INF100 Kræsjkurs

Tekna H24

Jakob Sverre Alexandersen

```
agenda = {
    'basics', 'betingelser', 'funksjoner',
    'strenger', 'lister',
    'tupler', 'løkker', 'oppslagsverk',
    'mengder', 'filer',
    'exceptions', 'moduler', 'problemløsing'
}
```

0 – Basics

Variabler

- Lagre informasjon som er tilgjengelig for senere

```
name = 'Jakob' #
age = 23
height = 181.2569
is_student = True
```

Datatyper

- Streng (string / str)
- Heltall (integer / int)
- Desimaltall (float)
- Boolsk verdi (boolean / bool)

```
name = 'Jakob'          # string
age = 23                # int
height = 181.2569       # float
is_student = True       # bool
```

- Bruke type() for å finne ut hvilken type en verdi er

Kommentarer

- Gjør koden mer forståelig
- “Kommentere” ut linjer med kode (hopper over)

```
# jeg er en kommentar

“””
Jeg er også en
kommentar
“””


‘’’
Jeg og
‘’’
```

Eksamen

- Kommentarer
- Om du ikke får til:
  - Formuler ideen din til sensor
  - Vær nøyaktig på hva du prøver å gjøre

- Eks:

```
"""
For å sjekke om ballen
treffer raqueten vil jeg
sjekke opp koordinatene til
ballen med raqueten. Om den
treffer vil jeg endre retning
på ballen og få den til å
sprette andre veien.
"""
```

Matematiske uttrykk

- Vi har tilgang til de fleste matematiske operatorene
  - +, -, *, /, ^
- Heltallsdivisjon og modulo

```
3 + 4 == 7
8 - 3 == 5
2*7 == 14
16/4 == 4
3**2 == 9
5//2 == 2
5%2 == 1
```

print()

- Tekst i terminalen

```
name = 'Jakob'
number_of_pets = 2
print('My name is ' + name)
print(name, 'has', number_of_pets, 'pets')
```

```
My name is Jakob
I have 2 pets
```

```
name = 'Jakob'
number_of_pets = 2
print(f'My name is {name}\nI have {number_of_pets} pets')
```

- F-strings

input()

- Informasjon fra brukeren
- Str by default

```
name = input("Enter your name: ")
print(f'Hello, {name}!')
```


```
Enter your name: Jakob
Hello, Jakob!
```

- OBS:

```
Enter a number: 1
Enter another number: 2
The sum of the two numbers is 12
```

len()

- Finne lengden på sekvenser
- Funker med alt som kan ha en lengde
  - Lister, strings, sets, dictionaries

```
names = ["Einar", ”Thone", "Paal"]
print(len(names)) #printer 3
```

Konvertering av typer

- Nyttig å kunne konvertere fra en type til en annen
- Eks: input()

```
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')
sum = float(num1) + float(num2)
print('The sum of {num1} and {num2} is {sum}')
```

2

- a = []
- b = "True"
- c = 42
- d = -1.5

Velg riktig data type

	int	list	(-error-)	bool	float	str
a*b	○	○	○	○	○	○
b*c	○	○	○	○	○	○
'b' + c	○	○	○	○	○	○
c*a	○	○	○	○	○	○
c == 10.3	○	○	○	○	○	○
a+a	○	○	○	○	○	○
len(c)	○	○	○	○	○	○
f"{int(d)+c}"	○	○	○	○	○	○
[b]	○	○	○	○	○	○
a+b	○	○	○	○	○	○

Oppgave 1

- Lag en enkel kalkulator som spør brukeren om to tall, a og b. Programmet skal regne ut summen $a + b$.
- Eksempelkjøring:

Enter a number: *4*
Enter another number: *5*
The sum is *9*

1 – betingelser

Datatype: boolean

- En bool har alltid en av to verdier: True eller False

- Eks:

```
we_love_python = True

semester_complete = False
```

Operasjoner for sammenligning

- Vi har følgende operatorer for sammenligning: ==, !=, <, >, >= , <=

```
"hi" == 'h' + 'i'        # True
5 == 3                   # False
4 > 2                    # True
5 < 1                    # False
4 >= 4                   # True
2 <= 3.14                # True
"cat" != "bunny"         # True
```

Betingelser (if, else)

```
passed_inf100 = True

if passed_inf100:
      print("You passed INF100!")
else:
      print("You did not pass INF100!")
```

Merk: vi trenger ikke skrive if passed_inf100 == True

Betingelser (if, elif, else)

- Programmet kan kun havne i *en* av kodeblokkene. Først sjekkes if, og om det er False, vil programmet sjekke elif. Om alle betingelser er False, vil det gå direkte til else.

```
num = -5

if num > 0:
        print("Positive number")

elif num == 0:
        print("Zero")

else:
        print("Negative number")
```

Velg de riktige linjene slik at outputet blir:
A
B
C

a = 450

(if a < 100:, if 'a' < 100:, if 'a' > 100:, if a > 100:)

- print('A')

(if a > 400:, elif a < 400:, else:, elif a > 400:)

- print('B')

(elif a % 10 == 0:, if a % 10 == 0:, if a % 10 != 0:, elif a % 10 != 0:)

- print('C')

(elif a < 1000:, if a < 500:, elif:, if a < 1000:)

- print('D')

Betinget verdi

```
num = 5

is_even = (num % 2 == 0)

print(f'The number is {"even" if is_even else "odd"}')
```

Printer: The number is odd

and, or, not operatorer

- Lar oss kombinere betingelser

```
True and True          # True
False and True         # False
False and False        # False

True or False          # True
True or True           # True
False or False         # False

not True               # False
not (not True)         # True
True and not False     # True
not True or False      # False
```

a	b	a and b	a or b
True	True	Select alternati (True, False)	Select alternati (False, True)
True	False	Select alternati (True, False)	Select alternati (True, False)
False	False	Select alternati (False, True)	Select alternati (True, False)
5 > 3	5 > 7	Select alternati (False, True)	Select alternati (False, True)

Velg resultatet av hvert *boolsk* uttrykk.

