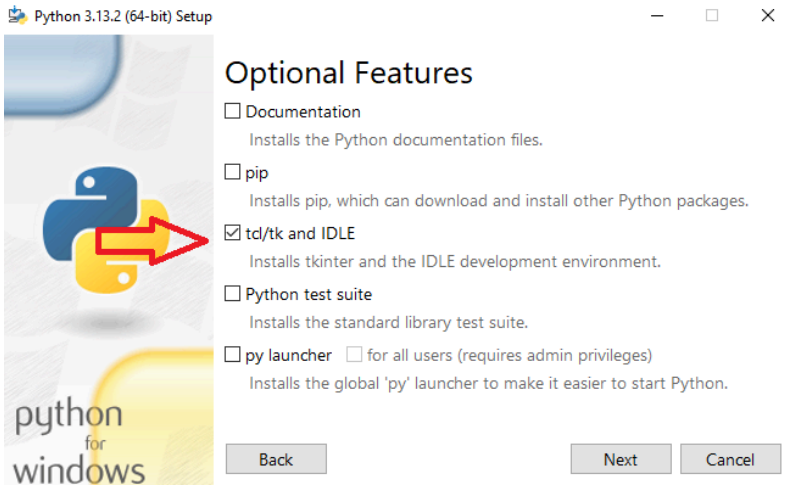
Задание №6, Черепашка

Теория

Черепаха (turtle, turtle graphics) - графический исполнитель, следующий командам для перемещения и изменения параметров.

> В Python представляет собой встроенную библиотеку **turtle**, для работы которой необходим **tkinter:**



> *Если эта галочка не была нажата при первой установке, то запустите установщик ещё раз, выберите Modify и добавьте её, далее выполняйте действия как при обычной установке*

Исполнитель **Черепаха** перемещается на экране компьютера, оставляя след в виде линии, если предварительно была выполнена команда **Опустить хвост,** при команде **Поднять хвост** черепаха перестаёт оставлять линию. По умолчанию у черепахи хвост опущен, а её голова направлена вдоль положительной оси абсцисс (направо). Команды **Вперед** и **Назад** непосредственно двигают Черепаху вдоль направления головы, а **Налево** и **Направо** - поворачивают голову на указанное количество градусов в указанном направлении.

📖 Соотнесение команд из условия с их написанием на языке программирования (k - это масштаб):

Задание	Кумир	Python
опустить хвост	опустить хвост	pendown()
поднять хвост	поднять хвост	penup()
вперед x	вперед(x)	forward(x * k)
назад x	назад(x)	back(x * k)
направо x	вправо(x)	right(x)
налево x	влево(x)	left(x)
повтори x [действия]	нц x раз	for i in range(x):
	действия	действия
	кц	

Также важно знать следующие команды:

done() - ставится в самом конце, чтобы при завершении движения не закрывалось графическое окно.

tracer(0) - черепаха завершит своё движение моментально, но часто работает некорректно. Стоит использовать только при построении координат.

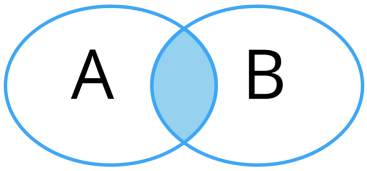
speed(3000) - ускоряет движение черепахи. Помогает, когда tracer не работает.

screensize(3000,3000) - позволяет перемещаться по графическому окну, используя ползунки справа и снизу.

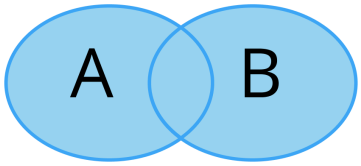
goto(x * k, y * k) - перемещение черепахи на указанные координаты. Используется для построения координат.

dot(2-5) - ставит точку указанного размера

Пересечение A и B (**A ∩ B)** - множество общих элементов, т.е. всех элементов, принадлежащих и множеству A, и множеству B.



Объединение A и B (**A ∪ B**)- множество элементов, принадлежащих множеству A, или множеству B, или обоим множествам.



🔥

Формулы подсчёта точек

Часто можно понять, какая именно фигура подразумевается в условии, можно и без написания программы. В случае прямоугольников, точки легко с помощью известных формул нахождения площади и периметра.

Важно: длину и ширину мы считаем начиная с 0, т.е. первая точка - это нулевая точка.

Пусть A и B - это длина и ширина прямоугольника.

