Chap I Langage et constructions élémentaires

BRUNO DARID

18 août 2019

PLAN

1 Repères historiques

En 1954, John Backus(*1924-2007*), fig. 1, présente le premier vrai langage de programmation tel qu'on l'entend aujourd'hui : le Fortran. Il mit au point vers la fin des années 50 le langage Algol. Il a, par la suite, proposé la *forme Backus-Naur*, notation qui permet de décrire la *grammaire* des langages de programmation.

2 Le langage Python

2.1 Introduction

Un programme est un texte qui décrit un algorithme que l'on souhaite faire exécuter par une machine. Ce texte est écrit dans un langage particulier, appelé **langage de programmation**. Le premier langage abordé en spécialité NSI est le langage **Python** en version 3. C'est un langage de haut niveau (*d'abstraction*) et interprété (*un programme appelé interpréteur se charge d'exécuter ligne par ligne le code « source »*).

Fig. 1 – John Backus

2.2 Les entrées/sorties

Dans un langage de programmation on appelle **entrées/sorties** des constructions permettant une communication avec l'utilisateur, donnant ainsi un aspect interactif au programme.

En python l'instruction `print()` affiche à l'écran les objets passés en arguments. Dans les exemples qui vont suivre, on affiche des chaînes de caractères (en python ces dernières sont délimitées par des doubles quotes `" "` ou des simples quotes `' '`).

```
In [ ]: print("Hello World!")
        print("Python version 3.", "Environnement", "Jupyter Notebook 5.7")
        print("Python version 3.", "Env.", "Jupyter", sep="//", end=" ")
        print("Pas de retour à la ligne.")
```

Des arguments optionnels comme `sep` ou `end` par exemple, permettent d'adapter l'affichage aux besoins. Par ailleurs, l'instruction `input()` interrompt le déroulement du programme et attend que l'utilisateur tape quelque chose au clavier.

```
In [ ]: niveau = input("Quel est votre niveau (novice/intermédiaire/avancé)?")
        moyenne = input("Quelle était votre moyenne annuelle en math ?")
        print("Vous avez répondu:",niveau,"avec ", moyenne, "en math!")
```

Remarques

— `input()` retourne toujours une chaine de caractères (même si l'utilisateur entre un nombre !) ;

— la syntaxe employée à l'intérieur de la fonction `print()` bien que correcte, sera simplifiée dans les cours ultérieurs tenant compte des dernières évolutions du langage.

3 Les instructions simples

Pour pouvoir accéder aux données, un programme d'ordinateur fait abondamment usage de **variables**. Une variable est une *référence* désignant une **adresse mémoire**, c'est-à-dire un emplacement précis dans la mémoire vive.

Les instructions *simples* sont la **déclaration et l'affectation de variables**. En python cette opération est réalisée de la manière suivante : `nom_variable = valeur`. Par exemple

```
n = 7
msg = "Affectation"
pi = 3.14
```

Remarques

— Le nom de variable doit commencer par une lettre et ne pas être un mot réservé (voir liste) ;

— Contrairement à d'autres langages, on n'indique pas le type d'une variable lors de sa déclaration en python. Le type est déterminé lors de l'affectation. Dans l'exemple précédent, `n` est du type entier, `msg` est du type chaîne de caractères, `pi` est du type *flottant* (décimal).

4 Les instructions composées

4.1 La séquence

On peut définir une séquence d'instruction comme un ensemble de deux instructions exécutées l'une **à la suite** de l'autre (l'ordre est important !). Par exemple, les lignes

```
x = x + 1
x = x * 2
```

constituent une séquence d'instructions (*à condition que la variable x ait été affectée au préalable !*). Nos programmes comporteront généralement des séquences de plus de deux instructions, on parlera alors de **bloc d'instructions**.

