Nom :	Prénom :	Groupe :

Durée : 90’ Documents, calculettes, téléphones et ordinateurs interdits

1 Fonctions numériques

1.1 Calcul de π

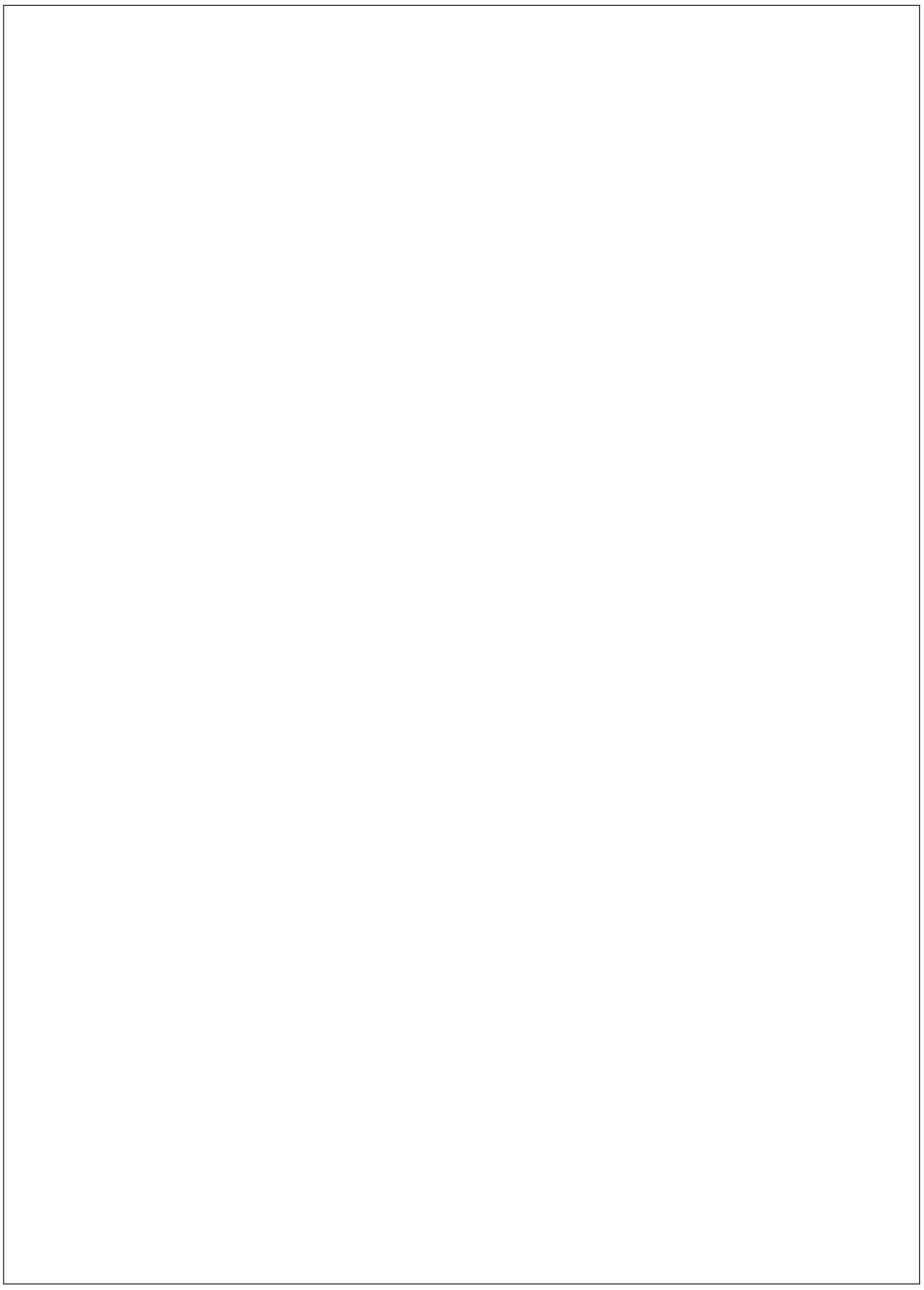
Définir une fonction qui calcule π à l’ordre n selon l’approximation suivante :

$$\pi \approx \sum_{k=0}^{n} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

On n’utilisera pas la fonction *puissance* (`x**k`).

