Forside TDT4110

Institutt for datateknologi og informatikk

Eksamensoppgave i TDT4110 - Informasjonsteknologi, grunnkurs

Faglige kontakter under eksamen:

- Børge Haugset (tlf.: 934 20 190)
- Yngve Dahl (tlf.: 905 27 892)

Eksamensdato: 1. desember 2018
Eksamenstid (fra-til): 15:00 – 19:00
Hjelpemiddelkode/Tillatte hjelpemidler: D

Annen informasjon:
Det er angitt i prosent hvor mye hver deloppgave i eksamenssettet teller ved sensur. Les gjennom hele oppgavesettet før du begynner å løse oppgaven. Disponer tiden godt!

Merk! Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

1 Oppgave 1 - Teori (25%)

Marker det du mener er det mest riktige alternativet. Svaralternativene kommer i tilfeldig rekkefølge.
Feil svar gir IKKE minuspoeng.

1) Hvilket alternativ er IKKE et lag i TCP/IP-stabelen (stack)?

- Sammenkoblingslaget (Connection layer)
- Det fysiske laget (Physical layer)
- Internettlaget (Internet layer)
- Nettverksgrensesnitt-laget (Network layer)

2) Hvilken metode brukes for å håndtere duplikater og at pakker kommer i feil rekkefølge?

- ACK (retransmisjon)
- Sequencing (sekvensering)
- Replay
- Flow control (flytkontroll)

3) Hva er "replay error" i nettverkssammenheng?

- At en forsinket pakke fra tidligere sesjon blir akseptert i senere sesjon, og at korrekt pakke dermed blir avvist som duplikat.
- At en bekreftelse (ACK) sendes dobbelt, noe som kan føre til at avsender tror en pakke har kommet fram selv om den ikke har det.
- At det blir sendt duplikater til mottaker, som da må håndtere dette.
- At en forsinket etableringspakke oppretter en tidligere terminert forbindelse, og begynner å sende informasjonen på nytt.

4) Hva er "jitter" innenfor nettverk?

- Tiden det tar å overføre data over et nettverk fra et endepunkt til et annet.
- Mengden data som kan overføres innenfor en gitt tidsenhet.
- Endringer i forsinkelser og lengden på forsinkelsene.
- Variasjonen i mengden data som mottas til de forskjellige tidene av døgnet.

5) Hva går "buffer overflow" ut på?

- At det sendes masse data slik at det hoper seg opp. Dette kan føre til minneaksessproblemer og krasj, og må derfor rettes opp ved hjelp av en buffer.
- At det sendes enormt mange pakker med mye data til mottakeren. Kan føre til overbelastning slik at legitime forespørsler ikke kan betjenes.
- At maskinen må nøytraliseres etter store operasjoner. En buffer virker nøytraliserende.
- At det oversendes mer data enn hva mottaker forventer. Overskrider en databuffers grenser og skriver til nabolokasjoner i minnet. Kan føre til minneaksessproblemer, feil resultater og krasj.

6) Hva er "wiretapping"?

- Et trådløst system som stiller svakt mot angrep utenfra.
- At uvedkommende tar kontroll over en brukers datamaskin.
- At en angriper oppgir feilaktige opplysninger om en tjeneste eller et produkt, eller leverer tjenesten/produktet i dårligere tilstand enn forventet.
- At en angriper kopiere datapakker som traverserer nettet for å få tak i informasjon.

7) Hva er sant om linje- og pakke-svitsjing?

- Informasjonen i linje-svitsjede nettverk deles inn i mindre blokker med data kalt datagram.
- Kommunikasjonen mellom to parter i pakkesvitsjede nettverk påvirkes ikke av kommunikasjon mellom andre.
- Linje-svitsjing gir mer effektiv utnyttelse av nettverksressurser.
- I linje-svitsjing opprettes koblingen mellom partene ved behov, og avsluttes etter endt bruk.

