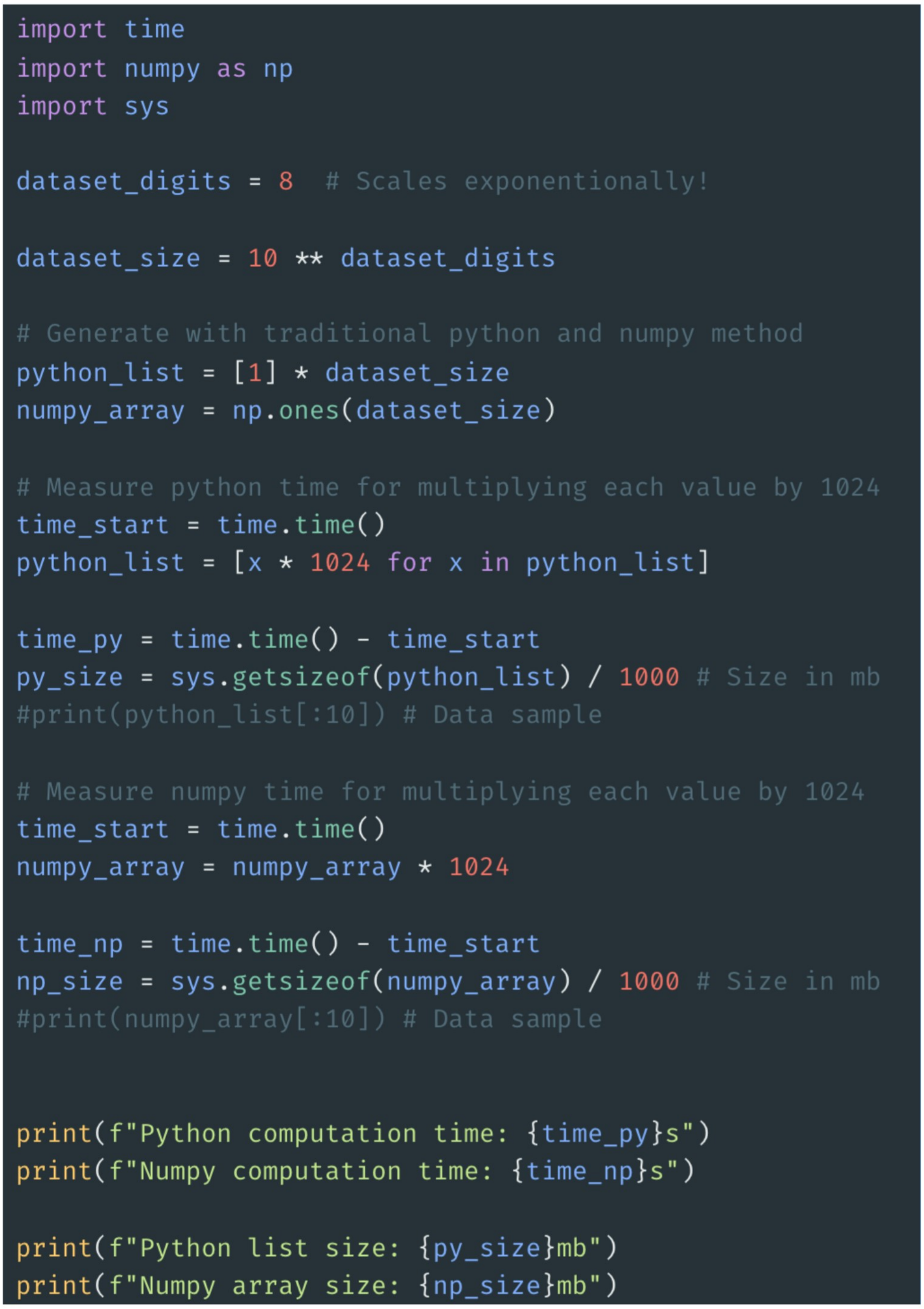
1.0 - Numpy

Numpy is like python lists but faster and more memory efficient simply because numpy arrays (equivalent to python lists) does not allow different datatypes in the same dataset.

