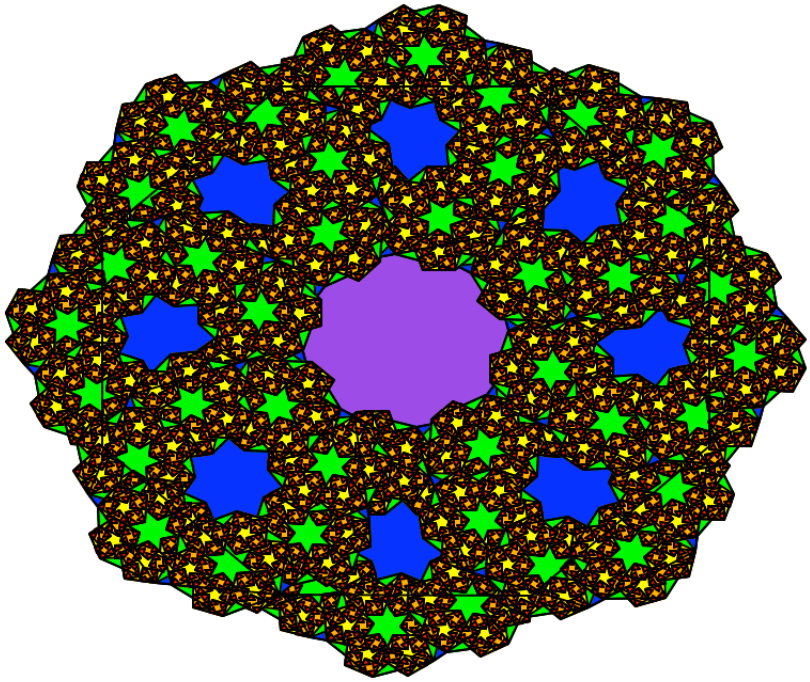
Computer Science AP/X — CSCI-140/242
Polygons — Lab 2

8/30/2018



Sides: 8, Fill:True

1 Implementation

You will now implement a program that handles the general case of drawing polygons of decreasing sides, recursively.

1.1 Program

Your program should be named `polygons.py`. When it is run on the command line it requires two arguments (in order):

1. `#_sides`: The number of sides of the main polygon (an integer). This value is guaranteed to be between `3 and 8`, inclusively.
2. `[fill|unfill|]`: If the string `fill` is specified, the polygons will be filled when drawn. Otherwise it is assumed the polygon will be unfilled and just a line drawing.

If the correct number of command line arguments is not supplied, a usage message should be displayed and the program should exit:

```
$ python3 polygons.py #_sides [fill|unfill]
```

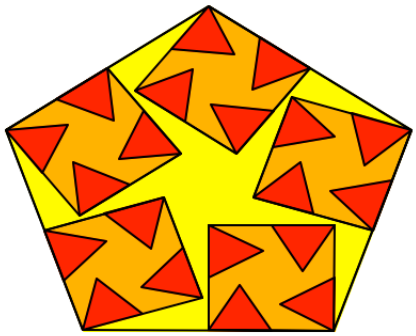
1.2 General Requirements

When the program is run, the main polygon that is drawn should be the number of sides supplied at run time. As the sides of the main polygon are drawn, recursion should be used to draw smaller polygons of decreasing size until the shape is a triangle.

When the programming finishes, the total length of all the sides of the polygons drawn should be displayed to standard output.

Here is an example of a potential image you might make if the program was run as (using `200 as the side length of the main polygon, and 50% as the reduction size for each sub-polygon`)

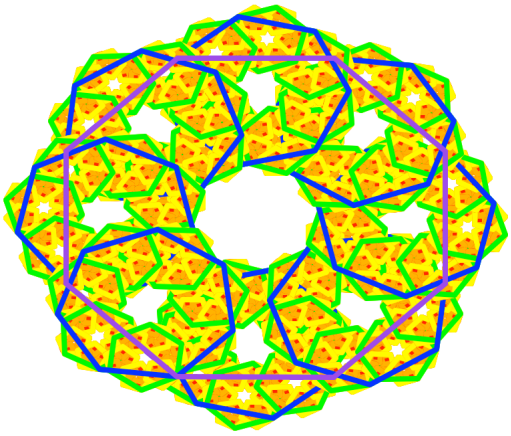
```
$ python3 polygons.py 5 fill
Sum: 60000
```



Sides: 5, Fill:True

Here is an example of an 8 sided unfilled polygon (using same constants as the previous):

```
Usage: $ python3 polygons.py 8 unfill
Sum: 276000.0
```



Sides: 8, Fill:False

We are encouraging you to be creative with this lab and make something that looks interesting using recursion and polygons.

