FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

HOOFDSTUK 1

START

prof. Helga Naessens

Ben Van Herbruggen
Leen Brouns

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

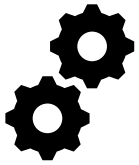
- **Een eerste programma**
- Inhoud programma
- Algoritme
- Testen

Een eerste programma

cirkel.py

```
1  # bereken de omtrek en oppervlakte van een cirkel
2  # als je de straal kent
3  # stap 1: vraag de straal
4  # stap 2: bereken
5  # stap 3: print de resultaten
6
7  import math
8
9  straal_tekst = input("Geef de straal: ")
10 straal_getal = int(straal_tekst)
11
12 omtrek = 2 * math.pi * straal_getal  # pi uit math-bib
13 oppervlakte = math.pi * (straal_getal ** 2)
14
15 print(f"De omtrek is {omtrek}")
16 print(f"De oppervlakte is {oppervlakte}")
```

Uittesten…



Commentaar

```
1   # bereken de omtrek en oppervlakte van een cirkel
2   # als je de straal kent
3   # stap 1: vraag de straal
4   # stap 2: bereken
5   # stap 3: print de resultaten
```

- # …
- Informatie voor lezer

- Wordt niet uitgevoerd
- Mag ook na een opdracht staan

```
12   omtrek = 2 * math.pi * straal_getal  # pi uit math-bib
```

Module importeren

```
7   import math
```

- om bestaande programmacode te gebruiken
- `import` staat meestal aan het begin van een programma
- module **`math`** → oplossingen wiskundeproblemen

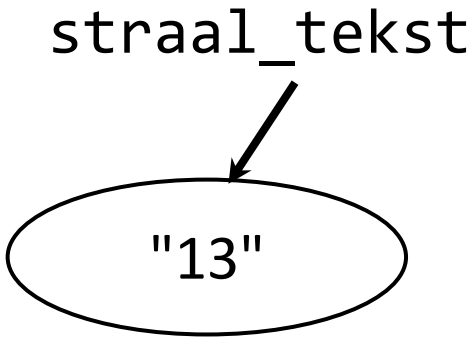
  ➢ Het getal π

  `math.pi`

Invoer

```
9   straal_tekst = input("Geef de straal: ")
```

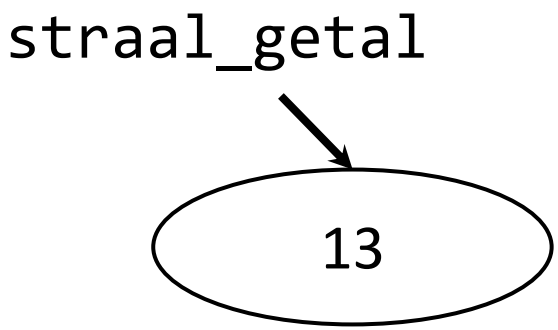
- **input(…)**
  - zie API (Application Programming Interface)
  - functie: stukje programmacode dat een taak uitvoert
  - de rode tekst (tussen " " of ' ') wordt afgedrukt
  - resultaat: de tekst die de gebruiker intikt (tekstvorm!)
- **straal**
  - variabele: naam voor een waarde
- **=**
  - Toekenning (krijgt de waarde van)



Conversie

```
10   straal_getal = int(straal_tekst)
```

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



Berekeningen

```
12   omtrek = 2 * math.pi * straal_getal  # pi uit math-bib
13   oppervlakte = math.pi * (straal_getal ** 2)
```

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

  / (reële deling)

  // (gehele deling)

  % (modulo = rest na gehele deling)

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

  9%3 = ?

  9%5 = ?

Uitvoer

```
15  print(f"De omtrek is {omtrek}")
16  print(f"De oppervlakte is {oppervlakte}")
```

