Ett internationellt programmeringsspråk

Elias Sjögreen

Skriv min abstract hääär!

Innehållsförteckning

# Inledning

Datorprogrammering har gått från att vara en nisch skicklighet till något fler och fler människor lär sig, för jobb och som en hobby. Industrin och den globala efterfrågan för programmerare ökar i takt med internet och i princip all teknologi, från din mikrovågsugn till dator till det komplexa system som styr exempelvis elnätet. Detta är inte heller något lokalt fenomen, digitaliseringen är global men majoriteten av programmeringsspråken som är ett väsentligt verktyg för denna teknologiska utveckling är fortfarande begränsade av en språkbarriär då de flesta programmeringsspråk använder sig av Engelskan. Denna rapport undersöker möjligheten att skapa en specifikation samt prototyp av ett programmeringsspråk med mål att vara översättningsbart mellan olika naturliga språk i utbildningsyfte.

# Bakgrund

## Begreppslista

**Programmeringspråk** eller **programspråk** är en representation i text eller visuellt av de instruktioner som man önskar ska kompileras för att sedan köras, antingen som maskinkod, mellanrepresentation eller tolkas i en interpretator.

**Maskinkod** är den binära kod, bestående av olika instruktioner som en dator direkt kan tolka och genomföra.

**Plattformsoberoende kod** eller **mellanrepresentation** är en typ av kod som kan köras på en virtuell maskin på datorn istället för att köras direkt på datorn genom maskinkod.

**Källkod** eller **källprogram** är ett datorprogram representerat som det programmeringsspråk det från början var skrive i.

**Program** eller **datorprogram** är vilket som helst program som en dator kan tolka och genomföra. Detta innefattar maskinkod men även källkod och olika typer av plattformsoberoende kod.

**Virtuell maskin** är en virtuell dator som i programmeringssyfte används för att skapa en miljö där plattformsoberoende kod eller olika mellanrepresentationer kan köras så som om dess kod var äkta maskinkod.

**Interpretator** eller **programtolk** är ett datorprogram som tolkar och genomför de instruktioner som beskrivs i programmet direkt utan översättning eller kompilation till maskinkod eller någon mellanrepresentation.

**Kompilator** är ett datorprogram som kan översätta ett program skrivet i källkod till datorprogram som går att köra representerade som maskinkod eller olika mellanrepresentationer.

**Integrerad utvecklingsmiljö** är ett datorprogram som innehåller ett antal olika verktyg för att arbeta med exempelvis programmering. Detta kan inkludera exempelvis en textredigerare, visuell programmeringsmiljö och en kompilator eller interpretator.

**Datastruktur** är en abstraktion av flera olika datavärden så som exempelvis en lista, ett träd eller en tabell. Detta görs för att datorn effektivt skall kunna arbeta med icke-primitiva datavärden och för att förenkla användandet.

## Programmeringsspråk

I grunden kan alla datorprocessorer tolka vissa instruktioner vid namn maskinkod (”maskinkod”, 2021). Allt en dator gör bygger på dessa, ofta mycket enkla instruktioner, representerade binärt i datorns minne. Maskinkoden kan vara svåra att läsa och skriva som människa. Komplexiteten för program skrivna i maskinkod snabbt ökar för större program, detta på grund av maskinkodens närhet till hårdvaran som tolkar programmet. För att tackla detta problem så har olika programmeringsspråk skapats för att göra programmeringen av datorer mer abstrakt och därmed möjliggjort mer avancerad, effektivare och enklare programmering.

Källkod är den centrala delen i ett programmeringsspråk som representerar det som ett datorprogram gör och är skriven för hand. Vanligtvis består källkoden av olika kodord, operationer, uttryck och värden i en text. Dessa kodord, operationer, uttryck och värden tolkas sedan på olika sätt av programmeringsspråket för att till vara körbart av en dator som maskinkod.

