

Bijlage 2: DNA mutaties wiskundig berekend

1 Wiskundig gedeelte

Zoals we in de bookwidgets hebben gezien is DNA een belangrijke factor om informatie over te dragen naar volgende generaties. Tijdens de replicatie van DNA kunnen er echter foutjes optreden bij het dupliceren van de DNA-streng, wat ook **mutaties** worden genoemd. Deze mutaties zullen een wijziging in de verdeling van de nucleotiden van de DNA-streng veroorzaken. In dit project zullen we op een wiskundige manier het effect bestuderen van mutaties op de verdeling van de nucleotiden voor toekomstige generaties.

Om te starten gebruiken we de volgende twee DNA-strengen:

- S_0 : ACTTGTCGGATGATCAGCGGTCCATGCACCTGACAACGGT
- S_1 : ACATGTTGCTTGACGACAGGTCCATGCGCCTGAGAACGGC

Hierbij gebruiken we de notatie S_0 om aan te geven dat deze streng komt van een individu uit generatie 0 en S_1 voor een individu uit generatie 1. Analoog kunnen we de notatie S_i op een algemene manier gebruiken voor de DNA-streng van generatie i .

Tel het aantal A , G , C , T - nucleotiden in de DNA-streng voor de S_0 -generatie. Doe dit ook voor de S_1 -generatie.

Wat valt er op als je de frequenties bekijkt van de twee DNA-strengen?

Wat voor type mutatie heeft hier plaatsgevonden? Wees hierbij zo specifiek mogelijk en leg uit!

Deze veranderingen kunnen we duidelijker voorstellen aan de hand van Tabel 1. In deze tabel wordt er een overzicht gegeven van het behoud en de verandering van de nucleotiden in de twee DNA-strengen. De kolommen geven hierbij de frequenties weer van de S_0 -generatie en de rijen de S_1 -generatie. De nucleotiden die worden behouden tussen de twee generaties staan op de hoofddiagonaal van de tabel. De nucleotiden die veranderd zijn, staan bijgevolg niet op deze hoofddiagonaal.

Er werd reeds een aanzet gegeven om de tabel in te vullen. Vul nu zelf de tabel verder aan.

- 7: 7 A-nucleotiden uit de S_0 -generatie zijn behouden in de S_1 -generatie.
- 1: 1 A-nucleotide uit de S_0 -generatie is veranderd naar een G-nucleotide in de S_1 -generatie.

S_0 en S_1		A	S_0 G	C	T
S_1	A	7			
	G	1			
	C				
	T				
Totaal					

Tabel 1: Gezamenlijke frequentietabel nucleotiden in S_0 en S_1 -generatie.

Bereken de kansverdeling voor de A-, G-, C- en T-nucleotiden voor de S_0 -generatie en noem deze kansen p_A, p_G, p_C en p_T respectievelijk. Deze kansen kunnen we vervolgens invullen in onderstaande kolommatrix $\mathbf{p}_0$. Doe dit vervolgens ook voor de nucleotiden in de S_1 -generatie en noem deze kolommatrix $\mathbf{p}_1$

$$\mathbf{p}_0 = \begin{bmatrix} p_A \\ p_G \\ p_C \\ p_T \end{bmatrix} = \begin{bmatrix} \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \end{bmatrix} \quad \text{en} \quad \mathbf{p}_1 = \begin{bmatrix} p_A \\ p_G \\ p_C \\ p_T \end{bmatrix} = \begin{bmatrix} \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \end{bmatrix}$$

Deze kolommatrix $\mathbf{p}_0$ noemen we de **basenverdeling van de S_0 -generatie** en geeft de verdeling weer van de nucleotiden in de S_0 -generatie. Analoog is $\mathbf{p}_1$ de basenverdeling van de S_1 -generatie en kunnen we algemeen $\mathbf{p}_i$ definiëren als de basenverdeling van de S_i -generatie.

