Logique et approche mathématique de la programmation

M. Rigo, septembre 2019

Pourquoi les mathématicien(ne)s doivent être capables de programmer ?

- Traitement / analyse / visualisation de données complexes, statistique / *data science*
- Calcul et approximations numériques (zéros de polynômes, équations différentielles, intégration, …)
- Exploration mathématique : émettre des conjectures
- Méthode de Monte-Carlo, modélisation
- Preuves par ordinateur : passer en revue un grand nombre (fini) de cas, méthodes formelles “certifiées”
- Dans le secondaire, à qui demandera-t-on d’enseigner la programmation, l’algorithmique ?

Pourquoi les mathématicien(ne)s doivent être capables de programmer ?

Si vous pensez aux débouchés :

- Secteur bancaire / assurance
- Consultance stat. / finance
- Enseignement
- Recherche
- Modélisation, logistique, travail dans une équipe…
- Votre capacité d’abstraction est un atout recherché !

Exemple :

Au Yahtzee, quelle est la probabilité de faire un *brelan* en un coup, i.e., au moins 3 dés identiques sur 5 ?

On supposera que les 5 dés sont ordonnés.

Exemple :

Au Yahtzee, quelle est la probabilité de faire un *brelan* en un coup, i.e., au moins 3 dés identiques sur 5 ?

23 / 108 = 0.213

Nombres de cas favorables :

3 dés identiques : 5! / (3! 2!) . 6 . 1 . 1 . 5 . 5
+ 4 dés identiques : 5! / (4! 1!) . 6 . 1 . 1 . 1 . 5
+ 5 dés identiques : 6 . 1 . 1 . 1 . 1 = 1656

Nombres de cas possibles : $6^5 = 7776$

```
def categorie():
    resultat = [ ]
    for i1 in range(1,7):
        for i2 in range(1,7):
            for i3 in range(1,7):
                for i4 in range(1,7):
                    for i5 in range(1,7):
                        resultat.append([i1,i2,i3,i4,i5])
    return resultat

def test_brelan(lance):
    if sorted([ lance.count(1),lance.count(2),
                lance.count(3),lance.count(4),
                lance.count(5),lance.count(6) ])[-1]>=3:
        return True
    else:
        return False
```

```
possible = categorie()

total_brelan = 0

for x in possible:              # compter les brelans
    if test_brelan(x):          # parmi tous les jets possibles
        total_brelan += 1

print(total_brelan)
```

```
1656
```

Pourquoi choisir Python ?

- Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability.
- The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++.
- Python is widely used in bigger organizations because of its multiple programming paradigms.
- It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols.
- Python executes with the help of an interpreter instead of the compiler, which causes it to slow down because compilation and execution help it to work normally.

Pourquoi choisir Python ?

Vous pourrez transposer vos connaissances à d'autres langages :

As you will see, you can start coding now with Python. Besides being awesome, Python should be your first programming language because you will quickly learn how to think like a programmer.

Développer une "pensée algorithmique"

De nombreuses ressources en ligne !

- Tutoriel en français, https://docs.python.org/fr/3/tutorial/
- Exercices pour s’entraîner, https://codingbat.com/python
- http://www.france-ioi.org/algo/chapters.php
- CS for all, https://www.cs.hmc.edu/twiki/bin/view/CSforAll/
- https://www.sololearn.com/
- YouTube : CS Dojo Python for absolute beginners https://www.youtube.com/channel/UCxX9wt5FWQUAAz4UrysqK9A
- MIT OpenCourseWare
- https://www.grahamwheeler.com/posts/python-crash-course.html

Programmer ?

Données → Structures de données & Algorithmes

Ingrédients & Recettes

Rem : pour formaliser la notion → *machine de Turing*

Rem : plusieurs algorithmes peuvent mener à la même solution, mais certains sont plus efficaces que d'autres → notion de *complexité* (en temps / espace)

Les mots clés (réservés) de Python 3

```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Rappels de base et compléments

Indentation indispensable (analyse syntaxique du code)

La “casse” (majuscule/minuscule) est respectée

Commenter votre code pour VOUS et pour les autres,
e.g., # ceci est un commentaire

- Variables
- Chaînes de caractères, listes
- Condition True, False, if, else, elif
- Boucles for, while
- Définir une fonction def

In [1]:
```
a = 1  # on définit des variables
b = 2  # des entiers
c = 'bonjour ' # des chaînes de caractères
d = 'hello'
e = 7.92  # des flottants
```

On ne doit pas spécifier par avance le type d'une variable

In [2]:
```
print(a+b)
```

```
3
```

In [3]:
```
print(c+d)  # l'opérateur + est aussi défini pour des chaînes
```

```
bonjour hello
```

In [4]:
```
print(a+c)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-29620a3af8d7> in <module>
----> 1 print(a+c)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Apprenez à lire les exceptions retournées !

In [5]:
```
print(b*e)  # produit d'un entier et un flottant -> flottant
```

```
15.84
```

In [6]:
```
int(b*e)    # on change son type (partie entière)
```

Out[6]:
```
15
```

In [7]:
```
round(b*e)  # arrondi à l'entier le plus proche
```

Out[7]:
```
16
```

Un peu d'arithmétique

In [13]:
```
print( 7 + 2 )
print( 7 - 2 )
print( 7 * 2 )
print( 7 / 2 )   # division
print( 7 // 2 )  # quotient après division par 2
print( 7 % 3 )   # reste après division par 3 (modulo)
print( 7 ** 3 )  # exponentiation
```

```
9
5
14
3.5
3
1
343
```

In [14]:
```
a = 9
b = 3
print( a**b - b/2)
```

```
727.5
```

Quelques changements de types, valides ou non...

integer

In [5]:
```
var = "17"
print( int(var) + 2 )
```

```
19
```

In [6]:
```
var = "17.33"
print( float(var) + 2 )
```

```
19.33
```

In [7]:
```
var = "17.33"
print( int(var) + 2 )
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-82492b045c2b> in <module>
      1 var = "17.33"
----> 2 print( int(var) + 2 )

ValueError: invalid literal for int() with base 10: '17.33'
```

In [8]:
```
var = "17.33"
print( int(float(var)) + 2 )
```

```
19
```

string

In [5]:
```
a = 7
print('bonjour ' + str(a))
```

```
bonjour 7
```

Variables et affectations

In [24]:
```
a = 9
b = 3
b = a    # on évalue d'abord le membre de droite, qu'on affecte ensuite à b
print(b)
```

```
9
```

In [25]:
```
# comment échanger les contenus de deux variables ?
a = 9
b = 3
aux = a  # on utilise une variable auxiliaire
a = b
b = aux
print('a =', a)
print('b =', b)
```

```
a = 3
b = 9
```

In [26]:
```
# affectations multiples
a, b = 9, 3
print('a =', a)
print('b =', b)
```

```
a = 9
b = 3
```

C’est un point plus délicat

In [27]:
```
# un échange plus "subtil"
a, b = 9, 3
a, b = b, a      # le membre de droite est évalué avant les affectations
print('a =',a)
print('b =',b)
```

```
a = 3
b = 9
```

Incrémenter une variable

In [16]:
```
a = 17
a = a + 2
print(a)
```

19

On évalue toujours
le membre de droite en premier

In [17]:
```
a = 17
a += 2
print(a)
```

19

Raccourci d'écriture
fréquemment utilisé

In [19]:
```
a = 17
a -= 2
print(a)
```

15

In [20]:
```
a = 17
a *= 2
print(a)
```

34

Les chaînes de caractères (string)

In [43]:
```
une_chaine = 'bonjour'

autre_chaine = " et au revoir"

encore_une = """On peut avoir une chaîne sur
plusieurs lignes..."""

print(une_chaine)
print(une_chaine + autre_chaine)
print(encore_une)
```

tient compte du retour à la ligne

+ est une concaténation
“bon” + “jour” = “bonjour”

```
bonjour
bonjour et au revoir
On peut avoir une chaîne sur
plusieurs lignes...
```

