Sous-programmes

Introduction à la programmation en Python

Sommaire

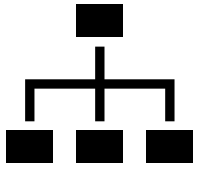
1. Généralités algorithmiques.
2. Les sous-programmes en Python.

1. Généralités algorithmiques.

1. Généralités algorithmiques.

Principe

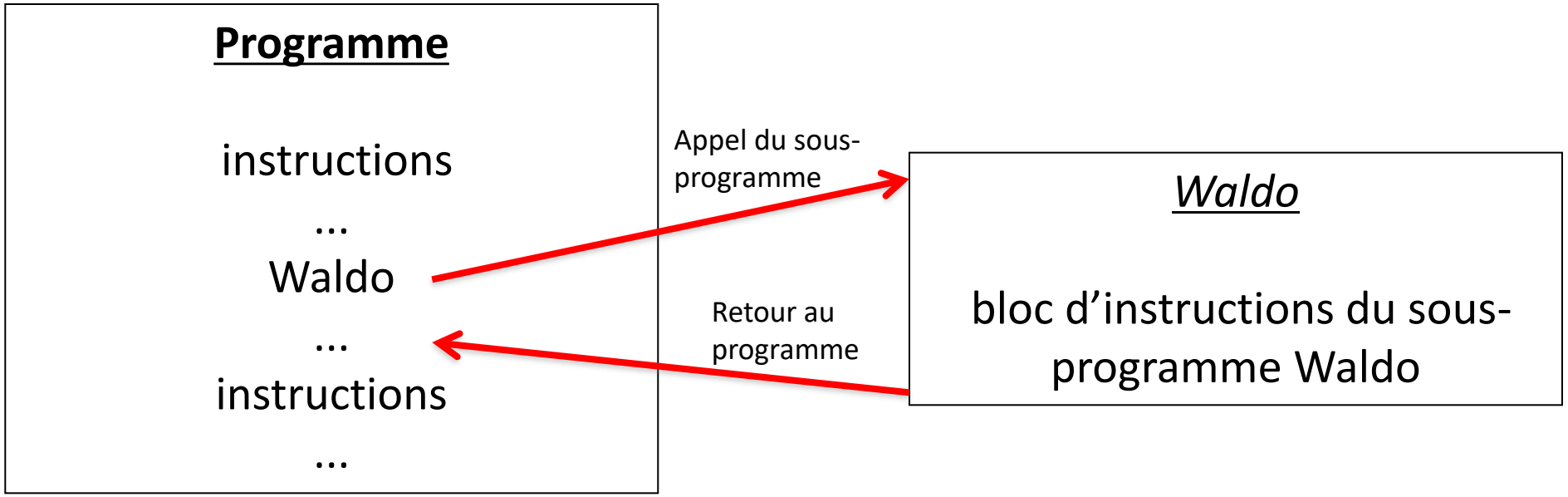
- Un sous-programme est un bloc d'instructions réalisant une certaine tâche.
- Il possède un nom et est exécuté lorsqu'on l'appelle.
- Un script bien structuré contiendra un programme dit "principal", et plusieurs sous-programmes dédiés à des fonctionnalités spécifiques.



1. Généralités algorithmiques.

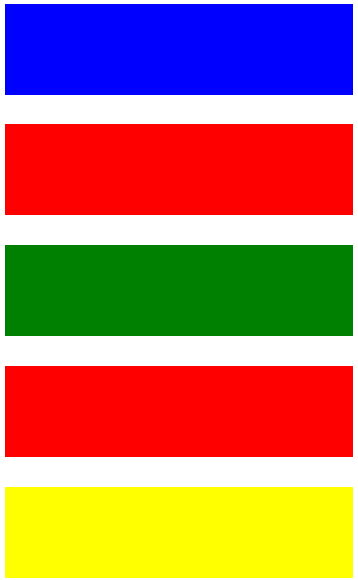
Déroulement

- Quand le programme principal fait appel à un sous–programme, il suspend son propre déroulement, exécute le sous-programme en question, et reprend ensuite son fonctionnement.

