

Programming 1

Table of Contents

1. Flödesscheman	1
1.1. Komponenter	1
1.1.1. Input	1
1.1.2. Output	1
1.1.3. State Change	2
1.1.4. Conditions	2
1.1.5. Loops	2
1.2. Funktioner i flödesscheman	3
2. Verktygslådan	3
2.1. Verktyg för ändring av State	3
2.1.1. Tilldelning	3
2.1.2. Inkrementering / Dekrementering	3
2.2. Verktyg för uppdelning av flödet (Villkor)	4
2.2.1. If	4
2.2.2. If-Else	5
2.2.3. If-Elseif-Else	5
2.3. Verktyg för upprepning av flödet (Loopar)	6
2.3.1. Räknande loop	6
2.3.2. Okänt antal iterationer	7
2.4. Verktyg för funktioner	8
2.4.1. Outputvariabel	8
2.4.2. Iterativt uppbyggd output	8

1. Flödesscheman

För att modellera hur en funktionsmasking fungerar använder vi flödesscheman. Flödesscheman består av komponenter som antingen

- ändrar på state eller
- styr flödet beroende på state.

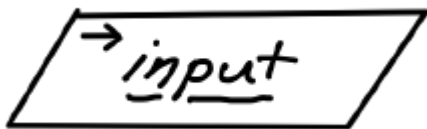
Komponenterna kopplas ihop med pilar och bildar ett flöde.

1.1. Komponenter

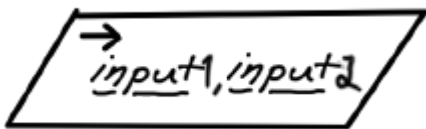
Alla komponenter i flödesscheman interagerar med **state** eller **input** på något vis. State består av variabler som indikeras med understruken text.

1.1.1. Input

Input-komponenten illustreras med en parallelogram med en liten pil till vänster. I rutan skrivs namnet på input så man kan referera till den som en variabel senare i flödet.

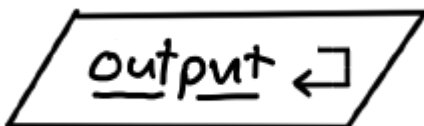


Om en situation kräver flera inputs separeras dessa med komma-tecken.



1.1.2. Output

Output-komponenten illustreras också med en parallelogram fast med en retur-pil på högersidan. I rutan skrivs variabeln som innehåller värdet som ska ges som output.



Om man i flödet når en **output**-komponent så avslutas flödet direkt.

1.1.3. State Change

För att ändra på state, antingen genom att

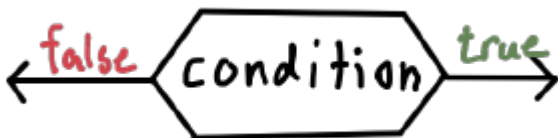
- Skapa en ny variabel, eller
- Ändra på en befintlig variabel

illustreras detta med en rektangel. I rutan skriver man på vilket sätt som variabeln ska ändras. Antingen genom att använda tilldelningstecken eller med ord beskriva vad som händer.



1.1.4. Conditions

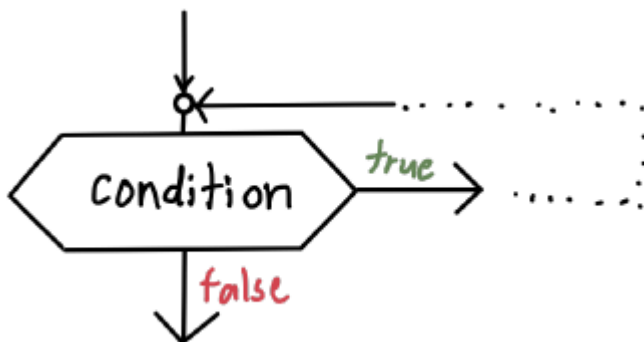
En **condition**-komponent illustreras med en utdragen hexagon. Den används för att dela flödet i två olika vägar. Den har alltid **en** ingång och **två** utgångar, en **true**-utgång och en **false**-utgång. I rutan skriver man en jämförelse mellan två variabler.



true-utgången går vanligtvis åt höger och **false**-utgången går vanligtvis rakt ner.

