Föreläsning 5

Strängar, lexikon, filer och några centrala begrepp

Vi spelar in de olika avsnitten separat och lägger länkar i Studentportalen. Tom läser chatten.

Föreläsning 5 handlar om:

1. Hantera text i Python, bild 3-11
2. Lexikon (dictionary), bild 12-15
3. Samlingar, itererbara, ordnade, oordnade, ändringsbara, inte ändringsbara… bild 16-21
4. Filhantering bild 23-29
5. Och lite extramaterial t ex om referenser till objekt och reguljära uttryck

Vi spelar in de olika avsnitten separat (utom 5) och lägger länkar i Studentportalen.

Hantera text i Python

- ”Python” och ’Python’ är text-strängar och hanteras likadant av Python

print(”Text inom **citattecken** kan innehålla **’apostrof’** och tvärt om”)

FEL: print(”Men inte ”så här”. Det blir fel”)

print(”””” Med s k "**triple-quoted strings**" kan en text

sträcka sig över flera rader””””) #skrivs ut på 2 rader

- print(**f'**Prefixet f, f-String, formaterar utskriften: {3.1415927 :.3f}\n') #ger 3 decimaler

Se minilektionen Formatering!

- **Escape sekvenser**, exempel (det finns fler)
  - \n (newline)
  - \t (horisontal tab)

print('\t ger en tab, medan \n ger ny rad \a\n’) # \a, bell

print(“Använd end=' ' om du *inte* vill få ny rad”, **end=''**)

print(x, x*2, x+3, x**4, **sep='; '**) # Eller sep=', ' eller sep=’\n’, eller sep=’ och sen ’

Hantera text i Python

List, tuple, range, str är *sekvenser* som man kan göra operationer på:

Exempel:

```
namn = ’Svea Lundmark’
print(type(namn))  #<class 'str'>: variabeln namn får typen str
```

Funktioner (t ex vissa inbyggda) på strängar, och

Metoder (sträng-klassen) på strängar

Variabeln som ***parameter*** vs variabeln som ***anropande objekt*** (punktnotation).

Hantera text i Python

namn = ’Svea Lundmark’ #variabeln *namn* får typen **str**

Funktioner på strängar

Variabeln som ***parameter,*** t ex vissa inbyggda funktioner kan användas, se https://docs.python.org/3/library/functions.html:

```
print(’Det finns’, len(namn), ’tecken i strängen ”namn”.’) # 13 (varför inte 12?)
print(’Gör en lista av strängen ”Nils”: ’, list(’Nils’) # ['N', 'i', 'l', 's']
print(’Vi sorterar strängen: ’, sorted(namn) # Returnerar en sorterad lista.
print(’Efter sorteringen: ’, namn)  # Vad skrivs ut?
print(’Största tecknet:’, max(namn)) # v, min(namn) ger blanksteg
```

Hantera text i Python

namn = ’Svea Lundmark’ #variabeln *namn* får typen **str**

Metoder (i string-klassen) på strängar

Variabeln namn som ***anropande objekt*** (punktnotation).

Exempel klassen string. *Alla metoder i string-klassen returnerar värde men* ***ändrar inte sträng-objektet****:*

print(’Bokstaven a finns’, **namn**.**count**(’a’), ’gånger.’) # 2

print(’Bokstaven v finns på plats ’, **namn**.**index**(’v’)) # 1 Starta på noll

print(Gör alla bokstäver till gemener:’, **namn**.**casefold**())

print(’Är alla bokstäver versaler?’,**namn**.**isupper**()) # False. Kollar bara bokstäver

Sträng-metoder, se https://www.w3schools.com/python/python_ref_string.asp

Hantera text i Python

Några operatorer på strängar: in, +, *, >, <, ==

```
if ’a’ in namn: # operatorerna in och not in
    print(’Strängen innehåller minst ett a’)
namn += ’ –Granström’ # + operatorn: konkatenering (engelska: concatenate)
namn +=3 * ” :-) ” # * operatorn
print(namn)


not_letters = []
for b in namn:
    if not b.isalpha(): # Om b inte är bokstav
        not_letters.append(b)
print('Tecken som inte är bokstäver:’, not_letters) #
```

