TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

TW1090 Inleiding Programmeren
Tentamen
30 januari 2017, 9:00 – 12:00 uur

Normering: Opgave 1 t/m 3 ieder 6 punten. Score: totaal aantal punten + 2. Cijfer: score / 2

Voorbereiding

Lees eerst de instructies op het aparte Instructie blad (Inloggen, IDLE starten, pdf van het boek openen).

Gebruik bij de implementatie van de opgaven alleen de modules / bibliotheken die nu al door middel van imports worden geïmporteerd. Voeg dus geen import statements toe. Verander ook de hoofdprogramma's niet, indien al een hoofdprogramma is gegeven. Voeg dan alleen implementaties toe van de gevraagde functies en test deze functies in het hoofdprogramma. Vul je naam en studienummer ook in het commentaar van de programma's in (zie de inhoud van de .py files). Open in de IDLE omgeving de .py file die je wilt gaan editen. Zorg ervoor dat je steeds maar 1 .py file tegelijk geopend hebt. Save je .py files regelmatig.

Opgave 1 (punten: a: 2, b: 2, c: 2)

Gebruik voor deze opdracht het programma in de file **`A1.py`**. Een object beweegt van tijdstip *tstart* tot *teind* langs een baan in de 3D ruimte.

De positie van het object op het tijdstip *t* wordt gegeven door $\begin{cases} x = 10\cos t \\ y = t\sin t \\ z = 2t \end{cases}$ met $t \in [ts, te]$.

Hierbij wordt [*ts*, *te*] verdeeld in 50 tijdstappen van gelijke grootte. Waarden van *ts* en *te* worden aan de gebruiker van het programma gevraagd. Implementeer de volgende functies:

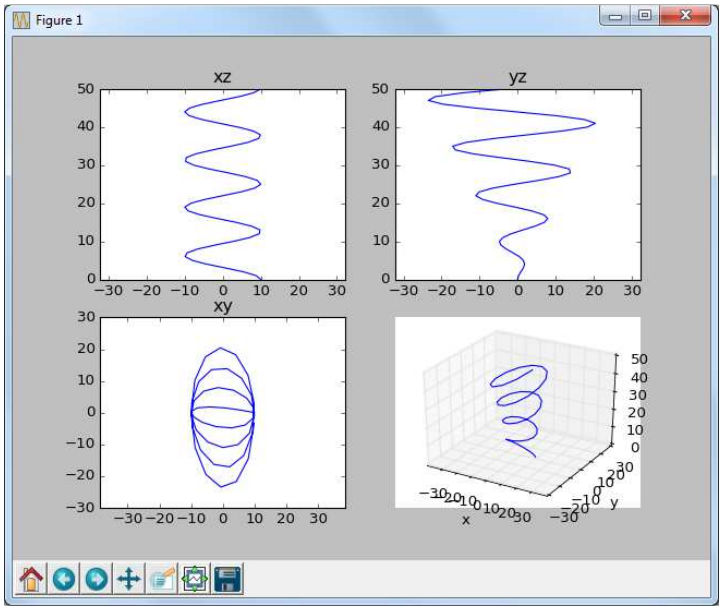
a) `createCurve(ts, te):`
Deze functie geeft 3 numpy arrays `x`, `y` en `z` terug met voor ieder tijdstip de x-, y- en z-coordinaat van een punt op de baan. Hierbij wordt [*ts*, *te*] verdeeld in 50 tijdstappen van gelijke grootte. Voer de berekeningen uit op complete numpy arrays, dus zonder gebruik van een lus.

b) `drawCurve(x, y, z):`
Deze functie plot een array van 2 bij 2 subplots zoals in de figuur op de volgende bladzijde. De eerste drie subplots tonen respectievelijk de projectie van de baan van het object op het xz-vlak (links boven), op het yz-vlak (rechts boven) en op het xy-vlak (links onder). De vierde subplot is een 3D plot van de baan (rechts onder). De code voor deze laatste subplot is al in de functie geïmplementeerd. Zorg ervoor dat de schaal op de assen overeenkomt (niet uitgerekt in x- of y-richting). Zet ook de juiste tekst boven de plots.

c) `averageVelocity(delta_t, x, y, z):`
Deze functie rekent de gemiddelde snelheid van de beweging van het object langs de baan uit. Dit wordt gedaan door voor ieder tijdsinterval $[t_i, t_{i+1}]$ de afgelegde weg in dat tijdsinterval te benaderen met de lengte

van een lijnstukje en deze lengtes op te tellen. Tenslotte wordt er gedeeld door de totale verstreken tijd (`delta_t` = *teind* - *tstart*).

