```python
# Récursivité_une_correction.py

from random import randint


n= int(input("Donner le nombre d'entiers n à afficher "))
def afficher_iteratif(n) :
    i = 1
    while( i <= n) :
        print(i)
        i+= 1
def afficher_recursif(n):
    #cas de base
    if n == 1 :
        print("Décollage")
        print(n)
    #cas recursif
    else :
        print(f"appel afficher recursif n : {n}")
        afficher_recursif(n-1)
        print(n)
afficher_iteratif((n))
afficher_recursif(n)


########### factoriel n récursif ###############


def factoriel_recursif(n) :
    """
    Calcul du factoriel de n de manière récursive
    """
    print(f"factoriel_recursif({n}) ")
    if n == 0 :
        fact = 1
        #print(f"factoriel_recursif({0}) :
```

```
{fact} ")
035|           return fact
036|       else :
037|           fact = n*factoriel_recursif(n-1)
038|           print(f"factoriel_recursif({n}) :
{fact} ")
039|           return fact
040|
041| # n= int(input("Donner le nombre  n factoriel
"))
042|
043|
044| print(factoriel_recursif(10))
045|
046| ########### tests ###############
047|
048| assert factoriel_recursif(0) == 1
049| assert factoriel_recursif(4) == 24
050| assert factoriel_recursif(5) ==  120
051|
052| ########### Calcul d'une suite mathématique
###############
053|
054|
055| def U(n):
056|       """
057|        U0 = 5
058|        Un+1 = Un +3
059|        Soit une suite arithmetique de raison 3 :
on ajoute 3 à chaque incrément de n
060|        """
061|       print(f"Appel à U{n}")
062|       #cas particulier
063|       if  n == 0 :
064|           print(f"partiel  U{0} : 5 " )
065|           return 5
066|           """
067|            le return est très important la
fonction s'apelant elle-même elle doit
068|            retourner une valeur
```

```
069|           """
070|       #cas général
071|       else :
072|           partiel = U(n-1) + 3
073|           print(f"partiel  U{n} : {partiel} " )
074|           return partiel # idem
075|
076|  print(U(5))
077|
078|  ########### tests ###############
079|
080|  assert U(0) == 5
081|  assert U(5) == 20
082|
083|  print(U(5))
084|
085|  ########### Calcul des valeurs de la suite de
Fibonnacci ###############
086|
087|  def Fibonnaci_iteratif(n) :
088|       F0=0
089|       print(f"F({0}) = {F0}")
090|       F1 = 1
091|       print(f"F({1}) = {F1}")
092|       for i in range(2,n ):
093|           suivant  = F0+F1
094|           print(f"n = {i} F({i}) = {suivant}")
095|           temp = F1
096|           F1 = suivant
097|           F0= temp
098|
099|  Fibonnaci_iteratif(10)
100|
101|
102|
103|  def Fibonnacci(n) :
104|       """
105|       Fibonnacci récursif
106|       """
107|       #print(f"Fibonnacci ({n}) ")
```

```
108|     if n == 0 :
109|         return 0
110|     elif n == 1 :
111|         return 1
112|     else :
113|         return Fibonnacci(n-1) +
Fibonnacci(n-2 )
114|
115| print(Fibonnacci(10))
116|
117| for i in range (11) :
118|     print(f"Fibonnacci ({i})  : {Fibonnacci
(i)}")
119|
120|
121|
122|
123| ########### CQuick sort ###############
124|
125|
126| ll =[-10,5,8,45,-96,12,897,-50]
127|
128|
129| def tri_rapide(list):
130|     inferieure = []
131|     pivot = []
132|     superieure = []
133|     #cas particulier
134|     if len(list) < 2:
135|         return list
136|     #cas general
137|     # on détermine aléatoirement un pivot
138|     pivot_valeur =
list[randint(0,len(list)-1)]
139|     for i in list:
140|         if i < pivot_valeur:
141|             # on insère  la valeur dans
inférieure si elle est inférieure à la valeur du
pivot
142|             inferieure.append(i)
```

```
143|         elif i > pivot_valeur:
144|             # on insère  la valeur dans
supérieuree si elle est supérieure à la valeur du
pivot
145|             superieure.append(i)
146|         else:
147|             # on stockeles différntes valeurs
des pivots
148|             pivot.append(i)
149|     # on recolle les morceaux
150|     return tri_rapide(inferieure) + pivot +
tri_rapide(superieure)
151|
152|
153| ########### tests ###############
154| print(tri_rapide(ll))
155|
156|
157| ###################### Courbe de Koch
##################"
158| from turtle import *
159|
160|
161| def courbeVonKoch_1(cote):
162|     forward(cote)
163|     left(60)
164|     forward(cote)
165|     left(-120)
166|     forward(cote)
167|     left(60)
168|     forward(cote)
169|
170|
171| def courbeVonKoch(  n, cote  ) :
172|     if n == 0 :
173|         forward(cote)
174|     else :
175|         # chaque segment est divisé en trois
176|         courbeVonKoch(n-1, cote/3)
177|         left(60)
```

```
178|         courbeVonKoch(n-1, cote/3)
179|         left(-120)
180|         courbeVonKoch(n-1, cote/3)
181|         left(60)
182|         courbeVonKoch(n-1, cote/3)
183|
184| setheading(0) # orientation intiale de la tête
: vers la droite de l'écran
185| hideturtle() # on cache la tortue
186| speed(1000)     # on accélère la tortue
187| sc  = Screen()
188| sc.setup(800,800 )
189| up()
190| goto(-300,-200)
191| down()
192| color('blue')
193| courbeVonKoch_1(200)
194| up()
195| goto(-300,0)
196| down()
197| color('green')
198| n=2
199| courbeVonKoch(  5 ,  6*200/n  )
200| exitonclick() # pour  garder ouverte la
fenêtre
```