Hantera text i Python

Jämför textsträngar

Python jämför tecken för tecken tills strängarna är slut eller det skiljer:

```
print(’Anna’ == ’anna’) # False
print (’betar’ < ’het’) # True
print(’bus’ < ’busa’) # True
print('buss' < 'busar') # False
```

Hantera text i Python

Typomvandla av textsträngar

a=’124’ # En textsträng

min_lista = **list**(a) # ['1', '2', '4’] men *a är fortfarande en sträng*

x = **int**(a) # omvandlar strängen ***med tal*** till heltal, inte t ex ’xy34’

x = x/2 # Går nu att räkna med

print('Omvandla till float:',float(s)) # 124.0

print('Omvandla till tuple:',tuple(s)) # ('1', '2', '4')

…

Hantera text i Python

Teckenrepresentation

- Pythons string-typ använder Unicode Standard för att representera tecken , se https://www.unicode.org/ Det finns plats för c:a 1.1 miljoner tecken.
- Varje tecken i en text har en egen teckenkod: ‘A’, ‘%‘, mellanslag etc.

print(ord(’A’)) # 65 decimal representation

print(chr(65)) # A decimal representation

print('I feel \u26A1! I need \u26FE’) # I feel ⚡! I need ☾. Hexadecimal repr.

https://www.unicode.org/charts/

“UTF-8 is an encoding format for representing Unicode characters as binary data of one or more bytes per character.”

```
”Kryptering:”
s = 'AB’
t = ord(s[0])+1  # tecken -> ascii
s2 = chr(t)      # ascii -> tecken
```

Rast 5 minuter

```
# Fundera: vad händer här:
for i in range(26):
    print(chr(65+i), '->’ ,chr(97+i), ';’ ,end='')
print('\n’)
```

Lexikon (dictionary)

Lexikon: (dictionary, avbildningstabell, typ **dict**)

En samling av *key-value* par.

Ex 1:	Ex 2:	Ex 3:	Ex 4:
Anna 070-1122334	A .-	Sverige SEK	Eva 23
Pelle 0732255889	B -…	UK £	Carl 19
Olle 0708877441	C -.-	USA $	Svea 19

Key : **Value**

Syntax Python: Måsvingar { och }

friends = {’Anna’: ’070-1122334’, ’Pelle’: ’0732255889’, ’Olle’: ’0708877441’}

data = {’Eva’:23, ’koordinater’: (2,3), 44:[1,9,3]}

{key:value, key:value, …} söknyckel kopplas till värde

Key måste vara unikt och kan inte ändras (immutable).

Value kan vara vad som helst och kan återkomma och ändras.

Lexikon (dictionary)

```
print(friends)  # {’Anna’: ’070-1122334’, ’Pelle’: ’0732255889’, ’Olle’: ’0708877441’}

print(friends[’Olle’])  # 0708877441
print(friends.get('Anna’)) # 070-1122334

friends['Olle']='0739966551’ # Ändra ett value
friends['Nisse'] = '0704455772’  # Lägg till en post

del friends[’Anna’] # tar bort Anna
print(friends.pop('Pelle’))  # Tar bort Pelle, returnerar värdet 0732255889

if 'Gun' in friends:
    print('Gun is my friend')
else:
    friends['Gun']='0705566773’ # lägg till Gun
list_of_friends = list(friends) # Ska man importera en modul för att använda
                                # list-funktionen? Hur ser utskriften ut?
```

Lexikon (dictionary)

```
print('\nPrint the keys:’) # ELLER:
for x in friends.keys():     # for i in friends:
    print(x)                 #       print(i)

print('\nPrint the values:')
for z in friends.values():
    print(z)

print('\nPrint the pairs:’)   # ELLER:
for a, b in friends.items():  # for a in friends.items():
    print(a, b)               #     print(a)
```

Rast 5 minuter

Lexikonbyggare (dictionary comprehension)**, ett exempel:**
Vad skriv ut?

```
code = {chr(i):i  for  i  in range(65,91)}
print(code)
```

Samlingar (collections) och sekvenser i Python

Vissa typer av objekt i Python kan itereras (loopas) över, exempel:

```
total = 0
for tal in [3, -4, 7, 0 ,1]:
        total = total + tal
```

Sekvensen efter ”*in*” måste vara itererbar, en *iterabel*, dvs går att loopa över och returnera ett element i taget.

