

Algorithmique

Chapitre 1: Rappels sur les fondamentaux

Dr. N'golo KONATE

konatengolo@ufhb.edu.ci

Rappels sur les fondamentaux

1. Identificateurs
2. Conditions
3. Boucles
4. Types de données

Identificateurs

Variables

Variables

Une variable est une zone de la mémoire dans laquelle une valeur est stockée.

Une variable possède 4 propriétés:

- Un nom
- Une adresse
- Un type
- Une valeur

Variables

Variables

Une variable est une zone de la mémoire dans laquelle une valeur est stockée.


Une variable possède 4 propriétés:

- Un nom
 - Une adresse
 - Un type
 - Une valeur
-
- Pour le programmeur, une variable est définie par son **nom**
 - Pour l'ordinateur, c'est une adresse

Affectation d'une valeur à une variable

Affectation

L'affectation est l'opération qui consiste à attribuer une valeur à une variable.

Elle est réalisé grâce a l'opérateur  en algo et = en python

- Pour le programmeur, une variable est définie par son **nom**
- Pour l'ordinateur, c'est une adresse

Affectation et égalité

Affectation : affecter n'est pas comparer

Attention de ne pas confondre **affectation** et **égalité mathématique**

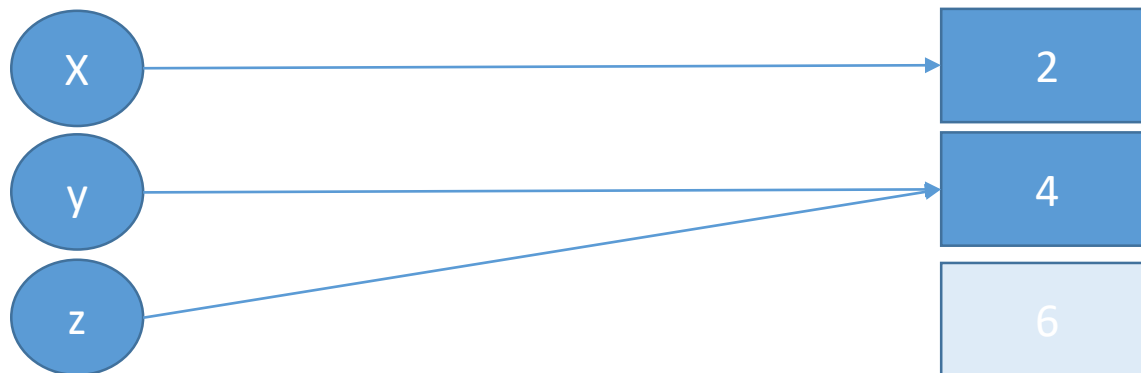
- **L'affectation(=)** a un effet (associe une valeur à une variable), mais n'a pas de valeur
- **La comparaison(==)** a une valeur (**True** si la comparaison est vraie, **False** sinon) mais n'a pas d'effet

Ré-affecter une valeur à une variable

Ré-affectation

La valeur d'une valeur peut évoluer au cours du temps par des opérations **réaffectation**.

```
>>> x=2  
>>> y=4  
>>> z=y  
>>>
```



Séquence temporelle des affectations

Séquence temporelle des affectations

Le membre droit d'une affectation est évalué avant de réaliser l'affectation.

La variable affectée peut donc se trouver en partie droite de l'affectation.

```
>>> x = 2
>>> x = x + 1      #incrémentatation : on calcule x+1,
>>> x              #puis on affecte cette valeur à x
3
>>> x = x - 1      #décrémentatation : on calcule x-1,
>>> x              #puis on affecte cette valeur à x
2
>>> u_n = 4
>>> u_n = 3 * u_n + 1  #définition d'une suite numérique
>>> u_n
13
```

Affectation parallèles et évaluation d'expressions

Affectation parallèles et évaluation d'expressions

Dans une affectation, la valeur du membre de droite est évaluée avant de l'affecter au membre de gauche, ce qui est important pour les affectations parallèles.

```
>>> a = 3
>>> b, a = a + 2, a * 2
#toutes les expressions sont évaluées avant la 1ère affectati
>>> a
6
>>> b
5
```

Types et opérations

Activités 1:

- ✓ Rappelez les différents types de données en algorithmique
 - Types de données simple
 - Types de données complexes

Types et opérations

Activités 1:

- ✓ Rappelez les différents types de données en algorithme
 - Types de données simple
 - Types de données complexes

Activités 2:

- ✓ Donner les opérations possibles sur les types de données

Conditions

Séquences d'instructions

Séquences d'instructions

Les instructions d'une **séquences d'instructions** sont exécutés **en séquences**, les unes après les autres, **dans l'ordre dans lesquelles elles apparaissent**

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
>>> print(a, b)
7 7
```

Séquences d'instructions

Séquences d'instructions

Les instructions d'une **séquences d'instructions** sont exécutés **en séquences**, les unes après les autres, **dans l'ordre dans lesquelles elles apparaissent**

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
>>> print(a, b)
7 7
```

```
>>> a, b = 3, 7
>>> b = a
>>> a = b
>>> print(a, b)
3 3
```

Séquences d'instructions

- Les séquences d'instructions sont indispensables, mais pas suffisantes
- Nécessité d'aiguiller le déroulement des programmes dans différentes directions, en fonction des circonstances rencontrées
- Ou encore de répéter plusieurs fois des instructions données (mais ce sera vu à la prochaine section)

————→ Instructions composés

Instructions conditionnelles

Instructions conditionnelles

Une instruction conditionnelle, ou alternative, est une instruction qui permet de choisir entre deux séquences d'instruction selon la valeur d'une condition.