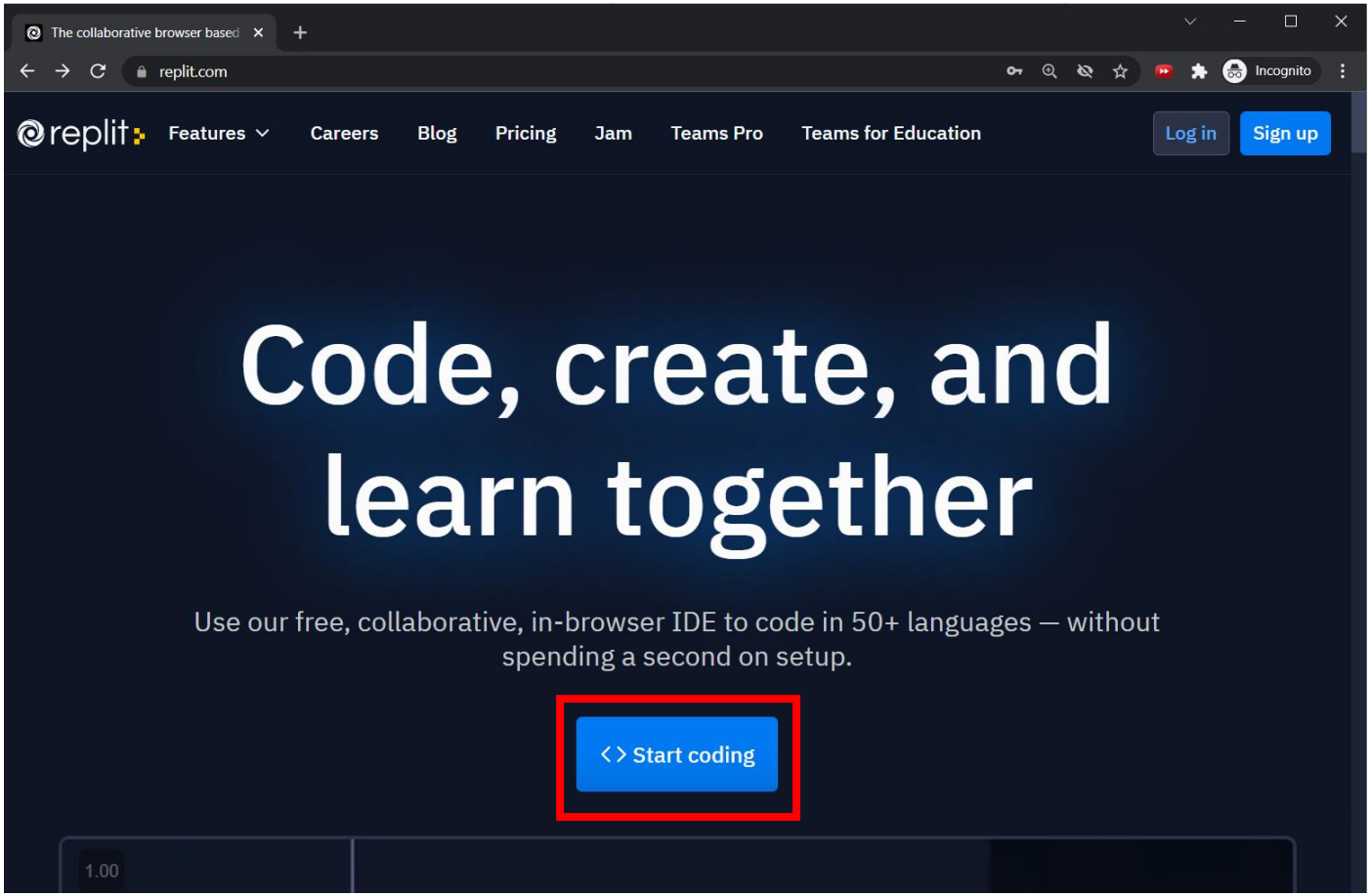
- funtie **`print(…)`**
  - informatie op scherm printen
  - tekst (tussen " " of ' ')
  - letter f voor " " om {} juist te interpreteren
  - waarde variabelen tussen {} binnen de tekst