Периметр: **P = (A + B) * 2**

Площадь или точки внутри фигуры:

Не считая точки на границах фигуры: **S = (A - 1) * (B - 1)**

Считая точки на границах фигуры: **S = (A + 1) * (B + 1)**

Объединение фигур: S1 + S2 - S3, где S3 - это площадь пересечения. В зависимости от условия, либо все площади будут учитывать точки на границах, либо нет

План решения

демо-вариант 2025

Исполнитель Черепаха действует на плоскости с декартовой системой координат. В начальный момент Черепаха находится в начале координат, её голова направлена вдоль положительного направления оси ординат, хвост опущен. При опущенном хвосте Черепаха оставляет на поле след в виде линии. В каждый конкретный момент известно положение исполнителя и направление его движения. У исполнителя существует 6 команд: **Поднять хвост**, означающая переход к перемещению без рисования; **Опустить хвост**, означающая переход в режим рисования; **Вперёд *n*** (где n – целое число), вызывающая передвижение Черепахи на n единиц в том направлении, куда указывает её голова; **Назад *n*** (где n – целое число), вызывающая передвижение в противоположном голове направлении; **Направо *m*** (где m – целое число), вызывающая изменение направления движения на m градусов по часовой стрелке, **Налево *m*** (где m – целое число), вызывающая изменение направления движения на m градусов против часовой стрелки.
Запись **Повтори *k* [Команда1 Команда2 … КомандаS]** означает, что последовательность из S команд повторится k раз.

Черепахе был дан для исполнения следующий алгоритм:
Повтори 9 [Вперёд 22 Направо 90 Вперёд 6 Направо 90]
Поднять хвост
Вперёд 1 Направо 90 Вперёд 5 Налево 90
Опустить хвост
Повтори 9 [Вперёд 53 Направо 90 Вперёд 75 Направо 90]

Определите периметр области пересечения фигур, ограниченных заданными алгоритмом линиями.

Построение фигуры происходит по одному и тому же алгоритму. Отличаются только способы подсчитать нужное количество точек.

1. Для начала выполним вспомогательные команды, для каждой задачи они почти одни и те же:
    a. Укажем **k**, который будет отвечать за масштаб. Чем он больше, тем больше будет фигура. Чтобы не терять пропорции, все последующие

движения Вперед, Назад, а также построение координат будем умножать на этот коэффициент.

b. **screensize(3000,3000)** - позволит нам перемещаться по окну. Очень помогает, когда фигура выходит за границы первоначального окна.

c. **tracer(0)** - т.к. мы будем ещё строить координаты-точки, то tracer нам подходит, но в других случаях лучше использовать speed(3000).

d. **left(90)** - по умолчанию в python голова черепахи направлена вдоль оси абсцисс, а в условии задачи всегда вдоль оси ординат. Поэтому разворачиваем в правильном направлении.

```
k = 10
screensize(3000,3000)
tracer(0)
left(90)
```

