THEME 6

Langage et programmation

Partie A : Les bases de Python

Partie B : Prototype d'une fonction

Partie C : Documentation d'une bibliothèque

*

Sommaire

Partie I

Algorithme et langage de programmation

Algorithme

i

Le mot « algorithme » vient du nom du grand mathématicien persan Al Khwarizmi (vers l'an 820), qui introduisit en Occident la numération décimale (rapportée d'Inde) et enseigna les règles élémentaires des calculs s'y rapportant.

La notion d'algorithme est donc historiquement liée aux manipulations numériques, mais elle s'est progressivement développée pour porter sur des objets de plus en plus complexes, des textes, des images, des formules logiques, des objets physiques, etc.

Source : https ://interstices.info

Définition

Algorithme

Un algorithme est suite non ambiguë d'instructions ou d'opérations permettant de résoudre un certain type de problème.

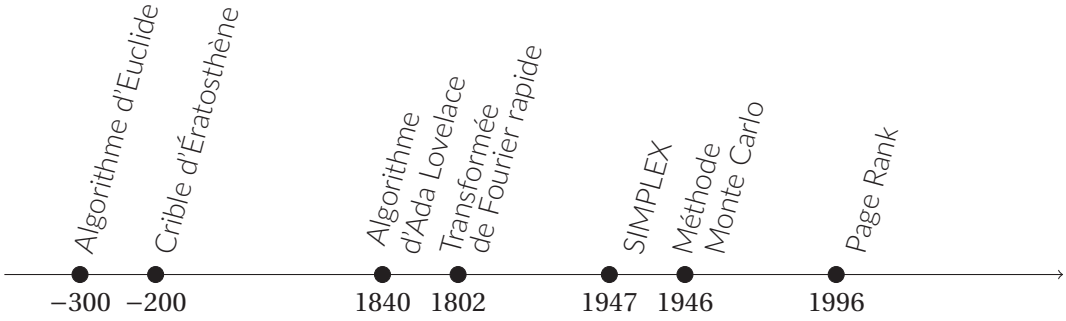
Par exemple pour résoudre le problème de détermination du plus grand diviseur commun de 2 nombres, on peut utiliser l'algorithme d'Euclide.

Exemple

Un algorithme c'est donc un peu comme une recette de cuisine.
Mais attention, un algorithme ne doit pas laisser de place à des instructions ambiguës.

L'instruction "saler et poivrer" que l'on peut retrouver dans une recette de cuisine n'ai pas suffisamment claire pour figurer dans un algorithme. Il faudrait préciser la masse de sel et de poivre.

• Quelques algorithmes



Langage de programmation

Un ordinateur ne comprend que le code binaire (une suite de 0 et de 1). La programmation directement dans ce langage est donc très fastidieuse et ne se prête pas à notre style de programmation actuel. Il faut donc trouver un traducteur entre l'humain et la machine. C'est le rôle du langage de programmation.

Source : OpenClassroom

i Grace Murray Hopper (1906-1992) est une informaticienne américaine, connue pour sa contribution à la conception et à la traduction des premiers langages de programmation. Elle conçoit le premier compilateur (A-0 System) pour l'UNI VAC-1 (UNIVersal Au tomatic Computer I) , le premier ordinateur commercial réalisé aux États-Unis.
À partir de 1957, Grace Hopper, employée par la firme IBM, défend l'idée qu'un programme devrait être écrit dans un langage proche du langage naturel plutôt qu'en langage machine. et sera considérée comme l'une des principales inspiratrices du langage COBO

Partie II

Importation des modules et packages

i Les modules et les packages sont un ou des fichiers contenant un ensemble de fonctions, de classes et de variables prédéfinies

• Importation classique

L'importation d'un module de manière classique se fait par l'instruction d'importation.

Exemple

Par exemple le module `turtle`, on utiliser l'instruction `import turtle` Ensuite on peut utiliser les instructions du module `turtle` comme par exemple la fonction `forward`.
⚠ Avec ce type d'import, il faudra préfixer l'instruction du nom du module

```
1 import turtle
2
3 turtle.forward(100)
```

• Importation avec alias

Pour éviter de devoir préfixer toutes les instructions avec le nom du module, on peut utiliser un **alias** à l'aide de l'instruction `as`.On peut ainsi raccourcir le nom du module

Exemple

Toujours avec l'exemple du module `turtle`

```
1 import turtle as tl
2
3 tl.forward(100)
```

Importation tout ou une partie d'un module

On peut aussi choisir d'importer seulement quelques instructions d'un module, dans ce cas on utilisera les instruction `from module import instruction`.

