Initiation à l'algorithmique

— procédures et fonctions —
1. Définition d'une fonction

Jacques TISSEAU

Enib–Cerv

Remarque (Notes de cours : couverture)

Ce support de cours accompagne le chapitre 3 des notes de cours « Initiation à l'algorithmique ».

— Cours d'Informatique S1 —

Initiation à l'algorithmique

Jacques TISSEAU

Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr

Ces notes de cours accompagnent les enseignements d'informatique du 1er semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (Enib : www.enib.fr). Leur lecture ne dispense en aucun cas d'une présence attentive aux cours ni d'une participation active aux travaux dirigés.

version du 21 octobre 2014

Avec la participation de Romain Bénard, Stéphane Bonneaud, Cédric Buche, Gireg Desmeulles, Céline Jost, Sébastien Kubicki, Eric Maisel, Alexis Nédélec, Marc Parenthoën et Cyril Septseault.

RÉUTILISABILITÉ

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
```

Elément de réponse

Encapsuler le code dans des fonctions ou des procédures.

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

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


Définitions

réutilisabilité aptitude d'un algorithme à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.

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
```

Elément de réponse

Utiliser des fonctions et des procédures.

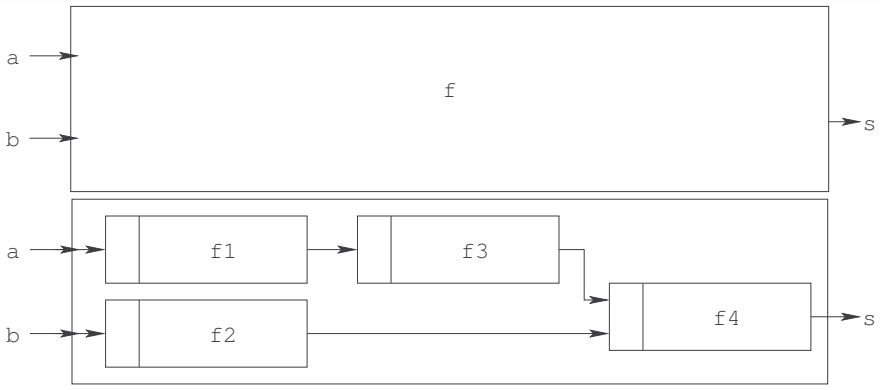
```
# calcul du signe
sign = int(x < 0)

# 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
```


Définitions

encapsulation action de mettre une chose dans une autre

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).

Fonction ≡ 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` → `sin(x):float`

Définitions

fonction bloc d'instructions nommé et paramétré, réalisant une certaine tâche. Elle admet zéro, un ou plusieurs paramètres et renvoie toujours un résultat.

Exemples de fonctions prédéfinies en Python

Function	Result
`abs(x)`	Returns the absolute value of the number `x`.
`chr(i)`	Returns one-character string whose ASCII code is integer `i`.
`help([object])`	Invokes the built-in help system. No argument → interactive help ; if `object` is a string (name of a module, function, class, method, keyword, or documentation topic), a help page is printed on the console ; otherwise a help page on `object` is generated.
`input([prompt])`	Prints `prompt` if given. Reads input and evaluates it.
`len(obj)`	Returns the length (the number of items) of an object (sequence, dictionary).
`ord(c)`	Returns integer ASCII value of `c` (a string of len 1).
`print([s1] [, s2 ]*)`	Writes to `sys.stdout`. Puts spaces between arguments `si`. Puts newline at end unless arguments end with `end=` (ie : `end=' '`).
`range([start,] end [, step])`	Returns list of ints from >= `start` and < `end`. With 1 arg, list from `0..arg-1`. With 2 args, list from `start..end-1`. With 3 args, list from `start` up to `end` by `step`.

