

Programmering og sproganalyse

Skrevet af: Jonas Reventlow Petersen og David Jonas Nitze

Antal anslag: 10.921

Indledning

Denne rapport er skrevet ud fra ni opgaver stillet af underviser. Disse ni spørgsmål omhandler ni forskellige problemstillinger angående programmering. For at udfører disse opgave, har vi valgt at skrive i programmeringssproget Python3. Kodningerne skriver vi i platformen Jupyter Notebook, og filerne vil dermed være .ipynb filer.

Vores primære viden af besvarelserne af opgaverne er udformet af bogen:

Gries, Paul., Campbell, J. og Montoyo, J. 2017. *Practical Programming. An introduction to Computer Science Using Python 3.6*. T. Coron (editor). 3. Edition.

Vores primære kodning er især baseret på kapitel 3, 7, 8, 9 og 10 i bogen. Disse kapitler giver en god forståelse af, hvordan functions, methods, loops, lists og indlæsning af filer fungerer, samt naturligvis den opstøvede viden fra forelæsnings- og øvelsesregi.

For så vidt muligt, at få skabt de bedste og mest enkle programmer, har vi også udnyttet af os den faglighed der ligger på internettet. Disse hjemmesider har hjulpet med viden til for eksempel en oversigt over metoder, som har bidraget til vores programmering.

Hjemmesiderne vil blive refereret til som parentes i denne rapport.

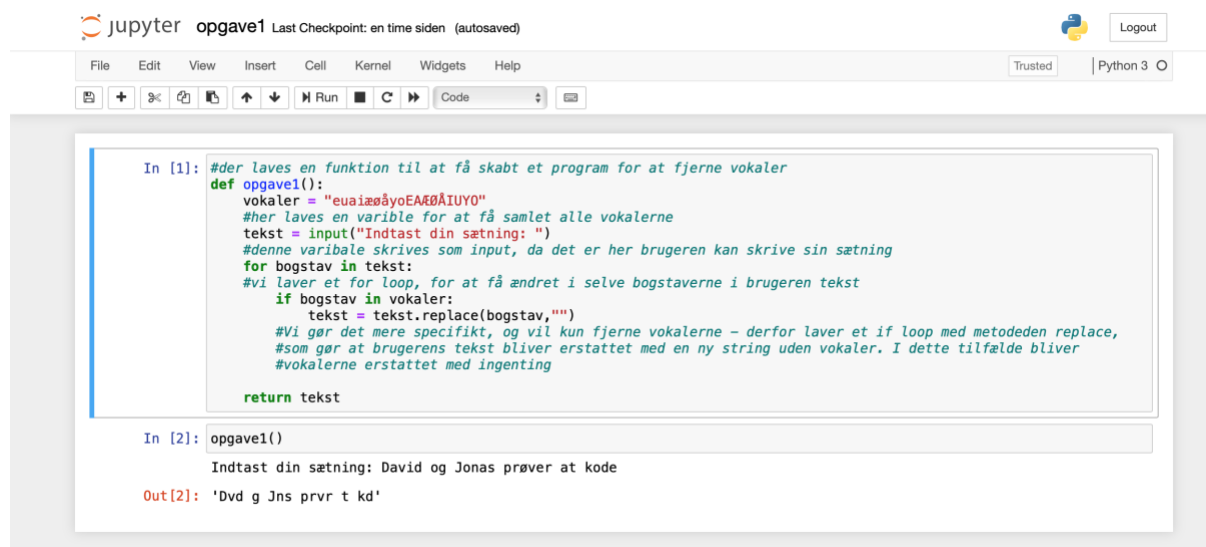
Denne rapport vil forklare processen af hver opgave. Hver opgave vil få sit eget afsnit, hvor vi vil forklare vores kodningssprog (python) med danske ord (alfabetet). For at give den bedste forklaring, vil der også blive refereret til vores bilag, som består af momentane skærbilleder, der viser hver kodnings ageren. Derudover er der skrevet kommentar i selve kodningerne, så man kan få en forklaring på, hvorfor vi har valgt de løsninger, vi gør brug af. Det er vigtigt at pointere, at opgaverne kan løses på mange forskellige måder. Vi har løst dem med de metoder, som vi har forstået bedst gennem kursets undervisning. Vi henviser også til alternative løsningsmodeller, når det har været relevant gennem udarbejdelsen af opgaverne.

Opgave1

Vi bliver i denne opgave bedt om at fjerne vokaler fra en brugers sætning. Vi har valgt at definere en funktion for at udføre opgaven. Til funktionen har vi et input - brugerens valgfrie sætning - samt defineret alle vokaler: store og små. På den måde bevarer vi sætningens helhed uden at ændre på hvorvidt en vokal skrives som versal eller ej, som ellers ville kunne være gjort med `.lower` eller `.upper`, hvis dette havde interesse. Dog ville sætningens udtryk ændre karakter, og derfor valgte vi muligheden for at bevare sætningens oprindelige konstruktion - med udeladelse af vokaler.

En anden metode, der kunne være brugt, kunne være RegEx, som kan bruges til at finde mønstre i strings (https://www.w3schools.com/python/python_regex.asp). Vores mønster er i denne opgave vokaler, og de kunne erstattes med en `reg.sub` kommando, hvor de ville blive erstattet af ingenting, da de blot skal fjernes fra brugerens input (sætning).

