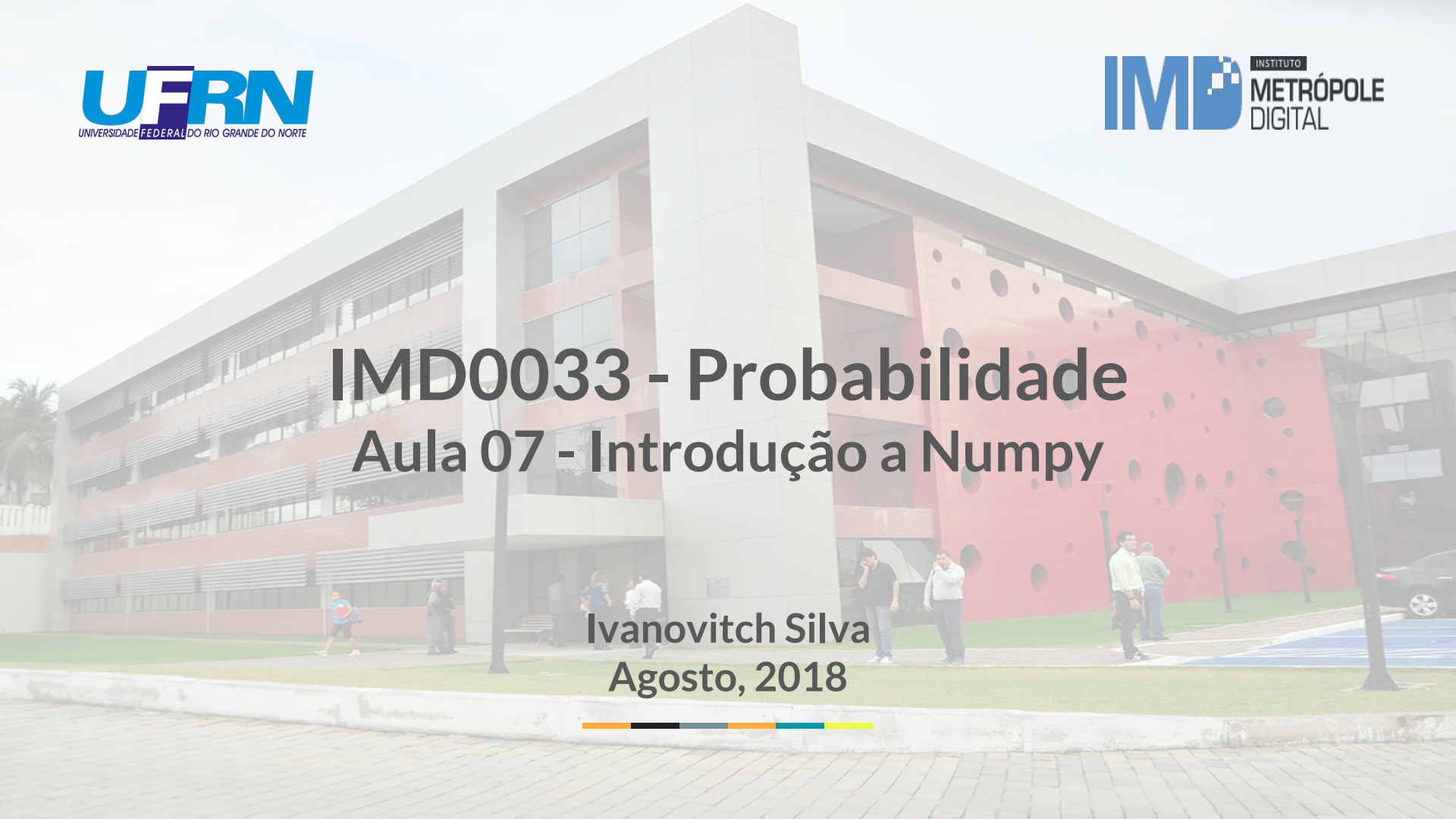


IMD0033 - Probabilidade

Aula 07 - Introdução a Numpy

Ivanovitch Silva

Agosto, 2018

Agenda

- Numpy - Overview
- Understanding NumPy ndarrays
- Selecting and slicing
- Arithmetic operations
- Adding rows and columns
- Sorting