När man talar om programmeringsspråk talar man ofta om olika hög- eller lågnivåspråk. Detta är ett koncept som används för att definiera hur nära källkoden är till den slutgiltiga maskinkoden. Historiskt tidiga programmeringsspråk är ofta närmare maskinkod och skulle därmed kunna kallas låga medan moderna ofta är väldigt abstraherade från den faktiska maskinkoden som körs och därmed är högnivåspråk. Fördelen med lågnivåspråk är just dess närhet till hur en dator fungerar och tänker vilket gör att det går att skriva mycket effektiv kod, ett bra exempel på ett sådant språk är assembler (”assemblerspråk”, 2021) vilket är ett samlingsnamn för språk där maskininstruktioner är översatta till ord och värden för olika hårdvara. Till skillnad från lågnivåspråk är högnivåspråk mycket enklare för utvecklaren att skriva samt förstå och är de vanligast använda (Carbonnelle, 2021; *TIOBE Index*, 2021).

## En global efterfrågan för programmerare

Programmering som yrke har under de senaste hundra åren växt fram otroligt snabbt och behovet för mjukvaruutvecklare är idag mycket stort. I endast Sverige beräknas det saknas omring 70000 personer för att möta det växande behovet inom IT branshen (*IT-Kompetensbristen*, 2020) till år 2024. I en artikel från mjukvaruutvecklarföretaget Future Processing (*How Many Developers Are There In The World In 2021?*, 2021) skriver dom om den förväntade utvecklingen samt nuvarande efterfrågan för utvecklare. I år, 2021, uppskattas det finnas omkring 26,9 miljoner mjukvaruutvecklare globalt och ökas till kring 45 miljoner år 2030.

I en årlig undersökning av internet- och mjukvaruutvecklingsforumet Stack Overflow har man frågat 80000 utvecklare, studenter och andra användare om bland annat utbilding, land, erfarenhet och andra relevanta frågor (*Stack Overflow Developer Survey 2021*, 2021). I undersökningens statistik om vart undersökningsdeltagarna bor är det endast kring 27,31%[[1]](#footnote-23) som bor i länder där engelska är ett officielt förstaspråk. av dom mer än 37 olika programmeringsspråken som det deltagarna i undersökningen har använt sig av som svar i olika frågor är endast ett språk[[2]](#footnote-24) ej baserat på engelskan.

Att programmeringsspråk inte vanligtvis är skrivna i användarens modersmål utan på engelska skulle kunna påverka spridningen, utbildningen och användingen av programmeringsspråk i icke engelskspråkiga länder[[3]](#footnote-25).

## Block- och flödesprogrammering

I ett programmeringsspråk där översättning till andra språk prioriteras finns det ett antal olika tillvägagångssätt, alla med olika för- och nackdelar.

Blockprogrammering är bland det vanligaste av de olika typerna av översättningsbara programmeringsspråk. Dessa språk är ofta till för utbildningssyfte och fungerar på sätt att de består av block, liknande dom i Lego. Blocken kan ha olika funktioner som till exempel värden, uttryck eller olika satser. Man bygger ett program genom att koppla ihop dessa block i en specialbyggd programmeringsmiljö. De kändaste och mest använda programmeringsspråket (*Scratch - Statistics*, 2021) implementerat på detta vis är Scratch (*Scratch - About*, 2021), originellt utvecklat av MIT senare Scratch Foundation. Detta språk utvecklades i utbildningssyfte med översättning och lättanvändning i åtanke och dess målgrupp är framförallt grundskolebarn.

Flödesprogrammeringsspråk är egentligt på många sätt lika de block baserade programmeringsspråk i att dem använder sig av visuella block. I ett flödesbaserat språk är dock dessa block inte sekventiella utan ihopkopplade med sladdar som representerar ett värdes flöde genom programmet. Spelmotorer som Unreal (*Blueprint Visual Scripting*, 2021), Unity (*Bolt Visual Scripting Unity*, 2021) och Godot (Godot Engine contributors, 2021) samt 3d modelleringsprogram som Blender (*Introduction — Blender Manual*, 2021) använder sig av nodbaserad programmering för ett enklare alternativ till traditionell textbaserad programmering. Detta alternativ är likt blockprogrammering i det att etiketterna och texterna på noderna är översättningsbara.

## Textprogrammering

Ett annat alternativ till visuella programmeringsspråk så som de block- och flödesbaserade språken tidigare nämnda är ett mer traditionellt textbaserat språk. Det finns många textbaserade programmeringsspråk varav dom flesta är av den typen men väldigt få uppfyller kravet att vara översättningsbart. Vad som menas med ett textbaserat programmeringsspråk är ett språk varav programmen består av vanliga tecken och bokstäver men som följer en viss grammatik och syntax, likt hur naturliga språk också följer en grammatik.

