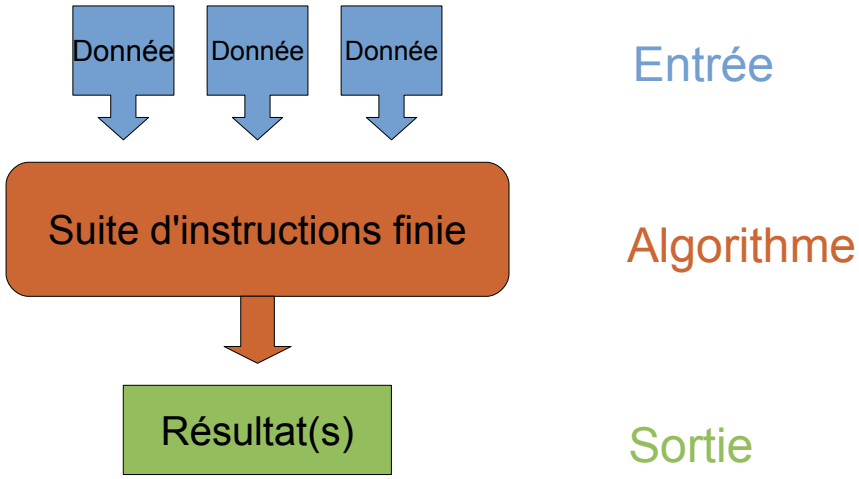
Algorithmique

Informatique

L'informatique est le domaine concernant le traitement automatique de l'information. Ce traitement est réalisé par des algorithmes.

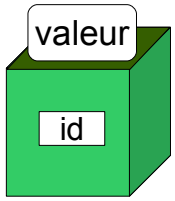
Algorithme

On pourrait définir un algorithme comme une suite finie d'instructions permettant de résoudre un problème ou, qui permet à partir de données de départ, d'obtenir un résultat recherché.



Variables

Les données (en entrée ou créées par l'algorithme) sont stockées dans des **variables** (qui correspondent à des emplacements de la mémoire). Une variable peut être vue comme une boîte pouvant contenir des objets (des valeurs). Les variables ont un nom (un identificateur) qui permet de les manipuler.
Faire une **affectation** consiste à mettre une valeur dans une variable.



L'affectation est schématisée par ← ou → ou = suivant les machines.
Ainsi :

a ← 13 ou 13 → a ou a = 13

signifient :

« la variable portant l'identifiant « a » prend la valeur 13 »

Différentes écritures d'un algorithme

Un algorithme informatique peut être décrit par du pseudo-code qui peut ensuite être traduit dans un langage informatique (exemple : Python, Java, etc.), ce qui donne un programme exécutable par une machine.

Voici un exemple de traduction d'un algorithme du langage naturel vers le pseudo-code puis en programme Python :

Langage naturel	**Pseudo-code**
Un camarade me donne trois de ses notes : 12, 18, 8. Je calcule leur moyenne en les ajoutant puis en divisant le résultat par 3. Je regarde s'il a la moyenne et je lui adresse un message en conséquence.	note1 ← 12 note2 ← 18 note3 ← 08 moy ← (note1 + note2 + note3) / 3 **Si** moy ≥ 10 **Alors Afficher** "Vous avez la moyenne" **Sinon Afficher** "Peut mieux faire" **FinSi**

Langage Python

```
note1 = 12
note2 = 18
note3 = 8
moy = (note1 + note2 + note3) / 3
if moy >= 10:
        print("Vous avez la
moyenne")
else:
        print("Peut mieux faire")
```

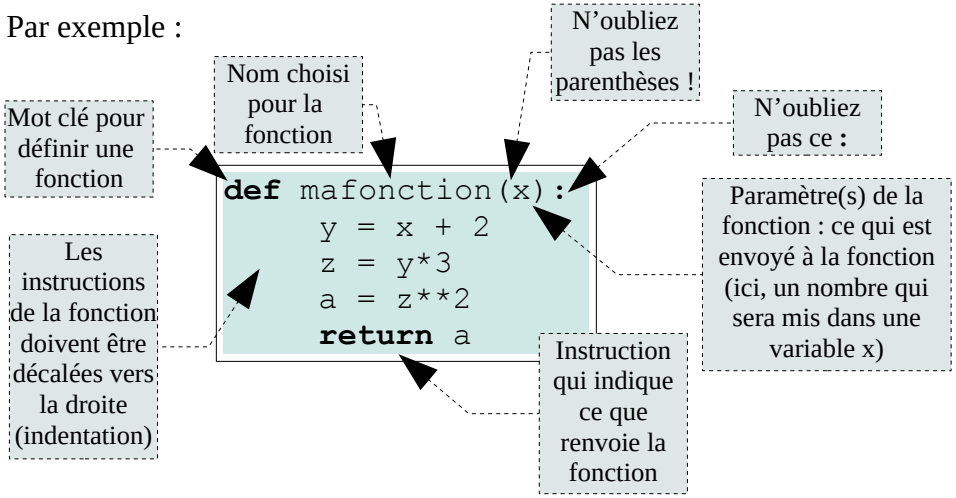
Les fonctions en Python (sous-programmes)

