Math93.com

Cours - NSI
Adresse IP

1 Rappel sur les opérateurs bit à bit

Python dispose de nombreux opérateurs qui agissent sur les nombres au niveau des bits en respectant les tables de vérités. On les nomment **opérateurs bit-à-bit** (*bitwise* en anglais).
Par exemple on a :

```
# Dans la console PYTHON

# Pour AND (et) noté & en Python
> bin(0b1010 & 0b1100)
'0b1000'

# Pour OR (ou) noté | en Python
> bin(0b1010 | 0b1100)
'0b1110'

# Pour XOR (ou exclusif) noté ^ en Python
> bin(0b1010 ^ 0b1100)
'0b110'

# Pour décaller les bits vers la droite ou la gauche
# Utile pour les divisions et mult. par puissances de 2
> bin(0b101010<<2)
'0b10101000'
> bin(0b101010>>2)
'0b1010'
```

Remarquer que ici on a utilisé l'écriture binaire mais on peut tout écrire en décimal.

```
# Pour AND (et) noté & en Python
> bin(0b1010 & 0b1100)
'0b1000'
#C'est pareil que
>10 & 12
8
# Pour OR (ou) noté | en Python
> bin(0b1010 | 0b1100)
'0b1110'
#C'est pareil que
>10 | 12
14
# Pour XOR (ou exclusif) noté ^ en Python
> bin(0b1010 ^ 0b1100)
'0b110'
#C'est pareil que
>10 ^ 12
6
```

```
# Pour XOR (ou exclusif) noté ^ en Python
> bin(0b1010 ^ 0b1100)
'0b110'
#C'est pareil que
>10 ^ 12
6
# Pour décaller les bits vers la droite ou la gauche
# Utile pour les divisions et mult. par puissances de 2
> bin(0b101010<<2)
'0b10101000'
#C'est pareil que
>42<<2
168
> bin(0b101010>>2)
'0b1010'
#C'est pareil que
>42>>2
10
```

