+1/1/60+

# Programmation impérative en Python — SPUF21

## Année 2023-2024 — Examen terminal

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]
```

+1/2/59+

**Exercice 1** Convertir en flottant . . . . . 7 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5 ☐5,5 ☐6 ☐6,5 ☐7

1. Écrire une fonction `français_vers_python(chaîne)` qui prend en argument une chaîne de caractères représentant un nombre au format français (avec virgule) et qui renvoie une nouvelle chaîne au format informatique où la virgule a été remplacée par un point. La fonction doit fonctionner même lorsque la chaîne ne représente pas un flottant valide. On rappelle que la méthode `replace` est interdite.

```
>>> français_vers_python("6,55957")
'6.55957'
>>> français_vers_python("3,14")
'3.14'
>>> français_vers_python("un,deux,troix")
'un.deux.troix'
```

```
def français_vers_python(chaîne):
    rés = ""
    for c in chaîne:
        if c == ",":
            rés = rés + "."
        else:
            rés = rés + c
    return rés
```

2. Écrire une fonction `chiffre_vers_nombre` qui convertit un chiffre (un caractère) en nombre.

```
>>> chiffre_vers_nombre('9')
9
>>> chiffre_vers_nombre('0')
0
```

```
def chiffre_vers_nombre(chiffre):
    return ord(chiffre) - ord('0')
```

3. Écrire une fonction `découper_chiffres(chaîne)` qui renvoie un couple de listes. La première liste correspondant aux chiffres (sous forme de nombres et non de caractères) de la partie entière (à gauche du point) ; la seconde liste correspondant aux chiffres de la partie décimale (à droite du point).

```
>>> découper_chiffres("153.123")
([1, 5, 3], [1, 2, 3])
>>> découper_chiffres("3.14159")
([3], [1, 4, 1, 5, 9])
>>> découper_chiffres("18")
([1, 8], [])
```

On rappelle que vous ne pouvez **pas** utiliser la méthode `split`.

```
def découper_chiffre(chaîne):
    L1 = []
    L2 = []
    partie_entière = True
    for c  in chaîne:
        if c == ".":
            partie_entière = False
        elif partie_entière:
            L1.append(chiffre_vers_nombre(c))
        else:
            L2.append(chiffre_vers_nombre(c))
    return (L1,L2)
```

4. Écrire une fonction `liste_vers_entier(liste)` qui transforme une liste de nombres entre 0 et 9 en un unique nombre. *Indice, pour ajouter* 3 *à la fin du nombre* 12*, il suffit de faire* $12 \times 10 + 3 \rightarrow 123$

```
>>> liste_vers_entier([1,2,3])
123
>>> liste_vers_entier([])
0
>>> liste_vers_entier([0,5,0,1,7])
5017
```

```
def liste_vers_entier(liste):
    n = 0
    for chiffre in liste:
        n = 10*n + chiffre
    return n
```

5. À partir de ces fonctions, en déduire une fonction `chaîne_vers_flottant` qui à partir d'une chaîne représentant un nombre à virgule renvoie le flottant correspondant. On n'utilisera évidemment pas la fonction `float`.

```
>>> chaîne_vers_flottant("123,456")
123.456
```

```
def chaîne_vers_flottant(chaîne):
    L1,L2 = découper_chiffres(français_vers_python(chaîne))
    n1 = liste_vers_entier(L1)
    n2 = liste_vers_entier(L2)
    return n1 + n2/10**len(L2)
```

**Exercice 2** Parcoursup et dictionnaire . . . . . 5 points

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

Dans cet exercice, on cherche à traiter des dossiers parcoursup. Pour cela chaque dossier étudiant est représenté par un tuple contenant, 1) le numéro de dossier, 2) l'année de passage du bac, 3) une suite de notes 4) le type de bac (générale, professionnelle, scientifique, etc.).

```
>>> alice = ("655957", "23/24", [18 , 9.5],    "Gén.")
>>> bob =   ("766068", "22/23", [8, 7, 11, 2], "Pro.")
```

1. Écrire une fonction `accepter(étudiant)` qui prend en argument un tuple représentant un dossier étudiant et renvoie `True` si la moyenne des notes est supérieure ou égale à 10. Sinon (ou dans le cas où la liste de notes est vide) on renverra `False`.

