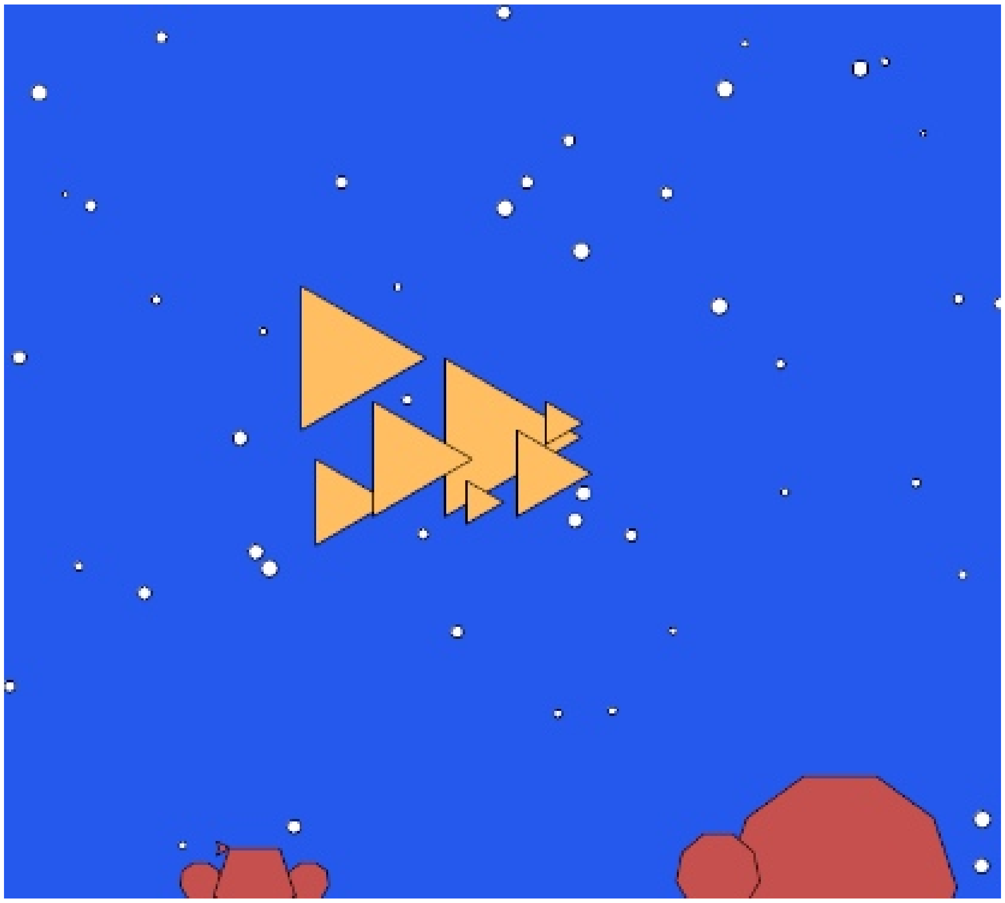
CS30 - Assignment 2

Due: Thursday February 5, at 11:59pm

For this assignment, we will be using the `turtle` graphics module to draw a picture. You will have two options, a seascape with fish and rocks or a space scene with spaceships and planets (come talk to me if you have an idea of another scene type that uses similar shapes). Below is an example completed picture:



1 Turtle documentation

A crucial programming paradigm is code reuse. Rather than writing your own code, say a `turtle` module, you can use one that someone has already written. An important part of coding then is documentation, both generating it for your programs and being able to understand other people's documentation.

The documentation for the `turtle` module can be found at:

`http://docs.python.org/library/turtle.html`

If you scroll down to section 24.5.2.2 you can see an overview of all of the functions available to you.

To get ready for this assignment, you're going to need to read a bit more about some of the functionality of the `turtle` module. Below are a few methods that may be useful. Click on each of them in the documentation and make sure you understand what they do as well as what parameters they take and what, if anything, they return.

- `speed`
- `set_heading`
- `fill_color`
- `begin_fill`
- `end_fill`
- `penup`
- `pendown`
- `bgcolor`
- `goto`
- `bye`

Two other methods that will be useful for us are `window_width` and `window_height`. Unfortunately, if you look at the documentation for these they say that they have been *deprecated*, which means discontinued. In general, it's not good to use deprecated functions, since eventually, they may not be around. In our case, however, we will find it useful to use these, so we'd like to find out more about them.

The `help` function takes as an argument the name of a function and outputs the docstring (i.e. function description) for us. Start up WingIDE then use `help` to get the documentation for these two functions. Remember you will need to import the `turtle` module first by typing `from turtle import *`.

2 Style

Before you start coding, a few brief comments on style. We've talked about style components in class and now we're going to start utilizing these in your assignments. In particular, make sure you keep the following in mind as you're writing your program:

- All functions should have appropriate docstrings
- Comment your code appropriately. Comment complicated parts of the code and include your name, date, etc. at the top of the file.
- Follow the variable naming conventions discussed in class and use good variable names
- Use whitespace appropriately to make your program easier to read

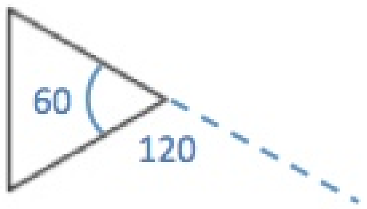
3 Basic Shapes

To get started, we're going to write some functions to make some basic shapes for us. Make sure that you have these working before moving on to the next part.

