+1/1/60+

Programmation impérative en Python — SPUF21

Année 2024-2025 — Seconde session

Nom : .....

Prénom : .....

Numéro d'étudiant : .....

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

Durée : 2 heures.

Aucun document n'est autorisé. L'usage de la calculatrice ou de tout autre appareil électronique est interdit.

Les exercices sont indépendants. Au sein d'un même exercice, vous pouvez utiliser les variables et fonctions des questions précédentes, même si vous n'avez pas su les faire ; chaque question est donc indépendante.

À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncé vous conseille l'utilisation de certaines d'entre elles.

```python
# Fonctions autorisées
range(…) len(…)
print(…) int(…)

# Méthodes
L.append(x)
```

```python
# Par exemple les méthodes et fonctions suivantes sont entre autres interdites
max(…) min(…) sum(…) abs(…) eval(…)
s.split(…) s.index(…) L.extend(…)

# Vous n'avez pas le droit d'utiliser des compréhensions ou des slices
# À la place vous devez utiliser des boucles.
x in L
[ x for x in range(L) ]
chaine[début:fin:pas]
```

Exercice 1 Calculs binaires . . . . . 4 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4

1. Écrivez une fonction `affiche_calcul_binaire(n)` qui affiche le calcul de la représentation binaire de n, de sorte que l'on peut lire *verticalement* (de bas en haut) la représentation en binaire de n. Par exemple, pour $13 = (1101)_2$, on obtiendra dans la console :

```
>>> affiche_calcul_binaire(13)
1 car 13 = 2 × 6 + 1
0 car 6 = 2 × 3 + 0
1 car 3 = 2 × 1 + 1
1
>>> affiche_calcul_binaire(14)
0 car 14 = 2 × 7 + 0
1 car 7 = 2 × 3 + 1
1 car 3 = 2 × 1 + 1
1
```

```
def affiche_calcul_binaire(n) :
    while n >= 2:
        q = n//2
        r = n%2
        print(r , 'car' , n , '=' , '2 ×' , q ,'+', r )
        n = q
    print(n)
```

2. Écrivez une fonction **récursive** `binaire(n)` qui renvoie l'écriture binaire de n sous forme de chaîne dans le sens de lecture **usuel**. On pourra utiliser la fonction `str` (rappel : `str(0)` vaut `"0"`).

```
>>> binaire(2)
'10'
>>> binaire(13)
'1101'
>>> binaire(14)
'1110'
```

```
def binaire(n) :
    if n<=1 :
        return str(n)
    else:
        return binaire(n//2)+str(n%2)
```

Exercice 2 Des formules et des arbres . . . . . 5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5

Dans cet exercice nous allons travailler avec des arbres. Pour manipuler ces derniers, vous ne pouvez utiliser que les fonctions suivantes :
— `arbre(r,Ag,Ad)` renvoie un arbre de racine `r` et de fils `Ag` (gauche) et `Ad` (droit) ;
— `est_feuille(A)` renvoie `True` si `A` est une feuille, et `False` sinon ;
— `racine(A)` renvoie la racine de l'arbre `A` : `'+'`, `'-'`, `'*'` ou `'/'` ;
— `fg(A)` renvoie le fils gauche de `A` ;
— `fd(A)` renvoie le fils droit de `A`.

1. Avec les fonctions ci-dessus, définir une variable `F` correspondant à la formule $\frac{x+y}{2}$ représentée sous forme d'arbre.

```
F = arbre('/', arbre('+', 'x', 'y'), 2)
```

2. Écrire une fonction `contient(v,arbre)` qui renvoie `True` si et seulement si une feuille de l'arbre est égale à `v`. Sinon, la fonction renverra `False`.

```
>>> contient('x',F)
True
>>> contient('a',F)
False
>>> contient(5,F)
False
>>> contient(2,F)
True
```

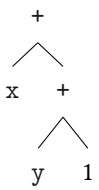
```
def contient(x, arbre):
    if est_feuille(arbre):
        return x == arbre
    else:
        return contient(x, fg(arbre)) or contient(x, fd(arbre))
```

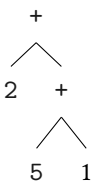
3. On appelle environnement une liste de couples `(nom, valeur)`.
Écrire une fonction `variable(env, nom)` qui renvoie la valeur associée à `nom` dans la liste `env`. Si la variable n'est pas dans la liste on renverra 0. Si le nom apparaît plusieurs fois on renverra la première occurence.

```
>>> env = [ ("a",3), ("x",2) , ("y",5), ("x", 4)]
>>> variable(env,"x")
2
>>> variable(env,"b")
0
```

```
def variable(env, x):
    for couple in env:
        (nom,valeur) = couple
        if nom==x:
            return valeur
    return 0
```

4. Écrire une fonction `remplacement(env,arbre)` qui remplace chaque variable de l'arbre par sa valeur dans l'environnement `env`. Par exemple avec l'environnement `env = [("a",3), ("x",2), ("y",5)]`, et l'arbre de gauche en paramètre, la fonction devra renvoyer l'arbre de droite.






```
def remplacer(env, arbre):
    if est_feuille(arbre):
        if type(arbre) == int:
            return arbre
        else:
            return variable(arbre, env)
    else:
        r = racine(arbre)
        g = remplacer(fg(arbre), env)
        d = remplacer(fd(arbre), env)
        return arbre(r,g,d)
```

Exercice 3 Collections . . . . . 6 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5 ☐5,5 ☐6

Dans cet exercice, on travaille sur des collections avec multiplicités. Par exemple si on se donne les valeurs suivantes : « A A D A B D A », on les stockera sous la forme d'une liste de tuples indiquant pour chaque valeur combien de fois elle apparaît : `[("A",4), ("B",1), ("D",2)]`. L'ordre des élements dans la liste n'a pas d'importance.

