Chapitre 5. Entiers relatifs, réels et caractères

Table des matières

Remplissez le jupyter notebook suivant en vous aidant de votre livre de Première NSI de Serge BAYS .

- Pour répondre, double-cliquez sur **Réponse** et complétez la zone en-dessous. Puis cliquez sur le bouton *Exécuter*.
- **Important : pour fermer votre jupyter notebook, cliquez sur :**

Fichier / Créer une nouvelle sauvegarde

puis sur :

Fichier / Fermer et Arrêter

- Ecrivez ci-dessous votre prénom et votre nom :

Réponse :

Chapitre 5. Entiers relatifs, réels et caractères

1. Entiers relatifs

1) Qu'est-ce qu'un entier relatif ?

Réponse :

1.1 Le complément à deux

Lisez le paragraphe **Entiers relatifs** p. 97 et en haut de la p. 98

En informatique, le **complément à deux** est une opération qui permet de **changer le signe d'un nombre binaire**.

Le complément à deux opère toujours sur des nombres binaires ayant *un certain nombre de bits donné bien connu*. Par exemple on sait qu'on travaille avec des mots de n = 4 bits ou des mots de n = 6 bits ou des mots de n = 8 bits etc. Dans une telle écriture, *le bit de poids fort* (bit le plus à gauche) donne *le signe* du nombre représenté (positif ou négatif). C'est le **bit de signe**.

Si le bit le plus à gauche est 0 alors le nombre est positif.

Si le bit le plus à gauche est 1 alors le nombre est négatif.

- Première idée (qui conduit à des erreurs) :

Méthode : le bit de poids fort donne le signe et les autres bits donnent la valeur absolue du nombre. Pour changer le signe de l'entier, on change le bit de poids fort.

Exemple : On choisit ici de travailler uniquement avec 4 bits. Si on travaille avec 4 bits, celui de poids fort étant réservé pour le signe, on obtient la table :

Bit de signe	Autres bits	En décimal
0	000	+0
0	001	+1
0	010	+2
0	011	+3
0	100	+4
0	101	+5
0	110	+6
0	111	+7
1	000	-0
1	001	-1
1	010	-2
1	011	-3
1	100	-4
1	101	-5
1	110	-6
1	111	-7

2) a) Avec cette méthode, quel est le résultat de l'addition de 4 et de -2 ? (puisqu'on a choisi de travailler uniquement sur 4 bits, alors on ignore un éventuel 5 ème bit à gauche qui pourrait provenir d'une retenue)

Réponse : *Complétez les pointillés*

4 est représenté par 0100 et -2 est représenté par 1010.

0100 additionné avec 1010 donne ..... qui d'après notre table a pour écriture décimale .....

On voit que le résultat est faux. Donc on change de méthode.

- Deuxième idée (qui est correcte) : **le complément à 2^n**, qu'on appelle plus rapidement **le complément à 2** :

Méthode : Comme précédemment, le bit de poids fort donne le signe du nombre, mais pour changer le signe de l'entier, on effectue trois étapes :

1) **On prend le NOT** de *chaque bit* (y compris le bit de poids fort) c'est à dire qu'on remplace les 0 par des 1 et les 1 par des 0.

2) **On additionne 1 et on regarde la somme**.

3) **On ignore un éventuel bit supplémentaire à gauche** qui proviendrait d'une retenue.

Exemple : On choisit ici de travailler uniquement avec 4 bits. Puisqu'on travaille avec 4 bits, le 4e bit (celui de gauche) sera réservé pour le signe et il restera seulement 3 bits pour le nombre lui-même. On pourra donc seulement travailler avec des entiers dont la valeur absolue va de 0 à 7 :

En décimal	En binaire	1) on prend le NOT	2) On ajoute 1	3) On ignore le 5e bit	En décimal
+0	0000	1111	10000	0000	-0
+1	0001	1110	1111	1111	-1
+2	0010	1101	1110	1110	-2
+3	0011	1100	1101	1101	-3
+4	0100	1011	1100	1100	-4
+5	0101	1010	1011	1011	-5
+6	0110	1001	1010	1010	-6
+7	0111	1000	1001	1001	-7

2) b) Avec cette méthode, quel est le résultat de l'addition de 4 et de -2 ? (puisqu'on a choisi de travailler uniquement sur 4 bits, alors on ignore un éventuel 5 ème bit à gauche qui pourrait provenir d'une retenue)

Réponse : *Complétez les pointillés*

4 est représenté par 0100 et -2 est représenté par 1110.

0100 additionné avec 1110 donne ..... qui d'après notre table a pour écriture décimale .....

- On voit sur cet exemple que le résultat est correct. Cela provient du fait que -2 est correctement représenté par son complément à 16 qui est 1110 soit 14 en décimal (le complément à 2^4 de 2 est $2^4 - 2 = 14$).

De façon générale, le complément à 2^4 de r est $2^4 - r$.

Si on additionne $2^4 - r$ et r on obtient $r + 2^4 - r = 2^4$ soit en binaire 10000, mais comme on ne tient pas compte du 5 ème bit, cela fait 0000 soit 0.

Quel est l'entier, qui, additionné à r donne 0 ?

Réponse : l'opposé de r.

Donc le complément à 2^4 de r (ou tout simplement **complément à** 2) de r est effectivement bien **son opposé**.

De façon encore plus générale, le complément à 2^n de r est $2^n - r$.

On obtient $r + 2^n - r = 100.....00$, mais comme on ne tient pas compte du n + 1 ème bit, cela fait 0.

Remarque : Dans le tableau précédent, on n'a que 15 arrangements de 0 et de 1 sur 16 arrangements possibles. Le 16 ème arrangement **permet d'ajouter -8 qui se code avec 1000**. En effet c'est le résultat par exemple de l'addition (-4) + (-4) (c'est à dire 1100 + 1100 en ignorant le 5 ème bit). Mais on note aussi que son opposé +8 ne pourra de toutes façons pas être mis dans le tableau puisque toutes les possibilités pour les entiers positifs ont déjà été utilisées.

3) Si on travaille avec des mots de 6 bits, quel est le codage de -10 ?

Réponse :

4) Si on travaille avec des mots de 8 bits (c'est à dire des octets), quel est le codage de -10 ?

Réponse :

Restons avec des mots de 8 bits. Comme on doit garder le bit de poids fort pour le signe (0 si c'est un nombre positif et 1 si c'est un nombre négatif) on n'a que 7 bits pour le nombre lui-même. On va ainsi de 0000 0000 à 0111 1111 pour les nombres positifs.

