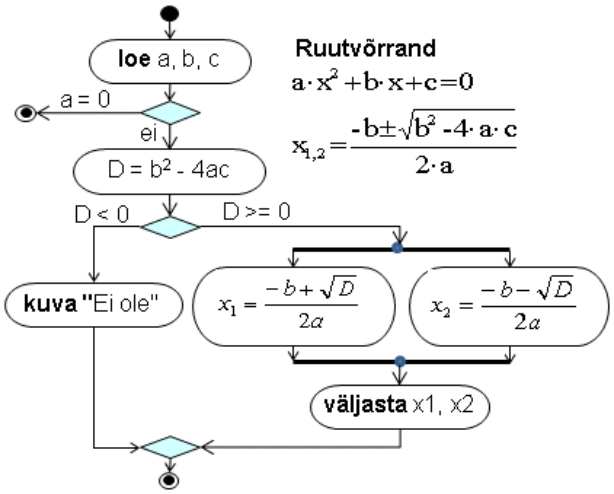
Algoritmimine



Algoritm on täpne ja ühemõtteline eeskiri antud liiki ülesannete lahendamiseks või tegevuste täitmiseks kindla eesmärgi saavutamisel. Algoritm määrab, milliseid tegevusi ja millises järjekorras peab selleks täitma. Mõiste algoritm pärineb matemaatikast, kuid seda kasutatakse ka mujal: retseptid, juhendid, stsenaariumid – kõik need on eeskirjad e algoritmid. Eriti palju kasutatakse seda mõistet seoses rakenduste loomise ja programmeerimisega.

Sisu

Algoritmi olemus

Algoritm on täpne ja ühemõtteline **eeskiri** antud liiki ülesannete lahendamiseks või tegevuste täitmiseks kindla eesmärgi saavutamisel. Algoritm määrab, milliseid tegevusi ja millises järjekorras peab selleks täitma. Mõiste algoritm pärineb matemaatikast, kuid seda kasutatakse ka mujal: retseptid, juhendid, stsenaariumid – kõik need on eeskirjad e algoritmid. Eriti palju kasutatakse seda mõistet seoses rakenduste loomise ja programmeerimisega. Võib öelda, et programmeerimise sisu ja kunst seisneb suurel määral just algoritmide loomises. Programmi teksti kirjutamist nimetatakse kodeerimiseks. Programm on üks võimalikest algoritmi esitusviisidest. Algoritmi sisu ja esitusviis sõltub täitjast, olgu selleks siis inimene, arvuti, robot vms.

Siin arvestatakse, et täitjaks on **programmjuhtimisega seade** (n arvuti) – protsessori ja mäluga varustatud seade, millel võivad olla võimalused (täpne koosseis ei ole oluline) infovahetuseks väliskeskkonnaga. Algoritmi võib sellisel juhul vaadelda loodava programmi mudelina, so tegevuste kirjeldusena üldisel kujul, mis otseselt ei sõltu konkreetsest programmeerimiskeelest. Algoritm on eeskätt mõeldud inimesele ning on orienteeritud ülesande lahendamise sisule ja lahenduse kirjelduse ülevaatlikkusele. Põhimõtteliselt võiks silmas pidada, et algoritmi saaks realiseerida erinevate programmeerimiskeelte abi. Algoritme kasutatakse nii rakenduste loomisel kui ka nende dokumenteerimiseks nt kas või selleks, et neid hiljem realiseerida mõne muu vahendiga.

Rakenduse loomise faasis kasutatakse sageli algoritmi järk-järgulist detailiseerimist. Algfaasis võib algoritm olla üsna üldisel kujul, arenduse käigus see täpsustub. Lõppfaasis peaks algoritm üsna täpselt arvestama kasutatavate andmete liike ja organisatsiooni ning tegevusi, mida arvuti saab täita andmetega protsesside juhtimiseks. Tüüpiliselt detailiseeritakse algoritm tasemele, millelt oleks lihtne koostada programmi.

Andmed algoritmides

Nii nagu programmid, on ka algoritmid ja nende esitus otseselt sõltuvad kasutatavate andmete liigist ja organisatsioonist. Edaspidi eeldame, et arvuti saab salvestada ja töödelda erinevat liiki andmeid: märkandmeid (arvud, tekstid, tõeväärtus jm) ning graafika- ja heliandmeid. Algoritmides saab kasutada muutujaid, andmekogumeid (loendid, massiivid ja tabelid) ja objekte.

Skalaarandmed esitatakse algoritmides konstantidena ja muutujatena. Nende olemus ja käsitlus on põhimõtteliselt sama nagu programmides, kuid nö formalismi on vähem. Näiteks ei kasutada eriti tüüpe (erineva täpsusega täis- ja reaalarvud). Liikidega (arv, tekst, tõeväärtus) peab arvestama näiteks tehete ja funktsioonide kasutamisel. Konstandid kirjutakse otse algoritmi, muutujad esitatakse nimede abil. Võiks arvestada, et algoritmis kasutatavad nimed sobiksid kasutamiseks ka programmis. Muutujate kasutamisel algoritmides peab arvestama, et programmis hakkavad neile vastama mälupesad ning omistamine seisneb väärtuse salvestamises arvuti mälus muutujale eraldatud pesas. Peamine korraldus muutujale väärtuse andmiseks on **omistamine**, ning selle võib esitada kujul:

```
muutuja = avaldis
```

Avaldiste esitamisele algoritmides erilisi nõudeid ja piiranguid ei seata. Oluline on, et see oleks arusaadav inimesele ja vastaks väärtuste liigile. Näiteks aritmeetikatehteid ja matemaatikafunktsioone saab rakendada ainult arvudele.

