Sous-programmes
De quoi s'agit-il ?

Imaginons une application bancaire qui doit demander des informations sur un client avant de décider si la banque peut ou pas accorder un crédit à ce client.

Algorithme Application_Bancaire

```
...
Ecrire ("Êtes-vous marié ?")
Répéter
    Lire ("Tapez Oui ou Non." ; Rep1)
Jusqu'à (Rep1 = "Oui" OU Rep1 = "Non")
Ecrire ("Avez-vous des enfants ?")
Répéter
    Lire ("Tapez Oui ou Non." ; Rep2)
Jusqu'à (Rep2 = "Oui" OU Rep2 = "Non")
Ecrire ("Avez-vous déjà bénéficié d'un crédit ?")
Répéter
    Lire ("Tapez Oui ou Non." ; Rep3)
Jusqu'à (Rep3 = "Oui" OU Rep3 = "Non")
...
```

Qu'en est-il lorsqu'on a à saisir une dizaine de réponses par oui ou non ?

Sous-programmes
De quoi s'agit-il ?

Une application, surtout si elle est longue, a des chances de devoir procéder aux **mêmes traitements**, ou à des **traitements similaires**, à **plusieurs endroits** de son déroulement.

Solution possible :

- Programmer des traitements similaires peut se faire en répétant le code correspondant autant de fois que nécessaire.
- Recopier les lignes de codes voulues en ne changeant que le nécessaire.

Inconvénients : Le programme ...

- Il devient lourd
- Il contient des répétitions
- Il peut devenir parfaitement illisible
- Sa structure peut poser des problèmes de maintenance

Sous-programmes
De quoi s'agit-il ?

Solution

Il faut **séparer** les **traitements similaires** du **corps du programme** et **regrouper** les instructions qui les composent en un **module séparé**. On peut, par la suite, **appeler** ce **groupe d'instructions** chaque fois qu'on en a besoin dans le corps du programme.

Conséquences :

- **Lisibilité** assurée. On dit que le programme devient **modulaire**.
- Pour la **maintenance**, il suffit de faire une seule modification dans le groupe d'instructions pour que cette modification prenne effet dans la totalité de l'application.
- Le **corps du programme** s'appelle alors le **programme principal**.
- Les **groupes d'instructions** auxquels on a recours s'appellent des **fonctions** et des **procédures**. Ils forment ce qu'on appelle des **sous-programmes**.

Sous-programmes
Les fonctions

Algorithme Application_Bancaire

```
...
Ecrire ("Êtes-vous marié ?")
Répéter
    Lire ("Tapez Oui ou Non." ; Rep1)
Jusqu'à (Rep1 = "Oui" OU Rep1 = "Non")
Ecrire ("Avez-vous des enfants ?")
Répéter
    Lire ("Tapez Oui ou Non." ; Rep2)
Jusqu'à (Rep2 = "Oui" OU Rep2 = "Non")
Ecrire ("Avez-vous déjà bénéficié d'un crédit ?")
Répéter
    Lire ("Tapez Oui ou Non." ; Rep3)
Jusqu'à (Rep3 = "Oui" OU Rep3 = "Non")
...
```

- **isoler** les instructions demandant une réponse par Oui ou Non ;
- **appeler** ces instructions à chaque fois que nécessaire ;
- créer une **fonction** dont le rôle est de **renvoyer** la réponse (oui ou non) de l'utilisateur.

Sous-programmes
Les fonctions

```
Fonction Rep_Oui_Non() : chaîne
Var : Truc : chaîne
   Répéter
      Lire ("Tapez Oui ou Non." ; Truc)
   Jusqu'à (Truc = "Oui" OU Truc = "Non")
   Renvoyer   Truc
Fin
```

- Le mot-clé **Renvoyer** indique quelle valeur doit prendre la fonction lorsqu'elle est utilisée par le programme.
- La valeur renvoyée par la fonction est contenue dans le **nom de la fonction lui-même**.
- Une fonction s'écrit toujours **en-dehors du programme principal**.

Sous-programmes
Les fonctions

Algorithme Application_Bancaire

...
Ecrire ("Êtes-vous marié ?")
Rep1 ← Rep_Oui_Non()
Ecrire ("Avez-vous des enfants ?")
Rep2 ← Rep_Oui_Non()
Ecrire ("Avez-vous déjà bénéficié d'un crédit ?")
Rep3 ← Rep_Oui_Non()
...

- Pour chaque question, on écrit un **message** à l'écran, et on **appelle** la fonction **Rep_Oui_Non()**

- **Comment inclure l'écriture du message directement dans la fonction ?**

Sous-programmes
Passage d'arguments dans une fonction

Pour faire figurer le message dans la fonction **Rep_Oui_Non()** :

1. Lorsqu'on appelle la fonction, on doit lui **préciser** quel message elle doit afficher avant de lire la réponse.
2. La fonction doit être **prévenue** qu'elle recevra un message, et être capable de le récupérer pour l'afficher.

▶ En algorithmique, on dit que le **message** devient un **argument** (ou un **paramètre**) de la fonction.

Sous-programmes
Passage d'arguments dans une fonction

Fonction Rep_Oui_Non(Msg : chaîne) : chaîne
Var : Truc : chaîne
  Ecrire (Msg)
  Répéter
    Lire ("Tapez Oui ou Non." ; Truc)
  Jusqu'à (Truc = "Oui" **OU** Truc = "Non")
  Renvoyer Truc
Fin

La fonction inclue entre les parenthèses une variable dont on précise le type et qui signale à la fonction qu'un argument doit lui être envoyé à chaque appel.