Les fonctions permettent de découper un gros programme en sous-programmes plus faciles à comprendre et à modifier. Les fonctions permettent aussi d'éviter de répéter plusieurs fois le même code dans un programme. Comme les fonctions sont aussi des programmes, elles prennent des données en entrée et renvoient d'autres données en sortie :

Pour définir une fonction, utilisez le mot clé **def** suivi du nom de la fonction puis de parenthèses contenant le nom des variables qui vont recevoir les données en entrée, terminez par :.

Les valeurs renvoyées par la fonction seront après la commande **return**.

Par exemple :



Attention : le programme ci-dessus sert à **définir une fonction** : celle-ci n'est pas exécutée mais stockée pour servir plus tard dans un programme principal.

Pour **l'exécuter** (pour **l'« appeler »**), il suffit de taper son nom suivi de parenthèses contenant des valeurs.

Par exemple ici, pour demander l'exécution de la fonction « `mafonction` » avec comme valeur 5 pour x :

```
>>> mafonction(5)
```

dans la console Python ou

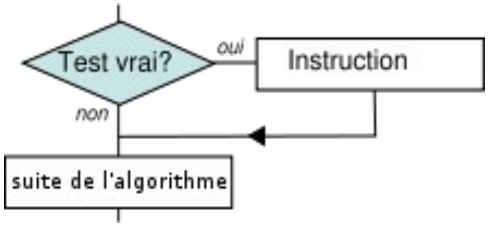
```
print(mafonction(5))
```

dans le programme.

Les tests (structures conditionnelles)

Il est parfois nécessaire qu'un algorithme ait besoin de s'adapter à différentes situations, pour cela il y a les tests. Il y en a deux sortes :

Forme 1 : **si** *condition vérifiée* **alors** *instructions à effectuer*



La condition est une comparaison, telles que, par exemple :

temperature = 0 ou hauteur < 3 ou $x \geq -1$

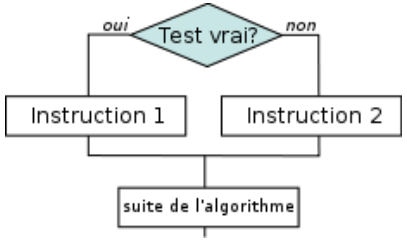
Voici un exemple volontairement simpliste :

Si *il pleut* **alors** *je prends mon parapluie*

qu'on pourrait traduire ainsi :

Si *temps=pluie* **alors** *prendreparapluie=oui*

Forme 2 : **si** *condition vérifiée* **alors** *instructions à effectuer* **sinon** *autres instructions à effectuer*



Exemple :

Si j'ai assez d'argent **alors** je m'achète la TX3000 **sinon** j'achète la Z20 qu'on pourrait traduire ainsi (en supposant que la TX3000 coûte 500 €) :

Si *argent>500* **alors** *achat= « TX3000»* ***sinon*** *achat = « Z20»*

Exemple 1 :

```
i ← 5
p ← 25
Si i < 10
alors p ← 32
Afficher p
```

L'algorithme affichera 32 car la condition i < 10 est vérifiée (5 < 10).

Exemple 2 :

```
f ← 5
b ← 3
s ← 20
Si b + f  >  s
alors s ← b + f
sinon s ← s + 1
Afficher s
```

L'algorithme affichera 21 car la condition « b + f > s » n'est pas vérifiée (3 + 5 ≯ 20) donc il exécute « s ← s + 1 » (donc s ← 21).

Si f prenait la valeur 26 au départ alors l'algorithme afficherait 29 car b + f = 3 + 26 = 29 > 20 donc il exécuterait l'instruction « s ← b + f ».