Samlingar är ”iterabler” i Python

Samlingar (collections) är datastrukturer med relaterade element. Pythons inbyggda samlingar gör att man kan lagra och få åtkomst till data enkelt och effektivt. Sekvenser är iterabla.

Ordnade samlingar, sekvenser: man kan nå elementen via deras **index**.

Listor:

li = [3, -4, 7, 0, 1]

Strängar: (?)

namn = ’Anna-Stina, 21’

Tuples:

min_t = (15, ’Abba’, 21, ’Bethoven’)

```
li = [3, -4, 7, 0, 1]
print(li[1]) # -4
li[0] = 8
print(li)    # [8, -4, 7, 0, 1]
```

Samlingar är ”iterables” i Python

Exempel på Pythons inbyggda oordnade samling (non-sequence collections):

Lexikon: (dictionary - avbildningstabell)

```
pris = {'äpplen': 12, 'bananer': 14, 'citroner': 20}
# print(pris[0]) FUNGERAR INTE! Lexikon är oordnad.
print(pris[’äpplen’]) # 12
for e in pris.items(): # e kommer successivt anta alla nyckel-värdepar
    print(e)
```

”Python’s dictionary is a shining star among its data structures; it is compact, fast, versatile, and extremely useful. It can be used to create a wide variety of mappings.”

Man måste skilja på:

Mutable (går att ändra):

Listor:

li = [3, -4, 7, 0, 1]

li[0] = 8 # [8, -4, 7, 0, 1]

Lexikon:

pris = {'äpplen': 12, 'bananer': 14, 'citroner': 20}

pris['bananer'] = 15

OBS! Key immutable; value mutable

Immutable (går inte att ändra):

Strängar:

namn = ’Anna-Stina, 21’

Tuples:

min_t = (15, ’Abba’, 21, ’Bethoven’)

Men: namn = ’Ludvig, 23’ funkar??

Variabeln namn pekar nu på en *ny* sträng.

Hantera text i Python

- Men strängar är ’imutable’, de går inte att ändra (men t ex listor kan ändras)
  Ex: namn[1]=’ä’ FEL!
- Men + och *operatorerna? Skapar *nya* strängar i minnet.
- Jämför: en ny sträng skapas och skrivs ut:

print(namn.replace('a', 'i’)) # Svei Lundmirk–Grinström

print(namn) # Svea Lundmark–Granström

- Jämför: en ny sträng skapas, namn pekar nu på den nya strängen:

namn = namn.replace(‘a’, ‘i’)

print(namn) # Svei Lundmirk–Grinström

Den inbyggda funktionen enumerate()

Man kan få både indexet och värdet när man itererar genom en sekvens med hjälp av den inbyggda funktionen **enumerate()**

Default-parameter

Syntax: enumerate(interable, start=0) #returnerar ett enumerate-objekt

```
town = ’Hjo’ #stäng
print('Index   Letter')
for i, v in enumerate(town):
    print(f'{i}{v:>7}')
```

Ger:

Index	Letter
0	H
1	j
2	o

Två variabler. Första blir index, andra elementet

```
animals = ('dog','cat','bird’) #tuple
print(list(enumerate(animals,10)))
```

Ger:

[(10, 'dog'), (11, 'cat'), (12, 'bird’)]

```
for i in enumerate(animals):
    print(i)
```

Ger:

(0, 'dog')
(1, 'cat')
(2, 'bird')

Rast 5 minuter

Datahantering

Hittills: Inmatning från *tangentbordet* och utskrift till *skärm*:

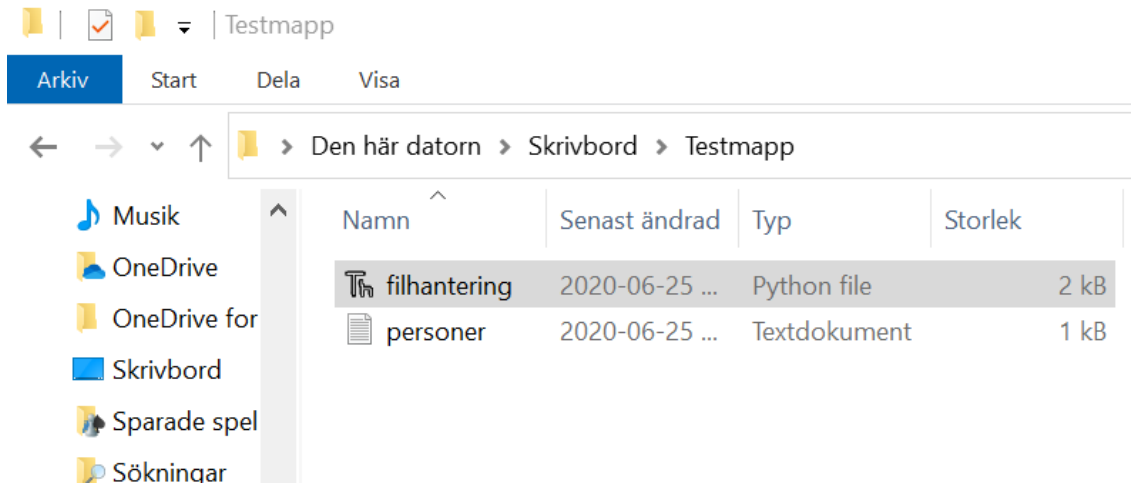
```
li = []
for i in range(4):
    li.append(int(input('Mata in medeltemperaturen: ')))
```

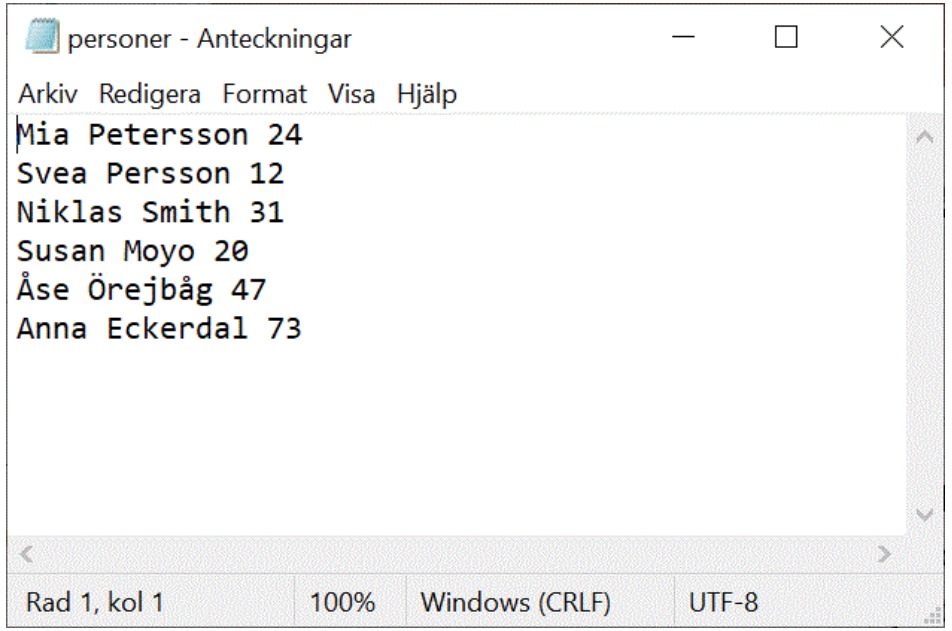
- Problem: inmatade och utskrivna data försvinner efter exekveringen.
- Problem: klumpigt, tidsödande och lätt att det blir fel vid manuell inmatning.
- Vi vill att datorn ska kunna läsa och processa *stora* datamängder från filer.

