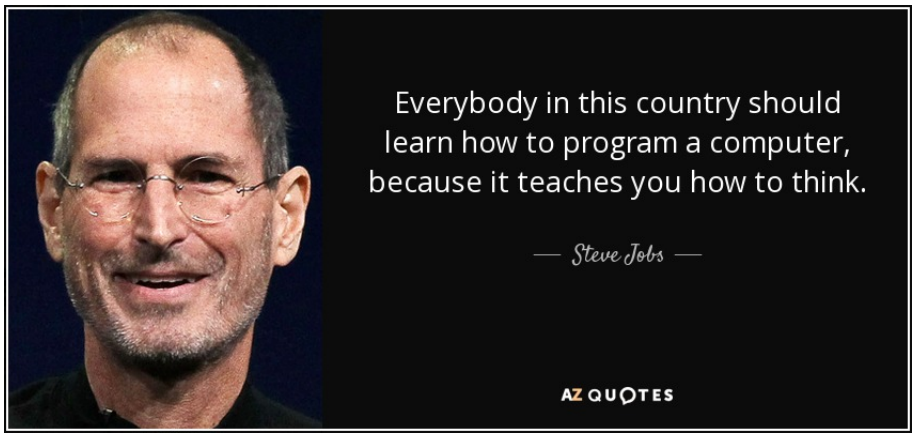
Practicum Wiskunde 3
Parametrisch Rekenmodel

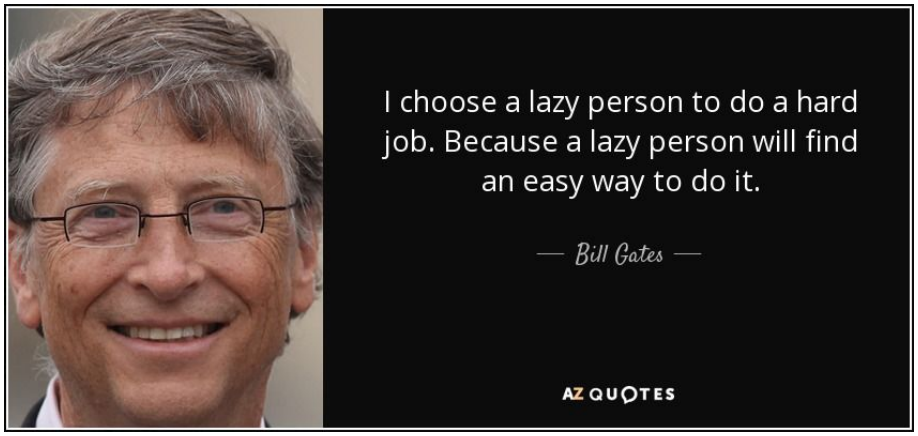
Datum 5/6/20
Docent: J.Antonissen

Doel

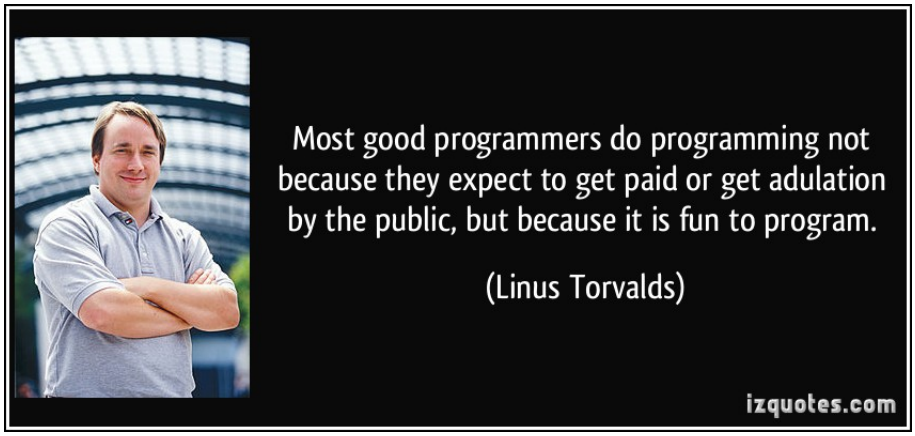
Programmeren is de meest belangrijke eigenschap van een ingenieur van de 21e eeuw. Hierdoor is het belangrijk dat ook jij leert programmeren. Programmeren lijkt op zichzelf iets dat lastig en abstract is, maar als je eenmaal eraan begint zal je merken dat programmeren vooral heel leuk is en je heel erg veel tijd gaat besparen. De grootste denkers van deze tijd sluiten zich hierbij aan:



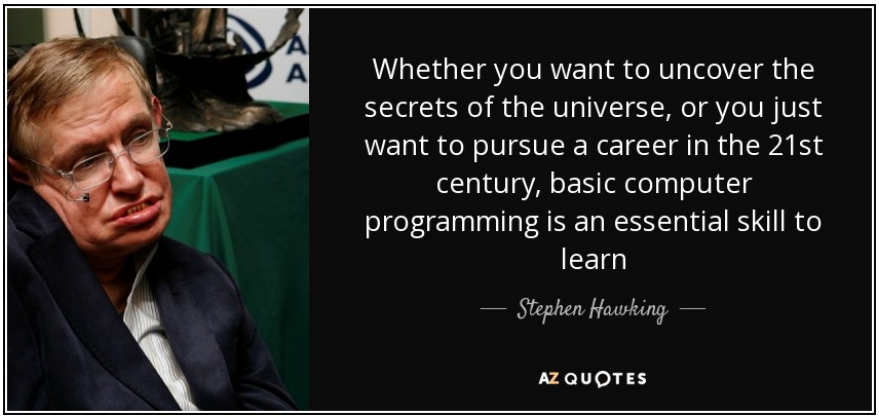
Figuur 1 - Apple



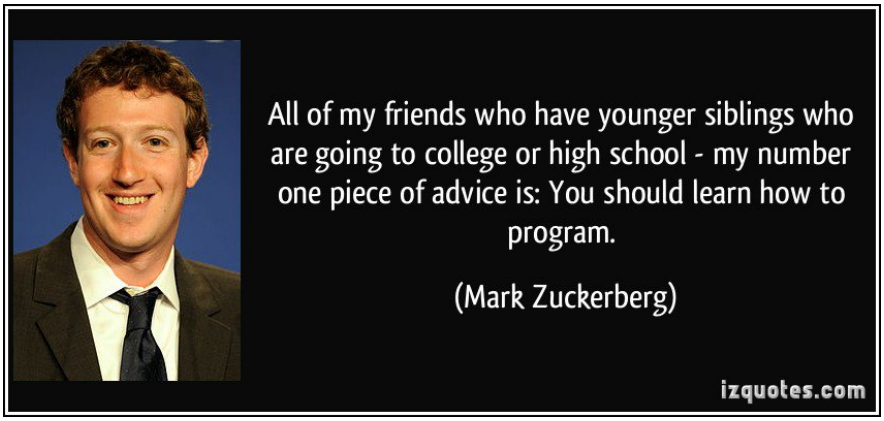
Figuur 2 - Windows



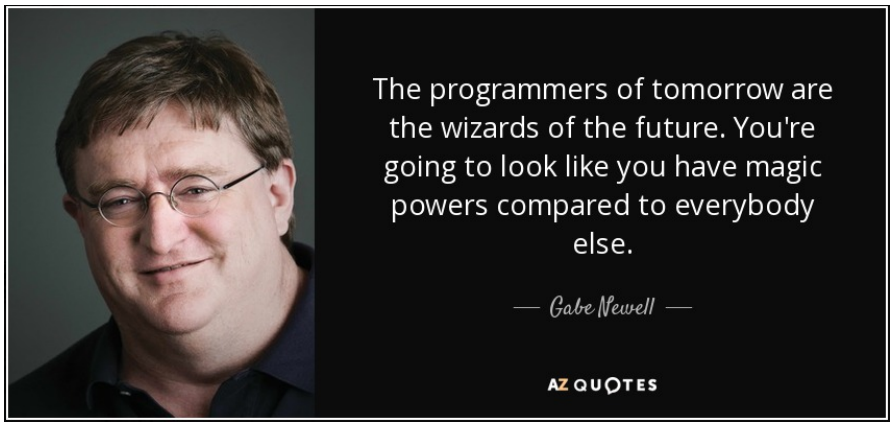
Figuur 3 – Linux/Git



Figuur 4 - Belangrijkste man voor natuurkunde na Newton en Einstein

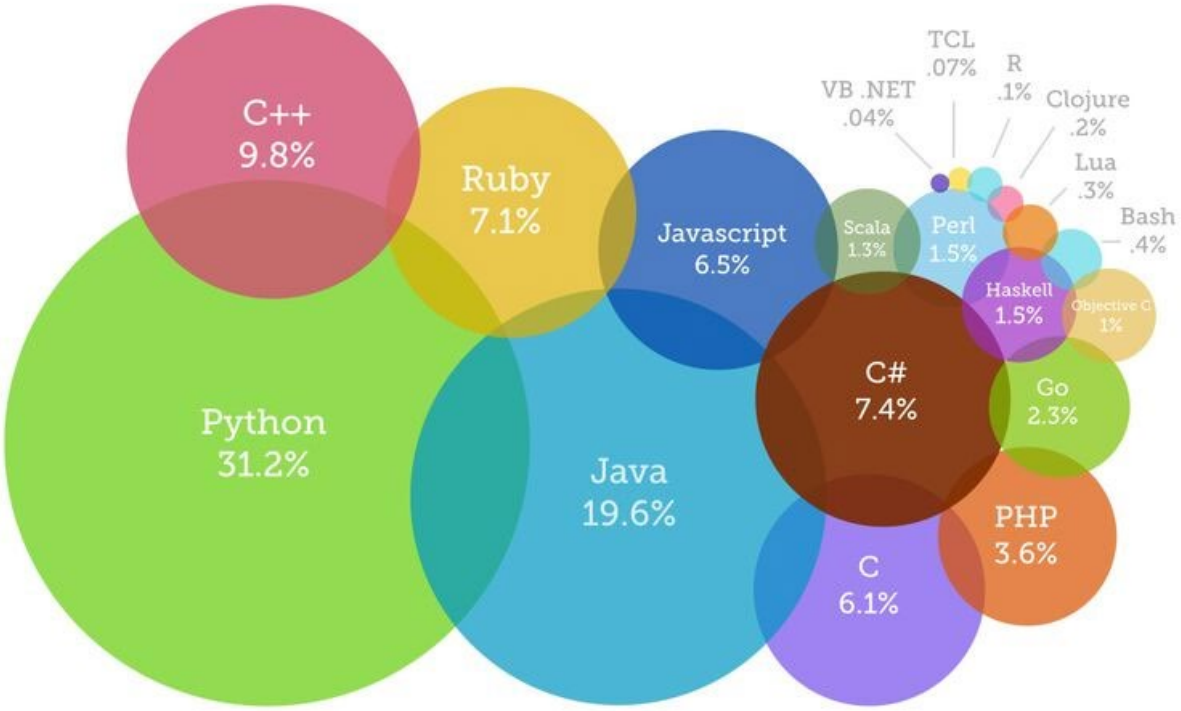


Figuur 5 - Facebook



Figuur 6 - Vale/Steam

In dit practicum leer maak je voor het eerst kennis met Python. Deze programmeertaal is momenteel wereldwijd de meest gewaardeerde taal als gevolg van de veelzijdigheid, eenvoud van programmeren en de leesbaarheid van de syntax (taal). Google, Nasa, Amazon, Netflix, Facebook, Instagram en Dropbox…. Ze draaien allemaal op Python.



