Les types de données simples et la chaîne de caractères

Table des matières

I. Comprendre les variables et la notion de « type »

Contexte

L'algorithmie va s'atteler à résoudre des problèmes à l'aide d'une suite d'opérations élémentaires. Nous voulons utiliser la puissance de calcul de la machine et l'algorithmie va nous permettre d'organiser nos idées pour lui faire comprendre ce que nous attendons d'elle. Cette puissance de calcul va nous aider, par exemple, à résoudre des problèmes scientifiques complexes, mais aussi effectuer des prévisions grâce à un gros volume de données. À chaque fois que le volume de données sera important, ou que les opérations seront complexes, l'ordinateur sera un allié de choix. Il ne faut pas confondre l'algorithmie avec la programmation. La programmation est la mise en application d'un algorithme.

Dans cette première partie, nous allons parler des types de données et des types de variables. Il s'agit d'une notion très importante puisqu'elle va nous permettre d'indiquer à la machine ce que représentent les données que nous lui demandons de manipuler. C'est grâce aux types qu'elle va savoir quelles opérations elle est autorisée à réaliser.

A. Qu'est-ce qu'une variable ?

Définition

Une variable va nous permettre de stocker temporairement une valeur dans la mémoire de l'ordinateur. Vous avez calculé un taux de croissance, un total de facture ou la solution d'une équation complexe, et vous voulez sauvegarder le résultat pour pouvoir le réutiliser. C'est ici que rentre en jeu la notion de « *variable* ». Une variable a donc deux fonctions essentielles :

- Stocker une valeur en mémoire,
- Pouvoir récupérer une valeur stockée préalablement.

Si on veut pouvoir accéder à la valeur d'une variable, il va falloir trouver un moyen d'y faire référence. Cette référence va se faire à travers le nom de la variable. Le choix du nom de la variable est laissé à la libre appréciation des développeurs. Dans la mesure du possible, il est judicieux de la nommer, de telle façon à comprendre immédiatement ce qu'elle contient :

- températureMoyenne
- distanceMin
- vitesseMax

Attention

Les langages de programmation imposent souvent des contraintes sur les noms des variables. Par exemple, ils doivent commencer par une lettre. Les noms des variables ne peuvent pas non plus être des mots-clés du langage (*« if », « then », « else », « for », « while »,* etc.). Cela risquerait de rendre votre programme incompréhensible.

Complément

Il ne faut pas non plus confondre les variables et les constantes, même si ces deux éléments partagent quelques propriétés. Une constante est une variable dont il n'est plus possible de modifier la valeur. Cela permet au développeur d'être certain que la valeur fixée au départ reste la même jusqu'à la fin du programme. Les constantes servent notamment à sauvegarder des constantes mathématiques comme : PI, e.

Nous allons voir qu'en plus de la valeur et d'un nom, une variable va être définie par son « *type* ». Le type va nous amener encore une information supplémentaire sur la nature de la donnée contenue dans la variable.

B. Quelles informations comprend la machine ?

Si on ouvre l'unité centrale d'un ordinateur, nous ne trouverons pas quelque chose qui ressemble à un cerveau humain, mais nous nous retrouverons en présence d'un enchevêtrement de câbles et de circuits électriques.

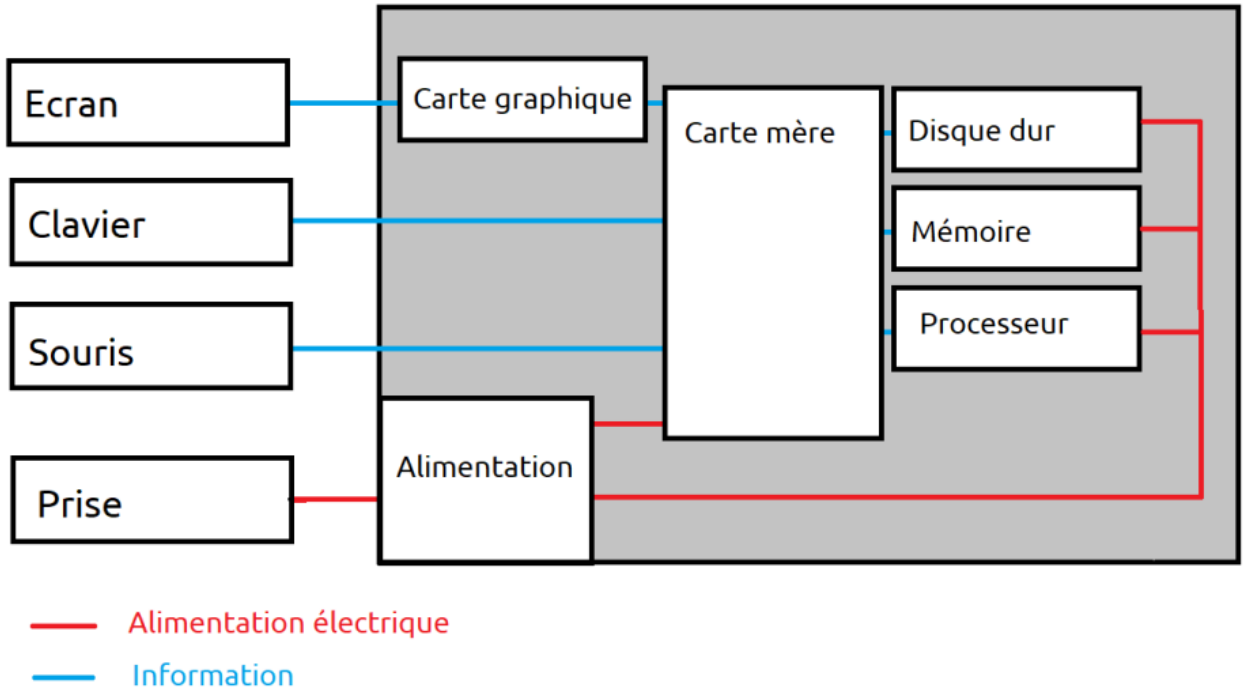


Schéma simplifié de l'unité centrale d'un ordinateur

La construction même de l'ordinateur va nous permettre de comprendre ce que la machine est capable de manipuler. L'information est véhiculée sous forme de signal électrique.

Nous ne pouvons faire passer par un câble électrique que deux informations différentes : soit le courant passe dans le câble, soit il ne passe pas. Cela semble bien peu, mais si nous utilisons deux câbles, nous pouvons faire passer 4 informations différentes.

Câble 1	Câble 2
Passe pas	Passe pas
Passe pas	Passe
Passe	Passe pas
Passe	Passe

Par convention, nous allons utiliser le 1 pour dire que du courant passe par le câble, et 0 pour dire qu'il ne passe pas. Une information devient donc une succession de 0 et de 1.

