# Kuidas mõista programmi: programmeerimiskeeled, korrektsus ja abstraktne interpretatsioon

Karoliine Holter

Tartu Ülikool, tarkvarateaduse töörühm

8. veebruar 2026

# Minust

2004 Mart Reiniku Kool
2013 Hugo Treffneri Gümnaasium
2016 „Paus“ (Olümpia)
2018 Informaatika BSc (cum laude)
2021 Tarkvaratehnika MSc (cum laude)
2023 Informaatika nooremteadur


http://www.awshop.xyz/image/cache/catalog/opidtootad-500x500-0.jpg

# Tarkvarateadus

Programmeerimiskeelte uurimine (ja loomine)

> Programmeerimiskeeli nähakse keskse rolliga arvutuslike probleemide lahendamisel ning neid kasutatakse vahendina inimkeele „tõlkimiseks“ masinkoodiks.

**Väljendusrikkus** arendatakse uusi keelelisi võimalusi, et lahendada aina keerukamaid arvutuslikke probleeme üldiselt ja selgelt.

**Suhtluse korrektsus** analüüsitakse, kas ja millal programmid (ka olemasolevad ja juba kasutusel olevad) käituvad korrektselt ning vastavad sellele, mida nendelt oodatakse.

# Miks on programmide korrektsus niivõrd oluline?

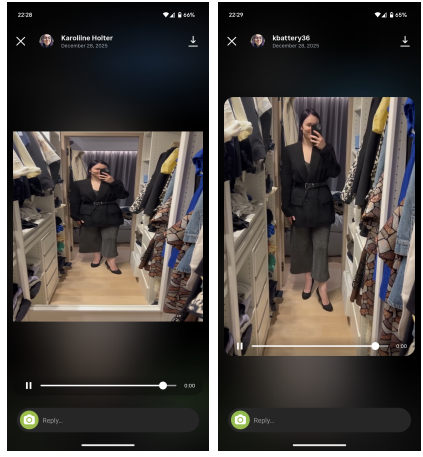
- Tüüpilised näited on raketid, lennukid või tuumaelektrijaamad.
- Therac-25 kiiritusravimasin: andmejooksu tõttu inimestele üledoos
- Aga väiksemad asjad meie elus ...
- Kui palju on „kehv programmeerimine“ Sinule haiget teinud?

# Miks on programmide korrektsus niivõrd oluline?

- Tüüpilised näited on raketid, lennukid või tuumaelektrijaamad.
- Therac-25 kiiritusravimasin: andmejooksu tõttu inimestele üledoos
- Aga väiksemad asjad meie elus . . .
- Kui palju on „kehv programmeerimine“ Sinule haiget teinud?

# Miks on programmide korrektsus niivõrd oluline?

- Tüüpilised näited on raketid, lennukid või tuumaelektrijaamad.
- Therac-25 kiiritusravimasin: andmejooksu tõttu inimestele üledoos
- Aga väiksemad asjad meie elus . . .
- Kui palju on „kehv programmeerimine“ Sinule haiget teinud?



# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon

**Keele spetsifikatsioon**

- süntaks - mis sõnad on lubatud?

```
if 5 > 2:
  print("Viis on suurem kui kaks")
```

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon

**Keele spetsifikatsioon**

- süntaks - mis sõnad on lubatud?

```
if 5 > 2:
  print("Viis on suurem kui kaks")
```

- semantika - mida programm *tegelikult tähendab*?

```
x = x + 1
```

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon

**Keele spetsifikatsioon**

- süntaks - mis sõnad on lubatud?

```
if 5 > 2:
  print("Viis on suurem kui kaks")
```

- semantika - mida programm *tegelikult tähendab*?

```
x = x + 1
```

*„Programm ei ole lihtsalt sümbolite jada, mida niisama ~~stackoverflow'st~~ keelemudelist kopeerida.“*

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika

**Formaalne semantika**

Leiame arvutuste taga peituvaid matemaatilisi struktuure

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika

**Formaalne semantika**

Leiame arvutuste taga peituvaid matemaatilisi struktuure

- kuidas programm suhtleb keskkonnaga?

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika

**Formaalne semantika**

Leiame arvutuste taga peituvaid matemaatilisi struktuure

- kuidas programm suhtleb keskkonnaga?
- programmide teisendamine

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika

**Formaalne semantika**

Leiame arvutuste taga peituvaid matemaatilisi struktuure

- kuidas programm suhtleb keskkonnaga?
- programmide teisendamine
- kas arvutuste järjekord loeb?
- jpm.

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika
3. Staatiline arutluskäik

**Staatiline arutluskäik**

Arutleme programmi üle *ilma seda käivitamata*

- tüübid

```
x = ''abc'' + 1
```

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika
3. Staatiline arutluskäik

**Staatiline arutluskäik**

Arutleme programmi üle *ilma seda käivitamata*

- tüübid
  ```
  x = ''abc'' + 1
  ```
- staatiline analüüs
  ```
  x = [1,2,3]
  x[4]
  ```

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika
3. Staatiline arutluskäik
4. Dünaamiline arutluskäik

**Dünaamiline arutluskäik**

Uurime programmi käitumist *käivitamise ajal*

- testid
  **assert** summa(2,3) == 5

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika
3. Staatiline arutluskäik
4. Dünaamiline arutluskäik

**Dünaamiline arutluskäik**

Uurime programmi käitumist *käivitamise ajal*

- testid
  **assert** summa(2,3) == 5
- veateated

```
>>> x = [1,2,3]
>>> x[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

# Programmeerimiskeelte põhilised uurimisvaldkonnad

1. Keele spetsifikatsioon
2. Formaalne semantika
3. Staatiline arutluskäik
4. Dünaamiline arutluskäik
5. Implementatsioon

**Implementatsioon**

Kuidas programm masinas tööle pannakse?

- interpreteerimine
- kompileerimine

# Interpretatsioon

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

# Interpretatsioon, mis juhtub sisendiga 10?

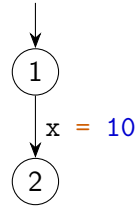
```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

# Interpretatsioon, mis juhtub sisendiga 10?


```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

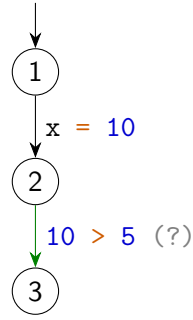
# Interpretatsioon, mis juhtub sisendiga 10?

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



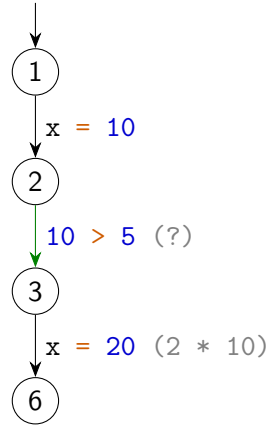
# Interpretatsioon, mis juhtub sisendiga 10?

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



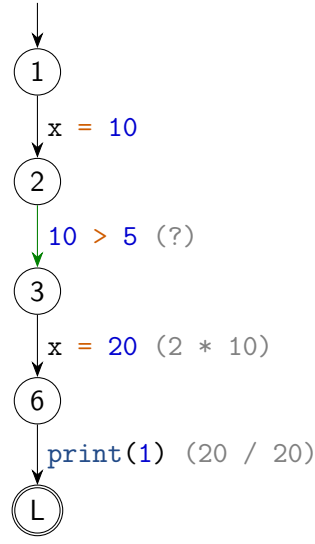
# Interpretatsioon, mis juhtub sisendiga 10?

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



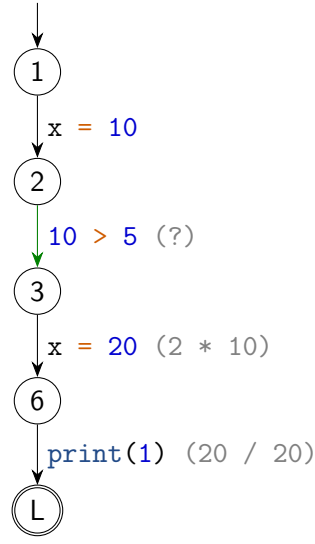
# Interpretatsioon, mis juhtub sisendiga 10?

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



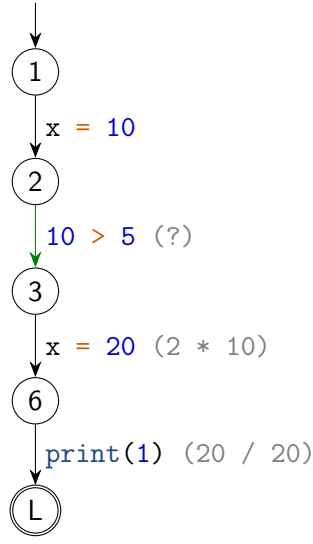
# Interpretatsioon, mis juhtub sisendiga -5?

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



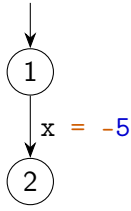
# Interpretatsioon, mis juhtub sisendiga -5?

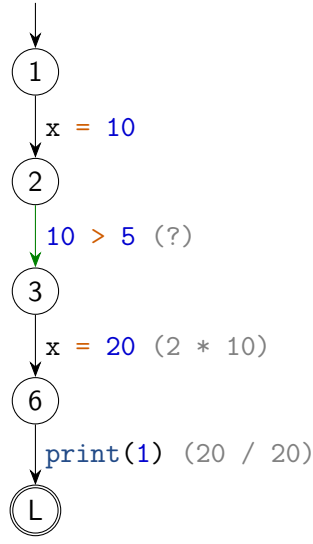

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



# Interpretatsioon, mis juhtub sisendiga -5?

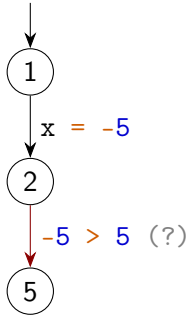
```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