Ett tidigt exempel på ett översättningsbart programmeringsspråk är det av ALGOL 68 (Wikipedia contributors, 2021a). Detta språk standardiserades (Van Wijngaarden m.fl., 1968) och dess standard publicerades i flera olika naturliga språk. Denna standard översattes till ryska, tyska, franska, bulgariska, kinesiska och senare japanska samt en för dess använding i blindskrift. Detta ledde till att standarden antogs och accepterades av både UNESCOs organisation IFIP samt Sovjets och senare Rysslands standardorganisation.

Citrine (Mooij (Gabor), 2021) är ett programmeringsspråk där lokalisation är en av kärnfunktionerna vilket har lett till dess översättning till 111 olika naturliga språk. Språket lokaliserar nyckelord, nummer och skiljetecken. För att konvertera mellan olika språk kan användaren själv översätta programmet men inget inbyggt verktyg i programmet verkar göra detta åt användaren. Detta då alla olika naturliga språk som Citrine stödjer publiceras som separata program utan vetskapen om hur man skulle översätta ett program från ett till ett annat naturligt språk.

| Skriftspråk | Exempelkod |
| --- | --- |
| Engelska | write: ‘Hello World’, stop. |
| Svenska | skriva: ‘Hej Världen’, sluta. |
| Tyska | schreiben: ‘Hallo Welt’, stop. |

Scheme (*The Scheme Programming Language*, 2003) är ännu ett standardiserat programmeringsspråk med möjlighet till internationalisation. Detta är dock inte en kärnfunktion i språket utan har istället utvecklats av användare som har använt språkets flexibilitet för att skapa ett bibliotek ("metaphorm", 2021) där olika översättningar finns. Eftersom olika språk kan laddas dynamisk går Scheme program att vara flerspråkiga. Detta leder dock till den nackdelen att språket ej enkelt kan översättas då flera olika språk skulle kunna existera i samma program samt det faktum att språket inte är byggt med översättning, lokalisation eller internationalisation som en kärnfunktion.

| Skriftspråk | Exempelkod |
| --- | --- |
| Engelska | (display "Hello World") |
| Svenska | (visa "Hej Världen") |
| Tyska | (anzeige "Hallo Welt") |

## Symbolprogrammering

Det sista typen som är relevant som ett alternativ för att skapa ett översättningsbart eller internationellt programmeringsspråk är den av vad jag väljer att kalla symbolprogrammeringsspråk. Detta då dom använder sig av symboler istället för nyckelord vilket leder till att det inte är bundet till ett naturligt språk. Ett exempel på ett sådant språk är exempelvis APL (Iverson, 1962) men även helt vanlig matematisk notation (Helmenstine, 2019).

## Ett programmeringsspråks uppbyggnad

Ett programmeringsspråk är egentligen i sig ett datorprogram vars uppgift är att översätta en viss inmatning till ett program som datorn kan köra. Detta görs genom att dela upp uppgiften i ett antal olika delar. Det kan se olika ut för olika språk och tillvägagångssätt men generellt delar man in det i tre större delar: lexikalanalys (”lexikalanalys”, 2021), syntaxanalys (”syntaxanalys”, 2021) och till sist kompilation (”kompilator”, 2021) eller interpration (”interpretator”, 2021). Man har programmerat dessa delar i ett annat[[4]](#footnote-30) programmeringsspråk vilket har skapat ett program som kan genomföra till exempel lexikalanalys, syntax analys och till sist kompilation för att sedan producera det slutgiltiga programmet.

### Lexikalanalys

Lexikalanalysen är oftast första steget i programmet och delar upp och klassificerar källkoden till en lista av lexikala element så som nyckelord, värden, operatorer och skiljetecken. Exempelvis skulle en sträng så som “(visa "Hej Världen")” delas upp i följande element:

| Elementtyp | Värde |
| --- | --- |
| Skiljetecken | ( |
| Nyckelord | visa |
| Sträng | Hej Världen |
| Skiljetecken | ) |

### Syntaxanalys

För att konvertera de lexikala elementen till något som datorn effektivt kan använda och “förstå” krävs ytterligare analys, så kallad syntaxanalys. Detta innebär vanligtvis att dessa lexikala element analyseras utifrån en formell grammatik för att producera en datastruktur som representerar programmets uppbyggnad samt validerar språkets grammatik.

