

Scripting using Python

Inhoudsopgave.

1. Inleiding.	3
• Literatuur.	4
• Informatie op internet.	4
2. De installatie van Python in Windows.	5
3. Berekeningen.	6
• Getallen.	6
• Typen.	9
• Variabelen.	9
• Rekenkundige uitdrukkingen.	11
4. Strings.	12
• Subscripting.	13
• Slicing.	13
• De lengte van een string bepalen.	14
• Een gedeelte van een string zoeken.	14
• Een gedeelte van een string vervangen.	15
• Strings achter elkaar plaatsen.	15
• Strings herhalen.	15
• Characters.	16
• Typecasting.	17
5. De print-opdracht.	19
6. Lijsten.	21
• Subscripting.	22
• Lijsten achter elkaar plaatsen.	23
• Lijsten herhalen.	23
• De lengte van een lijst bepalen.	23
• Een element in een lijst zoeken.	23
• Elementen uit een lijst verwijderen.	24
• Elementen in een lijst wijzigen.	24
• Elementen aan een lijst toevoegen.	25
• Een lijst sorteren.	25
• Een lijst omkeren.	25
• Het minimum en het maximum van een lijst bepalen.	26
• De som van de elementen van een lijst bepalen.	26
• De frequentie van een element in een lijst bepalen.	26
7. Splitsen en samenvoegen van strings.	27
8. Tupels.	28
9. Beweringen.	29
• Relationale operatoren.	29
• Logische operatoren.	29
• Lazy evaluation.	31

10. Opdrachten.	32
• De toekenningsopdracht.	32
• Samenvoegen en uitpakken.	33
• De toekenningsopdracht bij lijsten.	34
• Besturingsopdrachten.	35
• Het if-statement.	36
• Het if-else statement.	37
• Het geneste if-else statement.	38
• Range.	39
• Het for-statement.	40
• Het while-statement.	42
11. Dictionaries	43
• Het toevoegen van een (key,value)-paar aan een dictionary.	44
• Het wijzigen van een (key,value)-paar in een dictionary.	44
• Het verwijderen van een (key,value)-paar uit een dictionary.	44
• Alle keys opvragen.	44
• Alle values opvragen.	45
• Een dictionary doorlopen.	45
• Dictionaries samenvoegen.	46
• Een dictionary maken met keywords.	46
12. Functies	47
• De definitie van een functie.	47
• De aanroep van een functie.	48
• Globale en lokale variabelen.	48
• De waarde, die een functie teruggeeft.	50
• Parameters.	51
• Meerdere waarden teruggeven.	53
• Parameters met default-waarden.	54
13. Modulen.	55
• Een module importeren.	56
• Een module vinden.	57
• De inhoud van een module achterhalen.	57
• Win32-opdrachten.	59
14. Files.	60
• Lezen van en schrijven naar files.	60
• Tekstfiles en binaire files.	63
• Files kopiëren.	64
• Het filesysteem.	65
15. Foutafhandeling.	67
16. Klassen.	69
• De opbouw van een klasse.	69
• Overerving.	73
• Klasse-attributen.	75
• De methoden van een object achterhalen.	76
• De stack.	78
• Email verzenden.	80

1. Inleiding.

Scripting kan gebruikt worden voor het automatiseren van de dagelijkse systeembeheer-taken. Hiermee kan het systeembeheer vereenvoudigd worden.

Voorbeelden van systeembeheer-taken zijn:

- installeren en configureren van (netwerk-)systemen
- installeren, configureren en updaten van software
- performance meten
- informatie van (remote) computers en devices opvragen
- monitoren netwerkverkeer
- overzicht-documenten genereren
- toevoegen van gebruikers en het vastleggen van gebruikersrechten
- backup verzorgen
- automatisch een mail sturen
- een log-file analyseren

Het automatiseren heeft de volgende voordelen:

- de taken worden sneller en met minder fouten uitgevoerd
- de taken kunnen onbeperkt lang en op de gekste momenten worden uitgevoerd
- de beheerder hoeft niet steeds dezelfde handelingen te doen.

Het verschil tussen scripting en programmeren is:

- scripts zijn klein
- scripts worden direct uitgevoerd
- scripts maken gebruik van andere programma's
- scripts zijn dynamisch

Er zijn heel veel scripttalen.

Bekende scripttalen op Unix zijn: Perl, Python en Ruby.

Bekende scripttalen op Windows zijn: Kixstart, Vbscript, Powershell, Python en Ruby

Bij een web-omgeving worden de scripttalen javascript en php gebruikt.

In dit vak is gekozen voor de taal Python.

Hiervoor zijn een aantal redenen:

- Python is platform-onafhankelijk
- Python is breed toepasbaar: systeembeheer, gaming, wiskunde, website
- Python is vrij beschikbaar
- Python is goed leesbaar en heeft een nette structuur
- Python is geschikt om de basisprincipes van scripting te leren.
Wat geleerd is kan ook toegepast worden bij andere scripttalen.
- Python is ook geschikt voor Windows. Er is veel beschikbaar: win32api-module, IronPython, etc

In dit vak leer je de basisbegrippen van scripting.

Literatuur.

Geschiedte boeken zijn:

- Learning Python , Mark Lutz & David Asher, O'Reilly, ISBN 0-596-00281-5
- Programming Python, Mark Lutz, O'Reilly, ISBN 0-596-00925-9

Informatie op internet.

Op internet kun je allerlei sites over Python raadplegen:

python: <http://www.python.org>

tutorial1: <http://www.swaroopch.com/notes/Python>

tutorial2: <http://docs.python.org/3.1/tutorial/index.html>

IDLE: http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/IDLE-vertaling.html

howto: <http://docs.python.org/howto/>

python editors: <http://wiki.python.org/moin/PythonEditors>

Voor beginners: <http://wiki.python.org/moin/BeginnersGuide>

Let op: Dit dictaat is gebaseerd op Python 3.x.

Veel materiaal op internet is gebaseerd op Python 2.x

Python 3.x is echter niet compatibel met Python 2.x

2. De installatie van Python in Windows.

In 'Installatie Python.pdf' staat beschreven, hoe Python op de computer geïnstalleerd wordt.

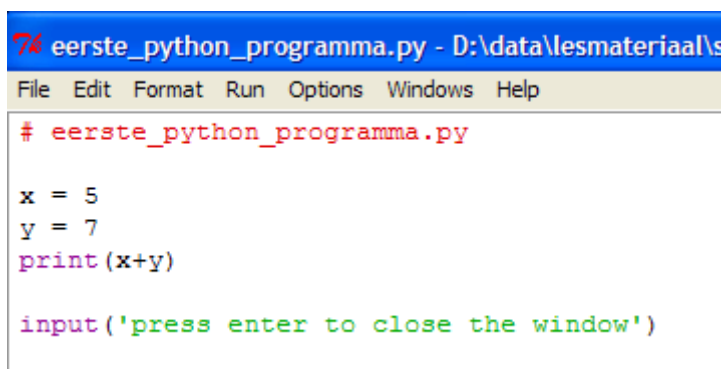
Een korte demonstratie.

- Start het programma 'IDLE'
- Tik in: 3+4
x = 5
y = 7
x+y

Resultaat:

```
>>> 3+4
7
>>> x=5
>>> y=7
>>> x+y
12
>>>
```

Deze opdrachten kunnen ook in een file met extensie 'py' geschreven worden en vervolgens uitgevoerd worden:



```
# eerste_python_programma.py

x = 5
y = 7
print(x+y)

input('press enter to close the window')
```

De file heeft extensie 'py'.

Alles wat achter '#' staat wordt gezien als commentaar en genegeerd.

In een python-file moet de opdracht 'x+y' vervangen worden door : 'print(x+y)'

Het programma kan op verschillende manieren opgestart worden

- IDLE, menu 'Run', submenu 'Run Module' (sneltoets: F5)
- op 'eerste_python_programma.py' dubbelklikken
(de regel “input('press enter to close the window')” is nodig, omdat anders het dos-scherm zeer kort wordt getoond)

3. Berekeningen.

Voor het uitvoeren van berekeningen heb je getallen, variabelen en rekenkundige uitdrukkingen nodig.

Getallen.

Getallen zijn nodig voor het uitvoeren van berekeningen.

Python kent drie soorten getallen:

- gehele getallen
- gebroken getallen
- complexe getallen

Complexe getallen zullen niet worden behandeld.

Gehele getallen

Gehele getallen zijn: 0, 1, 2, 3, ... en -1, -2, -3, -4, ...

Op gehele getallen kunnen bewerkingen worden uitgevoerd.
De bewerkingen worden gedaan met een operator

Het volgende tabel geeft een overzicht van de bewerkingen.

bewerking	operator
optellen	+
aftrekken	-
vermenigvuldigen	*
delen	/
geheel delen	//
de rest na een gehele deling bepalen	%
tot de macht verheffen	**

De gehele deling.

Bij de gehele deling wordt het gedeelte achter de punt weggelaten.

Voorbeeld:

normale deling: $17/3 = 5.666666666666667$

gehele deling: $17//3 = 5$

De rest na deling wordt verkregen met de operator '%' (spreek uit: modulo)

Voorbeeld: $17\%3 = 2$ want $17//3 = 5$ en de rest na deling is $17 - 3*5 = 2$

Machtsverheffen.

Bij het machtsverheffen wordt een getal een aantal keer met zichzelf vermenigvuldigd.

a tot de macht n:
$$a^n = \underset{\substack{\longleftarrow \hspace{1cm} \longrightarrow \\ n \text{ factoren}}}{a * a * \dots * a}$$

Bij de computer komen machten van twee vaak voor.

- 1 byte = 8 bits = 2^3 bits
- 1 kilobyte = 1024 byte = 2^{10} byte
- 1 gigabyte = 1024 megabyte = 1024.1024 kilobyte = 1024.1024.1024 byte = 2^{30} byte

In python kan 2^{30} als volgt worden berekend:

```
>>> 2**30
1073741824
```

Hoe wordt 2^{3*4} berekend?

Dit gaat als volgt:

- eerst wordt 3^4 berekend. Uitkomst: 81
- vervolgens wordt 2^{81} berekend. Uitkomst: 2417851639229258349412352

Er geldt dus: $2^{3*4} = 2^{(3^4)}$

Machtsverheffen gebeurt dus van rechts naar links.

De volgorde van berekening is van belang.

Er geldt: $(2^3)^4 \neq 2^{(3^4)}$

want:

$$(2^3)^4 = 8^4 = 4096$$
$$2^{(3^4)} = 2^{81} = 2417851639229258349412352$$

Om vergissingen te voorkomen is het verstandig haakjes te gebruiken.

Vraag: hoeveel bytes is 1 terabyte = 1024 gigabyte?

Binaire, octale en hexadecimale representatie.

Gehele getallen kunnen ook binair, octaal en hexadecimaal weergegeven worden:

Binaire weergave:

bin(1023): 0b1111111111

Octale weergave:

oct(8): 0o10

Hexadecimale weergave:

hex(16): 0x10

hex(255) 0xff

Gebroken getallen

Gebroken getallen bevatten een decimale punt of een exponent.

Voorbeelden van gebroken getallen zijn:

- 17.0
- 0.33333333333333
- 8e3 (= $8 \cdot 10^3 = 8000.0$)
- 43.5e-2 (= $43.5 \cdot 10^{-2} = 0.435$)

Gebroken getallen na 16 decimalen afgebroken.

Hierdoor worden ze niet altijd exact opgeslagen.

Je moet daarom rekening houden met een kleine afwijking.

Voorbeeld 1:

In de wiskunde geldt: $25 \cdot 0.28 = 7$, want $25 \cdot 0.28 = 25 \cdot 28/100 = 700/100 = 7$
De computer slaat echter 0.28 in de binaire representatie niet exact op.

In Python is

$$25 \cdot 0.28 - 7 = 8.881784197001252e-16 \text{ (dit is erg klein)}$$

Voorbeeld 2:

In de wiskunde is $1.0000000000000001 - 1 = 0.0000000000000001 = 10^{-16}$
 $1/3 - 0.3333333333333333 = (1/3) \cdot 10^{-16}$

In Python is

$$1.0000000000000001 - 1 = 0.0$$
$$1/3 - 0.3333333333333333 = 0.0$$

Typen

In Python zijn gegevens van een bepaald type.

Gehele getallen zijn van het type 'int' ('int' staat voor 'integer').

Gebroken getallen zijn van het type 'float' ('float' staat voor 'floating point').

'typecasting' zet het ene type getal om in het andere type getal.

