

Cours d'introduction à l'informatique

Partie 2 : Comment écrire un algorithme ?
Qu'est-ce qu'une variable ? Expressions et instructions...

Qu'est-ce qu'un algorithme ?

- Une recette de cuisine
- Protocole expérimental
- Des instructions pour aller quelque part
- Des consignes de sécurité
- **Dans ce cours, une méthode de résolution d'un problème, écrite de manière non ambiguë et susceptible d'être codée sur ordinateur**

Ambiguité

- Quelques exemples à éviter:
 - vends tricycle pour infirme en bon état
 - Deux conducteurs étaient interpellés par les gendarmes en état d'ivresse. (Var Matin, 13/07/1994)
 - J'ai lu la critique de Chomsky
- Le « français » ne convient pas, il faut avoir des règles de syntaxe et un vocabulaire précis (si possible pas trop grand pour ne pas s'encombrer l'esprit !!!).

Les deux bases des algorithmes

- Les **variables**, dans lesquelles on stocke l'information et qui décrivent la manière permettant d'accéder à la mémoire de l'ordinateur.
- Les **instructions** qui permettent de modifier l'état de ces variables

Les deux bases des algorithmes

- Un **algorithme** est donc composé d'une suite d'instructions qui, partant d'une description en mémoire d'un problème non résolu, donnent les modifications de la mémoire permettant d'arriver à une description en mémoire du problème résolu.
- ~~Les instructions qui permettent de modifier l'état de ces variables~~

Un exemple

Un exemple

Ex: calcul de x^{2k} ,

Le problème est bien **décrit** dès qu'on connaît les valeurs de x et de k (deux variables).

Le problème est **résolu** après l'exécution d'une suite d'instructions du type *puissance* \leftarrow *puissance*²

Le résultat est décrit par la valeur courante de la variable *puissance*.

*Les interactions avec l'utilisateur sont également décrites par l'algorithme grâce à des instructions d'entrée/sortie (ex: entrée de x et k par l'utilisateur et sortie de *puissance*).*

Ecrire un algorithme

Algorithme Puissance

// algorithme qui calcule une puissance d'un nombre

Variables

x,puissance : réels;

k,i : entier;

Début

x ← Saisie(); // L'utilisateur doit entrer un réel

k ← Saisie(); // L'utilisateur doit entrer un entier

puissance ← x; // initialisation de la variable puissance

Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...

 puissance ← puissance * puissance;

fin pour

Ecrire(puissance);

Fin

Ecrire un algorithme

Algorithme Puissance

// algorithme qui calcule une puissance d'un nombre

Variables

x,puissance : réels;
k,i : entier;

Début

x ← Saisie(); // L'utilisateur doit entrer un réel
k ← Saisie(); // L'utilisateur doit entrer un entier
puissance ← x; // initialisation de la variable puissance
Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...
 puissance ← puissance * puissance;

fin pour

Ecrire(puissance);

Fin

Entête

Déclarations

Corps=Description du calcul et des interactions

Ecrire un algorithme

Commentaires

Algorithme Puissance

// algorithme qui calcule une puissance d'un nombre

Variables

x,puissance : réels;

k,i : entier;

Début

x ← Saisie(); // L'utilisateur doit entrer un réel

k ← Saisie(); // L'utilisateur doit entrer un entier

puissance ← x; // initialisation de la variable puissance

Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...

 puissance ← puissance * puissance;

fin pour

Ecrire(puissance);

Fin



Ecrire un algorithme

Saisie par l'utilisateur

Commentaires

Algorithme Puissance

// algorithme qui calcule une puissance d'un nombre

Variables

x,puissance : réels;

k,i : entier;

Début

x ← Saisie(); // L'utilisateur doit entrer un réel

k ← Saisie(); // L'utilisateur doit entrer un entier

puissance ← x; // initialisation de la variable puissance

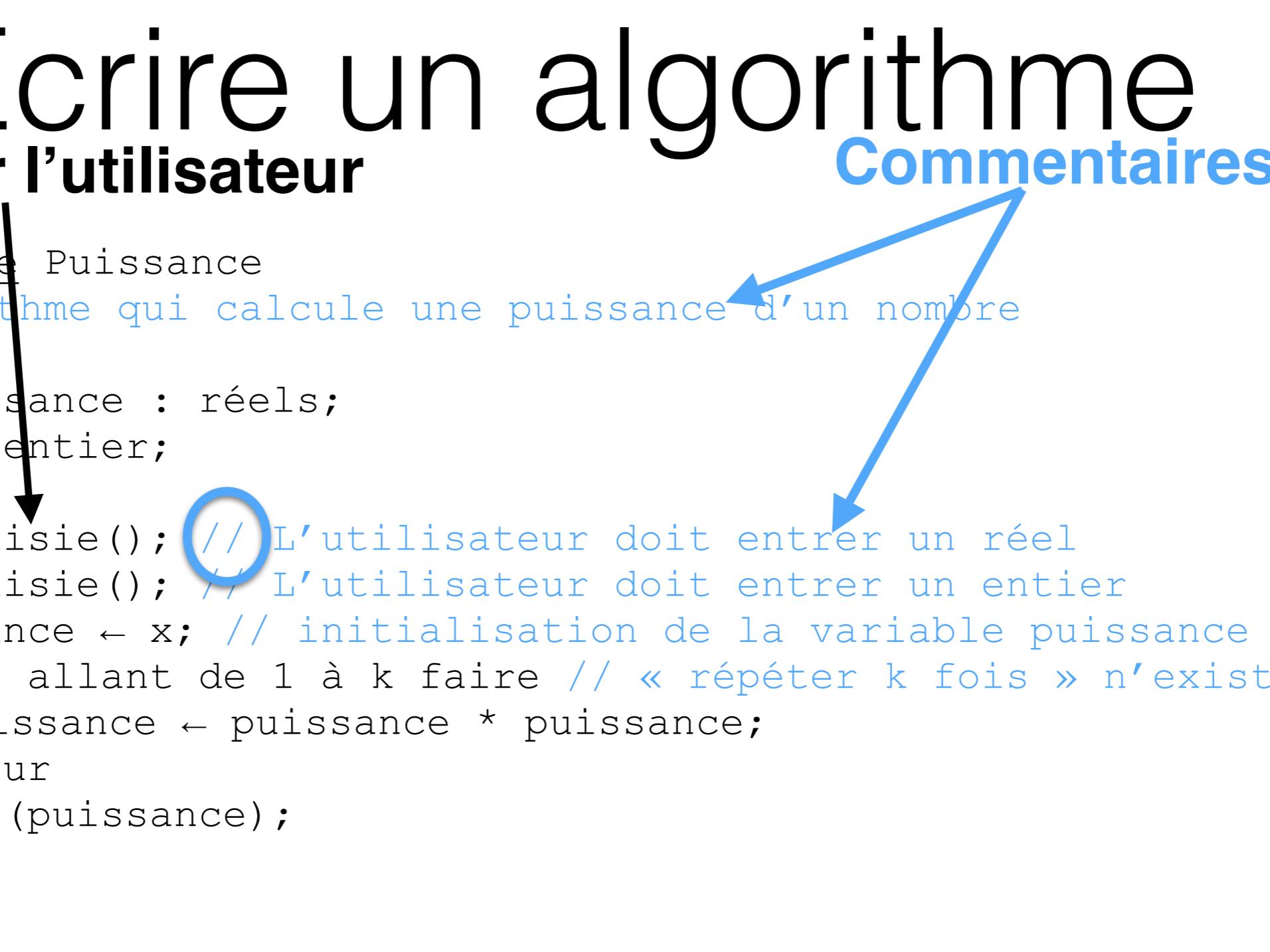
Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...

 puissance ← puissance * puissance;

fin pour

Ecrire(puissance);