1.1 Python vs Numpy illustration

On Canvas as *python_vs_numpy.py*



```
import time
import numpy as np
import sys

dataset_digits = 8  # Scales exponentionally!

dataset_size = 10 ** dataset_digits

# Generate with traditional python and numpy method
python_list = [1] * dataset_size
numpy_array = np.ones(dataset_size)

# Measure python time for multiplying each value by 1024
time_start = time.time()
python_list = [x * 1024 for x in python_list]

time_py = time.time() - time_start
py_size = sys.getsizeof(python_list) / 1000 # Size in mb
#print(python_list[:10]) # Data sample

# Measure numpy time for multiplying each value by 1024
time_start = time.time()
numpy_array = numpy_array * 1024

time_np = time.time() - time_start
np_size = sys.getsizeof(numpy_array) / 1000 # Size in mb
#print(numpy_array[:10]) # Data sample


print(f"Python computation time: {time_py}s")
print(f"Numpy computation time: {time_np}s")

print(f"Python list size: {py_size}mb")
print(f"Numpy array size: {np_size}mb")
```

2.0 - Straight to the task

The flying gardeners have measured car data outside their school which they now happily share for demonstration purposes.

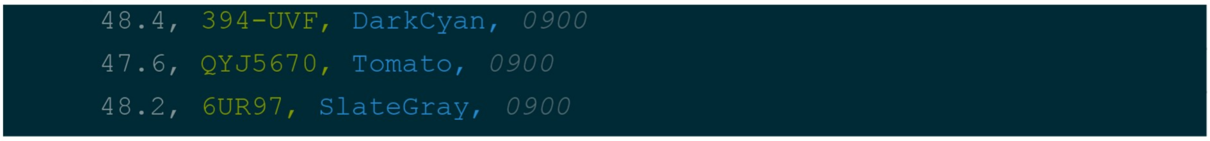
The data they collected is contained in a txt file called “speed_measurements.txt” (download from canvas) and contains row entries with the format:

Speed LicensePlate Color Time

2.0.1 - Data format

Speed -	Formatted as kmh
LisencePlate -	No predetermined standard, spaces can occur
Color -	Named colors
Time -	HHMM

A sample of the file

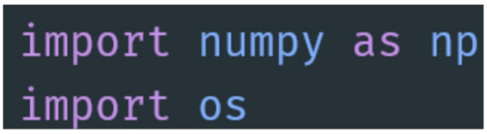


```
48.4, 394-UVF, DarkCyan, 0900
47.6, QYJ5670, Tomato, 0900
48.2, 6UR97, SlateGray, 0900
```

2.1 - Code task

A. Create a file called “process_car_data.py”

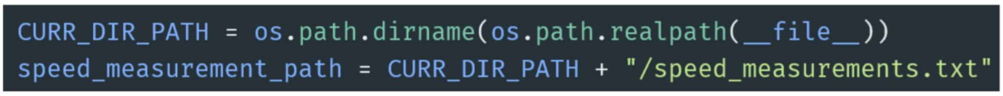
B. Import numpy and os



```
import numpy as np
import os
```

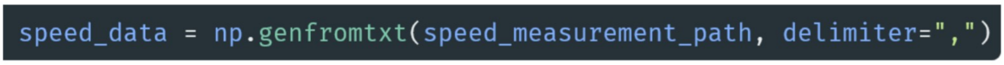
C. Put the file “speed_measurements.txt” in the same directory as the python file created in step A.

D. Create local references to the aforementioned file



```
CURR_DIR_PATH = os.path.dirname(os.path.realpath(__file__))
speed_measurement_path = CURR_DIR_PATH + "/speed_measurements.txt"
```

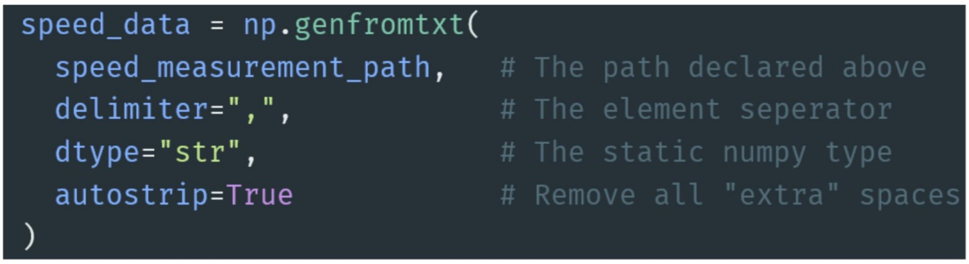
E. Fetch the content of the file using numpy’s function “genfromtxt”



```
speed_data = np.genfromtxt(speed_measurement_path, delimiter=",")
```

However, if you **print** the result of *speed_data*, do you notice anything strange?