In [44]:
```
print(une_chaine[0])     # les caractères sont indexés à partir de 0
print(une_chaine[2])
print(une_chaine[-1])    # on peut démarrer de la fin de la chaîne
print(une_chaine[-2])
print(une_chaine[0:2])   # un intervalle d'indices [0,2[
print(une_chaine[2:5])
print(une_chaine[0:6:2]) # un intervalle avec un pas de deux : 0, 2, 4
```

On peut renvoyer une “sous-chaîne”

```
b
n
r
u
bo
njo
bno
```

On prend le premier indice mais pas le dernier

Des sélections plus complexes

In [13]:
```
chaine = "0123456789"
print( chaine[:5] )    # sélection depuis le début
print( chaine[5:] )    # sélection jusqu'à la fin de la chaîne
print( chaine[:5:2] )  # avec un pas de 2
print( chaine[5::2] )  # avec un pas de 2

print( chaine[8:2:-1] ) # on peut aussi avoir un pas négatif
print( chaine[:2:-1] )  # depuis la fin du mot
print( chaine[8::-1] )  # jusqu'au début du mot
print( chaine[::-1] )   # moyen simple de prendre le miroir
```

```
01234
56789
024
579
876543
9876543
876543210
9876543210
```

In [24]:
```
x = "bon"
print( x + x + x )
```

```
bonbonbon
```

Cool ! Multiplier des chaînes...

In [26]:
```
print( 10 * x )
```

```
bonbonbonbonbonbonbonbonbonbon
```

Il y a plus de soixantes fonctions/méthodes applicables aux chaînes...

```
In [3]: chaine = "aZertYuiOp"
        chaine[::2][2:4]
Out[3]: 'tu'

In [4]: chaine.upper()
Out[4]: 'AZERTYUIOP'

In [5]: chaine.lower()
Out[5]: 'azertyuiop'

In [6]: chaine.endswith("op")
Out[6]: False

In [7]: chaine.startswith("aZ")
Out[7]: True

In [8]: chaine.index("ert")
Out[8]: 2

In [9]: chaine.isalpha()
Out[9]: True
```

On peut combiner les sélections

Notation standard pour appliquer
une “méthode” à une instance d’un objet.
La fonction renvoie ici une nouvelle chaîne.

swapcase, isdecimal, isdigit, islower, title, replace, …

Les chaînes sont des objets “*immuables*”

Objet dont la valeur ne change pas. Les nombres, les chaînes et les n-uplets sont immuables. Ils ne peuvent être modifiés. Un nouvel objet doit être créé si une valeur différente doit être stockée.

```
In [15]: chaine = "0123456789"
         chaine[3] = 'b'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-4fee4b490f97> in <module>
----> 1 chaine[3] = 'b'

TypeError: 'str' object does not support item assignment
```

Du tutoriel Python : Pourquoi les chaînes de caractères sont-elles immuables ?

La première concerne la performance : savoir qu'une chaîne de caractères est immuable signifie que l'allocation mémoire allouée lors de la création de cette chaîne est fixe et figée.

Un autre avantage est que les chaînes en Python sont considérées aussi "élémentaires" que les nombres. Aucun processus ne changera la valeur du nombre 8 en autre chose, et en Python, aucun processus changera la chaîne de caractère "huit" en autre chose.

```
In [15]: chaine = "0123456789"
         chaine[3] = 'b'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-4fee4b490f97> in <module>
----> 1 chaine[3] = 'b'

TypeError: 'str' object does not support item assignment
```

```
In [27]: nouvelle_chaine = chaine[0:3] + 'b' + chaine[4:]
Out[27]: '012b456789'
```

Avec le bon choix d’intervalles, on s’en sort

Les listes

Tout comme les chaînes de caractères, les listes sont des objets de type “séquentiel” (on y trouve aussi les *tuples*, *range* et les *séquences binaires*). On pourra donc leur appliquer toute une série d’opérations génériques aux séquences.