Vanligtvis så skapar syntaxanalysen ett syntaxträd som representerar programmets uppbyggnad på ett mer konkret sätt än en lista av lexikala element. Detta “träd” byggs upp utav olika syntaxnoder där varje nod är som en förgrening alternativt slutet på en gren i ett träd.

### Formell grammatik

Ett formellt språk eller system defineras som en delmängd av en ändlig uppsättning av tecken (i många fall är denna uppsättning av tecken de lexikala element). Vad som ingår i denna delmängd är definerat av det formella språkets grammatik, en så kallad formell grammatik. Grammatiken kan vara beskrivas på flera olika sätt, bland annat genom ett formellt system så som EBNF, kort för Extended Backus-Naur-form Pattis (2015) eller informellt genom en skriftlig beskrivning eller instruktion.

Grammatiken av exempelvis ett formellt språk som beskriver matematisk heltals aritmetik skulle följande formell grammatik gälla, i detta fall i form av en EBNF definition samt en informell skriftlig definition som förklarar EBNF definitionen. Även en visuell representation finns i figur 1.

siffra = "0" | "1" | "2" | "3" | "4"  
 | "5" | "6" | "7" | "8" | "9"  
nummer = "-"? siffra+  
operator = "+" | "-" | "×" | "÷"  
operation = uttryck operator uttryck  
gruppering = "(" uttryck ")"  
uttryck = nummer | operation | gruppering

Ovan definition går att beskriva som följande med ord:

* En siffra defineras som någon utav tecknen för siffrorna 0 till 9
* Ett nummer defineras som först ett valfritt minus tecken följt utav en eller flera siffror
* En operator defineras som ett plus, minus, gånger eller divisions tecken
* En operation defineras som ett uttryck följt av en operator och sedan ett till uttryck
* En gruppering defineras som ett uttryck inom paranteser
* Ett uttryck defineras som antingen ett nummer, operation eller gruppering

Med hjälp av denna definiton går det att analysera en lista av tecken eller lexikala element för att bygga ett syntaxträd. Exempelvis skulle ett uttryck som “123 + (-456 \* 789)” skapa det syntaxträd som finns i figur 2. Hur man genomför denna analys finns det ett antal olika sätt men vanligtvis delar man in syntaxanalysmetoderna i två familjer: “top-down” respektive “bottom-up” metoder (Lunell, 1991).

### Kompilation eller interpretation

(”träd”, 2021) kompileras, det vill säga konverteras till maskinkod eller interpreteras, det vill säga tolka och utföra instruktionerna angivna datastrukturen.

# Syfte och Frågeställning

Målet med projektet är att utveckla ett prototypprogram samt specifikation av ett programmeringsspråk. Denna prototyp skall vara översättningsbart och lättförståeligt. Syftet med detta är för att undersöka möjligheten och olika tillvägagångssätt för att skapa ett programmeringsspråk som går att översätta till olika språk och fortfarande vara enkelt att förstå.

# Metod

## Val av programmeringsspråkstyp

I valet av programmeringsspråkstyp finns det flera olika alternativ som på olika sätt uppnår det konstaterade målet.

Problemet som uppkommer för både flödes- och blockbaserad programmering är det att dom kräver en integrerad utvecklingsmiljö så som en hemsida för Scratch eller respektive programs skrivbordsapplikation för Unreal, Unity, Godot och Blender. Att utveckla en integrerad utvecklingsmiljö utöver ett programmeringsspråk och dess beståndsdelar skulle utgöra ytterligare problem och är egentligen utanför projektets omfattning.

## Verktyg och implementationsspråk

Det programmerings som valdes som implementationsspråk för prototyp implementationen av programmeringsspråket var TypeScript (Bierman m.fl., 2014), ett JavaScript baserat språk med tillägget av explicita datatyper. Anledningen till detta val var en avvägning mellan simplicitet, abstraktion, prestanda och möjligheten till plattformsoberoende kod för att göra språket körbart även i en webbläsare. Stöd för körning utav programmeringsspråket i både webbläsare och som program görs genom ett verktyg och körtidsmiljö vid namn Deno (Dahl, 2018).