5) Quelle est la valeur décimale du nombre $0111\,1111_2$?

Réponse :

On peut donc, sur un octet, écrire les nombres de 0 à +127.

6) Quel que soit le nombre de bits n dont on dispose, la suite de bits tous égaux à 1 (par exemple $1111\,1111$ si on travaille avec des octets ou $1111\,1111\,1111\,1111$ si on travaille sur 16 bits) représente toujours le même nombre. Quel est ce nombre écrit en base 10 ?

Réponse :

7) Dans les négatifs, jusqu'où peut-on descendre avec des mots de longueur 8 bits ? indication : considérez $1000\,0000_2$. C'est un nombre négatif puisque son bit de poids fort est 1. Mais lequel ?

Réponse :

Conclusion : Avec $n = 8\,bits$ on peut écrire les entiers depuis -128 jusqu'à $+127$.

- De façon générale, si on dispose de $n\,bits$, on peut coder les entiers depuis -2^{n-1} jusqu'à $+2^{n-1}-1$.

8) Quel est le plus petit nombre entier relatif écrit en base 10 qu'on peut représenter si on travaille sur 32 bits (c'est à dire 4 octets) ?

Réponse :

9) Quel est le plus grand nombre entier relatif écrit en base 10 qu'on peut représenter si on travaille sur 32 bits ?

Réponse :

1.2 Programmation

Lisez le paragraphe **Programmation** p. 98 et en haut de la p. 99

- Voici un programme qui n'aura à manipuler que des nombres **a** à virgule flottante (on dit plus rapidement des nombres flottants) :

In [ ]:
```
from time import time

start_time = time()
for i in range(50000):
    a = 1.001**i  # Calcul des 50000 premières puissance du flottant a
end_time = time()

print(" a est de type ", type(a))
print(" Durée de la boucle en secondes : ", end_time - start_time)
```

10) a) Quelle est la durée du calcul des 50000 premières puissances du nombre flottant 1.001 ?

Réponse :

- Voici un programme qui n'aura à manipuler que des nombres **a** entiers :

In [ ]:
```
from time import time

start_time = time()
for i in range(50000):
    a = 2**i  # Calcul des 50000 premières puissance de l'entier a
end_time = time()

print(" a est de type ", type(a))
print(" Durée de la boucle en secondes : ", end_time - start_time)
```

10) b) Quelle est la durée du calcul des 50000 premières puissances du nombre entier 2 ?

Réponse :

11) Le second programme prend combien de fois plus de temps à s'exécuter que le premier ?

Réponse :

- Python est un langage de programmation qui s'adapte automatiquement à la taille des nombres entiers qu'il doit manipuler. Ce n'est pas le cas de tous les langages.

Test en langage C :

1. Créez, par un clic droit, sur votre Raspberry, dans votre dossier "Documents" un sous dossier "langage_c".

1. Cliquez sur la framboise en haut à gauche, allez dans programmation puis ouvrez **Geany** (Geany est un éditeur de texte pour programmeur).

1. Dans Geany, allez dans le menu **Fichier** et choisissez **Enregistrer sous...** Déplacez-vous dans le dossier votre_username/Documents/langage_c et enregistrez ce nouveau fichier sous le nom test_depassement.cpp

cpp est l'extension des *fichiers sources* (c'est à dire des fichiers écrits *en langage compréhensible par des humains*) en langage C++.

1. Saisissez dans **Geany** les 13 lignes du programme en langage C++ donné à la p. 99 en respectant les indentations. La différence entre le langage C et le langage C++ est que ce dernier est orienté objet. Par ailleurs, en langage C les commentaires s'écrivent entre les signes / *et* / tandis qu'en C++ on met simplement en début de ligne de commentaire le signe //.

```
#include <iostream>
using namespace std;

int main(){
    unsigned short a;
    a = 65500 + 40;
    cout<<"Valeur de a: "<<a<<endl;
    a = 32*2048;
    cout<<"Valeur de a: "<<a<<endl;

    short b;
    b = 32000 + 1000;
    cout<<"Valeur de b: "<<b<<endl;
    return 0;
}
```

Terminez en cliquant sur Fichier / Enregistrer.

Quittez Geany.

Remarques :

- La directive **#include < iostream >** sert à gérer les flux d'entrée et sortie
- **using namespace std;** permet d'utiliser l'espace de nom standard qui contient par exemple cout (lire "c out") pour l'affichage.
- Le coeur du programme est la fonction **main** (c'est à dire la fonction **principale** en anglais) dont le nom est précédé de **int** qui est le type de la valeur renvoyée par la fonction, suivie de parenthèses (puisque c'est une fonction). La fonction **main()** contient très souvent des appels à d'autres fonctions. Le corps de la fonction est écrit entre deux accolades.
- Notez qu'en langage C, **chaque instruction se termine par un point virgule** (*semi-colon* en anglais).
- Le type des variables doit obligatoirement être écrit au moment de leur déclaration.

unsigned short est le type entier *non signé court* (Les entiers non signés de type unsigned short sont codés sur 16 bits : de 0 à $2^{16} - 1 = 65535$).

short est le type entier *signé court* (Les entiers signés de type short sont codés sur 16 bits : de $-2^{15} = -32768$ à $2^{15} - 1 = 32767$).

$a = 65000 + 40 = 65540$: On sort de la plage des entiers de type *unsigned short*.

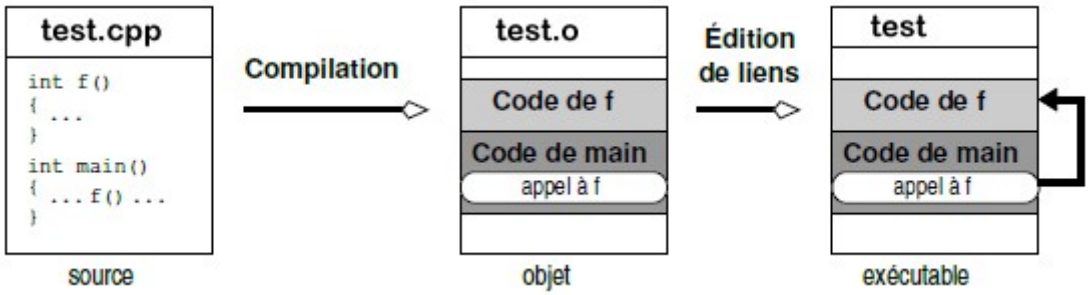
$a = 32 * 2048$: On est tout juste **en dehors** de la plage des entiers de type *unsigned short*.

