

# Programmeermethoden NA

## Week 2: Types

Kristian Rietveld

<http://liacs.leidenuniv.nl/~rietveldkfd/courses/prna2016/>



Universiteit Leiden  
The Netherlands

# Eerste programmeeropgave

- De eerste programmeeropgave: “IQ”.

<http://liacs.leidenuniv.nl/~rietveldkfd/courses/prna2016/opdr1.html>

- Te gebruiken Python versie: 2.7.

- Inleveren:

- Digitaal (Python-file) per e-mail naar  
`prna2016@handin.liacs.nl`

Vermeld studentnummers, namen in e-mail.

- Geprinte versie van verslag: doos in kamer 156,  
Snelliusgebouw.

- Vragen?

# Programmeeromgeving

- Heeft iedereen een werkend ULCN account?
  - (Vergeet de e-mail forward niet).
- Heeft iedereen een werkende programmeeromgeving?
- Voorbeelden:
  - Linux (Live USB, dual boot, virtual machine, Rasp. Pi, ...).
    - (Of inloggen via ssh).
  - Mac OS (alles staat al klaar, alleen editor nodig).
  - Windows (bijv. Enthought Canopy Express).

# Stoomcursus UNIX (vervolg)

- De belangrijkste commando's zijn:

<code>ls</code>	overzicht files in directory (=map)
<code>more</code>	file op het beeldscherm
<code>less</code>	file op het beeldscherm (handiger scrollen)
<code>rm</code>	file verwijderen
<code>cp</code>	file kopiëren
<code>mv</code>	file verplaatsen / hernoemen
<code>cd</code>	van directory veranderen (change directory)
<code>mkdir</code>	maak een nieuwe directory
<code>chmod</code>	rechten bij files regelen
<code>man</code>	hulpprogramma (manual pages)

- Enkele voorbeelden:

```
ls -lrt ; cp een twee ; cd Abc ; chmod 644 een ; man ls
```

# UNIX - geavanceerd

- Vanuit het terminalvenster kun je de web-browser Firefox opstarten: “`firefox &`”.
  - De ampersand (&) zorgt er voor dat Firefox “op de achtergrond” wordt opgestart. Je kunt dan in het terminalvenster verder werken.
- Een actief programma in UNIX heet een “*proces*”.
  - CTRL-C stopt een proces, CTRL-Z pauzeert een proces.
  - Verdere controle: `ps`, `top`, `kill`
  - Voorbeeld met *pipelining*: `ps ux | grep username`
- Gebruik “history” in het terminalvenster: pijl omhoog/omlaag en “tab” completion.
- Zie verder dictaat “Computers en programmeren”, hoofdstuk 2.

