1ère Partie : Représentation des données : types et valeurs de base

Question 1 :

Pour trouver l'entier positif codé en base 2 sur 8 bits par le code 0010 1010, on affecte à chacun des bits de droite à gauche une puissance de 2 croissante en commençant par 2^0, ainsi on calcule :
$0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 0 \cdot 2^7$ soit : $1 \cdot 2^1 + 1 \cdot 2^3 + 1 \cdot 2^5 = 2 + 8 + 32 = 42$.

Question 2 :

```
x=1
for i in range(10):
   x=x*2
print(x)
```

La boucle multiplie x par 2 dix fois de suite avec une valeur de départ de x égale à 1.
x=1
A l'issue de la boucle n°1 : x=2
A l'issue de la boucle n°2 : x=2*2=2**2
….
A l'issue de la boucle n°10 : x=2**10=(2**8)*(2**2)=256*4=1024

Question 3 :

L'entier relatif codé en complément à 2 sur un octet par le code 1111 1111 est -1 : résultat que l'on peut connaître.
Rappel :
Le premier bit de gauche indique que l'entier est négatif.
Le code est celui de 2^8-1=255
On peut aussi faire le complément à 2 de 1111 1111 : 0000 0000 et ajouter 1

Question 4 :

On rappelle que de la même manière qu'il n'est pas possible de représenter 1/3 de manière exacte en base 10, on ne peut pas représenter 0.1 de manière exacte en base 2.
Aussi le programme :

```
x=1.0
while x!=0.0:
   x=x-0.1
```

ne s'arrêtera jamais car on ne tombera jamais exactement sur 0.

Question 5 :

Il s'agit ici de comparer les tables de vérité

A	B	not A	(not A) or B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

(A and B) or (not A or B) or (not A or not B)

A	B	A and B	(not A) or B	(not A or not B)	(A and B) or (not A or B) or (not A or not B)
0	0	0	1	1	1
0	1	0	1	1	1
1	0	0	0	1	1
1	1	1	1	0	1

(A and B) or (not A or B)

A	B	A and B	(not A) or B	(A and B) or (not A or B)
0	0	0	1	1
0	1	0	1	1
1	0	0	0	0
1	1	1	1	1

(not A or B) or (not A or not B)

A	B	(not A) or B	(not A) or (not B)	(not A or B) or (not A or not B)
0	0	1	1	1
0	1	1	1	1
1	0	0	1	1
1	1	1	0	1

(A or B) or (not A or not B)

A	B	A or B	(not A) or (not B)	(A or B) or (not A or not B)
0	0	0	1	1
0	1	1	1	1
1	0	1	1	1
1	1	1	0	1

Question 6 :

Le codage UTF-8 est sur 1 à 4 octets.

Rappel :

Le codage UTF-8 est un codage de longueur variable. Certains caractères sont codés sur un seul octet, ce sont les 128 caractères du codage ASCII. Les autres caractères peuvent être codés sur 2, 3 ou 4

octets. Ainsi l'UTF-8 permet en théorie de représenter *2^{21} = 2097152* caractères différents, en réalité un peu moins. Il y a actuellement environ une centaine de milliers de caractères Unicode (incluant les caractères des langues vivantes ou mortes et également de nombreux emojis indispensables 🙂)

Les caractères en UTF-8 doivent avoir une forme particulière décrite dans la table ci-dessous :

Nbre octets codant	**Format de la représentation binaire**
1	0xxxxxxx
2	110xxxxx 10xxxxxx
3	1110xxxx 10xxxxxx 10xxxxxx
4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

2ième Partie : Représentation des données : types construits

Question 7 :

```
def f(x,y):
    if x > y:
        return y,x
    else:
        return x,y
```

La fonction retourne un tuple, ici un doublet (les parenthèses sont facultatives) avec une inversion dans l'ordre de présentation par rapport à l'ordre croissant.

