Variables

Variable

Une **variable** est un **espace de stockage** identifié par un **nom** et contenant une **valeur**.

Variables IRL

Type : Dé

Valeurs possibles ?

Valeur actuelle ?

Variables IRL

Type : D20

Valeurs possibles ?

Valeur actuelle ?

Variables IRL

Type : Pièce

Valeurs possibles ?

Valeur actuelle ?

Variables IRL

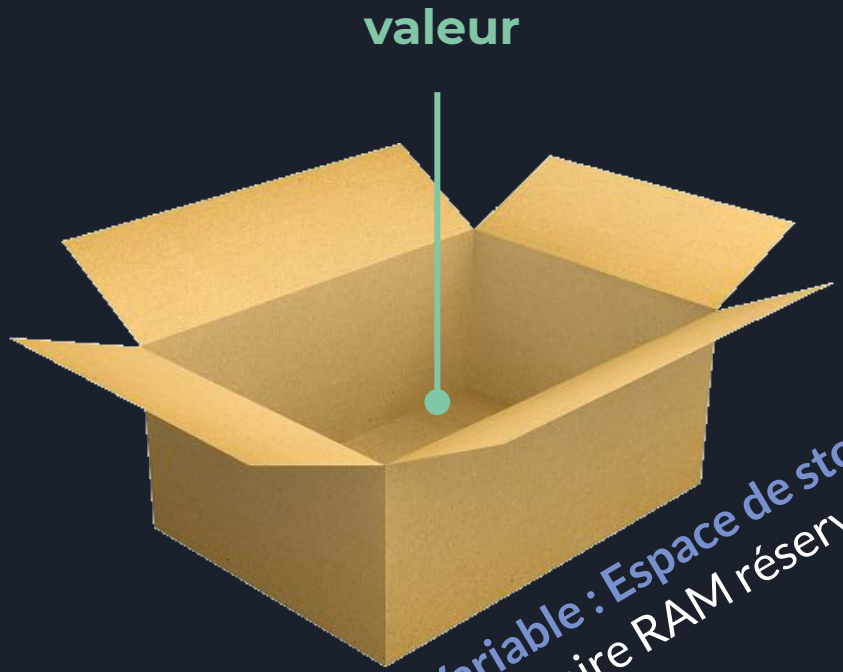
Type : Cryptex

Valeurs possibles ?

Valeur actuelle ?

Variable

maVariable
(identifiant)



La RAM

RAM : Mémoire à accès rapide.

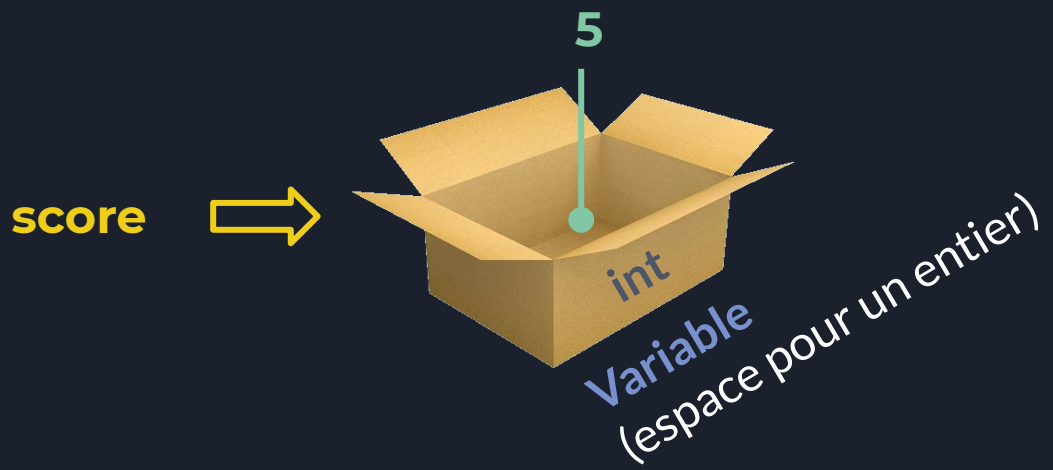
Les programmes réservent et gèrent des **espaces mémoires** dans la **RAM**.

maVariable est rangé là !

Déclaration de variables

```
int score = 5;
```

Type **Identifiant** **Affectation** **Valeur**



Type de valeur

Le **type** est le format d'une donnée. Il indique entre autre :

- Une **plage de valeurs** (i.e entre 0 et 255)
- Une **taille mémoire** (i.e. 4 octets).