Une liste est une suite ordonnée d’objets (dont le type peut varier) ;
on peut en modifier les éléments

Emploi de crochets […]

```
In [1]: une_liste = [1, 'a', 3, 17]

In [2]: print( une_liste[2] )
3
```

Les listes sont “muables” !

```
In [3]: une_liste[2] = 'tada'
        print( une_liste )
[1, 'a', 'tada', 17]
```

```
In [4]: b = 42
        une_liste[0] = b
        b = -1
        print( une_liste )
[42, 'a', 'tada', 17]
```

n’a pas d’effet sur la liste,
l’affectation a déjà été effectuée !

In [7]:
```
matrice = [[1,2],[3,4]]

print( matrice[0][1] )
```

```
2
```

In [7]:
```
matrice = [[1,2],[3,4]]

print(matrice[0])
```

```
[1, 2]
```

In [8]:
```
liste = [1, 2, 4, 8, 16, 32, [-1, -2], 64]

print( liste[3:5] )
print( liste[5:] )
print( liste[4::2] )
print( liste + liste )
```

Exactement comme pour les chaînes de caractères, opérations applicables à tout objet “séquentiel”

```
[8, 16]
[32, [-1, -2], 64]
[16, [-1, -2]]
[1, 2, 4, 8, 16, 32, [-1, -2], 64, 1, 2, 4, 8, 16, 32, [-1, -2], 64]
```

opérations génériques aux séquences (issu du tutoriel Python)

Opération	Résultat
`x in s`	`True` si un élément de *s* est égal à *x*, sinon `False`
`x not in s`	`False` si un élément de *s* est égal à *x*, sinon `True`
`s + t`	la concaténation de *s* et *t*
`s * n` or `n * s`	équivalent à ajouter *s* *n* fois à lui même
`s[i]`	i^{e} élément de *s* en commençant par 0
`s[i:j]`	tranche (*slice*) de *s* de *i* à *j*
`s[i:j:k]`	tranche (*slice*) de *s* de *i* à *j* avec un pas de *k*
`len(s)`	longueur de *s*
`min(s)`	plus petit élément de *s*
`max(s)`	plus grand élément de *s*
`s.index(x[, i[, j]])`	indice de la première occurrence de *x* dans *s* (à ou après l'indice *i* et avant indice *j*)
`s.count(x)`	nombre total d'occurrences de *x* dans *s*

paramètres optionnels

Méthodes pour les listes :

del, append, extend, insert, remove, pop, clear, count, sort, reverse, index

In [12]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]

del liste[2]
print(liste)
```

```
[1, 7, 12, 7, 33, 42, 21]
```

In [14]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]

del liste[2:4]
print(liste)
```

```
[1, 7, 7, 33, 42, 21]
```

In [15]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]

liste.pop(2)
print(liste)
```

```
[1, 7, 12, 7, 33, 42, 21]
```

In [19]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]

liste.append(1728)
print(liste)
```

```
[1, 7, 9, 12, 7, 33, 42, 21, 1728]
```

Notation standard pour appliquer
une “méthode” à une instance d’un objet.
Ces méthodes modifient la liste !
Une liste peut être très longue, cela
serait trop coûteux de créer une 2ième liste

In [20]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]

liste.remove(7)
print(liste)
```

Supprime la première occurrence de 7

```
[1, 9, 12, 7, 33, 42, 21]
```

In [22]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]

liste.insert(5, 123)
print(liste)
```

précise la position où on insère l'élément

```
[1, 7, 9, 12, 7, 123, 33, 42, 21]
```

In [23]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]
autre = [8, 8, 8]

liste.extend(autre)
print(liste)
```

```
[1, 7, 9, 12, 7, 33, 42, 21, 8, 8, 8]
```

In [26]:
```
liste.clear()
print(liste)
```

```
[]
```

In [30]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]
n = liste.count(7)
print(n)
print(liste.count(17))
```

```
2
0
```

La méthode renvoie une valeur

NB : la méthode renvoie une valeur, dans Jupyter vous avez accès à de l'aide en ligne

In [6]:
```
liste = [2, 3, 5, 7, 11, 13, 17]

a = liste.pop(3)

print(a)
print(liste)
```

```
7
[2, 3, 5, 11, 13, 17]
```

In [8]:
```
?list.pop
```

```
Docstring:
L.pop([index]) -> item -- remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.
Type:      method_descriptor
```

