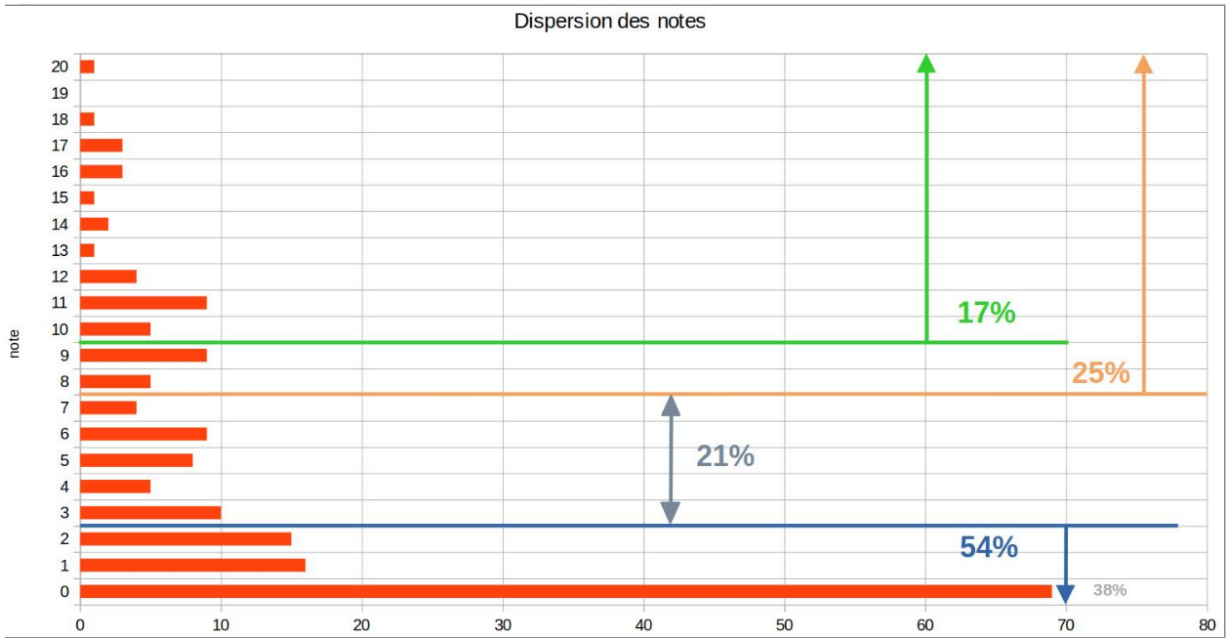
Contenue de l'UE (non contractuel)

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

Attention

- **Taux de réussite au bac : 91,1%**
- **Taux de réussite d'une UE en L1 : environ 20%**

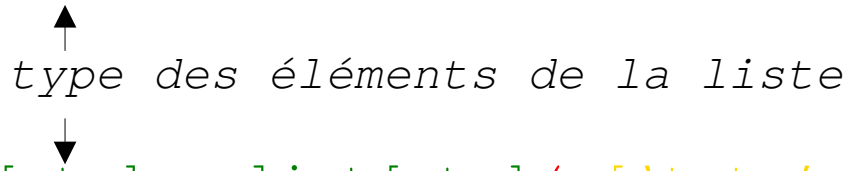
⇒ vous devez travailler en dehors des séances !



Résultat du contrôle numero 1 de 2022

Créer une variable

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



  - `liste2: list[str] = list[str]( ['toto','titi'] )`
  - `i3: int = i1 + i2 #attention`

Référence ou valeur référencée ?

- À gauche du signe d'affectation (=), on parle de l'étiquette elle même, c'est à dire de la référence. Ailleurs, on parle de la valeur référencée.
  - `i1: int = int(0)`
  - `i2: int = int('123')`
  - `i3: int = i1 + i2`

L'étiquette (ou référence) `i3`

La valeur référencée par l'étiquette `i1`

- `i2 = i3`

L'étiquette `i2`

La valeur référencée par l'étiquette `i3`

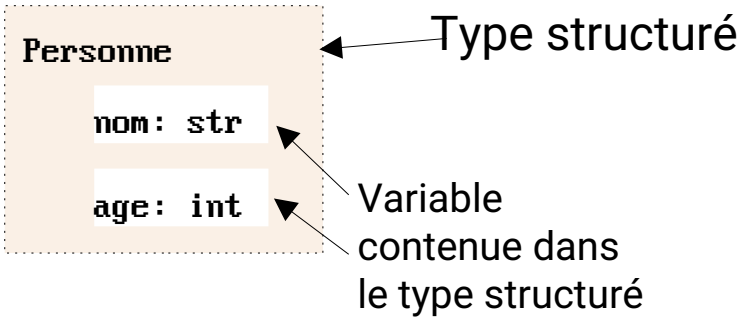
`print(i2, i3)`

La valeur référencée par l'étiquette `i2`

La valeur référencée par l'étiquette `i3`

Créer une variable d'un type structuré

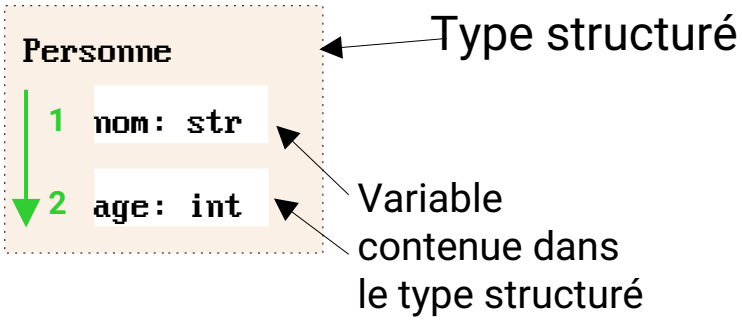
- NomVariable **:** type **=** type**(** valeurInitiale **)**




```
p1: Personne = Personne('titi',19)
p2: Personne = Personne('zaza',18)
```

Créer une variable d'un type structuré

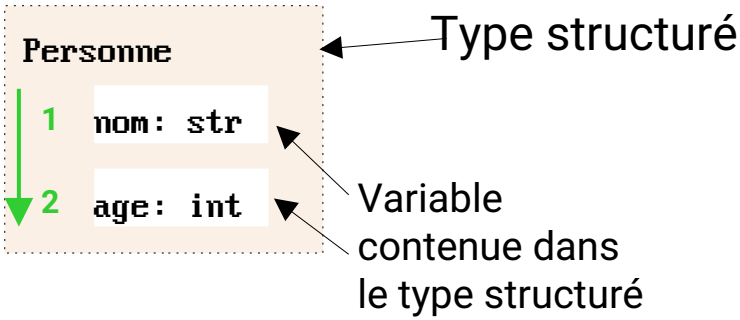
- NomVariable **:** type **=** type**(** valeurInitiale **)**

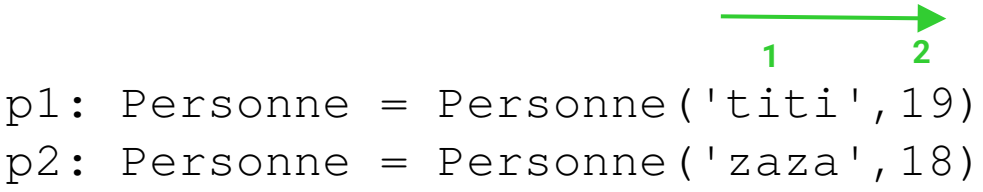



```
p1: Personne = Personne('titi',19)
p2: Personne = Personne('zaza',18)
```

Créer une variable d'un type structuré

- NomVariable **:** type **=** type**(** valeurInitiale **)**





- Accès aux variables internes des variables structurées

p1.nom

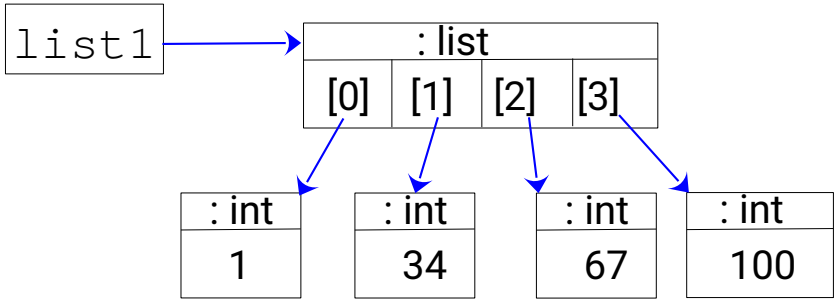
p1.age == 18 ?

p2.nom

p2 .nom == p1 .nom ?