Andmekogumite puhul arvestame esialgu **ühemõõtmeliste massiivide** ehk **loenditega**, mis on järjestatud elementide (muutujate) kogum. Loend (massiiv) tähistatakse ühe nimega, viitamiseks elementidele kasutatakse nime koos indeksiga: V(1), Y(k), X(i + 1) jmt.

Objektide osas arvestame praegu, et tegemist on süsteemis määratletud ja realiseeritud objektide klassidega (nagu on Scratchis, dokumendipõhistes süsteemides jmt). Siia kuuluvad eeskätt graafikaobjektid. Teatud juhtumeil võivad tulla mängu ka dokumendid ja nende osad, vormid, dialoogiboksid, aknad jms.

Objekti käsitlemisel algoritmis peab mingil viisil viitama objektile, tema omadustele ning meetoditele. Viitamiseks objektile kasutatakse **nime**. Kui algoritm kehtib ainult kindla objekti jaoks (nagu Scratchis), siis võib objekti nime algoritmis jätta ära. Viitamiseks omadustele ja meetoditele kasutatakse määratud, kokkulepitud või üldarusaadavaid nimesid. Meetodite puhul kasutatakse sageli ka argumente.

pall.X = 0; pall.liigu x0, y0; pööra 180; muudaX 10

Sprait
nimi x asukoht, y asukoht suund, suurus värvus, nähtavus, ..
liigu(), pööra() muudaX(), muudaY() vaheta kostüümi() muuda värvi() ütle(), mängi heli(), ..

Näiteks on toodud valik Scratchi põhiobjektide (spraitide) omadusi ja meetodeid. Analoogsed omadused ja meetodid on graafikaobjektidel mitmetes teises süsteemis (omaduste ja meetodite nimed on erinevad). Siin omadused **x asukoht** ja **y asukoht** määravad spraidi asukoha laval (X ja Y on koordinaadid). Omadus **suund** näitab spraidi pööret ning objekti liikumise suunda. Spraidi meetodid on realiseeritud käsuplokkide abil. Üldjuhul ei pea Scratchis realiseerimiseks mõeldud algoritmides tingimata kasutama täpselt käsuplokkide nimesid, kuid teatud määral tasub sellega siiski arvestada, kui algoritmi detailiseerimisel on jõutud viimasele tasemele. Algoritmi koostaja võib spetsifitseerida tegevused ja nende sisu. Koostaja võib ise määratleda ka oma objektid (**klassid**) ja spetsifitseerida nende jaoks omadused ja meetodid.

Üldiselt peaks algoritmi juurde kuuluma andmete spetsifikatsioon, milles näidatakse kasutatavad muutujad, andmekogumid ja objektid.

Algoritmide esitusviisid

Algoritmide esitamiseks kasutatakse erinevaid viise ja vahendeid:

- tavaline tekst
- valemid
- plokkskeemid
- **UML**i tegevusdiagrammid
- algoritmikeeled

Antud materjalis kasutatakse peamiselt **UML**i tegevusdiagramme ja algoritmikeelt. **UML**i diagrammides kasutatakse teatud standardseid sümboleid ja kujundeid. Antud juhul kõiki standardite detaile väga täpselt ei arvestata. Peamised tegevusdiagrammide komponendid on järgmised:

● **protsessi** (protseduuri, skripti) **algus** ◉ **protsessi lõpp** (katkestus)

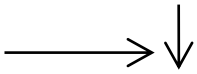
tegevus — Tegevusskeemide põhielement. Täidetav tegevus. Võib esitada erineva detailsusastmega

leia hind | koosta arve | lahenda võrrand | arvuta pindala

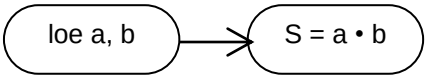
pall platsi keskele | pall lenda | **liigu** 20 sammu | **pööra** 30 kraadi

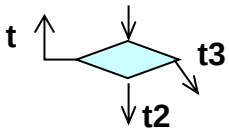
loe a, b, c | **arvuta** S, P | **kirjuta** S, P | **kuva** teade

S = 2πr(r + h) | D = b2 – 4ac | nimi = „Juku“ | S = 0; k = 0



Siire. Näitab järgmist tegevust.

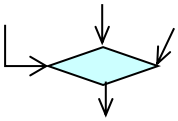




Hargnemine

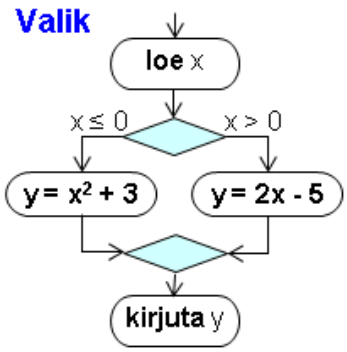
Tüüpiliselt üks sisend ja mitu väljundit

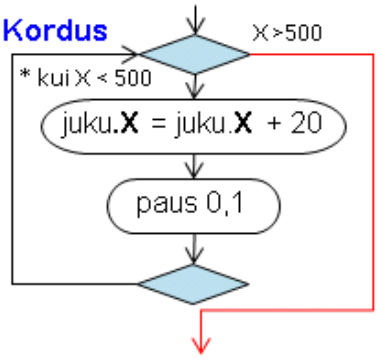
t1, t2, t3 – tingimused.