8) Hvor mange bit består en IPv4-adresse av?

- 128
- 64
- 16
- 32

9) Hva identifiserer prefikset i en IPv4-adresse?

- En spesifikk datamaskin på nettverket.
- Det unike fysiske nettverket en datamaskin er tilknyttet.
- Hvilke datamaskiner som tilhører et nettverk.
- Om en datamaskin er en ruter eller ikke

10) Hva er en sub-nett maske (addresse maske)?

- Ingen av de andre alternativene.
- En verdi som spesifiserer den eksakte grensen mellom prefikset og suffikset i subnetting og klasseløs adressering.
- Matrise for tildeling av prefiks og suffiks i subnetting og klasseløs adressering.
- Funksjon som muliggjør deling mellom prefiks og suffiks.

11) Hva kalles mindre kretskort som plugges inn i hoved(krets-)kortet?

- Arvekort
- Anti hovedkort
- Sekundærkort
- Datterkort

12) Hva er Photolithography (fotolitografi)?

- En digitalisering av Punch Cards (hullkort) som har gjort det mulig for en datamaskin å lese digital informasjon.
- En teknikk som har gjort at integrerte kretser har blitt enklere å lage. Ledninger printes på chipene i flere lag.
- Noe som brukes når man skal fotografere kretser ned på hovedkortet
- En type RAM som lagrer data permanent.

13) Hva er riktig om harddisker?

- Permanent og random access.
- Volatilt og random access.
- Volatilt og sequential access.
- Permanent og sequential access

14) Hva er hovedoppgaven til ALU?

- Sette i gang operativ systemet
- Utføre regneoperasjoner
- Peke på riktig minneadresse
- Hente data fra minnet

15) Hvilke 2 registre finner vi i kontrollenheten?

- Programteller og instruksjonsregister
- Instruksjonsregister og kontrollregister
- Programteller og dataregister
- Programteller og kontrollregister

16) Hvilket tall i titallssystemet representerer det binære tallet 100111?

- 71
- 39
- 43
- 27

17) Hvilken webfargekode (heksadesimalfargekode) representerer hvit?

- #808080
- #AAAAAA
- #000000
- #FFFFFF

18) Informasjon som beskriver informasjon kalles:

- kollating
- spesialdata
- sirkulær data
- metadata

19) Hva sier Nyquist-regelen for sampling?

- Nyquist-regelen sier at samplingsfrekvensen må være minst like rask som den raskeste frekvensen. Ettersom menneskelige ører kan høre lyder opp til ca. 20 000Hz, vil samplingsfrekvens på 20 000Hz oppfylle Nyquists regel for digital lydopptak.
- Nyquist-regelen sier at samplingsfrekvensen må være minst fire ganger så rask som den raskeste frekvensen. Ettersom menneskelige ører kan høre lyder opp til ca. 20 000Hz, vil samplingsfrekvens på 80 000Hz oppfylle Nyquists regel for digital lydopptak.
- Nyquist-regelen sier at samplingsfrekvensen må være minst dobbelt så rask som den raskeste frekvensen. Ettersom menneskelige ører kan høre lyder opp til ca. 20 000Hz, vil samplingsfrekvens på 40 000Hz oppfylle Nyquists regel for digital lydopptak.
- Nyquist-regelen sier at samplingsfrekvensen må være minst halvparten av den raskeste frekvensen. Ettersom menneskelige ører kan høre lyder opp til ca. 20 000Hz, vil samplingsfrekvens på 10 000Hz oppfylle Nyquists regel for digital lydopptak.

20) Hva er sampling?

