

Presentasjon 2

Funksjoner

- Gjenbrukbar blokk med kode
- Hvorfor?
 - Unngå repetisjon
 - Bedre leselighet
 - Enklere testing

Grunnleggende syntaks

- Definere funksjonen
- Kalle funksjonen

```
1  def MyFunction():
2      print("Hello World")
3
4
5  MyFunction()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\haron\OneDrive\Studie\inf100\lab3> & C:\Users\haron\AppData\Local\Programs\Python\Python313\python.exe c:/Users/haron/OneDrive/Studie/inf100/lab3/presentasjon.py

Hello World

PS C:\Users\haron\OneDrive\Studie\inf100\lab3> █

Parametere og argumenter

- Parameter -> midlertidig variabel
- Argument -> verdi inn

```
6 numbers = [1,4,6,3]
7
8 def multiply_list_elements(myList):
9     for i, num in enumerate(myList):
10         myList[i] = num * 2
11
12 multiply_list_elements(numbers)
13
14 print(myList)      "myList" is not defined
15 print(numbers)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\haron\OneDrive\Studie\inf100\lab3> & C:\U
thon.exe c:/Users/haron/OneDrive/Studie/inf100/lab3/p
[2, 8, 12, 6]
PS C:\Users\haron\OneDrive\Studie\inf100\lab3> |
```

Returverdi

- Ikke alltid en returverdi
- Verdi eller objekt som blir returnert

```
def MyFunction():  
    print("Hello World")
```

```
#MyFunction()
```

```
numbers = [1,4,6,3]
```

```
def multiply_list_elements(myList):  
    for i, num in enumerate(myList):  
        myList[i] = num * 2  
    return myList
```

```
result = multiply_list_elements(numbers)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
PS C:\Users\haron\OneDrive\Studie\inf100\lab3> & C:\Users\haron\OneDrive\Studie\inf100\lab3\present  
thon.exe c:/Users/haron/OneDrive/Studie/inf100/lab3/present  
[2, 8, 12, 6]  
PS C:\Users\haron\OneDrive\Studie\inf100\lab3> █
```

Lokale og globale variabler

```
presentasjon.py > ...
1  def MyFunction():
2      print("Hello World")
3
4  #MyFunction()
5
6  numbers = [1,4,6,3]
7
8  def multiply_list_elements(myList):
9      for i, num in enumerate(myList):
10         myList[i] = num * 2
11         return myList
12
13  result = multiply_list_elements(numbers)
14
15  print(result)
16  print(myList)      "myList" is not defined
17
```

Lesbarhet

- Lettere leselig
- Debugging
- Større prosjekter

```
eksempel.py > ...  
1  from presentasjon import multiply_list_elements  
2  
3  list_1 = [1,5,6,1,6,8]  
4  list_2 = [6,1,3,8]  
5  list_3 = [6,8,1,3,4]  
6  
7  print(f'List 1: {multiply_list_elements(list_1)}\n'  
8      f'List 2: {multiply_list_elements(list_2)}\n'  
9      f'List 3: {multiply_list_elements(list_3)}\n')
```

```
presentasjon.py > ...  
1  def multiply_list_elements(myList):  
2      for i, num in enumerate(myList):  
3          myList[i] = num * 2  
4      return myList  
5  
6  def main():  
7      numbers = [1,4,6,3]  
8      result = multiply_list_elements(numbers)  
9      print(result)  
10  
11  if __name__ == "__main__":  
12      main()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
  
PS C:\Users\haron\OneDrive\Studie\inf100\lab3> 8  
thon.exe c:/Users/haron/OneDrive/Studie/inf100/I  
List 1: [2, 10, 12, 2, 12, 16]  
List 2: [12, 2, 6, 16]  
List 3: [12, 16, 2, 6, 8]  
  
PS C:\Users\haron\OneDrive\Studie\inf100\lab3>
```

Obs! Mutasjon og destruktivitet

```
presentasjon.py > main
1  def multiply_list_elements(myList):
2      for i, num in enumerate(myList):
3          myList[i] = num * 2
4      return myList
5
6  def multiply_list_elements_nondestructive(myList):
7      newList = []
8      for num in myList:
9          newList.append( num * 2)
10     return newList
11
12  def main():
13     numbers = [1,4,6,3]
14
15     result = multiply_list_elements_nondestructive(numbers)
16     print(f'Ikke destruktiv:\n{result}')
17     print(f'{numbers}\n')
18
19     result = multiply_list_elements(numbers)
20     print(f'Destruktiv:\n{result}')
21     print(numbers)
22
23  if __name__ == "__main__":
24     main()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\haron\OneDrive\Studie\inf100\lab3> & C:\Us
udie/inf100/lab3/presentasjon.py
Ikke destruktiv:
[2, 8, 12, 6]
[1, 4, 6, 3]

Destruktiv:
[2, 8, 12, 6]
[2, 8, 12, 6]
PS C:\Users\haron\OneDrive\Studie\inf100\lab3> 
```


Løkker

- Gjentar kode
- Mindre repetisjon
- Større datamengder
- To typer løkker i Python....

For-løkke

- Itererer over en samling
 - Liste, streng, range, osv.
- Løkker kan nøstes
- Syntaks feil om ikke-itererbart

