DU2 - Inlämningsuppgift (U1)
Guide

När man tänker igenom ett program bör man alltid ha **papper och penna**, precis som med HTML och CSS.

Om du inte vet vad det är du ska koda så ska du inte sitta framför VSC och stirra i skärmen. Du ska istället sitta framför papper och penna och försöka tänka på problemets olika delar.

I denna guide föreslår jag några steg att följa när man ska lösa ett programmerings problem, om det inte är så att du känner att du vet exakt vad du ska göra.

Rekommenderas varmt:

- Skriv ut dessa ark på papper, jobba inte från datorn!
- Läs noggrann och förstå ALL text i guiden. Fråga i lektionen om det är något du undrar över!
- Anteckna på papperet: fyll arken med dina tankar och frågor kring koden.
- När du väl har förstått programmets struktur. Göm arken och skapa
  - Din egen lista över saker som programmet ska göra
  - Ditt eget flödesschema
  - Din egen pseudokod
- Använd andra ark för att skapa ett flödesschema över funktionerna (om du behöver det) och för att pseudokoda funktionerna.
- Börja koda i VSC först när du känner att du har en bra bild över det du ska koda.

Steg 1: Lista allt som programmet ska göra

Börja med att gå igenom briefen och videon och lista allt du kommer på som behöver hända.

Ordningen väntar du med, nu ska du bara lista allt som ska ske i programmet.

I U1:

- Användaren ombeds skriva namnet på en stad (vi kallar den *target*)
- Vi uppdaterar fliken, h2 och h3 med relevant information
- Vi måste kontrollera om staden finns på databasen
- Listan över alla städer (i de grå lådorna) skapas på sidan
- Tabellen över alla distanser skapas på sidan
- Vi ska hitta staden som ligger närmast *target*
- Vi ska hitta staden som ligger längst bort från *target*
- Vi markerar (ändrar bakgrundsfärg och textfärg) på stad-boxarna som innehåller städerna target, närmast och längst-bort.
- Vi uppdaterar stad-boxarna av den närmaste och längst-borta städerna så att distansen till target-staden också visas.

Steg 2: Ordna det programmet ska göra

Om du kollar på listan ovan så ser du att allt ska inte hända alla gånger programmet körs.

Vissa skeenden ska endast hända om staden som användaren anger finns i databasen (alltså arrayen cities).

Andra ska bara hända om staden som användaren anger INTE finns i databasen.

Så vi kan direkt dela upp det som ska hända i: Saker som ska hända om staden finns i databasen och saker som ska hända oavsett.

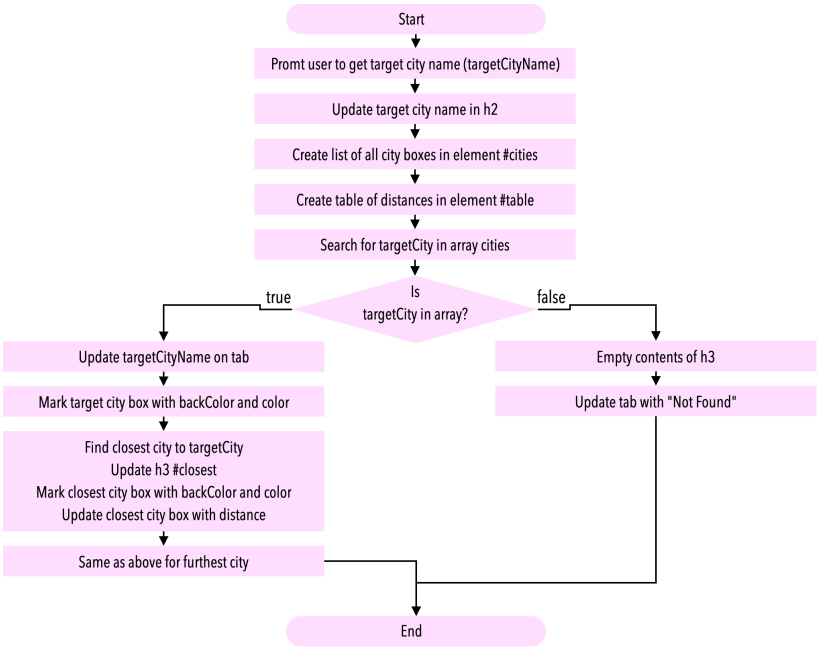
1. Saker som ska hända oavsett:
    1. Användaren ombeds skriva namnet på en stad (vi kallar den target)
    2. Vi uppdaterar h2 med namnet som användaren har angett
    3. Listan med städer i databasen skapas
    4. Tabellen med avstånd skapas
    5. Vi kontrollerar om staden finns i databasen
2. Saker som ska hända om staden finns i databasen:
    1. Vi söker staden som ligger närmast *target*.
    2. Vi söker staden som ligger längst bort från target.
    3. Vi uppdaterar h3 (städerna som ligger närmast och längst bort) och fliken (namnet på staden)
    4. Vi markerar (ändrar bakgrundsfärg och textfärg) på städerna target, närmast och längst-bort.
    5. Vi uppdaterar textContent för närmast och längst-bort så att distansen kommer också med.
3. Saker som ska hända om staden inte finns i databasen:
    1. h3 ska tömmas
    2. Fliken ska uppdateras med ”Not Found”