Fin



Ecrire un algorithme

Saisie par l'utilisateur

Commentaires

```
Algorithme Puissance
// algorithme qui calcule une puissance d'un nombre
Variables
x,puissance : réels;
k,i : entier;
Début
    x ← Saisie(); // L'utilisateur doit entrer un réel
    k ← Saisie(); // L'utilisateur doit entrer un entier
    puissance ← x; // initialisation de la variable puissance
    Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...
        puissance ← puissance * puissance;
    fin pour
    Ecrire(puissance);
Fin
```

Affichage pour l'utilisateur

Ecrire un algorithme

Saisie par l'utilisateur

Commentaires

Algorithme Puissance

// algorithme qui calcule une puissance d'un nombre

Variables

x,puissance : réels;

k,i : entier;

Début

x ← Saisie(); // L'utilisateur doit entrer un réel

k ← Saisie(); // L'utilisateur doit entrer un entier

puissance ← x; // initialisation de la variable puissance

Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...

 puissance ← puissance * puissance;

fin pour

Ecrire(puissance);

Fin

**Modification de la valeur d'une variable
= affectation**

Affichage pour l'utilisateur



99% des algorithmes se décomposent en 3 parties

- 1) ce qu'on demande à l'utilisateur
- 2) ce qu'on calcule (la partie difficile en général)
- 3) ce qu'on restitue à l'utilisateur (qui ne peut pas démonter l'ordinateur pour voir l'état de la mémoire !)

Début

```
1 x ← Saisie(); // L'utilisateur doit entrer un réel
2 k ← Saisie(); // L'utilisateur doit entrer un entier
puissance ← x; // initialisation de la variable puissance
Pour i allant de 1 à k faire // « répéter k fois » n'existe pas...
    puissance ← puissance * puissance;
fin pour
3 Ecrire(puissance);
Fin
```

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme

Variables

Début

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

Début

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

Début

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

année ← Saisie();

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

année ← Saisie();

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

année ← Saisie();

age ← 2020 - année;

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

année ← Saisie();

age ← 2020 - année;

Ecrire(age);

Fin

Un exercice typique ?

Ecrire un algorithme qui demande à l'utilisateur de saisir une année de naissance et calcule l'âge qu'aura la personne au 31 décembre 2020 à minuit.

Algorithme Calcule âge

Variables

année : entier

age: entier

Début

année ← Saisie();

age ← 2020 - année;

Ecrire(age);

Fin

en Javascript:

// Algorithme Calcule age

var annnee,age;

annee = Saisie();

age = 2020 - annnee;

Ecrire(age);

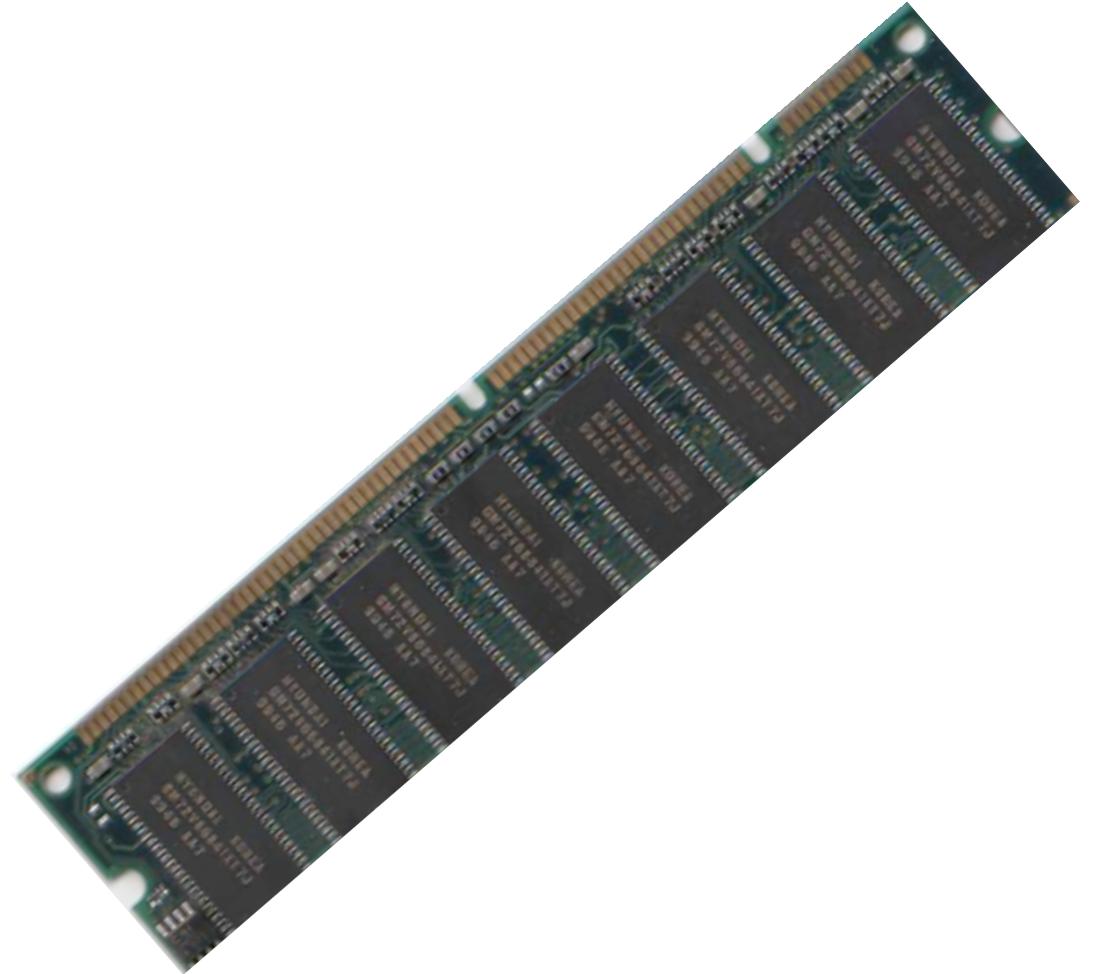
Les Variables

Nom, types, etc....

Accéder à la mémoire de l'ordinateur....

