

Datastrukturer och algoritmer (Python)

5DV150

Obligatorisk uppgift 1 (OU1)

Författare: Alfons Edbom Devall (alde0033)

Inlämningsdatum: 21/9 - 2021

Innehållsförteckning

Introduktion	3
Syfte	3
Introduktion	3
Bakgrundsinformation.....	3
Abstrakt datatyp.....	3
Implementerad datatyp	3
Abstrakta datatypen Lista:	4
Del 1	5
Del 2	7
Reflektion	8
Testkörningar.....	9
Testkörningar Del 1	9
Testkörningar Del 2.....	12
Referenser.....	14

Introduktion

Syfte

1. Ge en förståelse varför man använder sig av abstrakta datatyper
2. Ge en förståelse för skillnaden mellan en abstrakt datatyp (ADT) och en implementation av en datatyp
3. Öva på skriftlig presentation och dokumentation

I rapporten har den abstrakta datatypen Lista blivit implementerad på två olika sätt (Lista som fält och Lista som 2-länkade celler) och använts för att göra två olika program. Eftersom dessa implementationer är gjord på samma abstrakta datatyp ska det även gå att använda deras gränssnitt på samma sätt, detta kontrolleras även i testkörningar av programmet genom att ändra vilken implementation man använder högst upp i programmet (genom att ändra vilken implementation man importerar).

Introduktion

I Del 1 har ett program gjorts som först läser in data från en fil, skapar en lista med data och sedan skriver ut innehållet, element för element.

I Del 2 har ett andra program gjorts som först läser in data från en fil till en lista och frågar användaren i vilken ordning den ska sorteras, sorterar den enligt anvisningarna, och till sist skriver ut resultatet. Programmet erbjuder även att sortera listan igen eller avsluta programmet efter en sortering är gjord.

I rapporten har programmen skrivits i Python 3

Bakgrundsinformation

För att förstå rapporten kan det vara bra att ha koll på följande begrepp

Abstrakt datatyp

När man beskriver eller diskuterar hur man använder en datatyp utan att ta någon hänsyn till antingen *om* eller *hur* den är realiserad eller implementerad i ett programspråk. Man fokuserar alltså inte på vad dataobjekten "egentligen" består av eller hur de olika operationerna utförs, utan snarare *vad* operationerna gör samt vad de får för resultat¹

Implementerad datatyp

En implementerad datatyp är komplett konstruerad från grunden och är klar att användas i ett program.²

¹ Janlert, Lars-Erik; Wiberg och Torbjörn Wiberg. *Datatyper och algoritmer*. Uppl. 2:6. Malmö AB, Sverige 2011. S.27

² Janlert, Lars-Erik; Wiberg och Torbjörn Wiberg. *Datatyper och algoritmer*. Uppl. 2:6. Malmö AB, Sverige 2011. S.27

Abstrakta datatypen Lista:

En lista består av ett ändligt antal linjärt ordnade element. Elementen är ordnade med en före/efter relation till varandra. Varje element i en lista har två egenskaper: De har ett värde och en position; ett visst läge i strukturen. I Figur 1 kan gränsytan till Lista ses.

```
abstract datatype List(val)
auxiliary pos
  Empty() -> List(val)
  Insert(v:val,p:pos,l:List(val))->(List(val),pos)
  Iempty (l:List(val)) -> Bool
  Inspect (p:pos,l:List(val)) -> val
  First (l:List(val)) -> pos
  End (l:List(val)) -> pos
  Next(p:pos,l:List(val)) -> pos
  Previous(p:pos,l:List(val)) -> pos
  Remove((p:pos,l:List(val)) ->(List(val),pos)
```

3

Figur 1 Visar gränsytan till datatypen Lista. Dvs. vilka operationer som ska kunna utföras på datatypen samt vad som ska returneras av sagd operation

Gränsytan till en abstrakt datatyp visar vilka operationer som kan utföras på ett objekt som tillhör den datatypen, samt vad operationen ska returnera.⁴

Notera, I de implementerade datatyperna heter Empty() istället List().

