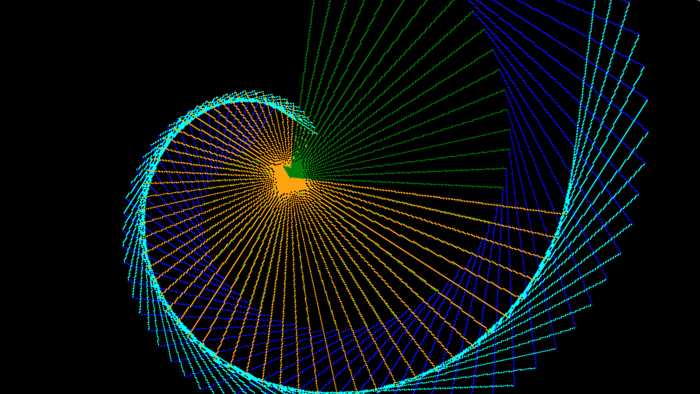
**Графика в Python при помощи модуля Turtle. Часть 1**

****

От [astler](https://geekstand.top/author/astler/) / 5 минут чтения / 4 февраля, 2020 / [python](https://geekstand.top/tags/python/), [turtle](https://geekstand.top/tags/turtle/)

Изучая обилие каналов в любимой телеге я увидел ссылку на плейлист [Модуль turtle. Создаем графику в Python](https://www.youtube.com/watch?v=WIWHPDN7CTI&list=PLQAt0m1f9OHvowenYcOHrRP_v1VN-0TWF). Чего уж говорить, а рост интереса к змейке игнорировать просто нельзя! Я уже молчу о том, что он занял все топы рекомендаций к изучению, а есть еще и машинное обучение, в котором без него сейчас просто никак. Иными словами: хочешь быть в теме — учи Python. Ну и я начал учить, не то чтобы прям нырнул, но постепенно ~~иду ко дну~~ погружаюсь, разбираюсь.

Рекомендую к просмотру все эти видео, очень интересно. А тут я распишу основы работы с turtle и интересные моменты.

Писать все буду в своей любимой [PyCharm](https://www.jetbrains.com/ru-ru/pycharm/), а если во время выполнения возникают ошибка похожая на **ImportError: DLL load failed while importing \_tkinter**, то видимо у вас система 64 бит, а вот Python скачан для 32. Просто удалите его и [скачайте](https://www.python.org/downloads/windows/) заново, уточнив предварительно, какая у вас система.

**Импорт черепашки**

Ладно. Перейдем к самой Черепашке (Turtle). Создали новый проект, файлик, и приступаем.

Как и с любым модулем, для работы с **turtle** её нужно импортировать. Делаем это вот так:

|  |
| --- |
|  |
| import turtle | |

Далее создадим объект библиотеки **turtle**, который назван **Turtle**. Значение запишем в переменную **turtlePen**.

|  |
| --- |
|  |
| turtlePen = turtle.Turtle() | |

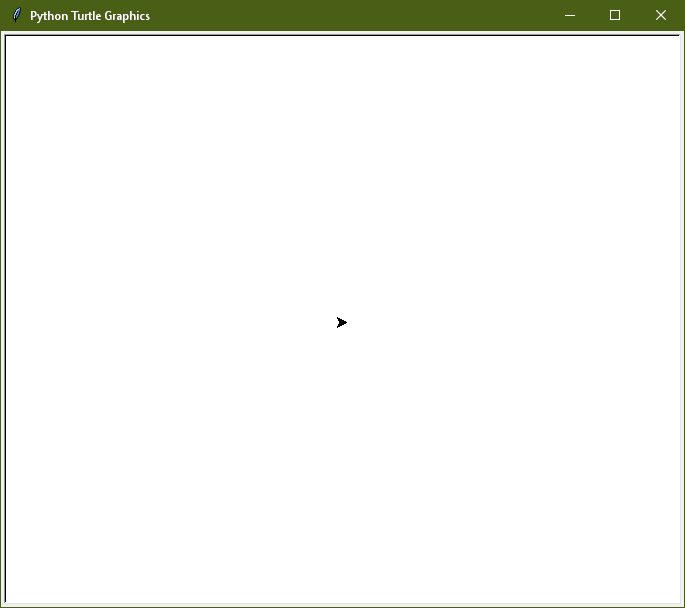
Попробуем запустить? Если все выполнено правильно, то вы должны увидеть белое окно, которое тут же исчезнет, так как программа завершена.