Câble 1	Câble 2
0	0
0	1
1	0
1	1

Remarque

À chaque fois que nous ajoutons un câble, le nombre d'informations différentes que nous pouvons propager est multiplié par 2. Le nombre d'informations différentes peut donc être exprimé à l'aide de puissance de 2 :

1 câble = 2 informationsn = 2^1

2 câbles = 4 informations = 2^2

3 câbles = 8 informations = 2^3

…

8 câbles = 256 informations = 2^8

Attention

L'information transmise par un câble est appelée « *bit* ». On comprend donc facilement les notions de 32 *bits* ou 64 *bits*. 32 *bits* permettent de propager 2^32 informations différentes alors que 64 *bits* permettent de propager 2^64 informations différentes. Les systèmes 64 *bits* ont donc une meilleure précision lorsqu'il s'agit de travailler avec des nombres.

8 câbles ou 8 *bits* représentent un octet. Un octet en anglais se dit « *byte* », qu'il ne faut donc pas confondre avec le *bit* (8 *bits* = 1 *byte* = 1 octet).

Exemple

Imaginons maintenant que nous aimerions représenter un nombre entier. La représentation binaire est parfaitement adaptée à cela. Nous pouvons donc associer à chaque nombre entier le nombre binaire qui lui correspond.

Décimal	Binaire
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Décimal	Binaire
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011

Décimal	Binaire
20	10100
21	10101
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101

L'ordinateur est donc naturellement fait pour manipuler des nombres. Si nous voulons utiliser des nombres entiers signés, il suffit de considérer que l'un des *bits* contient le signe (0 : nombre négatif ; 1 : nombre positif). Une représentation de 8 *bits* peut nous permettre de représenter 256 nombres positifs (de 0 à 255), ou 128 nombres négatifs (de -128 à -1), et 128 nombres positifs (de 0 à 127). Le fait d'utiliser un *bit* pour le signe limite donc à 7 *bits* la partie disponible pour représenter le chiffre.

Représenter un nombre réel va revenir à utiliser un nombre de bits pour représenter un exposant. Cette partie sera plus détaillée par la suite. Enfin, si la machine arrive à manipuler facilement les nombres, les caractères vont poser plus de problèmes. Mais là encore, l'ordinateur va convertir les caractères en nombres à l'aide d'une table de conversion : la table ASCII.

Nous venons de voir que l'ordinateur sait uniquement manipuler des nombres entiers, et il va convertir tous les éléments que nous lui fournissons en nombre entier.

Rappel

La compréhension du fonctionnement de la machine nous permet déjà de comprendre comment l'information est échangée, mais aussi pourquoi le binaire est si important pour les informaticiens. Nous verrons qu'en plus du binaire, d'autres bases ont leur importance, comme l'octal ou l'hexadécimal. Ces deux bases sont en fait également à base de puissance de 2 :

Octal : base 8 = 2^3

Hexadécimal : base 16 = 2^4

C. Les types donnent du sens aux nombres

Le type permet de donner une signification au contenu de la variable. Nous avons vu que la machine va convertir toutes les notions manipulées en nombre entier. Pour pouvoir faire le chemin en sens inverse, et retourner la valeur correcte au développeur, il est nécessaire de savoir de quel type de valeur il s'agit.

Exemple

En tant qu'être humain, nous arrivons facilement à faire la différence entre les différentes notions que nous manipulons. Toutefois, à certains moments, un complément d'information peut nous aider à comprendre ce que représente un chiffre.

Nombre	Signification
75000	Code postal
0102030405	Téléphone
12345123451234567890112	RIB

Sans cette information, il serait parfois difficile de définir de quel type d'information il s'agit. Il en est de même pour les variables. L'ordinateur a besoin de connaître le type manipulé pour savoir ce qu'il peut en faire. Par exemple, nous savons qu'il n'est pas possible d'additionner 2 caractères. En précisant le bon type à la machine, elle saura aussi comment réagir face à un caractère.

Si nous prenons la table ASCII, et que nous voulons convertir le caractère « *A* », nous allons voir que « *A* » se convertit en la valeur entière 65. Que le développeur stocke la valeur 65 ou 'A' dans une variable, le programme va stocker la représentation binaire de 65 en mémoire. Pour récupérer le caractère « *A* », il faut indiquer que ce que l'on attend est un caractère, et ceci se fait au travers des types.

Un type est donc juste une information sur la variable, qui permet de communiquer une information sur la nature de son contenu. En algorithmie, les variables que nous utilisons sont déclarées en en-tête de l'algorithme. Les types sont systématiquement donnés pour les variables que nous utilisons :

```
Variable :
 n : entier
 c : chaîne de caractères
 b : Booléen
 d :décimal

Début
…
Fin
```

Rappel

Nous détaillerons ces types dans la partie suivante. Pour l'instant, gardons à l'esprit qu'il existe des types simples :

- Entier : nombre entier, signé ou non ;
- Chaîne de caractères : suite de caractères ;
- Booléen : variable issue de l'algèbre booléenne valant soit VRAI, soit FAUX ;
- Décimal : nombre à virgule.

Nous verrons qu'il existe des types de variables plus complexes comme les tableaux. Enfin, « *la programmation orientée objet* » nous permettra de déclarer nos propres types. En entreprise par exemple, nous travaillons avec des clients, des fournisseurs, sur des articles, plutôt qu'avec des entiers, des réels et des booléens. La programmation orientée objet permet de développer des types, en adéquation avec les entités manipulées. Le but est de pouvoir définir une variable de type « *Client* », ou une variable de type « *Fournisseur* ».

Cela permet, dès le départ, de comprendre ce que nous pourrons faire avec ces variables. Les entiers pourront être additionnés, les chaînes de caractères concaténées (création d'une nouvelle chaîne de caractères contenant les deux chaînes mises bout à bout). Il sera donc plus facile de savoir comment manipuler les différentes données.

D. Typage implicite et typage explicite

Certains langages de programmation imposent un typage explicite, ce qui veut dire que vous êtes obligés de spécifier le type de la variable. Cela est vrai par exemple pour le C++ :

```
Int a ; // Déclaration d'une variable « a » de type entier (int : integer)
Double b ; // Déclaration d'une variable « b » de type réel (double)
```

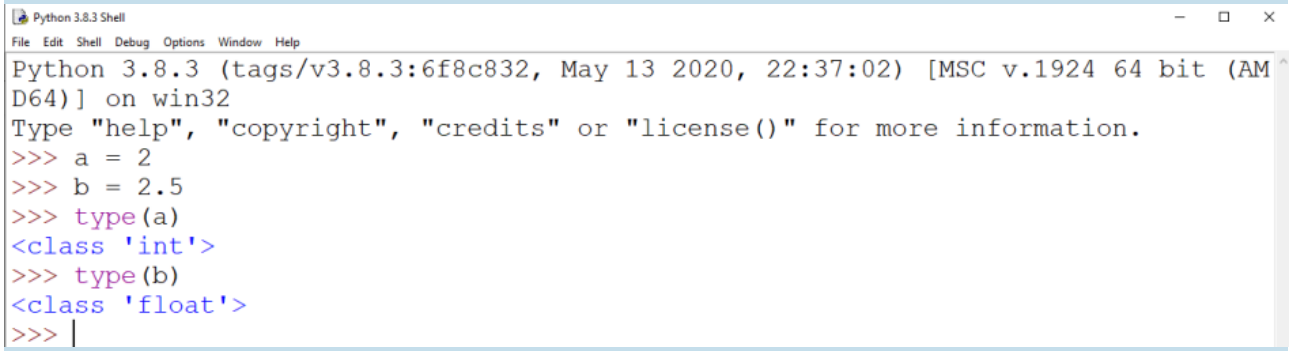
D'autres langages utilisent un typage implicite, ce qui veut dire que le type est déterminé à la première utilisation de la variable. C'est le cas de *Python* :

```
a = 2 # 2 étant un entier, Python considère que la variable qui va stocker cette valeur sera également de type entier (int : integer)
b = 2.5 # 2.5 étant un réel, Python considère que la variable qui va stocker cette valeur sera également de type réel (double)
```