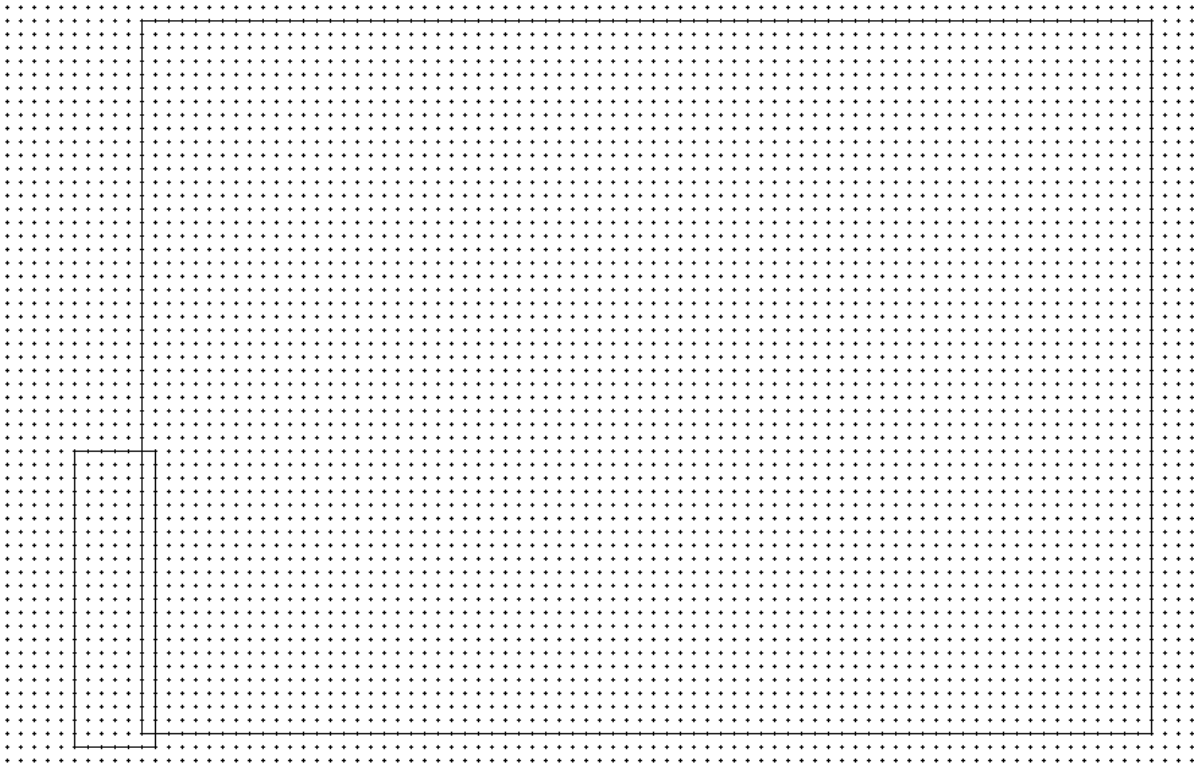
2. Выполняем алгоритм из условия:

```
for i in range(9):
    forward(22*k)
    right(90)
    forward(6*k)
    right(90)
penup()
forward(1*k)
right(90)
forward(5*k)
left(90)
pendown()
for i in range(9):
    forward(53*k)
    right(90)
    forward(75*k)
    right(90)
```

3. Строим координаты в виде точек:

```
penup()
for x in range(-60, 100): # range подбираем "на ощупь"
    for y in range(-60, 60): # range подбираем "на ощупь"
        goto(x*k,y*k)
        dot(3)
```

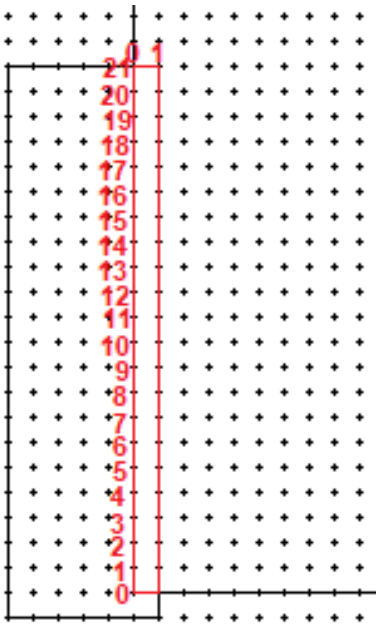
4. Не забываем дописать в конце **done()**, чтобы окно не закрывалось. Запускаем программу и получаем следующие фигуры:



```
# Вспомогательные команды
k = 10
screensize(3000,3000)
tracer(0)
left(90)
# Алгоритм из условия
for i in range(9):
```

```
    forward(22*k)
    right(90)
    forward(6*k)
    right(90)
penup()
forward(1*k)
right(90)
forward(5*k)
left(90)
pendown()
for i in range(9):
    forward(53*k)
    right(90)
    forward(75*k)
    right(90)
# Координаты:
penup()
for x in range(-60, 100):
    for y in range(-60, 60):
        goto(x*k,y*k)
        dot(3)
done()
```

5. Пересечение фигур выделено красным цветом. Это прямоугольник с длиной 21 и шириной 1. Находим периметр: (21 + 1) * 2 = **44**. Это и будет ответом



Примечание: в большинстве задач фигуры небольшие и точки легко можно подсчитать вручную по плану выше. Для достаточно больших прямоугольников нужно использовать формулы для длин сторон из условия. Для треугольников и других нестандартных фигур лучшим способом будет искать точки через уравнения прямых, либо через заливку. Решения этими способами можно посмотреть в разделе “Алгоритм решения”.

🖌️ Метод заливки

Внимание: метод корректно работает только при достаточно больших k > ~60, при использовании speed вместо tracer и не очень подходит для нахождения пересечений, т.к. нужно существенно изменить код.

