FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

DÉPARTEMENT DE MATHÉMATIQUES ET INFORMATIQUE

Organisation du code Python (fonctions, modules)

Master : MNSA

Pr: Youssef Ouassit

Les fonctions

Plan

- Définition
- Passage d’arguments aux fonctions
- Retourner une valeur depuis une fonction
- Valeur par défaut d’un argument
- Arguments à longueur variable
- Portée des variables
- La fonction lambda

Introduction

Exemple 1: Ecrire un programme qui calcule C_n^p

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))
# calculer n!
fn = 1
for i in range(1, n+1):
    fn = fn * i
# calculer p!
fp = 1
for i in range(1, p+1):
    fp = fp * i
# calculer (n-p)!
fnp = 1
for i in range(1, n-p+1):
    fnp = fnp * i

# calculer la combinaison
C = fn // (fp*fnp)
print(C)
```

Les fonctions

Définition

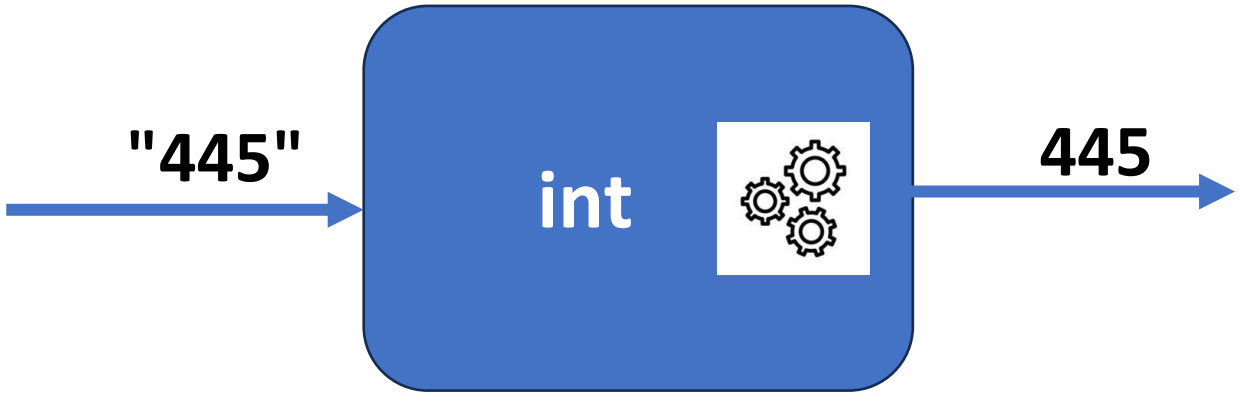
Une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle peut prendre des paramètres (ou arguments) en entrée, effectuer des opérations sur ces données et renvoyer un résultat.

Définition

Fonctions intégrées : **sqrt**

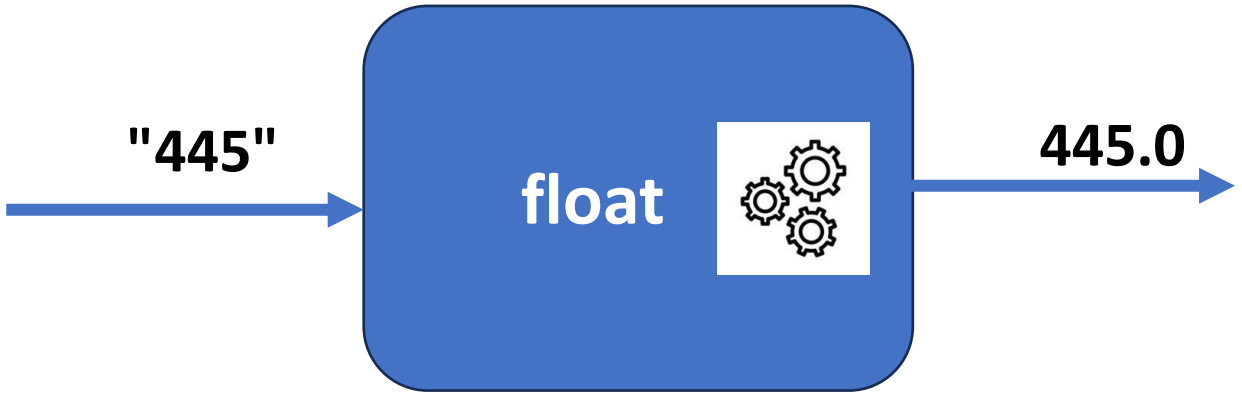
Définition

Fonctions intégrées : **int**



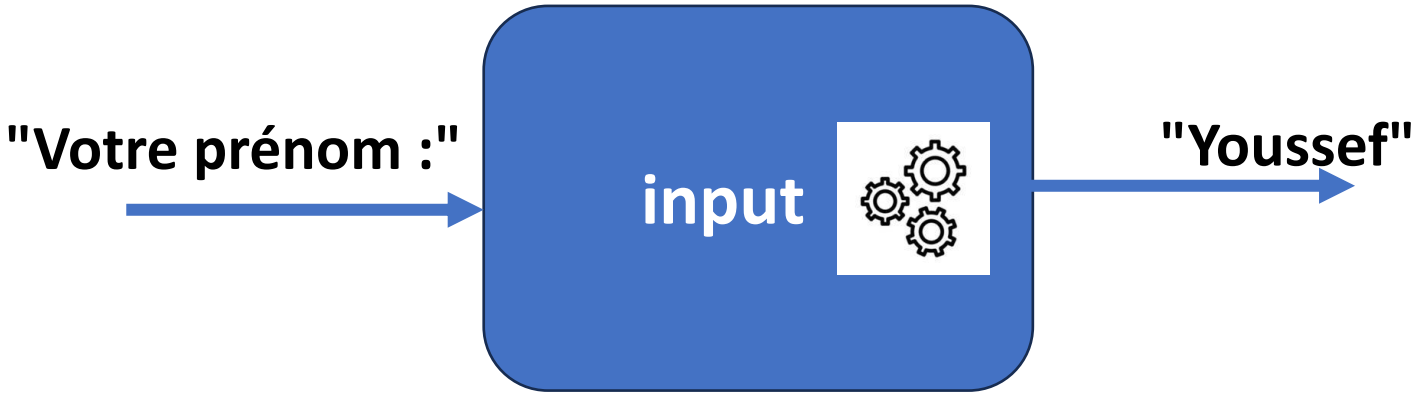
Définition

Fonctions intégrées : float



Définition

Fonctions intégrées : **input**



Définition

Une boîte noir :