Exercice E1C1

Soit un disque de centre O et de rayon r. Écrire un programme qui calcule et affiche l'aire S de ce disque à partir de la valeur du rayon qui sera demandée à l'utilisateur. *Indication* : $S = \pi * r^2$.

```
In [23]: pi = 3.14
         #Code à rajouter au dessous.
         #Toute suite de caractères (jusqu'à la fin de ligne) placés après un
         #dièse est ignorée.On s'en sert pour placer des commentaires.
```

4.2 Le test

4.2.1 Le test simple

Une instruction (ou un bloc d'instructions) conditionnelle n'est executée que si une condition est vérifiée. En python, la syntaxe est la suivante :

```
if condition:
    bloc_instructions
```

Dans l'exemple ci-dessous, % désigne le reste de la division euclidienne et == désigne l'égalité au sens test (et non l'affectation !)

```
In [1]: x = int(input("Entrer un nombre: "))# 'int' convertit l'entrée en entier
        if x%2 == 0:#Si le reste de la division de x par 2 est nul
            print("Le reste de sa division par 2 est nul, ")
            print(x,"est un nombre pair.")#alors on affiche la parité
```

```
Entrer un nombre: 6
Le reste de sa division par 2 est nul,
6 est un nombre pair.
```

Pour identifier sans ambiguités les instructions appartenant au bloc vérifiant la condition, on **DOIT LES INDENTER** en python. Indenter signifie décaler d'un certain nombre de colonne (*la bonne pratique indique 4 espaces*) par rapport au bloc d'instructions précédent.
La condition est souvent exprimée avec des opérateurs de comparaisons, dont voici la liste :

```
x == y # x est égal à y
x != y # x est différent de y
x > y # x est plus grand que y
x < y # x est plus petit que y
x >= y # x est plus grand que, ou égal à y
x <= y # x est plus petit que, ou égal à y
```

Exercice E2C1 : nombre pair divisible par 8
Écrire un programme python qui affiche le message "N est pair et divisible par 8" si le nombre N demandé à l'utilisateur vérifie cette condition.
Exercice E3C1 : valeur absolue
Écrire un programme qui transforme un nombre demandé à l'utilisateur, en sa valeur absolue.

4.2.2 Le test avec alternative

Parfois il est intéressant d'indiquer un bloc à traiter lorsque la condition *n'est pas* vérifiée. La syntaxe devient alors :

```
if condition:
    bloc_instructions_SI
else:
    bloc_instructions_SINON
```

L'exemple précédent devient :

```
In [2]: x = int(input("Entrer un nombre: "))# 'int' convertit l'entrée en entier
        if x%2 == 0:#Si le reste de la division de x par 2 est nul
            print("Le reste de sa division par 2 est nul, ")
            print(x,"est un nombre pair.")#alors on affiche la parité
        else:#x%2 est donc différent de 0
            print("Le reste de sa division par 2 n'est pas nul, ")
            print(x,"est un nombre impair.")

Entrer un nombre: 57
Le reste de sa division par 2 n'est pas nul,
57 est un nombre impair.
```

Question : aurait-on pu utiliser le code suivant pour tester la parité d'un nombre ?

```
x = int(input("Entrer un nombre: "))#int sert à convertir notre entrée en entier
if x%2 == 0:#Si le reste de la division de x par 2 est nul
    print("Le reste de sa division par 2 est nul, ")
    print(x,"est un nombre pair.")#alors on affiche la parité
    else:#x%2 est donc différent de 0
        print("Le reste de sa division par 2 n'est pas nul, ")
        print(x,"est un nombre impair.")
```

Exercice E4C1 : Que doit-on emmener ?

Écrire un programme capable de conseiller deux objets utiles à emporter en voyage, en fonction de la température de l'air. On demande à l'utilisateur la température prévue. Si celle-ci est supérieure à 20°, il conseille "maillot" et "crème bronzante". Sinon, il conseille "parapluie" et "pull".

Exercice E5C1 : année bissextile

Une année est bissextile si son millésime est multiple de 4. Cependant, les années dont le millésime est multiple de 100 ne sont bissextiles que si c'est aussi un multiple de 400 (*1900 n'était pas bissextile, 2000 l'a été*). Écrire un programme qui détermine si une année N est bissextile ou non.

Coup de pouce ?

Problème P1C1 : Affichage de l'heure

Écrire un programme qui récupère l'heure d'un utilisateur sous la forme HH mm ss. Le programme doit afficher cette heure augmentée d'une seconde.

