Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2018-2019**

Partiel

2018

(durée 2h - sans document)

Répondez directement sur le sujet.
Si de la place manque, vous pouvez joindre une de vos feuilles.

Numéro (**reporter ici le numéro inscrit sur votre copie cachetée**) : _____

Groupe de TD no : _____

Attention : *Dans vos programmes Python, si vous utilisez des fonctions ou méthodes, seules celles présentées en cours sont autorisées.*

Exercice 1 [2 points]

1. *Cochez la ou les bonne(s) réponse(s)*

   (a) Comment lire une valeur entière au clavier et l'affecter à `x` ?

   - ☐ `int(input(x))`
   - ☐ `x = input()`
   - ☐ `int(input()) = x`
   - ■ **`x = int(input())`**

   (b) Quelle(s) instruction(s) n'est (ne sont) pas valide(s), lorsque `ch="abcd"` ?

   - ☐ `print(ch)`
   - ☐ `ch = ch[:5]`
   - ■ **`ch[0]='A'`**
   - ☐ `ch = input(ch)`

   (c) Dans le bloc d'instructions d'une fonction, l'instruction `return` permet :

   - ☐ De ré-exécuter la fonction
   - ■ **D'interrompre l'exécution du bloc d'instructions**
   - ■ **De renvoyer une valeur**
   - ☐ D'afficher une valeur
   - ☐ De terminer le programme

   (d) Quelle(s) est (sont) l'expression(s) équivalente(s) à la négation de l'expression suivante ?

   ```
   NON(a >= 2) OU (b == 1)
   ```

   - ■ **`(a >= 2) ET (b != 1)`**
   - ☐ `(a >= 2) OU (b == 1)`
   - ☐ `(a < 2) ET (b != 1)`
   - ■ **`NON(NON(a >= 2) OU (b == 1))`**

2. Transformer la boucle `POUR` ci-dessous en boucle `TANT_QUE` équivalente :

```
POUR i ALLANT DE 10 A 20
    DEBUT
        ECRIRE i
    FIN
```

Solution :

```
i <- 10
TANT_QUE i <= 20 FAIRE
    DEBUT
        ECRIRE i
        i <- i + 1
    FIN
```

3. Qu'affiche le programme suivant ?

```
x = 'Partiel algorithmique et Python'
if 'Python' in x: print("Langage")
elif 'Partiel' in x: print("Evaluation")
else: print("Math")
```

Solution :
La condition du premier `if` est vérifiée, le programme affiche donc `Langage` et rien d'autre car les `elif` ne sont pas testés.

4. Dans la fonction `print()`, que permettent de faire les paramètres `sep` et `end` ? Quelle est leur valeur par défaut ?

Solution :
`sep` définit ce qui sépare l'affichage des différents arguments d'un même `print()` ; sa valeur par défaut est une espace.
`end` définit ce qui est ajouté à l'affichage après avoir affiché tous les arguments du `print()` ; sa valeur par défaut est un retour à la ligne.

5. Quelle est la différence entre un algorithme et un programme ?

Solution :
un algorithme qui est écrit en pseudo-code, proche du langage naturel, n'est pas exécutable par un ordinateur, alors qu'un programme, écrit dans un langage informatique, avec une syntaxe spécifique, est interprétable ou exécutable par un ordinateur.

Exercice 2 [3,5 points]

On souhaite réaliser un **programme Python** permettant d'afficher une table d'addition de la forme :

```
Saisir un entier inférieur ou égal à 10 :7
 0+0 0+1 0+2 0+3 0+4 0+5 0+6 0+7
     1+0 1+1 1+2 1+3 1+4 1+5 1+6
         2+0 2+1 2+2 2+3 2+4 2+5
             3+0 3+1 3+2 3+3 3+4
                 4+0 4+1 4+2 4+3
                     5+0 5+1 5+2
                         6+0 6+1
                             7+0
```

