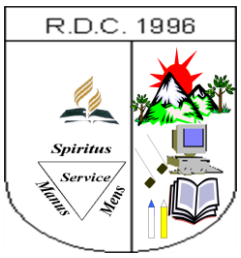
REPUBLIQUE DEMOCRATIQUE DU CONGO

PROVINCE DU NORD KIVU

UNIVERSITE ADVENTISTE DE LUKANGA

B.P. 180 BUTEMBO




# COURS D'ALGORITHMIQUE ET METHODES DE PROGRAMMATION

Travail effectué par :

MUMBERE WASUKUNDI SAMMY #4283

L2 SCIENCES INFORMATIQUE

2022-2023

# Chapitre 1: Présentation d'algorithmes

Un **algorithme** est une suite finie et non ambiguë d'instructions et d'opérations permettant de résoudre une classe de problèmes. Le mot *algorithme* a une longue histoire. Sous l'influence de l'ancien espagnol *algorismo*, il apparaît en français déjà vers 1230 sous la forme *augorisme*, puis *algorisme* au XIII$^{e}$ siècle, pour désigner le calcul en chiffres, l'arithmétique.

Le domaine qui étudie les algorithmes est appelé l'algorithmique. On retrouve aujourd'hui des algorithmes dans de nombreuses applications telles que le fonctionnement des ordinateurs4, la cryptographie, le routage d'informations, la planification et l'utilisation optimale des ressources, le traitement d'images, le traitement de textes, la bio-informatique, etc.

## Définition générale

Un algorithme est une méthode générale pour résoudre un type de problèmes. Il est dit correct lorsque, pour chaque instance du problème, il se termine en produisant la bonne sortie, c'est-à-dire qu'il résout le problème posé.

L'efficacité d'un algorithme est mesurée notamment par :

- Sa durée de calcul ;
- sa consommation de mémoire vive (en partant du principe que chaque instruction a un temps d'exécution constant) ;
- la précision des résultats obtenus (par exemple avec l'utilisation de méthodes probabilistes) ;
- sa scalabilité (son aptitude à être efficacement parallélisé) ;
- etc.

Les ordinateurs sur lesquels s'exécutent ces algorithmes ne sont pas infiniment rapides, car le temps de machine reste une ressource limitée, malgré une augmentation constante des performances des ordinateurs.

## Quelques définitions connexes

- Finitude : « un algorithme doit toujours se terminer après un nombre fini d'étapes » ;

- Définition précise : « chaque étape d'un algorithme doit être définie précisément, les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas » ;
- Entrées : « quantités qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié » ;

## Algorithmes numériques

Les algorithmes sont des objets historiquement dédiés à la résolution de problèmes arithmétiques, comme la multiplication de deux nombres. Ils ont été formalisés bien plus tard avec l'avènement de la logique mathématique et l'émergence des machines qui permettaient de les mettre en œuvre, à savoir les ordinateurs.

## Algorithmes non numériques

La plupart des algorithmes ne sont pas numériques.

On peut distinguer :

- des algorithmes généralistes qui s'appliquent à toute donnée numérique ou non numérique : par exemple les algorithmes liés au chiffrement, ou qui permettent de les mémoriser ou de les transmettre ;
- des algorithmes dédiés à un type de données particulier (par exemple ceux liés au traitement d'images).

## Algorithmes dans la vie quotidienne

L'algorithmique intervient de plus en plus dans la vie quotidienne :

- Une recette de cuisine peut être réduite à un algorithme si on peut réduire sa spécification aux éléments constitutifs :
    - Des entrées (les ingrédients, le matériel utilisé) ;
    - Des instructions élémentaires simples (frire, flamber, rissoler, braiser, blanchir, etc.) dont les exécutions dans un ordre précis amènent au résultat voulu ;
    - Un résultat : le plat préparé.

    Cependant, les recettes de cuisine ne sont en général pas présentées rigoureusement sous forme non ambiguë : il est d'usage d'y employer des termes vagues laissant une

liberté d'appréciation à l'exécutant alors qu'un algorithme non probabiliste stricto sensu doit être précis et sans ambiguïté.

- Le tissage, surtout tel qu'il a été automatisé par le métier Jacquard, est une activité que l'on peut dire algorithmique.

