1. Data: Lister, tuples og dictionaries

```python
# Liste
ord = ["hej", "farvel", "goddag"]
print(ord)
```

```
['hej', 'farvel', 'goddag']
```

```python
# Tuple
farver = ("red", "green", "blue")
print(farver)
```

```
('red', 'green', 'blue')
```

```python
# Dictionary
person = {"navn": "Jeppe", "alder": 30}
print(person)
```

```
{'navn': 'Jeppe', 'alder': 30}
```

1.1 `list`

En liste i Python er en samling af elementer, som kan indeholde forskellige datatyper. Lister er mutable, hvilket betyder, at deres indhold kan ændres (tilføjes, fjernes eller ændres).

1. Oprettes med: [] eller ved hjælp af list().
2. Kan indeholde elementer af forskellig type: tal, strenge, objekter osv.
3. Elementerne kan tilgås med indeksering (0-baseret).

Egenskaber for lister:

- Mutable (kan ændres).

- Tillader dubletter.
- Bestiller elementer (har en specifik rækkefølge).

Lister er en af de mest brugte datastrukturer, da de er alsidige og lette at manipulere. Deres mutability gør dem ideelle til situationer, hvor man skal kunne ændre, tilføje eller fjerne elementer dynamisk. Anvendelser:

- Sekvensering af data: Når du har brug for at gemme en samling elementer i en bestemt rækkefølge, som kan ændres over tid.
- Søgelister: Du kan bruge lister, hvis du skal søge efter eller filtrere data, f.eks. gennem brug af loops.
- Dynamisk størrelse: Hvis du ved, at mængden af data kan ændre sig under runtime, som ved at tilføje eller fjerne elementer fra en indkøbsliste eller en samling brugere.

Use-cases:

- Gemning af dynamisk data: En liste over brugere på et website, hvor nye brugere kan registrere sig, og gamle brugere kan slettes.
- Behandling af data: En liste over opgaver i et program, som løbende kan opdateres, når opgaverne udføres.
- Resultater af API-kald: Når man får en liste med data (fx JSON-listestruktur), som man ønsker at gennemløbe og ændre.

```
# Oprettelse af en liste
min_liste = [1, "æble", 3.14, True]
print(min_liste)
```

```
[1, 'æble', 3.14, True]
```

```
# Tilføje et element
min_liste.append("ny værdi")
print(min_liste)
```

```
[1, 'æble', 3.14, True, 'ny værdi']
```

```
# Ændre et element
min_liste[0] = "banan"
print(min_liste)
```

```
['banan', 'æble', 3.14, True, 'ny værdi']
```

```
# Fjerne et element
min_liste.remove(3.14)
print(min_liste)
```

```
['banan', 'æble', True, 'ny værdi']
```

```
# Fjerne et element
min_liste.remove(min_liste[3])
print(min_liste)
```

```
['banan', 'æble', True]
```

1.2 Tuple

En tuple ligner en liste, men er immutable, hvilket betyder, at den ikke kan ændres, når den først er oprettet. Tuples bruges, når man vil sikre, at dataene ikke utilsigtet ændres.

- Oprettes med: () eller ved hjælp af tuple().
- Kan også indeholde elementer af forskellig type.
- Elementerne kan tilgås med indeksering, ligesom lister.

Egenskaber for tuples:

- Immutable (kan ikke ændres).
- Tillader dubletter.
- Bestiller elementer (har en specifik rækkefølge).

Tuples er nyttige, når du vil oprette en uforanderlig samling af elementer. Fordi tuples er immutable, bruges de ofte i situationer, hvor dataens integritet skal bevares. Anvendelser:

- Konstante data: Brug tuples til data, der ikke skal ændres, såsom geografiske koordinater, faste konfigurationer eller andre værdier, der ikke bør justeres.
- Nøgler i dictionaries: Da tuples er immutables, kan de bruges som nøgler i dictionaries (modsat lister, der ikke kan bruges som nøgler).
- Tilbagevendende funktioner: Funktionen kan returnere flere værdier i en tuple, hvilket gør det nemt at håndtere og udpakke data.

