NSI: Faire le point :

Représentation des données : types et valeurs de base.

1. Quel est l'entier positif codé en base 2 sur 8 bits par le code 00101010 ?

   $1 \times 2^5 + 1 \times 2^3 + 1 \times 2 = 32 + 8 + 2 = 42$

   en Python :

   ```
   >>>øb00101010
   42
   ```

2. Quelle est la valeur affichée à l'exécution du programme Python suivant ?

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

   Cette fonction calcule 2^{10}, elle retourne donc la valeur 1024. Pour bien comprendre le fonctionnement de la boucle :

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

   retourne :

```
0 2
1 4
2 8
3 16
4 32
5 64
6 128
7 256
8 512
9 1024
1024
```

3. On considère la fonction python suivante :

```python
def fonction (x,y):
  p = x
  for i in range (y - 1):
    p = p * x
  return p
```

Selon vous que permet de faire cette fonction ? Elle permet de calculer la valeur de x^y. En Python :

```python
def fonction (x,y):
    p = x
    for i in range (y - 1):
        p = p * x
        print(i,p) # pour bien comprendre la boucle
    return p
```

retourne

```
>>> fonction (3,5)
0 9
1 27
2 81
3 243
243
```

A quelle conditions sur le paramètre `y` effectue-t-elle bien ce qu'elle est censée faire ?

y doit être un entier naturel supérieur ou égal à 1.

```
>>> fonction (3,1)
3
>>> fonction (3,0)
3
>>> fonction (3,-2)
3
>>> fonction (3,0.5)
line 3, in fonction
    for i in range (y - 1):
TypeError: 'float' object cannot be interpreted as an integer
```

4. Les entiers positifs ou nuls dont l'écriture en base 16 (hexadécimal) est constituée par un 1 suivi de 0 (par exemple 1, 10, 100, 1000, etc.) sont des puissances de ..... Ces nombres sont de la forme 1×16^k où k est un entier naturel, ce sont donc des puissances de 16.
5. Dans l'algorithme ci-dessous, qui prend en entrée un entier naturel non nul et renvoie son écriture binaire, par quel opérateur faut-il remplacer les pointillés ?

```
def cascade(n):
  chiffres = ''
  while n != 0:
    chiffres = str(n … 2) + chiffres
    n = n //2
  return chiffres
```

Il faut remplacer les pointilles par % :

```
def cascade(n):
  chiffres = ''
  while n != 0:
    chiffres = str(n%2) + chiffres
    n = n //2
  return chiffres
```

On a alors :

```
>>> cascade(12)
'1100'
```

6. Que retournent les trois commandes suivantes :

```
>>>int(øb11) # la commande øb11 retourne le même résultat, cette commande convertit un int en un int..., ce qui n'est pas très utile.
3
# Rappel øb11 représente l'écriture en binaire du nombre 3
```

```
>>>bin(14) # donne l'écriture bianire de 14
'0b1110'
```

```
>>>hex(øb1110110) # retourne l'écriture hexadécimale du nombre binaire 1110110
'0x76'
```

7. On considère la commande suivante :

```
>>> a="xcvds"
```

Que retournent les commandes suivantes ?

```
>>> a[2]
'v'
>>> len(a)
5
>>> a[-1]
's'
>>> a[len(a)]
IndexError: string index out of range
>>> a[len(a)-1]
's'
```

8. On considère la fonction suivante :

```
def alpha(message):
  retour=message
  for i in range(3):
    retour+=retour
  return retour
```

Que retournent, après exécution de cette fonction, les commandes suivantes :

```
>>> alpha('az')
'azazazazazazazaz'
>>> alpha(5)
40
```

Pour mieux comprendre le fonctionnement de cette fonction, on l'a modifiée comme suite :

```
def alpha(message):
    retour=message
    for i in range(3):
        retour+=retour
        print(i,retour)
    return retour
```

On a alors :

```
>>> alpha('az')
0 azaz
1 azazazaz
2 azazazazazazazaz
'azazazazazazazaz'
>>> alpha(5)
0 10
1 20
2 40
40
```

9. On considère les commandes suivantes :

```
>>> a='AZE'
>>> b=12
>>> c=120.6
>>> d=b==c
```

Que retournent alors les commandes :

```
>>> type(a)
<class 'str'>
>>> type(b)
<class 'int'>
>>> type(c)
<class 'float'>
>>> type(d)
<class 'bool'>
>>> d
False
>>> a+b
TypeError: can only concatenate str (not "int") to str
>>> type(b+c)
<class 'float'>
```