Steg 3: Flödesschema (flow-chart)

Ett flödesschema är en visuell representation av hur koden kommer att ”flöda”, dvs, hur det som ska göras ska struktureras. Kontrollstrukturer som loopar, if-satser och funktioner blir synliga.

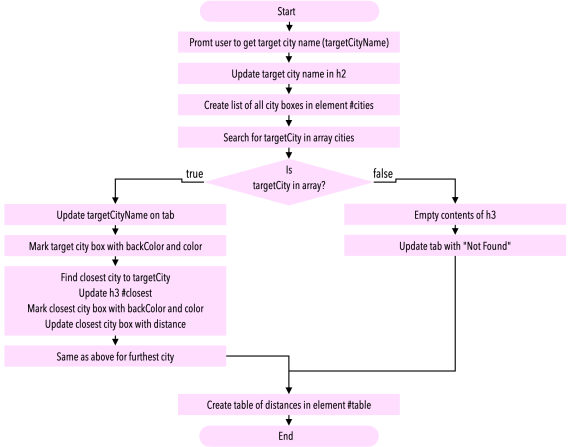
Ett flödesschema kan vara mer eller mindre detaljerat, beroende på hur säker man känner sig på de olika delarna av koden. Ju mindre säker man känner sig, desto mer detaljerat bör flödesschemat vara, eftersom det kommer att hjälpa oss komma fram till koden.

Det kan kännas svårt att skapa flödesscheman, men det är mycket svårare att hoppa över flödesschemat och börja koda direkt.



Saker att notera:

- Det finns pilar mellan lådorna, som visar hur koden ”flödar”, alltså i vilken ordning de olika delarna exekveras.
- Romben representerar en if-sats, där koden kan gå den ena eller den andra vägen beroende på om villkoret uppfylls (true) eller inte (false).
  - Hade det kommit fler instruktioner efter if-satsen så hade vi placerat dem där pilarna från båda grenarna kommer ihop igen (alltså där vi har placerat ”End”).
- Start och End är inte instruktioner, de översätts inte till kod. De är bara för att markera vart programmet startar och slutar (ett flödesschema kan bli rätt komplext efter ett tag)
- Visa lådor känns rätt enkla att programmera (som ”Promt user to get target city name (targetCityName”). Andra lådor representerar mycket mer komplex kod (som ”Create table of distances in element #table”). Dessa mer komplexa lådor kan detaljeras i ett annat flödesschema, som bara visar vad som händer i den lådan.
- Tabellen över distanser kan skapas också efter det att allt annat har hänt. Då skulle flödesschemat se ut såhär:



Steg 4: Pseudokod

Nästa steg är att skriva ner programstrukturen med hjälp av sk. ”pseudokod” (pseudo-code på engelska). Pseudokod är lite som kod men inte riktigt… Såhär skulle programmet kunna se ut i pseudokod (om vi skapar avståndstabellen efter det att vi har fixat allt annat:

```
cityTargetName = Prompt the user for target city name
Update cityTargetName in h2

createAllCityBoxes()

targetCityObject = getCityByName(cityName) // return null if no city with cityTargetName in array

If cityObject != null

        Update cityTargetName on tab

        markCityBox(targetCityObject, ”target”)

        closestCityObject = getClosestCity(targetCityObject) // return whole object. Include distance to target
        Update h3 #closest
        markCityBox(closestCityObject, ”closest”)
        updateBoxDistance(closestCityObject)

        furthestCityObject = getFurthestCity(targetCityObject) // return whole object. Include distance to target
        Update h3 #furthest
        markCityBox(furthestCityObject, ”furthest”)
        updateBoxDistance(furthestCityObject)

Else

        Empty h3
        Update tab with ”Not found”

createDistanceTable()
```

