CS/CSE-A1111 Basic Course in Programming Y1 Exam 24.10.2016

Write the following information clearly on top of each paper you submit: name of the course, date of the exam, your full name, student ID, the total number of papers you submit, and your signature.

Important instructions: Use indentations of the length of two squares in your code. If your indentations are not clear enough, you lose points. You do not have to write any comments in your code. You do not have to format output in any problem. You can assume that the input given by the user is correct, if it is not told in the problem that you should handle the incorrect input. **Calculators and any extra material are not allowed. However, students whose mother tongue is not Finnish may use a dictionary, if it does not contain any extra markings. Those students may also obtain both Finnish and English exam sheet, if they want.**

1. Parts a, b, c, and d: What is printed when the given Python program is executed? It is enough to give the correct output without any explanations. Parts e, f, and g: Explain, in plain English using a couple of sentences, the *purpose* of the given Python function. (Do not say *how* the code works. Instead say what the function would be used for, for example "the function calculates and returns the sum of the integers in the list given as a parameter".) Notice that some of the programs or functions may contain errors. In that case, explain what the incorrect program prints or how the incorrect function works.

   a) (2 p)
```
def main():
    bacteria = 200
    if bacteria > 10:
        print("Water quality is OK.")
    elif bacteria > 150:
        print("Water quality is bad.")
    else:
        print("Water quality is good.")

main()
```

   b) (2 p)
```
def double1(number):
    print(2 * number)

def main():
    value = 15
    number = 4
    double1(value)

main()
```

   c) (3 p)
```
def main():
    result = 100
    list1 = [35, 10, 50, 10, 5]
    for number in list1:
        if result > 30:
            result = result - number
    print(result)

main()
```

d) (5 p)

```
def change(number, numbers):
    numbers[0] = 2 * numbers[1]
    number = 5 * number
    return number

def main():
    list1 = [2, 12, 15]
    result = 10
    change(result, list1)
    print(result)
    for element in list1:
        print(element)

main()
```

10 2 12 15

50 24 12 15

e) You may assume that the parameters of the function are two equally-long lists containing integers. (4 p)

```
def mystery1(list1, list2):
    i = 0
    result = 0
    while i < len(list1):
        if list1[i] != list2[i]:
            result += list1[i] + list2[i]
        i += 1
    return result
```

0 1 2 len=3

i=0 1 2 3

0 1 4

f) You may assume that the parameter of the function is a string, which contains at least one character. (4 p)

```
def mystery2(str1):
    i = len(str1) - 1
    str2 = ""
    while i >= 0:
        str2 += str1[i]
        i -= 1
    return str2
```

hello i=4

g) You may assume that the parameter of the function is a list containing integers. The length of the list is at least 2. (5 p)

10 12 15 i=1 len=3

```
def mystery3(list1):
    i = 1
    while i < len(list1):
        if list1[i] > list1[i - 1]:
            return True
        else:
            return False
        i = i + 1
```

2. a) A shop gives a 5 % discount on the purchases whose value is at least 100 euros, if the customer has a bonus card. If the customer has not the bonus card or the value of the purchase is under 100 euros, the customer pays the normal price. Write a Python program which asks for the normal price and whether the customer has the bonus card. The program must output the sum which the customer has to pay. You may decide the exact format of the input (for example, how the user tells whether the customer has the bonus card) and output yourself. (10 p.)

   b) The monthly salaries of all employees of a firm has been stored in the list such that each element (a decimal number) of the list is the salary of one employee in euros. Write a Python function `how_many_greater(salaries, gap)`. The first parameter of the function is the list described above and the second parameter is a decimal number. The function must return the number of the employees whose salary exceeds the average salary of all employees at least by the sum given as the second parameter. For example, if the average of the salaries in the list is 2500 and the second parameter is 300, the function must return the number of the salaries which are at least 2800 euros. (20 p)

3. A shop has stored information about the purchases of its customers in a text file as follows. Each line containts information about one purchase of one customer: the date, the customer ID and the value of the purchase. The different fields are separated by the semicolon. Three lines of the file could look like this:

```
14.1.2015;A11223;55.30
14.1.2015;B7755;63.75
15.1.2015;A11223;75.00
18.1.2015;A11223;25.80
```

   Write a Python program which reads the file and calculates and outputs the number of purchases with the value of over 50 euros made by a certain customer specified by the user. The program asks the user to input the file name and the ID of the customer whose purchases are calculated. For example, if the user gives the customer ID A11223, the program should output that the customer has 2 purchases with the value of over 50 euros in the case above. If the file does not contain the ID of the customer, the program should output that the customer has 0 purchases with the value of over 50 euros.

