Algorithmique et Programmation

Instructions répétitives

Elise Bonzon

elise.bonzon@mi.parisdescartes.fr

LIPADE - Université Paris Descartes
http://www.math-info.univ-paris5.fr/∽bonzon/

Instructions répétitives

1. Pourquoi des instructions répétitives ?
2. Boucle `while`
3. Notion de terminaison
4. Boucle `while` : quelques exemples
5. Boucle `for`
6. Boucle `for` : quelques exemples
7. Pour conclure

Pourquoi des instructions répétitives ?

Pourquoi des instructions répétitives ?

Objectif des instructions répétitives

Pouvoir **répéter** une (suite d')action(s) **aussi longtemps que nécessaire**

- Analogie culinaire :
  - monter des blancs en neige
  - cuire un gâteau
- Répéter des actions *similaires*, potentiellement *différentes*
- Comment exprimer **aussi longtemps que nécessaire** ?
- Comment s'assurer que le programme termine ?

Somme des premiers entiers

Comment calculer la somme des 5 premiers entiers ?

(On suppose que $\sum_{i=1}^{n} \frac{n(n+1)}{2}$ n'existe pas)

Somme des premiers entiers

Comment calculer la somme des 5 premiers entiers ?

(On suppose que $\sum_{i=1}^{n} \frac{n(n+1)}{2}$ n'existe pas)

```
>>> somme5 = 1 + 2 + 3 + 4 + 5
>>> somme5
15
```

Somme des premiers entiers

Comment calculer la somme des 5 premiers entiers ?

(On suppose que $\sum_{i=1}^{n} \frac{n(n+1)}{2}$ n'existe pas)

```
>>> somme5 = 1 + 2 + 3 + 4 + 5
>>> somme5
15
```

Ok. Mais quelle est la somme des 50 premiers entiers ?

Somme des premiers entiers

Comment calculer la somme des 5 premiers entiers ?

(On suppose que $\sum_{i=1}^{n} \frac{n(n+1)}{2}$ n'existe pas)

```
>>> somme5 = 1 + 2 + 3 + 4 + 5
>>> somme5
15
```

Ok. Mais quelle est la somme des 50 premiers entiers ?

```
>>> somme50 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 +
12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20 + 21 + 22 + 23 +
24 + 25 + 26 + 27 + 28 + 29 + 30 + 31 + 32 + 33 + 34 + 35 +
36 + 37 + 38 + 39 + 40 + 41 + 42 + 43 + 44 + 45 + 46 + 47 +
48 + 49 + 50
>>> somme50
1275
```

Somme des premiers entiers

Comment calculer la somme des 5 premiers entiers ?

(On suppose que $\sum_{i=1}^{n} \frac{n(n+1)}{2}$ n'existe pas)

```
>>> somme5 = 1 + 2 + 3 + 4 + 5
>>> somme5
15
```

Ok. Mais quelle est la somme des 50 premiers entiers ?

```
>>> somme50 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 +
12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20 + 21 + 22 + 23 +
24 + 25 + 26 + 27 + 28 + 29 + 30 + 31 + 32 + 33 + 34 + 35 +
36 + 37 + 38 + 39 + 40 + 41 + 42 + 43 + 44 + 45 + 46 + 47 +
48 + 49 + 50
>>> somme50
1275
```

Ok... Mais quelle est la somme des 100 000 premiers entiers ?