Exemple

Toujours avec l'exemple du module `turtle`

```
1 from turtle import forward
2
3 forward(100)
```

Il est aussi possible d'importer de cette façon toutes les instructions d'un module avec `*`.

⚠ Cette méthode n'est pas conseillée.

Exemple

```
1 from turtle import *
2
3 forward(100)
```

Exercices

Exercice 1 ★

Quelle instruction permet d'importer l'intégralité du module `math` sans préfixe ?

. . . . . .

. . . . . .

Exercice 2 ★

Quelle instruction permet d'utiliser l'instruction `forward(100)` du module `turtle` après l'import fait en ligne 1 ?

```
1 import turtle as tortue
```

. . . . . .

Partie III

Les variables

i Il est souvent intéressant de pouvoir en conserver et réutiliser une valeur, par exemple le résultat d'un calcul. C'est le principe des variables : mémoriser une valeur.

• Affectation

En Python, les variables fonctionnent comme des étiquettes. Quand on assigne une valeur à une variable, on pose une étiquette avec un nom sur une valeur.
C'est-à-dire que la valeur existe quelque part en mémoire et qu'on vient lui attacher une étiquette. On peut aisément placer plusieurs étiquettes sur une même valeur, mais aussi retirer une étiquette pour la placer sur une autre valeur.

L'affectation se fait à l'aide du signe **=**

Affectation

```
1 a = 1
```

Réaffectation

```
a = 2
```

Double affectation

```
a = b
```

Source : http ://foobarnbaz.com/2012/07/08/understanding-python-variables/

• Les types de variables

Il existe plusieurs type de variables. Parmi les plus courants, on retrouve :

Variable	Type	Remarques
`a = 2`	`int` : Les nombres entiers relatifs	
`a = 3.5`	`float` : Les nombres réels	Le séparateur est le . et non la ,
`a = "Bonjour"`	`str` : Les chaines de caractères	Entourer par des guillemets (`"`) ou des apostrophes (`'`) ou des triples guillemets (`"""`) ou des triples apostrophes (`'''`)
`a = True`	`bool` : Les booléens True ou Fasle	

• Le nom des variables

•• Obligations :

— Ne pas contenir un signe opératoire + - * / %/ ni le caractères # (*Il est réserver aux commentaires*)

— Doit commencer par une lettre

— Ne doit pas être un mot clé Python (`if`, `in`, `as` ;), ...)

•• Préconisations :

— le nom décris le contenu variable et est constituer d'au moins 3 caractères.

— le nom est entièrement écrit en minuscules et les mots sont séparés par des espaces soulignés (underscores _). Exemple : ma_variable

— le nom doit permettre de comprendre le rôle de la variable, éviter les caractères spéciaux et accentuer

• Opérations arithmétiques

Opérateur	Équivalent
a + b	Somme de `a` et `b`
a - b	Différence de `a` et `b`
a * b	a = a * b
a ** b	`a` puissance `b` : a^b
a / b	Quotient de `a` par `b`
a // b	Dividende de `a` par `b`
a % b	Reste de la division euclidienne de `a` par `b`

• Sucre syntaxique

Comme dans d'autres langages certains opérateurs sont introduits pour raccourcir les processus de réaffectation des variables.

Exemple

Plutôt que d'écrire

```
1 i = i + 1
```

on préférera le raccourci suivant :

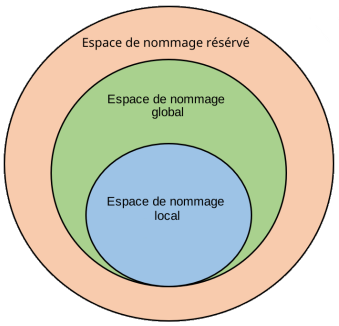
```
1 i += 1
```

Opérateur	Équivalent
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a **= b	a = a ** b
a /= b	a = a / b
a //= b	a = a // b
a %= b	a = a % b

Espace de nommage

Quand vous faites référence à une variable, Python cherche la référence à l'objet correspondant dans l'espace de nommage. Un espace de nommage est donc simplement une collection de noms associés à des références d'objet. C'est ce qui fait la correspondance entre le nom de la variable et ce à quoi elle se réfère.
Il existe de nombreux espaces de noms :

- L'espace des nommage réservé de Python (built-in) : créé quand on lance l'interpréteur.
- L'espace de nommage global : créé quand on lance l'interpréteur.
- L'espace de nommage locaux : espace lié à un module, une fonction ou une classe



Exercices

Exercice 3 ★Affectation

Que vaut les variables `a`, `b` et `c` après l'exécution de ces instructions.

```
1 a=3
2 b=7
3 c=b+a+2
```

. . . . . .

