Object-oriented Motif Mark
Feb 19, 2021

Your assignment is to convert Motif Mark into object-oriented code!

I'm going to ask you to stick to what we hope is a straightforward and simple style of object-oriented programming.

I recommend that you do **not** try to "write it all at once, and when finished, see if it works!". Instead, I recommend you either:

a) Convert your previous Motif Mark code piecemeal, bit by bit, frequently testing to ensure that your code still works (i.e., produces pretty and correct looking motif mark figures), and fixing it if it doesn't.
b) Start from scratch, and start simple, implementing one thing at a time (e.g., just draw a single gene), and then more and more, frequently testing to ensure that your code still works (i.e., produces pretty and correct looking motif mark figures), and fixing it if it doesn't.

Also, keep historical copies of your code that you know work, either via lots of git commits, or what I do when I'm dealing with a completely new paradigm:

```
cp  my_code.py  my_code.WORKS_FOR_GENE___MOVING_ONTO_EXON.py
vim  my_code.py
```

You can always revert if things get out of hand.

```
mv  my_code.py  my_code.THIS_PUPPY_TOO_BROKEN.py
cp  my_code.WORKS_FOR_GENE___MOVING_ONTO_EXON.py  my_code.py
```

Create a repository in your GitHub profile called "motif-mark-oop". Copy your previous assignment into it as a starting point.

You're going to build multiple classes. Instead of telling you the order in which to implement things, I'll just give you the big picture.

Data

Gene class – A Gene object stores its start, length (in # of nucleotides), and maybe its "width" (i.e., how "thick" of a line to use when drawing itself). *Consider storing drawing coordinates that are relative to its parent GeneGroup.*

Exon class – An Exon object stores its start, length, and maybe width, similar to Gene.

Motif class – A Motif object stores its start, length, and maybe width, similar to Gene.

FastaHeader class – A FastaHeader object stores its start, text, and maybe font.

GeneGroup class – A GeneGroup object is composed of (keeps track of / stores references to) other objects, specifically: one Gene object, one Exon object, multiple Motif objects, and a FastaHeader object. Since there are multiple GeneGroup objects (i.e., one for each gene), it would be super helpful if each GeneGroup stored its "rank" (e.g., "Hey everyone, I'm the third gene!")

Behavior

Gene class – A Gene object should be given enough information that it can figure out how to draw itself. Genes should be black. Here's the one piece of example code I'll give you for this assignment, it's an example implementation of a Gene's draw **method** (not function). Note that variables in all UPPERCASE are **global constants** that should be defined in one central place near the top of your code. (You are free to use this code, or not, in your submission, your choice.)

```
def draw(self, context, gene_number):
    y = HEIGHT_GENE_GROUP * gene_number + Y_OFFSET_GENE
    context.set_line_width(self.width)
    context.move_to(LEFT_MARGIN, y)
    context.line_to(LEFT_MARGIN + self.length(), y)
    context.stroke()
```

Exon class – An Exon object should be given enough information that it can figure out how to draw itself. Exons should be black.

Motif class – A Motif object should be given enough information that it can figure out how to draw itself. Each distinct Motif should have a different color (e.g., all 'catag' Motifs should be one color, while all 'ygcy' motifs should be a different color). I may faint if you pick any garish or unsightly colors. (Just kidding, random colors are fine.)

FastaHeader – A FastaHeader object should be given enough information that it can figure out how to draw itself. FastaHeaders should be black.

GeneGroup class – A GeneGroup object is responsible for telling its children (e.g., Genes, Motifs) to go draw themselves. Hopefully its children can take on some responsibility for once and for example do the dishes and vacuum every now and again. Oops, shared my dirty laundry, apologies. Yes, the GeneGroup object should hand off some (most?) responsibility of how and where its "children" should be drawn.

Side note: Try to make the GeneGroup as dumb as possible when it comes to code that calculates where to draw Genes, Exons, Motifs, and FastaHeaders. Try to push that responsibility onto the Genes/Exons/Motifs/FastaHeaders as much as possible.

Overview and Thoughts

You may notice that the Gene, Exon, Motif classes are very very similar to one another. Gasp, you may even find yourself copying and pasting code around the place. What is a style(s) of object-oriented programming that might help reduce the amount of copy/pasted code (though potentially at the cost of simplicity)? (If you know the answer, please do *not* use that technique for your submitted code!)

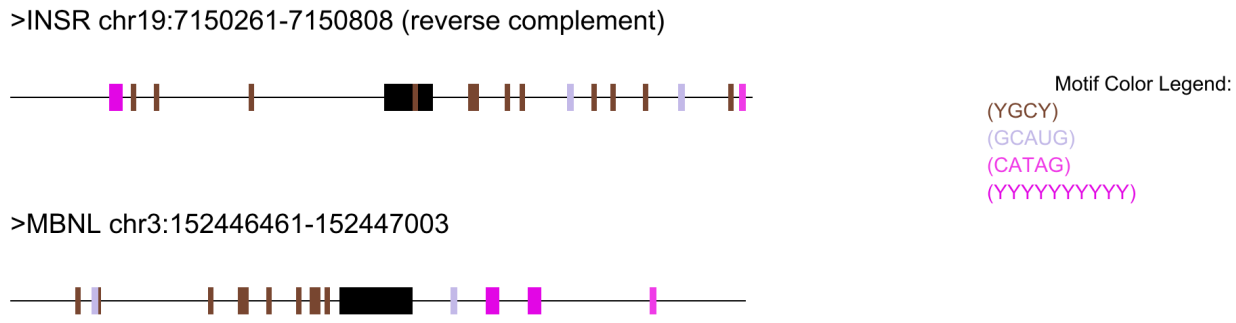
If you find that storing other data is useful, feel free to do so. For example, for debugging only, I found it useful to store the FASTA sequence of a given gene in each GeneGroup object.

Some of your code will live outside of your classes! For example, you'll want to instantiate a pycairo Context object in your "main", and then pass that single pycairo object around to your other objects, so that they can draw themselves onto that Context object.

```
surface = cairo.SVGSurface('plot.svg', …)
context = cairo.Context(surface)
```

Once you've conquered the object-oriented implementation and successful testing of the above requirements (and only then), apply your newfound knowledge and OOP perspective to implement the drawing of the figure legend. Depending on how everyone's feeling, we may turn this aspect into a stretch goal.

We're looking for something that looks (roughly) like the following. The color scheme for the Motifs does *not* matter.



Rereading this entire assignment would probably be very useful.

I encourage you to read this assignment, and then marinate on it for awhile, away from a keyboard.

This may seem like a big assignment, but if you follow the advice within, it's really just about rearranging / reorganizing your existing working non-OOP code so that it metamorphosizes into OOP code. FUN! And as always, PLEASE ASK QUESTIONS. I'm available.