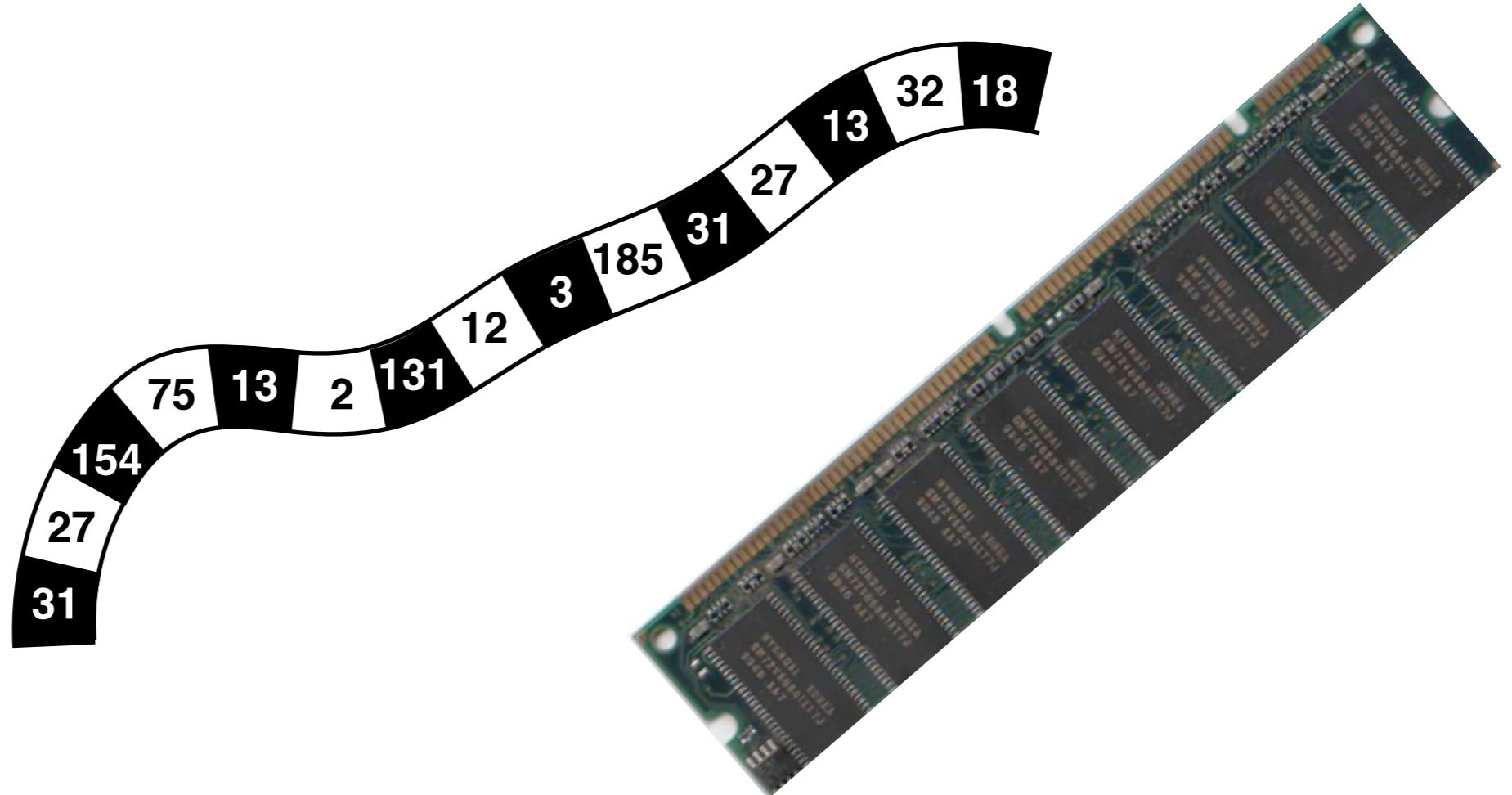
- Mémoire de l'ordinateur \approx très grand espace découpé en cases.
- Chaque case porte ainsi un numéro (**adresse**) pour s'y retrouver.
- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