Par exemple :

- Le type **int** décrit un nombre entier
- Le type **float** décrit un nombre décimal

Type de valeur

Les valeurs stockées en mémoire sont de différentes tailles en fonction de leurs **types**.

La **taille minimale** d'une valeur est l'**octet** (dans la majorité des langages).

Taille	Bits	Combinaisons
Bit = 0|1	1	2 (2^1)
Octet (byte)	8	256 (2^8)

"00000101" = 5

Type de valeur

Tous les types de valeurs ont une taille en **multiple d'octets**.

Type	Déscription	Octets	Pow	Plage de valeurs
char	Caractère	1	2^8	0 à 255
short	Entier court signé	2	2^{16}	-32 768 à 32 767
unsigned int	Entier non-signé	4	2^{32}	0 à 4 294 967 295
float	Nombre décimale	4	2^{32}	+/- 3.4 x 10^{38}

La taille des types dépend de l'architecture physique (16, 32, 64, ...) et du compilateur.

Type de valeur

La représentation binaire d'une valeur dépend de son type.

Par exemple, le 1er bit d'une valeur signée correspond au signe +/- :

Type	Déscription	Octets	Pow	Plage de valeurs
unsigned short	Entier court non-signé	2	2^{16}	0 à 65535
short	Entier court signé	2	2^{16}	-32 768 à 32 767

10100110 10100101 = -22875

Certains formats (ie float) sont beaucoup plus complexes.

Type : En résumé

Il faut des espaces mémoires de tailles différentes pour stocker des valeurs de types différents.

Type : En résumé

On ne peut pas stocker une valeur d'un type dans une variable d'un autre type !

Types de base

Les principaux types en C/C++ :

Type	Déscription	Exemple de valeur
bool	Valeur booléenne (true ou false)	true
char	Caractère	'c'
short	Entier court	5
int	Entier	-325361
long	Entier long	300 000 000 000 000
float	Nombre décimale	3.1415927f
double	Nombre décimale à double précision	3.14159265359793

Hormis bool, tous ces types peuvent être "signed" ou "unsigned".
Par exemple : unsigned int.

Langages et types

En fonction du langage, le type de valeur contenu par une variable est plus ou moins explicite.

On parle de langages :

- **Fortement typées** : C++, Java, Pascal, ...
- **Faiblement typés** : JavaScript, LUA, ...

Langages et types

Le typage est dit **fort** quand :

- Le type de valeur contenu par une variable est immuable,
- Le compilateur vérifie les erreurs de type possibles,
- La conversion d'un type à un autre doit être explicite.

Langages et types

Le C++ est fortement typé.

Le type d'une variable doit être défini lors de sa déclaration.

La variable ne pourra jamais stocker que des valeurs de ce type.

```
int score = 5;
score = 6.4f; // <= Conversion de float en int
```

Affectation

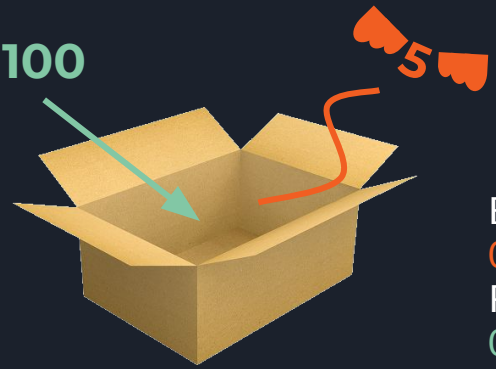
L'**affectation "="** écrit dans la zone mémoire de la variable.

Cette instruction se fait toujours de **droite à gauche**.

L'affectation **écrase** la valeur précédemment stockée.

```
int score = 5;
score = 100;


10 = score; // Erreur !
```



En mémoire, 5 :
00000101
Remplacé par, 100 :
01100100

Initialisation

```
int score;
score = 5;
```

La première affectation (symbole "=") d'une variable est appelée **initialisation**.

Initialisation

En fonction du langage et de l'environnement, une variable non-initialisée vaut :

- une valeur par défaut,
- la valeur de l'espace mémoire avant sa réservation.

```
int score;
```

Initialisation

En C++, la valeur sera celle de l'espace mémoire réservé par la variable. **Cette valeur peut être n'importe laquelle !**

Toujours initialiser ses variables après déclaration !

```
int score;
```

Cette règle est une bonne pratique quel que soit le langage !