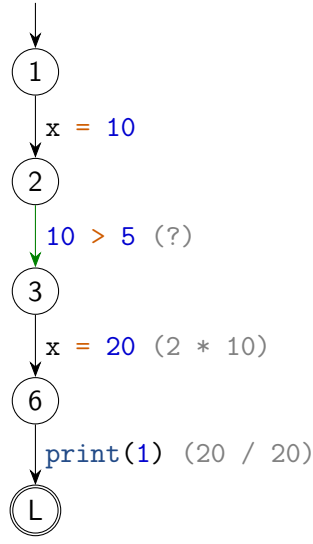


# Interpretatsioon, mis juhtub sisendiga -5?

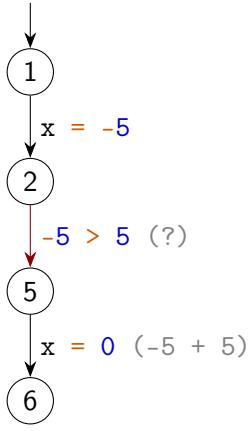
```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

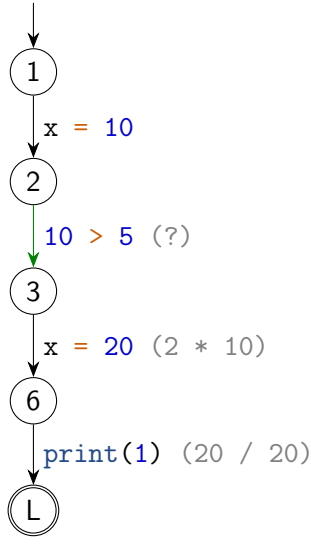


# Interpretatsioon, mis juhtub sisendiga -5?

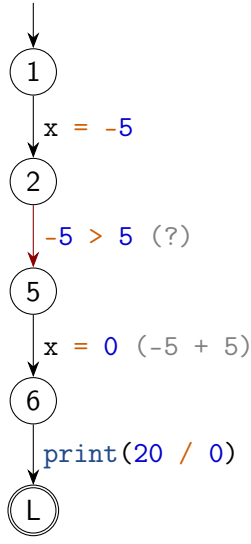
```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

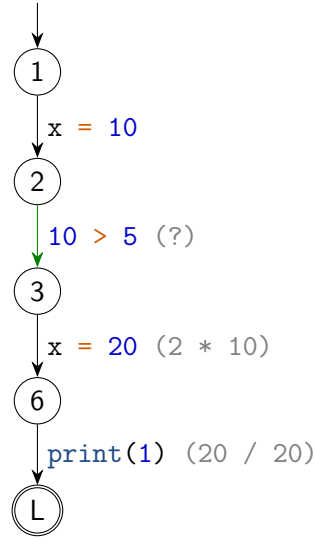


# Interpretatsioon, mis juhtub sisendiga -5?

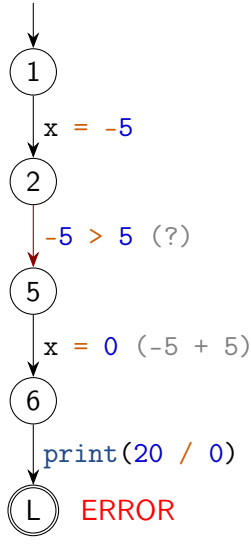
```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

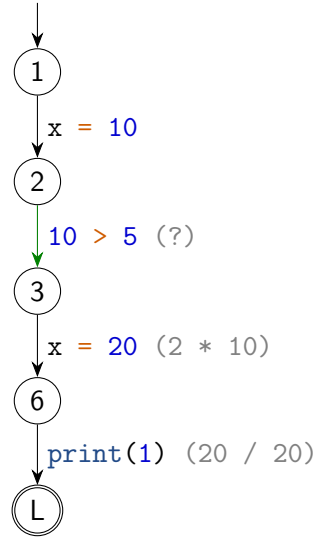


# Interpretatsioon, mis juhtub sisendiga -5?

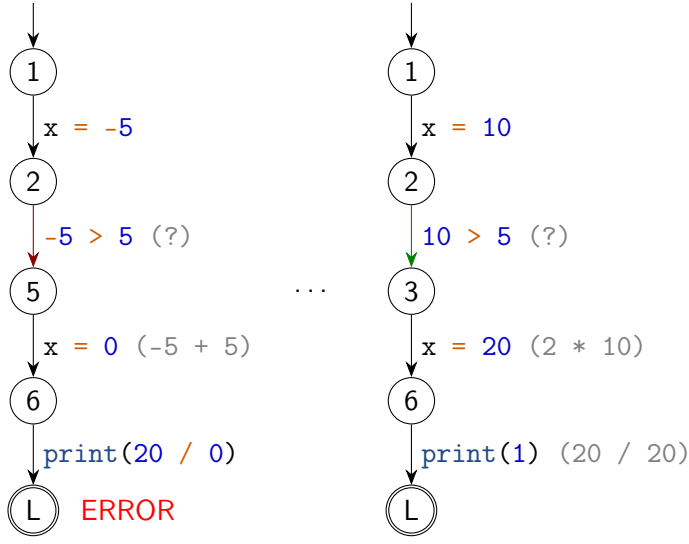
```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



