FONCTIONS

- relation qui associe à une variable (ou plusieurs) une et une seule valeur
- Ex : f(x)=2x+3, définie de *R* dans *R*.

$$\begin{cases} f : R \longrightarrow R \\ x \longmapsto 2x + 3 \end{cases}$$

MATHS ⇒ INFO (C)

$$\begin{cases} f : R \longrightarrow R \\ x \longmapsto 2x + 3 \end{cases}$$

```
float f(float x)
{
  return 2.0*x+3.0;
}
```


MATHS ⇒ INFO (PYTHON)

$$\begin{cases} f : R \longrightarrow R \\ x \longmapsto 2x + 3 \end{cases}$$

```
def f(x):
  return 2.0*x+3.0
```


FONCTION INFORMATIQUE

- Paramètres formels, paramètres effectifs

```
float f(float x)
{
  return 2.0*x+3.0;
}
```

$$\begin{cases} f : R \longrightarrow R \\ x \longmapsto 2x + 3 \end{cases}$$

- Appels de fonctions

```
float y, z=5.0;
y=f(3.0); y=f(y); z=f(y)+f(2.1);
printf("%g\n ",f(2.*z)); y=sqrt(f(f(4.)));
```

FONCTION INFORMATIQUE

- Paramètres formels, paramètres effectifs

```
def f(x):
  return 2.0*x+3.0
```

- Appels de fonctions

```
z=5.0
y=f(3.0)
y=f(y);
z=f(y)+f(2.1)
print(f(2.*z))
```

$$\begin{cases} f : R \longrightarrow R \\ x \longmapsto 2x + 3 \end{cases}$$

STRUCTURE DES PROGRAMMES AVEC FONCTIONS

- Constantes (*)
- Fonctions
- Constantes (*)
- Variables
- Début
  - Instructions;
- Fin

EXEMPLE

```
/* ceci est un  commentaire */
float f(float x)
{
  return 2.0*x+3.0;
}
int main() {
  float z=5., y;
  y=f(3.0);
  printf("%f", f(2.*z));
  y=sqrt(f(f(4.)));
  return 0;
}
```


EXEMPLE

```
# ceci est un  commentaire
def f(x) :
  return 2.0*x+3.0

z=5
y=f(3.0)
print(f(2.*z))
y=f(f(4.))**0.5
```

PÉRIMÈTRE

```
float perimetre(float r)
{
  const float PI=3.141592;
  float res;
  res = 2.0*PI*r;
  return res;
}
```


```
int main() {
  float p, rayon;
  printf("entrez le rayon:");
  scanf("%f",&rayon);
  p = perimetre(rayon);
  printf(" perimetre : ");
  printf("%f\n",p);
  return 0;
}
```


PÉRIMÈTRE

```
def perimetre(r):
  PI=3.141592
  res = 2.0*PI*r
  return res

rayon=float(input("entrez le rayon:"))
p = perimetre(rayon)
print("perimetre : ",p)
```

EN CAML

```
let PI = 3.141592 ;;
let perimetre = function r ->
  2.0 *. PI *. r ;;

perimetre 100.0;;
```

EN PASCAL ...

```
function perimetre(r:real):real
const PI=3.1416;
var res:real;
begin
  res := 2.0*PI*r;
  result := res {ou perimetre:=res}
end

begin
 writeln('perimetre :',perimetre(100.))
end.
```


PLUSIEURS VARIABLES

```
float moyenne(float a, float b)
{
  float res;

  res = (a+b)/2.0;
  return res;
}
```

$$f : R \times R \longrightarrow R \qquad (a,b) \longmapsto (a+b)/2$$


```
int main() {
  float  x,y,m;
  printf("entrez 2 nombres : ");
  scanf("%f",&x); scanf("%f",&y);
  m=moyenne(x,y);
  printf("la moyenne vaut : ");
  printf("%f",m);
  return 0;
}
```


PLUSIEURS VARIABLES

```
def moyenne(a, b):
  res = (a+b)/2.0
  return res
```

$$f : R \times R \longrightarrow R \qquad (a,b) \qquad (a+b)/2$$


```
print("entrez 2 nombres : ")
x=float(input())
y=float(input())
m=moyenne(x,y)
print("la moyenne vaut : ",m)
```

PROCÉDURE

= fonction à résultat *void*

```
void afficherEntier(int x)
{
  printf("valeur : %d\n", x);
  return ; // facultatif
}
int main() {
  afficherEntier(10);
  return 0;
}
```