Accès aux éléments d'une liste

```
list1: list = list([1,34,67,100])
```



- On utilise la position de l'élément dans la liste. On parle de l'indice de l'élément dans la liste.

```
i1 : int = int( list1[0] + 2)
i2 : int = int(list[i1])
i : int = int(0)
while (i < 4) :
        print(list1[i])
        i = i + 1
```

Mutable versus immutable data types

```
1  i: int = int(0)
2  j: int = i
3
4  i = i +1
5  print('j == i -->', j==i)
6
7  p1: Pixel = Pixel(0,0,0)
8  p2: Pixel = p1
9
10 p1.r = 255
11 print('p1 == p2 -->', p1 == p2)
```

???

???

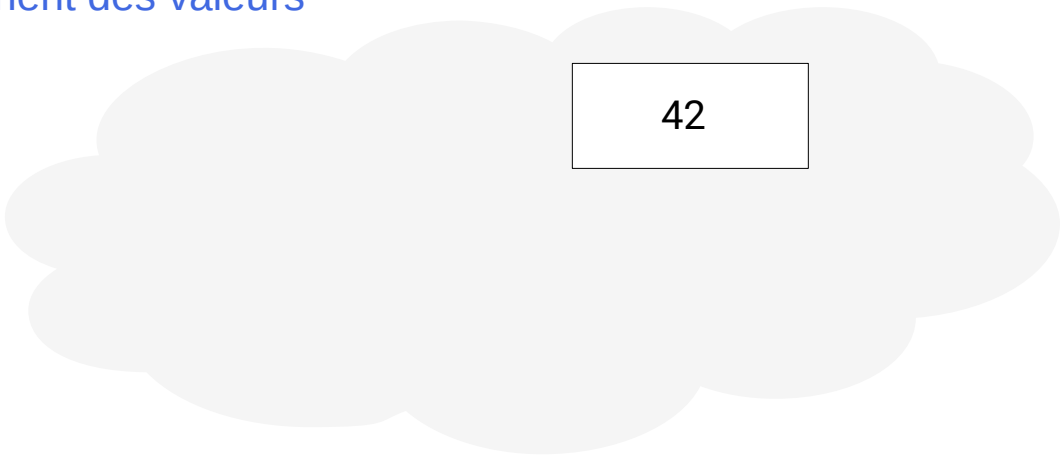
Mutable versus Immutable data types

- 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, tous les types structurés
    - 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
```

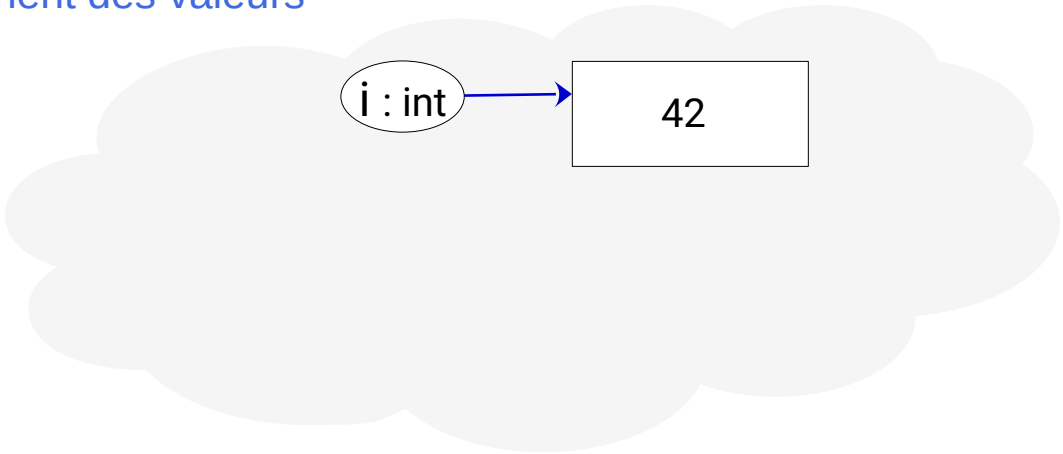


- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

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
```

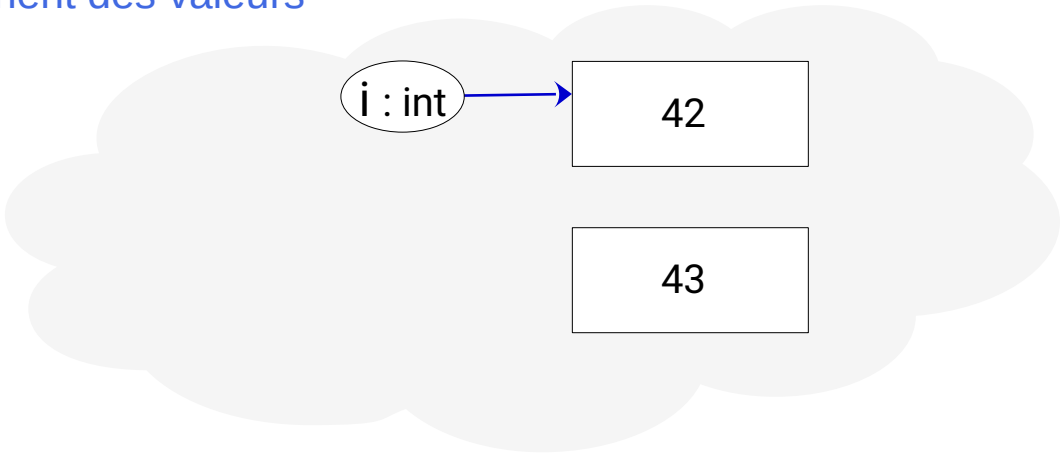


- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

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
```

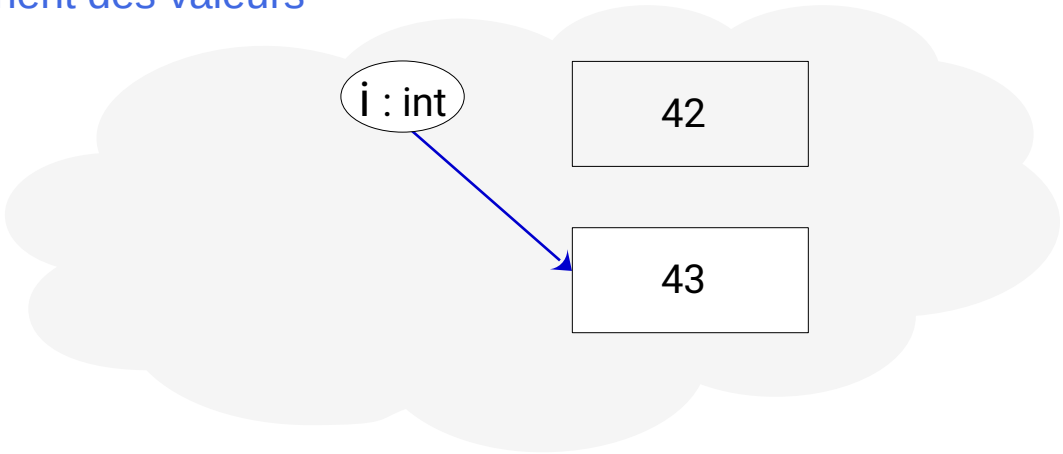


- Mutable : list, Point2D, etc
  - Les cases mémoires associées aux variables de ces types peuvent être modifiées car elles contiennent des étiquettes

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

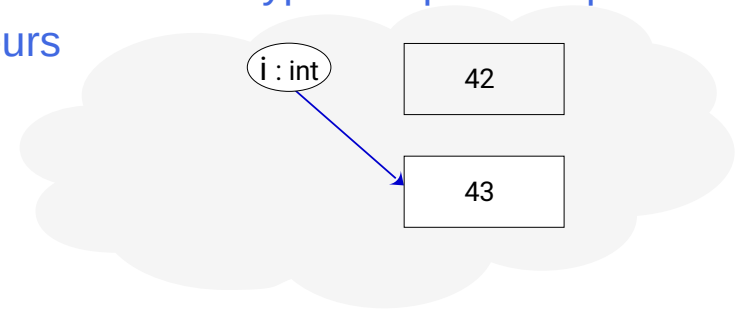
Mutable versus Immutable data types

- 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

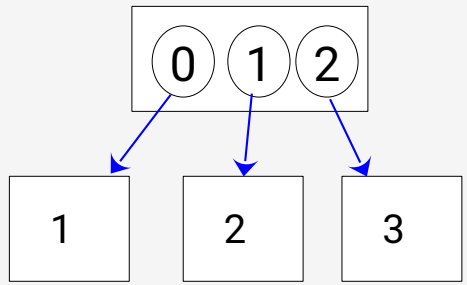
```
1  i: int = int(42)
2
3  i = i + 1
```



- 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
```