Remarque

Ce n'est pas parce que le typage des variables est implicite que les variables ne sont pas définies par un type. Il s'agit juste de deux façons différentes d'associer un type à une variable. Pour preuve, vous pouvez utiliser la fonction type () en *Python* pour déterminer le type d'une variable. *Python* est donc bien un langage qui se repose sur les types de variables comme le C++.

Si vous installez *Python*, vous installez également IDLE qui est un interpréteur de commandes *Python*. Ouvrez IDLE, et tapez les commandes suivantes.



Python indique que « *a* » est un entier (*int*), alors que « *b* » est un réel (*float*). Il a donc su déterminer le type en fonction de la valeur de la variable.

Complément

Pour plus d'information sur l'installation de *Python*, rendez-vous sur le site officiel : https://www.python.org/downloads/

Exercice : Quiz

[solution n°1 p.23]

Question 1

Une variable permet de :

- Stocker une valeur et récupérer la valeur stockée
- Modifier la vitesse du processeur
- Ajouter de la mémoire à un ordinateur

Question 2

Quel nom puis-je donner à une variable ?

- 99
- puissanceMoteur
- while

Question 3

Quel est le seul type de données que peut manipuler un ordinateur ?

- Les nombres réels
- Les caractères
- Les nombres entiers

Question 4

Le typage implicite de certains langages de programmation :

- Détermine le type d'une variable en fonction de la valeur qui lui est affectée
- N'utilise pas la notion de type
- Utilise un type par défaut commun à toutes les variables du langage

Question 5

Quelles sont les bases issues de puissances de deux ?

- Décimal, binaire, hexadécimal
- Binaire, octal, hexadécimal
- Binaire, décimal, octal

III. Les types numériques

A. Les entiers

Rappel

L'entier est le type qui se rapproche le plus de la représentation machine. Nous avons vu dans la première partie que l'ordinateur sait avant tout manipuler des nombres, et cela vient de son architecture. Nous avons pris l'habitude de manipuler les nombres en base 10, alors que la machine se base sur la base 2, car elle transmet l'information au travers de câbles électriques.

Heureusement, il est facile de passer d'une base à l'autre, et cela fait l'objet du cours d'arithmétique du BTS SIO. Compter en base 10 revient à utiliser les premiers chiffres dans l'ordre : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. À ce niveau, nous avons utilisé tous les chiffres de la base 10. Il faut donc ajouter une dizaine et recommencer : 10, 11, etc. Une fois arrivés à 99, nous avons utilisé tous les chiffres disponibles au niveau des dizaines et des unités. Nous ajoutons donc une centaine : 100, 101, etc.

Compter en base 2 revient à compter en base 10, mais nous ne pouvons utiliser que les chiffres 0 et 1. Nous comptons donc 0, 1, et nous avons utilisé tous les chiffres disponibles en base 2. Il faut donc ajouter une dizaine pour passer à 10, puis à 11. À ce niveau-là, nous allons ajouter une centaine pour passer à 100. Comptez en base 2 revient à utiliser les mêmes règles qu'en base 10.

Exemple

Cette façon de compter nous permet rapidement d'avoir une correspondance entre les nombres en base 10, et les nombres en base 2. La calculatrice pour programmeur donne également la conversion des nombres décimaux en plusieurs bases (binaire, octale, hexadécimale).

Décimal	Binaire
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Décimal	Binaire
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011

Décimal	Binaire
20	10100
21	10101
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101

En algorithmie, nous ne faisons pas la distinction entre entier signé et entier non signé. Nous considérons que les entiers peuvent avoir un signe. La déclaration des variables se fait en en-tête de l'algorithme dans une partie dédiée :

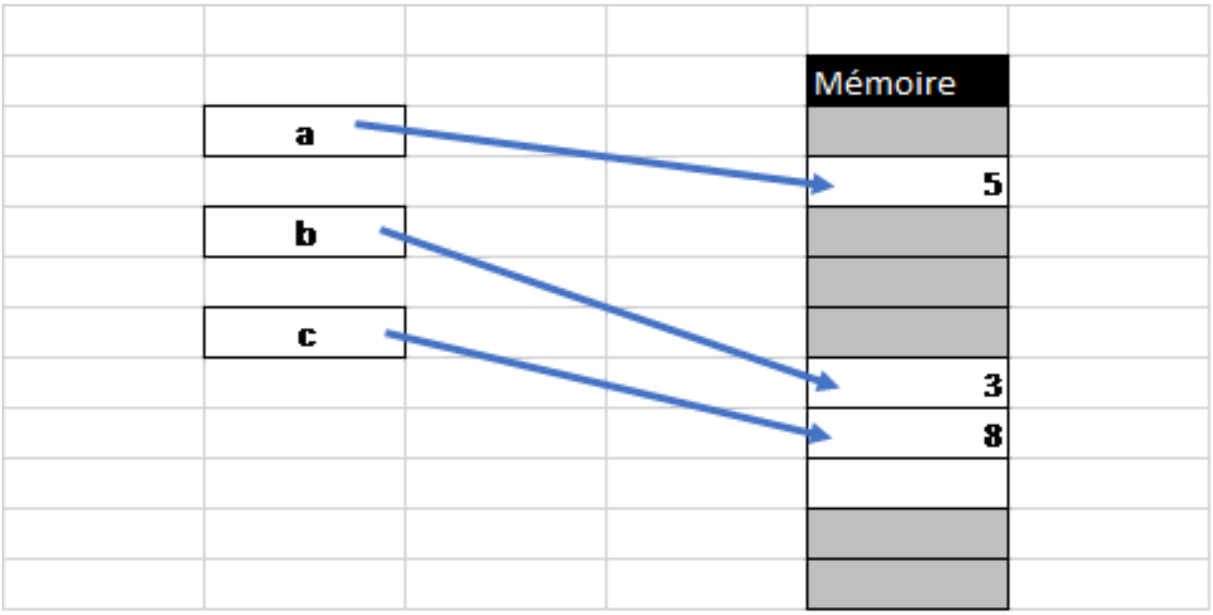
```
Variable :
 a, b, c : entier


Début
 a ← 5
 b ← 3
 c ← a + b
Fin
```

Ce premier algorithme est simple à comprendre. Nous déclarons 3 variables entières et nous leur donnons les noms : a, b et c. Nous signifions le début de notre algorithme avec le mot-clé « *Début* ». Nous affectons 5 à la variable « *a* ». Donc à partir de ce moment-là, à chaque fois que nous ferons appel à la variable « *a* », la valeur 5 sera retournée. Ensuite, nous affectons la valeur 3 à « *b* ». Là aussi, à partir de maintenant, à chaque fois que nous utiliserons la variable « *b* », la valeur 3 sera retournée. Enfin, nous affectons la variable « *c* » avec la somme de « *a* », et« *b* ». La somme de 3 + 5 = 8 est donc affectée à « *c* ».

