Encodage, Algorithmie, Langage C

- Nom - prénom - age ?
- D'où est-ce-que vous venez ?
- Votre expérience en informatique ?
- Vos objectifs ?
- Langage préféré ?
- Langage que vous maitrisez le mieux ?
- Langage qui vous amuse ?
- Projet wow ?
- Projet dont vous êtes le plus fière ?

Un petit quiz

Qu'est-ce-qu'un ordinateur ?

Qu'est-ce-qu'un ordinateur ?

- Un outil qui effectue des calculs sur de l'information.

- $\Rightarrow$ Comment est représentée l'information ?
- $\Rightarrow$ Quels sont les calculs ?
- $\Rightarrow$ Comment programmer un ordinateur ?

Comment est représentée l'information ?

Le binaire

- Le binaire utilise des 'bits' à la place des chiffres.
- Un bit représente un état vrai ou faux (1 ou 0).

- $\Rightarrow$ Qu'est ce que l'on peut représenter ?

Les booléens

- Un booléen est un état qui peut être soit vrai soit faux.
- Il suffit d'un seul bit pour représenter un booléen.

Les entiers

- On utilise plusieurs bits
- On associe une puissance de deux à chaque bit par ordre décroissant.
- On somme le résultat

$$\text{nombre de bits} = \lceil \log_2(n) \rceil$$

Les entiers négatifs

- On utilise un bit pour indiquer si le nombre et négatif ou non.
- Il s'agit par convention du bit le plus à gauche

$$\text{nombre de bits} = \lceil \log_2(n) \rceil + 1$$

Les entiers

- Les entiers sont généralement représentés avec 8, 16, 32, 64 ou 128 bits.
- On distingue les entiers signés avec le bit de signe et les entiers non signés qui n'ont pas de bits de signes.

Les nombres décimaux

- Petite expérience : demandez à python ou à javascript de calculer $0.1 + 0.2$

Les nombres décimaux

- On utilise le format double, qui utilise 32 (ou 64) bits, en plaçant la virgule au milieu.
- On utilise le format flottant (IEEE754), qui utilise 32 (ou 64) bits, qui représente les valeurs au format scientifique ($a \times 10^b$)
- On utilise le format décimal qui représente les valeurs au format décimal avec du texte.

Représenter les caractères

- On utilise les nombres et une table de conversion comme la table ASCII ou UTF.
- Exercice : traduire la séquence "67-79-68-65"
- Table ASCII

Représenter des séquences

- Il faut un espace mémoire suffisant.
- Pour représenter une séquence de 10 nombres au format 8 bits, il faut 80 bits consécutifs.

L'hexadécimal

- Fonctionne comme le binaire et le format décimal mais avec 16 chiffres (de 0 à F).
- Est utilisé pour plus simplement représenter des octets.
- Exemple : 255 = 1111 1111 = FF

Binaire VS Décimal VS Hexadécimal

- La valeur "10" correspond-elle à 3 en binaire, 10 en décimal ou 16 hexadécimal ?
- Pour différencier on utilise les préfixes '0b' pour le binaire et '0x' pour l'heexadécimal.
- Sans préfixe, il s'agit du format décimal.

Les opérations

- Qu'est-ce qu'une opération pour un ordinateur ?
- Comment sont elles phisiquement réalisées ?

Les opérations

- Il existe deux types d'opérations :
  - Les opérations pour manipuler la mémoire (lire ou écrire).
  - Les opérations sur les bits. On appelle cela l'algèbre de Boole.

L'algèbre de Boole

- Ce sont des règles mathématiques.
- On utilise le concept de "porte binaire" en informatique et en électionique.
- On peut représenter une porte par son nom, ou son symbole algébrique ou son diagramme électronique.
- On peut visualiser les opérations grâce aux tables de vérités.

La porte NON

- La porte NON, notée ¬ renvoie l'inverse l'état du bit donné.

A	¬ A
0	1
1	0

La porte ET

- La porte ET, notée ., renvoie vrai si et seulement si les deux bits sont vrais.

A	B	A . B
0	0	0
0	1	0
1	0	0
1	1	1

La porte ET

- La porte ET, notée +, renvoie vrai si et seulement si les deux bits sont vrais.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