Veranderingen in het DNA tussen opeenvolgende generaties zijn echter relatief zeldzaam. De verdeling van de nucleotiden in de S_1 -generatie, genoteerd $\mathbf{p}_1$, hangt bijgevolg in grote mate af van de basenverdeling in de vorige S_0 -generatie!

Vanuit de frequenties in Tabel 1 kunnen we de **voorwaardelijke kansverdeling van $\mathbf{p}_1$** berekenen. In de Bookwidget over kansen hebben we de definitie gezien voor een voorwaardelijke kans bij twee gebeurtenissen A en B : $P(B|A) = \frac{P(A \text{ en } B)}{P(A)}$.

Deze definitie kunnen we ook schrijven in termen van frequenties: $P(B|A) = \frac{\#(A \text{ en } B)}{\#(A)}$.

Dit kunnen we ook vertalen naar Tabel 2 door de gezamenlijke frequenties te delen door de overeenkomstige kolomtotalen.

Opnieuw werd er reeds een aanzet gegeven om Tabel 2 in te vullen. Vul nu zelf de tabel verder aan.

Tip: Laat de uitkomst in breukvorm staan, zodat je bij de toekomstige bewerkingen nauwkeurigere resultaten krijgt.

- $P(S_1 = A|S_0 = A) = \frac{\#(S_1 = A \text{ en } S_0 = A)}{\#(S_0 = A)} = \frac{7}{9}$.
- $P(S_1 = G|S_0 = A) = \frac{\#(S_1 = G \text{ en } S_0 = A)}{\#(S_0 = A)} = \frac{1}{9}$.

$S_0 S_1$		S_0			
		A	G	C	T
S_1	A	$\frac{7}{9}$			
	G	$\frac{1}{9}$			
	C				
	T				
Totaal					

Tabel 2: Voorwaardelijke kansverdeling van de nucleotiden in de S_1 -generatie, gegeven de nucleotiden van de S_0 -generatie.

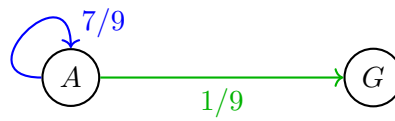
Op deze manier krijgen we een 4×4 -matrix met voorwaardelijke kansen voor de nucleotiden van de S_1 -generatie, conditioneel op de nucleotiden van de gekende nucleotiden in de S_0 -generatie. Deze overgangsmatrix, die we in dit project noteren als M , noemen we ook een **Markov matrix**.

DEFINITIE: Een matrix M is een **Markov matrix** als volgende voorwaarden gelden:

- M is een vierkante matrix.
- De elementen van M zijn positief.
- De som van de elementen in iedere kolom is gelijk aan 1 of de som van de elementen in elke rij is gelijk aan 1.

Ga na of de matrix in Tabel 2 een Markov matrix is.

Geef het bijbehorende pijlenschema, horende bij matrix M .



Met behulp van dit pijlenschema kunnen we visueel de mutatiekansen uit de Markov matrix (= overgangsmatrix) M voorstellen tussen twee opeenvolgende generaties. Hieruit kunnen we de basenverdeling $\mathbf{p}_1$ voor de S_1 -generatie berekenen door de matrix M te vermenigvuldigen met de basenverdeling $\mathbf{p}_0$ van de S_0 -generatie:

$$\mathbf{p}_1 = \begin{bmatrix} P(S_1 = A) \\ P(S_1 = G) \\ P(S_1 = C) \\ P(S_1 = T) \end{bmatrix} = \begin{bmatrix} P_{A|A} & P_{A|G} & P_{A|C} & P_{A|T} \\ P_{G|A} & P_{G|G} & P_{G|C} & P_{G|T} \\ P_{C|A} & P_{C|G} & P_{C|C} & P_{C|T} \\ P_{T|A} & P_{T|G} & P_{T|C} & P_{T|T} \end{bmatrix} \cdot \begin{bmatrix} P_A \\ P_G \\ P_C \\ P_T \end{bmatrix} = M \cdot \mathbf{p}_0$$

Ga dit voor ons voorbeeld na door de juiste matrixvermenigvuldiging uit te voeren.

