Oppgave 1, Python

Høst 2023

I disse oppgavene skal vi introdusere Python, og vise hvordan vi kan jobbe med tekst i Python. Nærmere bestemt kommer vi her til å jobbe med strenger, lister, hvordan vi kan printe verdier, lese inn filer, og jobbe med tekststrenger i Python. Dersom dere ønsker å jobbe videre med Python på egenhånd anbefaler vi at dere jobber med Trix-oppgavene for IN1140 (se emnesiden).

1 Hva er Python?

Python er et programmeringsspråk. Dere skal lære å programmere med Python i dybde i IN1000. I IN1140, skal vi i hovedsak fokusere på bruk av Python for tekstdata.

Python har et eget vokabular, eller mengde med reserverte ord, som har en spesiell betydning i Python og er svært viktige. Når Python møter slike ord i et program/kode har ordet én og bare én betydning for Python. Noen av disse ordene er: `and`, `break`, `class`, `continue`, `def`, `elif`, `else`, `for`, `from`, `if`, `import`, `in`, `not`, `or`, `while`.

Når du skal skrive et program skal du kunne bruke dine egne ord som har betydning for deg og koden din. Disse ordene kalles variabler. Dine variabelnavn kan være alle ordene i vårt menneskelig vokabular, men du må aldri bruke et av Python sine reserverte ord som variabelnavn (ellers kommer Python til å tro at du mener noe annet enn det du faktisk mener, siden reserverte ord kun har én betydning for Python).

1.1 Installasjon

Om du ikke allerede har installert Python på maskinen din, vil gruppelæreren hjelpe deg med det.

2 Strenger i Python

En streng er en sekvens av symboler som også kan inneholde mellomrom og siffer. For eksempel både ordet "sitron" og frasen "kurven inneholder 3 sitroner" er strenger. En streng kan også være en sekvens av tall, feks "123456". For at Python skal kunne gjenkjenne at en variabel er av datatypen streng, må du alltid ha strengverdien mellom anførselstegn.

Du kan be Python om å vise deg verdien av en variabel ved å printe den på skjermen. Dette gjøres ved bruk av den innebygde funksjonen `print()`.

[1]:
```
minstreng = 'Hello world!'
ny_streng = '123456'
siste_streng = 'Jeg sto opp kl.5.00 i dag morges.'
```

```
print(siste_streng)
```

```
Jeg sto opp kl.5.00 i dag morges.
```

Strenger kan lages både med enkle eller doble anførselstegn. Pass på at du enten bruker `'` eller `"`. Det finnes andre anførselstegn som likner veldig, men de fungerer ikke. Nøyaktig hvordan de ser ut i din editor avhenger av skrifttypen den bruker.

[2]:
```
minstreng = "Hello world!"
ny_streng = "123456"
siste_streng = "Jeg sto opp kl.5.00 i dag morges."

print(siste_streng)
```

```
Jeg sto opp kl.5.00 i dag morges.
```

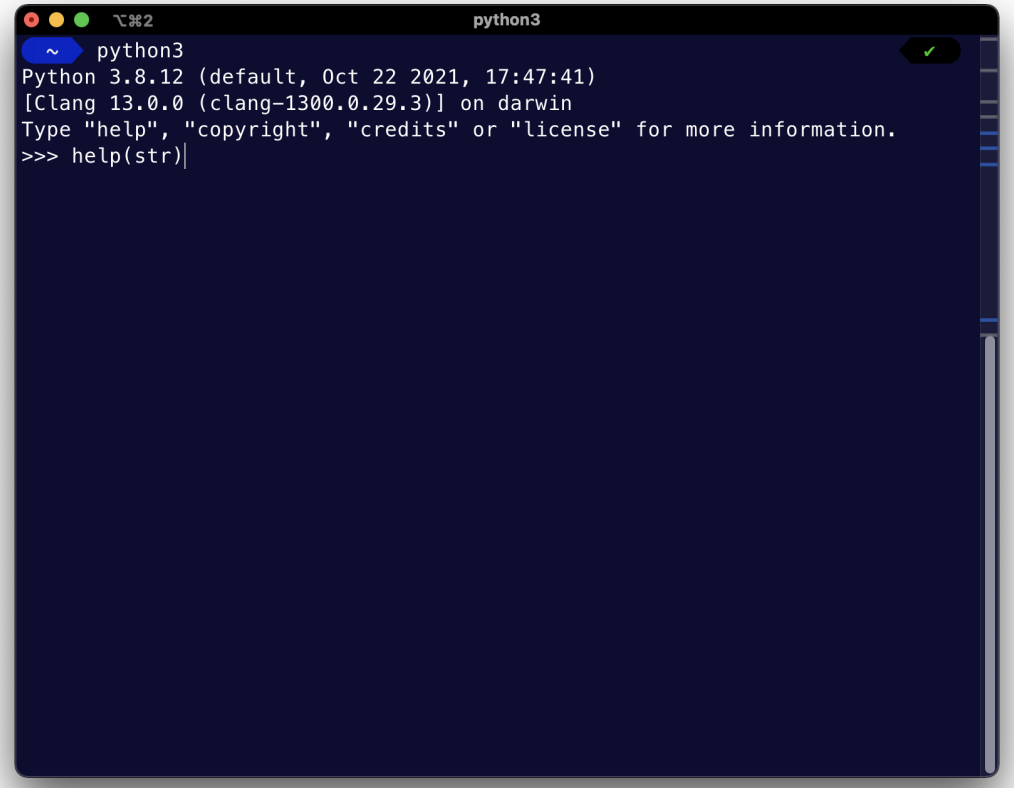
Strengen `"Hello world!"` kan også skrives ut uten å kalle på variabelen `minstreng`.