   The program must be able to handle the following errors
   - The file does not exist or it is not possible to read the file because of some other reason.
   - In some line, the value of the purchase is not given as a decimal number.

   In these cases, the program must output the type of the error occured and stop. The program does not have to continue reading after the defective line. You may also assume that each line of the file consists of three fields separated by a semicolon. Your program does not have to handle cases where the file contains empty lines or lines consisting of only one field, for example. (20 p)

4. A firm offers a net account to help individual people sell and buy items using Internet. A person creates a net account and deposits money into it. When he/she wants to buy something, he/she tells his/her net account to make a provision (reservation) for the sum which is the same as the price of the item he/she wants to buy. The firm tells the seller that the provision has been made. Then the seller sends the item to the buyer. When the buyer receives the item, he/she tells to the firm to withdraw the sum corresponding to the provision from the account and pay it to the seller. Thus, the buyer does not have to actually pay the item before he/she has received it, but the seller does not have to send the item without making sure that he/she will receive the payment. The buyer may have several unfinished trades simultaneously. Thus, the account may have several provisions at the same time.

CONTINUES

Write a Python class `Netaccount` to describe one net account. A `Netaccount` object should have the following data attributes (fields):

- `__owner` name of the owner of the account (the buyer)
- `__balance` amount of money saved currently in this net account (contains provisions)
- `__provisions` sum of the valid provisions in this net account (to make this problem easier, we do not specify various provisions made, but just store the sum of the valid provisions).

The class should have the following methods (unless otherwise told, the method does not have to return anything):

- `__init__(self, name)` creates a new `Netaccount` object. The name of the owner is given as a parameter. The balance of the new account is 0.0 and there are no valid provisions.
- `get_owner(self)` returns the name of the owner of the account.
- `deposit(self, amount)` deposits the amount given as a parameter to the accont (i.e. increases the balance), if the sum is positive. If the sum is negative or zero, the method does nothing.
- `make_provision(self, amount)` makes a provision for the amount given as a parameter. This is possible only if the total sum of valid provisions is less than or equal to the balance after this action. The amount given as a parameter also has to be positive. If it is possible to make the provision, the method returns the value `True`. If it is not possible, the method returns the value `False`.
- `withdraw(self, amount)` withdraws an amount given as a parameter thus that it can be paid to a seller, if the amount is positive and less than or equal to the sum of the valid provisions. This means that both balance and the sum of valid provisions are decreased by the given amount, if the withdrawal is possible. (Paying the amount to the seller is not performed in this method or class, but it is performed elsewhere.) If the withdrawal is possible, the method returns `True`. If it is not possible, the method does not change any values of the data attributes and returns `False`. (To simplify the problem, it is not checked that there is an earlier provision of exactly the same amount. It is enough that the amount to be withdrawn is not greater than the sum of the valid provisions. Other actions outside this class ensure that no seller is paid more than he/she should. You are not to asked to write that check.)
- `cancel_provision(self, amount)` cancels an earlier provision of the amount given as a parameter. This is possible only if the amount is positive and less than or equal to the sum of the valid provisions. Cancelling means that the sum of the valid provisions is decreased by the amount. If cancelling is possible, the methtod returns `True`. If it is not possible, the method does not change any values of the data attributes and returns `False`. This method is meant for situations where the trade is cancelled, for example.
- `__str__(self)` returns a string which contains the name of the owner of the account, the balance and the sum of the valid provisions.

In addition, write a main program which creates two `Netaccount` objects and after that performs depositons into them. Then the main program has to make a provision to the account created first and output whether it succeeded. Next, the program should withdraw money from the first account and output whether it succeeded. Finally, the program should output information about both accounts: the name of the owner, the balance and the sum of the valid provisions. Your main program does not have to ask any input from the user. You may decide the names and the amounts needed yourself. (25 p)