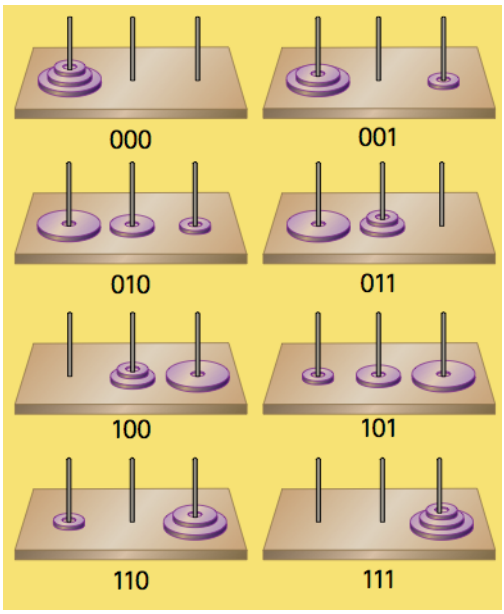
Chapitre 2 : Algorithmes, fonctions et variables

I Introduction

Les algorithme existent bien avant les ordinateurs !

- Règles de calcul (multiplication ...)
- Division euclidienne

Définition : [naive] Un **algorithme** est une méthode systématique de résolution d'un problème donné.

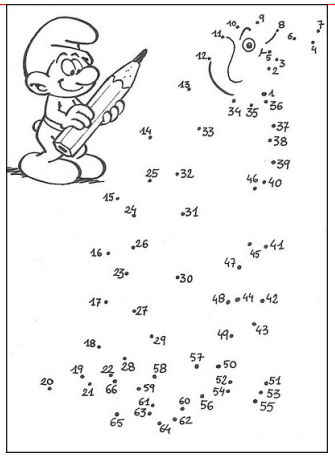


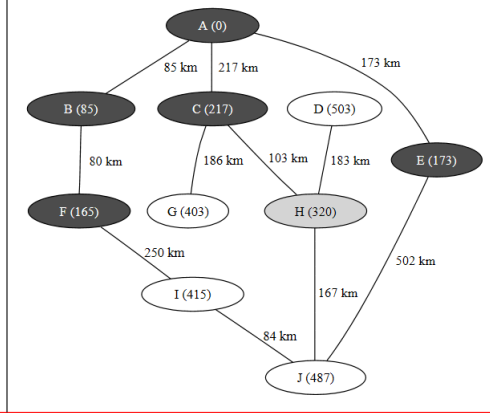
Décider si une liste L de nombres est triée (par ordre croissant).

$$L = [1, 5, 6, 8, 25, 36, 99]$$
$$L = [1, 5, 6, \ldots, \ldots, 291, 1029, 1033, 1056, \ldots, \ldots, 9999, 102935]$$

II Les algorithmes de nos jours

II.1 D'autres exemples





II.2 Instructions

II.2.1 Instruction IF ...THEN ...ELSE...

SI $a \geq 0$ ALORS
{
    ECRIRE "a est positif ou nul"
}
SINON
{
    ECRIRE "a est negatif"
}
FIN SI

```
def signe(a):
  if a >= 0:
    return "a est positif ou nul"
  else:
    return "a est negatif"
```

Remarque : Sur la condition SI ... ALORS :

- ne s'exécute qu'une seule fois, ne se répète pas

Sur le langage Python :

- pas besoin d'accolade
- pas de FIN SI

II.2.2 Instruction WHILE ...DO

$a = 10$
TANT QUE $a \geq 0$ FAIRE
{
    ECRIRE a
    $a \leftarrow a - 1$
}

```
a = 10
while a >= 0:
  print(a)
  a = a-1
```

Remarque : Sur la condition TANT QUE ... FAIRE :

- on ne sait pas combien de fois on boucle
- on ne sait pas si cela s'arrête bien !

Sur le langage Python :

- le FAIRE est implicite
- pas besoin d'accolade

II.2.3 Instruction FOR ...DO

POUR i allant de 1 à 10 FAIRE
{
    ECRIRE i
}

```
for i in range(1, 11):
  print(i)
```

Remarque : Sur la boucle POUR ... FAIRE :

- on sait à priori combien de fois on boucle
- on sait qu'elle se termine !
- on aurait pu utiliser While, mais on préfère FOR, c'est plus explicite

Exercice 1
Réécrire l'algorithme avec la boucle TANT QUE en utilisant la boucle FOR.

Donner l'affichage des exécutions suivantes :

```
for i in range(0, 11):
  print(i)
```

```
for i in range(3, 10):
  print(i)
```

```
for i in range(8):
  print(i)
```

II.3 Définition

- Un algorithme n'est pas un langage de programmation.
- Un même algorithme n'a pas d'écriture propre et s'interprète par l'ordinateur dès lors qu'il est écrit dans un langage de programmation.