Pour cela, on a écrit **en Python** le programme suivant :

```
1  n = 11
2  while n > 10:
3      n = input("Saisir un entier inferieur ou egal a 10: ")
4  while i <= n
5      print("    "*i, sep="")
6      j = i
7      ch = ""
8      while j <= n:
9          ch + " " + str(i) + "+" + (j-i)
10     print(ch)
11 i = i+1
```

Ce programme comporte **8 erreurs**. Corriger ces erreurs avec un stylo de couleur directement sur le programme ci-dessus : utiliser l'espacement entre les lignes pour mettre une version corrigée d'une ligne comportant une (ou des) erreur(s) et/ou ajouter d'éventuelles instructions manquantes.

Solution :

```
n = 11
while n > 10:
    n = int(input("Saisir un entier inférieur à 10: "))
        # 3 : manque le int
i = 0   # 4: il faut initialiser i
while i <= n:   # 4 : ajouter : après le while
    print("    "*i, end="")   # 5 : end plutôt que sep
    j = i
    ch = ""
    while j <= n:
           # 9 : il faut ajouter str devant (j-i)
           #   et mettre ch= au début de l'instruction
        ch = ch + " " + str(i) + "+" + str(j-i)
        j = j+1   # 10 : il faut ajouter j=j+1
    print (ch)
    i = i+1   # 11 : incr. à mettre dans le 1er while
```

Exercice 3 [7 points]

Un administrateur [1] d'un site web veut assurer un maximum de sécurité pour les utilisateurs du site. Pour cela, il décide de réaliser un programme Python qui évalue la force des mots de passe des différents utilisateurs du site, sachant qu'un mot de passe est une chaîne de caractères qui ne comporte ni espaces ni lettres accentuées.

La force d'un mot de passe varie, selon la valeur d'un score, de 'Très faible' jusqu'à 'Très fort' :

— si le score <20, la force du mot de passe est 'Très faible' ;
— sinon si le score<40, la force d'un mot de passe est 'Faible' ;
— sinon si le score <80, la force du mot de passe est 'Fort' ;
— sinon la force du mot de passe est 'Très fort'.

Le score se calcule en additionnant des bonus et en retranchant un malus.

Les bonus attribués sont :

— Nombre total de caractères * 4
— (Nombre total de caractères – nombre de lettres majuscules) * 2
— (Nombre total de caractères – nombre de lettres minuscules) * 3
— Nombre de caractères non alphabétiques * 5

Le malus se calcule en additionnant :

— La longueur de la plus longue séquence de lettres minuscules * 2 et
— La longueur de la plus longue séquence de lettres majuscules * 3

Pour cela l'administrateur a commencé par programmer 2 fonctions :

```
def minuscules():
    return "abcdefghijklmnopqrstuvwxyz"

def majuscules():
    return "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Écrire **en Python** :

1. Une fonction, appelée `nbLettresMin()`, qui prend en argument une chaîne de caractères et renvoie le nombre total de lettres minuscules contenues dans la chaîne. Cette fonction doit faire appel à la fonction `minuscules()`.

   Par exemple, l'appel de `nbLettresMin("PL1_promo2018")` doit renvoyer `5`.

   Solution :

   ```
   def nbLettresMin(ch):
       nbMin = 0
       i = 0
       while i < len(ch):
           if ch[i] in minuscules(): nbMin += 1
           i += 1
       return nbMin
   ```

2. Une fonction, appelée `nbCaraNonAlpha()`, qui prend en argument une chaîne de caractères et renvoie le nombre total de caractères non alphabétiques (qui ne sont ni des lettres

1. Repris et adapté de
http://www.developpement-informatique.com/cours/1ère-année-prépas-scientifiques-(Sup)/94/Exercices-corrigés-Python-(Série-6)

minuscules, ni des lettres majuscules) contenus dans la chaîne. Cette fonction doit faire appel aux fonctions `minuscules()` et `majuscules()`.

Par exemple, l'appel de `nbCaraNonAlpha("PL1_promo2018")` doit renvoyer `6`.

Solution :