Voorbeeld : $\text{int} \rightarrow \text{float} : \text{float}(3) = 3.0$
 $\text{float} \rightarrow \text{int} : \text{int}(4.7) = 4$

Bij 'typecasting' van 'float' naar 'int' wordt dus afgebroken i.p.v. afgerond.

Het afronden gaat met de functie 'round'.

Voorbeeld: $\text{round}(4.7) = 5$

Variabelen

Een variabele is een naam (identifier) die aan een gegeven wordt toegekend.

Bij Python is een identifier opgebouwd uit letters, cijfers en underscores (_) en mag niet beginnen met een cijfer.

In een eerder voorbeeld werden de variabelen x en y gebruikt:

```
x = 5          # spreek uit: x wordt 5
y = 7          # y wordt 7
print(x+y)
```

De variabelen worden in een tabel (dictionary) bijgehouden:

<i>Variabele</i>	<i>Type</i>	<i>Waarde</i>
x	int	5
y	int	7

De opdracht 'print dir()' geeft de lijst van variabelen. Ook de variabelen, die door Python zelf worden gebruikt, worden getoond.

De variabele 'y' kan worden verwijderd met de opdracht: 'del y'.

Voor een variabele wordt geheugenruimte gereserveerd. Als een variabele wordt verwijderd, wordt de geheugenruimte vrijgegeven.

Bekijk het programma 'var_demo.py':

```
x = 7
y = 9
print('x:',x)
print('y:',y)
print('type(x):',type(x))
print('x+y:',x+y)
print('dir():',dir())
print()
del y
x = 3.14159
print('type(x):',type(x))
print('dir():',dir())
```

De variabelen-tabel verandert tijdens de uitvoering van het programma:

variabelen-tabel

x = 7

<i>Variabele</i>	<i>Type</i>	<i>Waarde</i>
x	int	7

y = 9: tabel bevat nu ook 'y'

<i>Variabele</i>	<i>Type</i>	<i>Waarde</i>
x	int	7
y	int	9

del y: 'y' wordt uit tabel verwijderd

<i>Variabele</i>	<i>Type</i>	<i>Waarde</i>
x	int	7

x = 3.14159 : 'x' is van een ander type

<i>Variabele</i>	<i>Type</i>	<i>Waarde</i>
x	float	3.14159

Uitvoer:

```
x: 7
y: 9
type(x): <class 'int'>
x+y: 16
dir(): ['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'idlelib', 'x', 'y']

type(x): <class 'float'>
dir(): ['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'idlelib', 'x']
```

Rekenkundige uitdrukkingen.

Een rekenkundige uitdrukking is een combinatie van variabelen, constanten, operatoren en haakjes.

Voorbeelden van rekenkundige uitdrukkingen zijn:

$(a+3)*(b-4)$
 $a*a*a*a$
 $a**4$
 $1.0/(1+x**2)$

a, b, x : variabelen
 $1, 2, 3, 4, 1.0$: constanten
 $+, -, *, /, **$: operatoren

De berekeningen worden standaard van links naar rechts uitgevoerd.

Er geldt echter een prioriteitsvolgorde bij de operatoren:

- eerst machtsverheffen
- vervolgens vermenigvuldigen of delen
- tenslotte optellen of aftrekken

$3*2**10 = 3*(2**10)$
 $4+3*5 = 4+(3*5)$

Machtsverheffen gebeurt van rechts naar links: $3**2**5 = 3**(2**5)$

Advies: als je het niet precies weet, kun je het beste haakjes gebruiken.

4. Strings.

Een string is een stuk tekst.

Strings worden omsloten door '(enkele quote) of "(dubbele quote).

Voorbeelden van strings zijn:

'string'	# een string kan omsloten worden door enkele quotes
"nog een string"	# en ook door dubbele quotes
'a'	# een string, die bestaat uit één teken
"	# de lege string (twee enkele quotes)

Speciale strings zijn de volgende strings:

- een string die een enkele quote bevat
- een string die een dubbele quote bevat
- een string met speciale tekens (newline, tab, return, backslash)

Een string die een dubbele quote bevat kun je omsluiten met enkele quotes

Een string die een enkele quote bevat kun je omsluiten met dubbele quotes

Voorbeeld:

```
"aa'bb"  
'aa"bb'
```

Een string, die een enkele quote en een dubbele quote bevat, wordt beschreven met het escape-teken '\' (backslash)

Voorbeeld:

```
'aa'bb\"cc' # de tweede backslash is in dit voorbeeld niet nodig.
```

Het escape-teken '\' wordt ook gebruikt voor speciale tekens: \n (newline)

\t (tab)

\r (return)

\\ (backslash)

Met een string kun je allerlei dingen doen:

- subscripting (nummering).
- slicing (snijden).
- de lengte van een string bepalen.
- een gedeelte van de string zoeken.
- strings achter elkaar plaatsen.
- een string herhalen.

Subscripting.

De elementen van een string zijn op twee manieren genummerd:

s	s	t	r	i	n	g
van links naar rechts:	0	1	2	3	4	5
van rechts naar links:	-6	-5	-4	-3	-2	-1

Het nummer van een element uit de string wordt de index genoemd.

Met de index kan een element aangegeven worden.

Dit wordt subscripting genoemd.

```
s = 'string'
```

Er geldt:

```
s[0] = s[-6] = 's'
```

```
s[1] = s[-5] = 't'
```

```
s[2] = s[-4] = 'r'
```

```
s[3] = s[-3] = 'i'
```

```
s[4] = s[-2] = 'n'
```

```
s[5] = s[-1] = 'g'
```

Slicing.

Met een slice kan een gedeelte van een rij geselecteerd worden.

Voorbeeld

s	s	t	r	i	n	g
	0	1	2	3	4	5
	-6	-5	-4	-3	-2	-1

```
s = 'string'
```

```
s[2:4] = 'ri' (laatste index doet niet mee)
```

```
s[2:] = 'ring' (vanaf 2, 2 doet mee)
```

```
s[:4] = 'stri' (tot aan 4, 4 doet niet mee)
```

```
s[:] = 'string' (vanaf het begin tot het einde)
```

```
s[:-1] = 'strin' (tot één positie voor het einde)
```

```
s[:-2] = 'stri' (tot twee posities voor het einde)
```

Er kan een stapgrootte opgegeven worden.

```
s = 'abcdefghijk'
```

```
s[3:9] = 'defghi'
```

```
s[3:9:2] = 'dfh'    (stapgrootte 2)
```

De stapgrootte kan negatief zijn. De string wordt dan van achteren naar voren doorlopen.

```
s[9:3:-1] = 'jihgfe'
```

```
s[9:3:-2] = 'jhf'
```

```
s[::-1] = 'kjihgfedcba'
```

```
s[-1::-1] = 'kjihgfedcba'
```

```
s[-1:0:-1] = 'kjihgfedcb'
```

De lengte van een string bepalen.

De lengte van een string wordt met 'len'-functie verkregen:

```
len('string') = 6
```

```
len('dit is een lange string') = 23 ( de spaties worden meegerekend)
```

```
len('DitIsEenLangeString') = 19
```

Een gedeelte van een string zoeken

Met de find-methode wordt een gedeelte van een string opgezocht.

```
s = 'string'
```

```
s.find('tri') = 1    # beginpositie van gezochte string is 1
```

```
s.find('tri',2) = -1  # zoek vanaf positie 2
```

```
                    # -1 betekent: niet gevonden
```

```
s.find('tri',1,4) = 1  # zoek vanaf positie 1 tot aan positie 4
```

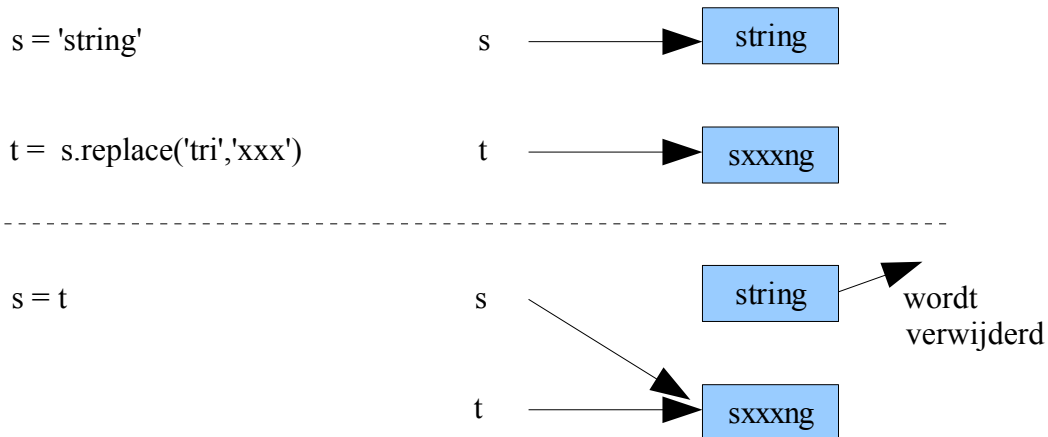
```
s.find('tri',1,3) = -1
```

Een gedeelte van een string vervangen.

Strings zijn niet te wijzigen. Als je een string wilt aanpassen, dan moet je het vervangen door een andere string.

```
s = 'string'
t = s.replace('tri','xxx')    # t = 'sxxxng' , t is een nieuwe string
s = t                        # s = 'sxxxng'
```

Hoe werkt het?



De opdrachten

```
t = s.replace('tri','xxx')
s = t
```

kunnen tot één opdracht worden samengevoegd:

```
s = s.replace('tri','xxx')
```

Let op:

```
s.replace('tri','xxx') : s is niet veranderd
s = s.replace('tri','xxx') : s is veranderd
```

Strings achter elkaar plaatsen.

Met de '+'-operator kunnen twee strings aan elkaar gekoppeld worden:

```
'abc' + 'xyz' = 'abcxyz'
```

Strings herhalen.

Met de '*'-operator kan een string herhaald worden.

```
'abc'*4 = 'abcabcabcabc'
```


Characters

Een character is een string met lengte 1.

Van een character kan de bijbehorende ord-waarde worden bepaald.

De ord-waarde is de positie van de character in de 7-bit ASCII-tabel.

	0	1	2	3	4	5	6	7
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
8	BS	HT	LF	VT	FF	CR	SO	SI
16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
24	CAN	EM	SUB	ESC	FS	GS	RS	US
32	SP	!	"	#	\$	%	&	'
40	()	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{		}	~	DEL

Uit de tabel kun je afleiden:

`ord('a') = 97`

`chr(97) = 'a'`

Voor de waarden 128 t/m 255 zijn verschillende ASCII-uitbreidingen gedefinieerd, zoals Latin-1 en Windows-1252

De dos-console kent een andere ASCII-uitbreiding. Deze is op te vragen met het commando 'chcp'.

Hiermee is het mogelijk een aantal speciale symbolen, zoals á, ë, ï en € te tonen.

Typecasting.

'40' is een string

40 is een geheel getal.

Strings en getallen zijn verschillend.

Vb1.

De operator '+' werkt bij strings anders dan bij getallen. Bij strings worden de twee strings achter elkaar geplaatst.

'10' + '2' = '102'

10+2 = 12

Vb2. Strings laten zich onderling anders vergelijken dan getallen.

'groot' < 'klein', want 'g' komt eerder in alfabet voor dan 'k'

'10' < '2', want '1' komt in het ASCII-alfabet eerder voor dan '2'

'+10' < '1', want '+' komt in het ASCII-alfabet eerder voor dan '2'

Soms is het nodig een string om te zetten naar een getal of een getal om te zetten naar een string.

```
str(40)    = '40'
int('40')  = 40
float('40') = 40.0
```

Voorbeeld: interactie met de gebruiker.

Bij dit programma wordt de gebruiker gevraagd twee getallen in te tikken. Vervolgens wordt de som getoond.

Het programma 'interactie_demo.py' bevat de volgende tekst:

```
s = input('x: ')
x = int(s)
s = input('y: ')
y = int(s)
print('de som van', x, 'en', y, 'is:', x+y)
input('press enter to close the window')
```

Twee getallen worden ingelezen en de som wordt uitgevoerd. .

Uitvoer:

```
x: 34
y: 56
de som van 34 en 56 is: 90
press enter to close the window
```

De regel "input('press enter to close the window')" is binnen IDLE niet nodig. Als je in een dosbox op 'interactie_demo.py' dubbelklikt is de regel wel nodig. Vraag: wat gebeurt er als de invoer geen getal is?

5. De print-opdracht.

Standaard geldt het volgende:

- bij een print-opdracht worden de gegevens gescheiden door een spatie.
- na de print-opdracht wordt begonnen met een nieuwe regel.

