1. (a) Lists are mutable (changeable). Tuples are immutable (non-changeable).
(b) The Pickle module provides data serialization and corresponding file I/O for Python. It is used to store the state of an object to a file and restore the state later. This is especially useful for large and complicated objects.
(c) One normally starts with analyzing where the performance bottlenecks are. This can be done in different ways, the fastest way might be to leave out certain parts of the program and see if the program responds faster. For quantitative results, you can use a timer in the code or a code profiler.
To deal with slow pieces of code, different measures can be taken:
- Implement using fast libraries if available (such as NumPy)
- Implement subparts in a lower level language such as C/C++
- Removing slow constructions, such as:
  - Code that requires creation of many new elements
  - Many nested for-loops

2. (a) The function takes a list and creates all permutations (variants) of it. It is a recursive function, meaning that it uses itself to solve the problem for a smaller subset of the input list. More specifically, the function calls itself with the input list except for the first element. You can then assume that the intermediate result will be all permutations of the subset. For each of these permutations, the first element is placed in every possible position, to create all new permutations that arise when adding that element to the set.
The yield commands act as return statements, but with the difference that the state of the function is saved before execution leaves the function. The next time the function is called (from the same outer scope), execution continues after that yield statement with local variables intact.
(b) The two problems are:
  i. Since the values in the input list are integers, division with a larger value will create a fraction below 1.0 which is rounded to 0. This will map all values except the old maximum to the new minimum, in this case 0.
  ii. The assignment in the for-loop changes the reference of ‘elem’ to a new object, leaving the list unchanged. It is not possible to change the actual value of an integer in Python since an integer is an immutable object.
What can be changed, is the object references that the list contains.
To make it work, the program can be modified like this:

```python
def remap(input_list, new_start, new_end):
    min_val = min(input_list)
    max_val = max(input_list)
    for i in range(len(input_list)):
```
input_list[i] = (input_list[i]-min_val)/(max_val-min_val)* \\
(new_end-new_start) + new_start

A = [0.0, 20.0, 50.0, 10.0]
remap(A, 0.0, 5.0)
print A

Notes:

- The changes to the list will persist in the outside scope where the function is called, since a list is a mutable type. There is no need for a return statement, although one often want to have one to increase readability and to leave the input parameters unchanged.
- To clearly see how the reference of a variable changes when performing an assignment, try this code out.

list = [1,2,3]
for elem in list:
    print "Id before: ", id(elem)
    elem = elem*2 + 1.
    print "Id after: ", id(elem)
print list
double atof(char * c) {
    double d = 0, mult = 1, dec=0;
    double div = 10;

    // make sure that we have a string
    assert(c != NULL && strlen(c) > 0);

    // check if there is a leading sign
    if (*c == '-' || *c == '+') {
        if (*c == '-') mult = -1; // remember the minus
        c++; // skip sign
    }
    assert(*c != '\0'); // must not end here
    while(isdigit(*c)) { // iterate as long as we have digits
        d = d*10 + (*c - '0'); // add them in, scale by 10 for each
        c++; // iteration, convert the char to a
    } // number

    // here we can have a decimal point or end of string
    if(*c == '\0') return d*mult; // finished, add in sign
    assert(*c == '.'); // must have a decimal point
    c++;
    while(isdigit(*c)) { // get the decimals
        dec = dec + (*c - '0')/div;
        c++;
        div *= 10;
    }
    assert(*c == '\0'); // we must be at the end now
    return mult*(d + dec); // add integral part and fraction
} // part, apply sign
#include <math.h>
#include <stdio.h>

// pointer to a square matrix and its size

int diagonal(double *m, int size) {
    int lines = size, kols = size; // use kols and lines for readability
    int result = 1; // my result
    int i, j;
    double comp, sum; // check
    for (i = 0; i < lines; i++) { // one line at a time
        sum = 0; // initial sum
        comp = fabs(*(m + i*kols + i)); // diagonal element
        for(j = 0; j < kols; j++) // sum up
            if (i != j) sum += fabs(*(m + i*kols + j));
        if (comp <= sum) { // if condition is not true
            result = 0; // we give up and return
            break; // false
        }
    }
    return result;
}

// some testing

int main () {
    double a[3][3] = {{ 5, 1, 1}, { 1,7,1}, { 2, 6, 10}};
    double b[2][2] = {{ 5, 1}, { 7,1}};

    int m1, m2;

    m1 = diagonal(&a[0][0], 3);
    m2 = diagonal(&b[0][0], 2);
    printf("%d %d\n", m1, m2);
    return 0;
}
5.

// a coordinate in (x,y)-space

#include <cmath>
#include <iostream>
using namespace std;

class Point {
private:
    double x,y;
public:
    Point():x(0),y(0) {}
    Point(double xp, double yp):x(xp),y(yp) {}

double getX() {return x;}
void setX(double xp) {x = xp;}
double getY() {return y;}
void setY(double yp) {y = yp;}

    // distance from me to the point p

double distance(const Point& p) {
    return sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
}
};

class Triangle {
private:
    Point p1, p2, p3;
public:
    Triangle(Point a, Point b, Point c):p1(a), p2(b), p3(c) {}

double area() {
    double a,b,c,s,area;
    a = p1.distance(p2);
    b = p2.distance(p3);
    c = p3.distance(p1);
    s = (a + b + c) / 2;
    area = sqrt(s*(s-a)*(s-b)*(s-c));
    return area;
}
};

int main() {
    Triangle t(Point(1,0), Point(0,1), Point(0,0));
    cout << "The area is " << t.area() << endl;
6. One possible solution is:

```cpp
template <class T>
class Stack
{
    int currentSize;
    int maxSize;
    T *data;

public:
    Stack(int m) : currentSize(0), maxSize(m), data(new T[m]) {} ~Stack() {delete [] data;}

    T pop();
    void push(T x);
};

template <class T>
T Stack<T>::pop()
{
    if(currentSize == 0)
        throw "No elements to pop!";
    else
        return data[--currentSize];
}

template <class T>
void Stack<T>::push(T x)
{
    if(currentSize == maxSize)
        throw "Max size already reached!";
    else
        data[currentSize++] = x;
}
```