Python i praksis - Af Henrik Sterner (hst@nextkbh.dk)

- Introduktion til Python
- Introduktion til variabler og datatyper i Python
- Brugerinput i Python
- Strengformatering i Python
- Betinget udførelse af kode i Python
- Løkker i Python
- Liste comprehension i Python
- Funktioner i Python

Om brugen af disse slides

- Disse slides forsøger at eksemplifere en lang række af de vigtige begreber i Python
- De må på ingen måde kopieres uden tilladelse fra Henrik Sterner
- De er lavet i markdown og kan derfor nemt konverteres til andre formater
- Ved brug af Visual Studio Code kan de konverteres til HTML, PDF, PowerPoint, Word, LaTeX og mange andre formater
- Slides er tilgængelige på github.com/henriksterner/IntelligenteSystemer/

Konvertere slides til andre formater

- Fra markdown til pdf ved brug af pandoc:

```
pandoc -t beamer -o slidespython.pdf slidespython.md
```

- Fra markdown til word ved brug af pandoc:

```
pandoc -t docx -o slidespython.docx slidespython.md
```

Øvelser og software brugt

Det er ekstremt vigtigt at lave en masse øvelser. Da det træner brugen af de grundlæggende strukturer. Programmering er en praktisk disciplin - et håndværk. Og man bliver ikke god til et håndværk uden at øve sig.

Øvelserne er lavet så de kan løses i Python ved brug af eksempelvis Jupyter Notebook eller Google Colab.

Undlad at få hjælp fra AI m.m. Det lærer du ikke noget af.

Hvad er Google Colab?

- Google Colab er et interaktivt programmeringsmiljø
- Google Colab er en webapplikation
- Google Colab er gratis
- Google Colab er udviklet i Python
- Google Colab kan bruges til mange forskellige programmeringssprog
- Google colab tilgås via en Google konto og Google Drev
- Alle filer gemmes i Google Drev
- Google Colab kan bruges til at træne maskinlæring og deep learning modeller
- Google Colab kan bruges til at træne modeller på GPU og TPU
- Adressen er: https://colab.research.google.com/

Hvad er Jupyter Notebook?

- Jupyter Notebook er et interaktivt programmeringsmiljø
- Jupyter Notebook er en webapplikation
- Jupyter Notebook er gratis
- Jupyter Notebook er udviklet i Python
- Jupyter Notebook kan bruges til mange forskellige programmeringssprog
- Jupyter Notebook kan tilgås lokalt på computeren
- Jupyter Notebook kan installeres via Anaconda og Visual Studio Code
- Anaconda kan downloades fra: https://www.anaconda.com/products/individual
- Visual Studio Code kan downloades fra: https://code.visualstudio.com/

Introduktion til Python

- Python: Et populært og læsbar programmeringssprog
- Bruges bredt indenfor udvikling, automatisering og dataanalyse
- Udtales “Pai-thon”
- Navngivet efter Monty Python
- Skabt af Guido van Rossum i 1991

Hvad er Python?

- Python er et højniveausprog
- Python er et objektorienteret sprog
- Python er et dynamisk typet sprog
- Python er et open source sprog
- Python er et sprog med mange biblioteker

Datascience biblioteker med Python

Vi vender tilbage til disse biblioteker senere i kurset: - Numpy: Matematik og videnskabelige beregninger - Pandas: Dataanalyse og datahåndtering - Matplotlib: Visualisering af data - Scikit-learn: Maskinlæring - TensorFlow: Maskinlæring og deep learning - PyTorch: Maskinlæring og deep learning

Overblik over hvad vi skal igennem

- Introduktion til variabler i Python
- Datatyper i Python
- Brugerinput i Python
- Strengformatering i Python
- Betinget udførelse af kode i Python
- Løkker i Python
- Funktioner i Python
- Objektorienteret programmering i Python
- Fejlhåndtering i Python
- Filhåndtering i Python

Introduktion til variabler og datatyper i Python

- Variabler gemmer værdier til brug i programmet
- Navngivne lagerpladser til forskellige typer af data
- Python bruger dynamisk typetildeling
- Variabler behøver ikke erklæres med en bestemt type
- Data i variabler kan ændres undervejs i programmet