nyc_taxis.csv

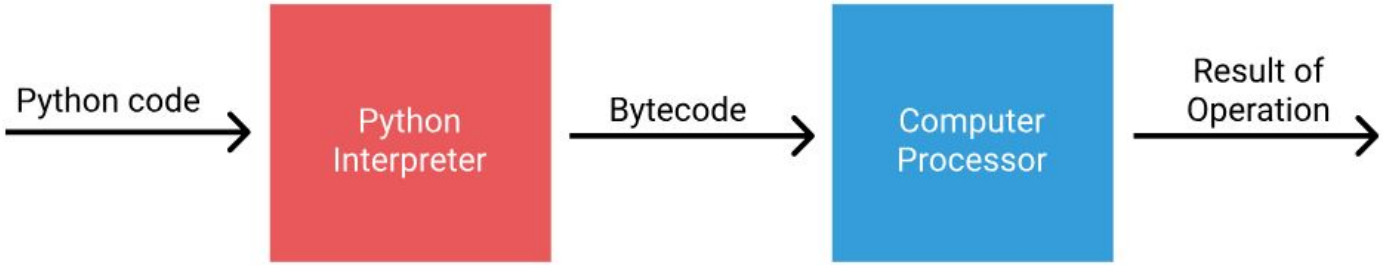
Atualizar o repositório

```
git clone https://github.com/ivanovitchm/imd0033_2018_2.git
```

Ou ....

```
git pull
```

Understanding Vectorization



Language Type	Example	Time taken to write program	Control over program performance
High-Level	Python	Low	Low
Low-Level	C	High	High

How can Python be so efficient?

Unvectorized code using list of lists

6	5
1	3
5	6
1	4
3	7
5	8
3	5
8	4

Two columns of numbers

```
my_numbers = [
                [6, 5],
                [1, 3],
                [5, 6],
                [1, 4],
                [3, 7],
                [5, 8],
                [3, 5],
                [8, 4]
              ]
```

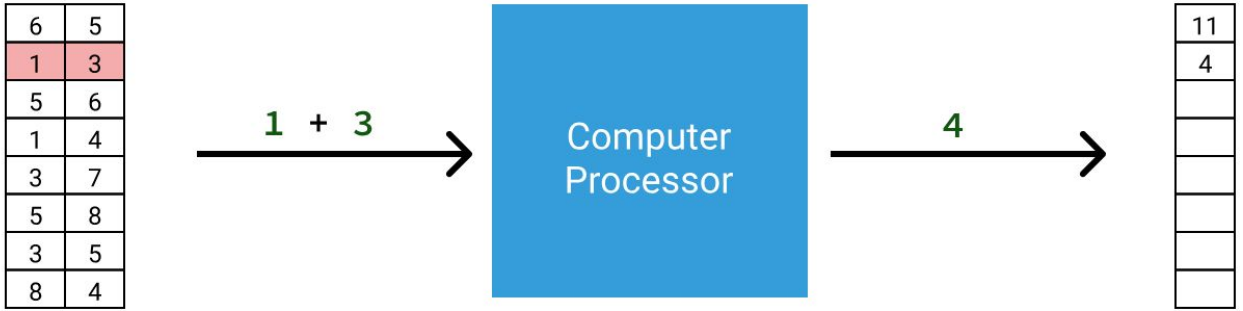
List of lists representation

```
sums = []

for row in my_numbers:
    row_sum = row[0] + row[1]
    sums.append(row_sum)
```