Bilag 1



```
In [1]: #der laves en funktion til at få skabt et program for at fjerne vokaler
def opgave1():
    vokaler = "euaieåyoEÅøŒIUYO"
    #her laves en variable for at få samlet alle vokalerne
    tekst = input("Indtast din sætning: ")
    #denne variabale skrives som input, da det er her brugeren kan skrive sin sætning
    for bogstav in tekst:
        #vi laver et for loop, for at få ændret i selve bogstaverne i brugeren tekst
        if bogstav in vokaler:
            tekst = tekst.replace(bogstav, "")
        #Vi gør det mere specifikt, og vil kun fjerne vokalerne - derfor laver et if loop med metodeden replace,
        #som gør at brugerens tekst bliver erstattet med en ny string uden vokaler. I dette tilfælde bliver
        #vokalerne erstattet med ingenting

    return tekst

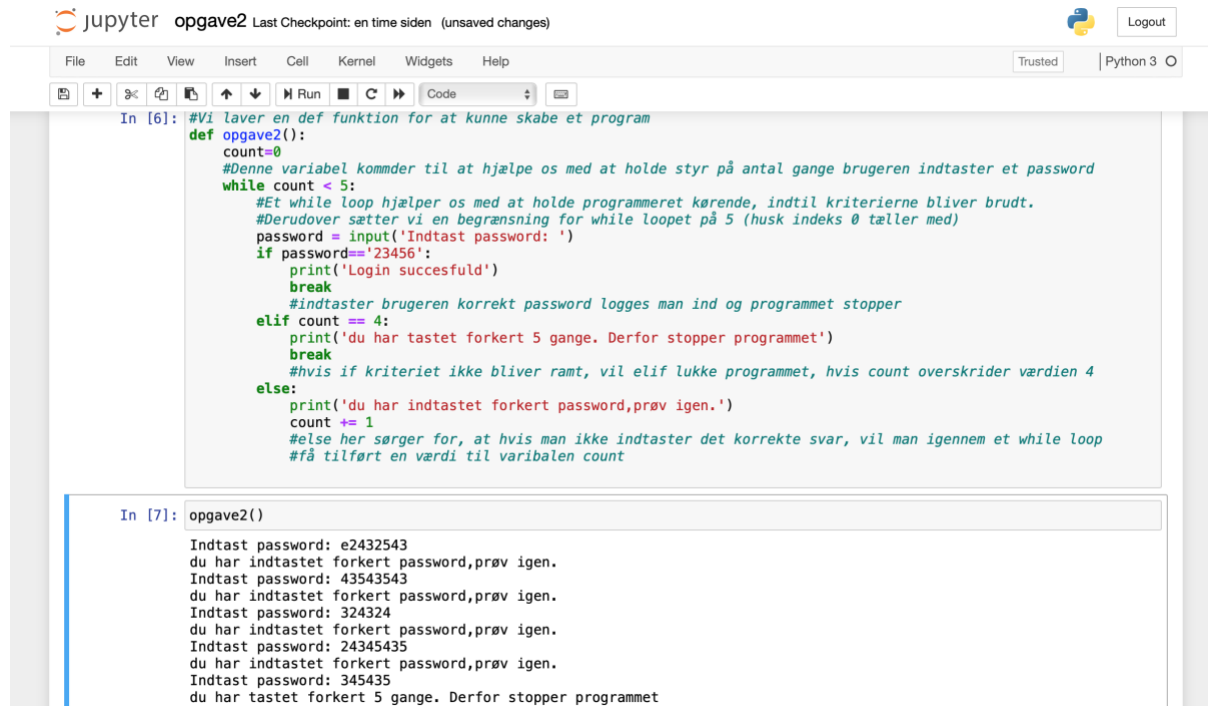
In [2]: opgave1()
Indtast din sætning: David og Jonas prøver at kode
Out[2]: 'Dvd g Jns prvr t kd'
```

Opgave2

Vi bliver i denne opgave bedt om at standse et program efter korrekt indtastet kodeord eller fem forkerte indtastninger. Vi har valgt at bruge et while loop for at udføre opgaven. Et while loop virker indtil en præmis bliver opfyldt - her enten af et korrekt udfyldt kodeord eller et ukorrekt kodeord gættet fem gange. Det ses også i kodningen: vi har indtastet præmissen om korrekt kodeord (`==23456`), og hvad der sker hvis denne præmis opfyldes (adgang). Indtastes

der ukorrekt kodeord (!=23456), tæller while loopet op imod de fem gange, som vi har sat som præmisse for antal forsøg af kodeordets indtastning.