Добавим переменную **window**, которая будет содержать объект **Screen** (Не забывайте о **скобках ()**!). С его помощью можно изменить параметры этого самого окна, вроде цвета фона или возможность сохранить окно даже после всех операций отрисовки.

Последнее нам и нужно, для этого добавляем в самый конец такую команду:

|  |
| --- |
|  |
| import turtle    turtlePen = turtle.Turtle()  window = turtle.Screen()    window.mainloop()   # &lt;- не даст закрыть окно | |

Запускаем и видим:



Это прекрасно!

**Рисуем при помощи Turtle!**

Уже можно рисовать? Да!

Для этого управляем нашим **turtlePen**. Допустим, мы можем указать ему идти вперед на определенное расстояние.

|  |
| --- |
|  |
| turtlePen.forward(123) | |
|  | |

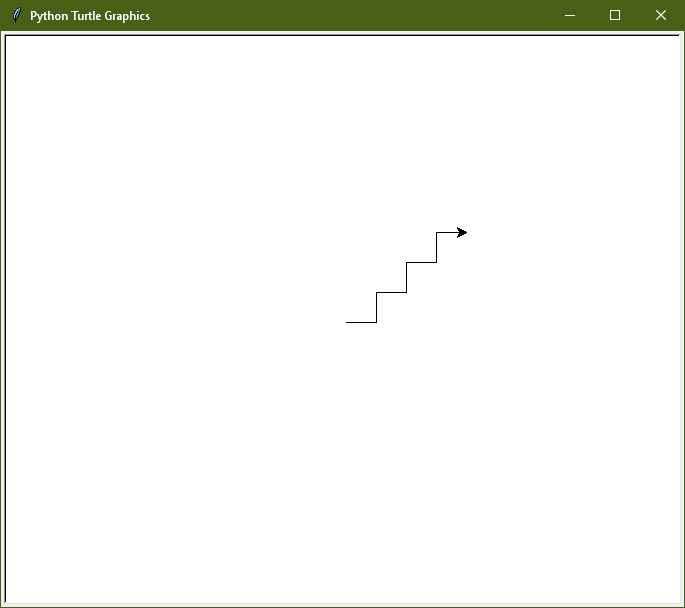
Я художник!

Вперед — это в правую сторону из центра? Любая черепашка начинает с координаты **(0;0)**, т.е. в центре экрана и смотрит в правую сторону.

Чтобы изменить направление движения используем команды **left** или **right**. Для примера нарисуем лесенку при помощи уже известных команд.

|  |
| --- |
|  |
| import turtle    turtlePen = turtle.Turtle()  window = turtle.Screen()    turtlePen.forward(30)  turtlePen.left(90)  turtlePen.forward(30)  turtlePen.right(90)  turtlePen.forward(30)  turtlePen.left(90)  turtlePen.forward(30)  turtlePen.right(90)  turtlePen.forward(30)  turtlePen.left(90)  turtlePen.forward(30)  turtlePen.right(90)  turtlePen.forward(30)    window.mainloop() | |

Выглядит кошмар, но на выходе получаем такую вот картинку:



Как и обещал

Можно увидеть, что у нас получилось очень много одинаковых команд, а потому для выполнения таких заданий разумно использовать циклы. А потому перепишем этот пример более корректно.

Обсудим скорость движения черепашки. При помощи команды **speed** мы можем изменять скорость рисования. Чем больше значение, тем быстрее рисует, но если укажем 0, то рисунок будет нарисован мгновенно.