Pseudokoden ovan kan skrivas lite hur som helst, syntaxen är inte så noga, bara det funkar för dig. Så småningom kommer du att utveckla din egen stil vad gäller pseudokod.

Saker att notera:

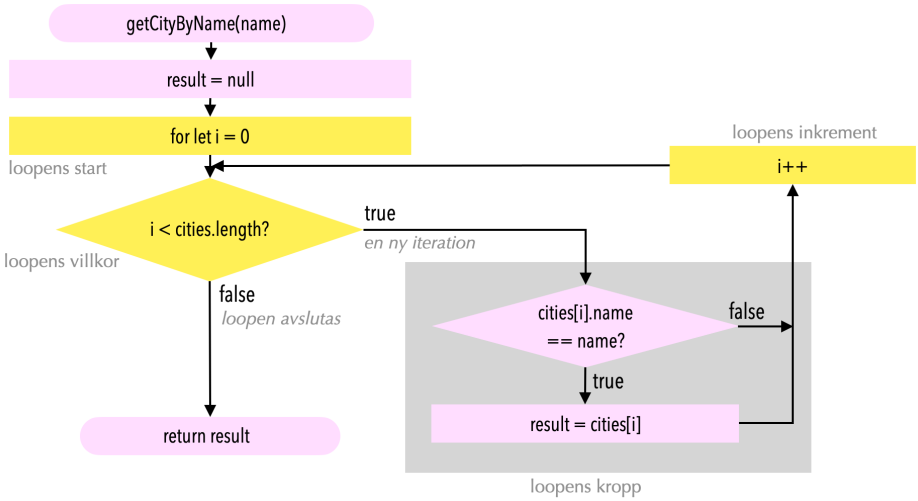
- Indenteringen! Den är absolut nödvändig om du ska kunna förstå pseudokoden. I vissa programmeringsspråk (som det mycket populära Python) har man helt tagit bort måsvingarna och det är endast indenteringen som visar vilka kodrader som ingår i i if-satsens kropp (och alla andra kroppar).
- Det finns flera funktionsanrop i koden men där finns inte någon kod för funktionerna.
  Men man kan se vilka argument funktionerna tar emot (om några).
  Funktionens returvärde förklaras där det är viktigt för programmet.
  En del av funktionerna är endast sideffekt-funktioner, så deras returvärdet är ointressant.
- Att funktionerna inte har ”stavats ut” i koden betyder inte att funktionerna är enkla. Men i detta läge behöver man inte tänka på hur de löses. Just nu försöker du bara få en överblick över hur programmet ska struktureras. Funktionerna löser du i efterhand.
  - Skulle man jobba i ett lag så kan man efter detta steg bara dela ut arbetet med funktionerna till olika personer. De vet vilka parametrar som funktionen tar emot och vad de ska göra på webbsidan, eller vilket värde de ska returnera. Samarbetet blir mycket enklare.

Steg 5: Flödesschema och pseudokod för funktioner

Nästa steg är att tänka igenom hur funktionerna ska kodas. Vissa är mer komplexa än andra.

Ibland är funktionen rätt enkel och vet vi exakt hur den ska kodas. Är funktionen lite mer komplex, eller om vi känner oss osäkra, måste vi först ta fram pseudokod. Är funktionen ännu mer komplex behöver vi ta fram ett flödesschema. Om man riktigt fastnar vid en funktion så angriper man den från steg 1 i denna guide (alltså börja med att tänka på allt som ska hända i funktionen), och följer alla steg efter det.

Jag visar här flödesschemat för funktionen getCityByName, eftersom den innehåller en loop, så ni ser hur man



kan representera den i ett flödesschema.