Bilag 2



The screenshot shows a Jupyter Notebook window titled 'opgave2' with the subtitle 'Last Checkpoint: en time siden (unsaved changes)'. The interface includes a top menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The main area contains two input cells. The first cell, labeled 'In [6]:', contains a Python function definition for 'opgave2'. The function initializes a 'count' variable to 0 and enters a 'while count < 5' loop. Inside the loop, it prompts the user for a password. If the password is '23456', it prints 'Login succesfuld' and breaks. If the password is incorrect, it prints a message and increments the count. If the count reaches 5, it prints a message and breaks. Otherwise, it prints a message and increments the count. The second cell, labeled 'In [7]:', calls the 'opgave2' function. The output shows the function being called, followed by five password prompts and corresponding messages, ending with a message indicating that the program has stopped after 5 failed attempts.

```
In [6]: #Vi laver en def funktion for at kunne skabe et program
def opgave2():
    count=0
    #Denne variabel kommer til at hjælpe os med at holde styr på antal gange brugeren indtaster et password
    while count < 5:
        #Et while loop hjælper os med at holde programmeret kørende, indtil kriterierne bliver brudt.
        #Derudover sætter vi en begrænsning for while loopet på 5 (husk indeks 0 tæller med)
        password = input('Indtast password: ')
        if password=='23456':
            print('Login succesfuld')
            break
        #Indtaster brugeren korrekt password logges man ind og programmet stopper
    elif count == 4:
        print('du har tastet forkert 5 gange. Derfor stopper programmet')
        break
    #hvis if kriteriet ikke bliver ramt, vil elif lukke programmet, hvis count overskrider værdien 4
    else:
        print('du har indtastet forkert password,prøv igen.')
        count += 1
        #else her sørger for, at hvis man ikke indtaster det korrekte svar, vil man igennem et while loop
        #få tilført en værdi til varibalen count

In [7]: opgave2()

Indtast password: e2432543
du har indtastet forkert password,prøv igen.
Indtast password: 43543543
du har indtastet forkert password,prøv igen.
Indtast password: 324324
du har indtastet forkert password,prøv igen.
Indtast password: 24345435
du har indtastet forkert password,prøv igen.
Indtast password: 345435
du har tastet forkert 5 gange. Derfor stopper programmet
```

Opgave3

Vi bliver i denne opgave bedt om at fjerne dubletter af navne fra to lister ved brug af en funktion. Denne opgave blev løst en def function. I funktionens argument sætter man to værdier ind som hedder list1 og list2, som i sidste ende ville være inputs af listerne.

I funktionen er det vigtigt først at få defineret i programmet, at det kun er lister der må puttes ind i argumentet. For at give programmet denne information, kan man lave et if statement. Inde i if statementet skriver vi at hvis listetypen for list1 og list2 ikke er præcist en liste, så fungerer programmet ikke. For at være flinke overfor brugere, har vi også valgt at brugeren får informationen om, at hendes/hans indtastninger ikke er en liste.

Vi samler de to lister, der bliver skrevet i funktionens argument. Denne tredje liste (samlet_liste) laver vi som en variabel, hvor dens værdi er at man har plusset list1 og list2 med hinanden.

For at få fjernet dubletter fandt vi en hjemmeside med løsningen

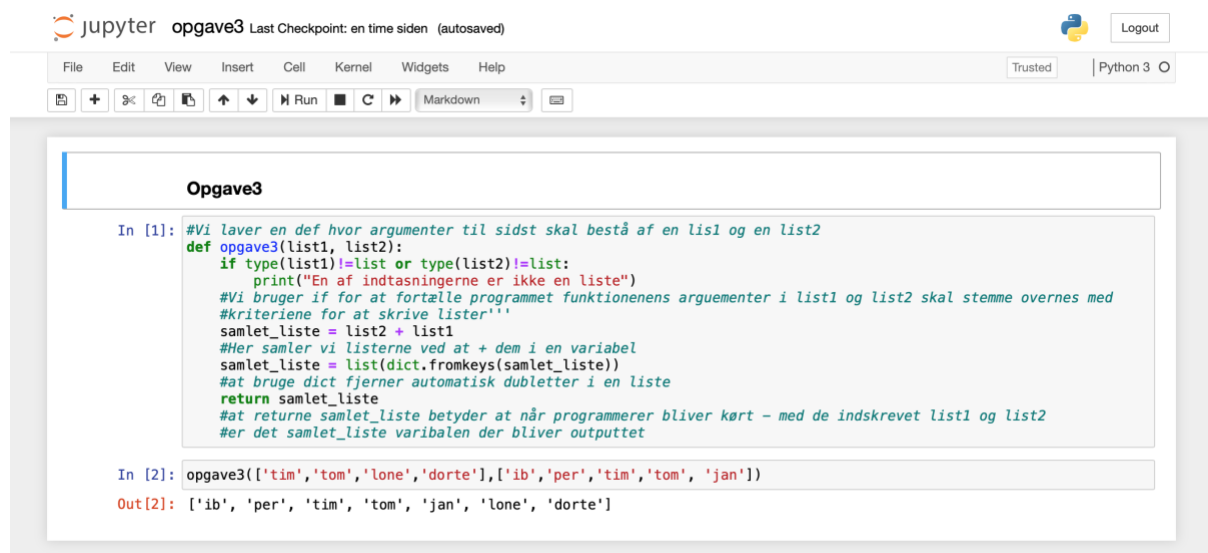
(https://www.w3schools.com/python/python_howto_remove_duplicates.asp). Her står der ordret:

“Create a dictionary, using the List items as keys. This will automatically remove any duplicates because dictionaries cannot have duplicate keys.”

Da det ikke kan lade sig gøre for en liste at have dubletter når man laver en dictionary inde i listen, vil dubletter automatisk blive fjernet.