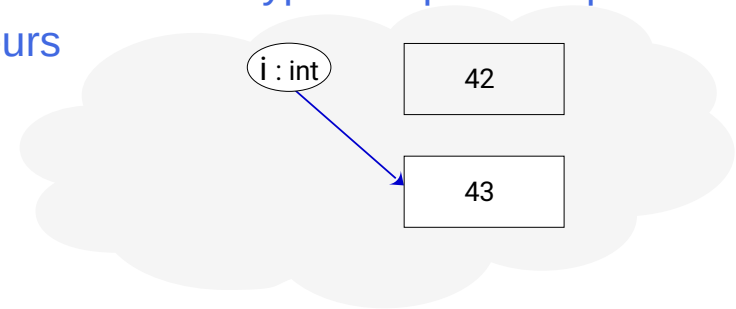
Mutable versus Immutable data types

- 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

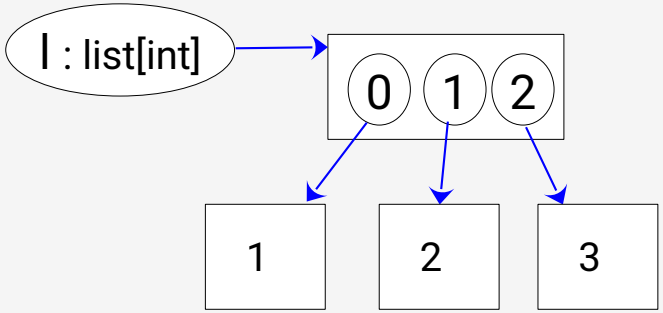
```
1 i: int = int(42)
2
3 i = i + 1
```



- 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
```



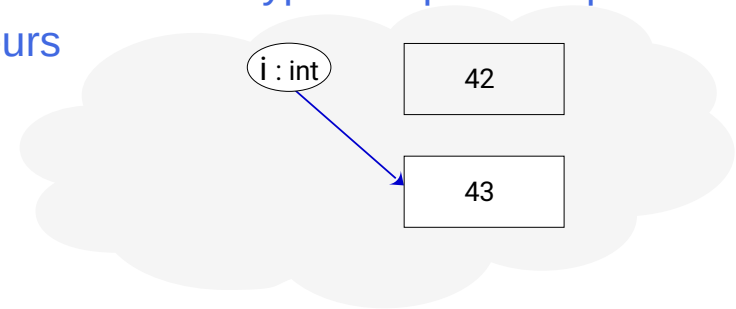
Mutable versus Immutable data types

- 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
```

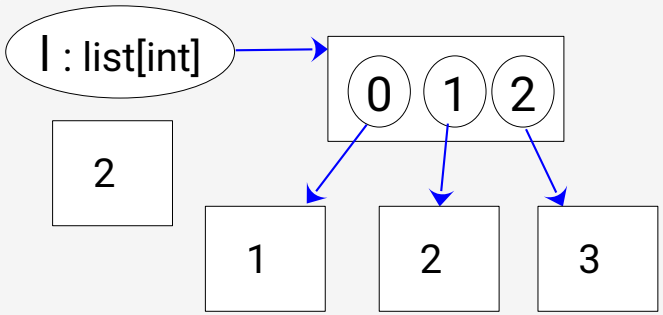


- Mutable : list, Point2D, etc

  - 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'



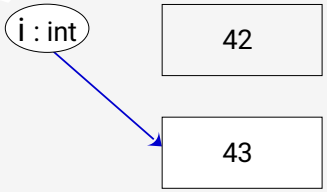
Mutable versus Immutable data types

- 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
```

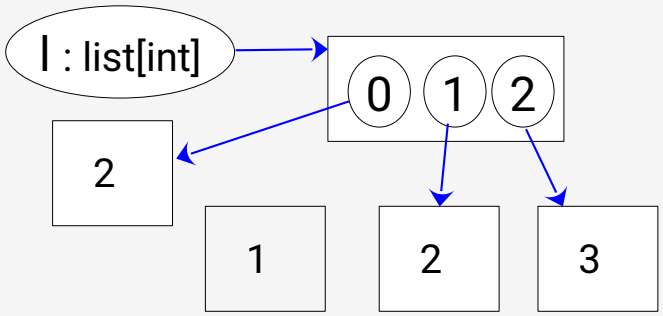


- Mutable : list, Point2D, etc

  - 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'



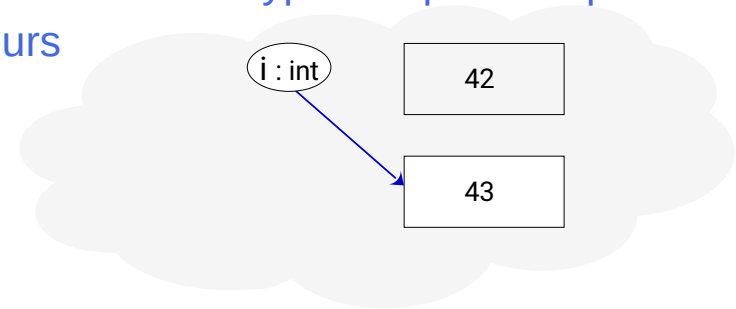
Mutable versus Immutable data types

- 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
```