La mémoire va stocker toutes les valeurs des variables. On peut se représenter la mémoire comme une grande étagère, dans laquelle il est possible de ranger les éléments à partir du moment où l'étagère est vide.

Voici l'état de la mémoire à la fin de l'algorithme :



Ce premier algorithme est assez simple, mais il met en place énormément de notions. Le fait de pouvoir affecter les valeurs entières 5 et 3 aux variables « *a* » et « *b* » vient du fait que les valeurs sont en adéquation avec les types de variables. L'addition entre « *a* » et « *b* » est également possible, car nous avons indiqué à la machine que les deux variables contiennent des entiers. Enfin, l'affectation de l'entier 8 à la variable « *c* » est également possible grâce au type entier de la variable « *c* ».

Conseil

Le typage des variables n'est pas seulement une contrainte, il va également permettre à la machine de comprendre ce qu'elle manipule, et potentiellement, aider le développeur en lui indiquant les opérations qui n'ont aucun sens (additionner un entier et un caractère).

B. Les réels

Attention

Un nombre réel permet de représenter un nombre à virgule. On pourrait se dire que finalement, le type entier est inutile, et qu'il serait beaucoup plus facile de n'utiliser que des nombres réels. Ce serait une erreur, et pour le comprendre, il faut une fois de plus, revenir à la représentation des nombres dans la mémoire de l'ordinateur.

Comme nous l'avons vu, la machine ne sait manipuler que des nombres entiers. Dans ce cas-là, comment fait-elle pour stocker un nombre réel ? L'ordinateur va procéder à une conversion, une fois de plus. Un nombre réel sur 32 *bits* sera représenté de la façon suivante :

1 bit de signe	**8 bits d'exposant**	**23 bits de partie significative**

Exemple

Un nombre sur 32 *bits* peut représenter 4 294 967 296 informations différentes. Par exemple, les nombres entiers de 0 à 4 294 967 295. Pour la représentation d'un nombre réel, il faut utiliser un *bit* pour le signe. Il en reste, donc 31. 8 *bits* sont utilisés comme exposant. Ils permettent de donner le nombre de chiffres après la virgule. Il reste donc 23 *bits* pour représenter la partie significative, soit 8 388 608 possibilités.

Imaginons que je veuille stocker le nombre 1.25, mais que je ne sache manipuler que les entiers. Je pourrais donc considérer que 1.25 = 125 x 10^{-2}. Je me retrouve avec une partie significative entière (1 25) et un exposant entier (-2). C'est exactement le même principe qui est utilisé par l'ordinateur, mais il le fait en base 2. Cela serait donc une mauvaise idée d'utiliser des nombres réels là où des entiers conviendraient, car nous perdrions en précision. En effet, les *bits* consacrés à l'exposant réduisent d'autant les *bits* restant pour la partie significative.

Le fait d'utiliser des entiers ou des réels peut influencer les résultats d'opération.

```
Variable :
 a : entier
 b : réel


Début
 a ← 3 / 2
 b ← 3 / 2
Fin
```

« *a* » étant défini comme un nombre entier, lui affecter le résultat de 3/2 va tronquer le résultat. Seule la partie entière sera donc affectée à « *a* ». « *a* » vaut, à la fin de l'algorithme, la valeur entière 1. « *b* » ayant été défini comme réel va pouvoir contenir des chiffres après la virgule grâce à la conversion vue plus haut. « *b* » contiendra donc à la fin de l'algorithme la valeur réelle 1.5.

C. Les booléens

Les variables booléennes tirent leur nom du mathématicien et logicien George Boole (1815 - 1864). Ce dernier a mis au point une algèbre bâtie sur la logique, pour transformer en condition des phrases du quotidien. Ces conditions doivent pouvoir être interprétées sans ambiguïté par la machine, et aboutir, soit à un résultat VRAI, soit à un résultat FAUX.

Encore une fois, la machine nous dévoile son fonctionnement binaire. Il n'y a pas de demi-mesure ; soit une condition est VRAIE, soit elle est FAUSSE. Les variables booléennes sont capables de stocker les résultats des conditions, et ne peuvent donc avoir que deux valeurs : VRAI et FAUX. Le type booléen est souvent considéré dans les langages de programmation comme une extension du type entier. Bien souvent, les langages de programmation considèrent la valeur 0 comme étant FAUSSE, et la valeur 1 comme étant VRAIE. Il y a des disparités au niveau des langages à ce niveau.

La possibilité de conditionner l'exécution des programmes permet de les rendre plus flexibles. En effet, ils ne se contentent plus d'exécuter des instructions qui se suivent, mais peuvent aussi répondre à des choix. Cette partie sera plus détaillée dans la partie traitant de la structure conditionnelle (« *si...* », « *alors...* », « *sinon...* »), et dans la partie mathématique traitant de la logique.

Gardons pour le moment en tête qu'une variable booléenne ne peut prendre que deux valeurs (VRAI ou FAUX), et qu'elle peut stocker le résultat d'un test.

```
Variable :
 a : booléen
 b : booléen


Début
 a ← 5 < 9
```

```
 b ← 5 > 9
Fin
```

Dans cet algorithme, nous avons défini deux variables booléennes « *a* » et « *b* ». Nous avons affecté à « *a* » le résultat du test 5 < 9. Nous savons que 5 est inférieur à 9. L'ordinateur affecte donc la valeur VRAI à « *a* ». Cependant, nous savons également que 5 n'est pas supérieur à 9. L'ordinateur affecte donc la valeur FAUX à « *b* ». Ces variables booléennes pourront être évaluées dans les structures conditionnelles et dans les boucles. Nous en reparlerons dans ces parties.

D. Les opérations sur les nombres

À partir du moment où nous avons donné un type à nos variables, nous savons quelles opérations nous pouvons réaliser avec elles. En effet, en tant qu'être humain, nous savons que certaines opérations n'ont pas de sens. Par exemple, multiplier un code postal et un numéro de sécurité sociale. Le type de variable va permettre de comprendre le comportement de certains opérateurs.