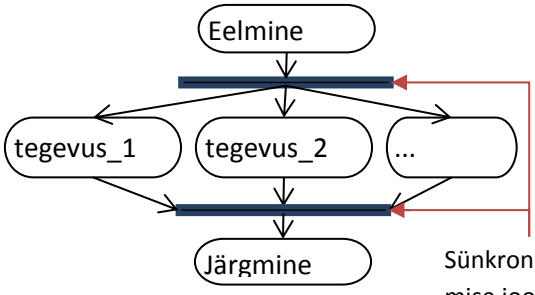
Ühinemine

Tüüpiliselt mitu sisendit ja üks väljund





Paralleelprotsessid ja tegevuste sünkroniseerimine



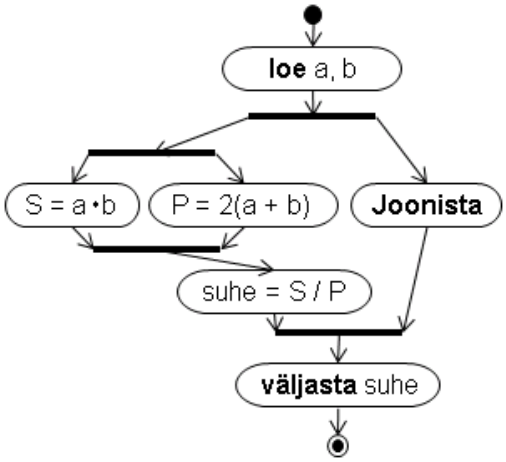
Sünkroniseerimise jooned

Tegevusi võib täita **paralleelselt**. Seda ei pea tegema, kui süsteem ei toeta paralleelsust või sellest ei ole erilist kasu. Ühtlasi tähendab see, et tegevuste täitmise järjekord pole oluline!

Algoritmikeeles esitatakse tegevused sarnaselt programmeerimiskeelte lausetele, kuid olulisemalt nõrgema formalismiga ning nõuetega süntaksi õigsusele. Tegemist on kokkuleppega, mida rakendatakse algoritmide kirjeldamiseks nt töömeeskonnas, koolis või õpetamisel. Kasutatakse mõningaid kindla tähendusega sõnu ja fraase nagu nt võtmesõnad programmeerimiskeeltes või ka tegevusskeemides: **loe**, **kirjuta**, **kui**, **kordus** …

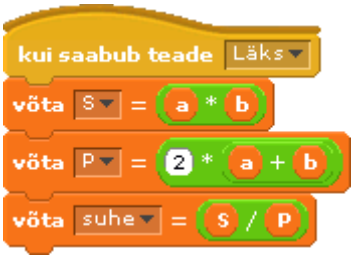
Allpool on mõned algoritmide näited, mis on esitatud **UML** tegevusskeemide ja algoritmikeele abil.

Antud on ristküliku külgede pikkused (a, b). Leida selle pindala (S), ümbermõõt (P) ning pindala ja ümbermõõdu suhe. Joonistada ka ristkülik.



protseduur **Joonista**
mine 0, 0
pliiats alla
liigu a, 0
liigu a, b
liigu 0, b
liigu 0, 0

protseduur **Rist**
loe **a, b**
teavita **Joonista**
S = a * b
P = 2(a + b)
suhe = S / P
väljasta **suhe**

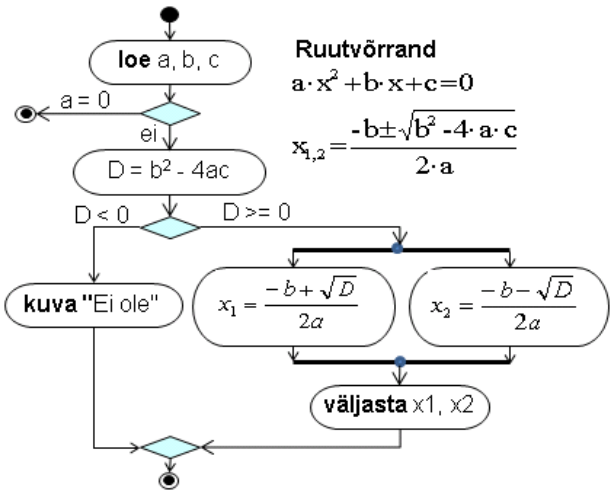


UML diagramm näitab, et arvutused ja joonistamine võivad toimuda paralleelselt, kuid enne peab lugema algandmed (**a**, **b**). Sellise variandi võimaldab realiseerida Scratch. Peale algandmete lugemist käivitatakse teatega **Läks** samaaegselt joonestamise ja arvutamise skriptid. Need võivad töötada paralleelselt, sest ei sõltu üksteisest.