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$
  2. ajouter à s le prochain entier naturel, 1 : $s = 1$

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$
  2. ajouter à s le prochain entier naturel, 1 : $s = 1$
  3. ajouter à s le prochain entier naturel, 2 : $s = 3$

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$
  2. ajouter à s le prochain entier naturel, 1 : $s = 1$
  3. ajouter à s le prochain entier naturel, 2 : $s = 3$
  4. ajouter à s le prochain entier naturel, 3 : $s = 6$

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$
  2. ajouter à s le prochain entier naturel, 1 : $s = 1$
  3. ajouter à s le prochain entier naturel, 2 : $s = 3$
  4. ajouter à s le prochain entier naturel, 3 : $s = 6$
  5. ajouter à s le prochain entier naturel, 4 : $s = 10$

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$
  2. ajouter à s le prochain entier naturel, 1 : $s = 1$
  3. ajouter à s le prochain entier naturel, 2 : $s = 3$
  4. ajouter à s le prochain entier naturel, 3 : $s = 6$
  5. ajouter à s le prochain entier naturel, 4 : $s = 10$
  6. ajouter à s le prochain entier naturel, 5 : $s = 15$

Somme des premiers entiers

- On veut calculer la valeur de $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5$
- Approche itérative :
  1. Initialement, $s = 0$
  2. ajouter à s le prochain entier naturel, 1 : $s = 1$
  3. ajouter à s le prochain entier naturel, 2 : $s = 3$
  4. ajouter à s le prochain entier naturel, 3 : $s = 6$
  5. ajouter à s le prochain entier naturel, 4 : $s = 10$
  6. ajouter à s le prochain entier naturel, 5 : $s = 15$
- Le traitement est (quasiment) **identique à chaque étape**

Somme des premiers entiers

Ce qu'on veut faire :

- Avoir une variable s pour mémoriser les additions successives déjà effectuées
- Avoir une variable i pour représenter successivement les entiers 1 à 5 que l'on veut sommer
- Répéter les opérations suivantes 5 fois :
  1. Ajouter la valeur du i courant à s
  2. incrémenter la valeur de i

Somme des premiers entiers

Ce qu'on veut faire :

- Avoir une variable s pour mémoriser les additions successives déjà effectuées
- Avoir une variable i pour représenter successivement les entiers 1 à 5 que l'on veut sommer
- Répéter les opérations suivantes 5 fois :
  1. Ajouter la valeur du i courant à s
  2. incrémenter la valeur de i

Instruction pour cela : `while`

Boucle `while`

Syntaxe de la boucle `while`

```
while condition:
    instruction_1
    instruction_2
    ...
    instruction_n
autre_instruction
```

- Avec
  - `condition` : **expression booléenne**, de type `bool`, appelée condition de boucle
  - `instruction_1`, `instruction_2`, ..., `instruction_n` sont des **instructions**, qui forment le corps de la boucle
- Attention : c'est l'indentation qui délimite le corps de la boucle !
  - `autre_instruction` ne fait pas partie du corps de la boucle
- Et ne pas oublier les :

Evaluation de la boucle `while`

```
while condition:
    instruction_1
    instruction_2
    ...
    instruction_n
autre_instruction
```

1. On évalue la valeur de `condition`

Evaluation de la boucle `while`

```
while condition:
    instruction_1
    instruction_2
    ...
    instruction_n
autre_instruction
```

1. On évalue la valeur de `condition`
2. Si `condition` n'a pas la valeur `False`, on interprète le corps de la boucle

   ```
   instruction_1
   instruction_2
   ...
   instruction_n
   ```

   et on revient à l'étape 1.

Evaluation de la boucle `while`

```
while condition:
    instruction_1
    instruction_2
    ...
    instruction_n
autre_instruction
```

1. On évalue la valeur de `condition`
2. Si `condition` n'a pas la valeur `False`, on interprète le corps de la boucle

   ```
   instruction_1
   instruction_2
   ...
   instruction_n
   ```

   et on revient à l'étape 1.
3. Si `condition` a la valeur `False`, on sort de la boucle et on interprète

   ```
   autre_instruction
   ```

Somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

Simulation de boucle

- Il est parfois difficile de "voir" le calcul qui est effectué dans une boucle `while`
- On s'appuie alors sur les simulations de boucle

Simulation de boucle

- Il est parfois difficile de "voir" le calcul qui est effectué dans une boucle `while`
- On s'appuie alors sur les simulations de boucle