Tableau

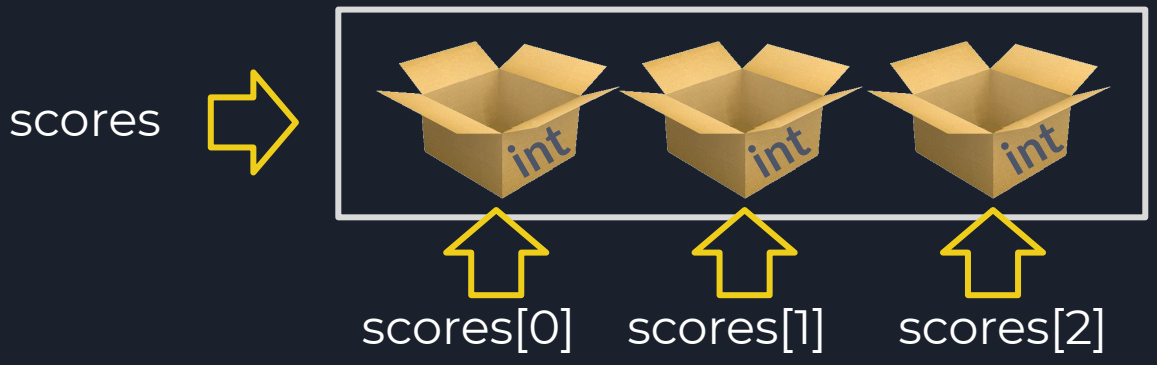
Un **tableau** est une variable stockant plusieurs valeurs contiguës en mémoire.

monTableau

Tableau

Déclaration d'une variable contenant un tableau de trois entiers :

```
int scores[3];
```



L’index d’un tableau commence toujours à 0 et fini à “taille-1”

Tableau

On **accède** à un élément grâce à son **index**.

```
scores[1] = 5;
```

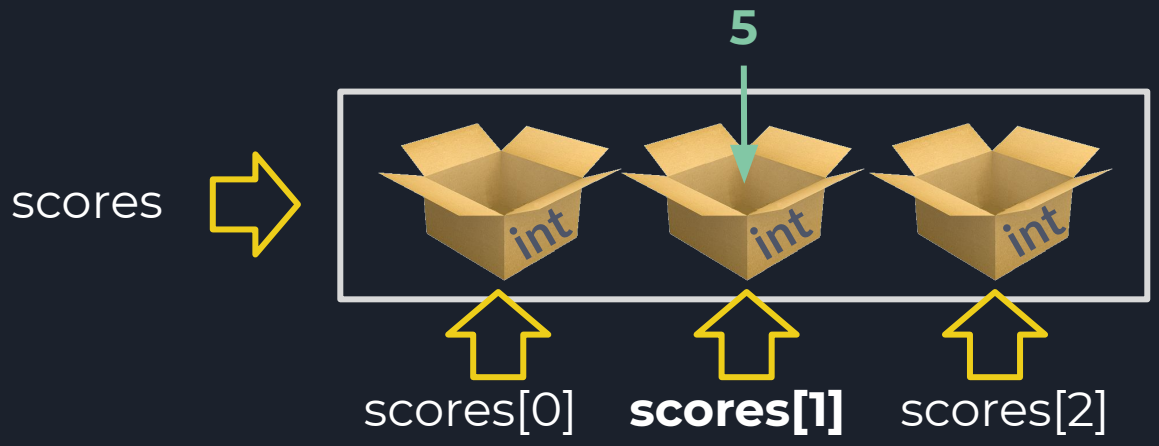


Tableau à dimensions

Un **tableau à deux dimensions** est simplement un tableau qui contient des tableaux.

tableau2

Tableau à dimensions

Déclaration d'une variable contenant un tableau à deux dimensions de valeurs entières :

```
int scores[3][2];
```