We laten zien hoe we hiervan kunnen afwijken

De opdracht

```
print(27,39,68)
```

heeft de volgende uitvoer:

```
27 39 68
```

Tussen de getallen staat één spatie.

Wat tussen de getallen staat wordt een 'separator' genoemd.

Je kunt de separator een andere waarde geven.

Voorbeeld:

```
print(27,39,68,sep='#')
```

heeft de volgende uitvoer:

```
27#39#68
```

Een separator kan uit meer dan één karakter bestaan.

De opdracht

```
print(27,39,68,sep = '%&$')
```

heeft de volgende uitvoer:

```
27%&$39%&$68
```

Na een print-opdracht wordt standaard met een nieuwe regel begonnen.

De opdrachten

```
print(27)  
print(39)
```

geven de volgende uitvoer:

```
27  
39
```

Het volgende voorbeeld laat zien hoe je kunt voorkomen, dat je naar een volgende regel gaat.

```
print(27,end = ' ')
print(39)
```

levert de volgende uitvoer op:

```
27 39
```

Je kunt de twee getallen achter elkaar plaatsen zonder iets ertussen.

Voorbeeld:

```
print(27,end = " ")      # ": twee enkele quotes achter elkaar: de lege string
print(39)
```

Uitvoer

```
2739
```

'sep' en 'end' zijn ook te combineren:

Voorbeeld:

```
print(27,39,sep='@',end='#')
print(99)
```

Uitvoer:

```
27@39#99
```

6. Lijsten.

Binnen python spelen lijsten een centrale rol.

Een lijst is een rij met elementen. De elementen worden gescheiden door een komma. De elementen worden omsloten door blokhaken.

Voorbeelden van lijsten zijn:

- een lijst van getallen: [3, -40 , 1e-4]
- een lijst van strings: ['aap','noot','mies']
- een lijst met van alles: ['aap', 3, [3, -40 , 1e-4], ['noot','mies']]

Voorbeelden, die we later tegenkomen, zijn:

- een lijst van filenamen uit een map
- een lijst van woorden uit een tekstfile
- een lijst van regels uit een file
- een lijst van partities

Ook bij lijsten kun je het volgende doen:

- subscripting
- slicing
- aan elkaar koppelen
- herhalen

Verder is het volgende mogelijk:

- de lengte van de lijst bepalen
- een element opzoeken, verwijderen, wijzigen of toevoegen
- de lijst sorteren
- het minimum en maximum van de lijst bepalen
- de som van de elementen van de lijst berekenen.
- bepalen hoe vaak een element in de lijst voorkomt

Subscripting

Voorbeeld:

$a = [1, 99, 2, 98, 3, 97]$

a:	1	99	2	98	3	97
	0	1	2	3	4	5
	-6	-5	-4	-3	-2	-1

Er geldt:

$a[0] = 1$	$a[-6] = 1$
$a[1] = 99$	$a[-5] = 99$
$a[2] = 2$	$a[-4] = 2$
$a[3] = 98$	$a[-3] = 98$
$a[4] = 3$	$a[-2] = 3$
$a[5] = 97$	$a[-1] = 97$

Slicing

Voorbeelden:

$a[1:3] = [99, 2]$
 $a[1:] = [99, 2, 98, 3, 97]$
 $a[:4] = [1, 99, 2, 98]$
 $a[:] = [1, 99, 2, 98, 3, 97]$
 $a[1:5] = [99, 2, 98, 3]$

Slicing met stapgrootte:

$a[1:5:2] = [99, 98]$
 $a[1::2] = [99, 98, 97]$
 $a[:4:2] = [1, 2]$
 $a[::2] = [1, 2, 3]$

Slicing met negatieve stapgrootte

$a[4:1:-1] = [3, 98, 2]$
 $a[4::-1] = [3, 98, 2, 99, 1]$
 $a[:: -1] = [97, 3, 98, 2, 99, 1]$
 $a[:: -2] = [97, 98, 99]$

Lijsten achter elkaar plaatsen.

`[71,72,73] + [1,2,3] = [71, 72, 73, 1, 2, 3]`

Lijsten herhalen.

`[1,2,3]*3 = [1, 2, 3, 1, 2, 3, 1, 2, 3]`
`3*[1,2,3] = [1, 2, 3, 1, 2, 3, 1, 2, 3]`
`[8]*10 = [8,8,8,8,8,8,8,8,8,8]`

De lengte van de lijst bepalen.

`len([3, -40 , 1e-4]) = 3`
`len(['aap', 3, [3, -40 , 1e-4], ['noot','mies']]) = 4`

Een element in de lijst zoeken.

`lijst = [1,2,100,2,1]`
`lijst.index(100) = 2` # zoek vanaf index 0
`lijst.index(2) = 1` # resultaat: laagste index
`lijst.index(2,2) = 3` # zoek vanaf index 2
`lijst.index(1) = 0`
`lijst.index(1,2) = 4`

`lijst.index(1,1,4) : foutmelding` # zoek vanaf index 1 tot aan index 4

foutmelding:

Traceback (most recent call last):

File "<interactive input>", line 1, in <module>

ValueError: list.index(x): x not in list

Elementen uit de lijst verwijderen.

Demo:

```
a = [1,2,3,4,5,6,7,8]
```

opdracht	resultaat
del a[6]	a = [1, 2, 3, 4, 5, 6, 8] # verwijder het zevende element
del a[1:3]	a = [1, 4, 5, 6, 8] # verwijder het 2e en het 3e element
del a[:2]	a = [5, 6, 8] # verwijder de eerste twee elementen
del a[1:]	a = [5] # verwijder alle elementen behalve het eerste element
del a[:]	a = [] # verwijder alle elementen

```
b = [1,2,3,2,1]
```

opdracht	resultaat
b.remove(2)	b = [1,3,2,1] # verwijder 2 met de laagste index # hetzelfde als: del b[b.index(2)]

Er is een verschil tussen 'del a[:]' en 'del a'

```
a = [1,2,3,4,5,6,7,8]
```

opdracht	resultaat
del a	a bestaat niet
del a[:]	a = []

Elementen uit de lijst wijzigen.

Demo:

```
a = [1,2,3,4,5,6,7,8]
```

opdracht	resultaat
a[2] = 99	a = [1, 2, 99, 4, 5, 6, 7, 8] # vervang het derde element door 99
a[1:3] = [88]	a = [1, 88, 4, 5, 6, 7, 8] # vervang a[1] en a[2] door 1 element
a[1:3] = [77,177,277]	a = [1, 77, 177, 277, 5, 6, 7, 8] # vervang a[1] en a[2] door 3 elementen
a[2:5] = []	a = [1, 77, 6, 7, 8] # hetzelfde als del a[2:5]

Vraag wat gebeurt er als " a[1:3] = [88] " wordt vervangen door " a[1:3] = 88 " ?

Elementen aan een lijst toevoegen.

Demo:

```
b = [1, 1, 1, 1]
```

opdracht

resultaat

```
b[2:2] = [8,88,888]    b = [1, 1, 8, 88, 888, 1, 1]    # voeg voor positie 2 toe:
                        8,88,888
b[:0] = [77]           b = [77, 1, 1, 8, 88, 888, 1, 1] # voeg voor de lijst toe: 77
b.extend([99,999])     b = [77, 1, 1, 8, 88, 888, 1, 1, 99, 999]
                        # voeg aan het eind van
                        de lijst toe: 99,999
```

*Vraag: wat gebeurt er als " b[2:2] = [8,88,888] " wordt vervangen door
" b[2] = [8,88,88] " ?*

Een lijst sorteren.

Demo:

```
a = [6,2,1,8,-14]
```

opdracht

resultaat

```
a.sort()              a = [-14, 1, 2, 6, 8] # sorteer van klein naar groot
```

```
b = [5,1,2,-7,23]
```

opdracht

resultaat

```
b.sort(reverse=True)  b = [23, 5, 2, 1, -7] # sorteer van groot naar klein
b.sort()              b = [-7, 1, 2, 5, 23]
```

Een lijst omkeren.

Demo:

```
a = [1,2,3,4,5]
```

opdracht

resultaat

```
a.reverse()           a = [5,4,3,2,1]
```

Het minimum en het maximum van de lijst bepalen.

```
a = [5,2,7,1,9,4]
```

```
min(a)      1
max(a)      9
```

De som van de elementen van de lijst berekenen.

```
sum([5,2,-7,1,9,4])    14
sum([5,2,-7,1.0,9,4])  14.0
sum([])                 0
```

De frequentie van een element in een lijst bepalen.

Demo:

```
a = [1,2,3,2,1,2,3,4,3,2,1]
```

```
a.count(2)      4
a.count(10)     0
```

Recapulatie.

Op de lijst a zijn de volgende functies van toepassing:

- len(a)
- max(a)
- min(a)
- sum(a)

(Functies berekenen iets. Functies worden later uitgebreid behandeld)

De volgende methoden kunnen op een lijst a toegepast worden:

- a.remove(...)
- a.index(...)
- a.extend(...)
- a.sort(...)
- a.reverse()

(Methoden voeren een bewerking uit. Methoden worden later behandeld)

Functies kunnen a niet veranderen.

Methoden kunnen a wel veranderen.

7. Splitsen en samenvoegen van strings.

Een string kan op verschillende manieren gesplitst worden. Het resultaat van een dergelijke splitsing is een lijst.

Splitsen in karakters.

Op basis van de string wordt een lijst met afzonderlijke karakters gemaakt

```
s = 'string'
list(s) = ['s', 't', 'r', 'i', 'n', 'g']
```

Splitsen in regels.

Een string kan meerdere regels bevatten. Het einde van een regel wordt aangegeven met '\n'. De regels kunnen afzonderlijk in een lijst worden geplaatst.

```
regels = 'een\ntwee\ndrie'
regels.splitlines() = ['een', 'twee', 'drie']
```

Splitsen op basis van een zelfgekozen deelstring.

```
regel = 'een twee drie'
regel.split(' ') = ['een', 'twee', 'drie']
```

Vraag 1: wat gebeurt er als de opdracht "b = regels.splitlines(True)" wordt gegeven?

Vraag 2: wat is het verschil tussen regel.split(' ') en regel.split() ?

Het samenvoegen gebeurt m.b.v. 'join'

Het samenvoegen van karakters tot één string.

```
a = ['s', 't', 'r', 'i', 'n', 'g']
''.join(a) = 'string'
```

Het samenvoegen van regels tot één string.

```
b = ['regel1', 'regel2', 'regel3']
'\n'.join(b) = 'regel1\nregel2\nregel3'
```

Het samenvoegen van woorden tot één string.

```
b = ['woord1', 'woord2', 'woord3']
''.join(b) = 'woord1 woord2 woord3'
```

8. Tupels

Tupels zijn lijsten, die niet gewijzigd kunnen worden. (Engels: immutable)

Lijsten worden omsloten door blokhaken.

Tupels worden omsloten door ronde haken.

Een tuple kan m.b.v. 'typecasting' worden omgezet in een lijst en andersom.

Demo:

```
>>> t = (1,2,3)
>>> type(t)
<class 'tuple'>
>>> a = list(t)
>>> a
[1, 2, 3]
>>> u = tuple(a)
>>> u
(1, 2, 3)
```

Een lege tuple wordt aangegeven met : ()

Hoe wordt een tuple met één element weergegeven?

Je bent geneigd een tuple met alleen het getal 3 weer te geven met: (3)

Er geldt echter $3 == (3) == ((3)) == (((3)))$

Daarom wordt een tuple met één element voorzien met een extra komma:

een tuple met het getal 3 wordt als volgt weergegeven: (3,)

Samengevat:

(3,) is een tuple met één element. Het element is in dit geval 3

(3) is gelijk aan 3

Recapitulatie

We hebben drie nu drie datastructuren behandeld: string, list en tuple

datastructuur	beschrijving	omsluiting	
string	een rij karakters	quotes	immutable
list	een rij elementen	blokhaken	mutable
tuple	een rij elementen	ronde haken	immutable

9. Beweringen.

Voorbeelden van beweringen zijn:

i is kleiner dan j
de lengte van de string is gelijk aan 4
het kwadraat van k is deelbaar door 3
het verschil tussen a en b is kleiner of gelijk aan 6

Beweringen kunnen waar of niet waar zijn.

Bij Python wordt 'waar' aangegeven met: 'True'
en 'niet waar' aangegeven met 'False'

Bij beweringen (ook wel logische expressies genoemd) worden relationele en logische operatoren gebruikt.

Relationele operatoren.

De relationele operatoren staan in de volgende tabel:

operator	betekenis
<	kleiner dan
<=	kleiner of gelijk aan
==	is gelijk aan
>=	groter of gelijk aan
>	groter dan
!=	is ongelijk aan

Logische operatoren.