Zie hiernaast de matplotlib figuur en hieronder de uitvoer in de Python shell voor *tstart* = 0 en *teind* = 25 na implementatie van de drie gevraagde functies.



Python 2.7.10 Shell

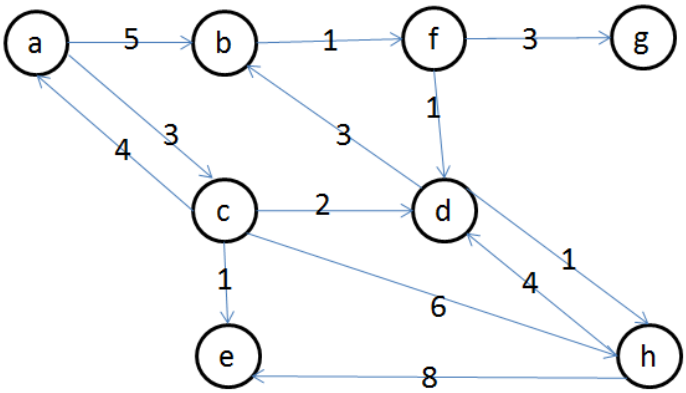
File Edit Shell Debug Options Window Help

```
>>>
van t = 0
tot t = 25
x =
[ 10.          8.77582562   5.40302306   0.70737202  -4.16146837
  -8.01143616  -9.89992497  -9.36456687  -6.53643621  -2.10795799
   2.83662185   7.08669774   9.60170287   9.76587626   7.53902254
   3.46635318  -1.45500034  -6.02011903  -9.11130262  -9.97172156
  -8.39071529  -4.75536928   0.04425698   4.83304759   8.43853959
   9.97798279   9.07446781   5.94920663   1.36737218  -3.54924267
  -7.59687913  -9.78453463  -9.5765948   -7.02397058  -2.75163338
   2.19439963   6.60316708   9.39524894   9.88704618   7.9581497
   4.08082062  -0.79563567  -5.4772926   -8.81791728  -9.99960826
  -8.7330464   -5.3283302   -0.61905294   4.24179007   8.06409494
   9.91202812]

y =
[  0.          0.23971277   0.84147098   1.49624248   1.81859485
   1.49618036   0.42336002  -1.2277413   -3.02720998  -4.39888553
  -4.79462137  -3.88047179  -1.67649299   1.39827992   4.59890619
   7.03499983   7.91486597   6.78714046   3.70906637  -0.71393564
  -5.44021111  -9.23680548 -10.99989227 -10.06770001  -6.43887502
  -0.82902372   5.46217148  10.85108976  13.86850298  13.55597831
   9.7543176    3.20024597  -4.60645307 -11.74445815 -16.34375736
 -17.0734551  -13.51777044  -6.33589144   2.84766698  11.80802746
  18.25890501  20.43501078  17.56976841  10.14023857  -0.1947288
 -10.96142653 -19.4630693  -23.45492766 -21.73388069 -14.48825948
  -3.30879375]

z =
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.  12.  13.  14.
  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.
  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.
  45.  46.  47.  48.  49.  50.]
De gemiddelde snelheid van t = 0.0 tot t = 25.0 is 11.6365511665
>>>
```

Ln: 39 Col: 4

Opgave 2 **(punten: a: 2, b: 4)**

In het programma **`A2.py`** worden gerichte grafen gerepresenteerd op de manier zoals op de collegeslides van lecture 7, dus als een dictionary van dictionaries. Knoopnamen zijn strings. Bijvoorbeeld:
`{'a': {'c': 3, 'b': 5}, 'c': {'a': 4, 'h': 6, 'e': 1, 'd': 2}, 'b': {'f': 1}, 'e': {}, 'd': {'h': 1, 'b': 3}, 'g': {}, 'f': {'d': 1, 'g': 3}, 'h': {'e': 8, 'd': 4}}` is de interne representatie van de hieronder getoonde graaf.