1.2 Conversion décimal $\rightarrow$ base b

Définir une fonction qui code sur k chiffres un entier positif n du système décimal au système en base b.

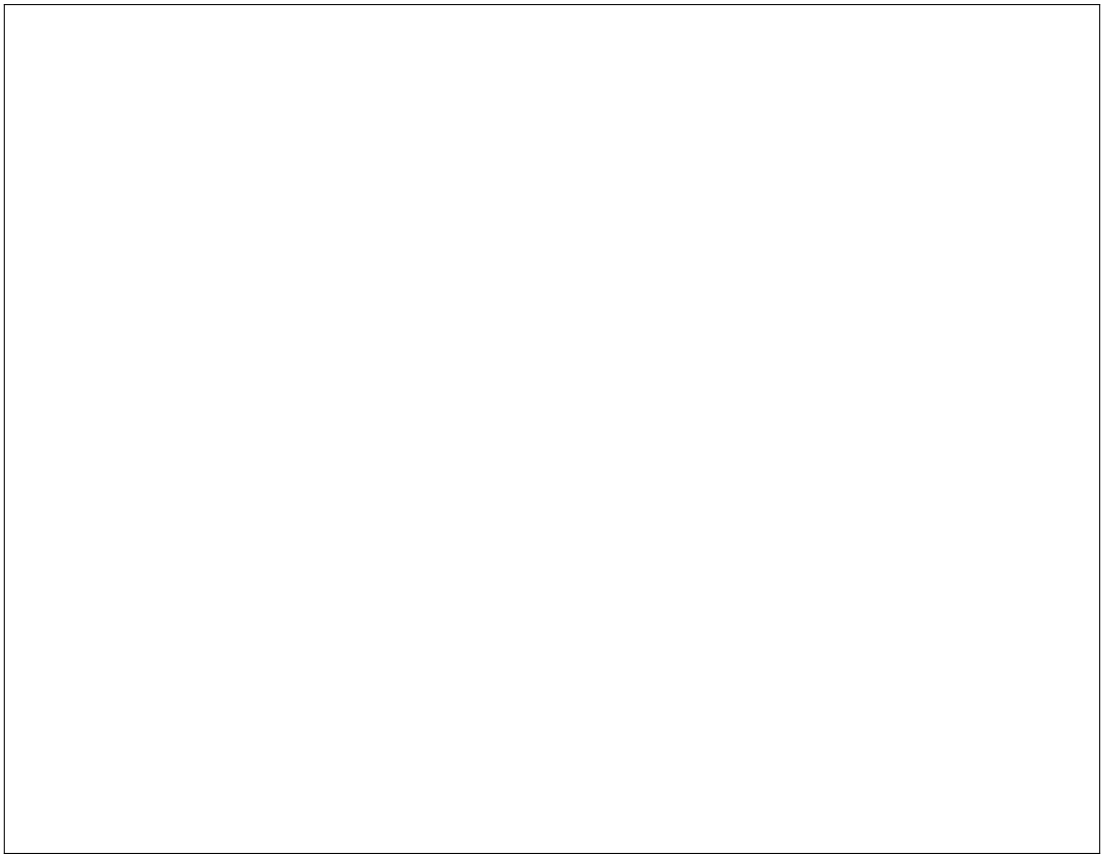


2 Fonctions graphiques

2.1 Courbes paramétrées

Ecrire une fonction qui permettent le tracé de courbes paramétrées ($x = f(t)$, $y = g(t)$) en utilisant les instructions *à la Logo*. Les courbes paramétrées seront données sous la forme de fonctions `lambda` comme dans l'exemple ci-dessous du cercle de centre (x_0, y_0) et de rayon r.

```
#---------------------------------------------------------------------------
def parametric_circle(x0,y0,r):
#---------------------------------------------------------------------------
    """
    cercle paramétrique : x = x0 + r*cos(t), y = y0 + r * sin(t)
    """
    return lambda(t): x0 + r * cos(t),
           lambda(t): y0 + r * sin(t)
#---------------------------------------------------------------------------
```