Indien je mag veronderstellen dat deze mutatiekansen constant blijven voor alle volgende generaties, bereken de kansverdeling $\mathbf{p}_2$ voor de S_2 -generatie. Doe dit op 2 manieren:

1. Door verder te rekenen vanuit de $\mathbf{p}_1$ -kansverdeling.
2. Door te beginnen vanuit de $\mathbf{p}_0$ -kansverdeling.

Schrijf telkens de matrixvermenigvuldiging op en gebruik je grafisch rekenmachine om de bewerkingen uit te voeren!

Bereken de kansverdeling $\mathbf{p}_3$ voor de S_3 -generatie, door gebruik te maken van de kansverdeling van $\mathbf{p}_0$ in de berekening. Schrijf ook hier matrixvermenigvuldiging op en gebruik je grafisch rekenmachine om de bewerkingen uit te voeren!

We kunnen dit verder veralgemenen naar een algemene formule voor de basenverdeling $\mathbf{p}_n$ voor de S_n -generatie. Hoe kunnen we $\mathbf{p}_n$ uitdrukken in functie van de basenverdeling $\mathbf{p}_0$?

Gebruik bovenstaande formule om de basenverdeling $\mathbf{p}_{200}$ te berekenen voor de S_{200} -generatie.

*Kun je met het rekenmachine ook **P500** berekenen?*

Indien de macht van de matrix M te hoog wordt, zal het grafisch rekenmachine moeilijkheden hebben om de basenverdeling te berekenen. Om deze computationele moeilijkheden te omzeilen kunnen we ook gebruik maken van wiskundige software. In het volgend hoofdstuk onderzoeken we hoe dit precies gebeurt met het programma **Python**.

2 DNA mutaties in Python berekenen

In het vorige deel hebben jullie het aantal nucleotiden in een DNA-sequentie met de hand geteld. Dit lukt nog net als je DNA-streng niet te lang is. Bij een langere DNA-streng is niet enkel tijd een probleem, maar de kans op fouten wordt ook steeds groter. In plaats van dit manueel te berekenen, zullen we dit proces in Python implementeren.

2.1 Python gebruiken op je computer

Om Python te kunnen gebruiken moet je eerst Python zelf op je computer installeren en nadien een programma waarmee je Python kan gebruiken. Wij zullen het programma PyCharm CE gebruiken.

2.1.1 Installeren van Python

Voer de volgende stappen uit:

- 1) Ga naar de officiële website van Python: <https://www.python.org/downloads/>.
- 2) Klik op de gele knop ‘Download Python [versienummer]’.
- 3) Open het gedownloade bestand (python-[versienummer]-amd64.exe).
- 4) Zorg ervoor dat je ‘Add python.exe to PATH’ aanvinkt.
- 5) Klik op ‘Install Now’ en wacht tot de installatie is voltooid.
- 6) Klik op ‘Close’ als de installatie klaar is.

2.1.2 Installeren en instellen van PyCharm Community Edition (CE)

PyCharm CE is een gratis en open-source code-editor speciaal ontworpen voor het programmeren in Python. Het helpt programmeurs met handige functies zoals automatisch aanvullen, foutdetectie en debugging.