$b = 32000 + 1000 = 33000$: On est **en dehors** de la plage des entiers de type *short*.

- Il ressort de cela que :
  - l'affichage n°1 ne peut pas donner la valeur prévue 65540 (en dépassement de 4 par rapport à la valeur 65536 où commence le débordement d'un bit supplémentaire)
  - l'affichage n°2 ne peut pas donner la valeur prévue 65536 (en dépassement de 0 par rapport à la valeur 65536 où commence le débordement d'un bit supplémentaire)
  - l'affichage n°3 ne peut pas donner la valeur prévue 33000

1. Compilation.

Le langage C est compilé, c'est à dire que le fichier source en langage C ne peut pas être exécuté directement. Le compilateur (qui est un programme) va traduire *le fichier source* **test_depassement.cpp** en *un fichier objet* **test_depassement.o** qui contient des instructions en langage machine qui lui-même va être traduit par l'éditeur de liens en *un fichier exécutable* **test_depassement**.



En réalité, les commandes que vous allez saisir dans le terminal de votre Raspberry vont transformer directement *le fichier source* **test_depassement.cpp** en *un fichier exécutable* **test_depassement**.

- Ouvrez un terminal et tapez la commande

```
$ pwd
```

pour voir dans quel dossier vous êtes.

- Placez-vous dans :

```
/home/mon_username/Documents/langage_c
```

- Pour vérifier la présence du fichier **test_depassement.cpp**, tapez la commande :

```
/home/mon_username/Documents/langage_c $ ls
```

Vous devriez voir son nom.

On veut à présent compiler (c'est à dire traduire en langage machine) le fichier source **test_depassement.cpp**

- Saisissez la commande de compilation g++ (g++ fait partie de GNU Compiler Collection est un ensemble de compilateurs libres) suivante :

```
$ sudo g++ test_depassement.cpp -o test_depassement
```

Des messages d'alerte de dépassements de capacité dûs à des entiers trop grands sont signalés par le compilateur; Vous n'en tenez pas compte.

1. Exécution du programme test_depassement

- En passant par l'explorateur de fichiers (qui a pour icône un dossier jaune) vérifiez la présence dans le dossier Documents/langage_c de *deux* fichiers :

L'un est le fichier source écrit en C++ avec l'extension .cpp

L'autre est le fichier exécutable qui porte le même nom, mais sans extension.

- On va exécuter le fichier exécutable **test_depassement**

Pour cela saisissez la commande :

```
$ ./test_depassement
```

12) Quel est le résultat (faux) de a = 65500 + 40 ?

Réponse :

13) Quel est le résultat (faux) de a = 32 * 2048 ?

Réponse :

14) Quel est le résultat (faux) de b = 32000 + 1000 ?

Réponse :

15) Expliquez pour quelle raison en langage C, une variable de type short égale à 32000 + 1000 prend la valeur -32536.

Réponse :

2. Nombres réels

Lisez le paragraphe **Nombres réels** en haut de la p. 105

16) Les nombres à virgule flottante servent à représenter les réels. Parmi les nombres réels suivants, lesquels ne sont représentés **que d'une manière approchée** par des nombres à virgule flottante :

```
-4.25
 0.1
 5/8
 8/5
 3
 pi
```

Réponse :

2.1 Représentation

2.1.1 Impossibilité de représenter certains réels

Lisez le paragraphe **Représentation** p. 105, p. 106 et en haut de la p. 107

17) De quel signe est un nombre dont le bit de signe est 1 ?

Réponse :

Comment écrire un nombre à virgule en binaire ?

Soit par exemple à écrire 5,1875 en binaire.

Il nous faut déjà représenter 5. Pour cela on sait que c'est 101.

Comment représenter le " ,1875 " ?

- On multiplie 0,1875 par 2 : 0,1875 x 2 = 0,375. On obtient 0,375 que l'on écrira ***0*** + 0,375 (ce 0 est le premier chiffre après la virgule).
- On multiplie 0,375 par 2 : 0,375 x 2 = 0,75. On obtient 0,75 que l'on écrira ***0*** + 0,75 (ce 0 est le deuxième chiffre après la virgule).
- On multiplie 0,75 par 2 : 0,75 x 2 = 1,5. On obtient 1,5 que l'on écrira ***1*** + 0,5 (ce 1 est le troisième chiffre après la virgule).
- On multiplie 0,5 par 2 : 0,5 x 2 = 1,0. On obtient 1,0 que l'on écrira ***1*** + 0,0 (ce 1 est le quatrième chiffre après la virgule).

La partie décimale est nulle donc on arrête.

Finalement :

$5,1875_{10} = 101,0011_2$

- Vérifions :

poids	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
$x =$	1	0	1,	0	0	1	1

D'où :

$x = 1 \times 2^2 + 1 \times 2^0 + 1 \times 2^{-3} + 1 \times 2^{-4}$

$x = 4 + 1 + 0,125 + 0,0625$

$x = 5,1875$

- De même qu'en décimal, on utilise la notation scientifique - par exemple 137,414 s'écrit $1,37414 \times 10^2$, on utilisera $x = 101,0011_2 = 1,010011 \times 2^2$. $\times 2^2$ correspond à décaler la virgule de deux rangs vers la droite.

- Autre exemple :

$y = 0,00011101_2 = 1,1101 \times 2^{-4}$. $\times 2^{-4}$ correspond à décaler la virgule de quatre rangs vers la gauche.

18) Pour le nombre x de l'exemple ci-dessus, donnez sa mantisse m et son exposant p.

Réponse :

19) Pour le nombre y de l'exemple ci-dessus, donnez sa mantisse m et son exposant p.

Réponse :

20) a) Dans quel intervalle semi-ouvert se trouve toujours la mantisse m ?

Réponse :

Mantisse infinie

De même qu'on connait des nombres réels dont l'écriture décimale est infinie (par exemple 1/3 = 0,3333333...) on peut avoir des nombres réels dont l'écriture binaire à virgule est infinie.

Exemple :

Comment représenter en binaire le nombre qui s'écrit " 0,1 " en décimal (un dixième)?

