FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

HOOFDSTUK 1

START

Helga Naessens

UNIVERSITEIT
GENT

De eerste 3 vuistregels

- Eerst nadenken, dan programmeren

- Een programma is een leesbare tekst over de oplossing van een probleem
  en kan op een computer uitgevoerd worden

- Oefening baart kunst: hoe meer je experimenteert hoe beter je programmeert en problemen kan oplossen

Inhoud Hoofdstuk 1

- **Een eerste programma**
- Inhoud programma
- Algoritme
- Testen

Een eerste programma

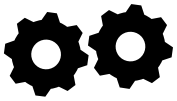
eerste.py

```
# berekent de oppervlakte en omtrek van een cirkel
# Stap 1: vraag de straal van de cirkel
# Stap 2: bereken de oppervlakte en omtrek
# Stap 3: print de resultaten

import math

straal_str = input("Straal van de cirkel in mm: ")
straal_int = int(straal_str)

omtrek = 2 * math.pi * straal_int  # omtrek berekenen
opp = math.pi * (straal_int ** 2)  # oppervlakte berekenen
print(f"omtrek: {omtrek:.2f}mm; oppervlakte: {opp}mm^2")
```

Uittesten… 

Commentaar

```
# berekent de oppervlakte en omtrek van een cirkel
# Stap 1: vraag de straal van de cirkel
# Stap 2: bereken de oppervlakte en omtrek
# Stap 3: print de resultaten
```

- # …
- Informatie voor lezer
- Wordt niet uitgevoerd
- Mag ook na een opdracht staan

```
omtrek = 2 * math.pi * straal_int  # omtrek berekenen
opp = math.pi * (straal_int ** 2)  # oppervlakte berekenen
```

Module importeren

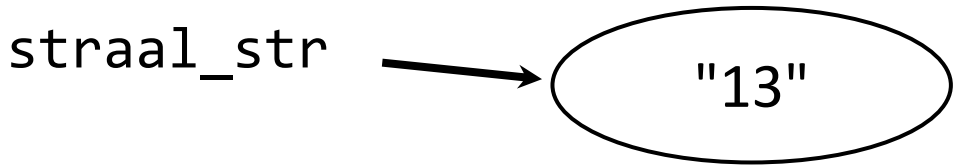
```
import math
```

- om bestaande programmacode te gebruiken
- `import` staat meestal aan het begin van een programma, maar mag ook vlak voor gebruik staan
- module **`math`** (zie API)→ oplossingen wiskundeproblemen
  - ➢ het getal π `math.pi`

Invoer

```
straal_str = input("Straal van de cirkel in mm: ")
```

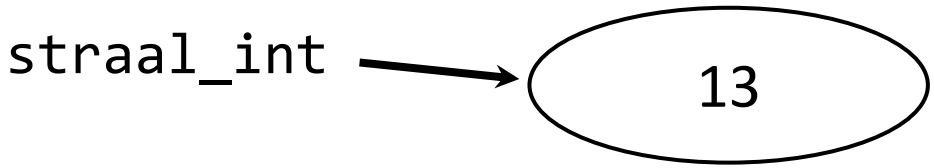
- **input(…)**
  - zie API
  - functie: stukje programmacode die een taak uitvoert
  - tekst (tussen " " of ' ') afdrukken
  - resultaat: wat de gebruiker intikt (tekst!)
- **straal_str**
  - variabele
- **=**
  - toekenning



Conversie

```
straal_int = int(straal_str)
```

- functie **`int(...)`**
  - tekst → geheel getal
  - string (`str`) → integer (`int`)
  - error indien tekst niet kan omgezet worden naar een geheel getal



- functie **`float(...)`**
  - conversie naar reëel getal

Berekeningen

```
omtrek = 2 * math.pi * straal_int  # omtrek berekenen
opp = math.pi * (straal_int ** 2)  # oppervlakte berekenen
```

- maal: *
- macht: **
- andere operatoren: + -

  / (reële deling)

  // (gehele deling)

  $\Rightarrow$ 8/3 $\neq$ 8//3

  % (modulo = rest na gehele deling)

  9%3 = ?

  9%5 = ?

