

Initiation à l'algorithmique

— procédures et fonctions —
2. Appel d'une fonction

Jacques TISSEAU

ÉCOLE NATIONALE D'INGÉNIEURS DE BREST
Technopôle Brest-Iroise
CS 73862 - 29238 Brest cedex 3 - France

enib©2009

```
1 def fibonacci(n):
2     """
3     u = fibonacci(n)
4     est le nombre de Fibonacci
5     à l'ordre n si n:int >= 0
6     """
7     assert type(n) is int
8     assert n >= 0
9     u, u1, u2 = 1, 1, 1
10    for i in range(2,n+1):
11        u = u1 + u2
12        u2 = u1
13        u1 = u
14    return u
```

Définition

```
def fibonacci(n):  
    ...  
  
    return u
```

Appel

```
>>> x = ...  
>>> y = fibonacci(x)
```

Définition

```
def fibonacci(n)
    ...
    return u
```

paramètres formels



Appel

```
>>> x = ...
>>> y = fibonacci(x)
```

Définition

```
def fibonacci(n)
    ...
    return u
```

paramètres formels

Appel

```
>>> x = ...
>>> y = fibonacci(x)
```

paramètres effectifs

Définition

```
def fibonacci(n)
    ...
    return u
```

paramètres formels

Appel

```
>>> x = ...
>>> y = fibonacci(x)
```

paramètres effectifs

à l'appel copie des paramètres effectifs dans les paramètres formels ($n = x$)

Définition

```
def fibonacci(n)
    ...
    return u
```

paramètres formels

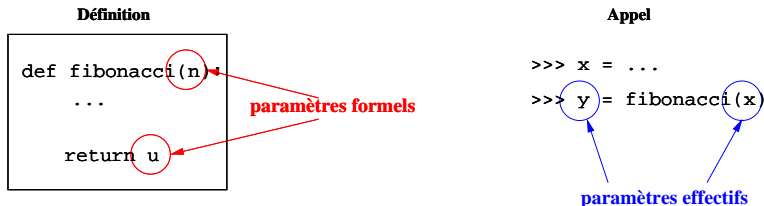
Appel

```
>>> x = ...
>>> y = fibonacci(x)
```

paramètres effectifs

à l'appel copie des paramètres effectifs dans les paramètres formels ($n = x$)

à la sortie copie des paramètres formels dans les paramètres effectifs ($y = u$)



à l'appel copie des paramètres effectifs dans les paramètres formels ($n = x$)

à la sortie copie des paramètres formels dans les paramètres effectifs ($y = u$)

Passage par valeur ce ne sont pas les paramètres effectifs qui sont manipulés par la fonction mais des copies de ces paramètres

Appel :

```
>>> x = 9
```

```
>>> y = fibonacci(x)
```

```
>>> y
```

```
55
```

Appel :

```
>>> x = 9
```

```
>>> y = fibonacci(x)
```

```
>>> y
```

```
55
```

Appel équivalent :

```
>>> x = 9
```

Appel :

```
>>> x = 9
```

```
>>> y = fibonacci(x)
```

```
>>> y
```

```
55
```

Appel équivalent :

```
>>> x = 9
```

```
>>> n = x
```

Appel :

```
>>> x = 9
>>> y = fibonacci(x)
>>> y
55
```

Appel équivalent :

```
>>> x = 9
>>> n = x
>>> u, u1, u2 = 1, 1, 1
>>> for i in range(2,n+1) :
...     u = u1 + u2
...     u2 = u1
...     u1 = u
... 
```

Appel :

```
>>> x = 9
>>> y = fibonacci(x)
>>> y
55
```

Appel équivalent :

```
>>> x = 9
>>> n = x
>>> u, u1, u2 = 1, 1, 1
>>> for i in range(2,n+1) :
...     u = u1 + u2
...     u2 = u1
...     u1 = u
...
>>> tmp = u
```

Appel :

```
>>> x = 9
>>> y = fibonacci(x)
>>> y
55
```

Appel équivalent :

```
>>> x = 9
>>> n = x
>>> u, u1, u2 = 1, 1, 1
>>> for i in range(2,n+1) :
...     u = u1 + u2
...     u2 = u1
...     u1 = u
...
>>> tmp = u
>>> del n, u, u1, u2, i
```