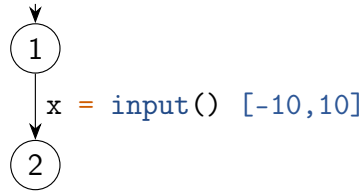



# Interpretatsioon, mis juhtub teiste sisenditega?

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



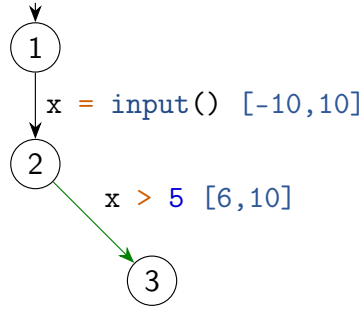
# Abstraktne interpretatsioon, sisend [-10, 10]




```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```

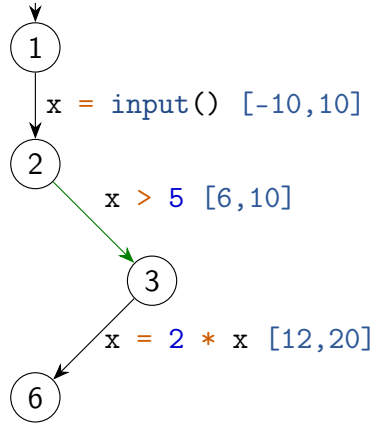
# Abstraktne interpretatsioon, sisend [-10, 10]

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



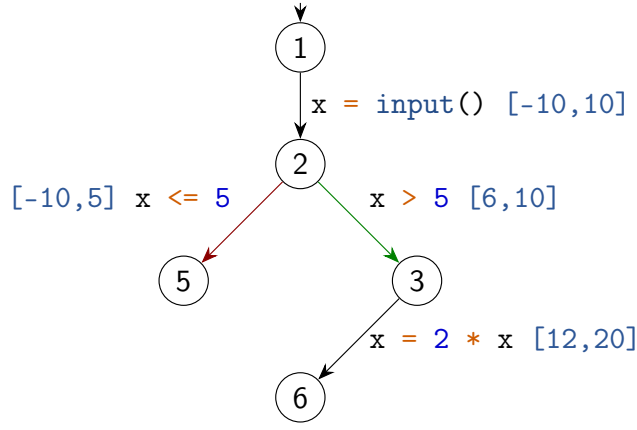
# Abstraktne interpretatsioon, sisend [-10, 10]

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



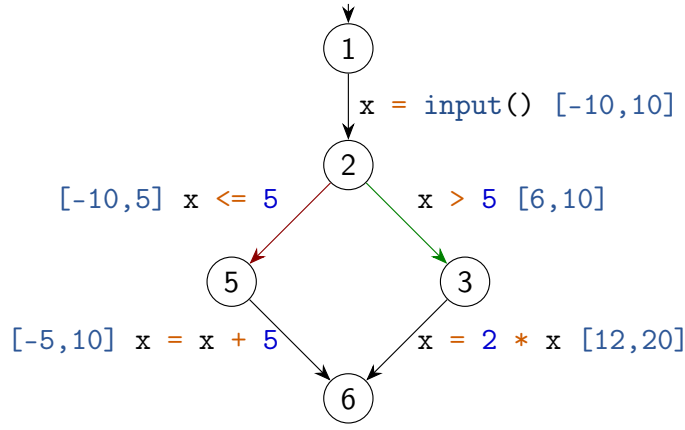
# Abstraktne interpretatsioon, sisend [-10, 10]

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



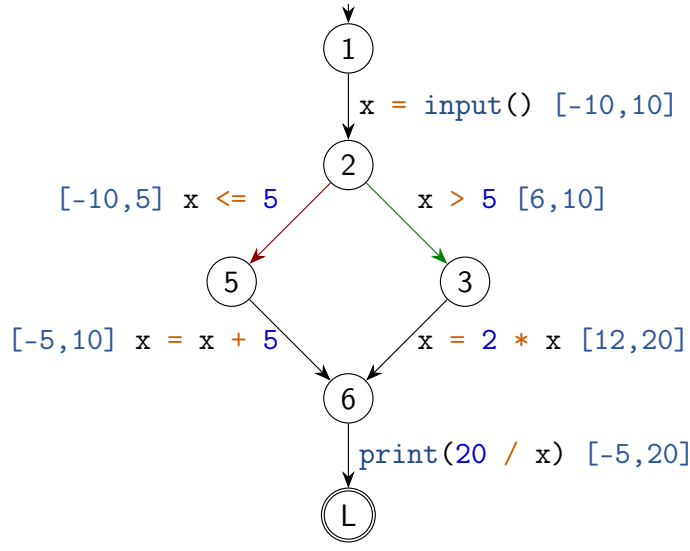
# Abstraktne interpretatsioon, sisend [-10, 10]

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



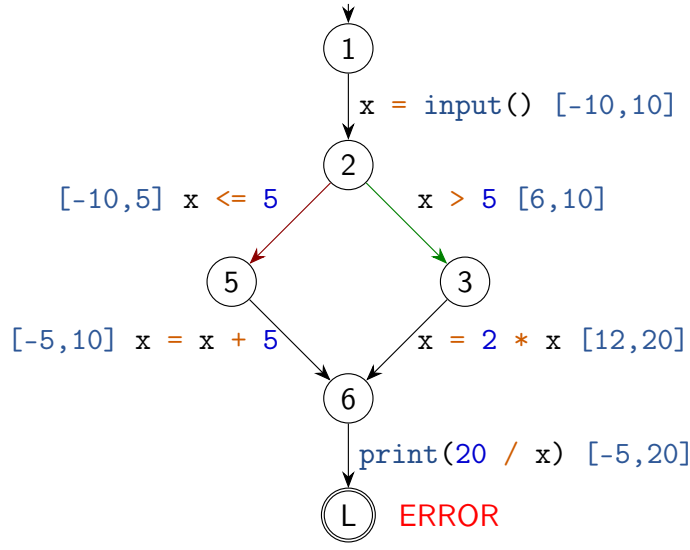
# Abstraktne interpretatsioon, sisend [-10, 10]

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



# Abstraktne interpretatsioon, sisend [-10, 10]