Een eerste programma

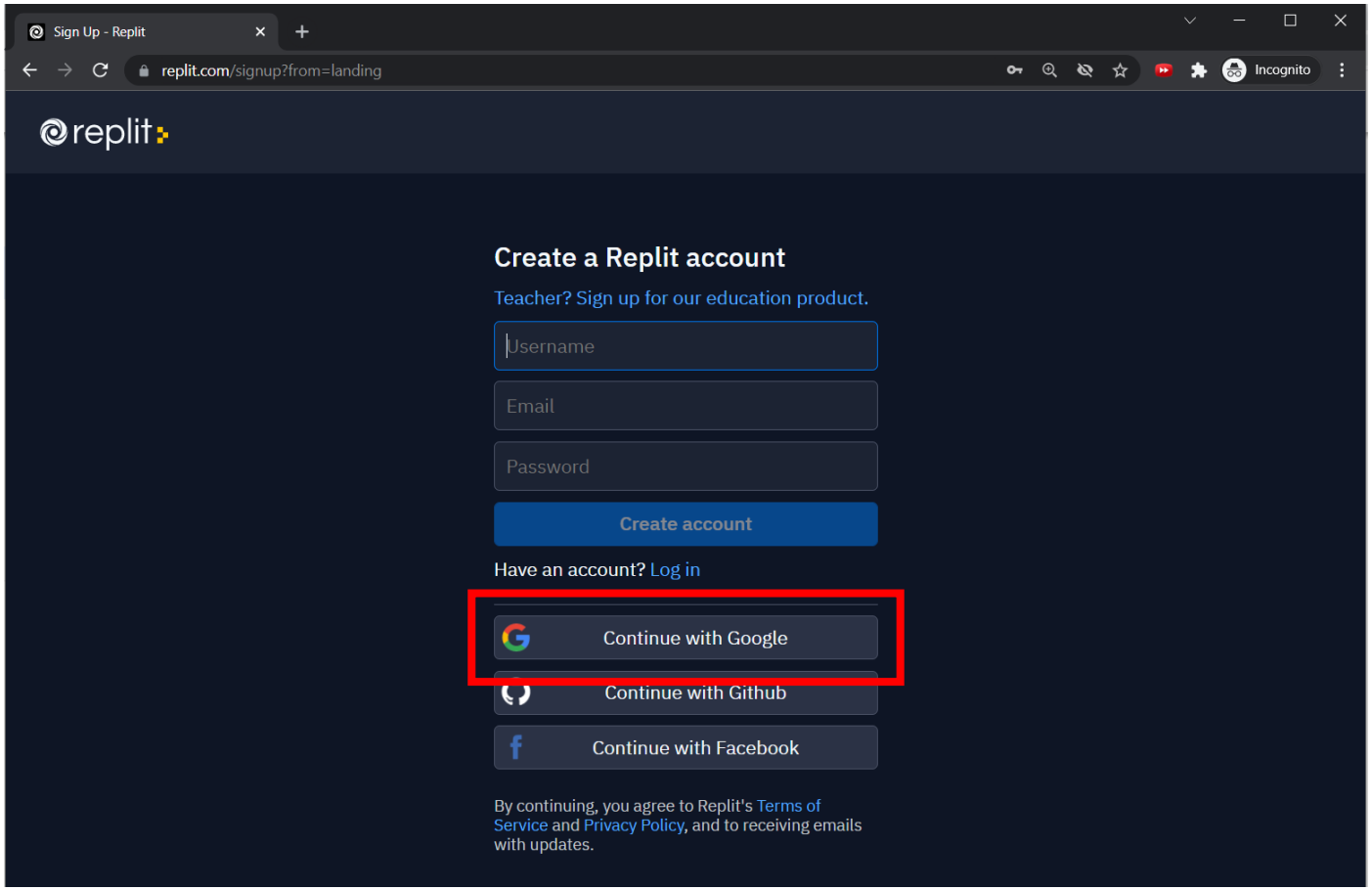
cirkel.py

```
1  # bereken de omtrek en oppervlakte van een cirkel
2  # als je de straal kent
3  # stap 1: vraag de straal
4  # stap 2: bereken
5  # stap 3: print de resultaten
6
7  import math
8
9  straal_tekst = input("Geef de straal: ")
10 straal_getal = int(straal_tekst)
11
12 omtrek = 2 * math.pi * straal_getal  # pi uit math-bib
13 oppervlakte = math.pi * (straal_getal ** 2)
14
15 print(f"De omtrek is {omtrek}")
16 print(f"De oppervlakte is {oppervlakte}")
```