Définition : [plus formelle] Un **algorithme** est une suite finie d'instructions permettant à partir de données connues, d'obtenir un résultat. Ces instructions sont sans ambiguité (une seule manière de les exécuter), sont exécutables et se terminent.

✌ Intérêt principal ✌ automatiser des méthodes systématiques. Un algorithme est bon :

- si la suite d'instructions est minimale (rapidité, complexité de l'algorithme ...)
- s'il donne le bon résultat pour toutes les entrées admissibles et non pas seulement sur un exemple précis
- s'il est agrémenté de commentaires

III Fonctions

Lorsqu'un algorithme dépend de plusieurs variables, on peut l'écrire sous forme de fonctions. La syntaxe est la suivante :

```
def nom_de_la_fonction(argument 1, argument 2, ... argument n) :
    # instuction 1
    # instruction 2

    # instruction m
    return ...
```

On essayera d'utiliser des fonctions le plus souvent possible.

IV Exercices

Exercice 2
On considère trois nombres stockés dans les variables $N1, N2, N3$. Ecrire une fonction qui prend en argument 3 nombres et renvoie le maximum de ces trois nombres.

Hint ☞ *On n'utilisera que les opérateurs SI ... ALORS ... SINON et PRINT.*

Exercice 3
On considère l'algorithme suivant :

```
N = 100
S = 0
for i in range(1, N+1):
  S = S+i
print(S)
```

1. Détailler ce que renvoie l'algorithme.
2. Que faut-il changer si l'on souhaite calculer la somme des nombres entre 50 et 200 (compris) ?
3. Ecrire ce programme sous forme d'une fonction qui prend en argument N et renvoie $\sum_{k=0}^{N} k$.
4. Ecrire un algorithme qui calcule $\sum_{i=1}^{100} 2^i$.

Exercice 4
Critiquer l'algorithme suivant :

resultat $\leftarrow 0$
POUR k allant de 1 à N FAIRE
{
resultat $\leftarrow$ **resultat** $* k$
AFFICHER "Le résultat est + **resultat**"
}
	`resultat = 0`
	`for k=1:N+1`
	`  resultat = resultat*k`
	`  print "Le r\'esultat est, resultat"`

Exercice 5
On considère l'algorithme suivant :

```
s = 2
while s <= 100:
  s = s*s
print(s)
```

1. Qu'affiche cet algorithme ?
2. Que cela aurait-il changé si on avait initialisé s à 1 ?

V Variables

V.1 Généralités

V.1.1 Déclaration d'une variable

Un nom de variable doit répondre aux exigences suivantes :

1. pas de caractère spéciaux sauf _
2. pas d'accent
3. pas d'espace
4. ne commence pas par un chiffre
5. représentatif de son usage

✌ Attention ✌ Python est sensible à la casse :

✗ monnom **✓ mon_nom** **✓ monNom** **✗ monPrénom** **✗ mon-prenom**

Quelques mots clefs réservés par Python (à ne pas utiliser comme nom de variables) : **and**, **as**, **assert**, **break**, **class**, **continue**, **def**, **elif**, **else**, **except**, **false**, **for**, **from**, **global**, **if**, **import**, **in**, **is**, **lambda**, **none**, **not**, **or**, **return**, **true**, **while**, **with**.

✌ En Python ✌ Pour déclarer une variable, il suffit de lui affecter une valeur.

V.1.2 Affectation d'une valeur à une variable

En algorithmique	En Python
variable ← valeur	**variable** = valeur

Tout ce qui se situe à droite du signe = est exécuté avant d'être affecté à la variable. Ex : $a = 2 + 1$

V.2 Différents types

- Entiers (*int*)
- Flottants (*float*)
- Complexes (*complex*)
- Chaine de caractères (*str*)
- Booléen : deux valeurs possible VRAI ou FAUX (1 ou 0) (*bool*)

VI Quelques algorithmes fondamentaux

Calculer une somme

$$\sum_{k=0}^{n} \sqrt{k}$$

```
import math
def somme(n):
  S = 0 # la somme au depart vaut toujours 0
  for k in range(n+1):
    S = S + math.sqrt(k)
  return S
```

Echanger la valeur de deux variables a et b

```
c=a # on cree une troisieme variable
a=b # on affecte Ã  a la valeur de b
b=c # b prend la valeur initiale de a
```

Compter sous condition

Par exemple les nombres pairs entre 0 et N

```
def compter_pair(N):
  c = 0 # variable compteur au d\'epart 0
  for i in range(N+1):
    if i % 2 == 0:
      c = c + 1
  return c
```

Trouver la valeur n à partir de laquelle

$$\sum_{i=0}^{n} 10^i > 1000$$

```
n = 0 # variable qui compte le nombre de tours
s = 0 # valeur de la somme au d\'ebut
while s <= 1000:
  s = s + 10 ** n
  n = n + 1
print("Nombre de tours :", n)
```