- Å ta målinger på regelmessige intervall
- Å konvertere fra digital til analog lyd
- Å konvertere fra analog til digital lyd
- Å ta målinger på uregelmessige intervall

Maks poeng: 20

```
def myst(val1, val2):
    if (val1 and val2):
        return 'a'
    elif (val1 and not val2):
        return 'b'
    elif (not val1 and val2):
        return 'c'
    else:
        return 'd'
```

Hva returneres ved funksjonskallet under?
myst(not(not False and True), not(not False or False))

Velg ett alternativ

- c
- d
- a
- b

Maks poeng: 1

3 Oppgave 2b - Kodeforståelse (5%)

Gitt funksjonen under:

```
def myst(numbers, flag):
    n = []
    for i in numbers:
        if (flag and (i%2 == 0)) or (not flag and (i%2 != 0)):
            if i not in n:
                n.append(i)
        n.sort()
    return n
```

Hva skrives ut når følgende funksjonskall kjøres:
print(myst([1,8,6,2,5,4,9,4,8,0], True))

Velg ett alternativ

- [9 ,5, 1], True
- [1 ,5, 9]
- [8 ,6, 2, 4, 4, 8, 0]
- [0, 2, 4, 4, 6, 8, 8]
- [0, 2, 4, 6, 8]
- [2, 4, 6, 8]

Maks poeng: 1

4 Oppgave 2c - Kodeforståelse (5%)

Funksjonen **count_items** tar inn ei liste med navn på personer (tekststrenger) og teller antall

forekomster av hvert navn i lista. Funksjonen skal returnere en *dictionary* hvor hvert navn i lista er representert én gang og som inneholder informasjon hvor mange ganger navnet forekommer i lista.

```
def count_items(names)
  d = {}
  for name in names
    KODE#1
  return d
```

Eksempel på kall av funksjonen count_items:

```
>>> print(count_items(['Anna', 'John', 'John', 'Thomas', 'Jane', 'Anna', 'Lis']))
{'Anna': 2, 'John': 2, 'Thomas': 1, 'Jane': 1, 'Lis': 1}
```

Hvordan skal linjen med innholdet **#KODE1** i koden til *count_items* se ut for at funksjonen skal fungere på måten beskrevet ved kjøring?

Velg ett alternativ

- d[name] = d.get(name,0)+1
- d[name] = d.get(name,0)
- d[names] = d.get(name,0)+1
- d[name] = d.get(name,1)+1
- d[name] = d.get(name,1)
- d == names.count(name)

Maks poeng: 1

5 Oppgave 2d - Kodeforståelse (5%)

Gitt funksjonen:

```
def myst(t1,t2):
    n = 2
    while (t1 % n != 0) or (t2 % n != 0):
        n+= 1
    return n
```

Hva skrives ut når følgende kode kjøres?

print(myst(49,42))

Velg ett alternativ

- 4
- 9
- 8
- 6
- 5
- 7

Maks poeng: 1

6 Oppgave 2e - Kodeforståelse (10%)

Funksjonen **bin_search** beskrevet under er ment å utføre et et iterativt binærsøk:

```
def bin_search(values, val, imin, imax):
    while imin < imax:
        #KODE1
        if #KODE2
            return True
        elif #KODE3
            imin = imid+1
        else:
            imax = imid-1
    return False
```

Eksempel på kall av funksjonen **bin_search**:

```
>>> A=[1,2,3,9,11,13,17,25,57,90]
>>> print(bin_search(A,57,0,len(A)-1))
True
```

Hva er riktig kode for **#KODE1**, **#KODE2** og **#KODE3**?

