21PNSI

Exercice 1

1) $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1$

Donc 1000_2 vaut 8_{10}

$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$

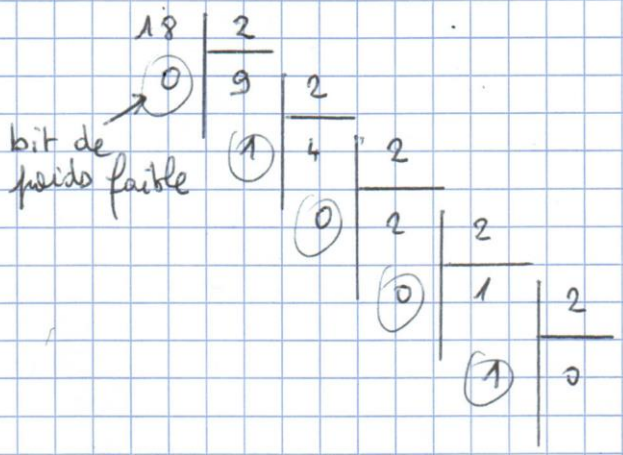
Donc 10011_2 vaut $16 + 2 + 1$ soit 19_{10}

$1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 32 + 4 + 1$

Donc 100101_2 vaut 37_{10}

2) $4_{10} = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$ Donc 4_{10} vaut 100_2

$12_{10} = 1 \times 2^3 + \times 2^2 + 0 \times 2^1 + 0 \times 2^0$ Donc 12_{10} vaut 1100_2



Donc 18_{10} vaut 10010_2

3) a) Compléter le fichier HTML : voir la feuille de l'énoncé ci-après

b) Compléter le fichier JavaScript : voir la feuille de l'énoncé ci-après

- script.js

```
function leibniz(){
    var s = .0...;  // Initialisation de la variable qui cumule la somme des bits.
     if (document.getElementById("b0").checked){s = s + 1}
     if (document.getElementById("b1").checked){s = s + 2}
     if (document.getElementById("b2").checked){S.=.S.+.4..}
     if (document.getElementById("b3").checked){S.=.S.+.8..}
     if (document.getElementById("b4").checked){S.=.S.+16.}
     if (document.getElementById("b5").checked){S.=.S.+32.}
     if (document.getElementById("b6").checked){S.=.S.+.64.}
     if (document.getElementById("b7").checked){S.=.S+128.}
     document.getElementById(".Numbae....").innerHTML = s
}
```

- binaire.html

```
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Convertisseur binaire</title>
  <link href="style.css..." type="text/css" rel="stylesheet">
</head>
<body>
  <h3>Convertisseur binaire vers d&eacutecimal &agrave 8 bits</h3>
  <label>Cochez :</label>
  <form autocomplete="off"> <!-- Pour effacer les cases lors du rafraichissement -->
  <div class="ligne">
    <input type="checkbox" id="b7" onChange="...leibniz()...">
    <label for="b7"><var>2<sup>7</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b6" onChange="...leibniz()...">
    <label for="b6"><var>2<sup>6</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b5" onChange="...leibniz()...">
    <label for="b5"><var>2<sup>5</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b4" onChange="...leibniz()...">
    <label for="b4"><var>2<sup>4</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b3" onChange="...leibniz()...">
    <label for="b3"><var>2<sup>3</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b2" onChange="...leibniz()...">
    <label for="b2"><var>2<sup>2</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b1" onChange="...leibniz()...">
    <label for="b1"><var>2<sup>1</sup></var></label>
  </div>
  <div class="ligne">
    <input type="checkbox" id="b0" onChange="...leibniz()...">
    <label for="b0"><var>2<sup>0</sup></var></label>
  </div>
  </form>
  <p>Le nombre repr&eacutesent&eacute en binaire ci-dessus est
  <code id="nombre">0</code>.</p>
  <script src="...script.js..."></script>
</body>
</html>
```