In [46]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]
n = liste.index(42)
print(n)
```

```
6
```

In [47]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]
liste.reverse()
print(liste)
```

```
[21, 42, 33, 7, 12, 9, 7, 1]
```

In [48]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]
liste.sort()
print(liste)
```

```
[1, 7, 7, 9, 12, 21, 33, 42]
```

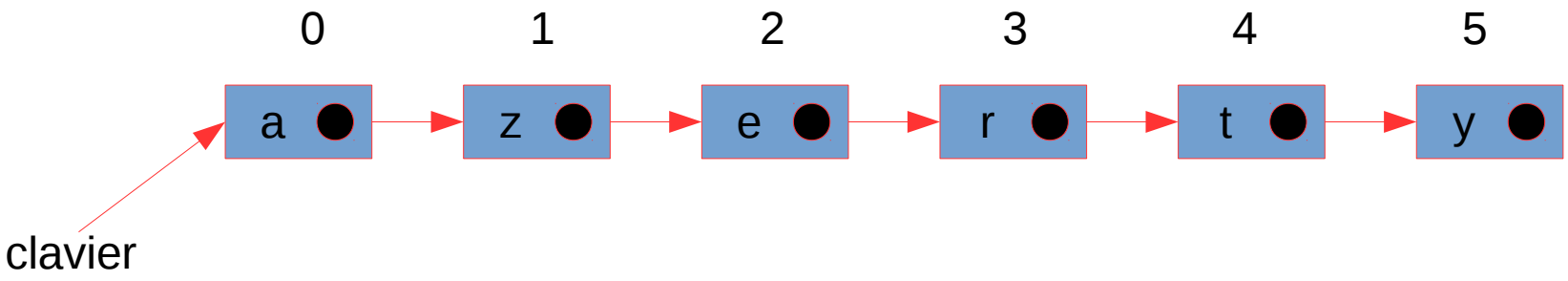
Les méthodes modifient l'objet

In [43]:
```
liste = [1, 7, 9, 12, 7, 33, 42, 21]
liste_triee = sorted(liste)
print(liste)
print(liste_triee)
```

```
[1, 7, 9, 12, 7, 33, 42, 21]
[1, 7, 7, 9, 12, 21, 33, 42]
```

Ici, on n'applique pas une méthode. L'objet n'est pas modifé. La fonction renvoie une nouvelle liste.

Représentation mémoire et alias



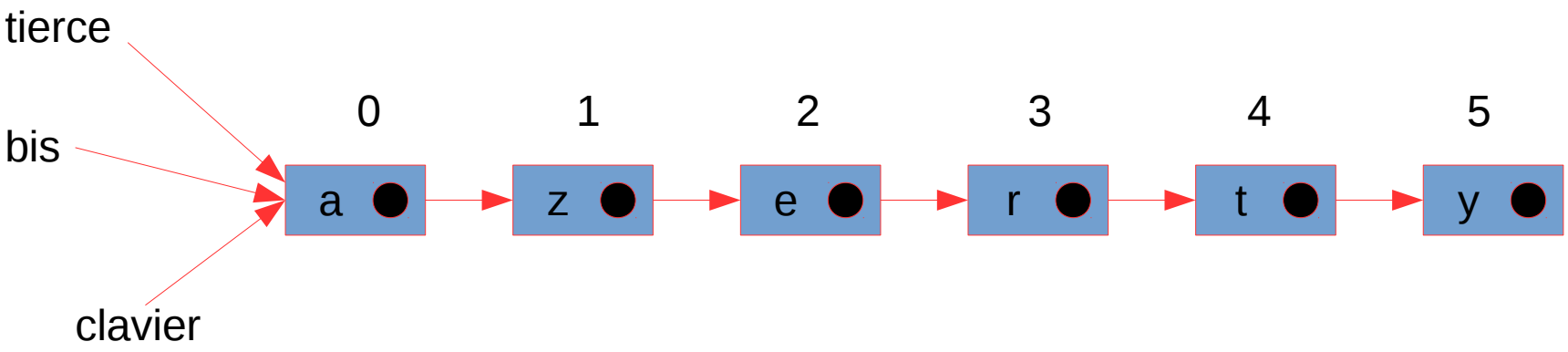