Use-cases:

- Koordinatsystem: Et system, der gemmer faste geografiske koordinater (latitude, longitude) som tuples for at sikre, at de ikke ændres.

- Returnering af flere værdier fra en funktion: Hvis du vil returnere både successtatus og resultatet af en operation.
- Faste opsætninger: I et spil kan du bruge tuples til at gemme uforanderlige data om en spillers oprindelige position eller karakteristika.

```
# Oprettelse af en tuple
min_tuple = (1, "orange", 2.71, False)

# Tilgå et element
print(min_tuple[1])
```

```
orange
```

```
# Forsøg på at ændre et element vil give en fejl
min_tuple[1] = "æble"  # Output: TypeError
```

```
TypeError: 'tuple' object does not support item assignment
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[15], line 2
      1 # Forsøg på at ændre et element vil give en fejl
----> 2 min_tuple[1] = "æble"  # Output: TypeError
TypeError: 'tuple' object does not support item assignment
```

```
# Geografiske koordinater som tuple
koordinater = (55.6761, 12.5683)  # Latitude, Longitude for København

# Returnere flere værdier fra en funktion og sikre at de ikke bare kan ændres
def beregn_area_og_omkreds(radius):
    area = 3.14 * radius ** 2
    omkreds = 2 * 3.14 * radius
    return (area, omkreds)
```

```
areal = beregn_area_og_omkreds(100)
print(areal)
```

```
(31400.0, 628.0)
```

1.3 `dict`

En dictionary er en samling af nøgle-værdi-par. Hver nøgle er unik, og man bruger nøglen til at få adgang til den tilsvarende værdi. Dictionary er mutable, så både nøgler og værdier kan ændres.

- Oprettes med: {} eller ved hjælp af dict().
- Nøgler skal være unikke og typisk af en immutable type (som strenge, tal eller tuples), mens værdierne kan være af enhver type.
- Tilgår elementer via nøgler, ikke via indeks.

Egenskaber for dictionaries:

- Mutable (kan ændres).
- Nøgler er unikke (ingen dubletter af nøgler).
- Ingen specifik rækkefølge før Python 3.7. Fra Python 3.8 bevares rækkefølgen.

Dictionaries er en datastruktur, hvor du arbejder med nøgle-værdi-par. De er meget effektive til opslag, hvor man hurtigt skal finde en værdi baseret på en unik nøgle. Dictionaries er mutable, hvilket betyder, at du kan tilføje, ændre eller fjerne nøgle-værdi-par efter behov. Anvendelser:

- Hurtig opslag af data: Når du har brug for at få adgang til data hurtigt baseret på en unik nøgle (som en bruger-id, produkt-id osv.).
- Data med relationer: Brug dictionaries til at repræsentere relationer mellem data, f.eks. en persons navn knyttet til deres alder eller adresse.
- Mapping af data: Brug dictionaries til at kortlægge og associere data, såsom ordbøger, telefonbøger eller API-parameteropslag.

Use-cases:

- API-svar: Når du modtager data i JSON-format, er det ofte struktureret som en dictionary med nøgler som f.eks. “status”, “data”, og “error”.
- Gemning af egenskaber: Du kan bruge en dictionary til at gemme egenskaber for et objekt, f.eks. en persons navn, alder, køn osv.
- Optælling og statistik: Ved at bruge nøgler til at tælle forekomster af bestemte elementer, som antallet af gange et ord optræder i en tekst.

```
# Oprettelse af en dictionary
min_dict = {"navn": "Jeppe", "alder": 31, "by": "Aalborg"}
print(min_dict)
```

```
{'navn': 'Jeppe', 'alder': 30, 'by': 'Aalborg'}
```

```python
# Tilgå en værdi
print(min_dict["navn"])
```

```
Jeppe
```

```python
# Ændre en værdi
min_dict["alder"] = 23
print(min_dict)
```

```
{'navn': 'Jeppe', 'alder': 31, 'by': 'Aalborg'}
```

```python
# Tilføje et nyt nøgle-værdi-par
min_dict["land"] = "Danmark"
print(min_dict)
```

```
{'navn': 'Jeppe', 'alder': 31, 'by': 'Aalborg', 'land': 'Danmark'}
```

```python
# Fjerne et nøgle-værdi-par
del min_dict["by"]
print(min_dict)
```

```
{'navn': 'Jeppe', 'alder': 31, 'land': 'Danmark'}
```