Variabelnavne

```
navn = "Alice"
alder = 30
point_1 = 95.5
er_student = True
```

Navngivningsregler for variabler

- Variabelnavne kan indeholde bogstaver, tal og understregning
- Variabelnavne må ikke starte med et tal
- Variabelnavne må ikke indeholde mellemrum
- Variabelnavne må ikke indeholde specialtegn
- Variabelnavne må ikke være Python nøgleord
- Variabelnavne bør være sigende
- Variabelnavne kan være på engelsk

Datatyper i Python

```
tekst = "Hej verden"
heltal = 42
decimaltal = 3.14
sandt_eller_falsk = True
```

Streng (Tekst)

```
besked = "Velkommen til Python"
underbesked = besked[8:12]
længde = len(besked)
```

Heltal

```
alder = 25
højde = -160
antal_børn = 3
```

Sandt eller Falsk (Boolean)

```
er_solopgang = True
er_nedbør = False
```

Listetyper

```
farver = ["rød", "grøn", "blå"]
tal = [1, 2, 3, 4, 5]
blandet = ["æble", 3, True]
```

Tupletyper

```
koordinater = (3, 7)
dataræk ke = (1, "tekst", True)
```

Dictionarietyper

```
person = {
    "navn": "Alice",
    "alder": 30,
    "er_student": True
}
navn = person["navn"]
```

Variabelskift

```
a = 5
b = 7
a, b = b, a
print("a:", a)  # Resultat: 7
print("b:", b)  # Resultat: 5
```

Globale og Lokale Variabel

```
global_var = 10
if global_var > 5:
    lokal_var = 5
    print(global_var)
    print(lokal_var)
```

Vi vender snart tilbage til if-betingelser

Typekonvertering

```
alder = 25
alder_tekst = str(alder)
pris = "9.99"
pris_decimal = float(pris)
```

Indbyggede Funktioner til Typer

```
længde = len("Hej verden")
maksimum = max([4, 7, 2, 9])
minimum = min([4, 7, 2, 9])
```

Variabler som Referencer

```
liste_a = [1, 2, 3]
liste_b = liste_a
liste_a.append(4)
print(liste_b)  # Resultat: [1, 2, 3, 4]
```

Identitet og Lighed

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)  # Resultat: False
print(a == b)  # Resultat: True
```

Typekontrol med isinstance()

```
x = 5
if isinstance(x, int):
    print("x er et heltal")
else:
    print("x er ikke et heltal")
```

Hvad er brugerinput i Python?

Brugerinput er: - Input fra brugeren - Input fra tastaturet - Input fra en fil - Input fra en database - Input fra en sensor - Input fra en anden computer - Input fra en anden enhed Vi kan bruge kommadoen input() til at få brugerinput i Python

Brugerinput med input()

```python
navn = input("Hvad hedder du? ")
alder = int(input("Hvor gammel er du? "))
```

Brugerinput bmi-beregner

```
højde = float(input("Hvor høj er du i meter? "))
vægt = float(input("Hvor meget vejer du i kg? "))
bmi = vægt / (højde ** 2)
print("Din BMI er: " + str(bmi))
```

Brugerinput andengradsligning

```
a = float(input("Indtast a: "))
b = float(input("Indtast b: "))
c = float(input("Indtast c: "))
d = b ** 2 - 4 * a * c
if d < 0:
    print("Ligningen har ingen løsninger")
elif d == 0:
    x = -b / (2 * a)
    print("Ligningen har en løsning: " + str(x))
else:
    x1 = (-b + d ** 0.5) / (2 * a)
    x2 = (-b - d ** 0.5) / (2 * a)
    print("Ligningen har to løsninger: " + str(x1) + " og "
```

Vi vender tilbage til if-else om lidt

Brugerinput med Fejlhåndtering

Måske brugeren taster forkert eller indtaster noget ugyldigt. Det skal vi håndtere.

