Package Imports

In [1]:

```
import numpy as np
```

Assignment 1: Array Basics

Hi there,

Can you import Numpy and convert the following list comprehension (I just learned about comprehensions in an awesome course by Maven) into an array?

Once you've done that report the following about the array:

- **The number of dimensions**
- **The shape**
- **The number of elements in the array**
- **The type of data contained inside**

In [2]:

```
my_list = [x * 10 for x in range(1, 11)]

my_array = np.array(my_list).reshape(5, 2)

my_array
```

Out[2]:

```
array([[ 10,  20],
       [ 30,  40],
       [ 50,  60],
       [ 70,  80],
       [ 90, 100]])
```

In [3]:

```
my_array.ndim
```

Out[3]:

```
2
```

In [4]:

```
my_array.shape
```

Out[4]:

```
(5, 2)
```

In [5]:

```
my_array.size
```

Out[5]:

```
10
```

In [6]:

```
my_array.dtype
```

Out[6]:

```
dtype('int32')
```

Assignment 2: Array Creation

Thanks for your help with the first piece - I'm starting to understand some of the key differences between base Python data types and NumPy arrays.

Does NumPy have anything like the range() function from base Python?

If so:

- **create the same array from assignment 1 using a NumPy function.**
- **Make it 5 rows and 2 columns.**
- **It's ok if the datatype is float or int.**

In [7]:

```
# With arange

my_array = np.arange(10, 101, 10).reshape(5, 2)

my_array
```

Out[7]:

```
array([[ 10,  20],
       [ 30,  40],
       [ 50,  60],
       [ 70,  80],
       [ 90, 100]])
```

In [8]:

```
# With linspace

my_array = np.linspace(10, 100, 10).reshape(5, 2)

my_array
```

Out[8]:

```
array([[ 10.,  20.],
       [ 30.,  40.],
       [ 50.,  60.],
       [ 70.,  80.],
       [ 90., 100.]])
```

In [9]:

```
# For fun: Use Array math to create multiples of 10 from single digit integers

my_array = (np.arange(1, 11) * 10).reshape(5, 2)

my_array
```

Out[9]:

```
array([[ 10,  20],
       [ 30,  40],
       [ 50,  60],
       [ 70,  80],
       [ 90, 100]])
```

Looking good so far! One of our data scientists asked about random number generation in NumPy.

Can you create a 3x3 array of random numbers between 0 and 1? Use a random state of 2022.

Store the random array in a variable called `random_array`.

In [10]:

```
rng = np.random.default_rng(2022)

random_array = rng.random(9).reshape(3, 3)

random_array
```

Out[10]:

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778],
       [0.1108689 , 0.04305584, 0.41441747]])
```

Assignment 3: Accessing Array Data

Slice and index the `random_array` we created in the previous exercise. Perform the following:

- **Grab the first two 'rows' of the array**
- **Grab the entire first column**
- **Finally, grab the second selement of the third row.**

Thanks!

In [11]:

```
random_array
```

Out[11]:

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778],
       [0.1108689 , 0.04305584, 0.41441747]])
```

In [12]:

```
random_array[:2, :]
```

Out[12]:

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778]])
```

In [13]:

```
random_array[:, 0]
```

Out[13]:

```
array([0.24742606, 0.06066207, 0.1108689 ])
```

In [14]:

```
random_array[2, 1]
```

Out[14]:

```
0.04305584439252108
```

Assignment 4: Arithmetic Operations

The creativity of our marketing team knows no bounds!

They've asked us to come up with a simple algorithm to provide a random discount to our list of prices below.

Before we do that,

- **Add a 5 dollar shipping fee to each price. Call this array `total`.**

Once we have that, we want to use the random_array created in assignment 2 and apply them to the 6 prices.

- **Grab the first 6 numbers from `random_array`, reshape it to one dimension. Call this `discount_pct`.**
- **Subtract `discount_pct` FROM 1, store this in `pct_owed`.**
- **Multiply `pct_owed` by `total` to get the final amount owed.**

In [15]:

```
prices = np.array([5.99, 6.99, 22.49, 99.99, 4.99, 49.99])

total = prices + 5

total
```

Out[15]:

```
array([ 10.99,  11.99,  27.49, 104.99,   9.99,  54.99])
```

In [16]:

```
discount_pct = random_array[:2, :].reshape(6)

pct_owed = 1 - discount_pct

final_owed = total * pct_owed

final_owed.round(2)
```

Out[16]:

```
array([ 8.27, 10.88, 10.67, 98.62,  3.39, 13.46])
```

In [17]:

```
((1 - (random_array[:2, :].reshape(6))) * total).round(2)
```

Out[17]:

```
array([ 8.27, 10.88, 10.67, 98.62,  3.39, 13.46])
```

In [18]:

```
print(discount_pct)
print(pct_owed)
print(final_owed.round(2))
```

```
[0.24742606 0.09299006 0.61176337 0.06066207 0.66103343 0.75515778]
[0.75257394 0.90700994 0.38823663 0.93933793 0.33896657 0.24484222]
[ 8.27 10.88 10.67 98.62  3.39 13.46]
```

Assignment 5: Filtering Arrays

Filter the product array to only include those with prices greater than 25.

Modify your logic to include cola, despite it not having a price greater than 25. Store the elements returned in an array called `fancy_feast_special`.

Next, create a shipping cost array where the cost is 0 if price is greater than 20, and 5 if not.

In [19]:

```
products = np.array(
    ["salad", "bread", "mustard", "rare tomato", "cola", "gourmet ice cream"]
)

products
```

Out[19]:

```
array(['salad', 'bread', 'mustard', 'rare tomato', 'cola',
       'gourmet ice cream'], dtype='<U17')
```

In [20]:

```
products[prices > 25]
```

Out[20]:

```
array(['rare tomato', 'gourmet ice cream'], dtype='<U17')
```

In [21]:

```
mask = (prices > 25) | (products == "cola")

fancy_feast_special = products[mask]

fancy_feast_special
```

Out[21]:

```
array(['rare tomato', 'cola', 'gourmet ice cream'], dtype='<U17')
```

In [22]:

```
shipping = np.where(prices > 20, 0, 5)

shipping
```

Out[22]:

```
array([5, 5, 0, 0, 5, 0])
```

Assignment 6: Aggregating and Sorting Arrays

First, grab the top 3 highest priced items in our list.

Then, calculated the mean, min, max, and median of the top three prices.

Finally, calculate the number of unique price tiers in our `price_tiers` array.

In [23]:

```
prices = np.array([5.99, 6.99, 22.49, 99.99, 4.99, 49.99])

prices.sort()
```

In [24]:

```
prices
```

Out[24]:

```
array([ 4.99,  5.99,  6.99, 22.49, 49.99, 99.99])
```

In [25]:

```
top3 = prices[-3:]
```

In [26]:

```
print(f"Mean: {top3.mean()}")
print(f"Min: {top3.min()}")
print(f"Max: {top3.max()}")
print(f"Median: {np.median(top3)}")
```

```
Mean: 57.49
Min: 22.49
Max: 99.99
Median: 49.99
```

In [27]:

In [27]:

```
price_tiers = np.array(["budget", "budget", "mid-tier", "luxury", "mid-tier", "luxury"])
```

In [28]:

```
np.unique(price_tiers)
```

Out[28]:

```
array(['budget', 'luxury', 'mid-tier'], dtype='<U8')
```

Assignment 7: Bringing it All Together

Ok, final NumPy task - let's read in some data with the help of Pandas.

Our data scientist provided the code to read in a csv as a Pandas dataframe, and has converted the two columns of interest to arrays.

- **Filter `sales_array` down to only sales where the product family was produce.**
- **Then, randomly sample roughly half (random number < .5) of the produce sales and report the mean and median sales. Use a random seed of 2022.**
- **Finally, create a new array that has the values 'above_both', 'above_median', and 'below_both' based on whether the sales were above the median and mean of the sample, just above the median of the sample, or below both the median and mean of the sample.**

In [30]:

```
import pandas as pd
import numpy as np

retail_df = pd.read_csv(
    "retail_2016_2017.csv", skiprows=range(1, 11000), nrows=1000
)

family_array = np.array(retail_df["family"])
sales_array = np.array(retail_df["sales"])
```

In [31]:

```
produce_array = sales_array[family_array == "PRODUCE"]
```

In [32]:

```
rng = np.random.default_rng(2022)

random_array = rng.random(30)

sampled_array = produce_array[random_array < 0.5]
```

In [33]:

```
mean = sampled_array.mean()

mean
```

Out[33]:

```
2268.102470588235
```

In [34]:

```
median = np.median(sampled_array)

median
```

Out[34]:

```
1272.755
```

In [35]:

```
np.where(
    sampled_array < median,
    "below_both",
    np.where(sampled_array > mean, "above_both", "above_median"),
)
```

Out[35]:

```
array(['above_median', 'below_both', 'below_both', 'below_both',
       'above_both', 'below_both', 'below_both', 'above_both',
       'below_both', 'above_median', 'above_both', 'above_both',
       'below_both', 'above_median', 'above_both', 'below_both',
       'above_both'], dtype='<U12')
```

In [36]:

```
sampled_array
```

Out[36]:

```
array([1662.394,  447.064,  962.866, 1077.44 , 3404.531,  962.96 ,
       1089.319, 7860.031,  446.038, 1272.755, 2775.771, 2339.906,
        722.333, 1567.843, 2458.456,  673.885, 8834.15 ])
```

In [ ]:

```

```