Tables de simulation

1. Créer un **tableau** avec :
   1.1 Une colonne par tour de boucle
   1.2 Une colonne par variable modifiée par la boucle
2. Première ligne : entrée et valeurs des variables avant la boucle
3. Lignes suivantes : tours effectués et valeurs des variables à la fin du tour
4. Dernière ligne : (sortie) valeurs des variables au dernier tour

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: True

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: True
tour 1	1	2	$2 \leq 5$: True

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: True
tour 1	1	2	$2 \leq 5$: True
tour 2	3	3	$3 \leq 5$: True

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: True
tour 1	1	2	$2 \leq 5$: True
tour 2	3	3	$3 \leq 5$: True
tour 3	6	4	$4 \leq 5$: True

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: True
tour 1	1	2	$2 \leq 5$: True
tour 2	3	3	$3 \leq 5$: True
tour 3	6	4	$4 \leq 5$: True
tour 4	10	5	$5 \leq 5$: True

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: `True`
tour 1	1	2	$2 \leq 5$: `True`
tour 2	3	3	$3 \leq 5$: `True`
tour 3	6	4	$4 \leq 5$: `True`
tour 4	10	5	$5 \leq 5$: `True`
tour 5	15	6	$6 \leq 5$: `False`

Simulation de boucle : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

tour de boucle	variable s	variable i	condition
entrée	0	1	$1 \leq 5$: `True`
tour 1	1	2	$2 \leq 5$: `True`
tour 2	3	3	$3 \leq 5$: `True`
tour 3	6	4	$4 \leq 5$: `True`
tour 4	10	5	$5 \leq 5$: `True`
tour 5 (sortie)	15	6	$6 \leq 5$: `False`

Trace du programme : utilisation du print

L'instruction `print` peut être utilisée pour tracer des boucles, en affichant la valeur des différentes variables :

```
s = 0
i = 1

print("En entrée, s vaut", s, " et i vaut", i,"\n")

while i <= 5 :
    s = s + i
    i = i + 1
    print("A la fin du tour, s vaut", s, "et i vaut", i)

print("\nSortie de boucle")
print("La valeur finale de s est", s)
```

Trace du programme : utilisation du print

```
En entrée, s vaut 0  et i vaut 1

A la fin du tour, s vaut 1 et i vaut 2
A la fin du tour, s vaut 3 et i vaut 3
A la fin du tour, s vaut 6 et i vaut 4
A la fin du tour, s vaut 10 et i vaut 5
A la fin du tour, s vaut 15 et i vaut 6

sortie de boucle
la valeur finale de s est 15
```

Notion de terminaison

Notion de terminaison

Terminaison

Une boucle `while` termine quand sa condition est fausse.

- Peut-on être sur que la condition sera fausse à un moment donné ?
- Possibilité d'avoir une boucle infinie

Notion de terminaison : erreur classique

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i

print(s)
```

Notion de terminaison : erreur classique

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i

print(s)
```

Erreur classique !

Notion de terminaison : erreur classique

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i

print(s)
```

Erreur classique !

Règle pour un `while`

Il faut obligatoirement qu'une des instructions du corps de la boucle modifie *potentiellement* la valeur de la condition de sortie de la boucle.

Notion de terminaison : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

Notion de terminaison : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

Pourquoi ce script termine ?

Notion de terminaison : somme des 5 premiers entiers