La porte XOU

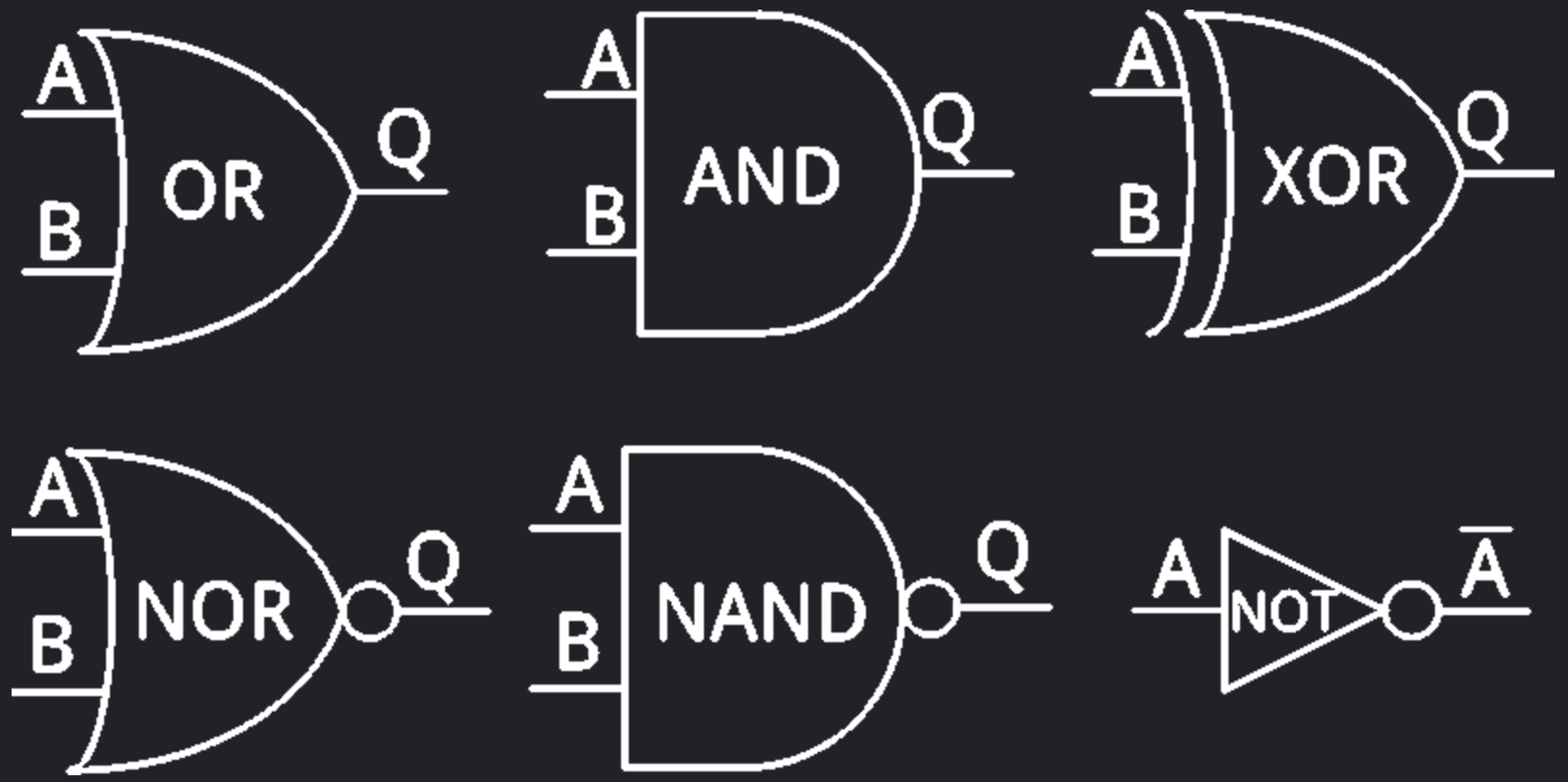
- La porte XOU, notée $\oplus$, renvoie vrai si et seulement si les deux bits sont vrais.

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

La porte XOU : Exercice

- Ouvrir ce lien : https://logic.ly/demo/
- Construire la porte XOU à partir des portes NON, ET et OU.

Les diagrammes des portes



Algrèbre de Boole : propriétés

- Priorités : Paranthèse > ET > OU > XOU.
- Associativité :
  - de la porte OU : $(A + B) + C = A + (B + C)$
  - de la porte ET: $(A.B).C = A.(B.C)$
- Commutativité: $A + B = B + A$
- Indempotence: $A + A + \cdots = A$ et $A.A.\cdots = A$

Algrèbre de Boole : propriétés

- Distributivité:
  - $A.(B+C) = (A.B) + (A.C)$
  - $A + (B.C) = (A+B).(A+C)$
- Loi de Morgan:
  - $\neg(A.B) = \neg A + \neg B$
  - $\neg(A+B) = \neg A.\neg B$

Algrèbre de Boole : propriétés

- Complémentarité:
  - $\neg\neg A = A$
  - $A + \neg A = 1$
  - $A.\neg A = 0$

Addition binaire

- Même principe que pour l'addition au format décimal.

	+1	+1		
		1	1	1
+			1	0
=	1	0	0	1

Addition binaire : Exercice

- Ouvrir ce lien : https://logic.ly/demo/
- Créer l'addition binaire avec les 3 bits d'entrée.
- Créer l'addition binaire de deux entiers positifs représentés sur 4 bits.

Addition binaire

- Que ce passe-t-il si l'on ajoute 7 ($0b0111$) et 1 ($0b0001$) ?
- Le résultat est -8 !
- C'est ce qu'on appelle l'erreur "bit overflow".
- Les langages de programmation ne réagissent pas de la même mannière.

