

TP – Adresse IP



Exercice 1

1. Quelle fonction **zeroAdroite(N)** mettrait les bits de droite à zéro pour un nombre de 10 bits?
Par exemple le nombre 10110 11110 en 10110 00000 (734 en 704),
c'est-à-dire ne garder que les bits de la partie gauche (ceux de droite sont à 0) ?
2. Quelle fonction **zeroAgauche(N)** mettrait les bits de gauche à zero pour un nombre de 10 bits?
Par exemple transformer le nombre 10110 11110 en 00000 11110 (734 en 30),
c'est-à-dire ne garder que les bits de la partie droite (ceux de gauche sont à 0) ?.

```
# Dans la console PYTHON
# Pour mettre des 0 à droite
> zeroAdroite(bin(0b1011011110))
'0b1011000000'
#C'est pareil que
>zeroAdroite(734)
704
# Pour mettre des 0 à gauche
> zeroAgauche(bin(0b1011011110))
'0b11110'
#C'est pareil que
>zeroAgauche(734)
30
```

Comme on peut le voir ci-dessus le but ici est de diviser un nombre binaire de 10 bits en 2 parties de 5 bits chacune, puis on met, soit les bits de gauche ou de droite à une valeur de 0.

Afin de résoudre cette problématique, j'ai décidé d'utiliser un masque (tout comme cela fonctionne avec les masques de sous-réseau). Cela signifie donc que :

- Pour mettre les bits de droites à 0 le masque sera : **0b1111100000**
- Pour mettre les bits de gauches à 0 le masque sera : **0b0000011111**

Il ne me reste donc plus qu'à développer les deux fonctions.

Commençons par celle qui met les bits de **droites** à 0 :

```
def zeroAdroite(valeur_donnee):
    try :
        valeur_donnee = int(valeur_donnee, 2)
    except :
        pass
    mask = int(bin(0b1111100000), 2)
    return (bin(valeur_donnee & mask), valeur_donnee & mask)
```

Cette fonction essaie dans un premier temps de convertir une valeur de type string (représentant un chiffre binaire) en entier numérique.

Ici j'utilise la méthode « try...except » car cela permet à l'utilisateur de mettre en argument soit un chiffre qui est déjà un entier, soit un chiffre binaire, peu importe son choix la fonction réalisera bien la conversion !

Dans un second temps, je définis le masque qui est donc ici : **0b1111100000**, comme vu plus tôt !

Enfin je retourne un tuple, qui contient d'abord la valeur modifiée en valeur binaire puis en valeur entière. Comme vous pouvez le remarquer j'utilise l'opérateur AND (et) noté « & » en Python, afin d'appliquer le masque sur la valeur initiale.

Selon cette méthode :



L'adresse de sous-réseau est obtenue avec l'adresse machine et le masque de sous-reseau

L'adresse du réseau est donnée par :

IP & MASQ (& est le AND bit à bit)

Fonction identique mais qui met les bits de **gauches** à 0 :

```
def zeroAgauche(valeur_donnee):
    try :
        valeur_donnee = int(valeur_donnee, 2)
    except :
        pass
    mask = int(bin(0b0000011111), 2)
    return (bin(valeur_donnee & mask), valeur_donnee & mask)
```

Je vérifie donc si ces fonctions sont bel et bien fonctionnelles en utilisant les exemples donnés dans l'énoncé :

```
>>> %Run nb_binaires_et_adressage.py
>>> zeroAdroite(bin(0b1011011110))
('0b1011000000', 704)

>>> zeroAdroite(734)
('0b1011000000', 704)

>>> zeroAgauche(bin(0b1011011110))
('0b11110', 30)

>>> zeroAgauche(734)
('0b11110', 30)
```

Cela fonctionne donc parfaitement !



Exercice 2

Dans un fichier appelé *adresseIP.py* (Dans lequel vous écrirez toutes les fonctions suivantes), écrire une fonction python **convIPstr_int(ipstr)** qui convertit une adresse IP écrit sous la forme d'une chaîne de caractère "A.B.C.D" en un entier int : $A \times 2^{24} + B \times 2^{16} + C \times 2^8 + D$. On peut utiliser la méthode **split("."')** pour casser la chaîne suivant les points

Code Python

```
# Dans l'éditeur PYTHON
def convIPstr_int(ipstr):
    """
    In: ipstr est un str
    Out: ipint est un int
    """
    ...

    return ipint
```

```
# Dans la console PYTHON
> convIPstr_int("115.250.207.0")
1945816832
```

Le but ici est de récupérer en input une adresse IP de la forme « A.B.C.D », puis la fonction sépare chaque « bloc » grâce au point et donc la méthode « **split("."')** » !

Puis la fonction convertie chacun de ces blocs, qui sont alors des « strings », en entier !