Antud ülesande juures aja kokkuhoidu praktiliselt ei teki ning lähenemisviis demonstreerib lihtsalt põhimõttelisi võimalusi. Skeem näitab, et ka **S** ja **P** arvutused võivad toimuda paralleelselt. Seda realiseerida ei ole mingit mõtet. Kuid taoline esitus näitab, et nende kahe suuruse arvutamise järjekord ei ole oluline: **S** ja **P** arvutamise käsuplokkide järjekord võib olla programmis suvaline, kuid mõlemad peavad eelnema muutuja **suhe** väärtuse leidmisele.

Muutuja **suhe** väärtuse väljastamiseks ei ole joonestamise lõpetamise ootamine tingimata vajalik, see võib toimuda kohe peale arvutuste lõppu. Võite vaadata ka Scratchi projekti, kus on mõningaid täiendusi joonise mastaabi valimisel – see ei ole kajastatud algoritmis.

Järgnevalt on esitatud ruutvõrrandi lahendamise algoritm. Peale algandmete (a, b, c) lugemist toimub kontroll, kas **a=0**; kui jah, siis programmi töö katkestatakse. Jätkamisel leitakse kõigepealt **D** väärtus. Kui **D<0**, lahend puudub, vastupidisel juhul leitakse ja väljastatakse juured **x1** ja **x2**. Vaatamiseks on demo.



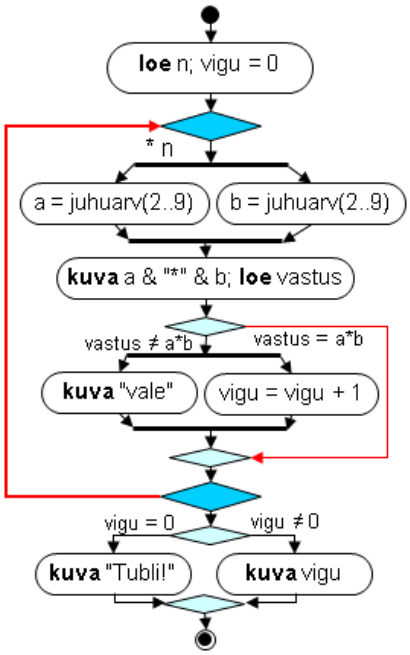
Ruutvõrrand

$$a \cdot x^2 + b \cdot x + c = 0$$

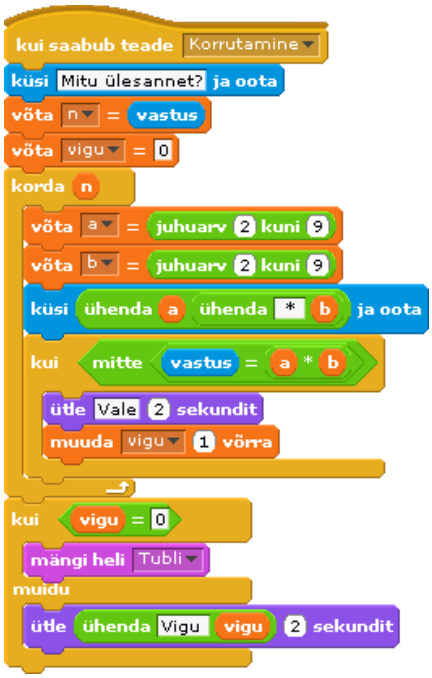
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

```
protseduur Ruutvrd
  loe a, b, c
  kui a = 0 siis lõpp
  D = b² - 4ac
  kui D < 0 siis
    kuva "Juured puuduvad"
  muidu
    x1 = (-b + √D) / 2a   x2 = (-b - √D) / 2a
    kuva x1, x2
  lõpp kui
lõpp Ruutvrd
```

Allpool on **algoritm** ja Scratchi **skript** rakenduse jaoks, mis esitab **n korrutamisülesannet**, kontrollib kasutaja vastuseid ja annab hinnangu. Vt ka Scratchi projekti.




```
protseduur Korrutamine
  loe n
  vigu = 0
  kordus n korda
      a ja b juhuarvud 2..9
      kuva küsimus
      loe vastus
      kui vastus <> a* b siis
        kuva "Vale"
        vigu = vigu + 1
  lõpp kordus
  kui vigu = 0 siis
      kuva  "Tubli!"
  muidu
      kuva vigu
  lõpp kui
```



Tüüptegevused märkandmetega

UML	Algoritmikeel	Scratch	Visual Basic (VB) Python

Väärtuste leidmine ja salvestamine (omistamine)

Leitakse avaldise väärtus ja salvestatakse see antud muutujas

muutuja = avaldis	muutuja = avaldis		muutuja = avaldis

muutuja – lihtmuutuja nimi, massiivi element: nimi(indeks), nimi(rida, veerg)
avaldis – eeskiri väärtuse leidmiseks: operandid, tehtemärgid, funktsioonid
arvavaldised, tekstavaldised, ajaavaldised, võrdlused, loogikaavaldised