```
while True:
    try:
        alder = int(input("Hvor gammel er du? "))
        break
    except ValueError:
        print("Indtast venligst et heltal")
```

Brugerinput med tal

```
while True:
    alder = input("Hvor gammel er du? ")
    if alder.isdigit():
        alder = int(alder)
        break
    else:
        print("Indtast venligst et heltal")
```

Strenge i Python

- Strenge er en sekvens af tegn
- Strenge er omgivet af anførselstegn
- Strenge kan indeholde bogstaver, tal og specialtegn
- Strenge kan indeholde et hvilket som helst tegn
- Strenge kan indeholde et hvilket som helst antal tegn
- Strenge er en datatype i Python
- Strenge er arrays af tegn

Strenge med anførselstegn

Strenge som tekst:

```
tekst = "Hej verden"
```

Strenge med tal og tekst:

```
tekst = "Hej verden 123"
```

Strengformatering

```
navn = "Alice"
alder = 30
print("Hej, jeg hedder " + navn + " og er " + str(alder) +
print("Hej, jeg hedder %s og er %d år gammel" % (navn, alde
print("Hej, jeg hedder {} og er {} år gammel".format(navn,
print(f"Hej, jeg hedder {navn} og er {alder} år gammel")
```

Metoder på strenge

```
besked = "Hej verden"
print(besked.upper())  # Resultat: HEJ VERDEN
print(besked.lower())  # Resultat: hej verden
print(besked.capitalize())  # Resultat: Hej verden
print(besked.replace("Hej", "Hello"))  # Resultat: Hello v
```

Flere metoder på strenge

```
besked = "Hej verden"
print(besked.startswith("Hej"))  # Resultat: True
print(besked.endswith("Hej"))  # Resultat: False
print(besked.find("verden"))  # Resultat: 4
print(besked.find("ikke"))  # Resultat: -1
```

Brugen af split og join

```
besked = "Hej verden"
ord = besked.split(" ")
print(ord)  # Resultat: ["Hej", "verden"]
ny_besked = " ".join(ord)
print(ny_besked)  # Resultat: Hej verden
```

Brugen af strip

```
besked = "   Hej verden   "
print(besked.strip())  # Resultat: Hej verden
```

Brugen af format

```
navn = "Alice"
alder = 30
print("Hej, jeg hedder {} og er {} år gammel".format(navn,
print("Hej, jeg hedder {0} og er {1} år gammel".format(navn
print("Hej, jeg hedder {navn} og er {alder} år gammel".form
print(f"Hej, jeg hedder {navn} og er {alder} år gammel")
```

Brugen af format med tal

```
tal = 3.14159265359
print("{:.2f}".format(tal))  # Resultat: 3.14
print("{:.4f}".format(tal))  # Resultat: 3.1416
print("{:e}".format(tal))  # Resultat: 3.141593e+00
print("{:d}".format(42))  # Resultat: 42
print("{:b}".format(42))  # Resultat: 101010
print("{:x}".format(42))  # Resultat: 2a
```

Betinget udførsel af kode i Python

- Betinget udførsel også kaldet selektion/forgrening
- Kode udføres kun hvis en eller flere betingelser er opfyldt
- Vi bruger if, elif og else til betinget udførsel af kode

Opbygning af if-sætninger

```
if betingelse:
    instruktioner
```

Betingelse skal evalueres til sand eller falsk. Instruktioner udføres kun hvis betingelsen er sand.

Opbygning af if-else-sætninger

```
if betingelse:
    instruktionerSand
else:
    instruktionerFalsk
```

Betingelse skal evalueres til sand eller falsk. InstruktionerSand udføres hvis betingelsen er sand. InstruktionerFalsk udføres hvis betingelsen er falsk.

Opbygning af if-elif-else-sætninger

```
if betingelse1:
    instruktioner1
elif betingelse2:
    instruktioner2
else:
    instruktioner3
```

Betingelse1 skal evalueres til sand eller falsk. Instruktioner1 udføres hvis betingelse1 er sand. Betingelse2 skal evalueres til sand eller falsk. Instruktioner2 udføres hvis betingelse2 er sand. Instruktioner3 udføres hvis betingelse1 og betingelse2 er falsk.

Betinget udførelse af kode

```
if alder >= 18:
    print("Du er myndig")
else:
    print("Du er ikke myndig")
```

Betinget udførelse af kode med elif

```
if alder < 0:
    print("Ugyldig alder")
elif alder < 18:
    print("Du er ikke myndig")
else:
    print("Du er myndig")
```

Betinget udførelse af kode med flere betingelser