1.4 Opsummering og use-cases

Opsummering

Funktion	Liste	Tuple	Dictionary
Mutable?	Ja	Nej	Ja
Indeksering	Ja	Ja	Nej (brug nøgler)

Datastruktur	Anvendelser	Typiske Use-cases

Use-cases

Datastruktur	Anvendelser	Typiske Use-cases
Lister	Dynamisk samling, tilføje/fjerne elementer, gemme data i rækkefølge	Indkøbslister, dynamiske samlinger, behandlingskøer
Tuples	Uforanderlige samlinger, sikre data, returnere flere værdier	Geografiske koordinater, funktionstilbageværdier, faste opsætninger
Dictionaries	Nøgle-værdi-par, hurtig adgang, associationsrelationer	API-data, databaser, statistik og optælling, ordbøger

2. Funktioner og argumenter

I Python er funktioner grundlæggende byggesten, der gør koden mere modulær, genanvendelig og struktureret. En funktion er en blok af organiseret, genanvendelig kode, der udfører en specifik opgave. Funktioner kan tage input (argumenter), udføre operationer på dem og returnere et resultat. Oprettelse af funktioner

Funktioner defineres ved hjælp af nøgleordet def, efterfulgt af et funktionsnavn og en parentes, der kan indeholde eventuelle parametre.

```
def funktionsnavn(parametre):
    # Funktionskrop
    # Valgfri return statement
    return resultat
```

- funktionsnavn: Navnet på funktionen.
- parametre: Variabler, som du kan give som input til funktionen.
- Funktionskrop: Koden, der udføres, når funktionen kaldes.
- return: Værdien, som funktionen kan returnere. Det er valgfrit.
- Funktioner gør koden mere genanvendelig og modulær.Du kan definere funktioner med positionelle argumenter, - nøgleord-argumenter, eller bruge standardværdier.
- Python tillader variabel længde argumenter via *args og **kwargs.
- Funktioner kan returnere flere værdier på én gang som en tuple.

- Lambda-funktioner er enkle, anonyme funktioner, der kan bruges, når en fuld funktionsdefinition er unødvendig.

Funktioner er et af de vigtigste værktøjer i Python og gør det nemt at organisere og strukturere din kode, så den bliver lettere at vedligeholde og læse.

```
def hej(navn):
    print(f"Hej, {navn}!")
```

```
hej("Jeppe")
```

```
Hej, Jeppe!
```

```
# Returnering af værdier
def kvadrat(x):
    return x ** 2
```

```
resultat = kvadrat(5)
print(resultat)
```

```
25
```

```
# Returnering af flere værdier (som Tuple)
def beregn(a, b):
    sum_ = a + b
    forskel = a - b
    return sum_, forskel
```

```
resultat = beregn(10, 5)
print(resultat)
```

```
(15, 5)
```

```
# Funktioner uden input
def velkommen():
    print("Velkommen til AAU!")
```

```
velkommen()
```

```
Velkommen til AAU!
```

```
# Funktioner med flere argumenter
def læg_sammen(a, b):
    return a + b
```

```
resultat = læg_sammen(3, 4)
print(resultat)
```

```
7
```

```
def hils(navn, dag):
    print(f"Hej, {navn}. Vi ses på {dag}.")
```

```
hils("Peter", "tirsdag")
```

```
Hej, Peter. Vi ses på tirsdag.
```

```
hils(dag = "onsdag", navn = "Lars")
```

```
Hej, Lars. Vi ses på onsdag.
```

```
# default-værdier i funktioner
def hils(navn, dag="fredag"):
    print(f"Hej, {navn}. Vi ses på {dag}.")
```

```
hils("Jens")
hils("Kim", "tirsdag")
```

```
Hej, Jens. Vi ses på fredag.
Hej, Kim. Vi ses på tirsdag.
```