# Python basics

AAMS - DLE

November 23, 2025

# Table of Contents

# What is Python?

- Højniveau programmeringssprog
- Let at lære og læse
- Dynamisk typet
- Stort bibliotek af moduler
- Tværplatform (Windows, Linux, MacOS)
- Bruges til webudvikling, dataanalyse, maskinlæring, automation, spiludvikling, m.m.
- Gratis og open source
- Populært i både industri og akademia
- Stadig under aktiv udvikling
- Udviklet af Guido van Rossum i slutningen af 1980'erne

# Choosing an Editor

- IDLE: Standard editor, enkel og let at bruge
- VS Code: Populær, mange udvidelser, god til større projekter (vores valg)
- PyCharm: Kraftfuld IDE, velegnet til professionelle udviklere
- Jupyter Notebook: Ideel til dataanalyse og videnskabelig computing
- Sublime Text: Hurtig og letvægts editor med mange plugins
- Atom: Open source editor med mange tilpasningsmuligheder

# Basic Syntax

Syntaks er læren om, hvordan ord sættes sammen for at danne meningsfulde sætninger og andre sproglige enheder dvs. I programmering refererer syntaks til de regler og strukturer, der bestemmer, hvordan kode skal skrives for at blive forstået af en computer.

- Kommentarer: Start med #
- Variabler: Ingen deklaration nødvendig
- Indrykning: Blokke defineres ved indrykning (typisk 4 mellemrum eller en tab)
- Print: Brug `print()` funktionen
- Datatyper: int, float, str, list, dict, tuple, set
- Kontrolstrukturer: if, for, while
- Funktioner: Defineres med `def` nøgleordet
- Classer: Defineres med `class` nøgleordet

# Python Code Example

```
# Dette er en kommentar
x = 5
y = 10
z = x + y
print(z)
```

# Variables I

- Bruges til at gemme data
- Ingen deklaration nødvendig
- Navngivningsregler:
  - Må ikke starte med et tal
  - Må ikke indeholde mellemrum eller specialtegn (undtagen underscore _)
  - Følg konventioner (f.eks. snake_case)
- Eksempler:
  - `alder = 25`
  - `navn = "John"`
  - `pris = 19.99`

# Variables II

**konventioner for variabelnavne:**

- Brug meningsfulde navne
- Brug henholdsvis:
  - Snake_case (f.eks. `min_variabel`)
  - CamelCase (f.eks. `MinVariabel`)
  - PascalCase (f.eks. `MinVariabel`)
- Undgå at bruge Python-reserverede ord (f.eks. `if`, `for`, `while`)
- Vær konsekvent i din navngivning

# Operators

- Assigneringsoperatoren (=) bruges til at tildele værdier til variabler
- Andre operatorer bruges til at udføre operationer på variabler
  - Aritmetiske operatorer: +, -, *, /,
  - Tildelingsoperatorer: =, +=, -=, =, /=
  - Sammenligningsoperatorer: ==, !=, >, <, >=, <=
  - Logiske operatorer: and, or, not
  - Eksempler:
    - `x += 5` (tilføjer 5 til x)
    - `if x > 10:` (tjekker om x er større end 10)

# Statements, Conditions, Expressions

- Statement: En komplet instruktion, der udfører en handling (f.eks. tildeling, kontrolstruktur)
- Condition: Et udtryk, der evalueres til sandt eller falsk (bruges i kontrolstrukturer)
- Expression: En kombination af værdier, variabler og operatorer, der evalueres til en enkelt værdi

extbfEksempler:

- Statement: `x = 5`
- Condition: `x > 10`
- Expression: `y = x + 2`

# Control Flow: If / Else

- Bruges til at træffe beslutninger i koden.
- Indrykning er afgørende i Python!

```
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

# Control Flow: Loops

- **For Loop:** Itererer over en sekvens (som en liste eller range).
- **While Loop:** Gentages så længe en betingelse er sand.

### For Loop