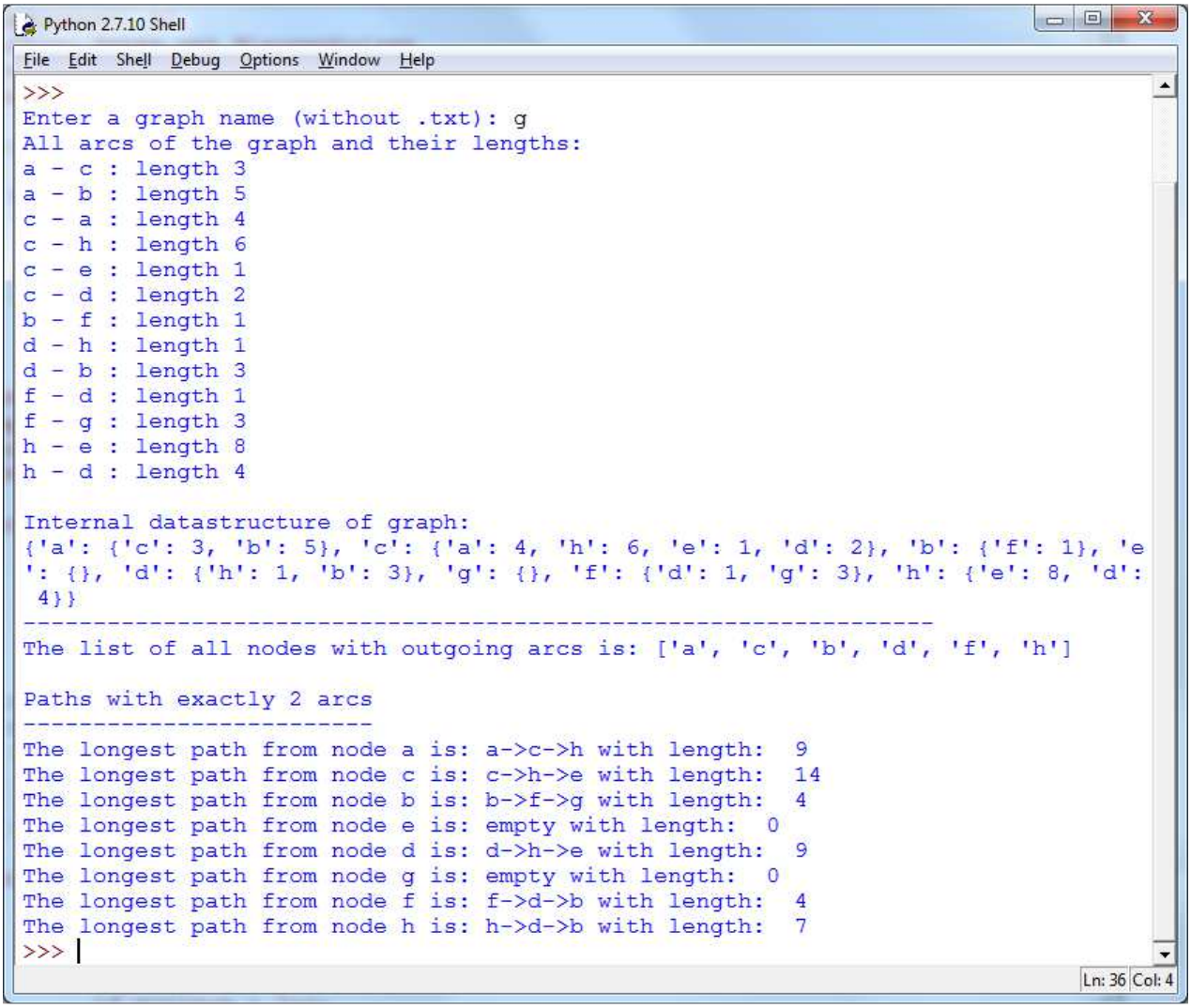
Deze graaf wordt ingelezen in het programma **`A2.py`** uit een file `g.txt` indien je als invoer van het programma alleen de letter `g` tikt.

a) Implementeer een functie `nodes(graph)`. Deze functie geeft een lijst terug van alle knopen uit de graaf die minstens 1 uitgaande pijl hebben.

b) Implementeer een functie `longestPath(node, graph)`. Deze functie geeft een pad vanuit knoop `node` terug dat aan de volgende voorwaarden voldoet:
- Het pad bestaat uit precies 2 pijlen.
- Het pad heeft de grootste lengte van alle paden, bestaande uit precies 2 pijlen, vanuit knoop `node`. Indien er meer dan één pad met de grootste lengte bestaat, dan wordt een van deze paden teruggegeven. Het maakt in dat geval niet uit welk van deze paden wordt teruggegeven.
- Het pad wordt teruggegeven als string (bestaande uit de knoopnamen met `->` ertussen).
- Behalve dit pad geeft de functie ook de lengte van het betreffende pad terug.
- Als er geen pad van 2 pijlen bestaat vanuit knoop `node`, dan geeft de functie `'empty', 0` terug.

