A/ Soit la fonction suivante :

```
fun sayHello(): Unit {
    val nom: String = "Hello !"
    for (c: Char in nom) println(c)
}
```

A.1 Donner ci-dessous la sortie réalisée lors de l'appel de cette fonction.

H
e
l
l
o

!

(ne pas oublié l'espace entre le o et !)

A.2 Que signifie *Unit* dans l'interface de la fonction ?

Comme les variables, une fonction est toujours typée. Unit est un type spécial qui signale que la fonction ne retourne aucune valeur.

Unit est la valeur par défaut de retour des fonctions

Par exemple :

```
fun main() { }
```

est équivalent à

```
fun main() : Unit { }
```

(Remarque : question prématurée, ne rentre pas dans le barème de notation)

==========

F/ Voici un extrait (doc java) de la documentation de la méthode next() classe Scanner

next

`public String next()`

Finds and returns the next complete token from this scanner. A complete token is preceded and followed by input that matches the delimiter pattern. This method may block while waiting for input to scan, even if a previous invocation of `hasNext()` returned `true`.

Soit la fonction suivante :

```
fun main() {
    val input = "1  2 red  blue"
    val s: Scanner = Scanner(input)
    val a = s.nextInt()
    val b = s.nextInt()
    val c = s.next()
    val d = s.next()
    println("$a\n$b\n$c\n$d")
}
```

Après compilation, l'exécution produit le résultat suivant :

```
1
2
red
blue
```

F1/ Expliquez le résultat produit.

L'appel successif aux méthodes *nextInt* et *next* permet de « consommer » les tokens (les mots) de la donnée d'entrée (input). Le délimiteur de token est par défaut représenté par une zone composée d' « espaces blancs » (tout caractères inférieur ou égal au caractère espace)

L'affichage en ligne est provoqué par le caractère \n inséré en argument de *println*.

F2/ Soit la fonction suivante:

```
fun main() {
    val input = "1  2 red"
    val s: Scanner = Scanner(input)
    val a = s.nextInt()
    val b = s.nextInt()
    val c = s.next()
    val d = s.next()
    println("$a\n$b\n$c\n$d")
}
```

F2/ Expliquez le résultat produit.

D'après la documentation technique fournie (voir question précédente), **le programme sera bloqué** par l'instruction val d = s.next(), en attente d'un token qui ne viendra jamais...

Donc pas de sortie à afficher.

B/ Soit la fonction suivante :

```
fun main() {
    val scanner = Scanner(System.`in`)
}
```

B.1/ Quel est le type de la variable scanner ?

justifiez votre réponse : Scanner

Le type de la variable est déduit du constructeur utilisé lors de l'initialisation. Comme dans un bon nombre de langages, le constructeur porte le même nom que sa classe.

(Remarque : question prématurée, ne rentre pas dans le barème de notation)

B.2/ L'instruction en ligne 3 est-elle valide ?

```
1 :  fun main() {
2 :      val scanner = Scanner(System.`in`)
3 :      scanner = 1+2
4 :  }
```

justifiez votre réponse :

Non car la variable est déclarée val, qui n'accepte qu'une seule opération d'affectation (initialisation). Elle est ensuite en lecture seule.

De plus, le type de la valeur à affecter (1+2), soit 3, est du type Int et non Scanner.

==========

C/ Soit la fonction

```
fun main(){
    val nom: String = "Hello "

    nom = nom + 1
    println(nom)
}
```

Ce code provoque une erreur de compilation.

Déterminer pourquoi

Non car la variable est déclarée val, qui n'accepte qu'une seule opération d'affectation (initialisation). Elle est ensuite en lecture seule.

==========

D/ Soit la fonction

```
fun main(){
    val x: String = "4"

    println(x + 2.toString())
}
```

Ce code provoque-il une erreur de compilation ou une valeur lors de son exécution ?

justifiez votre réponse :

Il n'y a pas d'erreur. La valeur passée à *println* est le résultat de la concaténation de "4" avec "2".

Affiche : 42

==========

E/ Soit la fonction suivante :

```
    fun main() {
        val scanner = Scanner(System.`in`)
    }
```

Dans cet extrait de code, **System.`in`** est

[x] Un argument
[ ] Un paramètre
[x] Une valeur
[ ] Une variable

Rappel : un **argument** est une **valeur** affectée à un **paramètre**

Cette valeur peut provenir directement d'une variable, d'une constane ou plus généralement d'une expression évaluée pour l'occasion.

==========

Comment déclarer une variable de type chaîne de caractères en Kotlin ? (Elian)

[ ] variable s = "my string"
[x] var s = "my string"
[ ] String s = "my string"
[x] var s: String = "my string"

==========

Comment se nomme la première ligne de cette methode ? : (d'après Alexandre)

```
fun salaireEmploye(nbHeure: Int , prixHeure: Int):  Double{
   return ... // code qui calcul le salaire (non montré ici)
}
```

[x] Interface
[ ] Body
[ ] Appel de méthode
[x] Entête
[ ] Corps de la méthode

==========

D’après cette fonction

```
fun salaireEmploye(nbHeure: Int , prixHeure: Int):  Double{
   return ... // code qui calcul le salaire (non montré ici)
}
```

[x] nbHeure est une variable de portée locale
[ ] nbHeure est une variable de portée au-delà de la fonction
[x] nbHeure est un paramètre
[ ] nbHeure est une constant
[ ] nbHeure est de type String

==========

D’après cette fonction

```
39 : fun salaireEmploye(nbHeure: Int , prixHeure: Int):  Double{
40 :   val prime = 42
41 :   return ... // code qui calcul le salaire (non montré ici)
42 : }
```

[x] la ligne 40 déclare une variable locale
[x] la ligne 40 affecte une variable
[x] la ligne 40 initialise une variable
[x] la ligne 40 définit une variable