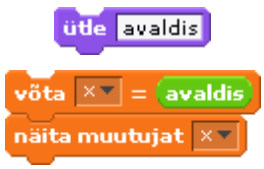
Väärtuste lugemine väliskeskkonnast

Loetakse väärtused väliskeskkonnast (boks, vorm, dokument, fail, …) ja salvestatakse muutujates

sisesta muutuja,	**küsi** muutuja,… **loe** muutuja,… **sisesta** muutuja, …		**Visual Basic** muut = **InputBox**() **Python** muut = **input**()

Väärtuste kirjutamine väliskeskkonda

Leitakse avaldise väärtus ja kuvatakse või kirjutakse väliskeskkonda (boks, vorm, dokument, fail,…)

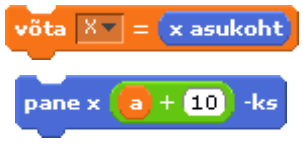
väljasta avaldis	**kuva** avaldis **kirjuta** avaldis		**Visual Basic** **MsgBox** avaldis **Python** **print**(arg, …)

Tüüptegevused objektidega

UML	Algoritmikeel	Scratch	Visual Basic Python

Objektide omaduste lugemine ja muutmine

Viitamiseks objekti omadusele kasutatakse tüüpiliselt konstruktsiooni **objekt.omadus**

muutuja = objekt.omadus objekt.omadus=avaldis	muutuja = objekt.omadus objekt.omadus = avaldis		X = auto.Left auto.Left = a+10

Objektide meetodite rakendamine

Viitamiseks objekti meetodile kasutatakse tüüpiliselt konstruktsiooni **objekt.meetod *argumendid***

objekt.meetod arg-d	**objekt.meetod arg_d**		auto.IncrementLeft 10 auto.setx(120)

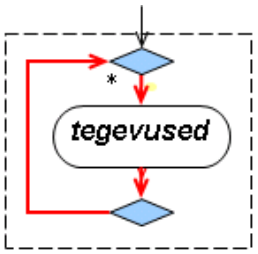
Protsesside juhtimine

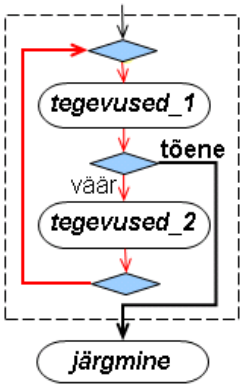
Protsesside juhtimine seisneb tegevuste täitmise järjekorra määramises. Eristatakse nelja liiki protsesse:

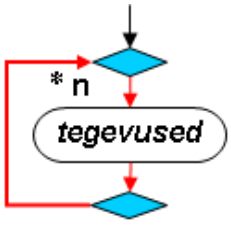
- järjestikune protsess ehk jada
- tsükliline protsess ehk kordus
- hargnev protsess ehk valik
- paralleelne protsess.

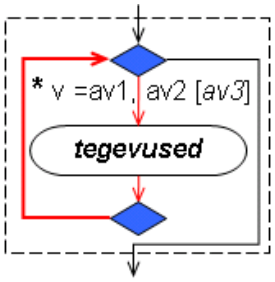
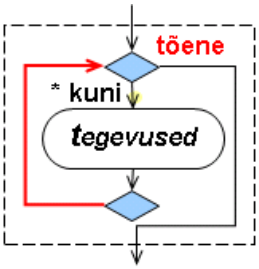
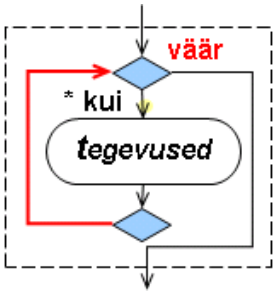
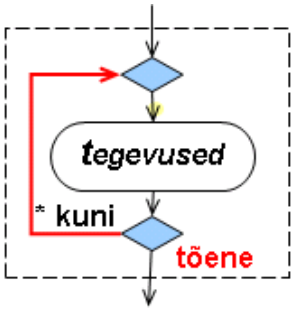
Nende kirjeldamiseks kasutatakse vastavaid kokkuleppeid, korraldusi või käske. Järjestikuste protsesside määramiseks mingeid spetsiaalseid vahendeid ei kasutada: eeldatakse, et tegevusi täidetakse selles järjekorras nagu need on esitatud algoritmis.

Kordused

UML	Algoritmikeel	Scratch	Visual Basic	Python
Lõputu kordus				
	kordus *tegevused* lõpp kordus		Do *laused* Loop	while True: *laused*
Lõputu kordus katkestusega				

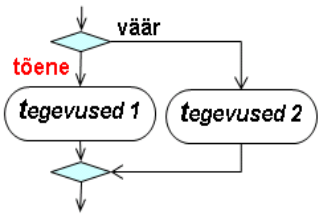
	kordus
	tegevused 1
	kui *ting* siis välju
	tegevused 2
	lõpp kordus
	NB! Katkestamise tingimusi võib olla mitu!
			Do
			laused 1
			If *ting* Then
			Exit Do
			End If
			laused 2
			Loop
				while True:
				laused 1
				if ting:
				break
				laused 2

Etteantud korduste arvuga kordus				
	kordus *n* korda *tegevused* lõpp kordus **NB!** Tegevuste seas võivad olla katkestajad!		For i = 1 T0 n *laused* Next i **NB!** VBs näitab käsu jätkumist _	for i in \ range(n): *laused* **NB!** Pythonis näitab käsu jätkumist \