- Rust détecte le problème et stoppe le programme :

```rust
fn main() {
    let mut num: u8 = 0;
    for _ in 0..1000 {
        num += 1;
        println!("Num : {num}");
    }
}
```

```
thread 'main' panicked at src\main.rs:4:9:
attempt to add with overflow
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
error: process didn't exit successfully: `target\debug\bit_overflow.exe` (exit code: 101)
```

- C ne détecte pas le problème

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    char num = 0;
    for (int _ = 0; _ < 130; _++) {
        num++;
        printf("Num = %d\n", num);
    }
    return EXIT_SUCCESS;
}
```

```
Num = 127
Num = -128
```

Programmer un ordinateur

Qu'est-ce qu'un programme ?

- Un programme est une suite d'instructions.

- $\Rightarrow$ Comment est représentée une instruction ?
- $\Rightarrow$ Comment un ordinateur exécute un programme ?

Représenter une instruction

- Une instruction est composée d'une opération et de paramètres.
- On utilise la ROM (Read Only Memory) pour choisir l'instruction.
- On utilise des registres comme paramètres.

Représenter une instruction : La ROM

- La ROM est une table qui associe un entier à chaque instructions réalisables par l'ordinateur.
- Chaque ROM diffère en fonction du constructeur.

Exemple de ROM

Op	Num	Param 1	Param 2	sortie
Read	1	Pos. mémoire	Taille à lire	Registre
Write	2	Valeur	Pos. mémore	
Add	3	Registre 1	Registre 2	Registre

Représenter une instruction : Exercice

- Traduire le pseudo programme suivant. Les entiers sont écris sur 8 bits :

```
Ecrire 10 dans l'emplacement mémoire 150
Ecrire 25 dans l'emplacement mémoire 238
Lire l'emplacement mémoire 150 dans le registre 2
Lire l'emplacement mémoire 238 dans le registre 3
Ajouter les registres 2 et 3 dans le registre 5
Ecrire le registre 5 à l'emplacement mémoire 96
```

Représenter une instruction : Exercice

- Les instructions, les paramètres et les sorties sont représentées sur 4 bits.

Comment est-ce que l'ordinateur exécute le programme ?

- Il y a un curseur qui pointe sur l'instruction à exécuter.
- Une horloge émet un signal régulièrement.
- A chaque signal, l'instruction pointée est exécutée et le curseur avance.

Le système d'exploitation

- Il s'agit d'un programme qui facilite l'utilisation de l'ordinateur:
  - Il assure la sécurité
  - Il normalise l'utilisation des périphériques
  - Il gère l'exécution de nos programmes

Les langages de programmation

Qu'est-ce que c'est ?

- Programmer octets par octets n'est pas envisageable pour de grosses applications.
- Un langage de programmation permet d'écrire de façon plus lisible les instructions et de ne pas dépendre de la plateforme.

Le premier langage notable

- L'assembleur est le premier langage de programmation notable.
- Il ressemble à la programmation en octet mais utilise des alais à la place

```
LDI 10 3
LDI 7 4
MUL 3 4 5
```

Le langage C

- Le langage C est une révolution dans le monde de la programmation.
- Le langage est plus ergonomique et permet de développer plus facilement et plus rapidement des applications.

Les langages modernes

- Le langage C est vieux et a quelques problèmes.
- Les langages récents ont hérités du langage C et tentent d'améliorer la programmation.
- Des langages tentent même de remplacer le langage C (Rust, Carbon et Zig par exemple).

Exemple d'un programme : Assembleur

```
global _start

section .data
    message db "Hello, World!"

section .text
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, message
    mov rdx, 13
    syscall

    mov rax 60
    mov rdi, 0
    syscall
```

Exemple d'un programme : Langage C

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("Hello, World!");
    return EXIT_SUCCESS;
}
```

Exemple d'un programme : Python

```
print("Hello, World!")
```

Comment comparer les langages de programmation ?

Langages compilés VS interprétés

- Les langages compilés crééent un fichier exécutable. Ils sont moins souples à programmer mais sont plus performants.
- Les langages interprétés utilisent une application pour exécuter les programmes. Ils sont plus souples à programmer mais moins performants.
- Des langages comme OCaml peuvent être interprétés ou compilés.

Statiquement VS dynamiquement typés

- Un type correspond à la famille d'une valeur (nombre, texte, booléen, etc...).
- Les langages statiquement typés imposent au programmeur de définir le type des valeurs. Une valeur ne peut pas changer de type.
- Les langages dynamiquement typés ne requièrent pas de définir le type des valeurs. Une valeur peut changer de type.

Statiquement VS dynamiquement typés

- Les langages dynamiquement typés sont moins souples à programmer mais plus performant.
- Les langages statiquement typés sont plus souples à programmer mais moins performant.
- De plus en plus de langages statiquement typés sont en mesure de déterminer le type d'une valeur.

Gestion de la mémoire

- Il y a deux principaux modes de gestion de la mémoire:
  - Gestion manuelle : Le programmeur doit allouer et libérer la mémoire manuellement.
  - Collecteur de déchet (Garbage Collector) : Un programme secondaire met en pause le programme principal pour nettoyer la mémoire.

Gestion de la mémoire

- La gestion manuelle complexifie la programmation mais permet produit des programmes plus performant.
- Le collecteur de déchet facilite la programmation mais diminue les performances des programmes.

Gestion de la mémoire

- Le langage Rust gère la mémoire de façon astucieuse.
- Le langage génère automatiquement les instructions d'allocation et de libération de la mémoire à la place du programmeur.
- C'est le meilleur des deux options.

