Introduction to Data Science & Programming

BSINDSP1KU

By Mie Jonasson

miejo@itu.dk

Question 1:

Consider the following two lists of prices and stock of different fruits from an inventory.

```
stock_list = [
  ("banana", 65),
  ("apple", 5),
  ("orange", 325),
  ..., # the list continues here
  ("pear", 15)]

prices = [
  ("banana", 4),
  ("apple", 2),
  ("orange", 1.5),
  ..., # the list continues here
  ("pear", 3)]
```

(a) (30 points) Suppose these list data structures each contain the same n words. Say we need to change both the price and the stock for one of the fruits. This fruit is identified by the string, say, `fruit = "grape"` - we do not know its position in the list. In the worst case scenario, explain how long it would take, in relation to n, to perform this operation with this data structure? What about the best case scenario?

Because the data is saved in two lists, iteration through them for finding a specific item will take a time that is dependent on n. This means that the worst-case scenario would be if the item we are looking for is the last item in the list – then we will have iterated through every single item in the list before finding what we are looking for. I.e. worst case scenario is looking through and comparing *n* items.

(b) (30 points) Which Python data structure would be most efficient out of the data structures you learned to carry out the same task? Explain, i.e. give the best and worst case amount of time in relation to n.

The most efficient data structure out of the ones we've learned would be a dictionary. With a dictionary we could use the names of the fruits as keys (because each fruit should only appear once) and save the stock and price as a value. We might even concatenate the data into one dictionary, were the value be a list of 2 numbers, i.e.

>>>fruit_dict["banana"]

[65, 4]

Question 2:

Consider the block of Python code below.

```
i = 1
j = 500
while i>=1:
  a =  " "*j + "*"*i + " "*j
  print(a)
  i = i+2
  j = j-1
  if i>1001:
    break
```

(a) (5 points) How many times does the above `while` loop execute exactly?

The while loop will iterate 501 times, which on 501st iteration after printing for i=1001 and j=0, it will redefine i = i+2, and then the if-statement will be true (because i now holds the value 1003), thus arriving at 'break'.

(b) (15 points) What does the code print out? In particular, what exactly is printed in the last line?

The code prints out a pyramid of asterisks with the first line printing 500 spaces followed by an asterisk and then followed by another 500 spaces. Zooming in on the top middle of the pyramid would reveal:

```
  *
 ***
*****
```

And the last line of code would contain no spaces, but only 1001 asterisks.

(c) (10 points) Write an implementation for the same functionality by replacing the `while` loop with a `for` loop.

```
js = list(range(501))
js.reverse()
for i,j in zip(range(1,1002,2),js):
    print(" "*j + "*"*i + " "*j)
```

Question 3:

Consider the block of Python code below.

```
def some_operation(s, i=0, j=0):
  if i == len(s):
    return j == 0
  if j < 0:
    return False
  if s[i] == "(":
    return  some_operation(s, i + 1, j + 1)
  elif s[i] == ")":
    return  some_operation(s, i + 1, j - 1)

some_operation(s)
```

Let the object `s` that is passed to the function be a string.

(a) (20 points) What is the secret operation that the above function performs with the input?

It analyzes a string that only contains '(' and ')' since there are no conditional statements to catch any other characters in any iteration, these would return the default None. Here are the cases it would return something:

Empty String – i.e. ""	True
String "()" or "(())" and so on	True
String "(()(()()))" or other 'valid' parentheses	True
All other string with a comb. Of "(" & ")"	False

(b) (20 points) Demonstrate that you are correct by showing what happens to two short but informative strings.

