# Dette bilag er en del af inspirationsmaterialet til faget [Erhvervsinformatik](https://emu.dk/eud/erhvervsinformatik/digital-myndiggorelse) på emu.dk og er udarbejdet som supplement til vejledningen. ”[Teknologisk handleevne og computational tankegang](https://emu.dk/eud/erhvervsinformatik/teknologisk-handleevne-og-computationel-tankegang)” er et af de tre kompetenceområder i faget. Data er grundlæggende for it-systemer, og derfor bør eleverne få en forståelse for hvilke typer af data it-systemer anvender, og hvordan data repræsenteres og er modelleret. Nedenfor finder du en kort introduktion til datatyper, variable og datastrukturer.

Datatyper, variable og datastrukturer

### Datatyper

En **datatype** en klassifikation, der identificerer og indskrænker hvilke værdier en given mængde data repræsenterer. Klassiske datatyper er heltal (integers), decimaltal (floats), tekst og tegn (strings) og boolske værdier (booleans), som kan være sande eller falske. Se siden om programmering.

### Variable

En **variabel** er en symbolsk repræsentation for en mængde eller et udtryk.

I matematik har variable oftest en "ukendt" mængde, så de kan ændre sig, eller *variere*. I datalogi repræsenterer variable allokeringer af data, som kan ændres.

Når man opretter (erklærer) en variabel, skal man angive dens *navn* og *datatype*. Derefter tildeler man den en *dataværdi*. Når en variabel får tildelt en bestemt værdi, kaldes den enten et “aktuelt parameter” eller et ”argument”.

Selve variablen er ikke en datastruktur - den er jo blot en symbolsk repræsentation for noget - men den kan godt indeholde en reference til en værdi, som er organiseret i en datastruktur, fx en liste eller et array.

### Datastrukturer

**Datastrukturer** er en fællesbetegnelse for data, der er organiserede i elementer, som kan tilføjes eller fjernes fra strukturen. Nogle datastrukturer forudsætter, at dataelementerne hver har et nøglefelt, der kan sorteres efter. Se ressourcen vedrørende databaser, som ikke gennemgås nedenfor.

* Man kan som udgangspunkt udføre følgende operationer på datastrukturer:
* Indsætte elementer
* Fjerne elementer
* Søge efter et bestemt element
* Teste om strukturen er tom

Eksempler på datastrukturer er linkede lister og arrays. Begge dele er måder at organisere flere dataforekomster lineært på.

### Linkede lister:

Data-elementer kan lagres på tilfældige pladser. Kronologi imellem pladserne skabes ved pegning fra et element til det næste.

### Arrays:

Data-elementer lagres fortløbende og tildeles et indeks-nummer, således at første plads får index-nummer 0, anden plads får indeks-nummer 1, osv

### Sammenligning af linkede lister og arrays:

Hvis man tænker på alfabetet, vil en liste gennem pegning kunne organisere dette kronologisk, altså a, b, c, d, e,..., selvom de enkelte bogstaver måske ikke er indskrevet på den måde i listen, men fx som b, e, c, a, d. Pegningen vil betyde, at der hoppes frem og tilbage i listen, så bogstaverne læses i den rigtige rækkefølge alligevel, altså at a peger på b, som peger på c, osv. Et array er nødt til at lagre data kronologisk, dvs. at a skal være det første element, b det næste osv.

En liste kan udvides uendeligt, mens et array har en bestemt størrelse fra starten, som på et tidspunkt vil blive fyldt, og det er omkostningstungt at udvide det. Derudover kan elementer nemt tilføjes eller fjernes på en liste - også midt i listen -, idet man bare skal ændre på pegningen. I et array vil fjernelse af elementer efterlade tomme områder, som kun kan fjernes ved at flytte alle de efterfølgende elementer. Ligeledes vil indsættelse af et nyt element kræve, at evt. efterfølgende elementer flyttes en plads.

Til gengæld tillader et array vilkårlig adgang til ethvert element (hvis man altså kender indeks-nummeret), mens man kun kan tilgå data på en liste sekventielt, dvs. at programmet skal gennemløbe listen fra start frem til det element, man skal bruge. Linkede listers brug af pegning til det næste element (og i dobbelt-linkede lister, hvor man også kan gå baglæns, til det foregående) gør, at linkede lister optager mere lagerplads i computerens hukommelse end et array, som ikke skal lagre denne information sammen med dataforekomsterne.

Som en konsekvens af ovenstående bør man bruge linkede lister, når man arbejder med store lister af data, hvor antallet af elementer på listen kan variere, mens arrays passer bedre til små lister, hvor man kender det maksimale antal af elementer på listen, og hvor der er en statisk sammenhæng mellem dem.

