**Funktionel noter til eksamen**

**Lisp - Map, FlatMap:**

***What is the difference between an array and a linked list?***

**Basic:**

Array: Array er en samling af elementer med samme datatype.

Linkedlist: Tilknyttet liste er en ordnet samling af elementer, der er forbundet med links / pointers.

**Size:**

Array: Angivet under erklæring.

Linkedlist: Ingen grund til at angive; vokser og skrumper under udførelsen.

**Storage Allocation:**

Array: Element placering er allokeret under kompileringstid.

Linkedlist: Element placering er tildelt under kørslen.

**Order of the elements:**

Array: Opbevares efter hinanden.

Linkedlist: Gemt tilfældigt.

**Accessing the element:**

Array: Direkte eller tilfældigt adgang, dvs. Angiv array indekset.

Linkedlist: Sekventielt adgang, dvs. fra den første node i listen af markøren.

**Insertion and deletion of element:**

Array: Langsomt relativt som forskydning er påkrævet.

Linkedlist: Nemmere, hurtig og effektiv.

**Searching:**

Array: Binær søgning og lineær søgning.

Linkedlist: Lineær søgning.

**Memory required:**

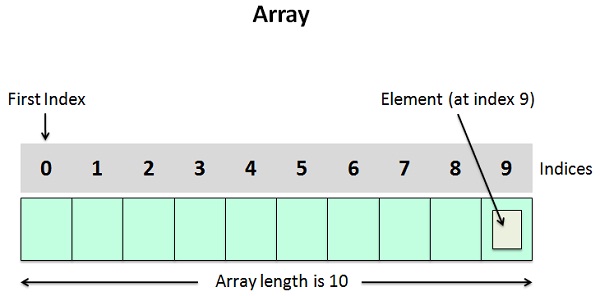
Array: mindre

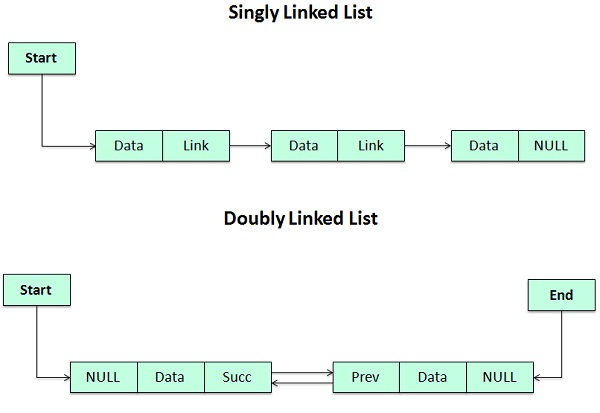
Linkedlist: mere

**Memory Utilization:**

Array: Ineffektiv

Linkedlist: Effektiv

******

******

***What is recursion?***

Rekursion i datalogi er en metode, hvor løsningen på et problem afhænger af løsninger til mindre forekomster af det samme problem. Tilgangen kan anvendes på mange typer problemer, og rekursion er et af de centrale ideer inden for datalogi.

Rekursion er at en funktion at kalde sig selv i funktionens kode. Nogle funktionelle programmeringssprog definerer ikke nogen looping konstruktioner, men er udelukkende afhængige af rekursion til gentagne opkaldskoder

public void fact ( int n) {

if (n == 0) return 1;

return n\* fact (n - 1); }

fact(5)

5\*fact(4)

5\*4\*fact(3)

5\*4\*3\*fact(2)

5\*4\*3\*2\*fact(1)

5\*4\*3\*2\*1\*fact(0)

5\*4\*3\*2\*1\*1

5\*4\*3\*2\*1

5\*4\*3\*2

5\*4\*6

5\*24

120

***What is a higher-order function?***

* Det er når en funktion enten tager en funktion som parameter eller returnere en funktion som resultat.

***What does mapping mean?***

* Mapping er en Higher-Order funktion som man kalder på hvert element af en liste og den returnere en liste af samme antal elementer.

***What does flattening a list mean?***

* Flattening er en process med at konvertere flere lister og gemt i en enkelt liste, der indeholder alle de elementer i de lister, du havde før.

[[[a, b], [c, d]], [[e, f], [g, h]]]

Kan blive flattend til:

[[a, b, c, d], [e, f, g, h]]

Også:

[a, b, c, d, e, f, g, h]