# Chapitre 2 : variables et types de données simples

Le premier objectif est de pouvoir l'utiliser plus tard. Pour placer un élément dans une variable, nous lui donnons un nom, puis définissons une valeur.

Donc, une variable est créée pour réserver un emplacement en mémoire dans lequel on va stocker une valeur. Selon le type de variable, l'interpréteur allouera de la mémoire et déterminera le type de valeur à stocker.

Par exemple :

poids = 70

Nous avons affecté la valeur 70 à une variable dont le nom est poids à l'aide du signe '='

Rappelez-vous, **à gauche, c'est le nom de la variable** et **à droite, c'est sa valeur**, le signe **'=' n'est pas symétrique**. En Python, ce signe ne représente pas une égalité comme en mathématique, mais une affectation.

En fait, une variable Python est tout simplement un nom qui fait référence à **un emplacement mémoire** utilisé par un programme. Vous pouvez l'utiliser pour demander à l'ordinateur d'enregistrer ou de récupérer des données vers et depuis cet emplacement de mémoire. **Python diffère** considérablement des langages tels que **Java, C ou C ++** en ce qui concerne le traitement des variables.

D'autres langages déclarent et lient une variable à un type de données spécifique. Cela signifie que vous pouvez stocker un type de données unique. Par conséquent, si une variable est de type entier, vous ne pouvez mettre que des entiers lors de l'exécution de votre programme.

## Affecter une valeur à plusieurs variables

Python est beaucoup plus flexible lorsqu'il s'agit de gérer des variables. Vous choisissez un nom et lui assignez une valeur. Si nécessaire, vous pouvez modifier la valeur et le type de données.

De plus, il est possible d'affecter une valeur à deux variables, elles feront en fait référence au même objet ou à la même valeur en mémoire.

Une ligne commençant par le signe # indique à l'interpréteur Python qu'il s'agit d'un commentaire. Donc, l'interpréteur va l'ignorer et passer à la ligne suivante.

```
# x et y font référence au même objet

x = 100

y = x
```

Le nommage d'une variable en Python doit respecter certaines règles :

1- Le nom de variable peut être une combinaison de **lettres en minuscules** (a à z) ou en **majuscules** (A à Z) ou des **chiffres** (0 à 9) ou un **trait de soulignement** ( _ ).
2- Il ne peut pas commencer par un chiffre.
3- Vous ne pouvez pas utiliser des symboles spéciaux comme ! , @ , # , $ , % etc.
4- Python est sensible à la casse. **Age** et **age**, c'est deux variables différentes.
4- Les mots clés ne peuvent pas être utilisés.

## Les types de variables simples en Python

Contrairement à d'autres langages de programmation, vous n'êtes pas obligés de déclarer le type d'une variable en Python.

### Les nombres entiers – int

Un entier 'int' ou 'integer' en anglais, est une valeur numérique sans virgule.

```
nombre_entier = 2
```

### Les nombres flottants – float

Un nombre flottant est **un nombre à virgule**, en anglais ‘float’. Mais nous devons remplacer la virgule par un point.

nombre_flottant = 2.5

**Les chaînes de caractères – str**

Une chaîne de caractères est **une série de caractères** entre **apostrophes** ' ', **guillemets** " " ou **triples guillemets** """ """.
‘str’ du mot ‘string’, qui veut dire une chaîne de caractères en anglais.

chaîne_de_caractère = "Bonjour!"

**Les booléens**

Les booléens peuvent ne pas sembler très utiles au premier abord, mais ils sont importants lorsque vous utiliserez des instructions conditionnelles. Les booléennes sont True ou False, ce qui veut dire **vraie** ou **faut**. True et False doivent commencer par une majuscule.

# Chapitre 3 : Présentation des listes

Les listes (ou list / array) en python sont une variable dans laquelle on peut mettre plusieurs variables.

**Créer une liste en python**

Pour créer une liste, rien de plus simple :

```
>>> liste = []
```

Vous pouvez voir le contenu de la liste en l'appelant comme ceci :

```
>>> liste
<type 'list'>
```

Ajouter une valeur à une liste python, vous pouvez ajouter les valeurs que vous voulez lors de la création de la liste python :

```
>>> liste = [1,2,3]
>>> liste
[1, 2, 3]
```

Ou les ajouter après la création de la liste avec la méthode append (qui signifie "ajouter" en anglais) :

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

On voit qu'il est possible de mélanger dans une même liste des variables de type différent. On peu t d'ailleurs mettre une liste dans une liste.