³ Kallin, Lena. *F2_cell_lista_stack_testning.pdf*. Föreläsning 5DV150 Datastrukturer och algoritmer (Python) 2021. Canvas. S.8

⁴ Janlert, Lars-Erik; Wiberg och Torbjörn Wiberg. *Datatyper och algoritmer*. Uppl. 2:6. Malmö AB, Sverige 2011. S.39–47

Del 1

I uppgift 1 gjordes ett program som: Först gör en tom lista, läser in data från en fil rad för rad, där värdet från varje rad i filen läggs till i slutet av listan och till sist skriver ut innehållet i listan från första elementet till sista.

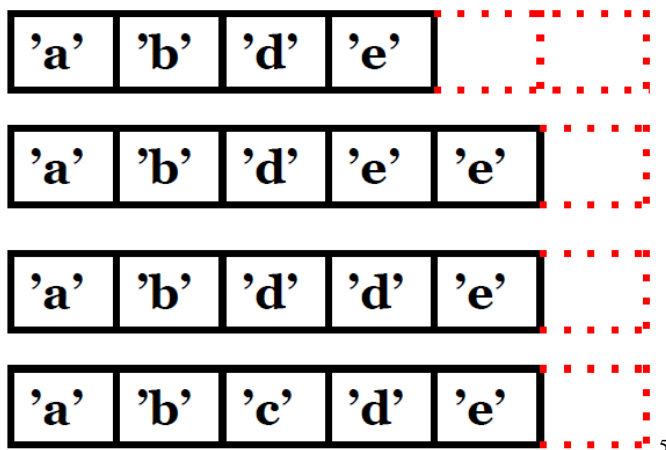
För att se fullständiga programmet, se OU1-1.py. Programmet ska fungera med båda implementationerna av datatypen Lista (ListAsArray samt ListAsTwoCell). Testkörningar för del 1 kan ses under (Testkörningar Del 1)

För att lägga till element i listan användes funktionen *insert(position, obj)* och här nedan kommer en mer genomgående beskrivning på hur *insert* är implementerad i de två fallen (ListAsArray och ListAsTwoCell):

ListAsArray

I den här implementationen av lista så fyller man först ett fält med ett "stort nog" antal element med värdet None och sätter slutpositionen i listan till den första TOMMA platsen i fältet.

När listan är implementerad som ett fält och man använder *insert* så måste man först tilldela alla element placerade efter *position* till indexet "bakom" sig själv och sedan kan man lägga till ett det nya värdet *obj* framför *position*, sedan returneras positionen för det insatta elementet, detta medför att man även måste flytta slutpositionen för listan ett steg bakåt i fältet. För att visualisera detta, kan ett liknande exempel ses i Figur 2, där de röda rutorna är tomma index i fältet samt att man flyttar alla element efter positionen man vill sätta in ett element.



Figur 2 Exempel på hur man kan sätta in ett nytt värde ('c') i en lista implementerad som ett fält

Om man sätter in ett värde i en tom lista sätter man helt enkelt in ett värde på första lediga platsen i arrayen och flyttar sista positionen i listan ett steg bakåt.

Ska man sätta in ett värde först i listan måste man alltså flytta alla värden ett steg bakåt i arrayen och till sist lägga till värdet först i listan.

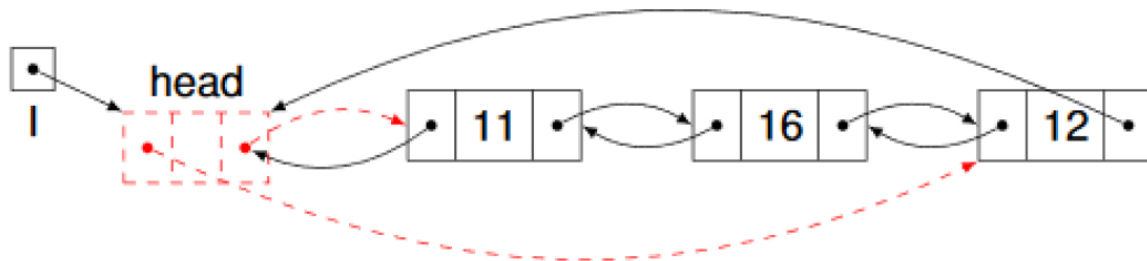
Sätter man in ett värde sist i listan behöver man bara ändra slutpositionen i listan ett steg längre bakåt i fältet efter att man satt in värdet.