Appel :

```
>>> x = 9
>>> y = fibonacci(x)
>>> y
55
```

Appel équivalent :

```
>>> x = 9
>>> n = x
>>> u, u1, u2 = 1, 1, 1
>>> for i in range(2,n+1) :
...     u = u1 + u2
...     u2 = u1
...     u1 = u
...
>>> tmp = u
>>> del n, u, u1, u2, i
>>> y = tmp
```

Appel :

```
>>> x = 9
>>> y = fibonacci(x)
>>> y
55
```

Appel équivalent :

```
>>> x = 9
>>> n = x
>>> u, u1, u2 = 1, 1, 1
>>> for i in range(2,n+1) :
...     u = u1 + u2
...     u2 = u1
...     u1 = u
...
>>> tmp = u
>>> del n, u, u1, u2, i
>>> y = tmp
>>> del tmp
```

Appel :

```
>>> x = 9
>>> y = fibonacci(x)
>>> y
55
```

Appel équivalent :

```
>>> x = 9
>>> n = x
>>> u, u1, u2 = 1, 1, 1
>>> for i in range(2,n+1) :
...     u = u1 + u2
...     u2 = u1
...     u1 = u
...
>>> tmp = u
>>> del n, u, u1, u2, i
>>> y = tmp
>>> del tmp
>>> y
55
```

```
def f(x) :  
    y = 3  
    x = x + y  
    print('liste :', dir())  
    print('intérieur :', locals())  
    print('extérieur :', globals())  
    return x
```

```
def f(x) :  
    y = 3  
    x = x + y  
    print('liste :', dir())  
    print('intérieur :', locals())  
    print('extérieur :', globals())  
    return x
```

```
>>> y = 6  
>>> f(6)
```

```
def f(x) :  
    y = 3  
    x = x + y  
    print('liste :', dir())  
    print('intérieur :', locals())  
    print('extérieur :', globals())  
    return x
```

```
>>> y = 6  
>>> f(6)  
liste : ['x', 'y']
```

```
def f(x) :  
    y = 3  
    x = x + y  
    print('liste :', dir())  
    print('intérieur :', locals())  
    print('extérieur :', globals())  
    return x
```

```
>>> y = 6
```

```
>>> f(6)
```

```
liste : ['x', 'y']
```

```
intérieur : {'y' : 3, 'x' : 9}
```

```
def f(x) :  
    y = 3  
    x = x + y  
    print('liste :', dir())  
    print('intérieur :', locals())  
    print('extérieur :', globals())  
    return x
```

```
>>> y = 6  
>>> f(6)  
liste : ['x', 'y']  
intérieur : {'y' : 3, 'x' : 9}  
extérieur : {'f' : <function f at 0x822841c>,  
             'y' : 6,  
             '__name__' : '__main__',  
             '__doc__' : None}
```

```
def f(x) :  
    y = 3  
    x = x + y  
    print('liste :', dir())  
    print('intérieur :', locals())  
    print('extérieur :', globals())  
    return x
```

```
>>> y = 6  
>>> f(6)  
liste : ['x', 'y']  
intérieur : {'y' : 3, 'x' : 9}  
extérieur : {'f' : <function f at 0x822841c>,  
            'y' : 6,  
            '__name__' : '__main__',  
            '__doc__' : None}
```

9

$$u_0 = 1, u_1 = 1, u_n = u_{n-1} + u_{n-2} \quad \forall n \in \mathbb{N}, n > 1$$

$$u_0 = 1, u_1 = 1, u_n = u_{n-1} + u_{n-2} \forall n \in \mathbb{N}, n > 1$$

Version itérative :

```
def fibonacci(n) :  
    u, u1, u2 = 1, 1, 1  
    for i in range(2,n+1) :  
        u = u1 + u2  
        u2 = u1  
        u1 = u  
    return u
```

$$u_0 = 1, u_1 = 1, u_n = u_{n-1} + u_{n-2} \quad \forall n \in \mathbb{N}, n > 1$$

Version itérative :

```
def fibonacci(n) :  
    u, u1, u2 = 1, 1, 1  
    for i in range(2,n+1) :  
        u = u1 + u2  
        u2 = u1  
        u1 = u  
    return u
```