Et donc grâce à cette conversion le programme peut réaliser le calcul :

$$\underline{\mathbf{A \times 2^{24} + B \times 2^{16} + C \times 2^8 + D}}$$

CODE :

```
def convIPstr_int(ipstr):
    """
    In: ipstr est un str
    Out: ipint est un int
    """
    ip_list = ipstr.split(".")
    for i in range(len(ip_list)) :
        ip_list[i] = int(ip_list[i])
    ipint = ip_list[0] * 2**24 + ip_list[1] * 2**16 + ip_list[2] *
2**8 + ip_list[3]
    return ipint
```

- A correspond au premier élément de la liste (`ip_list[0]`)
- B correspond au deuxième élément de la liste (`ip_list[1]`)
- C correspond au troisième élément de la liste (`ip_list[2]`)
- D correspond au quatrième élément de la liste (`ip_list[3]`)

Je stocke donc le résultat du calcul dans la variable « ipint », que je renvoie ensuite en output.

TEST :

```
53  print(convIPstr_int("115.250.207.0"))

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL

[Running] python -u "d:\Professionel\Lycée VAUBAN\Python\Exercices\Exo_01.ipynb"
1945816832

[Done] exited with code=0 in 0.056 seconds
```



Exercice 3

| Écrire une fonction python `convIPint_str(ipint)` qui fait le contraire de la fonction précédente.

```
# Dans la console PYTHON
> convIPint_str(1945816832)
'115.250.207.0'
```

Le but ici est de récupérer en input un entier, pour que la fonction renvoie son équivalent en adresse IP sous le type « string ».

Cette adresse IP sera de la forme « A.B.C.D ».

Pour résoudre ce problème je vais d'abord convertir le nombre entier en hexadécimal, puis diviser cette valeur hexadécimale en 4 blocs.

Chaque bloc est de suite convertie en valeur entière.

Il ne reste plus qu'à mettre tous ces entiers (convertis en strings) dans une variable, en les séparant par un point !

Puis je retourne cette variable en output (ipstr).

CODE :

```
def convIPint_str(ipint):
    """
    In: ipint est un int
    Out: ipstr est un str
    """
    ip_value = hex(ipint)
    bloc1 = int(ip_value[0:4], 16)
    bloc2 = int("0x" + ip_value[4:6], 16)
    bloc3 = int("0x" + ip_value[6:8], 16)
    bloc4 = int("0x" + ip_value[8:10], 16)
    ipstr = str(bloc1) + "." + str(bloc2) + "." + str(bloc3) + "."
+ str(bloc4)
    return ipstr
```

TEST :

```
45 print(convIPint_str(1945816832))

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL

[Running] python -u "d:\Professionel\Lycée
115.250.207.0

[Done] exited with code=0 in 0.055 seconds
```



Exercice 4

En utilisant les fonctions précédentes, écrire une fonction python **adresseReseau(ip , masque)** qui renvoie l'adresse réseau sous la forme A.B.C.D correspondant à cet IP et ce masque. Attention les entrées doivent pouvoir être écrite sous la forme A.B.C.D, c'est à dire sous la forme d'une chaîne de caractère.

Code Python

```
# Dans l'éditeur PYTHON
def adresseReseau(ip , masque):
    """
    In: ip , masque str
    Out: adresReseau est un str
    """
    ...
    ...

    return adresReseau
```

```
# Dans la console PYTHON
>>> adresseReseau( "115.250.207.0", "255.255.240.0")
'115.250.192.0'
```

Le but ici de récupérer une adresse IP et de lui appliquer un masque.

Je vais convertir ces différents arguments (qui sont de base des strings) en entier via la fonction précédemment créée : « convIPstr_int() »

Puis j'utilise l'opérateur AND (et) noté « & » en Python, afin d'appliquer le masque sur la valeur initiale.

Avant de retourner la valeur, je la convertis une dernière fois en strings pour qu'elle soit plus compréhensible pour l'utilisateur.

J'utilise donc les deux fonctions : « convIPstr_int() » et « convIPint_str() » dont leur fonctionnement est décrit plus tôt dans ce compte-rendu.

CODE :

```
def adresseReseau(ip, masque):
    ip = convIPstr_int(ip)
    masque = convIPstr_int(masque)
    ip = ip & masque
    return convIPint_str(ip)
```

TEST :

```
41 def adresseReseau(ip, masque):
42     ip = convIPstr_int(ip)
43     masque = convIPstr_int(masque)
44     ip = ip & masque
45     return convIPint_str(ip)
46
47 print(adresseReseau( "115.250.207.0","255.255.240.0"))

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL
[Running] python -u "d:\Professionel\Lycée VAUBAN\Première Gé
115.250.192.0
[Done] exited with code=0 in 0.053 seconds
```



Exercice 5

Ecrire une fonction python **NbrAdresse(masque)** qui renvoie le nombre d'adresses dans un sous-réseau.
Il faut compter le nombre de zéro dans le masque.