Example 1:

```
def some_operation(s, i=0, j=0):  for s= ")"
  if i == len(s): i= 0 and len(s) = 1, so this is false
    return j == 0
  if j < 0:  j=0, so this is false
    return False
  if s[i] == "(": s[0]=")", so this is false
    return  some_operation(s, i + 1, j + 1)
  elif s[i] == ")":  This is true! thus we run some_operation(")",1,-1)
    return  some_operation(s, i + 1, j - 1)

some_operation(s)
```

```
def some_operation(s, i=0, j=0):  for s=")", i=1, j=-1
  if i == len(s):  i=1 and len(s)=1, so this is true!
    return j == 0  j=-1, thus this will return False
  if j < 0:
    return False
  if s[i] == "(":
    return  some_operation(s, i + 1, j + 1)
  elif s[i] == ")":
    return  some_operation(s, i + 1, j - 1)

some_operation(s)
```

From this first we also see, that even if the string was longer (no matter what would be in the rest of the string) it would be caught in the second step 'if j < 0' and therefore return False. I.e. any string starting with ')' would return false.

Example 2:

```
def some_operation(s, i=0, j=0): for s="()"
  if i == len(s): i=0 and len(s)=2, false
    return j == 0
  if j < 0:  j=0, false
    return False
  if s[i] == "(":  s[0]="(", TRUE!
    return  some_operation(s, i + 1, j + 1) some_operation("()" , 1 , 1)
  elif s[i] == ")":
    return  some_operation(s, i + 1, j - 1)

some_operation(s)
```

```
def some_operation(s, i=0, j=0): for s="()", i=1, j=1
  if i == len(s): i=1 and len(s)=2, false
    return j == 0
  if j < 0: j=1, false
    return False
  if s[i] == "(":   s[1]=")", false
    return  some_operation(s, i + 1, j + 1)
  elif s[i] == ")": s[1]=")", TRUE!
    return  some_operation(s, i + 1, j - 1)some_operation("()" , 2 , 0)

some_operation(s)
```

```
def some_operation(s, i=0, j=0): for s="()", i=2, j=0
  if i == len(s): i=2 and len(s)=2, TRUE!
    return j == 0  j=0, thus this will return True
  if j < 0:
    return False
  if s[i] == "(":
    return  some_operation(s, i + 1, j + 1)
  elif s[i] == ")":
    return  some_operation(s, i + 1, j - 1)

some_operation(s)
```

From this we see that i is used to iterate through each element of the string, and j will increase by 1 for each beginning parentheses and decrease by one for each ending parentheses, thus only ending up at 0 if there are an equal amount of beginning and starting parentheses in the right order (i.e. "())(" would return False by the second conditional statement at i=3, since j=-1).

Question 4:

Prove the following by induction. Explicitly state the base case and the induction step.

Prove for all $n \geq 4$, that:

$$2n < n!$$

1) Base Case

Base case is n=4, so we just paste this into the inequality:

$$2n < n!$$

$$2 * 4 < 4!$$

$$8 < 24$$

This is true, thus the base case is proven

2) Assumption for k

We assume that the inequality is true for a random k, greater than or equal to 4:

$$2 * k < k!$$

3) Proof of k+1 if k (Inductive step)

We write up the equation for k+1:

$$2 * (k + 1) < (k + 1)!$$

Since 2k < k! we can write up this inequality multiplied by (k+1) on both sides:

$$2k * (k + 1) < k!\,(k + 1) = (k + 1)!$$

This thereby implies:

$$2k * (k + 1) < (k + 1)!$$

$$k\big(2 * (k + 1)\big) < (k + 1)!$$

From this we can deem it obvious that the inequality holds since:

$k\big(2 * (k + 1)\big) < (k + 1)!$ This holds true from assumption

$2 * (k + 1) < k\big(2 * (k + 1)\big)$ Since k is a positive number above 4

$2 * (k + 1) < k\big(2 * (k + 1)\big) < (k + 1)!$ We concatenate the equations

$2 * (k + 1) < (k + 1)!$ And remove the middle part since there are only '<'

And thus we have proved the inequality to hold true for all n≥4, since proving k+1 by k being true creates a ripple effect for all n≥4

Question 5:

Consider the following data set with attributes and measurements taken from 13 seagulls in Denmark.

seagullID	wingspan_mm	weight_g	seagullIsTagged	sex
11	711	290	1	0
12	680	305	0	0
13	689	305	1	0
14	687	320	0	1
15	698	310	0	0
16	699	320	0	0
17	712	290	0	1
18	696	305	0	1
19	690	290	0	1
20	698	320	1	1
21	690	335	1	0
22	695	310	0	0
23	685	310	0	0

(a) (10 points) Which of the variables are quantitative?

