**Project Resources**

* Metaprogramming – treating programs as data such that the program can read, generate, analyze or transform other programs, including itself
* <https://stackabuse.com/python-metaclasses-and-metaprogramming/>
  + Good with an example of multiplication
* <https://www.ibm.com/developerworks/library/ba-metaprogramming-python/index.html>
  + Not as good as above resource but helpful in describing how to create a metaclass

**Mock Schedule**

|  |  |
| --- | --- |
| Week (Meet) | Description of Activities |
| 1 (18.10) | Create a mock schedule (this), find resources on how to begin metaprogramming in Python, begin to research what needs to be done to create a successful program and plan for how long each step will take |
| 2 (25.10) | At very least, create a parser for the incoming text to separate it into pieces of a function, begin work on implementing basic mathematical functions with specified domains |
| 3 (1.11) | Continue working on implementing basic mathematical functions |
| 4 (8.11) | Finish implementing basic mathematical functions (+, -, \*, /) |
| 5 (15.11) | Begin work on creating lambdas for basic cases (fill this in more once have experience working with the metaclasses from weeks 1-4) |
| 6 (22.11) | Continue work on creating lambas for basic cases (fill this in more once have experience working with the metaclasses from weeks 1-4) |
| 7 (29.11) | Have created and finished all deliverables required for the course, finished up work to have a clean, working prototype |

**Early Steps**

* Create the parser – define a format such as:
  + satisfy(function\_name, a\_value, b\_value, equals, constraints = [])
    - function\_name has domain [+, -, \*, /, F]
    - a\_value, b\_value, and equals check for type on the input and then either use to create equals or find values which work
    - constraints are set to empty unless specified otherwise
    - Considering putting a\_value and b\_value into an array eventually so can have multiple inputs but starting with 12
* Define a specific format for output:
  + One idea is a dict for ex: **satisfy(+, A, B, 2, [A >= 0, B >= 0])**
    - [{ ‘A’: 0, ‘B’: 2}, {‘A’: 1, ‘B’: 1}, { ‘A’: 2, ‘B’: 0}]
* Slowly add in variables:
  1. Begin with addition where a\_value, b\_value and equals are specified and return the answer (should only be one)
  2. Don’t specify b\_value and get that to work (should only be one value possible)
  3. Add in checking for constraints before next step
  4. Have a\_value or b\_value with very small possible domains (to make sure infinite possibilities aren’t checked)
     + satisfy(+, A, B, 2, [A > 0, B > 0])
  5. Do same process with subtraction, multiplication, and division

Notes:

Streams (Java)

Using own parser in Python (determine undefined things)

Interface for data structure and type in something and tries to run it

**Questions**

* How (if at all possible) could this be applied to non-perfect statistics such as modelling sports statistics, stock market, etc.