Dans le contexte de la programmation, une fonction peut être considérée comme une boîte noire si :

- **Entrées** : Tu fournis des arguments à la fonction (entrées).
- **Sorties** : La fonction renvoie un résultat (sortie).
- **Implémentation cachée** : Tu n'as pas besoin de comprendre comment la fonction réalise ses calculs internes pour l'utiliser. Tu te concentres uniquement sur ce qu'elle fait et comment l'utiliser.

Les fonctions

Syntaxe

En Python, une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle peut prendre des paramètres (ou arguments) en entrée, effectuer des opérations sur ces données et renvoyer un résultat. Voici comment définir une fonction :

```
def nomFonction( argument1, argument2, …) :

    instructions

    [ return résultat ]
```

Les fonctions

Syntaxe

Une fonction qui retourne le max de deux nombres :


```
def nomFonction( argument1, argument2, …) :

        instructions

        [ return résultat ]
```

```
def max( a, b) :
        if a>b :
                mx = a
        else :
                mx = b
        return mx
```

Les fonctions

Syntaxe

Une fonction qui retourne le max de deux nombres :


```
def nomFonction( argument1, argument2, …) :

        instructions

        [ return résultat ]
```

```
def max( a, b) :
    if a>b :
        return a
    else :
        return b
```

Les fonctions

Syntaxe

Une fonction qui retourne le max de deux nombres :