	False	True
5 < 7 or 4 > 5	○	○
True or False	○	○
18 < 20 < 21 < 27 < 25	○	○
list(range(3)) == [1,2,3]	○	○
not (not (not False))	○	○
False and True	○	○
5 in range(5)	○	○
25 // 2 == 12	○	○
list(zip([4],[7])) == [(4,7)]	○	○
[x**2 for x in range(3)] == [0,1,4,9]	○	○

Presedens

Dette forteller oss hvilken rekkefølge operasjoner gjøres i

Operatorer	
`()`	Parentes. Det som er mellom parentesene evalueres før en operator utenfor parentesen.
`**`	Eksponentiering. Assosierer høyre-til-venstre.
`* / // %`	Multiplikasjon, divisjon og modulo.
`+ -`	Addisjon og subtraksjon.
`< <= > >= == != in not in`	Relasjoner. Disse vil *ikke* assosiere verken til høyre eller venstre; dersom man har flere slike etter hverandre vil de *komponeres* som en konjunksjon i stedet. For eksempel, `-1 < 0 == False` gir det samme svaret som `-1 < 0 and 0 == False`, og altså ikke det samme som `(-1 < 0) == False` slik man ellers kunne trodd.
`not`	Logisk negasjon.
`and`	Logisk konjunksjon.
`or`	Logisk disjunksjon.
`if else`	Betinget verdi.

Matematisk presedens

Python følger PEMDAS (vanlig matematisk presedens)

```
3 + 2*4        # 11
4 + 6%3        # 4
3**2*2         # 18
```

Alle tegn assosierer venstre til høyre med unntak av potens (**)

```
3 - 2 - 1      # 0
3**3**2        # 19683
```

Logisk presedens

- Husk at de logiske operatorene har <, >, osv.. og ‘in’ høyest presedens. Etter det kommer ‘not’, så ‘and’, etterfulgt av ‘or’

```
True or True and False      # True
True or (True and False)    # Use parentheses!
not False or True           # True
(not False) or True         # True
not (False or True)         # False

2 < 3 < 4                   # True
not 3 < 2 < 1               # True
not 3 < 2 and 1 < 2         # True

"e" in "hello"              # True
"a" and "e" in "hello"      # True
```

Truthy og falsy verdier

Alle verdier i python har en gjemt egenskap. Man konverterer verdien til en bool, vil den enten være True eller False (truthy eller falsy). Dette lar oss bruke verdien alene som en betingelse. Regelen er at alle ‘tomme’ verdier (0, 0.0, “”, [], None) er falsy, og alt annet er truthy.

```
user_input = input("Write something: ")

if user_input:
    print(f"The user wrote \"{user_input}\"")
else:
    print("The user did not write anything")
```

```
Write something: test
The user wrote "test"

Write something:
The user did not write anything
```

Oppgave 2

Utvid kalkulatoren slik at den først spør burkeren om to tall, så en matematisk operasjon. Programmet skal bruke operasjonen for å regne ut resultatet.

Eksempelkjøring:

```
Enter a number: 3
Enter another number: 8
Pick an operation (add, subtract, multiply, divide): multiply
The answer is 24
```

2 – funksjoner

$$\zeta(x) = \sum_{n=1}^{\infty} n^{-x}$$

Syntaks

- For å definere en funksjon, skriver vi ‘def’ etterfulgt av navnet og et sett med paranteser.
- Kodeblokk som viser hva som er en del av funksjonen
- Vi ‘kaller på’ funksjonen

```
def my_func():
        print('halla fra inne i funksjonen')

print('halla fra utsiden av funksjonen')

my_func()
```

Argumenter og parametre

Brukes for å ‘kommunisere’ med funksjoner. Argumenter: det som blir sendt inn i funksjonen når vi kaller på den. Parametere: defineres i funksjonen når den opprettes (hvilke data den tar inn).

```
def print_n_plus_two(n):
    print(n + 2)
```

7

```
number = 5
print_n_plus_two(number)
```

returverdi

- Verdien som funksjonen lagrer i minnet, som man kan hente senere i programmet

```
def squared(x):
return x**2

apples = 2

print(squared(apples)) # 4
apples = squared(apples) # 4
apples = squared(apples) # 16
apples = squared(apples) # 256
apples = squared(apples) # 65536
```

Innebygde funksjoner

Verktøy som vi kan bruke i koden vår

Eksempler:

- `print()`, `input()`, `len()`, `type()`
- `str()`, `int()`, `float()`, `bool()`, `list()`
- `abs()`, `max()`, `min()`, `round()`, `sum()`
- `range()`, `enumerate()`, `zip()`

Oppgave 3

Hva vil denne koden printe ut?

```
def add(a, b=4):
    a += b
    return a

def find_sum_of_abc(a):
    b = add(a*2)
    c = add(b, a)
    return a + b + c

print(find_sum_of_abc(2))
```

3 – strings

Bygge strenger

```
message = name + ' is ' + str(age) + ' years old and is ' + str(height) + ‘ m tall.’
```

Merk at vi må konvertere **int** og **float** til **string**. Vi bruker **str()** funksjonen.

For at teksten skal bli lesbar tar vi med mellomrom mellom variablene

F-strings

F-strings lar oss enklere bygge en streng

```
message = f'{name} is {age} years old and {height} meters tall.'
message += f'\n{name} has {"not " if not has_passed else ""}passed INF100.'
```

Merk:

- Vi trenger ikke å konvertere variabler til strenger
- Vi veksler mellom ” og ‘

Jobbe med strenger

Escape tegn

```
print("start\n\t\\end")
```

```
start
    \end
```

Sette sammen strenger

```
"abc" + "def" # "abcdef"
"abc" * 3     # "abcabcabc"
```

Medlemskap

```
"a" in "abc"   # True
"bc" in "abc" # True
"ac" in "abc" # False
```

Hente ut tegn

- På samme måte som lister, kan vi indeksere strenger
- Starter på 0

```
s = "Hello, World!"


print(s[0])     # H
print(s[5])     # ,
print(s[-1])    # !
print(s[-4])    # r
```

Slicing

- Veldig likt som indeksering
- Bruker kolon for å skille
- String[start : stop : step]

```
s = "Hello, World!"

print(s[0:5])      # Hello
print(s[:5])       # Hello
print(s[7:])       # World!

print(s[0:5:2])    # Hlo
print(s[::2])      # Hlo ol!
```

Reversering av streng

Sett steg-delen til -1:

```
s = "Hello, World!"

print(s[::-1])  # !dlroW ,olleH
```

Streng-metoder

Metoder er (som innebygde funksjoner) verktøy som vi kan bruke i Python, men de brukes på en litt annerledes måte. Forskjellige typer har forskjellige metoder.

Streng-metode .upper(), .lower(), .capitalize() og .title()

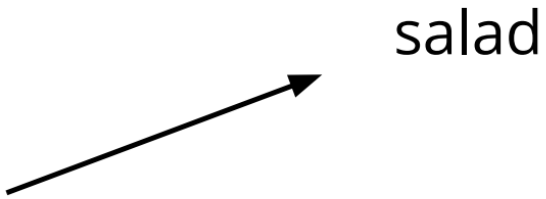
```
s = "hello, WORLD!"
```

`print(s.upper())`	→	HELLO, WORLD!
`print(s.lower())`	→	hello, world!
`print(s.capitalize())`	→	Hello, world!
`print(s.title())`	→	Hello, World!

