Integratie Externe Functionaliteiten Les 2 : NumPy zoeken, sorteren & filteren

Zoeken

We kunnen de array's doorzoeken om de indexering op te vragen waar aan een bepaalde conditie wordt voldaan.
We gebruiken hiervoor de methode **np.where*(conditie)***.

Deze methode zal door de array loopen en de opgegeven conditie controleren. Die waarde wordt dan bijgehouden in een nieuwe lijst.

De conditie wordt opgebouwd zoals we die kennen uit de if-statement.

Maak een 1D array aan met verschillende nummers.

1 nummer komt op verschillende plaatsen in de array voor.

Slaag in een nieuwe variabel de lijst met indexnummer op waar het getal voorkomt.

Print deze af.

Maak een nieuwe automatische array aan met 10 getallen, exclusief 0.

Maak hiervan 2 nieuwe lijsten met de indexering van de oneven getallen en de indexering van de even getallen.

Print deze beide lijsten af.

Er is nog een 2de methode om te zoeken, maar die wordt op gesorteerde array's toegepast. Dus we bekijken eerst hoe we een array sorteren.

Sorteren

Met de methode **np.sort(*de_array*)** ordenen we een array.

Voor getallen betekend dit van groot naar klein.

Voor strings wordt er alfabetisch gewerkt.

Een array met booleans zal eerst alle False groeperen, gevolgd door alle True's.

Probeer uit.

Maak 3 array's. Voor elk datatype 1 en gebruik de sort() methode om ze te sorteren.

Hoe werkt dit met een multidimensionale array?

Bij multidimensionale array's kunnen we ook sorteren over een bepaalde as. We voegen dan, gescheiden door een komma, de as toe. Op eenzelfde manier als we vorige les bij *join* gezien hebben.

We voegen de as *None* toe als we de array plat willen trekken. We krijgen dan een gesorteerde 1D array.

Print de 2D array 3 maal af.

Met de as 0.

Met de as 1.

Met de as None.

Merk je de verschillen?

Nu kunnen we ook een speciaal soort zoeken in deze gesorteerde array's toepassen.

Deze zoektocht gaat een indexering weergeven, maar niet van een bestaand item. We gaan wel van een opgegeven waarde de mogelijke indexering zoeken als deze aan de array wordt toegevoegd.

We gebruiken hiervoor de methode:

np.searchsorted(*naam_array*, *waarde)*

Maak een array met 5 niet aansluitende getallen.

Controleer op welke indexering een ontbrekend getal zou komen.

Werkt dit ook met strings?

Probeer eens uit met lijst van ontbrekende getallen als waarde op te geven. Wat merk je op?

Sorteren op keyword

Wanneer we iets uit een databank binnenhalen krijgen de resultaten in de vorm van een tuple. Deze zal ook doorgaans uit verschillende datatypes bestaan. Om dan per kolom te sorteren geven we elke kolom eerst een alias. Nadien gebruiken we die alias om onze array te sorteren. We gebruiken hiervoor **order** om het keyword te kiezen waarop we ordenen.

```
dtypes = [('id',int), ('naam', 'S'),('punten',float)]
studenten = [(1,'Jan', 94.2),(2,'Freya',88.2),(3,'Filip',33.3)]
studentArr = np.array(studenten, dtype = dtype)
sortStudentArr = np.sort(studentArr, order = 'punten')
print(sortStudentArr)
```

We geven een lijst van keywords bij order als we een volgorde van sorteren willen bij gelijke waarden.

argsort & lexsort

De methode **argsort()** geeft ons een array met indexering. Deze indexering staat dan op volgorde van de gesorteerde lijst.

```
1D array:
arr = np.array([13,11,10,12])
argSortArr = np.argsort(arr)
print(argSortArr)
print([arr[i] for i in argSortArr])
```

De methode **lexsort()** kunnen we gebruiken wanneer er 2 lijsten zijn die gesorteerd moeten worden.

We sorteren dan eerst op de laatst opgegeven waarde, vervolgens op de 2de om de rang te bepalen.

Het resultaat is een array met indexering.

```
voornaam = ['Jan', 'Lea', 'An', 'Jan']
achternaam = ['Peeters', 'Schoemaker', 'Ooms', 'Janssen']
lexSortArr = np.lexsort((achternaam, voornaam))
print(lexSortArr)
print([voornaam[i] + ' ' +  achternaam[i] for i in lexSortArr])
```

Filteren

Met de basismanier van filteren gaan we een array vergelijken met een lijst.
In de array zitten elementen opgeslagen, in de lijst booleans.
In de gefilterde array worden dan de elementen opgeslagen met dezelfde indexering als de indexering van de True waarden van de lijst.

We vullen dan de lijst tussen vierkante haken in achter de naam van de array. Net alsof we een waarde op een bepaalde index zouden opvragen.

> Maak een automatische array met 5 elementen.
>
> Maak een lijst met 5 booleans. Wissel af met True en False.
>
> Slaag de gefilterde array op in een nieuwe variabele en print deze af.

Het is niet handig als we zelf de lijst met booleans moeten aanleveren. In de praktijk gaan we dus een lijst van booleans creëren aan de hand van een conditie.

We zouden dit kunnen doen met door onze array te itereren en dan met een if/else-statement True of False toe te voegen aan een lijst.

> Maak een automatische lijst met de getallen tussen 20 en 40.
>
> Maak een variabele met een lege lijst voor de booleans.
>
> Schrijf een loop die alle elementen afloopt.
>
> Als de waarde deelbaar is door 3 voeg je True toe aan de lijst, anders voeg je false toe.
>
> Filter aan de hand van deze lijst de array.
>
> Print ze af.

Aangezien dit een veel voorkomende opdracht is, heeft NumPy hiervoor een afgekorte manier gemaakt. Zo moeten we niet telkens een loop met if-statements bouwen.

We kunnen namelijk de conditie rechtstreeks opslaan in een variabele. Deze zal den een lijst met booleans aanmaken. Die kunnen we dan weer gebruiken als filter op de originele array.

Kopieer de vorige opdracht.

Vervang de volledige loop met de if-statement door variabele met een conditie.

Pas deze variabele toe op te de array als filter.

Print ze af.

Oefenen

Oefn1.

Sorteer de volgende array op verschillende manieren: [[10,40],[30,20]]

Over de eerste as.

Over de laatste as.

Vlak ze uit(flattening).

Oefn2.

patienten = [('Jan', 5, 1.78), ('Jos', 6, 1.82),('Paul', 5, 1.72), ('An', 5, 1.70)]

De volgende array bevat de naam, aantal bezoeken en hoogte van patiënten.

Orden ze volgens aantal bezoeken en vervolgens volgens hoogte als deze gelijk zijn.

Oefn3.

cursistId = np.array([1023, 5202, 6230, 1671, 1682, 5241, 4532])
cursistScore = np.array([40.7, 42.1, 45.5, 41.9, 38.3, 40.7, 42.1])

Sorteer volgens cursistScore en vervolgens op Id als de score gelijk is.