1. Убираем у циклов лишние итерации. Часто в условии даётся намного больше повторений, чем нужно. Они ломают заливку. Нужно ровно столько итераций, чтобы фигура замкнулась (достроилась).
2. Для циклов (почти всегда именно циклы и строят фигуры) перед началом пишем **begin_fill()**, а после окончания - **end_fill()**.
3. До того, как строить координаты (хотя они для этого способа и не нужны), пишем следующий код, в котором range для циклов должен охватывать всю фигуру и где в условном операторе пишем "**== (5,)**", если мы не должны считать точки на линиях, и "**!= ()**", если должны:

```
canvas = getcanvas()
cnt = 0
for x in range(-200, 200):
    for y in range(-200, 200):
        if canvas.find_overlapping(x*k, y*k, x*k, y*k) == (5,): # != ()
            cnt += 1
print(cnt)
```

4. Если все сделано правильно, в консоли должен быть выведен правильный ответ, но лучше перепроверить его другим способом.

Алгоритм решения

A7F8FA, руками

Исполнитель Черепаха действует на плоскости с декартовой системой координат. В начальный момент Черепаха находится в начале координат, её голова направлена вдоль положительного направления оси ординат, хвост опущен. При опущенном хвосте Черепаха оставляет на поле след в виде линии. В каждый конкретный момент известно положение исполнителя и направление его движения. У исполнителя существует 6 команд: **Поднять хвост**, означающая переход к перемещению без рисования; **Опустить хвост**, означающая переход в режим рисования; **Вперёд *n*** (где *n* - целое число), вызывающая передвижение Черепахи на *n* единиц в том направлении, куда указывает её голова; **Назад *n*** (где *n* - целое число), вызывающая передвижение в противоположном голове направлении; **Направо *m*** (где *m* - целое число), вызывающая изменение направления движения на *m* градусов по часовой стрелке, **Налево *m*** (где *m* - целое число), вызывающая изменение направления движения на *m* градусов против часовой стрелки.

Запись **Повтори *k* [Команда1 Команда2 ... КомандаS]** означает, что последовательность из *S* команд повторится *k* раз.

Черепахе был дан для исполнения следующий алгоритм:

Повтори 9 [Вперёд 27 Направо 90 Вперёд 30 Направо 90]

Поднять хвост

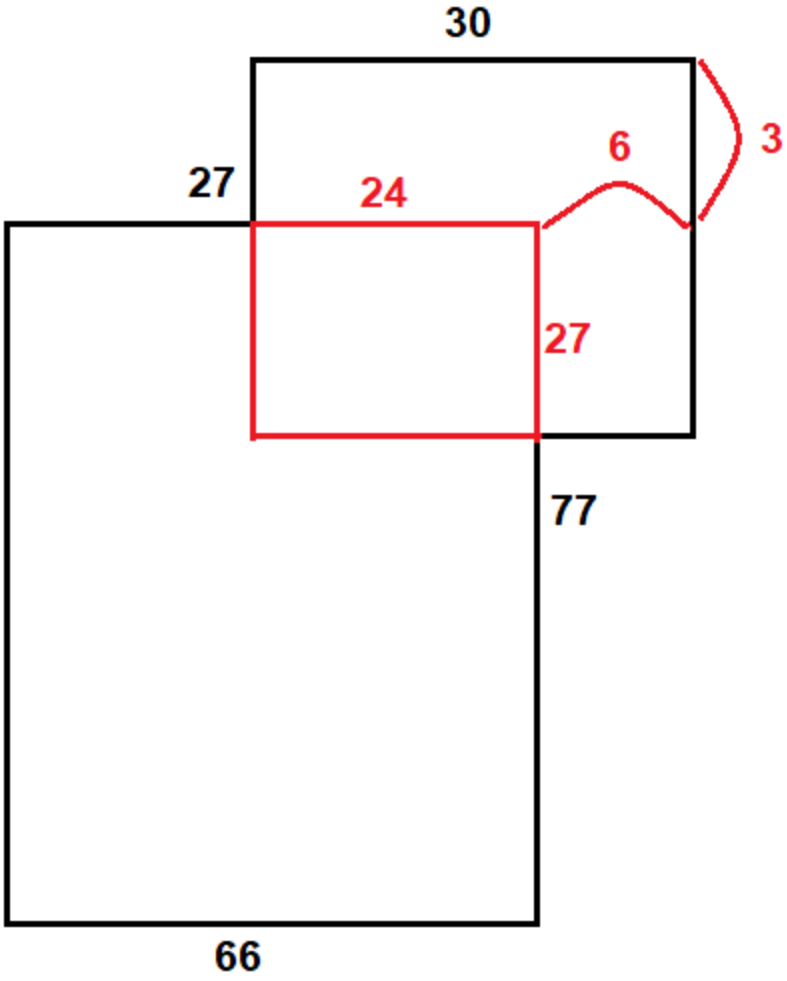
Вперёд 3 Направо 90 Вперёд 6 Налево 90

Опустить хвост

Повтори 9 [Вперёд 77 Направо 90 Вперёд 66 Направо 90]

Определите периметр области пересечения фигур, ограниченных заданными алгоритмом линиями.