Uitvoer

```
print(f"omtrek: {omtrek:.2f}mm; oppervlakte: {opp}mm^2")
```

- funtie **`print(…)`**
  - informatie op scherm printen (newline op het einde)
  - tekst tussen `" "` of `' '`
  - variabele
  - combinatie

```
print('omtrek:')
```

```
print(omtrek)
```

```
print('omtrek:',omtrek)
```

- combinatie tekst/variabele(n) kan m.b.v. een **`f-string`**:
  - `f" "` of `f' '`
  - variabelen tussen `{ }`

Een eerste programma

eerste.py

```
# berekent de oppervlakte en omtrek van een cirkel
# Stap 1: vraag de straal van de cirkel
# Stap 2: bereken de oppervlakte en omtrek
# Stap 3: print de resultaten

import math

straal_str = input("Straal van de cirkel in mm: ")
straal_int = int(straal_str)

omtrek = 2 * math.pi * straal_int  # omtrek berekenen
opp = math.pi * (straal_int ** 2)  # oppervlakte berekenen
print(f"omtrek: {omtrek:.2f}mm; oppervlakte: {opp}mm^2")
```

Inhoud Hoofdstuk 1

- Een eerste programma
- **Inhoud programma**
- Algoritme
- Testen

Inhoud programma

- **Importeren (deel) module**
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- Commentaar
- Tokens
- Variabelen
- Types
- Tekstopmaak

Als je iets uit een module (zie API) wilt gebruiken, moet je (een deel van) de module importeren

Importeren module

- Om bestaande code voor specifieke problemen te kunnen gebruiken

naam module

```
import module
```

Gebruik: *module*.xxx

naam module

```
import module as var
```

Gebruik: *var*.xxx

Importeren deel module

naam module

```
from module import functie/constante
```

Gebruik: *functie/constante*

(zonder vermelding module)

voorbeeld

```
from math import pi
print(pi)
```

Module math

- https://docs.python.org/3/library/math.html
- Voorbeeld functies

Functie	Betekenis
`math.sin`	sinus van hoek in radialen
`math.cos`	cosinus van hoek in radialen
`math.degrees`	radialen → graden
`math.radians`	graden → radialen
`math.sqrt`	vierkantswortel

```
import math
print(math.cos(math.pi))   # -1.0
print(math.sin(math.radians(90))) # 1.0
```

Oefening

- Lees de 3 gehele coëfficiënten van een vierkantsvergelijking in en bereken de 2 nulpunten.
- <u>Voorbeeld</u>

  nulpunten van $2x^2 + 5x - 3$ zijn -3 en 0.5

Berekening nulpunten vierkantsvergelijking $ax^2 + bx + c$:

$D(\text{iscriminant}) = b^2 - 4ac$

$w_{1\_2} = (-b \pm \sqrt{D})/2a$

Inhoud programma

- Importeren (deel) module
- **Opdrachten en uitdrukkingen**
- Witruimte (*whitespace*)
- Commentaar
- Tokens
- Variabelen
- Types
- Tekstopmaak

Uitdrukking (*expression*)

- Deel van een opdracht
- Heeft een resultaat/waarde
- Combinatie van literals/variabelen en operatoren

```
4 * zijde
```

```
basis * hoogte/2
```

```
(kleine_basis + grote_basis) * hoogte/2
```

Opdracht (*statement*)

- Voert een taak uit
- Geeft geen waarde terug
- Kan een zijeffect hebben
  - waarde van een variabele verandert
  - …

```
print(4 * zijde)  # waarde uitdrukking tonen
```

```
opp = basis * hoogte/2
# waarde uitdrukking in variabele opp
```

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- **Witruimte (*whitespace*)**
- Commentaar
- Tokens
- Variabelen
- Types
- Tekstopmaak

Witruimte

- Spatie, tab, …
- Wordt genegeerd
  - in een opdracht
  - lege lijnen

```
print(  4 *   zijde)
```

```
oppervlakte    =basis * hoogte  /2
```

Witruimte **vooraan** is belangrijk, wordt niet genegeerd en mag **niet** zomaar toegevoegd worden!

Witruimte - inspringen

- **Indentatie** = witruimte aan het begin van een lijn
- **Vereist** in Python om opdrachten te groeperen

Opdracht over meerdere lijnen

- Continuation
- \

```
print("Dit is een zin die over meerdere \
lijnen uitgespreid wordt.")
```

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- **Commentaar**
- Tokens
- Variabelen
- Types
- Tekstopmaak

Commentaar

- Alles na **#** wordt genegeerd

```
# oppervlakte trapezium
opp = (kleine_basis + grote_basis) * hoogte/2
```

```
opp = basis * hoogte/2   # oppervlakte 3hoek
```

Commentaar

Let us change our traditional attitude to the construction of programs. Instead of imaging that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

Donald Knuth, 1984

- Verhoogt leesbaarheid
- Niet uitgevoerd

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- Commentaar
- **Tokens**
- Variabelen
- Types
- Tekstopmaak

Token = symbool of woord met een speciale betekenis

Tokens: sleutelwoorden (keywords)

- Gereserveerde woorden
- Kan je niet gebruiken als namen van variabelen, …

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	class
exec	in	raise	continue	finally
is	return	def	for	lambda
try	True	False	None	

TABLE 1.1 Python keywords.

```
import = input("a:")
```

Tokens: operatoren

- Opdrachten zoals som, product, …

+	-	*	* *	/	//	%
<<	>>	&	|	^	~	
<	>	<=	>=	==	!=	<>
+=	-=	*=	/=	//=	%=	
&=	|=	^=	>>=	<<=	* *=	

TABLE 1.2 Python operators.