Les paradigmes

- Un paradigme est la façon dont est construit un langage de programmation pour résoudre un problème
- En voici quelques un:

Les paradigmes : la programmation impérative

- La programmation impérative qui définit un programme comme une simple séquence d'instructions.
- C, C++, Java, PHP, Python, ...

Les paradigmes : La programmation orientée objet

- La programmation orientée objet où le programme est un ensemble d'entités (objets) interagissant les unes avec les autres.
- Java, C#, Python, ...

Les paradigmes : La programmation fonctionnelle

- La programmation fonctionnelle où l'ensemble des ́éléments d'un programme se comporte comme des fonctions mathématiques.
- OCaml, F#, Rust, ...

Les paradigmes : la programmation évènementielle

- la programmation évènementielle où le programme répond à différents évènements.
- C#, Java, Javascript, ...

Les paradigmes : la programmation concurrente

- la programmation concurrente où le programme peut exécuter plusieurs instructions en parallèle.
- Rust, Javascript, C#, Java, ...

Des langages conçus pour des domaines spécifiques

- PHP : exclusivement pour le web
- Swift : IOS mobile
- OCaml : preuves scientifiques et compilateurs

Quel est le meilleur langage ?

- Il n'y a pas de règle générale : tout dépend du contexte.
- Il faut choisir un langage qui est techniquement compatible avec le projet.
- Il faut chosiir un langage qui est économiquement viable pour l'entreprise.

Quel est le meilleur langage ?

- Dans le cadre de votre cursus:
  - Prioriser les langages compilés
  - Prioriser les langages statiquement typés
  - Prioriser les langages à gestion manuelle de la mémoire

Conventions de nommage

Qu'est ce qu'une convention de nommage ?

- En programmation on doit donner des noms à des variables, fonctions, objets, ...
- Une convention de nommage est une norme qui définit quelles sont les règles pour nommer un élément.
- L'objectif est d'uniformiser le code d'une application.

Conventions de nommage : Pasal case

- Le Pascal Case consiste à mettre toutes les premières lettres de chaque mot en majuscule et de les coller.
- `MonSuperNom`

Conventions de nommage : Camel case

- Le Camel Case consiste à mettre les premières lettres de chaque mot en majuscule sauf pour le premier mot qui reste en minuscule.
- `monSuperNom`

Conventions de nommage : Snake case

- Le Snake Case consiste à séparer les mots par des '_' et de mettre toutes les lettres en minuscules.
- `mon_super_nom`

Conventions de nommage : Screaming snake case

- Le Screaming Snake Case consiste à relier les mots avec un '_' et à mettre toutes les lettres en majuscule.
- `MON_SUPER_NOM`

Algorithmie

Qu'est-ce-qu'un algorithme ?

- C'est une suite d'instructions pour résoudre un problème.
- Le GPS, une recette de cuisine, ... sont des algorithmes.
- En informatique, on utilise un pseudo-code pour écrire un algorithme.

Ecrire un aglorithme

Pour écrire un algorithme, il faut être précis !

```
ALGORIHTME: Faire des macarons
    Prendre les ingrédients
    Faire la pâte
    Cuire la pâte
FIN
```

Ecrire un aglorithme

Il faut utiliser un vocabulaire compréhensible par tout le monde !

```
ALGORITHME : Trouver le trésor
    Trouver les 4 Ponéglyphes
    Déchiffrer les Ponéglyphes
    Calculer les coordonnées
    Se rendre aux coordonnées
FIN
```

Définition d'un Ponéglyphe

Ecrire un aglorithme

Les algorithmes sont en anglais !

Ecrire un algorithme : Les commentaires

- Un commentaire est un texte informatif pour les programmeurs.
- Un commentaire est ignoré par l'ordinateur.
- Un commentaire sur une ligne commence par `//`
- Un commentaire sur plusieurs lignes est contenu entre les symboles `/*` et `*/`

Ecrire un algorithme : Les commentaires

```
// Un commentaire sur une ligne

/*
Un
commentaire
sur
plusieurs
lignes
*/
```

Ecrire un algorithme : Les types atomiques

- En algorithmie il faut définir le type des valeurs. Il y a 5 types par défaut :
  - **INT** : Les entiers (**UINT** entiers non signés)
  - **BOOL** : pour les booléens
  - **FLOAT** / **DOUBLE** / **DECIMAL** : pour les décimaux
  - **CHAR** : Pour les caractères
  - **STRING** : pour les textes

Ecrire un algorithme : Les variables

Une variable est un emplacement nommé dans lequel on peut stocker une valeur. Le nom d'une variable respecte la convention de nommage **Snake case** ou **Camel case**. Il est composé de lettres (minuscules et majuscules), de chiffres et du caracturère '_' et il ne peut pas commencer par un chiffre.

Ecrire un algorithme : Les variables

- Pour définir une variable, il faut écrire son type et son nom. On assigne une valeur à une variable avec le caractère '=' (ou '←' ou ':=').
- Il est possible de déclarer une variable sans lui assigner de valeur. Il faudra d'abord lui assigner avant de s'en servir.
- Pour utiliser une variable il faut écrire son nom.

Ecrire un algorithme : Les variables

```
INT my_variable = 10
PRINT(my_variable)

FLOAT my_variable_2
my_variable_2 = 3.14
PRINT(my_variable_2)
```

Ecrire un algorithme : Les opérations

- `+` pour l'addition ou la concaténation de textes.
- `++` pour l'incrément.
- `-` pour la soustraction.
- `--` pour le décrément.
- `*` pour la multiplication.
- `/` pour la division.
- `%` pour le modulo.