```
def nomFonction( argument1, argument2, …) :

        instructions

        [ return résultat ]
```

```
def max( a, b) :
    if a>b :
        return a
    return b
```

Appel d'une fonction

Une fonction peut être appelé par un programme ou une autre fonction, il suffit d'écrire le nom de la fonction et donner une valeur à chaque argument de la fonction.

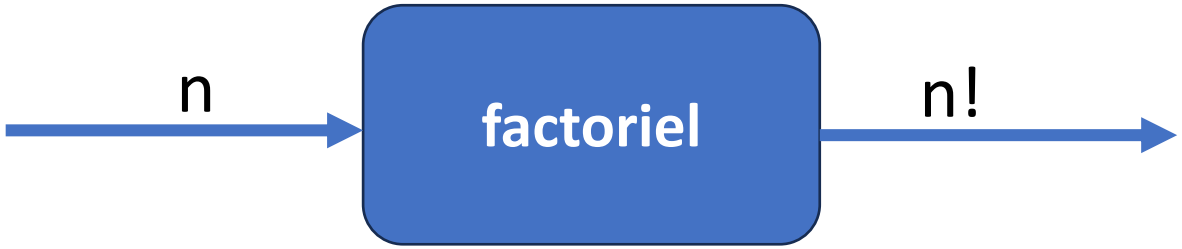
Syntaxe : **variable = nomFonction(Valeur1, Valeur2, …)**

Exemples :

```
x = max(10, 79)
y = max(39, 5)
```

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p




```
def factoriel(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f
```

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))
# calculer n!

fn = factoriel(n)

# calculer p!

fp = factoriel(p)

# calculer (n-p)!

fnp = factoriel(n-p)


# calculer la combinaison
C = fn // (fp*fnp)
print(C)
```

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))

fn = factoriel(n)
fp = factoriel(p)
fnp = factoriel(n-p)

# calculer la combinaison
C = fn // (fp*fnp)
print(C)
```

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p

```
def factoriel(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f

def combinaison(n, p):
    fn = factoriel(n)
    fp = factoriel(p)
    fnp = factoriel(n-p)
    return fn // (fp*fnp)
```

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))

C = combinaison(n, p)

print(C)
```

Arguments optionnels

Définition :

Python permet aux arguments de fonction d'avoir une valeur par défaut, si la fonction est appelée sans l'argument il a la valeur par défaut. De plus, les arguments peuvent être donnés dans n'importe quel ordre en utilisant les arguments nommés.

Arguments optionnels

Syntaxe :

```
def nomFonction( arg1, arg2, arg3=valeur1, arg4=valeur2) :
    instructions
    [ return résultat ]
```

Arguments optionnels

Exemple 2:

```
def montantTTC(puht , qte, tva) :
      mht = puht * qte
      mtva = mht * tva
      mttc = mht + mtva
      return mttc
```

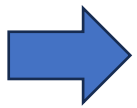
```
M1 = montantTTC(5000 , 10,  0.2)
M2 = montantTTC(7000 , 5  ,  0.2)
M3 = montantTTC(3000 , 10,  0.2)
M4 = montantTTC(100 , 10  ,  0.07)
```

Arguments optionnels

Exemple 2:

```
def montantTTC(puht , qte, tva=0.2) :
    mht = puht * qte
    mtva = mht * tva
    mttc = mht + mtva
    return mttc
```

M1 = montantTTC(5000 , 10, 0.2)
M2 = montantTTC(7000 , 5 , 0.2)
M3 = montantTTC(3000 , 10, 0.2)
M4 = montantTTC(100 , 10 , 0.07)



M1 = montantTTC(5000 , 10)
M2 = montantTTC(7000 , 5)
M3 = montantTTC(3000 , 10)
M4 = montantTTC(100 , 10 , 0.07)

Arguments optionnels

Exemple 2:

```
def montantTTC(puht , qte=10, tva=0.2) :
    mht = puht * qte
    mtva = mht * tva
    mttc = mht + mtva
    return mttc
```

M1 = montantTTC(5000)
M2 = montantTTC(7000 , 5)
~~M4 = montantTTC(100 , 0.07)~~
M4 = montantTTC(100 , tva = 0.07)
M5 = montantTTC(100 , tva = 0.07, qte=15)

Signatures des fonctions

Définition :

La signature d'une fonction en Python désigne la manière dont la fonction est définie, y compris le nom de la fonction, ses paramètres (et leurs types si des annotations de type sont utilisées) ainsi que le type de valeur retournée.

Exercice

Quelle la signature des fonctions :

sin	**sin(x:float) -> float**
input	**input(prompt:str = '') -> str**
range	**range(start:int=0, stop, step:int = 1) -> range**
print	**???????????**

Arguments de longueur variable

Définition :

Python permet aux fonctions d'avoir un nombre d'arguments avec une longueur indéfinie.

Arguments de longueur variable

Exemple 3:

```
def max(a , b) :
      if a>b :
            return a
      else:
            return b
```

```
def max(*a) :
      mx = a[0]
      for i in range(1, len(a)):
            if  a[i] > mx :
                  mx = a[i]
      return mx
```

Exercices

Exercice : Donner le prototype de la fonction print

print(*messages, end='\n', sep=' ') ➔ None

Les fonctions lambda

Les fonctions lambda en Python sont des fonctions anonymes, c'est-à-dire des fonctions sans nom, qui peuvent avoir un nombre quelconque d'arguments mais ne peuvent contenir qu'une seule expression. Elles sont souvent utilisées pour des opérations simples et courtes.

La syntaxe : lambda arguments : expression

Les fonctions lambda

Exemples:

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

Equivalent à :

```
f = lambda x : x**2
```

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

Equivalent à :

```
g = lambda x,y : x+y
```

```
def h(x):
    if x!=0 :
        return 1/x
    return x
```

Equivalent à :

```
h = lambda x : x!=0 if 1/x else x
```

Porté de variables

La « portée » d'une variable désigne l'espace du script dans laquelle elle va être accessible et visible.

On distingue deux types de portée de variables :

1. **Variable locale**

2. **Variable globale**

Porté de variables

1. Variable locale:

Est une variable déclarée à l'intérieur d'une fonction. Elle n'est accessible qu'à l'intérieur de la fonction dans laquelle elle a été déclaré.

```
def ma_fonction():
        x = 4
        print('ma variable vaut ', x)

ma_fonction()

print(x)
```

Erreur : x est non définie

Porté de variables

2. Variable globale:

Est une variable déclarée en dehors des fonctions. Elle visible et utilisable dans toute les fonctions (**mais la fonction ne peut le modifier**).

```
total = 10
def ma_fonction():
    x = total + 4
    print('ma variable vaut ', x)

ma_fonction()
print(total)
```

Porté de variables

2. Variable globale:

Est une variable déclarée en dehors des fonctions. Elle visible et utilisable dans toute les fonctions (**mais la fonction ne peut le modifier**).

```
total = 10
def ma_fonction():
    total = 100
    print('ma variable vaut ‘, total)

ma_fonction()
print(total)
```

```
total = 10
def ma_fonction():
    global total
    total = 100
    print('ma variable vaut ‘, total)

ma_fonction()
print(total)
```

Porté de variables

Résumé de l'ordre de recherche:

L'ordre de recherche des variables en Python est donc :

1. **Locale** : Variables définies dans la fonction actuelle.
2. **Englobante** : Variables définies dans la fonction englobante (si la fonction est imbriquée).
3. **Globale** : Variables définies au niveau du module.