Code Python

```
# Dans l'éditeur PYTHON
def NbrAdresse( masque):
    """
    In: masque str
    Out: Nbr est un int
    """
    ...
    return Nbr
```

```
# Dans la console PYTHON
>NbrAdresse('255.255.240.0')
4094
```

Le but de l'exercice ici est de déterminer le nombre d'adresses IP possible dans un sous-réseau.

Pour cela je convertis d'abord l'adresse du masque donnée en entier puis en binaire afin de compter le nombre de 0 dans cette valeur !

Lors du comptage de 0 je soustrais 1, car il ne faut pas oublier qu'il y a le 0 de « 0b ».

La formule pour calculer le nombre d'adresse est : $2^{\text{nombre de } 0} - 2$
Je réalise donc ce calcul et je renvoie le résultat.

CODE :

```
def NbrAdresse(masque):
    """
    In: masque str
    Out: Nbr est un int
    """
    masque = bin(convIPstr_int(masque))
    nb_0 = masque.count('0') - 1
    Nbr = 2**nb_0 - 2
    return Nbr
```

TEST :

```
47 v def NbrAdresse(masque):
48 v     """
49     In: masque str
50     Out: Nbr est un int
51     """
52     masque = bin(convIPstr_int(masque))
53     nb_0 = masque.count('0') - 1
54     Nbr = 2**nb_0 - 2
55     return Nbr
56
57 print(NbrAdresse("255.255.240.0"))
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL

[Running] python -u "d:\Professionel\Lycée VAU
4094

[Done] exited with code=0 in 0.056 seconds

3 Quelques défis pour aller plus loin

Défi 1

Corriger **adresseReseau(ip, masque)** pour qu'elle accepte les deux écritures. Par exemple, on pourrait faire **adresseReseau(90.98.100.3/21)** ou **adresseReseau(90.98.100.3,255.255.248.0)**

```
# Dans la console PYTHON
>>> adresseReseau( "115.250.207.0","255.255.240.0")
'115.250.192.0'
>>> adresseReseau( "115.250.207.0/20")
'115.250.192.0'
```

Le but ici est d'améliorer ma fonction « **adresseReseau()** » afin qu'elle accepte deux types d'entrées :

- "115.250.207.0","255.255.240.0" soit deux arguments.
- "115.250.207.0/20" soit un seul argument.

Je définis donc un masque par défaut qui sera : « "255.255.255.255" », qui ne masque rien en réalité mais qui est essentiel à la fonction !

Ensuite je mets une condition basée sur la présence du caractère « / » dans l'adresse IP pour séparer les deux types d'entrées afin de les exploiter différemment !

Ici nous allons voir le traitement du type d'entrée "115.250.207.0/20", pour l'autre méthode veuillez vous référer à l'exercice 4.

Il s'agit donc d'une chaîne de caractères, je la "split" au niveau du slash (« / »). Ce qui signifie donc que ma liste contiendra en premier élément l'adresse IP, et en deuxième élément le masque.

Je convertis donc ces deux éléments en entier via les fonctions développées précédemment.

Je peux donc maintenant créer le masque en binaire étant donné que je connais le nombre de 1 et de 0 à disposer.

Enfin j'utilise l'opérateur AND (et) noté « & » en Python, afin d'appliquer le masque sur la valeur initiale.

Avant de retourner la valeur, je la convertis une dernière fois en strings pour qu'elle soit plus compréhensible pour l'utilisateur.

CODE :

```
def adresseReseau(ip, masque="255.255.255.255"):  
    if « / » in ip :  
        ip_splited = ip.split("/")  
        ip = ip_splited[0]  
        masque = int(ip_splited[1])  
        ip = convIPstr_int(ip)  
        nb_0 = 32-masque  
        mask = "0b" + masque * "1" + nb_0 * "0"  
        mask = int(mask, 2)  
        ip = ip & mask  
    else :  
        ip = convIPstr_int(ip)  
        masque = convIPstr_int(masque)  
        ip = ip & masque  
    return convIPint_str(ip)
```

TEST :

```
67  print(adresseReseau("115.250.207.0/20"))  
  
PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL  
  
[Running] python -u "d:\Professionel\Lycée VAU  
115.250.192.0  
  
[Done] exited with code=0 in 0.06 seconds
```

La fonction me retourne bien la bonne adresse IP !

Défi 2

Ecrire une fonction python **adresseBroadcast(ip , masque)** pour qu'elle renvoie l'adresse de broadcast.

Code Python

```
# Dans l'éditeur PYTHON
def adresseBroadcast(ip , masque):
    """
    In: ip , masque sont des str
    Out: adresbrodcast est str
    """
    ...

