Programmeren in Python 1

http://www.win.tue.nl/~wstomv/edu/python/

Inleiding

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Constructie Groep

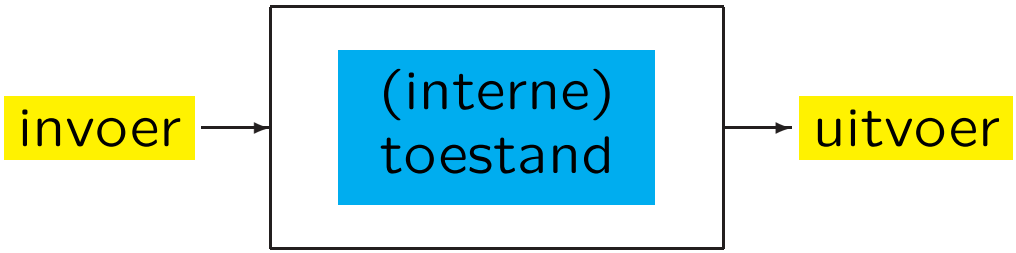
Opmerkingen aan T.Verhoeff@TUE.NL

Overzicht

Waar het om gaat:

- Algemene principes van professioneel programmeren
- Basiskennis van de programmeertaal Python
- Zelfstandig kleine Python programma's lezen, aanpassen, schrijven

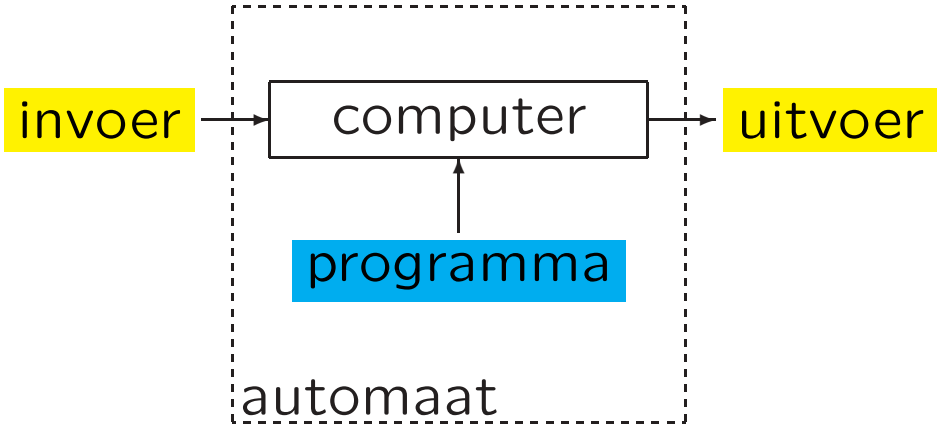
Abstracte automaat



"Berekenings" stappen:

- Invoer lezen
- Toestand veranderen
- Uitvoer schrijven

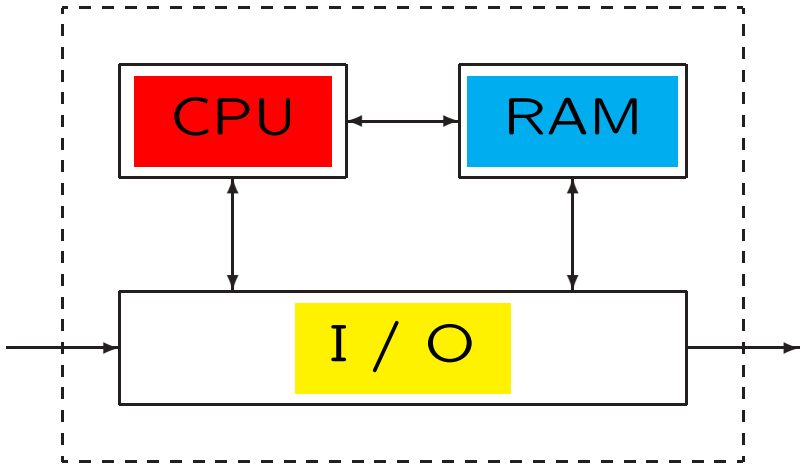
Computer als universele automaat



Automaat = Programma

Automatiseren = Programmeren

Structuur van een computer



CPU = Central Processing Unit

RAM = Random Access Memory (Programma + Gegevens)