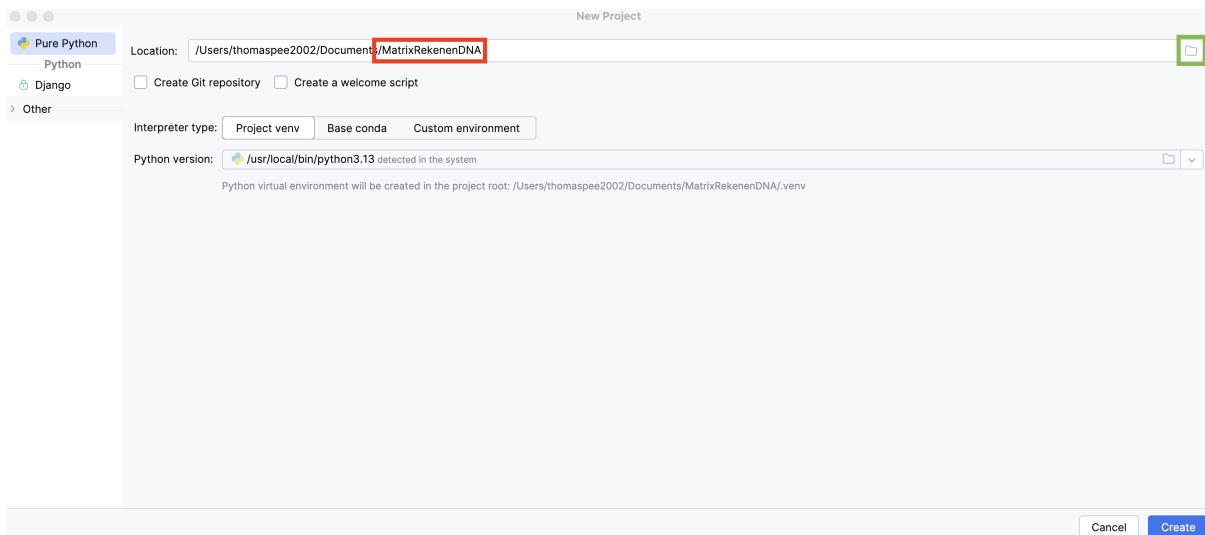
Voer de volgende stappen uit:

- 1) Ga naar de website van JetBrains: <https://www.jetbrains.com/pycharm/download/>.
- 2) Scroll naar beneden en klik op ‘Download’ onder Community Edition. Let op dat je niet PyCharm Professional downloadt (dit is geen gratis versie).
- 3) Open het gedownloade installatiebestand (pycharm-community-[versienummer].exe).
- 4) Volg de installatiewizard:
 - Klik op ‘Next’.
 - Kies de installatielocatie en klik op ‘Next’.
 - Klik op ‘Next’.
 - Klik op ‘Install’ en wacht tot de installatie is voltooid.
 - Klik op ‘Finish’.
- 5) Open PyCharm Community Edition.
- 6) Accepteer de licentievoorwaarden en klik op ‘Continue’.

7) Kies 'Skip Import' als dit de eerste keer is.

8) Klik op 'New Project':

- Selecteer een locatie voor je project. Dit kan je doen door rechtsboven op het map-symbool te klikken (zie groene kader in Figuur 1).
- Geef het project de naam 'MatrixRekenenDNA'. Dit kan door na de locatie '\MatrixRekenenDNA' te typen (zie rode kader in Figuur 1).
- Klik rechtsonder op 'Create'.



