1 Boucle simple

Nom :	Prénom :	Groupe :	Question :

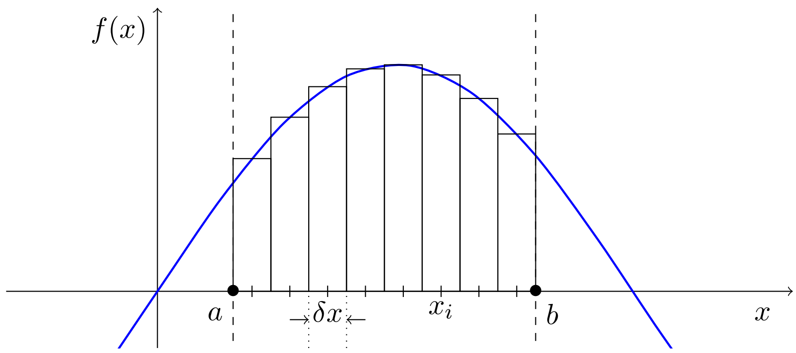
Durée : 15' Documents, calculettes, téléphones et ordinateurs interdits25'

Enoncé : Soit $f(x)$ une fonction continue de $R \to R$ à intégrer sur $[a,b]$ (on supposera que f à toutes les bonnes propriétés mathématiques pour être intégrable sur l'intervalle considéré). On cherche à calculer son intégrale $I = \int_a^b f(x)dx$ qui représente classiquement l'aire comprise entre la courbe représentative de f et les droites d'équations $x = a$, $x = b$ et $y = 0$. Les méthodes classiques d'intégration numérique (méthode des rectangles, méthode des trapèzes et méthode de Simpson) consistent essentiellement à trouver une bonne approximation de cette aire.

Dans la méthode des rectangles, on subdivise l'intervalle d'intégration de longueur $b - a$ en n parties égales de longueur $\delta x = \dfrac{b-a}{n}$. Soient x_1, x_2, ...,x_i, ..., x_n les points milieux de ces n intervalles. Les n rectangles formés avec les ordonnées correspondantes ont pour surface $f(x_1)\delta x$, $f(x_2)\delta x$, ..., $f(x_i)\delta x$, ..., $f(x_n)\delta x$. L'aire sous la courbe est alors assimilée à la somme des aires de ces rectangles, soit :

$$I = \int_a^b f(x)dx \approx (f(x_1) + f(x_2) + \cdots + f(x_i) + \cdots + f(x_n))\,\delta x$$

C'est la formule dite des rectangles qui repose sur une approximation par une fonction en escalier.



Question : Définir la fonction `integration` qui calcule l'intégrale I d'une fonction `f` sur `[a,b]` à l'ordre `n` par la méthode des rectangles.

```
>>> integration(cos,0.,pi,10000)
4.307866381890461e-16
>>> integration(cos,-pi/2,pi/2,10000)
2.000000008224676
>>> integration(sin,0.,pi,10000)
2.000000008224676
>>> integration(exp,0.,1.,10000)
1.7182818277430947
>>> integration(log,1.,2.,100)
0.38629644443195715
```

```
>>> integration(lambda x: 3,1.,2.,10)
3.0
>>> integration(lambda x: 3,2.,1.,10)
-3.0
>>> integration(lambda x: x,-1.,1.,10)
8.881784197001253e-17
>>> integration(lambda x: x*x,-1.,1.,100)
0.6666000000000001
>>> integration(lambda x: x**3,0.,1.,100)
0.24998750000000006
```

Réponse :

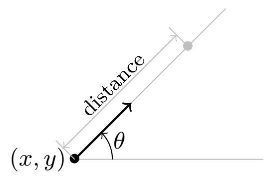
2 Appels récursifs

NOM :	PRÉNOM :	GROUPE :	QUESTION :

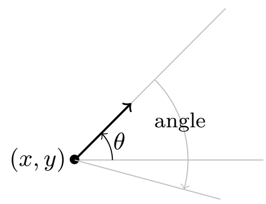
DURÉE : 15' DOCUMENTS, CALCULETTES, TÉLÉPHONES ET ORDINATEURS INTERDITS25'

Enoncé : On considère les fonctions `f` et `g` ci-dessous qui utilisent les primitives classiques de la tortue LOGO (module `turtle`) située en (x, y) avec une orientation θ par rapport à l'horizontale.

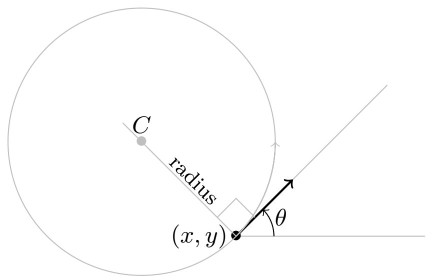
`forward(distance)` : Move the turtle forward by the specified `distance` (integer or float), in the direction the turtle is headed.