```
1 def fun(x):
2     if x > 5:
3         x = 2 * x
4     else:
5         x = x + 5
6     print(20 / x)
```



# Programmide *semantiline* analüüs

Mudelkontroll

$$\{\{x \mapsto 4\}, \{x \mapsto 5\}, \{x \mapsto 6\}, \dots\} \xrightarrow{x=x+1}$$

# Programmide *semantiline* analüüs

Mudelkontroll

$$\{\{x \mapsto 4\}, \{x \mapsto 5\}, \{x \mapsto 6\}, \dots\} \xrightarrow{x=x+1} \{\{x \mapsto 5\}, \{x \mapsto 6\}, \{x \mapsto 7\}, \dots\}$$

# Programmide *semantiline* analüüs

Mudelkontroll

$$\{\{x \mapsto 4\}, \{x \mapsto 5\}, \{x \mapsto 6\}, \dots\} \xrightarrow{x=x+1} \{\{x \mapsto 5\}, \{x \mapsto 6\}, \{x \mapsto 7\}, \dots\}$$

Abstraktne interpretatsioon

$$\{x \mapsto [4, \infty]\} \xrightarrow{x=x+1}$$

# Programmide *semantiline* analüüs

Mudelkontroll

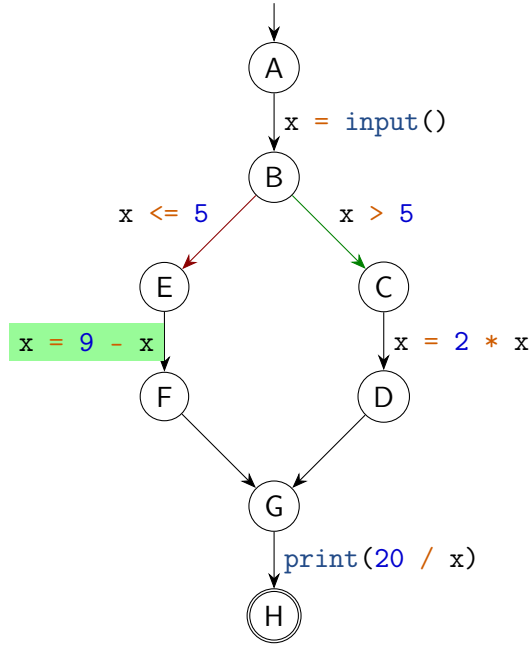
$$\{\{x \mapsto 4\}, \{x \mapsto 5\}, \{x \mapsto 6\}, \dots\} \xrightarrow{x=x+1} \{\{x \mapsto 5\}, \{x \mapsto 6\}, \{x \mapsto 7\}, \dots\}$$

Abstraktne interpretatsioon

$$\{x \mapsto [4, \infty]\} \xrightarrow{x=x+1} \{x \mapsto [5, \infty]\}$$

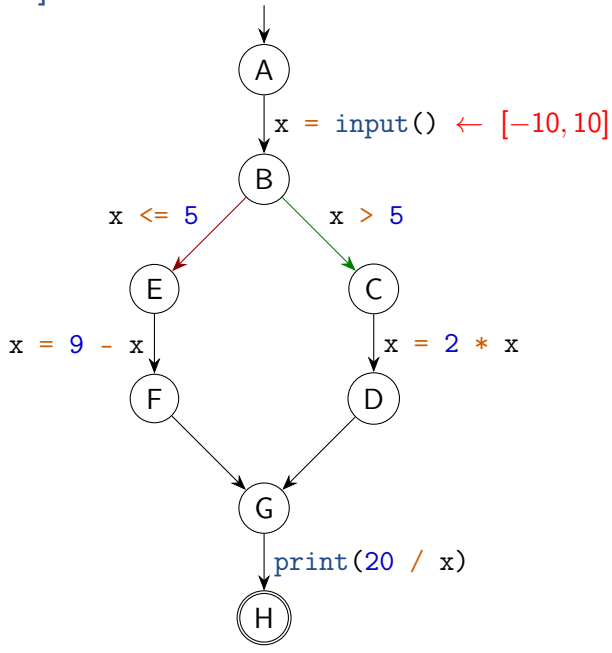
# Teine programm

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



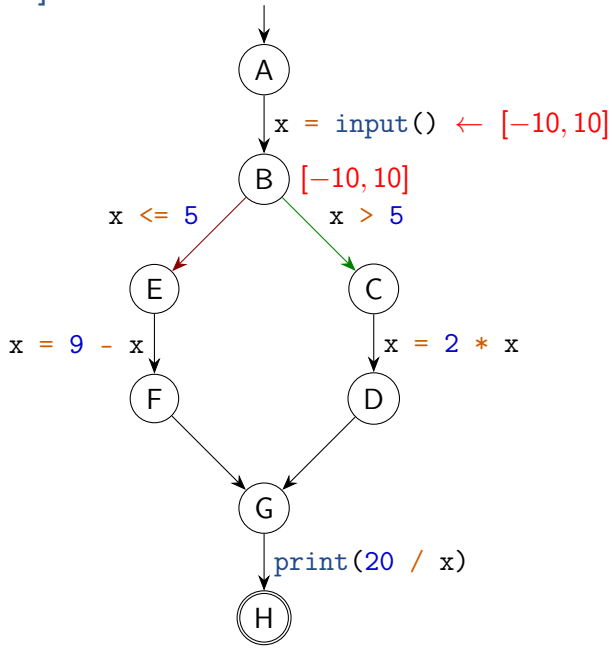
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



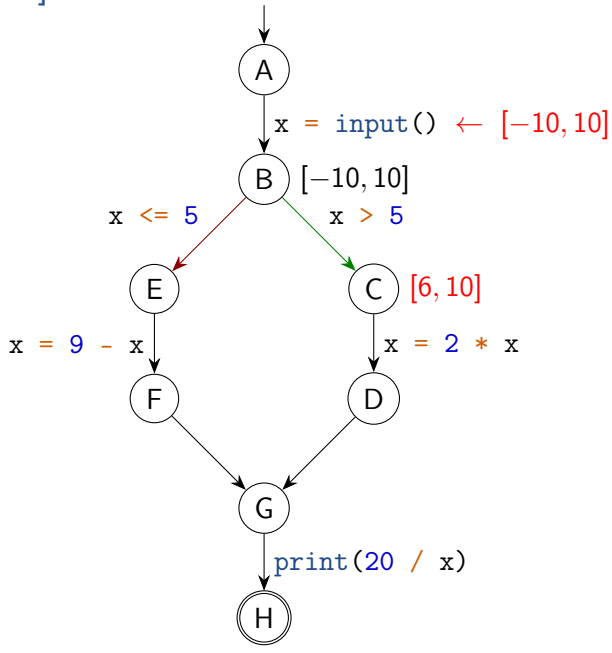
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



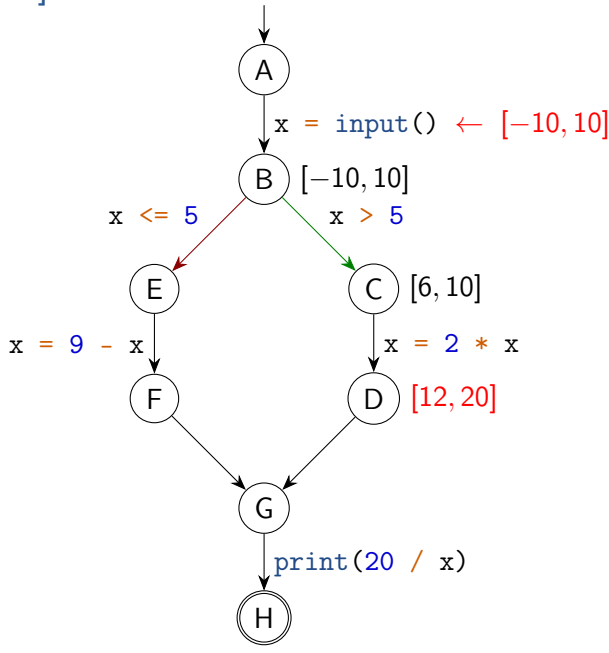
