Exercice 1

1) Il est le symbole pour obtenir le quotient de la division entière. Donc 7//2 vaut 3
Donc x et y ont la même valeur 3. Donc x == y est Vrai
La réponse est True

2) Le bit de poids faible des premier, troisième et quatrième nombres est 1. Donc ils sont impairs. Donc le seul nombre pair est le deuxième.
La réponse est 10 0 0 1 0

3) Faisons la table de vérité de (P et Q) ou (non(P) ou Q)

P	Q	P et Q	non P	non P ou Q	(P et Q) ou (non(P) ou Q)
F	F	F	V	V	V
F	V	F	V	V	V
V	F	F	F	F	F
V	V	V	F	V	V

La réponse est Faux

4) Parmi les quatre listes, 14 est présent dans la liste [11, 12, 13, 14, 15] (indices 0, 1, 2, 3, 4)
c'est à dire L[2]
14 est l'élément d'indice 3 dans cette liste.
Donc l'expression L[2][3] vaut 14.
La réponse est L[2][3]

5) liste = [1, [2, 3], [4, 5], 6, 7] . liste contient 5 éléments. Sa longueur est 5.
1er élément, 2e, 3e, 4e, 5e
La réponse est 5

6) La fonction foo contient une boucle for qui parcourt toute la liste en comparant chaque élément liste[i] avec son suivant.
Dès qu'on rencontre le cas liste[i] > liste[i+1] c'est à dire dès qu'on trouve un élément qui est strictement supérieur à son suivant, alors la fonction renvoie False
Dans le cas contraire (aucun élément supérieur à son suivant) alors la fonction renvoie True. Autrement dit la liste est triée croissante.
Parmi les listes proposées, seule la dernière est triée croissante.
La réponse est foo([1, 2, 2, 3, 4, 5])

7) lst est une liste en compréhension qui contient les éléments présents dans variable = [8, 12, -7, 52, -5, 32] si l'élément est strictement inférieur à 12.
Donc lst vaut [8, -7, -5].
La réponse est [8, -7, -5]

8) fruit est initialisé comme étant un dictionnaire vide
La fonction addone est appelée 5 fois avec comme paramètre le dictionnaire fruit.
A chaque appel, on lui fournit en premier paramètre une clé qui est le nom d'un fruit.
Si cette clé est dans le dictionnaire alors la valeur de l'item augmente de 1 (c'est l'instruction dic[index] += 1)
Sinon l'item est créé avec la valeur 1 (c'est l'instruction dic[index] = 1
Donc la réponse est {'Apple': 1, 'Banane': 1, 'Orange': 3}

Exercice 2

1) Le schéma 1 montre deux inconvénients de l'envoi en flux continu :
- Si le fil est coupé alors le flux est interrompu.
- Si un routeur est occupé alors le flux est bloqué.

2) Le schéma 2 montre deux avantages du découpage en paquets :
- Si un paquet est perdu, la totalité du message ne doit pas être renvoyée à nouveau. Seulement le paquet perdu doit être renvoyé à nouveau.
- le routeur n'est pas occupé exclusivement pour un flux. Donc les flux ne sont pas bloqués.

3) Le travail de la couche de transport du modèle OSI est
- de découper les données en paquets
- d'ajouter à chaque paquet une en-tête de transport.

Exercice 3

1 Remplacer une valeur

Écrire une fonction `remplacer` prenant en argument :

- une liste d'entiers `valeurs`
- un entier `valeur_cible`
- un entier `nouvelle_valeur`

et renvoyant une **nouvelle** liste contenant les mêmes valeurs que `valeurs`, dans le même ordre, sauf `valeur_cible` qui a été remplacé par `nouvelle_valeur`.

La liste passée en argument ne doit pas être modifiée.

Exemples

Si `valeurs = [3, 8, 7]`
alors `remplacer(valeurs, 3, 0)` renvoie `[0, 8, 7]` et valeurs vaut toujours `[3, 8, 7]`
Si `valeurs = [3, 8, 3, 5]`
alors `remplacer(valeurs, 3, 0)` renvoie `[0, 8, 0, 5]` et valeurs vaut toujours `[3, 8, 3, 5]`

```
def remplacer(valeurs, valeur_cible, nouvelle_valeur):
    liste = list(valeurs)    # Fait une copie de la liste valeurs
    for i in range(valeurs):
        if liste[i] == valeur_cible:
            liste[i] = nouvelle_valeur




    return liste
```

2 Maximum

Écrire une fonction `maximum` prenant en argument :

- une liste non vide de nombres `nombres`

et renvoyant le plus grand élément de cette liste.

Chacun des nombres utilisés est de type `int` ou `float`. On interdit ici d'utiliser `max`, ainsi que `sort` ou `sorted`.

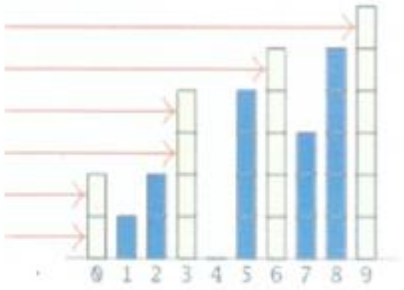
Exemple

Si `nombres = [98, 12, 104, 23, 131, 9]`
alors `maximum(nombres)` renvoie `131`

```
def maximum(nombres):
    maxi = nombres[0]
    for element in nombres:
        if element > maxi:
            maxi = element
    return maxi
```

3 Soleil couchant sur les bâtiments

Lorsque des bâtiments sont alignés, ils se font de l'ombre les uns les autres. Dans cet exercice, nous sommes au soleil couchant, les rayons du soleil sont donc supposés horizontaux.



Le soleil couchant éclaire neuf bâtiments, les rayons du soleil sont représentés par des flèches horizontales.

- Les bâtiments aux indices 0 et 3 reçoivent des rayons de soleil alors que les bâtiments aux indices 1 et 2 sont masqués.
- Les **quatre** bâtiments aux indices [0, 3, 6, 9] reçoivent des rayons de soleil sur au moins un étage et sont donc éclairés, alors que les autres ne le sont pas.

Écrire une fonction `nb_batiments_eclaires` qui prend en argument la liste `hauteurs` des bâtiments et qui renvoie le nombre de bâtiments éclairés.

- La hauteur des bâtiments (en nombre d'étages) est donnée par une liste d'entiers positifs. Une hauteur de zéro étage signifie l'absence de bâtiment.

Pour l'exemple ci-dessus, cette liste est [2, 1, 2, 4, 0, 4, 5, 3, 5, 6].

Exemple

Si `hauteurs = [2, 1, 2, 4, 0, 4, 5, 3, 5, 6]`
alors `nb_batiments_eclaires(hauteurs)` renvoie `4`

```
def nb_batiments_eclaires(hauteurs):
    nombre_eclaires = 0
    hauteur_max = 0
    for i in range(len(hauteurs)):
        if hauteurs[i] > hauteur_max:
            nombre_eclaires = nombre_eclaires + 1
            hauteur_max = hauteurs[i]

    return nombre_eclaires
```