# Python programma

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
```

# Onderdelen

- Een Python-programma bestaat uit verschillende onderdelen:
  - speciale symbolen: +, %, >=, = (toekenning), == (is gelijk)
  - woordsymbolen: if, else, print
  - identificers: straal
  - getallen: 42, 0
  - strings: "Einde van dit programma."
  - whitespace: spaties, tabs, lege regels.
- Vaste waarden in de source code, zoals 42 en de string "Ho i", worden ook wel *literals* genoemd.

# Keywords

- De woordsymbolen die we zagen heten ook wel gereserveerde woorden of keywords.
- Python heeft de volgende keywords:

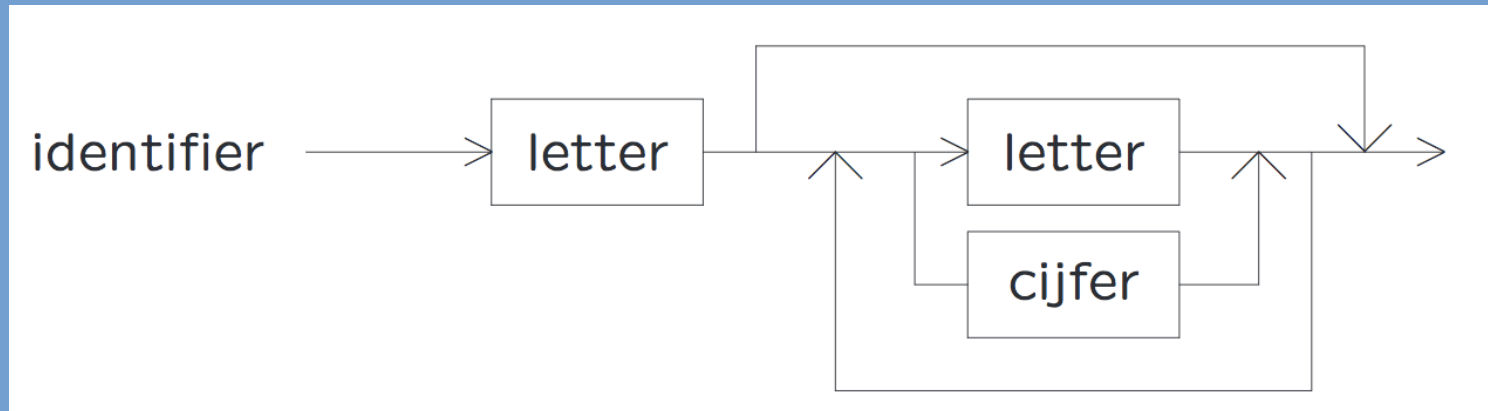
<b>and</b>	<b>as</b>	<b>assert</b>	<b>break</b>	<b>class</b>
<b>continue</b>	<b>def</b>	<b>del</b>	<b>elif</b>	<b>else</b>
<b>except</b>	<b>exec</b>	<b>finally</b>	<b>for</b>	<b>from</b>
<b>global</b>	<b>if</b>	<b>import</b>	<b>in</b>	<b>is</b>
<b>lambda</b>	<b>not</b>	<b>or</b>	<b>pass</b>	<b>print</b>
<b>raise</b>	<b>return</b>	<b>try</b>	<b>while</b>	<b>with</b>
<b>yield</b>				

- Keywords mogen niet worden gebruikt als naam voor een variabele (als identifier).



# Syntaxdiagrammen

- Programmeertalen zijn heel precies grammaticaal vastgelegd. Dit is nodig om de taal te kunnen “parsen”.
  - Ieder programma moet aan de regels van deze grammatica voldoen.
- Deze regels kunnen bijvoorbeeld worden vastgelegd in syntaxdiagrammen.
  - Voorbeeld: een *identifiser* is gedefinieerd als *een letter gevolgd door nul of meer cijfers*.



- Zie verder: Noam Chomsky, BNF (Backus Naur Form), context-vrije/formele talen.

# Commentaar

- Een goed programma is uitgebreid voorzien van commentaar.

```
#  
# Dit is ons Python programma.  
# En de interpreter slaat het allemaal over, zonder te lezen.  
#
```

- En ook als een variabele-declaratie uitleg behoeft:

```
stp = 4 # 4 EC studiepunten voor PRNA  
        # als je opgaven EN tentamen haalt.
```

- Commentaar wordt ook gebruikt om code tijdelijk “weg te commentariëren”.

# Commentaar (2)

- Er is een verschil tussen `#` (voor programmeurs, inclusief jezelf) en `print` (gebruikers, inclusief jezelf).
- Goed:

```
print "Geef eerste voorletter .. ",  
een = raw_input()  
print "Geef tweede voorletter .. ",  
twee = raw_input()
```

- Voor de gebruiker niet helemaal duidelijk:

```
print "Geef eerste voorletter en wanneer nodig ook  
de tweede .. ",  
letters = raw_input()
```

- Overbodig commentaar leidt alleen maar af:

```
print "Geef eerste voorletter .. ",  
voorletter = raw_input() # lees de voorletter in
```

# Globale structuur

- De globale structuur van een Python-programma is:

```
#!/usr/bin/env python  
#  
# Commentaar: wie, wat, waar(om), wanneer  
import ...  
import ..  
  
# Constante variabelen die we in het programma gaan gebruiken  
peildatum = ...  
  
... # allerlei functies  
  
# start van het programma  
print "hello world"
```

- Wanneer we uitgebreid met functies gaan werken, bekijken we een nog iets nettere versie van de globale structuur.

# Variabelen in Python

- In Python maken we variabelen met het toekenningsstatement (assignment).
- Om een variabele te kunnen gebruiken moet deze bestaan en dus zijn gemaakt met een toekenningsstatement.
- Toekenning op een al bestaande naam overschrijft de oude waarde.