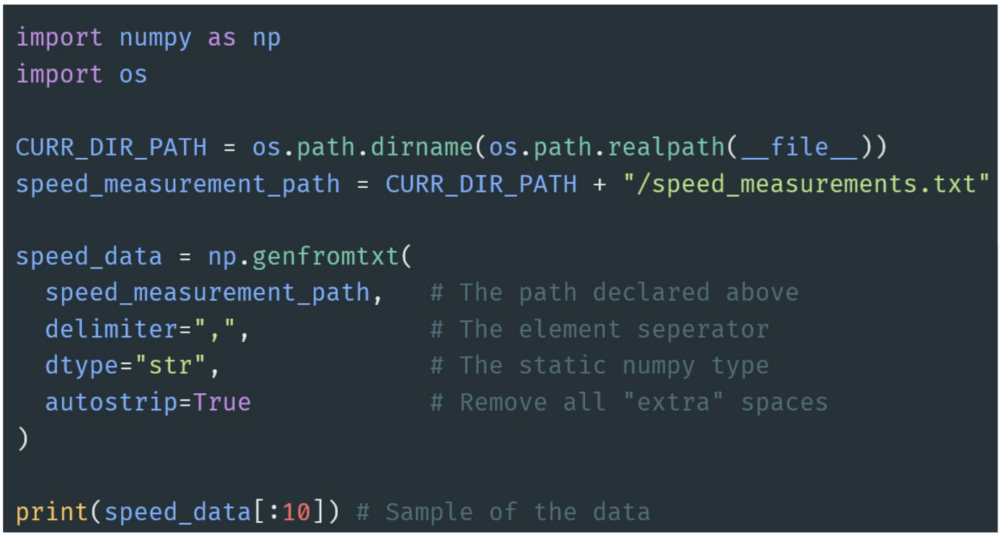
F. (**Replace** above) Add data type parameter and remove unnecessary spaces



```
speed_data = np.genfromtxt(
  speed_measurement_path,   # The path declared above
  delimiter=",",            # The element seperator
  dtype="str",              # The static numpy type
  autostrip=True            # Remove all "extra" spaces
)
```

If you wish to read up on more parameters, see https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html

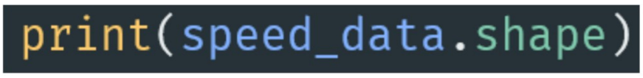
G. Well done, you have successfully loaded the measurements into the python environments memory. Full code below.



```
import numpy as np
import os

CURR_DIR_PATH = os.path.dirname(os.path.realpath(__file__))
speed_measurement_path = CURR_DIR_PATH + "/speed_measurements.txt"

speed_data = np.genfromtxt(
  speed_measurement_path,   # The path declared above
  delimiter=",",            # The element seperator
  dtype="str",              # The static numpy type
  autostrip=True            # Remove all "extra" spaces
)

print(speed_data[:10]) # Sample of the data
```

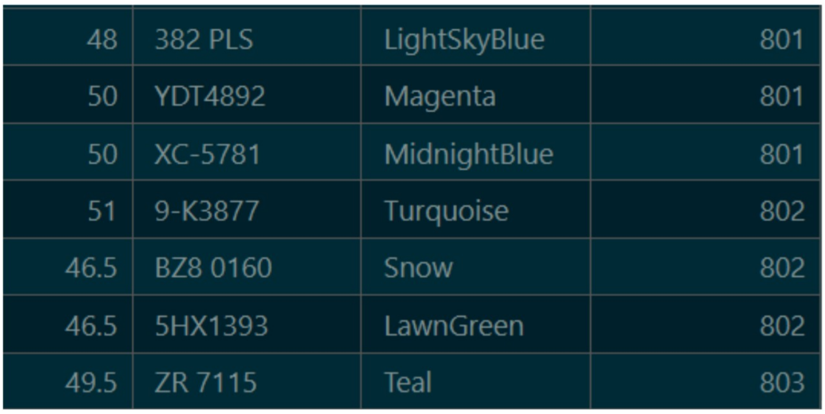
2.2 - Seperating the data

Right now we have a numpy array with the shape (834, 4).



```
print(speed_data.shape)
```

The shape explains how many rows and columns the dataset contains, 834 and 4 respectively. A sample of the data illustrated as a table below.