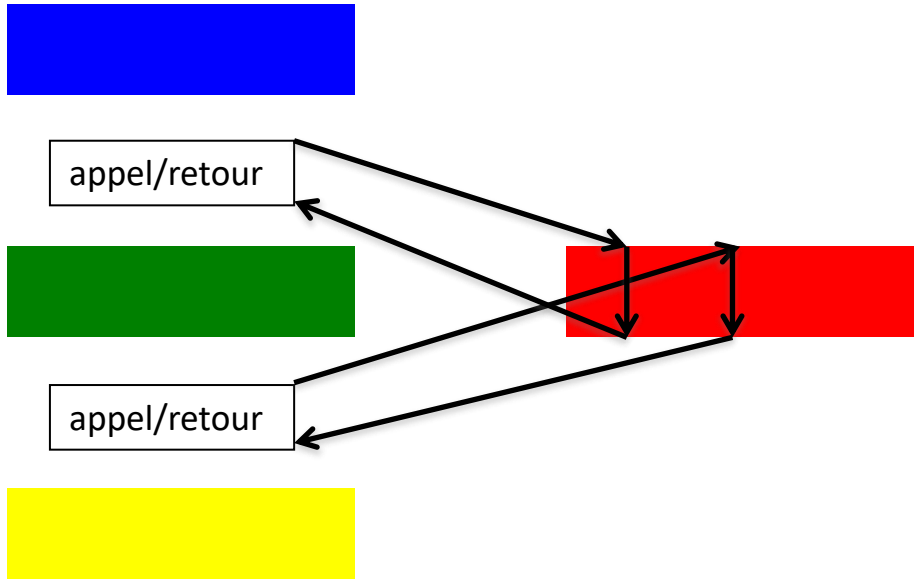


1. Généralités algorithmiques.

Premier avantage : éviter la duplication de code



Ici le bloc de code en rouge est dupliqué



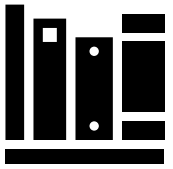
Ici on crée un sous-programme correspondant à ce bloc, et on l'appelle quand on en a besoin.

1. Généralités algorithmiques.

Second avantage : favoriser la réutilisation

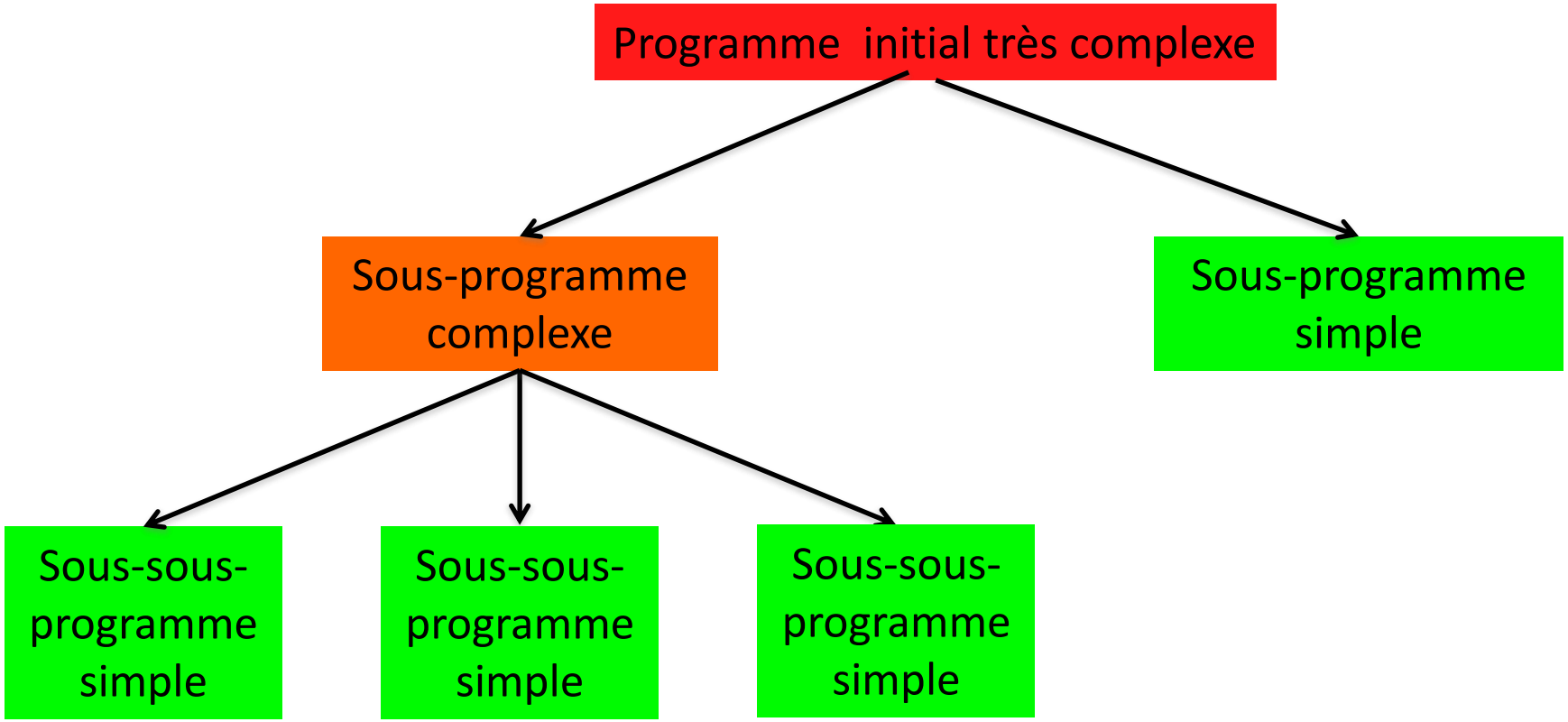
- Un sous-programme écrit pour résoudre un problème donné pourra servir de nouveau dans un autre contexte.