1. Вручную воспроизводим алгоритм и строим фигуры, подписывая длины сторон. По поворотам на 90 градусов очевидно, что это будут 2 прямоугольника с сторонами, указанными в командах Вперед:



2. Получившийся красный квадрат со стороной 24 - это пересечение двух исходных прямоугольников. Его **периметр равен 24 * 4 = 96.** Это и будет ответ.
3. Код для подсчёта точек вручную:

```
tracer(0)
screensize(3000,3000)
k = 10
pendown()
left(90)
for i in range(9):
    forward(27*k)
    right(90)
```

```
    forward(30*k)
    right(90)
penup()
forward(3*k)
right(90)
forward(6*k)
left(90)
pendown()
for i in range(9):
    forward(77*k)
    right(90)
    forward(66*k)
    right(90)
pu()
for x in range(-60,40):
    for y in range(-60,30):
        goto(x*k, y*k)
        dot(3)
done()
```

B99881 объединение, руками/заливкой

Исполнитель Черепаха действует на плоскости с декартовой системой координат. В начальный момент Черепаха находится в начале координат, её голова направлена вдоль положительного направления оси ординат, хвост опущен. При опущенном хвосте Черепаха оставляет на поле след в виде линии. В каждый конкретный момент известно положение исполнителя и направление его движения. У исполнителя существует 6 команд: **Поднять хвост**, означающая переход к перемещению без рисования; **Опустить хвост**, означающая переход в режим рисования; **Вперёд *n*** (где *n* - целое число), вызывающая передвижение Черепахи на *n* единиц в том направлении, куда указывает её голова; **Назад *n*** (где *n* - целое число), вызывающая передвижение в противоположном голове направлении; **Направо *m*** (где *m* - целое число), вызывающая изменение направления движения на *m* градусов по часовой стрелке, **Налево *m*** (где *m* - целое число), вызывающая изменение направления движения на *m* градусов против часовой стрелки.

Запись **Повтори *k* [Команда1 Команда2 ... КомандаS]** означает, что последовательность из *S* команд повторится *k* раз.

Черепахе был дан для исполнения следующий алгоритм:

Повтори 3 [Вперёд 27 Направо 90 Вперёд 12 Направо 90]