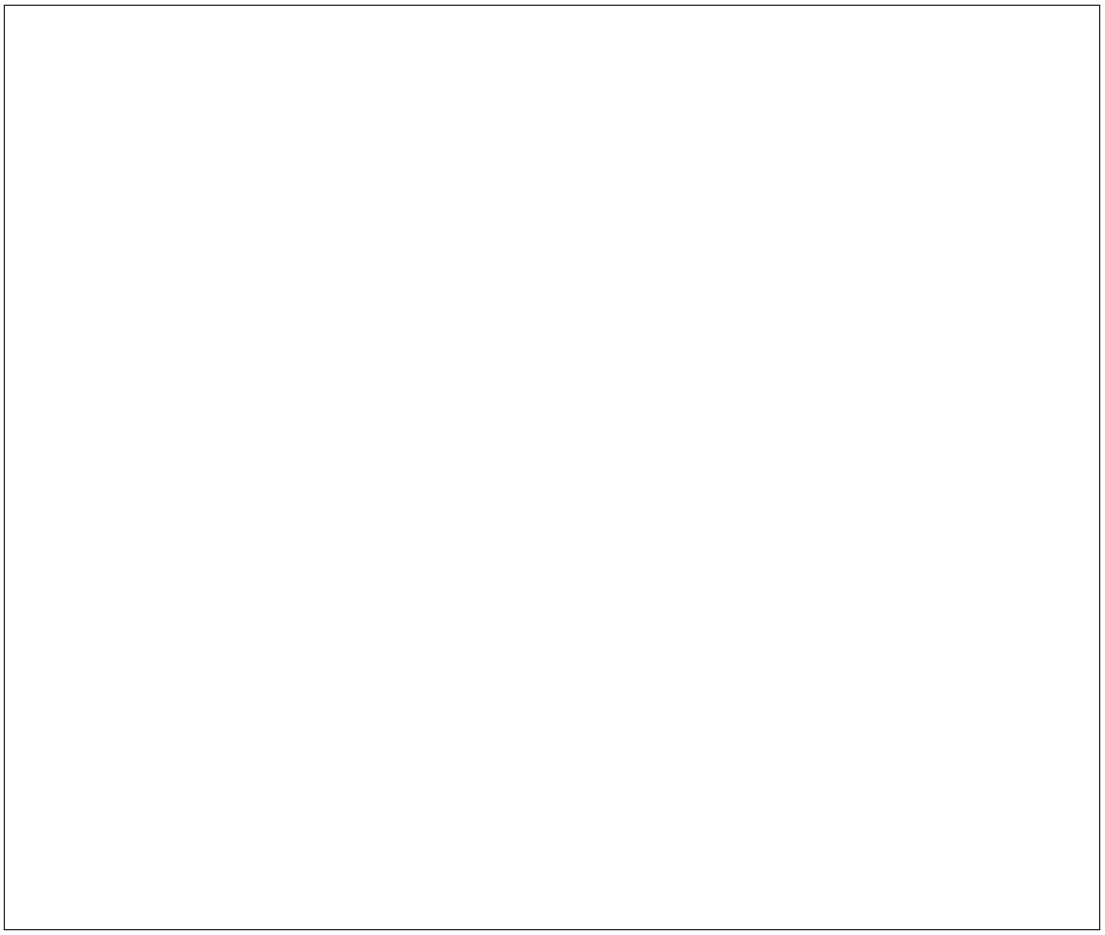


2.2 Courbes fractales

On considère la procédure `p` ci-contre :

```
def p(n,d):
    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
```

1. On considère l'appel `p(1,300)` et le crayon initialement en (0,0) avec une direction de 0. Dessiner le résultat de cet appel.
2. On considère l'appel `p(3,300)` et le crayon initialement en (0,0) avec une direction de 0. Dessiner le résultat de cet appel.