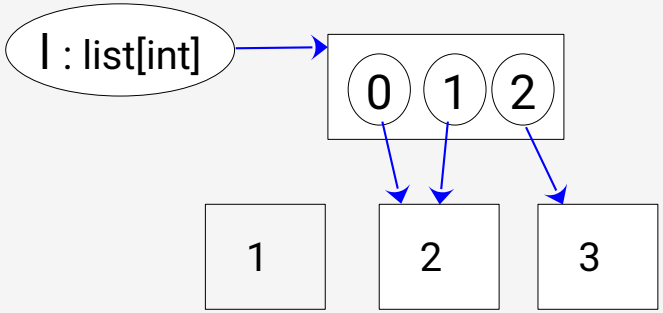


- Mutable : list, Point2D, etc

  - 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
```

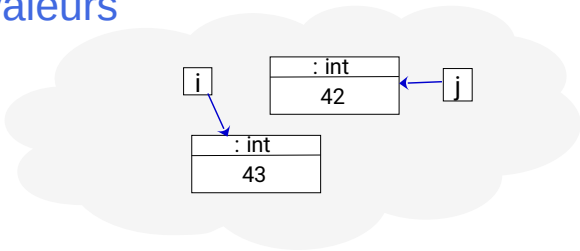
L'étiquette '0' de 'l'



Mutable versus Immutable data types

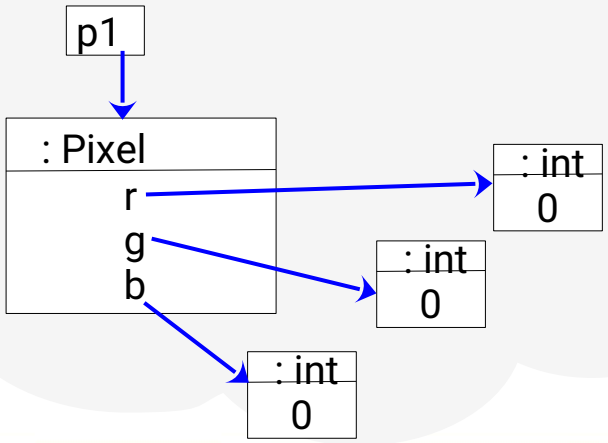
- Python, comme tout langage informatique définit deux sortes différentes de type de données : les mutable et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

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



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

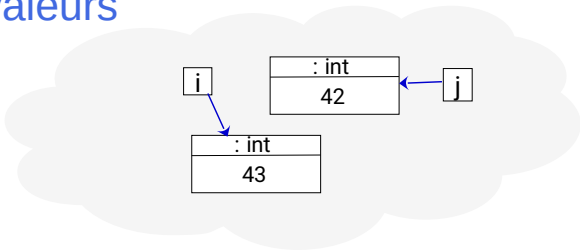
```
7  p1: Pixel = Pixel(0,0,0)
8  p2: Pixel = p1
9
10 p1.r = 255
```



Mutable versus Immutable data types

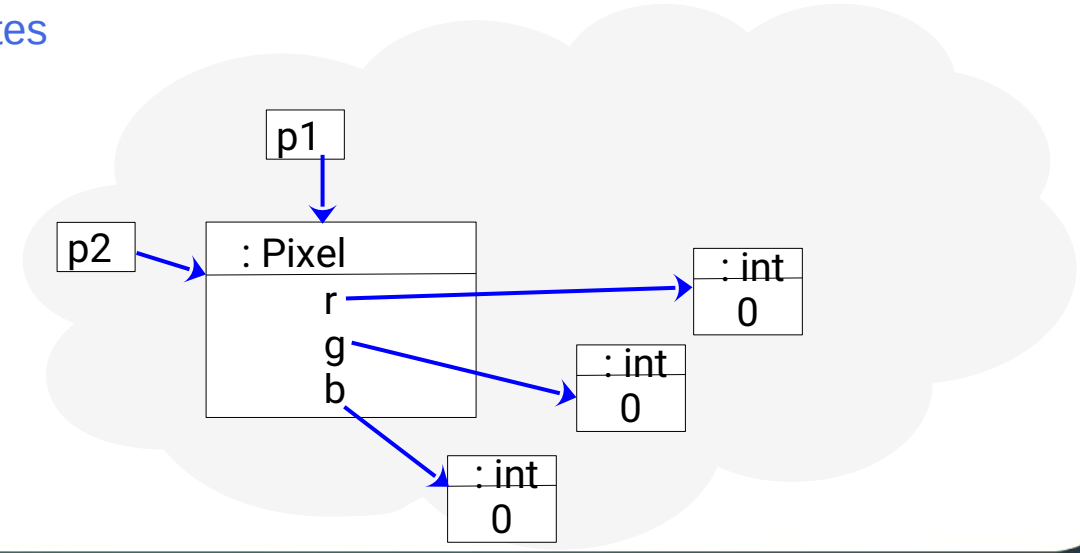
- Python, comme tout langage informatique définit deux sortes différentes de type de données : les mutable et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

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



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

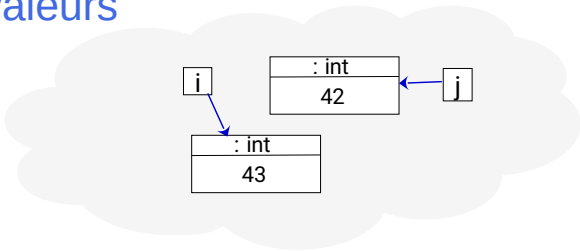
```
7  p1: Pixel = Pixel(0,0,0)
8  p2: Pixel = p1
9
10 p1.r = 255
```



Mutable versus Immutable data types

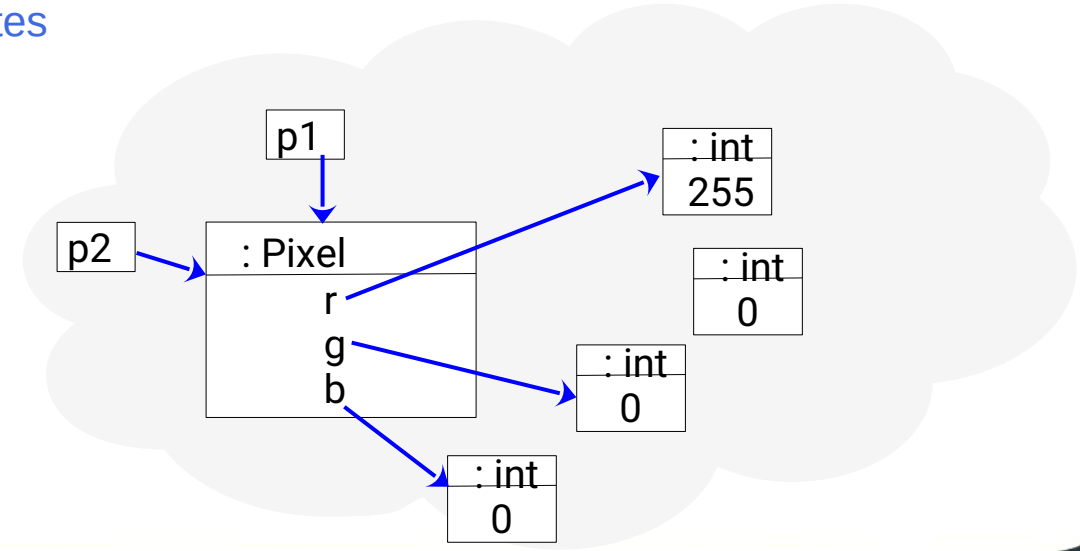
- Python, comme tout langage informatique définit deux sortes différentes de type de données : les mutable et les immutables.
- Immutable : int, str, float, bool
  - Les cases mémoires associées aux variables de ces types ne peuvent pas être modifiées car elles contiennent des valeurs

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



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

```
7  p1: Pixel = Pixel(0,0,0)
8  p2: Pixel = p1
9
10 p1.r = 255
```



Mutable versus immutable data types

```
1  i: int = int(0)
2  j: int = i
3
4  i = i +1
5  print('j == i -->', j==i)
6
7  p1: Pixel = Pixel(0,0,0)
8  p2: Pixel = p1
9
10 p1.r = 255
11 print('p1 == p2 -->', p1 == p2)
```

False

True

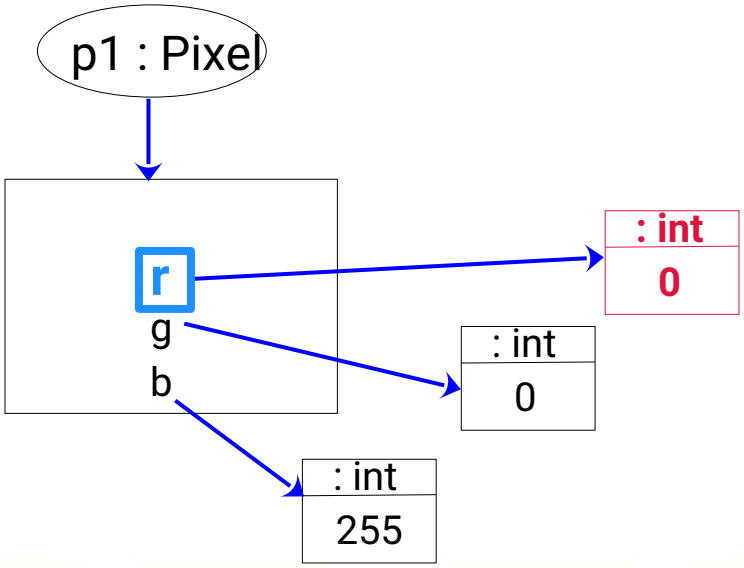
Étiquette (référence) et valeur/objet référencé

- À gauche du signe d'affectation (=), on parle de l'étiquette elle même, c'est à dire de la référence. Ailleurs, on parle de la valeur référencée.