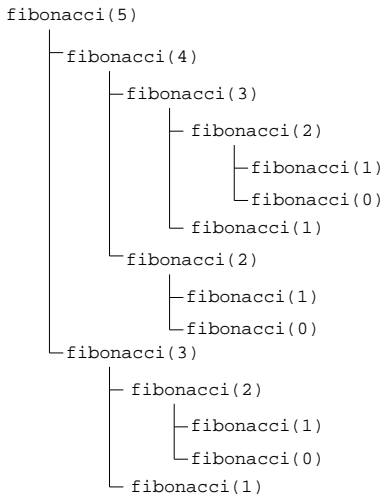
Version récursive :

```
def fibonacci(n) :  
    u = 1  
    if n > 1 :  
        u = fibonacci(n-1) +  
            fibonacci(n-2)  
    return u
```

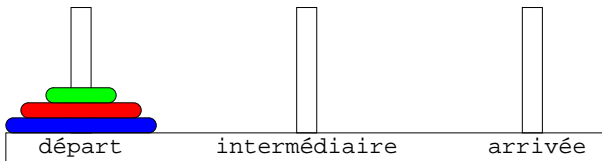
```
>>> fibonacci(5)
```

```
8
```

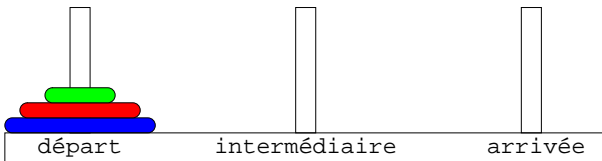
```
>>> fibonacci(5)  
8
```



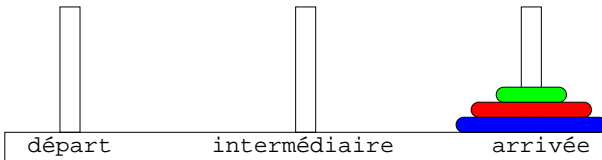
Etat initial :



Etat initial :



Etat final :



```
def hanoi(n, gauche, milieu, droit) :  
    assert type(n) is int  
    assert n >= 0  
    if n > 0 :  
        hanoi(n-1, gauche, droit, milieu)  
        deplacer(n, gauche, droit)  
        hanoi(n-1, milieu, droit, gauche)  
    return
```

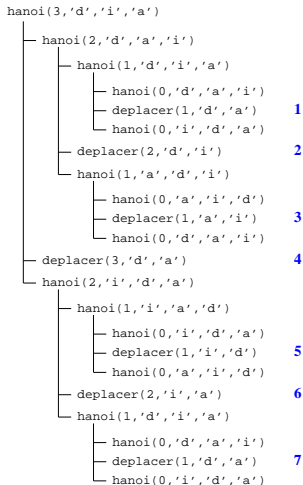
```
def hanoi(n, gauche, milieu, droit) :  
    assert type(n) is int  
    assert n >= 0  
    if n > 0 :  
        hanoi(n-1, gauche, droit, milieu)  
        deplacer(n, gauche, droit)  
        hanoi(n-1, milieu, droit, gauche)  
    return
```

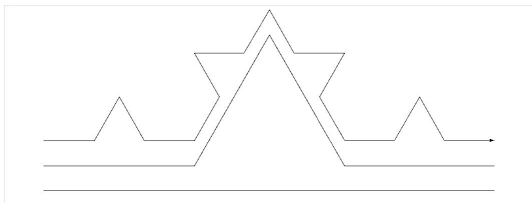
```
def deplacer(n, gauche, droit) :  
    print('déplacer disque', n,  
          'de la tour', gauche,  
          'à la tour', droit)  
    return
```

```
def hanoi(n, gauche, milieu, droit) :
    assert type(n) is int
    assert n >= 0
    if n > 0 :
        hanoi(n-1, gauche, droit, milieu)
        deplacer(n, gauche, droit)
        hanoi(n-1, milieu, droit, gauche)
    return
```

```
def deplacer(n, gauche, droit) :
    print('déplacer disque', n,
          'de la tour', gauche,
          'à la tour', droit)
    return
```

```
>>> hanoi(3, 'd', 'i', 'a')
```





$$\begin{cases} 0! = 1 \\ n! = n \cdot (n - 1)! \quad \forall n \in \mathbb{N}^* \end{cases}$$