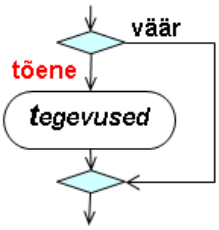
UML	Algoritmikeel	Scratch	Visual Basic	Python
Juhtmuutujaga kordus				
	kordus *v* = a1..a2 [,a3] *tegevused* lõpp kordus Juhtmuutuja ***v*** saab väärtusi a1-st a2-ni sammuga a3	Puudub. Saab modelleerida näiteks taoliselt 	For v=a1 To a2 _ [Step a3] *laused* Next v **NB!** VBs näitab käsu jätkumist _	for v in \ range \ (a1,a2,a3): *laused* **NB!** Pythonis näitab käsu jätkumist ****
Eelkontrolliga kordus kuni tingimus saab tõeseks				
	kordus kuni *ting* *tegevused* lõpp kordus korratakse, kuni tingimus **saab tõeseks**		Do Until *ting* *laused* Loop	while not *ting*: *laused*
Eelkontrolliga kordus kui tingimus on tõene				
	kordus kui *ting* *tegevused* lõpp kordus korratakse, kui tingimus **on tõene**	puudub	Do While *ting* *laused* Loop	while *ting:* *laused*
Järelkontrolliga kordus kuni tingimus saab tõeseks				
	kordus *tegevused* lõpp kuni *ting* korratakse, kui tingimus pole tõene (kuni saab tõeseks)	puudub	Do *laused* Loop Until *ting*	puudub

Valikud

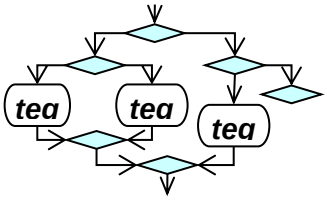
UML	Algokeel	Scratch	Visual Basic	Python
Valik kahest				

	kui *tingimus* siis
	tegevused 1
	muidu
	tegevused 2
	lõpp kui
			If *tingimus* Then
			laused 1
			Else
			laused 2
			End If
				if *tingimus*:
				laused 1
				else :
				laused 2

Valik ühest				

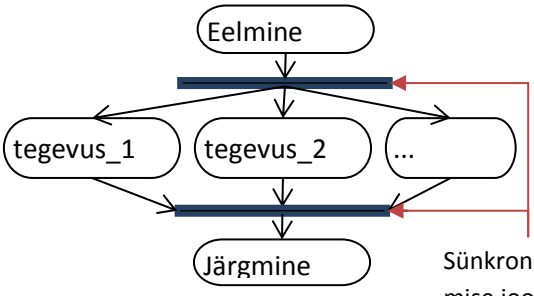
	kui *tingimus* siis
	tegevused
	lõpp kui
	
	kui *tingi* siis *tegevus*
			If *tingimus* Then
			laused
			End If
			
			If *ting* Then *lause*
				if *tingimus*:
				laused
				
				if *ting: lause*

Mitmene valik				

	kui *ting* siis
	kui ting siis
	tegevused 1
	muidu
	tegevused 2
	muidu
	kui *ting* siis *tegev*
	lõpp kui
			If *ting* Then
			laused
			[ElseIf *ting* Then
			laused]
			…
			[Else
			laused]
			End If
				if *ting*:
				laused
				[elif *ting*:
				laused]
				…
				[else:
				laused]

Üldjuhul võib valiku tegemiseks olla suvaline hulk tingimusi. Taolisi asju saab realiseerida valiku põhikorraldusi kasutades. Mitmetes keeltes on mitmese valiku jaoks olemas spetsiaalsed laused.

Paralleelsed protsessid

UML	Algoritmikeel	Scratch	Visual Basic
 Sünkroniseerimise jooned	Kokkuleppeline esitusviis	Ühetü ühilised päise-plokid 	puudub **Python** siin ei vaadelda

Mitmest üksusest koosnevad rakendused

Programmid koosnevad sageli mitmest omavahel seotud üksusest: protseduurist, funktsioonist või skriptist. Vastavalt sellele vastab igale osale omaette algoritm. Programmi üksuste st ka algoritmi üksuste vahel toimub teatud koostöö. Siin võib eristada kahte aspekti:

- pöördumine ühest üksusest teise poole
- andmevahetus üksuste vahel

Võib eristada kahte liiki üksusi:

- funktsioonid
- protseduurid

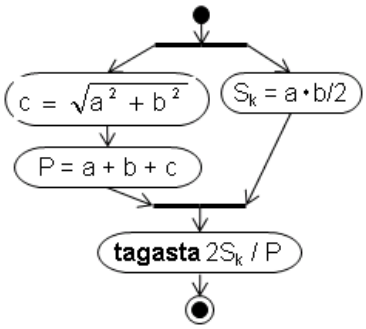
Erinevates programmeerimiskeeltes kasutatakse erinevaid nimetusi ja teatud erinevusi on nende üksuste kasutamisel, kuid üldised põhimõtted on samad.