Eftersom fält är en statisk datatyp måste man alltså först "flytta efter" alla element för att "ge plats" för det nya elementet när man använder *insert*.

⁵ Kallin, Lena. *F2_cell_lista_stack_testning.pdf*. Föreläsning 5DV150 Datastrukturer och algoritmer (Python) 2021. Canvas. S.18

ListAsTwoCell

I den här implementationen av lista så skapar man först en tom lista med en tom 2-cell som "huvud" i listan som håller reda på listans start och slut (vi ser till att First = Last i den tomma listan). När ett element sedan ska läggas till eller ta bort så ändras länkar (före/efter) mellan olika 2-cell-element. Ett exempel på hur man kan visualisera en lista implementerad med hjälp av 2-cell kan ses i Figur 3.



Figur 3 Exempel på hur man kan visualisera elementen i en lista implementerad som 2-cell.

När man använder *insert* i denna implementation så lägger man in ett nytt element före *position*. Detta görs genom att man först skapar ett nytt 2-cell-element och sätter dess värde till *obj*. Sedan sätts det nya 2-cell-elementets "efter-länk" till elementet bakom *position*, sedan sätts "före-länken" på det nya 2-cell-elementets till elementet på *position*. Sedan ändras elementet på *position* "efter-länk" till nya 2-cell-elementet och till sist ändras elementet bakom *position*:s "före-länk" till nya 2-cell-elementet och då är till sist det nya elementet sammanlänkat på rätt position i listan.

Sätter man in ett värde i en tom lista så görs länkarna så att både före och bakom länken till det nya 2-cell-elementet refererar till "huvudet"

Sätter man in ett värde först i en lista kommer "efter-länken" referera till "huvudet" och "före-länken" referera antingen till huvudet igen om listan tidigare var tom, eller till värdet som förut låg på första positionen.

Sätter man in ett värde sist i en lista så kommer "före-länken" antingen sättas till "huvudet" om listan tidigare var tom, eller till värdet som tidigare låg på sista positionen och "efter-länken" sätts till "huvudet"

Del 2

I uppgift 2 skulle ett program skrivas som läser i en fil och lagrar värdena i en lista, sorterar listan och sedan skriver ut resultatet. Den valda sorteringsalgoritmen skulle ha medelkomplexiteten $O(n \log(n))$. Dessutom skulle programmet innehålla ett enkelt menysystem så att användaren kan välja i vilken ordning data skulle sorteras, eller om man vill avsluta programmet.

För att se fullständiga programmet se OU1-2.py, men här kommer en beskrivning av hur programmet fungerar samt vilken sorteringsalgoritm som valdes och hur den implementerades.

För att läsa värdena från en fil och lagra värdena i en lista återanvändes koden från uppgift 1.

Sedan använde jag pseodokoden för Merge sort som finns i föreläsningssanteckningarna för kursen.⁶ För att den koden skulle fungera behövdes ett par hjälpfunktioner skapas. Dessa var funktioner som kunde ge en listans längd (ListLength) samt en funktion som kunde dela en lista på hälften. När den koden fungerade kunde mergesort funktionen modifieras så att den kunde sortera både i stigande och fallande ordning, detta gjordes genom att lägga till en extra parameter (comparefunc) som jämförde första elementet i två listor, och returnera antingen sant eller falskt beroende på om värdet i ena listan är större eller mindre än det andra värdet. Två funktioner gjordes där den ena returnerade True om första var mindre (LowToHigh), vilket då ger en lista sorterad i stigande ordning, samt en funktion som returnerade True om första var större (HighToLow), vilket då ger en lista sorterad i fallande ordning.