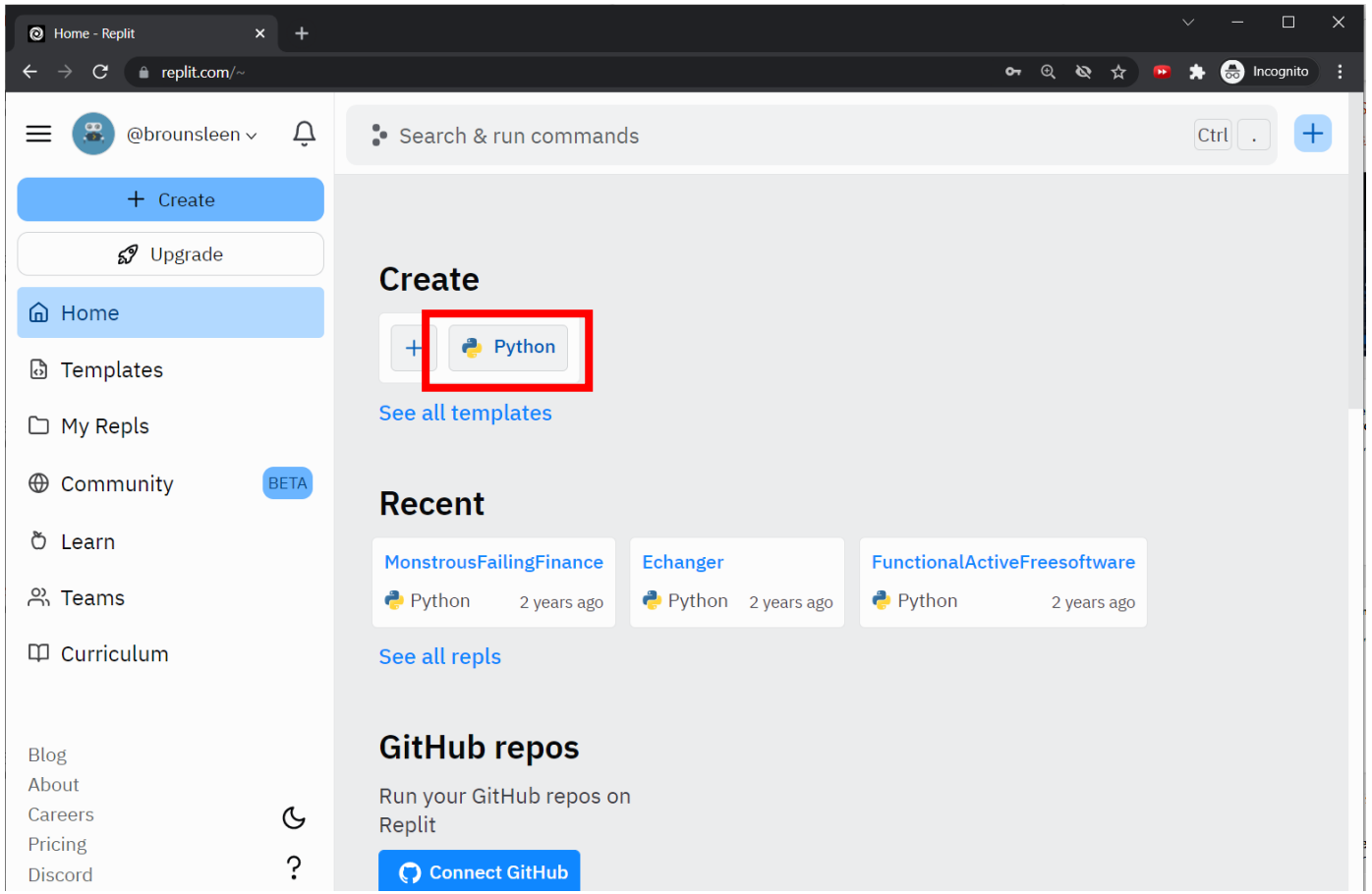
Een eerste programma in replit.com



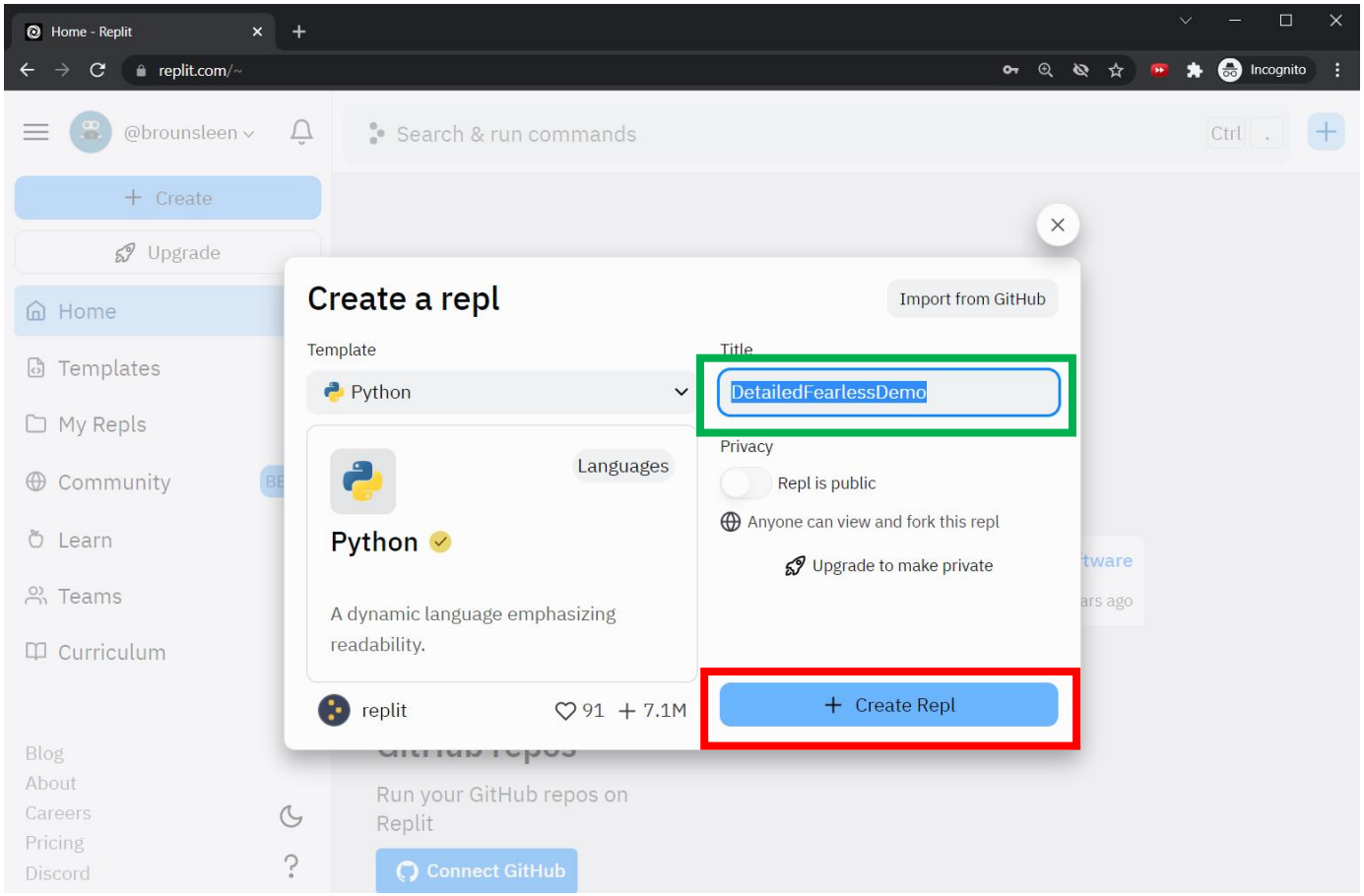
Een eerste programma in replit.com



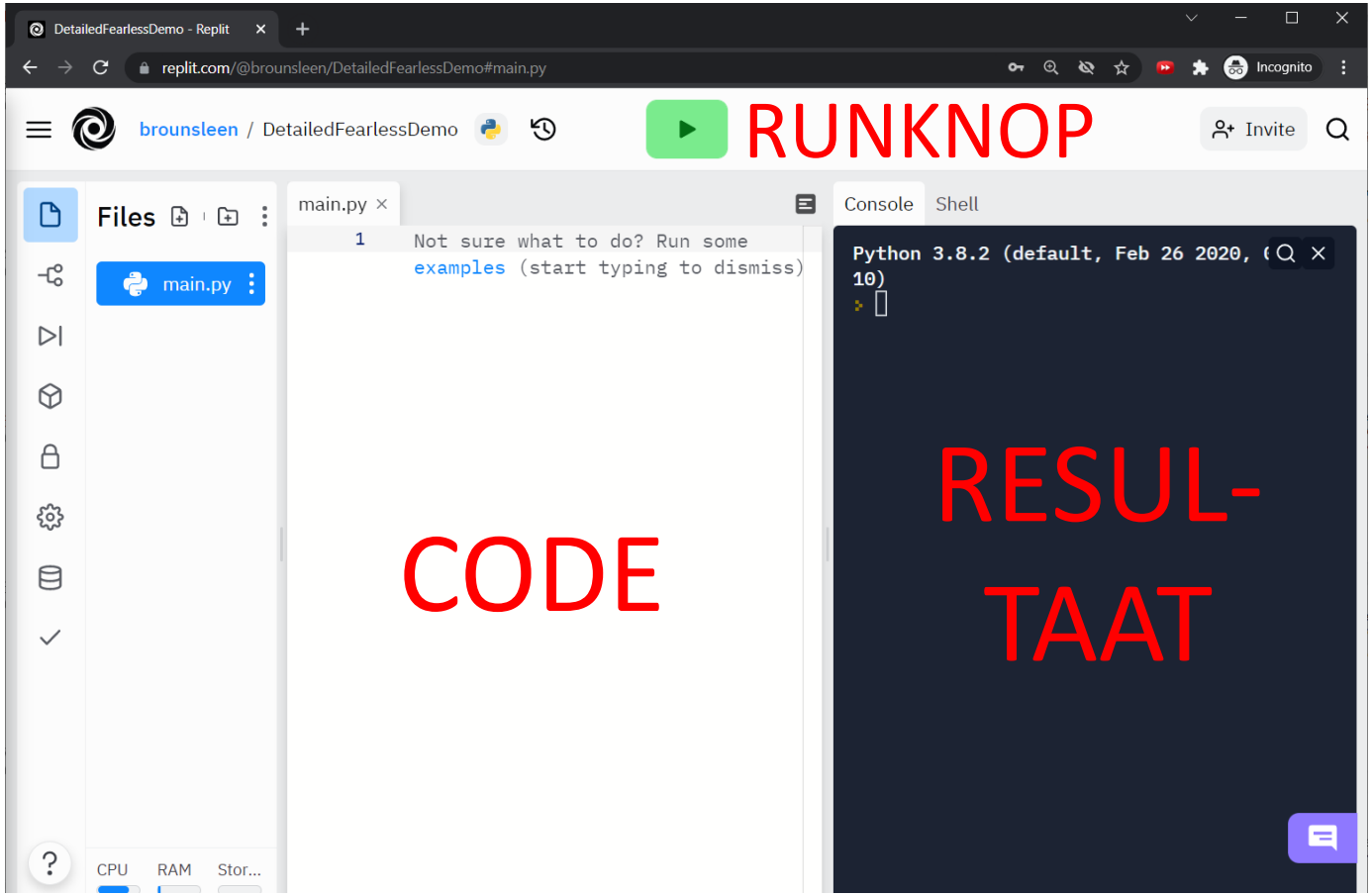
Een eerste programma in replit.com



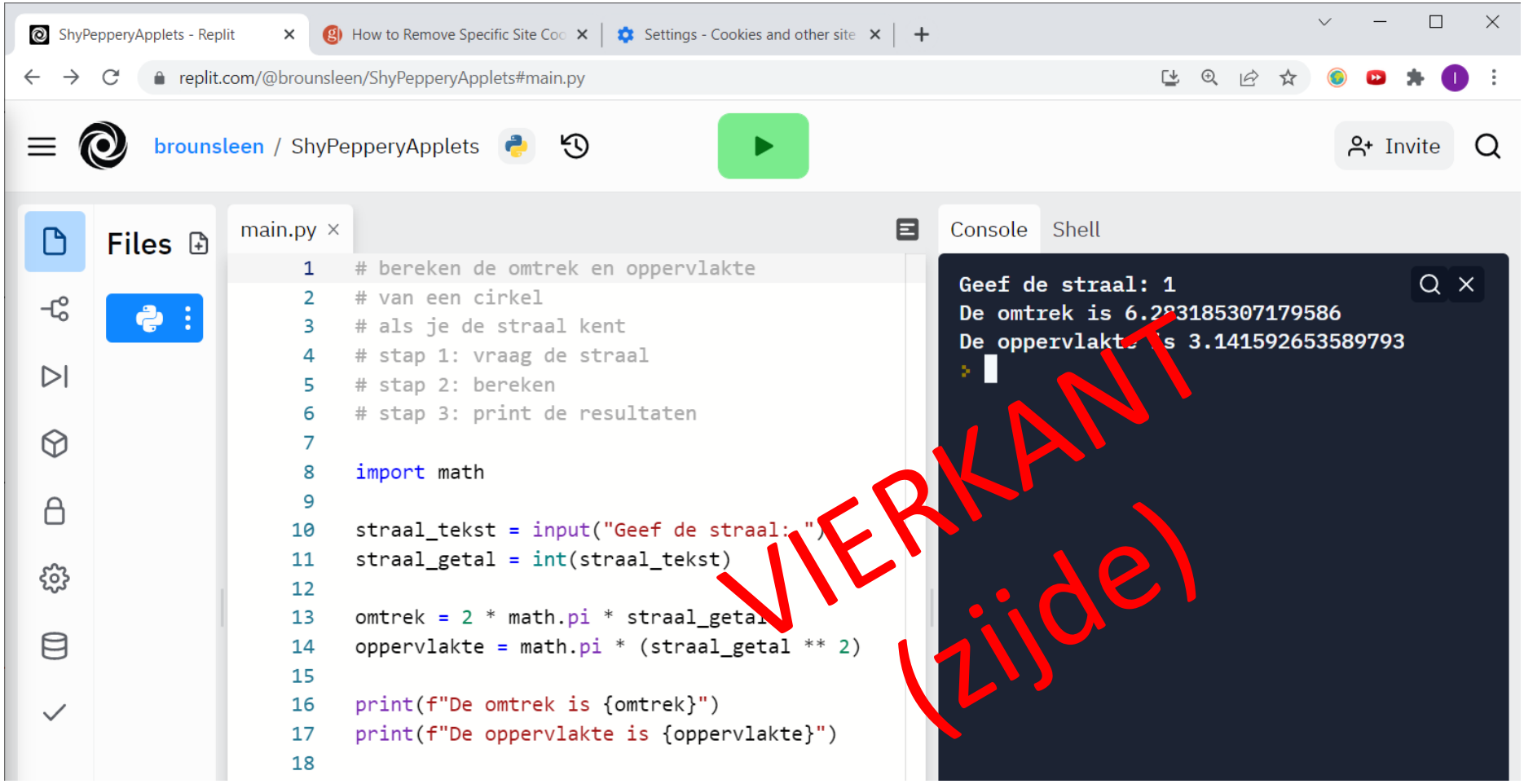
Een eerste programma in replit.com



Een eerste programma in replit.com



Een eerste programma in replit.com

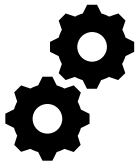


Een eerste programma

```
1   # bereken de omtrek en oppervlakte van een vierkant
2   # als je de zijde kent
3
4   zijde_tekst = input("Geef de zijde: ")
5   zijde_getal = int(zijde_tekst)
6
7   omtrek = 4 * zijde_getal
8   oppervlakte = zijde_getal ** 2
9
10  print(f"De omtrek is {omtrek}")
11  print(f"De oppervlakte is {oppervlakte}")
```

