Mats Vik Pettersen

Obilg 3

Teori 1

En **dictionary** i Python er en samling av elementer der hvert element har to deler: en nøkkel og en verdi. Du bruker nøkkelen for å få tilgang til verdien, som i et slags oppslagsverk. For eksempel, i en dictionary som dette: {"navn": "Mats", "alder": 19}, er "navn" en nøkkel, og "Mats" er verdien som hører til den nøkkelen.

Forskjellen mellom en dictionary og en liste:

- En **dictionary** bruker nøkler (for eksempel "navn") for å finne verdier, mens en **liste** bruker tall (indekser) for å finne elementer.
- I en **dictionary** kan du bruke en beskrivende nøkkel (som "navn" eller "alder") for å hente data, mens i en **liste** må du vite plasseringen (indeksen) til elementet for å få tilgang til det.

For eksempel:

- Dictionary: person["navn"] gir deg "Mats".
- Liste: liste[0] gir deg det første elementet.

Fordeler med dictionaries:

- **Rask tilgang**: Du kan raskt finne verdier basert på nøkler, uten å måtte lete gjennom hele samlingen.
- **Nøkler gir mening**: Nøklene kan være noe som beskriver verdiene godt, slik at dataene blir lettere å forstå.

Ulemper med dictionaries:

- **Mer minne**: De kan bruke mer minne enn lister, spesielt hvis du har mange elementer.
- **Ingen rekkefølge**: Elementene har ikke en fast rekkefølge, så det kan være vanskelig å holde ting i en bestemt rekkefølge.

Fordeler med lister:

- **Enkle og lette å bruke**: Lister er enkle å forstå og bruke for å holde en ordnet samling av data.
- **Fast rekkefølge**: Elementene er alltid i samme rekkefølge, noe som gjør dem ideelle for sekvenserte data.

Mats Vik Pettersen

Ulemper med lister:

- **Langsomt oppslag**: Hvis du vil finne et bestemt element i en stor liste, kan det ta tid, siden du må lete gjennom listen.
- **Kun indekser**: Du må huske hvor i listen (indeksen) et element er, noe som ikke alltid er praktisk.

Kort sagt: Dictionaries er bra når du trenger å lagre og få tilgang til data ved hjelp av nøkler (som et oppslagsverk), mens lister er gode når du vil ha en ordnet samling av elementer.

Teori 2

Funksjon

En funksjon i programmering er en selvstendig del av koden som utfører en bestemt oppgave. Du kan se på en funksjon som en liten maskin der du gir den inn data (kalt argumenter), den utfører en handling, og så gir den deg et resultat tilbake. Funksjoner gjør det lettere å gjenbruke kode, organisere programmet, og forenkle komplekse oppgaver.

Hvorfor er funksjoner så nyttige?

1. **Gjenbruk av kode**: Funksjoner lar deg bruke den samme koden flere ganger, uten at du må skrive den på nytt. Hvis du har en oppgave som gjentas flere steder i programmet, kan du lage en funksjon som gjør dette, og deretter bruke funksjonen så mange ganger du trenger. Dette sparer tid og gjør koden mer ryddig.
2. **Forenkling av koden**: Ved å bruke funksjoner kan du dele opp større og mer komplekse programmer i mindre, håndterbare biter. Dette gjør det mye enklere å forstå hva som skjer i programmet, spesielt når hver funksjon har et klart og tydelig formål. Det gjør også koden enklere å lese for andre.
3. **Unngå repetisjon**: En av de største fordelene med funksjoner er at de forhindrer unødvendig repetisjon av kode. Hvis du må gjøre den samme oppgaven flere steder i programmet, kan du bruke en funksjon i stedet for å kopiere den samme koden om igjen. Dette gjør at du bare trenger å oppdatere funksjonen ett sted, i stedet for å endre flere deler av programmet dersom noe må endres.

4. **Bedre struktur og oversikt**: Funksjoner hjelper med å organisere koden på en strukturert måte. De gir programmet en logisk oppdeling og gjør det enklere å finne og forstå de ulike delene. Når man leser koden, kan man enkelt se hva hver del gjør ved å se på funksjonsnavnene, noe som gjør det enklere å forstå programmet som helhet.
5. **Vedlikehold**: Når du bruker funksjoner, blir det mye enklere å oppdatere og vedlikeholde koden over tid. Hvis du har en funksjon som utfører en bestemt beregning, og det skjer en endring i hvordan denne beregningen skal gjøres, kan du bare oppdatere funksjonen én gang. Da vil alle stedene som bruker denne funksjonen automatisk ta i bruk den nye koden.
6. **Testbarhet**: En annen stor fordel med funksjoner er at de gjør det lettere å teste koden. Du kan teste en funksjon for seg selv, uten at du trenger å kjøre hele programmet. Dette gjør det lettere å finne feil, og du kan være sikker på at hver del av programmet fungerer som den skal.

Konsekvensene av å bruke funksjoner:

- **Enklere å skrive kode**: Når du bruker funksjoner, kan du konsentrere deg om én oppgave om gangen. Du lager små, veldefinerte biter av programmet som gjør bestemte oppgaver. Dette gjør det enklere å skrive riktig kode og redusere feil.
- **Enklere å lese kode**: Når koden er delt opp i små, meningsfulle funksjoner, blir det mye enklere å lese og forstå hva som skjer. Funksjonenes navn gir ofte en god indikasjon på hva de gjør, så man slipper å lese gjennom store mengder detaljert kode for å skjønne sammenhengen.
- **Enklere å oppdatere kode**: Fordi funksjoner brukes til å organisere kode, blir det mye lettere å oppdatere koden. Hvis du trenger å gjøre en endring, kan du ofte gjøre det direkte i funksjonen. Endringen vil da automatisk slå inn der funksjonen er brukt, uten at du må gjøre flere endringer i programmet.

Kort oppsummert:

Funksjoner gjør det enklere å **gjenbruke**, **forenkle** og **organisere** koden din. De hjelper deg med å skrive mer effektiv, oversiktlig og ryddig kode. Funksjoner sparer tid, reduserer feil og gjør programmet lettere å forstå og vedlikeholde. Ved å bruke funksjoner kan du lage kode som er mer fleksibel, testbar, og enklere å jobbe med både på kort og lang sikt.