. . . . . .

. . . . . .

Exercice 4 ★Affectation

Que vaut les variables `a`, `b` et `c` après l'exécution de ces instructions.

```
1 a = 5
2 b = 2
3 a = b + 3
4 b = a + 2
5 c = a + b
```

. . . . . .

. . . . . .

. . . . . .

Exercice 5 ★Bien nommer ces variables

Parmi les noms suivants, quels sont ceux qui ne peuvent être utiliser pour nommer des variables en Python. Justifier

- ma variable
- 1_nombre
- ma_variable
- %V4ri4ble$
- nombre1
- ma-variable
- maVariable
- mavariable

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Partie IV

Entrée - Sortie

Dans cette partie, nous verrons comment :

- afficher du texte,
- permettre à l'utilisateur de saisir du texte dans le terminale.

Sortie

Affichage du texte

En Python, pour afficher du texte, on utilise l'instruction **`print`**.
Si l'on veut afficher une chaine de caractère, on la placera entre guillemets :

Exemple

Pour afficher à l'écran le texte **`Bonjour`**.

Code

```
1 print("Bonjour")
```

Affichage

```
1 Bonjour
```

i En effet, **`print`** en anglais signifie « imprimer ».

Chaque ligne du programme représente une instruction.

Exemple

Code

```
1 print("Bonjour,")
2 print("Je m'appelle Camthalion")
3 print("Au revoir !")
```

Affichage

```
1 Bonjour,
2 Je me nomme Camthalion
3 Au revoir !
```

Source : FranceIOI

Formated string

Il peut être utile de placer un élément variable dans une chaine de caractère.
Par exemple, le résultat d'un calcul. Ainsi le code ci-dessous

Code

```
print(f"2 + 2 = {2+2}")
```

Affichage

```
2 + 2 = 4
```

L'instruction entre les accolades est évaluée avant l'affichage.

Code

```
nom = "John"
print(f"Bonjour {nom} !")
```

Affichage

```
Bonjour John !
```

Les erreurs

Quand on écrit un programme informatique, il faut être très rigoureux car un simple caractère mal placé peut perturber l'ordinateur du robot. Si le format de votre code n'est pas bon, il ne va pas aller plus loin et va simplement vous indiquer que vous avez commis une erreur. Quand ça arrive, il faut d'abord tenter de comprendre le message d'erreur et ensuite relire attentivement la zone de code concernée, en particulier pour s'assurer qu'il ne manque pas de symbole tel qu'une parenthèse ou un guillemet. Voici quelques exemples de messages d'erreur.
Attention aux parenthèses ! Si on oublie une parenthèse, cela produit une des erreurs suivantes :

```
    print("Bonjour"
    SyntaxError: unexpected EOF while parsing
```

```
    print "Bonjour")
    SyntaxError: invalid syntax
```

Attention aux guillemets !
Si on ne place pas correctement les guillemets, cela produit une des erreurs suivantes :

```
    print("Bonjour)
    SyntaxError: EOL while scanning string literal
```

```
    print(Bonjour)
    NameError: name 'Bonjour' is not defined
```

```
    print(Bonjour tout le monde)
    SyntaxError: invalid syntax
```

• Entrée

Dans certains cas, il peut être nécessaire de demander des informations à l'utilisateur.

•• L'instruction `input`

L'instruction **`input`** permet à l'utilisateur de saisir des données au clavier.

Exemple

```
1 prenom = input("Entrer votre prénom :")
2 print(f"Bonjour {prenom}")
```

Si l'utilisateur saisie **John**, l'affichage sera :

```
1 Bonjour John
```

i La chaine de caractères donnée en argument de la fonction **`input`** (**`"Entrer votre nom:"`**) est facultative mais est utile à l'utilisateur pour savoir ce qu'on attend.

•• La conversion

i L'instruction **`input`** considère les entrées utilisateurs comme des chaines de caractères. Il peut donc être utile de changer le type de la variable.

Exemple

Exécutons le code suivant.

```
1 nombre = input("Entrer un nombre")
2 nombre = nombre * 5
3 print(nombre)
```

Si l'utilisateur entre **3**
L'affichage sera :

```
1 33333
```

? Pourquoi l'interpréteur affiche **33333** et non **15** comme attendu ?

