1. (a) `yield` is a keyword used as `return`, but returns a generator which holds the state of the function, as well as the returned value. Example:

```python
def number_generator():
    for i in range(1,11):
        yield i

for i in number_generator():
    print(i),
```

(b) For example, any of these two is a valid answer:

- Simpleness
- Platform independence
- Encourage a clean programming style
- Code should be easy to read
- Minimize redundancy

(c) Identity, type and value for Python objects:

- Identity is a unique identity number for each object, associated to where it is stored in memory.
- The type tells what kind of the data the object contains, and can be for example list, int, function or generator.
- The value is the actual data a variable contains.

2.

```python
# Create a loop over 1-100:
# (range(100) would give 0-99)
for i in range(1,101):
    # Test if it's a third or fifth number.
    # The modulo operator '%' gives the remainder when
    # dividing with a certain number. If the result is
    # zero, then the number is a multiple of that number.
    third = i%3 == 0
    fifth = i%5 == 0

    # Do printing. Can't use elif (with the approach below)
    # since both Fizz and Buzz should be printed sometimes
    # (then you need to create a special test for the
    # FizzBuzz-case and place it first).
    if third:
        print "Fizz",
```
if fifth:
    print "Buzz",
if not (third or fifth):
    print i,
print ", ",

3.
#include <stdio.h>

/*
 function that tests if we have a golden ratio
 parameters:
    total: double, the total length of the segment
    lng: double, the length of the longest sub segment
 result:
    true (1) or false (0) depending of the outcome of the test
*/

int golden(double total, double lng) {
    double a = lng, b = total - lng;
    if (a < b) return 0; // just to be safe

    // now its meaningless to test for equality
    // here because that requires that
    // ALL decimals are identical and
    // we will never achieve that. Therefore
    // we test for a very small difference instead

    // do the proper test

    return fabs(total/a - a/b) < 1.0e-10;
}

int main() {
    double t,a;

    // test three times

    int i = 1;
    for (; i <= 3; i++) {
        printf("Give total length, and the longest segment: ");
        scanf("%lf",&t);
        scanf("%lf",&a);
int res = golden(t,a);
if(res) printf("Golden ratio %5.2f, %5.2f\n", a, t-a);
else printf("Not a golden ratio\n");
}
}

/* the quotient a/b should be (1 + sqrt(5))/2 to have a
golden ratio. So if b is 1, a should be
approx 1.6180339887

tragula.it.uu.se> gcc golden.c
tragula.it.uu.se> a.out
Give total length, and the longest segment:
2.6180339887 1.6180339887
Golden ratio 1.62, 1.00
Give total length, and the longest segment:
2 1
Not a golden ratio
Give total length, and the longest segment:
10 5
Not a golden ratio

*/

4.
#include <stdio.h>
#include <math.h>

// faculty function, this gives us problem for
// non small n's because it grows so fast.

double fac(int n) {
    double p = 1;
    int i;
    for (i = 1; i <= n; i++) {
        p = p*i;
    }
    return p;
}

double sinx(double x) {

double epsilon = 1.0e-10; // limit
double term = x; // our first term
double sum = term; // our sum

// now count, remember that we
// already have the first term

int n = 2;

// sum until the terms are small enough

while (fabs(term) > epsilon) {
    // compute the next term and add it in
    term = pow(-1,n-1) * pow(x, 2*n-1)/fac(2*n-1);
    sum += term;
    n++; // increment n
}
return sum; // return sum

// this is a much more efficient variant,
// it also avoids the fact that
// the faculty grow so fast. It uses the fact
// that if we have one arbitrary
// term in the series we can get the next
// term by changing the sign
// multiplying it by x twice and dividing
// it by (2*n)* (2*n+1) if n is the
// number of the current term
// thus if we have the first term, ie x,
// with number one, we can get the
// second term as -x *x*x/2*3, ie -x^3/6.
// having that we can get the third term
// as x^3/6 *x*x/4/5

// we don’t start from scratch each time

// this gives us

double sinx2(double x) {
    double epsilon = 1.0e-10;
    double term = x;
double sum = term;
int n = 1;
while (fabs(term) > epsilon) {
    term = -term*x*x/(2*n)/(2*n+1);
    sum += term;
    n++;
}
return sum;
}

// test this
int main() {
    double x;
    int i;
    for (i = 1; i <= 3; i++) {
        printf("Give x: ");
        scanf("%lf", &x);
        printf("approx sine is %10.5f and %10.5f, sin is %10.5f\n",
               sinx(x), sinx2(x), sin(x));
    }
    return 0;
}