```
1  p1: Pixel = Pixel(0,0,255)
2  p1.r = p1.r+1
```

L'étiquette (ou référence) `r` dans `p1`

La valeur référencée par l'étiquette r dans p1



Premiers algorithmes

Premiers algorithmes : compréhension

- 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

```
1 p1: Pixel = Pixel(0,0,255)
2 p2: Pixel = Pixel(255,255,255)
3 p3: Pixel = Pixel(r=255,g=0,b=0)
4
5 allPixels: list[Pixel] = list[Pixel]([p1, p2, p3])
```

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

Plus généralement le début du TD2 doit être clair pour vous.

```
1 index: int = int(0)
2 while (index < len(allPixels)):
3         p: Pixel = allPixels[index]
4         p = Pixel(0,0,0)
5         index = index + 1
```

Premiers algorithmes : écriture

- 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

• recopie d'une liste de pixels

Soit une liste de pixels nommée `inputPixels`. Écrire un algorithme permettant de créer une nouvelle liste nommée `outputPixels` contenant des pixels ayant la même valeur que ceux de la liste `inputPixels`. Cependant assurer vous que la modification des éléments de `inputPixels` n'impacte pas la liste `outputPixels`

Premiers algorithmes : écriture

- 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

• recopie d'une liste de pixels

Soit une liste de pixels nommée `inputPixels`. Écrire un algorithme permettant de créer une nouvelle liste nommée `outputPixels` contenant des pixels ayant la même valeur que ceux de la liste `inputPixels`. Cependant assurer vous que la modification des éléments de `inputPixels` n'impacte pas la liste `outputPixels`

```
1  #piege !
2  index: int = int(0)
3  while (index < len(inputPixels)):
4      pin: Pixel = inputPixels[index]
5      outputPixels = outputPixels + [pin]
6      index = index + 1
```

⇒ ici les étiquettes de `outputPixels` référencent les même objets que celle de `inputPixels`

Premiers algorithmes : écriture

- 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

• recopie d'une liste de pixels

Soit une liste de pixels nommée `inputPixels`. Écrire un algorithme permettant de créer une nouvelle liste nommée `outputPixels` contenant des pixels ayant la même valeur que ceux de la liste `inputPixels`. Cependant assurer vous que la modification des éléments de `inputPixels` n'impacte pas la liste `outputPixels`

```
8  index: int = int(0)
9  while (index < len(inputPixels)):
10     pin: Pixel = inputPixels[index]
11     pout: Pixel= Pixel(pin.r,pin.g,pin.b)
12     outputPixels = outputPixels + [pout]
13     index = index + 1
```

Création d'un nouveau pixel

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer **tous les pixels de `img`** qui sont égaux à `pRef` par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7
8
9
10
11
12      index = index + 1
```

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer **tous les pixels de `img`** qui sont égaux à `pRef` par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7      p: Pixel = img.pixels[index]
8
9
10
11
12     index = index + 1
```

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer tous **les pixels de `img` qui sont égaux à `pRef`** par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7      p: Pixel = img.pixels[index]
8      if(p == pRef):
9
10
11
12     index = index + 1
```

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer tous les pixels de `img` qui sont égaux à `pRef` par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7      p: Pixel = img.pixels[index]
8      if(p == pRef):
9          p.r = pNew.r
10         p.g = pNew.g
11         p.b = pNew.b
12     index = index + 1
```

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer tous les pixels de `img` qui sont égaux à `pRef` par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7      p: Pixel = img.pixels[index]
8      if(p == pRef):
9          p = Pixel(pNew.r, pNew.g, pNew.b)
10     index = index + 1
```

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer tous les pixels de `img` qui sont égaux à `pRef` par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7      p: Pixel = img.pixels[index]
8      if(p == pRef):
9          p = Pixel(pNew.r, pNew.g, pNew.b)
10     index = index + 1
```

Premiers algorithmes : écriture

- 2.4.2 Changement de couleur

  Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment. Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`. Écrire un algorithme permettant changer tous les pixels de `img` qui sont égaux à `pRef` par des pixels égaux à `pNew`.

```
5  index: int = int(0)
6  while (index < len(img.pixels)):
7      p: Pixel = img.pixels[index]
8      if(p == pRef):
9          img.pixels[index] = Pixel(pNew.r, pNew.g, pNew.b)
10     index = index + 1
```

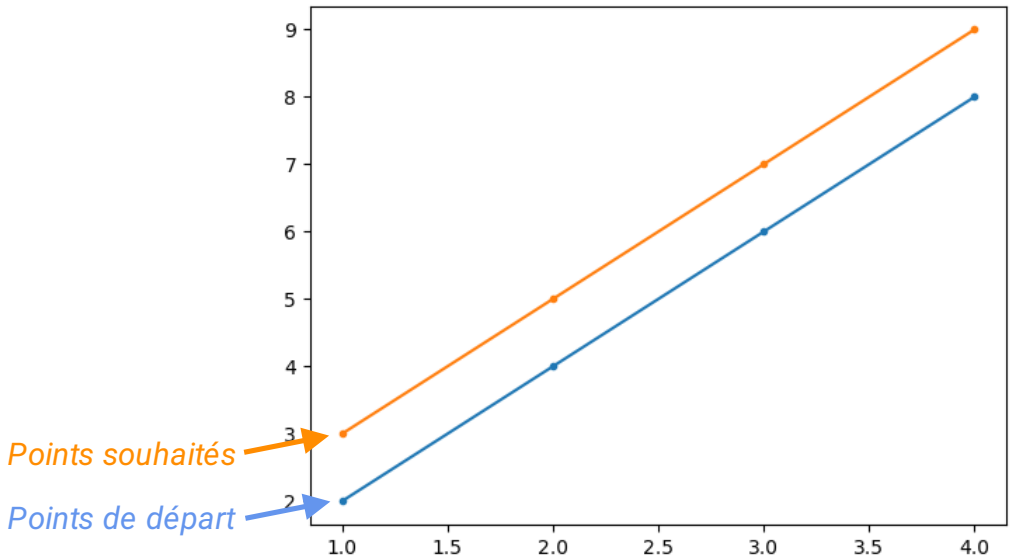
Premiers algorithmes

- 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

```
[(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')
```

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)]

```
[(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')
```

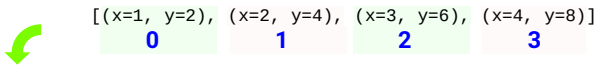
Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]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

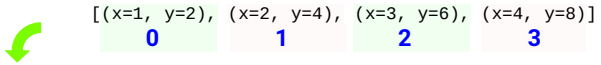
Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]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

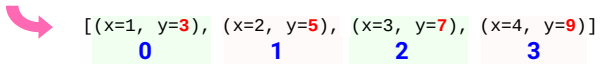
Premiers algorithmes

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')
```

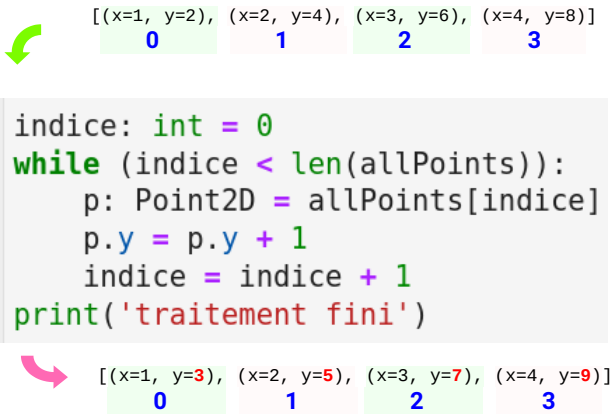


Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]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

