DS N°2 - REPRÉSENTATION DES DONNÉES / PYTHON
Représentation des données, programmation en langage Python.

Partie 1 - Représentation des données

Exercice 1

Donner la repésentation **en suivant la norme du complément à 2** des entiers suivants : -42 et -120.

La **norme IEEE 754** définit un **format standardisé** qui vise à unifier la représentation des nombres flottants. Cette norme propose **deux formats de représentation** : un format **simple précision sur 32 bits** et un format **double précision sur 64 bits** :

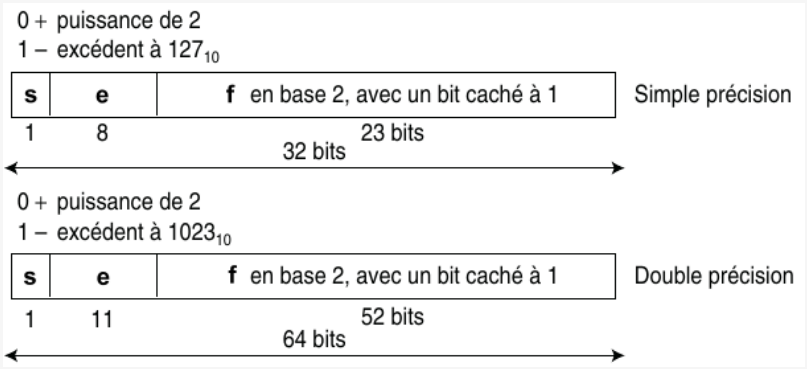

Figure 1: Formats IEEE754

En **simple précision**, la **chaîne de 32 bits** représentant le nombre est décomposée en :
- 1 bit de signe indiquant le **signe** de la mantisse,
- 8 bits pour **l'exposant**,
- 23 bits pour le codage de la **fraction de la mantisse**.

En **double précision**, la **chaîne de 64 bits** représentant le nombre est décomposée en :
- 1 bit de signe indiquant le **signe** de la mantisse,
- 11 bits pour **l'exposant**,
- 52 bits pour le codage de la **fraction de la mantisse**.

On rappelle que l'**exposant** est stocké sous une forme **décalée**. Ce décalage est de $2^{n-1} - 1$, où n est le **nombre de bits** utilisé pour stocker l'exposant.
En **simple précision**, ce décalage est donc de $+127_{10}$.
En **double précision**, il est de $+1023_{10}$

On rappelle également que pour économiser **1 bit** de précision pour représenter la **mantisse**, on ne représente que la partie après la virgule, appelée *fraction* de la **mantisse**.

Voici un tableau récapitulant **lécriture normalisée** d'un **nombre réel** selon la **norme IEEE 754** :

	exposant (e)	fraction (f)	forme normalisée
32 **bits**	8 *bits*	23 *bits*	$(-1)^s \times 1, f \times 2^{(e-127)}$
64 **bits**	11 *bits*	52 *bits*	$(-1)^s \times 1, f \times 2^{(e-1023)}$

Exercice 2

Donner la représentation flottante en **simple précision** de $-32,375$.

Exercice 3

Donner la **valeur décimale** du **nombres flottant** suivant codé en **simple précision** :

- 0 10000101 00101110000000000000000

Le **codage ASCII** (*American Standard Code for Information Interchange*) est un **codage à 7 bits** qui permet donc de représenter des **caractères**. Chacun des **codes** associés à un **caractère** est donné dans une **table à deux entrées**, la première entrée codant la valeur du **quartet de poids faible** et la seconde entrée codant la valeur des **3 bits de poids fort** du code associé au caractère.