```
for i in range(5):
    print(i)
# Prints 0, 1, 2, 3, 4
```

### While Loop

```
count = 0
while count < 5:
    print(count)
    count += 1
```

# Built-in Data Types I

- int: Heltal (f.eks. 5, -3, 0)
- float: Flydende tal (f.eks. 3.14, -0.5, 2.0)
- str: Strenge (f.eks. "Hello", 'Python')
- list: Liste af elementer (f.eks. [1, 2, 3], ['a', 'b', 'c'])
- dict: Ordbog med nøgle-værdi par (f.eks. 'navn': 'John', 'alder': 25)
- tuple: Uforanderlig liste af elementer (f.eks. (1, 2, 3), ('a', 'b', 'c'))
- set: Mængde af unikke elementer (f.eks. 1, 2, 3, 'a', 'b', 'c')

# Casting and Type Conversion

- Brug indbyggede funktioner til at caste datatyper fra en type til en anden
  - `int()`, `float()`, `str()`, `list()`, `dict()`, `tuple()`, `set()`
  - Eksempler:
    - `x = int("5")` (konverterer streng til heltal)
    - `y = str(3.14)` (konverterer flydende tal til streng)

# Special Data Types I

Der er fire datatyper som kan være nyttige at kende til, men hvornår skal man bruge dem?

- List []
  - Brug en list, når du har en samling af elementer, der skal være ordnede, og som kan ændres. Dette er den mest alsidige datastruktur og er god til lister, der kan vokse eller krympe. Elementer i en liste kan duplikeres.
  - Anvendelse: Gemning af en liste over studerende, en indkøbsliste, eller en sekvens af sensorværdier over tid.
  - Eksempel: studenter = ['Anders', 'Camilla', 'Mikkel']
- Tuple ()
  - Brug en tuple, når du har en samling af elementer, der skal være ordnede, men som er uforanderlige. Fordi de ikke kan ændres, er de hukommelsesmæssigt mere effektive og hurtigere end lister, og de kan bruges som nøgler i dictionaries.
  - Anvendelse: Repræsentation af koordinater (x,y), en persons fødselsdato (år, måned, dag), eller en samling af værdier, der ikke må ændre sig.
  - Eksempel: koordinater = (56.15, 10.21)

# Special Data Types II

- Dict
  - Brug en dict (dictionary), når du har brug for at gemme data som nøgle-værdi-par. Den er optimal, når du skal slå op i data baseret på en unik nøgle, hvilket gør den meget hurtig. Nøglerne skal være unikke og uforanderlige.
  - Anvendelse: Gemning af en persons oplysninger (navn, alder, by), en ordbog med oversættelser, eller konfigurationsindstillinger.
  - Eksempel: profil = 'navn': 'Anders', 'alder': 35, 'by': 'Aarhus'
- Set
  - Brug et set, når du har en samling af elementer, der ikke er ordnede, men som skal være unikke. Sæt er meget effektive til at fjerne duplikater og til at udføre matematiske operationer som union, intersection og difference.
  - Anvendelse: At finde unikke elementer i en liste, tjekke om et element er medlem af en samling, eller at sammenligne to samlinger af data.
  - Eksempel: unikke_tal = 1, 2, 3, 4, 4 (vil resultere i 1, 2, 3, 4)

# String Manipulation I

- Strenge er uforanderlige sekvenser af tegn, der bruges til at repræsentere tekst.
- Du kan manipulere strenge ved hjælp af forskellige metoder og operationer.

# String Manipulation II

**Eksempel:**

*Upper case*

- Eksempel:

```
a = "Hello,
World!"
print(a.upper())
» HELLO, WORLD!
```

*Lower case*

- Eksempel:

```
a = "Hello,
World!"
print(a.lower())
» hello, world!
```

*Remove white space*

- Eksempel:

```
a = "Hello,
World!"
print(a.strip())
```

*Replace*

- Eksempel:

```
a = "Hello,
World!"
print(a.replace("H",
"J"))
» Jello, World!
```

*Split*

- Eksempel:

```
a = "Hello,
World!"
print(a.split(","))
» ['Hello', '
World!']
```

*Join*

- Eksempel:

```
liste = ['Hello',
'World']
print("
".join(liste))
» Hello World
```

# String Manipulation III

### Slicing

```
# Slicing
a = "Hello, World!"
print(a[0:5])  # Output: Hello
```

*Negative tal kan også anvendes!*

### Joining

```
# Joining
liste = ['Hello', 'World']
print(" ".join(liste))  # Output: Hello World
```

# Functions

- Funktioner er genanvendelige blokke af kode, der udfører en specifik opgave.
- Defineres med `def` nøgleordet efterfulgt af funktionsnavn og parenteser.
- Kan tage parametre og returnere værdier.

**Eksempel:**

```
def greet(name):
    return "Hello, " + name + "!"

print(greet("Alice"))
```

# Building a Python Script

Et Python-program (script) er typisk opbygget i følgende rækkefølge:

- (Valgfrit) Import af moduler
- (Valgfrit) Globale konstanter og konfiguration
- Funktioner
- `if __name__ == "__main__":` blok ("main"-indgang)

**Eksempel:**

```
import time  # 1) Import

WELCOME_MESSAGE = "Velkommen til mit program"  # 2) Konstant

def greet(name):  # 3) Funktion
    print(f"Hej {name}!")

if __name__ == "__main__":  # 4) Main-blok
    print(WELCOME_MESSAGE)
    user_name = input("Hvad hedder du? ")
    greet(user_name)
    time.sleep(1)
    print("Programmet er færdigt.")
```

# Modules and Libraries

- Python har et enormt økosystem af biblioteker (moduler).
- Brug `import` til at bruge kode fra andre filer eller biblioteker.
- Ingen grund til at genopfinde den dybe tallerken!

**Eksempel: Random Module**

```
import random

# Generate a random number between 1 and 10
number = random.randint(1, 10)
print(number)

# Pick a random element from a list
choices = ['Rock', 'Paper', 'Scissors']
print(random.choice(choices))
```

# Classes

- Klasser er skabeloner til at skabe objekter, der indeholder både data og funktioner.
- Defineres med `class` nøgleordet efterfulgt af klassenavn.
- Kan have attributter (data) og metoder (funktioner).

**Eksempel:**

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        return "Hello, " + self.name + "!"

p = Person("Alice", 30)
print(p.greet())
```