```
In [51]: clavier = ['a', 'z', 'e', 'r', 't', 'y']
```

```
In [52]: bis = clavier
         tierce = clavier
```

```
In [53]: bis[2] = 99
         print( clavier )
```

```
['a', 'z', 99, 'r', 't', 'y']
```

Représentation mémoire et alias



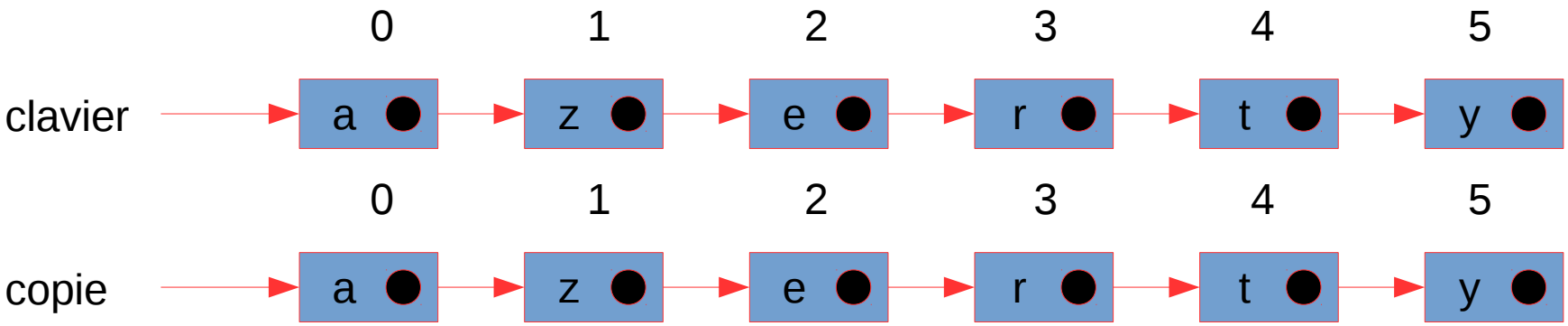

```
In [51]: clavier = ['a', 'z', 'e', 'r', 't', 'y']
```

```
In [52]: bis = clavier
         tierce = clavier
```

ne crée pas de copie de l'objet

```
In [53]: bis[2] = 99
         print( clavier )
```

```
['a', 'z', 99, 'r', 't', 'y']
```



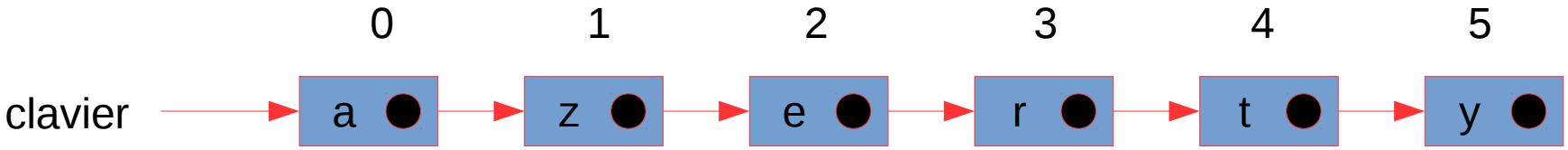
In [54]:
```
clavier = ['a', 'z', 'e', 'r', 't', 'y']
copie = clavier.copy()
copie[2] = 99
print(clavier)
print(copie)
```

```
['a', 'z', 'e', 'r', 't', 'y']
['a', 'z', 99, 'r', 't', 'y']
```

In [55]:
```
clavier = ['a', 'z', 'e', 'r', 't', 'y']
copie = clavier[:]
copie[2] = 99
print(clavier)
print(copie)
```

renvoie la liste (en entier)

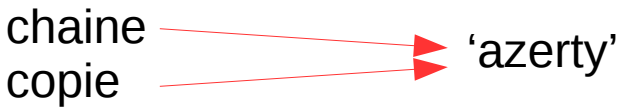
```
['a', 'z', 'e', 'r', 't', 'y']
['a', 'z', 99, 'r', 't', 'y']
```



id : identité, valeur unique (adresse mémoire) attribuée à un objet