Streng-metode .replace()

Med .replace() kan vi erstatte en substring med en annen:

```
food = "salmon"

print(food.replace("mon", "ad"))
```



Streng-metode .strip()

.strip() fjerner trailing og leading whitespace (før og etter)

Før:

```
s = '  hello   \n'
print(s)
print(repr(s))
print("length:", len(s))
```

```
  hello

' hello   \n'
length: 11
```

Etter:

```
s = s.strip()
print(s)
print(repr(s))
print("length:", len(s))
```

```
hello
'hello'
length: 5
```

Streng-metode .split()

.split() deler opp en streng i en liste. Tar inn tegn du vil splitte på som argument. Whitespace by default

```
sentence = "Once upon a time"
print(sentence.split())
```

→ ['Once', 'upon', 'a', 'time']

```
print(sentence.split("n"))
```

→ ['O', 'ce upo', ' a time']

Streng-metode .join()

.join() tar en liste med strenger og slår de sammen til en lang streng

```
sentence = " ".join(["Once", "upon", "a", "time"])

print(sentence)
```



```
Once upon a time
```

Streng som liste

Når vi programmerer i python kan det noen ganger hjelpe å tenke på en streng som en *liste av tegn*. Strenger og lister har, som vi kommer til å se, mange like egenskaper.

Hint: iterabel

```
print( list("Hello, World!") )
```

```
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
```

4 – lister

Lister

```
numbers = [1, 2, 3, 4, 5]
bools = [True, False, True]
misc = [1, "a", True, 3.14]

fruits = ["apple", "banana", "cherry"]

if "cherry" in fruits:
    print("Cherry is in the list")
else:
    print("Cherry is not in the list")
```

Cherry is in the list

Hente ut elementer

Vi henter ut elementer fra lister på samme måte som vi henter ut tegn fra strenger.

```
animals = ["elephant", "tiger", "giraffe", "lion"]

print(animals[3])  ⟶  lion
```

Slicing

På samme måte som på strenger, kan man også slice lister

```
fruits = ["apple", "banana", "cherry", "pear", "mango"]

# fruits[start:stop]
# fruits[start:stop:step]

print(fruits[1:3])      # ["banana", "cherry"]
print(fruits[:3])       # ["apple", "banana", "cherry"]
print(fruits[2:])       # ["cherry", "pear", "mango"]
print(fruits[::2])      # ["apple", "cherry", "mango"]
print(fruits[::-1])     # ["mango", "pear", "cherry", "banana", "apple"]
```

```
a = [1, 0, -2, 2, 5, 3]
```

Gitt at koden over er kjørt.
Hva skrives ut i de følgende setningene? (hvis programmet krasjer, skriv kun 'Error')

print(a[4])	
print(a[3 - 1])	
print(a[3] - 1)	
print(a[a[-1]-1])	
print(a[a[0] - a[1]])	
print(a[a[a[0]]])	

Unpacking

Å ‘unpacke’ en liste (eller tuppel) lar oss lagre innholdet i egne variabler

```
city, country, population = ["Oslo", "Norway", 1_000_000]

print(f"{city} is in {country} and has a population of {population}")
```

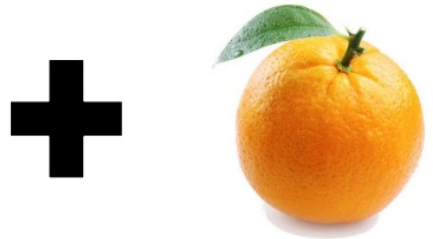
Alternativet er å hente ut elementer via indeksering

```
details = ["Oslo", "Norway", 1_000_000]
city = details[0]
country = details[1]
population = details[2]
```

.append() og .remove()

.append() legger til et element på slutten av listen

```
fruits = ["apple", "banana", "cherry"]
```



```
fruits.append("orange")
print(fruits)
```

→ ['apple', 'banana', 'cherry', 'orange']

.remove() fjerner et spesifikt element fra listen

```
fruits.remove("apple")
print(fruits)
```

→ ['banana', 'cherry', 'orange']

.insert() og .pop()

.insert() legger til et element på en gitt indeks i listen

```
fruits = ["apple", "banana", "cherry"]
fruits.insert(2, "kiwi")
print(fruits)
```



```
['apple', 'banana', 'kiwi', 'cherry']
```

.pop() fjerner og *returnerer* et element fra en gitt indeks i listen. Om man ikke oppgir en indeks, vil det siste elementet bli fjernet

```
fruits.pop(1)
print(fruits)
```



```
['apple', 'kiwi', 'cherry']
```

.extend()

Setter sammen to lister

```
fruits = ["apple", "banana", "cherry"]

fruits.extend(["mango", "papaya"])
print(fruits)
```

→ ['apple', 'banana', 'cherry', 'mango', 'papaya']

Man kan også gjøre det samme med +=

```
fruits += ["mango", "papaya"]
```

.sort()

Sorterer listen fra lavest til høyest verdi

```
numbers = [6, 2, 8, 3, 1]

numbers.sort()
print(numbers)
```

```
[1, 2, 3, 6, 8]
```

Metoden tar reverse som argument

```
numbers.sort(reverse=True)
print(numbers)
```

```
[8, 6, 3, 2, 1]
```

.reverse()

Reverserer listen

```
numbers = [6, 2, 8, 3, 1]

numbers.reverse()
print(numbers)
```



```
[1, 3, 8, 2, 6]
```

.index()

Gir oss indeksen av et spesifikt element i listen med frukt

```
fruits = ["apple", "banana", "cherry", "pear"]

print(fruits.index("cherry"))
```


2

.count()

Gir oss antallet av et gitt element i en liste

```
fruits = ["pineapple", "banana", "cherry", "pineapple", "pear"]

print(fruits.count("apple"))        ⟶   0

print(fruits.count("pineapple"))    ⟶   2
```

Ikke-destruktive funksjoner

- Funksjon som tar inn data
- Returnerer en endret kopi av argumentet

```
def double(nums):

    new_list = []

    for num in nums:
        new_list.append(num * 2)

    return new_list
```

Destruktive funksjoner

- Gjør endringer direkte
- Trenger ikke å returnere noe
- Listen er blitt endret på også utfor funksjonen

```
def double(nums):
    for i in range(len(nums)):
        nums[i] *= 2
```