Exercice 2

1) Algorithme : tri_insertion(tableau)

pour i allant de 1 à longueur(tableau) - 1

j ← i

x ← tableau[j]

tant que j > 0 et tableau[j-1] > x

tableau[j] ← tableau[j-1]

j ← j-1

tableau[j] ← x

renvoyer tableau

2)
```
def tri_insertion(tableau_a_trier):
    for i in range(1, len(tableau_a_trier)):
        j = i
        x = tableau_a_trier[j]
        while j > 0 and tableau_a_trier[j-1] > x :
            tableau_a_trier[j] = tableau_a_trier[j-1]
            j = j-1
        tableau_a_trier[j] = x
    return tableau_a_trier
```

3) a)
```
def tri_insertion(liste):
    # Tri par insertion
    tableau_a_trier = list(liste)  # Crée une copie de liste
    for i in range(1, len(tableau_a_trier)):
        j = i
        x = tableau_a_trier[j]
        while j > 0 and tableau_a_trier[j-1][0] < x[0]:
            tableau_a_trier[j] = tableau_a_trier[j-1]
            j = j-1
        tableau_a_trier[j] = x
    return tableau_a_trier
```

3) b)
```
def plus_hauts1(liste_triee, n):
    liste_des_n_plus_hauts = liste_triee[:n]
    return liste_des_n_plus_hauts
```

3) c)
```
def plus_hauts2(liste_triee, altitude):
    L = [x for x in liste_triee if int(x[0]) >= altitude]
    return L
```

Exercice 3

1) a) La présence du $ montre que Fredo est un simple utilisateur

b) Le ~ (tilde) montre qu'on est dans le repertoire personnel de l'utilisateur donc il s'agit de Fredo.

2) a) oui la commande est correcte.

b) Il a indiqué un chemin absolu puisque le chemin commence par la racine "/".

3) Pour aller dans le répertoire Exos depuis le répertoire Fredo on peut utiliser :

/home/Fredo/NSI/Exos (chemin absolu)

ou

NSI/Exos (chemin relatif)

4) Pour aller dans le répertoire DS depuis le répertoire Fiches on peut utiliser :

/home/Fredo/Maths/DS (chemin absolu)

ou

../DS (chemin relatif)

5) Un chemin relatif peut être plus rapide à écrire.

6) Un chemin absolu est correct depuis n'importe quel répertoire courant.

7) Fredo saisit la commande
mkdir Projets

8) Fredo saisit la commande
mkdir ../Anglais ou bien mkdir /home/Fredo/Anglais

9) cd ../Anglais ou bien cd /home/Fredo/Anglais

10) a) mv Activités Exos

b) Tout le contenu du repertoire Activités est ainsi déplacé.

11) a) Un message apparait lui disant qu'il n'est pas possible de supprimer un répertoire.

b) Fredo doit consulter le manuel de la commande rm en saisissant man rm.
Il verra qu'il doit utiliser l'attribut -R pour supprimer un répertoire.
Ainsi il doit saisir rm -R Maths (ou rm -r Maths)

Exercice 4

1)

Instructions données en langage machine (ici le microprocesseur 6502)	Instructions symboliques en langage assembleur (nom de l'instruction et de l'opérande)	Description de l'action réalisée
A9	LDA # $0B_{16}$	Charge $0B_{16}$ dans le registre A
0B		
69	ADC # 15_{16}	Ajoute 15_{16} au contenu du registre A.
15		
69	ADC # $0A_{16}$	Ajoute $0A_{16}$ au contenu du registre A
0A		

2) Ce programme réalise l'addition

$$\begin{array}{r} 0\,B \\ +\ 1\,5 \\ +\ 0\,A \end{array}$$

et stocke le résultat dans le registre A.

3) A la fin de l'exécution :

$0B_{16} = 11_{10}$

$15_{16} = 16 + 5 = 21_{10}$

$0A_{16} = 10_{10}$

donc l'addition est

$$\begin{array}{r} 11 \\ +\ 21 \\ +\ 10 \\ \hline 42_{10} \end{array}$$

42_{10} est écrit en base 16 dans l'accumulateur A

bit de poids faible → (10) = A

42 | 16
(10) | 2
2 | 16
(2) | 0

Donc l'accumulateur à la fin contient **2A**