ALGORITHME : Ensemble de règles opératoires, qui prend en **entrée** une valeur, ou un ensemble de valeurs et qui donne en **sortie** une valeur ou un ensemble de valeurs, dont l'application permet de résoudre un problème **bien spécifié**, énoncé au moyen d'un nombre fini d'opérations.

Un algorithme peut être traduit, grâce à un langage de programmation, en un programme exécutable par un ordinateur.

Premier exemple : a et b sont deux entiers naturels

x $\leftarrow a$	Affectation
y $\leftarrow b$	Affectation
m $\leftarrow 0$	Affectation
tant que y $\neq 0$	Boucle
m $\leftarrow m + x*$(unité de y)	Affectation
x $\leftarrow 10 * x$	Affectation
y $\leftarrow$ nbre de dizaines de y	Affectation
fin tant que	

Combien vaut m pour a=23 et b=45 ?

```
x ← a
y ← b
m ← 0
tant que y ≠ 0
    m ← m + x*(unité de y)
    x ← 10 * x
    y ← nbre de dizaines de y
fin tant que
```

$$
\begin{array}{r}
\times \begin{array}{r} 2\ 3 \\ 4\ 5 \end{array} \\
\hline
1\ 1\ 5 \\
9\ 2\ \cdot \\
\hline
1\ 0\ 3\ 5
\end{array}
$$

C'est l'algorithme de la multiplication posée.

Deuxième exemple : a et b sont deux entiers naturels

x $\leftarrow a$
y $\leftarrow b$
m $\leftarrow 0$
tant que y $\neq 0$
    si y impair alors
        m $\leftarrow m + x$
    fin si
    x $\leftarrow 2 * x$
    y $\leftarrow$ E($y/2$)
fin tant que

Combien vaut m pour a=23 et b=45 ?

```
x ← a
y ← b
m ← 0
tant que y ≠ 0
    si y impair alors
        m ← m + x
    fin si
    x ← 2 * x
    y ← E(y/2)
fin tant que
```

m	y	x
0	45	23
23	22	~~46~~
23	11	92
115	5	184
299	2	~~368~~
299	1	736
1035	0	~~1472~~

```
x ← a
y ← b
m ← 0
tant que y ≠ 0
    si y impair alors
        m ← m + x
    fin si
    x ← 2 * x
    y ← E(y/2)
fin tant que
```

TERMINAISON : y étant divisé par 2 à chaque itération, il deviendra nécessairement nul après un nombre fini d'itérations.

VALIDITÉ : On établit que $m + x \cdot y$ reste en permanence égal à $a \cdot b$. Cet invariant joint à la condition d'arrêt $(y = 0)$ prouve, ici encore, que la variable m vaut $a \cdot b$ à la sortie de la boucle.

C'est l'algorithme de multiplication russe.

COMPLEXITÉ SPATIALE : Le codage en binaire des deux valeurs entrées donne un algorithme de multiplication russe plus efficace car multiplier et diviser par 2 consistent en des décalages d'un bit.

Validité: Un algorithme est dit correct si pour chaque entrée donnée, il donne la sortie attendue.

Un algorithme incorrect risque de ne pas se terminer ou de donner une sortie autre que celle désirée mais peut s'avérer utile dans certains cas, si son taux d'erreur est susceptible d'être contrôlé.

Complexité temporelle: Si rapides qu'ils puissent être, les ordinateurs ne le sont pas à l'infini. Le temps machine est donc une ressource limitée. Un algorithme performant permet d'économiser cette ressource.

PYTHON: Langage de programmation multiplateforme interprété (pas de compilation nécessaire) largement répandu.

EDUPYTHON: Environnement de développement intégré pour le langage Python.

MODULES: Bibliothèques contenant de nombreuses fonctions.

Module lycee.py de l'académie d'Amiens

```
pgcd(a,b)
cos(angle)
sqrt(x)
repere.plot(x, y,'optionnel')
randint(min,max)
vecteur(x,y,z='optionnel')
norme(v)
baton(xi,ni='optionnel',couleur='optionnel')
polygoneFCC(xi,ni='optionnel',couleur='b')
moyenne(xi,ni='optionnel')
mediane(xi,ni='optionnel',option='optionnel')
tirageBinomial(n,p)
normalFRep(a,b,mu,sigma)
...
```

La multiplication russe

```
from lycee import *
def multRusse(a,b) :
      x = a
      y = b
      m = 0
      while (y != 0) :
            if (reste(y,2)== 1):
                  m=m+x
            x=2*x
            y=quotient(y,2)
       return m
```

- fonction
- typages dynamiques par affectation
- comparaisons
- test
- boucle conditionnelle

Le tracé d'un polygone régulier sans règle ni compas

```
from lycee import *
def polyReg(n) :
      x,y= [],[]
      for k in range(n+1) :
            x.append(cos(2*k*pi/n))
            y.append(sin(2*k*pi/n))
      repere.plot(x,y)
      repere.show()
```

- liste
- fonction (procédure)
- boucle itérative

Effet de bord: Une fonction est dite à effet de bord si elle modifie un état autre que sa valeur de retour. Par exemple, une fonction pourrait :

- modifier une variable statique ou globale,
- modifier un ou plusieurs de ses arguments,
- écrire des données vers un écran ou un fichier,
- lire des données provenant d'autres fonctions à effet de bord.

Ces effets de bord compliquent souvent la lisibilité du comportement des programmes et/ou nuisent à la réutilisabilité des fonctions et procédures.

Une fonction idéale aurait un comportement purement fonctionnel, sans effet de bord, afin de maximiser son ré-emploi dans un autre programme.