Поднять хвост

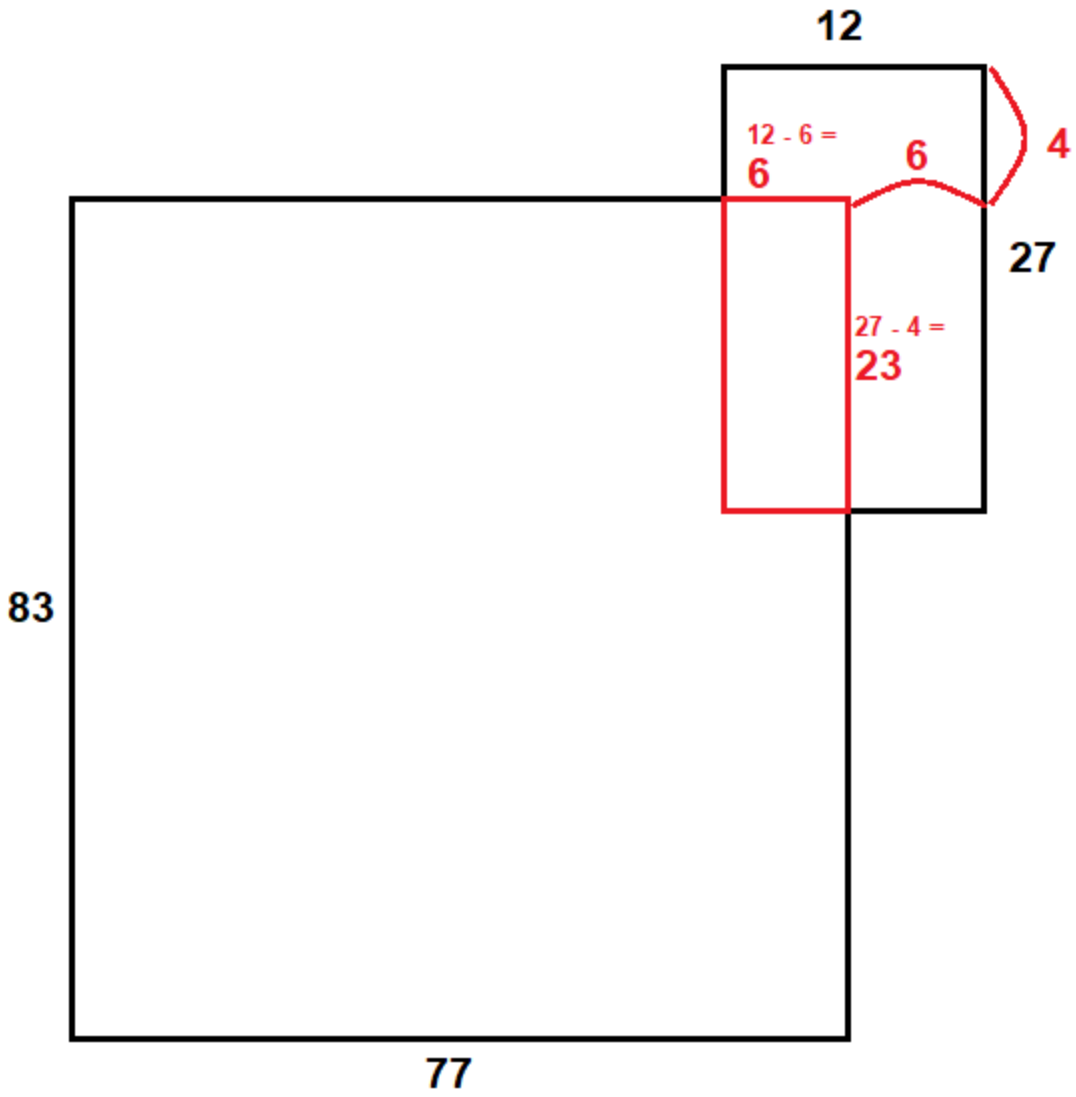
Вперёд 4 Направо 90 Вперёд 6 Налево 90

Опустить хвост

Повтори 3 [Вперёд 83 Направо 90 Вперёд 77 Направо 90]

Определите, сколько точек с целочисленными координатами будут находиться внутри объединения фигур, ограниченных заданными алгоритмом линиями, включая точки на границах этого объединения.

1. Вручную воспроизводим алгоритм и строим фигуры, подписывая длины сторон. По поворотам на 90 градусов очевидно, что это будут 2 прямоугольника с сторонами, указанными в командах Вперед:



2. Нам нужно посчитать количество точек объединения этих фигур, включая точки на границах. Для этого сложим площади этих фигур и вычтем площадь пересечения, представляющую собой красный прямоугольник со сторонами 6 и 23. Используем формулу: **S = (A + 1) * (B + 1)**

   (83 + 1) * (77 + 1) + (12 + 1) * (27 + 1) - (6 + 1) * (23 + 1) = 6748

3. Перепроверяем ответ с помощью заливки. Для этого выполняем шаги из плана решения, не доходя до постройки координат. Выставляем коэффициент масштаба побольше, например, 100. Используем обязательно speed вместо tracer. Уменьшаем range для циклов, чтобы не было лишних итераций и заливка не ломалась. Для этих же циклов

запускаем заливку и используем шаблонный код подсчёта “залитых” точек, включая линии. Весь код выглядит так (строчки для заливки помечены #):

```
left(90)
speed(3000) # Вместо tracer
screensize(3000,3000)
k = 100 # Побольше
begin_fill() # Начало заливки 1-ой фигуры
for _ in range(2): # Уменьшаем до замыкания фигуры
    forward(27*k)
    right(90)
    forward(12*k)
    right(90)
end_fill() # Конец заливки
pu()
forward(4*k)
right(90)
forward(6*k)
left(90)
down()
begin_fill() # Начало заливки 2-ой фигуры
for _ in range(2): # Уменьшаем до замыкания фигуры
    forward(83*k)
    right(90)
    forward(77*k)
    right(90)
end_fill() # Конец заливки
canvas = getcanvas() #
cnt = 0 # Счётчик точек
for x in range(-300, 300): # Следим, чтобы вся фигура входила
    for y in range(-300, 300): # Следим, чтобы вся фигура входила
        if canvas.find_overlapping(x*k, y*k, x*k, y*k) != (): # Линии включ.
            cnt += 1 #
```