```
a = 4
b = "testje!"
a = "overschrijven" # oude waarde van variabele
a wordt overschreven
d = a + g
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'g' is not defined
```

# Variabelen in Python (2)

- Alle variabelen worden opgeslagen in het computergeheugen in een binaire vorm.
  - Verschillende 'soorten' data worden op een verschillende manier opgeslagen.
  - In feite werken we met verschillende *typen* data en met elke variabele in Python is een *type* geassocieerd.
- Python kent een aantal basistypes die we nu zullen bekijken.
- In een later college leren we ook zelf types maken.

# Getallen

- *int*: Integers, meestal 4 of 8 bytes groot.  
Bereik  $-2^{31}$  tot  $2^{31} - 1$  of  $-2^{63}$  tot  $2^{63} - 1$ .  
Ongeveer -2 tot 2 miljard of -9 tot 9 triljoen.
- *long*: Long integers, zo groot als maar past in het geheugen van de computer. Zeer grote getallen mogelijk!
- *bool*: True of False.
- *float*: Benaderingen (!) van reële getallen. In Python altijd double precision (8 bytes).
- *complex*: Complexe getallen. Ingebouwd!

# Floating-point getallen

- Een variabele van type *float* bevat een benadering van reëel getal.
  - (Stiekem zijn het rationale getallen uit de verzameling  $\mathbb{Q}$ ).
  - Afrondingsfouten komen dus voor en test floats niet zomaar voor gelijkheid maar hanteer een foutmarge.
  - En irrationale getallen als  $\pi$ ,  $\sqrt{2}$  worden nooit exact gepresenteerd.
- Met `round()` kun je netjes afronden:
  - `round(234.2341234, 3)` geeft `234.234`
- In de module `math` vind je standaardfuncties als `sin`, `cos`, `floor`, `ceil`, ...

```
import math
print math.floor(6.8)
```



# Complexe getallen

- Python heeft een type voor complexe getallen ingebouwd!

```
>>> z = 6+9j # "j" is de imaginaire eenheid, in de  
wiskunde i geheten
```

```
>>> type(z)  
<type 'complex'>
```

```
>>> z.real  
6.0
```

```
>>> z.imag  
9.0
```

# Operaties op getallen

```
a, b = 3, -5
getal = a + b      # getal wordt -2
a = a + 7         # a wordt 10
b += 1           # Python kent geen ++ operator
a -= 1
getal += a
a = 19 / 7        # Integer deling: a wordt 2
b = 19 % 7        # Rest bij deling (mod): b wordt 5
# Optelling complexe getallen: resultaat (10+11j)
q = (6+9j) + (4+2j)
q = (6+9j) * 2    # Resultaat: (12+18j)
```

# Integer vs. floating point

```
i = 9 / 5          # Geeft 1, i wordt een integer
x = 9 / 5.0        # Geeft 1.8, x wordt een float
x = float(9 / 5)   # Geeft 1.0, 9 / 5 geeft een
                  # integer resultaat dat wordt
                  # omgezet naar een float.
x = 9 / float(5)   # Geeft 1.8, x wordt een float
x = 9.0 // 5.0     # Geeft 1.0, // is delen met
                  # integer-afroning
m = 3 ** 4         # Python heeft een ingebouwde operator
                  # voor machtsverheffing, het resultaat is 81
```

# Booleaanse variabelen

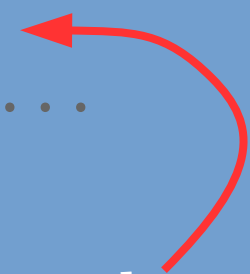
- Boolese/Booleaanse variabelen hebben het type *bool* en kunnen de waarde `True` of `False` aannemen.
  - Naar George Boole (1815-1864), uitvinder Booleaanse algebra.
- Waarheidstabel:

<code>p</code>	<code>q</code>	<code>not p</code>	<code>p and q</code>	<code>p or q</code>	<code>p == q</code>
False	False	True	False	False	True
False	True	True	False	True	False
True	False	False	False	True	False
True	True	False	True	True	True

# Boolean expressies

- Boolese algebra wordt gebruikt om Boolese expressies te schrijven.
- Als je zowel `and` als `or` in een expressie gebruikt: gebruik haakjes om verwarring te voorkomen!