Programma vertalen en uitvoeren





programma **interpreteren** binaire code

- interpreteren
  - omzetten lijn per lijn
  - elke lijn onmiddellijk uitgevoerd
- in 1e bach: Visual Studio Code  software om software te maken

  nu: replit.com website, geen installatie nodig

Inhoud Hoofdstuk 1

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

Inhoud programma

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

Importeren module

naam module

```
import module
```

- Bestaande code voor specifieke problemen
- Bewaard in een bestand
- Gebruik: `module.xxx`

```
math.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

Oefening

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

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

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

D(iscriminant) = $b^2 - 4ac$

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

Inhoud programma

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

Uitdrukking (*expression*)

- Nieuwe waarde - resultaat
- Combinatie waarden 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 module
- Opdrachten en uitdrukkingen
- **Witruimte (*whitespace*)**
- Commentaar
- Tokens
- Variabelen
- Types

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
- Tab of vier spaties
  - ➢ consistent!

Opdracht over meerdere lijnen

- Continuation
- Soms nodig voor leesbaarheid
- \

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

Inhoud programma

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

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

- Verhogen leesbaarheid
- Niet uitgevoerd

Commentaar

- Alles na # genegeerd

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

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

Inhoud programma

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

Token = symbool of woord met een speciale betekenis

Tokens: sleutelwoorden (keywords)

- Gereserveerde woorden
- Opdrachten voor interpreter
- 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
oppervlakte = basis * hoogte/2
```

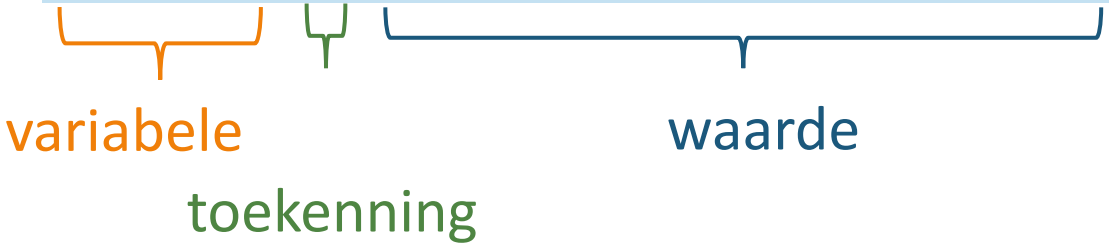
Inhoud programma

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

Variabele

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

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



Variabelen gerealiseerd in Python

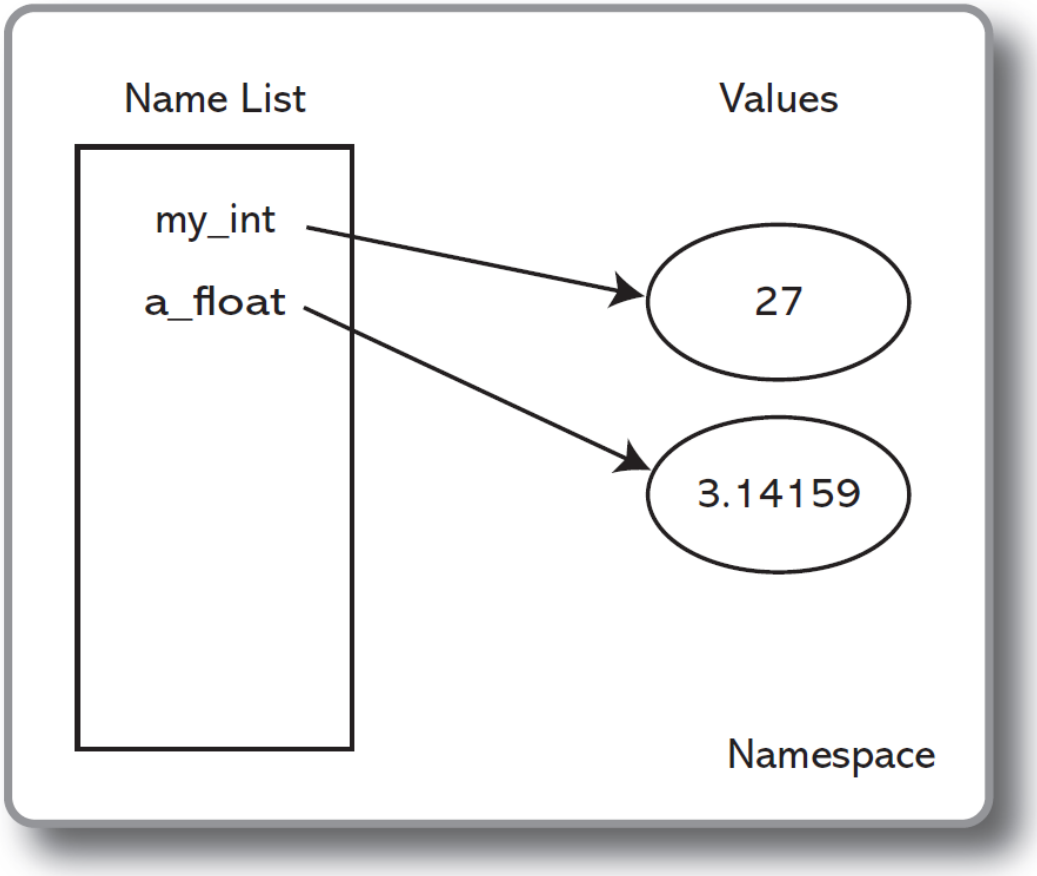


FIGURE 1.1 Namespace containing variable names and associated values.

