Intro à l'algorithmique et la programmation | **Note: /25**
Pearson - IUT Villetaneuse - Dpt. STID
Contrôle - 13 octobre 2020 **Nom:** _____
(Durée: 1 h. 30 min.) **Prénom:** _____

- Le barème est susceptible de changer.
- Aucune correspondance, aucun appareil électronique n'est autorisé durant l'examen.
- Les seuls documents autorisés sont les fiches résumés distribuées avec le cours sans annotations.
- Vous répondrez aux questions dans les espaces disposés à cet effet.
- La note tiendra compte de la concision, de l'efficacité et de l'expressivité du code ainsi que de la présentation.
- Toute sortie est définitive.

1. (5 points) **Test**

   (a) (2 points) Écrire une fonction `element_1( borne_min, borne_max, val)` prenant en paramètre 3 valeurs numériques et qui *affiche* la valeur `val` si celle-ci est comprise strictement dans l'interval `[ borne_min; borne_max]` et n'affiche rien sinon. On suppose que les valeurs fournies `borne_min` et `borne_max` sont telles que $borne\_min \leq boorne\_max$.

   Solution:

```
1 def element_1( borne_min, borne_max, val) :
2         if ( val > borne_min and val < borne_max ) :
3                 print( val )
```

   (b) (2 points) Écrire une fonction `element_2( borne1, borne2, val)` qui doit proposer le même résultat que `element_1` mais tenir compte du fait que les bornes de l'interval peuvent ne pas être donnés dans l'ordre (par exemple `borne1` peut être supérieur ou inférieur à `borne2`). Pour avoir tous les points vous devez utiliser la fonction `element_1` à bon escient dans la fonction `element_2`.

   Solution:

```
1 def element_2( borne1, borne2, val) :
2         if ( borne_min <= borne_max ) :
3                 element( borne_min, borne_max, val)
4         else :
5                 element( borne_max, borne_min, val)
```

   (c) (0.5 points) Proposez un programme principal permettant de mettre en évidence que la fonction `element_2` fonctionne comme attendu au moyen de 4 exemples d'appels testant différents aspects de la fonction.

   Solution:

```
1 element_2( -2, 2, -3)
2 element_2( -2, 2, -2)
3 element_2( -2, 2, 2)
4 element_2( -2, 2, 3)
```

2. (3 points) **Validation de semestres**
   Le règlement de passage et de validation d'un semestre dans un IUT imaginaire est le suivant.

— Si un étudiant a moins de 8 de moyenne générale au semestre courant, son semestre n'est pas validé.
— S'il a entre 8 (inclus) et 10 (exclus) alors il est recevable pour le semestre courant.
— S'il a au moins 10 de moyenne générale il valide son semestre courant.

De plus s'il est recevable, il peut compenser avec le semestre précédent si la moyenne des moyennes des 2 semestres est supérieure ou égale à 10. Dans ce cas, son semestre courant est validé.

(a) (2 points) Écrire une fonction `jury( semCourant, semPrecedent)` prenant en paramètre les moyennes du semestre courant `semCourant` et du précédent semestre `semPrecedent` et qui à partir de ces moyennes *affiche* si le semestre courant est validé.

Solution:

```
1 def jury( semCourant,  semPrecedent) :
2         moyenne = ( semCourant + semPrecedent ) / 2.0
3         if ( semCourant >= 10 ) :
4                 print( "Votre semestre est valide" )
5         else if ( semCourant >= 8 and moyenne >=10) :
6                 print( "votre semestre est valide par compensation" )
7         else :
8                 print( 'Vous ne validez pas votre semestre courant' )
```

(b) (0.5 points) Écrire un programme appelant 4 fois la fonction `jury` avec des valeurs différentes permettant de tester tous les cas de figure.

Solution:

```
1 jury( 12, 9 )
2 jury(9,11);
3 jury(9,9);
4 jury(7,13);
```

3. (3 points) **Définition**

(a) (0.5 points) Donnez la défintion d'une *affectation* :

Solution: Une affectation constite à associer une valeur à un nom de variable.

(b) (0.5 points) Donnez la défintion d'une *itération* :

Solution: Une itération correspond à une exécution du bloc d'instruction d'une boule.

(c) (0.5 points) Comment un programme principal et une fonction échangent-ils des valeurs :

Solution: Le programme transmet une/des valeur(s) à une fonction par les paramètres de la fonction. La fonction transmet une valeur au programme principale par la valeur de retour de la fonction.

(d) (1 point) Donnez la table de vérité de l'opérateur logique ET (and)

Solution:

ET	True	False
True	True	False
False	False	False

(e) (1 point) Donnez la table de vérité de l'opérateur logique OU (or)

Solution:

OU	True	False
True	True	True
False	True	False

(f) (0.5 points) Donnez la table de vérité de l'opérateur logique NON (not)

Solution:

p1	NON(p1)
True	False
False	True

4. (6 points) **Puissances de un**

   (a) (1 point) Écrire une fonction `saisieControlee()` qui ne prend pas de paramètre mais *retourne* un entier saisi par l'utilsateur compris entre 1 inclus et 9 inclus. La fonction redemandera un entier jusqu'à ce que la valeur fournie par l'utilisateur soit valide. Le message d'invite sera `"Saisir la valeur de n"` à chaque fois.