4.3 La boucle

4.3.1 La boucle conditionnelle, *non bornée*

Nécessité

1. Exécuter le programme ci-dessous pour des valeurs de N entières comprises entre 1 et 4. Que réalise-t-il ?
2. Quel commentaire peut-on faire à propos du bloc d'instruction conditionnel ?
3. Que faudrait-il apporter comme modification (*ne pas le faire !*) si on souhaitait rendre possible le calcul jusqu'à $N = 10$?

```
In [7]: N = 4
        c = N
        p = 1
        if c != 0:
            p = p * 2
            c = c - 1
            if c != 0:
                p = p * 2
                c = c - 1
                if c != 0:
                    p = p * 2
                    c = c - 1
                    if c != 0:
                        p = p * 2
                        c = c - 1
        print(p)
```

```
16
```

La méthode précédente montre ses limites : pour des valeurs quelconques de N il faudrait répéter un grand nombre de fois les mêmes lignes. On aimerait exécuter les instructions

```
p = p * 2
c = c - 1
```

tant que le test `c != 0` est vérifié. Il s'agit d'une **boucle conditionnelle** ou encore **boucle while** (*en anglais*) ou **boucle tant que** (*en français*).

4.3.2 Syntaxe

En python, la boucle conditionnelle a la forme suivante :

```
while condition:
    bloc_instructions
```

Remarques

— la condition s'exprime souvent avec les opérateurs de comparaisons vu plus haut ;

— le bloc d'instructions est indenté.

Le programme précédent peut ainsi s'écrire (*le vérifier en l'exécutant*) :

```
In [18]: N = 8
         c = N
         p = 1
         while c > 0:
             p = p * 2
             c = c - 1
         print(p)
```

```
256
```

Exercice E6C1 : table de multiplication

Écrire un programme qui affiche les 10 premiers termes de la table de multiplication par 7 en utilisant une boucle conditionnelle.

Exercice E7C1 : affichage

Écrire un programme qui produit l'affichage ci-dessous, en utilisant une boucle conditionnelle :

```
*
**
***
****
*****
******
*******
```

4.3.3 Danger lié à l'utilisation des boucles conditionnelles

Il faut s'assurer que la condition ne soit plus vérifiée à un moment donné pour assurer la **terminaison** de la boucle et éviter le bouclage infini. Ce point sera détaillé dans le cours d'algorithmique.

Dans le cas précédent, c est un entier positif et décroit à chaque tour de boucle. On est sûr d'avoir $c = 0$ et sortir ainsi de la boucle au bout d'un certain nombre d'*itérations*.

4.3.4 La boucle inconditionnelle ou *bornée*

Lorsque le nombre d'itérations est connu d'avance la boucle devient inconditionnelle. On l'appelle encore **boucle for** ou **boucle pour**. En python, la syntaxe de la boucle `for` est la suivante :

```
for variable_compteur in sequence:
    bloc_instructions
```

Remarques

— `sequence` doit une collection d'objets itérables, comme une suite d'entiers par exemple (d'autres séquences seront vus dans les prochains cours) ; typiquement une séquence d'entiers est obtenue par l'objet `range(n)` qui fourni les entiers allant de 0 à $n - 1$.

— Le bloc d'instructions doit être indenté.

Le programme utilisé pour exposer la boucle conditionnelle peut s'écrire de la manière suivante :

```
In [21]: N = 8
         p = 1
         for c in range(N):
             p = p * 2
         print(p)
```

```
256
```

Exercice E8C1 : boucle for
Résoudre les exercices E6C1, E7C1 en utilisant inconditionnelle (*for*).

Problème à résoudre Étape la plus longue *(Origine : France-IOI)*
Dans votre petit carnet de voyage, vous avez noté la distance que vous avez parcourue chaque jour. Aujourd'hui, vous êtes particulièrement en forme et vous décidez donc de marcher plus que les jours précédents. Vous souhaitez utiliser un programme pour déterminer quel est votre record pour l'instant.

Ce que doit faire votre programme :
Votre programme doit d'abord lire un entier strictement positif, le nombre de jours de marche effectués jusqu'à présent. Il doit ensuite lire, pour chaque jour, la distance parcourue ce jour-là. Il doit alors afficher la distance maximale parcourue en une journée.

Exemple d'entrée

```
4
8
3
12
2
```

Sortie

```
12
```