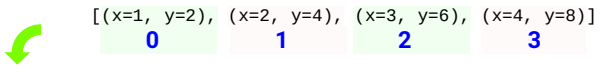


Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]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

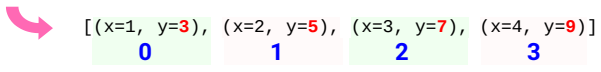
Premiers algorithmes

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')
```

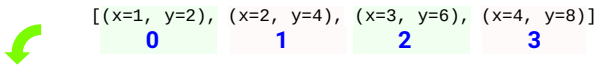


Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]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

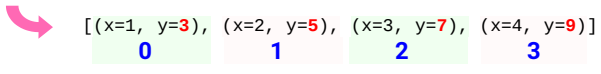
Premiers algorithmes

La boucle While

- On a un ensemble de 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')
```

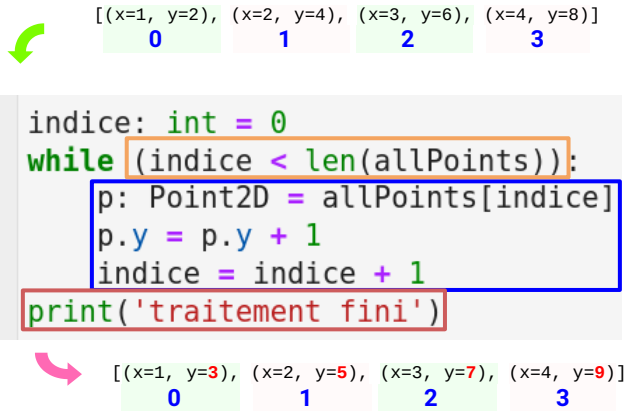


Step #	Line #	i	p	allPoints
1	1	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
2	2	0	undefined	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
3	3	0	(x=1,y=2)	[(x=1,y=2),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
4	4	0	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]
5	5	1	(x=1,y=3)	[(x=1,y=3),(x=2,y=4),(x=3,y=6),(x=4,y=8)]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

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



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

- 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')
```

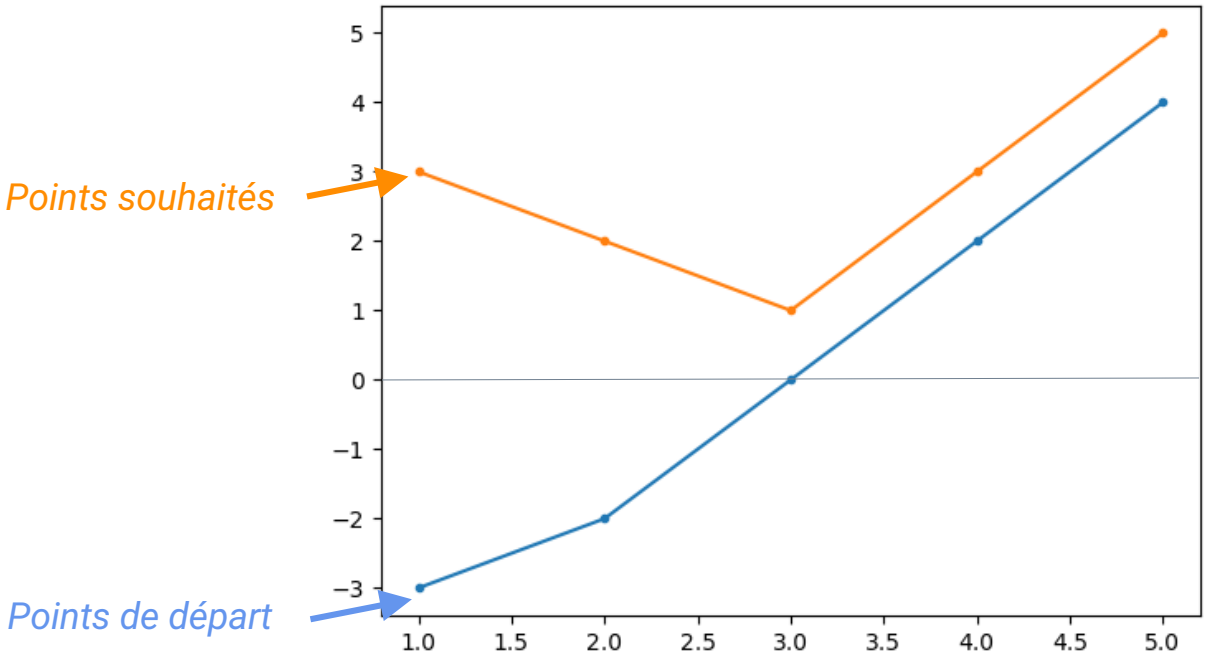
Le ***bloc de code*** identifié par le retrait du bord de la page est le code à exécuter **tant que la condition** est vraie

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

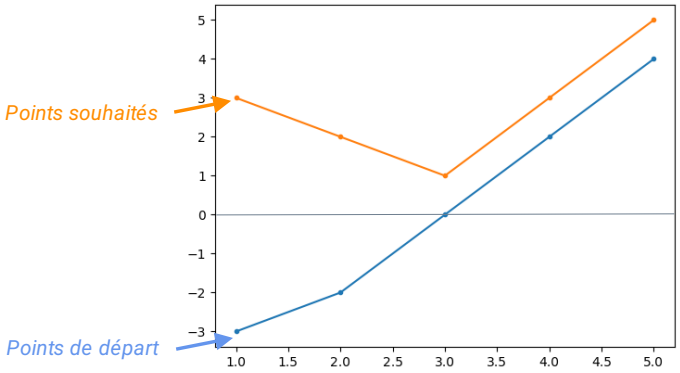


Premiers algorithmes

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)

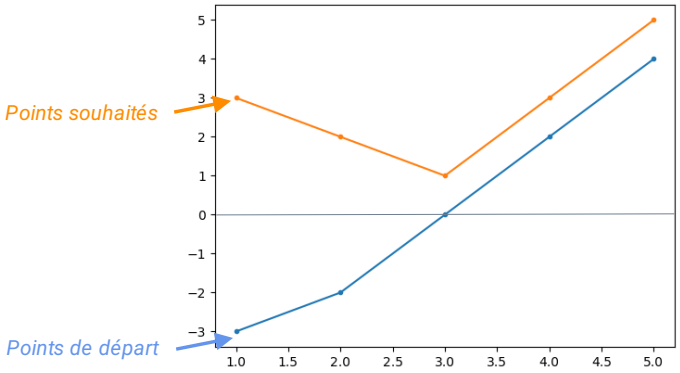
Premiers algorithmes

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.

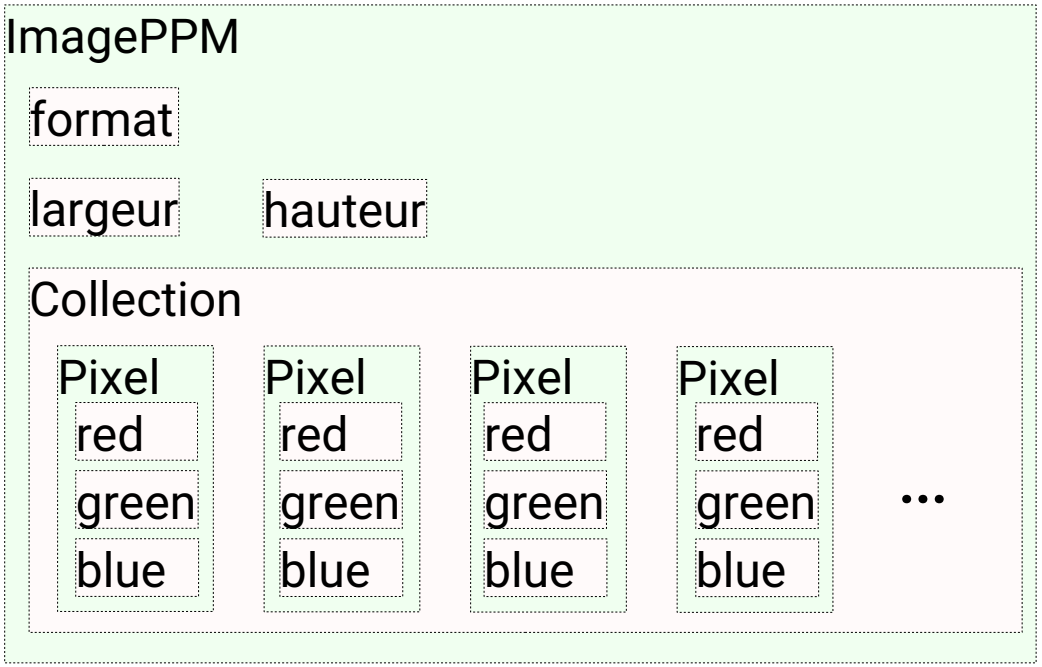
optionnel

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

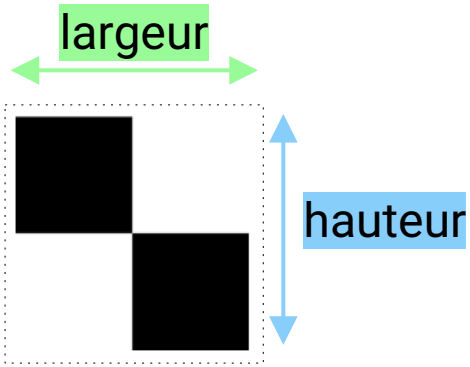


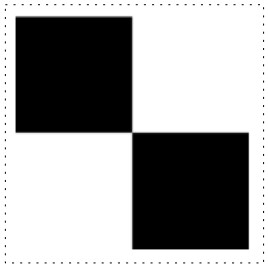
Image miniDamier

Le format PPM P3

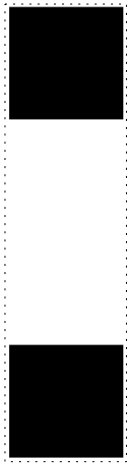
- Nous allons utiliser le format PPM et plus particulièrement son sous format P3 pour les images couleurs. L'avantage est que les images sont enregistrées dans un fichier texte sur le disque dur.

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

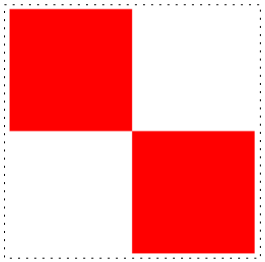
```
P3
2 2 255
0 0 0
255 255 255
255 255 255
0 0 0
```