1) Velg ett alternativ for #KODE1

- imid = imin+imax
- imid = (imin+imax)//2
- imid = (imin+imax)*2
- imid = imax//2
- imid = (imax-imin)//2
- imid = (imin+imax)%2

2) Velg ett alternativ for #KODE2

- val == imid:
- val == imax:
- val == imin:
- val == liste[imin]
- val == liste[imid]:
- val == liste[imax]

3) Velg ett alternativ for #KODE3

- val < values[imid]:
- val == imax:
- val > values[imid]:
- val >= values[imid]:
- val > values[imin]:
- val > values[imax]:

Maks poeng: 3

7

Oppgave 2f - Kodeforståelse (5%)

Funksjonen **sumofn** (beskrevet under) er ment å være en rekursiv funksjon som beregner summen av alle heltall fra og med 1 til og med verdien funksjonen tar inn.

```
def sumofn(n):
    if n==1:
        return 1
    #Kode1
```

Eksempel på kall av funksjonen **sumofn**:

```
>>> sumofn(5)
15
```

Hvordan skal linjen merket **#Kode1** i funksjonen **sumofn** se ut for at funksjonen skal fungerer slik beskrevet over?

Velg ett alternativ

- return sumofn(sumofn-1)
- return n*(n-1)
- return sumofn(n-1)
- return n+sumofn(n+1)
- return n+sumofn(n-1)
- return sumofn*n

Maks poeng: 1

Useful Python functions and commands

Built-in:

format(numeric_value, format_specifier)

Formats a numeric value into a string according to the format specifier, which is a string that contains special characters specifying how the numeric value should be formatted. Examples of various formatting characters are “f=floating-point, e=scientific notation, %=percentage, d=integer”. A number before the formatting character will specify the field width. A number after the character “.” will format the number of decimals.

%

Remainder (modulo operator): Divides one number by another and gives the remainder.

len(s)

Return the length (the number of items) of a string, tuple, list, dictionary or other data structure.

int(x)

Convert a string or number to a plain integer.

float(x)

Convert a string or a number to floating point number.

str([object])

Return a string containing a nicely printable representation of an object.

range(stop)

Returns a list with integers from 0, up to but not including stop. range(3) = [0, 1, 2]. Often used in combination with for loops: for i in range(10)

range([start], stop[, step])

start: Starting number of the sequence.

stop: Generate numbers up to, but not including, this number.

step: Difference between each number in the sequence.

chr(i)

Return a string of one character whose ASCII code is the integer i. For example, chr(97) returns the string 'a'. This is the inverse of ord()

ord()

Given a string of length one, return an integer representing the Unicode code point of the character when the argument is a unicode object, or the value of the byte when the argument is an 8-bit string. For example, ord('a') returns the integer 97.

tuple(iterable)

Accepts something iterable (list, range etc.) and returns the corresponding tuple.

if x in iterable:

Returns True if x is an item in iterable.

for idx, x in enumerate(iterable)

Enters a loop with x being one and one item from iterable. idx is an integer containing the loop number.

try: except: else: finally:

```
try:
   # Code to test
except:
   # If code fails. Many exception types, like IOError for file operations.
   # Variant: except Exception as exc # Let's you print the exception.
else:
```

```
    # Runs if no exception occurs
finally:
    # Runs regardless of prior code having succeeded or failed.
```

String operations:

s.isalnum()

Returns true if the string contains only alphabetic letters or digits and is at least one character of length. Returns false otherwise.

s.isalpha()

Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.

s.isdigit()

Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.

s.isspace()

Returns true if the string contains only whitespace characters, and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (\n), and tabs (\t)).

s.ljust(width)

Return the string left justified in a string of length width.

s.rjust(width)

Return the string right justified in a string of length width.

s.join(list)

Returns a string listing all items in the list, separated by s.

s.lower()

Returns a copy of the string with all alphabetic letters converted to lowercase.

s.upper()

Returns a copy of the string with all alphabetic letters converted to uppercase.

s.strip()

Returns a copy of the string with all leading and trailing white space characters (spaces, newlines and tabs) removed.

s.strip(char)

Returns a copy of the string with all instances of char that appear at the beginning and the end of the string removed.

s.split(str)

Returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified).

str.splitlines([keepends])

Return a list of the lines in the string, breaking at line boundaries. This method uses the universal newlines approach to splitting lines. Line breaks are not included in the resulting list unless keepends is given and true.