Don't forget to include the import statement for the `turtle` module at the top of your file:

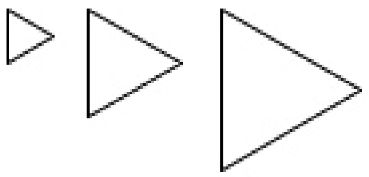
```
from turtle import *
```

- **triangle** - Write a method called `triangle` that draws an equilateral triangle (i.e. a triangle with all three sides the same length). Your function should take three parameters: the x and y coordinate where it should be drawn and the length of the side of the triangle. The triangle should be drawn so that the left edge of the triangle is vertical. For those rusty on geometry, the interior angles of an equilateral triangle are all 60 degrees, which means the angle between a straight line drawn from one side and the adjacent side is 120 degrees.



  Your x and y coordinates may indicate any part of the triangle (e.g. the top left, the center, etc.). Pick whichever seems easiest.

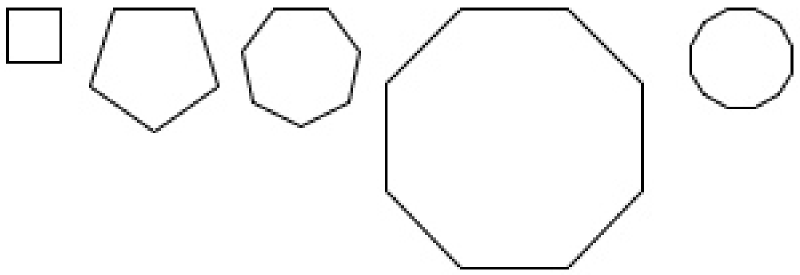
  For example, here are three triangles of increasing size:



- **polygon** - Write a method called `polygon` that draws a polygon. An n-sided polygon has n equal length edges and the angle between a straight line drawn from one side and the adjacent side is $360/n$. Your function should take 4 parameters: the x and y location of the polygon, the number of sides and the length of the each side. Again, the x and y coordinates may indicate any point on the polygon, so pick whatever seems easiest.

  Unlike the triangle where you know when you're writing the function exactly how many sides the object will have, for the polygon you can't just hard-code the different line segments. Instead, you'll have to use a `for` loop.

  For example, below are some polygons of differing number of sides and size. All can be drawn with the `polygon` function.



4 The Background

At this point, you should have two functions written that draw triangles and polygons anywhere on the screen. We'll now work on filling in the background.

The key component of the background is randomly spaced circles. Write a function called `add_circles` that takes as a parameter the number of circles to add and randomly places circles of radius 4 throughout the screen. Some hints for how to do this:

- The `circle` function draws a circle with a given radius
- You'll need to figure out the dimensions of the screen (look at section 1 again if this doesn't seem familiar)
- The `randint` function from the random module may be useful

- To make this available, don't forget to include `from random import randint` at the top of your program

To check that everything is working right, try adding differing numbers of circles and make sure that they're distributed throughout the screen and that the right number are actually being drawn.

Once you're sure it's working, add a bit more code to make the size of the circles random. You'll have to play with different size ranges to see what looks best. For example, if you look at the picture on the first page of this handout, you'll see that the bubbles range in size. Think about using constants here rather than hard-coding numbers in your code.

5 Putting it all together

You now have all the components required to put together your final picture. Create a function called `generate_picture` that draws the entire picture. One way to do this is:

1. Change the background to the appropriate color.
2. Change your `triangle` and `polygon` functions so that they create filled shapes (see the `turtle` documentation on how to do this if you don't remember).
3. Change your `add_circles` method to also draw filled circles.
4. Actually create your picture using your three functions. Your picture should include both triangles and polygons as well as some randomly filled circles using your `add_circles` function. I encourage you to get creative here!
5. Change the fill color of the objects to make it look more realistic (e.g. orange fish and brown rocks).

6 Conditional Creativity

We learned about conditionals (i.e. `if`-statements and their variants). Find at least 1-2 ways to incorporate conditionals into your program (if it's a more involved `if-elif-else` statement, you only need one, but if it's just simple `if` statements then you should have two). Here are a few ideas, but get creative!

- Have your fish/ships be random colors.
- Have your polygons be colored based on their size.

7 Extra Credit

You can earn up to 2 points of extra credit on this assignment. Extra credit will be awarded by including additional elements to your picture. The amount of points given will be assigned based on creativeness and difficulty of implementation. Here are some examples, but I encourage you to include your own:

- Instead of circles, write a function called `star` that draws stars (not asterisks) (use a `for` loop)
- Include other types of objects in your scene
- Modify the `triangle` function to draw more interesting triangles, for example isosceles triangles or triangles with fins

To receive extra credit you MUST include in your comments at the top of the program what the extra credit additions you added were (otherwise, it can be hard to figure out sometimes).

8 When you're done

When you're all done you should have a working program that draws your picture. Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course, assignment number and the date.
- Each function should have an appropriate docstring
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good *style*. This includes:

- Following naming conventions, e.g. all variables and functions should be lowercase.
- Using good variable names.
- Proper use of whitespace, including indenting and use of blank lines to separate chunks of code that belong together.

Submit your .py file online using the courses submission mechanism.

Grading

		points
`triangle`		
	equilateral	1
	correct x, y	1
	correct direction	1
`polygon`		
	correct shape	3
	varying sizes	1
	x, y	1
`add_circles`		
	correct number of circles	1
	covers entire area	1
	random locations	2
	random sizes	1
`generate_picture`		
	background color	1
	proper fills	1
	6+ fish/ships	2
	6+ rocks/planets	2
Use of conditionals		3
Comments, style		3
extra credit		2
total		25 (+2)