Tokens: scheidingstekens

- *Punctuators* of *delimiters*
- Afscheiden verschillende delen van een opdracht

(	)	[	]	{	}
,	:	.	`	=	;
'	"	#	\	@	

TABLE 1.3 Python punctuators.

Tokens: constanten

- *Literals*
- Vaste waarden
- Niet aanpasbaar tijdens uitvoering programma

```
straal = 25
```

```
# bereken oppervlakte driehoek
opp = basis * hoogte/2
```

```
print('oppervlakte =',opp)
```

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- Commentaar
- Tokens
- **Variabelen**
- Types
- Tekstopmaak

Variabele

- Aanpasbaar
- Aangemaakt bij eerste gebruik
- **Toekenning** (**=**) associeert waarde met de naam

```
straal = 20
omtrek = 2 * straal * math.pi
```

variabele

toekenning

waarde

Variabele aanpassen

```
straal = 20
straal = 30
straal = straal/3
getal = straal
```

- Waarde bepalen **uitdrukking rechts**
- Resultaat associëren met de **variabele links**
- Variabele heeft twee rollen:
  - RL: waarde ophalen `straal/3`
  - LL: waarde associëren `straal =`

Naamgeving

The practitioner of … programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, ***chooses the names of variables carefully*** *and explains what each variable means. He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding, using a mixture of formal and informal methods that reinforce each other.*

Donald Knuth, 1984

- Duidelijke namen verhogen de leesbaarheid

Naamgeving: voorwaarden

- Combinatie van: letters, cijfers en _
  - géén spatie, leesteken, ...
- Eerste teken: letter of _
  - géén cijfer
- Geen sleutelwoorden
- Hoofdlettergevoelig

```
straal
```

```
aantal_lln
```

```
my_name ≠ my_Name
```

Stijlgids: namenconventie variabelen

- Start met kleine letter
- Verschillende woorden verbinden met _

```
straal
```

```
radius_int
```

```
mijn_naam
```

Kwis

Welke van de onderstaande namen zijn als naam voor een variabele in Python:

- Slecht = voldoet niet aan de voorwaarden
- Aanvaardbaar = voldoet aan de voorwaarden, maar niet aan de stijlgids
- Goed = voldoet aan voorwaarden en stijlgids

a) `1waarde`

b) `ab`

c) `c&a`

d) `Save2db`

e) `verbinding_db`

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- Commentaar
- Tokens
- Variabelen
- **Types**
- Tekstopmaak

Type

- Elke gegeven is van een bepaald type
  - 598, 8, -54 zijn van het type `int` (gehele getallen)
  - 24.26, -85475.25e-15 zijn van het type `float`
  - 'tekstje', "dit zijn 4 woorden" zijn van het type `str`
- Type bepaalt
  - interne structuur
  - operaties: wat je ermee kan doen
- Type bepalen (zie API): functie `type(…)`

```
>>> s = '3'
>>> type(s)
<class 'str'>
>>> s = 3
>>> type(s)
<class 'int'>
>>> s = 3.0
>>> type(s)
<class 'float'>
```

Merk op

- Waarde van een variabele kan van type veranderen!
- Reële getallen (`float`): benaderingen → afrondingsfouten

  Let op als je reële getallen met elkaar vergelijkt (zie later)!
- Type resultaat deling (`/` en `//`):

```
>>> print(f"{5/3} {type(5/3)}")
1.6666666666666667 <class 'float'>
>>> print(f"{5//3} {type(5//3)}")
1 <class 'int'>
>>> print(f"{5.0//3} {type(5.0//3)}")
1.0 <class 'float'>
```

⇒ resultaat `/` : steeds `float` (ook bij `6/3`)

⇒ resultaat `//` : `int` als type teller én noemer `int` zijn;
anders `float`

Ingebouwde types

Type	Operatoren	Voorbeelden
`int`	`+ - * ** / // %`	27, -8521
`float`	`+ - * ** /`	11.658, -5.88e5
`bool`	…	True, False
`str`	…	'bla bla', "tekst"
`list`	…	[23, -4.89, 'code']
`dict`	…	{'BE':'België', 'D':'Duitsland', 'NL':'Nederland'}
`tuple`	…	(1, 3, 5, 7)

Volgorde bewerkingen (int, float)

Operator	Description
( )	Parenthesis (grouping)
* *	Exponentiation
+x , -x	Positive, Negative
* , / , % , //	Multiplication, Division, Remainder, Quotient
+ , -	Addition, Subtraction

TABLE 1.4 Precedence (order) of arithmetic operations: highest to lowest.