## Planering och struktur

Programmeringsspråkets lexikal-, syntax- och semantiskaanalys planerades och specifierades med hjälp av EBNF. Även en kompilator till JavaScript samt en översättare planerades. JavaScript valdes som kompilationsmål på grund av dess liknande struktur till det planerade språket samt dess stöd på dem flesta datorplatformerna.

Översättaren planerades fungera endast på lexikal nivå, det vill säga den är kontextfri och ej bryr sig om exempelvis ordning eller elementtyp. Detta går eftersom den endast planeras översätta lexikalelement av typen “nyckelord” som vanligtvis är kontextfria[[5]](#footnote-40).

# Genomförande

## Specifikation

En specifikation som beskriver programmeringsspråkets syntax och grammatik skapades i EBNF format, delar av denna specifikation är dock beroende utav det skriftspråk som önskas användas för språkets nyckelord.

### Värden

De värdetyper som valdes till specifikationen var "boolesk", "nummer", "sträng" och "inget". Detta är endast ett litet urval utav de traditionella värdetyperna i ett programmeringsspråk för att avgränsa specifikationens och implementationens komplexitet. Dessa fyra värdetyper är grunden till alla algoritmer, uttryck och satser som programmeringsspråket kan processera. EBNF representationen är följande:

boolesk = sant | falsk  
  
siffra = "0" | "1" | "2" | "3" | "4"  
 | "5" | "6" | "7" | "8" | "9"  
nummer = siffra+ "." {siffra+}  
  
sträng\_värde = valfri\_karaktär\*  
sträng = '"' (sträng\_värde - "\"?) '"'  
  
värde = boolesk | nummer | sträng | inget\_nyckelord

### Identifierare

En identifierare är en textuell referens eller ett namn till en funktion, variabel eller konstant. Detta skulle exempelvis kunna vara variabel- och funktionsnamn inom matematiken så som “x” eller “pi”. Detta språk definerar en identifierare som en sekvens som börjar med en karaktär i unicode kategorin bokstav följt utav något av unicode kategorierna bokstav, symbol eller nummer.

identifierare = unicode\_bokstav  
 (unicode\_bokstav  
 | unicode\_symbol  
 | unicode\_nummer)\*

### Kommentarer

Kommentarer är inom programmeringsspråket är ett sätt för användaren att kommentera kod med text som ignoreras utav implementationen. Dessa är definerade på samma sätt som i många andra språk så som C, JavaScript och Rust. Det finns två typer av kommentarer: en-rads- och fler-rads-kommentarer. En-rads-kommentarer börjar där användaren har skrivit “//” och slutar vid radens slut medan fler-rads-kommentarer börjar vid “/\*” och slutar vid “\*/”. Båda dessa kommentarstyperna ignorerar allt inom dessa avgränsare.

kommentar = en\_rads\_kommentar | fler\_rads\_kommentar  
en\_rads\_kommentar = "//" valfri\_karaktär\* ny\_rad  
fler\_rads\_kommentar = "/\*" valfri\_karaktär\* "\*/"

### Makron

Ett makro används i programmeringsspråket som ett sätt att kommunicera med kompilatorn för att exempelvis importera kod skriven i andra programmeringsspråk än detta. Likt kommentarer finns två olika typer av makron, en-rads-makron och fler-rads-makron. Dessa består utav en typidentifierare och ett värde. Typidentifieraren är till för att tipsa kompilatorn om hur makrots värde skall tolkas. Exempelvis genom att importera makrots värde som obearbetad kod.

makro = en\_rads\_makro | fler\_rads\_makro  
makro\_identifierare = valfri\_karaktär\*  
makro\_värde = valfri\_karaktär\*  
en\_rads\_makro = "#(" makro\_identifierare ")" makro\_värde ny\_rad  
en\_rads\_makro = "#(" makro\_identifierare ")" "{" makro\_värde "}"

### Operatorer

Operatorerna som är definerade för språket agerar endast på nummer- och booleskvärden på samma sätt som dess motsvarigheter i matematiken.