- On pourra ainsi créer des librairies de sous-programmes.



1. Généralités algorithmiques.

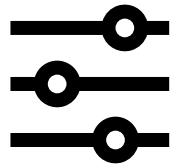
Troisième avantage : améliorer la conception



1. Généralités algorithmiques.

Paramètres

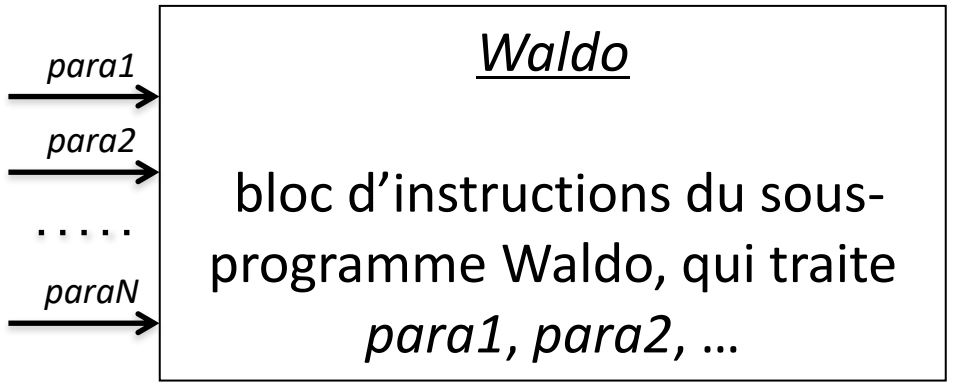
- Un sous-programme sert donc à effectuer un traitement générique.
- Ce traitement porte sur des données, dont la valeur pourra ainsi changer d'un appel du sous-programme à un autre.
- Ce que l'on appelle paramètres ce sont justement ces données transmises au sous-programme par le programme principal.