```
if y >= 3 and y <= 7: ...
if not (y < 3 or y > 7): ...
if y >= 3 and (x == 4 or x == 5): ...
if s == "hello": ...
if y >= 3 and (x == 4 or x == 5) and \
    z == 12 and (q >= 10 or q <= -10): ...
```



- Een if-statement mag **niet** zomaar op een volgende regel verder gaan. Als je dit doet moet je de voorgaande regel afsluiten met een backslash.

# Boolean expressies (2)

- Als in de volgende test  $x$  gelijk is aan 0:

$(x \neq 0 \text{ and } y / x == 7)$

dan wordt de tweede test niet eens meer gedaan. We noemen dit “short-circuiting” of “lazy evaluation”.

# Strings

- Een rijtje van karakters noemen we een “*string*” (of characters). Type: str.
- Strings laten zich gemakkelijk maken, de lengte bepalen en vergelijken:

```
>>> woord = "De."  
>>> len(woord)  
3  
>>> woord == "test"  
False  
>>> woord == "De."  
True
```

# ASCII

- Hoe worden die karakters in een computergeheugen opgeslagen?
  - Ieder karakter correspondeert met een getal tussen 0 en 255 (precies de range van een enkele byte).
  - Meest gebruikt: de ASCII tabel.

*American Standard Code for Information Interchange*

...	36	37	38	...	47	48	49	...	57	58	...
...	\$	%	&	...	/	0	1	...	9	:	...
65	66	...	90	91	...	97	98	...	122	123	...
A	B	...	Z	[	...	a	b	...	z	{	...

- Carriage Return ('\r', naar begin regel) heeft waarde 13, line feed ('\n', nieuwe regel) waarde 10. (Regelovergang UNIX: LF, Windows: CR, LF).
- Op een UNIX machine: “man ascii”



# Escaping

- We zagen net al een voorbeeld van “escaping”: `'\n'`.
- De backslash dient als escape-karakter, samen met het karakter dat nog volgt heeft dit een speciale betekenis.
  - `'\n'`: line feed, `'\r'` carriage return, `'\t'` tab.
- Wat nu als we letterlijk “`\n`” op het scherm willen zetten (en dus niet een line feed willen krijgen)?
  - Dan moet je het escape-karakter ook “escapen”:
  - Dus: `print "\\n"`

# Strings - Indexing en slicing

- Het is mogelijk om individuele karakters uit een string te lezen. Hiervoor moeten we de string “indexen”.
- We geven in zo'n geval een object van het type `str` en een integer index (positie van het vakje):
  - `woord[1]` - tweede (!) karakter in woord.
  - We beginnen met tellen bij 0!
  - Negatieve index: terugtellen vanaf einde string.
- Je mag ook een start- en eind- index geven, bijvoorbeeld om een substring uit te lezen. We noemen dit slicing.
  - `zin[3:14]`
  - De eind-index telt **niet** mee!

# Strings - Indexing en slicing

```
>>> s = "een lange test string"
>>> s[2]
'n'
>>> s[-4]
'r'
>>> s[3:8]
' lang'
>>> s[6:]
'nge test string'
```

# Strings - Andere operaties

- Is “aaa” een substring van s?
- Eindigt string f met “.txt”?
- Strings concateneren (aan elkaar plakken).

```
>>> s = "aaa bbb ccc eee fff ggg"
```

```
>>> "aaa" in s
```

```
True
```

```
>>> "zzz" in s
```

```
False
```

```
>>> f = "testbestand.txt"
```

```
>>> f.endswith(".txt")
```

```
True
```

```
>>> s + f
```

```
'aaa bbb ccc eee fff gggtestbestand.txt'
```

# Strings - Andere operaties

```
>>> s + 12
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
objects
>>> s + str(12)
"aaa bbb ccc eee fff ggg12"
```

```
>>> a = "aaa"
>>> b = "bbb"
>>> a * 3
'aaaaaaaaa'
>>> a * 3 + b * 3
'aaaaaaaaabbbbbbbb'
```

# Algemene opmerkingen

- Zodra je een string hebt gemaakt, kun je deze niet meer aanpassen.
  - Dus je kunt een individueel karakter mbv de index-notatie **niet** aanpassen.
  - Als je een aanpassing wilt maken, maak je simpelweg een nieuwe string.
- Wat een operator, zoals bijv. +, precies doet hangt af van het type van het object waarop deze wordt toegepast.
  - Bij integer: + telt op.
  - Bij strings: + concateneert.

# Functie "type()"

- Met de functie `type()` kunnen we voor een variabele opvragen van welk type deze is:

```
>>> a, b, c = 9, 3.14, "strrrr"
```

```
>>> type(a)
```

```
<type 'int'>
```

```
>>> type(b)
```

```
<type 'float'>
```

```
>>> type(c)
```

```
<type 'str'>
```

```
>>> a = "strrrr2" # oude waarde van a wordt overschreven
```

```
>>> type(a)
```

```
<type 'str'>
```

# Conversie van getallen

- Je kunt getallen omzetten naar de verschillende typen met behulp van conversiefuncties.
- `float( )` zet het argument om naar een floating-point getal. Accepteert ook strings: `float( "3.14" )`.
- Andere typeconversies: `int( )`, `complex( )`, `str( )`.
  - Voorbeeld: `int(3.14)` resulteert in de waarde 3.
- Een operatie op twee verschillende typen resulteert in een impliciete conversie: type coercion.
  - Bijv. `int + float` geeft een float.
  - `int + long` geeft een long.



# Expressies

- Met operatoren (+ - \* % / \*\* <<), variabelen en literals bouw je *expressies*.
  - Voorbeeld:  $(4 + 5) * 9 - 3$
- Er gelden prioriteiten, welke vastleggen op welke volgorde operatoren worden geëvalueerd.
  - Bij twijfel: zet haakjes!!
- Expressies worden altijd van links naar rechts geëvalueerd.
  - Uitzondering, toekenningen: dan eerst de rechterkant, gevolgd door de linkerkant.

# print statement

- `print` zet tekst op het scherm.
- Keyword `print`, gevolgd door een rijtje van expressies (gescheiden door komma's).
  - De expressies worden een-voor-een geëvalueerd, het resultaat wordt impliciet geconverteerd naar een string.
  - De resultaten worden achter elkaar op het scherm gezet, gescheiden door spaties.

```
>>> a = 110
>>> b = 12
>>> print "Test:", "a =", a, "b =", b, "en samen
maakt dat", a + b
Test: a = 110 b = 12 en samen maakt dat 122
```

# print statement (2)

- Nog twee handige trucjes:
  - Een 'kale' print, zonder iets erachter, geeft alleen een lege regel.
  - Als je grote blokken tekst op het scherm wilt zetten, gebruik dan driedubbele quotes:

```
print """Hallo, dit is een groot stuk tekst.
```

Bijvoorbeeld een infoblokje.

De interpreter leest door tot de afsluitende driedubbele quote."""

# Uitvoerformattering

- Met behulp van *uitvoerformattering* kunnen we (veel) meer controle uitoefenen over hoe variabelen op het scherm worden gezet.
- We schrijven hiervoor een *format string* waarin *format fields* zijn opgenomen.
  - Op de plek van elk format field, te herkennen aan accolades, wordt een variabele ingevuld. Het getal geeft aan welke variabele.
  - De in te vullen variabelen worden opgegeven als argumenten aan `.format()`.

```
print "{0} {1}".format(13, 42)
```

```
print "{een} {twee}".format(een=13, twee=42)
```

# Uitvoerformattering (2)

- Format fields kunnen in meer detail worden ingesteld door een dubbele punt toe te voegen en daarna verdere parameters.
  - De eerste parameter is een getal dat de breedte (in karakters) van het veld specificeert.
  - Het karakter aan het einde geeft het type veld aan: f (float), d (integer), s (string), e (wetenschappelijke notatie).
  - Voor floating-point getallen kan worden aangegeven hoeveel cijfers we achter de komma wensen.

```
a, b = 123, 3.1409134091023
```

```
print "{0:6d} {1:8.4f} {2:20s}".format(a, b, "test")
```

# Uitvoerformattering (3)

- We kunnen het veld ook links of rechts uitlijnen, of centreren:

```
print “|{0:<8d}|{0:>8d}|{0:^8d}|”.format(123)
```

- Het resultaat kun je ook opvangen in een string:

```
test = “{0:8.3f}”.format(123.2452312341234)
```

- Voor een compleet overzicht, zie de Python-documentatie hierover:

```
https://docs.python.org/2/library/string.html#formatspec
```

# Afsluitend

- Zorg dat je een werkende programmeeromgeving hebt!
- Vervolg stoomcursus UNIX.
- Werken met verschillende typen data.
- Werkcollege: de eerste programmeeropgave.
  - Deadline vrijdag 23 september 2016!
- Website:

<http://liacs.leidenuniv.nl/~rietveldkfd/courses/prna2016/>