

Initiation à l'algorithmique

— procédures et fonctions —

1. Spécification et implémentation

Jacques TISSEAU

ÉCOLE NATIONALE D'INGÉNIEURS DE BREST

Technopôle Brest-Iroise

CS 73862 - 29238 Brest cedex 3 - France

enib©2009

Problème

Comment réutiliser un algorithme existant sans avoir à le réécrire ?

Problème

Comment réutiliser un algorithme existant sans avoir à le réécrire ?

```
>>> n = 3
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
6
```

Problème

Comment réutiliser un algorithme existant sans avoir à le réécrire ?

```
>>> n = 3
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
6
```

```
>>> n = 5
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
120
```

Problème

Comment réutiliser un algorithme existant sans avoir à le réécrire ?

```
>>> n = 3
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
6
```

```
>>> n = 5
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
120
```

Élément de réponse

Encapsuler le code dans des fonctions ou des procédures.

Problème

Comment réutiliser un algorithme existant sans avoir à le réécrire ?

```
>>> n = 3
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
6
```

```
>>> n = 5
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
120
```

Élément de réponse

Encapsuler le code dans des fonctions ou des procédures.

```
>>> factorielle(3)
6
```

Problème

Comment réutiliser un algorithme existant sans avoir à le réécrire ?

```
>>> n = 3
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
6
```

```
>>> n = 5
>>> f = 1
>>> for i in range(1,n+1) :
...     f = f*i
...
>>> f
120
```

Élément de réponse

Encapsuler le code dans des fonctions ou des procédures.

```
>>> factorielle(3)
6
```

```
>>> factorielle(5)
120
```

Problème

Comment structurer un algorithme pour le rendre plus compréhensible ?

Problème

Comment structurer un algorithme pour le rendre plus compréhensible ?

```
ieee_code = []
k_exponent = 8
k_significand = 23
k_ieee = 32
bias = code(127,2,k_exponent)
x_int = int(abs(x))
x_frac = abs(x) - x_int
expo_2 = 0
for i in range(k_ieee) : append(ieee_code,0)

# calcul du signe
sign = int(x < 0)

# calcul de la mantisse
i = 0
significand = []
while (x_int != 0) and (i < k_significand) :
    insert(significand,0,x_int%2)
    x_int = x_int/2
    i = i + 1
```

```
if len(significand) > 0 and significand[0] == 1 :
    del significand[0]
    expo_2 = len(significand)
i = len(significand)
while (x_frac != 0) and (i < k_significand) :
    x_frac = x_frac * 2
    x_int = int(x_frac)
    x_frac = x_frac - x_int
    if (x_int == 0) and (i == 0) :
        expo_2 = expo_2 - 1
    else :
        append(significand,x_int)
        i = i + 1
```

Problème

Comment structurer un algorithme pour le rendre plus compréhensible ?

```
ieee_code = []
k_exponent = 8
k_significand = 23
k_ieee = 32
bias = code(127,2,k_exponent)
x_int = int(abs(x))
x_frac = abs(x) - x_int
expo_2 = 0
for i in range(k_ieee) : append(ieee_code,0)

# calcul du signe
sign = int(x < 0)

# calcul de la mantisse
i = 0
significand = []
while (x_int != 0) and (i < k_significand) :
    insert(significand,0,x_int/2)
    x_int = x_int/2
    i = i + 1
```

```
if len(significand) > 0 and significand[0] == 1 :
    del significand[0]
    expo_2 = len(significand)
i = len(significand)
while (x_frac != 0) and (i < k_significand) :
    x_frac = x_frac * 2
    x_int = int(x_frac)
    x_frac = x_frac - x_int
    if (x_int == 0) and (i == 0) :
        expo_2 = expo_2 - 1
    else :
        append(significand,x_int)
        i = i + 1
```

et quelques 20 lignes plus loin...

```
ieee_code[0] = sign
ieee_code[1 :9] = exponent
ieee_code[9 :32] = significand
```

Élément de réponse

Utiliser des fonctions et des procédures.

Élément de réponse

Utiliser des fonctions et des procédures.

```
# calcul du signe
sign = int(x < 0)

# calcul de la mantisse
significand, expo_2 = mantisse(x)

# calcul de l'exposant
exponent = exposant(expo_2, 127)

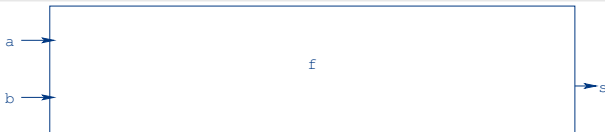
# code IEEE 754
ieee_code[0] = sign
ieee_code[1 :9] = exponent
ieee_code[9 :32] = significand
```

Structuration

Les fonctions et les procédures permettent de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés eux-mêmes en fragments plus petits, et ainsi de suite.