De logische operatoren staan in de volgende tabel:

operator	betekenis
and	en
or	of
not	niet

De logische operatoren worden volgens de volgende waarheidstabel vastgelegd:

b1	b2	b1 and b2	b1 or b2	not b1
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

In woorden : 'b1 and b2' is waar als b1 en b2 beide waar zijn.

'b1 or b2' is waar als b1 waar is of b2 waar is of b1 en b2 beide waar zijn.

Voorbeeld:

i = 3

j = 5

bewering

waarde

i < j

True

i == j

False

i < j and j < 10

True

i < j < 10

True

i < j < 10 < j*j <= 100 < j*j*j

True

i == 0 or j < 6

True

i == 3 or j == 5

True

Lazy evaluation

Het verwerken van logische expressies gebeurt op basis van "lazy evaluation".

Neem als voorbeeld de bewering:

$$(0 \leq i < \text{len}(a)) \text{ and } (a[i] > 100)$$

Eerst wordt nagegaan of " $0 \leq i < \text{len}(a)$ " waar is.

Als " $0 \leq i < \text{len}(a)$ " niet waar is dan weten we dat de hele bewering niet waar is.
Er wordt dan niet meer nagegaan of " $a[i] > 100$ " waar is.

Bekijk de volgende bewering::

$$(i > 100) \text{ or } (i < 0)$$

Eerst wordt nagegaan of " $i > 100$ " waar is.

Als " $i > 100$ " waar is, dan weten we dat de hele bewering waar is.
Er wordt dan niet meer nagegaan of " $i < 0$ " waar is.

Kortom:

"b1 and b2" : eerst wordt b1 geëvalueerd
als b1 niet waar is, wordt b2 niet geëvalueerd.

"b1 or b2" : eerst wordt b1 geëvalueerd
als b1 waar is, wordt b2 niet geëvalueerd

10. Opdrachten.

De volgende opdrachten worden behandeld:

- de toekenningsoopdracht
- pack en unpack
- de toekenningsoopdracht bij lijsten
- de keuzeopdracht
- de herhalingsopdracht

De toekenningsoopdracht.

De toekenningsoopdracht kent aan een gegeven een waarde toe.

Voorbeelden van toekenningsoopdrachten zijn:

```
x = 3
z = 8
y = (x + 1)*(z-4)  # x en z moeten bestaan
i = 11
i = i + 1           # verhoog i met 1
a = [i, 2*i, 3*i]
v = a[x-1]
```

Eerst wordt het gedeelte achter het '='-teken berekend. Vervolgens wordt het resultaat aan het object voor het '='-teken toegekend.

Er is een belangrijk verschil tussen "**x == 3**" en "**x = 3**".

"**x == 3**" betekent: "x is gelijk aan 3" . Dit is een bewering. Deze kan waar of niet waar zijn.
"**x = 3**" betekent : "x wordt 3" of "geef x de waarde 3" . Dit is een opdracht.

Verkorte schrijfwijzen.

De volgende tabel laat zien hoe je opdrachten korter kunt schrijven.

Statement(s)	Verkorte schrijfwijze
a = 3 b = 3	a = b = 3
a = 3 b = 4	a,b = 3,4
i = i + 5	i += 5
j = j -4	j -= 4
k = k*2	k *= 2
d = d/3	d /= 3

Samenvoegen en uitpakken.

Meerdere expressies kunnen worden samengevoegd tot één tuple. (pack)

Voorbeeld:

```
t = i+5, j-4, k*2, d/3
```

t is een tuple met vier elementen

Een tuple kan weer opgesplitst worden (unpack)

```
i,j,k,d = t
```

Demo:

```
>>> x = 3
>>> z = 8
>>> y = (x+1)*(z-4)
>>> x,y,z
(3, 16, 8) # tuple
>>> i = 11
>>> i += 1
>>> a = [i,2*i,3*i]
>>> a
[12, 24, 36]
>>> a = b = 3
>>> a,b
(3, 3)
>>> a,b = 3,4
>>> a,b
(3, 4)
>>> i,j,k,d = 10,20,30,40
>>> i+=5;j-=4;k*=2;d//=3
>>> i,j,k,d
(15, 16, 60, 13)
>>> i,j,k,d = 10,20,30,40
>>> t = (i+5,j-4,k*2,d//3) # pack in tuple
>>> i,j,k,d = t
>>> i,j,k,d
(15, 16, 60, 13)
>>> c1,c2,c3 = 'xyz' # unpack string
>>> c1,c2,c3
('x', 'y', 'z')
```

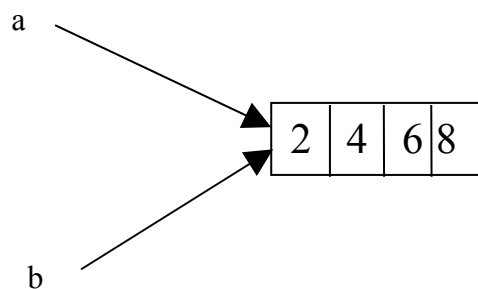
De toekenningsoopdracht bij lijsten.

Tupels en strings kunnen niet gewijzigd worden. Lijsten kunnen wel gewijzigd worden. Lijsten zijn verraderlijk: een lijst kan ook via een andere variabele gewijzigd worden.

Bekijk het volgende voorbeeld:

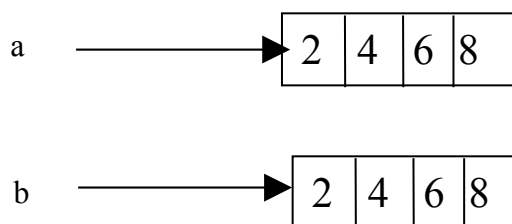
```
>>> a = [2,4,6,8]
>>> b = a
>>> b[2] = 22
>>> a
[2, 4, 22, 8]
```

De lijst a is veranderd via b. Dit komt omdat a en b naar hetzelfde object verwijzen.



Dit kun je oplossen door "b = a" te vervangen door "b = list(a)". Dan wordt een kopie van a gemaakt en aan b toegekend.

```
>>> a = [2,4,6,8]
>>> b = list(a)
>>> b[2] = 22
>>> a
[2, 4, 6, 8]
>>>
```



Dit gaat goed zolang de elementen van de lijst getallen of strings zijn. Als een element zelf een lijst is, dan moet een "diepe kopie" gemaakt worden.

Besturingsopdrachten.

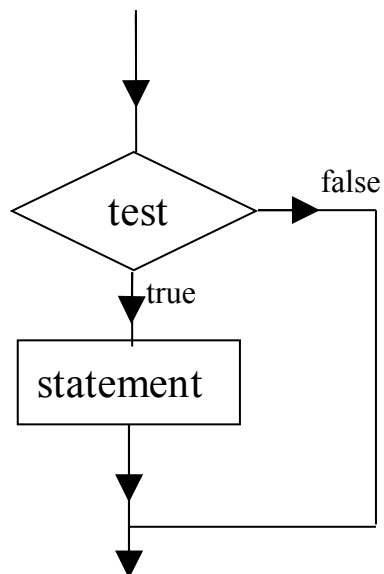
Besturingsopdrachten geven aan welke opdrachten uitgevoerd moeten. Ook wordt aangegeven in welke volgorde dit moet gebeuren.

Python kent de volgende besturingsopdrachten:

- keuzeopdrachten
 - o if statement
 - o if else statement
- herhalingsopdrachten
 - o for statement
 - o while statement
- sprongopdrachten
 - o break statement (wordt niet behandeld)
 - o continue statement (wordt niet behandeld)

Het if-statement

Het stroomdiagram van het if-statement is als volgt:



Het statement wordt uitgevoerd als 'test' de waarde 'True' heeft

In Python heeft het if-statement de volgende opbouw:

if test:	# na test wordt een ':' geplaatst
statement	# eerst inspringen, dan de statement

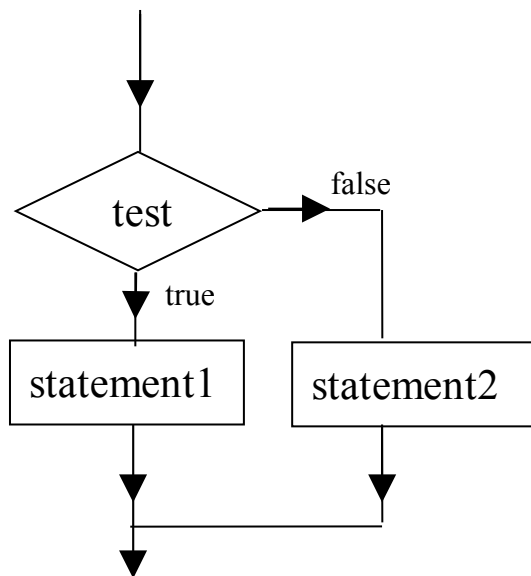
Voorbeelden:

```
s = input('geef de waarde van i: ')
i = int(s)
if i%3 == 0:
    print(i, 'is deelbaar door 3')
```

```
s = input('geef saldo: ')
saldo = int(s)
if saldo < 0:
    print('waarschuwing!!!')
    print('het saldo is negatief')
```

Het if-else statement

Het stroomdiagram van het if-else statement ziet er als volgt uit:



```
if test:
    statement1
else:
    statement2
```

Voorbeelden:

```
s = input('geef de waarde van i: ')
i = int(s)
if i%3 == 0:
    print(i, 'is deelbaar door 3')
else:
    print i, 'is niet deelbaar door 3'
```

```
s = input('geef saldo: ')
saldo = int(s)
if saldo < 0:
    print('waarschuwing!!!')
    print('het saldo is negatief')
else:
    print('gefeliciteerd!')
    print('het saldo is positief')
```

Het geneste if-else statement.

Bij een geneste if-else opdracht wordt een if-else opdracht binnen een andere if-else opdracht geplaatst.

Voorbeeld:

```
x = int(input('geef de waarde van x: '))  
y = int(input('geef de waarde van y: '))
```

```
if x<y:  
    print('x is kleiner dan y')  
else:  
    if x>y:  
        print('x is groter dan y')  
    else:  
        print('x is gelijk aan y')
```

In Python kan dit ook anders:

```
if x<y:  
    print('x is kleiner dan y')  
elif x>y:  
    print('x is groter dan y')  
else:  
    print('x is gelijk aan y')
```

De laatste vorm heeft twee voordelen:

- het vraagt minder tikwerk
- er wordt minder diep ingesprongen.

Range.

Het for-statement kan gebruik maken van een range.

Met een range kan een aantal waarden worden doorlopen.
De volgende tabel laat dit zien.

range	waarden
range(10)	0,1,2,...,9
range(3,10)	3,4,..., 9
range(-4,7)	-4,-3,-2,-1,0,1,...,6
range(0,10,2)	0,2,4,...,8
range(10,3,-1)	10,9,8,...,4
range(7,-4,-2)	7,5,3,1,-1,-3

Je kunt de waarden ook in een lijst opslaan:

list(range(7,-4,-2)) = [7,5,3,1,-1,-3]

Het for-statement.

Het for-statement wordt o.a. gebruikt voor het doorlopen van een range, een lijst, een tuple of een string.

We geven enkele voorbeelden:

Voorbeeld 1: druk 11^2 , 12^2 , ..., 20^2 af

Dit gaat als volgt:

```
for i in range(11,21):    # i heeft achtereenvolgens de waarden 11,12,...,20
    print(i*i)
```

Na iedere print-opdracht begint de uitvoer op een nieuwe regel

Dit kan voorkomen worden door bij de print-opdracht een andere 'end'-waarde mee te geven.

```
for i in range(11,21):
    print(i*i,end=' ')
```

De uitvoer is nu:

121 144 169 196 225 256 289 324 361 400

Voorbeeld 2: plaats 11^2 , 12^2 , ..., 20^2 in een lijst.

Oplossing:

```
b = []                # b is een lege lijst
for i in range(11,21):
    b.append(i*i)
print('b = ',b)
```

Uitvoer: [121, 144, 169, 196, 225, 256, 289, 324, 361, 400]

Het kan ook korter:

```
b = [i*i for i in range(11,21)]
print('b = ',b)
```

Voorbeeld 3: plaats van de kwadraten 11^2 , 12^2 , ..., 20^2 , alleen de kwadraten, die niet deelbaar zijn door 3, in een lijst.

Oplossing:

```
a = []
for i in range(11,21):
    k = i*i
    if k%3 != 0:
        a.append(k)
print('a = ',a)
```

Het kan ook korter:

```
a = [i*i for i in range(11,21) if i%3 != 0]
print('a = ',a)
```

Het for-statement kan i.p.v. ranges ook toegepast worden op strings, lijsten en tupels.

