Introduction à la Programmation en Python
Partiel – **Durée : 2 heures**

Université Paris-Diderot – Samedi 26 Octobre 2019

— Aucun document ni aucune machine ne sont autorisés. Les téléphones doivent être rangés.
— Les exercices sont tous indépendants.
— Cet énoncé a deux pages.
— Une réponse peut utiliser les réponses attendues à une question précédente même si elle est non traitée.
— Les fragments de code doivent être correctement indentés.

Exercice 1.

1. Quelles sont les valeurs des variables `x`, `y` et `z` à la fin de l'exécution du programme suivant :

```
def f(x,y):
    return(x + 3)

x = 1
y = 2 * x + 1
for i in range(0, 3, 1):
    x = f(x, y)
if y > x:
    z = f(y, y)
else:
    z = f(y, f(x, y))
```

2. Quelles sont les valeurs des variables `x`, `y`, `z` et `nc` à la fin de l'exécution du programme suivant :

```
x = "abc"
y = "ABC"
z = ""
for i in range(0, len(x), 1):
    z = z + x[i] + y[i]
nc = 0
for i in range(0, len(z), 1):
    if z[i] == "c":
        nc = nc + 1
    elif z[i] == "C":
        nc = nc - 1
if nc > 0:
    x = y
else:
    y = x
```

Exercice 2. Dans les restaurants de la ville de Pythonpolis, le serveur apporte à la fin du repas la note avec le prix du repas. Pour déterminer la somme totale à régler, on doit ajouter 20 % du prix du repas pour le service, plus la taxe de la ville qui est calculée sur la base du prix du repas (on ne paye pas de taxe sur le service). Le taux de taxe est de 20 % pour un prix de repas strictement inférieur à 100, et de 10 % pour un repas coûtant 100 ou plus. Il est la coutume à Pythonpolis que les convives se partagent équitablement la somme à payer.

La monnaie de Pythonpolis est simplement comptée par des nombres entiers, il n'y a pas de centimes. On se permet d'arrondir les sommes vers le bas quand nécessaire.

1. Écrire une fonction `part_personne` qui prend en paramètre le prix du repas et le nombre de convives, et qui renvoie la somme que chacun des convives doit payer, service et taxe inclus.

   Par exemple, `part_personne(100, 4)` doit renvoyer 32 (car taxe de 10 %), et `part_personne(50, 2)` doit renvoyer 35 (car taxe de 20 %).
2. Écrire une procédure `print_part_personne` qui prend en paramètre le prix du repas et le nombre de convives, et qui affiche la chaîne de caractères "`chacun doit payer : `", suivi par la somme que chacun des convives doit payer, service et taxe inclus. Par exemple, un appel `print_part_personne(100, 4)` doit afficher :

   ```
   chacun doit payer : 32
   ```

Exercice 3.

1. Écrire une fonction `est_premier` qui prend un entier strictement positif n en paramètre, et qui renvoie `True` si n est un nombre premier, et `False` sinon. On rappelle qu'un nombre n est premier quand ses seuls diviseurs positifs sont 1 et n.
2. Écrire une procédure `premieres_etoiles` qui prend un entier strictement positif n en paramètre. Un appel `premieres_etoiles(n)` doit afficher une ligne de n symboles, tel que le i-ème symbole affiché est `*` quand i est un nombre premier, et `#` sinon. Par exemple, l'appel `premieres_etoiles(10)` doit afficher la ligne

   ```
   #**#*#*###
   ```

   car $2, 3, 5, 7$ sont premiers et $1, 4, 6, 8, 9, 10$ ne le sont pas.

Exercice 4.

1. Une chaîne de caractères est un *carré* quand sa longueur est paire, et quand la première moitié de la chaîne est égale à la deuxième moitié. Par exemple, `"abcabc"` est un carré, tandis que `"abcdef"` ne l'est pas. Écrire une fonction `est_carre` qui prend une chaîne w de caractères en paramètre, et qui renvoie `True` quand w est un carré, et `False` sinon.
2. Une chaîne de caractères est un *palindrome* quand elle se lit de gauche à droite exactement comme de droite à gauche. Par exemple, `"abcba"` et `"abba"` sont des palindromes, tandis que `"abbc"` ne l'est pas. Écrire une fonction `est_palindrome` qui prend une chaîne w de caractères en paramètre, et qui renvoie `True` quand w est un palindrome, et `False` sinon.

Exercice 5. Le but de cet exercice est de trier des listes d'entiers en ordre croissant.

1. Écrire une procédure `echanger_cases(lis, i, j)` qui prend une liste `lis` d'entiers et deux entiers `i`, `j` en paramètre. Vous pouvez supposer que `i` et `j` sont des indices de la liste `lis`.

   Un appel `echanger_cases(lis, i, j)` doit modifier la liste `lis` en échangeant les contenus des cases aux positions `i` et `j`.

   Par exemple, si `lis` vaut `[2, 0, 4, 1]`, alors après un appel `echanger_cases(lis, 0, 1)`, `lis` vaut `[0, 2, 4, 1]`.
2. Écrire une fonction `index_min_from` qui prend une liste `lis` d'entiers et un entier `i` en paramètre. Vous pouvez supposer que `i` est un index de la liste de `lis`. Un appel `index_min_from(lis, i)` doit renvoyer l'index de la valeur minimale parmi tous les éléments de `lis` à partir de la position `i` (incluse). Par exemple, si `lis` vaut `[2, 0, 4, 1]`, alors `index_min_from(lis, 1)` renvoie 1 et `index_min_from(lis, 2)` renvoie 3.
3. Écrire une procédure `trier_croissant` qui prend une liste `lis` d'entiers en paramètre et qui la trie en place, c'est à dire qui modifie la liste `lis` en mettant ses éléments en ordre croissant. Vous devez vous servir des fonctions des deux questions précédentes, comme suit : on détermine l'index d'un élément minimal de `lis` à partir de la position 0 et on l'échange avec l'élément à la position 0, puis on détermine l'index d'un élément minimal de `lis` à partir de la position 1 et on l'échange avec l'élément à la position 1, et ainsi de suite. Par exemple, si `lis` vaut `[2, 0, 4, 1]`, alors après un appel `trier_croissant(lis)`, `lis` vaut `[0, 1, 2, 4]`.