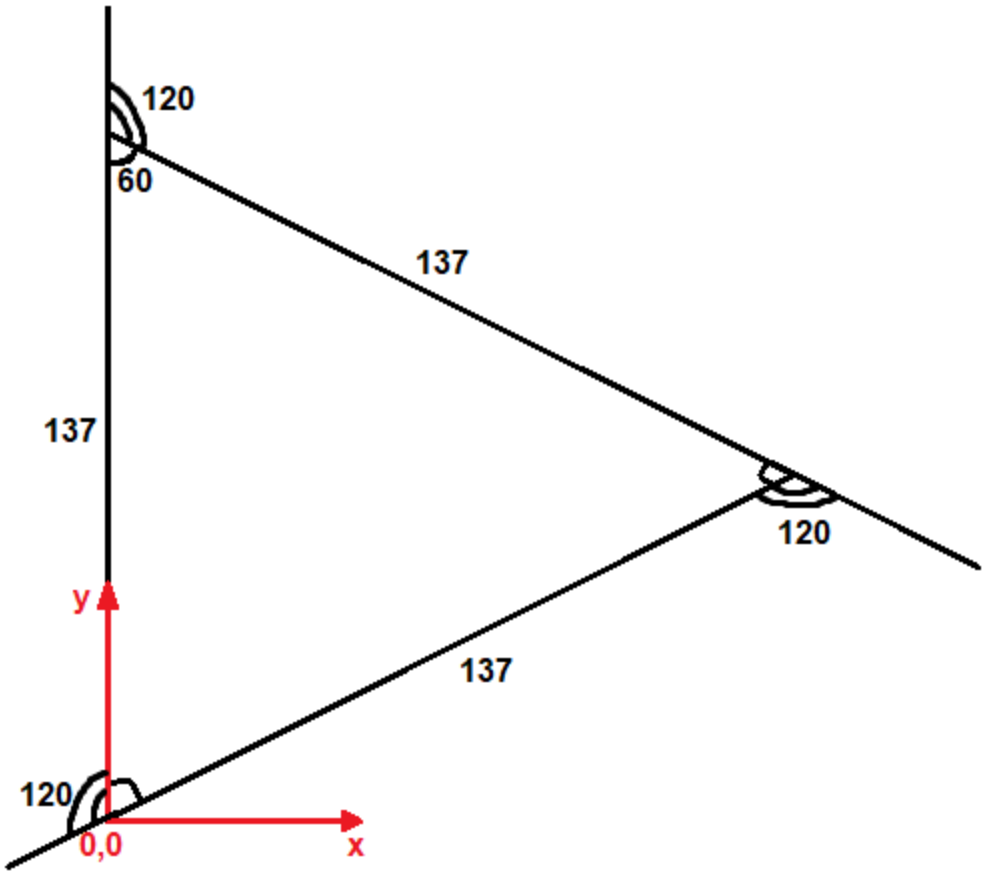
```
print(cnt) #
done()
```

4. Получаем тот же ответ: **6748**

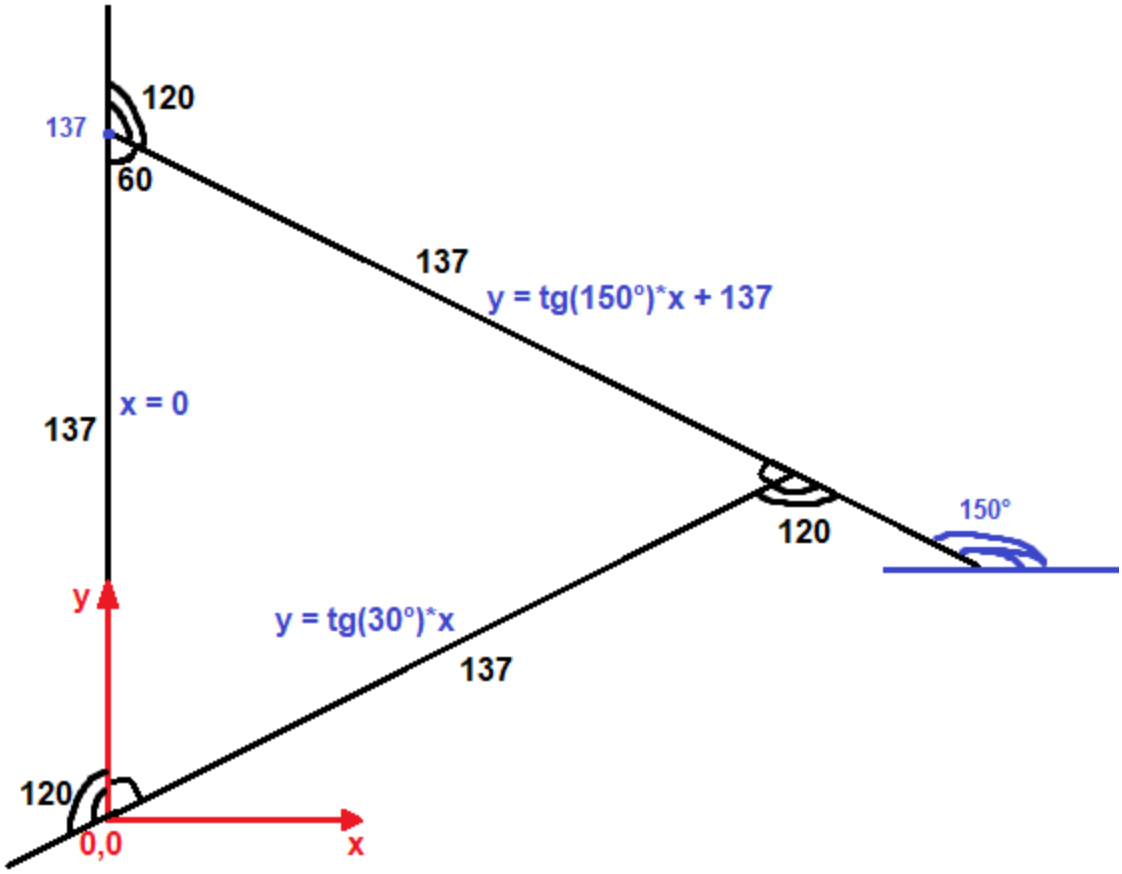
Треугольник, решение формулой/заливкой

Учебный алгоритм Черепаха работает на плоскости с декартовой системой координат. В начальный момент Черепаха находится в начале координат, а её голова направлена вдоль положительного направления оси ординат, хвост опущен. Если у Черепахи опущен хвост, то она оставляет следы на поле в виде линий. В каждый конкретный момент известно положение исполнителя и направление его движения.
У исполнителя существует две команды: **Вперёд n** (где n – целое число), вызывающая передвижение Черепахи на n единиц в том направлении, куда указывает её голова, **Направо m** (где m – целое число), вызывающая изменение направления движения на m градусов по часовой стрелке.
Запись **Повтори k [Команда1 Команда2 ... КомандаS]** означает, что последовательность из S команд повторится k раз.
Черепахе был дан для исполнения следующий алгоритм: **Повтори 21 [Вперед 137 Направо 120]**. Определите, сколько точек с целочисленными координатами будет находиться внутри области, ограниченной линией, заданной данным алгоритмом. Точки на линии учитывать не следует.