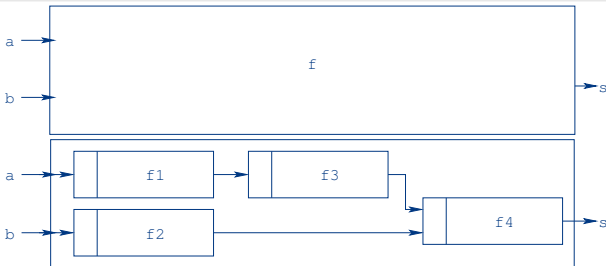
Structuration

Les fonctions et les procédures permettent de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés eux-mêmes en fragments plus petits, et ainsi de suite.



Structuration

Les fonctions et les procédures permettent de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés eux-mêmes en fragments plus petits, et ainsi de suite.



Fonctions

Une fonction est une suite ordonnée d'instructions qui *retourne* une valeur (bloc d'instructions nommé et paramétré).

Fonctions

Une fonction est une suite ordonnée d'instructions qui *retourne* une valeur (bloc d'instructions nommé et paramétré).

Fonction \equiv expression

Une fonction joue le rôle d'une expression.

Fonctions

Une fonction est une suite ordonnée d'instructions qui *retourne* une valeur (bloc d'instructions nommé et paramétré).

Fonction \equiv expression

Une fonction joue le rôle d'une expression.
Elle enrichit le jeu des expressions possibles.

Fonctions

Une fonction est une suite ordonnée d'instructions qui *retourne* une valeur (bloc d'instructions nommé et paramétré).

Fonction \equiv expression

Une fonction joue le rôle d'une expression.
Elle enrichit le jeu des expressions possibles.

Exemple

`y = sin(x)` renvoie la valeur du sinus de x

nom : sin

paramètres : x :float \rightarrow sin(x) :float

Procédures

Une procédure est une suite ordonnée d'instructions qui *ne retourne pas* de valeur (bloc d'instructions nommé et paramétré).

Procédures

Une procédure est une suite ordonnée d'instructions qui *ne retourne pas* de valeur (bloc d'instructions nommé et paramétré).

Procédure \equiv instruction

Une procédure joue le rôle d'une instruction.

Procédures

Une procédure est une suite ordonnée d'instructions qui *ne retourne pas* de valeur (bloc d'instructions nommé et paramétré).

Procédure \equiv instruction

Une procédure joue le rôle d'une instruction.
Elle enrichit le jeu des instructions existantes.

Procédures

Une procédure est une suite ordonnée d'instructions qui *ne retourne pas* de valeur (bloc d'instructions nommé et paramétré).

Procédure \equiv instruction

Une procédure joue le rôle d'une instruction.
Elle enrichit le jeu des instructions existantes.

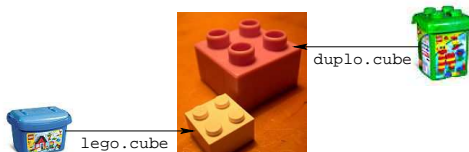
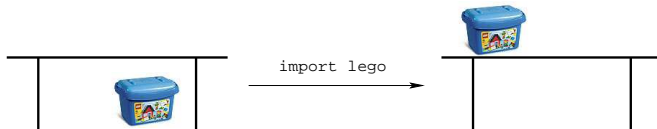
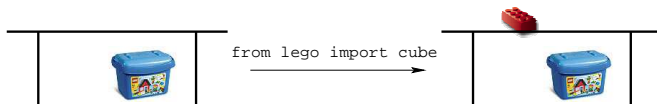
Exemple

```
print(x, y, z)
```

affiche les valeurs de x, y et z

nom : print

paramètres : $x, y, z \rightarrow \square$



Les 6 étapes de définition

Nom : un identificateur suffisamment explicite.

Les 6 étapes de définition

Nom : un identificateur suffisamment explicite.

Paramètres : la liste des paramètres d'entrée-sortie de l'algorithme.

Les 6 étapes de définition

Nom : un identificateur suffisamment explicite.

Paramètres : la liste des paramètres d'entrée-sortie de l'algorithme.

Préconditions : une liste d'expressions booléennes qui précisent les conditions d'application de l'algorithme.

Les 6 étapes de définition

Nom : un identificateur suffisamment explicite.

Paramètres : la liste des paramètres d'entrée-sortie de l'algorithme.

Préconditions : une liste d'expressions booléennes qui précisent les conditions d'application de l'algorithme.

Appel : des exemples d'utilisation de l'algorithme avec les résultats attendus.

Les 6 étapes de définition

Nom : un identificateur suffisamment explicite.

Paramètres : la liste des paramètres d'entrée-sortie de l'algorithme.

Préconditions : une liste d'expressions booléennes qui précisent les conditions d'application de l'algorithme.

