1 Fonctions numériques

1.1 Calcul de π

```python
# -*- coding: utf-8 -*-

def calculPi(n):
    """
    y = calculPi(n)
    calcul de pi à l'ordre n
    >>> from math import fabs, pi
    >>> fabs(pi - calculPi(0)) < 1.e0
    True
    >>> fabs(pi - calculPi(1)) < 1.e-1
    True
    >>> fabs(pi - calculPi(2)) < 1.e-2
    True
    >>> fabs(pi - calculPi(5)) < 1.e-5
    True
    >>> fabs(pi - calculPi(10)) < 1.e-10
    True
    >>> fabs(pi - calculPi(100)) < 1.e-100
    True
    """
    assert type(n) is int and n >= 0

    u = 1.
    y = 4./1. - 2./4. - 1./5. - 1./6.
    for k in range(1,n+1):
        u = u/16.
        y = y + u*(4./(8*k + 1) - 2./(8*k + 4) - 1./(8*k + 5) - 1./(8*k + 6))

    return y

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

1.2 Conversion décimal $\rightarrow$ base b

```python
# -*- coding: utf-8 -*-

def conversion(n,b=2,k=8):
    """
    code = conversion(n,b,k)
    code en base b sur k bits de l'entier décimal n -> list

    >>> conversion(23,2,8)
    [0, 0, 0, 1, 0, 1, 1, 1]
    >>> conversion(23,5,3)
    [0, 4, 3]
    >>> conversion(23,21,3)
    [0, 1, 2]
    >>> conversion(23,25,2)
    [0, 23]
    """
```

```
17      assert type(n) is int
18      assert type(b) is int
19      assert type(k) is int
20      assert n >= 0 and b > 1 and k > 0
21      assert n < b**k - 1
22
23      code = []
24      quotient = n
25      for i in range(k): code.append(0)
26
27      i = k - 1
28      while quotient != 0 and i >= 0:
29          code[i] = quotient%b
30          quotient = quotient//b
31          i = i - 1
32
33      return code
34
35  #-------------------------------------------------------------------------
36  if __name__ == "__main__":
37      import doctest
38      doctest.testmod()
```

2 Fonctions graphiques

2.1 Courbes paramétrées

```
1   # -*- coding: utf-8 -*-
2
3   from math import *
4
5   #-------------------------------------------------------------------------
6   def parametric_circle(x0,y0,r):
7       """
8       cercle paramétrique : x = x0 + r*cos(t), y = y0 + r * sin(t)
9       """
10      return lambda t: x0 + r * cos(t), lambda t: y0 + r * sin(t)
11
12  #-------------------------------------------------------------------------
13  def drawCurve(f,t1,t2,dt):
14      """
15      trace une courbe paramétrée pour t dans [t1,t2] par pas de dt
16      pour les fonctions x = f[0](t) et y = f[1](t)
17      >>> drawCurve(parametric_circle(-150,0,100),0.,2*pi,0.1)
18      >>> drawCurve(parametric_circle(-200,0,50),0.,pi,0.1)
19      """
20      assert type(t1) is float and type(t2) is float and type(dt) is float
21      assert type(f) is tuple
22
23      from turtle import up, down, goto
24      values = []
25      t = t1
26      while t < t2:
27          values.append(t)
28          t = t + dt
29
30      up()
```

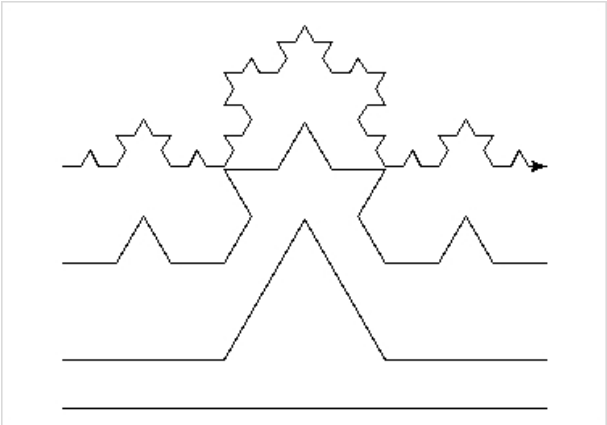
```
    goto(f[0](t1),f[1](t1))
    down()