Voorbeeld 4: druk de karakters van een string afzonderlijk af.

Dit gaat als volgt:

```
for ch in 'string':
    print(ch, end = ' ')
```

Uitvoer: s t r i n g

Voorbeeld 5: plaats 2^n met n in $[41,73,52]$ in een lijst.

Oplossing:

```
a = [41,73,52]
b = [2**n for n in a]
print('b = ',b)
```

Uitvoer: b = [2199023255552, 9444732965739290427392, 4503599627370496]

Voorbeeld 6: Van een lijst van strings wordt een frequentie-lijst gemaakt. De frequentie-lijst geeft aan hoe vaak de letter 'e' voorkomt in de string.

Oplossing:

```
a = ['hoeveel','genezen','bereveel']
u = [list(i).count('e') for i in a]
print('u = ',u)
```

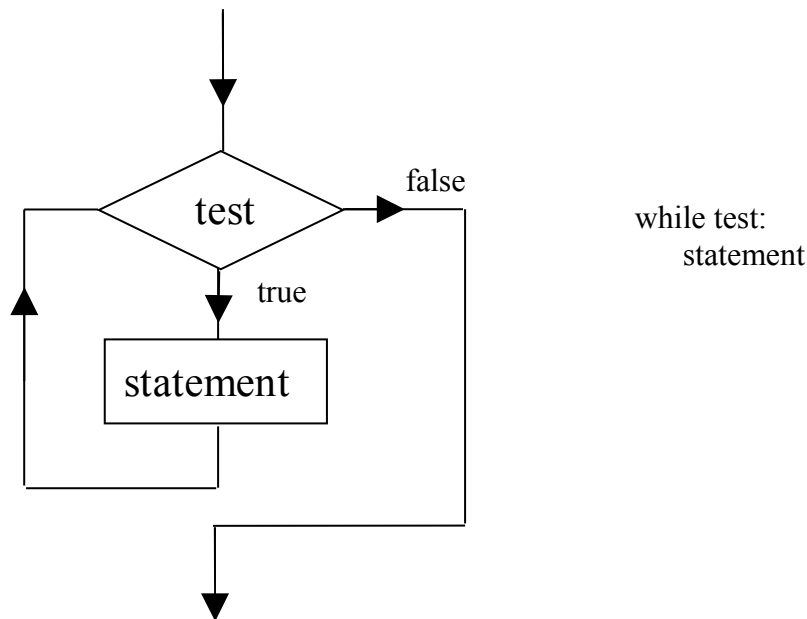
Uitvoer: u = [3, 3, 4]

Het while-statement

For-statements kunnen alleen gebruikt worden als bekend is welke lijst van elementen doorlopen moet worden.

Het while-statement wordt gebruikt als een dergelijke lijst ontbreekt.

Het stroomdiagram van het while-statement ziet er als volgt uit:



Voorbeeld:

Lees een rij getallen in. De rij wordt afgesloten met een 0. Druk de som van de ingelezen getallen af.

Oplossing:

```
som = 0
getal = int(input('voer een getal in: '))
while getal != 0:
    som += getal
    getal = int(input('voer een getal in: '))
print('de som van de ingelezen getallen is:', som)
```

11. Dictionaries.

Een dictionary is een verzameling (key,value)-paren.

De sleutels zijn uniek: ze mogen maar één keer voorkomen.

Voorbeeld:

key	value
Piet	7.4
Jan	3.4
Kees	5.5

```
cijferlijst = {'Piet':7.4, 'Jan':3.4, 'Kees':5.5}  
print(cijferlijst)
```

Uitvoer:

```
{'Jan': 3.4, 'Piet': 7.4, 'Kees': 5.5}
```

Bij dit voorbeeld is de volgorde bij de uitvoer anders dan bij de invoer.

Bij een dictionary doet de volgorde er niet toe. Voor een efficiënte opslag wordt een hashing-techniek toegepast. Deze plaatst de elementen in een willekeurige volgorde.

Het volgende is mogelijk bij een dictionary:

- het toevoegen van een (key,value) paar
- het wijzigen van een (key,value) paar
- het verwijderen van een (key,value) paar
- alle keys opvragen
- alle values opvragen
- een dictionary doorlopen
- dictionaries samenvoegen

Het toevoegen van een (key,value) paar.

Voorbeeld:

```
cijferlijst['Karel'] = 8.7 # Karel met cijfer 8.7 wordt toegevoegd  
print(cijferlijst)
```

Uitvoer:

```
{'Jan': 3.4, 'Karel': 8.7, 'Piet': 7.4, 'Kees': 5.5}
```

Het wijzigen van een (key,value) paar.

Voorbeeld:

```
cijferlijst['Jan'] = 6.1 # Jan had eerst 3.4  
print(cijferlijst)
```

Uitvoer:

```
{'Jan': 6.1, 'Karel': 8.7, 'Piet': 7.4, 'Kees': 5.5}
```

Het verwijderen van een (key,value) paar.

Voorbeeld:

```
del cijferlijst['Kees'] # Kees is gestopt met de cursus  
print(cijferlijst)
```

Uitvoer:

```
{'Jan': 6.1, 'Karel': 8.7, 'Piet': 7.4}
```

Alle keys opvragen.

Voorbeeld:

```
print('keys:', cijferlijst.keys()) # keys tonen
```

Uitvoer:

```
keys: dict_keys(['Jan', 'Karel', 'Piet'])
```

Alle values opvragen.

Voorbeeld:

```
print('values:', cijferlijst.values()) # values tonen
```

Uitvoer:

```
values: dict_values([6.1, 8.7, 7.4])
```

Een dictionary doorlopen.

Voorbeeld 1

```
for i in cijferlijst:      # alleen de keys worden getoond
    print(i, end = ' ')
```

Uitvoer:

```
Jan Karel Piet
```

Voorbeeld 2

```
for i in cijferlijst.items(): # keys en values worden getoond
    print(i)
```

Uitvoer:

```
('Jan', 6.1)
('Karel', 8.7)
('Piet', 7.4)
```

Voorbeeld 3

```
for i in cijferlijst:      # keys en values worden getoond
    print(i,':',cijferlijst[i])
```

Uitvoer:

```
Jan : 6.1
Karel : 8.7
Piet : 7.4
```

Dictionaries samenvoegen.

Voorbeeld

```
cijferlijst2 = {'Peter':9.5, 'Henk': 2.3, 'Rob':7.9}  
cijferlijst.update(cijferlijst2)  
print(cijferlijst)
```

Uitvoer:

```
{'Jan': 6.1, 'Karel': 8.7, 'Henk': 2.3, 'Piet': 7.4, 'Peter': 9.5, 'Rob': 7.9}
```

De 'symbol table', die wordt bijgehouden, is ook een dictionary.

Demonstratie:

```
print(vars()) # variabelen van het programma
```

Uitvoer: (nadat het netter is gemaakt)

```
{'__builtins__': <module 'builtins' (built-in)>,  
  '__package__': None,  
  'cijferlijst': {'Jan': 6.1, 'Karel': 8.7, 'Henk': 2.3, 'Piet': 7.4, 'Peter': 9.5,  
                  'Rob': 7.9},  
  'cijferlijst2': {'Rob': 7.9, 'Peter': 9.5, 'Henk': 2.3},  
  '__name__': '__main__',  
  '__doc__': None}
```

Recapitulatie

We hebben drie nu vier datastructuren behandeld: string, list, tuple, dictionary

datastructuur	beschrijving	omsluiting	wijzigbaar
string	een rij karakters	quotes	niet
list	een rij elementen	blokhaken	wel
tuple	een rij elementen	ronde haken	niet
dictionary	een set (key,value) paren	accolades	wel

12. Functies.



Een functie heeft de volgende kenmerken:

- het bevat een aantal opdrachten (statements)
- aan een functie kunnen parameters (argumenten) meegegeven worden.
- een functie geeft een waarde terug

Een functie heeft de volgende voordelen:

- door het programma op te bouwen uit functies, wordt het overzichtelijker
- een functie kun je opnieuw en met verschillende parameters gebruiken

We geven een voorbeeld van een functie:

```
def som(x,y):    # definitie van functie , x en y zijn formele parameters
    z = x+y      # z is een lokale variabele
    return z     # de waarde z wordt teruggegeven

print(som(2,3))    # de waarde z wordt teruggegeven

print(som('aap','je')) # parameters zijn van type 'string'
a = 3
b = 1e100          # a en b zijn globale variabelen
print(som(a,b))    # Deze worden toegekend aan de formele parameters.
```

Python heeft ingebouwde functies:

```
len([1,2,3,4]) = 4
min([1,2,3,4]) = 1
max([1,2,3,4]) = 4
sum([1,2,3,4]) = 10
```

De definitie van een functie.

Een voorbeeld van een functie-definitie is:

```
def som(x,y):
    z = x+y
    return z
```

De parameters (argumenten) die met de definitie van een functie worden meegegeven heten: formele parameters.

De aanroep van een functie

Een functie kan op verschillende manieren worden aangeroepen:

```
som(2,3)
som('aap','je')
som(a,b)
```

De parameters, die met een functie-aanroep worden meegegeven heten actuele parameters

Globale en lokale variabelen

Variabelen, die binnen een functie gedefinieerd zijn, heten lokale variabelen. Ze zijn niet bekend buiten de functie. Variabelen, die voor de functie-definitie zijn gedefinieerd, heten globale variabelen. Zij zijn wel bekend binnen de functie, tenzij binnen de functie een lokale variabele dezelfde naam heeft.

Voorbeeld:

```
a = 3      # a is globaal
b = 7      # b is globaal

def g(x,y):
    z = x+y # z is lokaal
    print('locals:', locals().keys())
    print('globals:', globals().keys())
    return z

print(g(a,b))
```

Uitvoer:

```
locals: dict_keys(['y', 'x', 'z'])
globals: dict_keys(['a', 'b', 'g', '__builtins__', '__file__', '__name__', '__doc__'])
10
```

Een lokale variabele en een globale variabele kunnen dezelfde naam hebben.

Voorbeeld:

```
a = 3      # a is globaal
b = 7      # b is globaal

def g2(x,y):
    a = 55
    print('a = ',a)
    z = x+y
    print('locals:', locals().keys())
    print('globals:', globals().keys())
    return z

print(g2(a,b))
print()
print('a = ',a)
```

Uitvoer:

```
a = 55      # lokale a heeft waarde 55
locals: dict_keys(['a', 'y', 'z', 'x'])
globals: dict_keys(['a', 'b', 'g2', 'g', '__builtins__', '__file__', '__name__',
'__doc__'])
10

a = 3      # globale a heeft waarde 3
```

Het is mogelijk een globale variabele binnen een functie te wijzigen. Dit moet dan expliciet aangegeven worden.

Voorbeeld:

```
a = 3      # a is globaal
b = 7      # b is globaal

def g3(x,y):
    global a
    a = 55
    print('a = ',a)
    z = x+y
    print('locals:', locals().keys())
    print('globals:', globals().keys())
    return z

print(g3(a,b))
print()
print('a = ',a)
print()
```

Uitvoer:

```
a = 55
locals: dict_keys(['y', 'x', 'z'])
globals: dict_keys(['a', 'b', 'g3', 'g2', 'g', '__builtins__', '__file__', '__name__',
'__doc__'])
10

a = 55
```

Advies: Wijzig een globale variabele niet.
De kans op fouten wordt groter.

De waarde die de functie teruggeeft.

Een functie geeft altijd een waarde terug.
Als geen return-opdracht wordt gegeven dan wordt de waarde 'None' teruggeven

Voorbeeld:

```
def doeNiets(i): pass # doe niets

x = doeNiets(0)
print(x)
```

Uitvoer:

```
None
```

Meestal wordt aan het einde van de functie-declaratie een return-opdracht gegeven. Het is ook mogelijk op andere plaatsen binnen de functie een return-opdracht te laten uitvoeren. Na het uitvoeren van de return-opdracht wordt de functie verlaten.

We geven een voorbeeld van een functie met meerdere return-opdrachten.

```
def myabs(i):  
    if i < 0:  
        return -i  
    return i  
  
print(myabs(-3))  
print(myabs(6))
```

Uitvoer:

```
3  
6
```

Vraag: hoe kan de functie worden aangepast, zodat maar één return-opdracht nodig is?

Parameters.

Het is mogelijk een functie geen parameters mee te geven. Ook kan een functie geen return-statement bevatten.

