**Рекурсивный вызов функции**

**Обращаемся к цвету по номеру**

Каждому цвету присвоим номер. Будем обращаться к цвету по номеру.

tones = [

'violet', # tones[0]

'blue', # tones[1]

'green', # tones[2]

'yellow', # tones[3]

'gold', # tones[4]

'orange', # tones[5]

'red' # tones[6]

]

t.color(tones[1]) # t.color('blue')

t.fillcolor(tones[6]) # t.fillcolor('red')

t.fillcolor(tones[10%7]) # t.fillcolor('yellow')

i = 10

t.fillcolor(tones[i%7]) # t.fillcolor('yellow')

t.fillcolor(tones[i % len(tones)]) # t.fillcolor('yellow')

**Пример 1: вложенные квадраты через цикл**

|  |  |
| --- | --- |
| sqn(200, 20, 7) | sqn(100, 20, 7) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/ex11_1.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/ex11_2.png |

# -\*- coding: utf-8 -\*-

import turtle

import time

tones = [

'violet', # tones[0]

'blue', # tones[1]

'green', # tones[2]

'yellow', # tones[3]

'gold', # tones[4]

'orange', # tones[5]

'red' # tones[6]

]

def sq(size, col): # нарисовать квадрат размера size цвета col

t.color(col)

for i in range(4):

t.fd(size)

t.left(90)

# n квадратов, первый размера size, каждый следующий на d меньше

def sqn(size, d, n):

for i in range(n): # i меняется от 0 до n-1

sq(size, tones[i]) # tones[i] - взять цвет номер i из tones

size -= d

if size < 0: # если следующий квадрат маленький, больше не рисовать

return

t = turtle.Turtle()

t.shape("turtle")

t.width(3)

t.speed(0)

# sqn(200, 20, 7) # 7 квадратов, первый размером 200, другие на 20 меньше

sqn(100, 20, 7) # 7 квадратов, первый размером 100, другие на 20 меньше

turtle.done()

**Пример 2: вложенные квадраты через рекурсивный вызов функции (хвостовая рекурсия)**

DEA! Изменилась только функция **sqn** и ее вызов.

# -\*- coding: utf-8 -\*-

import turtle

import time

tones = [

'violet', # tones[0]

'blue', # tones[1]

'green', # tones[2]

'yellow', # tones[3]

'gold', # tones[4]

'orange', # tones[5]

'red' # tones[6]

]

def sq(size, col): # нарисовать квадрат размера size цвета col

t.color(col)

for i in range(4):

t.fd(size)

t.left(90)

# нарисовать n квадратов,

# первый размера size,

# каждый следующий на d меньше

# уже нарисовали i квадратов

def sqn(size, d, n, i):

if n == 0: # если не надо больше рисовать квадратов

return

if size <= 0: # если следующий квадрат маленький, больше не рисовать

return

sq(size, tones[i]) # рисовать 1 квадрат

sqn(size-d, d, n-1, i+1) # рисовать следующий квадрат - РЕКУРСИЯ

t = turtle.Turtle()

t.shape("turtle")

t.width(3)

t.speed(0)

sqn(200, 20, 7, 0) # 7 квадратов, первый размером 200, другие на 20 меньше

turtle.done()

**Вызов в коде функции sqn самой функции sqn называется рекурсивным вызовом функции**

**Пример 3: вложенные квадраты через рекурсивный вызов функции**

Попробуем вызывать sqn до и после рисования одного квадрата.

|  |  |
| --- | --- |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/ex11_1.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/ex11_3.png |
| def sqn(size, d, n, i):  if n == 0:  return  if size <= 0:  return  sq(size, tones[i])  sqn(size-d, d, n-1, i+1) | def sqn(size, d, n, i):  if n == 0:  return  if size <= 0:  return  sqn(size-d, d, n-1, i+1)  sq(size, tones[i]) |

**Задача 1: Квадраты центрированные**

Написать функцию **sqn(size, d, n, i)**, которая с помощью РЕКУРСИИ рисует такие квадраты. Если квадрат слишком маленький, не рисовать.

* size - размер стороны квадрата
* d - на сколько сторона следующего квадрата меньше предыдущего
* n - сколько еще квадратов надо нарисовать
* i - сколько квадратов уже нарисовали

|  |  |
| --- | --- |
| sqn(200, 20, 7, 0) | sqn(100, 20, 7, 0) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/ex11_00.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/ex11_01.png |

**Задача 2а: Квадраты с поворотом на 30o**

Следующий квадрат повернут на 30o

Написать функцию **sqn(size, n, i)**, которая с помощью РЕКУРСИИ рисует такие квадраты. Если квадрат слишком маленький, не рисовать.

* size - размер стороны квадрата
* n - сколько еще квадратов надо нарисовать
* i - сколько квадратов уже нарисовали

|  |
| --- |
| sqn(200, 7, 0) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_2.png |

**Задача 2b: Квадраты с поворотом (произвольный угол)**

Написать функцию **sqn(size, ang, n, i)**, которая с помощью РЕКУРСИИ рисует такие квадраты. Если квадрат слишком маленький, не рисовать.