Traduction en Python

Le dernier algorithme s'écrirait ainsi :

```
f = 5
b = 3
s = 20
if b + f  >  s:
    s ← b + f
else:
    s ← s + 1
print(s)
```

Les boucles (répétitions d'instructions)

Les boucles Pour (boucle « bornées » ; for)

L'idée est ici de demander à la machine d'effectuer une répétition d'instructions un certain nombre de fois.

Exemple 1 :

```
Pour i allant de 0 à 99
Afficher « Bonjour ! »
FinPour
```

Ceci affiche « Bonjour ! » **100** fois.

Exemple 2 :

```
Pour k allant de 1 à 5
Afficher k²
FinPour
```

Ceci affiche les carrés de tous les nombres de 1 à 5.

Exemple 3 :

```
s ← 2
Pour n allant de 0 à 9
s ← s²
FinPour
Afficher s
```

Ceci met 2 au carré puis le résultat (4) au carré puis le résultat au carré (donc 16) et ceci 10 fois de suite. Affiche seulement le résultat final.

Traduction en Python

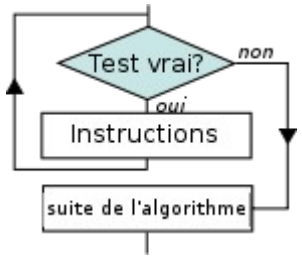
Exemple 1	Exemple 2	Exemple 3

`for i in range(100):`
`  print("Bonjour")`
	`for k in range(1, 6):`
	`  print(k**2)`
		`s = 2`
		`for n in range(10):`
		`  s = s**2`
		`print(s)`

Attention : remarquez que range(100) donne les nombres de 0 à 99, pas de 0 à 100 (ou de 1 à 100). Attention aussi à l'*indentation*.

Les boucles Tantque (boucle « non bornées » ; while)

L'idée est ici de demander à la machine d'effectuer une répétition d'instructions tant qu'une condition est vérifiée. Ceci est utile quand nous ne savons pas à l'avance combien de répétitions doivent être faîtes.
TantQue *condition vérifiée* **Faire** *instructions*
FinTantQue



Remarque : le mot Faire n'existe pas dans les langages de programmation.

Exemple 1 :

```
i ← 1
TantQue  i ⩽ 100
Faire
        Afficher « Je dois être attentif en classe. »
        i ← i + 1
FinTantQue
```

Cet algorithme affiche le message « Je dois être attentif en classe. » et augmente la valeur de i à 2. Comme i est encore inférieur ou égal à 100, la machine ré-affiche le message et augmente i à 2 et ainsi de suite 100 fois. Cet algorithme affiche donc 100 fois le texte « Je dois être attentif en classe. »et sert donc à faire des punitions !

Exemple 2 : (hasard(1 ; 6) choisit un nombre entier entre 1 et 6)

```
h ← hasard(1 ; 6)
Demander n
TantQue  n ≠ h
Faire
        Afficher « Mauvaise réponse. »
        Demander n
FinTantQue
Afficher « Bonne réponse ! »
```

Ici, la machine choisit un nombre entier au hasard entre 1 et 6 (appelé h). L'utilisateur entre des valeurs (dans n) tant qu'il n'a pas trouvé la valeur de h.

Exemple 3 :

```
Demander n
i ← 1
p ← 1
TantQue  i < n
Faire
        i ← i + 1
        p ← p × i
FinTantQue
Afficher p
```

Cet algorithme donne le nombre d'anagramme d'un mot de n lettres.
Par exemple, on peut prouver que « MOT » a $1\times2\times3=6$ anagrammes et que « MATHS » a $1\times2\times3\times4\times5=120$ anagrammes.

Traduction en Python

Exemple 1

```
i = 1
while i <= 100:
    print("Je dois être attentif en classe.")
    i = i + 1
```

Exemple 2

```
From random import randint
h = randint(1, 6)
n = int(input("Entrez un nombre")
while n!= h:
    print("Mauvaise réponse.")
    n = int(input("Entrez un autre nombre")
print("Bonne réponse !")
```

Attention
Observons l'exemple 1 :
– la variable i doit être définie avant le `while i<= 100` car sinon i n'a pas de valeur et « i<= 100 » donne une erreur ;
– la condition doit devenir fausse à un moment, sinon il y a une boucle infinie. Pour cela l'instruction `i = i + 1` est indispensable.
– attention aussi à l'*indentation*.