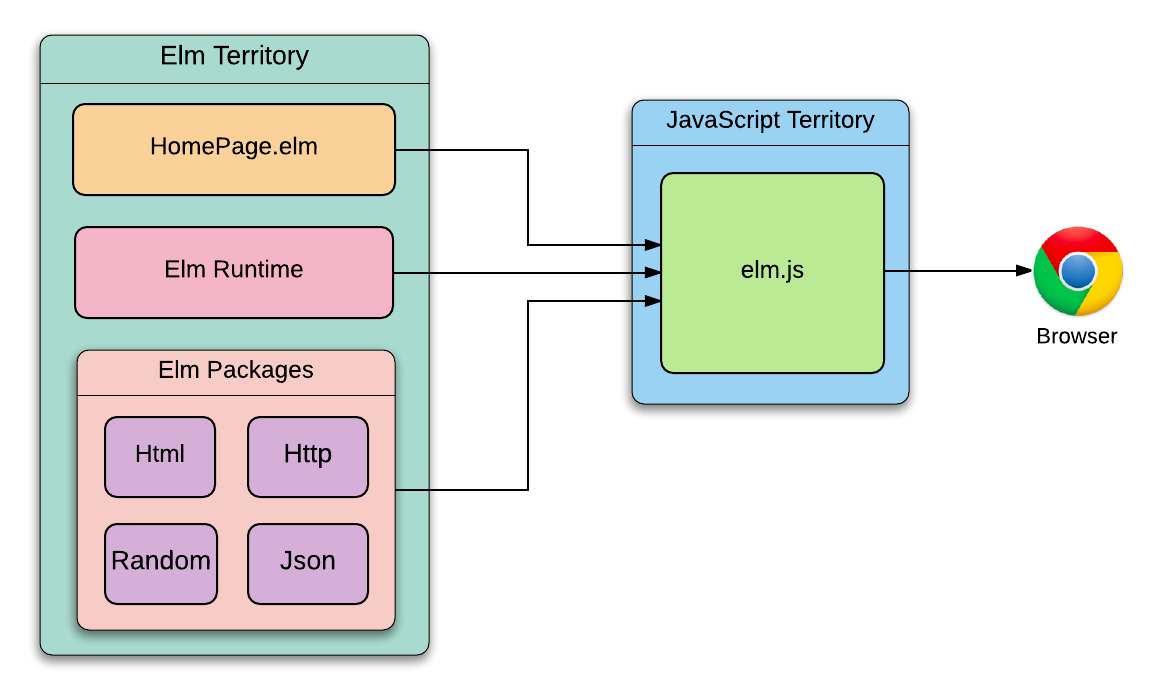
**Elm:**

***What are some of the differences between OOP and FP?***

* OOP - Object Oriented Programming: Når man programmere OOP binder man data og handling sammen hvor vi gør dem til units som hedder en Class. En Class kan blive set som et “blueprint” hvor man definerer “Types”, “Constructor” og “Metodes”. Når man så opretter et Objekt og tilføjer data til via. Constructoren og når man skal tilgå denne data kalder man objektet og de metoder der er lavet i den Class. OOP håndtere “State” og “Objekter” så hvis man ændre en property i et Objekt ændre det State’en, det er hvad man kalder “Mutable” og det er BESKIDT.
* FP - Functional Programming: Det er det modsatte af OOP, her deler vi Data og Handling op hver for sig. Handling er Funktioner og Data er indput, der bliver så udført en handling på data’en og hvis data’en bliver ændret så laver man en kopi af data’en og returnere det nye data. Ved at gøre det på den måde bliver det “Stateless” data er “immutable”. Det vil sige at PURE programming languages er “Stateless”.

***Why do we need the Elm runtime?***

* Elm runtime er et system designet til at understøtte udførelsen af programmer skrevet på Elm programmeringssprog. Det er her, hvor Elm runtime kommer ind. Elm.js filen indeholder ikke kun koden men også hele Elm runtime og alle de andre Elm pakker som er installerede. Browseren forstår kun JavaScript. Så hvert stykke Elm kode skal omdannes til det, herunder koden for selve runtime.



***What are some of the benefits of using Elm?***

* fordelen ved elm er at vi får funktionelle tankegang at se, og hvor hjælpsom en compiler faktisk kan være. Elm er pure funktionel programmering. På grund af designet i elm kan der ikke forekomme nogen runtime exception.

***What is the Maybe type in Elm? When is it used?***

* En funktion i Elm er den af typen Maybe. Maybe i Elm indkapsler effektivt data, som måske eller måske ikke eksisterer. Dette er ret praktisk, især hvis du håndterer komplekse data, som du ikke har meget viden om eller kontrol over. Noget af typen Maybe er det altid forbundet med Just eller Nothing, som i det kan være der, eller det kan ikke være.

type Maybe a = Just a | Nothing

(a her er bare en pladsholder til en bestemt type som bool, String, Int osv.).

List vegetarians = [“Ben”, “Emily”, “Alex”]

List.head vegetarians = Just “Ben” : Maybe.Maybe String

***What is the Result type in Elm? When is it used?***

* Result er nyttigt, når du har logik, der kan mislykkes. For eksempel kan parse af en string i en Int mislykkes. For at gøre dette så konkret som muligt, lad os se den egentlige definition af Result. Det ligner Maybe typen meget, men det har to typer variabler: **Den har to konstruktører: Err for fejl og Ok for succeser.**

type Result error value

= Err error

| Ok value

***Can you reverse a string in Elm?***

import Html

exposing (Html, ol, li, text)

import String

main = viewString "desserts"

viewString : String -> Html

msg viewString string =

text (String.reverse string)