Algorithme Application_Bancaire

  ...
  Rep1 ← Rep_Oui_Non("Êtes-vous marié ?")
  Rep2 ← Rep_Oui_Non("Avez-vous des enfants ?")
  Rep3 ← Rep_Oui_Non("Avez-vous déjà bénéficié d'un crédit ?")
  ...

Sous-programmes
Les fonctions

Format général d'une fonction :

Fonction *Nom_Fonction*(*Arg_1* : Type, *Arg_2* : Type, ...) : Type
Var : [*Variables_Locales*]
...
[*Traitements*]
...
Renvoyer [*Valeur_Renvoyée*]
Fin

Sous-programmes
L'analyse fonctionnelle

- En algorithmique, le plus **difficile**, mais aussi le plus **important**, c'est d'acquérir le réflexe de constituer systématiquement les fonctions **adéquates** quand on doit **traiter un problème donné**.
- Il faut prendre l'habitude de **découper** son algorithme en **différentes fonctions** pour le rendre **léger**, **lisible** et **performant**.
- Cette partie de la réflexion s'appelle l'**analyse fonctionnelle d'un problème** et c'est toujours par elle qu'il faut commencer.
- Dans un **premier temps**, on **découpe** le traitement en **modules** (algorithmique fonctionnelle) et dans un **deuxième temps**, on **écrit** chaque **module** (algorithmique classique).

Sous-programmes
Exemples

Écrire une fonction qui renvoie la moyenne de deux entier.

Fonction Moyenne(A : entier, B : entier) : réel
Var : Moy : réel
  Moy $\leftarrow$ (A + B) / 2
  Renvoyer Moy
Fin

Sous-programmes
Exemples

Écrire une fonction qui renvoie la factorielle d'un entier.

```
Fonction Factorielle(N : entier) : entier
Var : Fact, i : entier
   Fact ← 1
   Pour i ← 2 à N Faire
      Fact ← Fact * i
   Fin Pour
   Renvoyer   Fact
Fin
```

Sous-programmes
Exemples

Écrire une fonction qui renvoie le maximum de deux réels.

Fonction Maximum(A : réel, B : réel) : réel
  Si (A $\geq$ B) **Alors**
    Renvoyer A
  Sinon
    Renvoyer B
  Fin Si
Fin

Sous-programmes
Exemples

Écrire un algorithme qui permet de déclarer un tableau de 10 réels, de le remplir par des valeurs saisies par l'utilisateur et d'en chercher la valeur maximale.

Algorithme Table_Maximum

Var : R(9), Max : réel , i : entier

Début
  Pour i ← 0 à 9 **Faire**
    Ecrire ("Entrez la valeur numéro " ; i + 1 ; " dans le tableau")
    Lire (R(i))
  Fin Pour
  Max ← R(0)
  Pour i ← 1 à 9 **Faire**
    Max ← Maximum(Max , R(i))
  Fin Pour
  Ecrire ("La valeur maximale est " ; Max)
Fin

Fonctions et procédures

La définition d'une fonction est introduite par le mot-clé `def`

- Suivi du nom de la fonction
- Entre parenthèses, ses arguments (parenthèses vides si aucun)
- Enfin, la ligne est terminée par deux points `:`

Le corps de la fonction est un bloc : on l'**indente** donc d'un cran vers la droite

On peut sortir d'une fonction de deux façons :

- En arrivant à la fin de la fonction : retour à l'indentation de niveau maximal (complètement à gauche), dans le cas d'une procédure
- En exécutant le mot-clé `return`
  - Soit seul : fin d'une procédure (ne retournant rien)
  - Soit suivi d'une variable qui est retournée par la fonction

```
1 def maFonction( a ):
2         print "parametre : " + str( a ) + " de type " + str( type( a ) )
3         return type( a )
```

Appel d'une fonction

On appelle une fonction **par son nom**, en lui passant ses **paramètres entre parenthèses**

```
1  def maFonction( a ):
2          print "parametre : " + str( a ) + " de type " + str( type( a ) )
3          return type( a )
4
5  # ailleurs dans le programme
6  maFonction( 5 )
7  maFonction( "toto" )
```

Attention : pas de vérification du type des arguments passés

- Source d'erreurs pas directement détectées : c'est lors de l'utilisation de la variable de type incorrect dans la fonction que l'erreur est annoncée (si elle l'est...)
- Rend possible l'appel de la fonction avec des arguments de différents types

Point d'entrée dans le programme

L'exécution d'un script Python commence par la **première ligne en-dehors de toute fonction**

- On exécute la première ligne du bloc de plus haut niveau qui ne soit pas une définition de fonction

```
1 def maFonction( a ):
2         print "parametre : " + str( a ) + " de type " + str( type( a ) )
3         return type( a )
4
5 maFonction( 5 ) # premiere ligne executee
```

Les programmes Python complexes sont souvent composés de plusieurs *modules* (plus de détails plus tard)

- Nom du module dans lequel on se trouve : variable `__name__`
- Module principal = `__main__`

Généralement, on commence par tester si on se trouve dans le module principal

- Si c'est le cas, on effectue nos appels de fonction
- Intérêt : écrire des modules auto-suffisants ou utilisables par d'autres scripts

```
1 if __name__ == "__main__":
2     appel_fonction()
```