Accéder à la mémoire de l'ordinateur



- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

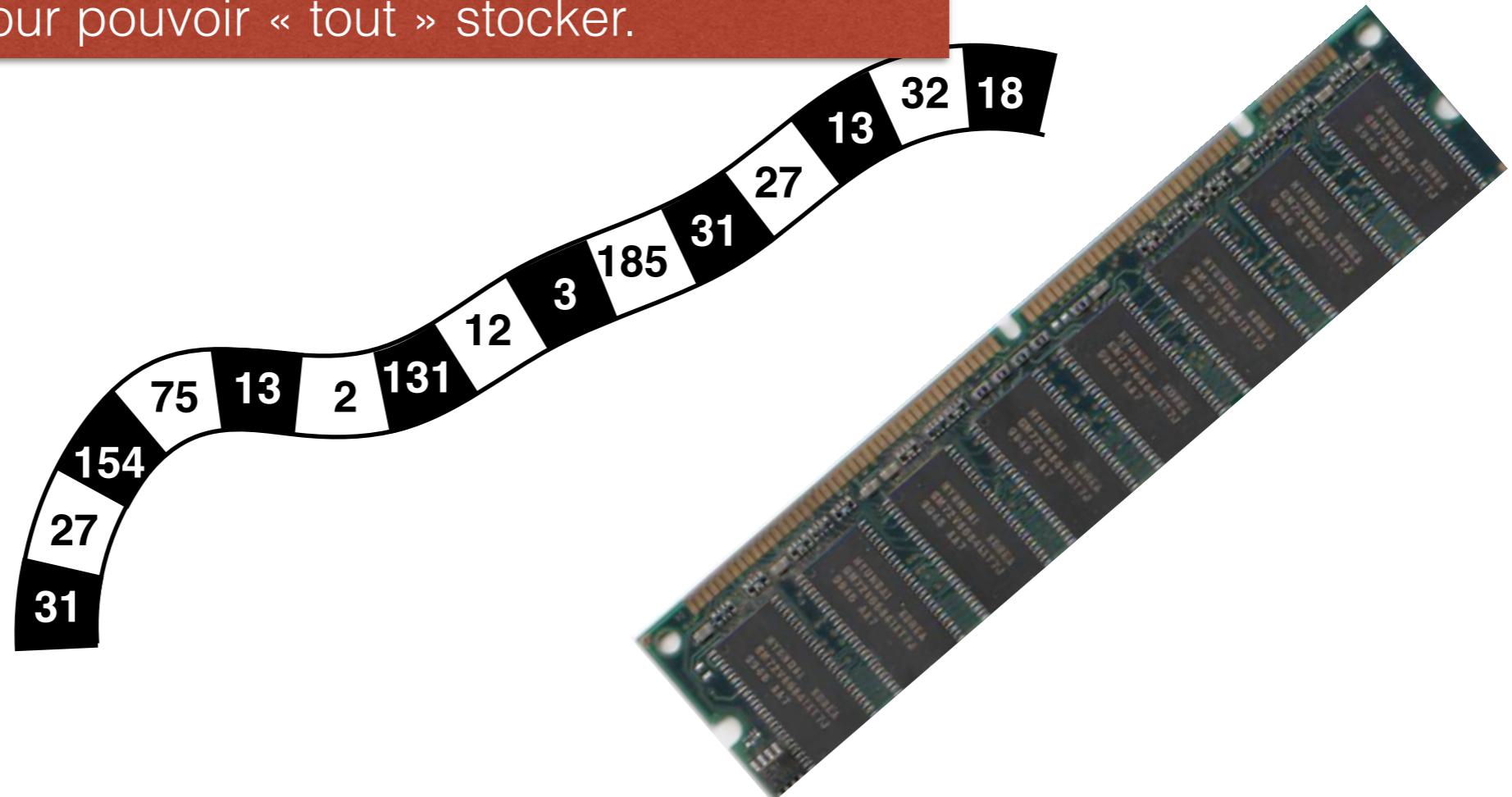
Accéder à la mémoire de l'ordinateur



- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

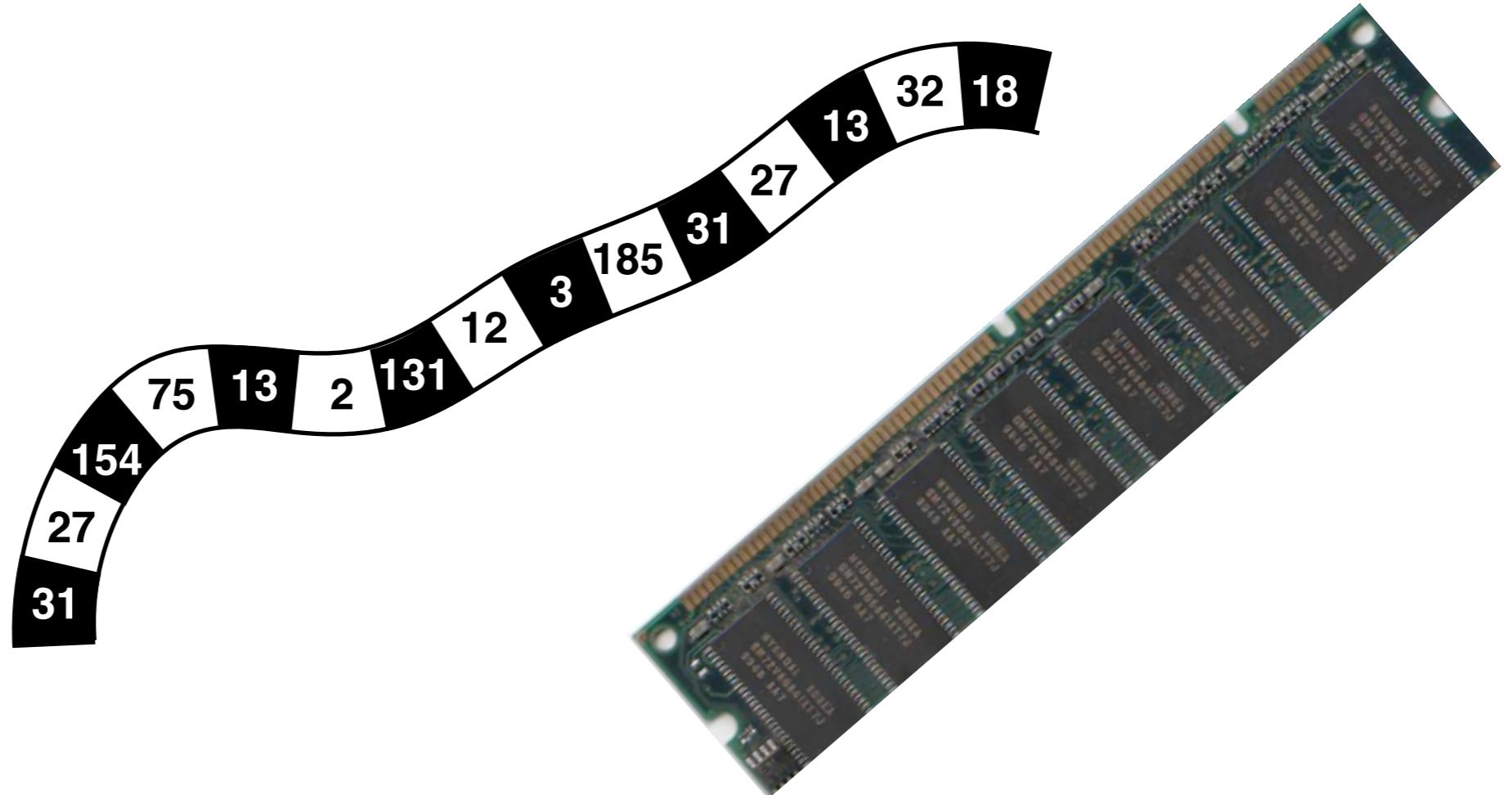
Accéder à la mémoire de

Petite remarque en passant, il suffit de savoir stocker des entiers (et même, uniquement les valeurs 0 et 1) pour pouvoir « tout » stocker.



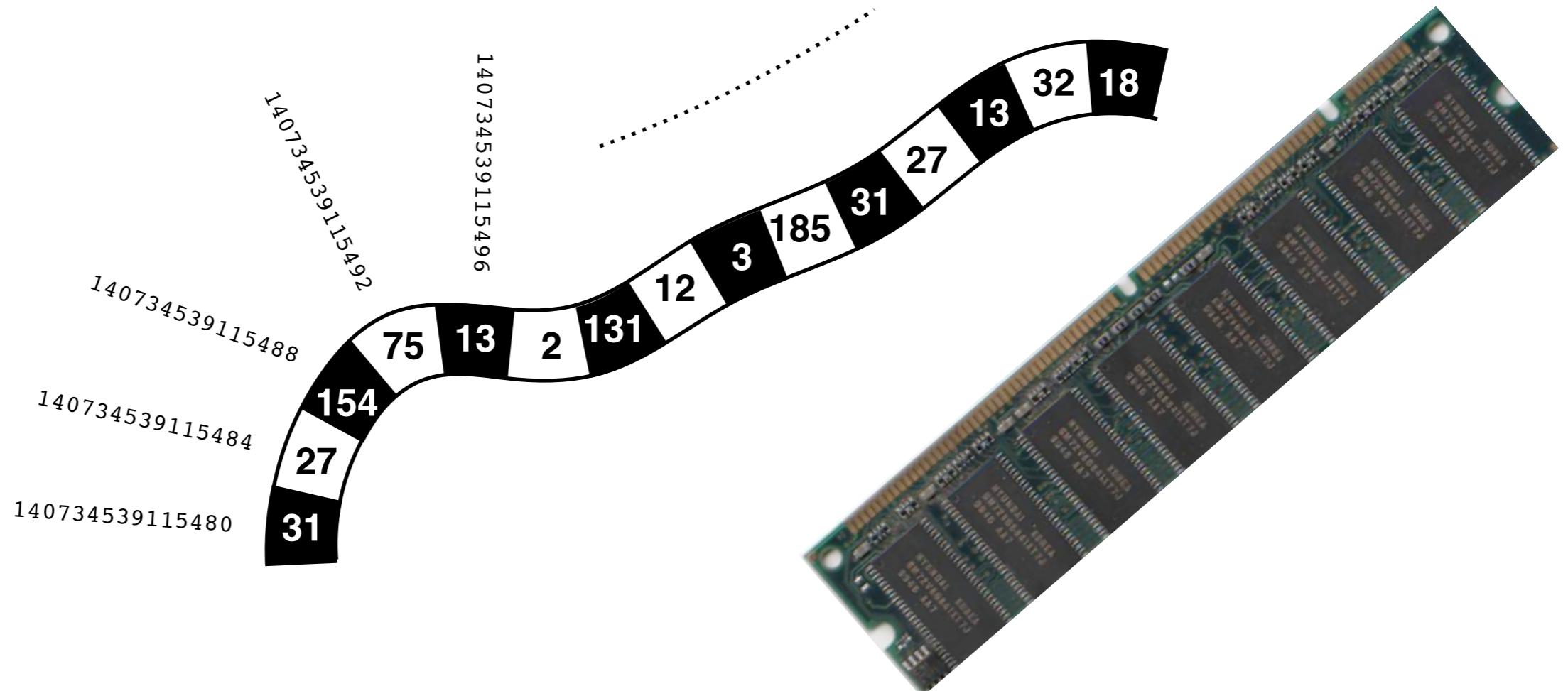
- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

Accéder à la mémoire de l'ordinateur



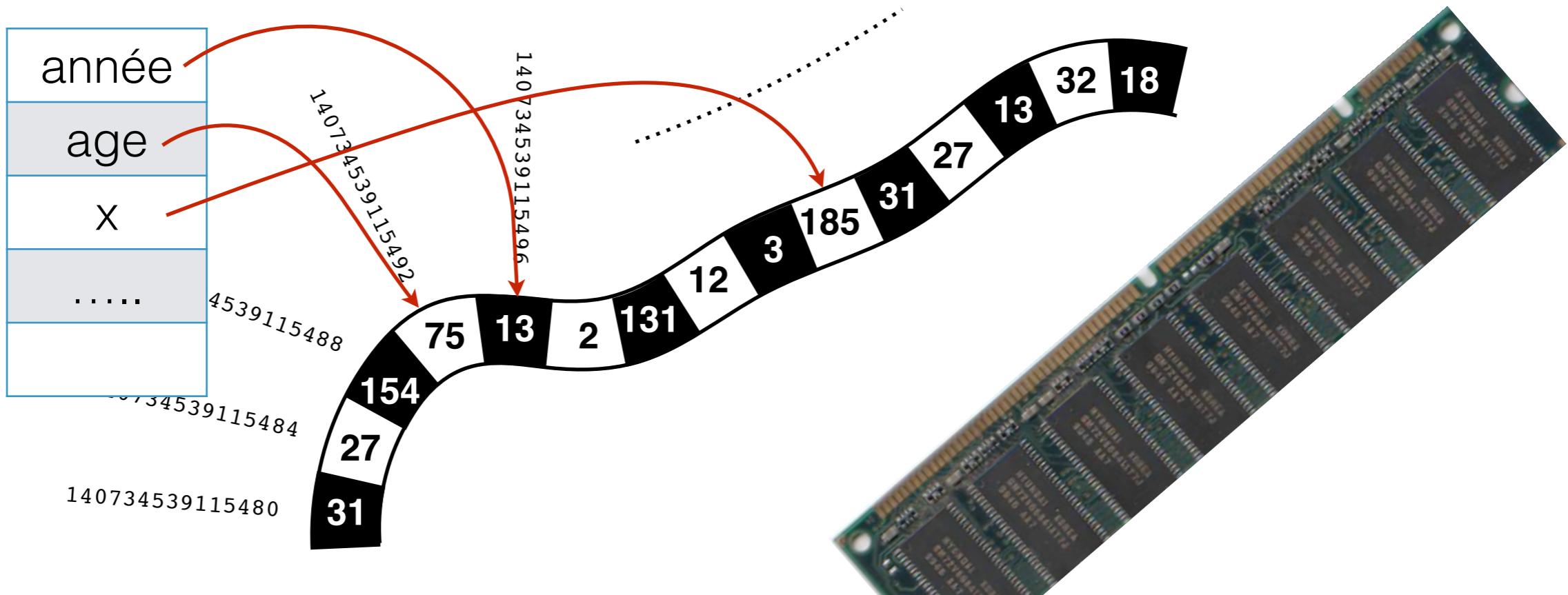
- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