    return adresbrodcast
```

```
# Dans la console PYTHON
>adressebroadcast( "90.98.100.3", "255.255.255.128")
90.98.100.127
>adressebroadcast( "90.98.100.3/25")
90.98.100.127
```

Ici le but est d'obtenir l'adresse de « broadcast » en connaissant l'adresse IP et le masque.

Tout comme la fonction « adresseReseau() », cette fonction accepte aussi deux types d'entrées :

- "115.250.207.0","255.255.240.0" soit deux arguments.
- "115.250.207.0/20" soit un seul argument.

Je définis donc un masque par défaut qui sera : « "255.255.255.255" », qui ne masque rien en réalité mais qui est essentiel à la fonction !

Ensuite je mets une condition basée sur la présence du caractère « / » dans l'adresse IP pour séparer les deux types d'entrées afin de les exploiter différemment !

Pour ce qui est de la séparation et de la création du masque, veuillez vous référer à la page précédente.

Je récupère donc le nombre d'adresses différentes possibles sur le sous-réseau via la fonction créée précédemment « NbrAdresse() », tout en ajoutant une adresse supplémentaire qui sera donc celle de « broadcast » !

Enfin, il ne me reste plus qu'à réaliser une « addition » de bits, notée OR (ou) qui s'écrit | en Python.

CODE :

```
def adresseBroadcast(ip, masque="255.255.255.255"):  
    """  
        In: ip , masque sont des str  
        Out: adresbrodcast est str  
    """  
    if "/" in ip :  
        ip_splited = ip.split("/")  
        ip = ip_splited[0]  
        masque = int(ip_splited[1])  
        nb_0 = 32-masque  
        mask = "0b" + masque * "1" + nb_0 * "0"  
        masque = convIPint_str(int(mask, 2))  
        nb_possible = NbrAdresse(masque) + 1  
        ip = convIPstr_int(ip)  
        adresbrodcast = nb_possible | ip  
    else :  
        nb_possible = NbrAdresse(masque) + 1  
        ip = convIPstr_int(ip)  
        adresbrodcast = nb_possible | ip  
    return convIPint_str(adresbrodcast)
```

TEST :

```
88 print(adresseBroadcast("90.98.100.3","255.255.255.128"))  
89 print(adresseBroadcast("90.98.100.3/25"))  
  
PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL  
[Running] python -u "d:\Professionel\Lycée VAUBAN\Première Gén  
90.98.100.127  
90.98.100.127  
[Done] exited with code=0 in 0.056 seconds
```

Cela fonctionne parfaitement, peu importe le type d'entrée je trouve la même et bonne adresse de broadcast.

Défi 3

Ecrire une fonction python **plageAdresse(ip , masque)** pour qu'elle renvoie une liste d'adresses possibles dans son réseau.

Code Python

```
# Dans l'éditeur PYTHON
def plageAdresse(ip , masque):
    """
    In: ip , masque sont des str
    Out: plageadr est une liste de str
    """
    ...
    return plageadr
```

```
# Dans la console PYTHON
>plageAdresse("115.250.207.0","255.255.255.252")
['115.250.207.1', '115.250.207.2']
```

Le but ici est de créer une fonction qui renvoie une liste complète des adresses possibles dans son réseau.

Je détermine d'abord le nombre d'adresse possibles, puis grâce à une boucle j'incrémente l'adresse initiale et ainsi de suite jusqu'à que toutes les adresse possibles soit dans la liste !

CODE :

```
def plageAdresse(ip , masque):
    """
    In: ip , masque sont des str
    Out: plageadr est une liste de str
    """
    nb_possible = NbrAdresse(masque) + 1
    ip = convIPstr_int(ip)
    count = 1
    plageadr = []
    while count < nb_possible:
        ip_temp = convIPint_str(ip + count)
        plageadr.append(ip_temp)
        count += 1
    return plageadr
```

TEST :

```
88 def plageAdresse(ip, masque):
89     """
90     In: ip , masque sont des str
91     Out: plageadr est une liste de str
92     """
93     nb_possible = NbrAdresse(masque) + 1
94     ip = convIPstr_int(ip)
95     count = 1
96     plageadr = []
97     while count < nb_possible:
98         ip_temp = convIPint_str(ip + count)
99         plageadr.append(ip_temp)
100        count += 1
101    return plageadr
102
103 print(plageAdresse("115.250.207.0","255.255.255.252"))
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL

```
[Running] python -u "d:\Professionel\Lycée VAUBAN\Première Gé
['115.250.207.1', '115.250.207.2']

[Done] exited with code=0 in 0.06 seconds
```

Je trouve bel et bien les deux adresses possibles sur le réseau !

Si vous souhaitez explorer le code plus en détail et réaliser vos propres tests, il se trouve ci-joint à ce compte-rendu, sous le nom de «adresseIP.py».