List comprehension

List comprehension kan brukes som en snarvei for å bygge lister. Følgende list comprehension gjør det samme som den ikke-destruktive funksjonen double()

```
def double(nums):
    return [num * 2 for num in nums]
```

Vi kan gjøre mye på en linje ved bruk av list comprehension:

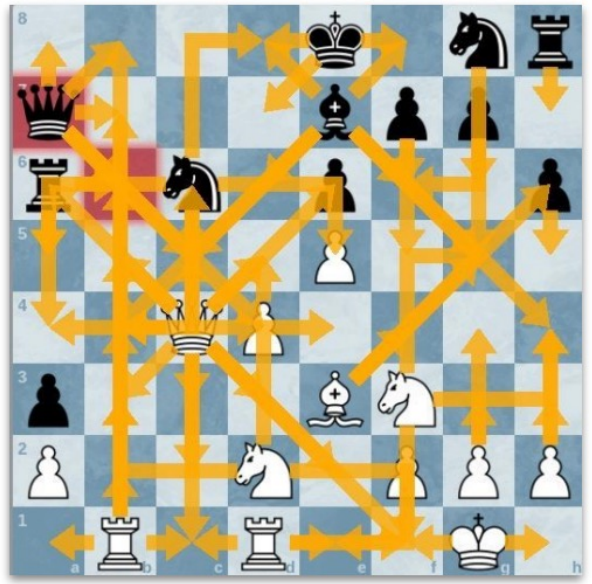
```
# The sum of all numbers up to 1000 that are divisible by 3 or 5
print(sum([num for num in range(1000) if num % 3 == 0 or num % 5 == 0]))
```

(Strengt tatt ikke nødvendig å kunne)

2D-lister

Lister kan inneholde andre lister

```
board = [
    ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'],
    ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],
    ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R']
]
```



5 – tupler

Tupler

- Som en liste, men immutable
  - Kan ikke endres
- 2D koordinater

```
coordinate = [3, 4]
```

```
coordinate = (3, 4)
```

- Gjør koden sikrere det vi vet vi kan bruke tupler

Tuppel-metoder

```
my_tuple = (1, 6, 4, 3, 1, 2, 4, 1)

print(my_tuple.count(1))
```

```
3

print(my_tuple.index(4))
```

```
2
```

5 – løkker

While-løkker

Kjører så lenge en betingelse er sann (True)

```
n = 1

while n < 5:
    print('n =', n)
    n += 1

print('Done')
```



```
n = 1
n = 2
n = 3
n = 4
Done
```

14

In the game "Rock Paper Scissors"

- Rock wins over Scissors,
- Scissors wins over Paper and
- Paper wins over Rock.

Write a program that plays Rock Paper Scissors against the user, keeping a score over several rounds.

For-løkker

Brukes når vi vet hvor mange ganger løkken skal kjøre

```
animals = ["dog", "cat", "bird"]

for animal in animals:
    print(animal)
```



```
dog
cat
bird
```

```
string = "Hi!"

for letter in string:
    print(letter)
```



```
H
i
!
```

range()

En ‘liste’ med tall som teller på en spesifikk måte

```
range(5)  ⟶  [0, 1, 2, 3, 4]

range(4, 8)  ⟶  [4, 5, 6, 7]
```

Tar opp til tre argumenter:

```
# range(start, stop, step)
# range(start, stop)        <- step = 1
# range(stop)               <- start = 0, step = 1
```

For-løkker med range()

Vi bruker ofte range() i sammenheng med for-løkker for å telle gjennom indeksene i en sekvens

```
soda = ["Pepsi", "Coke", "Sprite"]

for i in range(len(soda)):
    print(i, soda[i])
```

→

```
0 Pepsi
1 Coke
2 Sprite
```

enumerate()

En blanding av iterering på indeks og element!

```
soda = ["Pepsi", "Coke", "Sprite"]

for i, drink in enumerate(soda):
    print(i, drink)
```



```
0 Pepsi
1 Coke
2 Sprite
```

continue og break

Hopper rett til neste iterasjon av en løkke

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```

```
while True:
    user_input = input("Enter a number: ")
    if user_input == "quit":
        break
    total += int(user_input)

print("The total is", total)
```

Break hopper ut av en løkke

zip()

Henter ut elementer og ‘zipper’ elementene indeksvis inn i tupler i en ny liste

```
xs = [3, 2, 6]
ys = [2, 4, 1]

print( list( zip(xs, ys) ) )
```



```
[(3, 2), (2, 4), (6, 1)]
```

```
for x, y in zip(xs, ys):
    print(f"{x}, {y}")
```



```
3, 2
2, 4
6, 1
```

Nøstede løkker

- En løkke inni en annen løkke
- 2D-lister

```
rows = 3
cols = 3

for row in range(rows):
    print("Outer loop. Row:", row)
    for col in range(cols):
        print("- Inner loop. Col:", col)
```

```
Outer loop. Row: 0
- Inner loop. Col: 0
- Inner loop. Col: 1
- Inner loop. Col: 2
Outer loop. Row: 1
- Inner loop. Col: 0
- Inner loop. Col: 1
- Inner loop. Col: 2
Outer loop. Row: 2
- Inner loop. Col: 0
- Inner loop. Col: 1
- Inner loop. Col: 2
```

Oppgave 4

Ved å bruke en løkke, skriv et program som teller hvor mange unike tall som finnes i en liste

Bruk gjerne koden til høyre for å komme i gang

```
numbers = [1, 6, 2, 9, 4, 2, 1]

def unique_words(numbers):
    # Your code here
    ...

print(unique_words(numbers))
```

7 – oppslagsverk

Oppslagsverk (dictionaries)

Datastruktur som bruker nøkkel : verdi (key : value). For å hente ut verdier bruker vi nøklene, og ikke indeks

```
pet_translator = {
    'woof': 'Feed me, human!',
    'meow': 'Pet me, but only exactly 3 times',
    'chirp': 'Clean my cage, please',
    'hiss': 'You are not my favorite human right now',
}
```

Som navnet foreslår er det praktisk å bruke når man trenger slå-opp type funksjonalitet. OBS: man kan ikke ha duplikater av keys i dict

Hente ut verdier fra dict

Man bruker key for å hente ut en verdi:

```
pet_phrase = 'woof'
print(f"When your pet says '{pet_phrase}', it means '{pet_translator[pet_phrase]}'")
```

```
When your pet says 'woof', it means 'Feed me, human!'
```

Her kan man få KeyError om nøkkelen ikke eksisterer

Derfor kan man bruke .get() som tillater en default-verdi dersom nøkkelen ikke eksisterer

.get()