Figuur 7 - Populariteit programmeer talen 2019

Daarnaast is Python ook een Nederlandse trots! Python is namelijk als een hobbyproject in 1989 ontwikkeld door de Amsterdammer Guido van Rossum omdat hij de bestaande andere programmeertalen onnodig moeilijk vond. Het idee achter python was het volgende:

- Krachtige maar simpele en intuïtieve programmeertaal
- Open Source (gratis voor iedereen)
- Code die te lezen is net zoals Engels
- Geschikt voor iedereen van ingenieur, tot wiskundige tot econoom.

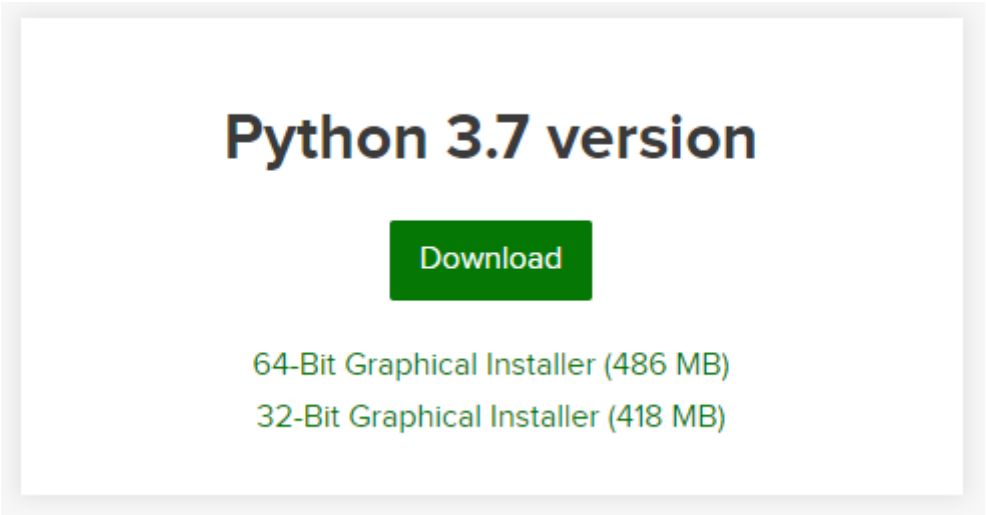
Python is vernoemd naar de favoriete film van Guido, een cult comedy uit de jaren negentig:

Monthy Python and the quest for the holy grail.

Figuur 8 - Guido van Rossum

Python 3, Anaconda en Spyder

Jullie gaan leren programmeren in de derde versie van Python, Python 3. Echter is Python op zich niet erg fijn om mee te werken, het is een taal, geen programma. Om makkelijk te programmeren werken we met een zogenaamde Integrated Development Environment (IDE), of in het Nederlands een 'geïntegreerde ontwikkel omgeving'. Zo'n IDE is een programma waarin Python wordt samengevoegd met een aantal handige bibliotheken die bijvoorbeeld je code op typfouten controleren of het makkelijk maken om grafieken te tekenen (matplotlib), te rekenen (Scipy) of met matrixen te werken (Numpy). Al deze handige tools zijn samengevoegd in een platform genaamd Anaconda. Laten we beginnen met onze reis en eerst eens Anaconda downloaden, let op dat je wel je besturingssysteem selecteert, respectievelijk; Windows, Mac of Linux.



Figuur 9 - Python 3 download scherm

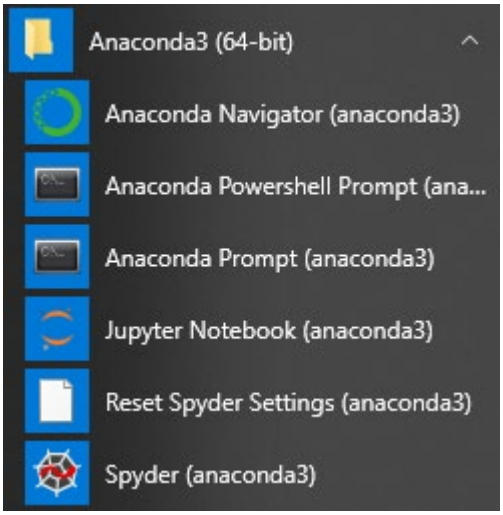
Als je Anaconda hebt gedownload installeer je deze en verschijnt na het installeren de Anaconda map in je start menu. In dit menu staan verschillende snelkoppelingen.

Anaconda navigator is het verzamelbestand van alles wat mogelijk is binnen Anaconda.

Anaconda Powershell en Prompt zijn mogelijkheden om extra plug-ins te installeren via een command-line-tool genaamd 'conda'.

Jupyter is een Python web-applicatie waarbij code wordt gecombineerd met tekst. Een voorbeeld is hier te vinden.

Spyder is de IDE voor data wetenschappers om berekeningen in te maken. Met deze IDE gaan wij werken in de komende jaren. Een introductie van Spyder vind je hier.



Figuur 10 - Windows start menu

Kom je er niet uit? Check deze video!

Opdracht

De werkelijke kracht van programmeren ten opzichte van een handberekening is dat je heel makkelijk waarden kan aanpassen en dan meteen de berekening weer kunt uitvoeren met één druk op de knop in plaats van alles weer opnieuw in je rekenmachine te moeten toetsen. Ten opzichte van Excel is Python veel makkelijker om je code te onderzoeken, je hoeft bijvoorbeeld in python niet eerst op een cell te klikken om je formule te inspecteren.

Binnen deze opdracht ga je kennis maken met de voordelen van Python doormiddel van het maken van een *parametrisch rekenmodel*. In zo'n parametrisch model reken je niet met waarden (4, 18, 12) maar met parameters (x, y, a). Op deze manier kan je door de parameter aan te passen de hele berekening automatisch aanpassen.