Only the variables wingspan and weight are quantitative, since they are the only quantities that are measurable on a given scale (i.e. for these it's mm & g). The others are categorical, meaning they only categorize each seagull by chosen traits and order them by ID (which are not measurable).

(b) (15 points) Suppose you load the data into Python like this:

```
data = numpy.loadtxt("data.csv", skiprows=1, dtype="int")
```

The National Ornithological Society of Denmark is conducting a study on obesity in urban seagulls. A bird is obese when *both* the following conditions are true: it has a wing span of less than 700 mm and a weight of more than 300 g. Create a mask that selects all obese seagulls:

```
mask = (                    ) & (                    )
```

where `data[mask,0]` should return the obese seagull IDs.

For the mask we want to firstly have True for each row, where the entry is below 700 in the column with index 1 (i.e. when the wingspan is less than 700 mm). For the second part of the mask, we want True for each row, where the entry is above 300 in the column with index 2 (i.e. when the weight is above 300 grams):

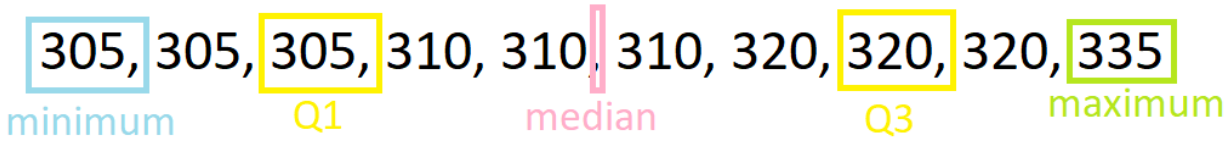
mask = (data[: , 1] < 700) & (data[: , 2] > 300)

(c) (25 points) Calculate the five number summary for the weights over all obese seagulls and report their sorted weights. If a five number summary value falls between two data points, take the midpoint (average) between the two data points.

Firstly I will write up the unsorted weights of the seagulls that classify as obese by the above mask:

305, 305, 320, 310, 320, 305, 320, 335, 310, 310

There are 10 obese seagulls out of the 13 data points. Their sorted weights and five number summary is given by:



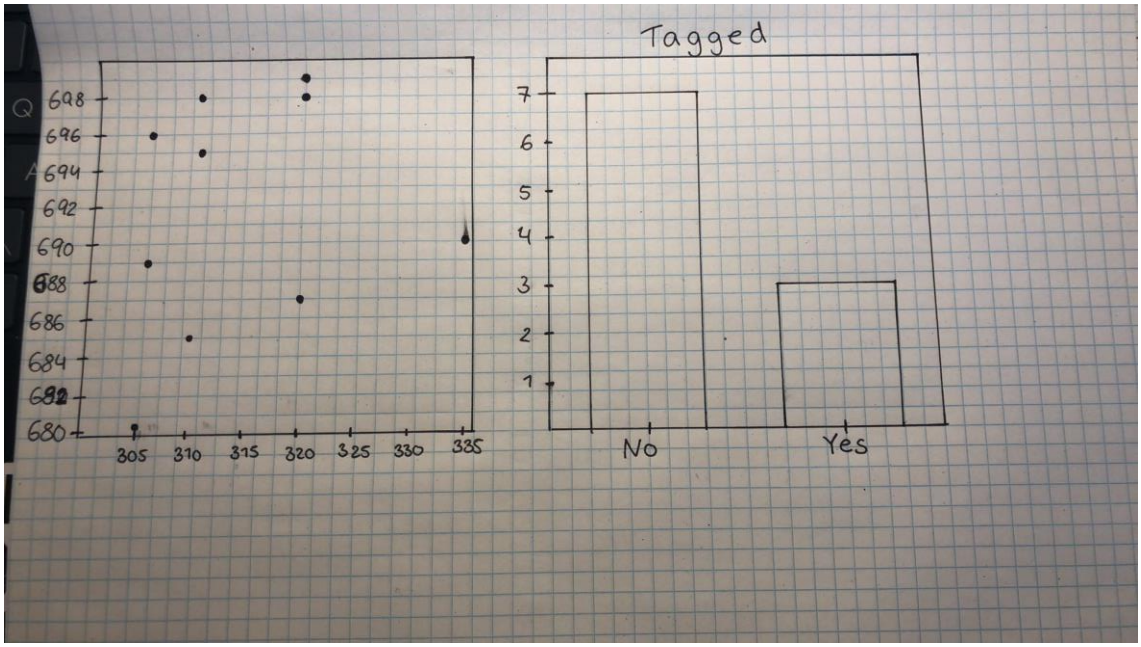
The five number summary is thereby [305 ; 305 ; 310 ; 320 ; 335]

Question 6:

Consider the data set `data_obese = data[mask]`, where `data` and `mask` are taken from the previous question.

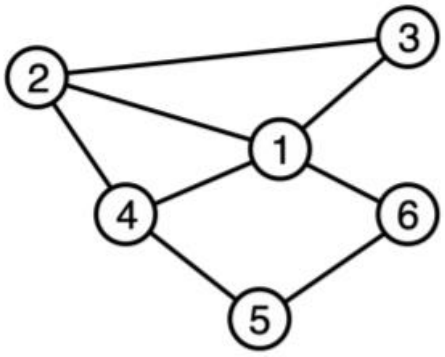
Draw by hand as exactly as possible what the following code will output. Draw two boxes next to each other, as shown below the code, for `axis1` and `axis2`. Draw all elements, including ticks, labels, titles, bars, scatterplot markers, at their correct locations. You do not need to get the colors right.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

categories, counts = np.unique(data_obese[:, 3], return_counts=True)

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axis1 = axes[0]
axis2 = axes[1]

axis1.scatter(data_obese[:,2], data_obese[:,1])