Si nous utilisons des variables de type numérique (entier ou réel), les opérateurs de mathématiques classiques sont disponibles.

```
Variable :
 a, b : entier
 som, sous, mult, div, div_ent, mod : entier

Début
 som ← a + b
 sous ← a - b
 mult ← a * b
 div ← a / b
 div_ent ← a // b
 mod ← a % b
Fin
```

Voici les opérations de base sur les variables numériques :

- Addition : +
- Soustraction : -
- Multiplication : *
- Division : /
- Puissance : **
- Division entière : // (quotient de la division euclidienne)
- Modulo : % (reste de la division euclidienne)

Les opérateurs de division seront vus plus en détail dans le chapitre des mathématiques consacré à l'arithmétique.

Attention

Si vous tentez d'effectuer une opération avec des variables qui ont des types non prévus, le langage vous signalera immédiatement une erreur. Cela peut être une bonne chose pour comprendre ce qui ne va pas. Les types vont donc permettre de déterminer quelles sont les opérations possibles, et lesquelles ne le sont pas.

Enfin, si nous utilisons des variables booléennes, il est possible d'utiliser des opérateurs booléens :

Et : et logique, renvoie VRAI si la valeur à gauche de l'opérateur et la valeur à droite de l'opérateur sont toutes les deux à VRAI et renvoie FAUX sinon.

A	B	A et B
FAUX	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

Ou : ou logique, renvoie VRAI si la valeur à gauche de l'opérateur ou la valeur à droite de l'opérateur est à VRAI et renvoie FAUX sinon.

A	B	A ou B
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	VRAI

Non : inverse la valeur d'une variable booléenne (VRAI devient FAUX, FAUX devient VRAI).

A	non A
FAUX	VRAI
VRAI	FAUX

Remarque

Les tableaux que nous avons utilisés ici sont des tables de vérité. Ils permettent de comprendre facilement comment fonctionne un opérateur. A et B sont des variables booléennes qui ne peuvent prendre que deux valeurs qui sont : VRAI et FAUX. À partir de là, nous essayons de retrouver toutes les conditions possibles de valeur. Pour chacune des conditions, nous allons évaluer l'expression dans la colonne de droite.

Les tables de vérité permettent d'aller beaucoup plus loin, et de représenter des expressions plus complexes, composées de plusieurs opérateurs. Elles permettent également de vérifier si deux expressions sont équivalentes.

Exercice : Quiz

[solution n°2 p.24]

Question 1

Quel type numérique n'est pas un type de base en programmation ?

- Les nombres complexes
- Les nombres entiers
- Les nombres réels

Question 2

J'ai le droit d'additionner un réel et un entier ?

- Non
- Oui

Question 3

Quelle est la valeur de « *c* » à la fin de cet algorithme ?

```
Variable :
 a, b, c : entier

Début
 a ← 3
 b ← 6
 c ← a * b
Fin
```

- 18
- 9
- 3

Question 4

Quelle est la valeur de « c » à la fin de cet algorithme ?

```
Variable :
 a, b, c : réel

Début
 a ← 2
 b ← 5
 c ← b / a
Fin
```

- 2
- 2.5
- 7

Question 5

Quelles sont les valeurs possibles pour une variable booléenne ?

- ○ VRAI et FAUX
- ○ OUI et NON
- ○ BLANC et NOIR

V. Les chaînes de caractères

A. Un caractère

Rappel

La machine ne sait manipuler que des valeurs entières. Nous avons vu que par quelques conversions, nous avons pu lui faire traiter des nombres décimaux et booléens. Les caractères constituent une nouvelle difficulté à relever.

Les caractères sont tout simplement convertis en entier à l'aide d'une table de conversion que nous appelons la table ASCII (*American Standard Code for Information Interchange*) :

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

(Source : Wikipédia[1])

Exemple

Par exemple, si je veux stocker en mémoire le caractère « *A* », je vais le convertir grâce à la table ASCII en entier à savoir 65. Ensuite, la machine va stocker 65, ce qu'elle va faire très facilement puisqu'elle manipule maintenant un entier et non plus un caractère.

1 https://fr.wikipedia.org/wiki/Fichier:ASCII-Table-wide.svg)

C'est le type qui permet une fois de plus de donner du sens aux données. En effet, sans le type, il serait impossible de savoir si la mémoire contient le nombre entier 65 ou le caractère « *A* » qui a été converti en 65 et stocké. La représentation en mémoire va être identique dans les deux cas. Cependant, le type va permettre au langage de programmation de savoir de quoi on parle.

La plupart des langages intègrent même des fonctions qui peuvent faire la conversion entre un caractère, et sa représentation numérique. Par exemple, en *Python* :

- Chr(65) retourne « *A* », le caractère correspondant au nombre 65 ;
- Ord('A') retourne 65, la valeur numérique associée au caractère « *A* ».

Attention

Il ne faut pas confondre la valeur numérique 3 avec le caractère « *3* ». Ces deux valeurs ne sont pas égales. Les langages de programmation utilisent souvent les quotes (‘), ou les doubles quotes (‘’) pour distinguer les caractères des nombres.

B. La chaîne de caractères

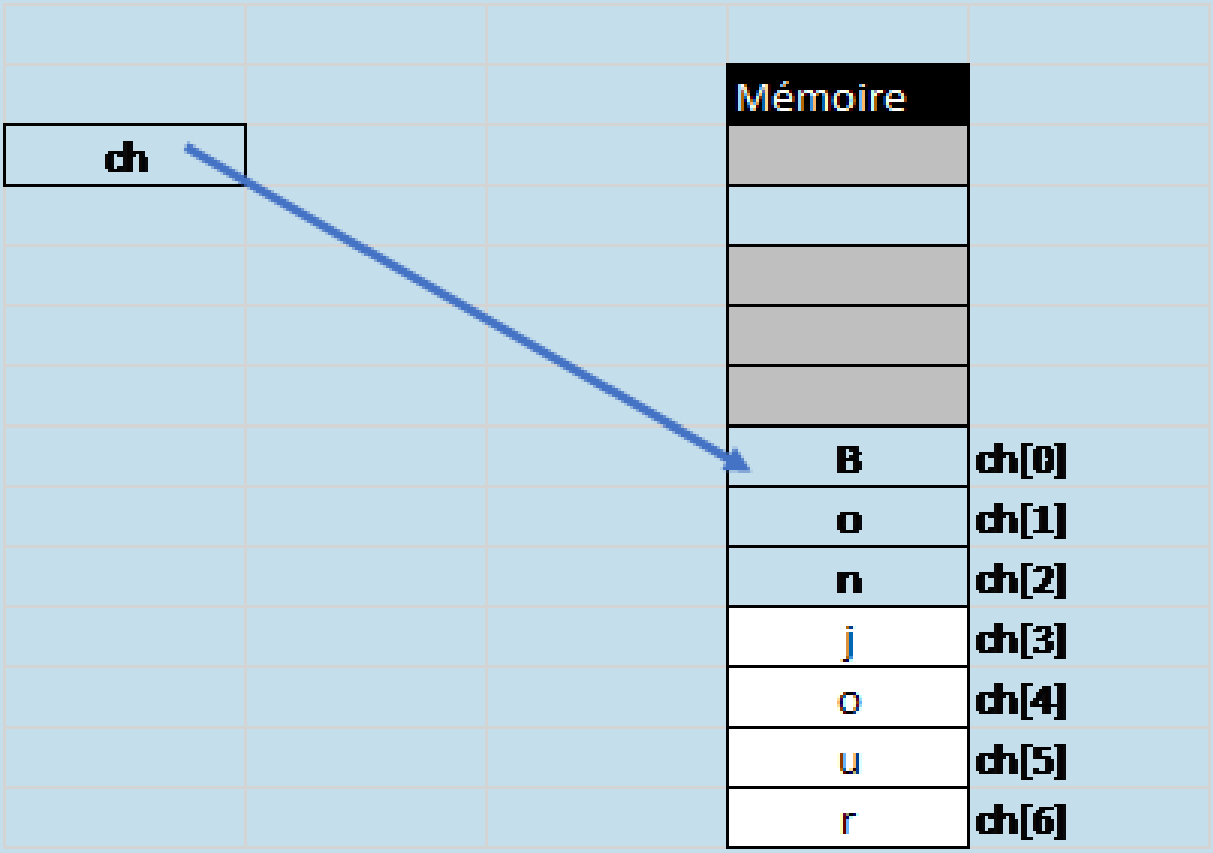
La chaîne de caractères est composée de plusieurs caractères les uns à la suite des autres. Une chaîne de caractères nous permet donc de représenter des mots ou des phrases.