Verkorte notaties

Opdracht	Verkorte notatie
`getal = getal + 5`	`getal += 5`
`getal = getal - 9`	`getal -= 9`
`getal = getal / 3`	`getal /= 3`
`getal = getal // 3`	`getal //= 3`
`getal = getal * 2`	`getal *= 2`
`getal = getal ** 6`	`getal **= 6`
`getal = getal % 4`	`getal %= 4`

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- Commentaar
- Tokens
- Variabelen
- Types
- **Tekstopmaak**

Tekstopmaak met f-string

- tekst opmaken kan m.b.v. een **`f-string`**:

```
f'tekst {waarde1} tekst {waarde2}'
```

```
naam = "Tom"
leeftijd = 8
print(f"{naam} is {leeftijd} jaar oud")
# prints: Tom is 8 jaar oud
el, symb, nr = "boor", "B", 5
print(f"{el} ({symb}): atoomnr {nr}")
# prints: boor (B): atoomnr 5
```

Formaat f-string

```
{var[:[align][width][.precision][descriptor]]}
```

- ***align***: **<** (links), **>** (rechts) of **^** (center)
- ***width***: minimale breedte (default: juist genoeg)
- ***precision***: aantal cijfers na de komma (voor reëel getal, laatste cijfer wordt afgerond)
- ***descriptor***:

s	string
d	geheel getal (decimaal)
f	kommagetal
e	kommagetal (wet. notatie)
%	reëel getal als percent

Voorbeelden tekstopmaak

```
import math
print(math.pi)
print(f"Pi = {math.pi:.4f}")
print(f"Pi = {math.pi:8.3f}")
```

```
3.141592653589793
Pi = 3.1416
Pi =    3.142
```

Inhoud Hoofdstuk 1

- Een eerste programma
- Inhoud programma
- **Algoritme**
- Testen

Algoritme

- **Recept**
  - Probleem oplossen
  - Eindig aantal stappen

- Voorbeeld: selection sort

Programma?

- Een programma is een leesbare tekst
  over de oplossing van een probleem
  en kan op een computer uitgevoerd worden

- Programmeren = omzetten van het algoritme naar een programma in een bepaalde programmeertaal
  - ➢ Leesbaar
  - ➢ Uitvoerbaar

Oefening

- Lees een geheel getal in (=`getal`)
- Wissel de 2 laatste cijfers van `getal` en print `getal` opnieuw uit.

```
Geef een getal in: 6428
getal = 6428
nu is getal = 6482
```

Inhoud Hoofdstuk 1

- Een eerste programma
- Inhoud programma
- Algoritme
- **Testen**

Vuistregel

Test je code vaak en grondig!

Debuggen

- Programma werkt niet of werkt verkeerd
  → fouten zoeken en oplossen
- Drie soorten programmeerfouten

  a) **Syntax fout**: fout in een instructie.

  b) **Runtime fouten**: fout tijdens het uitvoeren; programma crasht door onverwachte gegevens van buitenaf.

  c) **Logische fout**: Code is correct geschreven, maar werkt niet zoals het moet.
- **Lees de foutboodschap**!

Syntax fout

```
>>> straal = input('Geef straal)
  File "<stdin>", line 1
    straal = input('Geef straal)
                                ^
SyntaxError: EOL while scanning string literal
>>>
```

Runtime fout

```
>>> aantal_appels = 35
>>> aantal_kinderen = 0
>>> aantal_appels_per_kind = aantal_appels/aantal_kinderen
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> int('30 graden')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '30 graden'
```

Inhoud Hoofdstuk 1

- Een eerste programma
- Inhoud programma
- Algoritme
- Testen

Inhoud programma

- Importeren (deel) module
- Opdrachten en uitdrukkingen
- Witruimte (*whitespace*)
- Commentaar
- Tokens
- Variabelen
- Types
- Tekstopmaak

Vuistregels

- Eerst nadenken, dan programmeren

- Een programma is een leesbare tekst over de oplossing van een probleem en kan op een computer uitgevoerd worden

- Oefening baart kunst: hoe meer je experimenteert hoe beter je programmeert en problemen kan oplossen

- Test je code vaak en grondig!