```
if alder < 0:
    print("Ugyldig alder")
elif alder < 18:
    print("Du er ikke myndig")
elif alder < 67:
    print("Du er i arbejde")
else:
    print("Du er på pension")
```

Betinget udførsel med flere instruktioner

```
if alder < 0:
    print("Ugyldig alder")
elif alder < 18:
    print("Du er ikke myndig")
    print("Du må ikke købe alkohol")
elif alder < 67:
    print("Du er i arbejde")
    print("Du må gerne købe alkohol")
else:
    print("Du er på pension")
    print("Du må gerne købe alkohol")
```

Eksempel på if-sætninger i if-sætninger

```
if alder < 0:
    print("Ugyldig alder")
elif alder < 18:
    print("Du er ikke myndig")
    if alder < 15:
        print("Du må ikke køre bil")
    else:
        print("Du må køre knallert")
elif alder < 67:
    print("Du er i arbejde")
    print("Du må gerne køre bil")
else:
    print("Du er på pension")
    print("Du må gerne køre bil")
```

BMI beregner med if-betingelse i if-betingelse

```
højde = float(input("Hvor høj er du i meter? "))
vægt = float(input("Hvor meget vejer du i kg? "))
bmi = vægt / (højde ** 2)
if bmi < 18.5:
    print("Du er undervægtig")
elif bmi < 25:
    print("Du er normalvægtig")
elif bmi < 30:
    print("Du er overvægtig")
else:
    print("Du er svært overvægtig")
    if bmi > 40:
        print("Du er i livsfare")
```

Intro til løkker

- Løkker bruges til at gentage kode
- Løkker bruges til at udføre kode et bestemt antal gange
- Løkker bruges til at udføre kode indtil en betingelse er opfyldt
- Løkker bruges til at udføre kode på hvert element i en liste
- Løkker bruges til at udføre kode på hvert bogstav i en streng
- Løkker bruges til at udføre kode på hvert bogstav i en fil
- Mange andre ting
- Der findes to slags løkker: while-løkker og for-løkker

While løkkers opbygning

```
while betingelse:
    instruktioner
```

Betingelse skal evalueres til sand eller falsk. Instruktioner udføres så længe betingelsen er sand.

For løkkers opbygning

```
for variabel in sekvens:
    instruktioner
```

Variabel er en variabel der bruges til at gemme hvert element i sekvensen. Sekvens er en liste, tupel, streng eller anden sekvens. Instruktioner udføres for hvert element i sekvensen.

While-løkker

```
i = 0
while i < 10:
    print(i)
    i += 1
```

For-løkker

```
for i in range(10):
    print(i)
```

For-løkker med liste

```
farver = ["rød", "grøn", "blå"]
for farve in farver:
    print(farve)
```

For-løkker med streng

```
besked = "Hej verden"
for bogstav in besked:
    print(bogstav)
```

Løkker indeni løkker

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

While løkker indeni while løkker

```
i = 0
while i < 3:
    j = 0
    while j < 3:
        print(i, j)
        j += 1
    i += 1
```

While løkker indeni for løkker

```
for i in range(3):
    j = 0
    while j < 3:
        print(i, j)
        j += 1
```

For løkker indeni while løkker

```
i = 0
while i < 3:
    for j in range(3):
        print(i, j)
    i += 1
```

Løkker med break

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

i = 0
while i < 10:
    print(i)
    if i == 5:
        break
    i += 1
```

Løkker med continue

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

i = 0
while i < 10:
    i += 1
    if i == 5:
        continue
    print(i)
```

Løkker med else

```
for i in range(10):
    print(i)
else:
    print("Færdig")

i = 0
while i < 10:
    print(i)
    i += 1
else:
    print("Færdig")
```

Løkker med else og break

```
for i in range(10):
    print(i)
    if i == 5:
        break
else:
    print("Færdig")

i = 0
while i < 10:
    print(i)
    if i == 5:
        break
    i += 1
else:
    print("Færdig")
```

Løkker med else og continue

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

i = 0
while i < 10:
    i += 1
    if i == 5:
        continue
    print(i)
else:
    print("Færdig")
```

