Paradigmes de programmation

I./ Qu'est-ce qu'un paradigme de programmation :

Un *paradigme de programmation* est un modèle général pour décomposer les problèmes et composer des solutions. Il permet de concevoir des systèmes complexes à partir de briques élémentaires simples et que l'on peut composer.

La nature du paradigme de programmation utilisé dépend des propriétés de ses briques élémentaires. Le choix du paradigme se fait selon le type de problème que l'on veut résoudre. Certains langages de programmation incitent à l'utilisation d'un paradigme particulier ; d'autres langages permettent l'usages de plusieurs types de paradigmes. Python fait partie de ces langages.

Un même problème peut être résolu à l'aide de plusieurs paradigmes différents, et plusieurs paradigmes peuvent être utilisés dans un même programme.

Nous allons illustrer cette notion en décrivant trois paradigmes particuliers : le *paradigme impératif*, le *paradigme fonctionnel* et le *paradigme orienté objet*. Mais il en existe un très grand nombre d'autres, comme vous pourrez le voir sur la page suivante : https://fr.wikipedia.org/wiki/Paradigme_(programmation)

II./ Le paradigme impératif :

Dans le paradigme impératif, la brique élémentaire est *l'instruction*.

L'idée de base est que la machine possède un *état* (mémoire, registres, variables, …) et que chaque instruction modifie cet état. Les instructions se composent en *séquence*.

Quelques exemples de langages utilisant le paradigme impératif : **Fortran** (1954), **BASIC** (1964), **C** (1972), …

Dans ce paradigme, le programme est une suite d'instructions que l'ordinateur va exécuter les unes après les autres. Certaines instructions permettent de modifier la séquence : saut, tests, boucles…

Reprenons l'exemple de la suite de Fibonacci (vue dans le cours sur la récursivité) :

$F_0 = 0$	$F_1 = 1$	$F_n = F_{n-1} + F_{n-2}$ si $n > 1$.

Exemple de code en Fortran IV :

```
C     FIBONACCI SEQUENCE - FORTRAN IV
      NN=46
      DO 1 I=0,NN
    1 WRITE(*,300) I,IFIBO(I)
  300 FORMAT(1X,I2,1X,I10)
      END
C
      FUNCTION IFIBO(N)
      IF(N) 9,1,2
    1 IFN=0
      GOTO 9
    2 IF(N-1) 9,3,4
    3 IFN=1
      GOTO 9
    4 IFNM1=0
      IFN=1
      DO 5 I=2,N
      IFNM2=IFNM1
      IFNM1=IFN
    5 IFN=IFNM1+IFNM2
    9 IFIBO=IFN
      END
```

Exemple de code en BASIC :

```
10 INPUT N
20 LET A=0
30 LET B=1
40 FOR I=2 TO N
50 LET C=B
60 LET B=A+B
70 LET A=C
80 NEXT I
90 PRINT B
```

Exemple de code en C :

```
long long int fibb(int n) {
      int fnow = 0, fnext = 1, tempf;
      while(--n>0){
            tempf = fnow + fnext;
            fnow = fnext;
            fnext = tempf;
            }
            return fnext;
}
```

III./ Le paradigme fonctionnel :

Dans le paradigme fonctionnel, la brique élémentaire est la *fonction* ou *l'expression*.

Le programme est vu comme un ensemble de fonctions, qui peuvent être combinées pour former des expressions complexes, qui vont être appliquées à des données (nombres, listes, …). C'est par exemple le paradigme utilisé lorsqu'on réalise un programme récursif. Dans ce paradigme un critère important est la pureté : les fonctions doivent être pures, c-à-d que le résultat d'une fonction ne doit dépendre que de ses paramètres, et pas de facteurs externes.

Un exemple en Python montrant la différence entre fonction pure et fonction impure :

```
def fonction_pure(a):
    return a + 2

def fonction_impure(a):
    return a + B

B = 2   # B est une variable globale
print(fonction_pure(6))
print(fonction_impure(6))
print()
B = 5
print(fonction_pure(6))
print(fonction_impure(6))
```

Rappel : une *variable globale*, définie à l'extérieur des fonctions, est accessible de partout, c'est-à-dire à partir de n'importe quelle fonction. Une *variable locale*, par contre, définie à l'intérieur d'une fonction, n'est accessible qu'à l'intérieur de celle-ci. Après exécution du script Python précédent, si on tape **`print(B)`** dans la console, on obtiendra **`5`**. Mais si on tape **`print(a)`**, on obtiendra un message d'erreur, la variable **`a`** n'étant pas définie en dehors des fonctions **`fonction_pure`** et **`fonction_impure`**.

Quelques exemples de langages utilisant le paradigme fonctionnel : **Lisp** (1958), **Scheme** (1975), **Haskell** (1990), …

Exemple de code en Lisp :

```
(defun fibo (x)
 "
   Calcule le nombre de  fibonacci pour x
 "
     (if (<= x 2)
           1
     (+ (fibo (- x 2))(fibo (1- x)))))

     (loop for i from 1 to x do
          (print (fibo i)))
```

Exemple de code en Scheme :

```
(define fibo
 (lambda (x)
   (if (< x 2)
     x
     (+ (fibo (- x 1)) (fibo (- x 2))))))
```

Exemple de code en Haskell :

```
fib x =
  if x < 1
    then 0
    else if x < 2
           then 1
           else fib (x - 1) + fib (x - 2)
```

IV./ Le paradigme orienté objet :

Dans le paradigme orienté objet, la brique élémentaire est l'*objet* ou la *classe*.

Le programme est vu comme un ensemble d'objets, qui contiennent des données (attributs) et des fonctions (méthodes) qui permettent d'effectuer des traitements sur les données. Nous détaillerons plus en profondeur la programmation orienté objet dans un prochain chapitre.

Quelques exemples de langages utilisant le paradigme orienté objet : **Simula** (1967), **Smalltalk** (1972), **C++** (1983), **Java** (1995), …

Pour voir d'autres exemples de codes dans différents langages de programmation, voir le site Rosetta Code :

http://www.rosettacode.org/wiki/Rosetta_Code

BY NC SA