Til sidst er det den samlede liste vi godt vil have som output. Derfor vælger vi at returnerer samlet_liste. Alt programmeringen foregår også inde i selve programmet, som betyder at kodningerne kun har betydning, hvis listerne kommer ind i argumentet i programmet.

Bilag 3



```
Opgave3

In [1]: #Vi laver en def hvor argumenter til sidst skal bestå af en list1 og en list2
def opgave3(list1, list2):
    if type(list1)!=list or type(list2)!=list:
        print("En af indtastningerne er ikke en liste")
    #Vi bruger if for at fortælle programmet funktionens argumenter i list1 og list2 skal stemme overens med
    #kriteriene for at skrive lister'''
    samlet_liste = list2 + list1
    #Her samler vi listerne ved at + dem i en variabel
    samlet_liste = list(dict.fromkeys(samlet_liste))
    #at bruge dict fjerner automatisk dubletter i en liste
    return samlet_liste
    #at returne samlet_liste betyder at når programmerer bliver kørt - med de indskrevet list1 og list2
    #er det samlet_liste variablen der bliver outputtet

In [2]: opgave3(['tim', 'tom', 'lone', 'dorte'], ['ib', 'per', 'tim', 'tom', 'jan'])
Out[2]: ['ib', 'per', 'tim', 'tom', 'jan', 'lone', 'dorte']
```

Opgave4

I denne opgave, skal vi have lavet et program der tæller antal ord med kriterierne af at ordet består af mere end tre bogstaver og hvor det første og sidste bogstav er det samme. Det betyder vi reelt set har tre kriterier, da vi også får at vide at det specifikt er ord vi skal arbejde med, og ikke tal eller tegn.

Vi har løst det igennem en def function. Vi vælger igen her at få et argument ind i vores def function, da det giver mulighed for at skrive den liste man nu vil, når vi skal bruge programmet til vores output.

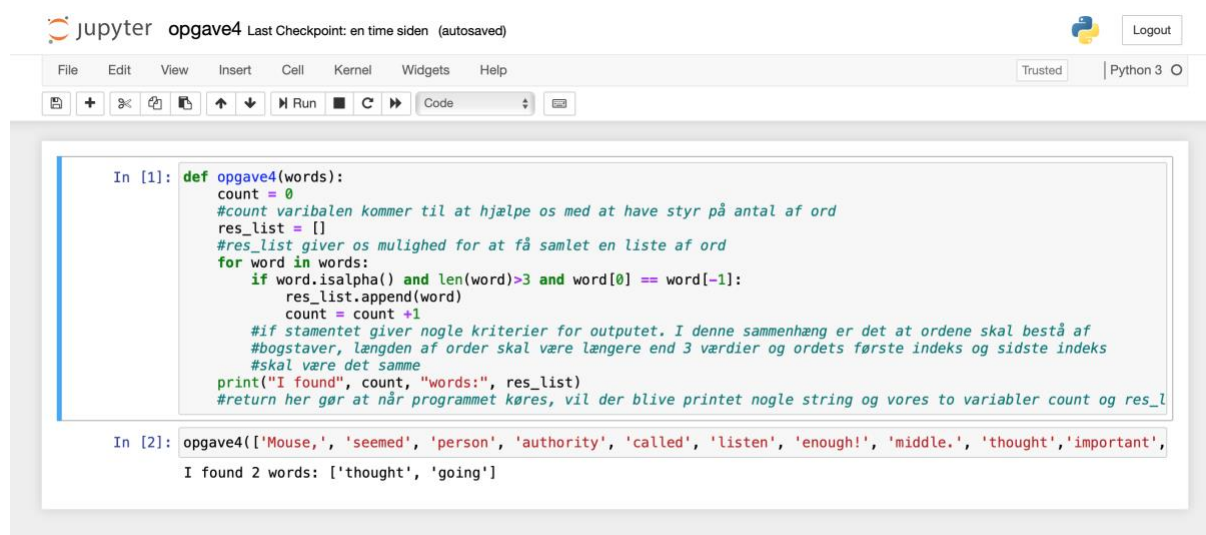
Opgaven vil gerne have, at programmet skal udskrive både antal ord som indgår i kriterierne, men også hvilke ord der arbejdes med. Disse to kriterier vil vi gerne have hver for sig, og derfor laver vi en variabel med værdi 0, og en variabel med en tom liste.

Herefter laves et for loop, for at få arbejdet specifikt med hvert ord, der skal undersøges om de er i overensstemmelse med 'if'-kriterierne.

Da det er ord, vi leder efter, kan man bruge metoden `.isalpha()`. Længden på orden skal være længere end tre bogstaver. Her har vi valgt den built-in function der hedder 'len'(gh), som arbejder med længden af en værdi. Til sidst indikerer vi, at det første indeks i hvert ord [0] skal være det samme som (==) det sidste indeks i ordet [-1]. Python's første værdi starter altid med 0 (ligesom i for eksempel Java), og -1 indikerer sidste indeks i variabelen.

Hvis et ord opfylder 'if'-kriterier, vil de gennem `.append` metoden blive tilføjet til vores tomme liste, og vores variabel count vil få tilføjet én værdi. Programmet slutes med at returnerer vores indskrevet print, hvor outputet bliver printet som vist i bilaget.