* size - размер стороны квадрата
* ang - на сколько **градусов** повернут следующий квадрат
* n - сколько еще квадратов надо нарисовать
* i - сколько квадратов уже нарисовали

|  |
| --- |
| sqn(200, 30, 7, 0) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_2.png |

**Задача 3a: Вписанные правильные треугольники**

Написать функцию **tri(size, n, i)**, которая с помощью РЕКУРСИИ рисует вложенные правильные треугольники. Если сторона треугольника меньше 20, не рисовать.

* size - размер стороны треугольника
* n - сколько еще фигур надо нарисовать
* i - сколько фигур уже нарисовали

|  |
| --- |
| sqn(200, 7, 0) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_3_3.png |

**Задача 3b: Вписанные правильные многоугольники**

**Сумма углов правильного k-угольника (k-2)\*180**

Написать функцию **sqn(size, k, n, i)**, которая с помощью РЕКУРСИИ рисует вложенные правильные многоугольники. Если сторона многоугольника меньше 20, не рисовать.

* size - размер стороны многоугольника
* k - количество углов в многоугольнике
* n - сколько еще многоугольников надо нарисовать
* i - сколько многоугольников уже нарисовали

|  |  |  |
| --- | --- | --- |
| sqn(200, 3, 7, 0) | sqn(200, 4, 3, 0) | sqn(200, 5, 5, 0) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_3_3.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_3_4.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_3_5.png |

**Задача 4: Спираль вовнутрь (чередование 3 цветов)**

Написать функцию **spi(size, d, n, col)**, которая с помощью РЕКУРСИИ рисует вложенные спираль. Если сторона спирали меньше 10, не рисовать.

* size - длина отрезка спирали
* d - на сколько следующий отрезок меньше предыдущего
* n - сколько еще отрезков надо нарисовать
* col - цвет отрезка; цвета меняются по очереди: red, gold, blue

|  |
| --- |
| spi(200, 10, 15, 'red') |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonRecursion/t11_4.png |

**Задача 6: Узор из рекурсивных веток**

Функция **branch(n, size0, dsize, ang0, dang)** рисует 1 ветку из n отрезков. Первый отрезок длины size0, каждый следующий меньше на dsize. Первый угол поворота налево ang0, каждый следующий больше на dang.

Функция возвращает точку, в которой стоит черепаха, после того, как нарисовали половину отрезков ветки.

def branch(n, size0, dsize, ang0, dang):

size = size0

ang = ang0

for i in range(n):

if i == n/2:

p = t.pos()

t.fd(size)

t.left(ang)

size -= dsize

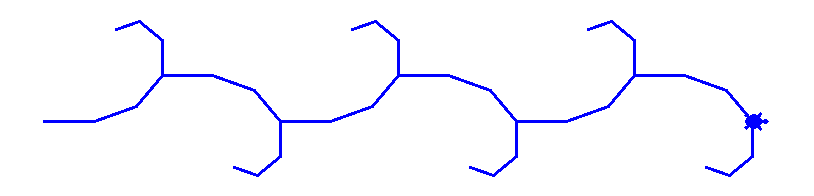
ang += dang

return p

Одну ветку, как на рисунке, можно нарисовать так:

pc = branch(6, 50, 5, 20, 10)

Напишите функцию **uzor(n, size0, dsize, ang0, dang, k)**, которая использует функцию **branch** и рисует узор из **k** веток РЕКУРСИВНО.

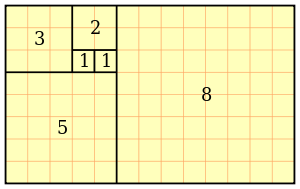


**Задача 7: Квадраты Фибоначчи**

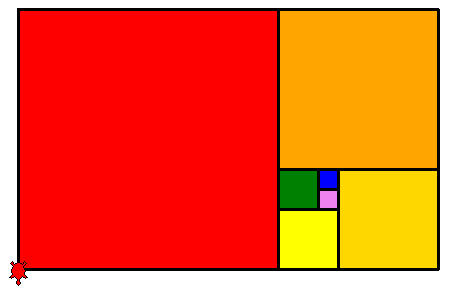
[Числа Фибоначчи](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%3Cbr%20/%3E%E2%80%A2D0%B0_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8) - это последовательность 1, 1, 2, 3, 5, 8, 13, 21...

Fn = Fn-1 + Fn-2

F1 = F2 = 1



Нарисовать n квадратов со сторонами, равными первым n числам Фибоначчи.



Подсказка: можно написать функцию fib0(size, n), которая рисует n квадратов Фибоначчи, size - размер стороны первого квадрата. Она рисует первые 1 или 2 квадрата и вызывает функцию fib2(f0, f1, size, i, n), чтобы нарисовать остальные квадраты.

**Задача 8. Путь с поворотами**

Нарисовать дорогу, без тупиков и перекрестков, с n левыми и k правыми поворотами.