

Quelques principes de programmation
(ceci n'est pas un cours d'algo !)

Réfléchir avant de coder ...

- les programmes (surtout non triviaux) nécessitent de réfléchir avant de coder
- algorithmes !
- permet : analyse modulaire, support de test, documentation
- exemple : faire une omelette ...
(dur ! -- surtout pour certains)

Omelette

- faire une omelette
- fini ? NON

Omelette

- faire en séquence :
 - trouver les ingrédients
 - faire la base
 - faire cuire
- fini ? NON

Omelette

- faire en séquence :
 - trouver les ingrédients = faire en séquence :
 - trouver les oeufs (4)
 - trouver le sel
 - trouver le poivre
 - faire la base
 - faire cuire
- fini ? NON

Omelette

- faire en séquence :
 - trouver les ingrédients = ...
 - faire la base = faire en séquence :
 - casser les oeufs
 - mettre du sel
 - mettre du poivre
 - mélanger
 - faire cuire
- fini ? NON

Omelette

- faire en séquence :
 - trouver les ingrédients = ...
 - faire la base = faire en séquence :
 - casser les oeufs
= tant qu'il reste des oeufs (non cassés) faire :
 - prendre un de ces oeufs
 - casser l'oeuf
 - mettre du sel
 - mettre du poivre
 - mélanger
 - faire cuire
- fini ? NON

Omelette

- faire en séquence :
 - trouver les ingrédients = ...
 - faire la base = faire en séquence :
 - casser les oeufs
= tant qu'il reste des oeufs (non cassés) faire [...]
 - si pas régime sans sel : mettre du sel
 - mettre du poivre
 - mélanger
= tant que pas homogène :
 - mélanger
 - faire cuire
- fini ? NON

Omelette

- faire en séquence :
 - trouver les ingrédients = ...
 - faire la base = ...
 - faire cuire
 - = faire en séquence :
 - minutage = 10 min
 - tant que minutage > 0 faire :
 - faire chauffer l'omelette
 - minutage = minutage - 1min
- fini ? OUI

Omelette (synthèse)

- pour résoudre ce problème complexe (omelette)
nous avons utilisé :
- des séquences
- des conditionnelles
- des boucles
- des variables

Plan

- variables
- espaces de visibilité des variables
- conditionnelles
- boucles
- fonctions
- quelques programmes de base

Variable

- une variable a un nom
- désigne à la base un espace de la mémoire où est stockée une valeur
- associé(e) ou non à un type
- affectation :
nom <- 'Jean' nom := 'Jean'
nom = 'Jean' let nom='Jean' in ... (*)
- utilisation :
écrire(nom) age = age+1

Visibilité

- Variables globales
définies dans le cadre d'un programme
visibles partout (sauf si ...)
- Variables locales
définies dans un sous-cadre
(ex: fonction définie dans le cadre d'un
programme)
visibles dans le sous-cadre
- Paramètres des fonctions
(nous verrons cela plus tard)

Conditions

- Quelque chose qui est vrai ou faux
 - il fait beau à Evry en ce moment
 - mon chien s'appelle Marcel
 - tous les oeufs sont cassés
 - Jean est majeur
- Dans un(le) langage de programmation
 - temps(Evry) = 'beau'
 - nom(chien(Prof)) = 'Marcel'
 - oeufNonCassés = 0
 - age(Jean) > 17

Conditions

- bien sur il est possible de combiner des conditions : ET, OU, NON, ...
 - il fait beau et mon chien s'appelle Marcel
temps(Evry) = 'beau' ET nom(chien(prof)) = 'Marcel'
 - Jean a entre 13 et 18 ans (non compris)
age(Jean)>13 ET age(Jean)<18
 - Jean est majeur ou a un chien appelé Marcel
age(Jean)>17 OU nom(chien(Jean))='Marcel'
(vrai si ... ou ... ou les 2 !)

Conditionnelle

- Permet de séquencer en fonction d'une condition
 - s'il fait beau alors prendre mes lunettes de soleil sinon prendre mon parapluie
 - SI temps(Evry)='beau' ALORS
 objetPris = 'lunettes'
SINON
 objetPris = 'parapluie'
FIN SI
- (on décale pour que ce soit lisible !)

Boucles

- TANT QUE condition FAIRE
 qqe chose
FIN TANT QUE

- majeur = faux
age = 0

TANT QUE age < 200 FAIRE

 age = age + 1 # vieillir

 SI age = 18 ALORS

 majeur = vrai # on passe majeur

 FIN SI

FIN TANT QUE

Boucles

- Itérer dans une liste
- Très courant :
 - somme des éléments d'une liste
 - rechercher un élément dans une liste
 - pour tout i dans $[1, N]$: ...
- POUR variable DANS liste FAIRE
FIN POUR
 - somme = 0, liste_valeurs = [1,2,3,4]
POUR element DANS liste_valeurs FAIRE
 somme = somme + liste_valeurs
FIN POUR

Fonctions

- Réutiliser du code
- Paramètres formels (définition de la fonction)
- Paramètres effectifs (lors de l'appel)
- Possible valeur de retour
 - DEFINITION FONCTION somme(x,y) COMME
 RETOURNER x+y
 FIN DEFINITION
 - $v = \text{somme}(2,3)*5$ # = $5 \times 5 = 25$

Ex1 : Somme de valeurs

somme = 0

liste_valeurs = [...]

POUR element DANS liste_valeurs FAIRE

 somme = somme + element

FIN POUR

Ex2 : Somme de valeurs (2)

DEFINITION FONCTION pair(nb) COMME

SI (modulo(nb,2)=0) ALORS

RETOURNER vrai

SINON

RETOURNER faux

FIN DEFINITION

Ex2 : Somme de valeurs (2)

somme = 0

liste_valeurs = [...]

POUR element DANS liste_valeurs FAIRE

 SI pair(element) ALORS

 somme = somme + element

 FIN SI

FIN POUR

Ex3 : Somme de valeurs (3)

somme = 0

liste_valeurs = [...]

indice = 0

TANT QUE indice < taille(liste_valeurs) FAIRE

 somme = somme + liste_valeurs[indice]

 indice = indice + 1

FIN TANT QUE

[2 4 55 7 9 10]

0 1 2 3 4 5 ...

Ex4 : Chercher une valeur

valeur = ..., liste_valeurs = [...]

t = taille(liste_valeurs)

indice = 0

TANT QUE (indice < taille) FAIRE

 SI liste_valeurs[indice] = valeur ALORS

 "SORTIE BOUCLE" # rang voulu = indice

 SINON

 indice = indice + 1

 FIN SI

FIN TANT QUE

si indice=taille alors non trouvé, si indice < taille alors trouvé

Ex4 : Chercher une valeur (2)

valeur = ..., liste_valeurs = [...]

t = taille(liste_valeurs)

indice = 0, trouvé = faux

TANT QUE (indice < taille) ET NON trouvé FAIRE

 SI liste_valeurs[indice] = valeur ALORS

 trouvé = vrai # rang voulu = indice

 SINON

 indice = indice + 1

 FIN SI

FIN TANT QUE

si NON trouvé alors non trouvé, sinon alors trouvé (indice)

Ex4 : Chercher une valeur (3)

Plus problématique avec un POUR

```
valeur = ..., liste_valeurs = [...]
```

```
POUR element DANS liste_valeurs FAIRE
```

```
  SI element = valeur ALORS
```

```
    "SORTIE BOUCLE"      # rang voulu = ??
```

```
  FIN SI
```

```
FIN POUR
```

```
# comment récupère-t-on le rang ?
```