Ecrire un algorithme : Les opérations raccourcis

- `+=` pour ajouter à la valeur d'une variable.
- `-=` pour soustraire à la valeur d'une variable.
- `*=` pour multiplier la valeur d'une variable.
- `/=` pour diviser la valeur d'une variable.

Ecrire un algorithme : Les opérations

- `!` pour la porte NON.
- `&` et `&&` la porte ET.
- `|` et `||` la porte OU.
- `^` la porte XOU.
- `>>` le décalage de bit à droite.
- `<<` le décalage de bit à gauche.

Ecrire un alogrithme : Les comparateurs

- `<` pour vérifier si une valeur est plus petite.
- `<=` pour vérifier si une valeur est plus petite ou égale.
- `>` pour vérifier si une valeur est plus grande.
- `>=` pour vérifier si une valeur est plus grande ou égale.
- `==` pour vérifier si deux valeurs sont égales.
- `!=` pour vérifier si deux valeurs sont différentes.

Ecrire un algorithme : Les tableaux

- Un tableau est un outil pour stocker une collection de valeurs de même type.
- Un tableau à une taille fixe.
- On accède aux valeurs du tableau à partir de leurs index.
  - La première valeur du tableau est à l'index 0.
  - La deuxième valeur du tableau est à l'index `taille` -1

Ecrire un algorithme : Les tableaux

```
INT[] my_array = INT[100];

my_array[0] = 52

my_array[99] = 99
my_array[my_array.length - 1] = 99
```

Ecrire un algorithme : Les tableaux

- Il est possible de créer un tableau en indiquant directement son contenu.
- Il n'est donc pas nécessaire de préciser la taille.

```
INT[] array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Ecrire un algorithme : Le controle de flux

Controle de flux

- Il est nécessaire de pouvoir controler l'exécution d'un programme.
- On peut conditionner l'exécution d'instructions.
- On peut répéter des instructions.

Controle de flux : Les conditions

- Il existe deux menières de conditionner l'exécution d'une instruction :
  - La structure `IF - ELSE`
  - La structure `SWITCH`

Les conditions : structure `IF` - `ELSE`

- La branche `IF` définie une liste d'instructions à exécuter si l'expression booléenne donnée s'évalue à vrai.
- La branche `ELSE` est optionnelle et permet de définir une liste d'instructions à exécuter si la branche `IF` n'est pas exécutée.
- Il est possible d'enchainer les branches `IF` et `ELSE`.

Les conditions : structure `IF` - `ELSE`

```
BOOL a = true
BOOL b = false

IF a & b THEN
    PRINT("A and B are true")
ELSE IF A THEN
    PRINT("A is true and B is false")
ELSE IF B THEN
    PRINT("B is true and A is false")
ELSE
    PRINT("A and B are false")
ENDIF
```

Les conditions : structure `SWITCH`

- La structure `SWITCH` permet d'éviter de répéter les branches `IF` - `ELSE` lorsque l'on a besoin de comparer une variable (ou autre) avec une série de valeurs.
- Chaque option est contenue dans une branche `CASE` - `BREAK`. La branche `DEFAULT`, dernière et optionnelle, s'exécute si aucune
- Une seule branche, au plus, s'exécute.

Les conditions : structure `SWITCH`

```
INT dice_roll = 4

SWITCH dice_roll THEN
    CASE 1 THEN PRINT("Dice roll 1") BREAK
    CASE 2 THEN PRINT("Dice roll 2") BREAK
    CASE 3 THEN PRINT("Dice roll 3") BREAK
    CASE 4 THEN PRINT("Dice roll 4") BREAK
    CASE 5 THEN PRINT("Dice roll 5") BREAK
    CASE 6 THEN PRINT("Dice roll 6") BREAK
    DEFAULT PRINT("Not a valid dice roll") BREAK
ENDSWITCH
```

Controle de flux : Les boucles

- Les boucles permettent de répéter une brique d'instructions. Il existe 5 structures de boucles :
  - Les boucles `LOOP`
  - Les boucles `WHILE`
  - Les boucles `DO - WHILE`
  - Les boucles `FOR`
  - Les boucles `FOREACH`

Controle de flux : Les boucles

- Une boucle répète une séquence d'instructions. Il est possible de manipuler son exécution :
  - Le mot clé `BREAK` quitte la boucle. Les instructions restantes dans la boucle ne sont pas exécutées.
  - Le mot clé `CONTINUE` passe directement à l'itération suivante. Le reste des instructions ne sont pas exécutées.

Les boucles : boucle `LOOP`

- La boucle `LOOP` permet de répéter indéfiniment des instructions.
- Il faut utilise manuellement en sortir (par exemple avec le mot clé `BREAK`).

Les boucles : boucle `LOOP`

```
INT i = 0

LOOP
    PRINT(i)
    IF i == 10 THEN
        BREAK
    ENDIF
    i++
ENDLOOP
```

Les boucles : boucle `WHILE`

- La boucle `WHILE` exécute une séquence d'instructions tant que l'expression booléenne passée en paramètre s'évalue à vrai.
- Si l'expression booléenne s'évalue à faux dès le début, la boucle n'éxécute pas la séquence d'instructions.
- Attention à ne pas créer une boucle infinie !

Les boucles : boucle `WHILE`