SANS PARAMÈTRE NI RÉSULTAT

```
void afficherMessageBienvenue(void)
{
printf("--------------------------\n");
printf("Bienvenue cher utilisateur\n");
printf("--------------------------\n");
}
int main() {
  afficherMessageBienvenue();
  return 0;
}
```


SANS PARAMÈTRE

```
int lireEntier(void)
{
  int x;

  scanf("%d",&x);
  return x;
}
```



```
int main ()
{
  int  a, b;
  a = lireEntier();
  b = lireEntier();
  afficherEntier(a+b);
  return 0;
}
```

PLUSIEURS RÉSULTATS : N-UPLET = STRUCTURES

- ou plusieurs fonctions (1 par résultat)

```
typedef struct {
  float m;
  float var;
} Couple;
typedef struct {
  float m, var;
} Couple;
```


```
Couple moyVariance(float a,float b,float c) {
  Couple res;
  res.m = (a+b+c)/3.0;
  res.var = ( (a-res.m)*(a-res.m)
    +  (b-res.m)*(b-res.m)
    +  (c-res.m)*(c-res.m)
    )/3.0;
  return res;
}
```



```
int main() {
  Couple c;
  float x, y, z;
  x=lireReel(); y=lireReel(); z=lireReel();
  c = moyVariance(x,y,z);
  printf("moyenne : %f\n",c.m);
  printf("variance : %f\n",c.var);
  return 0;
}
```



```
float moyenne(float a, float b, float c) {
  return (a+b+c)/3.0;
}
float variance(float a,float b,float c,float m) {
  float res;
  res=((a-m)*(a-m)+(b-m)*(b-m)+(c-m)*(c-m))/3.0;
  return res;
}
```

```
int main() {
  float m, var, x, y, z;
  x=lireReel(); y=lireReel(); z=lireReel();
  m = moyenne(x,y,z);
  var = variance(x,y,z,m);
  printf("moyenne : %f\n",m);
  printf("variance : %f\n",var);
  return 0;
}
```


PARAMÈTRES MODIFIABLES

- Passage par adresse
- Impossible dans la plupart des langages fonctionnels (Caml, Lisp)
- Possible en Pascal (var), en Php, …
- Délicat en C : pointeurs (cf. scanf !)

- Eviter si possible, et préférer n-uplets
- Fortement déconseillés dans ce cours !


```
void moyVariance(float a,float b,float c,float *m,float *v) {
   *m = (a+b+c)/3.0;
   *v =  ((a-*m)*(a-*m) + (b-*m)*(b-*m) + (c-*m)*(c-*m))/3.0;
   return; //facultatif
}
float moy, var;
moyEcartType(12.3, 9, 14.5,&moy, &var);
```



```
def moyVariance(a, b, c) :
  m = (a+b+c)/3.0
  var = ((a-m)**2+(b-m)**2+(c-m)**2))/3.0
  return (m,var)
```

```
x=float(input())
y=float(input())
z=float(input())
(m, v) = moyVariance(x,y,z)
print("moyenne : ", m)
print("variance : ",v)
```


PORTÉE DES VARIABLES

- Locales
  - déclarées dans une fonction
  - pas d'interférence entre variables
  - noms des paramètres
- Globales
  - effets de bord
  - difficulté de savoir qui modifie (voir aussi threads)
  - Eviter dans la mesure du possible = interdit dans ce cours !

EXEMPLE

```
int toto(int a, int b) {
  int c ;
  c = a + b;
  return c;
}
int a = 10, c = 1;
a = toto(5, 3); printf("%d ",c);
```


FONCTIONS EN PARAMÈTRE

```
float f1(float x)
{
  return 2.*x+3.;
}
float derivee(float(*f)(float),float x, float h)
{
  return (f(x+h)-f(x))/h;
}
```

FONCTIONS POUR L'ÉCROU

- Surface du triangle
- Surface du cercle
- Surface de l'octogone
- Volume de l'écrou

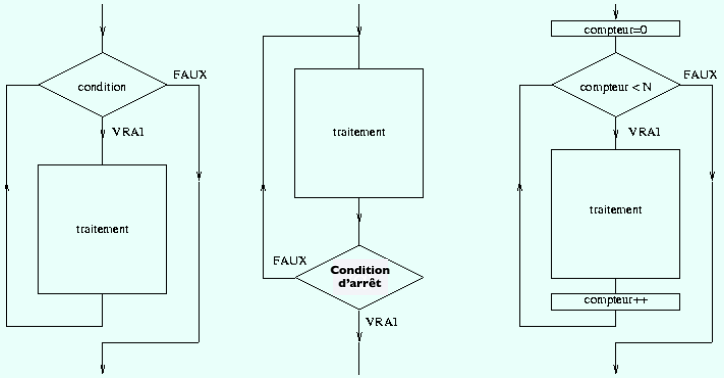
CALCUL DE LA DURÉE (H,M,S)