Python recognizes "\r", "\n", and "\r\n" as line boundaries for 8-bit strings.

s.endswith(substring)

The substring argument is a string. The method returns true if the string ends with substring.

s.startswith(substring)

The substring argument is a string. The method returns true if the string starts with substring.

s.find(substring)

The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found the method returns -1.

s.replace(old, new)

The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.

*str.format(*args, **kwargs)*

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

List operations:

s[i:j:k]

Return slice starting at position i extending to position j every k items. Can also be used for strings.

item in s

Determine whether a specified item is contained in a list.

min(list)

Returns the item that has the lowest value in the sequence.

max(list)

Returns the item that has the highest value in the sequence.

s.append(x)

Append new element x to end of s.

s.insert(index,item)

Insert an item into a list at a specified position given by an index.

s.index(item)

Return the index of the first element in the list containing the specified item.

s.pop()

Return last element and remove it from the list.

s.pop(i)

Return element i and remove it from the list.

s.remove(item)

Removes the first element containing the item.

s.reverse()

Reverses the order of the items in a list.

s.sort()

Rearranges the elements of a list so they appear in ascending order.

Dictionary operations:

d.clear()

Clears the contents of a dictionary

d.get(key, default)

Gets the value associated with a specific key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.

d.items()

Returns all the keys in a dictionary and their associated values as a sequence of tuples.

d.keys()

Returns all the keys in a dictionary as a sequence of tuples.

d.pop(key, default)

Returns the value associated with a specific key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.

d.popitem()

Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.

d.values()

Returns all the values in dictionary as a sequence of tuples.

File operations:

open()

Returns a file object, and is most commonly used with two arguments: open(filename, mode). Mode can be ‘r’ (read only), ‘w’ (writing only), ‘a’ (appending), ‘r+’ (both reading and writing). Adding 'b' to the read or write attribute lets you use binary mode to save the value of variables to file and load from them. For binary to work you need to import the pickle library.

f.read(size)

Reads data from file and returns it as a string. Size is an optional and if left out the whole file will be read.

f.readline()

Reads a single line from the file (reads until and including a newline character (\n) is found), and returns it as a string.

f.readlines()

Reads data from the file and returns it as a list of strings.

f.write(string)

Writes the contents of string to file.

f.writelines(list)

Writes a sequence of strings (typically a list of strings) to file.

f.close()

Close the file and free up any system resources taken up by the open file.

Libary operations:

pickle.dump(obj, file)

Write a pickled representation of obj to the open file object file.

pickle.load(file_object)

Read a string from the open file object file and interpret it as a pickle data stream, reconstructing and returning the original object hierarchy.

auctions.py:

```
# The module auction.py contains some helper functions.


# get_auction_data _really_ uses filename to load all auction
# values from file into a two-dimensional list. We won't show
# the complete code here, but instead return a set of values
```

```
    # that represents the kind of data one would get. The data
    # fits the examples given in the assignments.
    def get_auction_data(filename):
      # Code that collects the data, not shown here.
      # Instead we hard code the return value.
        return [['Customer','vase','maleri','sykkel','bilbane','tv'],
                ['Per',100,0,200,0,1500],
                ['Ida',110,50,200,0,1500],
                ['Ottar',200,600,200,0,1700],
                ['Dag',200,600,200,0,1700],
                ['Lise',400,600,0,0,0]]

# Bid returns the bid from customer on item, given dataset data.
def bid(customer, item, data):
    column = data[0].index(item)
    for i in data:
        if i[0] == customer:
            return i[column]



# Helper function: sorts a list of lists based on its second value
# Values sorted lowest first, increasing
def sort_list(liste,column):
    return sorted(liste,key=lambda l:l[column], reverse=True)
```

Oppgave 3 - Oppgavebeskrivelse

Auksjonshjelp

En venninne av deg jobber deltid i et lite auksjonsfirma. Firmaet holder auksjoner på nett. Hun har fått i oppdrag å programmere en løsning for å holde rede på auksjonsdata. Hun har kommet et stykke, men er ikke helt i mål. Siden du har tatt kurs i programmering har det nå blitt din oppgave å hjelpe henne et stykke på veien.