In appendix A staan de verschillende parametrische opdrachten A t/m J weergegeven. Voor dit practicum ga jij een parametrisch rekenmodel maken van de gegeven opdracht. Welk vraagstuk jij gaat beantwoorden wordt bepaald door het laatste cijfer van je studentnummer:

Laatste getal student nummer	0	1	2	3	4	5	6	7	8	9
Opgave	A	B	C	D	E	F	G	H	I	J

N.B.: Als je de opdracht niet hebt gehaald voor de 1^{e} kans, pak dan voor de herkansing de andere verbinding. Als je voor kans 1 een oplegging hebt dan heb je dus voor de herkansing een inklemming. Beide met dezelfde belasting.

In iedere afbeelding staan de parameters die je moet verwerken in je model. Maak van de opdracht een Python script dat het volgende kan:

- Stap 1: Korte uitleg over de som
- Stap 1: Vraag om parameters in te voeren
- Stap 2: Berekenen van reactiekrachten en momenten dmv een matrix met Numpy
- Stap 3: Berekenen van dwarskrachtlijn
- Stap 3: Berekenen van een momentlijn dmv integreren met Scipy
- Stap 4: Plotten van dwarskrachtlijn dmv matplotlib
- Stap 4: Plotten van momentlijn dmv matplotlib
- Stap 5: Plotten van VLS + dwarskrachtlijn + momentlijn dmv matplotlib

Zo'n complexe opdracht kan je het beste aanpakken door in stappen te werken. Één van de grootste fouten die je kan maken is door meteen te beginnen met programmeren, dan raak je echt verloren. (1) Begin daarom eerst met een handberekening te maken van je vraagstuk. Hierdoor krijg je inzicht in welke stappen je deze opdracht kan uitvoeren. (2) Zodra de berekening bekend is schrijf je je berekening om in 'pseudo code', probeer je code eerst eens op papier uit te schrijven in stappen die je los kan uitvoeren. (3) Zodra je weet wat de structuur van je script wordt kan he beginnen met het programmeren van je code in Spyder. (4) Natuurlijk lukt het je de eerste keer niet om de code perfect te schijven. Werk daarom in kleine stukjes die je los van elkaar kan controleren in plaats van het meteen schrijven van het gehele bestand.

Beoordeling

Voor de opdracht kan je een voldoende of onvoldoende behalen. Je haalt een voldoende voor de opdracht als het correcte resultaat uit de voor jou geselecteerde opdracht komt. De opdracht is correct als na het invoeren van de gevraagde informatie de juiste NVM-lijnen worden getoond in de gecombineerde afbeelding. Let dus op dat ook de statica juist is!

Beter goed gejat…

Nu zijn jullie natuurlijk slim genoeg om voor deze opdracht samen te werken door elkaars code de kopiëren. Heel goed! Een goede programmeur is namelijk niet eentje die zelf zijn code schrijft maar eentje die zijn code slim steelt van het internet. Een veel gebruikt voorbeeld is de volgende:

Figuur 11 - Programmeren: Beter goed gejat dan slecht bedacht.

Dus, werk samen, steel elkaars code, steel de docent zijn code (Appendix B en C) en pluk het internet helemaal leeg!

Succes!

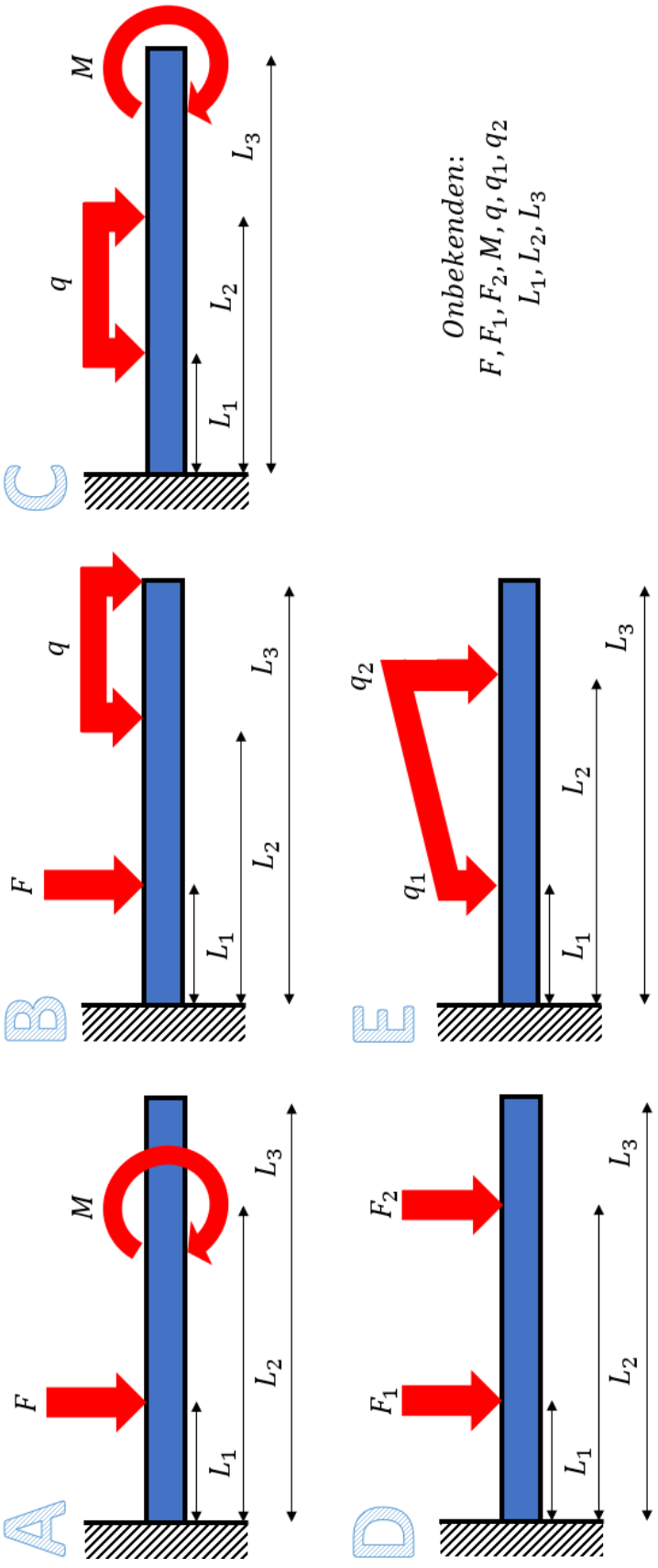
Tot slot, vind je het lastig om zo te beginnen? Het internet zit vol met handige tutorials, video's, websites en boeken. Een aantal hints om je alvast op gang te brengen:

- Google
- Youtube
- Boek
- Edx: Python for data science
- Edx: Essential math for Machine learning: Python Edition
- LearnPython.com
- SciPy – Getting Started
- Numpy – Getting Started
- Matplotlib – Getting Started

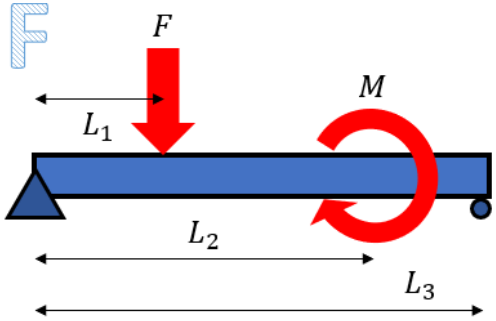
Zoals gezegd, de kern van programmeren is goed gebruik kunnen maken van google en van voorbeelden op het internet leren. Maak gebruik van de kracht van het internet! (En zoek in het engels!)

Figuur 12 - Het binaire leven van een Programmeur

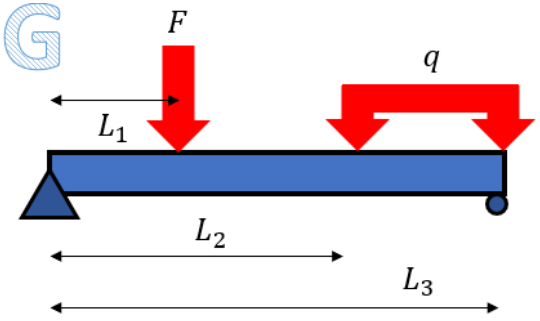
Appendix A - Opdracht



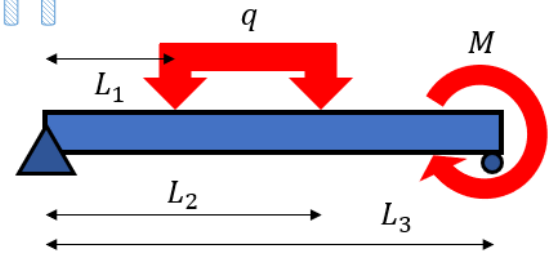
F



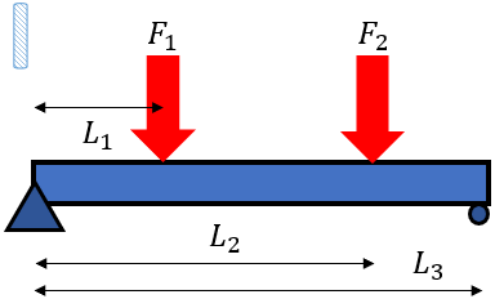
G



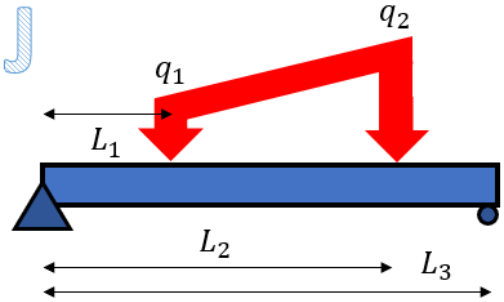
H



I



J



Onbekenden:

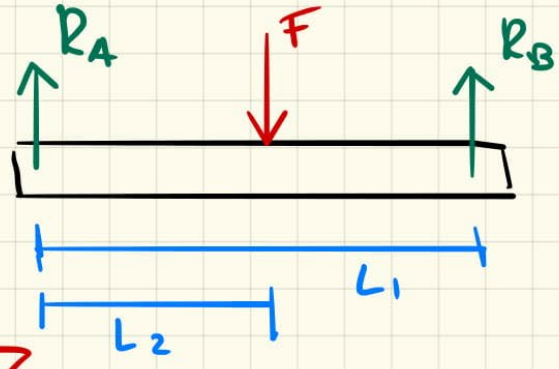
$F, F_1, F_2, M, q, q_1, q_2$

L_1, L_2, L_3

Appendix B - Handberekening

Wiskunde 3 - Practicum

Opgelegde balk



Variabelen:

- $F = ?$
- $L_1 = ?$
- $L_2 = ?$

Parametrisch oplossen.

Stap 1: Reactie krachten bepalen

$\Sigma F_y = 0 \rightarrow R_A + R_B - F = 0$

$\Sigma M_A = 0 \rightarrow -F \cdot L_2 + R_B \cdot L_1 = 0$

$\rightarrow R_B = F \cdot \frac{L_2}{L_1}$

$\rightarrow R_A = F - R_B = F \cdot (1 - \frac{L_2}{L_1})$

optie 1: Balans

Matrix opstellen

optie 2: Matrix

$1 \cdot R_A + 1 \cdot R_B = F \cdot 1$

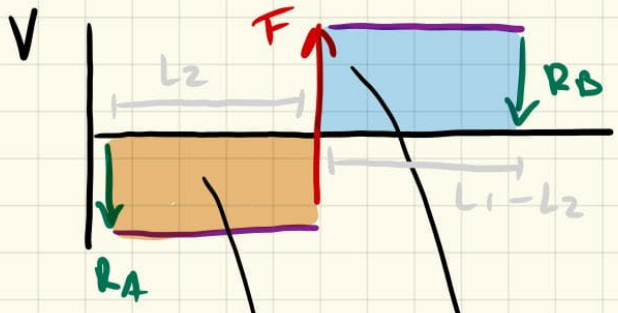
$0 \cdot R_A + L_1 \cdot R_B = F \cdot L_2$

$\rightarrow \left[\begin{array}{cc|c} 1 & 1 & F \\ 0 & L_1 & F \cdot L_2 \end{array}\right]$

deze zoeken we.