Accéder à la mémoire de l'ordinateur



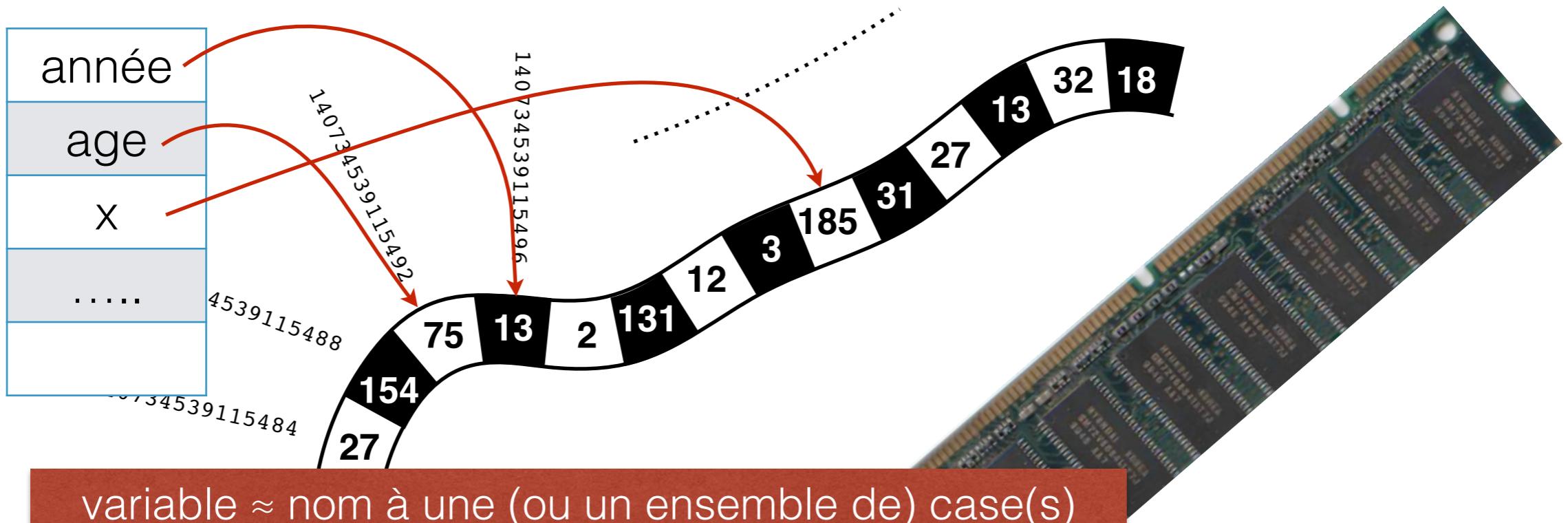
- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

Accéder à la mémoire de l'ordinateur



- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

Accéder à la mémoire de l'ordinateur



- On peut consulter/modifier le contenu d'une case en question dès qu'on en connaît l'adresse.

Les variables

Les variables sont caractérisées par:

- un **nom** (identifiant)
 - car nommer c'est abstraire
- un **type**
 - de ce qui est stocké dans la variable (i.e., combien de cases mémoires « physiques » sont nécessaires pour stocker l'information)
 - numérique (entier/réel), chaînes de caractères, booléens, autres choses plus compliquées (cf. plus tard)....
- une **valeur courante** (qui peut être modifiée).

Les variables

Les variables sont caractérisées par:

- un **nom** (identifiant)
 - NB: un jour il faudra se poser la question de comment gérer les cas où un programme est écrit à plusieurs
 - un qui peuvent utiliser les mêmes noms de variables....
Réponse dans 2 cours....
- de ce qui est stocké dans la variable (i.e., combien de cases mémoires « physiques » sont nécessaires pour stocker l'information)
- numérique (entier/réel), chaînes de caractères, booléens, autres choses plus compliquées (cf. plus tard)....
- une **valeur courante** (qui peut être modifiée).

Les types de base

1 - Les numériques « entiers »

- Les entiers en maths = éléments de \mathbb{Z}
- Les entiers en informatique = un sous ensemble des entiers en maths (ceux qu'on peut écrire selon une règle d'écriture donnée et qui dépend du langage informatique).
 - Les « int » en C (sur mon mac) sont les entiers entre $-2^{31} = -2147483648$ et $2^{31} = 2147483647$
 - Les entiers en javascript (partout) sont tous les entiers entre $-2^{53} = -9007199254740992$ et $2^{53} = 9007199254740991$ plus quelques autres qui ne se suivent pas forcément...

Les types de base

1

>>

La réalité: on utilise une décomposition binaire des entiers

$$135 = 10000111 = 1 \cdot 128 + 0 \cdot 64 + \dots + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$

- L
- L
- L

Si on utilise 8 « bits » on peut écrire tous les nombres entre

$$00000000 = 0 \text{ et } 11111111 = 255$$

ou encore tous les nombres entre -128 et 127 si on réserve un bit pour le signe....

- **Moralité:** on peut coder les entiers avec uniquement des bits (0 et 1).

- Les entiers en javascript (partout) sont tous les entiers entre $-9007199254740992 = -2^{53}$ et 9007199254740991 plus quelques autres qui ne se suivent pas forcément...

Les types de base

2 - Les numériques « réels »

- Les réels en maths = éléments de \mathbb{R}
- Les entiers en informatique = Les flottants un sous ensemble des réels en maths (ceux qu'on peut écrire selon une règle d'écriture donnée et qui dépend du langage informatique et souvent d'une norme).
 - Les « float » en C sont certains réels dont les positifs sont compris entre 3.4×10^{-38} et 3.4×10^{38}
 - Les « Numbers » en javascript sont...

Réels vs flottants: les nombres en (Algo/java)Script

- L'ordinateur ne sait manipuler nativement que des entiers et en nombre fini.
- L'astuce pour représenter un nombre réel, c'est de le noter sous une notation ingénieur:
 $0.00001123=1.123e-5=1123e-8$ et comme 1123 (**mantisso**) et -8 (**exposant**) sont des entiers, on peut les coder en machine.
- En javascript, la mantisse est un nombre compris entre -2^{53} et $2^{53}-1$ et l'exposant est compris entre -2^{10} et $2^{10}-1$.

Réels vs flottants: les nombres en (Algo/java)Script

- L'ordinateur ne sait manipuler nativement que des entiers et en nombre fini.
- L'astuce pour représenter un nombre réel, c'est de le noter sous une notation ingénieur:
 $0.00001123 = 1.123 \times 10^{-5} = 1123 \times 10^{-8}$ et comme 1123 (**mantisso**) et -8 (**exposant**) sont des entiers, on peut les coder avec deux entiers.
Moralité: on peut coder les flottants avec deux entiers et donc uniquement des bits (0 et 1) !
- En JavaScript, les nombres sont représentés avec 1 bit pour le signe, 53 bits pour la mantisse et 11 bits pour l'exposant. Le résultat est compris entre -2^{53} et $2^{53}-1$ et l'exposant est compris entre -2^{10} et $2^{10}-1$.