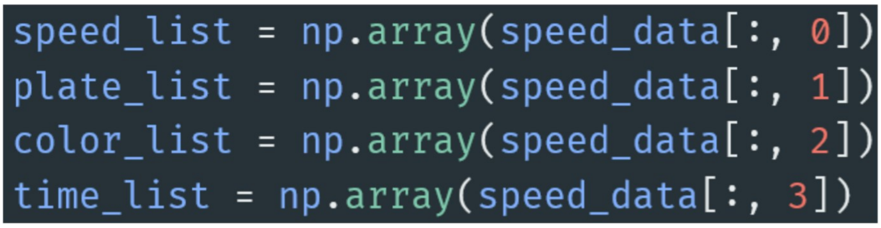


48	382 PLS	LightSkyBlue	801
50	YDT4892	Magenta	801
50	XC-5781	MidnightBlue	801
51	9-K3877	Turquoise	802
46.5	BZ8 0160	Snow	802
46.5	5HX1393	LawnGreen	802
49.5	ZR 7115	Teal	803

With numpy we can easily seperate the different data into 1 column (1d) arrays.

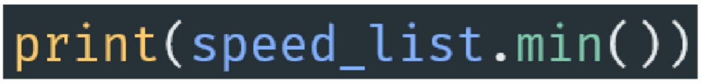
2.2.1 - Seperating in code

A. Using the list slicer on the numpy array, seperate speed, plate, color and time into their own arrays.



```
speed_list = np.array(speed_data[:, 0])
plate_list = np.array(speed_data[:, 1])
color_list = np.array(speed_data[:, 2])
time_list = np.array(speed_data[:, 3])
```

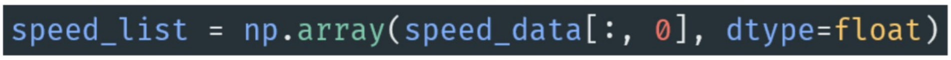
B. Use the min function to find the lowest speed that day



```
print(speed_list.min())
```

You will get a type error, **reflect** about why that happened. Clearly speed is compiled of float numbers? no?

C. Change the slicing above for the speed list to fixate the data type to a float

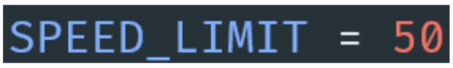


```
speed_list = np.array(speed_data[:, 0], dtype=float)
```

Would it work with an int type instead? Why (not)?

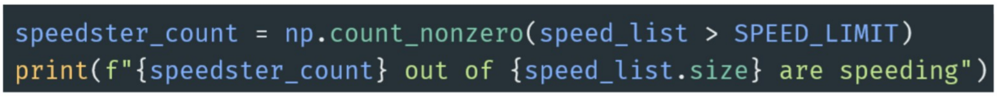
2.2.2 - Check speedsters

A. Create a speed limit variable. For this data the speed limit was 50.



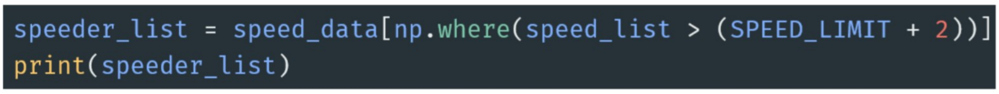
```
SPEED_LIMIT = 50
```

B. Count the non-zero (doesn't make sense to count zero values..?) values higher than the speed limit



```
speedster_count = np.count_nonzero(speed_list > SPEED_LIMIT)
print(f"{speedster_count} out of {speed_list.size} are speeding")
```

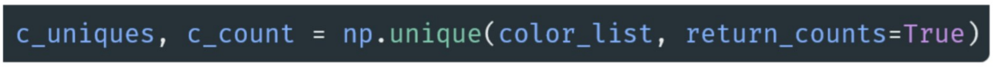
C. Let's bust those speedsters. Print a list of the worst speedsters, those who drive more than 2 kmh over the speed limit.



```
speeder_list = speed_data[np.where(speed_list > (SPEED_LIMIT + 2))]
print(speeder_list)
```