```
s = 0     #variable permettant de calculer la somme
i = 1     #prochain entier à ajouter

while i <= 5 :
    s = s + i
    i = i + 1

print(s)
```

Pourquoi ce script termine ?

- i vaut initialement 1
- i incrémenté à chaque étape, et sa valeur se rapproche un peu de 5 à chaque tour de boucle
- à une étape donnée, i sera strictement supérieur à 5, et on sortira de la boucle

Boucle `while` : quelques exemples

Factorielle

$$\begin{cases} n! & = & 1 * 2 * 3 * \ldots * (n-1) * n \text{ si } n > 0 \\ 0! & = & 1 \end{cases}$$

Factorielle

$$\begin{cases} n! & = & 1 * 2 * 3 * \ldots * (n-1) * n \text{ si } n > 0 \\ 0! & = & 1 \end{cases}$$

Approche itérative :

1. Initialement, $\texttt{fact}(1) = 1$
2. $\texttt{fact}(2) = \texttt{fact}(1) * 2$
3. $\texttt{fact}(3) = \texttt{fact}(2) * 3$
4. $\ldots$

Factorielle

```
fact = 1    #factorielle au rang 0, vaut 1 par convention
n = 8       #valeur de n
i = 2

while i <= n :
    fact = fact * i
    i = i + 1

print("La factorielle de", n,"est", fact)
```

Factorielle

```
fact = 1    #factorielle au rang 0, vaut 1 par convention
n = 5       #valeur de n
i = 2

while i <= n :
    fact = fact * i
    i = i + 1

print("La factorielle de", n,"est", fact)
```

tour de boucle	variable fact	variable i	condition
entrée	1	2	$2 \leq 5$: True
tour 1	2	3	$3 \leq 5$: True
tour 2	6	4	$4 \leq 5$: True
tour 3	24	5	$5 \leq 5$: True
tour 4	120	6	$6 \leq 5$: False

Suite récursive

Une suite récursive est définie par une condition initiale et une fonction de récursion :

$$\begin{cases} u_0 & = & k \\ u_{n+1} & = & f(u_n) \text{ pour } n \in \mathbb{N} \end{cases}$$

Par exemple :

$$\begin{cases} u_0 & = & 2 \\ u_{n+1} & = & 3 * u_n + 4 \text{ pour } n \in \mathbb{N} \end{cases}$$

Suite récursive

Une suite récursive est définie par une condition initiale et une fonction de récursion :

$$\begin{cases} u_0 & = & k \\ u_{n+1} & = & f(u_n) \text{ pour } n \in \mathbb{N} \end{cases}$$

Par exemple :

$$\begin{cases} u_0 & = & 2 \\ u_{n+1} & = & 3 * u_n + 4 \text{ pour } n \in \mathbb{N} \end{cases}$$

```
u = 2 #valeur au rang 0
n = 5 #valeur de la suite à calculer
i = 1

while i <= n :
    u = 3 * u + 4
    i = i + 1

print("La valeur de la suite pour n = ", n,"est", u)
```

Suite récursive : simulation

```
u = 2 #valeur au rang 0
n = 5 #valeur de la suite à calculer
i = 1

while i <= n :
    u = 3 * u + 4
    i = i + 1

print("La valeur de la suite pour n = ", n,"est", u)
```

Suite récursive : simulation