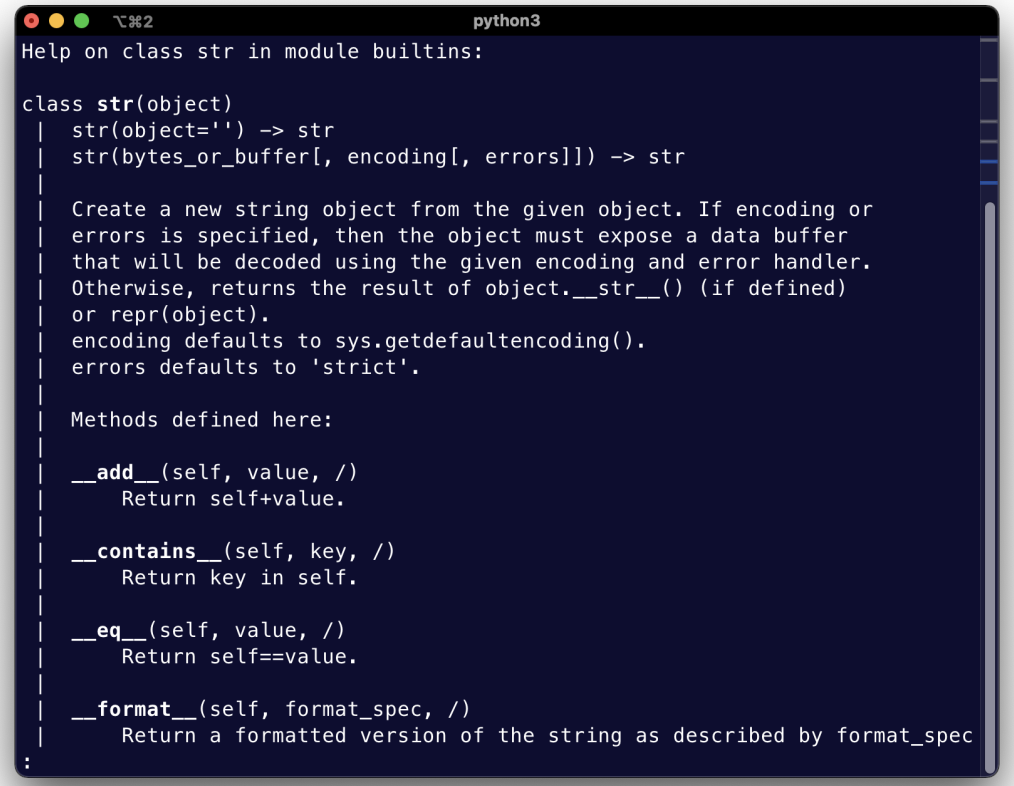
[3]:
```
print('Hello world!')
```

```
Hello world!
```

2.1 Strengmetoder

Python Library Reference inneholder en oversikt over metoder som er tilgjengelig for forskjellige datatyper. For strenger, finner du relevant informasjon på http://docs.python.org/library/stdtypes.html. Alternativt kan du skrive inn `help(str)` i et Python-shell for å få en oversikt over metodene som er tilgjengelig for strenger. Her vises alle metodene som er tilgjengelige for strenger, det er ganske mange fler enn de vi går gjennom her. Bruk pil opp og ned for å navigere. Trykk `Q` for å avslutte.