L'interpréteur Python considérant la variable **`nombre`** comme une chaine de caractère, il va interpréter l'instruction **`nombre * 5`** comme **`répéter 5 fois la valeur de la variable nombre`**.
De fait si l'on veut que le programme fonctionne correctement, il faut donc changer le type de la variable. Dans notre exemple, nous allons utilisant l'instruction **`float`** pour transformer la saisie en nombre à virgule flottante. Ce qui donnerait :

```python
nombre = float(input("Entrer un nombre"))
nombre = nombre * 5
print(nombre)
```

Partie V

Boucle

i Les boucles font partie des grandes structures de base de l'algorithmique, et donc de la programmation.
Il s'agit de répéter une suéquences d'instruction un certain nombre de fois.
On dit que l'on réalise une itération de la boucle à chaque fois que la séquence d'instruction est exécutée.

Il existe deux types de boucle :

- Les **boucles bornés ou itératives**, dont on connait par avance le nombre de répétitions souhaitées
- Les **boucles conditionnelles**, dont on ne connait pas le nombre de répétitions mais qui possèdent une condition d'arrêt.

Les boucles itératives

• structure d'une boucle itérative

En Python, lorsque que l'on connaît le nombre de répétitions à effectuer, on utilise généralement la boucle itérative `for`
A chaque itération, Python va prendre un élément de l'itérable et de la boucle à chaque fois que le corps de la boucle est exécuté. Le bloc d'instructions à répéter est délimité par l'**indentation**.

for variable **in** itérable **:**
    bloc d'instructions indenté à répéter

Définition

Un itérable est un objet dont on peut parcourir les valeurs une par une.
Les types `list`, `tuple`, `dict`, `set` sont des exemples d'itérable.

• Répétition à l'aide de l'instruction `range`

•• L'instruction `range`

Pour obtenir un itérable contenant `n` valeurs, on peut utiliser l'instruction `range`

Exemple

```
1 iterable = range(5)
```

A la suite de cette instruction, la variable nommée `iterable` contiendra les valeurs :

$$\underbrace{0,\ 1,\ 2,\ 3,\ 4}_{\text{5 valeurs}}$$

•• Construction d'une boucle avec l'instruction `range`

Comme l'instruction `range` nous fournit un itérable contenant un nombre de valeurs que l'on souhaite, il est fréquent de l'utiliser dans les boucles itératives.
La boucle sera donc construite à l'aide de l'instruction `for var in range(nb_de_repetition):` et le bloc d'instructions à répéter est délimité par l'**indentation**.

```
for i in range(n) :
    bloc d'instructions indenté à répéter
```

Exemple :

L'exécution du code suivant

```
1 for _ in range(4):
2     print("Hello")
```

Affichera :

```
Hello
Hello
Hello
Hello
```

i L'instruction `range(4)` crée un objet de type `range` qui contient les valeurs 0, 1, 2 et 3.

Quand on écrit l'instruction : `for i in range(4):`
A chaque répétition de la boucle, la valeur de la variable `i` change et prend successivement les valeurs de l'objet range

Exercice 6 ★

Quelle sera l'affichage, après l'execution du code ci-dessous

```
1 for i in range(4):
2     print("Hello")
3 print("Word")
```

.....

.....

.....

.....

.....

.....

Exercice 7 ★Debuguer

Retrouver les erreurs dans les codes ci-dessous.

```
for _ in range(4)
    print("Bla")
```

.....

.....

.....

```
for _ range(4):
    print("Hello")
```

.....

.....

.....

```
for _ in range(4):
print("Bonjour")
```

.....

.....

.....

Exercice 8 ★★

Après exécution du code ci-dessous, quelle sera la valeur de la variable `res`.

```
res = 1
for i in range(5):
    res += i*3
```

.....

.....

.....

Exercice 9 ★

Écrire un programme qui calcule et affiche la somme des nombres de 1 à 99.

.....

.....

.....

.....

.....

.....

• Itérer sur une chaine de caractères

i Les objets itérables sont les objets que l'on peut parcourir à l'aide d'un `for`. Il possède généralement la méthode `__iter__`.

Les types `str, list, dict, tuple` sont les itérables que nous allons voir cette année.

Exemples :

Code

```
1 for mot in ["Hello","Word"]:
2     print(mot)
```

Sortie

```
Hello
Word
```

Code

```
1 for lettre in "Hello":
2     print(mot)
```

Sortie

```
H
e
l
l
o
```

Exercice 10 ★★

Complete le programme ci-dessous pour qu'il affiche le nombre de caractères de la chaine ch.

```
1 ch = "Vive les NSI"
2 counter = ..........................................................................
3 for ..........................................................................
4 ..........................................................................
5 ..........................................................................
```