Bilag 4



```
In [1]: def opgave4(words):
count = 0
#count variabelen kommer til at hjælpe os med at have styr på antal af ord
res_list = []
#res_list giver os mulighed for at få samlet en liste af ord
for word in words:
    if word.isalpha() and len(word)>3 and word[0] == word[-1]:
        res_list.append(word)
        count = count + 1
    #if stamentet giver nogle kriterier for outputet. I denne sammenhæng er det at ordene skal bestå af
    #bogstaver, længden af ordet skal være længere end 3 værdier og ordets første indeks og sidste indeks
    #skal være det samme
print("I found", count, "words:", res_list)
#return her gør at når programmet køres, vil der blive printet nogle string og vores to variabler count og res_l

In [2]: opgave4(['Mouse', 'seemed', 'person', 'authority', 'called', 'listen', 'enough!', 'middle.', 'thought', 'important',
I found 2 words: ['thought', 'going']
```

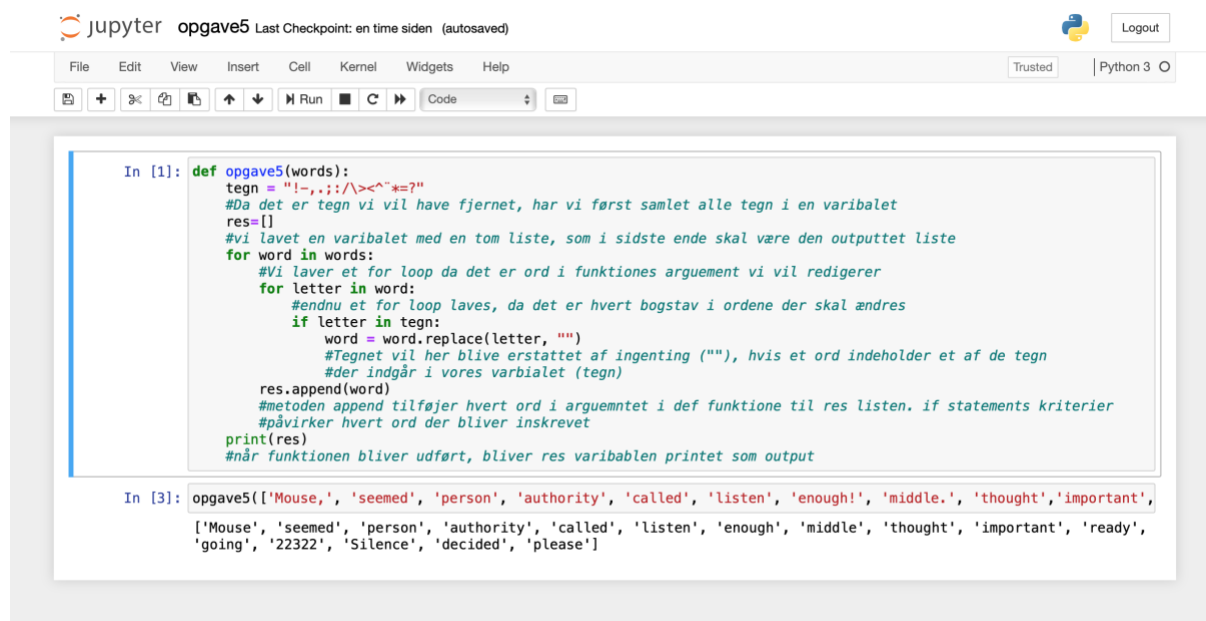
Opgave5

I dette program skal vi have fjernet tegnsætninger fra et input. Inputtet i dette program er en liste. Vores program skal kunne løse, at tegnsætninger skal fjernes fra listens indhold.

Dette løses ved at kode i en def function. Vi putter words ind som argumentet, da de i sidste ende skal være ordene som input i vores liste. For at få samlet alle tegn, lægger vi dem ind i en variabel. Dertil laver vi en ny liste som er tom, som i sidste ende bliver vores output.

Vi har valgt at bruge et for loop for at kunne revidere de specifikke tegn i hvert ord. Det første for loop gør at vi har et fokus på ordene i listen. Det næste loop gør at vi skaber et fokus på bogstaver i hvert ord. Igennem et 'if' statement, kan vi få skabt en kontakt mellem tegn og bogstav. Inde i 'if' statementet bruger vi .replace metoden, som gør, at hvis et ord består af nogle af de tegn oppe i vores variabel, så vil dette specifikke tegn bliver erstattet med ingenting (""). Ordet - som nu er ændret gennem .replace metoden - bliver puttet ind i vores tomme liste, og det er denne liste vi returnerer til sidst. Forloopet tager fat i funktionens argument (words), og går dermed igennem alle ord der bliver skrevet i listen - uanset om de har tegn i sig eller ej.