⌥⌘2
python3
~ python3
Python 3.8.12 (default, Oct 22 2021, 17:47:41)
[Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> help(str)


⌥⌘2
python3
Help on class str in module builtins:
class str(object)
| str(object='') -> str
| str(bytes_or_buffer[, encoding[, errors]]) -> str
|
| Create a new string object from the given object. If encoding or
| errors is specified, then the object must expose a data buffer
| that will be decoded using the given encoding and error handler.
| Otherwise, returns the result of object.__str__() (if defined)
| or repr(object).
| encoding defaults to sys.getdefaultencoding().
| errors defaults to 'strict'.
|
| Methods defined here:
|
| __add__(self, value, /)
| Return self+value.
|
| __contains__(self, key, /)
| Return key in self.
|
| __eq__(self, value, /)
| Return self==value.
|
| __format__(self, format_spec, /)
| Return a formatted version of the string as described by format_spec
:

2.1.1 `count()`

`count()` returnerer antall ikke-overlappende delstrenger (sendt med som argument mellom parantesene) i strengen den kalles på.

[4]:
```
mystring = 'hello'
antall = mystring.count('l')
print('Antall l-er:', antall)
```

```
Antall l-er: 2
```

I eksempelet over ser vi at `count()` kalles på strengen som ligger i variabelen `mystring`. Argumentet er strengen `'l'`.

2.1.2 `split()`

Dersom vi ikke sender noe argument inn i `split()`, deles strengen den kalles på, opp ved tomrom som mellomrom og linjeskift. Merk at når vi kaller på metoder, må vi alltid ha med parantesene bak, uansett om vi sender med et argument eller ikke.

[5]:
```
mystring = 'Hello world!'
oppdelt = mystring.split()
print(oppdelt)
```

```
['Hello', 'world!']
```

Dersom vi sender med en streng som argument, deles strengen opp ved argumentet.

[6]:
```
ny_streng = 'en blå badeball'
oppdelt = ny_streng.split('b')
print(oppdelt)
```

```
['en ', 'lå ', 'ade', 'all']
```

Resultatet er en liste av strenger. Som vi kan se, er argumentet `'b'` fjernet fra det oppdelte resultatet.

Vi kan også kalle `split()` direkte på en streng uten å lagre den i en variabel først.

[7]:
```
print('en blå badeball'.split('b'))
```

```
['en ', 'lå ', 'ade', 'all']
```

2.1.3 `endswith()`

`endswith()` returnerer `True` hvis strengen den kalles på, slutter på strengen som sendes med som argument. Ellers returnerer den `False`.

[8]:
```
mystring = 'Hello world!'
print(mystring.endswith('!'))
```

```
True
```

`True` skrives ut fordi strengen i variabelen `mystring` slutter på `'!'`.

[9]:
```
mystring = 'Hello world!'
print(mystring.endswith('?'))
```

```
False
```

2.1.4 `lower()`

`lower()` returnerer en kopi av strengen den kalles på, hvor alle store bokstaver (på fagspråket kalt *majuskler*) er gjort om til små (*minuskler*).

[10]:
```
mystring = 'HeLlO WoRlD!'
print(mystring.lower())
```

```
hello world!
```

2.2 Lister i Python

Lister i Python inneholder en samling elementer. Elementene i en liste kan være ord, tall, fraser, setninger, osv. I Python blir lister skrevet med firkantede parenteser og elementene er skilt av et komma:

[11]:
```
en_liste = ['hei!', 'kake', 42, True, False]
print(en_liste)
```

```
['hei!', 'kake', 42, True, False]
```

Vi lager en ny, tom liste:

[12]:
```
ny_liste = []
```

2.3 Listemetoder

2.3.1 `append()`

Du kan legge til elementer til listen din ved å bruke `append()`. Elementene vil alltid bli lagt til bakerst i listen. Du kan legge til det samme elementet flere ganger.

[13]:
```
ny_liste.append('første')
ny_liste.append('første')
ny_liste.append('andre')

print(ny_liste)
```

```
['første', 'første', 'andre']
```

2.3.2 `count()`

Du kan telle forekomster av en viss verdi ved å bruke metoden `count()`.

[14]:
```
antall = ny_liste.count('første')
print('Antall "første":', antall)
```

```
Antall "første": 2
```

2.3.3 remove()

Du kan slette den første forekomsten av en verdi ved å bruke metoden `remove()`.

[15]:
```
ny_liste.remove('første')
print(ny_liste)
```

```
['første', 'andre']
```

Vi kan gjøre det samme en gang til for å fjerne den andre forekomsten av `'første'`.