/* will give the following output
   you see that our approximation is bad for big x’s

vranx.it.uu.se> gcc sinx.c -lm
vranx.it.uu.se> a.out
Give x: 0.4
approx sine is 0.38942 and 0.38942, sin is 0.38942
Give x: 6.7
approx sine is 0.40485 and 0.40485, sin is 0.40485
Give x: 45.5
approx sine is 615.08104 and -101.43656, sin is 0.99859
*/
private:
    int * buffer; // data storage
    int size; // size of the buffer
    int number; // actual usage

public:
    Container(); // constructor
    Container(int size); // constructor
    ~Container(); // destructor
    void store(int data); // store data in the first free place
    int find(int index); // return the data in position 'index'
};

// A constructor MUST set up all attributes,
// use 10 as default size

Container::Container():size(10),number(0) {
    buffer = new int[10];
}

Container::Container(int size):size(size),number(0){
    buffer = new int[size];
}

Container::~Container() {
    delete [] buffer;
}

// store one data element in the next free position
// We use number to find this place

void Container::store(int data) {

    // check if there are any space left
    if(number >= size) {

        // no space, make some new
        int* temp = new int[size * 2]; // new space
        int i;

        // copy data from old space to new space
        for (i = 0; i < number; i++) temp[i] = buffer[i];
// remove the old space and refer the new instead

delete [] buffer;
buffer = temp;

// update size
size = 2*size;

}

// add the data

buffer[number++] = data;
}

int Container::find(int index) {
    if(index < 0 || index >= number) {
        cout << "find::invalid index " << index << endl;
        return -1;
    }
    else
        return buffer[index];
}

int main() {
    Container c(20);       // space for 20 number;
    int i;
    for (i = 1; i <= 30; i++)    // add 30 number
        c.store(i*i);
    for (i=0; i<=30; i++)     // get 31 numbers
        printf("Number at %d is %d\n", i, c.find(i));
    return 1;
}

/* output will be

vranx.it.uu.se> g++ Container.cc
vranx.it.uu.se> a.out
Number at 0 is 1
Number at 1 is 4
Number at 2 is 9
Number at 3 is 16
Number at 4 is 25
 */
Number at 5 is 36
Number at 6 is 49
Number at 7 is 64
Number at 8 is 81
Number at 9 is 100
Number at 10 is 121
Number at 11 is 144
Number at 12 is 169
Number at 13 is 196
Number at 14 is 225
Number at 15 is 256
Number at 16 is 289
Number at 17 is 324
Number at 18 is 361
Number at 19 is 400
Number at 20 is 441
Number at 21 is 484
Number at 22 is 529
Number at 23 is 576
Number at 24 is 625
Number at 25 is 676
Number at 26 is 729
Number at 27 is 784
Number at 28 is 841
Number at 29 is 900
find::invalid index 30
Number at 30 is -1

*/

6.

template <class T, int N>
T Vec<T,N>::p_norm(int p)
{
    T sum = 0;
    for (int i=0; i<N; i++)
    {
        sum += pow(data[i],p);
    }

    T result = pow(sum,1.0/(T)p);

    return result;
}
template<class T, int N>
Matrix<T> VectorField<T,N>::p_norm(int p)
{
    Matrix<T> out(mnWidth, mnHeight);
    for (int i=0; i<mnWidth; i++)
    {
        for (int j=0; j<mnHeight; j++)
        {
            out(i,j) = mData[i][j].p_norm(p);
        }
    }
    return out;
}

7.
public class Container {
    private int[] buffer; // data storage
    private int size;     // size of the buffer
    private int number;   // actual usage

    public Container()  // constructor
    {
        buffer = new int[10];
        size = 10;
        number = 0;
    }

    public Container(int size) // constructor
    {
        buffer = new int[size];
        this.size = size;
        number = 0;
    }

    // store data in the first free place
    public void store(int data)
    {
        // check if there are any space left
        if(number >= size) {

    }
// no space, make some new
int[] temp = new int[size * 2];  // new space
int i;

// copy data from old space to new space
for (i = 0; i < number; i++) temp[i] = buffer[i];

// remove the old space and refer the new instead
buffer = temp;

// update size
size = 2 * size;
}

// add the data
buffer[number++] = data;
}

// return the data in position 'index'
int find(int index)
{
    if(index < 0 || index >= number) {
        System.out.println("find::invalid index " + index);
        return -1;
    }
    else
        return buffer[index];
}

public static void main(String [] arg) {

    // space for 20 numbers;
    Container c = new Container(20);
    int i;
    for (i = 1; i <= 30; i++)  // add 30 number
        c.store(i*i);
for (i=0; i<=30; i++) // get 31 numbers
    System.out.println("Number at " + i + 
    " is " + c.find(i));
}