- Conversion en secondes de chaque temps
- Différence
- Conversion en heures, minutes et secondes

ITÉRATIONS

- Permettre de répéter des traitements
  - grand nombre
  - nombre inconnu à l'avance
- Structures de contrôle pour exprimer :
  - **tant que (***condition***) faire** *traitement* **fait**
  - **répéter** *traitement* **jusqu'à** (condition)
  - **faire pour** *compteur* **depuis** *début* **jusqu'à** *fin traitement*

CONTRÔLES DE L'ITÉRATION



RÉCURSIVITÉ

- Diviser un problème en sous-problèmes, un des sous-problèmes est le problème lui-même, dans un cas un peu plus simple
- Formule de récurrence
- Cas particuliers (arrêt)
- Fonction de simplification, strictement décroissante

RÉCURSIVITÉ

- Ne nécessite pas de comprendre les étapes du calcul : raisonnement descriptif, au niveau de la propriété
- Programmation
  - Plus facile (pas de syntaxe adhoc)
  - Beaucoup plus fiable
  - Réputée "moins" efficace

- Améliorations
  - Dérécursivation "automatique"
  - Récursivité terminale
- Récursivité : incontournable (pas d'autre solution)
- Récursivité : beaucoup mieux !

ITÉRATION

- Nécessite de comprendre les étapes du calcul : raisonnement au niveau du schéma d'exécution de la machine
  - décrire une variable (ou plus) qui va prendre toutes les valeurs successives du calcul
  - initialisation
  - condition d'arrêt
  - évolution de la condition
- Itération : plus difficile à mettre au point !

RÉCURSIF

- Afficher tous les nombres entre 1 et N

```
void afficherEntiers(int n) {
   if (n>0) {
         printf("%d ",n);
         afficherEntiers(n-1);
   }
}
```


ITÉRATIF

- Afficher tous les nombres entre 1 et N
  - une variable compteur
  - initialisation :

    compteur ← 1;
  - condition d'arrêt :

    compteur > N
  - traitement :

    afficher(compteur);
  - évolution de la condition :

    compteur←compteur+1;

TANT QUE ... FAIRE ... FAIT

```
i = 1;
while  !(i>N) {
    printf("%d ",i);
    i =  i+1;
}
```

```
i = 1;
while  !(i>N) :
    printf(i, end=' ')
    i =  i+1
```


FAIRE POUR... DEPUIS... JUSQUE

```
for (i=1; i<=N; i=i+1) {
        printf("%d ", i);
  }
```

```
for i in range (1,N) :
        print(i, end=' ')
```

FACTORIELLE

- 0! = 1
- n! = n * n-1 * n-2 * … * 1

  = n * (n-1 * n-2 * … * 1) = n * (n-1)!

- Relation de récurrence
- Domaine de définition, cas d'arrêt
- Fonction de simplification


```
int factorielle(int n)
{
  int f;
  if (n==0) f = 1;
  else f = n * factorielle(n-1);
  return f;
}
```



```
def factorielle(n) :
   if (n==0) :
      f = 1
  else :
      f = n * factorielle(n-1)
  return f
```

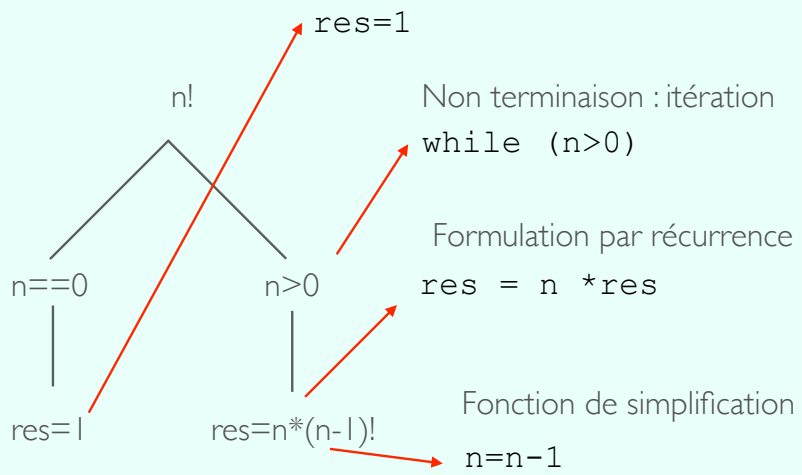

RÉCURSIVITÉ TERMINALE

```
int factorielle(int n, int acc) {
  int f;
  if (n==0) f =  acc;
  else f =  factorielle(n-1,acc*n);
  return  f;
}
```