```
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

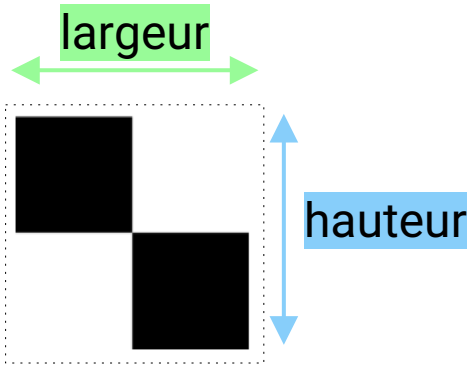


Image miniDamier

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

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

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 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
```

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.heigh)+'\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
```

Fonctions

L'un des concepts les plus importants en programmation est celui de fonction. Les fonctions permettent en effet de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés en fragments plus petits, et ainsi de suite. D'autre part, les fonctions sont réutilisables : si nous disposons d'une fonction capable de calculer une racine carrée, par exemple, nous pouvons l'utiliser un peu partout dans nos programmes sans avoir à la ré-écrire à chaque fois.

https://inforef.be/swi/download/apprendre_python3_5.pdf

Page 69

Fonctions prédéfinis

- Nous avons déjà utilisé des fonctions prédéfinies. Par exemple

  `print :` **`print('Bonjour', name)`**

  ⇒ Elle permet d'afficher les arguments donnés entre parenthèse sur l'écran.

  `len :` **`len(list1)`**

  ⇒ Elle retourne la longueur d'une liste passée en argument.

- Lorsque nous l'utilisons, nous disons que nous **appelons une fonction**. Ces fonctions sont directement accessible dans python

Fonctions importées

- Nous avons déjà utilisé des fonctions importées. Par exemple

```
sqrt : float = sqrt(1)
```

⇒ Elle retourne la racine carrée d'un nombre passé en arguments . Elle est définit dans le module `math` et on doit donc l'importer pour l'utiliser :

```
from math import sqrt
```

```
loadImage : ImagePPM = loadImage('./pict/img.ppm')
```

⇒ Elle permet de désérialiser une image ppm depuis le disque dur et de retourner une variable de type `ImagePPM`. Elle est définit dans le module `iiwHelper` et on doit donc l'importer pour l'utiliser :

```
from iiwHelper import loadImage
```

Fonctions originales

- C'est une fonction dont vous choisirez le comportement et que vous pourrez réutiliser à différent endroit dans votre code :

  La programmation est l'art d'apprendre à un ordinateur comment accomplir des tâches qu'il n'était pas capable de réaliser auparavant. L'une des méthodes les plus intéressantes pour y arriver consiste à ajouter de nouvelles instructions au langage de programmation que vous utilisez, sous la forme de fonctions originales

https://inforef.be/swi/download/apprendre_python3_5.pdf Page 81

- Les fonctions originales permettent de décomposer un problème complexe en sous problèmes plus simples. Chaque fonction traite une partie du problème clairement identifié et pourra elle même être décomposée en sous fonctions.

Définition d'une fonction originale

- ```
  def nomDeLaFonction(paramètre1, paramètre2, ...):
      #Bloc d'instruction
  ```

- Définition d'une fonction sans paramètre

```
def table7():
    n : int = int(1)
    while (n <11) :
        print(n * 7)
        n = n + 1
```
```

Définition d'une fonction originale

```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

- Définition d'une fonction sans paramètre :

```
def table7():
    n : int = int(1)
    while (n <11) :
        print(n * 7)
        n = n + 1
```

- Appel de la fonction ci dessus :

```
table7()
```

⇒

```
7
14
21
28
35
42
49
56
63
70
```

Définition d'une fonction originale

- ```
  def nomDeLaFonction(paramètre1, paramètre2, ...):
      #Bloc d'instruction
  ```

- Définition d'une fonction avec paramètre :

```
def table(base: int):
    n : int = int(1)
    while (n <11) :
        print(n * base, end =' ')
        n = n + 1
```

- Appel de la fonction ci dessus :

```
table(7)
print( )
table(8)
```

```
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
```
```

Définition d'une fonction originale

-
```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

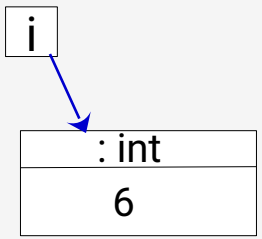
- Définition d'une fonction avec paramètre :

`nomParam1 : type, nomParam2 : type, etc`

```
def table(base: int):
    n : int = int(1)
    while (n <11) :
        print(n * base, end =' ')
        n = n + 1
```

- Appel de la fonction ci dessus :

```
i: int = int(6)
table(i)
```



Définition d'une fonction originale

-
```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

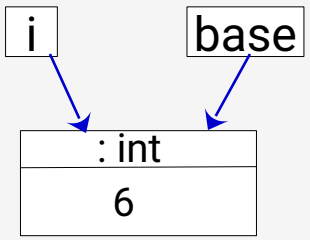
- Définition d'une fonction avec paramètre :

`nomParam1 : type, nomParam2 : type, etc`