Voorbeeld:

```
def printTafelVanVijf():  
    for i in range(1,11):  
        print(i,'keer 5 is',5*i)  
  
printTafelVanVijf()
```

Uitvoer:

```
1 keer 5 is 5  
2 keer 5 is 10  
3 keer 5 is 15  
4 keer 5 is 20  
5 keer 5 is 25  
6 keer 5 is 30  
7 keer 5 is 35  
8 keer 5 is 40  
9 keer 5 is 45  
10 keer 5 is 50
```

Bij de aanroep van een functie worden de actuele parameters in de lokale 'symbol table' geplaatst. De 'symbol table' bevat niet de objecten zelf maar de verwijzingen (references) naar de objecten.

Als een actuele parameter een getal, een string of een tuple is, dan kan deze binnen de functie niet gewijzigd worden.

Als de parameter een lijst of dictionary is, dan kan een element van de parameter gewijzigd worden. De parameter zelf kan niet gewijzigd worden.

Voorbeeld 1:

```
def f(i):
    i *=2;
    return i

j = 4;
print(f(j))
print(j)    # j blijft ongewijzigd
i = 5
print(f(i))
print(i)    # i blijft ongewijzigd
```

Uitvoer:

```
8
4
10
5
```

Voorbeeld 2:

```
def g(a):
    a = [4,5,6]
    print(a)

a = [4,2,6,1]
print('vooraf:',a)
g(a)
print('achteraf:',a) # a blijft ongewijzigd.
print()
```

Uitvoer:

```
vooraf: [4, 2, 6, 1]
[4, 5, 6]
achteraf: [4, 2, 6, 1]
```

Voorbeeld 3:

```
def h(a):  
    a[1] = 25  
    print(a)  
  
a = [4,2,6,1]  
print('vooraf:',a)  
h(a)  
print('achteraf:',a)    # een element van a is gewijzigd  
print()
```

Uitvoer:

```
vooraf: [4, 2, 6, 1]  
[4, 25, 6, 1]  
achteraf: [4, 25, 6, 1]
```

Meerdere waarden teruggeven.

M.b.v. tupels is het mogelijk meerdere waarden terug te geven.

Demo:

```
def machten(i):  
    return i*i,i*i*i,i*i*i*i # of return (i*i,i*i*i,i*i*i*i)  
  
u,v,w = machten(5)  
print(u,v,w)  
print(machten(6)[1])  
t = machten(2)  
print( t[-1])  
u,v,w = t  
print( u,v,w)
```

Uitvoer:

```
25 125 625  
216  
16  
4 8 16
```

Parameters met default-waarden

Parameters kunnen een default waarde krijgen.

Voorbeeld:

```
def f(i,t=0,h=0):  
    return i + 10*t + 100*h  
  
print(f(3))          # t == 0, h == 0  
print(f(3,4))        # h == 0  
print(f(3,4,5))  
print()  
print(f(1,t=9))      # h = 0  
print(f(1,h=2))      # t = 0  
print(f(1,t=9,h=7))  
print(f(1,h=8,t=5)) # t en h mogen in een andere volgorde staan.
```

Uitvoer:

```
3  
43  
543  
  
91  
201  
791  
851
```

Dit heeft de volgende voordelen:

- minder tikwerk
- de volgorde van de parameters hoeft niet onthouden te worden.

Een parameter, die bij de aanroep expliciet genoemd wordt, heet een keyword-parameter.

13. Modulen.

De library van Python is een verzameling modulen.

Een module is een tekstfile met Python-code.

Door het gebruik van modulen wordt de Python-code verdeeld over meerdere tekstfiles .

Een overzicht van de beschikbare modules is te vinden in de Python Documentation, Library Reference

De volgende tabel beschrijft een aantal modulen.

module	functie
sys	verkrijgen systeem-informatie, systeem-instellingen
os	werken met filesysteem
threading	werken met threads
socket	maken van internet-verbindingen
telnetlib	toegang tot telnetservers
ftplib	toegang tot ftpservers
poplib	opvragen van email
BaseHttpServer	maken van webserver
anydbm	database voor strings
shelve	database voor objecten
sqlite3	sql-database

Ook door derden worden Python modulen gemaakt.

Een voorbeeld is de pywin32 module. Deze is beschikbaar op de site

<http://sourceforge.net/projects/pywin32/reviews/>

Hiermee kunnen specifieke windows-problemen worden opgelost.

We komen hier later op terug.

Een module importeren.

Als een module geïmporteerd wordt, gebeurt het volgende:

- de module wordt opgezocht
- de module wordt (zo nodig) gecompileerd (een '.pyc'-file wordt gemaakt)
- de module wordt gerund (dus: importeren houdt ook runnen in!)

Het programma, dat de module importeert, heeft vervolgens toegang tot de variabelen en functies binnen de module.

Voorbeeld 1.

De module 'sys' geeft informatie over het huidige systeem.

Programma:

```
import sys  
  
print(sys.platform)
```

Uitvoer:

```
win32 ( bij een windows-platform)
```

Voorbeeld 2.

Het kan ook als volgt:

```
from sys import *  
  
print(platform)
```

Voordeel: minder schrijfwerk

Nadeel: minder overzichtelijke code (als je meerdere modules gebruikt, weet je niet bij welke module een variabele hoort)

Voorbeeld 3

Het is ook mogelijk een module-naam te vervangen.

```
import sys as s  
  
print(s.platform)
```

Vooraf bij lange module-namen biedt dat voordeel.

Een module vinden.

De zoek-volgorde is als volgt:

- de directory van de huidige (top-level) file
- de directories, die staan in de systeemvariabele PYTHONPATH
- de directories waar de standaard library modules staan
- de directories, die worden genoemd in een '.pth'-file.
(Deze file moet staan in de top level van de installatie directory)

Python kent ingebouwde modules. Voorbeelden van ingebouwde modules zijn: 'sys' en 'time'
De niet-ingebouwde modules zijn te vinden in: '<python-map>\Lib'

Modules van derden worden geplaatst in '<python-map>\Lib\site-packages'

De inhoud van een module achterhalen.

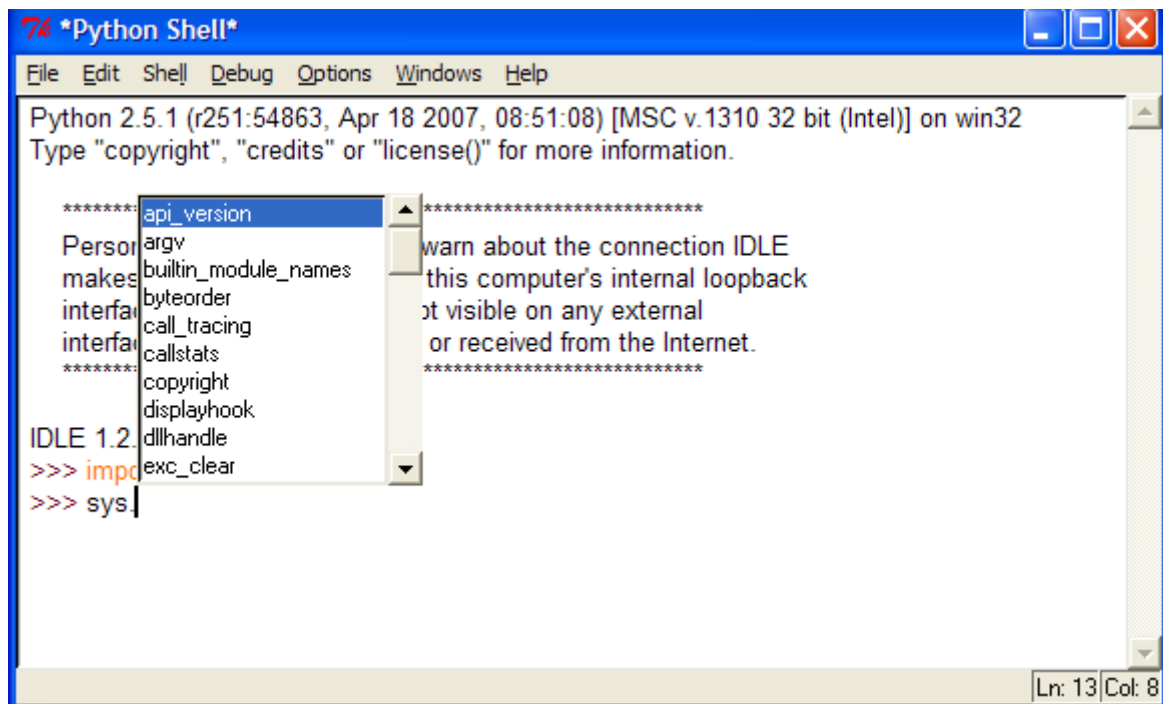
Hoe kun je nagaan wat een module voor mogelijkheden biedt?

Methode 1.

Tik in IDLE in:

```
import sys  
sys.
```

Er verschijnt dan een listbox met mogelijkheden.



Methode 2.

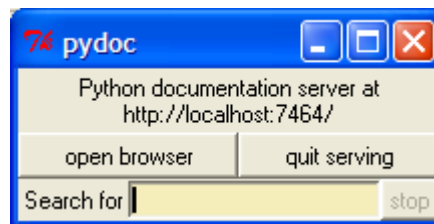
Geef de volgende opdrachten:

```
import sys
dir(sys)      # lijst met items
vars(sys)     # dictionary met items en beschrijvingen.
```

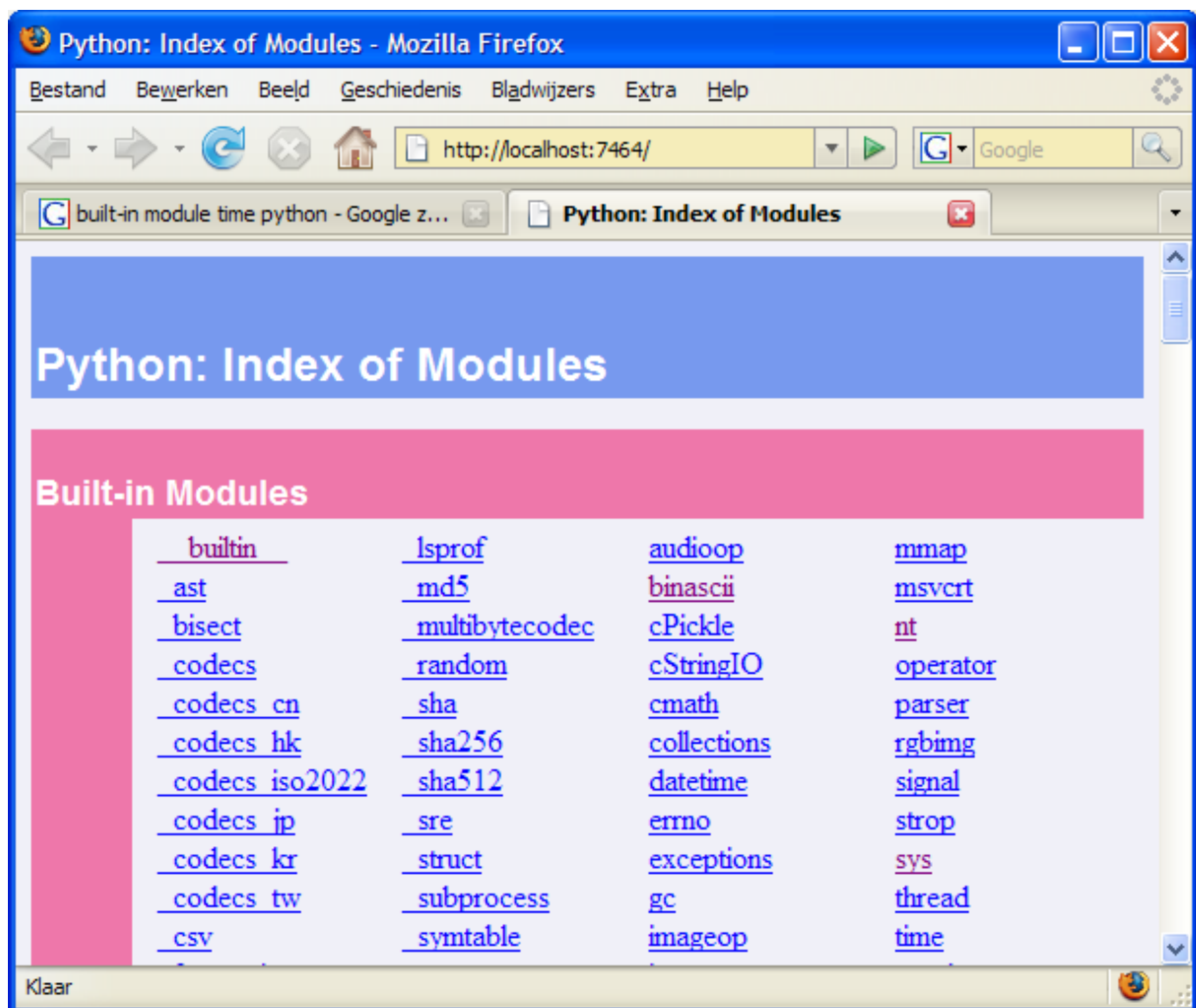
Methode 3.