```
u = 2 #valeur au rang 0
n = 5 #valeur de la suite à calculer
i = 1

while i <= n :
    u = 3 * u + 4
    i = i + 1

print("La valeur de la suite pour n = ", n,"est", u)
```

tour de boucle	variable u	variable i	condition
entrée	2	1	$1 \leq 5$: True
tour 1	10	2	$2 \leq 5$: True
tour 2	34	3	$3 \leq 5$: True
tour 3	106	4	$4 \leq 5$: True
tour 4	322	5	$5 \leq 5$: True
tour 5	970	6	$6 \leq 5$: False

PGCD

Les exemples donnés jusqu'à présent sont simples :

- La solution algorithmique est donnée dans l'énoncé du problème
- On connait à l'avance le nombre de passages dans la boucle `while`

Souvent, ce n'est pas le cas.

PGCD

Les exemples donnés jusqu'à présent sont simples :

- La solution algorithmique est donnée dans l'énoncé du problème
- On connait à l'avance le nombre de passages dans la boucle `while`

Souvent, ce n'est pas le cas.
Par exemple, pour trouver le PGCD de deux entiers :

PGCD

Calculer le **plus grand commun diviseur** de deux entiers positifs.

PGCD

Les exemples donnés jusqu'à présent sont simples :

- La solution algorithmique est donnée dans l'énoncé du problème
- On connait à l'avance le nombre de passages dans la boucle `while`

Souvent, ce n'est pas le cas.
Par exemple, pour trouver le PGCD de deux entiers :

PGCD

Calculer le **plus grand commun diviseur** de deux entiers positifs.

Il faut d'abord trouver un algorithme pour résoudre le problème.

Algorithme d'Euclide

Algorithme d'Euclide

On veut calculer le PGCD de deux entiers a et b, tels que $a \geq b$.

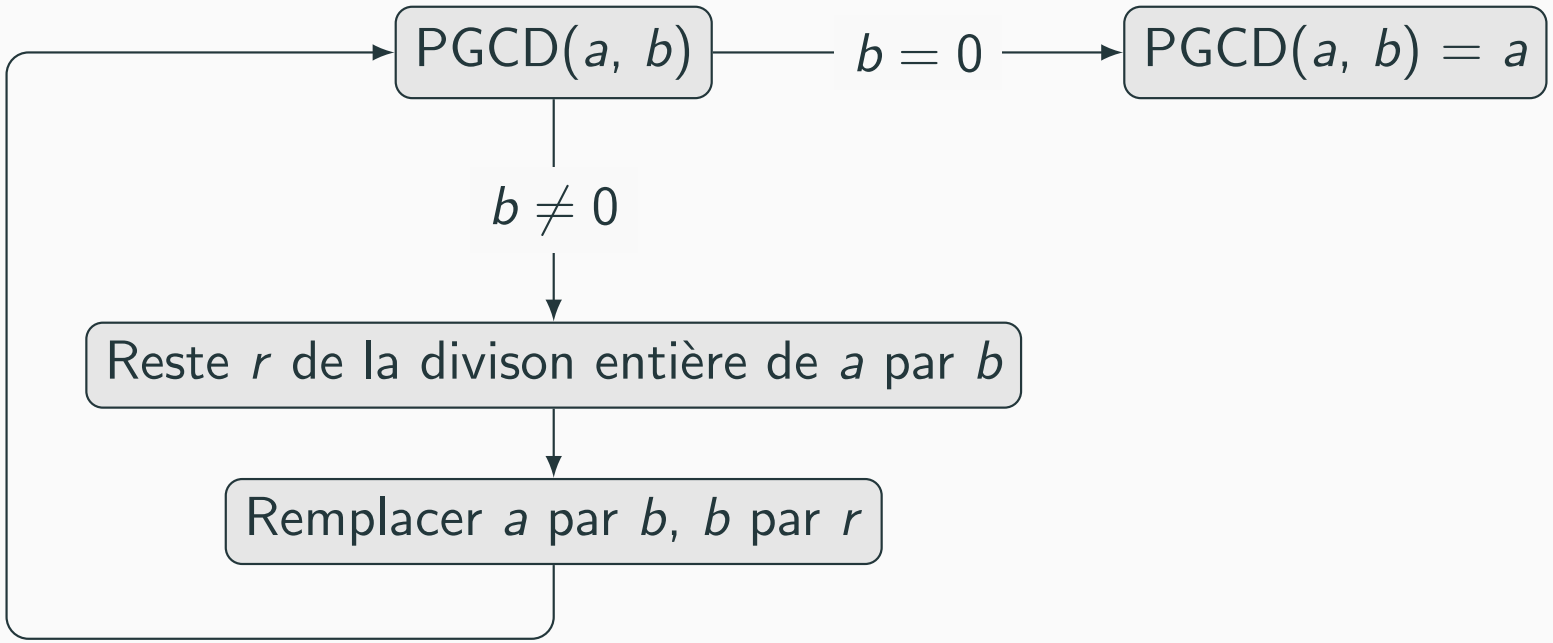
1. Si $b \neq 0$, le PGCD de a et b est le PGCD de b et du reste de la division euclidienne de a par b
2. si $b = 0$, le PGCD de a et b est a

Algorithme d'Euclide

Algorithme d'Euclide

On veut calculer le PGCD de deux entiers a et b, tels que $a \geq b$.