/* output will be

vranx.it.uu.se> g++ Container.cc
vranx.it.uu.se> a.out
Number at 0 is 1
Number at 1 is 4
Number at 2 is 9
Number at 3 is 16
Number at 4 is 25
Number at 5 is 36
Number at 6 is 49
Number at 7 is 64
Number at 8 is 81
Number at 9 is 100
Number at 10 is 121
Number at 11 is 144
Number at 12 is 169
Number at 13 is 196
Number at 14 is 225
Number at 15 is 256
Number at 16 is 289
Number at 17 is 324
Number at 18 is 361
Number at 19 is 400
Number at 20 is 441
Number at 21 is 484
Number at 22 is 529
Number at 23 is 576
Number at 24 is 625
Number at 25 is 676
Number at 26 is 729
Number at 27 is 784
Number at 28 is 841
Number at 29 is 900
find::invalid index 30
Number at 30 is -1
*/
public class Container {

    public class InvalidIndexException
        extends RuntimeException {
        public InvalidIndexException(String s) {
            super(s);
        }
    }

    private int[] buffer;  // data storage
    private int size;  // size of the buffer
    private int number;  // actual usage

    public Container() // constructor
    {
        buffer = new int[10];
        size = 10;
        number = 0;
    }

    public Container(int size) // constructor
    {
        buffer = new int[size];
        this.size = size;
        number = 0;
    }

    // store data in the first free place
    public void store(int data)
    {
        // check if there are any space left
        if(number >= size) {
            // no space, make some new
            int[] temp = new int[size * 2];  // new space
            int i;

            // copy data from old space to new space
        }
    }
for (i = 0; i < number; i++) temp[i] = buffer[i];

// remove the old space and refer the new instead

buffer = temp;
}

// add the data

buffer[number++] = data;
}

// return the data in position 'index'

int find(int index) {
    if(index < 0 || index >= number) {
        System.out.println("find::invalid index "+ index);
        return -1;
    }
    else
        return buffer[index];
}

// remove one item

public int remove(int index) {
    if (index < 0 || index >= number) 
        throw new InvalidIndexException(
            "Remove:: Invalid index "+ index);
    else {
        int data = buffer[index];

        // fill the hole by moving data one
        // step to left from
        // from here to the end

        for (int i = index; i < number-1; i++)
            buffer[i] = buffer[i + 1];
        number--;
        return data;
    }
}
// sort this

public void sort() {
    int n = number; // number of values
    while (n > 1) {
        int pos = 0; // assume this is the
        // position for the maximum
        int max = buffer[0]; // and the value
        for(int i = 1; i < n; i++)
            // find the max value of
            // the numbers in pos 1..n
            if (buffer[i] > max) {
                max = buffer[i];
                pos = i;
            }
        int temp = buffer[n-1];
        buffer[n-1] = buffer[pos];
        buffer[pos] = temp;
        n--;
    }
}

public static void main(String [] arg) {
    // space for 20 number
    Container c = new Container(20);
    int i;
    for (i = 1; i <= 30; i++) // add 30 number
        c.store((int)(Math.random()*50));
    for (i=0; i<30; i++) // get 30 numbers
        System.out.println("Number at " + i + " is " + c.find(i));
    System.out.println("Sorted:");
    c.sort();
    for (i=0; i<30; i++) // get 30 numbers
        System.out.println("Number at " + i + " is " + c.find(i));
}
/* output will be
vranx.it.uu.se> javac Container.java
vranx.it.uu.se> java Container
Number at 0 is 17
Number at 1 is 48
Number at 2 is 10
Number at 3 is 8
Number at 4 is 46
Number at 5 is 25
Number at 6 is 17
Number at 7 is 2
Number at 8 is 38
Number at 9 is 24
Number at 10 is 17
Number at 11 is 10
Number at 12 is 17
Number at 13 is 28
Number at 14 is 25
Number at 15 is 44
Number at 16 is 14
Number at 17 is 37
Number at 18 is 40
Number at 19 is 0
Number at 20 is 1
Number at 21 is 5
Number at 22 is 12
Number at 23 is 40
Number at 24 is 43
Number at 25 is 18
Number at 26 is 14
Number at 27 is 7
Number at 28 is 40
Number at 29 is 38
Sorted:
Number at 0 is 0
Number at 1 is 1
Number at 2 is 2
Number at 3 is 5
Number at 4 is 7
Number at 5 is 8
Number at 6 is 10
Number at 7 is 10
Number at 8 is 12
Number at 9 is 14
Number at 10 is 14
Number at 11 is 17
Number at 12 is 17
Number at 13 is 17
Number at 14 is 17
Number at 15 is 18
Number at 16 is 24
Number at 17 is 25
Number at 18 is 25
Number at 19 is 28
Number at 20 is 37
Number at 21 is 38
Number at 22 is 38
Number at 23 is 40
Number at 24 is 40
Number at 25 is 40
Number at 26 is 43
Number at 27 is 44
Number at 28 is 46
Number at 29 is 48

*/

9.

10.