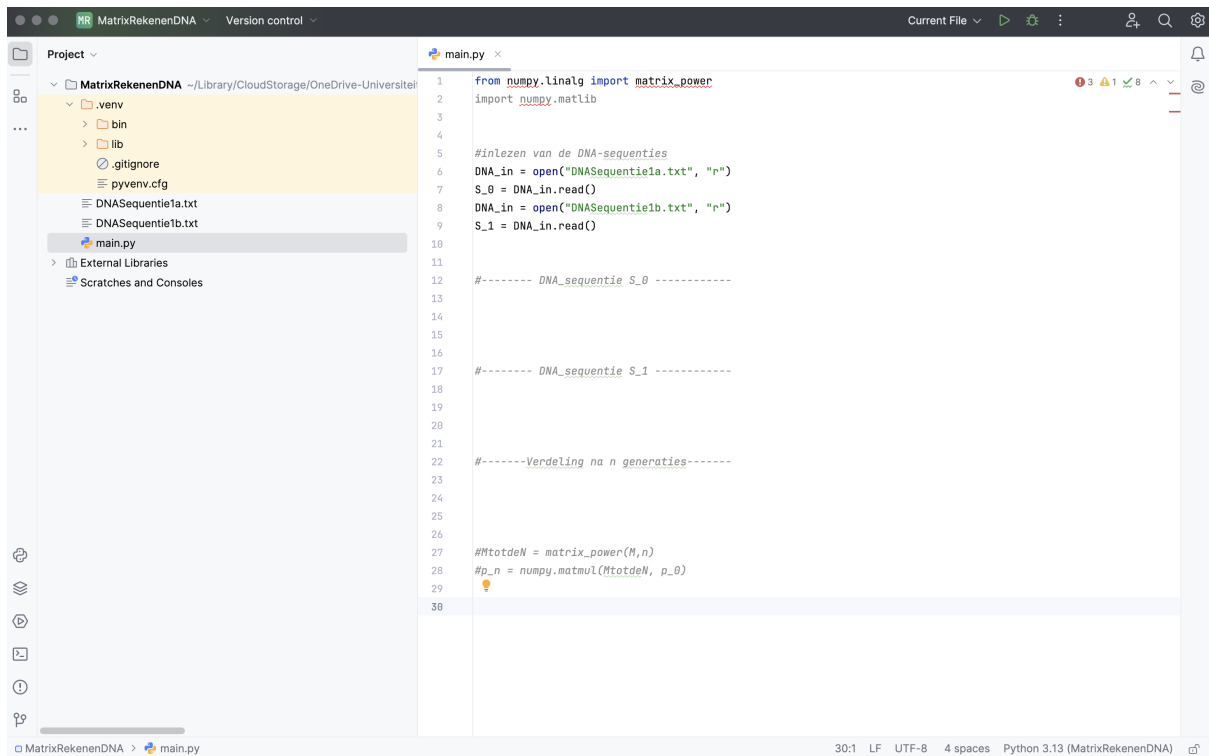
Figuur 1: Screenshot van PyCharm CE om een nieuw project aan te maken.

9) Je kunt nu beginnen met programmeren in Python!

Je plaatst de volgende bestanden in de map waar je je project hebt aangemaakt (deze staan op Smartschool):

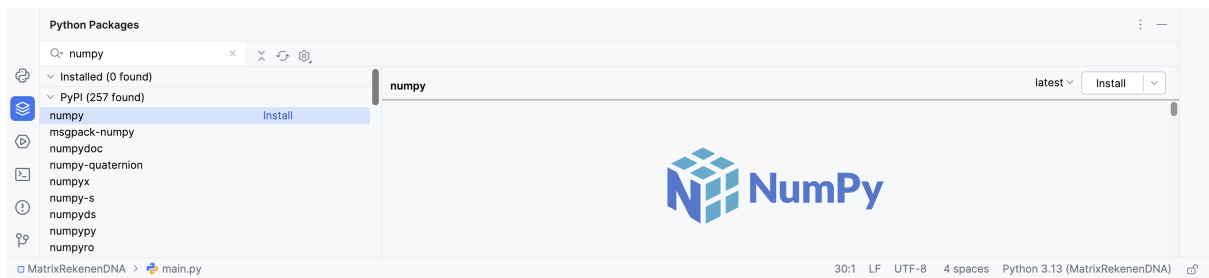
- DNASsequentieXa.txt
- DNASsequentieXb.txt
- main.py

met 'X' gelijk aan je groepsnummer. Wanneer je het project opent in PyCharm CE en `main.py` aanklikt, ziet je scherm er uit zoals in Figuur 2. Vergeet op lijn 6 en 8 van de code in `main.py` je groepsnummer niet aan te passen van 1 naar 'X'.