Appel : des exemples d'utilisation de l'algorithme avec les résultats attendus.

Description : une phrase qui dit ce que fait l'algorithme.

Les 6 étapes de définition

Nom : un identificateur suffisamment explicite.

Paramètres : la liste des paramètres d'entrée-sortie de l'algorithme.

Préconditions : une liste d'expressions booléennes qui précisent les conditions d'application de l'algorithme.

Appel : des exemples d'utilisation de l'algorithme avec les résultats attendus.

Description : une phrase qui dit ce que fait l'algorithme.

Code : la séquence d'instructions nécessaires à la résolution du problème.

❶ *nom*

```
def factorielle() :  
    return
```

❶ *nom*

```
>>> factorielle()  
>>>
```

1 *nom*

```
def factorielle() :  
    return
```

1 *nom*

```
>>> factorielle()  
>>>
```

2 *paramètres d'entrée-sortie*

```
def factorielle(n) :  
    f = 1  
    return f
```

2 *paramètres d'entrée-sortie*

```
>>> factorielle(5)  
1  
>>> factorielle(-5)  
1  
>>> factorielle('toto')  
1
```



3 *préconditions*

```
def factorielle(n)
    assert type(n) is int
    assert n >= 0
    f = 1
    return f
```

3 *préconditions*

```
>>> factorielle(5)
1
>>> factorielle(-5)
AssertionError :
    assert n >= 0
>>> factorielle('toto')
AssertionError :
    assert type(n) is int
```

3 préconditions

```
def factorielle(n)
    assert type(n) is int
    assert n >= 0
    f = 1
    return f
```

3 préconditions

```
>>> factorielle(5)
1
>>> factorielle(-5)
AssertionError :
    assert n >= 0
>>> factorielle('toto')
AssertionError :
    assert type(n) is int
```



📌 *jeu de tests*

```
def factorielle(n) :  
    """  
    >>> for i in range(8) :  
        ...  
    print(factorielle(i),end=' ')  
    1 1 2 6 24 120 720 5040  
    """  
    assert type(n) is int  
    assert n >= 0  
    f = 1  
    return f
```

📌 *jeu de tests*

```
>>> for i in range(8) :  
    ...  
    print(factorielle(i),end=''  
    ')  
    ...  
    1 1 1 1 1 1 1 1
```

📌 *jeu de tests*

```
def factorielle(n) :  
    """  
    >>> for i in range(8) :  
        ...  
    print(factorielle(i),end=' ')  
    1 1 2 6 24 120 720 5040  
    """  
    assert type(n) is int  
    assert n >= 0  
    f = 1  
    return f
```

📌 *jeu de tests*

```
>>> for i in range(8) :  
    ...  
    print(factorielle(i),end=''  
    ')  
    ...  
    1 1 1 1 1 1 1 1
```

5 *description*

```
def factorielle(n) :  
    """  
    f = n!  
    >>> for i in range(8) :  
        ...  
print(factorielle(i),end=' ')  
1 1 2 6 24 120 720 5040  
    """  
    assert type(n) is int  
    assert n >= 0  
    f = 1  
    return f
```

5 *description*

```
>>> for i in range(8) :  
    ...  
print(factorielle(i),end=''  
)  
...  
1 1 1 1 1 1 1 1
```

5 description

```
def factorielle(n) :  
    """  
    f = n!  
    >>> for i in range(8) :  
        ...  
print(factorielle(i),end=' ')  
1 1 2 6 24 120 720 5040  
    """  
    assert type(n) is int  
    assert n >= 0  
    f = 1  
    return f
```

5 description

```
>>> for i in range(8) :  
    ...  
print(factorielle(i),end=''  
)  
...  
1 1 1 1 1 1 1 1
```

6 implémentation

```
def factorielle(n) :  
    """  
    f = n!  
    >>> for i in range(8) :  
        ...  
print(factorielle(i),end=' ')  
1 1 2 6 24 120 720 5040  
    """  
    assert type(n) is int  
    assert n >= 0  
    f = 1  
    for i in range(1,n+1) :  
        f = f * i  
    return f
```

6 implémentation

```
>>> for i in range(8) :  
    ...  
    print(factorielle(i),end=''  
    ...  
1 1 2 6 24 120 720 5040
```

⑥ implémentation

```
def factorielle(n) :  
    """  
    f = n!  
    >>> for i in range(8) :  
        ...  
print(factorielle(i),end=' ')  
1 1 2 6 24 120 720 5040  
    """  
    assert type(n) is int  
    assert n >= 0  
    f = 1  
    for i in range(1,n+1) :  
        f = f * i  
    return f
```