Certains langages, comme le C / C++, font la différence entre un caractère (type char), et une chaîne de caractères (type String). La plupart du temps, vous pouvez vous passer de cette distinction, en partant du principe qu'un caractère peut être considéré comme une chaîne de caractères ne comportant qu'un seul élément.

Remarque

En termes de représentation mémoire, une chaîne de caractères est stockée dans des emplacements contigus de mémoire. Lorsque l'on accède à une chaîne de caractères, on accède en fait au début de la zone mémoire qui contient la chaîne. Il est ensuite possible d'accéder à un élément particulier de la chaîne en utilisant l'opérateur [].

Nous affectons ici la chaîne de caractères 'Bonjour' à la variable « *ch* » :



- Ch[0] accède au premier élément de la chaîne : 'B'
- Ch[3] accède au troisième élément de la chaîne : 'j'

Attention

En algorithmie, comme en programmation, le premier caractère de la chaîne se trouve à l'indice 0, et non à l'indice 1. Cela cause beaucoup de problèmes aux développeurs débutants. Vous pouvez retenir que Ch[0] accède à la tête de la chaîne, et fait 0 décalage. Ch[3] accède à la tête de la chaîne, et procède 3 fois à la recherche de l'élément suivant.

C. Les opérations sur les chaînes

Remarque

Une fois de plus, l'ordinateur va se baser sur le type de la variable pour savoir ce qu'il peut en faire. Le type chaîne de caractères va donc nous permettre d'utiliser des opérations comme :

- La concaténation : a + b (a : chaîne et b : chaîne)
- La répétition : a * b (a : chaîne et b : entier)
- Accès à l'élément d'indice i : ch[i]
- Comparaisons : a > b, a = b ,etc. (a : chaîne et b : chaîne)
- La récupération d'une sous-chaîne : ch[i :j] ou substring(ch, i, j)
- La longueur d'une chaîne : longueur(ch) ou len(ch) ou length(ch) ou ch.length selon le langage utilisé

La concaténation met une fois de plus en avant l'importance des types. Nous utilisons la même opération « + » pour faire la somme de deux nombres et pour concaténer deux chaînes. La différence entre les deux opérations se fait uniquement sur le type de variables.

Exemple

Nous allons utiliser quelques fonctions vues ci-dessus. Nous allons réaliser un algorithme qui manipule deux chaînes de caractères, qui les concatène, et qui donne la première lettre de la chaîne concaténée.

```
Variable :
 c1 : chaîne de caractères
 c2 : chaîne de caractères
 c3 : chaîne de caractères

Début
 c1 ← ''Bonjour''
 c2 ← ''tout le monde''
 c3 ← c1 + 2
 Afficher ''La première lettre de '' + c3 + '' est '' + c3[0]
Fin
```

Cet algorithme affiche : la première lettre de Bonjour tout le monde est B

Remarque

Ce premier algorithme basé sur les chaînes de caractères nous montre comment concaténer deux chaînes de caractères avec l'opérateur « + », et comment accéder à un élément bien précis avec l'opérateur []. Ces deux opérateurs fonctionnent comme attendu ici, car nous avons pris le soin de déclarer nos variables c1, c2 et c3 en tant que chaînes de caractères. Cela permet à la machine de savoir exactement comment elle doit réagir.

Exercice : Quiz

[solution n°3 p.25]

Question 1

À quoi sert la table ASCII ?

- O Elle sert à faire correspondre une valeur numérique à chaque caractère.
- O Elle sert à convertir des nombres binaires en hexadécimal.
- O Elle sert à traduire un texte français en anglais.

Question 2

Combien d'emplacements mémoire va utiliser la chaîne de caractère 'Bonjour' ?

- O 1
- O 7
- O 10

Question 3

Soit « *c* » une variable de type chaîne de caractères qui contient 'Salut'. Que vaut c [2] ?

- a
- l
- u

Question 4

Quelle est la valeur de « *c* » à la fin de cet algorithme ?

```
Variable
  a, b, c : chaîne de caractères

Début
  a ← '2'
  b ← '5'
  c ← a + b
Fin
```

- '7'
- '25'
- '52'

Question 5

Soit « *c* » une variable de type de chaîne de caractères qui contient 'Salut'. Que vaut longueur(c) ?

- 0
- 5
- 'Salut'

VII. Essentiel

La notion de type est très importante en algorithmie comme en programmation. C'est le type qui donne un sens aux données manipulées, et qui va permettre de comprendre les opérations que l'on va lancer sur nos variables. Que le langage possède un typage explicite ou implicite ne change rien, le type de variable est une notion présente systématiquement.

Le type détermine comment l'ordinateur va stocker nos données en mémoire, et comment il va pouvoir s'adapter pour manipuler exclusivement des valeurs entières. Nous avons identifié des types simples :

- Entier
- Réel
- Booléen
- Caractère
- Chaîne de caractères

Bien vite, les tableaux et les objets viendront s'ajouter à cette liste.

Les types permettent de comprendre ce que fait un algorithme, et comment les différentes opérations sont exécutées. Sans les types, la machine serait incapable de faire la différence entre un caractère et un entier.

VIII. Auto-évaluation

A. Exercice

Nous allons, pour la première fois, dialoguer avec la machine. Pour que ce dialogue ait lieu, il faut vérifier :

- Que la machine puisse nous informer ;
- Que nous puissions envoyer des informations à la machine.