On peut accéder aux données d'une liste à l'aide de leur indice associé.

```
>>> print(nombres[2])
13
>>> print(fromages[0])
roquefort
>>> print(fromages[-1])
comté
>>> print(nombres [0:3])
[2, 5, 13]
```

On peut accéder à la taille d'une liste à l'aide de la fonction len(). Elle renvoie le nombre d'éléments présents dans la liste.

```
>>> len(nombres)
5
>>> len(fromages)
4
```

# Chapitre 4 : utilisation des listes

## Contrôle du flux d'exécution

Dans la plupart des programmes que vous allez écrire vous aurez besoin d'utiliser des instructions qui permettront au programme de suivre des chemins différents selon les circonstances. Pour ceci il est nécessaire de disposer d'instructions capables de *tester une certaine condition* et de modifier le comportement du programme en conséquence.

L'instruction qui est sans doute la plus utile afin de permettre un tel comportement est l'instruction if. Son fonctionnement sous Python est très simple. Si la condition à droite du mot-clé if est vraie, alors le bloc d'instructions en dessus est exécuté.

```
>>> if age > 18 :
...     print ("La personne peut acheter de l'alcool.")
```

On peut, si on le souhaite ajouter une instruction else pour exécuter un autre bloc si la condition est fausse :

```
>>> if age > 18:
...     print ("La personne peut acheter de l'alcool.")
... else:
...     print ("Trop jeune !")
```

Plutôt que d'emboîter des instructions if, on peut utiliser une instruction if suivie par une ou plusieurs instructions elif (raccourci de *else if*) :

```
>>> if n == 0:
...     print("Le nombre est égal à zéro")
... elif n > 0:
...     print("Le nombre est positif")
... else:
...     print("Le nombre est négatif")
```

## Boucles

```
>>> a = 0
>>> while a < 10: # N'oubliez pas le deux-points !
...     print(a)   # N'oubliez pas l'indentation !
...     a += 2
...
```

Avant d'introduire l'instruction for parlons un peu de la fonction range(). Cette fonction peut nous être très utile lorsque on veut itérer sur une suite de nombres. Elle génère des progressions arithmétiques.

```
range(2, 10) # 2, 3, 4, 5, 6, 7, 8, 9
range(0, 15, 2) # 0, 2, 4, 6, 8, 10, 12, 14
```

La boucle for est très utile lorsque on veut répéter un bloc d'instructions un nombre des fois connu à l'avance. Si on veut par exemple imprimer tous les nombres de 0 à 5, voici comment on peut le faire à l'aide de l'instruction for et de la fonction range.

```
>>> for i in range(6): # N'oubliez pas le deux-points !
...     print(i)        # N'oubliez pas l'indentation !
...
```

Comme on va le voir un peu plus tard, la boucle for peut être utilisée très facilement pour parcourir les éléments d'une liste.

## Fonctions

Pour créer une fonction en Python, on commence par le mot-clé def (définition). Il doit être suivi par le nom de la fonction, la liste des paramètres entre parenthèses et un deux-points ':'. Le corps de la fonction commence à la ligne suivante et doit être écrit avec un retrait de quelques espaces.

Voici une fonction qui imprime les n premiers termes de la suite *Fibonacci*.

```
def fibonacci(n):
    a, b = 0, 1
    for i in range(20):
        print(a, end=' ')
        a, b = b, a + b
    print(a)
```

```
>>> fibonacci (20)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

On peut bien sûr écrire une fonction qui nous renvoie quelque chose. Ceci se fait avec le mot-clé return. Voici une fonction qui renvoie la somme des carrés des entiers de 0 à n.

```
def sommeCarres(n):
    sum = 0
    for i in range(n):
        sum += i**2
    return sum
```

```
>>> sommeCarres (20)
2470
```

## Listes

Les listes sont des structures ordonnées de données. En Python, une liste est définie à l'aide des crochets.