Voorbeelden:
De functie toegepast op node `'a'` uit de graaf in bovenstaande figuur geeft `'a->c->h', 9` terug.
De functie toegepast op node `'g'` uit de graaf in bovenstaande figuur geeft `'empty', 0` terug.

Het testprogramma in de file **`A2.py`** geeft na correcte implementatie van de functies bij invoer `g` uitvoer zoals in het Python shell window op de volgende bladzijde.



Opgave 3 (punten: a: 1, b: 2, c: 3)

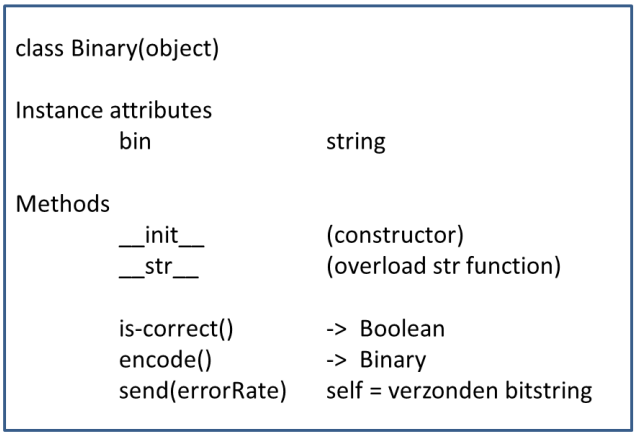
We willen boodschappen bestaande uit bits (nullen en enen) verzenden met gebruik van een fout detecterende code. Voor verzending voegen we aan een boodschap een 0 of een 1 toe (het zogenaamde pariteitsbit) zodat het aantal enen in de aangevulde boodschap altijd even is. Boodschappen worden in het programma in de file **`A3.py`** gerepresenteerd als strings (met uitsluitend nullen en enen).
Het nut van het toevoegen van het pariteitsbit is dat de ontvanger van een boodschap kan detecteren, wanneer er een fout in de verzending is opgetreden, mits er niet meer dan 1 bit verkeerd is overgekomen. Bij precies 1 fout bit zal namelijk het aantal enen in het ontvangen bericht oneven zijn, wat wijst op een fout in de verzending.

Voorbeeld:

Bericht	met pariteitsbit	ontvangen	fout in verzending
11010110111	110101101110	110**0**01101110	ja, wel gedetecteerd want 1 bit fout
11010110110	110101101110	110101101110	nee
01010110111	010101101111	010**0**01**0**01111	ja, niet gedetecteerd want 2 bits fout

In programma **`A3.py`** vind je een opzet van een klasse Binary voor binaire strings. De `__init__` en de `__str__` methoden zijn al geïmplementeerd. Het gegeven programma test de klasse. Indien je het

programma probeert te runnen dan komt er een foutmelding. De reden daarvan is dat een aantal methoden, die worden aangeroepen, nog niet in de klasse zijn geïmplementeerd. De klasse moet na aanvulling met de nu nog ontbrekende methoden voldoen aan de beschrijving in het klassendiagram hieronder.



Implementeer de methoden:

a) `is_correct()`: Deze methode controleert of een boodschap (na verzending) correct is, d.w.z. een even aantal enen bevat. De methode geeft True of False terug.
   Voorbeeld: `c = Binary('101110')` `c.is_correct()` → `True`
b) `encode()`: Deze methode maakt een boodschap gereed voor verzending door er een 0 of een 1 aan toe te voegen, zodat het aantal enen in de boodschap even is geworden.
   Voorbeeld: `c = Binary('100110')` `c.encode()` → `'1001101'`
c) `send(errorRate)`: Deze methode simuleert het verzenden van een bericht. Hierbij geeft `errorRate` voor ieder bit in de boodschap de kans in procenten aan op foute verzending. Ook het paritietsbit kan fout worden verzonden. Gebruik de module `random`, die al is geïmporteerd, om ieder bit met kans `errorRate` % te laten veranderen van een 0 naar een 1 of van een 1 naar een 0.
   Voorbeeld: zie de voorbeelden op de volgende bladzijde.

Test de methoden door runnen van het testprogramma en aanroepen van de te testen methoden in de Python shell na runnen van de klasse. Op de volgende bladzijde is de gewenste uitvoer van het testprogramma gegeven.