Oppgavens datastruktur

Du vil i denne oppgaven forholde deg til en datastruktur som består av en todimensjonal liste, kalt data. Du vil snart se hvordan du kan hente den ut, men den har dette formatet:
>>> data

[['Customer', 'vase', 'painting', 'bicycle', 'lego', 'tv'], ['Per', 100, 0, 200, 0, 1500], ['Ida', 110, 50, 200, 0, 1500], ['Ottar', 200, 600, 200, 0, 1700], ['Dag', 200, 600, 200, 0, 1700], ['Lise', 400, 600, 0, 0, 0]]

Første element i listen *data* er en liste som inneholder 'navnene' på alle gjenstandene som auksjoneres ut. **Gjenstandene står i en tilfeldig rekkefølge** – du kan altså ikke forvente at du finner gjenstanden 'vase' som element 1 som det gjør over. De resterende elementene i listen data er en ny 'subliste' med hver enkelt kundes navn, og så deres bud på hver gjenstand. Gjenstander en kunde ikke har bydd på vil stå oppført med budverdi 0. **Navnene står i tilfeldig rekkefølge**, du kan altså ikke forvente at Per sine bud står i data[1] slik eksempelet over viser. Du kan derimot forvente at dersom 'vase' er på plass 1 i data[0], så befinner hver kunde sitt bud på vasen seg også på plass 1 i kundens subliste.

Eksisterende funksjoner

Det eksisterer allerede noen funksjoner som det kan være lurt å forholde seg til. Disse funksjonene er skrevet i modulen *auction.py* som ligger i den samme folderen som du skal legge koden din. Du vil finne en lenger beskrivelse av funksjonsnavn, innparametre og returverdier nederst i funksjonshjelpen til venstre. De presenteres også under: Disse skal altså IKKE programmeres av deg, men du må vise hvordan du eventuelt bruker dem!

Venninnen din har allerede skrevet funksjonen *get_auction_data* for å lage den todimensjonale listen *data* basert på en fil som inneholder alle auksjonsdata.

def ***get_auction_data****(filename)*: # Existing function, populates your needed 2d-list

    return data # data is a two-dimensional list, format described below

get_auction_data tar inn et filnavn (du kan bruke '*auctiondata.txt*'). Den returnerer den todimensjonale listen data beskrevet over.

Eksempel:

>>> data = auction.get_auction_data('auctiondata.txt')

>>> data

[['Customer', 'vase', 'painting', 'bicycle', 'lego', 'tv'], ['Per', 100, 0, 200, 0, 1500], ['Ida', 110, 50, 200, 0, 1500], ['Ottar', 200, 600, 200, 0, 1700], ['Dag', 200, 600, 200, 0, 1700], ['Lise', 400, 600, 0, 0, 0]]

def ***bid****(customer, item, data)*: # finds one *customers* bid on a particular *item* on dataset *data*.

    return customer_bid_on_item # Returns an integer which is the customers bid on item.

bid(customer, item, data) har som input kundens navn, gjenstandens navn, og datastrukturen nevnt over. Den returnerer kundens bud på den aktuelle gjenstanden.

Eksempel:

>>> auction.bid('Per','tv',data)

1500 # Pers bid on the tv.

def ***sort_list****(mylist,column): #* list is a two-dimensional list, column is the column in each sub-list

to sort everything by

    return two-dimensional list   # sorted by values in column number *column*

sort_list(list, column) er en funksjon som tar imot en todimensjonal liste, og sorterer den etter kolonnenummeret som oppgis som parameter 2.