I / O = Input and Output

Levensloop van een programma

1. Probleemstelling, toepassing
2. Specificatie, eisenpakket
3. Ontwerp, oplossing
4. Productie, coderen, intikken
5. Controle, testen
6. Gebruik en onderhoud

Voorbeeld: Probleemstelling

Hoe een bedrag gepast betalen met zo min mogelijk euromunten?

96 eurocent gepast betalen met 5 euromunten:

- één halve euro (1×50 eurocent)
- twee kwintjes (2×20 eurocent)
- één stuiver (1×5 eurocent)
- één cent (1×1 eurocent)

Met minder munten kan niet

Voorbeeld: Specificatie van een programma

Invoer : Bedrag in eurocenten, tussen 0 en 500

Uitvoer : Tabel met aantal keer dat iedere munt nodig is om het ingevoerde bedrag met zo min mogelijk munten gepast te betalen

Beperkingen : Onbeperkt beschikbare euromunten met waarden:

1 2 5 10 20 50 100 200

Ongebruikte munten worden niet in de tabel vermeld

Voorbeeld: Dialoog met programma

```
96
1 x 50
2 x 20
1 x 5
1 x 1
```

Invoer is geel uitvoer is blauw

Voorbeeld: Ontwerp van een programma

Variabelen : Gehele getallen $bedrag$, $munt$, $aantal$

Stappen :

Lees $bedrag$ in

Laat $munt$ de waarden van de euromunten *dalend* doorlopen

Bepaal voor elke $munt$ het maximale $aantal$ dat gebruikt kan worden om het resterende $bedrag$ gepast te betalen en verlaag het $bedrag$ met elke gebruikte $munt$

Schrijf $aantal \times munt$, mits $aantal > 0$

Voorbeeld: Een eerste Python programma

```
# Een bedrag gepast betalen met zo min mogelijk euromunten

bedrag = input ( 'Bedrag tussen 0 en 500 eurocent: ' )

for munt in 200, 100, 50, 20, 10, 5, 2, 1 :
    aantal = 0

    while bedrag >= munt :
        aantal = aantal + 1
        bedrag = bedrag - munt

    if aantal > 0 :
        print aantal, 'x', munt
```

Voorbeeld: Systematisch testen van een programma

Invoer *bedrag*	**Verwachte uitvoer (tabel)**
0	(*geen*)
1	1x1
3	1x1, 1x2
4	2x2
200	1x200
388	1x200, 1x100, 1x50, 1x20, 1x10, 1x5, 1x2, 1x1
500	2x200, 1x100
−1	?
501	?
'abc'	?

Voorbeeld: Gebruik en onderhoud van een programma

- Een programma slijt niet, ook niet door intensief gebruik

- De omgeving van een programma evolueert mogelijk wel:

  andere muntsoorten

- De wensen van de gebruikers kunnen veranderen:

  herhaald een bedrag invoeren

  beperkte collectie munten gebruiken

  met wisselgeld passen

Trilogie van (programmeer)talen

- **Syntaxis** (vormleer): *Hoe ziet 't eruit, hoe schrijf je 't?*
- **Semantiek** (betekenisleer): *Wat betekent 't, hoe werkt 't?*
- **Pragmatiek** (gebuiksleer): *Hoe gebruik je 't in de praktijk?*

  conventies, methodes, technieken

Trilogie-voorbeeld: Verkeerslichten

Syntax (verschijningsvormen):

auto voetganger trein bus

Semantiek (bevelen): Stop Pas op Ga . . .

Pragmatiek (gewoontes): Door rood rijden, . . .