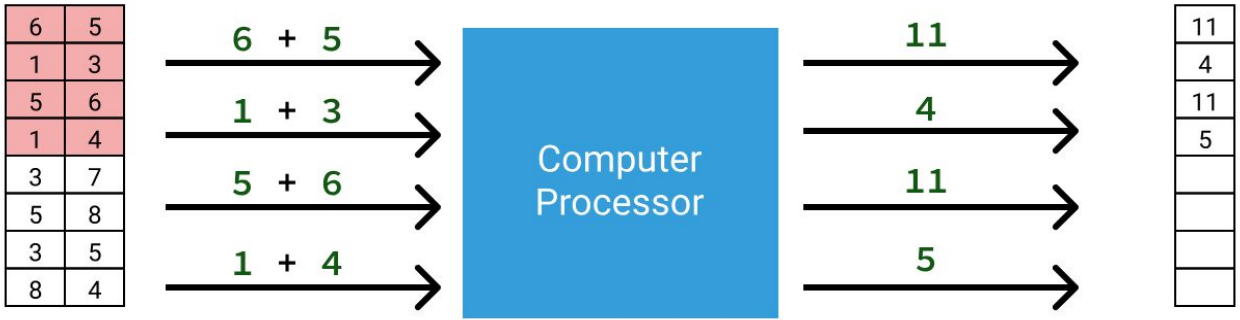
Python code to sum each row

How Vectorization Makes Code Faster

Single Instruction Multiple Data (SIMD)



Numpy



NYC Taxi-Airport Data

There is data on over 1.3 trillion individual trips, reaching back as far as 2009 and is regularly updated

nyc_taxis.csv

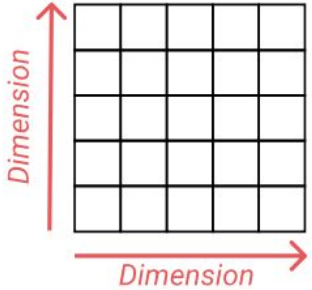
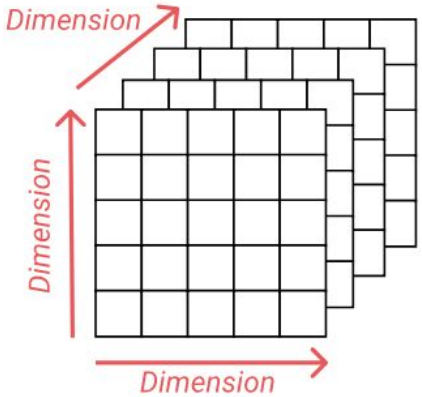
NYC Taxi-Airport Data

pickup_year	pickup_month	pickup_day	pickup_dayofweek	pickup_time	pickup_location_code	dropoff_location_code	trip_distance	trip_length	fare_amount	total_amount
2016	1	1	5	0	2	4	21.00	2037	52.0	69.99
2016	1	1	5	0	2	1	16.29	1520	45.0	54.30
2016	1	1	5	0	2	6	12.70	1462	36.5	37.80
2016	1	1	5	0	2	6	8.70	1210	26.0	32.76
2016	1	1	5	0	2	6	5.56	759	17.5	18.80
2016	1	1	5	0	4	2	21.45	2004	52.0	105.60
2016	1	1	5	0	2	6	8.45	927	24.5	32.25
2016	1	1	5	0	2	6	7.30	731	21.5	22.80
2016	1	1	5	0	2	5	36.30	2562	109.5	131.38
2016	1	1	5	0	6	2	12.46	1351	36.0	37.30

```
1  import csv
2  import numpy as np
3
4  # import nyc_taxi.csv as a list of lists
5  # remove the header row
6  # convert each element to float
7
8  taxi = np.array(
9        [[float(item) for item in row]
10         for row in list(csv.reader(open("nyc_taxis.csv", "r")))[1:]]
11      )
12 print(type(taxi))
13 taxi[:2]
14
15
```

```
<class 'numpy.ndarray'>
array([[2.016e+03, 1.000e+00, 1.000e+00, 5.000e+00, 0.000e+00, 2.000e+00,
        4.000e+00, 2.100e+01, 2.037e+03, 5.200e+01, 8.000e-01, 5.540e+00,
        1.165e+01, 6.999e+01, 1.000e+00],
       [2.016e+03, 1.000e+00, 1.000e+00, 5.000e+00, 0.000e+00, 2.000e+00,
        1.000e+00, 1.629e+01, 1.520e+03, 4.500e+01, 1.300e+00, 0.000e+00,
        8.000e+00, 5.430e+01, 1.000e+00]])
```

