Chapitre 3

Pierre, papier, ciseaux

3.1. Nouveaux thèmes abordés dans ce chapitre

- procédures, fonctions
- paramètres
- variables globales et locales
- effets de bord
- types mutables

3.2. Règles du jeu

Tout le monde connaît le jeu pierre-papier-ciseaux, aussi connu sous le nom de « chifoumi ». Deux joueurs se montrent simultanément leur main qui symbolisera une pierre (poing fermé), un papier (main tendue) ou des ciseaux (l'index et le majeur forment un V).

La pierre bat les ciseaux, les ciseaux battent le papier et le papier bat la pierre. Si les deux joueurs jouent le même symbole, il y a égalité.

Nous allons dans un ce chapitre écrire une version textuelle du jeu. Nous verrons au chapitre 4 comment utiliser les images ci-dessus.

Exemple de partie

```
Pierre-papier-ciseaux. Le premier à 5 a gagné !
1 : pierre, 2 : papier, 3 : ciseaux ? 3
Vous montrez ciseaux - Je montre pierre
vous 0    moi 1
1 : pierre, 2 : papier, 3 : ciseaux ? 2
Vous montrez papier - Je montre papier
vous 0    moi 1
1 : pierre, 2 : papier, 3 : ciseaux ? 2
```

```
Vous montrez papier - Je montre papier
vous 0    moi 1
1 : pierre, 2 : papier, 3 : ciseaux ? 2
Vous montrez papier - Je montre ciseaux
vous 0    moi 2
1 : pierre, 2 : papier, 3 : ciseaux ? 1
Vous montrez pierre - Je montre ciseaux
vous 1    moi 2
1 : pierre, 2 : papier, 3 : ciseaux ? 1
Vous montrez pierre - Je montre ciseaux
vous 2    moi 2
1 : pierre, 2 : papier, 3 : ciseaux ? 1
Vous montrez pierre - Je montre papier
vous 2    moi 3
1 : pierre, 2 : papier, 3 : ciseaux ? 1
Vous montrez pierre - Je montre papier
vous 2    moi 4
1 : pierre, 2 : papier, 3 : ciseaux ? 3
Vous montrez ciseaux - Je montre papier
vous 3    moi 4
1 : pierre, 2 : papier, 3 : ciseaux ? 2
Vous montrez papier - Je montre ciseaux
vous 3    moi 5
```

Remarque : les nombres en gras ont été entrés au clavier par le joueur.

3.3. Code du programme

ppc.py

```
# jeu pierre, papier, ciseaux
# l'ordinateur joue au hasard

from random import randint

def ecrire(nombre):
    if nombre == 1:
        print("pierre",end=" ")
    elif nombre == 2:
        print("papier",end=" ")
    else:
        print("ciseaux",end=" ")

def augmenter_scores(mon_coup,ton_coup):
    global mon_score, ton_score
    if mon_coup == 1 and ton_coup == 2:
        ton_score += 1
    elif mon_coup == 2 and ton_coup == 1:
        mon_score += 1
    elif mon_coup == 1 and ton_coup == 3:
        mon_score += 1
    elif mon_coup == 3 and ton_coup == 1:
        ton_score += 1
    elif mon_coup == 3 and ton_coup == 2:
        mon_score += 1
    elif mon_coup == 2 and ton_coup == 3:
        ton_score += 1

ton_score = 0
mon_score = 0
fin = 5
print("Pierre-papier-ciseaux. Le premier à",fin,"a gagné !")
no_manche = 0
while mon_score < fin and ton_score < fin:
    ton_coup = int(input("1 : pierre, 2 : papier, 3 : ciseaux ? "))
    while ton_coup < 1 or ton_coup > 3:
        ton_coup =int(input("1 : pierre, 2 : papier, 3 : ciseaux ? "))
    print("Vous montrez",end=" ")
    ecrire(ton_coup)
    mon_coup = randint(1,3)
```

```
    print("- Je montre",end=" ")
    ecrire(mon_coup)
    print()     # aller à la ligne
    augmenter_scores(mon_coup,ton_coup)
    print("vous",ton_score,"   moi",mon_score)
```

3.4. Analyse du programme

Reprenons ce programme ligne par ligne pour l'expliquer en détails.

```
# jeu pierre, papier, ciseaux
# l'ordinateur joue au hasard
```

Il est toujours utile de décrire ce que fait le programme au début du code.

```
from random import randint
```

Revoici le module `random` d'où nous importons la fonction `randint(a,b)`, qui renvoie un nombre entier compris entre les bornes `a` et `b`, les bornes étant comprises.

3.4.1. Procédures

Quand les programmes deviennent plus longs, ils deviennent évidemment plus difficiles à comprendre. Aussi est-il fortement conseillé de « découper » le programme en sous-programmes, aussi appelés procédures ou fonctions (nous verrons la différence entre les deux un peu plus loin). Idéalement, chaque sous-programme devrait être visible entièrement sur l'écran pour faciliter sa lecture, et donc sa compréhension.