```
nombres = [2,5,13,-35,0]
fromages = ['roquefort', 'camembert', 'saint-nectaire', 'comté']
```

Voici quelques astuces pour manipuler des listes :

**Boucler sur une liste**

Pour afficher les valeurs d'une liste, on peut utiliser une boucle :

```
>>> liste = ["a","d","m"]
>>> for lettre in liste:
...     print lettre
...
```

Si vous voulez en plus récupérer l'index, vous pouvez utiliser la fonction enumerate.

```
>>> for lettre in enumerate(liste):
...     print lettre
...
(0, 'a')
(1, 'd')
(2, 'm')
```

NB : Les valeurs retournées par la boucle sont des tuples.

**Copier une liste**

Beaucoup de débutants font l'erreur de copier une liste de cette manière

```
>>> x = [1,2,3]
>>> y = x
```

Or si vous changez une valeur de la liste y, la liste x sera elle aussi affectée par cette modification :

```
>>> x = [1,2,3]
>>> y = x
>>> y[0] = 4
```

>>> x

[4, 2, 3]

### Recherche dichotomique

La recherche dichotomique est un algorithme très simple et efficace pour rechercher un élément dans une liste triée.

# Chapitre 5 : Instructions IF

Cette notion est l'une des plus importante en programmation. L'idée est de dire que si telle variable a telle valeur alors faire cela sinon cela.

Un booléen est une valeur qui peut être soit vraie (true) soit fausse (false). Cette valeur est utilisée pour représenter deux états possibles, généralement "vrai" et "faux", "oui" et "non", "1" et "0", etc. Les booléens sont souvent utilisés dans les algorithmes pour effectuer des tests et prendre des décisions. Par exemple, un programme informatique peut utiliser un booléen pour vérifier si un nombre est pair ou impair. Si le nombre est pair, le booléen sera "vrai", sinon il sera "faux".

Les booléens peuvent également être utilisés dans les expressions conditionnelles. Par exemple, si on veut afficher un message seulement si une certaine condition est remplie, on peut utiliser un booléen pour savoir si la condition est vraie ou fausse. Si le booléen est "vrai", le message sera affiché, sinon il ne le sera pas.

En informatique, les booléens sont souvent utilisés en conjonction avec les opérateurs de comparaison, qui permettent de comparer deux valeurs entre elles. Par exemple, l'opérateur "égal à" (==) permet de vérifier si deux valeurs sont égales, et renvoie "vrai" si c'est le cas, "faux" sinon. Il existe d'autres opérateurs de comparaison tels que "différent de" (!=), "supérieur à" (>), "inférieur à" (<), etc.

En résumé, les booléens sont une valeur très utile en informatique qui permet de représenter deux états possibles et de prendre des décisions en fonction de ces états. Ils sont souvent utilisés dans les algorithmes et les expressions conditionnelles pour effectuer des tests et contrôler le flux d'exécution d'un programme.

Prenons un exemple, on va donner une valeur à une variable et si cette valeur est supérieure à 5, alors on va incrémenter la valeur de 1.

>>> a = 10

>>> if a > 5:

```
...     a = a + 1
...
>>> a
11
```

**Condition if else**

Il est possible de donner des instructions quelque soit les choix possibles avec le mot clé else.

```
>>> a = 20
>>> if a > 5:
...     a = a + 1
... else:
...     a = a - 1
...
>>> a
21
```

**Condition elif**

Il est possible d'ajouter autant de conditions précises que l'on souhaite en ajoutant le mot clé elif, contraction de "else" et "if", qu'on pourrait traduire par "sinon".

```
>>> a = 5
>>> if a > 5:
...     a = a + 1
... elif a == 5:
...     a = a + 1000
... else:
...     a = a - 1
...
>>> a
1005
```

**Les comparaisons possibles**

Il est possible de comparer des éléments :

== égal à

!= différent de (fonctionne aussi avec)

> strictement supérieur à

>= supérieur ou égal à

< strictement inférieur à

<= inférieur ou égal à

**AND / OR**

Il est possible d'affiner une condition avec les mots clé AND qui signifie " ET " et OR qui signifie " OU ".

On veut par exemple savoir si une valeur est plus grande que 5 mais aussi plus petite que 10 :

>>> v = 15

>>> v > 5 and v < 10

False

Testons maintenant la condition OR

>>> v = 11

>>> v > 5 or v > 100

True

# Chapitre 6 : Les dictionnaires

Un dictionnaire en python est une sorte de liste mais au lieu d'utiliser des index, on utilise des clés alphanumériques.

**Comment créer un dictionnaire python ?**

Pour initialiser un dictionnaire, on utile la syntaxe suivante :

>>> a = {} ou >>> a = dict ()

Comment ajouter des valeurs dans un dictionnaire python ?

Pour ajouter des valeurs à un dictionnaire il faut indiquer une clé ainsi qu'une valeur :

>>> a = {}