Eksempel:

>>> auction.sort_list([['a',3],['b',12],['c',10]] , 1)

[['b',12], ['c',10], ['a',3]]

the sublist ['b',12] comes first, as 12 is the highest of 3, 12 and 1.

number and we sort by column 1.

8 Oppgave 3a – Programmering (5%)

Skriv funksjonen **menu** som skal skrive ut til skjerm ulike oppgaver auksjonsprogrammet støtter, og la brukeren taste inn (via tastatur) hvilken oppgave han/hun vil ha gjennomført. Funksjonen har ingen parametre. Eksemplet under viser hvordan menyen skal se ut når funksjonen **menu** kalles. Dersom brukeren taster et gyldig valg (dvs. tegnet *i*, *s* eller *q* – se eksemplet under) skal funksjonen returnere en streng betående av tegnet som ble tastet. I motsatt tilfelle skal funksjonen returnere *None*.

Eksempel på bruk av funksjonen:

>>> menu()

Select task:

'*i*' *to show the highest bid of an item,*

'*s*' *to save all winning bids to file,*

'*q*' *to exit.*

Choice:

Skriv ditt svar her...

```
1
```

Maks poeng: 10

9 Oppgave 3b – Programmering (10%)

Skriv funksjonen **item_offers** som har inn-parametrene **item** og **data**. Parameteren **item** er en en streng angir navnet på en auksjonsgjenstand (f.eks. *tv*).
Parameteren **data** angir en to-dimensjonal liste med auksjonsdata formatert som beskrevet i den innledende oppgavebeskrivelsen (også vist først i eksemplet nedenfor). Funksjonen skal returnere en to-dimensjonal liste som inneholder navnet til budgiverne og budene de har gitt på den angitte auksjonsgjenstanden (se under for eksempel på struktur).

Eksempel på bruk:

>>> data
[['Customer', 'vase', 'maleri', 'sykkel', 'lego', 'tv'], ['Per', 100, 0, 200, 0, 1500], ['Ida', 110, 50, 200, 0, 1500], ['Ottar', 200, 600, 200, 0, 1700], ['Dag', 200, 600, 200, 0, 1700], ['Lise', 400, 600, 0, 0, 0]]

>>> item_offers('tv',data)

[['Per', 1500], ['Ida', 1500], ['Ottar', 1700], ['Dag', 1700], ['Lise', 0]]

Skriv ditt svar her...

```
1
```

Maks poeng: 10

10 Oppgave 3c – Programmering (10%)

Skriv funksjonen **item_winner** som tar inn-parametrene **item** og **data**. Parameteren item er en streng som angir navnet på en auksjonsgjenstand (f.eks. *tv*). Parameteren **data** angir en to-dimensjonal liste med auksjonsdata formatert som beskrevet i den innledende oppgavebeskrivelsen. Funksjonen skal returnere en liste som inneholder to elementer: Navnet på budgiveren med det høyeste budet og hvor høyt budet var. Dersom det ikke eksisterer et bud på en gitt gjenstand skal funksjonen returnere en tom liste. Forutsett at parameteren **item** alltid identifiserer en gjenstand som finnes på auksjonen.

Bruk gjerne funksjonen **sort_list** beskrevet i den innledende oppgavebeskrivelsen (bruken er illustrert nedfor) i forbindelse med sortering.

Dersom flere budgivere har budt høyeste verdi er vinneren den første – det er denne budgiveren som ga budet først hvis du bruker sort_list.

Eksempel på bruk av funksjonen **sort_list.**

>>> auction.sort_list([['a',3],['b',12],['c',10]] , 1) # Du trenger ikke programmere denne.