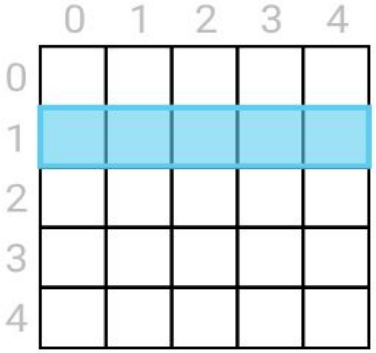
Understanding Numpy ndarray

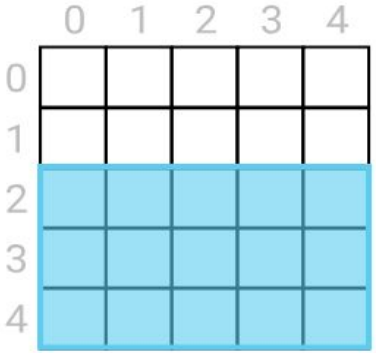
	Number of Dimensions	Known As
	One	One-dimensional array, array, list, vector, sequence
	Two	Two-dimensional array, matrix, table, list of lists, spreadsheet
	Three	Three-dimensional array, multi-dimensional array, panel

Introduction to Numpy

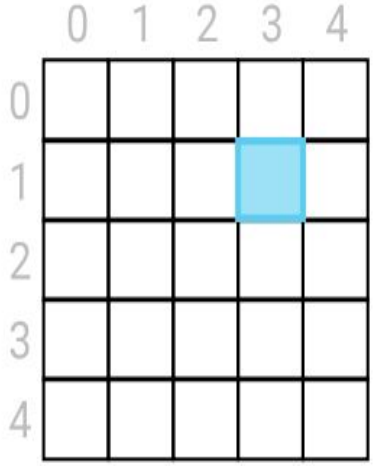
```
>>> print(taxi)

    [[ 2016.  1.   1.  ..., 11.65  69.99   1. ]
     [ 2016.  1.   1.  ...,  8.    54.3    1. ]
     [ 2016.  1.   1.  ...,  0.    37.8    2. ]
     ...,
     [ 2016.  6.  30.  ...,  5.    63.34   1. ]
     [ 2016.  6.  30.  ...,  8.95  44.75   1. ]
     [ 2016.  6.  30.  ...,  0.    54.84   2. ]]

 >>> taxi.shape
     (89560, 15)
```

	List of lists method	*NumPy method*
Selecting a single row	`sel_lol = data_lol[1]`	`sel_np = data_np[1]`
		Same syntax as list of lists. Produces a 1D ndarray.
Selecting multiple rows	`sel_lol = data_lol[2:]`	`sel_np = data_np[2:]`
		Same syntax as list of lists. Produces a 2D ndarray.





Selecting a single item



List of lists method

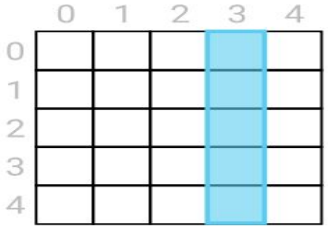
```
sel_lol = data_lol[1][3]
```

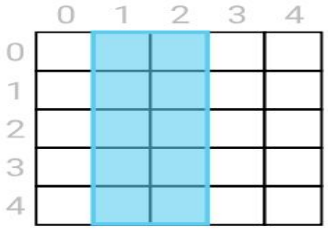
NumPy method

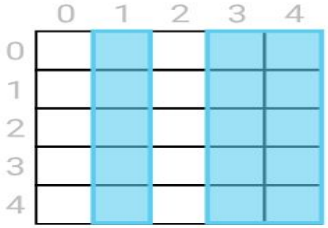
```
sel_np = data_np[1,3]
```

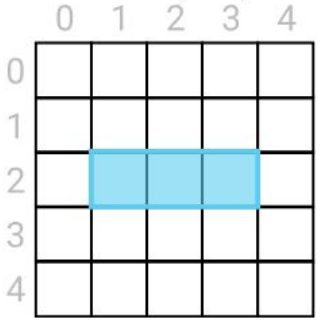
Comma separated row/column locations. Produces a single Python object.