- On multiplie 0,1 par 2 : 0,1 x 2 = 0,2. On obtient 0,2 que l'on écrira ***0*** + 0,2 (ce 0 est le premier chiffre après la virgule).
- On multiplie 0,2 par 2 : 0,2 x 2 = 0,4. On obtient 0,4 que l'on écrira ***0*** + 0,4 (ce 0 est le deuxième chiffre après la virgule).
- On multiplie 0,4 par 2 : 0,4 x 2 = 0,8. On obtient 0,8 que l'on écrira ***0*** + 0,8 (ce 0 est le troisième chiffre après la virgule).
- On multiplie 0,8 par 2 : 0,8 x 2 = 1,6. On obtient 1,6 que l'on écrira ***1*** + 0,60 (ce 1 est le quatrième chiffre après la virgule).
- On multiplie 0,6 par 2 : 0,6 x 2 = 1,2. On obtient 1,2 que l'on écrira ***1*** + 0,20 (ce 1 est le cinquième chiffre après la virgule)
- On multiplie 0,2 par 2 : 0,2 x 2 = 0,4. On obtient 0,4 que l'on écrira ***0*** + 0,4 (ce 0 est le deuxième chiffre après la virgule).

On est retombé sur 0,2 et donc on commence à tourner en rond dans une boucle infinie. Donc on arête.

Finalement :

$$x = 0,1_{10} = 0,00011001100110011....{}_2$$

Donc ce nombre 0,1 (un dixième) qui s'écrit de façon exacte en décimal ne peut que s'écrire d'une façon approchée en binaire. Il ne pourra donc pas être représenté de façon exacte dans un ordinateur.

Utilisons la norme IEEE754 (IEEE pour Institute of Electrical and Electronics Engineers) qui précise que pour un système utilisant des mots de 64 bits :

- 1 est réservé pour le signe. C'est le bit s.
- 11 sont réservés pour l'exposant décalé e (tel que e = p + 1023). De cette manière l'exposant e est toujours positif.
- 52 sont réservés pour la mantisse tronquée m' (tel que m' = m - 1). De cette manière on gagne 1 bit : le nombre à gauche de la virgule, qui, dans toute mantisse est toujours égal à 1.

20) b) Pour le nombre x = 0,1 en décimal de l'exemple ci-dessus, donnez son bit de signe s.

Réponse :

21) Pour le nombre x = 0,1 en décimal de l'exemple ci-dessus, donnez sa mantisse m.

Réponse :

22) Pour le nombre x = 0,1 en décimal de l'exemple ci-dessus, donnez sa mantisse tronquée m'.

Réponse :

23) Quel est l'intérêt d'utiliser la mantisse tronquée m' plutôt que la mantisse m ?

Réponse :

24) Pour le nombre x = 0,1 en décimal de l'exemple ci-dessus, donnez son exposant p.

Réponse :

25) Pour le nombre x = 0,1 en décimal de l'exemple ci-dessus, donnez son exposant décalé e.

Réponse :

26) Dans quel intervalle fermé se trouve toujours l'exposant décalé e ?

Réponse :

27) Donnez une valeur approchée du plus grand nombre flottant codable sur 64 bits (tous les bits sont à 1 sauf le bit de signe)

Réponse :

28) Ecrivez dans la cellule ci-dessous un programme en Python qui calcule et affiche s le plus grand nombre flottant codable sur 64 bits

In [ ]:
```
s = 0
for i in range(971, 1024):
    s = s + 2.0 ** i

print(s)
t = s * 2
print(t)
```

29) Que se passe-t-il si on calcule un nombre flottant t plus grand que s (essayez de calculer t = 2 * s) ?

Réponse :

Des erreurs parfois avec les nombres à virgule flottante

- On a vu que le réel $0,1_{10}$ a une infinité de décimales en écriture binaire. Or, $0,1$ est égal en binaire à $1,10011001100\ldots \times 2^{-4}$. Sa mantisse comporte 52 bits qui représentent les chiffres à droite de la virgule. Ce qui fait que le 53 ème bit à droite de la virgule ne peut pas exister et donc le bit de poids $2^{-57} \approx 10^{-17}$ ne peut pas exister.
- La représentation en machine du nombre de type *float* 0.1 contient donc **une erreur d'arrondi** de l'ordre de 10^{-17}.
- Par ailleurs, si la partie entière du nombre à virgule flottante est plus grande que 1, plusieurs bits sont nécessaires pour la représenter, ce qui entraine **une troncature** des bits de poids les plus faibles.

- On retiendra que l'on ne peut pas tester l'égalité de deux nombres à virgule flottante, le résultat étant incertain car il dépend à la fois des erreurs d'arrondi et de troncature.

- La bonne pratique est de tester si deux nombres à virgule flottante sont suffisamment proches en utilisant **la valeur absolue (toujours positive donc) de leur différence**.

- Voici deux programmes :

In [ ]:
```
def test_egal_1(a, b, c):
    somme = a + b
    if somme == c:
        print("a + b est egal a c")
    else:
        print("a + b est different de c")

a = 0.1
b = 0.2
c = 0.3

test_egal_1(a, b, c)
```

In [ ]:
```
# Importation de la fonction valeur absolue depuis la bibliotheque math.
from math import fabs

def test_egal_2(a, b, c):
    somme = a + b
    # On teste un eventuel ecart beaucoup plus grand que les erreurs d'arr
ondi.
    if fabs(somme - c) < 10**-14:
        print("a + b est egal a c")
    else:
        print("a + b est different de c")

a = 0.1
b = 0.2
c = 0.3

test_egal_2(a, b, c)
```

30) Testez les deux programmes précédents. Que pouvez-vous conclure ?

Réponse :

2.1.2 Exemples

Lisez le paragraphe **Exemples** en bas de la p. 107 et en haut de la p. 108

- Le codage de -0,375 est : Le bit de signe s = 1 puisque c'est un nombre négatif.
- Pour 0,375 :

$0,375 \times 2 = 0,75$. On a comme premier bit après la virgule un 0.

$0,75 \times 2 = 1,5$. On a comme deuxième bit après la virgule un 1.

$0,5 \times 2 = 1,0$. On a comme troisième bit après la virgule un 1.