Resultaat: $R_A = (\ldots)$, $R_B = (\ldots)$

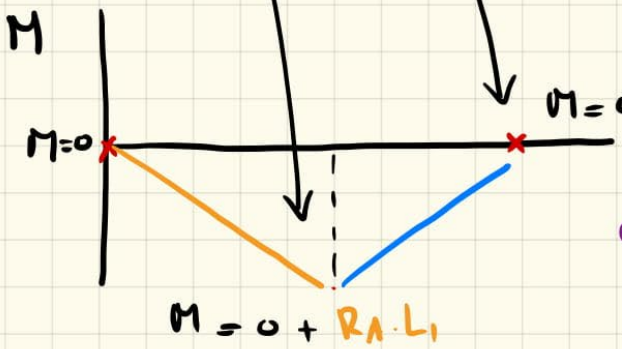
Stap 2: Berekenen V en M lijn



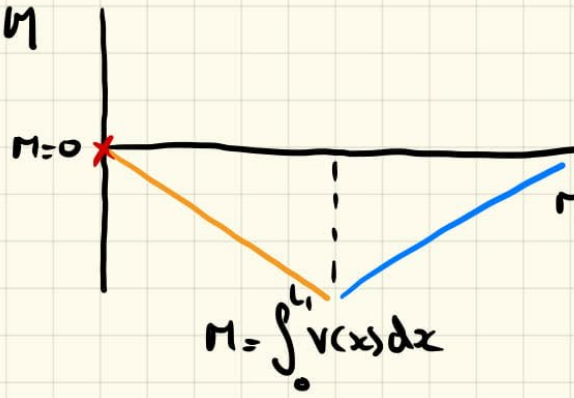
actie = -reactie

$V(x) = -R_A \quad 0 \geq x > L_2$

$V(x) = R_B \quad L_2 \geq x > L_1-L_2$



optie 1: oppervlakte methode



optie 2: Integreren

Appendix C - Script

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 26 11:54:12 2019

@author: Johan Antonissen
"""

'''''
Stap 1: Vraag om waarden

'''
print('Stap 1: Waarden invoeren')
#laten we beginnen met een tekening van het vraagstuk
print('Hieronder is een schets weergegeven van de som wie we willen oplossen:')
print('')
print('/^\ Ra        | F              /^\ Rb')
print(' |            |                 | ')
print(' |           \./                | ')
print('=A==============================B=')
print(' |--> x')
print(' |---------------------L1-------| ')
print(' |-----L2-----|                  ')

#We vragen dr kracht F aan de gebruiker en maken er een float van om er mee te kunn
en rekenen
F=float(input('Hoeveel is de kracht (F kN)? '))

#We vragen lengte L1, de totale lengte van de balk en maken er een float van om er
mee te kunnen rekenen
L1=float(input('Hoe lang is de balk (L1 m)? '))

#We vragen de afstanf van de kracht tot het beginpunt van de balk
L2=float(input('Op welke afstand van links is de balk belast op kracht (L2 m)? '))


#Nu mag natuurlijk de L2 niet langer zijn dan L1, anders valt de kracht buiten de b
alk. Dit moeten we dus controleren.
#We gebruiken hier een 'while' statement om de vraag net zo lang te blijven vragen
tot onze waarde kloppen.
#Dit is een zogenaamde 'catch functie'
while L2 >= L1:
    print('Kracht L2 moet wel binnen het bereik van L1 liggen he!')
    L2=float(input('Op welke afstand van links is de balk belast op kracht (L2 m)?
'))

print()
'''''
Stap 2: bereken de reactiekrachten

'''
print('Stap 2: Matrix rekenen')
#Matrix en vector berekeningen maken we met numpy, laten we eerst dus eens maar num
py erbij halen
import numpy as np
#https://docs.scipy.org/doc/numpy/index.html

#Nu gaan we onze matrix in numpy zetten, we stellen eerst onze krachtbalansen op
#In onze matrix staat:   [[1*Ra+1*Rb],
#                        [0*Ra+L1*Rb]]
matrix_A=np.array([[1,1],[0,L1]])
print('matrix A \n',matrix_A)

```

```
#In onze vector staat:  [F,
#                        F*L2]

#Ofwel:                 [1, 1] | [F]
#                       [0,L1] | [F*L2]
vector_b=np.array([F,F*L2])
print('vector b \n',vector_b)

#Nu lossen we de matrix op met lineare algebra:
#https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html
R=np.linalg.solve(matrix_A,vector_b)

#Het resultaat is onze vector R waarin Ra en Rb zijn verstopt
print('Oplossende vector voor de matrix \n',R)
Ra=R[0]
Rb=R[1]
print('Ofwel: Ra =',Ra,' en Rb =', Rb)

print()
'''''
Stap 3: opstellenfuncties nvm lijnen

'''
print('Stap 3: Opstellen functies voor inwendige belasting')
#Functies manipuleren doen we met de scipy (science python) module, laten we deze e
erst importeren
#In tegenstelling tot numpy is scipy een verzameling van modulen,
#We moeten dus eerst een module  uit scipy importeren voordat we er een naampje aan
 kunnen geven
from scipy import integrate as i
#https://docs.scipy.org/doc/scipy/reference/

#We beginnen met het opstellen van onze dwarskrachtlijn.
#Dit is een discontinue functie dus we moeten deze in meerdere stappen beschrijven.