On veut trouver la valeur absolue d'un nombre

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{sinon} \end{cases}$$

- On fait le choix entre deux calculs (x ou $-x$) selon une condition ($x \geq 0$)
- Il faut donc savoir **évaluer une condition**

Instructions conditionnelles

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{sinon} \end{cases}$$

- 3 éléments à distinguer:
 - ✓ l'expression qui permet de choisir : $x \geq 0$
 - ✓ l'instruction à appliquer si le choix est vrai : x
 - ✓ l'instruction à appliquer si le choix est faux : $-x$
- On nomme
 - ✓ la **condition** : l'expression qui permet de choisir
 - ✓ le **conséquent** : l'instruction à appliquer si le choix est vrai
 - ✓ l'**alternant** : l'instruction à appliquer si le choix est faux

Syntaxes des instructions conditionnelles

```
if condition:  
    consequent  
else:  
    alternant
```

- Avec
 - ✓ la condition : expression booléenne, de type **bool**
 - ✓ le conséquent : instruction ou suite d'instructions
 - ✓ l'alternant : instruction ou suite d'instructions
- Remarques : Le bloc **else** peut ne pas être présent
- Attention ! la condition est une expression booléenne !
 - ✓ On n'écrit jamais : condition == **True**
 - ✓ On n'écrit jamais : condition == **False**

Activités

1. Ecrire un programme permettant d'afficher la valeur absolue d'un entier entrée au clavier
2. Ecrire un programme permettant de déterminer si un entier entré au clavier est pair ou impair
3. Ecrire un programme permettant d'afficher le Ph d'une solution

Boucles

Pourquoi les instructions répétitives

Objectif des intructions répétitives

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

Pourquoi les instructions répétitives

Objectif des instructions répétitives

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

Les boucles se mettent en œuvre sous deux formes:

Pourquoi les instructions répétitives

Objectif des instructions répétitives

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

Les boucles se mettent en œuvre sous deux formes:

- La boucle **tant... que:** qui s'exécute tant qu'une condition est respectée

Pourquoi les instructions répétitives

Objectif des intructions répétitives

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

Les boucles se mettent en œuvre sous deux formes:

- La boucle **tant... que:** qui s'exécute tant qu'une condition est respectée
- La boucle **pour:** qui s'exécute quand on connaît le nombre d'itération