***Can you write a call to a HTTP endpoint?***

*type Msg = Get | Set | FetchCount (Result Http.Error Int) | ResetCount (Result Http.Error Int)*

fetchCount: Cmd Msg

fetchCount =

let

url = "http://localhost:5050/counter"

request = Http.get url counterDecoder

in

Http.send FetchCount request

resetCount: Cmd Msg

resetCount =

let

url = "<http://localhost:5050/counter/1>"

request = Http.post url Http.emptyBody Decode.int

**Haskell:**

***What is a side-effect?***

* Det er noget som kommer ude fra den “pure world” i funktionel programmering**.** En bestemt funktion kan ændre en global variabel eller statisk variabel, ændre en af ​​sine argumenter, ophæve en undtagelse, skrive data til en skærm eller fil, læse data eller aktivere til andre sideffekter.
* Det funktionelle sprog Haskell udtrykker side effects som I / O og andre stateful beregninger ved hjælp af monadiske handlinger
* Det er derfor vi har Monads til at indkapsle det som er “unpure” til at gøre det “pure” igen.

***How do Haskell handle side-effects? What does 'IO ()' mean?***

* Fra et abstrakt og idealiseret synspunkt findes der ikke side effects i Haskell. Hvad Haskell tillader, er evnen til at compile side effects (med IO-monaden) uden at faktisk udføre dem.
* fra et konkret og fysisk synspunkt kan side effects forekomme, da programmet (hoved) bliver evalueret, men på grund af monads arbejde må du ikke bekymre dig om side effects, der opstår i den forkerte rækkefølge eller muligheden af side effects ikke bliver ordnet..

***What does it mean that functions are first-class citizens?***

* I funktionel programmering kan funktioner tildeles variabler, overføres til andre funktioner som parametre, og returneres som værdier fra andre funktioner. Sådanne funktioner er kendt som førsteklasses funktioner.

***What is recursion?***

* Rekursion i datalogi er en metode, hvor løsningen på et problem afhænger af løsninger til mindre forekomster af det samme problem. Tilgangen kan anvendes på mange typer problemer, og rekursion er et af de centrale ideer inden for datalogi. Rekursion er at en funktion at kalde sig selv i funktionens kode. Nogle funktionelle programmeringssprog definerer ikke nogen looping konstruktioner, men er udelukkende afhængige af rekursion til gentagne opkaldskoder

public void fact ( int n) {

if (n == 0) return 1;

return n\* fact (n - 1); }

fact(5)

5\*fact(4)

5\*4\*fact(3)

5\*4\*3\*fact(2)

5\*4\*3\*2\*fact(1)

5\*4\*3\*2\*1\*fact(0)

5\*4\*3\*2\*1\*1

5\*4\*3\*2\*1

5\*4\*3\*2

5\*4\*6

5\*24

120

***What are some of the benefits of using Haskell?***

* **Pure** (Det tillader ikke nogen side-effects)
* **Laziness** (Det betyder, at intet er evalueret, indtil det skal evalueres)
* **Memory managed** (Der er ingen bekymring for at ting går galt for Garbage Collectoren tager sig af alt det)
* **Performance** (At det går hurtigt)

**Virtual CPU and tail recursion:**

***What is an instruction?***

* En instruktion er maskinkode, et sæt af instruktioner, der udføres direkte af CPU’en.

***What is a CPU?***

* Central processing unit - (CPU) er det elektroniske kredsløb i en computer, der udfører instruktionerne i et computerprogram ved at udføre de grundlæggende aritmetiske (matematik), logiske, kontrol- og input / output-operationer (I / O), der er angivet i instruktioner.

***What is tail recursion?***

* I Tail recursion udfører du først dine beregninger, og derefter udfører du det rekursive opkald, der overfører resultaterne af dit nuværende trin til det næste rekursive trin. I princippet er retur værdien af et givet rekursivt trin det samme som retur værdien af det næste rekursive kald.

***When can tail recursion be applied?***

I datalogi er et tail call en metode, der udføres som den endelige handling i en procedure. Hvis et tail call kan føre til, at den samme undermetode bliver kaldt igen, denne undermetode kaldes tail recursion, hvilket er et specielt tilfælde af rekursion. Tail recursion (eller tail-end rekursion) er særligt nyttig, og ofte nem at håndtere i implementeringer.

tail racusion kan implementeres uden at tilføje en ny stack frame til call stacken.

***Why is tail recursion necessary in FP?***

* Functional programming gør brug af mange funktioner og man undgår derfor et stackoverflow ved at køre tail recursion fremfor recursion
* tail recursion Det er en funktion, der slet ikke gør noget efter rekursering.  
    
  Dette er vigtigt, fordi det betyder, at du bare kan videregive resultatet af det rekursive opkald gennem direkte i stedet for at vente på det - du behøver ikke at forbruge stakplads. En normal funktion skal derimod have en stabelramme, så kompilatoren ved at komme tilbage til den (og have alle de nødvendige variable værdier) efter det rekursive opkald er afsluttet.