Following are some programming exercies. They aren't in any difficulty order. Some of them are easy enough to be asked in the exam.

P1

Read a positive integer number from the user and then display the numbers that divide this number without any remainder. Additionally, report how many such numbers are there and whether the input number is a prime number or not. Sample runs:

```
Enter a positive integer : 0
The number should be positive.

Enter a positive integer : 24
Divisors of 24 are : 1, 2, 3, 4, 6, 8, 12, 24
There are 8 divisors so 24 is NOT a prime number.

Enter a positive integer : 7
Divisors of 7 are : 1, 7
There are only 2 divisors so 7 IS a prime number.
```

P2

Write a program that will determine how an ATM will give cash back to a user. Let's assume that an ATM holds banknotes of type 100TL, 50TL, 10 TL, 5TL and 1TL. When the user enters the amount she wants to withdraw, the ATM tries to match this amount with the largest banknotes first. When the amount can't be matched with just this type of banknote, it uses the next largest banknote and this continues until the whole amount is matched. Print how many of each banknote will be given back if the ATM utilizes this algorithm. See examples for more clarification:

```
Enter withdrawal amount : 125
125TL = 1x100TL + 2x10TL + 1x5TL

Enter withdrawal amount : 873
882TL = 8x100TL + 1x50TL + 3x10TL + 2x1TL

Enter withdrawal amount : 399
399TL = 3x100TL + 1x50TL + 4x10TL + 1x5TL + 4x1TL
```

P3

Write a program that will read numbers from the user until the user enters a negative number. Then display the total and average of the numbers entered. Sample run:

```
Number please: 12
Number please: 6
Number please: 15
Number please: -2
Total : 33
Average : 11.0
```

P3

Here is a function that will find the maximum value in a numbers list:

```
def findMax(nums):
  result = nums[0]  # assume first number is the maximum one
  for num in nums[1:]:    # check the rest of the numbers
    if num > result: result = num  # if you find a larger num than the
  return result                    # current one found, modify your result
```

Write a similar function called findMax2 that will return the second largest value inside its nums parameter. This is similar to finding the largest value but this time you have to keep track of the largest (i.e max1) and the second largest value (i.e max2) while you are iterating over the values in the array. Whenever you find a new array value that is larger then one of them you have to update these variables accordingly (i.e. if new value is larger than max1 then max2 becomes max1 and max1 becomes this new value, if it is only larger than max2 then only update max2 as this new value).

P4

The following algorithm can be used to compute the greatest common divisor (GCD) of two numbers:

1. Read numbers A and B whose GCD will be computed
2. If A equals B then GCD is A (or B)
3. If A>B then make A equal to A-B, otherwise (if A < B) make B equal to B-A and then go back to step 2

For instance, for values A=60 and B=35 the algorithm will work like this:

```
 A   B
--- ---
 60  35   (initial values)
--------> (A>B so A becomes 60-35)
 25  35
--------> (B>A so B becomes 35-25)
 25  10
--------> (A>B so A becomes 25-10)
 15  10
--------> (A>B so A becomes 15-10)
  5  10
--------> (B>A so B becomes 10-5)
  5   5
-------->  (A=B so GCD is found and it is 5)
```

You can use a `while` loop containing some `if-else` statements to implement this algorithm. If GCD of two numbers are known, least common multiple (LCM) of these numbers can be computed by using the following equation: `A*B = GCD(A,B)*LCM(A,B)`. Read two numbers from the user and then display their GCD and LCD by using the information above.

P5

Read a number from the user that is between 5 and 9 and then produce the following patterns. Hint: Use nested loops or duplicating a string by * operator: “ab” * 5 -> “abababababab”

```
Enter a number: 7

Pattern 1:

1
22
333
4444
55555
666666
7777777

Pattern 2:

......1
.....22
....333
...4444
..55555
.666666
7777777

Pattern 3:

1
22
321
4444
54321
666666
7654321

Pattern 4:

*
**
*-*
*--*
*---*
*----*
*-----*
*------*
*-------*
**********

Pattern 5:

1000001
0200020
0030300
0004000
0050500
0600060
7000007
```

Pattern 6:

```
7-7-7-7-7-7-7
 6-6-6-6-6-6
  5-5-5-5-5
   4-4-4-4
    3-3-3
     2-2
      1
```

Pattern 7:

```
+++++++
+*****+
+*****+
+*****+
+*****+
+*****+
+++++++
```

Pattern 8 (you may assume input number is odd):

```
+-+-+-+
-*****+
+*****-
-*****+
+*****-
-*****+
+-+-+-+
```