[16]:
```
ny_liste.remove('første')
print(ny_liste)
```

```
['andre']
```

Dersom vi gjør det samme enda en gang, får vi en feil (en såkalt `ValueError`) fordi det ikke er flere forekomster av `'første'` som kan fjernes, i listen. Slike feilmeldinger viser hvor feilen ligger og kan være nyttige å se på når vi skal feilsøke koden vår.

[ ]:
```
ny_liste.remove('første')
print(ny_liste)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/l6/bswkk8t15td75835k2hc39hw0000gn/T/ipykernel_42581/3681198555.py in <module>
----> 1 ny_liste.remove('første')
      2 print(ny_liste)

ValueError: list.remove(x): x not in list
```

2.3.4 sort()

Sorterer liste alfabetisk. Vi må være litt forsiktige med hvordan vi bruker `sort()`. Den returnerer ingen verdi, slik de tidligere metodene vi har sett på, har gjort.

[17]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
handleliste.sort()
print(handleliste)
```

```
['avokado', 'banan', 'brød', 'juice', 'kake']
```

Det kan virke intuitivt å bruke metoden slik som i de to eksemplene under, men merk hvordan de kun returnerer `None`:

[18]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
sortert_handleliste = handleliste.sort()
print(sortert_handleliste)
```

```
None
```

[19]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
print(handleliste.sort())
```

```
None
```

Sortere ikke-alfabetisk:

[20]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
handleliste.sort(reverse=True)
print(handleliste)
```

```
['kake', 'juice', 'brød', 'banan', 'avokado']
```

Vi kan også sortere lister med tall:

[21]:
```
liste_med_tall = [3, 1, -2, 0.5]
liste_med_tall.sort()
print(liste_med_tall)
```

```
[-2, 0.5, 1, 3]
```

2.3.5 Indeksering

Med indeksering kan vi hente ut et element fra en spesifikk posisjon i en liste. Når vi indekserer må vi huske å telle fra null! For å hente ut elementet på posisjon 1, må vi altså bruke indeks 0:

[22]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
print(handleliste[0])
```

```
brød
```

Hent ut element på posisjon 3 i listen:

[23]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
print(handleliste[2])
```

```
banan
```

Hent ut siste element i listen:

[24]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
print(handleliste[-1])
```

```
juice
```

Hent ut nest siste element i listen:

[25]:
```
handleliste = ['brød', 'kake', 'banan', 'avokado', 'juice']
print(handleliste[-2])
```

```
avokado
```

Angi ny verdi på posisjon 1 i listen:

[26]:
```
handleliste[0] = 'godteri'
print(handleliste)
```

```
['godteri', 'kake', 'banan', 'avokado', 'juice']
```

2.3.6 "Slicing"

Vi kan også hente ut en delsekvens fra en liste. Vi gjør dette ved å kutte, eller "slice", vekk elementene vi ikke vil ha med. Indeksene markerer hvor vi kutter.

Hent ut verdier fra og med indeks 1 til (og ikke med) indeks 3:

[28]:
```
møbler = ['stol', 'bord', 'kommode', 'gyngestol', 'seng']
print(møbler[1:3])
```

```
['bord', 'kommode']
```

Hent ut alle verdier fra og med indeks 2:

[29]:
```
print(møbler[2:])
```

```
['kommode', 'gyngestol', 'seng']
```

Hent ut de tre siste verdiene:

[30]:
```
print(møbler[-3:])
```

```
['kommode', 'gyngestol', 'seng']
```

Hent ut de to første verdiene: *(husk "til og ikke med")*

[31]:
```
print(møbler[:2])
```

```
['stol', 'bord']
```

Vi kan bruke indeksering og "slicing" på alle sekvenstyper i Python. Strenger kan anses som bokstavsekvenser.

[32]:
```
streng = 'abcde'
print(streng[2:4])
```

```
cd
```

[33]:
```
print(streng[:-1])
```

```
abcd
```

3 Åpne og lese filer i Python

Følgende krever at filen `test.txt` ligger i samme mappe som koden din og at du står med terminalen i denne mappen når du kjører koden din. Denne finner du på semestersiden. Du kan enten

åpne tekstfilen i nettleseren din og kopiere innholdet inn i en ny fil som du kaller `test.txt`, eller du kan høyreklikke på lenken til filen og velge "Lagre lenken som ...".

Det finnes flere måter å lese inn en fil på. Felles for alle er at vi bruker metoden `open()` som tar inn filstien som argument. Her bør vi også huske å spesifisere `encoding` får å sikre at vi leser norske bokstaver som *Æ*, *Ø* og *Å* og andre spesialtegn riktig. Selv om filinnlesingen fungerer fint uten spesifisering av `encoding` på ditt eget system, kan det hende at et annet system som skal kjøre koden din, krever det for at filen leses slik du ønsker. Filstien kan være enten relativ eller absolutt og sendes med som en streng. Når vi er ferdige med filen, bør vi lukke den med `close()`. Denne metoden brukes på filobjektet som returneres av `open()`.

Les inn hele filen som én streng:

[34]:
```
fil = open('test.txt', encoding='utf-8')
filinnhold = fil.read()
fil.close()
```

Vi kan prøve å skrive ut de 200 første tegnene i filen.

[35]:
```
print(filinnhold[:200])
```