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



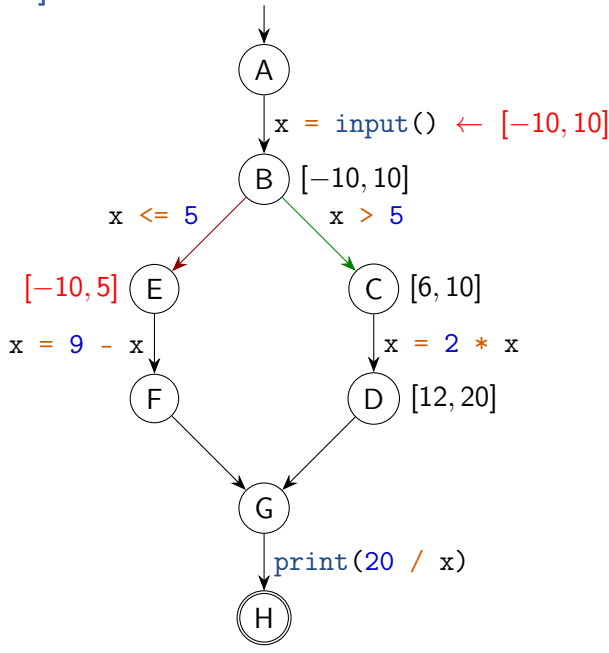
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



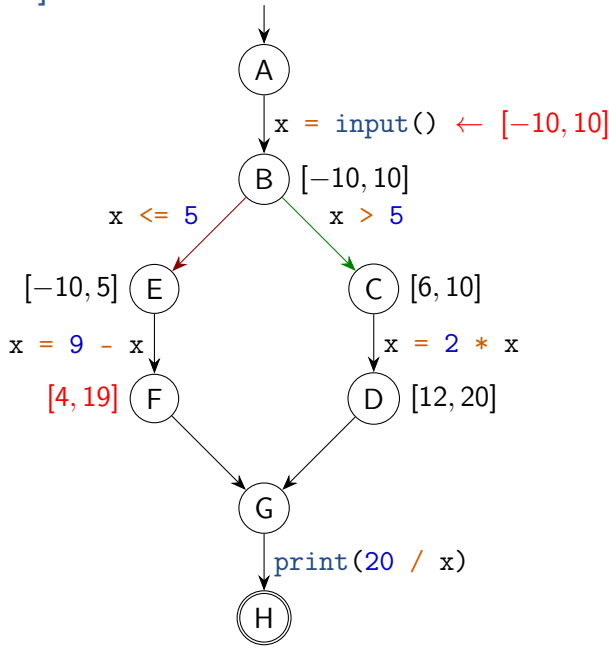
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



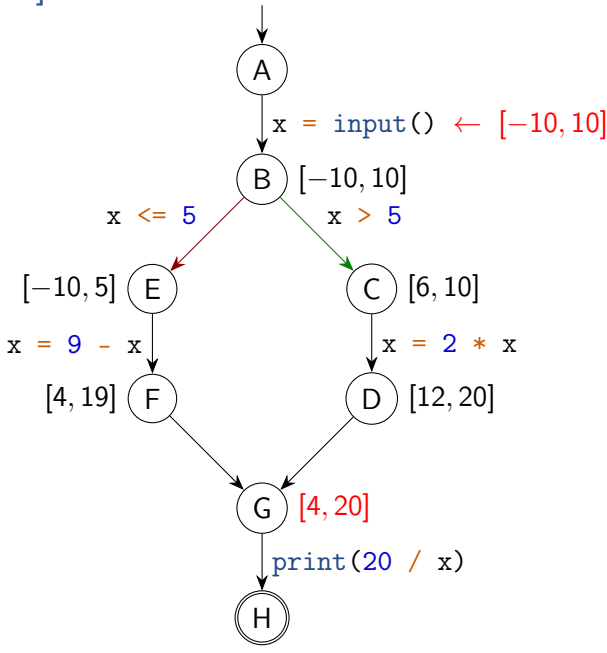
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



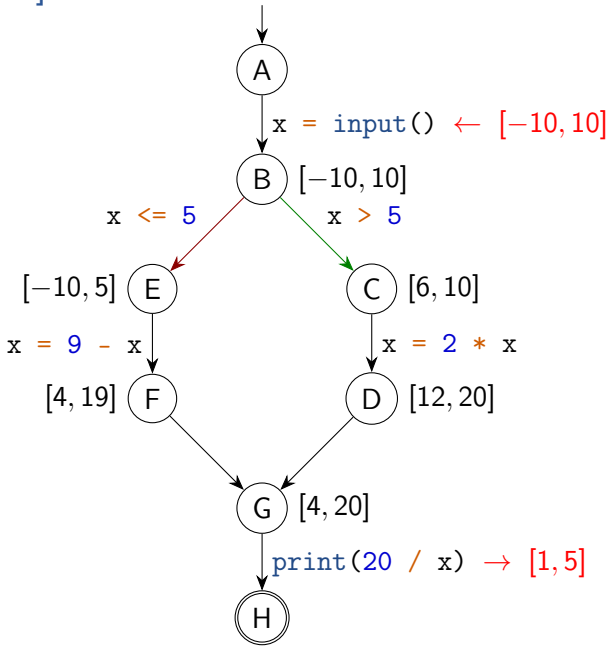
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



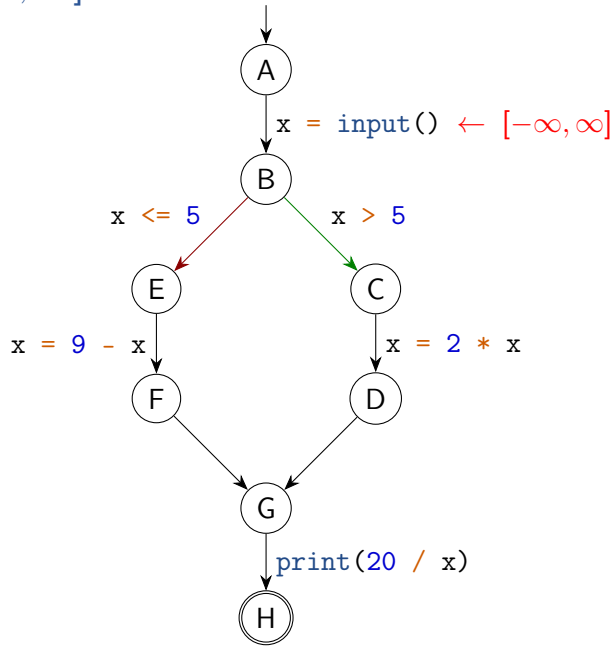
# Teine programm sisendiga $[-10, 10]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



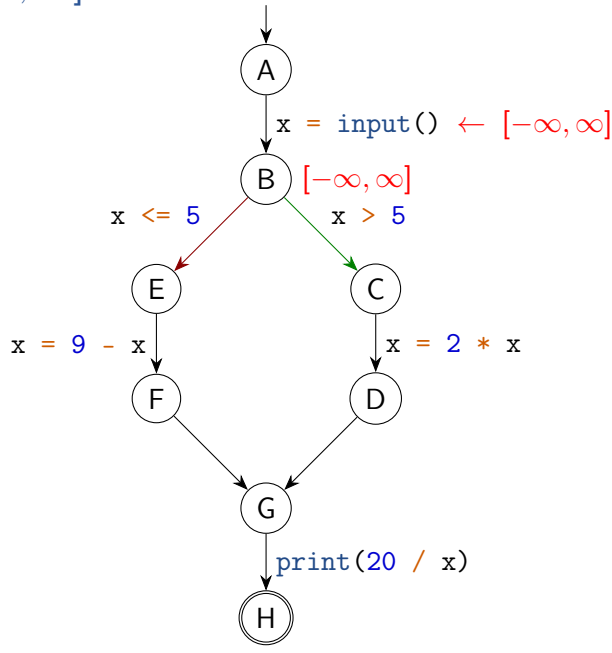
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



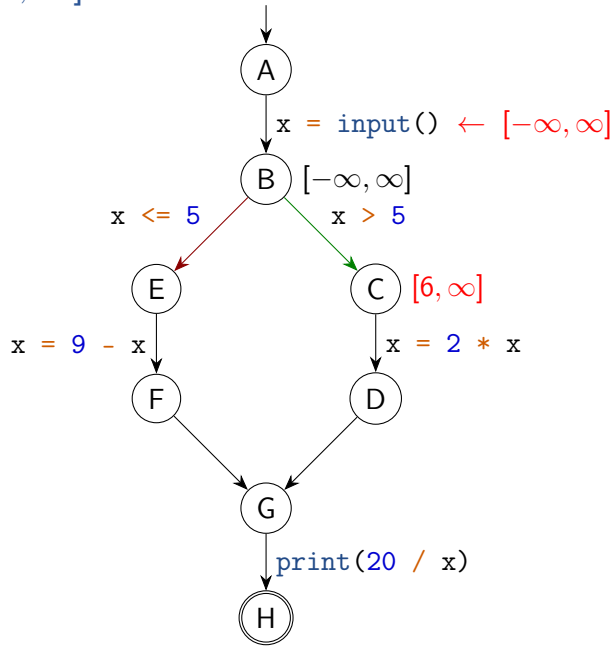
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



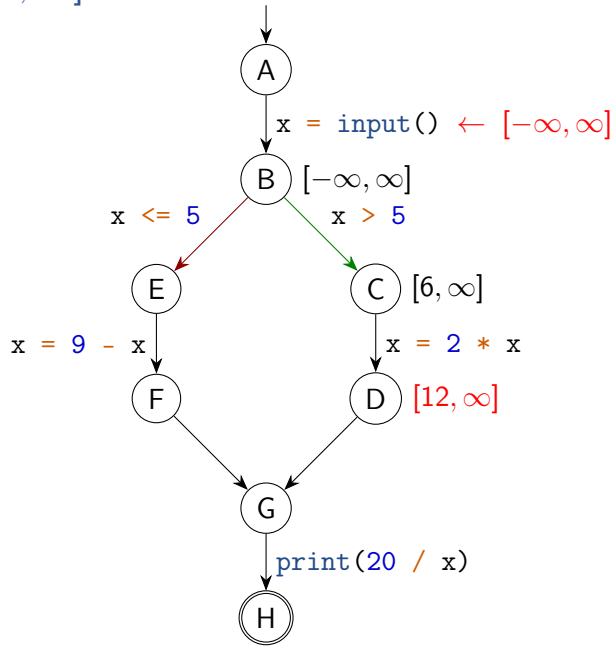
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



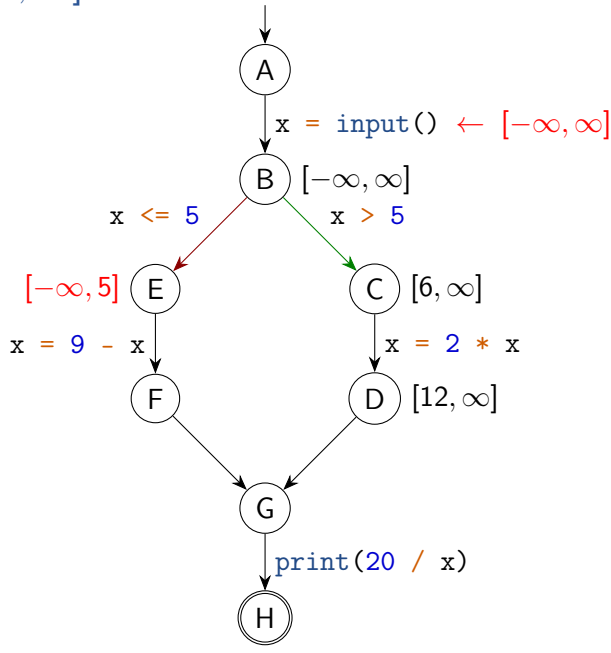
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



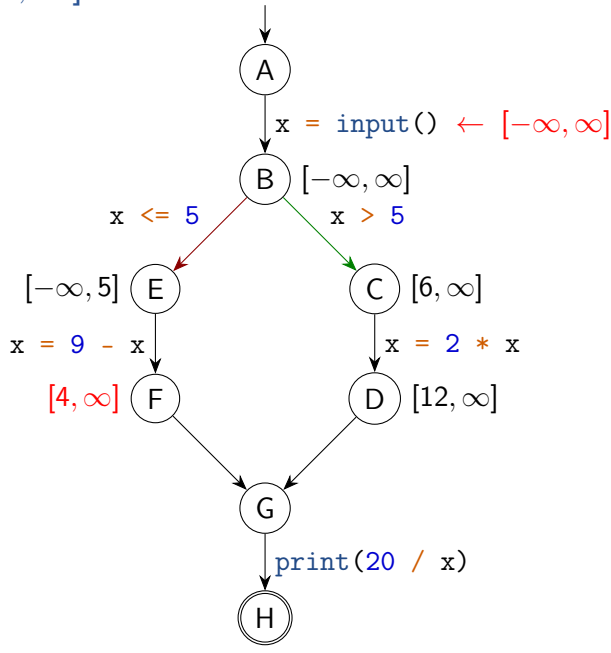
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



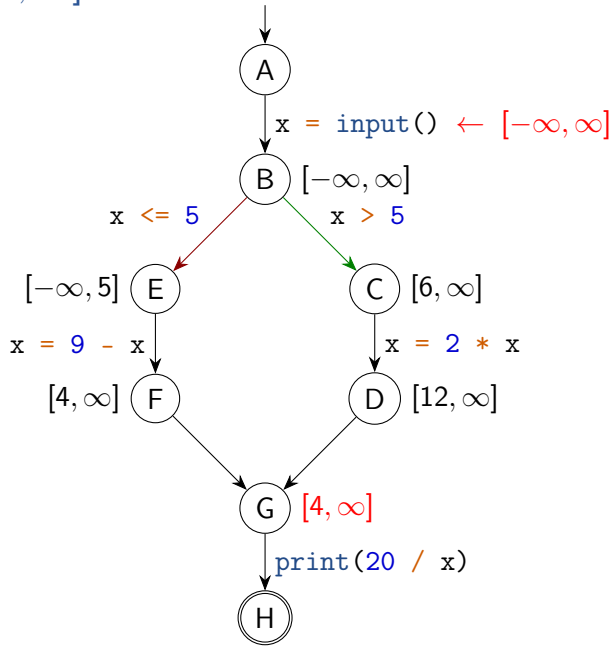
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



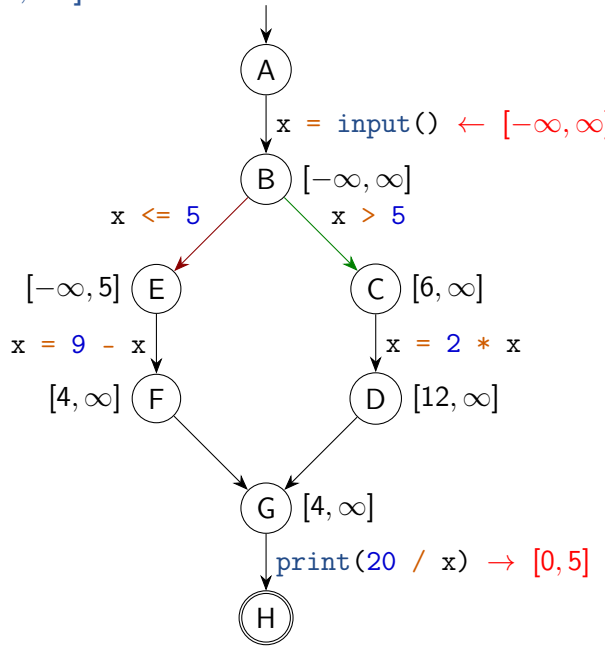
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



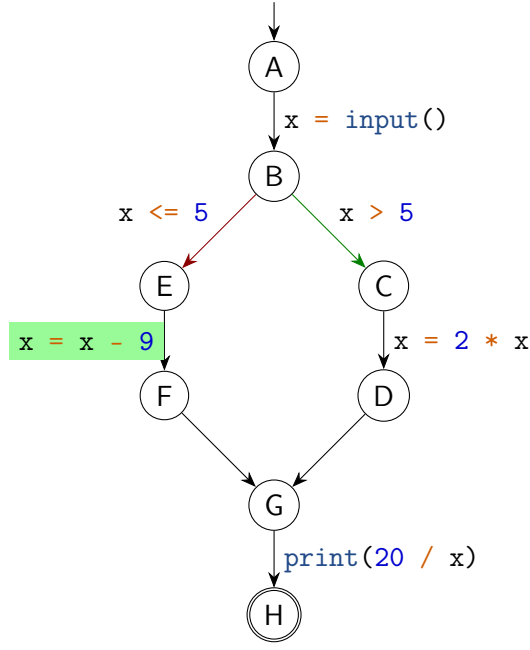
# Teine programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = 9 - x
6 print(20 / x)
```



