Integratie Externe Functionaliteiten
Les 1 : NumPy

Wat en waarom NumPy?

We hoeven bij programmeren niets steeds het warm water uit te vinden. In de startersmodule Start to Program zagen we reeds hoe we grote delen extra functionaliteit aan de standaard Python kunnen toevoegen.
We doen dit aan de hand van **modules.**

NumPy is een veelgebruikte en krachtige module als we Python willen gebruiken voor data analyse en later eventueel machine learning.
De module is open-source en al in gebruik sinds 2005. Ze is voornamelijk in C geschreven, maar dus toepasbaar in Python. Dit geeft NumPy meer rekensnelheid. Het array object in NumPy rekent zo tot 50 keer sneller dan normale lijsten in Python. Dit maakt het perfect voor snelle verwerking van data.

NumPy geeft ons veel extra vooral **wiskundige functies**.
NumPy werkt vooral rond **multidimensionale arrays en matrices.**

Vooral de multidimensionale arrays interesseren ons als we later data visueel willen voorstellen.

We kunnen een uitgebreide documentatie van alle functies terugvinden op de volgende site:

https://numpy.org/

NumPy installeren en importeren

Voor we NumPy kunnen gebruiken moeten we de module eerst installeren. Ze is namelijk geen standaard module van Python die we zomaar kunnen importeren.

NumPy is open-source dus is vrij te verkrijgen en makkelijk toe te voegen.

We openen de opdrachtenprompt in Windows, of de terminal op Mac, en geven volgend commando.

```
pip install numpy
```

NumPy is nu toegevoegd aan de lijst van modules die we in Python kunnen gebruiken. We moeten ze alleen nog importeren in het begin van onze nieuw bestand.

Maak een nieuw Python bestand.
Importeer de numpy module.

Voeg nadien volgende code toe aan je python bestand.

```
print(numpy.__version__)
```

Voer het python bestand uit in de cmd.

Als alles goed verlopen is hebben we nu een programmaatje dat de geïnstalleerde versie van NumPy laat zien.

In de meeste voorbeelden en documentatie online gaan we merken dat er een alias gebruikt wordt voor de module. Zo moeten we niet elke keer numpy volledig uitschrijven. Deze alias is doorgaans ***np***.

Pas de code aan zodat de alias np gebruikt wordt.

NumPy array's

Het array-object van NumPy is te vergelijken als een lijst in de originele Python code.
Het verschil zit hem eerst en vooral in de rekensnelheid.
Maar een 2de interessante eigenschap is het makkelijk opbouwen van multidimensionale array's. Heel handig voor data-analyse op een x en y as bijvoorbeeld, maar daar later meer over.

We kunnen van elke lijst, tuple,... uit Python een array-object van NumPy maken. Dit met de array() functie van NumPy.
Nadien zal het herkent worden als het datatype ***ndarry***.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

Welk groot verschil merken we op in de weergave van de lijst?

De code hierboven geeft ons een 1 dimensionale array.

We bekijken de meest voorkomende dimensionale arrays.

0-D array

Dit betekend dat er geen array's zijn. Het bestaat uit slechts 1 getal.

```
arr0D = np.array(13)

print(arr0D)
```

1-D array

Een één-dimensionale array. Te vergelijken met 1 lijst.

```
arr1D = np.array([0,1,2,3,4])

print(arr1D)
```

2-D array

Een twee dimensionale array. Dit is een array van array's. Dit komt van pas als we data in een soort schema willen laten zien. Of denk maar aan velden in een excel tabel met rijnen en kolommen. Hier beginnen ook de wiskundige technieken met matrices.

```
arr2D = np.array([[0,1,2,3,4] , [5,6,7,8,9]])

print(arr2D)
```

3-D array

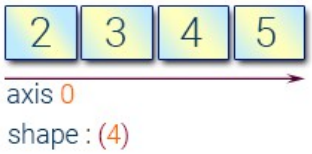
Bij een drie-dimensionale array wordt er nog extra as bijgehaald. Het is een array van array's met array's.

```
arr3D = np.array([[[1,2,3], [4,5,6]], [[1,2,3], [4,5,6]]])

print(arr3D)
```

Visueel ziet dit er als volgt uit:
Let nog even niet op de functies arange of reshape. Die bespreken we later.

1D Array

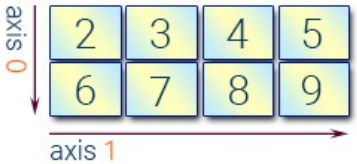
```
>>> import numpy as np
>>> x =  np.arange(2, 6).reshape(4)
>>> x
array( [ 2, 3, 4, 5 ] )

>>>
```



shape : (4)

2D Array

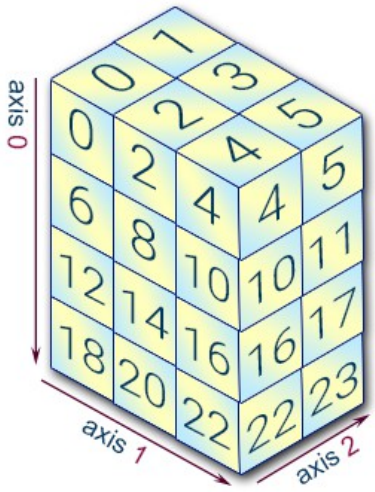
```
>>> import numpy as np
>>> x =  np.arange(2, 10).reshape(2, 4)
>>> x
array( [ [ 2, 3, 4, 5 ] )
         [ 6, 7, 8, 9 ] ] )
>>>
```



shape : (2, 4)