Start 'Module docs'

De volgende window wordt getoond:



Klik op 'open browser'



Klik op 'sys'

Er wordt dan informatie over de module 'sys' getoond.

Win32 opdrachten.

Bij dit voorbeeld maken we gebruik van de module 'win32api'.

Voor het ontwikkelen van programma's heeft Microsoft een win32-api ter beschikking gesteld. Op de site [http://msdn2.microsoft.com/nl-nl/default\(en-us\).aspx](http://msdn2.microsoft.com/nl-nl/default(en-us).aspx) is hierover informatie te verkrijgen.

Om met Python hiervan gebruik te maken moet de module 'win32api' gedownload worden. Deze is op de site <http://sourceforge.net/projects/pywin32/files/pywin32/> beschikbaar.

Het volgende programma toont de beschikbare schijven op de computer en de hoeveelheid geheugen die op de c-schijf beschikbaar is.

```
import win32api

drives = win32api.GetLogicalDriveStrings()
print('drives: ', repr(drives))
drives = drives.split('\x00')
print('lijst drives: ', drives)
print()

t = win32api.GetDiskFreeSpaceEx('c:\\')
print(t)
print(t[2], 'byte beschikbaar op c:\\')
print(t[2]//2**30, 'gigabyte beschikbaar op c:\\')
```

Uitvoer:

```
drives: 'A:\\x00C:\\x00D:\\x00E:\\x00F:\\x00'
lijst drives: ['A:\\', 'C:\\', 'D:\\', 'E:\\', 'F:\\', '']

(9067790336, 20957974528, 9067790336)
9067790336 byte beschikbaar op c:\\
8 gigabyte beschikbaar op c:\\
```

Het schrijven van dit soort programma's vereist kennis van de win32-api.

14. Files

Lezen van en schrijven naar files

We willen de volgende regels wegschrijven naar de tekstfile 'vb1.txt' :

```
1 een
2 twee
3 drie
```

Dit gaat als volgt:

```
f = open('vb1.txt','w') # open de file 'vb1.txt' om in te schrijven.
f.write('1 een\n')
f.write('2 twee\n')
f.write('3 drie')
f.close()
```

'vb1.txt' staat in dezelfde directory als het python-programma.

Het lezen van een file gaat als volgt:

```
f = open('vb1.txt','r') # open een file om uit te lezen
for line in f:
    print(line, end = '')
```

Het is mogelijk een lijst met regels weg te schrijven:

```
f = open('vb2.txt','w')
f.writelines(['1 een\n','2 twee\n','3 drie'])
f.close()
```

De regels kunnen ook tegelijk ingelezen worden.

De regels worden in een string-lijst geplaatst.

```
f = open('vb2.txt','r')
a = f.readlines()
f.close()
for s in a:
    print(s, end = '')
```

Een file kan per regel ingelezen worden.

Voorbeeld:

```
f = open('vb2.txt','r')
s = f.readline()
while s:          # bij het einde van de file is s == "" (de lege string)
    print(s, end = "")
    s = f.readline()
f.close()
```

Verkorte schrijfwijze:

```
f = open('vb2.txt','r')
for line in f:
    print(line, end = "")
f.close()
```

Een file kan in één keer ingelezen worden:

Voorbeeld:

```
f = open('vb1.txt','r')
s = f.read()
f.close()
print('repr(s):',repr(s))
print('s:',s,sep = '\n')
```

Uitvoer:

```
repr(s): '1 een\n2 twee\n3 drie'
s:
1 een
2 twee
3 drie
```

Een file kan karakter voor karakter gelezen worden.

Voorbeeld:

```
f = open('vb1.txt','r')
s = f.read(1)
while s:
    print(s, end = '')
    s = f.read(1)
f.close()
print()
print()

f = open('vb1.txt','r')
s = f.read(1)
while s:
    print(repr(s), end = ',')
    s = f.read(1)
f.close()
```

Uitvoer:

```
1 een
2 twee
3 drie
```

```
'1',' ','e','e','n','\n','2',' ','t','w','e','e','\n','3',' ','d','r','i','e',
```

Tekstfiles en binaire files.

Voor tekstfiles in Windows geldt het volgende:

- bij het schrijven wordt '\n' omgezet in '\r\n'
- bij het lezen wordt '\r\n' omgezet in '\n'.

Als 'aaa\nbbb\nccc' wordt weggeschreven naar een tekstfile, dan worden 11+2 karakters in de file geplaatst.

Als we 'aaa\nbbb\nccc' letterlijk willen wegschrijven naar een file, dat kan dit met de opdracht:

```
f.write('aaa\nbbb\nccc','wb')           # wb : write binary
```

Voorbeeld:

```
f = open('vb3.txt','w')  
s1 = 'aaa\nbbb\nccc'  
print('s1:',repr(s1))  
print('len(s1):',len(s1))  
f.write(s1)  
f.close();  
  
g1 = open('vb3.txt','r')  
s2= g1.read()  
print('s2:', repr(s2))  
print('len(s2):',len(s2))  
g1.close();  
  
g2 = open('vb3.txt','rb')  
s3= g2.read()  
print('s3:', repr(s3))  
print('len(s3):',len(s3))  
g2.close()
```

Uitvoer:

```
s1: 'aaa\nbbb\nccc'  
len(s1): 11  
s2: 'aaa\nbbb\nccc'  
len(s2): 11  
s3: b'aaa\r\nbbb\r\nccc'  
len(s3): 13
```


Files kopiëren.

We laten drie methoden zien:

Methode 1: karakter na karakter kopiëren

```
def copyfile1(src,dest):  
    f = open(src,'rb')  
    g = open(dest,'wb')  
    s = f.read(1)  
    while s:  
        g.write(s)  
        s = f.read(1)  
    f.close()  
    g.close()  
  
copyfile1('vb','copy1')
```

Nadeel: het kopiëren gaat erg traag

Methode 2: de hele file in één string inlezen en vervolgens de string wegschrijven

```
def copyfile2(src,dest):  
    f = open(src,'rb')  
    g = open(dest,'wb')  
    s = f.read()  
    if s:  
        g.write(s)  
    f.close()  
    g.close()  
  
copyfile2('vb','copy2')
```

Nadeel: deze methode werkt, als de file niet te groot is.

Methode 3: het inlezen en wegschrijven in datablokken van 2048 bytes.

```
def copyfile3(src,dest):  
    f = open(src,'rb')  
    g = open(dest,'wb')  
    s = f.read(2048)  
    while s:  
        g.write(s)  
        s = f.read(2048)  
    f.close()  
    g.close()  
  
copyfile3('vb','copy3')
```

Voordeel: de methode is snel en het werkt ook voor grote files.

Het filesystem.

Bij de opdracht ' **f = open('vb.txt','w')** ' wordt een file geopend in de huidige werk-directory.

De huidige werk-directory kan als volgt worden opgevraagd:

```
import os  
print(os.getcwd())
```

of

```
import os  
print(os.path.abspath('.'))
```

We kunnen ook eerst een directory maken en daarin de file 'vb1.txt' plaatsen.

Dit gaat als volgt:

```
dirname = 'testdir'  
if not os.path.exists(dirname):  
    print( 'creating dir ' + dirname + ' ...')  
    os.mkdir(dirname)  
else:  
    print( 'dir ' + dirname + ' exist')  
f = open(dirname + '/vb1.txt','w')  
f.write('aaa\nbbb\nccc')  
f.close()
```

Uitvoer bij 1e keer:

```
creating dir testdir ...
```

Uitvoer bij 2e keer:

```
dir testdir exist
```

(Twee keer '**os.mkdir(dirname)**' aanroepen leidt tot een foutmelding.)

Als we de file in een subdirectory van een directory willen plaatsen dan moeten we dat als volgt doen:

```
dirname = 'testdir/subdir'  
.....  
    os.makedirs(dirname)    # de directories 'testdir' en 'subdir' worden gemaakt.  
.....
```

We geven een overzicht van opdrachten die betrekking hebben op files en directories.

	dos	python
listing directory	dir	os.listdir('.')
change directory	cd	os.chdir(...)
file copy	copy	shutil.copy(src, dest)
dir copy	xcopy	shutil.copytree(src, dest)
make dir	mkdir	os.mkdir(...)
delete file	del	os.remove(...)
delete dir	rmdir	os.rmdir(...) (werkt alleen bij een lege directory) shutil.rmtree(...) (werkt ook bij een niet-lege directory)
rename	rename	os.rename(src, dst)

15. Foutafhandeling.

De volgende code kan leiden tot een foutmelding:

```
f = open('bestaat.niet','r')
for line in f:
    print(line, end = ' ')
f.close()
```

De foutmelding is: **IOError: [Errno 2] No such file or directory: 'bestaat.niet'**
Het programma wordt bovendien beëindigd.

We kunnen op verschillende manieren ons beschermen tegen foutsituaties.

Methode 1: ga vooraf na of de file wel bestaat.

```
import os
if not os.path.exists('bestaat.niet'):
    print('de file \'bestaat.niet\' bestaat niet')
else:
    f = open('bestaat.niet','r')
    for line in f:
        print(line, end = "")
    f.close()
```

De methode heeft als nadeel, dat vooraf een test wordt gedaan. Dit is extra werk.
Bovendien zijn niet alle fouten afgevangen. De file kan bijvoorbeeld wel bestaan, maar de gebruiker heeft geen toegangsrechten. Het is bewerkelijk alle mogelijke fouten af te vangen.

Methode 2: gebruik een "try... catch ..." -constructie.

```
try:
    f = open('bestaat.niet','r')
    for line in f:
        print(line,end = "")
    f.close()
except:
    print('de file \'bestaat.niet\' kan niet geopend worden')
```

Deze methode is gebaseerd op het EAFP-principe: Easier to Ask Forgiveness than Permission

Het is nu echter niet duidelijk waarom de file niet geopend kan worden.

We kunnen de "try... catch ..." -constructie als volgt verfijnen:

```
import sys
try:
    f = open('bestaat.niet','r')
    for line in f:
        print(line,i)
    i = 1/0 # delen door nul gaat fout
except IOError as error_info:
    print(error_info)
except: # alle andere fouten
    print(sys.exc_info())
finally:
    print('finally')
    try:
        f.close() # zorg er altijd voor dat de file afgesloten wordt
    except: pass
```

Uitvoer:

```
[Errno 2] No such file or directory: 'bestaat.niet'
finally
```

16. Praktische toepassingen.

In dit hoofdstuk behandelen we enkele toepassingen.

- 1. achterhalen van ip-adressen.
- 2. een webserver bouwen.
- 3. een programma starten en stoppen
- 4. dos-opdrachten op een remote computer uitvoeren.
- 5. random getallen genereren.
- 6. het verzenden van email
- 7. files verzenden naar en opslaan van een ftp-server
- 8. een interactieve calculator
- 9. python-objecten opslaan

In dit hoofdstuk laten we zien wat met Python allemaal mogelijk is.

Overigens:

Python wordt bij Linux op de achtergrond veel gebruikt. Het is een standaard onderdeel van Linux. Verder is Dropbox gebaseerd op Python. (zie <http://blip.tv/pycon-us-videos-2009-2010-2011/pycon-2011-how-dropbox-did-it-and-how-python-helped-4896698>)

Toepassing 1: achterhalen van ip-adressen.

(info: <http://docs.python.org/py3k/library/socket.html>)

Wat is het ip-adres van een gegeven site?

In windows kun hiervoor de volgende opdrachten geven:

- ping www.hu.nl
- tracert www.hu.nl

In Python gaat dit als volgt:

```
import socket # importeer module 'socket'  
s = socket.gethostbyname("www.hu.nl")  
print(s)
```

Uitvoer:

213.154.250.220

Het is mogelijk meer informatie te achterhalen:

```
t = socket.gethostbyname_ex("www.google.com") # ex : extended
print(t)
```

Uitvoer:

```
('www.l.google.com', ['www.google.com'], ['173.194.65.147', '173.194.65.99',
'173.194.65.105', '173.194.65.106', '173.194.65.103', '173.194.65.104'])
```

Je ziet dat www.google.com een alias is en dat aan deze site meerdere ip-adressen gekoppeld zijn.

Het achterhalen van je eigen ip-adres is iets lastiger.