- Appel

```
fc = factorielle(10,1);
```

Cas de terminaison :
donne valeur initiale



Non terminaison : itération

```
while (n>0)
```

Formulation par récurrence

```
res = n *res
```

Fonction de simplification

```
n=n-1
```

```
int factorielle(int n) {
  int res = 1;
   while (n>0) {
     res = n * res;
     n=n-1;
   }
  return res;
}
```

EXECUTION

```
fact(3)                    {n=3}
  3 * fact(2) //6          {n=2}
    2 * fact(1) //2        {n=1}
      1 * fact(0) //1      {n=0}
        1
```

PRODUIRE LES CONTEXTES

```
int factorielle(int n) {
  int res = 1;
   while (n>0) {
      res = n * res;
      n=n-1;
   }
  return res;
}
```

```
int factorielle(int n) {
  int i, res = 1;
   for (i=1;i<=n;i=i+1)
   {
      res = i * res;
   }
  return res;
}
```

X^N

- $x^n = x * x * x * \ldots * x$

  $= x * (x * x * \ldots * x) = x * x^{n-1}$

  $x^0 = 1$

- $x^n = x^{n/2} * x^{n/2}$, si n est pair

  $= x * x^{n/2} * x^{n/2}$, si n est impair

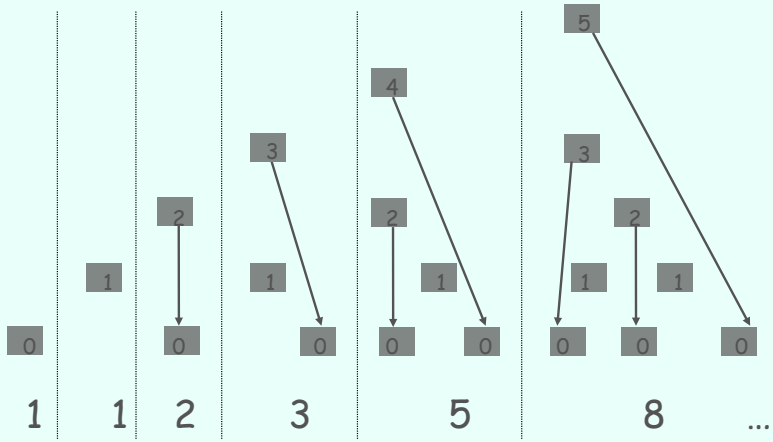
  $x^0 = 1$

DIVISION ENTIÈRE

- a div b
  - =0 si a<b
  - =1 + (a-b) div b
- Définition par récurrence
- Cas d'arrêt
- Fonction de simplification strictement décroissante

FIBONACCI

- $U_0=1, U_1=1$
- $U_n = U_{n-1}+U_{n-2}$
- Nombre d'or U_n / U_{n-1}
- Lapins
  - 1 couple donne 1 couple,
  - après une saison sans rien (si age >= 2)



RÉPÉTER UN CALCUL

- Avec question à l'utilisateur :

  Voulez vous continuer ?

- Réponse oui ou non : si non, on arrête

répéter

// calcul

afficher("encore oui/non"); rep← lire();

// que faire si autre réponse ?

TOURS DE HANOÏ

Règle du jeu

- *Vous ne pouvez déplacer qu'un seul palet à la fois.*
- *Vous ne pouvez pas poser un palet plus grand au-dessus d'un palet plus petit.*
- *Vous pouvez déplacer un palet sur n'importe quel autre pieu (contigu ou non) à condition de respecter les exigences sus-décrites.*

Malgré son nom exotique, ce jeu a été inventé à la fin du XIXe siècle (en 1883) par Édouard Lucas, un mathématicien français. À l'époque, le créateur prête au jeu des origines asiatiques, d'où le choix de son nom.