1. Généralités algorithmiques.

Paramètres

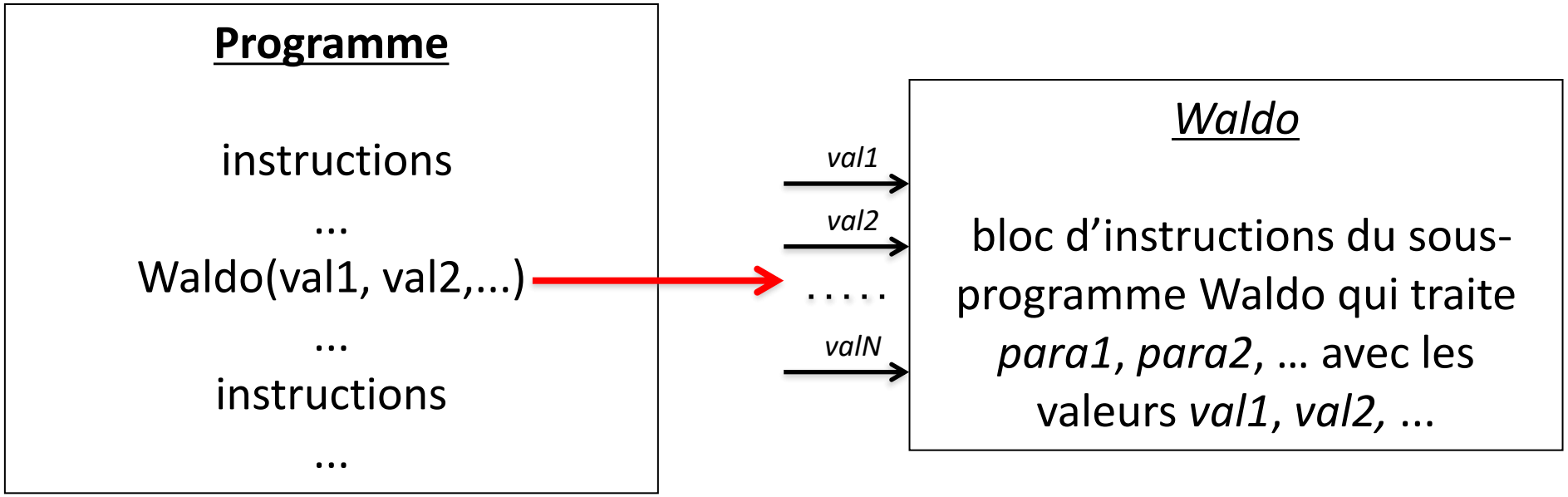
- Lors de l'implémentation d'un sous-programme, on précise la liste de tous les paramètres qu'il va utiliser.



1. Généralités algorithmiques.

Paramètres

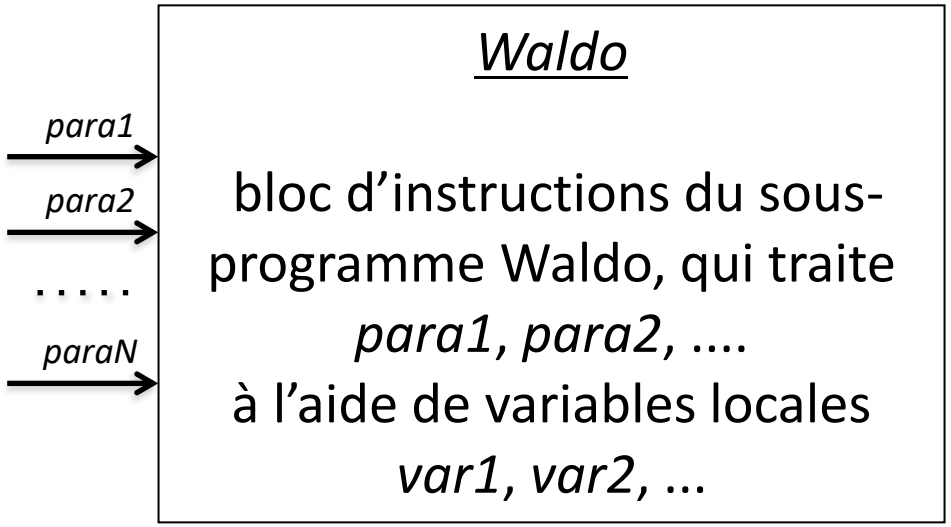
- Lors de l'utilisation d'un sous-programme, on va alors préciser la valeur de chacun des paramètres qu'il possède.