Waarom werkt het raadplegen van <http://www.mijnip.nl/> thuis meestal niet?

```
import socket
s = socket.gethostname()
print(s)
t = socket.gethostbyname_ex(s)
print(t)
```

Uitvoer:

```
J-PC
('J-PC', [], ['192.168.2.3'])
```

Wanneer heeft je eigen computer meerdere ip-adressen?

Complexer wordt het bij ipv6. We gaan hier niet verder op in.

Toepassing 2: een webserver bouwen.

(info: <http://docs.python.org/py3k/library/http.server.html>)

Tegenwoordig is ieder netwerk-apparaat voorzien van een webserver. Met de webserver kunnen loggegevens bekeken worden en kan het apparaat geconfigureerd worden.

Dergelijke webserver zijn klein.

Een dergelijke webserver kan ook in Python gemaakt worden.

```
import http.server

server_address = ('', 80) # poortnummer 80
HandlerClass = http.server.SimpleHTTPRequestHandler
HandlerClass.protocol_version = "HTTP/1.0"

httpd = http.server.HTTPServer(server_address, HandlerClass)

print("Serving HTTP on", httpd.server_name, "port", httpd.server_port, "...")

httpd.serve_forever() # handel onbeperkt lang cliënt-aanvragen af.
```

De webserver kan getest worden door in een webbrowser “<http://localhost>” in te tikken. Als de root-directory (de directory, waarin de webserver is opgestart) de file 'index.html' bevat, dan wordt deze getoond. Als 'index.html' niet bestaat dan wordt de directory-inhoud getoond

Vervang je de regel

```
server_address = ('', 80)
```

door

```
server_address = ('localhost', 80)
```

dan is de webserver niet te bereiken vanaf een andere computer.

Vervang je de regel

```
server_address = ('', 80)
```

door

```
server_address = ('145.89.1.1', 80)
```

dan is de webserver niet meer via <http://localhost> of <http://127.0.0.1> te bereiken.

Toepassing 3: een programma starten en stoppen

(info: <http://docs.python.org/py3k/library/subprocess.html#subprocess.Popen>)

Een toepassing kan als volgt worden opgestart:

```
import subprocess
p = subprocess.Popen("notepad")
```

De toepassing kan ook worden afgesloten:

```
p.terminate()
```

De volgende code start "notepad" op en sluit het na 5 seconden weer af:

```
import subprocess

p = subprocess.Popen("notepad")

import time

time.sleep(5)

p.terminate()
```

Het opstarten van "Internet Explorer" gaat moeilijker.
Op mijn laptop heb ik het als volgt opgelost:

```
import subprocess

p = subprocess.Popen("C:/Program Files (x86)/Internet Explorer/iexplore.exe")
```

Met dos-commando's gaat het als volgt:

```
import subprocess

def runcmd(cmd):
    p = subprocess.Popen(cmd,
                          shell = True,
                          stdin = subprocess.PIPE,
                          stdout= subprocess.PIPE,
                          stderr= subprocess.STDOUT)

    return p

p = runcmd("dir")
s1 = p.stdout.read() # lees in wat proces p uitvoert
print(s1)

# zet bytes om in string
s2 = s1.decode()
print(s2)
```

Het is ook mogelijk een ander python-programma op te starten en beëindigen.

```
p = subprocess.Popen("pythonw webserver.py") # pythonw opent geen dosbox
print("webserver gestart")

import time
time.sleep(15)

p.terminate()
print("webserver gestopt")
```

Toepassing 4: dos-opdrachten op een remote computer uitvoeren.

Let op: een moeilijk maar interessant onderwerp.

Hiervoor hebben we twee programma's nodig.

- SimpleTelnetServer.py
- SimpleTelnetClient.py

SimpleTelnetServer.py:

```
import socketserver
import subprocess

def runcmd(cmd):
    p = subprocess.Popen(cmd,
                          shell = True,
                          stdin = subprocess.PIPE,
                          stdout= subprocess.PIPE,
                          stderr= subprocess.STDOUT)
    return p

class TelnetRequestHandler(socketserver.StreamRequestHandler):
    def handleCommand(self,s):
        p = runcmd(s)
        s1 = p.stdout.read()
        s2 = s1.decode()
        self.wfile.write((str(len(s2)+2) + '\r\n').encode()) # stuur eerst aantal bytes op
        self.wfile.write((s2 + '\r\n').encode())

    def handle(self):
        print('telnet request from', self.client_address)
        self.endOfSession = False

        while True:
            try:
                bar = self.rfile.readline() # lees opdracht van client
                s = bar.decode()
                if not s:
                    print('client closed')
                    break
                if s.endswith('\n'):
                    s = s[:-1]
                if s.endswith('\r'):
                    s = s[:-1]
            except:
                print('client connection aborted')
                break
            print('<<',s)
            self.handleCommand(s); # verwerk opdracht en stuur antwoord
            print('connection closed:', self.client_address)
            self.request.close() # close socket
```

```
def run(port):
    server = socketserver.ThreadingTCPServer(("",port), TelnetRequestHandler)
    print('TelnetServer running at port', port, '...')
    server.serve_forever()
```

```
telnet_port = 23
run(telnet_port)
```

SimpleTelnetClient.py:

```
import socket

def run(host,port):
    addr = (host,port)
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect(addr)
    rf = s.makefile('rb',-1)
    wf = s.makefile('wb',0)

    wf.write(('ipconfig\r\n').encode()) # stuur opdracht 'ipconfig' op
    b2 = rf.read(1) # ga na hoeveel bytes gelezen moeten worden
    s2 = b"
    while b2 != b'\n':
        s2 = s2 + b2
        b2 = rf.read(1)
    n = int((s2+b2).decode()) # lees antwoord
    s3 = rf.read(n).decode()
    print(s3)

    wf.write(('dir\r\n').encode()) # stuur opdracht 'dir' op
    b2 = rf.read(1)
    s2 = b"
    while b2 != b'\n':
        s2 = s2 + b2
        b2 = rf.read(1)
    n = int((s2+b2).decode())
    s3 = rf.read(n).decode()
    print(s3)

    s.close()
    return

host = 'localhost'
telnet_port = 23
run(host,telnet_port)

input("Press Enter to finish")
```

Toepassing 5: random getallen genereren.

Het volgende programma simuleert 50 worpen van een dobbelsteen.

```
import random

for i in range(50):
    j = random.randrange(1,7)
    print(j,end = ' ')
print()
```

Uitvoer:

```
2 6 3 2 4 2 6 1 2 4 6 3 4 6 3 4 4 2 1 5 4 5 2 5 1 2 6 3 6 1 6 6 3 5 1 3 2 4 1 3 4 1 5 2 3 3 5 2 3 3
```

Met de module 'random' kunnen ook willekeurige wachtwoorden gegenereerd worden.
Dit gaat als volgt:

```
s = \
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

def gen_random_wachtwoord():
    random_wachtwoord = ""
    for i in range(8):
        random_wachtwoord += s[random.randrange(len(s))]
    return random_wachtwoord

print(gen_random_wachtwoord())
```

Toepassing 6: het verzenden van email

(zie <http://docs.python.org/py3k/library/email-examples.html>)

Bij het versturen van mail moeten een aantal gegevens worden ingevuld:

- smtp-server (wordt beschikbaar gesteld door de isp, bedrijf of instelling.
Bijv. smtp.wxs.nl)
- mail-adres afzender
- mail-adres ontvanger
- onderwerp mail
- inhoud mail

```
import smtplib
import email.mime.text

host = input('smtp-server? ')
me = input('mail-adres afzender? ')
you = input('ontvanger? ')

msg = email.mime.text.MIMEText('mail regel1\nmail regel2\nmail regel3')
msg['Subject'] = 'spam: door python gegenereerde mail!'
msg['From'] = me
msg['To'] = you

s = msg.as_string()
print('====\nmsg:',s,'\n====')

server = smtplib.SMTP(host,25)
server.set_debuglevel(1) # geeft extra informatie
server.sendmail(me, [you], s)
server.quit()
```

Het volgende wordt verstuurd:

```
msg: Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: spam: door python gegenereerde mail!
From: <<me>>
To: <<you>>

mail regel1
mail regel2
mail regel3
```

Bij het versturen via een gmail-account moet ook ingelogd worden. Bovendien is het een ssl-verbinding.

```
import smtplib
import email.mime.text
import getpass

gmail_me = input('gmail account? ')
password = getpass.getpass('password? ') # tekst niet zichtbaar op scherm
you = input('ontvanger? ')

msg = email.mime.text.MIMEText('gmail regel1\ngmail regel2\ngmail regel3')
msg['Subject'] = 'spam via gmail!'
msg['From'] = gmail_me
msg['To'] = you

server = smtplib.SMTP_SSL('smtp.gmail.com',465)
server.login(gmail_me,password)
server.set_debuglevel(1)
server.sendmail(gmail_me, [you], msg.as_string())
server.quit()
```

We gaan op aantal vragen niet verder in:

- hoe wordt email gestuurd naar meerdere ontvangers?
- hoe wordt een bijlage meegestuurd?
- hoe wordt ontvangen mail opgehaald?

Toepassing 7: files verzenden naar en opslaan van een ftp-server
(zie <http://docs.python.org/py3k/library/ftplib.html>)

Veel netwerk-apparaten zijn voorzien van een ftp-server.

In het volgende voorbeeld wordt de file toMap/toServer.txt opgeslagen en de file fromMap/fromServer.txt opgehaald.

```
import ftplib
import getpass

site = input('site? ')
user = input('user? ')
passwd = getpass.getpass('passwd? ')

connection = ftplib.FTP(site)
connection.login(user,passwd)
connection.set_debuglevel(1) # geef debug-informatie

lines=[]
connection.dir(lines.append)
for line in lines:
    print(line)
print

#get file
filename = 'fromServer.txt'
localfile =open(filename,'wb')
connection.retrbinary('RETR fromMap/'+filename,localfile.write,1024)

#put file
filename = 'toServer.txt'
localfile =open(filename,'rb')
connection.mkd('toMap')
connection.storbinary('STOR toMap/'+filename,localfile,1024)

#sluit af
connection.quit()
```


Toepassing 8: een interactieve calculator

De functie 'eval' berekent de opgegeven expressie van de gebruiker.

Een interactieve calculator kan als volgt gerealiseerd worden:

```
while True:
    s = input('expressie')
    try:
        print(eval(s))
    except:
        print('syntax error')
```

Een voorbeeld:

```
expressie: 2+3*4
14
expressie: 2**100
1267650600228229401496703205376
expressie: 2++4
6
expressie: 2+++4
6
expressie: 2+-+4
-2
expressie: 2---4
-2
expressie: 2--
syntax error
```

Toepassing 9: python-objecten opslaan
(zie <http://docs.python.org/py3k/library/shelve.html>)

Het volgende programma slaat een Python-object op en haalt het object weer op.

```
lijst = {'Jan': 6.1, 'Karel': 8.7, 'Henk': 2.3, 'Piet': 7.4, 'Peter': 9.5, 'Rob': 7.9}

import shelve

d = shelve.open('vb.shelve')

d['lijst'] = lijst

d.close()

d = shelve.open('vb.shelve')
print(d['lijst'])
d.close()
```

Er worden drie files gemaakt:

- vb.shelve.dat (binair)
- vb.shelve.dir
- vb.shelve.bak

Een uitgebreider voorbeeld staat op bovengenoemde site:

```
import shelve
```

```
d = shelve.open(filename) # open -- file may get suffix added by low-level  
                           # library
```

```
d[key] = data             # store data at key (overwrites old data if  
                           # using an existing key)
```

```
data = d[key]            # retrieve a COPY of data at key (raise KeyError if no  
                           # such key)
```

```
del d[key]               # delete data stored at key (raises KeyError  
                           # if no such key)
```

```
flag = key in d          # true if the key exists
```

```
klist = list(d.keys())   # a list of all existing keys (slow!)
```

```
# as d was opened WITHOUT writeback=True, beware:
```

```
d['xx'] = [0, 1, 2]      # this works as expected, but...
```

```
d['xx'].append(3)       # *this doesn't!* -- d['xx'] is STILL [0, 1, 2]!
```

```
# having opened d without writeback=True, you need to code carefully:
```

```
temp = d['xx']          # extracts the copy
```

```
temp.append(5)         # mutates the copy
```

```
d['xx'] = temp         # stores the copy right back, to persist it
```

```
# or, d=shelve.open(filename,writeback=True) would let you just code
```

```
# d['xx'].append(5) and have it work as expected, BUT it would also
```

```
# consume more memory and make the d.close() operation slower.
```

```
d.close()              # close it
```