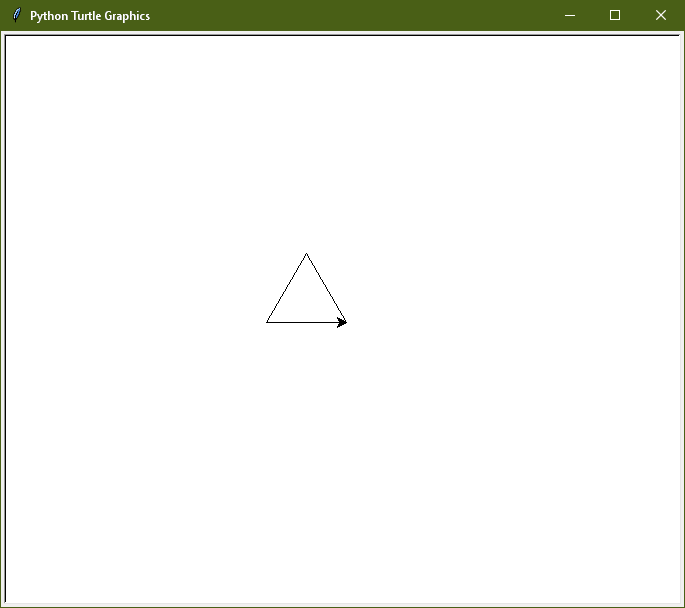
Итак, версия ступени версия 2.0:

|  |
| --- |
|  |
| import turtle    turtlePen = turtle.Turtle()  window = turtle.Screen()      def draw\_stairs(n, size=30):     # &lt;- создали функцию, для рисования      for i in range(0, n):        # &lt;- цикл нарисует указанное количество ступеней          turtlePen.left(90)          turtlePen.forward(size)          turtlePen.right(90)          turtlePen.forward(size)    turtlePen.speed(10)    draw\_stairs(8)   # &lt;- вызов функции    window.mainloop() | |

Уже лучше

Что насчет других геометрических фигур? Без проблем! Вот треугольник!

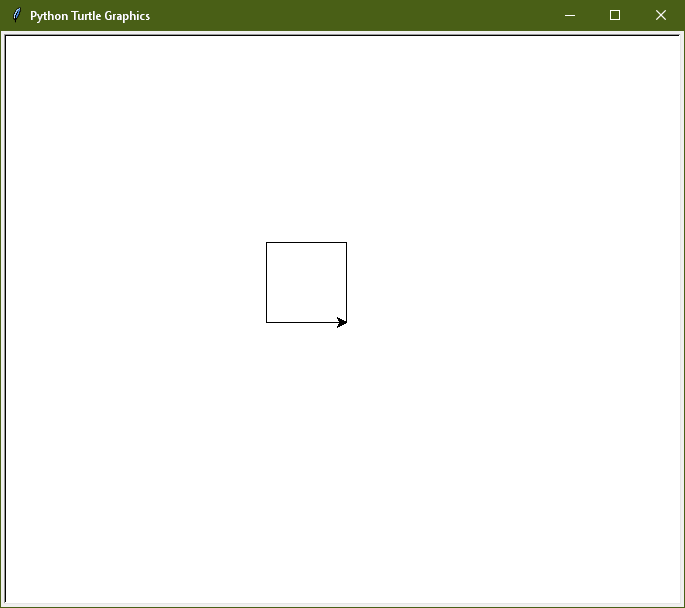
|  |
| --- |
|  |
| turtlePen.left(120)  turtlePen.forward(80)  turtlePen.left(120)  turtlePen.forward(80)  turtlePen.left(120)  turtlePen.forward(80) | |



Похож ведь?

А это квардат:

|  |
| --- |
|  |
| turtlePen.left(90)  turtlePen.forward(80)  turtlePen.left(90)  turtlePen.forward(80)  turtlePen.left(90)  turtlePen.forward(80)  turtlePen.left(90)  turtlePen.forward(80) | |



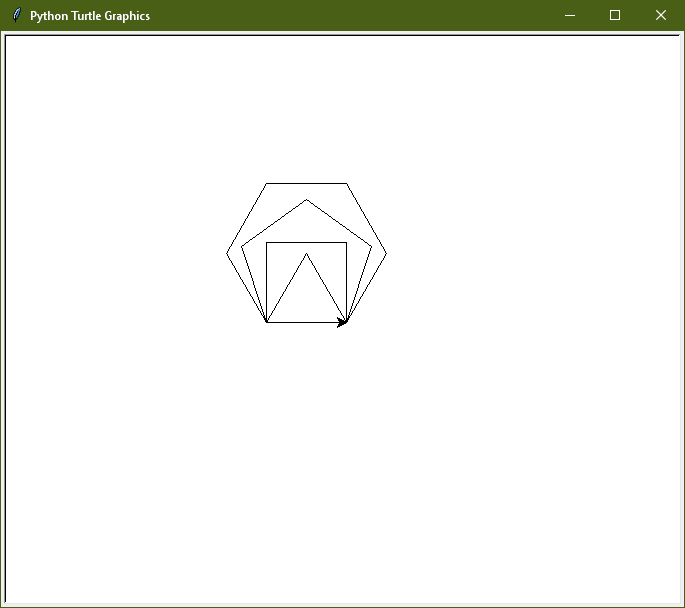
Стороны равны, гарантирую

Внимательные могли заметить, что и для **правильного** треугольника, и для квадрата, мы в сумме получаем поворот на 360 градусов, а значит можно создать функцию рисования правильных многоугольников!