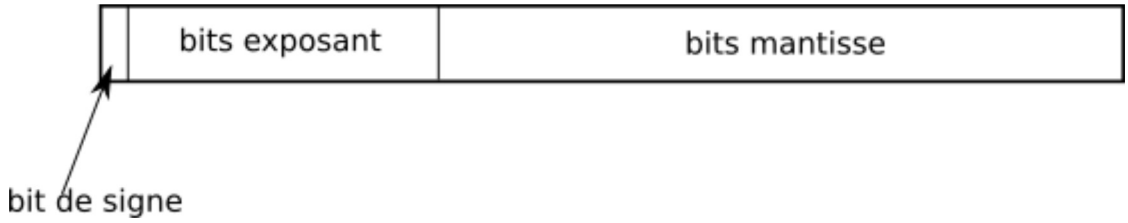
$0,0 \times 2 = 0,0$. On a comme quatrième bit après la virgule un 0.

On s'arrête car on n'a que des 0.

- Donc 0,375 s'écrit en binaire : $0,011000\ldots$ soit $1,1000\ldots \times 2^{-2}$

Donc on a $m \times 2^p$ avec la mantisse $m = 1,1000\ldots$ et la puissance $p = -2$

Et donc la mantisse tronquée est $m' = 1000\ldots$ et l'exposant décalé
$e = -2 + 1023 = 1021$ soit en binaire sur 11 bits $e = 011\,1111\,1101$



- Ainsi le nombre $-0,375$ a comme codage sur 64 bits :

s (1 bit)	e (11 bits)	m' (52 bits)
1	011 1111 1101	1000 00000000 00000000 00000000 00000000 00000000 00000000

31) Quel est le codage sur 64 bits du nombre -4,5 ?

Réponse :

- Bit de signe s =
- Exposant décalé e =
- Mantisse tronquée m' =

32) Quelle est l'écriture décimale du nombre x ainsi codé : ?

s (1 bit)	e (11 bits)	m' (52 bits)
1	011 1111 1110	1000 00000000 00000000 00000000 00000000 00000000 00000000

Réponse :

2.2 Calculs

2.2.1 L'écriture du nombre détermine son type

Lisez le paragraphe **Calculs** au milieu de la p. 108

33) Dans la cellule en-dessous de celle-ci (qui est une cellule où vous pouvez saisir du code Python) vous donnerez à x les différentes valeurs présentes dans le tableau ci-après et vous remplacerez les pointillés par le type de la variable x (int ou float).

In [ ]:
```
x = -2.15e-4
type(x)
```

x	de la forme	type
3 ; -5	p entier	int
3.5 ; 3.0 ; 3.	x nombre à virgule	.....
-5/2 ; 4/2	$p_1 \,/\, p_2$ quotient d'entiers	.....
5//2 ; 4//2	$a \,//\, b$ quotient de la division euclidienne (ou division " entière ")	.....
1e4 ; 2e-4 ; -2.15e-4	$m \times 10^p$ notation scientifique	.....

2.2.2 Quelques précautions

Lisez le paragraphe **Quelques précautions** en bas de la p. 108 et en haut de la p. 109

34) Pour la machine, l'expression 1e15 + 1 == 1e15 est-elle vraie ou fausse ?

In [ ]:
```
1e15 + 1 == 1e15
```

Réponse :

35) Pour la machine, l'expression 1e16 + 1 == 1e16 est-elle vraie ou fausse ?

In [ ]:
```
1e16 + 1 == 1e16
```

Réponse :

36) Pour la machine, le nombre de type flottant 1.0e308 (dix puissance 308) est-il représentable ?

In [ ]:
```
x = 1.0e308
print (x)
```

Réponse :

37) Pour la machine, le nombre de type flottant 1.0e309 (dix puissance 309) est-il représentable ?

In [ ]:
```
x = 1.0e309
print (x)
```

Réponse :

38) Pour la machine, le nombre de type flottant 2.0**1023 (c'est à dire $2,0^{1023}$) est-il représentable ?

In [ ]:
```
x = 2.0**1023
print (x)
```

Réponse :

39) Pour la machine, le nombre de type flottant 2.0**1024 (c'est à dire $2,0^{1024}$) est-il représentable ?

In [ ]:
```
x = 2.0**1024
print (x)
```

Réponse :

40) Pour la machine, le nombre de type entier 2**1024 (l'entier 2 à la puissance 1024) est-il représentable ?

In [ ]:
```
n = 2**1024
print (n)
```

Réponse :

41) Lorsqu'on travaille avec des nombres à virgule flottante, à quoi faut-il faire attention pour éviter que le programme s'arrête à cause d'une erreur " OverflowError " ?

Réponse :

42) A l'aide du programme ci-dessous, complétez le tableau ci-après.

In [ ]:
```
def nombre_de_chiffres(n):
    """
    Renvoie le nombre de chiffres de l'ecriture en base 10 de l'entier n

    Parametre nomme :
    -----------------
        n : de type int

    Retourne :
    ----------
        c : de type int
        le nombre de chiffres en base 10.


    """

    chaine = str(n)  # Transforme un entier enchaine de caracteres.
    c = len(chaine)
    return c
```

In [ ]:
```
nombre_de_chiffres(2**10)
```

n	Nombre de chiffres en base 10
2^{10}	.....
2^{1024}	.....
2^{2500}	.....
2^{5000}	.....
2^{10000}	.....

3. Textes

Lisez le paragraphe **Textes** au milieu de la p. 109

43) Dans la cellule en-dessous de celle-ci (qui est une cellule où vous pouvez saisir du code Python) vous donnerez à ma_chaine les différentes valeurs présentes dans le tableau ci-après et vous remplacerez les pointillés par le type de la variable ma_chaine (int ou float ou str). Rappel : int pour integer (entier) ; float pour floating point (à virgule flottante) ; str pour string (chaine de caractères).

In [ ]:
```
ma_chaine = 3
type(ma_chaine)
```

ma_chaine	de la forme	type
3	p entier	int
1.6e-19 ; 6.02e23	x nombre à virgule	.....
'3' ; '6.02e23'	'p' entouré d'apostrophes	.....
"3" ; "1.6e-19"	"p" entouré de guillemets	.....
n13 p.68	caractères alpha numériques	.....
'n13 p.68'	'caractères alphanumériques' entourés d'apostrophes	.....
"n13 p.68"	"caractères alphanumériques" entourés de guillemets	.....

44) Citez les deux opérations pour lesquelles il faut connaître la manière dont sont représentés les caractères en machine.

Réponse :

3.1 Représentation

Lisez le paragraphe **Représentation** en bas de la p. 109 et en haut de la p110

45) Quand il s'agit définir l'ensemble des caractères à coder en machine, de quoi parle-t-on (donnez le nom complet en anglais et son abréviation) ?