axis2.bar(categories, counts)
axis2.set_title("Tagged")
axis2.set_xticks(categories)
axis2.set_xticklabels(["No", "Yes"]);
```



Question 7:

Given the following network.



(a) (10 points) Calculate the degrees $k_1, k_2, \ldots, k_6$, and the average degree $\langle k \rangle$ of the network.

The degree is the number of edges directly related to each vertex. The average degree of a graph is just the average of the degrees over all nodes in the graph/network:

k_1	k_2	k_3	k_4	k_5	k_6	$\langle k \rangle$
4	3	2	3	2	2	2,6667

(b) (15 points) Calculate the clustering coefficients c_1 and c_6 of nodes 1 and 6.

We use the formula for the clustering coefficient, which tells us how well connected the neighbors of each node are:

$$c_1 = \frac{links\ between\ neighbors}{possible\ links\ betw.neighs.} = \frac{2}{6} = \frac{1}{3}$$

$$c_6 = \frac{0}{1} = 0$$

(c) (20 points) Calculate the average path length ℓ and the diameter D.

We write up all the shortest paths:

From:	To node 6	To node 5	To node 4	To node 3	To node 2
Node 1	1	2	1	1	1
Node 2	2	2	1	1	
Node 3	2	3	2		
Node 4	2	1			
Node 5	1				

Now we find the average of all the shortest paths. This will be the average path length in the graph:

$$\langle \ell \rangle = \frac{1+2+1+1+1+2+2+1+1+2+3+2+2+1+1}{15} = \frac{23}{15}$$

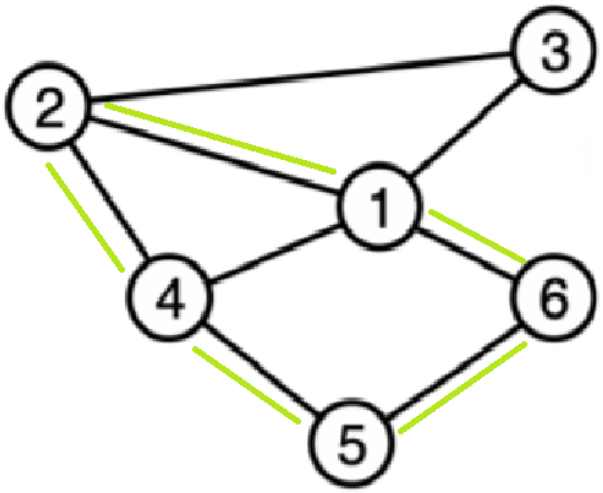
The Diameter is the longest of the shortest paths. In this case it would be the shortest path between 3 & 5:

$$D = 3$$

(d) (5 points) Give an example of a cycle of length 5.

An example of a length 5 cycle would be:

1 – 2 – 4 – 5 – 6 – 1



Question 8:

For the network from the previous question:

(a) (5 points) Write down the adjacency matrix.

The adjacency matrix has a 1 as an entry on each position i,j if there is an edge between i & j, otherwise 0:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(b) (5 points) Write down the edge list.