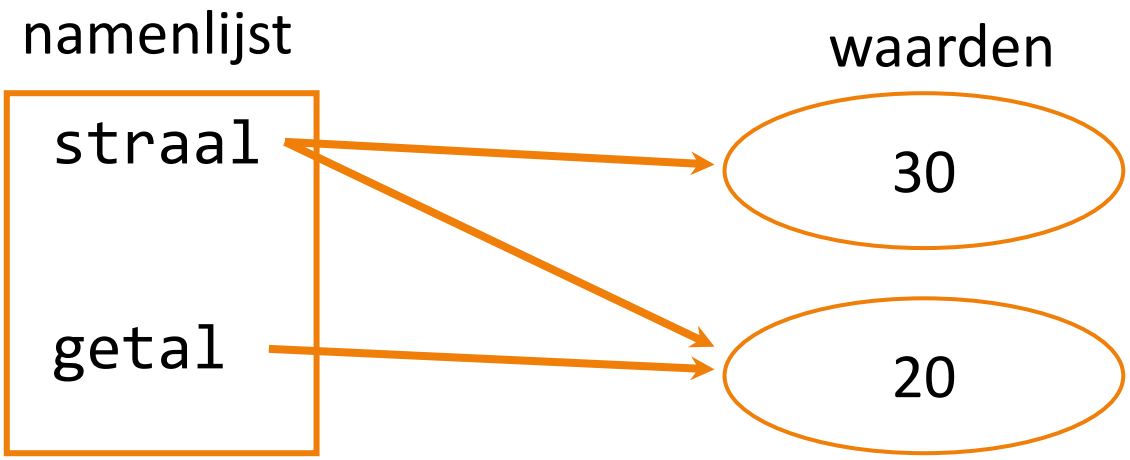
Variabele aanpassen

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

- Waarde bepalen **uitdrukking rechts**
- Resultaat associëren met de **variabele links**
- Variabelen heeft twee rollen
  - Waarde ophalen `straal/3`
  - Waarde associëren `straal =`

Toekenning

```
straal = 30
getal = 20
straal = getal
```



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

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

```
straal
```

```
aantal_lln
```

```
my_name ≠ my_Name
```

Stijlgidsen

- https://www.python.org/dev/peps/pep-0008/
- https://github.com/google/styleguide/blob/gh-pages/pyguide.md
- Vuistregel 4:

  A foolish consistency is the hobgoblin of little minds

  Rechtlijnigheid is de boeman van kleingeestige mensen

- We passen namenconventies toe, maar soms wijken we af om de leesbaarheid te verhogen

Namenconventie variabelen

- Start met kleine letter
- Verschillende woorden verbinden met _

```
straal
```

```
radius_int
```

```
mijn_naam
```

Kwis

Welke van de onderstaande namen zijn fout/aanvaardbaar/goed als naam voor een variabele in Python?

a) 1waarde

b) abab

c) c&a

d) Save2db

e) verbinding_db

Inhoud programma

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

Object

- Elke variabele verwijst naar een **object**
- Een object heeft
  - een unieke identificatie
    → functie **`id`**(*`object/variabele`*)
  - 0 of meer namen

```
>>> v = 15
>>> id(v)
2479310334704
>>> id(15)
2479310334704
>>> w = v
>>> id(w)
2479310334704
```

Type

- Elke object is een **instantie** van een 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: functie `type(…)`

```
>>> s = "3"
>>> type(s)
<class 'str'>
>>> s = 3
>>> type(s)
<class 'int'>
>>> s = 3.0
>>> type(s*2)
<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(5.2//3,' ',type(5.2//3))
1.0   <class 'float'>
>>> print(5/3,' ',type(5/3))
1.6666666666666667   <class 'float'>
>>> print(5//3,' ',type(5//3))
1   <class 'int'>
```

Ingebouwde types

Type	Operaties	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’}
`set`	…	{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`

Oefening

Controleer of een ingelezen rekeningnummer correct is.
Een rekeningnummer is een geheel getal bestaande uit 12 cijfers.
Als je van de eerste 10 cijfers de rest bij deling door 97 neemt, dan zou je de laatste 2 cijfers moeten bekomen.
Behalve als de rest 0 is, dan vormen de laatste 2 cijfers het getal 97.

Print de berekening en de laatste 2 cijfers.

Inhoud Hoofdstuk 1

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

Algoritme

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

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 5

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

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

- Rechtlijnigheid is de boeman van kleingeestige mensen

- Test je code vaak en grondig!