3 Appels de fonctions

3.1 Portée des variables

On considère les fonctions `f`, `g` et `h` suivantes :

```
def f(x):
    x = 2*x
    print('f', x)
    return x
```

```
def g(x):
    x = 2*f(x)
    print('g', x)
    return x
```

```
def h(x):
    x = 2*g(f(x))
    print('h', x)
    return x
```

Qu'affichent les appels suivants ?

1.
```
>>> x = 5
>>> print(x)

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

>>> z = g(x)
>>> print(x)

>>> t = h(x)
>>> print(x)
```

1.
```
>>> x = 5
>>> print(x)

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

>>> x = g(x)
>>> print(x)

>>> x = h(x)
>>> print(x)
```

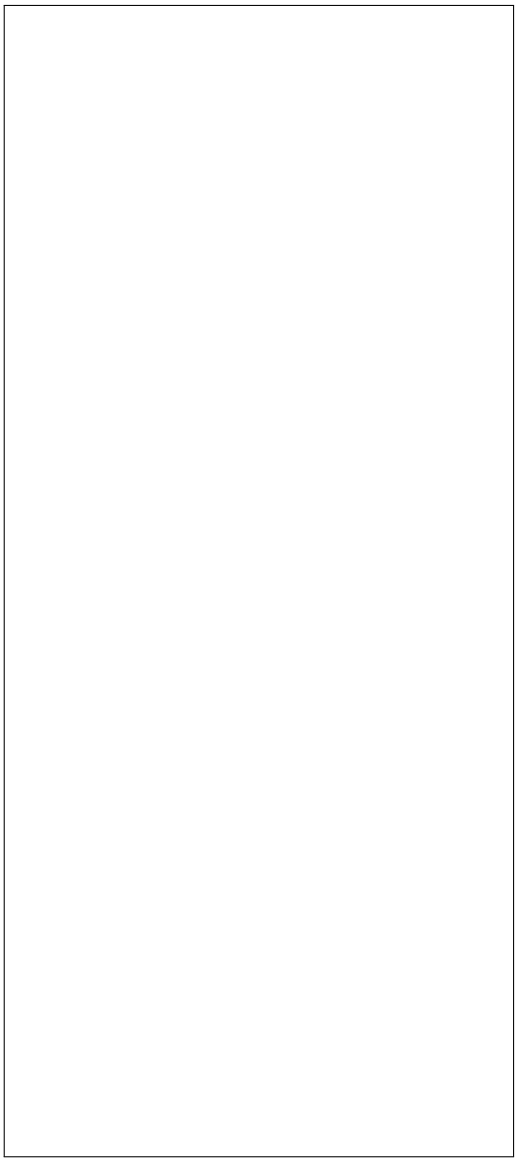
3.2 Récursivité

Définir une fonction récursive pour la recherche de la première occurrence d'un élément x dans un tableau t.

4 Exécutions de fonctions

4.1 Exécution d'une fonction itérative

Qu'affiche l'appel `f([3,6,4,5,2,1])` où `f` est la fonction itérative définie ci-contre ?



```
#---------------------------------------
def f(t):
#---------------------------------------
    assert type(t) is list

    for i in range(len(t)):
        m = i
        for j in range(i+1,len(t)):
            if t[j] < t[m]: m = j
        t[i],t[m] = t[m],t[i]
        print(i, t)  #---------- affichage

    return t
#---------------------------------------
```

4.2 Exécution d'une fonction récursive

Qu'affiche l'appel `f(3,4,5,6)` où `f` est la fonction récursive définie ci-contre ?

```
#---------------------------------------
def f(n,a,b,c):
#---------------------------------------
    assert type(n) is int
    assert n >= 0
    assert a != b
    assert b != c
    assert a != c

    if n > 0:
      f(n-1,a,c,b)
      print(a, c) #------------- affichage
      f(n-1,b,a,c)

    return
#---------------------------------------
```