Вот она:

|  |
| --- |
|  |
| def polygon(n, size=80):      if n > 2:                          # &lt;- многоугольников меньше 3 углов я не знаю :)          angle = 360/n                  # &lt;- получаем угол поворота в зависимости от количества углов            for n in range(0, n):         # &lt;- рисуем стороны              turtlePen.left(angle)              turtlePen.forward(size)    # рисуем разные фигуры  polygon(3)  polygon(4)  polygon(5)  polygon(6) | |

Для примера я выведу фигуры с 3, 4, 5 и 6 углами.

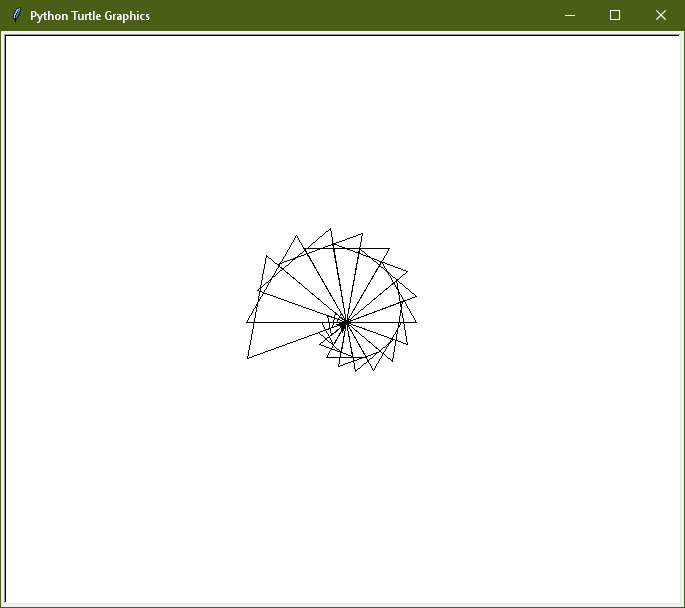


Все работает!

Если заметили, то я везде оставлял возможность менять длину стороны для каждой фигуры (2 параметр у каждой функции). Это вполне можно использовать для создания интересных узоров!

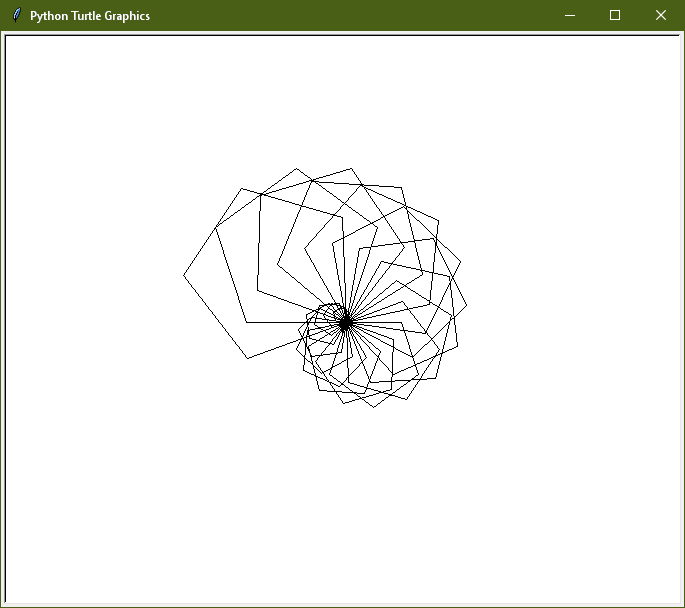
К примеру:

|  |
| --- |
|  |
| import turtle    turtlePen = turtle.Turtle()  window = turtle.Screen()      def polygon(n, size=80):      if n > 2:          angle = 360/n            for n in range(0, n):              turtlePen.left(angle)              turtlePen.forward(size)      turtlePen.speed(10)    for i in range(0, 100, 5):      polygon(3, 10 + i)      turtlePen.left(20)    window.mainloop() | |