| Symbol | Beskrivning |
| --- | --- |
| = | Anger en variabels värde |
| + | Addition vid operationer med två nummervärde, annars för att markera att ett nummervärde är positivt |
| - | Subtraktion vid operationer med två nummervärde, annars för att markera att ett nummervärde är negativt |
| \* | Multiplikation av två nummervärden |
| / | Division av två nummervärden |
| % | Restprodukten av två nummervärden vid division |
| == | Jämför två värden för likhet |
| != | Jämför två värden för olikhet |
| < | Jämför ifall det första nummervärdet är mindre än det andra nummervärdet |
| <= | Jämför ifall det första nummervärdet är mindre eller lika med än det andra nummervärdet |
| > | Jämför ifall det första nummervärdet är större än det andra nummervärdet |
| >= | Jämför ifall det första nummervärdet är större eller lika med än det andra nummervärdet |
| ! | Inverterar en boolesk, dvs gör om sanna booleskvärden till falska och falska till sanna |
| || | Boolesk eller operation som tar två booleskvärden och returnerar sant ifall något av värdena är sanna |
| && | Boolesk eller operation som tar två booleskvärden och returnerar sant ifall båda värdena är sanna |

EBNF definitionen är densamma som symbolerna i tabellen.

### Nyckelord

Nyckelorden i programmeringsspråket används för att markera olika uttryck och satser. Dessa nyckelord är dock skriftspråksspecifika vilket gör att språkets definition ändras beroende på skriftspråk.

| Engelska | Svenska | Tyska | Franska |
| --- | --- | --- | --- |
| import | importera | importieren | import |
| from | från | aus | des |
| export | exportera | exportieren | export |
| function | funktion | funktion | function |
| return | returnera | rückkehr | retourne |
| if | om | ob | si |
| while | medan | während | pendant |
| break | avbryt | abbrechen | casse |
| continue | fortsätt | fortsetzen | continue |
| variable | variabel | variable | variable |
| constant | konstant | konstante | constant |
| none | inget | null | rien |
| true | sant | wahr | vrai |
| false | falskt | falsch | faux |

EBNF definitionen är densamma som orden i kolumnen för det skriftspråk man använder.

### Uttryck

#### Binära- och unärauttryck

Binära och unärauttryck är en grupp av olika operationer som utförs på ett, två eller flera uttryck. Denna typ utav uttryck kräver en operator och defineras som följande:

binärt\_uttryck = uttryck binär\_operator uttryck  
unärt\_uttryck = unär\_operator uttryck

#### Villkorsuttryck

Villkorsuttryck används för att i uttryck välja antingen ett uttryck eller det andra beroende på om ett villkor (booleskt uttryck) är sant eller falskt likt en om-sats.

villkor = uttryck  
uttryck\_om\_sant = uttryck  
uttryck\_om\_falskt = uttryck  
villkors\_uttryck = om\_nyckelord villkor  
 uttryck\_om\_sant  
 annars\_nyckelord  
 uttryck\_om\_falskt

#### Funktionsanrop

Funktionsanrop är likt funktioner i matten anrop till tidigare definerade funktioner, dessa anrop kan ta noll eller flera uttryck som argument.

funktions\_argument = uttryck  
funktions\_anrop = identifierare  
 "(" funktions\_argument  
 ("," funktions\_argument)\* ")"

#### Variabelåtkomst

Variabel- och konstantåtkomst görs genom att specifiera variabeln eller konstantens identifierare.

variabel\_åtkomst = identifierare

#### Gruppering

En gruppering används i ett uttryck för att markera vilken ordning ett uttryck utförs, likt hur i matematiken paranteser används för prioritering. Det är även värt att notera att även operatorerna följer prioriteringsreglerna.

gruppering = "(" uttryck ")"

#### Sammanfattning av uttryck

Sammanfattningsvis kan ett uttryck i programmeringsspråket defineras som något av ovan definerade kategorierna eller ett värde.

uttryck = unärt\_uttryck  
 | binärt\_uttryck  
 | vilkors\_uttryck  
 | funktions\_anrop  
 | variabel\_åtkomst  
 | gruppering  
 | värde

### Satser

#### Villkorssats

Villkors-, om-, eller if-satsen är en konstruktion i programmeringsspråket för att programmera logik och kontrollera flödet av ett program. En if-sats består av ett uttryck som representerar villkoret för att köra dess inre satser. Valfritt är även en annars- eller else-sats som körs ifall villkorsuttrycket ej är sant.

villkor = uttryck  
villkors\_sats = om\_nyckelord "(" villkor ")" sats  
 (annars\_nyckelord sats)?