```
In [15]: chaine = 'azerty'
         copie = chaine
         print(id(chaine))
         print(id(copie))
```

```
140676757666648
140676757666648
```




```
In [16]: copie = 'test'
         print(id(chaine))
         print(id(copie))
```

```
140676757666648
140676855639824
```


```
In [17]: clavier = ['a', 'z', 'e', 'r', 't', 'y']
         print(id(clavier))
```

```
140676757828744
```

Une variable contient une adresse mémoire (l’endroit où est stocké l’information)

```
In [19]: liste_2 = clavier
         liste_2[3] = 'w'
         print(id(liste_2))
```

```
140676757828744
```

Lors d’une affectation, une unique référence est copiée

Tests et booléens

In [57]:
```
1 > 2
```

Out[57]: False

In [58]:
```
1+1 == 2
```

Out[58]: True

In [59]:
```
1 != 2
```

Out[59]: True

In [61]:
```
1 <= 2
```

Out[61]: True

In [62]:
```
True and False
```

Out[62]: False

In [63]:
```
True or False
```

Out[63]: True

In [64]:
```
not True
```

Out[64]: False

In [66]:
```
(not 1>2) and 1==1
```

Out[66]: True

Tests et booléens

In [67]:
```
liste = [1, 2, 4, 8, 16]
7 in liste
```

Out[67]:
```
False
```

In [68]:
```
liste = [1, 2, 4, 8, 16]
2 in liste
```

Out[68]:
```
True
```

In [71]:
```
liste = [1, 2, 4, 8, 16, 16]
7 not in liste
```

Out[71]:
```
True
```

In [80]:
```
mot = "bonjour à tous"
print( 'n' in mot )
print( "z" in mot )
```

```
True
False
```

Tests et booléens

In [81]:
```
x = 17
if x < 15:
    print("bonjour")
else:
    print("au revoir")
```

```
au revoir
```

In [87]:
```
x = 10
if x < 15:
    print("bonjour")
else:
    print("au revoir")
```

```
bonjour
```

In [88]:
```
x = 42
if x < 15:
    print("x < 15")
elif x<30:
    print("15 <= x < 30")
elif x<45:
    print("30 <= x < 45")
else:
    print("x >= 45")
```

```
30 <= x < 45
```

Entrée de l'utilisateur & un mot sur les exceptions

In [19]:
```
val = input("entrez un nombre entier :")
try:
     val = int(val)
except ValueError:
     print("ce n'est pas un nombre entier !")
```

```
entrer un nombre entier :17
```

In [16]:
```
val
```

Out[16]:
```
17
```

In [20]:
```
val = input("entrez un nombre entier :")
try:
     val = int(val)
except ValueError:
     print("ce n'est pas un nombre entier !")
```

```
entrez un nombre entier :a7.2
ce n'est pas un nombre entier !
```

In [ ]:
```

```

Entrée de l'utilisateur & un mot sur les exceptions

Les erreurs détectées durant l'exécution sont appelées des exceptions,
Elles peuvent être gérées au sein du programme

```
In [21]: 17 * 1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-21-223fa14b1104> in <module>
----> 1 17 * 1/0

ZeroDivisionError: division by zero
```

```
In [22]: 17 * x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-6037534c1709> in <module>
----> 1 17 * x

NameError: name 'x' is not defined
```

```
In [23]: 17 + '17'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-23-c930c3d0f282> in <module>
----> 1 17 + '17'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Entrée de l'utilisateur & un mot sur les exceptions

Les erreurs détectées durant l'exécution sont appelées des exceptions,
Elles peuvent être gérées au sein du programme

In [6]:
```
try:
    print("essayons ceci...")
    17/0
except NameError:
    print("un premier type d'erreur")
except ZeroDivisionError:
    print("un autre type d'erreur")
except:
    print("toute autre erreur")
```

```
essayons ceci...
un autre type d'erreur
```