Funktsioonid. Parameetrid ja tagastatav väärtus

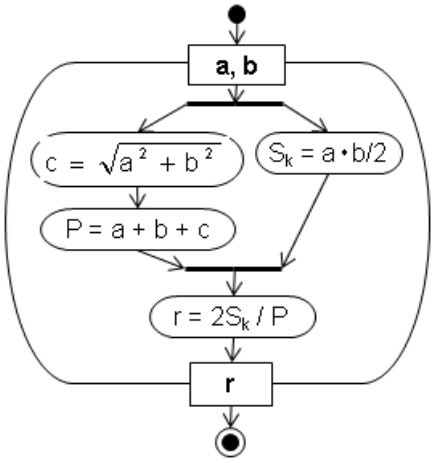
Funktsioonid on mõeldud peamiselt väärtuste leidmiseks (tuletamiseks). Tüüpiline funktsioon leiab ja **tagastab ühe väärtuse**. Tabelis on toodud funktsioon, mis leiab täisnurkse kolmnurga siseringi raadiuse. On toodud UML skeemi kaks varianti, funktsiooni esitus algoritmikeeles, Pythonis, Visual Basicus, Scratchi versioonis 1.4 ja 2.0.

Täisnurkse kolmnurga siseringi raadius: variant 1

parameetrid: a, b – kaatetid
tagastatav väärtus - r



Täisnurkse kolmnurga siseringi raadius: variant 2. Parameetrid ja tagastatav väärtus on skeemil



Algoritmikeel

funktsioon Sirira(a, b)
c = **sqrt**(a2 + b2)
P = a + b + c
Sk = a*b/2
tagasta 2Sk / P

Pythoni funktsioon

```
import math
def Sirira(a, b):
    c=math.sqrt(a*a+b*b)
    P=a+b+c
    Sk=a*b/2
    return 2*Sk/P
```

Kasutamine (pöördumine)
print(**Sirira**(4,5))

VBA funktsioon

```
Function Sirira(a, b)
Dim c, P, Sk
c = Sqr(a ^ 2 + b ^ 2)
P = a + b + c
Sk = a * b / 2
Sirira = 2 * Sk / P
End Function
```

Kasutamine

r = Sirira(k1, k2)

Scratch versioon 1.4

Scratch versioon 2.0 (BYOB)

Funktsioonides saab kasutada sisendandmete esitamiseks **parameetreid**. Siin on parameetriteks kaatetite pikkused **a** ja **b**. Parameetrid kujutavad endast tinglikke muutujaid. Nad saavad väärtused vastavatelt **argumentidelt** pöördumisel. Parameetreid kasutatakse (koos funktsiooni sisemuutujatega) tulemuse leidmisel. Nagu öeldud, leiab ja tagastab funktsioon tavaliselt **ühe väärtuse**, siin on selleks raadius **r**. Tagastatav väärtus määratletakse ühel või teisel moel vastavas protseduuris. Üsna levinud on *return* (**tagasta**) lause kasutamine või tulemuse omistamine funktsiooni nimele (VB). UMLi skeemidel võib parameetrid ja tagastava väärtuse näidata skeemi ümbritseva kujundi servadel. Algoritmikeeles, Pythonis, VBs ja enamikes programmeerimiskeeltes näidatakse parameetrid sulgudes funktsiooni päises nime järel.

Scratchi praeguses versioonis (1.4) funktsioone (ka parameetreid ja argumente) ei ole. Neid saab teatud määral modelleerida tavalise skriptina, mille võiks teha eraldi spraidi jaoks. Parameetritele ja tagastavale väärtusele vastavad suurused on otstarbekas määratleda globaalsete muutujate abil (siin **a**, **b**, **r**), nö abisuurused lokaalsete muutjate abil (siin **c, P, Sr**).

Scratchi järgmises versioonis (2.0) peaks saama kasutada funktsioone, parameetreid ja argumente luues vastavaid plokke. Praegu arendavad seda varianti koos Berkeley ülikool ja MIT BYOB nime all.

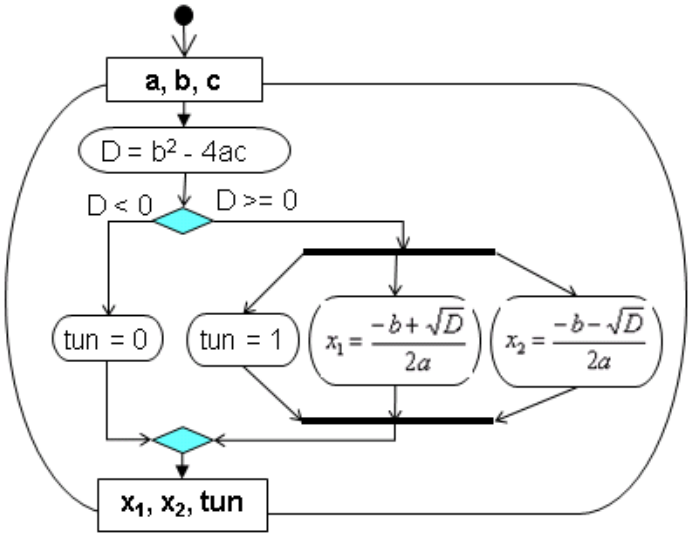
Funktsiooni kasutamine toimub nn **funktsiooniviida** abil, mis esitatakse kujul:

f_nimi(argumendid)

f_nimi - funktsiooni nimi; **argumendid** - parameetritele vastavad väärtused, mis võivad olla esitatud konstantidena, muutujatena või avaldistena. Argumentide arv ja järjestus peab vastama parameetritele.