Trygg måte å hente ut verdier. Metoden tar en nøkkel og en valgfri verdi som argument. Det andre argumentet er hva metoden skal returnere om nøkkelen ikke finnes

```
print(pet_translator.get("ribbit", "Translation not found"))
```



```
Translation not found
```

Gjøre endringer i dict

Vi kan legge til eller oppdatere et element ved å skrive følgende:

```
pet_translator['meow'] = "Pet me, but only if I'm in the mood"
print(pet_translator)
```

```
{'woof': 'Feed me, human!', 'meow': "Pet me, but only
if I'm in the mood", 'chirp': 'Clean my cage, please',
'hiss': 'You are not my favorite human right now'}
```

.keys() og .values()

.keys() gir oss en ‘liste’ som inneholder alle nøklene i en dict

```
print( list( pet_translator.keys() ) )
```

→ ['woof', 'meow', 'chirp', 'hiss']

.values() gir oss en ‘liste’ som inneholder alle verdiene i en dict

```
print( list( pet_translator.values() ) )
```

↘

['Feed me, human!', "Pet me, but only if I'm in the mood", 'Clean my cage, please',
'You are not my favorite human right now']

.items()

.items() gir oss parvis keys og items som tupler i en ‘liste’

```
print( list( pet_translator.items() ) )
```

```
[('woof', 'Feed me, human!'), ('meow', "Pet me, but only if I'm in the mood"), ('chirp',
'Clean my cage, please'), ('hiss', 'You are not my favorite human right now')]
```

Fjerne ting fra oppslagsverk

For å fjerne nøkkel/verdi par fra en dict kan man bruke ‘del’ (for delete):

```
del pet_translator['chirp']
print(pet_translator)
```

```
{'woof': 'Feed me, human!', 'meow': "Pet me, but only if I'm in
the mood", 'hiss': 'You are not my favorite human right now'}
```

8 – mengder

Mengder (sets)

Som lister, men kun unike elementer

```
numbers = {1, 4, 2, 1, 2}
print(numbers)
```

```
{1, 2, 4}
```

Derfor kan man fjerne duplikater fra en liste ved å konvertere den til et set, så tilbake til en liste

```
numbers = [1, 4, 2, 1, 2]
print( list( set(numbers) ) )
```

```
[1, 2, 4]
```

OBS: sets er ‘unordered’, så rekkefølgen beholdes ikke

.add()

Legger til elementer i en mengde

```
numbers = {1, 2, 3}

numbers.add(4)
print(numbers)
```

{1, 2, 3, 4}

.remove()

Fjerner elementer fra en mengde

```
numbers = {1, 2, 3}

numbers.remove(2)
print(numbers)
```

{1, 3}

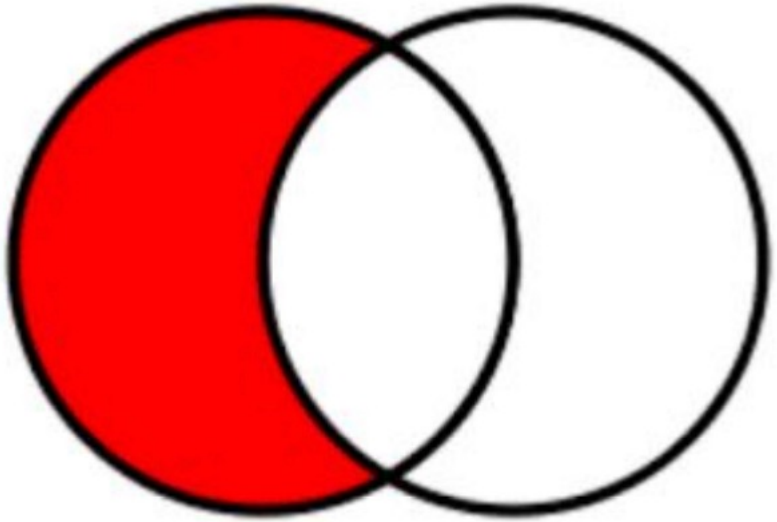
Operator: -

Om man er kjent med set theory vet man at det er diverse operasjoner som vi kan bruke på mengder

$x - y$ gir oss en mengde som inneholder kun elementer fra en mengde x som ikke er i mengden y

```
x = {1, 2, 3}
y = {2, 3, 4}

print(x - y)
```

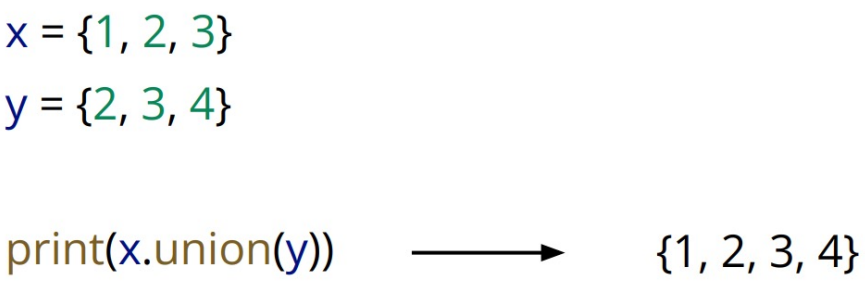
```
{1}
```

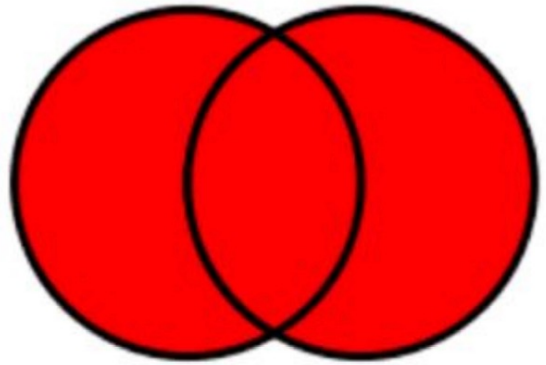


.union()

x.union(y) gir oss en mengde som inneholder alle elementene fra mengdene x og y

```
x = {1, 2, 3}
y = {2, 3, 4}

print(x.union(y))
```

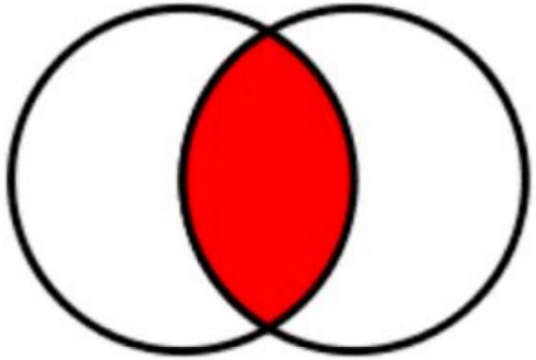




.intersection()

x.intersection(y) gir oss en mengde som kun inneholder elementene settene x og y har til felles

```
x = {1, 2, 3}
y = {2, 3, 4}

print(x.intersection(y))  ⟶  {2, 3}
```