I den här uppgiften använde jag mig merge sort, vilket är en rekursiv sorteringsalgoritm som delar upp listan i mindre och mindre delar och sorterar dem först för att sedan slå ihop dem till en stor sorterad lista. Merge sort har även tidskomplexiteten $O(n \log(n))$. Vill man veta mer om merge sort finns det en mycket bra Wikipedia artikel för algoritmen.⁷

⁶ Kallin, Lena. *F5_LS_Sortering_Sokning.pdf*. Föreläsning 5DV150 Datastrukturer och algoritmer (Python) 2021. Canvas. S.13-14

⁷ Wikipedia. Merge sort. 2021. https://en.wikipedia.org/w/index.php?title=Merge_sort&oldid=1042060512 (Hämtad 2021-09-21)

Reflektion

I uppgiften har jag fått jobba väldigt praktiskt med två olika implementationer av den abstrakta datatypen Lista. För att kunna klara uppgiften och få fungerande program krävdes det att jag lärde mig och fick en bra förståelse för vad en abstrakt datatyp faktiskt är samt hur den skiljer sig emot en faktiskt implementerad datatyp. För att få båda implementationerna att fungera fick man inte vara och gräva alltför mycket "under ytan" eller utnyttja hur operationerna i klassen var implementerade utan snarare använda sig av gränssytan för hur den abstrakta datatypen Lista för att få robusta och fungerande program.

I början var det svårt och otydligt att skilja mellan vad som faktiskt skiljde implementationerna åt och hur dessa var olika ifrån den abstrakta datatypen Lista, men när jag kommit i gång och fått ett fungerande program för båda implementationerna för Uppgift 1 kändes det ganska klart vad skillnaden var och hur man ska använda den här datatypen för att få fungerande program oavsett vilken implementation som användes.

I uppgift 2 var det svårt till en början att förstå sig på hur merge sort faktiskt fungerar, framför allt med de rekursiva anropen till sig själv och hur det faktiskt hjälpte till med att få en sorterad lista. Med tanke på att vi fått en pseudokod för hur merge sort skulle kunna implementeras var det egentligen inte särskilt svårt att implementera en fungerande algoritm (sedan blev det ändå något litet fel som var väldigt tidskrävande att hitta). När jag väl fått en fungerande algoritm gick det relativt fort och lätt att göra den så man kan välja i vilken ordning listan skulle sorteras.

I avslutande vill jag säga att jag lärt mig väldigt mycket kring skillnaden mellan abstrakt och implementerad datatyp samt att jag fått mycket god erfarenhet av att skriva program i Python och jag skulle säga att jag utvecklats som programmerare.

Testkörningar

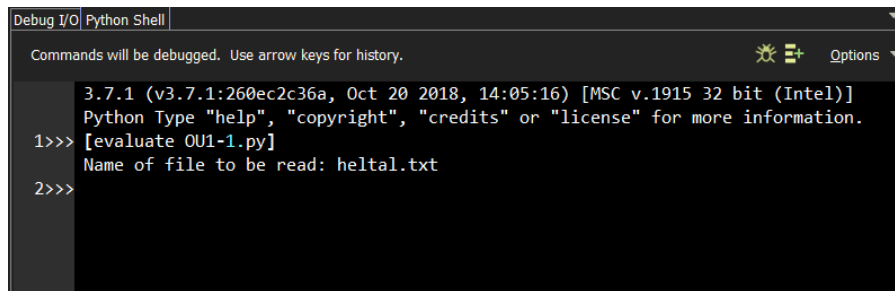
Testkörningar Del 1

I mina testkörningar har jag valt att testa fyra fall som beskrivs närmre nedan.

Fall 1: Köra programmet med en tom textfil

När programmet körs och användaren anger en tom fil ska programmet inte krascha och det ska inte heller skriva ut något till skärmen.