Solution:

```
1 def saisieControlee() :
2         val = int( print("Saisir la valeur de n"))
3         while (val<1 or val >9):
4                 val = int( print("Saisir la valeur de n"))
5         return val
```

Soit $\mathbb{1}$ l'ensemble des nombres en base 10 qui ne s'écrivent qu'avec le chiffre 1 et dont le nombre de digits (le nombre de chiffres) est inférieur à 9 inclus. On a :
$\mathbb{1} = \{1, 11, 111, 1111, ..., 111111111\}$.
On note $1_{..n}$ l'élément de $\mathbb{1}$ contenant exactement n digits. Ainsi on a :
— $1_{..2} = 11$
— $1_{..3} = 111$
— $1_{..5} = 11111$

   (b) (2.5 points) Écrire une fonction `uns( nb_de_1 )` qui prend en paramètre un entier et *retourne* l'entier correspondant à la valeur de $1_{..n}$.
   Plusieurs stratégies peuvent être mises en oeuvre pour répondre à cette question :
   — en construisant une chaîne de caractère et en la castant en entier,
   — en utilisant un calcul arithmétique.

Solution:

```
1 def uns( nb_de_1 ) :
2         cpt = 0
3         val = 0
4         while (cpt < n)  :
5                 val = 10 * val + 1 ;
6                 cpt += 1
7         return val
```

```
1 def uns( nb_de_1 ) :
2         cpt = 0
3         val = 0
4         digit = 1
5         while (cpt < nb_de_1 )  :
6                 val +=  digit
7                 digit *= 10
8                 cpt += 1
9         return val
```

```
1 def uns( nb_de_1 ) :
2         cpt = 0
3         val = ""
4         while (cpt < nb_de_1 )  :
```

```
5                   val += "1"
6                   cpt += 1
7           return int(val)
```

Un palindrome est une série de caractères dont l'ordre est le même que la série soit lue de droite à gauche ou de gauche à droite. Par exemple :
— K A Y A K
— engage le jeu que je le gagne
— 1279721
Tous les éléments de $\mathbb{1}$ admettent pour carré un palindrome particulier de la forme :
$1^2_{..n} = 123...n...321$
Ainsi on a :

$(1_{..1})^2 =$ $(1)^2 =$ 1
$(1_{..2})^2 =$ $(11)^2 =$ 121
$(1_{..3})^2 =$ $(111)^2 =$ 12321
$(1_{..7})^2 =$ $(1111111)^2 =$ 1234567654321

(c) (1.5 points) Écrire une fonction `palindromeNumerique( n )` qui prend en paramètre un entier compris entre 1 inclus et 9 inclus (le contrôle de la valeur passée en paramètre est réalisé avant l'appel de la fonction) et qui *affiche* la valeur de $(1_{..n})^2$ **sans utiliser la multiplication ni la fonction d'élévation à la puissance**.

Solution:

```
1  def palindromeNumerique( n ) :
2          chaine = ""
3          cpt = 0 ;
4          while( cpt < n ) :
5                  cpt += 1
6                  chaine += str( cpt )
7          while( ctp > 1 )  :
8                  cpt -= 1
9                  chaine += str( cpt )
10         print( chaine )
```

(d) (1 point) Écrire un programme principal qui utilise les fonctions précédemment développées qui demande une valeur n à l'utilisateur et affiche le carré du nombre contenant exactement n 1. En guise de spécification du programme voici une trace d'exécution :

```
1 Saisir la valeur de n:
2 12
3 Saisir la valeur de n:
4 4
5 Resultat:
6 1111^2=1234321
```

Solution:

```
1 n = saisieControlee()
2 print( str( uns(n) ) )
```

Tot:

```
3 palindromeNumerique( n )
```

5. (3 points) **Visibilité d'une variable**
   On considère le code suivant.

```
1  def tmp( i ) :
2          while ( i < 4 ) :
3                  print( "bonjour" )
4                  i += 1
5          return 2 * i
6
7  i = 0
8  j = 0
9  while ( i < 7 ) :
10         tmp( i )
11         i += 1
12
13 j = tmp( i )
```

   (a) (1 point) Combien de fois la fonction `tmp` est-elle appelée ?
   ☐ 2 ☐ 3 ☐ 7 √ **8** ☐ 9

   (b) (1 point) Combien de fois s'affiche `bonjour` ?
   ☐ 3 ☐ 4 ☐ 8 √ **10** ☐ 14

   (c) (1 point) Combien vaut `j` à la fin du programme ?
   ☐ 0 ☐ 6 ☐ 8 √ **14** ☐ 16

Espace supplémentaire