1. Généralités algorithmiques.

Variables locales

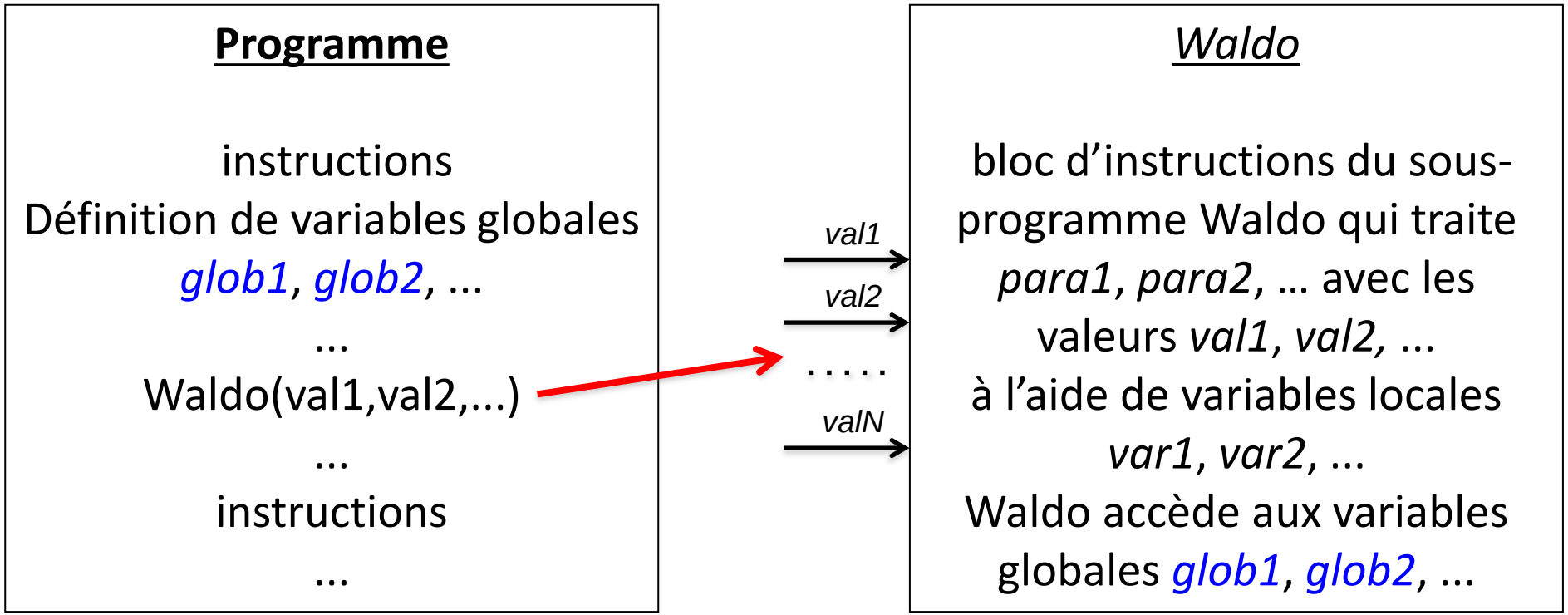
- Ce sont des variables définies à l'intérieur d'un sous-programme afin de procéder au traitement des paramètres.



1. Généralités algorithmiques.

Variables globales

- Ce sont des variables définies dans le programme principal et accessibles par un sous-programme.



1. Généralités algorithmiques.

Bonne pratique

- Si un sous-programme utilise des données issues du programme principal on préfèrera les passer en paramètres plutôt que les définir de façon globale.

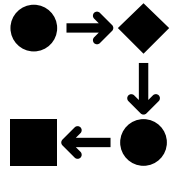
- Cela limite en effet beaucoup la réutilisabilité du code.

1. Généralités algorithmiques.

Les deux types de sous-programmes : procédures et fonctions

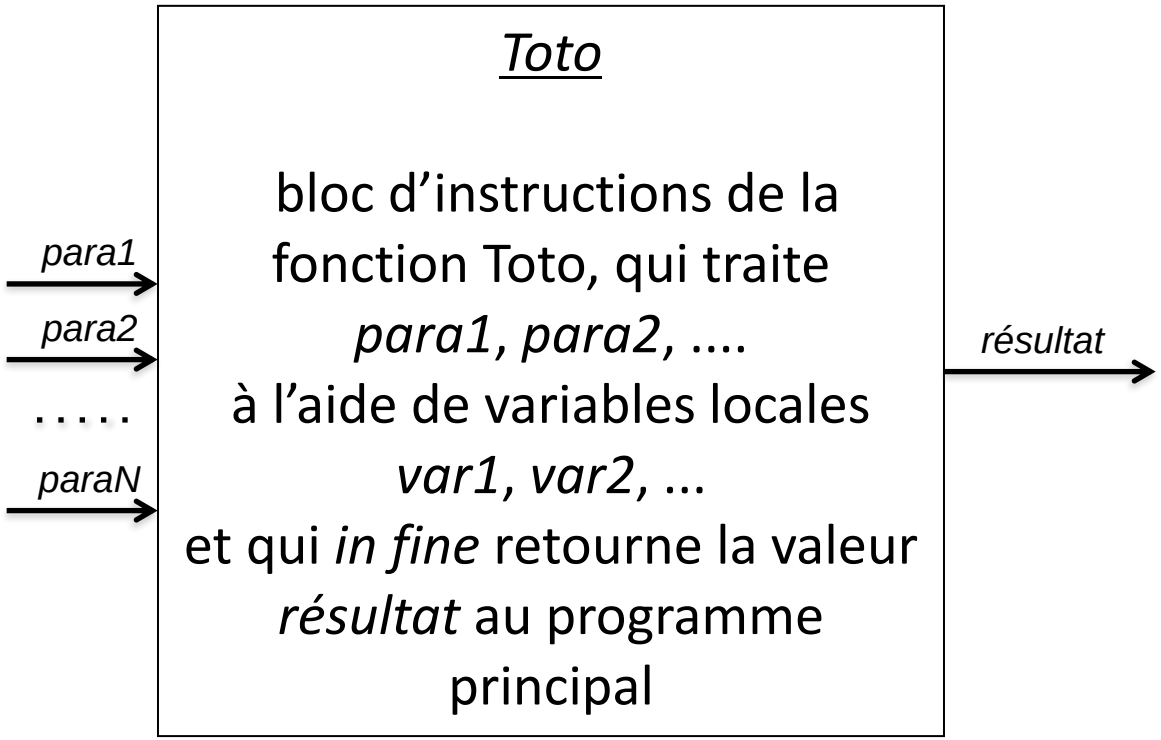
- Les procédures modifient l'état du programme sans retourner de résultat. Elles effectuent des effets de bord.