Filhantering

Läsa från *text-fil*

Antag att det finns en fil med text, t ex personer.txt, i samma mapp som ditt Python-program. Den filen kan t ex vara skapad t ex i Thonny eller Anteckningar, sparat på filformatet textfil.





Filhantering

Gör ett Python-program som läser text-filen och skriver ut innehållet.

Skriv Python-koden som:

– Skapar ett filobjekt

– Be det filobjektet att läsa från filen

- En rad i taget
- Eller hela filen

– Skriv ut innehållet på skärmen

Om ditt Pythonprogram körs i en annan katalog än filen finns i måste filnamnet ange sökvägen till katalogen. Enklast: spara filen där programmet körs

Filhantering

Läsa från *text-fil,* Pythonkod:

```
filnamn = input('Ange filens namn: ’) # t ex personer.txt
fi = open(filnamn, 'r', encoding='utf-8’) # fi är ett objekt. 'r': Öppna filen för läsning,
                                          # kan behöva utf-8 (eller 'latin-1’) för åäö.
for rad in fi: # fil-objekt är iterabla. Läs en rad i taget.
    print(rad, end='') # Utan end='' blir det 2 radbyten
# ELLER: print(fil.read()) #Läser hela filen, returnerar en sträng
fi.close() # Viktigt att stänga filen

# Inget import behövs, open(), print() och input() inbyggda funktioner.
```

Filer

Skriva på text-fil

```
utfil = input('Ange namn på filen som ska skapas: ’)
fu = open(utfil, 'w’) #fu refererar till utfil. ’w’: skriv över fil som finns eller skapa en ny fil
# fu = open(utfil, ’a’) Lägg till i existerande fil (a för append)
# fu = open(utfil, ’x’) Skriv på ny fil som skapas. Kastar fel om filen finns.
print('Mata in text, avsluta med 0:’)
text = input('Skriv en rad, avsluta med 0:')
while text !='0':
    fu.write(text+'\n')
    text = input('Skriv en rad till om du vill:')
fi.close()
```

Filer

Bättre att använda ’with’, *context manager.* Öppnade filer stängs automatiskt.

with open(n1, mode) as f1, open(n1, mode) as f2, …

```
readfile = input('Ange namnet på filen som ska läsas: ')
writefile = input('Ange namnet på filen som ska skrivas på: ’)
with open(readfile, 'r') as f1, open(writefile, 'w') as f2:
    for person in f1: #Läs en rad i taget från filen som f1 refererar till
        if 'Niklas' in person:
            continue # Hoppa över Niklas
        f2.write(person) # Skriv raden på filen som f2 refererar till
```

Filer

for r in f1: # Läser en rad i taget till r
f1.read() # Läser hela filen och returnerar en sträng
f1.readline() # Läser en rad. Returnerar ’ ’ vid filslut
f1.read(t) # Läser t st tecken. Returnerar ’ ’ vid filslut
f1.readlines() # Läser alla raderna. Ger en lista med rader.
f2.write() # Skriver en sträng på filen
f2.writelines() # Skriven en lista med ett listelement per rad.

Se minilektionen Läsa och skriva.

Extramaterial

Bild 31 - 35

Om funktionen part2 i lektion 7:

```
# sorted(iterable, *, key=None, reverse=False) # Se https://docs.python.org/3/library/functions.html#sorted
# 'key' specifies a function of one argument that is used to extract a comparison  key from each element in iterable (for example, key=str.lower).

def part2(e):
    return e[1]

# Funktionen sorted returnerar en sorterad lista
# Sortering baseras på värdet av funktionen part2 i fallande ordning
freq={'k': 5, 'v': 3, 'ä': 6, 'l': 8, 'e': 8, 'n': 13, 's': 5, 'g': 4, 'u': 1, 'm': 1, 'o': 3, 'f': 1, 't': 3, 'r': 6, 'a': 8, 'p': 1, 'å': 1, 'd': 3, 'c': 3, 'h': 1, 'b': 3, 'ö': 2, 'i': 3}
lista=list(freq.items())

freq_order = sorted(lista, key=part2, reverse=True)
for i, e in enumerate(freq_order, start=1):
    print(f'{e[1]:2d} {e[0]}', end='\t') # Skriv TAB efter varje utskrift
    if i % 6 == 0:
        print()        # Gör radbyte var 6:e varv i loope
```