Ce premier cas va nous permettre d'établir le contact. En termes d'algorithmie, nous allons utiliser deux instructions :

- Afficher msg : va nous permettre d'afficher un message (ici le contenu de la variable msg) sur l'écran et donc informer l'utilisateur ;
- Saisir c : va nous permettre de récupérer ce que l'utilisateur a saisi et le stocker dans une variable (ici la variable c).

Il s'agit d'une opération primordiale pour voir si les échanges sont possibles. Si la communication entre l'utilisateur et la machine ne fonctionne pas, nous ne pourrons pas aller plus loin.

Question 1 [solution n°4 p.27]

Comment faire en sorte que la machine me salue ?

Dans cette première question, nous allons faire en sorte que la machine vous dise bonjour. Pour cela, nous allons réaliser les étapes suivantes :

- Déclarer une variable de type chaîne de caractères qui va contenir votre prénom ;
- L'algorithme affiche une question du type : « *Quel est votre prénom* ? » ;
- Suite à cette question, l'algorithme lit la réponse de l'utilisateur ;
- Enfin, l'algorithme affiche : « *Bonjour* » votre prénom.

Question 2 [solution n°5 p.27]

Comment faire en sorte que la machine me demande mon âge ?

L'algorithme de la question précédente évolue et devient plus indiscret. Nous voulons maintenant qu'en plus de son prénom, l'utilisateur puisse saisir son âge. L'algorithme devra afficher : « *Bonjour* »votre prénom « *vous avez* » votre âge « *ans* ».

Prenez exemple sur la question 1 en réfléchissant au type le plus adapté pour stocker votre âge.

Question 3 [solution n°6 p.27]

Comment faire en sorte que la machine calcule mon année de naissance ?

L'évolution de notre algorithme ci-dessus va faire calculer la machine. Maintenant que l'algorithme connaît votre âge, il peut calculer simplement votre année de naissance (année courante - âge). Je vous propose, dans cette troisième partie, de déclarer une variable qui va contenir l'année courante, et une variable qui va contenir votre année de naissance. Là aussi, voyez quel est le type le plus adapté à ces données.

Demandez ensuite à la machine d'effectuer le calcule et d'afficher : « *Vous êtes nés en* » année de naissance.

B. Test

Exercice 1

[solution n°7 p.28]

Redonnez le bon type à chacune des valeurs ci-dessous :

"9"
VRAI
2.5
2
Entier
Réel
Booléen
Chaîne de caractères

Exercice 2 : Quiz

[solution n°8 p.28]

Question 1

Comment va réagir l'ordinateur quand on lui demande de manipuler un caractère ?

- ○ Aucun problème, un ordinateur est construit de manière à comprendre les caractères de base
- ○ Il va convertir ce caractère en nombre entier pour pouvoir le manipuler
- ○ Il va renvoyer un message d'erreur à l'utilisateur pour lui signaler qu'il ne comprend pas ce type.

Question 2

Avec 1 *bit*, je peux représenter 2 informations différentes ; avec 2 *bits*, je peux en représenter 4 ; avec 3 *bits*, je peux en représenter 8. Combien d'informations différentes puis-je représenter avec 16 *bits* ?

- ○ 16
- ○ 2 56
- ○ 65 536

Exercice 5

[solution n°9 p.29]

Dans quel cas de figure l'opérateur '+' produit une addition, une concaténation ou une erreur :

Chaîne + entier | Entier + réel | Réel + chaîne | Réel + réel | Chaîne + Chaîne | Entier + entier

Addition	Concaténation	Erreur

Exercice 6

[solution n°10 p.29]

Faites en sorte que la phrase ci-dessous soit vraie :

Dans les langages où la déclaration des variables est *[Facultative | Implicite | Explicite]*, il n'y a pas lieu de déclarer le type de la variable. Le langage va déterminer ce type lors de la première utilisation de la variable. C'est le cas pour le langage *Python*.

Solutions des exercices

Exercice p. 8 **Solution n°1**

Question 1

Une variable permet de :

- [x] Stocker une valeur et récupérer la valeur stockée
- [ ] Modifier la vitesse du processeur
- [ ] Ajouter de la mémoire à un ordinateur

La variable permet de stocker les données du programme que nous voulons conserver. Le fait de leur donner un nom nous permet de récupérer ensuite la valeur stockée.

Question 2

Quel nom puis-je donner à une variable ?

- [ ] 99
- [x] puissanceMoteur
- [ ] while

Les noms de variables doivent obligatoirement commencer par une lettre (99 ne convient pas), et ne pas être un mot du langage (*while* est une instruction de boucle).

Question 3

Quel est le seul type de données que peut manipuler un ordinateur ?

- [ ] Les nombres réels
- [ ] Les caractères
- [x] Les nombres entiers

La représentation binaire de la machine provient du fait que les échanges d'informations se font par courant électrique. Cette représentation binaire peut être assimilée facilement à des nombres entiers.

Question 4

Le typage implicite de certains langages de programmation :

- [x] Détermine le type d'une variable en fonction de la valeur qui lui est affectée
- [ ] N'utilise pas la notion de type
- [ ] Utilise un type par défaut commun à toutes les variables du langage

Que le typage soit implicite ou explicite, il est toujours présent. Le typage implicite déduit le type à partir de l'affectation d'une valeur à une variable.

Question 5

Quelles sont les bases issues de puissances de deux ?

- ○ Décimal, binaire, hexadécimal
- ◉ Binaire, octal, hexadécimal
- ○ Binaire, décimal, octal

Q Les bases issues de puissance de 2 sont le binaire (2 = 2^1), l'octal (8 = 2^3) et l'hexadécimal (16 = 2^4).

Exercice p. 14 **Solution n°2**

Question 1

Quel type numérique n'est pas un type de base en programmation ?

- ◉ Les nombres complexes
- ○ Les nombres entiers
- ○ Les nombres réels

Q Les nombres complexes ne sont pas considérés comme des types de bases, bien que des bibliothèques mathématiques vous permettront de les utiliser si vous en avez besoin. Les nombres entiers sont facilement compréhensibles par l'ordinateur. Les nombres réels subissent une conversion vers une représentation entière.

Question 2

J'ai le droit d'additionner un réel et un entier ?

- ○ Non
- ◉ Oui

Q Les nombres réels et entiers sont parfaitement compatibles du point de vue de l'addition. En *Python*, si vous sauvegardez l'addition entre un entier et un réel dans une variable, le résultat sera automatiquement du type réel pour ne pas perdre d'information.

Question 3

Quelle est la valeur de « *c* » à la fin de cet algorithme ?

```
Variable :
 a, b, c : entier

Début
 a ← 3
 b ← 6
 c ← a * b
Fin
```

- ◉ 18
- ○ 9
- ○ 3

L'opérateur « * » est l'opérateur de multiplication en présence de deux entiers. La variable « c » prend donc la valeur de 3 * 6, à savoir 18.

Question 4

Quelle est la valeur de « c » à la fin de cet algorithme ?

```
Variable :
 a, b, c : réel


Début
 a ← 2
 b ← 5
 c ← b / a
Fin
```

- ○ 2
- ◉ 2.5
- ○ 7

L'opérateur « / » est l'opérateur de division en présence de deux réels. La variable « c », étant définie en tant que réel, est capable de stocker la partie décimale du résultat de la division 5 / 2, à savoir 2.5.