Один из вариантов…

Изменим треугольники на пятиугольники, и получим:

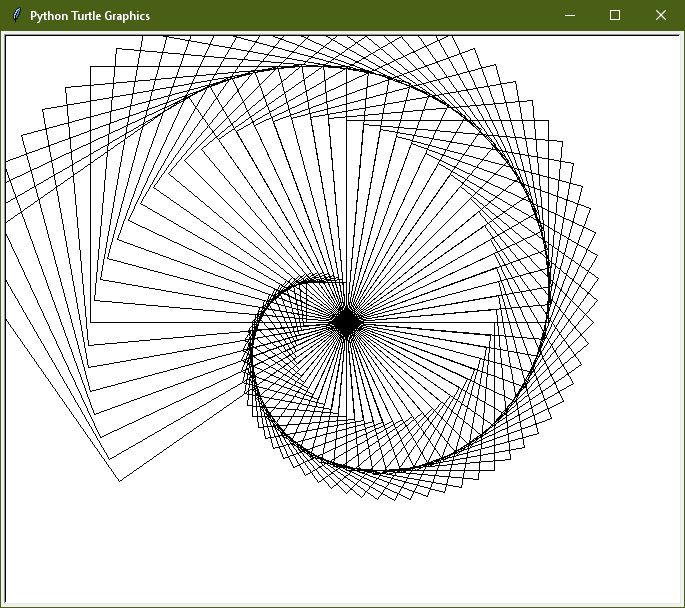


Все так же, но иначе

А этот цикл:

|  |
| --- |
|  |
| size = 40    for i in range(0, 80):      polygon(4, size)      turtlePen.left(5)      size = size + 3 | |

Нарисует:



Нужно будет переместить центр…

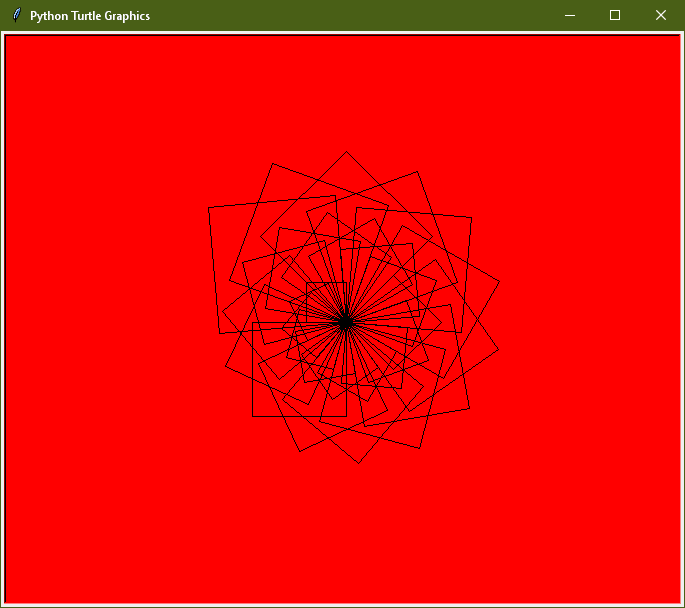
Иными словами — **экспериментируйте**! Так можно получить самые интересные изображения.

А пока добавим всему этому красок!

**Используем цвета**

Фон окна можно изменить командой **bgcolor**, куда передаем имя текста в виде строки:

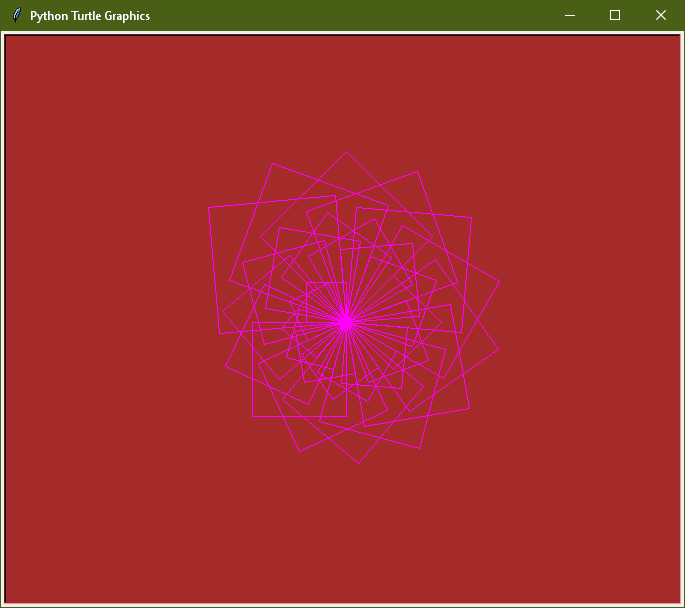
|  |
| --- |
|  |
| window.bgcolor("red") | |



Выбор явно не лучший, но для примера сойдет ^\_^. Далее буду использовать **brown**

Аналогично при помощи команды **color** указываем цвет для рисования:

|  |
| --- |
|  |
| turtlePen.color("magenta") | |

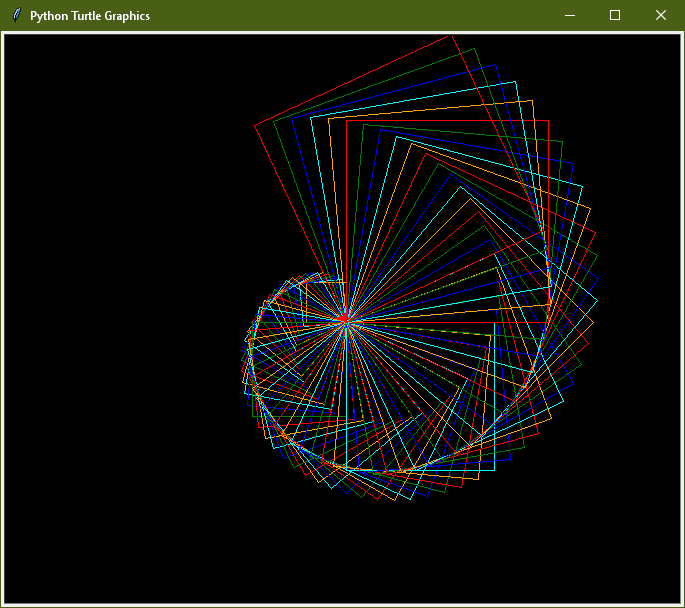


Думаю вполне неплохо

Цвет можно менять и во время рисования. Для этого создадим массив цветов и в зависимости от значения цикла (или иных параметров) будем менять цвет линий.

|  |
| --- |
|  |
| import turtle    turtlePen = turtle.Turtle()  window = turtle.Screen()    window.bgcolor("black")      def polygon(n, size=80):      if n > 2:          angle = 360 / n            for n in range(0, n):              turtlePen.left(angle)              turtlePen.forward(size)      turtlePen.speed(100)    colors = ['orange', 'cyan', 'blue', 'green', 'red']    size = 40    for i in range(0, 60):      turtlePen.color(colors[i % 5])      polygon(4, size)      turtlePen.left(5)      size = size + 3    window.mainloop() | |

Цвет фона я сделал черным, чтобы максимально рассмотреть это цветовое безумие.



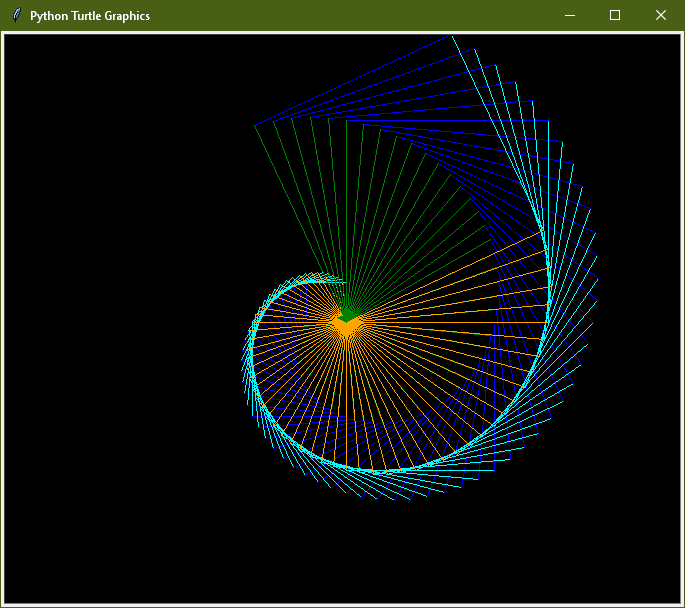
Все интереснее!