Les fonctions récursives

Par : Youssef Ouassit

Les fonctions récursives

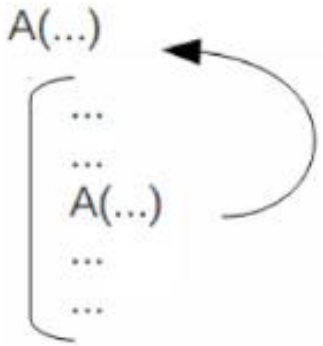
Plan

- Définition
- Evolution d’un appel récursif
- Types de récursivité
  - Terminale
  - Non terminale

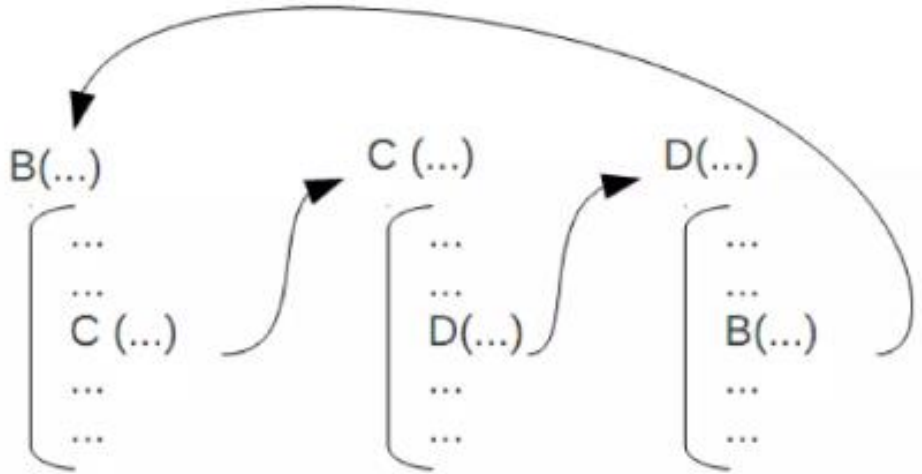
Les fonctions récursives

Définition

- Une fonction est dite récursive lorsqu'elle est auto référente, c-à-d elle fait référence à elle-même dans sa définition.
- Autrement dit, elle est définit en fonction d'elle-même, directement ou indirectement.



Récursivité directe



Récursivité indirecte

Les fonctions récursives

Exemple

- Calculer n!

```
def factoriel(n):
    f = 1
    for i in range(2, n+1):
        f = f * i
    return f
```

Version Itérative

Les fonctions récursives

Exemple

- Calculer n!

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

factoriel(n) = n* factoriel(n-1)
```

Les fonctions récursives

Exemple

- Calculer n!

```
def factoriel(n):
    return n*factoriel(n-1)
```


Appel récursif

Version Récursive

Evolution d'un appel récursif

L'exécution d'un appel récursif passe par deux phases, la phase de descente et la phase de la remontée :

- Dans la phase de descente, chaque appel récursif fait à son tour un appel récursif.

```
def factoriel(n):
    return n*factoriel(n-1)
```

factoriel(4) ← 4 * factoriel(3)

factoriel(3) ← 3 * factoriel(2)

factoriel(2) ← 2 * factoriel(1)

Phase de descente

factoriel(1) ← 1 * factoriel(0)

factoriel(0) ← 0 * factoriel(-1)

Les fonctions récursives

Evolution d’un appel récursif

Puisqu'un algorithme récursif s'appelle lui-même, il est impératif qu'on prévoit une condition d'arrêt à la récursion, sinon le programme ne s'arrête jamais!