Make sure that your name/s are displayed on the screen (using the `turtle.write`) command to write a string at a location in the window.

The program should pause when it is finished drawing so that we may admire your artistic masterpiece. Closing the window should end the program.

1.3 Code Specifics

1.4 Magic Numbers

Instead of hardcoding numbers everywhere you use them, we suggest you make "global constant variables". Of course Python doesn't have the notion of constants. As long as you do all your work in functions, and you don't use the `global` keyword, they will not be changeable.

For example, the instructor solution uses the following. Feel free to add others. This helps to reduce the amount of arguments you need to pass to your function/s.

```
# "constants" for the color used at each depth, e.g. depth=1 is 'red'
COLORS = 'red', 'orange', 'yellow', 'green', 'blue', 'blueviolet', 'violet'

# window dimensions
WINDOW_LENGTH = 800
SIDE_LENGTH = 200

# pen sizes to use for filled and unfilled polygons
FILL_PEN_WIDTH = 2
UNFILL_PEN_WIDTH = 8
```

1.5 Functions

Minimally you should have the following functions:

- A main function. It should be responsible for handling the command line arguments, usage message, invoking the recursive drawing function, and displaying the total side lengths.
- A function to initial turtle, e.g. `init`. This should set up the window, coordinate system and draw the team names.
- A recursive function to draw the polygons. You may find it useful to have other helper functions that the recursive one uses. Outside of the globals mentioned above, you should only have arguments for handling the polygon side counting, the length of the side of the polygon, and the fill state. **You are not allowed to use a global variable to count the total length drawn, It must be done recursively.**

1.6 Command Line

To specify the command line arguments using PyCharm, you will first need to run your program so you get a run configuration. Once you do this, the program will show up next to the green arrow and you can click/edit it. The field you want to change is called `Script parameters`. Here you should put the number of sides and fill state.

To read these in Python, you need to use the `sys` module. Inside there is a *Python list* called `argv` which will contain the script parameters. The catch is the first entry in `argv` contains your program name (full path). The script parameters will come as strings in the second and third entry, e.g.:

```
import sys

# imagine the program was run as: $ python3 polygons.py 5 fill
print(sys.argv)            # ['.../polygons.py', '5', 'fill]
len(sys.argv)              # 3
sides = int(sys.argv[1])   # sides = 5
```

1.7 Color

The `turtle.color()` method takes a string to use for the drawing color. Most of the basic colors are supported.

1.8 Filling Polygons

To fill a polygon, the format of the code should be:

```
turtle.begin_fill()
# draw shape
turtle.end_fill()
```

1.9 Speeding Up Drawing

Turtle can be very slow to animate. It is useful when you are developing and debugging, but eventually once you know things work you will want it to run faster. To turn off animation, use the command:

```
turtle.tracer(0,0)
```

Your program should end with the following two statements to update the drawing window and enter the main loop:

```
turtle.update()
turtle.mainloop()
```

2 Grading

This assignment will be graded using the following rubric:

- (15%) Problem Solving
- (10%) Design:
  - Functions are used to break up the problem.
  - Magic numbers are avoided.
  - There is no use of the `global` keyword.
  - The total length of all sides drawn is computed recursively.
- (65%) Functionality:
  - 5% Command line argument handling.

  - 10% Polygons are drawn and their sides count down recursively to a triangle.
  - 15% Program can handle filling in polygons.
  - 15% Program can handle unfilled polygons.
  - 10% How "interesting" are the final images drawn?
  - 10% Sum of all sides drawn is computed/displayed correctly.
- Version Control: 5%
- Code Style and Documentation: 5%

3 Submission

The try command to submit is:

```
try csapx-grd lab2-1 polygons.py log.txt
```

Recall that to verify your latest submission has been saved successfully, you can run try with the query option, -q:

```
try -q csapx-grd lab2-1
```