Les types de base

3 - Les caractères

- Une des finalités de l'informatique, c'est de communiquer....
- Les caractères en informatique = ce que l'on souhaite écrire.....
 - Avant 80, on souhaite écrire des messages en anglais (alphabet latin, pas d'accents, quelques caractères de ponctuation et de contrôle).
 - Après 80, on veut écrire des caractères accentués, des caractères dans des alphabets non latin....
 - Il faut des normes de codage pour cela (ASCII, Unicode,...)

Les types de base

3 - Les caractères

la table ASCII

	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	
0						(0	8	@	H	P	X	`	h	p	x	
1						!)	1	9	A	I	Q	Y	a	i	q	y
2						"	*	2	:	B	J	R	Z	b	j	r	z
3						#	+	3	;	C	K	S	[c	k	s	{
4						\$,	4	<	D	L	T	\	d	l	t	
5						%	-	5	=	E	M	U]	e	m	u	}
6						&	.	6	>	F	N	V	^	f	n	v	~
7						'	/	7	?	G	O	W	_	g	o	w	

- Il faut des normes de codage pour cela (ASCII, Unicode,...)

```
Ecrire(Ascii_vers_Caractere(36));  
Ecrire(Caractere_vers_Ascii('Z'));
```

\$
90

la table ASCII

-
-
-

	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	
0						(0	8	@	H	P	X	`	h	p	x	
1						!)	1	9	A	I	Q	Y	a	i	q	y
2						"	*	2	:	B	J	R	Z	b	j	r	z
3						#	+	3	;	C	K	S	[c	k	s	{
4						\$,	4	<	D	L	T	\	d	l	t	
5						%	-	5	=	E	M	U]	e	m	u	}
6						&	.	6	>	F	N	V	^	f	n	v	~
7						'	/	7	?	G	O	W	_	g	o	w	

- Il faut des normes de codage pour cela (ASCII, Unicode,...)

```
Ecrire(Ascii_vers_Caractere(36));  
Ecrire(Caractere_vers_Ascii('Z'));
```

\$
90

la table ASCII

- la table ASCII
- la table Unicode à partir du code 20000

	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
0	北		丰	丸	\	么	乐	乘	习	𠂇	买	乸	亩	予	亏	亘
1	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶
2	丢	个	串	为	义	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶
3	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶
4	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶
5	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶
6	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶
7	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶	丶

- Il faut des normes de codage pour cela (ASCII, Unicode,...)

```
Ecrire(Ascii_vers_Caractere(36));  
Ecrire(Caractere_vers_Ascii('Z'));
```

\$

90

la table ASCII

- NB: en javascript et dans le langage algorithme, un caractère est un symbole entouré de deux « quotes »

0	北		丰	丸	丶	𠂇	乐	乘	习	𠂔	𠂊	买	𩫕	龜	予	亏	𠂔
1	丂	丄	丂	丹	丶	义	𠂊	乙	乡	𠂔	乱	軋	乾	争	云	瓦	𠂔
2	丢	个	串	为	乂	丂	兵	し	𠂔	𠂔	姿	𠂔	亂	事	互	亚	𠂔
3	丂	丫	弗	主	乃	之	兵	丂	𠂔	𠂔	乳	𠂔	𠂔	事	亓	些	𠂔
4	丂	丂	临	丂	ㄨ	乌	乔	也	𠂔	𠂔	𠂔	𠂔	𠂔	二	五	𠂔	𠂔
5	严	中	辯	丽	久	乍	𠂔	九	𠂔	𠂔	乳	𠂔	𠂔	」	丂	巒	𠂔
6	並	丂	丂	举	乚	乎	乖	乞	书	𠂔	𠂔	乾	了	于	三	亞	𠂔
7	丧	丰	丂	ノ	毛	乏	乘	也	𠂔	𠂔	𠂔	𠂔	𠂔	尔	亏	𠂔	𠂔

- Il faut des normes de codage pour cela (ASCII, Unicode, ...)

```
Ecrire(Ascii_vers_Caractere(36));  
Ecrire(Caractere_vers_Ascii('Z'));
```

\$

90

la table ASCII

- NB: en javascript et dans le langage algorithme, un caractère est un symbole entouré de deux « quotes »

0	北		丰	丸	\	么	乐	乘	习	𠂇	买	乸	亾	予	亏	亘
1	丂	丄	丂	丹	八	义	乚	乙	乡	𠂇	乱	軋	乾	争	云	瓦
2	丢	个	串	为	乂	丂	兵	し	𠂇	𠂇	姿	𠂇	亂	事	互	亚
3	卯	丫	弗	主	乃	之	兵	丂	𠂇	𠂇	乳	菴	𠂇	事	亓	些
4	丂	丂	临	丂	メ	乌	乔	也	𠂇	𠂇	𠂇	𠂇	𠂇	二	五	𠂇
5	严	中	莘	丽	久	乍	𠂇	九	𠂇	𠂇	乳	壹	」	亍	井	巒
6	並	丂	丂	举	从	平	丂	乞	书	𡿵	𠂇	乾	了	干	三	𠂇

Moralité: on peut coder les caractères avec un entier en utilisant une table de correspondance « normalisée au niveau international» et donc uniquement des bits (0 et 1) ! ,....)

Les types de base

3 - Les chaînes de caractères

- Ce sont des suites de caractères (0, 1, 1000 caractères)...
- En javascript et dans le langage algorithmique, elles sont notées comme une suite de caractères entourée de deux quotes
 - 'Bonjour', 'Au revoir', 'première ligne\ndeuxième ligne',
....
 - 'X'
 - "" : la chaîne de caractères « vide » joue un rôle très important

Les types de base

3 - Les chaînes de caractères

- Ce sont des suites de caractères (0, 1, 1000 caractères)...
- En javascript et dans le langage algorithmique, elles sont notées comme une suite de caractères entourée de deux quotes
 - 'Bonjour', 'Au revoir', 'première ligne\ndeuxième ligne',
....
 - 'X'
- **Moralité:** on peut coder les chaînes de caractères par une suite de plusieurs entiers et donc uniquement avec des bits (0 et 1) !

Les types de base

4 - Les booléens

- Ce sont les valeurs vrai et faux (ou true et false)...

Les types de base

4 - Les booléens

- Ce sont les valeurs vrai et faux (ou true et false)...

Moralité: Voici finalement le type de base unique de l'informatique que l'on peut évidemment coder avec des bits (0=faux et 1=vrai) !

Pourquoi porter autant d'attention aux types alors que tout peut se coder par des bits ?

1. Parce qu'il est plus pratique d'écrire 'Bonjour' que
01000010011011101101110011010011011110111010101110010
2. Parce qu'on a besoin de savoir combien de place (en nombre de bits) il faut réservé en mémoire pour stocker une variable....
3. Parce que les calculs qu'on peut faire changent d'un type à l'autre.....

Les Expressions

Opérations de base, syntaxe,....

Opérations de base pour

1 - Les « entiers »

- Les opérations + (addition), - (soustraction), * (multiplication), div (division entière) et mod (reste de la division entière)
- $5+3$ vaut 8
- $17 \text{ div } 5$ vaut 3 et $17 \text{ mod } 5$ vaut 2
- Les comparaisons $=, \neq, <, >, \leq, \geq$ entre deux nombres entiers.... Le résultat d'une comparaison est un booléen ($12 = 4*3$ vaut vrai et $12 < 7$ vaut faux)

Opérations de base pour

2 - Les « réels »

- Les opérations + (addition), - (soustraction), * (multiplication), / (division)
- $5/3$ vaut 8
- $17/5$ vaut 3.6666666667
- Les comparaisons $=, \neq, <, >, \leq, \geq$ entre deux nombres réels....

