Boucle

- Structure itérative
- Nombre d'itération connue
- Nombre d'itération inconnue
- Quiz

Structure itérative

- Nous avons vue les structures impératives. Un bloc de code s'exécute de haut en bas.
- Nous avons vue les structures contionnelles. Un bloc code s'exécute seulement si une condition booléenne est respecté.
- Nous allons maintenant ajouter les structures itératives. Un bloc de code s'exécute plusieurs fois de suite.
    - Les deux types de base d'itération sont:
        - `POUR ... À` -> Quand le nombre d'itération à faire est connu d'avance.
        - `TANT QUE` -> Quand le nombre d'itération à faire est inconnue et doit être déterminé par une condition.
    - *Nous en verrons quelques autres plus tard dans la session, mais ces deux structures couvrent 70% de cas d'utilisation.*

Nombre d'itération connue: POUR ... À

Notez que je trouve la traduction `POUR ... À` *un peu bizarre alors je vais souvent l'appeler par son nom anglais,* `FOR`

- Le principe d'une boucle `POUR ... À` est de déclarer une variable de type ENTIER avec une intervalle de valeur
- Puis, on écrit le bloc de code qui sera exécuté en boucle.
- Le bloc de code sera exécuté autant de fois que l'intervalle de valeur défini au début.
    - De plus, à chaque itération, la variable de l'entier changera pour représenter l'intervalle.

Nombre d'itération connue: POUR ... À

```
POUR ENTIER i = 0 À 5 // i va prendre,  itérativement, toutes les valeurs entre 0 et 4 (la fin de la boucle est exclusive)
    ÉCRIRE i  // Ce bout de code s'exécutera donc 5 fois.
FINPOUR
```

Le pseudo-code suivant affichera:

```
0
1
2
3
4
```

Nombre d'itération inconnue: TANT QUE

- Le principe d'une boucle TANT QUE est de définir une condition booléene, et tant que cette condition est vrai, la boucle va se poursuivre.
    - On écrit ensuite le bloc de code à répéter tant que la condition est vrai.
- Avec une boucle TANT QUE, il n'y a pas de variable associée à l'itération, mais on peut en définir une hors de l'itération et la modifier dans l'itération.

Nombre d'itération inconnue: TANT QUE

```
ENTIER i = 0
TANT QUE i < 5 // La boucle vas donc se poursuivre, tant que i sera plus petit que 5.
    ÉCRIRE i  // Ce bout de code s'exécutera donc 5 fois.
    i++
FINTANTQUE
```

Le pseudo-code suivant affichera:

```
0
1
2
3
4
```

Boucle infini

- Attention, le plus gros danger avec les boucles est de créer une boucle infinie.
- Il s'agit d'un cas où la condition de sortie de la boucle ne sera jamais atteinte, le logiciel tournera donc en boucle à l'infinie.
- Le programme pourrait alors donner l'impression d'être *gelé*
- Par exemple (à ne pas faire):

```
ENTIER i = 0
TANT QUE i < 5 // La boucle vas donc se poursuivre, tant que i sera plus petit que 5.
    ÉCRIRE i  // Ce bout de code s'exécutera donc 5 fois.
    // Ici, j'ai oublié volontairement le i++ pour créer la boucle infini.
FINTANTQUE
```

Le pseudo-code suivant affichera:

```
0000000... jusqu'à l'infini...
```

QUIZ

1. Qu'est-ce que la boucle suivante affichera?

```
POUR ENTIER i = 0 À 7
    ÉCRIRE 1
FINPOUR
```

1. Réponse

```
1
1
1
1
1
1
1
1
```

2. Qu'est-ce que la boucle suivante affichera?

```
ENTIER i = 5
TANT QUE i < 11
    SI i % 2 == 0
        ÉCRIRE i
    FINSI
    i++
FINTANTQUE
```

2. Réponse

```
6
8
10
```

3. Qu'est-ce que la boucle suivante affichera?

```
ENTIER i = 0
TANT QUE i < 5
    ÉCRIRE i
    i--
FINTANTQUE
```

3. Réponse

```
0
-1
-2
-3
-4
-5
... boucle infinie...
```

4. Décrivez dans vos mots ce que fait ce programme.

```
STRING text
BOOLEEN loop = true

TANTQUE loop
  ECRIRE "Veuillez écrire une instruction: "
  LIRE text
  SI text == "exit"
    loop = false
  SINON
    ECRIRE "Vous avez écrit: " + text
  FINSI
FINTANTQUE
```

4. Réponse

- Le programme demande à l'utilisateur d'inscrire une intruction.
- Si l'utilisateur écrit autre chose que "exit", le programme répète l'instruction et demande à l'utilisateur d'inscrire une nouvelle instruction.
- Si l'utilisateur écrit "exit", le programme ne répète pas l'instruction et sort de la boucle. Ce sera alors la fin du programme.