Réponse :

46) Le codage américain ASCII (American Standard Code for Information Interchange) est codé sur 7 bits soit 128 codes différents théoriquement possibles depuis 000 0000 à 111 1111. En réalité, certains de ces codes ne servent pas. Combien de caractères différents (chiffres, lettres ou symboles) sont effectivement présents dans le jeu de caractères ASCII ?

Vous exécuterez le code Python ci-dessous pour connaitre la réponse.

In [ ]:
```
def imprime_code_ascii():
    for i in range(128):
        print('i = ', i, 'représente le caractère : ', chr(i))
    print('')

imprime_code_ascii()
```

Réponse :

47) En utilisant les résultats du programme Python précédent, donnez le code (en décimal) de la lettre A.

Réponse :

48) a)Quel est le code en hexadécimal de la lettre A ?

Réponse :

48) b) Quel est le code en binaire de la lettre A dans le jeu de caractères ASCII ?

Réponse :

49) Quel est le nom du caractère associé au code 97 (en écriture décimale) dans le jeu de caractères ASCII ?

Réponse :

50) Quel est le nom du caractère associé au code 61 (en écriture hexadécimale) dans le jeu de caractères ASCII ?

Réponse :

Exercice 1 - Décodage ASCII (codage des caractères sur 7 bits)

La table ci-dessous donne le code associé à chacun des caractères ASCII imprimables. Le code du caractère en hexadécimal s'obtient en écrivant le numéro de la ligne suivi du numéro de la colonne. Par exemple, la lettre M a pour code hexadécimal `4D`, c'est à dire `77` en décimal.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2	espace	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

51) À quel nom correspond la liste des codes ASCII suivants, donnés en hexadécimal ?

```
47 65 6F 72 67 65 20 42 6F 6F 6C 65
```

Réponse :

Codage ISO 8859-1 ou latin-1 (sur 8 bits)

52) Le jeu de caractères ASCII a été étendu pour former le jeu de caractères ISO 8859-1. Ce jeu, qu'on appelle aussi "latin-1" contient les mêmes caractères que dans ASCII pour la partie 0000 000 à 0111 1111, mais avec en plus des lettres accentuées dans la partie 1000 0000 à 1111 1111. Ainsi, il est codé sur 8 bits (un octet). Dans la table de caractères ISO 8859-1, le 'é' est codé en décimal par 233. Quel est le code de 'é' en hexadécimal dans la table de caractères latin 1 ?

Réponse :

- Contrôlez votre réponse dans la table ISO 8859-1 (ou 'latin 1') donnée ci-dessous :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8																
9																
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

53) Exécutez l'instruction de codage suivante, observez le résultat et lisez les explications qui suivent :

In [ ]:
```
def convertir_en_octets(chaine):
    """
    Convertit une chaine de caractères en octets en suivant la table ISO 8
859-1 (latin-1)

    Parametres nommes
    -----------------
        chaine : de type string
            Chaine de caracteres a encoder

    Retourne
    --------
        octet_utf : de type byte (octet)

    """

    octet_latin = chaine.encode("latin-1")
    return octet_latin


ma_chaine = 'Bonne fête de Noël'
mes_octets = convertir_en_octets(ma_chaine)
print(mes_octets)
type(mes_octets)
```

- Explication :
  - Toutes les lettres (et espaces) qui peuvent être encodées suivant la table de caractères ASCII le sont. C'est le cas pour 'Bonne f' ; 'te de No' et 'l'.
  - Les caractères (ici les e avec accents) qui ne sont pas dans le charset ASCII sont encodés sur 1 octet qui est dans la partie 1000 0000 à 1111 1111 de la table de caractères (charset) ISO 8859-1.

C'est le cas pour 'ê' qui est encodé par \xea Sachant que \x signifie "ce qui suite est en hexadécimal" on voit donc que 'ê' est encodé par l'octet ea.

C'est le cas aussi pour 'ë' qui est encodé par \xeb Sachant que \x signifie "ce qui suite est en hexadécimal" on voit donc que 'ë' est encodé par l'octet eb.

Caractère (type str)	**B**	**o**	**n**	**n**	**e**		**f**	**ê**	**t**	**e**		**d**	**e**		**N**	**o**	**ë**	**l**
Code utf-8 (type byte)	B	o	n	n	e		f	ea	t	e		d	e		N	o	eb	l
Code utf-8 (hexadecimal)	42	6f	6e	6e	65	20	66	ea	74	65	20	64	65	20	4e	6f	eb	l

Codage UTF-8 (sur 8 bits ou éventuellement 2 octets -ou même parfois 3 ou 4-)

- Cependant, dans les années 90 est apparue la nécessité d'étendre encore le jeu de caractères pour y inclure les autres symboles et langues du monde. Pensez par exemple au symbole de l'euro ou aux langues comme l'arabe ou le chinois. L'organisation privée à but non lucratif 'Consortium Unicode' fondée en 1991, a mis en place le système UNICODE qui utilise aujourd'hui des formats de table de caractères très répandus comme UTF-8 ou UTF-16.
- Pour la plage de caractères ASCII, UTF-8 est identique au codage ASCII et permet un jeu de caractères plus étendu. Toutefois, pour les textes français, UTF-8 peut nécessiter deux octets pour chaque caractère.

54) Exécutez l'instruction de codage suivante, observez le résultat et lisez les explications qui suivent :

In [ ]:
```
def convertir_en_octets(chaine):
    """
    Convertit une chaine de caractères en octets en suivant la table UTF-8

    Parametres nommes
    -----------------
        chaine : de type string
            Chaine de caracteres a encoder

    Retourne
    --------
        octet_utf : de type byte (octet)

    """

    octet_utf = chaine.encode("utf-8")
    return octet_utf


ma_chaine = 'Bonne fête de Noël'
mes_octets = convertir_en_octets(ma_chaine)
print(mes_octets)
type(mes_octets)
```

- Explication :
  - Toutes les lettres (et espaces) qui peuvent être encodées suivant la table de caractères ASCII le sont. C'est le cas pour 'Bonne f' ; 'te de No' et 'l'.
  - Les caractères qui ne sont pas dans le charset ASCII sont encodés sur 2 octets.

C'est le cas pour 'ê' qui est encodé par \xc3\xaa Sachant que \x signifie "ce qui suite est en hexadécimal" on voit donc que 'ê' est encodé par les deux octets c3 et aa.