```
for_loops.py > ...
1  MyNumList = [1,5,6,6]
2  MyStringList = ["for","loops","in","python"]
3
4  sum_of_numbers = 0
5  for num in MyNumList:
6      sum_of_numbers += num
7  print(sum_of_numbers)
8
9  sum_of_strings = ""
10 for string in MyStringList:
11     for char in string:
12         sum_of_strings += char
13         sum_of_strings += " "
14 print(sum_of_strings)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2> & C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2\for_loops.py
```

```
18
for loops in python
```

```
PS C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2> █
```

While-løkke

- Kjører så lenge betingelsen er sann
 - Uendelig løkker, spill-loop, input-sjekk
- Syntaksfeil om betingelsen ikke kan evalueres til True/False
 - Alle objekter er truthy/falsy

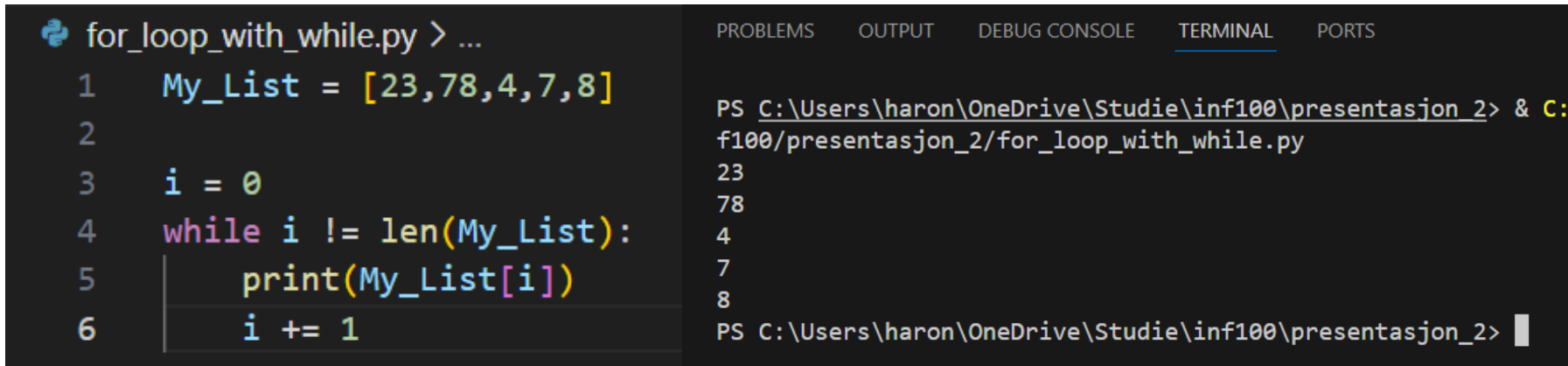
```
while_loop_truthy-falsy.py > ...
1  myList = [1,4,2,5,6,6,3,2]
2
3  while myList:
4      print(MyList.pop())
```

```
while_loops > ...
1  secret = "python"
2  guess = ""
3
4  print("Gjett ordet! (skriv 'quit' for å avslutte)")
5
6  while guess != secret:
7      guess = input("Skriv gjetning: ")
8
9      if guess == "quit":
10         print("Du avsluttet spillet.")
11         break
12     elif guess == secret:
13         print("Riktig! Du vant!")
14     else:
15         print("Feil, prøv igjen...")
```

```
PS C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2>
f100/presentasjon_2/while_loop_truthy-falsy.py
2
3
6
6
5
2
4
1
PS C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2>
```

Når skal man bruke hvilken?

- For-løkke hvis du skal iterere over en mengde som er kjent
- While-løkke hvis du ikke vet hvor lenge løkken skal gjentas
 - While-løkker kan kopiere For-løkker



The screenshot shows a code editor with a file named `for_loop_with_while.py`. The code is as follows:

```
1 My_List = [23,78,4,7,8]
2
3 i = 0
4 while i != len(My_List):
5     print(My_List[i])
6     i += 1
```

On the right, the `TERMINAL` tab shows the command to run the script and its output:

```
PS C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2> & C:\Python39\python.exe C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2\for_loop_with_while.py
23
78
4
7
8
PS C:\Users\haron\OneDrive\Studie\inf100\presentasjon_2>
```

Break/Continue

- Break avslutter løkke tidlig
 - Unngå uønskete uendelige løkker eller unødvendig gjenta løkke
- Continue går til neste runde av løkken

```
break.py > ...  
1  tall = [1, 3, 7, 9, 11]  
2  
3  for n in tall:  
4      if n == 7:  
5          print("Fant 7!")  
6          break  
7      print("Sjekker", n)
```

```
continue.py > ...  
1  for i in range(6):  
2      if i % 2 == 0:  
3          continue  
4      print(i)
```

Uendelige løkker

- While-løkke som alltid er True
- Hvis uønsket
- Ønskelige tilfeller:
 - Spill
 - Server
 - Logging
 - Ekstern hendelse som avslutter