1. Si $b \neq 0$, le PGCD de a et b est le PGCD de b et du reste de la division euclidienne de a par b
2. si $b = 0$, le PGCD de a et b est a



Algorithme d'Euclide : exemple

Calcul du PGCD de $a = 147$ et $b = 105$

- $b \neq 0$. $147 = 105 * 1 + 42 \ \rightarrow r = 42$. On remplace : a par $b = 105$ et b par $r = 42$

Algorithme d'Euclide : exemple

Calcul du PGCD de $a = 147$ et $b = 105$

- $b \neq 0$. $147 = 105 * 1 + 42 \ \rightarrow r = 42$. On remplace : a par $b = 105$ et b par $r = 42$
- $b \neq 0$. $105 = 42 * 2 + 21 \ \rightarrow r = 21$. On remplace : a par $b = 42$ et b par $r = 21$

Algorithme d'Euclide : exemple

Calcul du PGCD de $a = 147$ et $b = 105$

- $b \neq 0$. $147 = 105 * 1 + 42 \ \rightarrow r = 42$. On remplace : a par $b = 105$ et b par $r = 42$
- $b \neq 0$. $105 = 42 * 2 + 21 \ \rightarrow r = 21$. On remplace : a par $b = 42$ et b par $r = 21$
- $b \neq 0$. $42 = 21 * 2 + 0 \ \rightarrow r = 0$. On remplace : a par $b = 21$ et b par $r = 0$

Algorithme d'Euclide : exemple

Calcul du PGCD de $a = 147$ et $b = 105$

- $b \neq 0$. $147 = 105 * 1 + 42 \rightarrow r = 42$. On remplace : a par $b = 105$ et b par $r = 42$
- $b \neq 0$. $105 = 42 * 2 + 21 \rightarrow r = 21$. On remplace : a par $b = 42$ et b par $r = 21$
- $b \neq 0$. $42 = 21 * 2 + 0 \rightarrow r = 0$. On remplace : a par $b = 21$ et b par $r = 0$
- $b = 0$. Le PGCD de 21 et 0 est 21

Algorithme d'Euclide : exemple

Calcul du PGCD de $a = 147$ et $b = 105$

- $b \neq 0$. $147 = 105 * 1 + 42 \ \rightarrow r = 42$. On remplace : a par $b = 105$ et b par $r = 42$
- $b \neq 0$. $105 = 42 * 2 + 21 \ \rightarrow r = 21$. On remplace : a par $b = 42$ et b par $r = 21$
- $b \neq 0$. $42 = 21 * 2 + 0 \ \rightarrow r = 0$. On remplace : a par $b = 21$ et b par $r = 0$
- $b = 0$. Le PGCD de 21 et 0 est 21

Le PGCD de 147 et 105 est 21.

Algorithme d'Euclide : Implémentation