Question 5

Quelles sont les valeurs possibles pour une variable booléenne ?

- ◉ VRAI et FAUX
- ○ OUI et NON
- ○ BLANC et NOIR

Une variable booléenne contient habituellement le résultat de l'évaluation d'une condition. Une condition, une fois évaluée, ne peut prendre que deux valeurs : VRAI ou FAUX.

Exercice p. 19 Solution n°3

Question 1

À quoi sert la table ASCII ?

- ◉ Elle sert à faire correspondre une valeur numérique à chaque caractère.
- ○ Elle sert à convertir des nombres binaires en hexadécimal.
- ○ Elle sert à traduire un texte français en anglais.

La table ASCII est une correspondance entre chaque caractère disponible. C'est une représentation sous forme de nombre entier que l'ordinateur sera capable de comprendre. Elle servira également à retrouver un caractère à partir d'un entier stocké en mémoire.

Question 2

Combien d'emplacements mémoire va utiliser la chaîne de caractère 'Bonjour' ?

- 1
- 7
- 10

La chaîne de caractères va être sauvegardée en mémoire à raison d'une lettre par emplacement. 'Bonjour' contient 7 lettres, elle va donc utiliser 7 emplacements mémoire.

Question 3

Soit « *c* » une variable de type chaîne de caractères qui contient 'Salut'. Que vaut c [2] ?

- a
- l
- u

L'opérateur [] permet d'accéder à un élément précis de la chaîne de caractères. Chaque élément se voit attribuer un indice en commençant par 0 : c[0] = 'S', c[1] = 'a' et c[2] = 'l'.

Question 4

Quelle est la valeur de « *c* » à la fin de cet algorithme ?

```
Variable
  a, b, c : chaîne de caractères

Début
  a ← '2'
  b ← '5'
  c ← a + b
Fin
```

- '7'
- '25'
- '52'

L'opérateur « + » entre deux chaînes de caractères est l'opérateur de concaténation. Les deux chaînes de caractères « *2* » et « *5* » sont donc concaténées pour donner « *25* ». L'opérateur « + » n'est l'opérateur d'addition que lorsqu'il est entouré de données de type numérique.

Question 5

Soit « *c* » une variable de type de chaîne de caractères qui contient 'Salut'. Que vaut longueur(c) ?

- 0
- 5
- 'Salut'

La fonction longueur renvoie la longueur d'une chaîne de caractères, autrement dit, le nombre de lettres qui la composent. Ici, 'Salut' est un mot de 5 lettres.

p. 21 **Solution n°4**

```
Variables :
 prenom : chaîne de caractères

Début
 Afficher 'Quel est votre prénom ?'
 Saisir prenom
 Afficher 'Bonjour ' + prenom
Fin
```

p. 21 **Solution n°5**

```
Variables :
 prenom : chaîne de caractères
 age : entier

Début
 Afficher 'Quel est votre prénom ?'
 Saisir prenom
 Afficher 'Quel est votre âge ?'
 Saisir age
 Afficher 'Bonjour ' + prenom + ' vous avez ' + age + ' ans'
Fin
```

p. 21 **Solution n°6**

```
Variables :
 prenom : chaîne de caractères
 age : entier
 anneeCourante : entier
 anneeNaissance : entier

Début
 anneeCourante ← 2022
 Afficher 'Quel est votre prénom ?'
 Saisir prenom
 Afficher 'Quel est votre âge ?'
 Saisir age
 anneeNaissance ← anneeCourante - age
 Afficher 'Vous êtes nés en ' + anneeNaissance
```

```
Fin
```

Exercice p. 22 Solution n°7

Redonnez le bon type à chacune des valeurs ci-dessous :

Entier
2
Réel
2.5
Booléen
VRAI
Chaîne de caractères
"9"

Type	Valeur
Entier	2
Réel	2.5
Booléen	VRAI
Chaîne de caractères	"9"

Les chaînes de caractères sont toujours entourées de quotes (‘’). C’est une bonne façon de les reconnaître. Les booléens ne peuvent avoir que deux valeurs qui sont : VRAI et FAUX. Enfin, on différencie un nombre entier d’un nombre réel grâce à la partie décimale qui se trouve après la virgule.

Exercice p. 22 Solution n°8

Question 1

Comment va réagir l’ordinateur quand on lui demande de manipuler un caractère ?

- ○ Aucun problème, un ordinateur est construit de manière à comprendre les caractères de base
- ◉ Il va convertir ce caractère en nombre entier pour pouvoir le manipuler
- ○ Il va renvoyer un message d’erreur à l’utilisateur pour lui signaler qu’il ne comprend pas ce type.

L’ordinateur ne sait manipuler que des nombres entiers. Pour pouvoir manipuler des caractères, il va s’appuyer sur la table ASCII pour transformer chaque caractère en nombre entier et ainsi se retrouver avec un type qu’il sait manipuler

Question 2

Avec 1 *bit*, je peux représenter 2 informations différentes ; avec 2 *bits*, je peux en représenter 4 ; avec 3 *bits*, je peux en représenter 8. Combien d’informations différentes puis-je représenter avec 16 *bits* ?

- ○ 16
- ○ 2 56
- ◉ 65 536

Le nombre d’informations différentes que je peux représenter est systématiquement calculé par $2^{\text{nombres se bits}}$.je me base sur 16 bits,il me suffit de calculer 2^{16}, c’est-à-dire 65 536.

Exercice p. 22 Solution n°9

Dans quel cas de figure l'opérateur '+' produit une addition, une concaténation ou une erreur :

Addition	Concaténation	Erreur
Entier + entier	Chaîne + Chaîne	Réel + chaîne
Entier + réel		Chaîne + entier
Réel + réel		

Cas	Opération
Addition	**Entier + entier**
	Entier + réel
	Réel + réel
Concaténation	**Chaîne + chaîne**
Erreur	**Réel + chaîne**
	Chaîne + entier

L'opérateur « + » exécute une addition à partir du moment où les deux opérandes sont de type numérique (entier ou réel). La concaténation se fait uniquement si les deux opérandes sont des chaînes de caractères. Enfin, il y a une erreur quand l'opérateur « + » se trouve entre une valeur numérique et une valeur de type chaîne de caractères. Pour résoudre ce genre de problème, il faudra passer par une opération de transtypage.

Exercice p. 22 Solution n°10

Faites en sorte que la phrase ci-dessous soit vraie :

Dans les langages où la déclaration des variables est `Implicite`, il n'y a pas lieu de déclarer le type de la variable. Le langage va déterminer ce type lors de la première utilisation de la variable. C'est le cas pour le langage *Python*.

Dans les langages où la déclaration des variables est implicite, il n'y a pas lieu de déclarer le type de la variable. Le langage va déterminer ce type lors de la première utilisation de la variable. C'est le cas pour le langage *Python*.