Pseudokoden skulle kunna vara:

```
Function getCityByName (name)
        let result = null
        for i = 0; i < cities.length; i++
                if cities[i].name == name
                        result = cities[i]
        return result
```

Eftersom vi bara vill hitta den första staden i arrayen som uppfyller kravet (att dess namn är samma som argumentet) så kan vi efter det att vi har hittat det, avsluta loopen. Vi behöver inte kolla på resten av arrayen.

```
Function getCityByName (name)
        let result = null
        for i = 0; i < cities.length; i++
                if cities[i].name == name
                        result = cities[I]
                        break
        return result
```

En vända till, med en typisk tankegång hos en utvecklare: Eftersom instruktionen *return* avslutar funktionen helt, oavsett var den finns, så kan koden simplifieras till (och så passar jag på att använda for-of, som funkar bra i detta fall:

```
Function getCityByName (name)
        for city of cities
                if city.name == name
                        return city
        return null
```

De två extra-pseudokoderna följer inte flödesschemat.

Vi rekommenderar att du försöker rita flödesscheman som passar till dessa pseudokoder. Jättebra övning :-).

Steg 6: Koda!

Nu är du redo för att börja koda.

Koda funktionerna först, så att du kan testa att de fungerar som de ska.

Sen integrerar du funktionerna i programmet.

Var inte orolig om koden inte fungerar felfritt från början. Saker och ting blir sällan exakt som man hade planerat, man brukar missa både stora och små detaljer. Men nu har du en bra bild av vad du vill uppnå och det är mycket lättare att förstå vad det är som inte fungerar, och varför.

I detta fall hade jag kodat i denna ordning (men det är verkligen inte det enda bra sättet att göra det):

1. Koda funktionen createAllCityBoxes() som skapar listan över alla städer
2. Koda funktionen markCityBox(cityObject, kindOfCity). Funktionen tar emot två argument, ett för att veta vilken stad (box) som ska uppdateras och ett för att veta vilken typ av stad det är (”target”, ”closest” eller ”furthest”). Tips: Jag hade nog använt mig av CSS-classer med just de namnen…
    1. Tips: Testa funktionen genom att skicka vilken stad som helst som argument (cities[3], exvis) och testa med alla tre olika typer av städer.
3. Koda funktionerna getClosestCity and getFurthestCity. De kommer att vara ganska likt varandra.
    1. De tar emot targetCityOject som argument (ett av elementen i arrayen cities).
    2. De returnerar ett city-object (ett av elementen i arrayen cities)
    3. Notera att du senare kommer att behöva ha avståndet till targetCity för att uppdatera boxen
        1. I pseudokoden har jag kommenterat ”Include distance”. Med det menar jag att jag ska lägga till en egenskap i city-objektet med avståndet som värde.
    4. Testa funktionerna och kontrollera att de fungerar. Kolla på bilden över tabellen, välj en target (Graz, exvis) och hitta städerna som ligger närmast (Maribor, 10km) och längst bort (Bordeaux, 150km). Sen anropar du funktionerna med Graz-objektet som argument (cities[16]) för att se om du får rätt stad-objekt tillbaka (Maribor och Bordeaux).
4. Koda funktionen updateBoxDistance. Notera att den tar som argument objektet som funktionerna getClosestCity och getFurthestCity returnerar. Det objektet inkluderar bla. en egenskap som innehåller avståndet mellan den staden och targetCity (fiffigt!).
    1. Använd returvärden från förra punkten för att testa denna funktion. Kolla så att rätt stad-boxar uppdateras, och på rätt sätt.
5. Koda funktionen getCityByName.
    1. Den tar emot en string som argument, nämligen namnet på staden.
    2. Den ska returnera:
        1. null, om det inte finns någon stad i arrayen cities med det namnet.
        2. Hela objektet av staden som har det namnet, om det finns en stad med det namnet i arrayen.
    3. Testa funktionen med namn som finns och som inte finns (som ”asd” exvis)
6. Koda funktionen createDistanceTable. Den är lite klurig.
7. Integrera det hela med din pseudokod som guide.
8. Fixa till detaljerna som ”Update tab”, ”Update h2”, osv.
9. Testa att det funkar med både namn som finns och namn som inte finns.
    1. Tips: Testa i början utan prompt, använd bara
       const targetCityName = ”Graz”;
       eller
       const targetCityName = ”Grazo”;
       ..så slipper du skriva namnet varje gång du ska testa något. När allt funkar testar du med prompt.