C'est le cas aussi pour 'ë' qui est encodé par \xc3\xab Sachant que \x signifie "ce qui suite est en hexadécimal" on voit donc que 'ë' est encodé par les deux octets c3 et ab.

Caractère (type str)	**B**	**o**	**n**	**n**	**e**		**f**	**ê**		**t**	**e**		**d**	**e**		**N**	**o**	**ë**		**l**
Code utf-8 (type byte)	B	o	n	n	e		f	c3	aa	t	e		d	e		N	o	c3	ab	l
Code utf-8 (hexadecimal)	42	6f	6e	6e	65	20	66	c3	aa	74	65	20	64	65	20	4e	6f	c3	ab	6c

- Remarque :

On voit que l'encodage en **utf-8 prend plus de place** que l'encodage latin-1 (il faut deux octets au lieu d'un pour les e avec accent par exemple) mais son avantage est qu'il **permet d'encoder beaucoup plus de caractères** comme on l'a déjà dit.

55) A l'aide de la fonction convertir_en_octets, trouvez combien il faut d'octets pour coder en utf-8 le symbole €. Dans la zone de réponse qui suit, vous donnerez ce nombre ainsi que les octets utilisés.

In [ ]:
```
def convertir_en_octets(chaine):
    """
    Convertit une chaine de caractères en octets en suivant la table UTF-8

    Parametres nommes
    -----------------
        chaine : de type string
            Chaine de caracteres a encoder

    Retourne
    --------
        octet_utf : de type byte (octet)

    """

    octet_utf = chaine.encode("utf-8")
    return octet_utf


ma_chaine = '2€'
mes_octets = convertir_en_octets(ma_chaine)
print(mes_octets)
type(mes_octets)
```

Réponse :

56) A l'aide la fonction convertir_en_octets, décrivez ce qui se passe si on essaie de coder en ISO 8859-1 le symbole €.

In [ ]:
```
def convertir_en_octets(chaine):
    """
    Convertit une chaine de caractères en octets en suivant la table ISO 8
859-1 (latin-1)

    Parametres nommes
    -----------------
        chaine : de type string
            Chaine de caracteres a encoder

    Retourne
    --------
        octet_utf : de type byte (octet)

    """

..... a compléter vous-même .....
```

Réponse :

Exercice 2 - Décodage UTF-8

En UTF-8, le codage des caractères coïncide avec l'ASCII pour les 128 premiers caractères (table donnée plus haut). Les autres caractères sont représentés par plusieurs octets.

La série d'octets suivants, donnés en hexadécimal, a été relevée dans un fichier codé en UTF-8.

```
43 6F 64 C3 A9 20 65 6E 20 55 54 46 2D 38
```

57) Il contient uniquement des caractères de la table ASCII ainsi qu'un « é ». Quelle est la séquence d'octets qui représente le « é », et qu'est ce qui est inscrit dans le fichier ?

Réponse :


```
43 6F 64 C3 A9 20 65 6E 20 55 54 46 2D 38
```


58) Si ce même fichier avait été interprété en latin 1, qu'est ce qui se serait affiché ?

Réponse :

- En résumé, le schéma suivant provoque des erreurs d'affichage :

$$'mon\ message'\ de\ (str) \rightarrow \boxed{\text{Encodage en utf-8}} \rightarrow 'mes\ octets'\ (bytes) \rightarrow \boxed{\text{Decodage e}}$$

59) Faites quelques essais en utilisant les deux fonctions ci-dessous : l'une encode en UTF-8 et l'autre décode en ISO 8859-1. Remplacez les pointillées de 'mon_message' par une chaine de caractères de votre choix. Observez la chaine de caractères 'ma_chaine' en résultat.

In [ ]:

```
def convertir_en_octets(chaine):
    """
    Convertit une chaine de caractères en octets en suivant la table UTF-8

    Parametres nommes
    -----------------
        chaine : de type string
            Chaine de caracteres a encoder

    Retourne
    --------
        octet_utf : de type byte (octet)

    """

    octet_utf = chaine.encode("utf-8")
    return octet_utf




def convertir_en_chaine(octets):
    """
    Convertit des octets en une chaine en suivant la table ISO 8859-1

    Parametres nommes
    -----------------
        octets : de type byte
            octets a decoder

    Retourne
    --------
        chaine : de type str

    """

    chaine = octets.decode("latin-1")
    return chaine


mon_message = '.....'
mes_octets = convertir_en_octets(mon_message)
ma_chaine = convertir_en_chaine(mes_octets)
print(ma_chaine)
type(ma_chaine)
```

3.2 Gestion des fichiers textes en Python

3.2.1 Ouverture et fermeture d'un fichier

Lisez le paragraphe **Ouverture et fermeture d'un fichier** au milieu de la p. 110

60) Qu'est-il essentiel de faire lorsqu'on a utilisé par exemple dans un programme en Python l'instruction mon_fichier = open('nom_du_fichier', 'r') ?

Réponse :

3.2.2 Ecriture d'un fichier

Lisez le paragraphe **Ecriture et lecture** en bas de la p. 110 et en haut de la p. 111

61) Ouvrez à côté de la fenêtre du navigateur dans lequel vous travaillez actuellement, une fenêtre de l'explorateur de fichiers de façon à voir le contenu du dossier où vous avez enregistré ce jupyter-notebook

```
20pnsi_cours_05_entiers_relatifs_reels_et_caracteres.ipynb
```

Saisissez dans la cellule de code ci-dessous la commande :

```
mon_fichier = open('fichier.txt', 'w')
```

Au moment où vous exécutez cette commande Python open('fichier.txt', 'w') remarquez l'apparition dans le dossier où se trouve le fichier de ce jupyter-notebook du fichier.txt qui est en argument dans la commande open()

```
mon_fichier = open('fichier.txt', 'w')
```

In [ ]:

Saisissez dans la cellule de code ci-dessous la commande :

```
mon_fichier.write("J'écris dans un fichier\nA la fin je le ferme.")
```

In [ ]:

Saisissez dans la cellule de code ci-dessous la commande :

```
mon_fichier.close()
```

In [ ]:

62) En utilisant l'explorateur de fichiers, par un clic droit ouvrez-le avec Text Editor. Qu'y a-t-il d'écrit dans le fichier ?