```
INT i = 0

WHILE i <= 10 THEN
    PRINT(i)
    i++
ENDWHILE
```

Les boucles : boucle `DO - WHILE`

- La boucle `DO - WHILE` fonctionne comme la boucle `WHILE` sauf que la séquence d'instruction est au moins exécutée une fois.

Les boucles : boucle `DO - WHILE`

```
INT i = 0
    PRINT(i)
    i++
DO

WHILE i <= 10
```

Les boucles : boucles `FOR`

- La boucle `FOR` permet, à l'aide d'un compteur, de répéter un certain nombre de fois une séquence d'instructions.
- Il faut définir une variable compteur, une valeur limite et une fonction pour mettre à jour le compteur.
- Il est possible d'anonymiser le compteur si sa valeur n'est pas utile.

Les boucles : boucles `FOR`

```
FOR INT i = 0; i <= 10; i++ THEN
    PRINT(i)
ENDFOR


FOR INT _ = 10; _ > 0; _-- THEN
    PRINT("Hello")
ENDFOR
```

Les boucles : boucles `FOR`

- La boucle `FOR` est particulièrement utile pour parcourir un tableau :

```
INT[] array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

FOR INT i = 0; i < array.length; i++ THEN
    PRINT(array[i])
ENDFOR
```

Les boucles : boucles `FOREACH`

- La boucle `FOREACH` est une variante de la boucle `FOR` .
- Elle sert uniquement à parcourir les valeurs d'une séquence.
- La boucle `FOREACH` définie une variable qui contiendra itération après itération les valeurs de la séquence.

Les boucles : boucles `FOREACH`

```
INT[] array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

FOR INT i = 0; i < array.length; i++ THEN
    PRINT(array[i])
ENDFOR

FOEARCH value IN array THEN
    PRINT(value)
ENDFOREACH
```

Les fonctions et procédures

Fonctions et procédures

- Diviser le code en petites briques pour :
  - Ne pas se répéter
  - Faciliter la lecture du code
- Une fonction, comme en mathématiques, renvoie une valeur.
- Une procédure ne renvoie pas de valeurs.
- Pour renvoyer une valeur (ou quitter une procédure) on utilise le mot clé `RETURN` suivi de la valeur.

Fonctions et procédures

- Les fonctions et procédures acceptent des paramètres.
- Il faut définir le type et le nom de chaque paramètre.
- Pour appeler une fonction ou une procédure, on écrit son nom, suivi de parenthèses dans lesquelles on place les valeurs des paramètres dans l'ordre.

Fonctions et procédures

- Le nom des fonctions et des procédures sont composés de lettres, chiffres et du caractère '_'. Leurs noms ne doivent pas commencer par une chiffre. Ils respectent la convention de nommage "Snake case" ou "Camel case".
- Les noms doivent indiquer ce que fait la fonction. Ils commencent généralement par un verbe.

Les fonctions : exemple

```
FUNCTION user_to_string(STRING name, INT age, BOOL is_student) STRING THEN
    STRING text

    IF is_student THEN
        text += "Student "
    ELSE
        text += "Adult "
    ENDIF

    text += "is " + age.to_string() + " years old"

    RETURN text
ENDFUNCTION
```

Les procédures : exemple

```
PROCEDURE say_hi(STRING name) THEN
    PRINT("Hello, " + name)
ENDPROCEDURE
```

Les fonctions et procédures : la récursivité

- Une fonction peut s'appeler elle-même : c'est la récursivité.
- Comme pour les boucles il faut faire attention

Les fonctions et procédures : la récursivité

```
PROCEDURE print_counter(INT i) THEN
    IF i > 10 THEN
        RETURN
    ENDIF