```
def table(base: int):
    n : int = int(1)
    while (n <11) :
        print(n * base, end =' ')
        n = n + 1
```

- Appel de la fonction ci dessus :

```
i: int = int(6)
table(i)
```



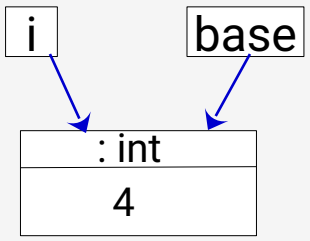
Définition d'une fonction originale

-
```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

- Définition d'une fonction avec paramètre :

```
def xxx(base: int):
    base = base * 2

i: int = int(4)
xxx(i)
print(i)
```



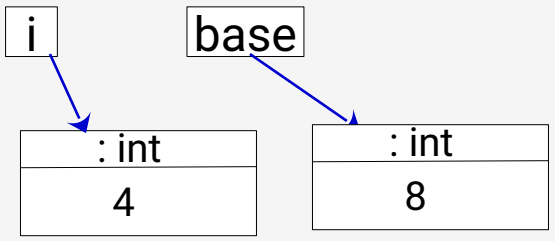
Définition d'une fonction originale

-
```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

- Définition d'une fonction avec paramètre :

```
def xxx(base: int):
    base = base * 2

i: int = int(4)
xxx(i)
print(i)
```



Définition d'une fonction originale

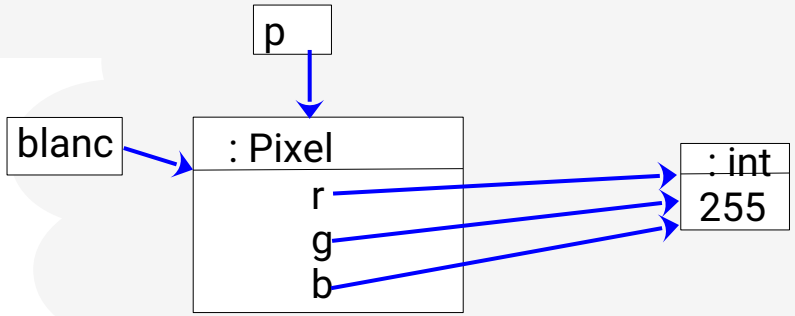
```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

- Définition d'une fonction avec paramètre :

```
from iiwHelper import Pixel

def yyy(p: Pixel):
    p.r = 0

blanc: Pixel = Pixel(255,255,255)
yyy(blanc)
print(blanc)
```



Définition d'une fonction originale

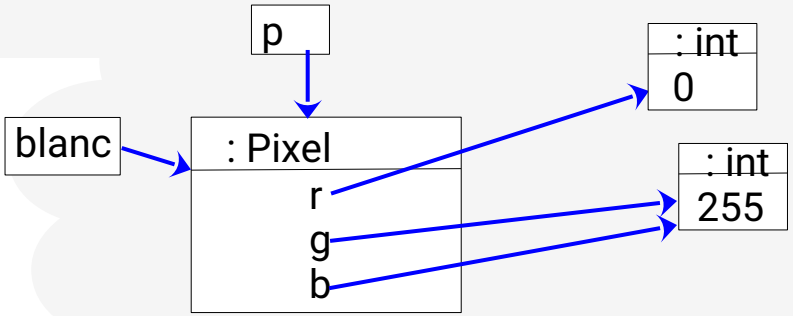
-
```
def nomDeLaFonction(paramètre1, paramètre2, ...):
    #Bloc d'instruction
```

- Définition d'une fonction avec paramètre :

```
from iiwHelper import Pixel

def yyy(p: Pixel):
    p.r = 0

blanc: Pixel = Pixel(255,255,255)
yyy(blanc)
print(blanc)
```



Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec paramètre et type de retour :

```
from iiwHelper import Pixel

def getR(p: Pixel)-> int:
    return p.r

blanc: Pixel = Pixel(255,255,255)
blancR: int = getR(blanc)
```

Le mot clé **`return`** stop le flux d'exécution de la fonction et revient à la ligne d'appel de la fonction.

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec 'code mort' :

```
from iiwHelper import Pixel

def getR(p: Pixel)-> int:
    return p.r
    #code jamais executé
    p.r=127

blanc: Pixel = Pixel(255,255,255)
blancR: int = getR(blanc)
```

Le mot clé **`return`** stop le flux d'exécution de la fonction et revient à la ligne d'appel de la fonction.

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec 'code mort' :

```
from iiwHelper import Pixel

def
    Code is structurally unreachable Pylance
    No quick fixes available
    p.r=127

blanc: Pixel = Pixel(255,255,255)
blancR: int = getR(blanc)
```

Le mot clé **`return`** stop le flux d'exécution de la fonction et revient à la ligne d'appel de la fonction.

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec multiples return:

```
def f(p: Pixel)-> int:
    if(p.r < 42):
        return p.r
    p.r = 127 * p.g // 1200
    return p.g

blanc: Pixel = Pixel(255,255,255)
val: int = f(blanc)
```

Le mot clé **`return`** stop le flux d'exécution de la fonction et revient à la ligne d'appel de la fonction.

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec paramètre et retours multiples:

```
from iiwHelper import Pixel

def getRG(p: Pixel)-> tuple[int,int]:
    return p.r,p.g

blanc: Pixel = Pixel(255,255,255)
blancR: int = 0
blancG: int = 0
blancR,blancG = getRG(blanc)
```

Le mot clé **`return`** stop le flux d'exécution de la fonction et revient à la ligne d'appel de la fonction.

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec type de retour 'None':

```
def xxx(v1:int) -> None:
    v1 = v1 * 2
    print(v1)

xxx(12)
xxx(v1=12)

val = xxx(12) # ????
print(val)
```

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Définition d'une fonction avec type de retour 'None':

```
def xxx(v1:int) -> None:
    v1 = v1 * 2
    print(v1)

xxx(12)
xxx(v1=12)

val = xxx(12) # ????
print(val)
```

⇒

```
24
24
None
```

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Appel d'une fonction avec paramètres par défaut:

```
def xxx(v0:int, v1: int=1) -> None:
    v1 = v1 * v0
    print(v1)

xxx(23) # -> print 23
```

Définition d'une fonction originale

```
def nomDeLaFonction(partam1: type, param2: type, ...) -> typeDeRetour:
    #bloc d'instruction
    return uneVariable
```

- Appel d'une fonction avec paramètres nommés lors de l 'appel:

```
def xxx(v0:int, v1: int=1) -> None:
    v1 = v1 * v0
    print(v1)

xxx(v0=3)
xxx(12,23)
xxx(v0=2,v1=3)
```

```
3
276
6
```

```
def xxx(v0:int, v1: int=1) -> None:
    v1 = v1 * v0
    print(v1)

xxx(v1=12,23)
```

```
  Cell In [35], line 5
    xxx(v1=12,23)
                ^
SyntaxError: positional argument follows keyword argument
```

```
def xxx(v0:int, v1: int=1) -> None:
    v1 = v1 * v0
    print(v1)

xxx(12,v0=23)
```

```
Cell In [36], line 5
      2     v1 = v1 * v0
      3     print(v1)
----> 5 xxx(12,v0=23)

TypeError: xxx() got multiple values for argument 'v0'
```

Définition d'une fonction originale

```
def isInCircle(img:ImagePPM, centerX:int, centerY:int, radius: int, pixelPos:int) → bool:
```

```
image: ImagePPM = loadImage('./pict/valrose.txt')

i: int = int(0)
while (i < len(image.pixels)):
    p: Pixel = image.pixels[i]
    if (isInCircle(image, image.width//2, image.height//2, 100, i)):
        p.g = p.g//3
        p.b = p.g//3
    else:
        p.r = p.r + 100
        p.g = p.g + 100
        p.b = p.b + 100
    i = i + 1
showImage(image)
```