.issubset()

x.issubset(y) returnerer True dersom alle elementene i mengden x eksisterer i mengden y. Ellers False

```
x = {1, 2, 3}
y = {2, 3, 4}

print(x.issubset(y))
```

False

```
x = {2, 3}
y = {2, 3, 4}

print(x.issubset(y))
```

True

Oppgave 5

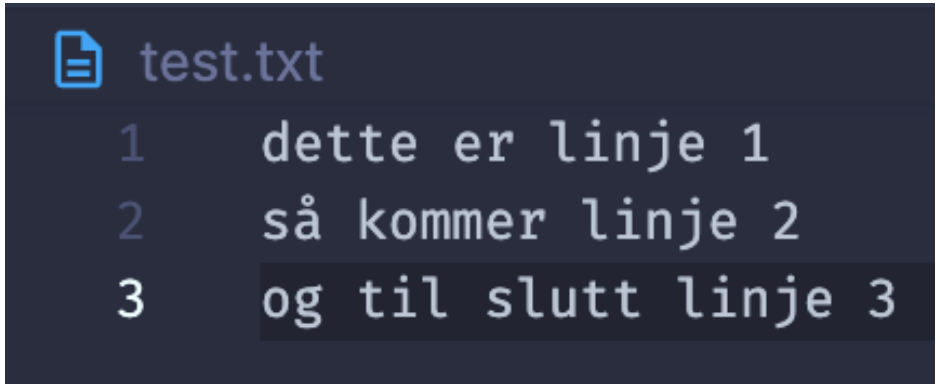
Ved å bruke mengder, skriv et program som teller hvor mange unike ord som finnes i en tekst

```
text = "hello test one two three test"


def unique_words(text):
    # Code goes here
    ...


print(unique_words(text))
```

9 – filer

Jobbe med filer

I følgende eksempler skal vi bruke en testfil som ser slik ut:



Gå gjennom fil linje for linje

```python
from pathlib import Path

content = Path("test.txt").read_text().splitlines()

for line in content:
    print(line)
```

```
dette er linje 1
så kommer linje 2
og til slutt linje 3
```

Lese en fil som *en* streng

```python
from pathlib import Path

content = Path("test.txt").read_text()
print(content)
```

```
dette er linje 1
så kommer linje 2
og til slutt linje 3
```

En liste med hver linje som element

```
from pathlib import Path

content = Path("test.txt").read_text().splitlines()
print(content)
```

```
['dette er linje 1', 'så kommer linje 2', 'og til slutt linje 3']
```

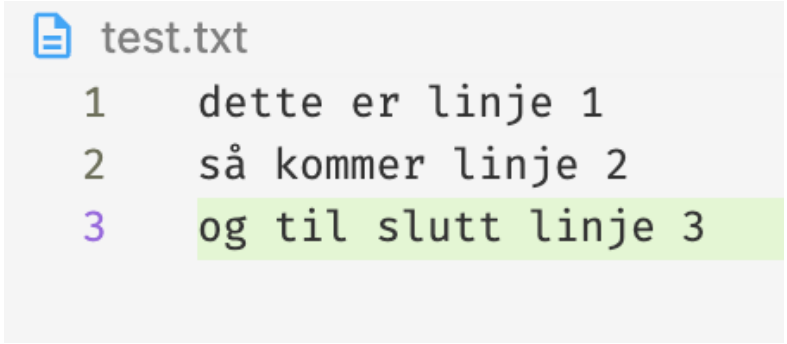
En liste med alle ord som element

```python
from pathlib import Path

content = Path("test.txt").read_text().split()
print(content)
```

```
['dette', 'er', 'linje', '1', 'så', 'kommer', 'linje', '2', 'og', 'til', 'slutt', 'linje', '3']
```

Skrive inn i fil (fjerner tidligere innhold)

```python
from pathlib import Path

content = Path("test.txt").write_text('Hello!')
```



Skrive inn i fil (legger til på slutten)

```python
from pathlib import Path

content = Path("test.txt").read_text()
add_content = '\nHello!'
Path("test.txt").write_text(content + add_content)
```

test.txt

```
1    dette er linje 1
2    så kommer linje 2
3    og til slutt linje 3
```

test.txt

```
1    dette er linje 1
2    så kommer linje 2
3    og til slutt linje 3
4    Hello!
```

CSV-fil: fra CSV til 2D-liste

```python
import csv
import io
from pathlib import Path

content = Path("test.csv").read_text()
reader = csv.reader(io.StringIO(content))
table = list(reader)
headers = table[0]
data = table[1:]

print(headers)
print(data)
```

test.csv > data

```
1   ID;alder;snitt
2   93;25;"B"
3   01;21;"A"
4   83;22;"C"
5   77;23;"C"
```

```
['ID;alder;snitt']
[['93;25;"B"'], ['01;21;"A"'], ['83;22;"C"'], ['77;23;"C"']]
```

CSV-fil: fra 2D-liste til CSV-fil

```python
import csv
import io
from pathlib import Path
headers = ['ID', 'bonus’]
data = [
    ['A', 100],
    ['B', 200],
    ['C', 300],
]
output = io.StringIO()
writer = csv.writer(output, delimiter=';', quotechar="'")
writer.writerow(headers)
writer.writerows(data)

Path('output.csv').write_text(output.getvalue())
```

output.csv > data

```
1    ID;bonus
2    A;100
3    B;200
4    C;300
```

2 Read each line from a text file and print the number of characters in the line

filename = "foo.txt"

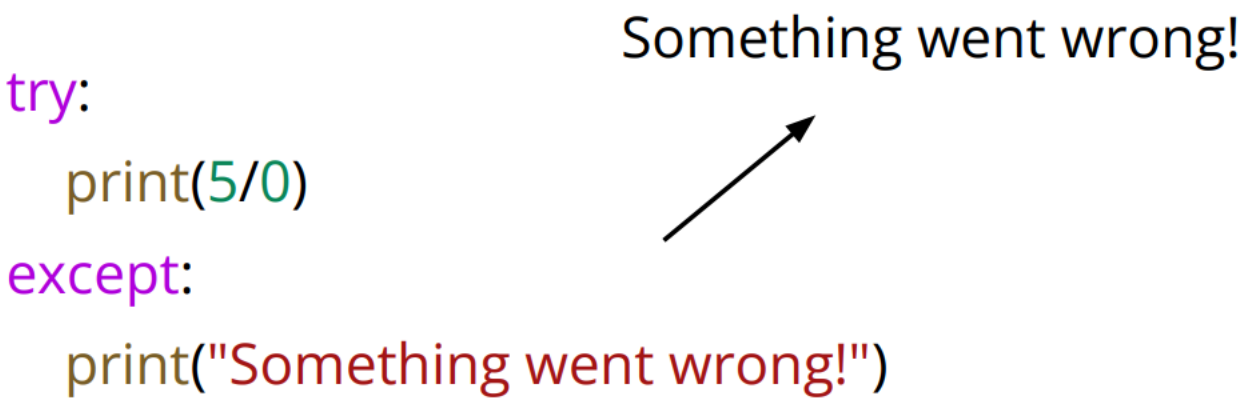
Select alternative (file, read, with, open) Select alternative (open(filename), filename, with(filename), read(filename)) Select alternative (to f:, as f:, with f:, from f:)