```
>>> accepter(alice) # Alice a 13.75 de moyenne
True
>>> accepter(bob) # Bob a 7 de moyenne
False
```

```
def accepter(étudiant):
    notes = étudiant[2]
    if notes == []:
        return False
    somme = 0
    for n in notes:
        somme = somme + n
    return somme/len(notes) >= 10
```

2. Écrire une fonction `incrémenter(dico, k)` qui modifie le dictionnaire donné en paramètre en ajoutant 1 à la valeur associée à la clé `k`. Si la clé n'est pas présente, on l'ajoute au dictionnaire en lui associant la valeur 1.

```
>>> dico = {"k1":20, "k2":31}
>>> incrémenter(dico, "k1")
>>> dico
{'k1': 21, 'k2': 31}
>>> incrémenter(dico, "k3")
>>> dico
{'k1': 21, 'k2': 31, 'k3': 1}
```

```
def incrémenter(dico, clé):
    if clé in dico:
        dico[clé] = dico[clé] + 1
        else:
        dico[clé] = 1
```

3. On souhaite maintenant faire des statistiques. Écrire une fonction `dico_stat(liste_étudiants)` qui à partir d'une liste de dossiers renvoie un dictionnaire qui indique, pour chaque année et par formation, le nombre d'étudiants. On générera la clé à partir de l'année et du type de bac. Par exemple Alice et Dalida correspondront à la clé `"23/24-Gén."` et Bob à la clé `"22/23-Pro."`.

```
>>> charles = ("877179", "22/23",[],"Gén.")
>>> dalida  = ("988280", "23/24",[20],"Gén.")
>>> elyes   = ("099390", "20/21",[17,18], "Sci.")
>>> liste =[ alice, bob, charles, dalida, elyes]
>>> stat = dico_stat(liste)
>>> stat
{'23/24-Gén.': 2, '22/23-Pro.': 1, '22/23-Gén.': 1, '20/21-Sci.': 1}
```

```
def dico_stat(liste_étudiants):
    dico = dict()
    for étudiant in liste_étudiants:
        (numéro, année, notes, formation) = étudiant
        incrémenter(dico, année+"-"+formation))
    return dico
```



**Exercice 3** Fichiers textes . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

1. On se donne une liste d'étudiants au même format que dans l'exercice précédent. Écrire une fonction `bilan(fichier,liste)` qui stocke les résultats dans un fichier texte dont le nom est donné en argument. On pourra utiliser la fonction `accepter` de l'exercice précédent et chaque ligne du fichier ne contiendra que le numéro de l'étudiant séparé par une virgule du résultat « OUI » ou « NON ». Par exemple :

```
>>> bilan("résultats.txt",[alice, bob])
```

créera le fichier `résultats.txt` avec comme contenu :

```
655957,OUI
766068,NON
```

```
def bilan(liste_étudiants, nom_fichier):
    fichier = open(nom_fichier,'w', encoding = "utf-8")
    for étudiant in liste_étudiants:
        (numéro, années, notes, formation) = étudiant
        if accepter(étudiant):
            fichier.write(f"{numéro},OUI\n")
        else:
            fichier.write(f"{numéro},NON\n")
    fichier.close()
```

2. Écrire une fonction `rechercher_étudiant(numéro, fichier)` qui affiche toutes les lignes du fichier contenant le numéro donné en paramètre. On admet que chaque numéro est codé sur 6 chiffres.

```
>>> rechercher_étudiant("766068", "résultats.txt")
766068,NON
```

```
def rechercher_étudiant(numéro, fichier):
    fichier = open(nom_fichier,'r', encoding = "utf-8")
    for ligne in fichier:
        égals  = True
        for i in range(6):
            if ligne[i] != numéro[i]:
                égals = False
        if égals:
            print(ligne)
    fichier.close()
```



**Exercice 4** 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,F)` qui renvoie `True` si et seulement si une feuille de l'arbre `F` 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
```

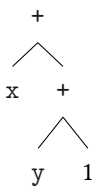
```
def contient(x, F):
    if est_feuille(F):
        return x == F
    else:
        return contient(x, fg(F)) or contient(x, fd(F))
```

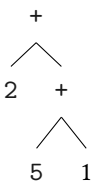
3. On appelle environnement une liste de couples `(nom, valeur)`.
Écrire une fonction `variable(env, nom)` qui renvoie la valeur associée à `nom` dans la liste `env`. Si la variable n'est pas dans la liste on renverra 0. Si le nom apparaît plusieurs fois on renverra la première occurence.

```
>>> 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)
```