Det här blir resultatet av att köra programmet med en fil som inte innehåller någon data och heter heltal.txt:



```
Debug I/O Python Shell
Commands will be debugged. Use arrow keys for history.
3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)]
Python Type "help", "copyright", "credits" or "license" for more information.
1>>> [evaluate OUI-1.py]
Name of file to be read: heltal.txt
2>>>
```

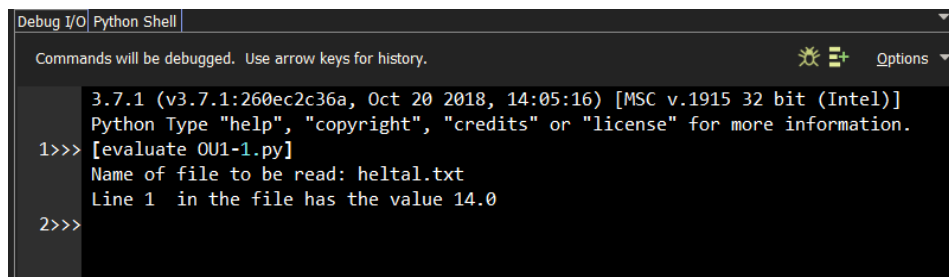
Figur 4 Testkörning av OUI-1.py där användaren anger en tom fil

Testkörningen kan ses i Figur 4 och programmet fungerar som förväntat.

Fall 2: Köra programmet med ett värde i textfilen

När programmet körs och användaren anger en fil med ett flyttal ska programmet skriva ut endast talet som finns i filen

Det här blir resultatet av att köra programmet med en fil som innehåller en rad med ett tal och heter heltal.txt. I detta fall innehåller heltal.txt endast talet 14:



```
Debug I/O Python Shell
Commands will be debugged. Use arrow keys for history.
3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)]
Python Type "help", "copyright", "credits" or "license" for more information.
1>>> [evaluate OUI-1.py]
Name of file to be read: heltal.txt
Line 1 in the file has the value 14.0
2>>>
```

Figur 5 Testkörning av OUI-1.py där användaren anger en fil med ett värde

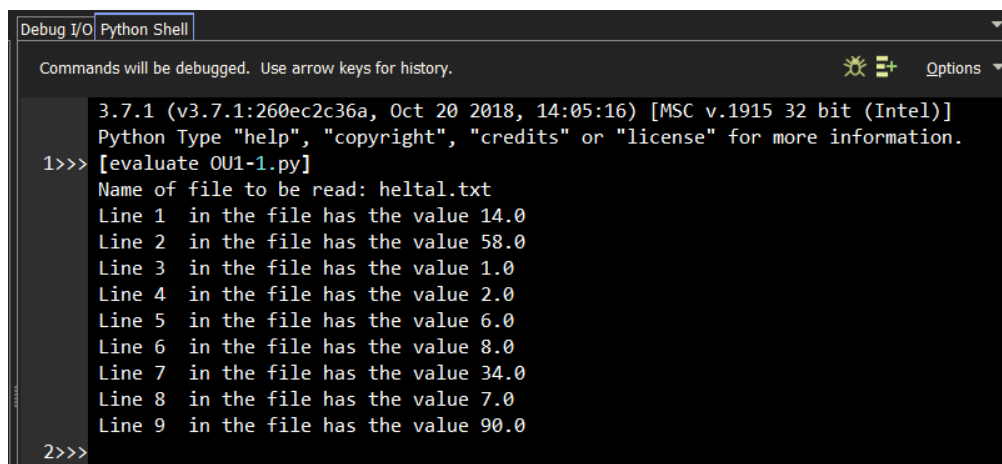
Testkörningen kan ses i Figur 5 och programmet fungerar som förväntat

Fall 3: Köra programmet med fler än ett värde och med implementationen ListAsArray

När programmet körs och användaren anger en fil med fler än ett flyttal ska programmet skriva ut alla tal i filen i rätt ordning.