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n - 1)! \quad \forall n \in \mathbb{N}^* \end{cases}$$

Version itérative :

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

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n - 1)! \quad \forall n \in \mathbb{N}^* \end{cases}$$

Version itérative :

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

Version récursive :

```
def factorielle(n) :  
    u = 1  
    if n > 1 :  
        u = n * factorielle(n-1)  
    return u
```

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

```
>>> factorielle(5)  
      (5*factorielle(4))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
      (5*(4*(3*(2*factorielle(1)))))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
      (5*(4*(3*(2*factorielle(1)))))
      (5*(4*(3*(2*1))))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
      (5*(4*(3*(2*factorielle(1)))))
      (5*(4*(3*(2*1))))
      (5*(4*(3*2)))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
      (5*(4*(3*(2*factorielle(1)))))
      (5*(4*(3*(2*1))))
      (5*(4*(3*2)))
      (5*(4*6))
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
      (5*(4*(3*(2*factorielle(1)))))
      (5*(4*(3*(2*1))))
      (5*(4*(3*2)))
      (5*(4*6))
      (5*24)
```

```
>>> factorielle(5)
      (5*factorielle(4))
      (5*(4*factorielle(3)))
      (5*(4*(3*factorielle(2))))
      (5*(4*(3*(2*factorielle(1)))))
      (5*(4*(3*(2*1))))
      (5*(4*(3*2)))
      (5*(4*6))
      (5*24)
```

120

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u  
  
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u
```

```
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

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

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u
```

```
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
>>> factorielle(5)  
      (factIter(5,1,1))
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u
```

```
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
>>> factorielle(5)  
      (factIter(5,1,1))  
      (factIter(5,2,2))
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u
```

```
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
>>> factorielle(5)  
      (factIter(5,1,1))  
      (factIter(5,2,2))  
      (factIter(5,3,6))
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u
```

```
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
>>> factorielle(5)  
    (factIter(5,1,1))  
    (factIter(5,2,2))  
    (factIter(5,3,6))  
    (factIter(5,4,24))
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u  
  
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
>>> factorielle(5)  
    (factIter(5,1,1))  
    (factIter(5,2,2))  
    (factIter(5,3,6))  
    (factIter(5,4,24))  
    (factIter(5,5,120))
```

```
def factorielle(n) :  
    u = factIter(n,1,1)  
    return u  
  
def factIter(n,i,fact) :  
    u = fact  
    if i < n :  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

```
>>> factorielle(5)  
    (factIter(5,1,1))  
    (factIter(5,2,2))  
    (factIter(5,3,6))  
    (factIter(5,4,24))  
    (factIter(5,5,120))  
  
120
```

```
def f(x) :  
    if cond : arret  
    else :  
        instructions  
        f(g(x))  
    return
```

```
def f(x) :  
    if cond : arret  
    else :  
        instructions  
        f(g(x))  
    return
```



```
def f(x) :  
    while not cond :  
        instructions  
        x = g(x)  
    arret  
    return
```

récursivité terminale → boucle

```
def f(x) :  
    if cond : arret  
    else :  
        instructions  
        f(g(x))  
    return
```



```
def f(x) :  
    while not cond :  
        instructions  
        x = g(x)  
    arret  
    return
```

```
def factIter(n,i,fact) :  
    if i >= n : u = fact  
    else :  
        pass  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```

récursivité terminale → boucle

```
def f(x) :  
    if cond : arret  
    else :  
        instructions  
        f(g(x))  
    return
```



```
def f(x) :  
    while not cond :  
        instructions  
        x = g(x)  
    arret  
    return
```

```
def factIter(n,i,fact) :  
    if i >= n : u = fact  
    else :  
        pass  
        u = factIter(n,i+1,fact*(i+1))  
    return u
```



```
def factIter(n,i,fact) :  
    while i < n :  
        pass  
        n,i,fact = n,i+1,fact*(i+1)  
    u = fact  
    return u
```