    for t in values: goto(f[0](t),f[1](t))

    return

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

2.2 Courbes fractales

```
# -*- coding: utf-8 -*-

from turtle import *

#--------------------------------------------------------------------------
def p(n,d):
    """
    >>> up(); goto(-150,-150); down(); p(0,300)
    >>> up(); goto(-150,-75); down(); p(1,300)
    >>> up(); goto(-150,0); down(); p(2,300)
    >>> up(); goto(-150,75); down(); p(3,300)
    """
    assert type(n) is int
    assert n >= 0
    if n == 0: forward(d)
    else:
        p(n-1,d/3.)
        left(60)
        p(n-1,d/3.)
        right(120)
        p(n-1,d/3.)
        left(60)
        p(n-1,d/3.)
    return

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

Les courbes ci-dessous correspondent, respectivement de bas en haut, aux appels suivants :

`p(0,300)`, `p(1,300)`, `p(2,300)` et `p(3,300)`.



3 Appels de fonctions

3.1 Portée des variables

```
>>> x = 5
>>> print(x)
5

>>> y = f(x)
>>> print(x)
f 10
5

>>> z = g(x)
>>> print(x)
f 10
g 20
5

>>> t = h(x)
>>> print(x)
f 10
f 20
g 40
h 80
5
```

```
>>> x = 5
>>> print(x)
5

>>> x = f(x)
>>> print(x)
f 10
10

>>> x = g(x)
>>> print(x)
f 20
g 40
40

>>> x = h(x)
>>> print(x)
f 80
f 160
g 320
h 640
640
```

3.2 Récursivité

```
1  # -*- coding: utf-8 -*-
2
3  def recherchePremier(t,x,debut):
4      """
5      ok,index = recherchePremier(t,x,debut)
6      recherche la première occurence de x dans la liste t en commençant
7      à l'indice debut
8      ok == True si x a été trouvé à l'indice index, False sinon
9
```

```
10      >>> recherchepremier([3,6,1,4,1,5,2,4],1,0)
11      (True, 2)
12      >>> recherchepremier([3,6,1,4,1,5,2,4],1,3)
13      (True, 4)
14      >>> recherchepremier([3,6,1,4,1,5,2,4],1,5)
15      (False, 8)
16      """
17      assert type(t) is list
18      assert type(debut) is int and 0 <= debut <= len(t)
19
20      ok, index = False, debut
21      if index > len(t) - 1: ok = False
22      else:
23          if t[index] == x: ok = True
24          else: ok, index = recherchePremier(t,x,index+1)
25
26      return ok, index
27
28  #-------------------------------------------------------------------------
29  if __name__ == "__main__":
30      import doctest
31      doctest.testmod()
```

4 Exécutions de fonctions

4.1 Exécution d'une fonction itérative

Il s'agit de l'algorithme du tri par sélection et l'appel correspond au tri du tableau `[3,6,4,5,2,1]`.

```
>>> f1([3,6,4,5,2,1])
0 [1, 6, 4, 5, 2, 3]
1 [1, 2, 4, 5, 6, 3]
2 [1, 2, 3, 5, 6, 4]
3 [1, 2, 3, 4, 6, 5]
4 [1, 2, 3, 4, 5, 6]
5 [1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>>
```

4.2 Exécution d'une fonction récursive

Il s'agit de l'algorithme des tours de Hanoï et l'appel correspond au déplacement de 3 disques de la tour 4 à la tour 6 en utilisant la tour 5.

```
>>> f2(3,4,5,6)
4 6
4 5
6 5
4 6
5 4
5 6
4 6
>>>
```