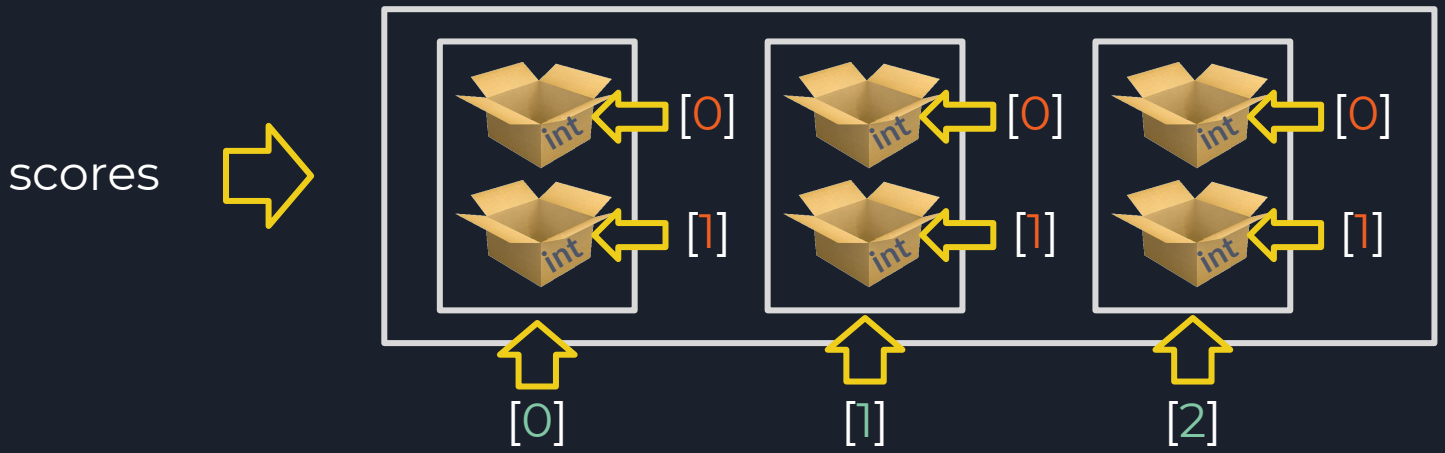
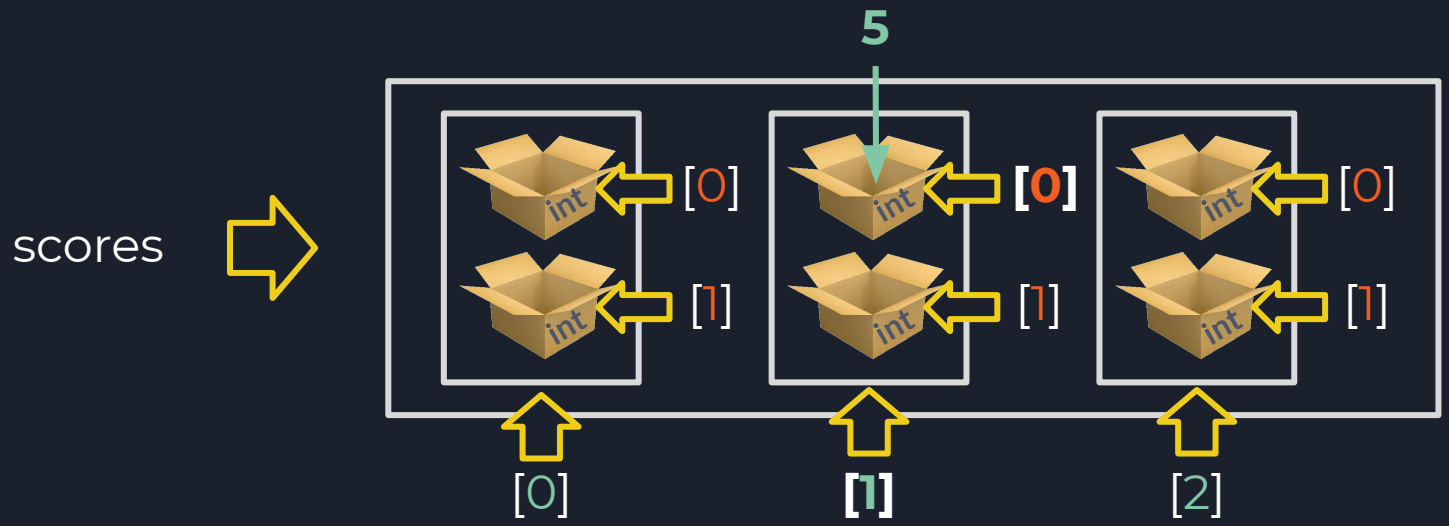