`right(angle)` : Turn turtle right by `angle` (integer or float) units. Units are by default degrees.



`circle(radius)` : Draw a circle with given `radius` (integer or float). The center C is radius units left of the turtle.




```python
from turtle import *

def f(n,m,d) :
    assert type(n) is int
    assert n >= 0
    assert type(m) is int
    assert m >= 0
    assert type(d) is float
    assert d >= 0.0

    for i in range(n):
        g(m,d)
        right(360/n)
    return
```

```python
def g(m,d) :
    assert type(m) is int
    assert m >= 0
    assert type(d) is float
    assert d >= 0.0

    if m > 0 :
        g(m-1,d/2)
        circle(d/2)
        g(m-1,d/2)
    else :
        forward(d)
    return
```

Question : Que dessinent les appels suivants ?

1. `>>> f(1,0,100)`
2. `>>> f(1,1,100)`
3. `>>> f(1,2,100)`
4. `>>> f(2,2,100)`
5. `>>> f(3,2,100)`
6. `>>> f(4,3,100)`

Réponses :

1. `>>> f(1,0,100)`

2. `>>> f(1,1,100)`

3. `>>> f(1,2,100)`

4. `>>> f(2,2,100)`

5. `>>> f(3,2,100)`

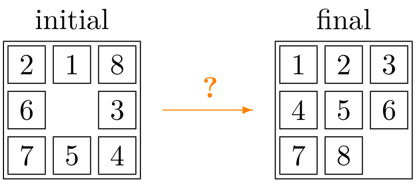
6. `>>> f(4,3,100)`

3 Liste de listes

Nom :	Prénom :	Groupe :	Question :

Durée : 15’ Documents, calculettes, téléphones et ordinateurs interdits25’

Enoncé : Le taquin est un jeu en forme de damier $(n \times n)$. Il est composé de $(n^2 - 1)$ petits carreaux numérotés de 1 à $(n^2 - 1)$ qui glissent dans un cadre prévu pour n^2 carreaux. Il consiste à remettre dans l’ordre les $(n^2 - 1)$ carreaux à partir d’une configuration initiale quelconque. L’état du taquin $(n \times n)$ sera représenté par une liste de n listes de n éléments chacune, chaque élément portant un numéro de 1 à $(n - 1)$, le carreau vide étant numéroté 0. La figure ci-dessous donne un exemple de taquin (3×3).



taquin (3×3)

Le carreau vide est numéroté `0` :

état initial : `[[2,1,8],[6,0,3],[7,5,4]]`

$\downarrow$?

état final : `[[1,2,3],[4,5,6],[7,8,0]]`

Questions :

1. Définir la fonction `listeCarree` qui teste si une liste `t` est une liste de n éléments dont chaque élément est lui-même une liste de n éléments.

```
>>> listeCarree([])
True
>>> listeCarree([[1]])
True
>>> listeCarree([[1],[2]])
False
>>> listeCarree([[1,7],[2,-3]])
True
>>> listeCarree([[2,1,8],[6,0,3]])
False
>>> listeCarree([[2,1,8],[6,0,3],[7,5,4]])
True
>>> listeCarree([[2,1,8],[6,0,3],[7]])
False
```

2. Définir la fonction `appartient` qui teste si un élément `e` appartient (`True`) ou non (`False`) à une liste carrée `t`.

```
>>> t = [[2,1,8],[6,0,3],[7,5,4]]
>>> 5 in t
False
>>> 9 in t
False
>>> t = [[2,1,8],[6,0,3],[7,5,4]]
>>> appartient(5,t)
True
>>> appartient(9,t)
False
```

3. Définir la fonction `Taquin` qui teste si une liste `t` représente (`True`) ou non (`False`) un jeu de taquin.

```
>>> Taquin([[0,1,2]])
False
>>> Taquin([[3,1],[0,2]])
True
>>> Taquin([[2,1,8],[6,0,3],[7,5,4]])
True
>>> Taquin([[2,1,8],[6,0,3],[7,5,0]])
False
```

4. Définir la fonction `jouerTaquin` qui, à partir d’une configuration donnée `jeu` d’un damier $(n \times n)$, retourne la liste des configurations possibles après un déplacement élémentaire du carreau vide.

```
>>> jeu  = [[3,1],[0,2]]
>>> jouerTaquin(jeu)
[[[0, 1], [3, 2]] ,
 [[3, 1], [2, 0]]]
```

```
>>> jeu  = [[2,1,8],[6,0,3],[7,5,4]]
>>> jouerTaquin(jeu)
[[[2, 0, 8], [6, 1, 3], [7, 5, 4]],
 [[2, 1, 8], [6, 5, 3], [7, 0, 4]],
 [[2, 1, 8], [0, 6, 3], [7, 5, 4]],
 [[2, 1, 8], [6, 3, 0], [7, 5, 4]]]
```

Réponses :