1.1.5. Loops

En **loop**-komponent illustreras nästan som en **condition**-komponent, egentligen är de samma sak. Skillnaden är en liten ring ovanför hexagonen och att false-utgången vanligtvis går rakt ner. I rutan skrivs **loopvillkoret**.



Loopvillkoret måste bero på något state som ändras i true-delen, och det måste ändras på ett sätt så att **loopvillkoret** någon gång blir falskt. Annars får man en oändlig loop.

1.2. Funktioner i flödesscheman

Flödesscheman kan beskriva bara en liten del av ett program eller så kan de beskriva en hel funktion. En hel funktion beskrivs genom att flödesscheman har en **input**-komponent och en **output**-komponent.


2. Verktygslådan

2.1. Verktyg för ändring av State

2.1.1. Tilldelning

För att kunna använda värden (t.ex tal) i program måste man först lagra dem i en variabel. Att lagra ett värde i en variabel gör man med hjälp av tilldelning.

En tilldelning består av tre komponenter: en **variabel**, ett **tilldelningstecken** och ett **värde**.

Flödesschema	Kod
	<pre>score = 5</pre>

2.1.2. Inkrementering / Dekrementering

Inkrementering

Att öka värdet på en variabel av datatypen **Integer** kallas *inkrementering*, eller att *inkrementera*.

Oftast används inkrementeringsverktyget i samband med loopar.

Flödesschema	Kod
	<pre>score = 5 score = score + 1</pre>

Dekrementering

Att minska värdet på en variabel av datatypen **Integer** kallas *dekrementering*, eller att *dekrementera*.

Flödesschema	Kod
	<pre>score = 5 score = score - 1</pre>

Oftast används dekrementeringsverktyget i samband med loopar.

2.2. Verktyg för uppdelning av flödet (Villkor)

2.2.1. If

Om du vill att ditt program ska göra en sak enbart om ett villkor är sant, kan du använda if. If-verktyget använder ett villkor, och om villkoret utvärderas till sant kommer något ske. Om villkoret inte utvärderas till sant kommer programflödet fortsätta efter villkoret, som om ingenting hänt.

Flödesschema	Kod
	<pre> if condition # True-delen end # Sammanfogningen </pre>

2.2.2. If-Else

Om du vill att ditt program ska göra en sak **om ett villkor är sant, och en annan sak om villkoret är falskt** kan du använda **if-else**.

Flödesschema	Kod
	<pre> if condition # True-delen else # False-delen end # Sammanfogning </pre>

2.2.3. If-Elseif-Else

Om ditt program har *fler än två* olika saker det ska göra baserat på **olika** villkor kan du använda if-else-if-else.

Flödesschema	Kod
<pre> graph TD Start(()) --> Cond1{Condition1} Cond1 -- true --> DoStuff1[do stuff] Cond1 -- false --> Cond2{Condition2} Cond2 -- true --> DoStuff2[do stuff] Cond2 -- false --> Cond3{Condition3} Cond3 -- true --> DoStuff3[do stuff] Cond3 -- false --> DoStuff4[do stuff] DoStuff1 --> Join(()) DoStuff2 --> Join DoStuff3 --> Join DoStuff4 --> Join Join --> End(()) </pre>	<pre> if condition1 # True-delen för condition1 elseif condition2 # True-delen för condition2 elseif condition3 # True-delen för condition3 else # False-delen för alla conditions end #Sammanfogning </pre>

2.3. Verktyg för upprepning av flödet (Loopar)

2.3.1. Räknande loop

Inkrementerande

En inkrementerande loop passar bra att använda om du i förväg vet (eller kan räkna ut) hur många gånger programflödet behöver upprepas, t.ex. om programmet alltid ska göra något 5 gånger, eller om vet att du ska göra något lika många gånger som du har tal i en lista.

