THÈME 4:
BOUCLES WHILE

Notion du thème:

- Boucle **`while`** :
  - Fonctionnement et syntaxe
  - Compteur de boucle
  - Accumulateur et drapeau
  - Boucles imbriquées

Fonctionnement et syntaxe

- But: **répéter** des instructions jusqu'à ce qu'une condition change.
- Syntaxe:

```
while condition:
    instructions
suite_du_programme
```

- Les `instructions` seront répétées tant que `condition` est vraie.
  Lorsque que `condition` devient fausse, on exécute `suite_du_programme`.

Exemple: division de A par B

Un programme qui demande un entier A, puis un entier B jusqu'à ce que celui-ci soit non-nul, puis qui calcule le quotient de A par B.

```
a = int(input("Donnez la valeur de A : "))
b = int(input("Donnez la valeur de B : "))
while b == 0 :
    b = int(input("B est nul! Recommencez : "))
print("A / B = ", a // b)
```

*Note: si b est non-nul dès le premier essai, on n'entre **pas** dans le while.*

Attention aux boucles infinies!

Si la condition de la boucle while ne devient jamais fausse, le programme boucle **indéfiniment**:

Exemple 1:

```
n=5
while n<10 :
      print("n vaut :", n)
print("Fin")
```

Exemple 2:

```
while True :
   print("Je boucle.")
print"("Fin")
```

Compteur de boucle

- Sert à compter **combien de fois l'on passe** dans la boucle (ou un nombre qui dépend de cela)

```
b = int(input("Donnez un entier b non-nul:"))
n = 0 # n est le compteur de boucle
while b==0:
    n=n+1
    b=int(input("Incorrect, recommencez: "))
print("Merci, il vous a fallu ", n , "essais supplémentaires.")
```

Important: Toujours pensez à initialiser le compteur!

Compteur de boucle: exemples

- On peut se servir du compteur dans la condition.

```
i = 0 # variable compteur
while i<10:
      print( 2 ** i ) # 2 puissance i
      i = i + 1 # incrémenter notre compteur
print("Fin")
```

Le **pas** = augmentation du compteur à chaque étape

Ici le pas est égal à 1.

Compteur de boucle: exemples

- On peut utiliser un pas différent de 1 (même négatif)

```
i = 0 # variable compteur
while i<100:
      print(i)
      i = i + 2
print("Fin")
```

Ici le pas = 2

Compteur de boucle: exemples

- On peut utiliser un pas différent de 1 (même négatif)

```
i = 10 # variable compteur
while i>0:
      print(i)
      i = i - 1
print("Fin")
```

Ici, pas: -1

Variable "accumulateur"

- Pour stocker des informations sur les valeurs parcourues, par exemple la somme:

```
i=1
somme = 0 # initialement, la somme est égale à 0
while i <= 10 :
        somme = somme + i
        # chaque valeur de i est rajoutée à la somme
        #(accumulation)
        i = i + 1
        # ne jamais oublier de mettre à jour le compteur
print("La somme des 10 premiers entiers est : ", somme)
```

Initialisation de « l'accumulateur »

1. Attention à ne pas l'oublier
2. Utiliser l'élément neutre de l'opération
   - Pour l'**addition** entre nombres: **0** (car x+0=x)
   - Pour la **multiplication** entre nombres: **1** (car x*1=x)
   - Pour la **concaténation** entre str: **""** (chaîne vide)
     (car ""+"bonjour"="bonjour")

Variable « drapeau »

- Un accumulateur booléen est appelé **drapeau**.
- Exemple: lire 10 entiers et vérifier qu'ils sont tous impairs:

```
i=0
tous_impairs = True
while i < 10:
        x = int(input('Entrez un entier:'))
        tous_impairs = tous_impairs and (x % 2 != 0)
        i=i+1
if tous_impairs:
        print('Tous les nombres entrés sont impairs')
else:
        print('Au moins un nombre entré n'était pas impair')
```

Initialisation d'un « drapeau »

- Utiliser l'élément neutre des opérations booléennes
  - Pour un **ET**: `True`
    car `True and b` vaut `b`
  - Pour un **OU** : `False`
    car `False or b` vaut `b`

Le mot-clé **break**

- Permet de **sortir immédiatement** de la boucle while

```
i=1
while i<100:
      if i % 2 == 0 :
            print("*")
            break
      i=i+1
      print("Incrementation de i")
print("Fin")
```

Le mot-clé **continue**

- Permet de **remonter immédiatement au début** de la boucle while en ignorant la suite des instructions dans la boucle.

```
i=1
while i<100:
      if i % 2 == 0 :
            print("*")
            continue
      i=i+1
      print("Incrementation de i")
print("Fin")
```

Break et continue

- Inconvénients:
  - Code plus difficile à lire/analyser si plusieurs niveaux d'imbrications et/ou longues instructions dans le while
  - N'a pas toujours d'équivalent dans les autres langages de programmation

➔ On essaiera tant que possible de se passer de break et continue.

Au lieu de break

```
while cond:
      instructions
      if … :
            …
            break
      other_instruct
```

```
stop=False
while (not stop) and cond:
      instructions
      if … :
            …
            stop=True
      if not stop:
            other_instruct
```

Boucles imbriquées

- Un instruction d'une boucle while peut être une boucle while

Ex : résultat produit par ce programme ?

```
i = 1
while i <= 3 :
    j = 1
    while j <= 2 :
        print(i, ", ", j)
        j = j + 1
    i = i + 1
```

Résultat :

```
1, 1
1, 2
2, 1
2, 2
3, 1
3, 2
```

Exemple d'application

On veut écrire un programme qui affiche un « carré » de `n` x `n` fois le caractère '*'. L'utilisateur choisit le côté `n` du carré.

Exemple de résultat :

```
Entrez la valeur de n : 5
*****
*****
*****
*****
*****
```

Correction

```
n = int(input("Entrez la valeur de n : "))
num_lig = 0 # compteur de ligne
while num_lig < n :

    num_col = 0 #cpt nb etoiles de la ligne

    while num_col < n :
        print("*", end="") # /!\ end
        num_col = num_col + 1
    print() # saut de ligne

    num_lig = num_lig + 1 # passer ligne suivante
```

- Ces slides ont été réalisés par:
  - Amir Charif
  - Lydie Du Bousquet
  - Aurélie Lagoutte
  - Julie Peyre
- Leur contenu est placé sous les termes de la licence **Creative Commons CC BY-NC-SA**