1. Вручную воспроизводим алгоритм и строим фигуру, подписывая длины сторон. По поворотам на 120 градусов очевидно, что это будет равносторонний треугольник со стороной 137. Обозначим точку старта черепашки за нулевую точку координатной плоскости.



2. Найдём функции прямых, образующих треугольник. Уравнение прямой: **y = k*x + b**, где k - это тангенс угла между прямой и положительной осью абсцисс (x), а b - точка пересечения прямой с осью ординат (y).



3. С помощью кода можем найти все точки, которые принадлежат фигуре, которую они образуют. Т.к. точки на линии учитывать не следует, то неравенства строгие. Ответ: **8058**

```
from math import *
cnt = 0
for y in range(-200, 200):
    for x in range(-10,200):
        if y < tan(radians(150))*x + 137 and y > tan(radians(30))*x and x > 0:
            cnt += 1
print(cnt)
```

4. Также можно перепроверить себя заливкой. Весь код:

```
k = 100
left(90)
```

```
speed(300)
screensize(3000,3000)
begin_fill()
for _ in range(3):
    forward(137*k)
    right(120)
end_fill()
canvas = getcanvas()
cnt = 0
for x in range(-100,200):
    for y in range(-200, 200):
        if canvas.find_overlapping(x*k, y*k, x*k, y*k) == (5,):
            cnt += 1
print(cnt)
done()
```