    PRINT(i)
    print_counter(i + 1)
ENDPROCEDURE
```

Gestion de la mémoire

- Un programme a besoin de stocker dans la mémoire deux choses:
  - Les variables
  - Les appels de fonctions et procédures.

- $\Rightarrow$ Comment ces données sont-elles stockées ?

Gestion de la mémoire

- Le programme demande de stocker une donnée à l'OS qui lui réserve un emplacement mémoire.
- Une variable est en réalité le numéro de l'emplacement mémoire réservé.

- $\Rightarrow$ Quels sont les espaces mémoires dont disposent les programmes ?

Gestion de la mémoire : Stack et Heap

- Il y a deux espaces mémoires:
  - Le stack : il s'agit d'un espace mémoire exclusivement réservé pour le programme.
  - Le heap : il s'agit d'un espace mémoire commun aux différents programmes.

Gestion de la mémoire : Stack

- Le stack permet de stocker plus rapidement des valeurs.
- Le stack stocke les appels de fonctions et de procédures.
- Son espace est limité il faut donc faire attention.
- S'il n'y a plus de place (trop d'appels de fonctions par exemple), on appelle cela un "Stack overflow".

Gestion de la mémoire : Heap

- Le heap est plus lent pour stocker des valeurs mais il est plus grand.
- Ce sont généralement les valeurs volumineuses qui y sont stockées.

Complexité

La complexité : notation "Big O"

- La complexité c'est l'étude de l'empreinte mémoire ou du temps d'exécution d'un algorithme.
- La notation "Big O" est une évaluation par ordre de grandeur de la compexité.
- La notation "Big O" évalue en fonction de 'n'. 'n' correspond à la charge de travail (une taille d'une collection par exemple).

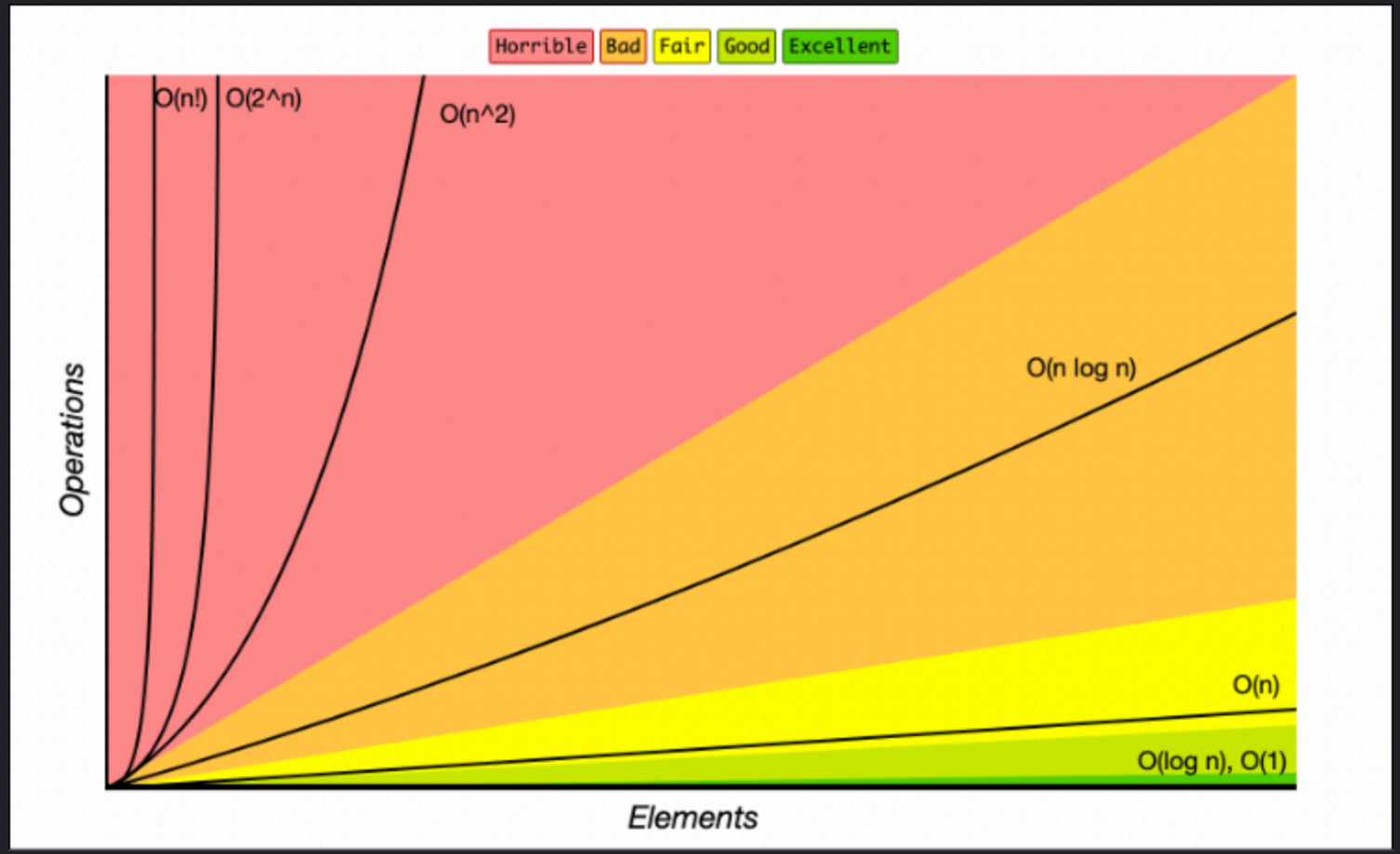
La complexité : notation "Big O"

- Il existe trois notations "Big" :
  - La notation Ω qui évalue la complexité dans le meilleure des cas.
  - La notation Θ qui évalue la complexité exacte.
  - La notation O qui évalue la complexité dans le pire des cas.

La complexité : notation "Big O"

- Les valeurs remarquables de la notation "Big O" sont:
  - $O(1)$
  - $O(n \log n)$
  - $O(n)$
  - $O(\log n)$
  - $O(n^2)$
  - $O(2^n)$
  - $O(n!)$

La complexité : notation "Big O"



Créer nos propres types

Combiner les valeurs

- Il est possible de combiner des valeurs afin de créer de nouveaux types.
- Un objet ou une entité peut être décomposé en types atomiques.

Combiner les valeurs

- Il existe plusieurs facon, en fonction des langages, de combiner les types :
  - Les structures (langage C)
  - Les classes (programmation orientée objet)

Combiner les valeurs

- Nous utiliserons les structures.
- Le nom d'une structure est composé de lettres et du caractère '_'. Il respecte la convention de nommage "Pascal case" ou "Snake case".
- Les attributs d'une structure sont définis comme des paramètres de fonction.

Combiner les valeurs