⑥ implémentation

```
>>> for i in range(8) :  
    ...  
    print(factorielle(i),end=''  
    ...  
1 1 2 6 24 120 720 5040
```

```
1 def factorielle(n):
2     """
3     f = n!
4     >>> for i in range(10):
5         ...     print factorielle(i),
6         1 1 2 6 24 120 720 5040 40320 362880
7     >>> factorielle(15)
8         1307674368000L
9     """
10    assert type(n) is int
11    assert n >= 0
12
13    f = 1
14    for i in range(1,n+1): f = f * i
15
16    return f
```

```
1  def sommeArithmetique(n):
2      """
3      somme s des n premiers entiers
4
5      >>> for n in range(7):
6          ...     print sommeArithmetique(n) == n*(n+1)/2,
7          True True True True True True True
8      """
9      assert type(n) is int
10     assert n >= 0
11     q = q + 1
12     s = n*(n+1)/2
13
14     return s
```

Spécification d'un algorithme

Quoi ?

La spécification décrit la fonction et l'utilisation d'un algorithme (ce que fait l'algorithme).

Spécification d'un algorithme

Quoi ?

La spécification décrit la fonction et l'utilisation d'un algorithme (ce que fait l'algorithme).

L'algorithme est vu comme une boîte noire dont on ne connaît pas le fonctionnement interne.

Spécification d'un algorithme

Quoi ?

La spécification décrit la fonction et l'utilisation d'un algorithme (ce que fait l'algorithme).

L'algorithme est vu comme une boîte noire dont on ne connaît pas le fonctionnement interne.

Implémentation d'un algorithme

Comment ?

L'implémentation décrit le fonctionnement interne de l'algorithme (comment fait l'algorithme).

Spécification d'un algorithme

Quoi ?

La spécification décrit la fonction et l'utilisation d'un algorithme (**ce que fait l'algorithme**).

L'algorithme est vu comme une boîte noire dont on ne connaît pas le fonctionnement interne.

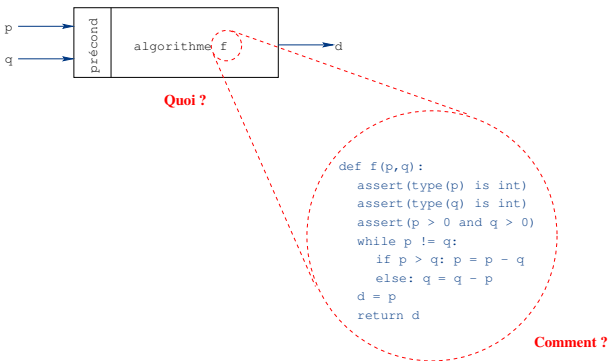
Implémentation d'un algorithme

Comment ?

L'implémentation décrit le fonctionnement interne de l'algorithme (**comment fait l'algorithme**).

L'implémentation précise l'enchaînement des instructions nécessaires à la résolution du problème considéré.

La spécification décrit la fonction et l'utilisation d'un algorithme



L'implémentation décrit le fonctionnement interne de l'algorithme

```
#-----  
def sommeArithmetique(n):  
#-----  
    """  
    somme s des n premiers entiers  
  
    >>> for n in range(7):  
    ...     print sommeArithmetique(n)\  
            == n*(n+1)/2  
    True True True True True True True  
    """  
    assert type(n) is int  
    assert n >= 0  
  
    s = n*(n+1)/2  
  
    return s  
#-----
```

```
#-----  
def sommeArithmetique(n):  
#-----  
    """  
    somme s des n premiers entiers  
  
    >>> for n in range(7):  
    ...     print sommeArithmetique(n)\  
            == n*(n+1)/2  
    True True True True True True True  
    """  
    assert type(n) is int  
    assert n >= 0  
  
    s = 0  
    for i in range(n+1): s = s + i  
  
    return s  
#-----
```


Concepteur

Le concepteur d'un algorithme définit l'interface et l'implémentation de l'algorithme.

Concepteur

Le concepteur d'un algorithme définit l'interface et l'implémentation de l'algorithme.

Utilisateur

L'utilisateur d'un algorithme n'a pas à connaître son implémentation ; seule l'interface de l'algorithme le concerne.

Concepteur

Le concepteur d'un algorithme définit l'interface et l'implémentation de l'algorithme.

Utilisateur

L'utilisateur d'un algorithme n'a pas à connaître son implémentation ; seule l'interface de l'algorithme le concerne.

Selon la spécification de l'algorithme, l'utilisateur **appelle** (utilise) l'algorithme sous forme d'une **procédure** ou d'une **fonction**.

Propriétés d'un algorithme

validité : être conforme aux jeux de tests

Propriétés d'un algorithme

validité : être conforme aux jeux de tests

robustesse : vérifier les préconditions

Propriétés d'un algorithme

validité : être conforme aux jeux de tests

robustesse : vérifier les préconditions

réutilisabilité : être correctement paramétré