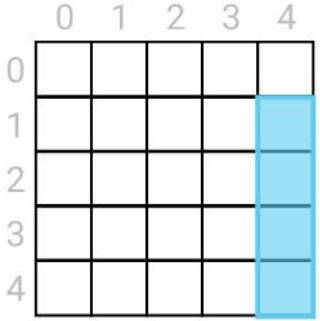
	List of lists method	*NumPy method*

Selecting a single column 	`sel_lol = []`
	
	`for row in data_lol:`
	`  col4 = row[3]`
	`  sel_lol.append(col4)`
		`sel_np = data_np[:,3]`
		
		Comma separated row wildcard and column location. Produces a 1D ndarray

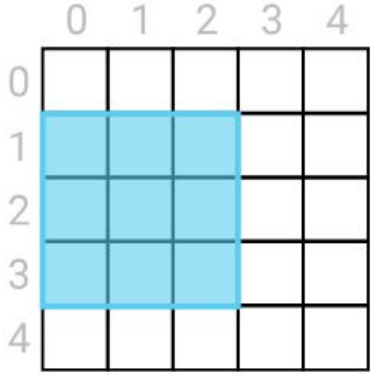
Selecting multiple columns 	`sel_lol = []`
	
	`for row in data_lol:`
	`  col23 = row[1:3]`
	`  sel_lol.append(col23)`
		`sel_np = data_np[:,1:3]`
		
		Comma separated row wildcard and column slice location. Produces a 2D ndarray

Selecting multiple, specific columns 	`sel_lol = []`
	
	`for row in data_lol:`
	`  cols = [row[1],`
	`  row[3],row[4]]`
	`  sel_lol.append(cols)`
		`cols = [1,3,4]`
		`sel_np = data_np[:,cols]`
		
		Comma separated row wildcard and list of column locations. Produces a 2D ndarray

	List of lists method	*NumPy method*
Selecting a 1D slice (row) 	`sel_lol = data_lol[2][1:4]`	`sel_np = data_np[2,1:4]` *Comma separated row location and column slice. Produces a 1D ndarray*

Selecting a 1D slice (column)
 
	`sel_lol = []`
	
	`rows = data_lol[1:]`
	`for r in rows:`
	`  col5 = r[4]`
	`  sel_lol.append(col5)`
		`sel_np = data_np[1:,4]`
		Comma separated row slice and column location. Produces a 1D ndarray