```
def nbCaraNonAlpha(ch):
    nb = 0
    i = 0
    while i < len(ch):
        if ch[i] not in majuscules()
           and ch[i] not in minuscules(): nb += 1
        i += 1
    return nb
```

3. Une fonction, appelée `bonus()`, qui prend en argument une chaîne de caractères et renvoie la valeur du bonus, calculé comme indiqué dans l'énoncé page 5. Cette fonction doit faire appel aux fonctions `nbLettresMin()` et `nbCaraNonAlpha()` et à une fonction `nbLettresMaj()` qu'on supposera existante qui prend en argument une chaîne de caractères et renvoie le nombre de lettres majuscules contenues dans la chaîne.

   Par exemple, l'appel de `bonus("PL1_promo2018")` doit renvoyer `128`, c'est-à-dire le résultat de $13 * 4 + (13 - 2) * 2 + (13 - 5) * 3 + 6 * 5$.

Solution :

```
def bonus(ch):
    return len(ch)*4 + ((len(ch) - nbLettresMaj(ch))*2) +
       ((len(ch) - nbLettresMin(ch))*3) +
       (nbCaraNonAlpha(ch)*5)
```

4. Une fonction, appelée `plusLongueSequenceMin()`, qui prend en argument une chaîne de caractères et renvoie le nombre de caractères de la plus longue séquence de minuscules. Cette fonction doit faire appel à la fonction `minuscules()`.

   Par exemple, l'appel de `plusLongueSequenceMin("ab2018toto")` doit renvoyer `4`.

Solution :

```
def plusLongueSequenceMin(ch):
    seqCourante = 0
    plusLongueSeq = 0
    i = 0
    while i < len(ch)):
        while (i < len(ch) and ch[i] in minuscules()):
            seqCourante += 1
            i += 1
        if seqCourante > plusLongueSeq:
```

```
                plusLongueSeq = seqCourante
            seqCourante = 0
            i += 1
        return plusLongueSeq
```

5. Une fonction, appelée `malus()`, qui prend en argument une chaîne de caractères et renvoie la valeur du malus, calculé comme indiqué dans l'énoncé page 5. Cette fonction doit faire appel à la fonction `plusLongueSequenceMin()` et à une fonction qu'on supposera existante, `plusLongueSequenceMaj()`, qui prend en argument une chaîne de caractères et renvoie le nombre de caractères de la plus longue séquence de majuscules contenues dans la chaîne.
   Par exemple l'appel de `malus("PL1_promo2018")` doit renvoyer `16`, c'est-à-dire le résultat de $5 * 2 + 2 * 3$.

Solution :

```
def malus(ch):
    return plusLongueSequenceMin(ch)*2
         + plusLongueSequenceMaj(ch)*3
```

6. Un programme principal qui demande à l'utilisateur de saisir une chaîne de caractères et qui affiche la force du mot de passe saisi, en faisant appel aux fonctions `bonus()` et `malus()`.
   Par exemple, si l'utilisateur saisit la chaîne `"PL1_promo2018"`, le programme doit afficher :
   `Le mot de passe a un score de 112 il est donc 'Tres fort'`

Solution :

```
mp = input("Mot de passe :")
score  =  bonus(mp) - malus(mp)
print("Le mot de passe a un score de",score,
      "il est donc ", end="")
if score < 20: print("'Tres faible'")
elif score < 40: print("'Faible'")
elif score < 80: print("'Fort'")
else: print("'Tres fort'")
```

Exercice 4 [7,5 points]

Un entier naturel n est appelé un nombre **puissant** lorsque, pour tout diviseur premier p de n, p^2 divise n. L'objectif de cet exercice est d'écrire un algorithme en pseudo-code permettant de tester si une valeur n saisie au clavier par un utilisateur est un nombre puissant. Par exemple, le nombre 72 est un nombre puissant : ses deux diviseurs premiers sont 2 et 3 et, 72 est également divisible par 4 et 9. Par contre, 63 n'est pas un nombre puissant car 7 est un diviseur premier de 63 mais 63 n'est pas divisible par 49.

1. Écrire, **en pseudo-code**, une fonction, appelée `premier()`, qui prend un entier e en argument et renvoie un booléen qui vaut `Vrai` si e est premier et `Faux` sinon.

Solution :