Réponse :

63) Refermez Text Editor. Saisissez et exécutez dans la cellule de code ci-dessous la commande :

```
mon_fichier = open('fichier.txt', 'a')
```

In [ ]:

Saisissez dans la cellule de code ci-dessous la commande :

```
mon_fichier.write("Je continue.")
```

In [ ]:

Saisissez dans la cellule de code ci-dessous la commande :

```
mon_fichier.close()
```

In [ ]:

64) En utilisant l'explorateur de fichiers, par un clic droit ouvrez-le avec Text Editor. qu'y a-t-il d'écrit dans le fichier ?

Réponse :

65) Refermez Text Editor. Saisissez et exécutez dans la cellule de code ci-dessous la commande. Attention, on fait exprès ici d'ouvrir fichier.txt avec l'attribut 'w' write et non l'attribut 'a' ajout pour voir ce qui va se passer :

```
a, b = 2, 5
mon_fichier = open('fichier.txt', 'w')
```

In [ ]:

Saisissez dans la cellule de code ci-dessous les commandes :

```
mon_fichier.write(str(a) + " x " + str(b) + " = " + str(a*b))
mon_fichier.close()
```

In [ ]:

66) En utilisant l'explorateur de fichiers, par un clic droit ouvrez-le avec Text Editor. qu'y a-t-il d'écrit dans le fichier ?

Réponse :

Refermez Text Editor

- En conclusion, il est important de se rappeler que si on ouvre un fichier **avec l'attribut 'w'** alors on **recommence à zéro**.
- Si on veut conserver ce qu'il y avait avant et simplement ajouter de nouvelles lignes, il faut ouvrir avec l'attribut 'a'.

3.2.3 Lecture d'un fichier

Lisez le paragraphe **Lecture** en bas de la p. 111, p.112 et p.113

67) Saisissez et exécutez dans la cellule de code ci-dessous la commande :

```
mon_fichier = open('fichier.txt', 'w')
```

In [ ]:

Saisissez et exécutez dans la cellule de code ci-dessous les lignes :

```
mon_fichier.write(str(5) + "\t" + str(8.3) + "\t" + str(1e-4) + "\n")
mon_fichier.write(str(8) + "\t" + str(32.7) + "\t" + str(1e2))
mon_fichier.close()
```

In [ ]:

En utilisant l'explorateur de fichiers, par un clic droit ouvrez-le avec Text Editor. Nous pouvons ainsi voir ce que produit la chaine de caractères spéciale de tabulation " \t ". Nous avions déjà vu la chaine de caractères spéciale de retour à la ligne " \n ". Dans Text Editor, faites avancer le curseur vers la droite ou vers la gauche plusieurs fois avec les flèches de direction du clavier. On constate que les tabulations sont bien présentes.

Refermez Text Editor.

- Lecture de fichier et récupération des données tabulées en 3 colonnes sur deux lignes :

68) Refermez Text Editor. Saisissez et exécutez dans la cellule de code ci-dessous la commande. Attention, on ouvre fichier.txt avec l'attribut 'r' read pour aller chercher les données qui s'y trouvent :

```
mon_fichier = open('fichier.txt', 'r')
for ligne in mon_fichier:
    liste = ligne.rstrip().split("\t")  # Supprime le caractere "\n" ET separ
e les 3 elements selon les "\t".
    a, b, c = [float(x) for x in liste]  # convertit les 3 elements de type s
tr en type float.
    print(a, b, c)

mon_fichier.close() # On n'oublie pas de refermer mon_fichier, même en cas de
lecture seulement.
```

In [ ]:

- En conclusion, on voit ici comment récupérer les données qui ont été écrites dans un fichier.
- On parcourt le fichier **mon_fichier** ligne par ligne à l'aide d'une boucle for.
- Sur chaque ligne il peut y avoir plusieurs données séparées par le caractère spécial de tabulation **\t**
- On récupère alors les données grâce à la méthode **.split("\t")**.
- On utilise aussi la méthode **.rstrip()** pour ne pas être gêné par le caractère spécial de saut de ligne **\n**
- Enfin, le stockage des données dans un fichier représente un stockage permanent (qui demeure quand on éteint l'ordinateur). En effet les fichiers sont écrits sur la mémoire de masse (Disque dur ou SD card) tandis que le stockage dans des listes au cours de l'exécution de Python n'est que temporaire, puisque la mémoire RAM de l'ordinateur est effacée à chaque extinction du système d'exploitation.

69) Où interviennent les notions d'encodage lorsqu'on écrit un ficher texte ? La réponse est que l'encodage est déterminé par les caractères qu'on utilise. Pour le vérifier, saisissez ci-dessous les 6 lignes de code qui sont en haut de la page 112.

In [ ]:

Vous avez obtenu deux fichiers texte : test1.txt qui ne comporte que des caractères alpha numériques simples et test2.txt qui comporte des caractères qui n'existent pas dans le code ASCII. Il devra donc être codé en UTF-8. Vérifiez-le en ouvrant un terminal (icône en forme de rectangle noir, dans la barre d'outils, en haut du bureau de votre Raspberry).

```
Dans le terminal, allez dans le dossier où vous avez votre jupyter-notebook.
Vous y trouverez aussi les deux fichiers test1.txt et test2.txt.
Saisissez les lignes de commande :

$ file test1.txt

$ file test2.txt
```

70) Nous avons déjà vu à la question 54) que le texte 'Bonne fête de Noël' est codé dans la charset (table de caractères) UTF-8 par la suite d'octets 42 6f 6e 6e 65 20 66 c3 aa 74 65 20 64 65 20 4e 6f c3 ab 6c, les caractères avec accent prenant deux octets à eux tous seuls. On va retrouver ce résultat en écrivant en mode binaire (avec l'attribut 'wb') cette chaine de caractères dans le fichier test3.txt Copiez et exécutez ces lignes de code dans la zone de code ci-dessous :

```
ma_chaine = 'Bonne fête de Noël'
ma_chaine = ma_chaine.encode('utf-8')
f = open('test3.txt', 'wb')
f.write(ma_chaine)
f.close()
```

In [ ]:

Vous avez obtenu le fichier texte : test3.txt qui est codé en UTF-8 puisqu'il comporte des lettres avec des accents. Vérifiez-le dans un terminal.

```
Dans le terminal, allez dans le dossier où vous avez votre jupyter-notebook.
Vous y trouverez le fichier test3.txt
Saisissez les lignes de commande :

$ file test3.txt

$ hexdump -C test3.txt
```

71) Retrouvez-vous les mêmes octets (en écriture hexadécimale) ?

Réponse :