Cas pratique: Somme des n-premiers nombres

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

Ecrire un algorithme permettant de calculer la somme des n premiers nombre

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
```

Boucle while

- Il est parfois difficile de "voir" le calcul qui est effectué dans une boucle **tant... que**
- 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 `a 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
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

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

Notion de terminaison

Terminaison d'une boucle

Une boucle while termine quand sa condition est fausse.

- Peut-on être sûr que la condition sera fausse à un moment donné ?
- Possibilité d'avoir une **boucle infinie**
- Comment matérialiser cette fin

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

Activités

1. Ecrire un programme qui permet de calculer la suite suivante

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

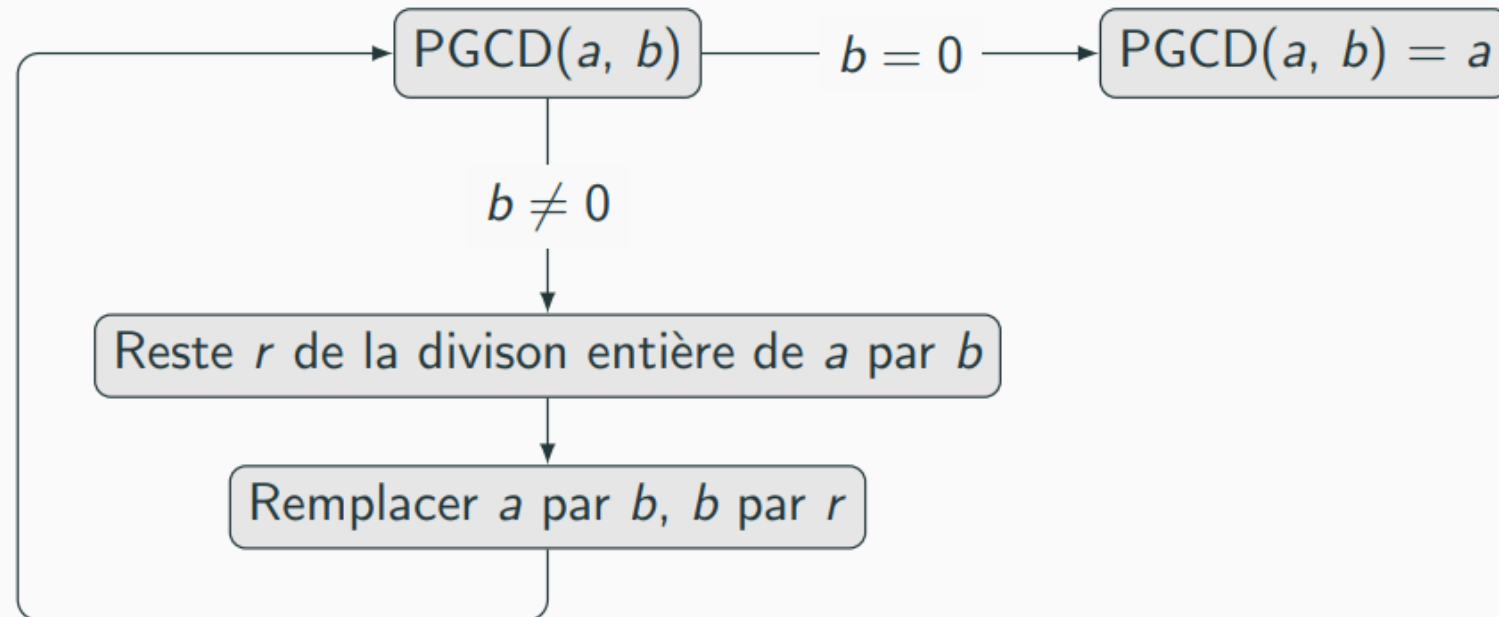
2. Ecrire un programme permettant d'obtenir le PGCD de deux nombres

Activités: PGCD

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



Activités: PGCD

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.

Boucle while & boucle for

- Une boucle **while** se répète **tant que** la condition est vraie
- Lorsque l'on connaît 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

Boucle while & 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 :

Boucle while & boucle for

```
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, stockées de façon **ordonnée**
- La variable var prend les valeurs de liste_de_valeurs dans **l'ordre dans lequel elles apparaissent** dans la séquence

Activité

1. Ecrire un programme qui parcourt un mot
2. Ecrire un programme qui compte le nombre de voyelles
3. 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
```

Types de données

Chaine de caractères

Chaine de caractères

Une chaine de caractères est une séquence de caractères. Son type en python est str.

Longueur d'une chaine de caractères

L'opérateur **len** retourne la **longueur** d'une chaine de caractères, c'est à dire le nombre de caractères qui la compose.

Signature :

len :: **str** → **int**

Chaine de caractères

Opérateurs

Concaténation de chaines de caractères

L'opérateur **+** permet de **concaténer** les chaines de caractères.

Signature :

$$+ :: \text{str} \times \text{str} \rightarrow \text{str}$$

Duplication de chaines de caractères

L'opérateur ***** permet de **dupliquer** les chaines de caractères.

Signature :

$$* :: \text{str} \times \text{int} \rightarrow \text{str}$$

Les listes

Listes

Une **liste**, de type **list** est une collection **ordonnée** et **modifiable** d'éléments éventuellement hétérogènes.

Une liste est formée d'éléments séparés par des virgules, et entourés de crochets.

La **liste vide**, notée **[]**, est une liste qui ne contient aucun élément

Longueur d'une liste

Comme pour les chaînes de caractères, l'opérateur **len** retourne la longueur d'une liste, c'est à dire le nombre d'éléments qui la compose.

Signature : **len** :: **list** → **int**

Les listes

Opérateurs

Appartenance d'un élément à une liste

L'opérateur **in** permet de déterminer si un élément appartient à une liste.

Signature :

$$\text{in} :: \text{elem} \times \text{list} \rightarrow \text{bool}$$

Concaténation de listes

L'opérateur **+** permet de **concaténer** les listes.

Signature :

$$+ :: \text{list} \times \text{list} \rightarrow \text{list}$$

Les ensembles

Ensembles

Un **ensemble**, de type **set** est une collection **non ordonnée** d'éléments **uniques**.

Un ensemble est formé d'éléments séparés par des virgules, et entourés d'accolades.

L'**ensemble vide**, noté **set()**, est un ensemble qui ne contient aucun élément.

Les ensembles

Opérations

Soit E et F deux ensembles, et x un élément quelconque

Notation Python	Notation mathématique
<code>len(E)</code>	$ E $: le cardinal de E
<code>set()</code>	\emptyset : l'ensemble vide
<code>x in E</code>	$x \in E$: l'appartenance
<code>x not in E</code>	$x \notin E$: la non-appartenance
<code>E < F</code>	$E \subset F$: l'inclusion stricte
<code>E <= F</code>	$E \subseteq F$: l'inclusion large
<code>E & F</code>	$E \cap F$: l'intersection
<code>E F</code>	$E \cup F$: l'union
<code>E - F</code>	$E \setminus F$: la différence