Selecting a 2D slice



List of lists method

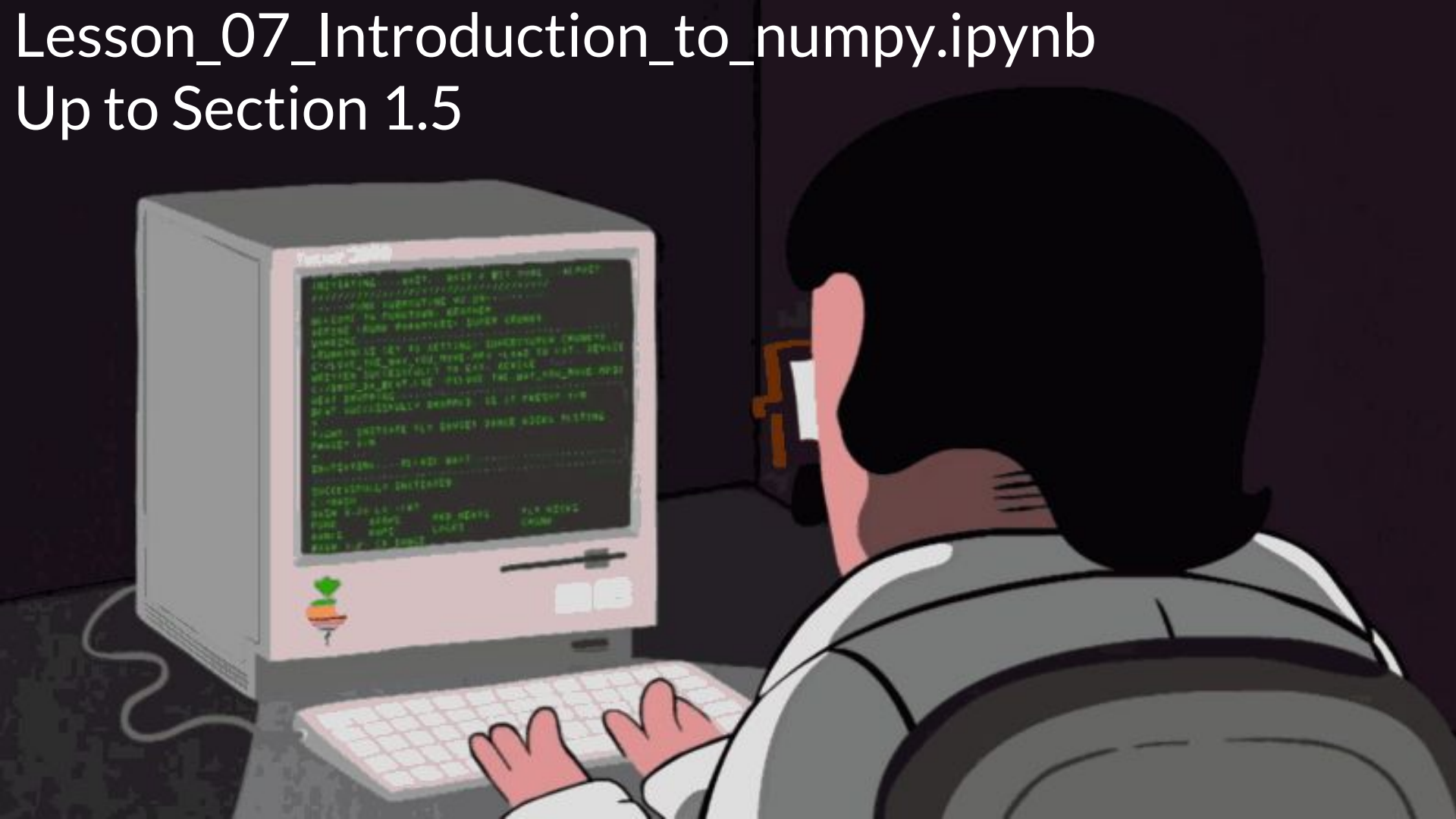
```
sel_lol = []

rows = data_lol[1:4]
for r in rows:
    new_row = r[:3]
    sel_lol.append(new_row)
```

NumPy method

```
sel_np = data_np[1:4,:3]
```

Comma separated row/column slice locations. Returns a 2D ndarray

Lesson_07_Introduction_to_numpy.ipynb
Up to Section 1.5



Vector Math (list of lists vs numpy)

```
import numpy as np

# create random (500,5) numpy arrays and list of lists
np_array = np.random.rand(500,5)
list_array = np_array.tolist()

def python_subset():
  filtered_cols = []
  for row in list_array:
    filtered_cols.append([row[1],row[2]])
  return filtered_cols

def numpy_subset():
  return np_array[:,1:3]
```

Vector Math (list of lists vs numpy)