Protseduurid. Sisend- ja väljundparameetrid

Protseduurid on tunduvalt üldisema iseloomuga kui funktsioonid. Nad võivad teha suvalisi tegevusi, mida saab kasutada antud keeles, sh ka leida väärtusi. Näiteks on toodud parameetritega protseduurina realiseeritud algoritm ruutvõrrandi lahendamiseks.




```
protseduur Ruutvrd (a, b, c, x1, x2, tun)
  D = b² - 4ac
  kui D < 0 siis
    tun = 0
  muidu
    tun = 1
    x1 = (-b+√D)/2a    x2 = (-b-√D)/2a
  lõpp kui
lõpp Ruutvrd
```

protseduur Ruutpea

```
loe a
kui a = 0 siis
  kuva „a ei tohi olla 0!“
  stopp
lõpp kui
loe  b, c
Ruutvrd a, b, c, x1, x2, tunnus
kui tunnus = 0 siis
  kuva “Juured puuduvad!“
muidu
   kuva x1, x2
lõpp kui
```

Sisendparameetriteks on siin ruutvõrrandi kordajad **a, b, c**, väljundparameetriteks on lahendid **x1** ja **x2** (kui on) ja tunnus **tun**, mis näitab, kas lahendid on (**tun = 1**) või ei ole (**tun = 0**).

Ruutpea kujutab endast peaprotseduuri. See loeb algandmed, kontrollides kas **a = 0**. Kui jah siis, kuvatakse vastav teade ja katkestatakse programmi täitmine. Kui **a** ei ole null, loetakse ka **b** ja **c** väärtused ning pöördutakse alamprotseduuri **Ruutvrd** poole.

Scratchi praeguses versioonis ei ole parameetrite ja argumentide kasutamine võimalik. Kuid saab teatud määral modelleerida.

Võib teha spetsiaalse spraidi, millel on kaks skripti. Esimene **RuutVrd** realiseerib ruutvõrrandi lahendamise ülaltoodud algoritmi järgi, kusjuures kasutatakse spraidi lokaalseid muutujaid, mille nimed algavad siin allkriipsuga, et neid oleks lihtsam eristada tavalistest muutujatest. Teine (**RV**) on liidese mall. Sprait nende skriptidega on otstarbekas eksportida. Hiljem saab seda lisada suvalisse projekti ja siduda sellega.

Sisendparameetritele **_a, _b** ja **_c** ja väljundparameetritele **_x1**, **x2**, **_tun** seatakse vastavusse argumendid: **a**, **b** ja **c**, ning **y1**, **y2** ja **tun**. Näiteks on toodud ka skript, mis loeb kordajad ja käivitab skripti **RV**. See seob parameetrid ja argumendid ja käivitab skripti **RuutVrd**. Võite tutvuda ka projektiga.

Näiteks on toodud sama algoritmi realiseerimine Visual Basicuga

```
Sub RuutPea()
  Dim a, b, c, y1, y2, tun
   a = InputBox("Anna a")
   If a = "" Or a = "0" Then
     MsgBox "a ei tohi olla 0!":
    End
   End If
   b = InputBox("Anna b")
   c = InputBox("Anna c")
   RuutVrd a, b, c, y1, y2, tun
   If tun = 0 Then
     MsgBox "Juured puuduvad!"
   Else
     MsgBox "y1="&y1&" y2="& y2
   End If
End Sub
```

```
Sub RuutVrd(a,b,c, x1,x2, tun)
  Dim D
  D = b ^ 2 - 4 * a * c
  If D < 0 Then
     tun = 0
  Else
     tun = 1
      x1 = (-b + Sqr(D)) / (2 * a)
      x2 = (-b - Sqr(D)) / (2 * a)
   End If
End Sub
```

Klassikaline parameetritega protseduur.
a, b, c on sisendparameetrid; **x1, x2, tun** – väljundparameetrid.
Sisendparameetrid saavad väärtused samanimelistelt argumentidelt. Kasutades neid, leitakse tulemused ja omistatakse need parameetritele **tun, x1, x2**.

Parameetrite väärtused omistatakse vastavalt argumentidele: **y1, y2, tun**. Peaprotseduuris on võetud muutujate nimedeks **y1** ja **y2** näitamaks, et parameetrite ja argumentide nimed ei pea olema samad.

Pöördumine ühest üksusest teise poole

Allpool on toodud tüüpilised elemendid koostöö kirjeldamiseks programmiüksuste vahel

UML	Algoritmikeel	Scratch	Visual Basic	Python
Pöördumine protseduuri poole				
kutsu p_nimi()	**kutsu** p_nimi(arg,)	 (1.4)  ver (2.0)	**Call p_nimi** (arg, …) **p_nimi arg, …**	**f_nimi** (…)
Pöördumine funktsiooni poole				
muut = f_nimi(…)	muut = f_nimi(…)	Scratchi vers. 1.4 ei ole  (2.0)	**muut = f_nimi(…)**	
Teadete saatmine				
t_nimi **teavita** t_nimi	**teavita** t_nimi	 	puudub	
Teadete vastuvõtmine				
t_ni	**teade** t_nimi		puudub	