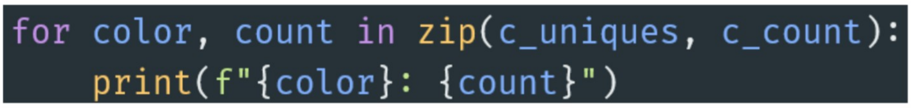
2.2.3 - Count uniques

A. Finally, why don't we count some unique colors too?



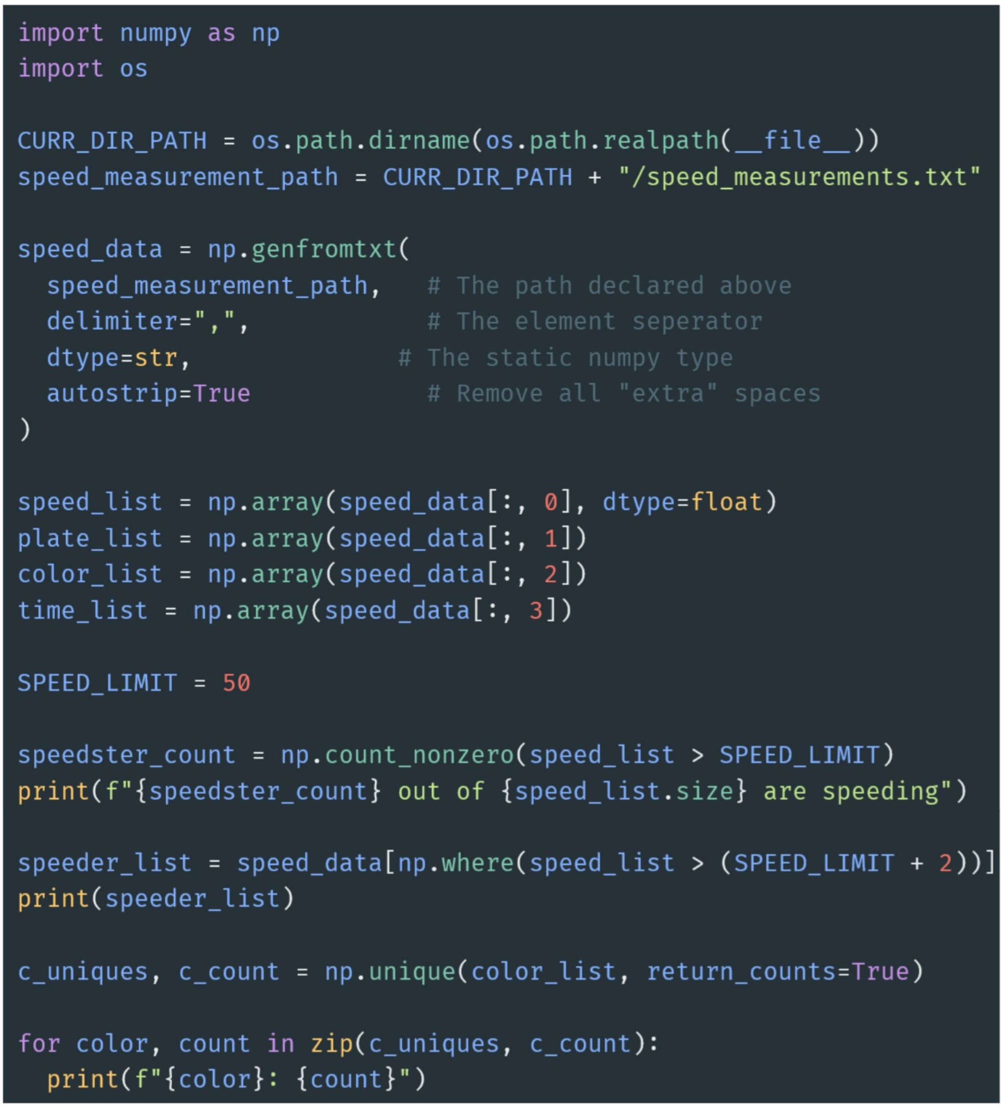
```
c_uniques, c_count = np.unique(color_list, return_counts=True)
```

print the result by your own design, however, using the function "zip" is helpful when iterating through two related collections.



```
for color, count in zip(c_uniques, c_count):
    print(f"{color}: {count}")
```

2.4 - Full code



```python
import numpy as np
import os

CURR_DIR_PATH = os.path.dirname(os.path.realpath(__file__))
speed_measurement_path = CURR_DIR_PATH + "/speed_measurements.txt"

speed_data = np.genfromtxt(
  speed_measurement_path,   # The path declared above
  delimiter=",",            # The element seperator
  dtype=str,              # The static numpy type
  autostrip=True            # Remove all "extra" spaces
)

speed_list = np.array(speed_data[:, 0], dtype=float)
plate_list = np.array(speed_data[:, 1])
color_list = np.array(speed_data[:, 2])
time_list = np.array(speed_data[:, 3])

SPEED_LIMIT = 50

speedster_count = np.count_nonzero(speed_list > SPEED_LIMIT)
print(f"{speedster_count} out of {speed_list.size} are speeding")

speeder_list = speed_data[np.where(speed_list > (SPEED_LIMIT + 2))]
print(speeder_list)

c_uniques, c_count = np.unique(color_list, return_counts=True)

for color, count in zip(c_uniques, c_count):
  print(f"{color}: {count}")
```