```
struct person {
    int age;
    char[] name;
};
```

```
class Person {
    int age;
    string name;
}
```

Combiner les valeurs

- Pour instancier une structure il faut lister les valeurs entre accolades dans l'ordre des attributs.
- Pour accéder auxattributs d'une structure on utilise le caractère '.'.

Combiner les valeurs

```
STRUCTURE Person
    INT age
    STRING name
ENDSTRUCTURE

Person person = { 10, "Tom" }
PRINT()
```

Les énumérations

Les énumérations

- Une énumération est un moyen de lister différentes options.
- Chaque option possède une valeur entière différente.
- L'objectif est de faciliter la lecture du code.
- Le nom d'une énumération est composé de lettres et du symbole '_'. Il respecte la convention "Pascal case" ou "Snake case". Les options d'une énumération sont nommées avec la convention "Screaming snake case".

Les énumérations

```
PROCEDURE move(INT direction) THEN
    SWITCH move THEN
        CASE 0 THEN PRINT("Move north") BREAK
        CASE 1 THEN PRINT("Move east") BREAK
        CASE 2 THEN PRINT("Move south") BREAK
        CASE 3 THEN PRINT("Move west") BREAK
    ENDSWITCH
ENDPROCEDURE

// Pas très explicite
move(0)
```

Les énumérations

```
ENUM Direction
    NORTH
    EAST
    SOUTH
    WEST
ENDENUM

PROCEDURE move(Direction direction) THEN
    SWITCH move THEN
        CASE NORTH THEN PRINT("Move north") BREAK
        CASE EAST THEN PRINT("Move east") BREAK
        CASE SOUTH THEN PRINT("Move south") BREAK
        CASE WEST THEN PRINT("Move west") BREAK
    ENDSWITCH
ENDPROCEDURE

move(NORTH)
```

Une variable de plusieurs types ?

Une variable de plusieurs types ?

- Pour simplifier et rendre générique le code d'une application on peut définir une valeur ayant différent types.
- L'idée est dire qu'une variable peut représenter différentes choses et qu'il faut bien bien distinguer les différents cas.

Une variable de plusieurs types ?

- On peut y parvenir de différentes façons en fonction des langages de programmation :
  - Les unions (langage C).
  - L'héritage, les classes abstraites (programmation orientée objet).
  - Les types énumérés (programmation fonctionnelle).
  - Les traits, interfaces et templates (programmation fonctionnelle).

Une variable de plusieurs types ?

- Nous utiliserons les unions.
- Le nom d'une union est composé de lettres et du symbole '_'. Il respecte la convention de nommage "Pascal case" ou "Snake case".
- Les variantes d'une union sont définies comme les attributs d'une structure.
- Pour instancier une union on place la valeur entre accolades.

Une variable de plusieurs types ?

- Pour accéder à une variante on utilise le symbole '.'.
- Pour savoir quelle variante est contenue dans l'union on utilise la structure `SWITCH`.

Une variable de plusieurs types ?

```
UNION MyUnion
    INT an_int;
    BOOL a_boolean;
    CHAR a_char;
ENDUNION

MyUnion my_union = { 10 }

SWITCH my_union THEN
    CASE INT int_value THEN PRINT("Its an integer : " + int_value.to_string()) BREAK
    CASE BOOL bool_value THEN PRINT("Its a boolean : " + bool_value. to_string()) BREAK
    CASE CHAR char_value THEN PRINT("Its a char : " + char_value.to_string()) BREAK
ENDSWITCH
```

Une variable de plusieurs types ?

- En C++

```
union MyUnion {
    int an_integer;
    bool a_boolean;
    char a_char;
}
```

Une variable de plusieurs types ?

- En C#

```
abstract class Value {
    abstract string to_string();
}

class Integer : Value {
    int val;
    string to_string() { return val.to_string(); }
}

class Char : Value {
    char val;
    string to_string() { return char.to_string(); }
}
```

Une variable de plusieurs types ?

- En OCaml

```
type MyValue =
| an_integer of int
| a_boolean of bool
| a_char of char
```

Une variable de plusieurs types ?

- En Rust 1/2

```
enum MyValue {
    AnInteger(i64),
    ABoolean(bool),
    AChar(char)
}
```

Une variable de plusieurs types ?

- En Rust 2/2

```
trait ToString {
    pub fn (&self) -> String;
}

struct AnInt(i64);
struct ABool(bool);
struct AChar(char);

impl ToString for AnInt { ... };
impl ToString for ABool { ... };
impl ToString for AChar { ... };
```

Les pointeurs

Les pointeurs

- Les pointeurs permettent de stocker une addresse dans la mémoire.
- Ils sont utilisés pour ne pas avoir à réserver d'espaces mémoires continus.
- Ils sont également utilisés pour modifier le contenu des structures et des unions.

Structures de données

Structure de données : définition

- Une structure de données permet d'organiser des informations.
- Elles permettent de représenter des relations, des priorités, etc…

Structure de données : Les listes