```
def factoriel(n):
    if n==0 :
        return 1
    return n*factoriel(n-1)
```

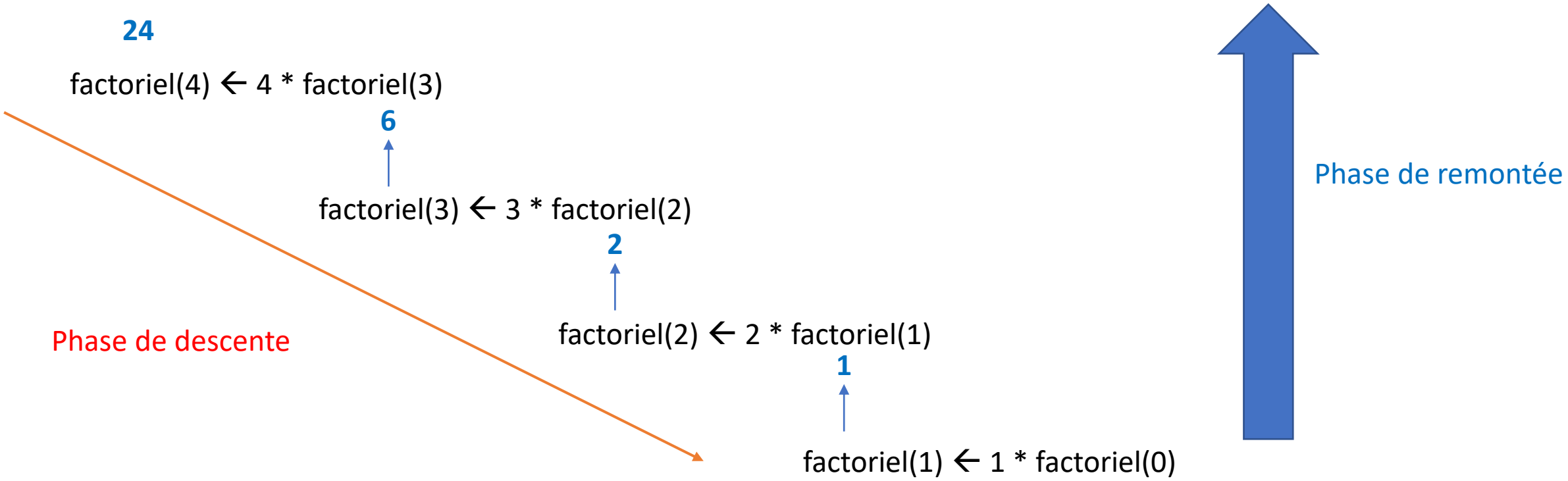
Condition d’arrêt

Appel récursif

Evolution d'un appel récursif

L'exécution d'un appel récursif passe par deux phases, la phase de descente et la phase de la remontée :

- En arrivant à la condition terminale, on commence la phase de la remontée qui se poursuit jusqu'à ce que l'appel initial soit terminé, ce qui termine le processus récursif.



Exemple

Une fonction qui calcule le reste de la division euclidienne de a sur b :

a mod b = (a-b) mod b

```
def Reste(a, b):
    while a>=b :
        a = a - b
    return a
```

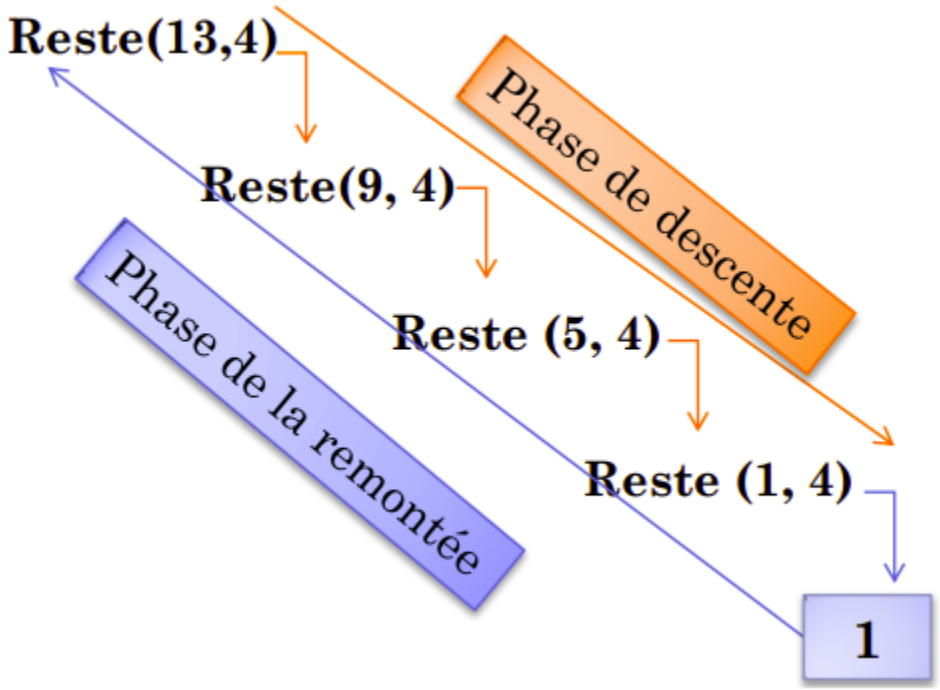
Version itérative

