Based on the code used during the 2nd class, let us implement a number of different components of the already working Genetic Algorithm and try to run them checking how they affect the efficiency and efficacy of the algorithm. Thus, modifying the code from the previous classes, let us:

Add a new problem, a very popular benchmark, Rastrigin function: http://www.geatbx.com/docu/fcnindex-01.html. This time it has a lot of local extrema - it is not a convex problem anymore.

Make the problems easily exchangeable by using Python reflection: getattr(sys.modules[name], "name-of-the-function")(parameters-of-the-function)

Our two functions have different constraints - modify the code so the hypercube encompassing our search space is defined in the main algorithm code, in an easily accessible place.

Check how the current parameters, legacy from the previous class work with a new problem. Try to modify the parameters of the search (mutation rate, mutation probability, number of generations, number of individuals...). Do check this for different number of dimensions (10, 50, 100).

In order to make the program better configurable - extract all its parameters (see 4) so they can be set in the main program loop. The parameters of the components (mutation, crossover etc) when needed should be passed as function arguments.

Seeking for better components of the algorithms implement new variation operators (implement those and see if introducing them helps you solving the Rastrigin problem) http://www.geatbx.com/docu/algindex-03.html:

- discrete crossover - a gene is taken randomly from first or second parent

- intermediate recombination- a gene is a mean value of both parents' genes

- introduce a gaussian mutation (instead of uniform one)

Finally, let us introduce the following selection methods:

- proportional (roulette) selection: The algorithm generates a random number between 0 and the sum of all fitnesses. Then, it walks through the array of frequencies, producing a running total. At some point the running total exceeds the generated threshold. The index at that point is the selection.

- modify the way of evaluation (rank is assigned - between 1 and NUMBER-Of\_INDIVIDUALS instead of fitness value) using any of the existing selection methods.

Draw overlay graphs (simply by adding the results one by one before calling plot.show()) showing different observations - to check how the efficacy and efficiency of the algorithms are affected (do not change everything at once - do it one by one):

- change the mutation component

- change the crossover component

- change the selection component.

Make sure all the components are easily callable in a dynamic way (using getattr) so the whole algorithm is easily configurable.

Check other problems and if there's time, do implement them:

- A very popular and difficult function for optimization (because of gradient pointing in wrong direction) - De Jong function 2 or Rosenbrock Valley: http://www.geatbx.com/docu/fcnindex-01.html

- A very popular and difficult function, Schwefel function:http://www.geatbx.com/docu/fcnindex-01.html "