Opérations de base pour

2 - Les « réels »

- Les opérations + (addition), - (soustraction), * (multiplication), / (division)

Attention ! Les calculs se font sur des nombres flottants ! On ne peut pas tout représenter ! Par exemple :

```
Ecrire(1.257*43*1000/43 == 1.257/43*43*1000); false
```

- Les comparaisons =, ≠, <, >, ≤, ≥ entre deux nombres réels....

Opérations de base pour

3 - Les caractères

- Les comparaisons $=, \neq, <, >, \leq, \geq$ entre deux caractères se fait en comparant leur deux codes Ascii (ainsi ‘A’ $<$ ‘a’ est vrai car le code ASCII de ‘A’ est 65 et celui de ‘a’ est 97, de même ‘e’ = ‘é’ est faux car ces deux symboles ont des codes différents).
- Les fonctions Caractere_vers_Ascii(‘Z’) et Ascii_vers_Caractere(110)....

Opérations de base pour

4 - Les chaînes de caractères

- La concaténation de deux chaînes de caractères notée +
 - ‘Bon’ + ‘jour’ vaut la chaîne ‘Bonjour’
- Obtenir la longueur d'une chaîne de caractères.
 - Longueur(‘Bonjour’) vaut 7
- Extraire un caractère d'une chaîne (notation ch[n], le premier caractère de la chaîne est ch[0]).
 - Si la variable ch contient comme valeur la chaîne de caractères ‘Bonjour’, ch[3] vaut ‘j’
- Les comparaisons $=, \neq, <, >, \leq, \geq$ entre deux chaînes de caractères se font en utilisant l'ordre lexicographique (i.e., l'ordre du dictionnaire).

Opérations de base pour

5 - Les booléens

- Les opérations logiques: et, ou et non

Tables de vérité (1)

non

	A	non(A)
non	Vrai	
	Faux	

Tables de vérité (1)

	A	$\text{non}(A)$
non	Vrai	Faux
	Faux	

Tables de vérité (1)

	A	$\text{non}(A)$
non	Vrai	Faux
	Faux	Vrai

Tables de vérité (2)

et

A	B	$A \text{ et } B$
Vrai	Vrai	
Vrai	Faux	
Faux	Vrai	
Faux	Faux	

Tables de vérité (2)

et

A	B	$A \text{ et } B$
Vrai	Vrai	Vrai
Vrai	Faux	
Faux	Vrai	
Faux	Faux	

Tables de vérité (2)

et

A	B	$A \text{ et } B$
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	
Faux	Faux	

Tables de vérité (2)

et

A	B	$A \text{ et } B$
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	

Tables de vérité (2)

et

A	B	$A \text{ et } B$
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Tables de vérité (3)

ou

A	B	A ou B
Vrai	Vrai	
Vrai	Faux	
Faux	Vrai	
Faux	Faux	

Tables de vérité (3)

ou

A	B	$A \text{ ou } B$
Vrai	Vrai	Vrai
Vrai	Faux	
Faux	Vrai	
Faux	Faux	

Tables de vérité (3)

ou

A	B	A ou B
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	
Faux	Faux	

Tables de vérité (3)

ou

A	B	A ou B
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	

Tables de vérité (3)

ou

A	B	A ou B
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Syntaxe d'une expression

Une expression est soit:

- une valeur (25, 3.8, 'Bonjour', la variable x,...)
- de la forme (expression) opération (expression), où les parenthèses sont facultatives et sont uniquement là pour lever des ambiguïtés....

Ainsi, tout ce qui suit représente des expressions valides (on suppose que x est une variable)

15

3 + 8

1 + (2 * x - 1)

'Bon'+ 'jour'

(5*3) + (12 -8)

1 + 2 + 3 + 4 + 5 * 2

3 + Longueur('Bonjour')

Syntaxe d'une expression

Une expression est soit:

Une expression représente donc un « calcul » valide que l'on peut utiliser au sein des instructions de l'algorithme !

Une expression possède donc une valeur (obtenue après l'évaluation de cette expression).

Attention, lorsqu'il y a des ambiguïtés dans une expression, elles sont levées en imposant des préférences (ex: la concaténation sera préférée à l'addition)

ex: dans le cas de l'expression `12 + 'Bonjour'`, l'opération choisie sera la concaténation et le résultat de l'évaluation de l'expression sera la chaîne de caractères `'12Bonjour'` ! (cf TD pour plus de détails).

`1 + 2 + 3 + 4 + 5 - 2`

`3 + Longueur('Bonjour')`

Les comparaisons en javascript

Attention aux notations en javascript !

- le test d'égalité « = » s'écrit == (ex: « x == 5 »)
- différent « ≠ » s'écrit != (ex: x != 5)
- « ou égal ≤ et ≥» s'écrivent <= et >= (ex: x <= 5)

Les opérations booléennes en javascript

Attention aux notations en javascript !

- le « et logique » s'écrit `&&` (ex: « `(5 < x) && (x<10)` »)
- le « ou logique » s'écrit `||` (ex: « `(x < -1) || (x > 1)` »)
- le « non logique » s'écrit `!` (ex: « `!(x < -1)` »)

Les Instructions

Instructions de base, séquentialité,....

Instructions de base

- L'exécution d'un algorithme consiste en l'exécution d'un suite d'instruction dans le but de changer l'état de la mémoire.
- Il y a trois instructions de base:
 - ★ les affectations
 - ★ Les saisies
 - ★ Les affichages
- Chaque instruction trouve les variables dans un état et les laisse dans un nouvel état

Les affectations

- Les instructions décrivent les changements de l'état de la mémoire
- Elles s'écrivent

nom_variable \leftarrow expression

- ex: $x \leftarrow 5$ ou $nom \leftarrow 'Bourdon'$, où x est une variable de type numérique et nom est une variable de type chaîne de caractères.

Les saisies

- Ce sont les instructions qui permettent d'interagir en entrée avec l'utilisateur (qui peut entrer des valeurs).
- Elles s'écrivent

nom_variable ← Saisie()

- ex: x ← Saisie()
- NB: en TP, on fera la distinction entre les différents types de valeurs que l'on peut saisir et on utilisera SaisieEntier(), SaisieReel() ou encore Saisie() qui permet quant à lui de saisir une chaîne de caractères.

Les affichages

- Ce sont les instructions qui permettent d'interagir en sortie avec l'utilisateur (affichage d'un résultat).
- Elles s'écrivent

Ecrire(expression)

- ex: Ecrire('Bonjour le monde')
- NB: en TP, on utilisera d'autres modes pour restituer un résultat, comme par exemple par un affichage graphique ou une sortie sonore...

En javascript

- L'affectation « `a ← 5` » s'écrit « `a=5` » (un seul égal)
- La saisie s'écrit « `a=Saisie();` »
- L'affichage s'écrit « `Ecrire('Bonjour');` »

La séquentialité

- Un algorithme comporte une suite d'instructions qui s'exécutent (par défaut) l'une après l'autre dans l'ordre d'écriture.
- on parle de séquentialité
- Afin de marquer qu'une instruction succède à une autre instruction, on utilise le symbole « ; » (et par convention et clarté d'écriture, on passe à la ligne).

ex: $x \leftarrow 5;$
 $y \leftarrow x+10; //$ La valeur de x utilisée ici est 5

Exemple d'évaluation

```
y ← 5 * (x + 10);
```

Exemple d'évaluation

$y \leftarrow 5 * (x + 10);$

Etape no 1 : on reconnaît que l'instruction est une affectation car elle est bien de la forme « variable \leftarrow expression ».