```
Da Natalie Netland hadde svart på 40 boligannonser uten å få ett eneste napp,
var hun i ferd med å gi opp drømmen om å bli barnehagelærer. Så fikk hun et
uventa tilbud om både kost og losji.
"Nå kan j
```

Legg merke til at det kommer et linjebrudd etter *"losji."*. I strengen som ligger i `filinnhold`, er dette egentlig markert med `\n`, men `print()` tolker dette som at den skal starte på en ny linje.

Vi ser nå på hvordan vi leser inn filen linje for linje:

Her bruker vi en for-løkke til å iterere over filobjektet. Prøv å få løkken under til å gi mening ved å si "for hver linje i fil, skriv ut linje".

[36]:
```
fil = open('test.txt', encoding='utf-8')

# Her er løkken
for linje in fil:
    print(linje)

fil.close()
```

```
Da Natalie Netland hadde svart på 40 boligannonser uten å få ett eneste napp,
var hun i ferd med å gi opp drømmen om å bli barnehagelærer. Så fikk hun et
uventa tilbud om både kost og losji.

"Nå kan jeg bo her til jeg får flytte for meg selv" sier Egersund-jenta
takknemlig der hun står utenfor ett av Tromsøs hoteller.

Flere av dem tilbyr nå rom til studentene til det de kaller "kostpris". Det
utgjør rundt tre tusen kroner i uka. Det er Studentparlamentet som har inngått
avtale med hotellene.
```

```
For mange ungdommer betyr det at de slipper å forlate byen, men kan ha is i
magen og håpe på at boligjakten ender lykkelig.

Øyvind Alapnes er direktør på Clarion Hotel The Edge. "Vi tenker langsiktig og
vet hvor viktig studentene er for Tromsø. Vi ønsker at de skal føle seg velkomne
til byen. Nå har vi rundt 15 studenter innlosjert" forteller han.

Ifølge Studentparlamentet er det private leiemarkedet blitt mindre fordi stadig
flere huseiere foretrekker å leie ut til turister i stedet for til studenter.

Airbnb tilbakeviser at de spiser av studentenes boligmarked.

På landsbasis manglet over ti tusen studenter tak over hodet ved skolestart.
```

Alternativt kan vi lese inn linjene slik:

[37]:
```
fil = open('test.txt', encoding='utf-8')
fillinjer = fil.readlines()
fil.close()

print(fillinjer)
```

```
['Da Natalie Netland hadde svart på 40 boligannonser uten å få ett eneste napp,
var hun i ferd med å gi opp drømmen om å bli barnehagelærer. Så fikk hun et
uventa tilbud om både kost og losji.\n', '"Nå kan jeg bo her til jeg får flytte
for meg selv" sier Egersund-jenta takknemlig der hun står utenfor ett av Tromsøs
hoteller.\n', 'Flere av dem tilbyr nå rom til studentene til det de kaller
"kostpris". Det utgjør rundt tre tusen kroner i uka. Det er Studentparlamentet
som har inngått avtale med hotellene.\n', 'For mange ungdommer betyr det at de
slipper å forlate byen, men kan ha is i magen og håpe på at boligjakten ender
lykkelig.\n', 'Øyvind Alapnes er direktør på Clarion Hotel The Edge. "Vi tenker
langsiktig og vet hvor viktig studentene er for Tromsø. Vi ønsker at de skal
føle seg velkomne til byen. Nå har vi rundt 15 studenter innlosjert" forteller
han.\n', 'Ifølge Studentparlamentet er det private leiemarkedet blitt mindre
fordi stadig flere huseiere foretrekker å leie ut til turister i stedet for til
studenter.\n', 'Airbnb tilbakeviser at de spiser av studentenes boligmarked.\n',
'På landsbasis manglet over ti tusen studenter tak over hodet ved
skolestart.\n']
```

Legg merke til at innholdet ligger lagret som en liste, hvor hvert element er en linje. Dette ser vi på hakeparantesene (de firkantede) og på kommategnene mellom hvert element i listen. Legg også merke til markøren for linjeskift `\n` i slutten av hver linje.