PROCÉDURES

Procédures

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

Procédure ≡ 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`

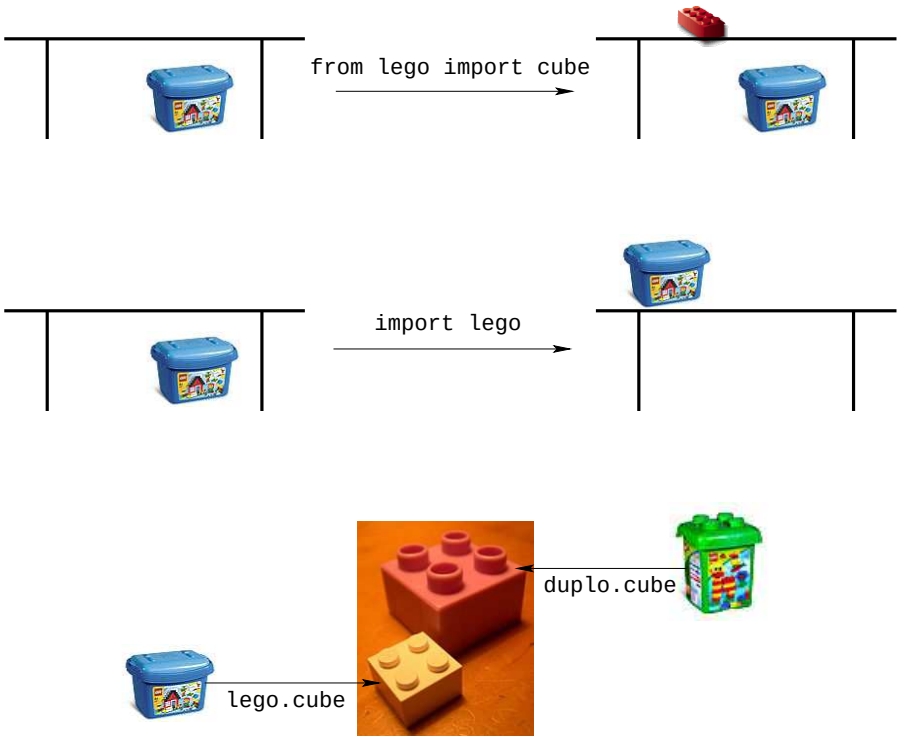
paramètres : `x`, `y`, `z` $\rightarrow \square$

Définitions

procédure bloc d'instructions nommé et paramétré, réalisant une certaine tâche. Elle admet zéro, un ou plusieurs paramètres et ne renvoie pas de résultat.

Remarque

Une fonction en informatique se distingue principalement de la fonction mathématique par le fait qu'en plus de calculer un résultat à partir de paramètres, la fonction informatique peut avoir des « effets de bord » : par exemple afficher un message à l'écran, jouer un son, ou bien piloter une imprimante.
Une fonction qui n'a pas d'effet de bord joue le rôle d'une expression évaluable.
Une fonction qui n'a que des effets de bord est appelée une procédure et joue le rôle d'une instruction.



Remarque

Les procédures et les fonctions intégrées au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment. Les autres fonctions sont regroupées dans des fichiers séparés que l'on appelle des modules.
Les modules sont donc des fichiers qui regroupent des ensembles de fonctions. Souvent on regroupe dans un même module des ensembles de fonctions apparentées que l'on appelle des bibliothèques. Pour pouvoir utiliser ces fonctions, il faut importer le module correspondant.

```
>> from math import sin, pi
>> sin(pi/2)
1.0
```

```
>> import math
>> math.sin(math.pi/2)
1.0
```

Les 6 étapes de définition

1 :	**Nom**	: un identificateur suffisamment explicite
2 :	**Paramètres**	: la liste des paramètres d'entrée-sortie de l'algorithme
3 :	**Préconditions**	: une liste d'expressions booléennes qui précisent les conditions d'application de l'algorithme (conditions sur les paramètres d'entrée)
4 :	**Appel**	: des exemples d'utilisation de l'algorithme avec les résultats attendus (entrées $\rightarrow$ sorties)
5 :	**Description**	: une phrase qui dit ce que fait l'algorithme en explicitant les paramètres d'entrée-sortie
6 :	**Code**	: la séquence d'instructions nécessaires à la résolution du problème