Listor som referenser

```
def mult_lista1(lista, tal):     #lista pekar på ursprungliga listan a
    for i in range(len(lista)):
        lista[i] = tal*lista[i]         # och ändrar i den.

def mult_lista2(lista, tal):
    temp = []                                #lokal variabel skapas
    for i in range(len(lista)):
        temp.append(tal*lista[i]) #Ändringar in temp, inte i lista, dvs a
    return temp

a=[1,2,3,4,5]
print('Min ursprungliga lista:', a)

print('Testa mult_lista1 med tal 2:')
mult_lista1(a, 2)
print('Nu är listan multiplicerad med 2:', a)

print('Testa mult_lista2 med tal 3:')
ny = mult_lista2(a,3)
print('Nu är listan multiplicerad med 3:', ny)
print('Den ursprungliga listan ser nu ut så här:', a)
```

Reguljära uttryck, (Carl Nettelblad)

```
import re
instr = ’här har vi några ord’
ordlista = re.findall(r’[a-zA-ZåäöÄÖÖ]+’, instr)
```

- Ett reguljärt uttryck är en ”mall” för hur en sträng ska se ut. Här använder vi hakparentes för att ange flera möjliga tecken. Inuti hakarna anger - ett intervall av tecken.
- + betydelser en eller flera förekomster av föregående tecken. (ba+b matchar bab, baab, baaab o.s.v.)
- * innebär 0 eller flera förekomster (så ba*b matchar bb, bab, baab o.s.v.)
- Parentes kan innebära grupperingar ((ba)+ innebär ba, baba, bababa, o.s.v.
- ? innebär 0 eller 1 förekomst (så b(aa)?b innebär bb eller baab)

Reguljära uttryck: Olika funktioner

- findall (dvs re.findall())returnerar en lista med alla icke-överlappande träffar
  - Får du något annat än alla ord på OU3? Kolla ditt reguljära uttryck *noga*. Ett missat bindestreck, ett missat plustecken, ett extra mellanslag kan ge en lista, men något helt annat än alla ord. (Testa!)
- search söker efter en match (None om ingen finns)
- finditer returnerar ett itererbart objekt med alla ickeöverlappande träffar
  - Om du ändå ska köra i en for-slinga kan finditer vara bättre än findall
- sub ersätter icke-överlappande träffar på ett reguljärt uttryck med en sträng

Reguljära uttryck: Filtrera rader

- ”Hitta alla rader som börjar med !, slutar med ö och innehåller ett mellanslag följt av mer än 6 siffror i följd”

```
with open('filen.txt', 'r') as fil:
    for ln in fil:
        if re.search(r'^!.* [0-9]{7,}.*ö$', ln):
            print(ln)
```

- ^ början på strängen/raden
- . valfritt tecken
- $ slutet på strängen/raden

- Jämför med att skriva ett eget villkor för att kolla detta…

Reguljära uttryck inte bara i Python

- Många olika språk och bibliotek har stöd för reguljära uttryck
  - Lite olika exakt vilken syntax
- Reguljära uttryck innehåller ofta specialtecken
  - I Python sätter vi r före strängen för att markera ”rå” sträng, så många specialtecken tappar sin betydelse
- I terminalen kan vi använda kommandot grep för att söka i filer med reguljära uttryck

```
grep "^.*[0-9]\{7,\}.*ö$" test.txt
```

- Notera citationstecken och \ före { för att inte specialtecken ska feltolkas