# Python

## Notions de modularité

Valérie Bellynck (2009-2010),
d'après un diaporama d'Alain Bozzi (2008),
lui-même venant de celui de Bob Cordeau (2008)

# Les sous-programmes

**sous-programme =**

suite nommée d'instructions, que l'on peut appeler chaque fois que nécessaire

**2 types de sous-programmes :**

- **la fonction :** suite d'instructions effectuant un calcul.
  Le résultat du calcul est retourné ou renvoyé par l'instruction :
  `return nom_Variable ou expression`
- **la procédure :** suite d'instructions pour effectuer une tâche.
  Il n'y a pas l'instruction `return` donc pas de résultat retourné.

On parlera toujours de fonction.
Si elle ne retourne pas de valeur (pas d'instruction `return`) ce sera une procédure.

**Un sous-programme dépend de valeurs entrées en paramètre**

**Définition d'un sous-programme en Python**

```
def nomFonction(param1, param2, ...) :
        suite d'instructions
```

Attention à l'indentation en Python : c'est elle qui délimite le bloc d'instructions

# Les procédures - un exemple

## Exemple :

Séquence minimale d'instructions

pour tracer un carré avec le module **Turtle** :

```
for i in range(4):
    forward(100)
    left(90)
```

On peut regrouper ces instructions

dans une fonction que l'on nommera **dessineCarre()**

# Les procédures

## Définition d'un sous-programme en Python

```
def nomFonction(param1, param2, ...) :
        suite d'instructions
```

## Soit, dans l'exemple du carré :

```
def dessineCarre() :
    for i in range(4):
        forward(100)
        left(90)
```

**Ce sous-programme**

- **ne contient pas l'instruction `return`, il ne retourne pas de valeur, c'est donc une procédure.**
- **ne contient pas de paramètres.**

# Les procédures paramètrées

## Introduction d'un paramètre :

- on veut pouvoir changer la longueur du côté et utiliser la même procédure

## Soit, dans l'exemple du carré :

```
def dessineCarre(cote) :
    for i in range(4):
        forward(cote)
        left(90)
```

**Ce sous-programme**

- **est une procédure (ne contient pas d'instruction `return`)**
- **contient un paramètre.**

# L'appel d'une procédure

## Utilisation d'un sous-programme dans un script Python

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from Turtle import *

# définition de la fonction
def dessineCarre(cote):
  for i in range(4):    # pour les 4 cotés
        forward(cote)   # pour la longueur de chaque coté
        rigth(90)       # pour tourner

# appel de la fonction
dessineCarre(100)
```

# Les fonctions

**Exemple 2 :**

```
def carre(val):
        return val * val
```

un entier

**Exemple 3 :**

```
def estPair(val):
        return val % 2 ==0
```

un booléen

**Exemple 4 :**

une chaîne

```
def messagerie(email,texte):
        return "<a href=mailto:" + email + ">"  + texte + "</a>"
```

**Ces sous-programmes contiennent l'instruction return, ils retournent respectivement un entier pour l'exemple 2, un booléen pour l'exemple 3, une chaîne de caractère pour l'exemple 4, ce sont donc des fonctions.**

# L'appel d'une fonction

## Utilisation d'un sous-programme dans un script Python

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
from Turtle import *
longueur = int(raw_input("Entrez la longueur d'coté : "))
# définition de la fonction
def dessineCarre(cote):
  for i in range(4):          # pour les 4 cotés
    forward(cote)             # pour la longueur de chaque coté
    rigth(90)                 # pour tourner
def carre(val):
    return val * val          # rend la valeur au carré
def estPair(val):
    return val % 2 ==0        # rend vrai si la valeur est paire ,faux sinon
# appel de la fonction
dessineCarre(longueur)
if(estPair(longueur)):
    write ("La surface du carré, " + carre(longueur) + ", est paire. " )
```

# Les modules

- **Contenu d'un module**
  - Regroupement de variable et de fonctions.
  - Le but d'un module est de proposer des opérations sur des domaines différents.

- **Création et utilisation d'un module**
  - définir des fonctions dans un fichier et le nommer **mNomFichier.py**
  - créer un autre fichier et utiliser les différentes fonctions en prenant soin de mettre au début du fichier la ligne suivante :

```
import mNomFichier

ou

from mNomFichier import *
```

# Les modules

**`import nomFichier`** et **`from nomFichier import *`**

- import nomFichier nécessite de préfixer les fonctions, sinon il se produit une erreur “NameError”.
- Si dans le module mDessin on trouve la fonction carre(largCote), il faut l’utiliser de la manière suivante :

```
import mDessin
mDessin.carre(100)
```

- Dans le cas de from mDessin import *, le préfixage n’est plus nécessaire,
  on utilise le module de la manière suivante :

```
from mDessin import *
carre(100)
```