```
def Reste(a, b):
    if a < b :
        return a
    return Reste(a-b, b)
```

Version récursive

Récursivité Terminale est Non terminale

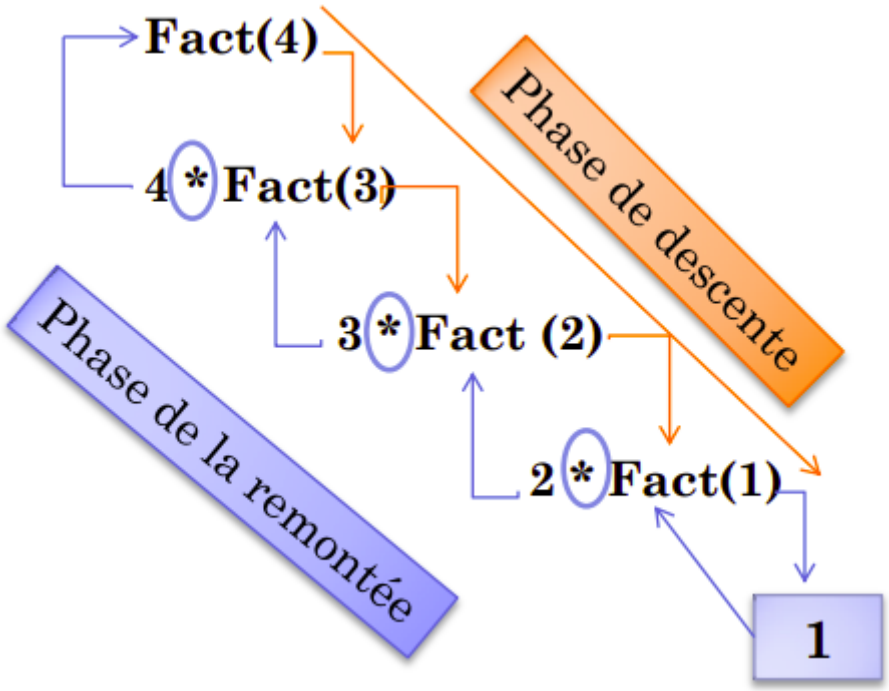
- Une fonction récursive est dit terminale si aucun traitement n'est effectué à la remontée d'un appel récursif (sauf le retour de la fonction).




```
def Reste(a, b):
    if a < b :
        return a
    return Reste(a-b, b)
```

Récursivité Terminale est Non terminale

- Une fonction récursive est dite non terminale si le résultat de l'appel récursif est utilisé pour réaliser un traitement (en plus du retour de la fonction).




```
def Fact(n):
    if n==0 :
        return 1
    return n*Fact(n-1)
```

Récursivité Terminale est Non terminale

- PARFOIS, il est possible de transformer une fonction récursive non terminale à une fonction récursive terminale.

```
def factoriel(n):
    if n==0 :
        return 1
    return n*factoriel(n-1)
```

```
def factoriel(n, resultat):
    if n==1 :
        return resultat
    return factoriel(n-1,n*resultat)
```

NB : la fonction doit être appelée en mettant resultat à 1

Récursivité Terminale est Non terminale

- PARFOIS, il est possible de transformer une fonction récursive non terminale à une fonction récursive terminale.