#### Medan, fortsätt och avbryt

För mer advancerade program finns medan- eller while-satsen med dess tillhörande nyckelord: “fortsätt” och “avbryt”. Denna sats finns likt en villkorssats för att kontrollera flödet av ett program men till skillnad från en villkorssats fortsätter medansatsen att köra dess inre satser tills villkoret är falskt.

villkor = uttryck  
medan\_sats = medan\_nyckelord "(" villkor ")" sats

#### Returnera

En funktion finns som en abstraktion för att återanvända och abstrahera exempelvis algoritmer. För att funktioner ska kunna användas i uttryck och skapa nya värden finns nyckelordet “returnera” för att returnera ett värde från en funktion och avbryta funktionens körning.

returnera = returnera\_nyckelord uttryck

#### Variabel- och konstantdeklaration

För att deklarera variabla och konstanta värden som kan användas i en lokal räckvidd, det vill säga exempelvis det nuvarande kodblocket eller funktionen.

variable\_deklaration = variabel\_nyckelord  
 identifierare "=" uttryck  
konstant\_deklaration = konstant\_nyckelord  
 identifierare "=" uttryck

#### Kodblock

Ett kodblock består av en lista utav satser som alla existerar i kodblockets egna räckvidd. Detta gör att variabler och konstanter, men även värden som är skapta i det lokala kodblocket inte kan användas av något på en högre nivå, som exempelvis ifall kodblocket skulle vara inuti ett annat kodblock.

kod\_block = "{" sats\* "}"

#### Sammanfattning av satser

Satser är sammanfattningsvis alla ovanstående rubriker samt uttryck och makron. Dessa satser används inne i exempelvis funktioner för att skapa logiken som utgör ett program.

sats = villkors\_sats  
 | medan\_sats  
 | returnera  
 | variable\_deklaration  
 | konstant\_deklaration  
 | kod\_block  
 | uttryck  
 | makro

### Program

Ett program är en lista utav syntaxelement på toppnivån. Dem enda elementen som räknas som detta är funktionsdeklarationer och importer. Funktioner är till för att abstahera och binda ihop sammanhängande satser av kod som lätt kan användas flera gånger i koden. Importer är till för att använda extern kod.

funktions\_argument = identifierare  
funktions\_deklaration = exportera\_nyckelord? funktion\_nyckelord  
 identifierare  
 "(" funktions\_argument  
 ("," funktions\_argument)\* ")"  
 kod\_block  
  
import\_element = identifierare  
importera = importera\_nyckelord  
 (import\_element ("," import\_element)\*  
 från\_nyckelord)?  
 sträng  
  
program = funktions\_deklaration | import

## Implementation

Vet typ inte vad jag ska skriva här? Jag programmerade lexikalanalysen, syntaxanalysen, kompilatorn och översättaren för språket.

# Resultat

## Exempel

Exempelkod på alla olika språk här!

# Diskussion och Slutsats

## Icke-europeiska språk

* Funkar kanske inte med right-to-left språk
* Funkar kanske inte med språk som använder tecken

## Läsbarhet

* Vem som helst kan inte läsa det utan en översättare

## Kontext och nyckelord

* problemet med exempelvis franska här, (hur type variable skulle ändras om variabelnamnet var maskulint/feminint)

# Källförtäckning

assemblerspråk. (2021). I *Nationalencyklopedin*. <https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/assemblerspr%C3%A5k>

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. *European Conference on Object-Oriented Programming*, 257–281.