Tableau à dimensions

On accède à une valeur grâce à ses index :

```
scores[1][0] = 5;
```

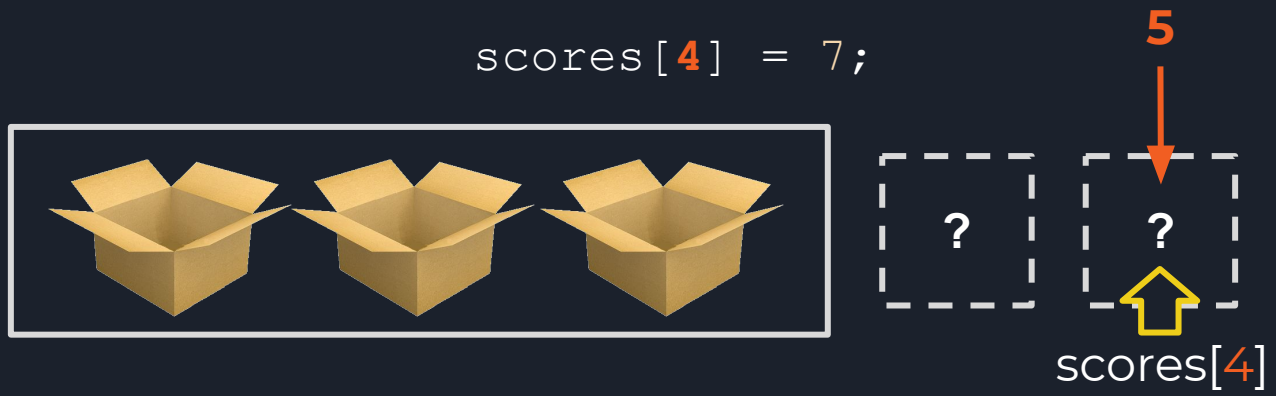


Index hors limites

Attention à **ne jamais utiliser un index supérieur ou égale à la taille d'un tableau** !

Cela revient à lire ou écrire sur une zone mémoire non-réservée par la variable !

```
scores[4] = 7;
```



Chaînes de caractères

Une **string** (chaîne de caractère) permet de stocker une **valeur textuelle**.

Deux formats différents :

- En C, une string est un **tableau de caractères**,
- En C++, le **type string** est défini dans la std (bibliothèque standard C++).

Chaînes de caractères C

Tableau de caractères en C :

```
char text[1024] = "Some text";
```

La bibliothèque C **"string.h"** propose des opérations entre tableaux de caractères.

```
#include "string.h"
…
strcat(text, " and more.");
printf(text); // Affiche : "Some text and more."
```

Chaînes de caractères C

Une chaîne de caractères doit finir par un **caractère de fin de chaîne '\0'**. La chaîne "Some text" compte donc 10 caractères.

Si le caractère '\0' n'est pas présent, les opérations manipulant la chaîne s'arrêteront au premier '\0' trouvé dans la mémoire. Aïe.

```
printf(text) // "Some textd3(1__zed à$d"
```

Chaînes de caractères C++

Type std::string C++ (standard library) :

```
#include "string"
…
std::string text = "Some text";
```

Le type std::string est une classe représentant une chaîne de caractères sécurisée.

Chaînes de caractères C++

Le type std::string propose des opérations simplifiées entre chaînes :

```
text += " and more.";
std::cout << text; // "Some text and more."

text.insert(10, "(hi) ");
std::cout << text; // "Some text (hi) and more."
```

Portée et durée de vie

Un **bloc d'instructions** se défini entre deux accolades **{ et }**.

Généralement, une variable **n'est accessible et n’existe** qu'à l'**intérieur du bloc** d'instructions dans lequel elle a été **déclarée**.

Portée et durée de vie

L'identifiant d'une variable doit être unique tant que cette variable est accessible :

```
{
    int score = 5;
    score = 8;
    int score = 3; // Erreur ! L'id score est déjà utilisé.
}

int score = 3; // Ok ! L'id score est disponible.
```

Portée et durée de vie

Une variable **définie hors des fonctions** va :

- vivre pendant la durée de l'exécution du programme,
- être accessible dans toutes les fonctions.

On parle de **variable globale** au programme.

On évitera autant que possible d'avoir des variables globales !

Transtypage

On appelle **transtypage**, ou **cast**, le fait de passer d'un type de valeur à un autre. Il est dit :

- **Implicite** si il est **déduit automatiquement** par le compilateur dans le cas ou le type de valeur ne correspond pas au type de variable ou à l'opération.
- **Explicite** si il est **précisé en instructions** dans le code.

Transtypage

Cast implicite (automatique) :

```
int someVar = 2.5f; // 2.5f passe de float à int. Résultat : 2 au lieu 2.5
```

Cast explicite avec l'instruction “(type)” :

```
float someVar = ((float)5) / 2; // 5 est converti en float (5.0f) avant la division. Résultat 2.5f et au lieu de 2.0f.
```

Transtypage

Les opérations de cast sont définies par le langage entre la majorité des types de base.

Si un valeur est transtypée en valeur de type ayant une définition inférieure, il y a risque de **perte de données**.

```
int entier = 1.3f; // entier vaut 1
float decimal = entier; //decimal vaut 1.0
```