Etapes 1–5 : **Spécification** Etape : 6 : **Implémentation**

Remarque

Pour encapsuler un algorithme dans une fonction, on suivra pas à pas la démarche suivante :

1. *donner un nom explicite à l'algorithme,*
2. *définir les paramètres d'entrée-sortie de l'algorithme,*
3. *préciser les préconditions sur les paramètres d'entrée,*
4. *donner des exemples d'utilisation et les résultats attendus,*
5. *décrire par une phrase ce que fait l'algorithme et dans quelles conditions il le fait,*
6. *encapsuler l'algorithme dans la fonction spécifiée par les 5 points précédents.*

Les 5 premières étapes relèvent de la spécification de l'algorithme et la dernière étape concerne l'encapsulation proprement dite de l'algorithme.

NOM ET ENTRÉES-SORTIES

1. *nom*

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

2. *paramètres d'entrée-sortie*

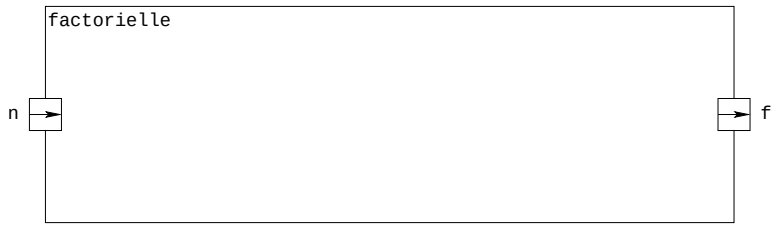
```
def factorielle(n):
    f = 1
    return f
```

1. *nom*

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

2. *paramètres d'entrée-sortie*

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



Définitions

paramètres d'entrée arguments de la fonction qui sont nécessaires pour effectuer le traitement associé à la fonction.

paramètres de sortie résultats retournés par la fonction après avoir effectué le traitement associé à la fonction.

PRÉCONDITIONS

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
```

Définitions

préconditions conditions que doivent impérativement vérifier les paramètres d'entrée de la fonction juste avant son exécution.

postconditions conditions que doivent impérativement vérifier les paramètres de sortie de la fonction juste après son exécution.

invariants conditions que doit impérativement vérifier la fonction tout au long de son exécution.

JEU DE TESTS

4. *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
```

4. *jeu de tests*

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


Remarque

A chaque étape de la spécification, le code de la fonction doit toujours être exécutable même s'il ne donne pas encore le bon résultat.
Le jeu de tests ne sera vérifié qu'une fois l'implémentation correctement définie.

Remarque (test automatique)

En Python, *le jeu de tests peut être automatiquement évalué par l'interpréteur en ajoutant les 3 lignes suivantes en fin de fichier source.*

```
if __name__ == "__main__" :
  import doctest
  doctest.testmod()
```

DESCRIPTION

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
```


Définitions

description phrase qui précise ce que fait la fonction et dans quelles conditions elle le fait.

Remarque

La description est une phrase (chaîne de caractères) qui doit expliciter le rôle des paramètres d'entrée et leurs préconditions, ainsi que toutes autres informations jugées nécessaires par le concepteur de la fonction.
Dans certains cas « difficiles », on pourra préciser une référence bibliographique ou un site Web *où l'utilisateur pourra trouver des compléments sur la fonction et l'algorithme associé.*
La description d'une fonction intègrera donc au moins :

- *un exemple typique d'appel de la fonction,*
- *la signification des paramètres d'entrée-sortie,*
- *les préconditions sur les paramètres d'entrée,*
- *un jeu de tests significatifs.*

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
```

factorielle(n) : TOUT EN UN

```
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
```


TD (Décodage base b → décimal)

La valeur décimale d'un nombre entier codé en base b peut être obtenue par l'algorithme suivant :

```
>>> n = 0
>>> for i in range(len(code)):
...     n = n + code[i]*b**(len(code)-1-i)
...
>>>
```

Spécifier une fonction qui encapsule cet algorithme.

sommeArithmetique(n) : TOUT EN UN

```
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
12     s = n*(n+1)/2
13
14     return s
```


TD (Division entière)

Le calcul du quotient q et du reste r de la division entière $a \div b$ ($a = bq + r$) peut être obtenu par l'algorithme suivant :

```
>>> q,r = 0,a
>>> while r >= b:
...     q = q + 1
...     r = r - b
...
>>>
```

Spécifier une fonction qui encapsule cet algorithme.