```
# Programme qui calcule le PGCD de deux entiers

a = 147
b = 105

if b > a :  #si b > a, on inverse les valeurs de a et de b
    tmp = a
    a = b
    b = tmp

a_init = a #pour garder la valeur de a pour l'affichage final
b_init = b #pour garder la valeur de b pour l'affichage final

while b != 0 :
    r = a % b #reste de la division euclidienne
    a = b
    b = r

print("Le pgcd de", a_init, "et", b_init, "est", a)
```

Algorithme d'Euclide : simulation

tour de boucle	r	a	b	condition
entrée	0	147	105	$105 \neq 0$: `True`
tour 1	42	105	42	$42 \neq 0$: `True`
tour 2	21	42	21	$21 \neq 0$: `True`
tour 3	0	21	0	$0 \neq 0$: `False`

Table de multiplications

Afficher une table de multiplication sous ce format :

```
Table de multiplication de 8 :
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

Table de multiplications

Afficher une table de multiplication sous ce format :

```
Table de multiplication de 8 :
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
```

```
n = 8
print("Table de multiplication de", n, ":")
i = 1
while i <= 9:
    m = n * i
    print(n, "*", i, "=", m)
    i = i + 1
```

Tables de multiplications

- On souhaite à présent afficher **toutes** les tables de multiplications entre 1 et un entier donné n
- Impossible de résoudre ce problème avec une boucle
- Utilisation de boucles imbriquées

Tables de multiplications

- On souhaite à présent afficher **toutes** les tables de multiplications entre 1 et un entier donné n
- Impossible de résoudre ce problème avec une boucle
- Utilisation de boucles imbriquées

```
# Programme qui affiche toutes les tables de multiplcations
# jusqu'à un entier entré au clavier
n = int(input("Jusqu'à qu'elle table souhaitez-vous aller ? "))
i = 1

while i <= n :
    print("Table de multiplication de", i, ":")
    j = 1
    while j <= 9 :
        print(i, "*", j, "=", i * j)
        j = j + 1
    i = i + 1
```

Tables de multiplications

```
Jusqu'à qu'elle table souhaitez-vous aller ? 2
Table de multiplication de 1 :
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
Table de multiplication de 2 :
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
```

Boucle `for`

Boucles `while` et boucles `for`

- Une boucle `while` se répète tant que la condition est vraie
- Lorsque l'on connait l'ensemble des valeurs à considérer, il peut être pratique de parcourir une séquence de valeurs
- C'est ce que fait une boucle `for`
  - Accède tour à tour à chaque valeur d'une séquence afin de la traiter dans le corps de la boucle

Syntaxe de la boucle `for`

```
for var in liste_de_valeurs :
    instruction_1
    instruction_2
    ...
    instruction_n
autre_instruction
```

- Avec
  - La variable `var` prend toutes les valeurs contenues dans `liste_de_valeurs`
  - `instruction_1`, `instruction_2`, ..., `instruction_n` sont des **instructions**, qui forment le corps de la boucle
- Attention : c'est l'indentation qui délimite le corps de la boucle !
  - `autre_instruction` ne fait pas partie du corps de la boucle
- Et ne pas oublier les `:`

Séquence

```
for var in liste_de_valeurs :
    instruction_1
    instruction_2
    ...
    instruction_n
autre_instruction
```

- `liste_de_valeurs` est un objet de type séquence
- Une séquence contient plusieurs éléments, stockés de façon ordonnée
- La variable `var` prend les valeurs de `liste_de_valeurs` dans l'ordre dans lequel elles apparaissent dans la séquence

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n - 1$
- `range`(i, j) : génère les entiers **de** i **à** $j - 1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j - 1$**, séparés par un pas de** k

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) :

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n - 1$
- `range`(i, j) : génère les entiers **de** i **à** $j - 1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j - 1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) :

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9
- `range`(10, 3) :

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9
- `range`(10, 3) : intervalle vide

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9
- `range`(10, 3) : intervalle vide
- `range`(3, 10, 2) :

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9
- `range`(10, 3) : intervalle vide
- `range`(3, 10, 2) : 3, 5, 7, 9

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9
- `range`(10, 3) : intervalle vide
- `range`(3, 10, 2) : 3, 5, 7, 9
- `range`(10, 3, -2) :

Intervalle d'entiers : `range`

Intervalle d'entiers : `range`

`range` construit un intervalle d'entiers.