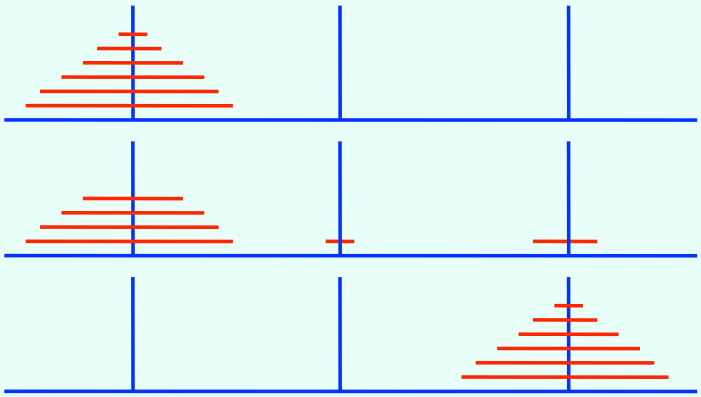
La première trace du jeu de la Tour de Hanoï, prétend-il, aurait été trouvée dans les écrits du Mandarin Fer-Fer-Tam-Tam. Le jeu serait également connu en Chine, au Japon ainsi qu'au Tonkin, pays dans lesquels ils sont fabriqués en porcelaine.

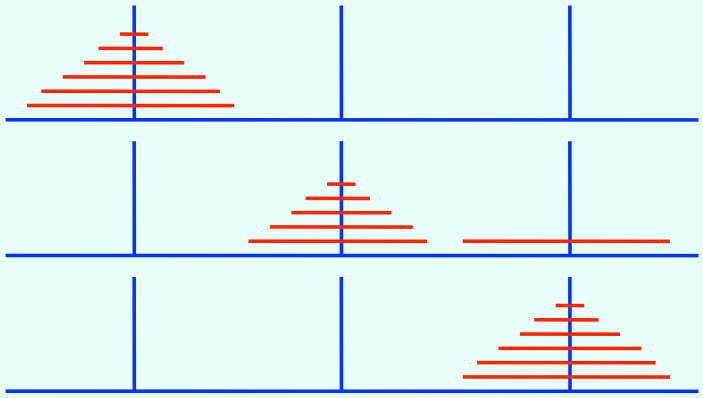
Un passionné des mélanges en tout genre...

On est cependant en mesure de dire que toutes ces informations liées à une provenance lointaine sont le fruit de l'imagination prolifique d'Édouard Lucas qui se fait d'ailleurs appeler Professeur N. Claus (de Siam), mandarin du collège de Li-Sou-Stian. En modifiant l'ordre des lettres qui composent le mot Li-Sou-Stian, on réalise qu'il s'agit de l'anagramme de Saint-Louis, précisément le collège où officie le professeur Édouard Lucas. Apparemment, l'individu aime les jeux de substitution. Le nom complet d'Édouard Lucas est Lucas d'Amiens, une nouvelle anagramme de N. Claus de Siam !

Un destin faste et tragique !

Né en 1842, Édouard Lucas est licencié de mathématiques à l'âge de 21 ans. Il enseigne dans différents lycées à partir de 1871. Réputé pour sa passion et ses compétences en arithmétique, il publiera un ouvrage en quatre volumes, intitulé *Les Récréations mathématiques*. Selon lui, tout théorème est capable d'engendrer un jeu de logique associé. Après une carrière brillante de mathématicien, il est emporté à l'âge de 49 ans par une septicémie. Cette infection du sang survient suite à une blessure au visage occasionnée par un morceau de porcelaine cassée...





```
void Hanoi(int n, char G, char D, char M) {
  if (n>0) {
  Hanoi(n-1, G, M, D);
  printf("mouvement de %c vers %c",G, D);
  Hanoi(n-1, M, D, G);
  }
}
```


RÉCURSIVITÉ CROISÉE

- Pair si pas impair
- Impair si pas pair

RACINE CARRÉE APPROCHÉE

- Recherche dichotomique
- $X_{J+1} = (X_J + A/X_J) / 2, X_1 = A$

NOMBRES RÉELS

- Pour être représenté en binaire, tout nombre réel X est décomposé en un triplet (S, E, F) de sorte que :
  - $X=(-1)^S * 2^{E-127} * 1.F$
  - S est facile à trouver (0 si le nombre est positif, 1 sinon). En revanche, pour obtenir E, on cherche la plus grande valeur de k=E-127 tel que $2^k < |X|$ (valeur absolue de X). Enfin, on obtient F ainsi : $F = (-1)^s * X * 2^{-(E-127)} - 1$
  - Ecrivez une fonction qui convertit tout nombre réel en un triplet (S, E, F).