*Blueprint Visual Scripting*. (2021). Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints>

*Bolt Visual Scripting Unity*. (2021). Unity Technologies. <https://unity.com/products/unity-visual-scripting>

Carbonnelle, P. (2021). *PYPL PopularitY of Programming Language index*. <https://pypl.github.io/PYPL.html>

Dahl, R. (2018). *10 Things I Regret About Node.js*. Youtube. <https://youtu.be/M3BM9TB-8yA>

Godot Engine contributors. (2021). *VisualScript*. <https://docs.godotengine.org/en/stable/getting_started/scripting/visual_script/index.html>

Helmenstine, A. M. (2019). Why Mathematics Is a Language. I *ThoughtCo*. <https://www.thoughtco.com/why-mathematics-is-a-language-4158142>

*How Many Developers Are There In The World In 2021?* (2021). Future Processing. <https://www.future-processing.com/blog/how-many-developers-are-there-in-the-world-in-2019>

interpretator. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/interpretator>

*Introduction — Blender Manual*. (2021). Blender Foundation. <https://docs.blender.org/manual/en/latest/render/shader_nodes/introduction.html>

*IT-Kompetensbristen*. (2020). IT&Telekom företagen. <https://www.techsverige.se/2020/12/it-kompetensbristen/>

Iverson, K. E. (1962). *A Programming Language*. John Wiley & Sons, Inc.

kompilator. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/kompilator>

lexikalanalys. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/lexikalanalys>

Lunell, H. (1991). *Kompilatorkonstruktion i teori och praktik*. Studentlitteratur.

maskinkod. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/maskinkod>

"metaphorm". (2021). *international-scheme*. <https://github.com/metaphorm/international-scheme>

Mooij (Gabor), G. J. G. T. de. (2021). *Localized Programming Language Citrine*. <https://citrine-lang.org>

Pattis, R. E. (2015). *EBNF: A Notation to Describe Syntax*.

*Scratch - About*. (2021). Scratch Foundation. <https://scratch.mit.edu/about>

*Scratch - Statistics*. (2021). Scratch Foundation. <https://scratch.mit.edu/statistics>

*Stack Overflow Developer Survey 2021*. (2021). Stack Overflow. <https://insights.stackoverflow.com/survey/2021>

syntaxanalys. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/syntaxanalys>

*The Scheme Programming Language*. (2003). <https://groups.csail.mit.edu/mac/projects/scheme>

*TIOBE Index*. (2021). TIOBE Software. <https://www.tiobe.com/tiobe-index/>

träd. (2021). I *Nationalencyklopedin*. <https://www.ne.se/uppslagsverk/encyklopedi/lång/träd-(2)>

Van Wijngaarden, A., Mailloux, B. J., Peck, J., & Koster, C. H. A. (1968). Draft Report on the Algorithmic Language ALGOL 68. *ALGOL Bull.*, *Sup 26*, 1–84. <https://doi.org/10.5555/1064072.1064073>

Wikipedia contributors. (2021a). *ALGOL 68 — Wikipedia, The Free Encyclopedia*. <https://en.wikipedia.org/w/index.php?title=ALGOL_68&oldid=1053624353>

Wikipedia contributors. (2021b). *Self-hosting (compilers) — Wikipedia, The Free Encyclopedia*. <https://en.wikipedia.org/w/index.php?title=Self-hosting_(compilers)&oldid=1048801699>

# Bilagor

## Figur 1. Diagram av en formell grammtik för heltals aritmetik

| Delmängd/Nodtyp | Diagram |
| --- | --- |
| Siffra | siffra |
| Nummer | nummer |
| Operator | operator |
| Operation | operation |
| Gruppering | gruppering |
| Uttryck | uttryck |

## Figur 2. Syntaxträd utav exempel program i heltals aritmetik

1. Inräknat i denna beräkning är USA med 18,33%, Stor Brittianien med 5,37% samt Kanada med 3,61%. [↑](#footnote-ref-23)
2. Programmeringsspråket APL [↑](#footnote-ref-24)
3. Det fjärde, åttånde och tionde Globala målet: god utbildning för alla, anständiga arbetsvillkor och ekonomisk tillväxt och minskad ojämlikhet. Något mer relevant nu än någonsin med tanke på den statistik som finns kring den växande IT branschen. [↑](#footnote-ref-25)
4. Eller samma, se Wikipedia contributors (2021b) [↑](#footnote-ref-30)
5. Se underrubriken [Kontext och nyckelord](#Kontext-och-nyckelord) för vidare reflektion [↑](#footnote-ref-40)