- for line in f:
  - Select alternative (line = line.strip(), line = f.readline(), line = line.split(), line = f.read()):
    - print(len(line))

10 – exceptions

try / except

Gir oss en trygg måte å håndtere kræsj på. Hvis koden i try-blokken fører til feilmelding, vil error-blokken 'catche' koden istedenfor å kræsje



try / except – ValueError

Eksempel: få heltall som input fra bruker. Om vi gjør det slik vil det føre til error:

```
num = int(input("Enter a number: "))
```

```
Traceback (most recent call last):
  File "/Users/jakobalexandersen/inf161/lab-9/haha.py", line 1, in <module>
    num = int(input("Enter a number: "))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: '2.1'
```

```
while True:
    try:
        num = int(input("Enter a number: "))
        print(f'the chosen number is {num}')
        break
    except ValueError:
        print("That's not a number!")
```

```
Enter a number: 1.2
That's not a number!
Enter a number: 1
the chosen number is 1
```

Andre exceptions

- KeyError
  - Når man slår opp i en dict og nøkkelen ikke finnes
- IndexError
  - Om man slår opp et element og indeksen er utfor listens lengde
- FileNotFoundError
  - ???

3 Fill in the name of an Exception-*type* such that the program prints "Unknown location". The t*ype* you choose must be as specific as possible:
Use e.g. "ZeroDivisionError", not just "Exception", since that can also catch other excpetions

```
location = {
  'Rick' : 'C-137',
  'Morty' : 'C500-A',
}
try:
```

- who = 'Summer'
- place = location[who]
- print(f"{who} is in {place}")

except [ ] :

- print("Unknown location")

except:

- print("Other error")

11 – moduler

Egne moduler

main.py

```python
from my_program import generate_random_number

print(generate_random_number())
```

my_program.py > ...

```python
import random

def generate_random_number():
    return random.randint(0, 100)
```

Egne moduler

Det kan være problematisk å ha kode i det gobale skopet når man lager en modul. Dette er fordi all koden i modulen vil kjøre om man ikke gjør slik:

```
def hello():
    print("Hello!")

hello() # testing my module
```

✗

```
def hello():
    print("Hello!")

if __name__ == "__main__":
    hello() # testing my module
```

✓

Standardbiblioteket

Flere verktøy som man kan bruke i python.

Random, math, datetime, csv, pathlib, etc.

```python
import math

r = 4
print(math.pi*r**2)
```

```python
from random import choice

print(choice(["red", "green", "blue"]))
```

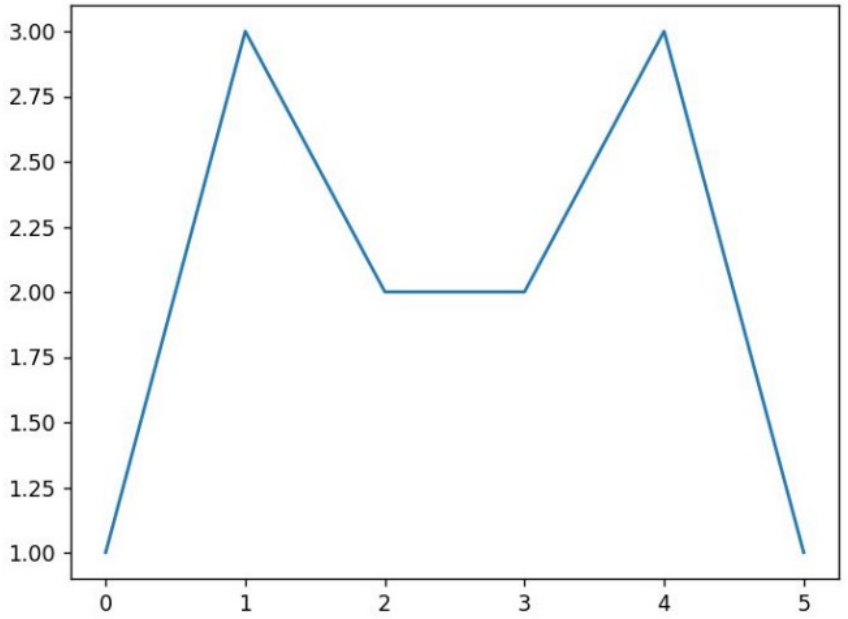
Eksterne pakker

Pakker som ikke inkluderes i standardbiblioteket må installeres. Da bruker vi pip (eller pip3, eller python3.xx –m pip install …)

Plotting av grafer

Man trenger en liste av x'er og en liste av y'er (med like lengder)

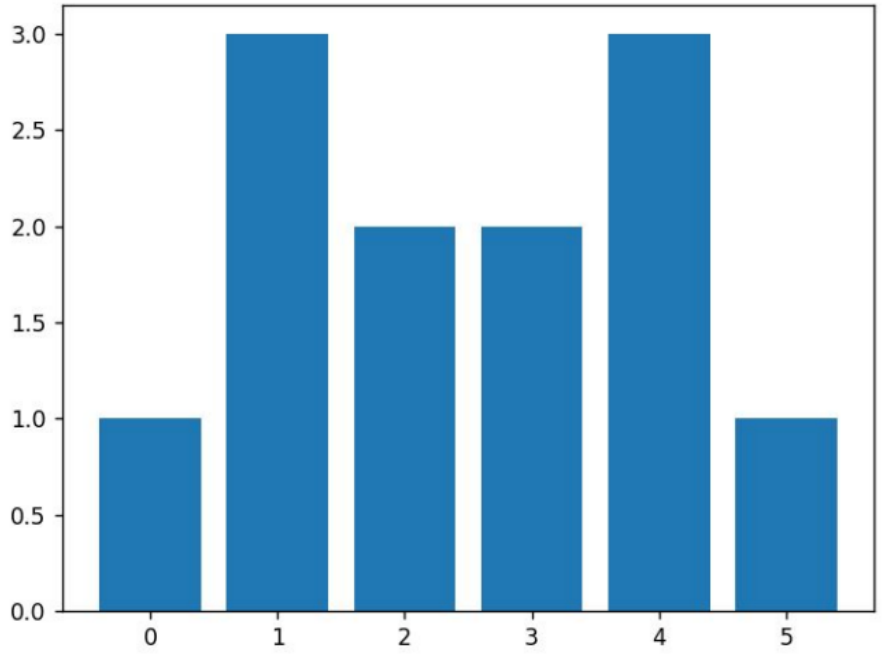
```
import matplotlib.pyplot as plt

xs = [0, 1, 2, 3, 4, 5]
ys = [1, 3, 2, 2, 3, 1]

plt.plot(xs, ys)

plt.show()
```