Figuur 2: Screenshot interface van PyCharm CE.

Omdat we enkele wiskundige berekeningen (met onder andere matrices) willen uitvoeren, moeten we nog het Python pakket **numpy** installeren. Dit doe je door linksonderaan je scherm op het symbool met de opeengestapelde parallellogrammen te klikken. Dit opent een nieuw venster waarbij je in de zoekbalk **numpy** moet intypen. Door het pakket **numpy** te selecteren, kan je rechts op de knop 'Install' klikken. **numpy** wordt hierdoor automatisch geïnstalleerd.



Figuur 3: Screenshot interface van PyCharm CE om **numpy** te installeren.

2.2 Programmeren in Python

Het eindresultaat van dit Python project kan je zien in Figuur 4. Doorheen de rest van deze bundel zullen jullie dit stap per stap namaken.

```
De eerste DNA-sequentie is gelijk aan:
AAACCCCGTT
De lengte van de eerste DNA-sequentie is 10

Het aantal nucleotiden in de DNA-streng is als volgt verdeeld:
aantal nucleotiden A is 3
aantal nucleotiden G is 1
aantal nucleotiden C is 4
aantal nucleotiden T is 2
aantal purines is 4
aantal pyrimidines is 6

De tweede DNA-sequentie is gelijk aan:
AAACCCCGTTTTT
De lengte van de tweede DNA-sequentie is 13

Het aantal nucleotiden in de DNA-streng is als volgt verdeeld:
aantal nucleotiden A is 3
aantal nucleotiden G is 1
aantal nucleotiden C is 4
aantal nucleotiden T is 5
aantal purines is 4
aantal pyrimidines is 9

De verdeling p_0 is gelijk aan
[[0.3], [0.2], [0.4], [0.1]]

De Markov matrix M is gelijk aan
[[0.1, 0.2, 0.3, 0.4], [0.1, 0.2, 0.3, 0.4], [0.1, 0.2, 0.3, 0.4],
[0.1, 0.2, 0.3, 0.4]]

De verdeling na 20 generaties is
[[0.23]
 [0.23]
 [0.23]
 [0.23]]
```

Figuur 4: Eindresultaat van dit Python project.

Opmerking: Probeer samen met je groepsgenoten op zoek te gaan naar een oplossing. Programmeren gebeurt immers (bijna) altijd in samenwerking met anderen. Indien bepaalde begrippen niet duidelijk zijn, aarzel dan niet om via het internet meer informatie te verzamelen! Dit is heel normaal bij het programmeren, niet enkel bij beginners.

Me when people ask
me how I learned
programming:



Tip: In het Engels vind je het meeste zoekresultaten.

2.3 Commentaar toevoegen met hash

Soms wil je extra informatie zetten in een `.py`-bestand. Dit doe je door een hash `#` te typen. Alle tekens hierna op dezelfde lijn worden dan genegeerd als je het programma runt. Deze lijn wordt dan een **comment** genoemd.

Zo een comment heeft twee mogelijke functies:

- Het hoofddoel van deze commentaar is om de code leesbaar te maken voor het menselijk oog. Dat wil zeggen dat er vaak in woorden staat uitgelegd wat Python nu eigenlijk aan het berekenen is.
- Dit symbool wordt vaak ook gebruikt om een lijn code tijdelijk ‘uit te schakelen’. Soms wil je code tijdelijk niet gebruiken of heb je een stukje code herschreven en wil je de eerdere versie niet weg doen omdat je deze misschien later nog nodig gaat hebben. Als je programma af is, moet deze commentaar wel altijd worden gewist! Anders wordt het programma vaak onleesbaar.