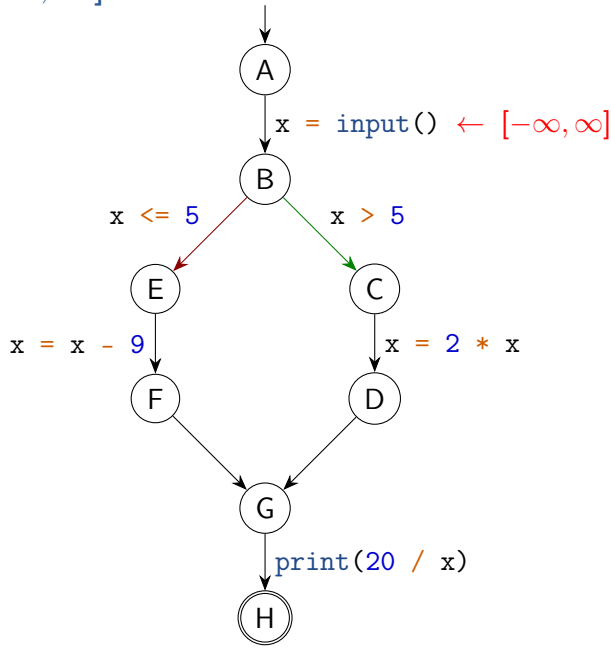
# Kolmas programm

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



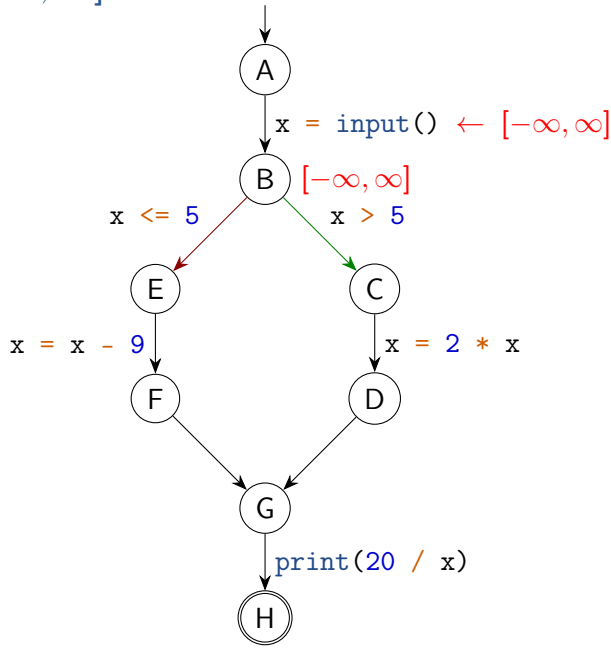
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