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	P
1	SOH	DC1	!	1	A	Q	a	Q
2	STX	DC2	«	2	B	R	b	R
3	ETX	DC3	#	3	C	S	c	S
4	EOT	DC4	$	4	D	T	d	T
5	ENQ	NAK	%	5	E	U	e	U
6	ACK	SYN	&	6	F	V	f	V
7	BEL	ETB	'	7	G	W	g	W
8	BS	CAN	(	8	H	X	h	X
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Figure 2: Table ASCII (Attention : sur la dernière colonne, les lettres sont en MINUSCULES)

Le caractère A est codé par la chaîne 1000001_2. soit le code hexadécimal 41_{16}.

Exercice 4

Combien de **caractères** peuvent être représentés en codage **ASCII** ?

Exercice 5

Donner le **codage ASCII** en **hexadécimal** et en **binaire** de la **chaîne de caractères** *Python* ci-dessous :
`"Salut !"`

Exercice 6

Indiquez les **valeurs affichées** après l'exécution des **deux instructions** suivantes dans un **interpréteur** Python :

```
>>> ord('%')
...
>>> chr(64)
...
```

Exercice 7

Écrire une fonction `printASCII(s)` qui **affiche** à l'écran les **codes ASCII** au **format hexadécimal** d'une **chaîne de caractères** `s` donnée.

Programmation Python

Exercice 1

Écrire une **fonction** `milieu(a, b)`, qui prend **deux entiers** `a` et `b`, et qui **renvoie** l'**entier du milieu** compris entre `a` et `b`.
Exemple : `milieu(1, 10)` renverrait `5`. `milieu(15, 30)` renverrait `22`.

Exercice 2

Écrivez une fonction `verifier_admission(age, moyenne, formation)` qui prend **trois paramètres** `age` (*int*), `moyenne` (*float*) et `formation` (*str*), et qui détermine si une personne est **admissible** à une **formation** donnée.

Les critères d'admission sont les suivants :
- L'âge doit être **supérieur ou égal** à `18`.
- La **moyenne** doit être **supérieure ou égale** à `12.0`.
- La **formation** doit être `"informatique"`, `"mathématiques"` ou `"physique"`.

Si toutes les conditions sont **remplies**, la fonction **renvoie** `True` (admis), sinon elle renvoie `False` (non admis).

Exercice 3

Écrire une **fonction** `pairs_impairs(nb, choix)`, qui prend un **entier** `nb` et une **chaîne de caractères** `choix` en **entrée**, et qui :

- **affiche** tous les entiers **pairs** de 0 à `nb` si `choix = "pairs"`,
- **affiche** tous les entiers **impairs** de 0 à `nb` si `choix = "impairs"`,
- **affiche** un message `"Choix incorrect"` sinon.

Exercice 4

Écrire un **programme** qui :

- génère un *nombre aléatoire* entre 1 et 100,
- demande à l'utilisateur de *saisir un nombre* entre 1 et 100,
- *tant que* l'utilisateur n'a pas trouvé le bon nombre :
  - si le *nombre à trouver* est *inférieur* au *nombre saisi*, on affiche `"C'est moins"`,
  - si le *nombre à trouver* est *supérieur* au *nombre saisi*, on affiche `"C'est plus"`.
- une fois que le nombre est trouvé, on affiche `"C'est gagné !"`.

On utilisera la fonction `randint` du module `random`.

```
1 from random import randint
2
3 ...
```

Exercice 5

Écrire une fonction `atteindre_objectif(objectif)` qui prend un **entier positif** `objectif` en **entrée**, et simule un processus pour atteindre cet **objectif** en incrémentant progressivement une valeur initiale de 0 par un nombre entier positif aléatoire entre 1 et 10. On affichera la progression à **chaque étape**, et on **renverra** finalement le **nombre total d'incréments** effectué.

Exemple :

```
1 >>> atteindre_objectif(20)
2 Progression actuelle : 0
3 Ajouté : 4
4 Progression actuelle : 4
5 Ajouté : 7
6 Progression actuelle : 11
7 Ajouté : 10
8 Progression actuelle : 21
9 Objectif atteint en 3 essais !
```