The edge list lists all the edges in the graph:

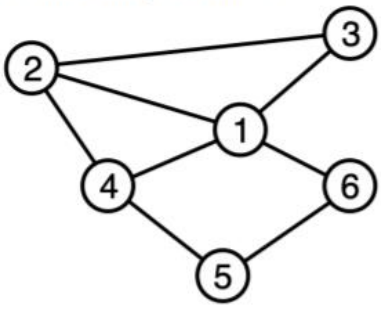
1	2
1	3
1	4
1	6
2	3
2	4
4	5
5	6

(c) (5 points) Write down the adjacency list.

The adjacency lists states for each node what other nodes are neighbors:

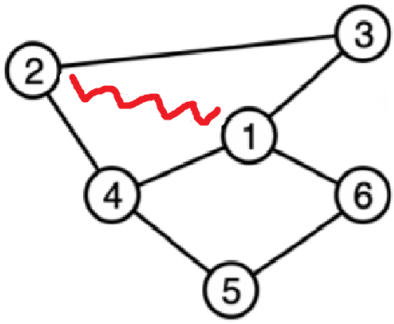
Initial	Terminal nodes
1	2, 3, 4, 6
2	1, 3, 4
3	1, 2
4	1, 2, 5
5	4, 6
6	1, 5

(d) i. (20 points) Remove from the same network below exactly one link to make the clustering coefficient $c_1 = 0$. Indicate in the figure below which link you remove.



ii. (15 points) For this new network: What is then the new average degree?

i. There are two links between neighbors of node 1, and since we are only allowed to remove 1 link, we can not just remove the links between the neighbors to obtain c_1=0. Instead, we see that the two links between the neighbors have node 2 in common, thus we can just remove the link between 1 & 2, arriving at:



Giving that none of the neighbors of node 1 are connected, i.e. $c_1 = 0$

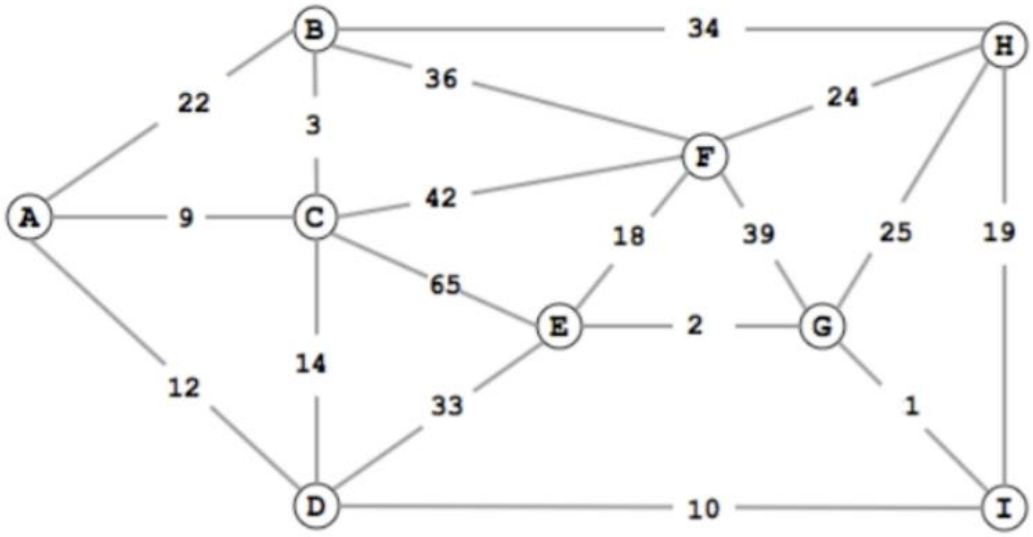
ii. We now decrease the degrees of node 1 & 2 by one, getting the average degree:

$$\frac{k_1 + k_2 + k_3 + k_4 + k_5 + k_6}{6} = \frac{3 + 2 + 2 + 3 + 2 + 2}{6} = \frac{14}{6} = \frac{7}{3} = 2{,}3333$$

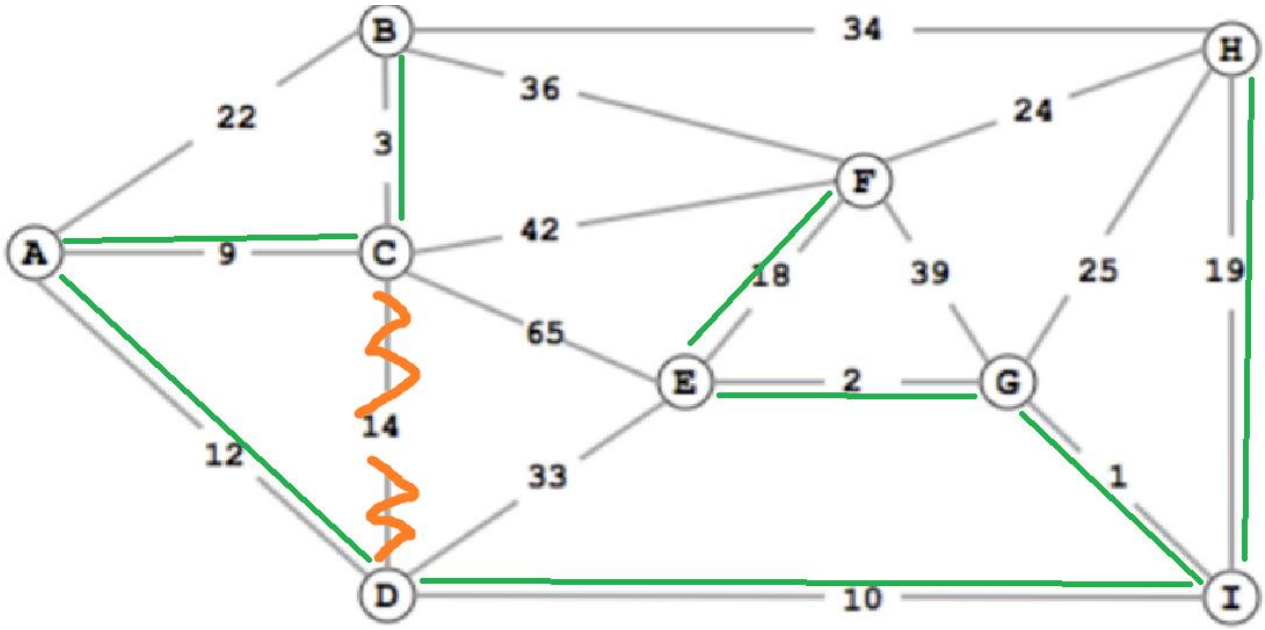
i.e. the average degree just decreases by 2/6, since it decreases by 2 in the numerator (1 for each terminal node), but stays the same in the denominator (same number of nodes still)😊

Question 9:

(a) (30 points) Find the minimum spanning tree of the graph below by using Kruskal's algorithm. List the edges in the order of selection by the algorithm.



With kruskals we add the edges in order with lowest weighted edge first, UNLESS adding it will create a cycle. We do this until it is a spanning tree:



Order of selection:

GI - EG - BC - AC - DI - AD - not CD EF - HI

Question 10:

Consider the following code:

```
class Client:
    bonus = 500

    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

        if self.balance < 4000:
            self.level = "Standard"
            self.balance += self.bonus
        else:
            self.level = "Premium"


    def deposit(self, amount):
        self.balance += amount
        return self.balance

class Savings(Client):
    interest_rate = 0.01

    def update_balance(self):
        self.balance += self.balance*self.interest_rate
        return self.balance
```

(a) (10 points) What is the relation between `Client` and `Savings`? Which key concept of object-oriented programming enables this relation?

'Savings' is a child of the parent-class 'Client' meaning it inherits the attributes from the parent-class. The key-concept behind this is the Object Oriented Programming (OOP) concept of *inheritance* that allows one class to inherit attributes from another class.

(b) (15 points) Suppose you own 5000 DKK in cash and you want to open a Savings account in this bank and deposit all your money there. What is an optimal way for you to do that and why? Also, write down the corresponding code that would be executed.

An optimal way to do this would be to begin by making an instance of 'Client' where we deposit less than 4000, since this would trigger the bonus and give us an extra 500. Then once we have created this instance, we can deposit the rest of the money, and then turn it to a savings-account. The code would look as so:

```
m_jona = Client("Mie Jonasson",3000)
m_jona.deposit(2000)
m_jona = Savings(m_jona)
```

And would give me a savings account with m_jona.balance = 5500

(c) (10 points) What is the output from the following code?

```
Jeff_Koons = Savings("Jeff Koons", 4000)
Jeff_Koons.update_balance()
```

This code will use the method of the savings-class, because we create Jeff_Koons as an instance of children-class 'Savings'. The method 'update_balance()' takes the balance and adds on 1% in interest, then returns the new balance. This means that

1) Self.balance = 4000 → self.balance += self.balance*self.interest_rate → self.balance += 4000*0.01 → self.balance = 4040
2) The output will thereby just be what the method returns:
   >>> *CODE*
   4040

(d) (10 points) What is the output from the following code?

```
bonus = 6000
Eve_Doe = Client("Eve Doe", 3000)
Client.deposit(Eve_Doe, 3000)
print(Eve_Doe.level)
```

In the first line we define a variable 'bonus=6000' but this is not the bonus of the class, since changing a class-attribute you would need to specify it to be Client.bonus

In the second line we initiate an instance of 'Client' where Eve deposits 3000. This runs the __init__(self, name, amount) function of the class, and initiates "Eve Doe" as a "standard"-level client and gives her the bonus of 500, meaning her balance is now 3500 by the conditional statement in the init-method.

The third line will add another 3000 to the account of instance Eve_Doe. This means that the balance of the instance will now be 6500. This line will return 6500, but will not print anything.

The fourth line will just print the level of the instance, which was created with the __init__ method, thus just printing 'Standard'

SUMMARY:

The code will output:

>>> *CODE*

Standard