- Les fonctions retournent un résultat au programme principal.



1. Généralités algorithmiques.

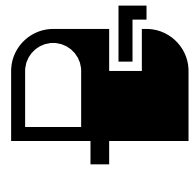
Fonction : visualisation



1. Généralités algorithmiques.

Les deux types de sous-programmes : remarque

- Tous les langages de programmation ne distinguent pas nommément ces deux types de sous–programmes.
- En Python, en C et en C++, on ne manipule ainsi a priori que des fonctions. Bien qu'en pratique la distinction se fasse.
- En Pascal ou PL/SQL en revanche, les deux types sont clairement séparés.



1. Généralités algorithmiques.

2. Les sous-programmes en Python.

2. Les sous-programmes en Python.

Syntaxe générale pour déclarer un sous-programme

- Procédure :

```
def myProcedure(para1, para2, ..., paraN):
    bloc d'instructions de la procédure
```

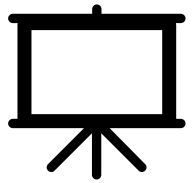
- Fonction :

```
def myFunction(para1, para2, ..., paraN):
    bloc d'instructions de la fonction
    return value
```

2. Les sous-programmes en Python.

Utilisation d'un sous-programme

- On l'appelle par son nom en lui passant autant de paramètres qu'il en possède.
- Ces paramètres peuvent être des variables ou des valeurs explicites.
- Dans le cas d'une fonction, on prend garde de ne pas perdre la valeur retournée en effectuant par exemple une affectation ou un affichage.



2. Les sous-programmes en Python.

Procédure : exemple 1

```
def rectangle(x, y):
    print("Périmètre :", 2*(x+y), ", Aire :", x*y)

rectangle(5, 2)
longueur, largeur = 6, 3
rectangle(longueur, largeur))
```

```
Périmètre : 14 , Aire : 10
Périmètre : 18 , Aire : 18
```

2. Les sous-programmes en Python.

Procédure : exemple 2

```
from math import sqrt

def afficheNombreDor():
    print(round((1+sqrt(5))/2, 2))

afficheNombreDor()
```

2. Les sous-programmes en Python.

Fonction : exemple 1

```
def cube(x):
    return x*x*x

print(cube(5))
```

2. Les sous-programmes en Python.

Fonction : remarques

- Une fonction peut retourner plusieurs valeurs, on les séparera dans ce cas par des virgules.

- Une fonction peut contenir plusieurs fois la commande “return”, mais elle cesse son fonctionnement dès qu’elle en rencontre une.

2. Les sous-programmes en Python.

Fonction : exemple 2

```
def calculMiniMaxi(x, y):
    if x < y:
        return x, y
    else:
        return y, x

a, b = 5, -2
mini, maxi = calculMiniMaxi(a, cube(b))
print("Minimum :", mini, ", Maximum :", maxi)
```

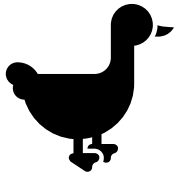
```
Minimum : -8 , Maximum : 5
```

2. Les sous-programmes en Python.

Duck Typing : principe

- En Python on ne précise pas les types attendus des paramètres des sous-programmes.