Liste comprehension i Python

- Liste comprehension er en smart måde at lave lister på
- Smart i den forstand at det er kortere og hurtigere
- Det er hurtigere fordi det er optimeret i Python
- Det er kortere fordi det er mere kompakt
- Men det er ikke altid nemmere at læse

Listecomprehension med for løkker

```
liste = [i for i in range(10)]
print(liste)  # Resultat: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listecomprehension med for løkker og betingelse

```
liste = [i for i in range(10) if i % 2 == 0]
print(liste)  # Resultat: [0, 2, 4, 6, 8]
```

Listecomprehension med for løkker og flere løkker

```
liste = [(i, j) for i in range(3) for j in range(3)]
print(liste)  # Resultat: [(0, 0), (0, 1), (0, 2), (1, 0),
```

Listecomprehension med for løkker og flere løkker og betingelse

```
liste = [(i, j) for i in range(3) for j in range(3) if i !=
print(liste)  # Resultat: [(0, 1), (0, 2), (1, 0), (1, 2),
```

Funktioner i Python

- Funktioner er en samling af kode/instruktioenr
- Funktioner kan udføre en bestemt opgave
- Det er smart at bruge funktioner fordi det gør koden mere overskuelig
- Det er smart at bruge funktioner fordi det gør koden mere genbrugelig, mere testbar og mere fejlsikker

Funktioners opbygning

- Funtioner defineres på formen:

```
def navn(argumenter):
    instruktioner
```

De behøver ikke at have argumenter eller returværdier. - Funktioner kaldes på formen:

```
navn(argumenter)
```

Funktioner uden argumenter

```
def hilsen():
    print("Hej verden")
hilsen()  # Resultat: Hej verden
```

Funktioner med argumenter

```
def hilsen(navn):
    print("Hej " + navn)
hilsen("Alice")  # Resultat: Hej Alice
hilsen("Bob")  # Resultat: Hej Bob
```

Funktioner med flere argumenter

```
def hilsen(navn, alder):
    print("Hej " + navn + ", du er " + str(alder) + " år ga
hilsen("Alice", 30)  # Resultat: Hej Alice, du er 30 år gam
hilsen("Bob", 25)  # Resultat: Hej Bob, du er 25 år gammel
```

Funktioner med returværdi

```
def hilsen(navn):
    return "Hej " + navn
hilsetekst = hilsen("Alice")
print(hilsetekst)  # Resultat: Hej Alice
```

Funktioner med returværdi og flere argumenter

```
def hilsen(navn, alder):
    return "Hej " + navn + ", du er " + str(alder) + " år g
hilsentekst = hilsen("Alice", 30)
print(hilsentekst)  # Resultat: Hej Alice, du er 30 år gam
```

Funktioner med returværdi og flere argumenter bestående af flere typer

```
def hilsen(navn, alder, er_student):
    tekst = "Hej " + navn + ", du er " + str(alder) + " år
    if er_student:
        tekst += " og er studerende"
    return tekst
hilsentekst = hilsen("Alice", 30, True)
print(hilsentekst)  # Resultat: Hej Alice, du er 30 år gam
```

Funktioner med for-løkker

```
def hilsen(navn, antal):
    for i in range(antal):
        print("Hej " + navn)
hilsen("Alice", 3)
```

Funktioner med while-løkker

```
def hilsen(navn, antal):
    i = 0
    while i < antal:
        print("Hej " + navn)
        i += 1
hilsen("Alice", 3)
```

Funktioner med for-løkker og returværdi

```
def hilsen(navn, antal):
    tekst = ""
    for i in range(antal):
        tekst += "Hej " + navn + "\n"
    return tekst
hilsentekst = hilsen("Alice", 3)
print(hilsentekst)
```

Funktioner med while-løkker og returværdi

```
def hilsen(navn, antal):
    tekst = ""
    i = 0
    while i < antal:
        tekst += "Hej " + navn + "\n"
        i += 1
    return tekst
hilsentekst = hilsen("Alice", 3)
print(hilsentekst)
```

Funktioner med for-løkker og break

```
def hilsen(navn, antal):
    for i in range(antal):
        print("Hej " + navn)
        if i == 1:
            break
hilsen("Alice", 3)
```