[['b',12], ['c',10], ['a',3]]

Eksempel på bruk av funksjonen **item_winner**:

>>> data

[['Customer', 'vase', 'maleri', 'sykkel', 'lego', 'tv'], ['Per', 100, 0, 200, 0, 1500], ['Ida', 110, 50, 200, 0, 1500], ['Ottar', 200, 600, 200, 0, 1700], ['Dag', 200, 600, 200, 0, 1700], ['Lise', 400, 600, 0, 0, 0]]

>>> item_winner('tv', data)

['Ottar', 1700]

Skriv ditt svar her...

```
1
```

Maks poeng: 10

11 Oppgave 3d – Programmering (5%)

Skriv funksjonen **all_winners_dict** som tar inn-parameteren **data**, som er en todimensjonal liste formatert som beskrevet i innledningen til programeringsoppgavene (også illustrert under). Funksjonen skal finne ut hvilken budgiver som har gitt det høyeste budet for hver enkelt auksjonsgjenstand. Funksjonen skal returnere en dictionary. Denne skal bruke navnet på hver gjenstand som nøkkel. Til hver nøkkel skal det knyttes en verdi i form av en liste bestående av to elementer: (1) navnet på budgiveren (vinneren av auksjonen) og (2) vinnerbudet.

Eksempel på bruk:

>>> data
[['Customer', 'vase', 'maleri', 'sykkel', 'lego', 'tv'], ['Per', 100, 0, 200, 0, 1500], ['Ida', 110, 50, 200, 0, 1500], ['Ottar', 200, 600, 200, 0, 1700], ['Dag', 200, 600, 200, 0, 1700], ['Lise', 400, 600, 0, 0, 0]]

>>> all_winners_dict(data)

{'vase': ['Lise', 400], 'painting': ['Ottar', 600], 'bicycle': ['Per', 200], 'lego': [], 'tv': ['Ottar', 1700]}

Skriv ditt svar her...

```
1
```

Maks poeng: 10

12 Oppgave 3e – Programmering (5%)

Skriv funksjonen **save_auction_data** som tar inn-parameteren **data**. Parameteren data er en to-dimensjonal liste med auksjonsdata formatert slik som beskrevet i den innledende beskrivelsen til Oppgave 3. Funksjonen skal lagre resultatet (return-verdien) av funksjonskallet **all_winners_dict(data)** til binærfilen *auction_winners.db* som ligger i den samme mappen/katalogen som auksjonsprogrammet. Brukeren skal få beskjed om hvorvidt det har lyktes å skrive dataene til binærfilen eller ikke (se eksempel under).

Eksempel på vellykket lagring:

>>> save_auction_data(data)

Dictionary data successfully written to file.

Eksempel på feil ved lagring:

>>> save_auction_data(data)

Could not write dictionary data to file.

Skriv ditt svar her...

```
1
```

Maks poeng: 10

13 Oppgave 3f - Programmering (5%)

Skriv funksjonen **task**. Funksjonen skal først opprette en variabel *data*, og sette denne til resultatet av kallet til funksjonen **get_auction_data('auctiondata.txt')** fra modulen *auction.*

Deretter skal funksjonen kalle **menu** og lagre returverdien fra denne i en variabel. Hva funksjonen **task** så skal gjøre avhenger av verdien til variabelen:

> *'i'*: Brukeren skal bli bedt om å skrive inn navnet på en auksjonsgjenstand. Deretter skal funksjonen kalle **item_winner** (Oppgave 3c) med navnet på auksjonsgjenstanden, og variabelen *data* som argument. Funksjonen skal så skrive ut hvem som har vunnet gjenstanden, og hva budet var (eksempelvis *Vinner av tv er Ottar som bød 1700.*)
> Hvis det ikke har kommet inn noe bud på en gjenstand (som bilbanen i eksempelet) skal det skrives ut *Ingen bud på lego.*
>
> '*s*': Funksjonen skal kalle **save_auction_data** (Oppgave 3e) med variabelen *data* som argument.
>
> '*q*': Programmet skal avsluttes.

Menyen skal presenteres igjen helt til brukeren velger å avslutte med *q.*

Skriv ditt svar her...

```
1
```

Maks poeng: 10