- Cela implique que l'on peut utiliser un sous-programme avec des paramètres de n'importe quel type, à la condition que les opérations du sous-programme soient compatibles avec les types des paramètres.



2. Les sous-programmes en Python.

Duck Typing : exemple

```
def addition(x, y):
    return x + y

print(addition(666, 1))
a, b = "Brown ", "Sugar"
print(addition(a, b))
```

```
667
Brown Sugar
```

2. Les sous-programmes en Python.

Valeurs par défaut des paramètres : principe

- On peut donner aux paramètres des valeurs par défaut, qui seront utilisées si lors de l'appel on n'en précise pas d'autres.

- Cela peut ne concerner que certains paramètres. Dans ce cas-là, ce seront les plus à droite dans la liste des paramètres du sous-programme.

2. Les sous-programmes en Python.

Valeurs par défaut des paramètres : exemple

```
def rectangle(x, y=1):
    print("Périmètre :", 2*(x+y), ", Aire :", x*y)

rectangle(2)
rectangle(7, 5)
```

```
Périmètre : 6 , Aire : 2
Périmètre : 24 , Aire : 35
```

2. Les sous-programmes en Python.

Paramètres immuables : principe

- Les paramètres de type “int”, “bool”, “float”, “complex” et “str” sont immuables.

- Cela signifie en particulier que si l’on passe une variable de l’un de ces types comme paramètre à un sous-programme, celui-ci ne pourra pas en modifier sa valeur.

2. Les sous-programmes en Python.

Paramètres immuables : exemple

```
def doubler1(x):
    x *= 2

y = 3
print("valeur de y avant :", y)
doubler1(y)
print("valeur de y après :", y)
```

```
valeur de y avant : 3
valeur de y après : 3
```

```
def doubler2(x):
    return 2*x

z = 3
print("valeur de z avant :", z)
z = doubler2(z)
print("valeur de z après :", z)
```

```
valeur de z avant : 3
valeur de z après : 6
```

2. Les sous-programmes en Python.

Variables locales : exemple

```
def sommeEntiers(n):
    somme = 0
    for i in range(n+1):
        somme += i
    return somme
```

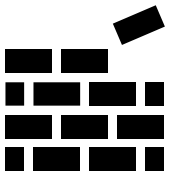
- Les variables ‘somme’ et ‘i’ ne servent qu’au bon déroulement de la fonction et ne seront d’ailleurs pas accessible en dehors de celle-ci.

2. Les sous-programmes en Python.

Module : principe

- Fichier d’extension “.py“ contenant des sous-programmes (regroupés si possible de façon cohérente).

- Pour utiliser ces sous-programmes dans un autre projet on devra importer le module.



2. Les sous-programmes en Python.

Import d'un module : trois possibilités

- Intégralité du module :

```
from myModule import *

import myModule
```

- Un sous-programme en particulier :

```
from myModule import mySubroutine
```

2. Les sous-programmes en Python.

Import d'un module : remarques

- Ces modules peuvent être des librairies développées par des tiers telles que "math", "Tkinter", etc. ou des librairies "maison" conçues par nous-mêmes.

- La syntaxe "`import myModule`" nécessite de rappeler le nom du module avant d'utiliser un sous-programme avec une syntaxe de la forme "`myModule.mySubroutine()`".

- Elle est donc à privilégier dans le cas où plusieurs modules différents pourraient contenir des sous-programmes portant le même nom.

2. Les sous-programmes en Python.

Import d'un module : exemples

```
from math import *
print(sqrt(2))
```

```
1.4142135623730951
```

```
import math
print(math.sqrt(5))
```

```
2.23606797749979
```

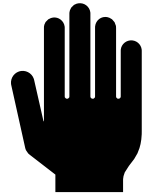
```
from math import pi
print(pi)
```

```
3.141592653589793
```

2. Les sous-programmes en Python.

Code non exécuté lors de l'import d'un module

```
if __name__ == "__main__":
    code non exécuté lors d'un import
    mais exécuté uniquement lorsque l'on
    exécute directement le script
```



2. Les sous-programmes en Python.