Lage søylediagram

Nesten samme som en linje, men .bar() istedet for .plot()

```
import matplotlib.pyplot as plt

xs = [0, 1, 2, 3, 4, 5]
ys = [1, 3, 2, 2, 3, 1]

plt.bar(xs, ys)

plt.show()
```

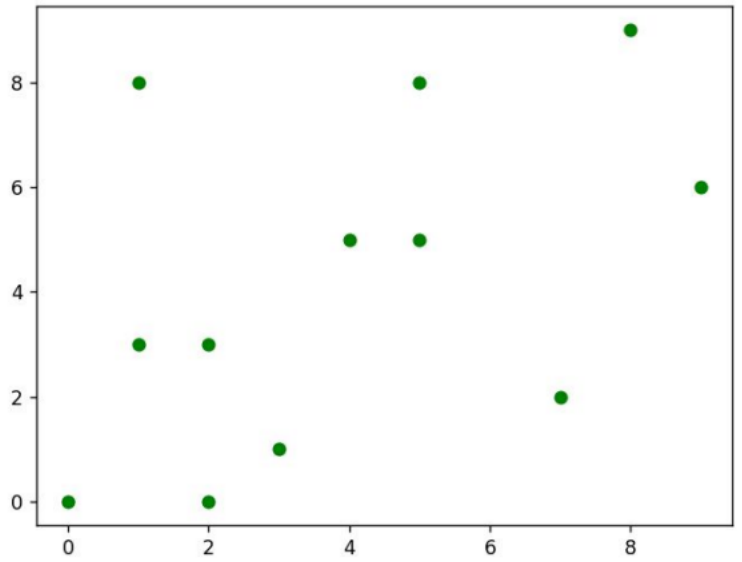


Lage scatter plot

For-løkke med zip(), og bruke .scatter() metoden

Også mulig å bruke .plot() slik som vist i kommentaren

```
import matplotlib.pyplot as plt

xs = [7, 1, 9, 2, 5, 1, 3, 5, 8, 2, 0, 4]
ys = [2, 3, 6, 0, 5, 8, 1, 8, 9, 3, 0, 5]

for x, y in zip(xs, ys):
    plt.scatter(x, y, color="green")
    # plt.plot(x, y, "o", color="green")

plt.show()
```

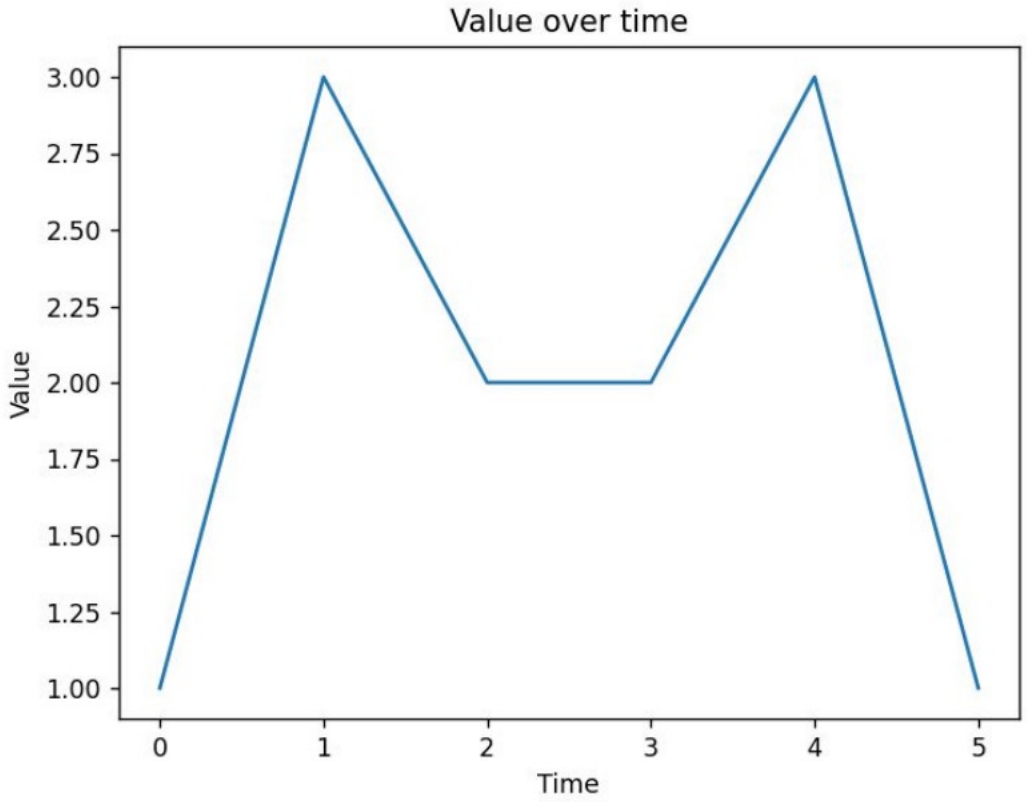


Labels

Metoder som .title(), xlabel() og ylabel() lar oss merke viktig informasjon

Gjøres før .show()

```
plt.title("Value over time")
plt.xlabel("Time")
plt.ylabel("Value")
```



12 – problemløsning

Kommenter koden din!

- Bedre enn ingenting
- Man kan få poeng man ellers ikke ville fått
- Sensor kan enklere forstå hvordan du har tenkt
- Lettere å ha oversikt over egen kode

Bruk gode ressurser

- Finn, gjør klar, og bruk gode ressurser på eksamen
- 6 tosidige A4-ark
- Kursnotater er tilgjengelige!
- Plagiat er fusk! Vær flink på å oppgi kilder

Gjennomgang av eksamensoppgaver

Lykke til på eksamen!