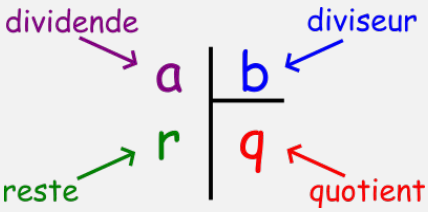
# LIFAP1 – TD 3 : Encore des algos…

*Objectifs* : Approfondir les notions vues dans le TD précédent (boucles, conditions, structures de données, entrées / sorties, …)

**La division euclidienne (ou division entière) et le modulo**




Le reste est également appelé **modulo** => a modulo b = r
Ici, a, b, q et r sont des entiers.

**Remarque** : en affectant dans un entier le résultat d'un calcul réel, on récupère la partie entière du résultat.

1. Écrire un algorithme qui teste si un entier choisi par l'utilisateur est multiple de 5 ou multiple de 7.

```
Début
Variables  valeur : entier

Afficher ('Donnez une valeur')
Saisir(valeur)
Si ((valeur modulo 5) = 0) ou ((valeur modulo 7) = 0)
        Alors Afficher (valeur, 'est multiple de 5 ou de 7')
        Sinon Afficher (valeur, 'n'est ni multiple de 5, ni multiple de 7')
FinSi
Fin
```

2. Écrire un algorithme qui calcule la somme des chiffres qui composent un nombre choisi par l'utilisateur.
   Exemple : valeur saisie : 1234 ➔ résultat : 10 (= 1 + 2 + 3 + 4)

```
Début
  Variables : nbre, sdc, i : entier
  Afficher ('Donnez une valeur')
  Saisir(valeur)
  sdc ← 0
  Tant que (valeur > 0) faire
      sdc ← sdc + (valeur  modulo 10)
      valeur ← valeur / 10
  Fin tant que
  Afficher ('La somme des chiffres qui composent ', nbre,' est : ', sdc)
Fin
```

3. Écrire un algorithme qui calcule les racines réelles (si elles existent) d'un polynôme du second degré décrit par 3 coefficients réels a, b et c. Les solutions seront affichées à l'écran.

```
Variables   a,b,c : réels
sol1, sol2, delta : réel
Début
Afficher('Entrez les 3 coefficients du polynôme')
Saisir(a,b,c)
delta ← b*b -4*a*c
Si (delta < 0)    Alors afficher ('pas de racines réelles')
```

```
                Sinon  Si (delta = 0)    Alors   sol1← -b /(2*a)
                                                  Afficher (‘une racine double :’, sol1)
                                           Sinon  sol1 ← (-b + sqrt(delta)) / (2*a)
                                                  sol2 ← (-b -  sqrt(delta)) / (2*a)
                                                  Afficher(sol1,sol2)
                          Fin Si
        Fin Si
        Fin
        Remarque : si a=b=0 alors on n’a pas un polynôme !
```

4. Écrire un algorithme permettant de trouver une valeur choisie aléatoirement par le programme. Le joueur disposera au maximum de 6 tentatives pour trouver cette valeur et le programme lui indiquera à chaque essai si sa valeur est trop grande ou trop petite.
   *Outil : pour choisir un nombre aléatoire, on utilisera en algorithmique : aleatoire()*

```
Variables : a_trouver, valeur, nb_essais : entiers
Début
  a_trouver ← aleatoire()
  nb_essais ← 0
  Faire
    Afficher(‘Donnez une valeur’)
    Saisir(valeur)
    Si (valeur > a_trouver)   Alors Afficher(‘trop grand’)
                              Sinon  Si (valeur < a_trouver)
                                        Alors Afficher(‘trop petit’)
                                     Fin si
    Fin si
    nb_essais ← nb_essais + 1
  Tant que ((valeur <> a_trouver) et (nb_essais < 6))
  Si (valeur = a_trouver)   Alors Afficher(‘gagné en ‘,nb_essais)
                            Sinon Afficher (‘perdu trop d essais’)
  Fin si
Fin
```

5. Écrire l’algorithme d’un programme permettant de vérifier si un entier est premier ou non. Un nombre premier est un nombre qui n’est divisible que par 1 et par lui-même.

```
variables locales     i : entier, est_premier : booléen
Début
    est_premier ← vrai
    Afficher ("Donnez un entier positif")
    Saisir(n)
    Pour i allant de 2 à racine(n) par pas de 1 faire
      Si (n modulo i ) = 0   Alors est_premier ← faux
      Fin si
    Fin pour
    Si est_premier Alors Afficher (n, " est un nombre premier")
                   Sinon Afficher (n, " n’est pas un nombre premier")
    fin si
Fin

Autre solution plus efficace (sortie de la boucle dès que l’on tombe sur un diviseur) :
variables locales     i : entier, est_premier : booléen
Début
    est_premier ← vrai
    Afficher ("Donnez un entier positif")
    Saisir(n)
    i ← 2
    Tant que ((est_premier) et (i <= racine(n))) faire
      Si (n modulo i ) = 0   Alors est_premier ← faux
      Fin si
      i ← i +1
    Fin pour
    Si est_premier Alors Afficher (n, " est un nombre premier")
                   Sinon Afficher (n, " n’est pas un nombre premier")
    fin si
Fin
```