Bilag 5



The screenshot shows a Jupyter Notebook interface with the following content:

```
In [1]: def opgave5(words):
    tegn = "!,.,;:/\><^*=?"
    #Da det er tegn vi vil have fjernet, har vi først samlet alle tegn i en varibale
    res=[]
    #vi lavet en varibale med en tom liste, som i sidste ende skal være den outputtet liste
    for word in words:
        #Vi laver et for loop da det er ord i funktionens argument vi vil redigerer
        for letter in word:
            #endnu et for loop laves, da det er hvert bogstav i ordene der skal ændres
            if letter in tegn:
                word = word.replace(letter, "")
                #Tegnet vil her blive erstattet af ingenting (""), hvis et ord indeholder et af de tegn
                #der indgår i vores varibale (tegn)
            res.append(word)
            #metoden append tilføjer hvert ord i arguemntet i def funktionen til res listen. if statements kriterier
            #påvirker hvert ord der bliver inskrevet
    print(res)
    #når funktionen bliver udført, bliver res variblen printet som output
```

```
In [3]: opgave5(['Mouse', 'seemed', 'person', 'authority', 'called', 'listen', 'enough!', 'middle.', 'thought', 'important',
    ['Mouse', 'seemed', 'person', 'authority', 'called', 'listen', 'enough', 'middle', 'thought', 'important', 'ready',
    'going', '22322', 'Silence', 'decided', 'please']
```

Opgave6

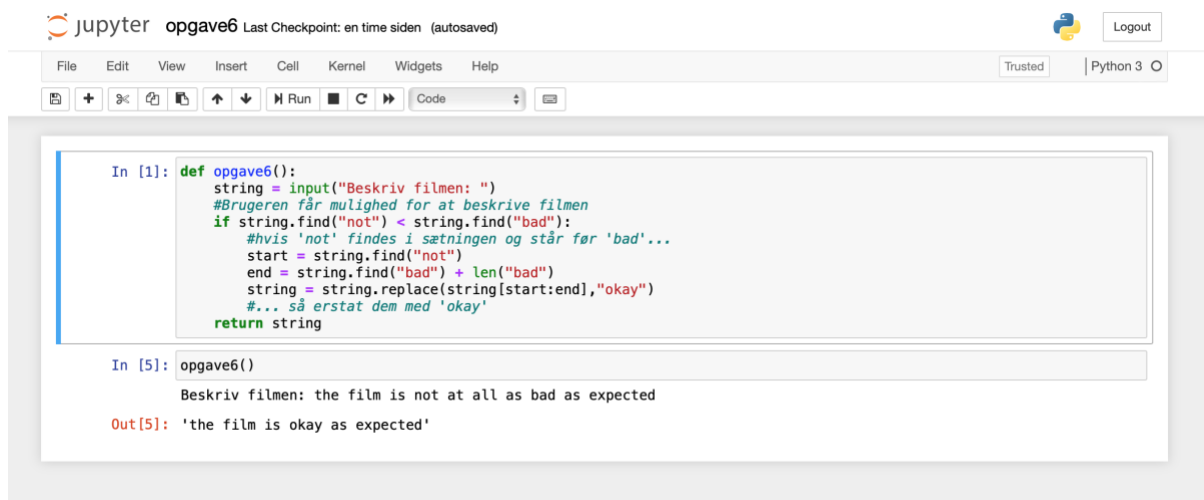
I denne opgave skal vi lave en funktion, der tager en string som input og returnerer en ny string. Indeholder stringen "not ... bad", skal dette udskiftes med "okay". Findes de i omvendt rækkefølge, eller kun det ene af ordene, ændres stringen ikke.

Vi har valgt at bruge 'if', og betingelsen er hvis "not" kommer før "bad", så skal det i inputtets string ændres til "okay" i stedet. For at sikre, at sætningerne kun ændres hvis "not"

kommer før “bad” indsætter vi < for at vise, at “not” nødvendigvis skal komme efter “bad” hvis ‘if’ betingelsen skal opfyldes. Dvs. index for “not” kommer før index for “bad”.

I ‘end’ sikrer vi os at hele ordet “bad” også kommer med, ved at bruge ‘len’(ght) funktionen, og bliver erstattet når stringen returneres.

Bilag 6



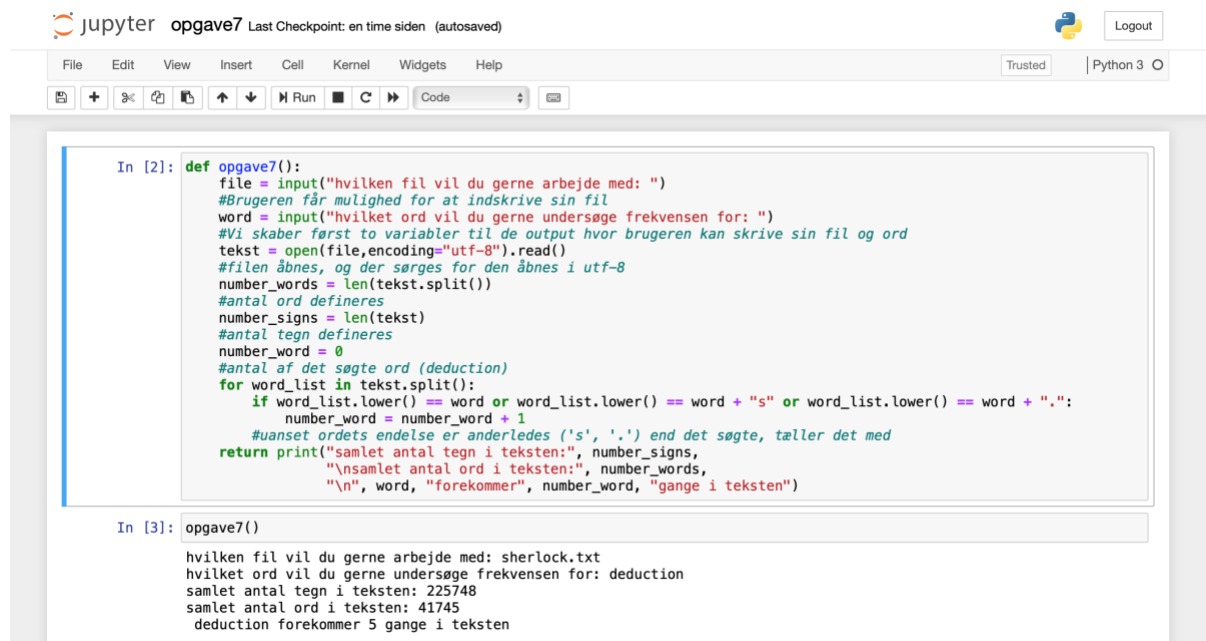
```
jupyter opgave6 Last Checkpoint: en time siden (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [1]: def opgave6():
        string = input("Beskriv filmen: ")
        #Brugeren får mulighed for at beskrive filmen
        if string.find("not") < string.find("bad"):
            #hvis 'not' findes i sætningen og står før 'bad'...
            start = string.find("not")
            end = string.find("bad") + len("bad")
            string = string.replace(string[start:end], "okay")
            #... Så erstat dem med 'okay'
        return string

In [5]: opgave6()
Beskriv filmen: the film is not at all as bad as expected
Out[5]: 'the film is okay as expected'
```