Syntactische bouwstenen van Python programmateksten

Commentaar	`# Een bedrag gepast betalen met ...`
Wit	*regelovergang* *spatie* *tab*
Namen	`bedrag` `input` `munt` `aantal`
Sleutelwoorden	`for` `in` `while` `if` `print`
Letterlijke waarden	`200` `100` `50` `20` `10` `5` `2` `1` `0` `'x'`
Operatoren	`>=` `+` `-` `>`
Scheiders	`=` `(` `)` `,` `:`

Commentaar en wit

```
# Een bedrag gepast betalen met zo min mogelijk euromunten

bedrag = input( 'Bedrag tussen 0 en 500 eurocent: ' )

for munt in 200, 100, 50, 20, 10, 5, 2, 1 :
    aantal = 0

    while bedrag >= munt :
        aantal = aantal + 1
        bedrag = bedrag - munt

    if aantal > 0 :
        print aantal, 'x', munt
```

Namen en sleutelwoorden

```
# Een bedrag gepast betalen met zo min mogelijk euromunten

bedrag = input ( 'Bedrag tussen 0 en 500 eurocent: ' )

for munt in 200, 100, 50, 20, 10, 5, 2, 1 :
    aantal = 0

    while bedrag >= munt :
        aantal = aantal + 1
        bedrag = bedrag - munt

    if aantal > 0 :
        print aantal, 'x', munt
```

Letterlijke waarden, operatoren en scheiders

```
# Een bedrag gepast betalen met zo min mogelijk euromunten

bedrag = input ( 'Bedrag tussen 0 en 500 eurocent: ' )

for munt in 200 , 100 , 50 , 20 , 10 , 5 , 2 , 1 :
    aantal = 0

    while bedrag >= munt :
        aantal = aantal + 1
        bedrag = bedrag - munt

    if aantal > 0 :
        print aantal , 'x' , munt
```

Geen commentaar, minimaal wit en kortste namen

Dit programma is syntactisch anders, maar semantisch hetzelfde:

```
a=input('Bedrag tussen 0 en 500 eurocent: ')
for b in 200,100,50,20,10,5,2,1:
 c=0
 while a>=b:
  c=c+1
  a=a-b
 if c>0:
  print c,'x',b
```

Maar zo doen we het niet!

Semantische bouwstenen van Python programma's

Naam–object bindingen :

Op elk moment refereert iedere naam aan één object

Uitdrukkingen evalueren :

```
input ( '...' )
```

```
bedrag >= munt
```

```
aantal + 1
```

```
bedrag - munt
```

Enkelvoudige opdrachten uitvoeren :

```
aantal = 0
```

```
aantal = aantal + 1
bedrag = bedrag - munt
```

```
print aantal, 'x', munt
```

Samengestelde opdrachten uitvoeren :

```
for munt in ... :
    ...
```

```
while ... :
    ...
```

```
if ... :
    ...
```

Kanttekeningen bij gepast betalen

- We hebben een zogenaamd gretig algoritme toegepast:

  Gebruik telkens *zoveel mogelijk* van de *grootste* kandidaat-munt

- Bij euromunten levert dit een minimale betaling

- In het algemeen is de gretige betaling echter *niet* minimaal

- Probeer maar eens 300 cent te passen met de munten

  5 10 100 250 500

  (de oude Nederlandse munten zonder kwartje)

Aandachtspunten bij programma's

- Functioneel correct (volgens specificatie)
- Nette opmaak, helder commentaar, zinvolle naamgeving
- Geschikt interface (gebruiksvriendelijke in- en uitvoer)
- Systematisch gecontroleerd

- Goede structuur (datastructuur, functies, modules)
- Efficiënt (tijd, geheugen)

Programmeertraining

- Taalkennis: Programmeertaal Python
- 'Tool'kennis: Programmeergereedschap Python IDE
- Algemene programmeer-technieken en -gewoontes
- Programmeer-oefeningen
- Oplossingen van opgaven laten beoordelen: PEACH