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
```

1.1 Structure d'une Adresse IP (iPv4)

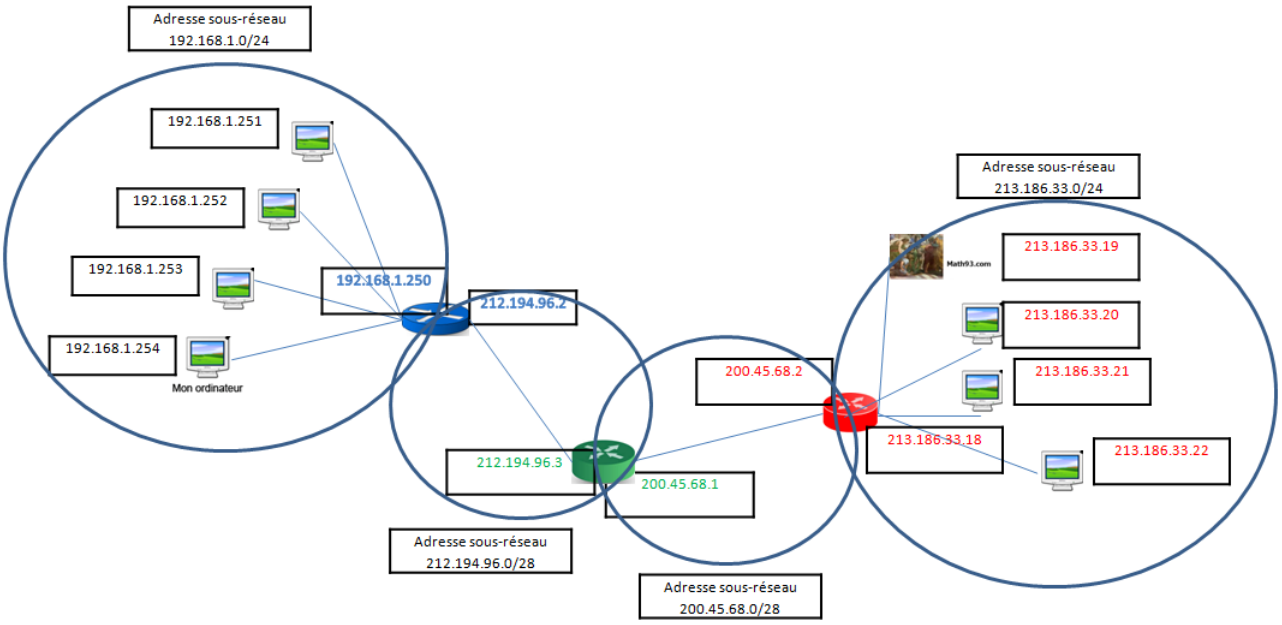

FIGURE 1 – *Quatre sous-réseaux forment un réseau. Chaque machine a une adresse dans son sous-réseau. Chaque sous-réseau a une adresse.*

Une adresse IP (avec IP pour Internet Protocol) est un numéro d'identification qui est attribué de façon permanente ou provisoire à chaque périphérique relié à un réseau informatique qui utilise l'Internet Protocol. L'adresse IP est à la base du système d'acheminement (le routage) des paquets de données sur Internet.

Structure d'une Adresse IP (iPv4)

- Une adresse IP est codée sur 32 bits (4 octets ou bytes)(Ipv4 1984)
- Une adresse IP est souvent présentés en 4 blocs de 4 octets : A.B.C.D, par exemple :

$$IP = 192.168.1.55$$

- Plus petite adresse 0.0.0.0
- La plus grande : 255.255.255.255
- Il y a 2^{32} adresses possibles.
- Une partie représente l'adresse de la machine
- Une autre représente l'adresse du sous-réseau
- Le nombre d'adresse commençant à s'épuiser on essaye de développer Ipv6 sur 128 bits (1996)

Elle est accompagnée du masque de sous-réseau qui est aussi de 4 octets : A',B',C',D', par exemple :

$$MASQ = 255.255.255.0$$

C'est avec le masque de sous-réseau que l'on peut déterminer quelle est l'adresse du sous-réseau et quelle est l'adresse de la machine.

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

L'adresse du réseau est donnée par :

$$IP \ \& \ MASQ \text{ (\& est le AND bit à bit)}$$

Remarque

Un masque ne peut pas contenir n'importe quel nombre. Il doit forcément n'avoir que des 1 à gauche et que des 0 à droite.
255.255.240.0 = 11111111.11111111.11100000.00000000 est un masque valide

240.255.240.0 = 11100000.1111111.11100000.00000000 n'est pas un masque valide

Méthode :

Voici une IPv4 90.98.100.3 et son masque de sous réseau 255.255.255.0
C'est à dire en binaire :

IP	=0101 1010.	0110 0010.	0110 0100.	0000 0011
& MASQ	=1111 1111.	1111 1111.	1111 1111.	0000 0000
Adresse Sous-reseau	=0101 1010.	0110 0010.	0110 0100.	0000 0000
	=90.	98.	100.	0

On peut penser que le masque ne sert à rien. On trouve facilement l'adresse réseau si le masque ne contient que des 0 ou 255 mais si le masques est du type 255.255.240.0, dans ce cas il faut faire attention.

IP	=0101 1010.	0110 0010.	0110 0100.	0000 0011
& MASQ	=1111 1111.	1111 1111.	1110 0000.	0000 0000
Adresse Sous-reseau	=0101 1010.	0110 0000.	0110 0000.	0000 0000
	=90.	98.	96.	0

Remarque

Une adresse finissant par zéro n'est pas forcément une adresse réseau. Par exemple l'adresse 115.250.207.0 avec le masque 255.255.240.0 n'est pas une adresse réseau.

IP	= 0111 0011.	1111 1010.	1100 1111.	0000 0000
& MASQ	=1111 1111.	1111 1111.	1111 0000.	0000 0000
Adresse Sous-reseau	=0111 0011.	1111 1010.	1100 0000.	0000 0000
	=115.	250.	192.	0

115.250.207.0 est donc une adresse de machine dans un réseau d'adresse 115.250.192.0.

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 caractére "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
```

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'
```

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'
```

2 Informations données par l'adresse IP

Nombre de machine possible dans un sous-réseau

Pour une IP et un masque donnée, il y a deux adresses réservées :

- Une pour le sous-réseau ,
- Une pour le broadcast (adresse de diffusion qui permet d'envoyer une trame à toutes les machines du sous-réseau).

On peut savoir combien il y a de machine dans un sous-réseau en comptant le nombre de bit à zéro dans le masque.

On obtient l'adresse broadcast en remplaçant les zéro du masque par des 1 dans l'adresse de réseau.

$$IP\ reseau\ OU \text{ le complément à 1 du masque}$$

Par exemple pour l'adresse 90.98.100.3/255.255.255.0, il y a 8 bits à zéro dans le masque donc il y a 2^8-2 machines possibles. L'adresse réseau est 90.98.100.0 et l'adressse broadcast est 90.98.100.255.

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
```

Remarque

Il existe une autre notation pour les masques de sous-reseau, on donne l'IP suivie d'un slash et du nombre de bits à gauches utilisés pour l'adresse du sous réseau.

Par exemple 90.98.100.3/21 indique que le masque est
11111111.11111111.11111000.00000000 soit 255.255.248.0

Plage d'adresses possibles dans un sous-réseau

- La première adresse possible dans un sous-réseau est l'adresse du sous réseau.
- La dernière adresse possible dans un sous-reseau est l'adresse de broadcast.
- Donc les adresses machines sont toutes les adresses possibles entre ces deux valeurs.

Exemple :
Pour 115.250.207.3/255.255.255.248.
L'adresse réseau est 115.250.207.0, l'adresse de broadcast est 115.250.207.7 donc les adresses possibles de machines sont :
115.250.207.1
115.250.207.2
115.250.207.3
115.250.207.4
115.250.207.5
115.250.207.6

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'
```

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
```

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']
```