Opgave7

I denne opgave skal vi arbejde med en tekstfil, hvor vi skal tælle antal ord, anslag mm. Vi har valgt at gøre brug af brugerinputs, hvor brugeren selv kan skrive navnet på den fil, der skal undersøges (skal dog være i samme sti som programmet, og skal være en .txt fil, dvs. i denne opgave er det nødvendigvis “sherlock.txt”). Samme gør sig gældende for det undersøgte valgfrie ord, der her skal være “deduction”. Tekstfilen (sherlock.txt) splitter vi ind i ord for derefter at finde antal anslag. Vi sørger for at gøre tekstens bogstaver små, så de er ens. Det er relevant for søgningen af ord, der er sensitiv for versaler. Søgningen af det valgfrie ord “deduction” er gjort således, at ordet også tager hensyn til, hvis det har en anden endelse, dvs. her flertals-‘s’ eller ‘.’. Søger man blot efter “deduction”, får man tre resultater - men vi lavede en hurtig i notepad med ctrl+F af sherlock.txt, som viste at der er ét resultat med “deductions” og ét med “deduction.”. Dem sørger vi for at få med. Skulle andre tegn med (fx: ?;!*’) kan man tilføje dem på samme måde. Det ville være relevant, hvis man ikke kendte det valgfrit efterspurgt ord.

Bilag 7



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. The notebook is titled 'opgave7' and shows 'Last Checkpoint: en time siden (autosaved)'. The code cell contains a function 'opgave7()' that prompts the user for a file name and a word, reads the file, and counts the occurrences of the word. The output cell shows the results of running the function with 'sherlock.txt' and 'deduction'.

```
In [2]: def opgave7():
        file = input("hvilken fil vil du gerne arbejde med: ")
        #Brugeren får mulighed for at indskrive sin fil
        word = input("hvilket ord vil du gerne undersøge frekvensen for: ")
        #Vi skaber først to variable til de output hvor brugeren kan skrive sin fil og ord
        tekst = open(file,encoding="utf-8").read()
        #filen åbnes, og der sørges for den åbnes i utf-8
        number_words = len(tekst.split())
        #antal ord defineres
        number_signs = len(tekst)
        #antal tegn defineres
        number_word = 0
        #antal af det søgte ord (deduction)
        for word_list in tekst.split():
            if word_list.lower() == word or word_list.lower() == word + "s" or word_list.lower() == word + ".":
                number_word = number_word + 1
            #uanset ordets endelse er anderledes ('s', '.') end det søgte, tæller det med
        return print("samlet antal tegn i teksten:", number_signs,
                    "\nsamlet antal ord i teksten:", number_words,
                    "\n", word, "forekommer", number_word, "gange i teksten")

In [3]: opgave7()

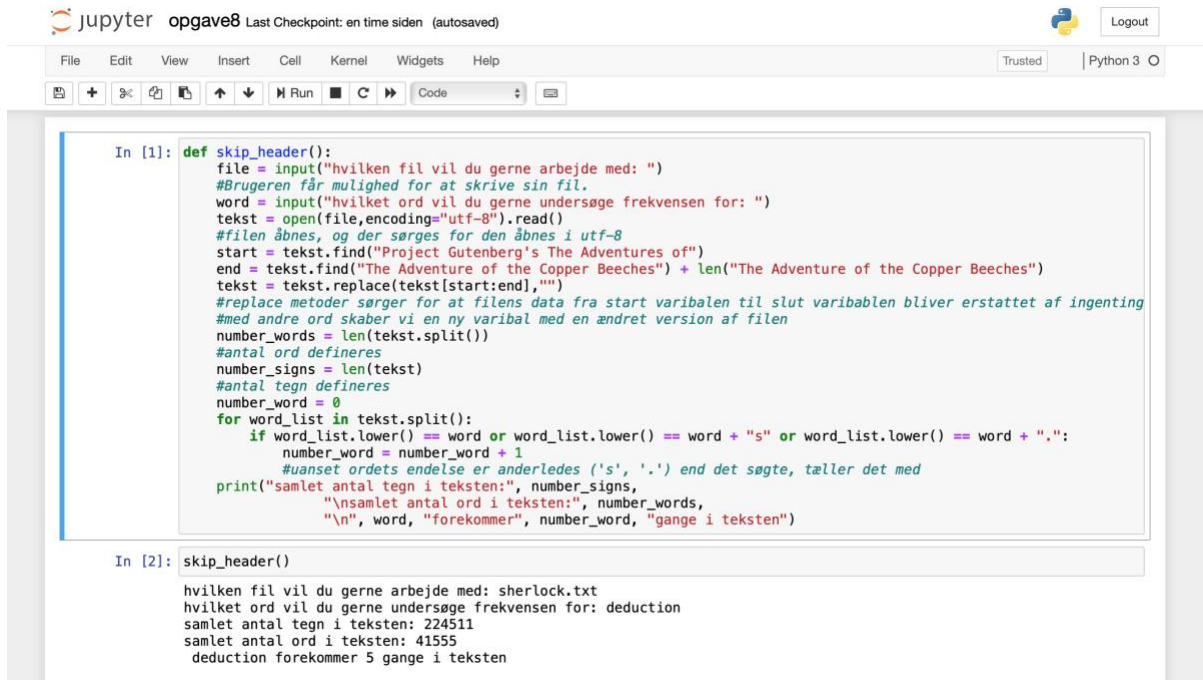
hvilken fil vil du gerne arbejde med: sherlock.txt
hvilket ord vil du gerne undersøge frekvensen for: deduction
samlet antal tegn i teksten: 225748
samlet antal ord i teksten: 41745
deduction forekommer 5 gange i teksten
```

