



Unit 4: Driving Features

Application: Custom Polygons

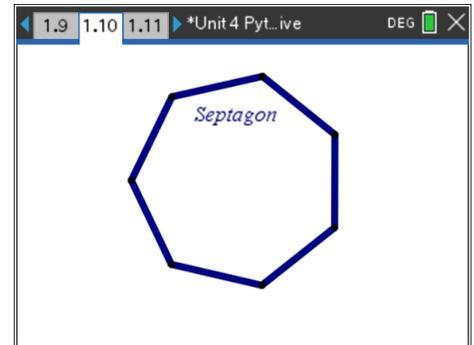
In this lesson, you will earn your Rover driver's license by designing a 'regular polygon' maker.

Objectives:

- Use **input()** statements to enter data for the number of vertices and the length of each side of a regular polygon
- Display lights along the sides and corners.

Now that you have successfully driven a square and a pentagonal path, you are ready to take your Rover driving test.

Create a program in which you enter the *number of vertices* and the *length of each side* of a regular polygon, drive that route, and light up the sky with a dazzling array of colors along the way. In addition to simply moving Rover, your program will seek input from the user, drive the proper distance for each side, and calculate the proper angle to turn at each vertex.



Teacher Tip: The turn angle is $360/(\text{number of vertices})$ degrees since the sum of the exterior angles of a polygon is 360 degrees.

1. Make a copy of your pentagon program from the last lesson. Your code should be similar to the image to the right. You will make some additions and changes to this code.

```

1.9 1.10 1.11 *Unit4 Pyt_ive DEG
#=====
# colorful pentagon:
for i in range(5):
  rv.color_rgb(0, 10 + 60 * i, 0) # green sides
  rv.forward(1)
  rv.wait_until_done()
  rv.color_rgb(10 + 60 * i, 0, 0) # red vertices
  rv.left(72)
  rv.wait_until_done()
rv.color_rgb(0,0,0)
  
```

2. Before the **for** loop, write two **input** statements to enter the number of vertices (**n**) and the length of each side (**s**).

n = (*an input statement*)
s = (*another input statement*)

It is *good* to use more informative variable names like **vertices** and **length** but be careful not to use Python reserved words.

```

1.9 1.10 1.11 *Unit4 Pyt_ive RAD
#=====
# colorful polygon:
from time import *
n= .....
s= .....
for i in range(5):
  rv.color_rgb(0, 10 + 60 * i, 0) # green sides
  rv.forward(1)
  rv.wait_until_done()
  rv.color_rgb(10 + 60 * i, 0, 0) # red vertices
  rv.left(72)
  
```



3. There are three values in the loop statements that need to change:

range(?), forward(?), and left(?)

After editing those three arguments, run your program, enter values for the input statements, and test your program.

Watch Rover carefully or insert a marker in the marker holder and draw the polygon on paper.

If you used an expression for the brightness of the LED along each side, then you will also have to adjust those LED statements to account for the change in the number of vertices. It would be better to use a variety of colors, not just red and green. Try random colors!

```

1.9 1.10 1.11 *Unit 4 Pyt...ive RAD 21/21
*u4app.py
S= .....
# colorful polygon:
for i in range(5):
  rv.color_rgb(0, 10 + 60 * i, 0) # green sides
  rv.forward(1)
  rv.wait_until_done()
  rv.color_rgb(10 + 60 * i, 0, 0) # red vertices
  rv.left(72)
  rv.wait_until_done()
rv.color_rgb(0,0,0)

```

Teacher Tip: Possible solution:

```

1.9 1.10 1.11 *Unit 4 Pyt...ive RAD 21/21
*u4app.py
=====
n= int(input("Vertices? "))
s= int(input("Length of sides? "))
# colorful polygon:
for i in range(n):
  rv.color_rgb(0, 10 + 245 / n * i, 0) # green
  rv.forward(s)
  rv.wait_until_done()
  rv.color_rgb(10 + 245 / n * i, 0, 0) # red
  rv.left(360 / n)
  rv.wait_until_done()
rv.color_rgb(0,0,0)

```

The LED can be tricky; avoid going out of range.

Need a challenge? Try programming *this* route:

