Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 2

Code UE : SPO1U57

Année 2023–2024

Premiers algorithmes formalisés

Les questions indiquées par des numéros entre parenthèses sont à faire à la maison.

Exercice 1 (variables et affectations)

```
1  def test1(a, b):
2      c = a - 2
3      b = c + a
4      a = b // c + 1
5      d = a + 5
6      e = a * d
7      return e
```

Considérez les deux algorithmes suivants :

```
1  def test2(a):
2      a = a * 2
3      d = a + d
4      d = e * (a + d)
5      b = d / 2
6      e = b - a
```

1. Analysez par une lecture attentive ces deux algorithmes et relevez leurs défauts s'ils en ont.
2. Déroulez l'exécution de `test1(-1, 3)`, de `test1(2, -3)` et de `test2(1)`. Vous présenterez ces trois exécutions sous la forme de tableaux dans lesquels chaque ligne notée l.k représente la mise à jour de la valeur d'une variables (`a`, `b`, `c`, `d` ou `e`, en colonnes), après l'exécution de l'instruction située à la ligne k.

Exercice 2 (Structures conditionnelles)

1. Soit l'algorithme `condition` suivant :

```
1  def condition(a, b, c, d, e):
2      if a < 0:
3          a = -a
4      if a == 0 or b > 0:
5          a = (b + d) * e
6          b = b // 2
7      c = b + a
8      if c != 0:
9          if d < 0:
10             e = a * b
11         else:
12             e = b // a
13     res = (a + b + c + d) / e
14     return res
```

   Déroulez l'exécution de `condition(-2, 5, 0, -3, 2)` en la présentant sous la forme d'un tableau permettant de suivre l'évolution des variables instruction par instruction. Qu'est-ce qui se trouve dans res à la fin de l'exécution ?

2. Écrivez une fonction qui prend en paramètre un entier n et qui retourne vrai si n est pair et faux sinon.
3. Écrire une fonction `division` qui prend en argument deux nombre entiers et qui retourne le résultat flottant de la division du premier par le second. Si le second argument est nul, la fonction retourne le flottant -1.

Exercice 3 (Structures itératives non bornées)

1. Soit l'algorithme `itération` suivant :

```
def itération(n):
    i = 0
    res = 0
    while i < n / 4:
        res = res + n - (i + 1)
        i = i + 1
    return res
```

Déroulez l'exécution de `itération(20)` en la présentant sous la forme d'un tableau permettant de suivre l'évolution des variables à chaque fin d'étape de l'itération sur `i`. Quelle est la valeur retournée ?

2. Écrivez une fonction `somme_entiers_impairs` utilisant une boucle `while` qui prend un entier n en paramètre et qui retourne la somme des n premiers entiers impairs.
3. Déroulez l'exécution de `somme_entiers_impairs(10)` en la présentant sous la forme d'un tableau permettant de suivre l'évolution des variables à chaque fin d'étape de l'itération. Quelle est la valeur retournée ?
4. Comptez le nombre d'instructions exécutées par l'appel de `somme_entiers_impairs(10)`. Généralisez ce résultat pour tout $n \in \mathbb{N}$.
5. Regardez attentivement le résultat de l'exécution de la question 3. Qu'observez-vous ? Imaginez un autre algorithme plus efficace qui, étant donné un entier n, retourne la somme des n premiers nombres impairs.

Lorsque l'on souhaite qu'un algorithme « interagisse » avec un utilisateur, on utilise généralement une fonction d'affichage `print(t)` qui permet d'afficher à l'écran le texte `t`. Par exemple, quand on appelle :

— `print('Il fait beau')`, le texte « Il fait beau » est écrit à l'écran (la sortie standard).
— `print('La valeur de x est ', x, ', et celle de y est ', y, '.')`, si l'on considère que les valeurs 4 et -1 sont respectivement affectées à `x` et `y` au moment de l'appel, le texte « La valeur de x est 4, et celle de y est -1. » est affiché à l'écran.

Exercice 4 (Structures itératives et appel de fonction) La suite de Syracuse associée à l'entier a, qu'on appelle altitude initiale, est l'unique suite définie par récurrence par $u_0 = a$ et pour tout entier naturel n :

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

Une conjecture célèbre affirme que, quelle que soit l'altitude a, la suite de Syracuse associée à a finit par rencontrer la valeur 1, puis continuer avec le cycle (1,4,2,1,...). On appelle *durée de vol* le plus petit entier n tel que $u_n = 1$. La conjecture de Syracuse (ou de Collatz) se résume donc à dire que la durée de tout vol est finie.

1. Écrire le code permettant de calculer et afficher à l'écran la valeur de u_{150} en supposant que $u_0 = 871$. *On utilisera une boucle* `for`.
2. Écrire une fonction calculant la durée de vol d'un entier donné en argument. *On utilisera une boucle* `while`.
3. Calculer la durée de vol maximale pour des altitudes initiales inférieures à 10^3. Même question jusqu'à 10^6 (sans utiliser la fonction `max` de la bibliothèque standard de Python). *On utilisera une boucle* `for` *utilisant la fonction de la question précédente.*

Exercice 5 Rappelons-nous qu'un nombre premier est un nombre entier supérieur ou égal à 2 qui n'est divisible que par 1 et par lui-même.

1. Écrire une fonction `est_premier` prenant en argument un entier n et qui renvoie un booléen qui est `vrai` si n est premier, `faux` sinon.
2. Écrire une fonction qui prend un entier i en argument et retourne le i-ième nombre premier, qu'on appellera p_i dans la suite, en s'aidant de la fonction précédente (sachant que 2 est le premier nombre premier, que 3 est le second, etc.) : lorsque $i > 1$, vous utiliserez une boucle `while` qui énumère tous les entiers impairs (en effet, inutile d'essayer les entiers pairs, puisque le seul premier pair est 2).

3. Jusqu'en 1536, on croyait que les nombres de Mersenne, c'est-à-dire les nombres de la forme $2^p - 1$, avec p premier, étaient premiers. À partir des fonctions écrites en réponse aux deux questions ci-dessus, écrire un algorithme qui montre l'invalidité de la conjecture d'avant 1536. L'algorithme devra écrire :
   — à partir de quel nombre premier p_m la conjecture est fausse (c'est-à-dire tel que $2^{p_m} - 1$ n'est pas premier) ;
   — à quel rang m se trouve p_m dans la liste des nombres premiers.