Det här blir resultatet av att köra programmet med en fil som heter heltal.txt och innehåller 9 värden i denna ordning:

1. 14
2. 58
3. 1
4. 2
5. 6
6. 8
7. 34
8. 7
9. 90



```
Debug I/O Python Shell
Commands will be debugged. Use arrow keys for history.
3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)]
Python Type "help", "copyright", "credits" or "license" for more information.
1>>> [evaluate OUI-1.py]
Name of file to be read: heltal.txt
Line 1 in the file has the value 14.0
Line 2 in the file has the value 58.0
Line 3 in the file has the value 1.0
Line 4 in the file has the value 2.0
Line 5 in the file has the value 6.0
Line 6 in the file has the value 8.0
Line 7 in the file has the value 34.0
Line 8 in the file has the value 7.0
Line 9 in the file has the value 90.0
2>>>
```

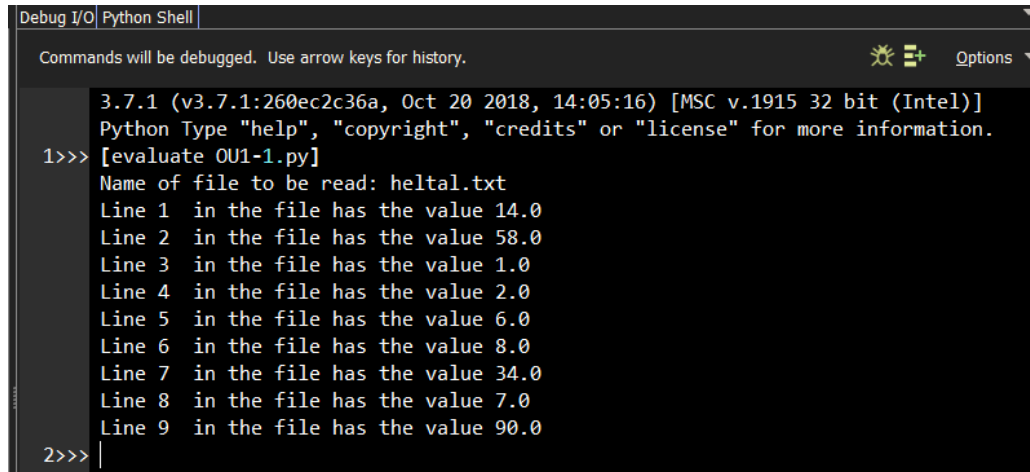
Figur 6 Testkörning av OUI-1.py där användaren ger en fil med 9 värden och använder implementationen ListAsArray

Testkörningen kan ses i Figur 6 och programmet fungerar som förväntat

Fall 4: Köra programmet med fler än ett värde och med implementationen ListAsTwoCell

Samma som föregående testkörning bara att implementationen ändras till ListAsTwoCell, Se **Fall 3** för mer info.

Utskriften blir:



```
Debug I/O Python Shell
Commands will be debugged. Use arrow keys for history.
3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)]
Python Type "help", "copyright", "credits" or "license" for more information.
1>>> [evaluate OUI-1.py]
Name of file to be read: heltal.txt
Line 1 in the file has the value 14.0
Line 2 in the file has the value 58.0
Line 3 in the file has the value 1.0
Line 4 in the file has the value 2.0
Line 5 in the file has the value 6.0
Line 6 in the file has the value 8.0
Line 7 in the file has the value 34.0
Line 8 in the file has the value 7.0
Line 9 in the file has the value 90.0
2>>> |
```

Figur 7 Testkörning av OUI-1.py där användaren ger en fil med 9 värden och använder implementationen ListAsTwoCell

Testkörningen kan ses i Figur 7 och programmet fungerade som förväntat även denna gång. Jag drar därför slutsatsen att mitt program fungerar som det ska med båda implementationerna av Lista.