Opgave8

I denne opgave skal vi arbejde videre på samme program som i opgave7, dog med den ændring, at vi gerne vil være foruden overskrift, indholdsfortegnelse mm., og vil starte fra selve historiens (sherlock.txt) begyndelse i stedet.

Vi forsøgte os først ad med metoden 'skip the header', som kan læses i onlineudgaven af bogens kapitel 10.6, som er relevant for netop denne opgave. Dog fangede vi ikke helt præcist hvordan den skulle bruges - for eftersom det var en .endswith kommando, ville den vel blot give et true/false, ræsonnerede vi. Vi tænkte lidt ekstra over den, og endte ud med en metode, der var meget sammenlignelig med en, vi havde brugt før - nemlig opgave6. Vi indsatte hvor den skulle starte (første anslag i filen) og hvor den skulle slutte (sidste anslag i indholdsfortegnelsen), som alt sammen blev erstattet af "", dvs. ingenting. Vi kan dog se, at der mangler at blive fjernet ét ord af syv anslag - hmm. Hvilket ord, hvordan eller hvorfor ved vi ikke.

Bilag 8



The screenshot shows a Jupyter Notebook interface. At the top, the header includes the Jupyter logo, the text "opgave8", and "Last Checkpoint: en time siden (autosaved)". On the right, there is a "Logout" button. Below the header is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A toolbar with various icons for file operations and execution is located below the menu bar. The main area contains two code cells. The first cell, labeled "In [1]:", contains a Python function definition for "skip_header()". The function prompts the user for a file name, reads the file, finds the start of "Project Gutenberg's The Adventures of the Copper Beeches", and replaces the header with an empty string. It then counts the number of words and signs in the text and prints the results. The second cell, labeled "In [2]:", shows the function being called, which produces the following output:

```
In [1]: def skip_header():
        file = input("hvilken fil vil du gerne arbejde med: ")
        #Brugeren får mulighed for at skrive sin fil.
        word = input("hvilket ord vil du gerne undersøge frekvensen for: ")
        tekst = open(file,encoding="utf-8").read()
        #filen åbnes, og der sørges for den åbnes i utf-8
        start = tekst.find("Project Gutenberg's The Adventures of")
        end = tekst.find("The Adventure of the Copper Beeches") + len("The Adventure of the Copper Beeches")
        tekst = tekst.replace(tekst[start:end],"")
        #replace metoder sørger for at filens data fra start variabelen til slut variablen bliver erstattet af ingenting
        #med andre ord skaber vi en ny variabel med en ændret version af filen
        number_words = len(tekst.split())
        #antal ord defineres
        number_signs = len(tekst)
        #antal tegn defineres
        number_word = 0
        for word_list in tekst.split():
            if word_list.lower() == word or word_list.lower() == word + "s" or word_list.lower() == word + ".":
                number_word = number_word + 1
            #uanset ordets endelse er anderledes ('s', '.') end det søgte, tæller det med
        print("samlet antal tegn i teksten:", number_signs,
              "\nsamlet antal ord i teksten:", number_words,
              "\n", word, "forekommer", number_word, "gange i teksten")

In [2]: skip_header()

hvilken fil vil du gerne arbejde med: sherlock.txt
hvilket ord vil du gerne undersøge frekvensen for: deduction
samlet antal tegn i teksten: 224511
samlet antal ord i teksten: 41555
deduction forekommer 5 gange i teksten
```