def dwarskrachtlijn(x,F,Ra,Rb,L1,L2):
    #Als (if) x kleiner is dan 0 dan moet er 0 uit de functie komen
    if x < 0:
        V=0.
    #Als dat niet zo is en de x is kleiner dan L2 (elif = else if) dan moet er -
Ra uit de functie komen
    #Ofwel 0 < x < L2
    elif x < L2:
        V=-Ra
    #Als dat niet zo is en de x is kleiner dan L1 (elif = else if) dan moet er Rb u
it de functie komen
    #Ofwel L2 < x < L1
    elif x < L1:
            V=Rb
        #Voor alle andere waarden (bijvoorbeeld x > L2) moet er natuurlijk weer
0 uit komen
        else:
            V=0.
        return V

    #De definitie (ofwel def = functie) beschrijven we met de variabele x, daarn
a zetten we de constanten neer waarmee we rekenen
    def momentlijn(x,F,Ra,Rb,L1,L2):
        #Nu weten we dat het integraal van de dwarskrachtlijn het moment is, lat
en we dus de dwarskrachtlijn integreren
        #Als (if) x kleiner is dan 0 dan moet er 0 uit de functie komen
        if x < 0:
            M=0.
        #Als x tussen 0 en L1 ligt moet er M uitkomen:
        elif x < L1:
```

```
                #Voor het integreren van een enkele integraal gebruiken we de scipy quad functie
                #https://docs.scipy.org/doc/scipy/reference/integrate.html
                #We sturen hier naar quad onze functie voor de dwarskrachtlijn, de eerste integratiegrens (0) en de 2e grens (x)
                #We nemen hier als grens x in plaats van L2 omdat we dan ieder punt tussen 0 en L2 kunnen berekenen ipva alleen L2.
                #De niet-variabelen moeten we doorsturen als zogenaamde arguments (args) zoals beschreven in de documentatie.
                #Ofwel M=scipy.integrate.quad(functie voor moment, linkergrens, rechtergrens, args=(constanten))
                M=i.quad(dwarskrachtlijn,0,x,args=(F,Ra,Rb,L1,L2))
                M=M[0]
            else:
                M=0.
            return M

        print()
        '''''
        Stap 4: Berekenen nvm lijnen

        '''
        print('Stap 4: NVM lijnen berekenen')
        #Nu we de functies hebben kunnen we deze berekenen

        #We maken eerst een lijstje waar we onze waarden kunnen opslaan
        DwarskrachtX=[]
        DwarskrachtY=[]
        MomentX=[]
        MomentY=[]

        #laten we in totaal 100 stappen nemen dan is het resultaat erg nouwkeurig.
        #We berekenen dus de functies tussen x=0 en x=L2 in een totaal van 100 stappen
        stappen=100
        #We tellen van 0 tot 100
        for stap in range(stappen):
            #We laten x van 0 naar L1 lopen per stap met in een totaal van 100 stappen
            x=L1*stap/stappen
            #We voegen x toe aan onze lijstjes
            DwarskrachtX.append(x)
            MomentX.append(x)

            #We berekenen de dwarskracht:
            #We roepen de functie op die we eerst hebben geschreven en sturen daar al onze waarden heen.
            V=dwarskrachtlijn(x,F,Ra,Rb,L1,L2)
            #En voegen V toe aan ons lijstje
            DwarskrachtY.append(V)

            #We berekenen nu het moment:
            #We roepen de functie op die we eerst hebben geschreven en sturen daar al onze waarden heen.
            M=momentlijn(x,F,Ra,Rb,L1,L2)
            #En voegen M toe aan ons lijstje
            MomentY.append(M)
            #En we herhalen dit nog een keer

        #Laten we eens kijken hoe onze data eruit ziet
        print('Dwarskracht:')
        print('X=',DwarskrachtX)
        print('Y=',DwarskrachtY)

        print('Moment:')
```

```
170.        print('X=',MomentX)
171.        print('Y=',MomentY)
172.
173.        print()
174.        '''''
175.        Stap 5: Tekenen nvm lijnen
176.
177.        '''
178.        print('Stap 5: NVM lijnen tekenen')
179.        #Om echt te weten of we onze data goed hebben berekend moeten we deze natuur
    lijk visualiser
180.
181.        #We beginnen met het importeren van onze matplotlib module
182.        from  matplotlib import pyplot as plt
183.        #https://matplotlib.org/3.1.1/users/index.html
184.
185.        #Nu tekenen we onze dwarskrachtlijn
186.        plt.plot(DwarskrachtX,DwarskrachtY,color='red')
187.        #We voegen een titel toe
188.        plt.title('Dwarskracht')
189.        #We beschrijven de x-as
190.        plt.xlabel('Lengte balk X (m)')
191.        #We beschrijven de y-as
192.        plt.ylabel('Kracht V (kN)')
193.        #We voegen een rooster toe om dit beter af te lezen
194.        plt.grid()
195.        #We voegen een horizontale x-
    as toe voor de duidelijkheid met een kleur zwart
196.        plt.axhline(color='black')
197.        #We slaan de plot op als een plaatje
198.        plt.savefig('Dwarskracht.png', bbox_inches='tight')
199.        #En we laten het plaatje ook even zien in de console (rechts)
200.        plt.show()
201.        #We maken de plot weer leeg voor de volgende tekening
202.        plt.clf()
203.
204.        #Nu tekenen we onze momentlijn
205.        #Laten we het maximale moment hierin aangeven
206.        #Dat moeten we eerst uitzoeken waar dat dan is
207.        #Het maximale moment is simpelweg de maximale waarde in de maxY lijst
208.        #Let wel we moet het absolute maximum nemen anders komt er 0 uit en niet ons
     gezochte maximum!
209.        maxY=max(MomentY,key=abs)
210.        #Voor de x-
    locatie van het maximum moment moetten we de locatie in de lijst vinden dit doen we
     met de index functie
211.        lijst_index=MomentY.index(maxY)
212.        #Vervolgens pakken we de die specifieke locatie in de lijst met onze x-
    waarden
213.        maxX=MomentX[lijst_index]
214.        #Ofwel in totaal:
215.        print('Maxima moment =',maxX,maxY)
216.        #Nu plotten we de reguliere functie en geven hieraan het label 'Moment'
217.        plt.plot(MomentX,MomentY,color='green',label='Moment')
218.        #We plotten hierin het maximum met een rode punt 'ro' en het label 'Maximum'