Testkörningar Del 2

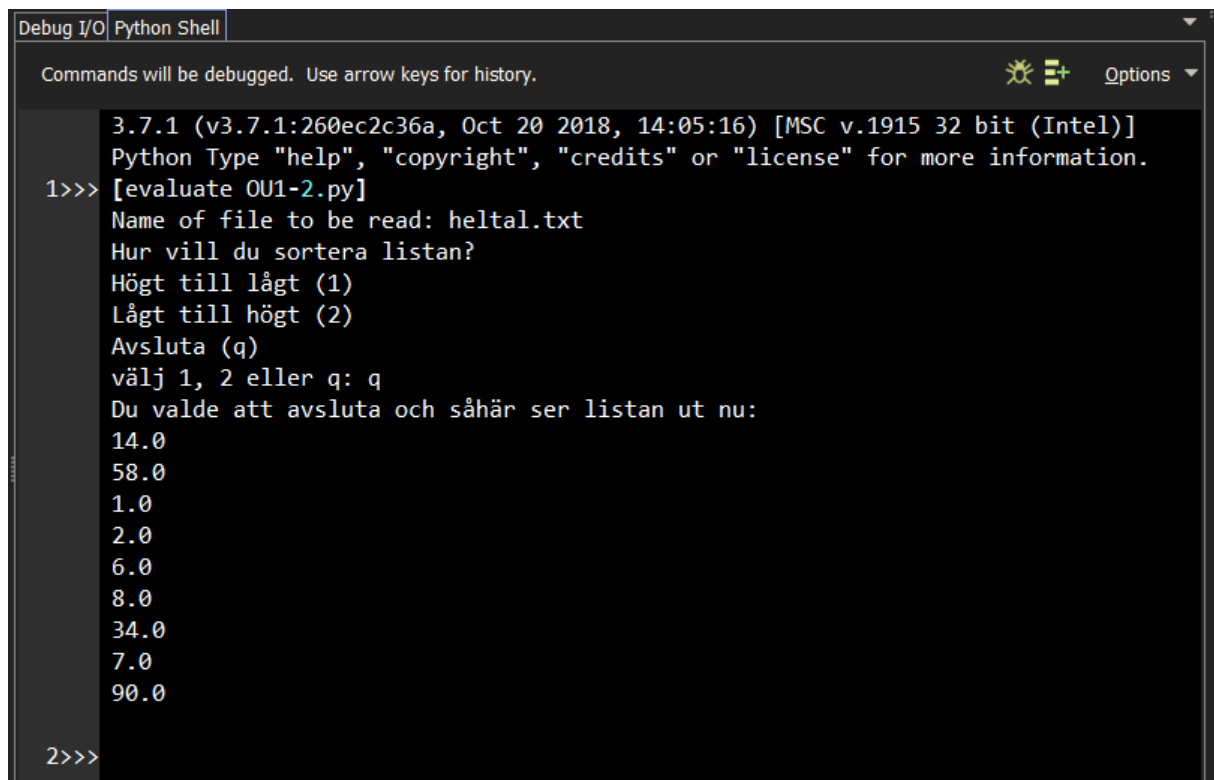
Jag har valt att testa två fall som beskrivs närmre nedan.

Fall 1: Köra programmet utan att sortera listan

När användaren gett en fil med flyttal utan att sortera listan ska programmet returnera listan i rätt ordning och osorterad

Det här blir resultatet av att köra programmet med en fil som heter heltal.txt och innehåller 9 värden i denna ordning:

1. 14
2. 58
3. 1
4. 2
5. 6
6. 8
7. 34
8. 7
9. 90



```
Debug I/O Python Shell
Commands will be debugged. Use arrow keys for history.
3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)]
Python Type "help", "copyright", "credits" or "license" for more information.
1>>> [evaluate OU1-2.py]
Name of file to be read: heltal.txt
Hur vill du sortera listan?
Högt till lågt (1)
Lågt till högt (2)
Avsluta (q)
välj 1, 2 eller q: q
Du valde att avsluta och såhär ser listan ut nu:
14.0
58.0
1.0
2.0
6.0
8.0
34.0
7.0
90.0
2>>>
```

Figur 8 Testkörning av OU2-2.py där användaren ger en fil med 9 värden och valde att avsluta programmet utan att sortera listan

Testkörningen kan ses i Figur 8 och programmet fungerar som förväntat.