Les procédures sont souvent appelées plusieurs fois dans un programme, et elles ont la plupart du temps des paramètres. Le résultat de la procédure dépendra du ou des paramètres.

Voici une procédure permettant d'écrire à l'écran soit « pierre », soit « papier », soit « ciseaux ». Cela dépendra du paramètre `nombre`. Ainsi `ecrire(2)` affichera « papier ».

```
def ecrire(nombre):
    if nombre == 1:
        print("pierre",end=" ")
    elif nombre == 2:
        print("papier",end=" ")
    else:
        print("ciseaux",end=" ")
```

On remarquera un deuxième paramètre dans les procédures `print` : `end=" "`. Cela indique à Python qu'il faut continuer d'écrire sur la même ligne, après avoir inséré un espace.

La deuxième procédure du programme est chargée de mettre à jour les scores des joueurs en fonctions de leur coup.

```
def augmenter_scores(mon_coup,ton_coup):
    global mon_score, ton_score
    if mon_coup == 1 and ton_coup == 2:
        ton_score += 1
    elif mon_coup == 2 and ton_coup == 1:
        mon_score += 1
    elif mon_coup == 1 and ton_coup == 3:
        mon_score += 1
    elif mon_coup == 3 and ton_coup == 1:
        ton_score += 1
    elif mon_coup == 3 and ton_coup == 2:
        mon_score += 1
    elif mon_coup == 2 and ton_coup == 3:
        ton_score += 1
```

Les variables `mon_coup` et `ton_coup` sont globales. Cela signifie qu'elles ont été déclarées dans le programme principal, donc hors de la procédure. Dans un tel cas, la ligne

```
    global mon_score, ton_score
```

est indispensable, sous peine du message d'erreur UnboundLocalError: local variable 'mon_score' referenced before assignment.

Les variables `mon_coup` et `ton_coup` ont été passées en arguments. Il n'y a donc pas lieu de les déclarer comme globales.

3.4.2. Effets de bord

En informatique, une procédure est dite à effet de bord si elle modifie autre chose que sa valeur de retour. Par exemple, une procédure peut modifier une variable globale, ou modifier un ou plusieurs de ses arguments. Les effets de bord rendent souvent le comportement des programmes difficile à comprendre et sont à éviter dans la mesure du possible.

La procédure `augmenter_scores(mon_coup,ton_coup)` est à effet de bord puisqu'elle modifie les variables globales `mon_score` et `ton_score`.

Comment éviter les effets de bord ?

On pourrait, plutôt que de les déclarer comme variables globales, les passer comme arguments de la fonction et les modifier. Mais, en Python, les arguments ne peuvent être modifiés que s'ils sont d'un type mutable.

Types	Mutables
Chaîne de caractères	non
Valeur numérique	non
Liste	oui
Tuple	non
Dictionnaire	oui

Tableau 3.1 : Types mutables et non mutables

Pour le moment, nous n'avons vu que les deux premiers types du tableau 3.1.

Comme `mon_score` et `ton_score` sont des entiers, ils ne sont pas mutables et ne pourront pas être modifiés. Cela ne produira pas de message d'erreur, mais les deux scores resteront à 0.

Le paragraphe suivant montre une façon d'éliminer les effets de bord en utilisant des variables locales.

3.4.3. Fonction

La différence entre une fonction et une procédure, c'est qu'une fonction renvoie une (parfois plusieurs) valeurs. La valeur retournée est toujours placée après l'instruction `return`. Une fois cette instruction exécutée, le programme quitte immédiatement la fonction.

Transformons la procédure `augmenter_scores` en une fonction :

ppc2.py

```
def augmenter_scores(mon_coup,ton_coup,mon_score,ton_score):
    mon_nouveau_score = mon_score
    ton_nouveau_score = ton_score
    if mon_coup == 1 and ton_coup == 2:
        ton_nouveau_score += 1
    elif mon_coup == 2 and ton_coup == 1:
        mon_nouveau_score += 1
    elif mon_coup == 1 and ton_coup == 3:
        mon_nouveau_score += 1
    elif mon_coup == 3 and ton_coup == 1:
        ton_nouveau_score += 1
    elif mon_coup == 3 and ton_coup == 2:
        mon_nouveau_score += 1
    elif mon_coup == 2 and ton_coup == 3:
        ton_nouveau_score += 1
    return mon_nouveau_score,ton_nouveau_score
```

Cette fonction va renvoyer deux valeurs (`mon_nouveau_score` et `ton_nouveau_score`) qu'il faudra stocker dans des variables (dans notre exemple `mon_score` et `ton_score`). On appellera donc cette fonction par cette ligne :

```
mon_score,ton_score = augmenter_scores(mon_coup,ton_coup,mon_score,ton_score)
```

Comme les variables `mon_nouveau_score` et `ton_nouveau_score` sont déclarées dans la fonction `augmenter_scores`, elles seront locales à cette fonction et ne pourront pas être appelées depuis l'extérieur, contrairement aux variables globales. Si d'aventure il y a dans le code des variables de même nom, que ce soit dans le programme principal ou dans d'autres fonctions, il s'agira de variables différentes ! Pour reprendre l'image du chapitre 2, il y aura deux boîtes de même nom mais différentes, ayant probablement des contenus différents.