1. Écrire une fonction `nouvelle_collection()` qui renvoie une collection vide.

```
def nouvelle_collection():
    return []
```

2. Écrire une fonction `lire(x,C)` qui renvoie la multiplicité associée à la valeur `x`. Si `x` n'est pas présent dans `C`, on renverra 0.

```
>>> C = [("A",4), ("B",1), ("D",2)]
>>> lire("B",C)
1
>>> lire("U",C)
0
```

```
def lire(x,C):
    for (y,n) in C:
        if x==y:
            return n
    return 0
```

3. Écrire une fonction `ajout(x,n,C)` qui modifie la collection `C` en ajoutant `n` occurences à `x`.

```
>>> C
[('A', 4), ('B', 1), ('D', 2)]
>>> ajout("D",10,C)
>>> C
[('A', 4), ('B', 1), ('D', 12)]
>>> ajout("E",3,C)
>>> C
[('A', 4), ('B', 1), ('D', 12), ('E', 3)]
>>> ajout("A",-4,C)
>>> C
[('A', 0), ('B', 1), ('D', 12), ('E', 3)]
```

```python
def ajout(x,n,C):
    for i in range(len(C)):
        (y,m) = C[i]
        if x == y:
            C[i] = (y, n+m)
            return
    C.append((x,n))
```

4. Écrire une fonction `ramasse_miette(C)` qui renvoie une nouvelle collection sans les éléments de multiplicité nulle. La fonction affichera aussi le nombre de valeur supprimées.

```
>>> C = [("A",4), ("B",0), ("D",0), ("E",-3)]
>>> C = ramasse_miette(C)
2 valeur(s) supprimée(s)
>>> C
[('A', 4), ('E', -3)]
>>> C = ramasse_miette(C)
0 valeur(s) supprimée(s)
>>> C
[('A', 4), ('E', -3)]
```

```python
def rammasse_miette(C):
    R = []
    s = 0
    for (x,n) in C:
        if n!=0:
            R.append((x,n))
        else:
            s = s+1
    print(f"{s} valeurs supprimées")
    return R
```

5. Écrire une fonction `plus_nombreux(C)` qui renvoie la valeur la plus présente dans la collection `C`.

```
>>> C = [("A",-5), ("B",4), ("D",2)]
>>> plus_nombreux(C)
'B'
```

```
def plus_nombreux(C):
    m = None
    r = None
    for (x,n) in E:
        if m==None or m<n:
            m = n
            r = x
    return r
```

Exercice 4 Recherche de texte . . . . . 5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5

L'objectif de cet exercice est de rechercher une chaîne de caractères dans un texte fourni sous forme de chaîne ou de fichier.

1. Écrire une fonction `présent(chaîne, texte, i)` qui regarde si à l'indice `i` du texte se trouve la chaîne en question. On fera attention de renvoyer `False` en cas d'impossibilité triviale (si l'indice `i` est trop grand par rapport à la longueur du texte, si `chaîne` est plus longue que le texte, etc).

```
>>> présent("thon", "on", 10)
False
>>> présent("thon", "python", 0)
False
>>> présent("thon", "python", 2)
True
```

```
def présent(chaîne, texte, i):
    if i + len(chaîne) > len(texte):
        return False
    for j in range(len(chaîne)):
        if chaîne[j] != texte[i+j]:
            return False
    return True
```

2. Écrire une fonction `rechercher(chaîne, texte)` qui renvoie le nombre d'apparitions de la chaîne dans le texte. On pourra évidemment utiliser la fonction précédente.

```
>>> rechercher("toto", "to")
0
>>> rechercher("thon", "Anthony le python dépasse un thon lors du marathon")
4
>>> rechercher("outou", "Groutoutou")
2
```

```
def rechercher(chaîne, texte):
    n=0
    for i in range(len(texte)):
        if présent(chaîne, texte, i):
            n = n+1
    return n
```

3. On se donne un fichier texte `nerval.txt` contenant un poème et représenté ci-contre. Écrire une fonction `début_ligne(chaîne, nom_fichier)` qui affiche toute les lignes commençant par `chaîne` parmi les lignes du fichier dont le nom est en paramètre.

```
>>> début_ligne("Un", "nerval.txt")
Un air très vieux, languissant et funèbre,
Un coteau vert, que le couchant jaunit,
>>> début_ligne("Choucroute", "nerval.txt")
>>> début_ligne("Il", "nerval.txt")
Il est un air pour qui je donnerais
```

```
Fantaisie (de Gérard de Nerval)

Il est un air pour qui je donnerais
Tout Rossini, tout Mozart et tout Weber,
Un air très vieux, languissant et funèbre,
Qui pour moi seul a des charmes secrets.

Or, chaque fois que je viens à l'entendre,
De deux cents ans mon âme rajeunit :
C'est sous Louis treize ; et je crois voir s'étendre
Un coteau vert, que le couchant jaunit,

Puis un château de brique à coins de pierre,
Aux vitraux teints de rougeâtres couleurs,
Ceint de grands parcs, avec une rivière
Baignant ses pieds, qui coule entre des fleurs ;

Puis une dame, à sa haute fenêtre,
Blonde aux yeux noirs, en ses habits anciens,
Que, dans une autre existence peut-être,
J'ai déjà vue… - et dont je me souviens !
```

```
def début_ligne(chaîne, nom_fichier):
    fichier = open(nom_fichier, "r", encoding="utf-8")
    for ligne in fichier:
        if présent(chaîne, ligne, 0):
            print(ligne,end="")
    fichier.close()
```