```
FONCTION premier(e TYPE NOMBRE) TYPE BOOLEEN
    VARIABLES_LOCALES
        d TYPE NOMBRE
        result TYPE BOOLEEN
    DEBUT
        d <- 2
        resul <- VRAI
        TANT_QUE (resul == VRAI ET d < e) FAIRE
            DEBUT
                SI (e%d == 0) ALORS
                    DEBUT
                        resul <- FAUX
                    FIN
                SINON
                    DEBUT
                        d <- d+1
                    FIN
            FIN
        RENVOYER resul
    FIN
```

2. Écrire, **en pseudo-code**, une fonction, appelée `diviseurPremier()`, qui prend un entier n en argument et renvoie un tableau de booléens. Dans ce tableau, si la valeur booléenne stockée à l'indice i vaut `Vrai`, cela signifie que i est un diviseur premier de n ; à l'inverse, si la valeur booléenne stockée à l'indice i vaut `Faux`, cela signifie que i n'est pas un diviseur premier de n. Cette fonction fera appel à la fonction `premier()`.

Solution :

```
FONCTION diviseurPremier(n TYPE NOMBRE)
    TYPE TABLEAU DE BOOLEEN
    VARIABLES_LOCALES
        i TYPE NOMBRE
        tab TYPE TABLEAU DE NOMBRE
    DEBUT
        tab <- CREER_TABLEAU(n+1)  # il y a aussi n
        tab[0] <- FAUX
        tab[1] <- FAUX   # 1 n'est pas premier
        i <- 2
        TANT_QUE (i <= n) FAIRE
            DEBUT
                SI (n%i == 0 ET premier(i)) ALORS
                    DEBUT
                        tab[i] <- VRAI
                    FIN
                SINON
                    DEBUT
                        tab[i] <- FAUX
                    FIN
                i <- i+1
            FIN
        RENVOYER tab
    FIN
```

3. Écrire, **en pseudo-code**, le programme principal qui lit une valeur n, appelle la fonction `diviseurPremier()` pour connaître les diviseurs premiers de n et indique si n est puissant en affichant le résultat à l'écran. Si n est puissant, il faut afficher la liste de ses diviseurs premiers. Si n n'est pas puissant, il faut afficher un diviseur premier d qui est tel que n n'est pas divisible par d^2.

Solution :

```
ALGO
VARIABLES
    i, j, n TYPE NOMBRE
    puissant TYPE BOOLEEN
    tab TYPE TABLEAU DE NOMBRE
DEBUT
    LIRE n
    puissant <- VRAI
    tab <- diviseurPremier(n)
    i <- 2
    TANT_QUE (puissant ET i <= n) FAIRE
        DEBUT
            SI (tab[i] ET n%(i*i) != 0) ALORS
                DEBUT
                    puissant <-FAUX
                FIN
            i <- i+1
        FIN
    SI (puissant) ALORS
        DEBUT
            POUR j ALLANT_DE 2 A n
                DEBUT
                    SI (tab[j]) ALORS
                        DEBUT
                            ECRIRE j
                        FIN
                FIN
            ECRIRE "le nombre", n, "est puissant"
        FIN
    SINON
        DEBUT
            ECRIRE "diviseur premier de ", n, ":", i-1
            ECRIRE "le nombre", n, "n'est pas puissant"
        FIN
FIN
```