Inkrementerande loopar använder sig av en *räknar-variabel*. En räknar-variabel håller koll på hur många gånger loopen upprepas. Vanliga namn på räknar-variabler är **counter**, **iterations** eller **i**, men en räknar-variabel kan heta precis vad som helst.

För att en räknande loop ska fungera behöver följande steg ske:

1. Innan loopen påbörjas måste man *initiera* räknar-variabeln, det vill säga, tilldela den ett värde. I de flesta fall initierar man räknar-variabeln med värdet 0 (eftersom det är så många gånger man upprepat programflödet innan man börjar loopa).
2. Räknar-variabeln används sen inne i villkoret som avgör om programflödet ska upprepas eller ej.
3. Inne i loopen måste räknar-variabeln *inkrementeras*, det vill säga ökas. Om man inte ökar räknar-variabeln kommer loopen fortsätta i all evighet, eftersom loop-villkoret aldrig blir falskt.

Flödesschema	Kod
<pre> graph TD Start(()) --> Init[number_of_turns = 0] Init --> Inc[number_of_turns = number_of_turns + 1] Inc --> Cond{number_of_turns < game_turn_limit} Cond -- True --> Inc Cond -- False --> Exit(()) </pre>	<pre> iterations = 0 number_of_iterations = 10 while iterations < number_of_iterations # Do stuff iterations = iterations + 1 end </pre>

Dekrementerande

En dekrementerande loop har i stort sett samma användningsområde som en inkrementerande loop, det vill säga, du vet (eller kan räkna ut) hur många gånger du behöver upprepa programflödet.

Flödesschema	Kod
<pre> graph TD Start(()) --> Dec[turns_left = turns_left - 1] Dec --> Cond{turns_left > 0} Cond -- True --> Dec Cond -- False --> Exit(()) </pre>	<pre> iterations_left = 10 while iterations_left > 0 # Do stuff iterations = iterations - 1 end </pre>

Skillnaden är att räknar-variabeln räknar neråt, det vill säga dekrementerar.

2.3.2. Okänt antal iterationer

Om du inte i förväg vet (eller kan räkna ut) hur många gånger programflödet ska upprepas behöver du använda en loop med brytvärde.

Flödesschema	Kod
<pre> graph TD Start(()) --> Cond{player_health* > 0} Cond -- True --> Loop[] Loop --> Cond Cond -- False --> Exit(()) </pre> <p>* förutsatt att det finns en variabel med detta namn</p> <p>någonstans i loopen måste "player_health" kunna ändra värde.</p>	<pre> while player_health > 0 # Do stuff end </pre>

2.4. Verktyg för funktioner

2.4.1. Outputvariabel

När en funktion ska behandla **input** på något vis, och returnera någon **output** beroende på beräkningen så är en outputvariabel väldigt användbar. Outputvariabeln håller värdet på det som ska ges som output när funktionen har körts klart. Det krävs att outputvariabeln **initieras** med något startvärde. Startvärdet kan vara vilket värde som helst eller att man kopierar input.

2.4.2. Iterativt uppbyggd output

Ett vanligt användningsområde för en **outputvariabel** är när man iterativt bygger upp en **output**. Detta verktyg är särskilt användbart när varje iteration beror på resultatet av iterationen innan.

Flödesschema	Kod
<pre> graph TD Input[/input/] --> Init[output = ?] Init --> Cond{loop-villkor} Cond -- True --> Mod[output += ?] Mod --> Cond Cond -- False --> Output[/output/] Output --> Stop([stop]) </pre> <p>Initiera output-variabeln med A) ett standardvärde (t.ex 0 eller "" eller []). B) En gissning eller ett uträknat värde (t.ex värdet på första elementet i input).</p> <p>ev. andra saker som behöver ske innan loopen börjar</p> <p>Modificera inte input!</p> <p>modificera output med nästa värde (t.ex inkrementera, konkatera eller byta värde)</p>	<pre> def function(input) output = ? # ... while condition # ... output = output + ? end return output end </pre>