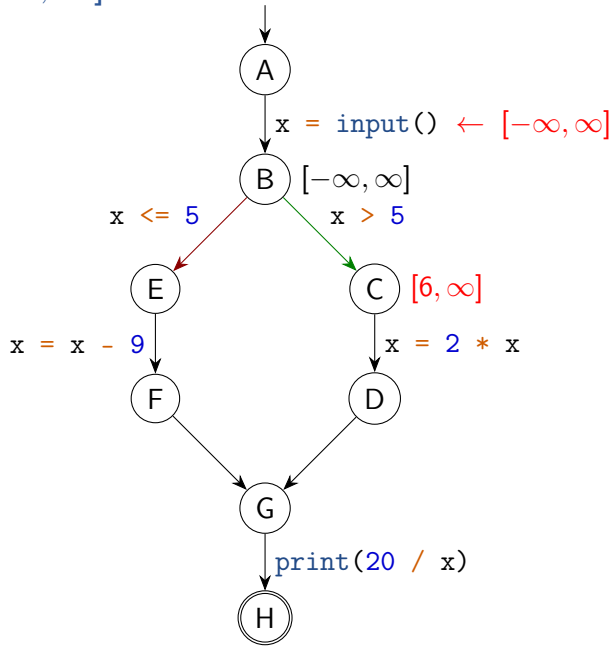
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



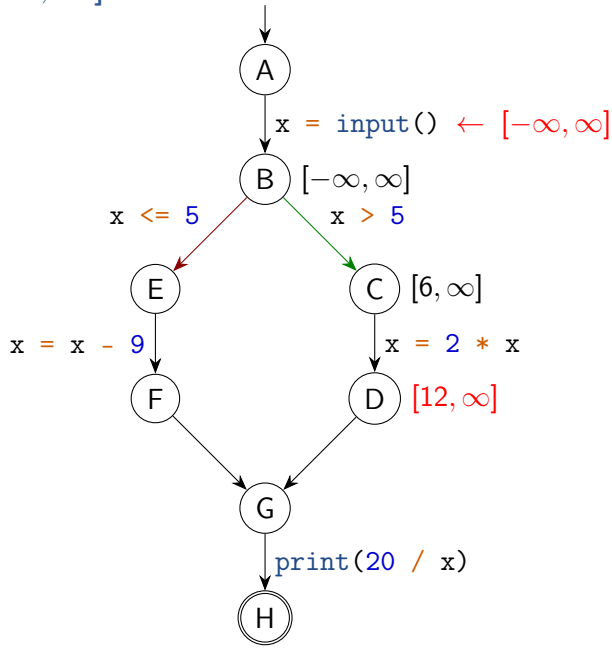
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



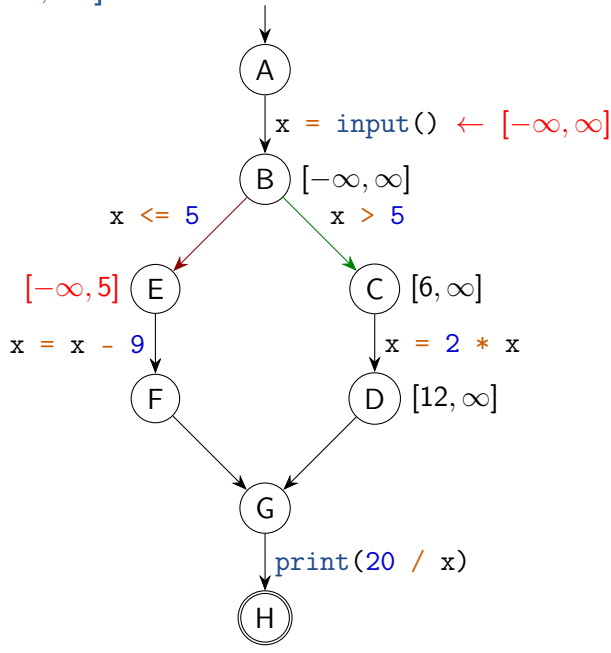
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



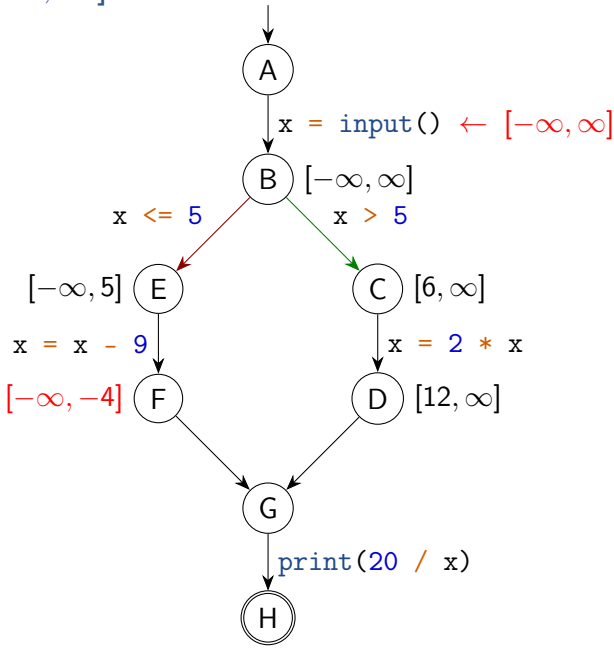
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



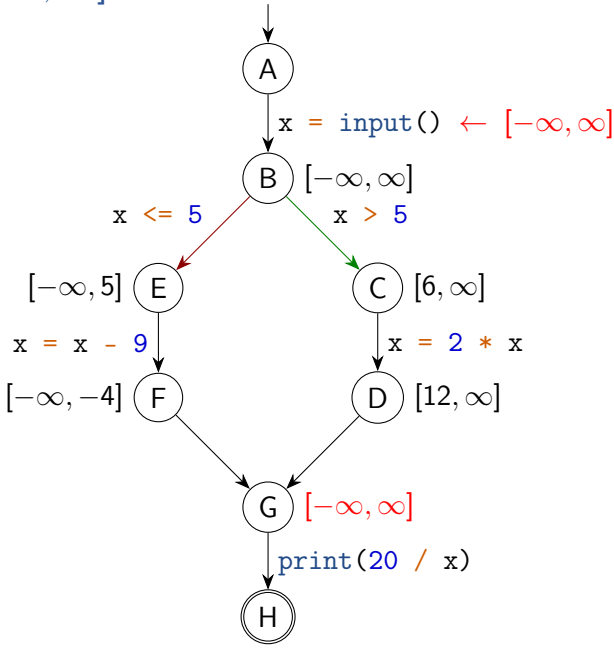
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