Exemple d'évaluation

$y \leftarrow 5 * (x + 10);$

Etape no 1 : on reconnaît que l'instruction est une affectation car elle est bien de la forme « variable \leftarrow expression ».

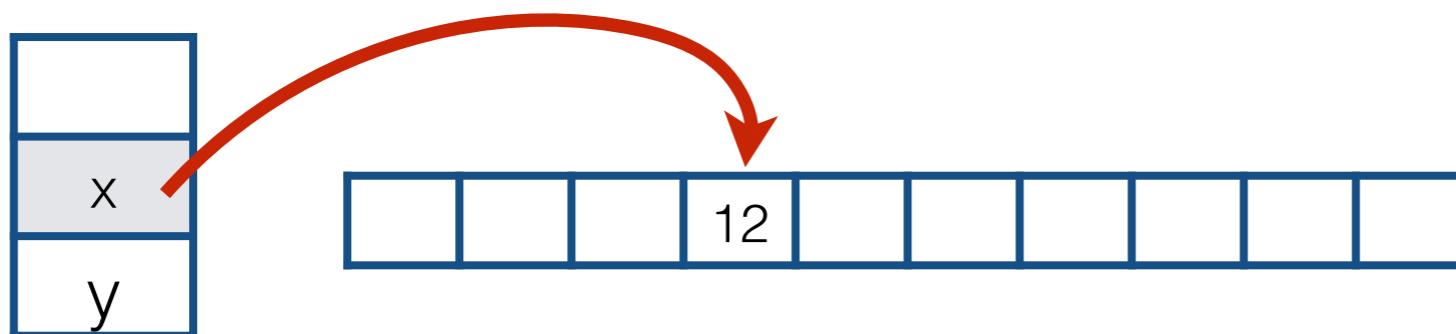
Etape no 2 : on évalue l'expression « $5 * (x + 10)$ ». Pour cela, il faut chercher en mémoire la valeur associée à la variable x

Exemple d'évaluation

$$y \leftarrow 5 * (x + 10);$$

Etape no 1 : on reconnaît que l'instruction est une affectation car elle est bien de la forme « variable \leftarrow expression ».

Etape no 2 : on évalue l'expression « $5 * (x + 10)$ ». Pour cela, il faut chercher en mémoire la valeur associée à la variable x



Exemple d'évaluation

$y \leftarrow 5 * (x + 10);$

Etape no 1 : on reconnaît que l'instruction est une affectation car elle est bien de la forme « variable \leftarrow expression ».

Etape no 2 : on évalue l'expression « $5 * (x + 10)$ ». Pour cela, il faut chercher en mémoire la valeur associée à la variable x

Etape no 3 : l'expression est donc « $5 * (12 + 10)$ ». Compte-tenu des priorités (classiques) des opérations, l'expression vaut 110.

Exemple d'évaluation

$y \leftarrow 5 * (x + 10);$

Etape no 1 : on reconnaît que l'instruction est une affectation car elle est bien de la forme « variable \leftarrow expression ».

Etape no 2 : on évalue l'expression « $5 * (x + 10)$ ». Pour cela, il faut chercher en mémoire la valeur associée à la variable x

Etape no 3 : l'expression est donc « $5 * (12 + 10)$ ». Compte-tenu des priorités (classiques) des opérations, l'expression vaut 110.

Etape no 4 : on affecte la valeur 110 à la variable y .

Exemple d'évaluation

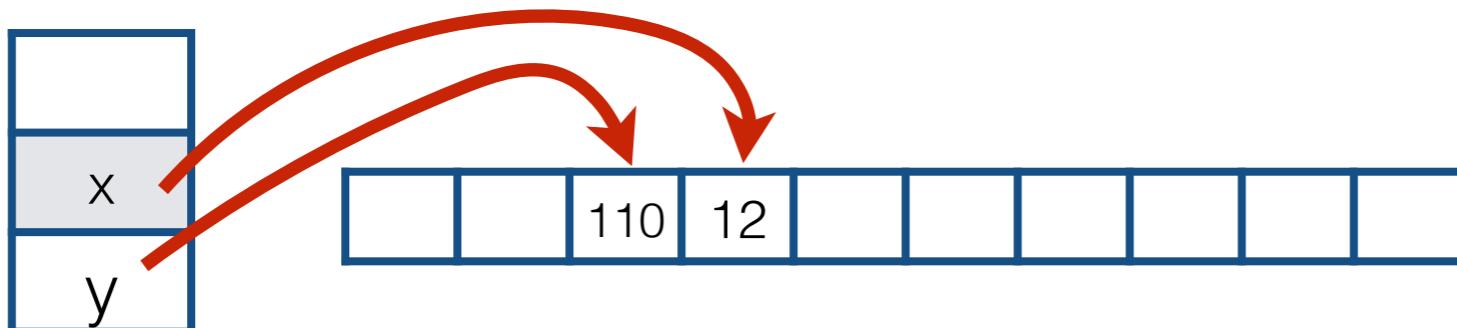
$y \leftarrow 5 * (x + 10);$

Etape no 1 : on reconnaît que l'instruction est une affectation car elle est bien de la forme « variable \leftarrow expression ».

Etape no 2 : on évalue l'expression « $5 * (x + 10)$ ». Pour cela, il faut chercher en mémoire la valeur associée à la variable x .

Etape no 3 : l'expression est donc « $5 * (12 + 10)$ ». Compte-tenu des priorités (classiques) des opérations, l'expression vaut 110.

Etape no 4 : on affecte la valeur 110 à la variable y .



L'historique d'exécution

- Pour bien comprendre le fonctionnement d'un algorithme, on peut construire un tableau représentant l'évolution de l'état de la mémoire au cours d'une exécution de l'algorithme.

Instructions	Variable 1	Variable 2
Avant exécution de l'instruction 1	?	?
Après exécution de l'instruction 1
Après exécution de l'instruction 2		
Après exécution de l'instruction 3		

Exemple

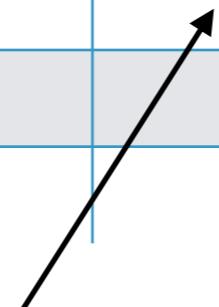
```
Algorithme Voyage
Variables:
    position, déplacement: entiers;
Début
① position ← 25 ;
② déplacement ← 11 ;
③ position ← position + déplacement ;
④ Ecrire(position);
Fin
```

Instructions	position	déplacement
Avant exécution de l'instruction 1	?	?
Après exécution de l'instruction 1	25	?
Après exécution de l'instruction 2	25	11
Après exécution de l'instruction 3	36	11
Après exécution de l'instruction 4	36	11

Construire un historique d'exécution

1. Numéroter les instructions (n)
2. Sélectionner les variables à suivre (m)
3. Construire un tableau n+2 lignes * m+1 colonnes :

Instructions	v	...	v	...	v
Avant					
Après					
Après					
...					
Après					



Valeur de la variable v_i après l'exécution
de toutes les instructions jusqu'à I_2

Bilan

- Un algorithme est une suite séquentielle d'instructions qui modifient l'état de la mémoire.
- On accède à la mémoire par le biais de variables qui possèdent un nom, un type et une valeur courante.
- Un historique d'exécution est un outil primordial pour comprendre et étudier un algorithme.

Bilan

Algorithme Calcule âge

- *Variables*

année : entier

age: entier

- *Début*

année ← Saisie();

age ← 2020 - année;

- *Ecrire(age);*

Fin

e séquentielle
t l'état de la mémoire.

par le biais de variables
type et une valeur

est un outil primordial
er un algorithme.

Bilan

Algorithme Calcule âge

- *Variables*

année : entier

age: entier

- *Début*

année ← Saisie();

age ← 2020 - année;

Ecrire(age);

Fin

Fin de l'algorithme

en Javascript:

```
// Algorithme Calcule age
```

```
var annnee,age;
```

```
annee = Saisie();
```

```
age = 2020 - annnee;
```

```
Ecrire(age);
```