On obtient donc :

```
>>> f(42,21)
(21, 42)
```

Question 8 :

```
alphabet = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
```

La variable alphabet est du type list.

```
>>> type(alphabet)
<class 'list'>
>>> alphabet[4]
'E'
```

Question 9 :

```
def somme(tab):
    s = 0
    for i in range(len(tab)):
        s+=tab[i]# ou s=s+tab[i]
    return s

>>> somme([10,11,12,13,14])
60
```

Question 10 :

```
>>> [n * n for n in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Il s'agit d'une liste créée par compréhension, constituée des carrées des entiers de 0 à 9.

Question 11 :

Considérons l'exemple :
frequences={'do4': 523.25, 'la3': 440, 'mi4': 659.26}
Il s'agit d'un dictionnaire dont les clés sont les notes et les valeurs associées sont les fréquences en Hz.
Pour accéder à la valeur de la clé 'la3', on écrit :

```
>>> frequences['la3']
440
```

Rappels **:** connaître les **méthodes keys(), values()** et **items()** des dictionnaires

Pour parcourir les clés d'un dictionnaire :

```
for i in frequences.keys():
   print(i)
```

```
>>> %Run qcm0programmes.py
do4
la3
mi4
```

Pour parcourir les valeurs d'un dictionnaires :

```
for i in frequences.values():
   print(i)
```

```
>>> %Run qcm0programmes.py
523.25
440
659.26
```

Pour parcourir les items d'un dictionnaire, c'est-à-dire les paires (clé,valeur) :

```
for cles,valeurs in frequences.items():
   print("La valeur associée à la clé "+cles+" est : "+str(valeurs))
```

```
La valeur associée à la clé do4 est : 523.25
La valeur associée à la clé la3 est : 440
La valeur associée à la clé mi4 est : 659.26
```

Question 12 :

```
ports = {'http' : 80, 'imap' : 142, 'smtp' : 25}
ports['ftp'] = 21
print(ports['ftp'])
```

```
>>> %Run qcm0programmes.py
```

21

Le programme a ajouté la clé 'ftp' et sa valeur 21 au dictionnaire ports, d'où le nouveau dictionnaire

```
>>> ports
{'http': 80, 'imap': 142, 'smtp': 25, 'ftp': 21}
```

Troisième Partie : Traitement des données en tables

Question 13 :

```
repertoire = [{'nom':'Dupont', 'tel':'5234'}, {'nom':'Tournesol', 'tel':'5248'}, {'nom':'Dupond',
'tel':'3452'}]
```

repertoire est une variable qui est du type liste et qui est constitué de dictionnaires ou p-uplets nommés.

```
>>> type(repertoire)
<class 'list'>
>>> type(repertoire[0])
<class 'dict'>
```

Pour obtenir le numéro de téléphone de Tournesol, on fait :

```
>>> repertoire[1]['tel']
'5248'
```

Question 14 :

```
repertoire = [{'nom':'Dupont', 'tel':'5234'}, {'nom':'Tournesol', 'tel':'5248'}, {'nom':'Dupond',
'tel':'3452'}]
for i in range(len(repertoire)):
    if repertoire[i]['nom']=='Dupond' :
        print(repertoire[i]['tel'])
```

À nouveau répertoire apparaît comme une liste de dictionnaires.
On parcourt les éléments de la liste, c'est-à-dire les différents dictionnaires qui la constituent par indice croissant, lorsque pour un dictionnaire donné la valeur de la clé 'nom' vaut 'Dupond', on va chercher dans ce même dictionnaire la valeur de la clé 'tel'.

```
>>> %Run qcm0programmes.py
3452
```

Question 15 :

['Chat', 'Cheval', 'Chien', 'Cochon'] est trié par ordre alphabétique croissant ; on trie au rang 0, puis au rang 1, puis au rang 2….

Question 16 :

['12', '142', '21', '8'] est trié par ordre numérique croissant ; on trie au rang 0, puis au rang 1, puis au rang 2….

Question 17 :

```
image = [[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]
for i in range(4):
   for j in range(4):
      if (i+j) == 3:
         image[i][j] = 1
```

On parcourt les indices i et les indices j pour lesquels la somme donne 3 auxquels cas on affecte la valeur 1 à l'élément d'indice j de la liste située à l'indice i de la liste de liste image.

Ces valeurs se produisent pour quatre possibilités :

i=0 et j=3 ou i=3 et j=0
i=1 et j=2 ou i=2 et j=1

image=[[0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0]]

Question 18 :

```
table = [12, 43, 6, 22, 37]
for i in range(len(table) - 1):
   if table[i]>table[i+1]:
      table[i],table[i+1] = table[i+1],table[i]
```

On parcourt tous les éléments de la table sauf le dernier, on regarde si sa valeur est supérieure à la valeur suivante, auquel cas on procède à une inversion.

Voici l'évolution de la table (évolution que l'on peut suivre avec le debugger en faisant des « step into » dans la boucle et en regardant l'évolution des variables).

i=0 table=[12, 43, 6, 22, 37]
i=1 table=[12, 6, 43, 22, 37]
i=2 table=[[12, 6, 22, 43, 37]
i=3 table=[[12, 6, 22, 37, 43]