Если же вы хотите разрисовать конкретные стороны своих фигур, то цвет нужно указать в цикле отрисовки сторон (и убрать из цикла отрисовки фигур).

В итоге функция **polygon** должна выглядеть приблизительно так:

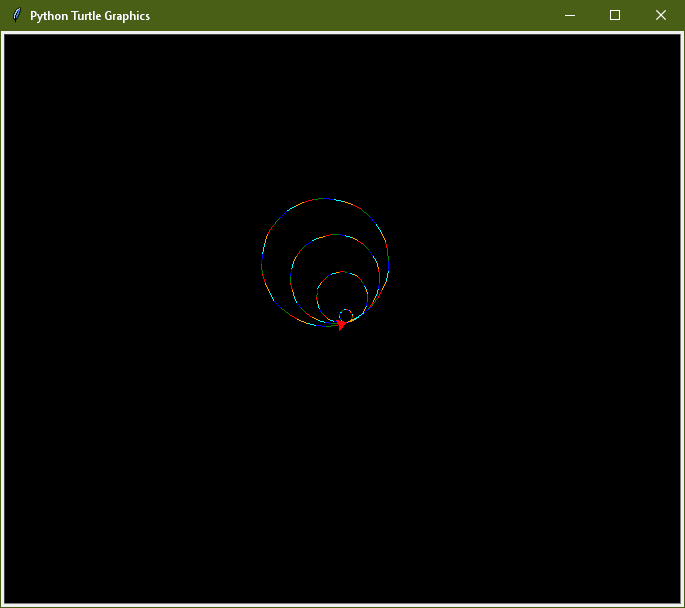
|  |
| --- |
|  |
| def polygon(n, size=80):      if n > 2:          angle = 360 / n            for n in range(0, n):              turtlePen.color(colors[n % 5])              turtlePen.left(angle)              turtlePen.forward(size) | |

А изображение получит совершенно новый вид:



Так даже круче!

Напоследок вспомним о кругах. Ведь наша программа рисует только многоугольники. И хотя можно указать какой-то 40-угольник, который в итоге даже будет похож на круг:

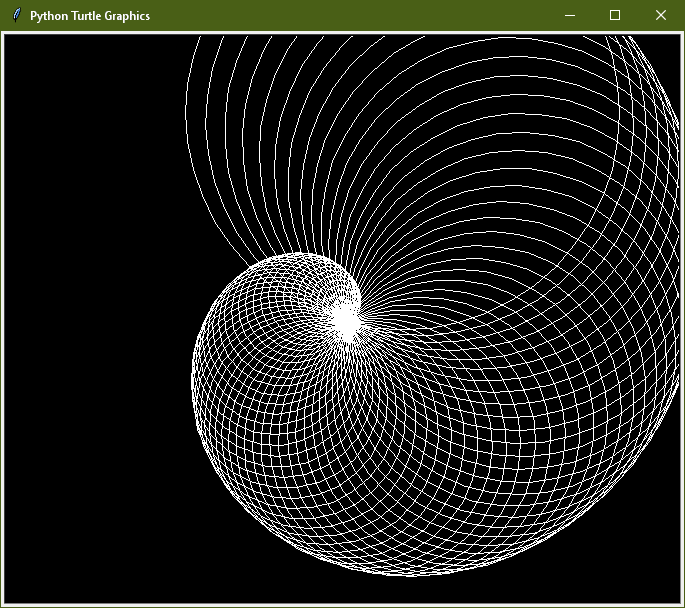


Ну почти. Зато красиво!

В программе есть для этого отдельная функция: **circle**. И заменив функцию **polygon** в цикле отрисовки:

|  |
| --- |
|  |
| for i in range(0, 70):      turtlePen.color("white")      turtlePen.circle(size)      turtlePen.left(5)      size = size + 3 | |

Получаем очередной интересный узор, но только с кружками:



Лучше так, чем 40 угольник!