Fall 2: Köra programmet och först ge en felaktig input, sedan sortera på ett sätt och till sist sortera på ett annat sätt

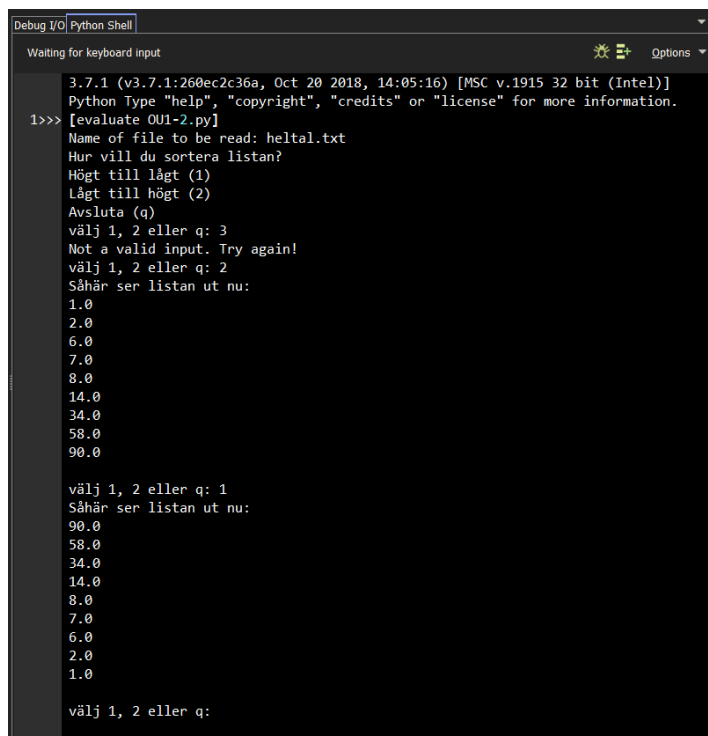
När användaren gett en fil med flyttal och först ger en felaktig input ska programmet säga att det blev fel fråga vad man vill göra igen, sedan ska man kunna sortera på två olika sätt och till sist ska listan skrivas ut i den ordningen man sist sorterade den i.

Det här blir resultatet av att köra programmet med en fil som heter heltal.txt och innehåller 9 värden i denna ordning:

1. 14
2. 58
3. 1
4. 2
5. 6
6. 8
7. 34
8. 7
9. 90

Samt att ge inputsen i denna ordning:

1. '3' (felaktig input)
2. '2' (sortera från lågt till högt)
3. '1' (sortera från högt till lågt)
4. 'q' (avsluta)



```
3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)]
Python Type "help", "copyright", "credits" or "license" for more information.
1>>> [evaluate OU1-2.py]
Name of file to be read: heltal.txt
Hur vill du sortera listan?
Högt till lågt (1)
Lågt till högt (2)
Avsluta (q)
välj 1, 2 eller q: 3
Not a valid input. Try again!
välj 1, 2 eller q: 2
Såhär ser listan ut nu:
1.0
2.0
6.0
7.0
8.0
14.0
34.0
58.0
90.0

välj 1, 2 eller q: 1
Såhär ser listan ut nu:
90.0
58.0
34.0
14.0
8.0
7.0
6.0
2.0
1.0

välj 1, 2 eller q:
```

Figur 9 Testkörning av OU2-2.py där användaren ger en fil med 9 värden gav först ett felaktigt input följt av två sorteringar innan programmet avslutades.

Testkörningen kan ses i Figur 9 och programmet fungerar som förväntat. Jag drar därför slutsatsen att programmet fungerar som det ska.

Referenser

1. Janlert, Lars-Erik; Wiberg och Torbjörn Wiberg. *Datatyper och algoritmer*. Uppl. 2:6. Malmö AB, Sverige 2011.
2. Kallin, Lena. *F2_cell_lista_stack_testning.pdf*. Föreläsning 5DV150 Datastrukturer och algoritmer (Python) 2021. Canvas
3. Kallin, Lena. *F5_LS_Sortering_Sokning.pdf*. Föreläsning 5DV150 Datastrukturer och algoritmer (Python) 2021. Canvas
4. *Wikipedia*. Merge sort. 2021.
https://en.wikipedia.org/w/index.php?title=Merge_sort&oldid=1042060512 (Hämtad 2021-09-21)