SPÉCIFICATION, IMPLÉMENTATION

QUOI ?, COMMENT ?

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.

Définitions

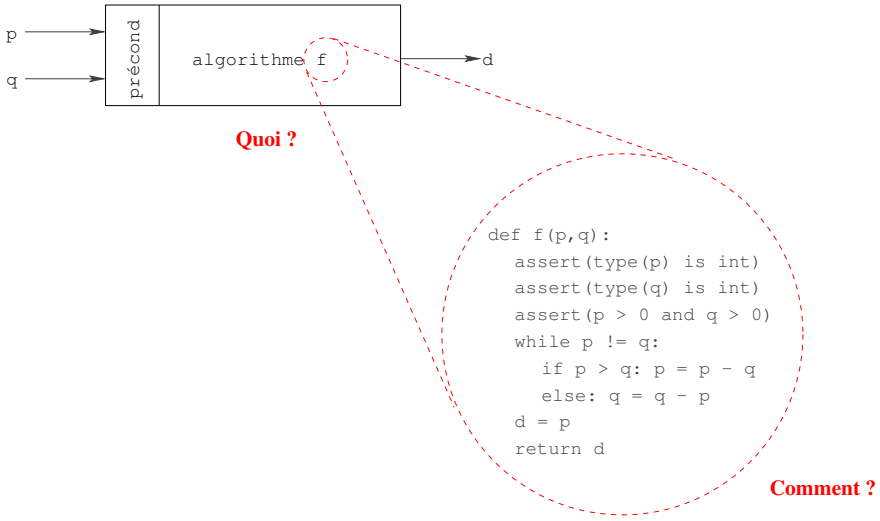
spécification décrit ce que fait l'algorithme et dans quelles conditions il le fait.

implémentation décrit comment fait l'algorithme pour satisfaire sa spécification.

SPÉCIFICATION, IMPLÉMENTATION

QUOI ?, COMMENT ?

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
#--------------------------------------
```


TD (Une spécification, des implémentations)

1. *Proposer deux implémentations du calcul de la somme* $s = \sum_0^n u_k$ *des* n *premiers termes d'une suite géométrique* $u_k = a \cdot b^k$.
2. *Comparer les complexités de ces deux implémentations.*

Indication :

$$s = \sum_{k=0}^{n} ab^k = a \sum_{k=0}^{n} b^k$$

où l'expression $N = (b^0 + b^1 + b^2 + \cdots + b^n)$ *peut être vue comme le nombre* $(111\cdots1)_b$ *en base* b. *Or en base* b, *le nombre* $(b-1)(b^0 + b^1 + b^2 + \cdots + b^n)$ *est le nombre immédiatement inférieur à* b^{n+1}, *soit* $(b-1)N = b^{n+1} - 1$.
Exemple en base $b = 10$: $999_{10} = 9(10^0 + 10^1 + 10^2) = 10^3 - 1$

$$S = \sum_{k=0}^{n} b^k = (b^0 + b^1 + b^2 + \cdots + b^n) = \frac{b^{n+1} - 1}{b - 1}$$

SPÉCIFICATION, IMPLÉMENTATION

CONCEPTEUR VERSUS UTILISATEUR

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
Robustesse : vérifier les préconditions
Réutilisabilité : être correctement paramétré

Définitions

Validité : aptitude à réaliser exactement la tâche pour laquelle il a été conçu.
ie : L'implémentation de la fonction doit être conforme aux jeux de tests.

Robustesse : aptitude à se protéger de conditions anormales d'utilisation.
ie : La fonction doit vérifier impérativement ses préconditions.

Réutilisabilité : aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.
ie : La fonction doit être correctement paramétrée.

Remarque

Une fois la fonction définie (spécifiée et implémentée), il reste à l'utiliser (à l'appeler).