3D Array

```
>>> import numpy as np
>>> x =  np.arange(24).reshape(4, 3, 2)
>>> x
array( [ [ [ 0, 1] ),      [ [ 6, 7] ),      [ [ 12, 13] ),    [ [ 18, 19] ),
           [ 2, 3] ),        [ 8, 9] ),        [ 14, 15] ),      [ 20, 21] ),
           [ 4, 5] ),        [ 10, 11] ),      [ 16, 17] ),      [ 22, 23] ] ] )
>>>
```



shape : (4, 3, 2)

Dimensies tellen

Om de tel niet te verliezen in kunnen we van de array's opvragen uit hoeveel dimensies ze bestaan.
Dit met het ***ndim*** attribuut.

```
print(arr0D.ndim)
```

Vraag van alle andere voorbeelden ook het aantal dimensies op.

Multidimensionale arrays

We kunnen ook array's maken met zoveel dimensies als we willen. Dit is niet altijd even overzichtelijk. Daarom schrijven we de deze niet helemaal uit.

Deze komen praktisch ook niet veel voor.

```
arrND = np.array([1,2,3,4], ndmin=5)

print(arrND)
print('number of dimensions :', arrND.ndim)
```

Automatische array's

Er zijn een aantal veel voorkomende 1 dimensionale array's die we automatisch kunnen aanmaken.

arange()

Met de **arange()** functie maken we automatisch een optellende array aan tussen 2 waarden.
We kunnen instellen met welke stappen de array wordt gevuld.
Ook kunnen bepalen om welk datatype het gaat. NumPy is uitgebreider in het gebruik van datatypes. Hierover later meer.

```
np.arange([start, ]stop, [step, ][dtype=None])
```

Maak een array van alle getallen tussen 0 en 10.(excl. 10)

Maak een array van alle getallen tussen 5 en 10.(excl. 10)

Maak een array van alle even getallen tussen 25 en 45.

linspace()

linspace() geeft ons de mogelijkheid om een array te maken met een begin en een eind getal. We geven op hoeveel waarden we in de array willen en er wordt dan een eerlijke verdeling gemaakt om de tussenwaarden in te vullen.

```
np.linspace(start, stop, num=50,
            [endpoint=True,]
            [retstep=False,]
            [dtype=None])
```

endpoint is optioneel en staat standaard op True. Dit wil zeggen dat het eindgetal mee opgenomen wordt in de array.

retstep is optioneel en staat standaard op False. Op True zal de grote van de tussenstappen ook gegeven worden.

Maak een array aan die van 10 tot 50 loopt.
De array bestaat uit 15 waarden.

Maak een array aan die van 6 tot 25 loopt.
De array bestaat uit 7 waarden.
Het eindgetal, 25, is niet 1 van de waarden.
Hoeveel ligt er tussen de tussenwaarden?
Is dit hetzelfde als we 25 wel in de array meetellen?

shape en reshape()

shape is een attribuut dat voor ons de vorm van de multidimensionale array gaat geven in een tuple.

Vraag op dezelfde manier als we het aantal dimensies hebben opgevraagd nu de vorm op van onze 0D, 1D, 2D en 3D array's.

Reshape() is dan weer een functie die ons toelaat de vorm aan te passen. Zo kunnen we van onze automatische 1D arrays bijvoorbeeld een 2D array maken.

```
arrNET2D = arrNET.reshape(2,5)

print(arrNET2D)
```

LET OP! Wat gebeurd er als we 2 rijen van 4 waarden willen?

Het wordt nog meer denkwerk als we van een 1D array naar een 3D willen. Dat gaat als volgt te werk:

```
arrTwaalf = np.arange(12)

arrTwaalf3D = arrTwaalf.reshape(2,3,2)
print(arrTwaalf3D)
```

geeft:
2 array's met 3 array's van 2 elementen.

Maar we zitten op een pc dus het zou handiger zijn om niet alles zelf te moeten berekenen. We mogen 1 onbekende waarde hebben. Hiervoor vullen we dan **-1** in.

```
arrTwaalf3D = arrTwaalf.reshape(2,-1,2)
print(arrTwaalf3D)
```

Dit gaat automatisch de 3 array's als onderverdeling nemen.

Dit kunnen we ook op meer dan alleen 1D array's toepassen. Zolang we weten om hoeveel waarden het gaat kunnen we ze hervormen.

Het omgekeerde wordt ook gedaan. Van een multidimensionale array een 1D array maken. Dit wordt wel eens flattenig genoemd.
Dan wordt er slechts 1 waarde, **-1**, meegegeven.

Maak van de 3D array uit het begin een 1D array.

Oefenen en herhaling:

Oefn1.

Maak een 1D array tussen 12 en 38 in stappen van 3.
Print deze array omgekeerd af.

Oefn2.

Maak een array aan met waarden tussen 2 en 10, incl. 10;
Herschik deze in een matrix(vorm) van een 2D array van 3x3.

Oefn3.

Ga op internet op zoek naar de volgende automatische types van aangevulde arrays.
Maak dan van elke array een voorbeeld.
Begrijp je wat ze juist wat ze doen?

- eyes()
- identity()
- ones()
- zeros()
- full()
- ones_like()
- zeros_like()
- full_like()