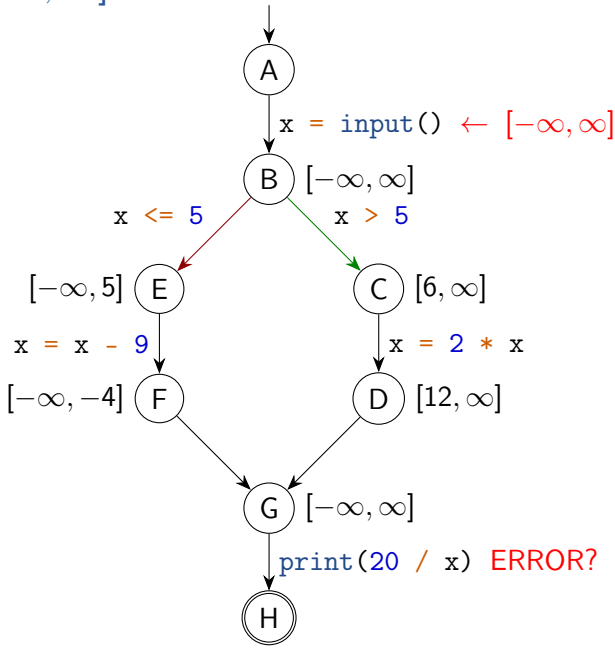
# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



# Kolmas programm sisendiga $[-\infty, \infty]$

```
1 x = input()
2 if x > 5:
3     x = 2 * x
4 else:
5     x = x - 9
6 print(20 / x)
```



# Abstraktne interpretatsioon

**Korrektne**

*Kui päriselt võib tekkida viga, siis abstraktselt tekib viga.*

ehk

*Kui abstraktselt ei teki viga, siis päriselt ei saa viga tekkida.*

Tõestab programmi korrektsuse!

# Abstraktne interpretatsioon

**Korrektne**

*Kui päriselt võib tekkida viga, siis abstraktselt tekib viga.*

ehk

*Kui abstraktselt ei teki viga, siis päriselt ei saa viga tekkida.*

Tõestab programmi korrektsuse!

**Ebatäpne**

*Kui abstraktselt tekib viga, siis päriselt ei pruugi viga tekkida.*

Eksib, aga ainult ühte pidi!

# Programmi omadused, sh. andmejooksud

Eesmärk: näidata, et programmi *kõik* täitmised omavad mingit kindlat omadust, näiteks

- Ei esine nulliga jagamist
- Lõpetab alati töö (termineerub)
- Ei saa esineda andmejookse
- jpm.

# Programmi omadused, sh. andmejooksud

Eesmärk: näidata, et programmi *kõik* täitmised omavad mingit kindlat omadust, näiteks

- Ei esine nulliga jagamist
- Lõpetab alati töö (termineerub)
- Ei saa esineda andmejookse
- jpm.

Andmejooks

- On ohtlik olukord paralleelselt töötavates programmides
- Kui kaks arvutust võivad omavahel kooskõlastamatult andmeid näppida, siis ...
- võib väga halvasti lõppeda!

# Goblint

- Paralleelsete C-programmide analüsaator

# Goblint

- Paralleelsete C-programmide analüsaator
- Abstraktsel interpreteerimisel põhinev

# Goblint

- Paralleelsete C-programmide analüsaator
- Abstraktsel interpreteerimisel põhinev
- Eriline fookus andmejooksudel

# Goblint

- Paralleelsete C-programmide analüsaator
- Abstraktsel interpreteerimisel põhinev
- Eriline fookus andmejooksudel
- Tartu Ülikooli ja Müncheni Tehnikaülikooli koostöö

# Goblint

- Paralleelsete C-programmide analüsaator
- Abstraktsel interpreteerimisel põhinev
- Eriline fookus andmejooksudel
- Tartu Ülikooli ja Müncheni Tehnikaülikooli koostöö
- Mitmekordne andmejooksude kategooria võitja rahvusvahelisel verifitseerimisvõistlusel

# Goblint

- Paralleelsete C-programmide analüsaator
- Abstraktsel interpreteerimisel põhinev
- Eriline fookus andmejooksudel
- Tartu Ülikooli ja Müncheni Tehnikaülikooli koostöö
- Mitmekordne andmejooksude kategooria võitja rahvusvahelisel verifitseerimisvõistlusel
- Andmejooksude maailmameister (Wettlaufweltmeister)

# *Competition on Software Verification (SV-COMP)*

Iga-aastane rahvusvaheline võistlus, kus võrreldakse ja hinnatakse tarkvara verifitseerimise tööriistu.

# *Competition on Software Verification (SV-COMP)*

Iga-aastane rahvusvaheline võistlus, kus võrreldakse ja hinnatakse tarkvara verifitseerimise tööriistu.

- Osalejad esitavad oma tööriistad, mis kontrollivad programmide korrektsust erinevatel meetoditel

# *Competition on Software Verification (SV-COMP)*

Iga-aastane rahvusvaheline võistlus, kus võrreldakse ja hinnatakse tarkvara verifitseerimise tööriistu.

- Osalejad esitavad oma tööriistad, mis kontrollivad programmide korrektsust erinevatel meetoditel
- Tööriistu jooksutatakse suurel hulgal erinevatel programmidel

# *Competition on Software Verification (SV-COMP)*

Iga-aastane rahvusvaheline võistlus, kus võrreldakse ja hinnatakse tarkvara verifitseerimise tööriistu.

Võistluskategooriad:

- *ReachSafety*
- *MemSafety*
- *Concurrency*
- *NoOverflows*
- *Termination*
- *SoftwareSystems*

# *Competition on Software Verification (SV-COMP)*

Iga-aastane rahvusvaheline võistlus, kus võrreldakse ja hinnatakse tarkvara verifitseerimise tööriistu.

Võistluskategooriad:

- *ReachSafety*
- *MemSafety*
- *Concurrency*
- *NoOverflows*
- *Termination*
- *SoftwareSystems*

Punktisüsteem:

| | vigane | korrektne |
|---|---|---|
| õige vastus | +1 | +2 |
| vale vastus | -16 | -32 |

# Verifitseerimisvõistlusest reaalsuseni

*Competition on Software Verification (SV-COMP)*, Dirk Beyer: SV-COMP = F1



Matti Blume: Mercedes-Benz, Techno-Classica 2018, Essen

# Abstraktse interpretatsiooni rakendusi teistes valdkondades


## Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law

Raphaël Monat[1⋆(✉)], Aymeric Fromherz[2⋆], and Denis Merigoux[2]

[1] Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[2] Inria Paris, Paris, France
{raphael.monat,aymeric.fromherz,denis.merigoux}@inria.fr

**Abstract.** Legal expert systems routinely rely on date computations to determine the eligibility of a citizen to social benefits or whether an application has been filed on time. Unfortunately, date arithmetic exhibits many corner cases, which are handled differently from one library to the other, making faithfully transcribing the law into code error-prone, and possibly leading to heavy financial and legal consequences for users.
In this work, we aim to provide a solid foundation for date arithmetic working on days, months and years. We first present a novel, formal semantics for date computations, and formally establish several semantic properties through a mechanization in the F⋆ proof assistant. Building upon this semantics, we then propose a static analysis by abstract interpretation to automatically detect ambiguities in date computations. We finally integrate our approach in the Catala language, a recent domain-specific language for formalizing computational law, and use it to analyze the Catala implementation of the French housing benefits, leading to the discovery of several date-related ambiguities.

# Mida endaga kaasa võtta

- Süntaktiliselt korrektne programm võib siiski oodatust valesti käituda
- Testid ei anna 100% garantiid
- Semantika aitab mõista programmi tähendust
- **Arusaamine on oluline**
- Mõningaid programmi omadusi saab *tõestada*, mitte ainult testida

# Ootame teid väga Tartu Ülikooli

Loodus-ja täppisteaduste valdkonna stipendiumid üle-eestiliste aineolümpiaadide parimatele 300 eurot kuus

- 11.-12. klassi arvestuses viie parima hulgas kolme aasta jooksul
- rahvusvahelistel olümpiaadidel osalemine
- aineolümpiaadid: astronoomia lahtine võistlus, bioloogia, füüsika, keemia, geograafia, maateadused, informaatika, matemaatika


Aitäh!