Beide soorten commentaar staan in het bestand `main.py`. Kijk naar Figuur 5 en schrijf hieronder het regelnummer en het doel van deze commentaar:

```

1 from numpy.linalg import matrix_power
2 import numpy.matlib
3
4
5 #inlezen van de DNA-sequenties
6 DNA_in = open("DNASequentie1a.txt", "r")
7 S_0 = DNA_in.read()
8 DNA_in = open("DNASequentie1b.txt", "r")
9 S_1 = DNA_in.read()
10
11
12 #----- DNA_sequentie S_0 -----
13
14
15
16
17 #----- DNA_sequentie S_1 -----
18
19
20
21
22 #-----Verdeling na n generaties-----
23
24
25
26
27 #MtotdeN = matrix_power(M,n)
28 #p_n = numpy.matmul(MtotdeN, p_0)

```

Figuur 5: main.py aan het begin van het project

2.4 De functie print()

De functie `print()` wordt uitgelegd in onderstaand filmpje: https://www.youtube.com/watch?v=_uoBdW-D9mg&list=PL2iW_rkiCt7UqxL47lGkBaTgApj_QHJ0K&index=4.



Wat doet de print-functie? Formuleer in je eigen woorden.

Je kan dus verschillende datatypes printen in de console:

- integers
- floats
- strings

Gebruik een print-functie voor elk datatype. Probeer deze eerst uit in de console. Schrijf vervolgens een voorbeeld van het print-statement hieronder neer voor de die verschillende datatypes.

Je kan ook deze datatypes combineren met elkaar. Dit doe je door tussen de verschillende datatypes een komma te zetten. Een voorbeeld hiervan is:

```
1 print("Het getal pi is gelijk aan", 3.14)
```

Zet nu de drie datatypes die je hierboven hebt gevonden in dezelfde print functie en noteer de code hieronder:

Tip: Als je een witregel wilt printen in de console, kan je gewoon `print()` gebruiken.

Je kan nu de volgende twee lijnen in `main.py` programmeren:

```
De eerste DNA-sequentie is gelijk aan:  
AAACCCCGTT
```

Deze code moet je onder de tussentitel: -----DNA-sequentie S_0----- zetten.

Tip: De DNA-sequentie die je moet printen is de DNA-sequentie van jouw groepje! Deze wordt in de variabele `S_0` gestopt op regel 6 en 7. Gebruik dus NIET `print("AAACCCCGTT")`.

2.5 De lengte van de DNA-sequentie en de functie `len()`

Nu willen we de lengte van de DNA-sequentie berekenen. Hiervoor maken we gebruik van de functie `len()`. De uitleg van deze functie kan je in dit filmpje vinden: https://www.youtube.com/watch?v=7krSoGswpd&list=PL2iW_rkiCt7UqxL47lGkBaTgApj_QHJOK&index=38.



Bereken eerst de lengte van de DNA-sequentie met `len()` en print dan deze tekst en de lengte van de DNA-sequentie in de code:

```
De lengte van de eerste DNA-sequentie is 10
```

2.6 Het tellen van de nucleotiden en de functie `count()`

De functie `count()` wordt gebruikt om te tellen hoe vaak een bepaald teken in een string voorkomt. Een voorbeeld kan je vinden in het volgende filmpje (van 2:20 tot 3:15): https://youtu.be/szIFFw_Xl_M?si=4zT447Nu03U3f4AZ&t=140.



Probeer de functie `count()` eens te gebruiken voor de string 'xyyyz'. Kijk hoe vaak x, y en z voorkomen en print dit aantal. Schrijf de code die je gebruikt hieronder:

Als dit gelukt is doe je nu hetzelfde voor de DNA-sequentie. We willen weten hoe vaak de nucleotiden A, G, C en T voorkomen in de DNA-streng. Ook zouden we willen weten hoeveel purines en pyrimidines er in totaal zijn. Het eindresultaat moet er als volgt uitzien:

```
Het aantal nucleotiden in de DNA-streng is als volgt verdeeld:  
aantal nucleotiden A is 3  
aantal nucleotiden G is 1  
aantal nucleotiden C is 4  
aantal nucleotiden T is 2  
aantal purines is 4  
aantal pyrimidines is 6
```