```
>>> a["nom"] = "Wayne"
>>> a["prenom"] = "Bruce"
>>> a
{'nom': 'Wayne', 'prenom': 'Bruce'}
```

Vous pouvez utiliser des clés numériques comme dans la logique des listes.

**Comment récupérer une valeur dans un dictionnaire python ?**

La méthode *get* vous permet de récupérer une valeur dans un dictionnaire et si la clé est introuvable, vous pouvez donner une valeur à retourner par défaut :

```
>>> data = {"name": "Wayne", "age": 45}
>>> data.get("name")
'Wayne'
>>> data.get("adresse", "Adresse inconnue")
'Adresse inconnue'
```

**Comment vérifier la présence d'une clé dans un dictionnaire python ?**

Vous pouvez utiliser la méthode *haskey* pour vérifier la présence d'une clé que vous cherchez :

```
>>> a.has_key("nom")
True
```

**Comment supprimer une entrée dans un dictionnaire python ?**

Il est possible de supprimer une entrée en indiquant sa clé, comme pour les listes :

```
>>> del a["nom"]
>>> a
{'prenom' : 'Bruce'}
```

# Chapitre 7 : Les boucles for et while

Une boucle (ou loop) vous permet de répéter à l'infini des instructions selon vos besoins.

**Le boucle while**

En anglais " while " signifie "Tant que". Pour créer une boucle, il faut donc utiliser ce mot clé suivi d'une indication qui dit quand la boucle s'arrête.

Un exemple sera plus parlant :

On désire écrire 100 fois cette phrase :

" Je ne dois pas poser une question sans lever la main "

Ecrire à la main prend beaucoup de temps et beaucoup de temps x 100 c'est vraiment beaucoup de temps, et peu fiable, même pour les chanceux qui connaissent le copier-coller. Et un bon programmeur est toujours un peu fainéant perfectionniste, il cherchera la manière la plus élégante de ne pas répéter du code.

```
>>> i = 0
>>> while i < 10 :
...     print ("Je ne dois pas poser une question sans lever la main")
...     i = i +1
...
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
```

**La boucle for**

La boucle for permet de faire des itérations sur un élément, comme une chaine de caractères par exemple ou une liste .

Exemple:

```
>>> v = "Bonjour toi"
>>> for lettre in v:
...     print(lettre)
...
B
o
n
j
o
u
r

t
o
i
```

**Range**

Il est possible de créer une boucle facilement avec range :

```
for i in range(0,100):
  print(i)
```

**Stopper une boucle avec break**

Pour stopper immédiatement une boucle on peut utiliser le mot clé break :

```
>>> liste = [1,5,10,15,20,25]
>>> for i in liste:
...     if i > 15:
...             print("On stoppe la boucle")
...             break
...     print(i)
...
1
5
10
15
On stoppe la boucle
```

## Références


1. Patrice Hernert, Les algorithmes, Paris, Presses universitaires de France, coll. « Que sais-je ? », 2002, 128 p. (ISBN 978-2-13-053180-7, OCLC 300211244), p. 5.
2. En particulier dans les systèmes d'exploitation et la compilation
3. (en) Donald E. Knuth, Algorithmes, Stanford, CSLI Publications, 2011, 510 p. (ISBN 978-1-57586-620-8).
4. Etc.
5. Le Goff, Vincent, Apprenez à Programmer en Python, Le livre du Zéro
6. Puiseaux, Pierre, Le Langage Python, Ellipses TechnoSup
7. Zelle, John, Python Programming, Franklin, Beedle, and Associates
8. Lee, Kent D., Python Programming Fundamentals, Springer