La durée de vie des variables locales est limitée. Elles n'existent que pendant l'exécution du sous-programme dans lequel elles se trouvent. Leur emplacement en mémoire n'est pas fixé à l'avance comme pour les variables globales mais un nouvel espace mémoire leur est alloué à chaque exécution du sous-programme. Cela a pour conséquence qu'elles ne conservent pas leur contenu d'une exécution à l'autre du sous-programme.

Exercice 3.1

Voici un exercice assez difficile, mais qui résume toutes les subtilités concernant les variables locales, les variables globales, les paramètres et les effets de bord.

Avertissement ! Les programmes ci-dessous sont mal écrits et ne sont surtout pas à prendre en exemple...

Vous remarquerez qu'ils se ressemblent beaucoup. Faites bien attention à tous les détails !

Que vont écrire les programmes suivants ?

Programme 1

```
def truc():
    n=10
    p=p+1

n, p = 5, 20   # on peut initialiser plusieurs variables sur une seule ligne
truc()
print(n,p)
```

Programme 2

```
def truc():
    global p
    n=10
    p=p+1

n, p = 5, 20
truc()
print(n,p)
```

Programme 3

```
def truc():
    global n, p
    n=10
    p=p+1

n, p = 5, 20
truc()
print(n,p)
```

Programme 4

```
def truc(p):
    n=10
    p=p+1

n, p = 5, 20
truc(p)
print(n,p)
```

Programme 5

```
def truc(p):
    global p
    n=10
    p=p+1

n, p = 5, 20
truc(p)
print(n,p)
```

Programme 6

```
def truc(p):
    global n
    n=10
    p=p+1

n, p = 5, 20
truc(p)
print(n,p)
```

Programme 7

```
def truc(n):
    global p
    n=10
    p=p+1

n, p = 5, 20
truc(p)
print(n,p)
```

Exercice 3.2

Réécrivez le programme de l'exercice 2.3 (partie 2), où il était question d'écrire les tables de multiplication de 2 à 12, mais en utilisant une procédure `table(n)` qui écrira la table de multiplication de `n`.

Exercice 3.3

Écrivez une fonction `maximum(a,b,c)` qui renvoie le plus grand des trois nombres `a`, `b`, `c` fournis en arguments.

Exercice 3.4

Écrivez un programme qui convertit des degrés Celsius en degrés Fahrenheit, et vice-versa. Définissez deux fonctions qui effectuent les conversions.

Formules : si F est la température en °F et C la température en °C, alors

$$F=\frac{9}{5}C+32 \qquad \text{et} \qquad C=\frac{5}{9}(F-32)$$

Exercice 3.5

D'après un sondage effectué auprès de 2000 personnes, un humain joue *pierre* dans 41% des cas, *papier* dans 30 % des cas, et *ciseaux* dans 29 % des cas. Sachant cela, on va tenter de rendre l'ordinateur plus « malin ».

Modifiez le code du § 3.3. Il faudra remplacer la ligne

```
mon_coup = randint(1,3)
```

par

```
mon_coup = coup_ordi()
```

et programmer la fonction `coup_ordi()` qui retournera *papier* dans 41 % des cas, *ciseaux* dans 30 % des cas et *pierre* dans 29 % des cas.

Exercice 3.6

Écrivez une fonction `f(x)` qui retourne la valeur de $f(x) = 2x^3 - 3x - 1$.

Écrivez une procédure `tabuler` avec quatre paramètres : `f`, `borneInf`, `borneSup` et `pas`. Cette procédure affichera les valeurs de la fonction `f`, pour x compris entre `borneInf` et `borneSup`, avec un incrément de `pas`.

Exemple : `borneInf` = –2.0, `borneSup` = 2.0 et `pas` = 0.5.

```
-2.0     -11.0
-1.5      -3.25
-1.0        0.0
-0.5        0.25
 0.0       -1.0
 0.5       -2.25
 1.0       -2.0
 1.5        1.25
 2.0        9.0
```

3.5. Ce que vous avez appris dans ce chapitre

- Les procédures et les fonctions permettent de découper le programme en sous-programmes. Cela permet d'augmenter la lisibilité et d'éviter d'écrire plusieurs fois la même chose.
- La différence entre une fonction et une procédure, c'est qu'une fonction renvoie une ou plusieurs valeurs (§ 3.4.3).
- Il existe des variables globales et des variables locales (§ 3.4.3).
- Les effets de bord sont à éviter, si possible (§ 3.4.2).
- Il existe des types mutables et d'autres non mutables (tableau 3.1). On ne peut pas modifier les valeurs de type non mutable.