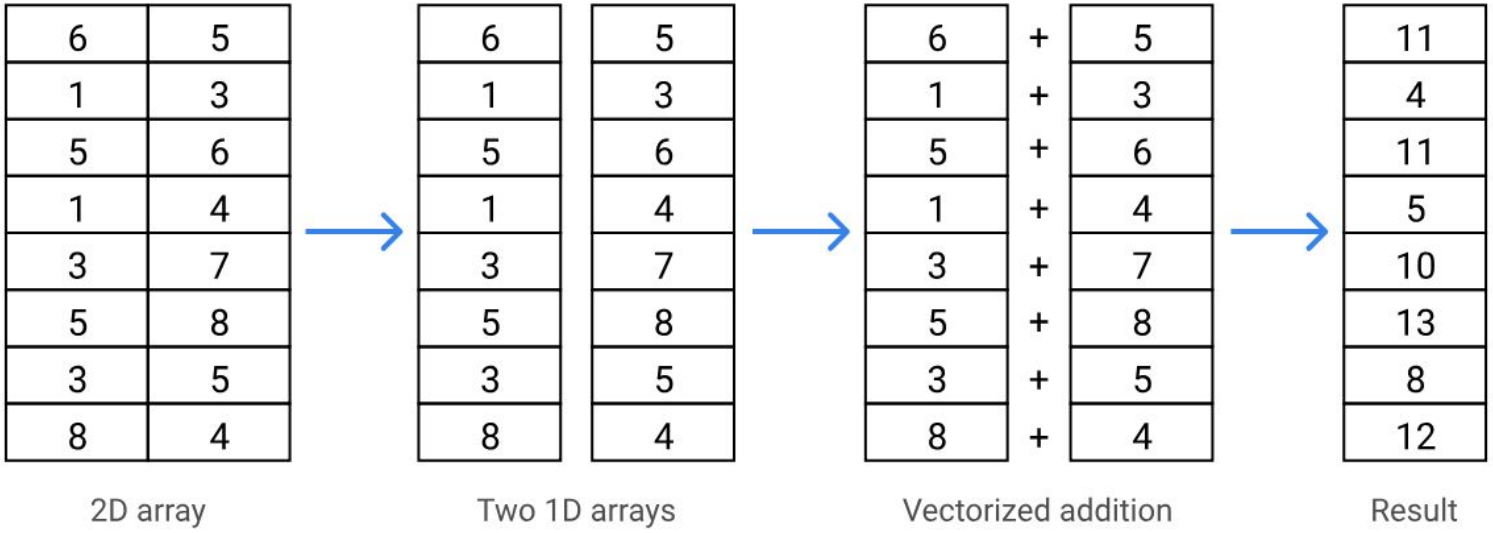
```
%%timeit -r 1 -n 1
list_of_list = python_subset()
```

```
1 loop, best of 1: 182 µs per loop
```

```
%%timeit -r 1 -n 1
numpy_array = numpy_subset()
```

```
1 loop, best of 1: 9.06 µs per loop
```

Vector Math (numpy)



Calculating Statistics for 1D ndarrays

Calculation	Function Representation	Method Representation
Calculate the minimum value of **trip_mph**	np.min(trip_mph)	trip_mph.min()
Calculate the maximum value of **trip_mph**	np.max(trip_mph)	trip_mph.max()
Calculate the mean average value of **trip_mph**	np.mean(trip_mph)	trip_mph.mean()
Calculate the median average value of **trip_mph**	np.median(trip_mph)	There is no ndarray median method

Calculating Statistics for 2D ndarrays

2D ndarray

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

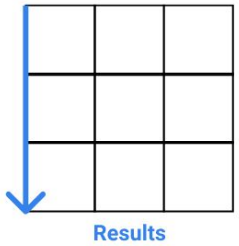
`ndarray.max(axis=0)`

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

3	1	4	3

Result

`ndarray.method(axis=0)`
Calculates along the **row** axis



Calculates result for each **column**.

2D ndarray

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

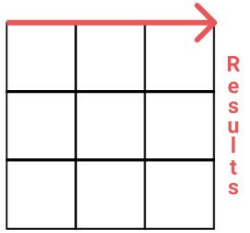
`ndarray.max(axis=1)`

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

Result

1
4
2
3

`ndarray.method(axis=1)`
Calculates along the **column** axis



Calculates result for each **row**.

Adding Rows and Columns to ndarrays (concatenate)

```
>>> print(ones)

    [[ 1  1  1]
     [ 1  1  1]]

>>> print(zeros)

    [ 0  0  0]
```

```
>>> print(ones.shape)

    (2, 3)

>>> print(zeros.shape)

    (3,)
```

Adding Rows and Columns to ndarrays (concatenate)

```
>>> zeros_2d = np.expand_dims(zeros,axis=0)

>>> print(zeros_2d)

    [[ 0  0  0]]

>>> print(zeros_2d.shape)

    (1, 3)
```

Adding Rows and Columns to ndarrays (concatenate)

```
>>> combined = np.concatenate([ones,zeros_2d],axis=0)

>>> print(combined)

    [[ 1  1  1]
     [ 1  1  1]
     [ 0  0  0]]
```

Sorting ndarrays

```
fruit = np.array(['orange', 'banana',
                  'apple', 'grape',
                  'cherry'])
```

orange	0
banana	1
apple	2
grape	3
cherry	4

```
sorted_fruit = fruit[sorted_order]
```

apple
banana
cherry
grape
orange

```
sorted_order = np.argsort(fruit)
```

2	1	4	3	0

Lesson_07_Introduction_to_numpy.ipynb
Up to Section 1.11