Plusieurs utilisations sont possibles :

- `range`(n) : génère les entiers **de 0 à** $n-1$
- `range`(i, j) : génère les entiers **de** i **à** $j-1$ (si $i > j$, aucun nombre ne sera généré)
- `range`(i, j, k) : génère les entiers **de** i **à** $j-1$**, séparés par un pas de** k

- `range`(8) : 0, 1, 2, 3, 4, 5, 6, 7
- `range`(3, 10) : 3, 4, 5, 6, 7, 8, 9
- `range`(10, 3) : intervalle vide
- `range`(3, 10, 2) : 3, 5, 7, 9
- `range`(10, 3, -2) : 10, 8, 6, 4

Boucle `for` : quelques exemples

Somme des n premiers entiers

```
# Programme qui calcule la somme des n premiers entiers

s = 0
n = 5

#Attention à bien paramétrer la fonction range avec n+1!
for i in range(1, n+1) :
    s = s + i

print("La valeur finale de s est", s)
```

Parcours d'un mot

```
# Programme qui parcourt un mot

mot = "bonjour"

for lettre in mot :
    print("*", lettre, "*")
```

Parcours d'un mot

```
# Programme qui parcourt un mot

mot = "bonjour"

for lettre in mot :
    print("*", lettre, "*")
```

```
* b *
* o *
* n *
* j *
* o *
* u *
* r *
```

Parcours d'une chaine de caractère

```
# Programme qui compte le nombre de voyelles
# d'une chaine de caractère

chaine = "Python est un langage fort sympa"

nb_voyelles = 0

for lettre in chaine :
    if lettre in "aeiouy" :
        nb_voyelles = nb_voyelles + 1

print("'", chaine, "' contient", nb_voyelles, "voyelles")
```

Parcours d'une chaine de caractère

```
# Programme qui compte le nombre de voyelles
# d'une chaine de caractère

chaine = "Python est un langage fort sympa"

nb_voyelles = 0

for lettre in chaine :
    if lettre in "aeiouy" :
        nb_voyelles = nb_voyelles + 1

print("'", chaine, "' contient", nb_voyelles, "voyelles")
```

```
' Python est un langage fort sympa ' contient 10 voyelles
```

Boucles imbriquées : affichage des multiples d'une série de nombres

On souhaite écrire un programme qui donne l'affichage suivant :

```
Quel est le nombre maximum ? 8
1 est un nombre premier
2 est un nombre premier
3 est un nombre premier
4 est égal à 2 * 2
5 est un nombre premier
6 est égal à 2 * 3
6 est égal à 3 * 2
7 est un nombre premier
8 est égal à 2 * 4
8 est égal à 4 * 2
```

Boucles imbriquées : affichage des multiples d'une série de nombres

On souhaite écrire un programme qui donne l'affichage suivant :

```
Quel est le nombre maximum ? 8
1 est un nombre premier
2 est un nombre premier
3 est un nombre premier
4 est égal à 2 * 2
5 est un nombre premier
6 est égal à 2 * 3
6 est égal à 3 * 2
7 est un nombre premier
8 est égal à 2 * 4
8 est égal à 4 * 2
```

Nécessité de deux boucles imbriquées :

1. Parcourir tous les entiers i de 1 à n
2. Pour chaque i, parcourir tous les entiers j de 2 à $i - 1$ pour vérifier si c'est un multiple de i

Boucles imbriquées : affichage des multiples d'une série de nombres

```
n = int(input("Quel est le nombre maximum ? "))

premier = True #pour savoir si le nombre est premier

for i in range(1, n+1) :
    for j in range(2, i) :
        if i % j == 0:
            print(i, "est égal à", j, "*", i//j)
            premier = False
    if premier :
        print(i, "est un nombre premier")
    premier = True
```

Pour conclure

Résumé du cours

Aujourd'hui, on a vu

- Les boucles `while` et `for`
- Quelques notions sur la terminaison des boucles
- La fonction `range`