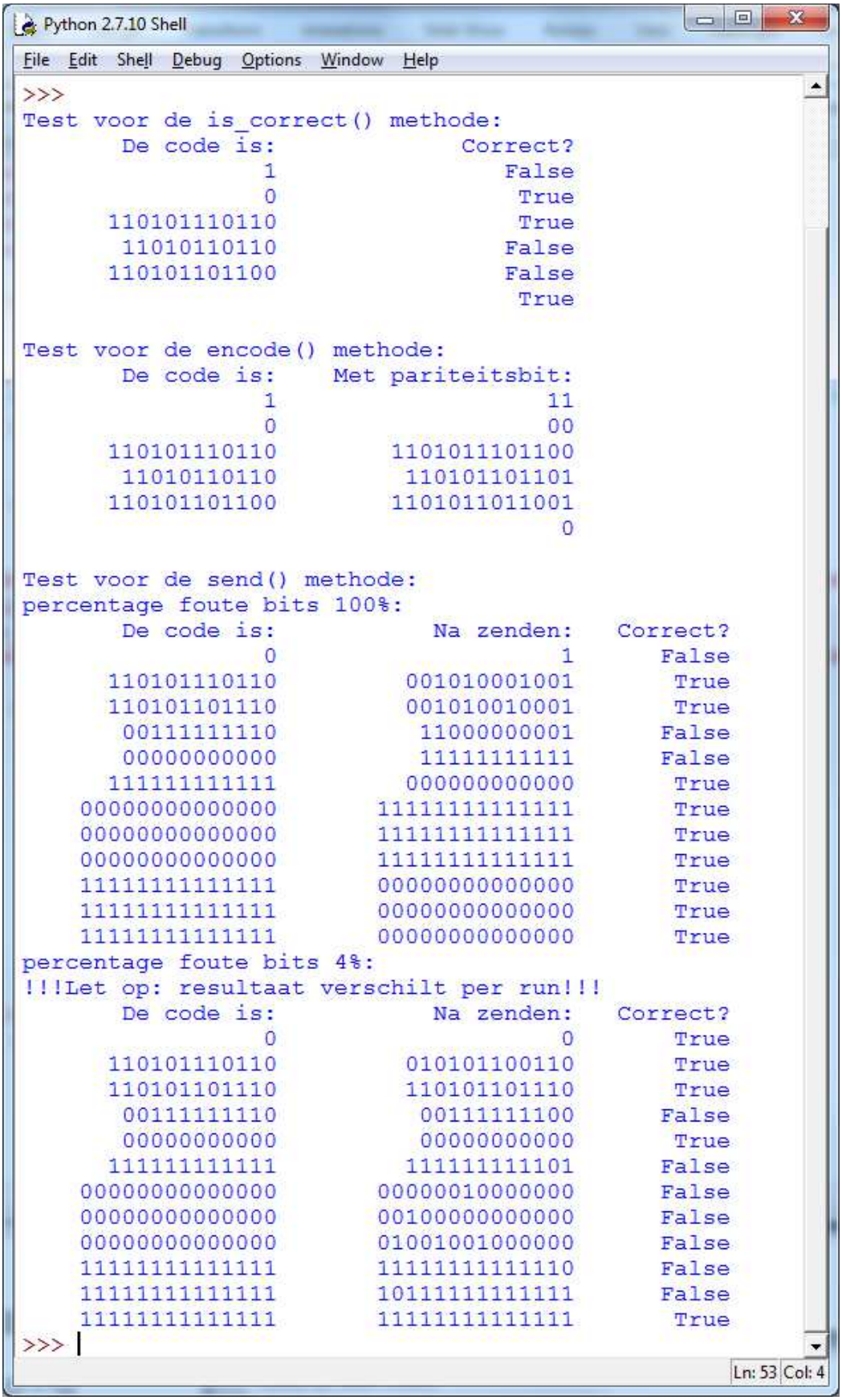
Afsluiten

Sluit alle .py bestanden. Controleer in IDLE door middel van Kies File | Open nog een keer extra of je in je directory **Homedrive(H:)** drie .py bestanden hebt staan met in de naam jouw naam en studienummer. **Deze drie bestanden vormen het werk dat je inlevert.**

Druk nu op het sluithokje rechts bovenin het IDLE window. Sluit op deze manier alle Editor en Python Shell windows van IDLE. Sluit ook de pdf van het boek. Kies tenslotte Start | Exit | OK. Je wordt nu uitgelogd.

AAN DE HAND VAN UITSLUITEND DE DRIE .PY BESTANDEN WORDT JE WERK BEOORDEELD.

INDIEN DE .PY BESTANDEN NIET OP HOMEDRIVE(H:) STAAN MAAR OP DE DESKTOP, DAN HEB JE GEEN WERK INGELEVERD.



einde opgaven