CPES Scientifique

Informatique

TP 4

Tests et boucles

Résumé

Ce TP aborde quelques **structures de contrôle** élémentaires : les **instructions de branchement conditionnel** et les **boucles while**.

1 Instructions de branchement conditionnel

1.1 Description

Les instructions de branchement conditionnel permettent d'exécuter ou non certaines instructions en fonction du résultat d'un test. Il en existe plusieurs variantes.

1.1.1 Test simple

La syntaxe des **tests simples** est la suivante :

```
if (condition):
    # Instruction (I1)
    # Instruction (I2)
    ...
    # Instruction (In)
```

Dans cette premiere variante, les instructions $(I1), \ldots, (In)$ sont exécutées si l'expression « `condition` » s'évalue à `True`. Par exemple, si on définit la fonction suivante :

```
def f (x):
    res = 0
    if (x >= 0):
       res = 1
    return res
```

alors dans la console on a

```
>>> f(0)
1
>>> f(-1)
0
```

1.1.2 Test à alternative

On peut aussi donner une alternative :

```
if (condition):
    # Instructions (T1)
    # Instructions (T2)
    ...
    # Instructions (Tn)
else:
    # Instructions (F1)
    # Instructions (F2)
    ...
    # Instructions (Fm)
```

Dans cette variante, les instructions $(T1), \ldots, (Tn)$ sont exécutées si l'expression « `condition` » s'évalue à `True`. Sinon, les instructions $(F1), \ldots, (Fm)$ sont exécutées. Par exemple, avec

```
def test_if_else (x):
    if x >= 0:
        return True
    else:
        return False
```

on a, dans la console :

```
>>> test_if_else(0)
True
>>> test_if_else(-1)
False
```

Attention. Dans le test à alternative, l'expression `condition` n'est évaluée qu'une seule fois.

1.1.3 Tests imbriqués

Enfin, Python offre un moyen simple d'imbriquer des tests :

```
if (condition_1):
    # Instructions a executer si "condition_1" est vraie
elif (condition_2):
    # Instructions a executer si:
    #   "condition_1" est fausse ET "condition_2" est vraie
```

Les tests imbriqués peuvent être combinés avec des tests à alternative :

```
if (condition_1):
    # Instructions a executer si "condition_1" est vraie
elif (condition_2):
    # Instructions a executer si:
    #   "condition_1" est fausse ET "condition_2" est vraie
else:
    # Instructions a executer si:
    #   "condition_1" est fausse ET "condition_2" est fausse
```

Par exemple,avec

```
def test_if_elif (x):
    if x > 0:
```

```
        return 1
    elif x == 0:
        return 0
    else:
        return -1
```

on a, dans la console :

```
>>> test_if_elif(5)
1
>>> test_if_elif(0)
0
>>> test_if_elif(-10)
-1
```

1.2 Exercices

Question 1.1 (Calcul du maximum de deux nombres)**.** *Écrire et tester une fonction Python prennant deux arguments et renvoyant le maximum (c.-à-d. le plus grand) de ses arguments.*

Question 1.2 (Calcul du maximum de trois nombres)**.** *Même chose qu'à la question précédente, mais avec trois arguments.*

Question 1.3 (Calcul du maximum de trois nombres)**.** *Même chose qu'à la question précédente, mais cette fois-ci* ***sans utiliser d'instruction de branchement*** *(on pourra toutefois utiliser la fonction de la Question 1.1).*

2 Boucles « tant que »

2.1 Description

Les « **boucles tant-que** » (ou boucles « **while** ») permettent de répéter l'exécution d'instructions tant qu'une certaine condition est satisfaite.

La syntaxe est la suivante :

```
while (condition):
    # Instructions a executer
    # tant que "condition" est vraie
```

Par exemple, avec

```
1 def f (n):
2     i = n
3     res = 0
4     while (i != 0):
5         i = i - 1
6         res = res + 1
7     return res
```

on a

```
>>> f(0)
0
>>> f(1)
```

```
1
>>> f(2)
2
>>> f(3)
3
```

2.2 Exercices

Question 2.1. *On considère la fonction* `f` *ci-dessus.*

1. *Donner, pour* `n` $= 4$*, les valeurs des variables* `i` *et* `res` *à chaque passage à la ligne* 4 *du programme.*
2. *Que calcule* `f` *?*

Question 2.2. *Considérons la fonction* `g` *ci-dessous :*

```
1 def g(n) :
2   res = 0
3   x = n
4   while (x > 0) :
5     res = res + x
6     x = x-1
7   return res
```

1. *Donner, pour* `n` $= 4$*, les valeurs des variables* `x` *et* `res` *à chaque passage à la ligne* 4 *du programme.*
2. *Que calcule* `g` *?*

Question 2.3. *Considérons la fonction* `h` *ci-dessous :*

```
1 def h(n) :
2   res = 1
3   k = n
4   while (k > 0) :
5     res = res * k
6     k = k-1
7   return res
```

1. *Donner, pour* `n` $= 4$*, les valeurs des variables* `k` *et* `res` *à chaque passage à la ligne* 4 *du programme.*
2. *Que calcule* `h` *?*

Question 2.4. *Écrire et tester une fonction Python prennant en argument un entier* `n` *et renvoyant* $\sum_{k=0}^{\texttt{n}} 2k$*.*

Question 2.5. *Écrire et tester une fonction Python prennant en argument un entier* `n` *et renvoyant* $\sum_{k=0}^{\texttt{n}} k^2$*.*

Question 2.6 (Division Euclidienne)**.**

1. *Écrire et tester une fonction Python calculant le reste de la division euclidienne.*
2. *Écrire et tester une fonction Python calculant le quotient de la division euclidienne.*

Question 2.7 (Partie entière)**.** *Écrire et tester une fonction Python qui prend en argument un nombre à virgule flottante positif et qui renvoie sa partie entière.*