2.7 De tweede DNA-sequentie S_1

*Doe nu exact hetzelfde voor de tweede DNA-sequentie S_1.
Schrijf deze code onder de titel --- DNA-sequentie S_1 ---.
Dit is het resultaat dat je moet bekomen:*

```
De tweede DNA-sequentie is gelijk aan:  
AAACCCCGTTTT  
De lengte van de tweede DNA-sequentie is 13  
  
Het aantal nucleotiden in de DNA-streng is als volgt verdeeld:  
aantal nucleotiden A is 3  
aantal nucleotiden G is 1  
aantal nucleotiden C is 4  
aantal nucleotiden T is 5  
aantal purines is 4  
aantal pyrimidines is 9
```

2.8 De verdeling van de nucleotiden na n generaties en matrices in Python

We weten nu voor elke DNA-sequentie hoelang deze is en hoeveel bepaalde nucleotiden voorkomen. Aan de hand van deze gegevens kan je de basenverdelingen van de S_0 en S_1 -generaties berekenen. Om de Markov matrix te berekenen, heb je nog de gezamenlijke frequentietabel nodig. Dit is momenteel nog iets te moeilijk om zelf in Python op te stellen. Daarom vinden jullie deze tabel op SmartSchool: `frequentietabelX.txt` met 'X' gelijk aan je groepnummer.

Bereken de basenverdelingen met de hand:

$\mathbf{p}_0 =$

$\mathbf{p}_1 =$

Bereken de Markov matrix met de hand en vul deze in onderstaande tabel in:

$S_1 S_0$		S_0			
		A	G	C	T
S_1	A	...	...	...	...
	G	...	...	...	...
	C	...	...	...	...
	T	...	...	...	...

2.9 Matrices en vectoren in Python

De notatie van matrices wordt uitgelegd op de volgende website: <https://www.programiz.com/python-programming/matrix>.



Begin nu te programmeren onder de volgende tussentitel: `---Verdeling na n generaties---`.

Stel M gelijk aan de Markov matrix die je hierboven hebt berekend en schrijf deze in de `main.py` file. Doe hetzelfde voor de verdeling van de eerste DNA-sequentie, noem deze $\mathbf{p}_0$. Definieer ook de generatie n , neem deze bijvoorbeeld gelijk aan 25.

Tip: Een vector is gewoon een matrix met 1 rij of 1 kolom!

Gebruik deze variabelen en print het volgende op de console:

```
De verdeling p_0 is gelijk aan
[[0.3], [0.2], [0.4], [0.1]]

De Markov matrix M is gelijk aan
[[0.1, 0.2, 0.3, 0.4], [0.1, 0.2, 0.3, 0.4], [0.1, 0.2, 0.3, 0.4],
[0.1, 0.2, 0.3, 0.4]]
```

2.10 De functies `matrix_power` en `matmul`

Je mag nu bij de volgende code de hash weghalen:

```
1 #MtotdeN = matrix_power(M,n)
2 #p_n = numpy.matmul(MtotdeN, p_0)
```

Wat doet de functie `matrix_power`?

Wat doet de functie `matmul`?

Wat is dan de formule om hier p_n te berekenen?

Print nu het resultaat op de console als volgt:

```
De verdeling na 20 generaties is
[[0.23]
 [0.23]
 [0.23]
 [0.23]]
```

2.11 De finishlijn is in zicht!

Zorg dat het resultaat in de console er uit ziet zoals in Figuur 4, inclusief de witregels en de rest van de opmaak. Uiteraard staan bij jullie andere getallen. Zorg ook dat je code leesbaar is voor iemand die het project niet kent door voldoende commentaar toe te voegen.

Als je hiermee klaar bent, mag je al je bestanden uploaden via Smartschool.