219.        plt.plot(maxX,maxY,'ro',label='Maximum')
220.        #En we plaatsen de legenda in de grafiek
221.        plt.legend()
222.        #We voegen een titel toe
223.        plt.title('Moment')
224.        #We beschrijven de x-as
225.        plt.xlabel('Lengte balk X (m)')
226.        #We beschrijven de y-as
227.        plt.ylabel('Moment M (kNm)')
228.        #We voegen een rooster toe om dit beter af te lezen
```

```
229.        plt.grid()
230.        #We voegen een horizontale x-
    as toe voor de duidelijkheid met een kleur zwart (k)
231.        plt.axhline(color='black')
232.        #We slaan de plot op als een plaatje
233.        plt.savefig('Moment.png', bbox_inches='tight')
234.        #En we laten het plaatje ook even zien in de console (rechts)
235.        plt.show()
236.        #We maken de plot weer leeg voor de volgende tekening
237.        plt.clf()
238.
239.        print()
240.        '''''
241.        Stap 6: Alles samenvoegen
242.
243.        '''
244.        print('Stap 6: Samengestelde tekening')
245.        #Nu is het natuurlijk ook leuk om alle data in 1 tekening te setten samen me
    t een schets van het VLS.
246.        #We creeeren een nieuw figuur met herin 3 plots, links een schets van het VL
    S en rechts de V em M plot.
247.        fig = plt.figure()
248.        #We defineren de assen (ax) waar we mee tekenen
249.        ax = fig.gca()
250.        #En beschrijven de grootte van het figuur dat we als uitkomst willen hebben
    (breedte 8 inch, hoogte 4 inch)
251.        fig.set_size_inches(8, 4)
252.        #Nu bouwen we de 3 deelplots die we willen maken van 2 rijen en 2 kolommen
253.        ax1 = plt.subplot2grid((2, 2), (0, 0), rowspan = 2)     #Plaatje begit links
    boven op rij 0 kolom 0 en leeft op 2 rijen
254.        ax2 = plt.subplot2grid((2, 2), (0, 1))                  #V lijn begint rij 0
     kolom 1
255.        ax3 = plt.subplot2grid((2, 2), (1, 1))                  #M lijn begint rij 1
     kolom 1
256.
257.        # We beschrijven de data van onze VLS tekening
258.        ax1.set_xlabel('x (globaal)',fontsize=10)
259.        ax1.set_ylabel('y (globaal)',fontsize=10)
260.        #We voegen een rooster (tick) toe (De streepjes op de x- en y- as)
261.        ax1.set_yticklabels([])
262.        ax1.set_xticklabels([])
263.        #En beschrijven de limieten van onze tekening
264.        #We maken het plaatje iets groter dan L1 x L1 door de 1.1 factor
265.        ax1.set_xlim(.1*-L1,1.1*L1)
266.        ax1.set_ylim((1.1*-L1/2.,1.1*L1/2.))
267.
268.        # We beschrijven de data van onze dwars en moment lijn
269.        ax2.set_ylabel('Dwars [kN]',fontsize=10)
270.        ax2.axhline(color='black')
271.        ax2.set_yticklabels([])
272.        ax2.set_xticklabels([])
273.        ax2.set_ylim(1.1*max(DwarskrachtY,key=abs),-1.1*max(DwarskrachtY,key=abs))
274.        ax2.set_xlim(0,L1)
275.
276.        ax3.set_ylabel('Moment [kNm]',fontsize=10)
277.        ax3.axhline(color='black')
278.        ax3.set_yticklabels([])
279.        ax3.set_xticklabels([])
280.        ax3.set_ylim(-abs(max(MomentY,key=abs)),abs(max(MomentY,key=abs)))
281.        ax3.set_xlim(0,L1)
282.
283.        #Voor het VLS willen we een vierkant tekenen in onze plot, hiervoor gebruike
    n we de patches module
284.        import matplotlib.patches as patches
285.        #https://matplotlib.org/3.1.1/api/patches_api.html
286.
```

```
287.        breedte=L1
288.        hoogte=L1*.1
289.
290.        #We teken de balk en geven hier eerst het punt linksonder aan vanaf waar we
    tekenen en dan de breedte en hoogte
291.        balk=patches.Rectangle((0,-
    hoogte/2.),breedte,hoogte,fill=True, color='grey',ec='black')
292.        ax1.add_patch(balk)
293.        #Nu tekenen we onze drie pijlen op het figuur
294.        #Let op we moeten hier de pijlen parametrisch opzetten anders schalen ze hee
    l raar met de afmetingen van de balk
295.        krachtF=patches.Arrow(L2,0,0,-
    F/F*breedte/3.,color='red',width=hoogte/2.,ec='black')
296.        krachtRa=patches.Arrow(0,0,0,Ra/F*breedte/3.,color='green',width=hoogte/2.,e
    c='black')
297.        krachtRb=patches.Arrow(L1,0,0,Rb/F*breedte/3.,color='green',width=hoogte/2.,
    ec='black')
298.        ax1.add_patch(krachtF)
299.        ax1.add_patch(krachtRa)
300.        ax1.add_patch(krachtRb)
301.
302.        #Nu gaan we onze plots tekenen, laten we beginnen met wat we al weten, de dw
    ars en moment lijn
303.        ax2.plot(DwarskrachtX,DwarskrachtY,color='red')
304.        ax3.plot(MomentX,MomentY,color='green')
305.
306.        plt.savefig('Combi.png', bbox_inches='tight')
307.        plt.show()
308.        plt.clf()
```