EEE110 Computer Programming

Week 12
Array-Oriented Programming with NumPy

Dr Kasım Zor

May 30, 2022

NumPy: **Num**erical **Py**thon

The Fundamental Package for Scientific Computing with Python

- Support multi-dimensional arrays
- Built-in array operations
- Simplified, but powerful array interactions (broadcasting)
- Integration of other languages such as Fortran, C, or C++
- Many of Python packages are built on NumPy (Pandas)

For further information related to array programming with NumPy, taking a glance at the following article is highly recommended:

- Harris et al. Array programming with NumPy. Nature, vol. 585, pp. 357–362, 2020. (Access Link)

NumPy: Introduction to ndarrays

ndarrays (Rank 1)

```
import numpy as np

an_array = np.array([3, 33, 33])

print(type(an_array))
## <class 'numpy.ndarray'>
print(an_array.shape)
## (3,)
print(an_array[0], an_array[1], an_array[2])
## 3 33 33
```


NumPy: Introduction to ndarrays

ndarrays (Rank 1)

```
an_array[0] = 888 #ndarrays are mutable

print(an_array)
## [888  33  33]
an_array[0] = 'Hello'
## Error in py_call_impl(callable, dots$args, dots$keywords):
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```

ndarrays (Rank 2)

```
another = np.array([[11,12,13], [21,22,23]])
print(another)
```

```
## [[11 12 13]
##  [21 22 23]]
```

```
print('The shape is 2 rows, 3 columns: ', another.shape)
```

```
## The shape is 2 rows, 3 columns:  (2, 3)
```

```
print('Accessing elements [0,0], [0,1], and [1,0] of the \
ndarray')
```

```
## Accessing elements [0,0], [0,1], and [1,0] of the ndarray
```

```
print(another [0,0], ',', another[0,1], ',', another[1,0])
```

```
## 11 , 12 , 21
```


Question 1 - ndarrays (Rank 2)

Which of the followings can result in a two dimensional array?

```
np.array([[11,12,13],[21,22,23]]) #1
np.array([11,12,13,21,22,23])     #2
np.array([11,12,13],[21,22,23])   #3
np.array[[11,12,13,21,22,23]]     #4
```


Solution 1 - ndarrays (Rank 2)

```
np.array([[11,12,13],[21,22,23]]) #1
```

```
## array([[11, 12, 13],
##        [21, 22, 23]])
```

```
np.array([11,12,13,21,22,23])     #2
```

```
## array([11, 12, 13, 21, 22, 23])
```


Solution 1 - ndarrays (Rank 2)

```
np.array([11,12,13],[21,22,23])   #3
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords):
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```

```
np.array[[11,12,13,21,22,23]]     #4
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords):
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```

NumPy: Introduction to ndarrays

Many NumPy Arrays

```
ex1 = np.zeros((2,2))
print(ex1)
```

```
## [[0. 0.]
##  [0. 0.]]
```

```
ex2 = np.full((2,2), 9.0)
print(ex2)
```

```
## [[9. 9.]
##  [9. 9.]]
```

```
ex3 = np.eye(2,2)
print(ex3)
```

```
## [[1. 0.]
##  [0. 1.]]
```


NumPy: Introduction to ndarrays

Many NumPy Arrays

```
ex4 = np.ones((1,2))
print(ex4)
```

```
## [[1. 1.]]
```

```
print(ex4.shape)
```

```
## (1, 2)
```

```
ex5 = np.random.random((2,2))
print(ex5)
```

```
## [[0.4869263  0.19594295]
##  [0.39021172 0.94495498]]
```


NumPy: Array Indexing

Slice Indexing

```
an_array = np.array([[11,12,13,14],[21,22,23,24],[31,32,33,34]
print(an_array)
```

```
## [[11 12 13 14]
##  [21 22 23 24]
##  [31 32 33 34]]
```

```
a_slice = an_array[:2,1:3] #reference
print(a_slice)
```

```
## [[12 13]
##  [22 23]]
```


NumPy: Array Indexing

Slice Indexing

```
print('Before:', an_array[0,1])
```

```
## Before: 12
```

```
a_slice[0,0] = 1000
print('After:', an_array[0,1])
```

```
## After: 1000
```

```
print(an_array)
```

```
## [[  11 1000   13   14]
##  [  21   22   23   24]
##  [  31   32   33   34]]
```

NumPy: Array Indexing

Slice Indexing

```
an_array = np.array([[11,12,13,14],[21,22,23,24],[31,32,33,34]
print(an_array)
```

```
## [[11 12 13 14]
##  [21 22 23 24]
##  [31 32 33 34]]
```

```
a_slice = np.array(an_array[:2,1:3]) #copy
print(a_slice)
```

```
## [[12 13]
##  [22 23]]
```


NumPy: Array Indexing

Slice Indexing

```
print('Before:', an_array[0,1])
```

```
## Before: 12
```

```
a_slice[0,0] = 1000
print('After:', an_array[0,1])
```

```
## After: 12
```

```
print(an_array)
```

```
## [[11 12 13 14]
##  [21 22 23 24]
##  [31 32 33 34]]
```


NumPy: Array Indexing

Question 2 - Slice Indexing

Consider that you try to access rows of a 3 x 3 numpy array called 'arr' using the following command:

```
arr[:2,]
```

How many rows will be returned?

- 1
- 2
- 3
- IndexError: index out of bounds

NumPy: Array Indexing

Solution 2 - Slice Indexing

```
arr = np.eye(3,3)
print(arr)
```

```
## [[1. 0. 0.]
##  [0. 1. 0.]
##  [0. 0. 1.]]
```

```
print(arr[:2,])
```

```
## [[1. 0. 0.]
##  [0. 1. 0.]]
```

NumPy: Array Indexing

Both Integer Indexing and Slice Indexing

```
an_array = np.array([[11,12,13,14],[21,22,23,24],[31,32,33,34]
print(an_array)
```

```
## [[11 12 13 14]
##  [21 22 23 24]
##  [31 32 33 34]]
```

```
row_rank1 = an_array[1, :]
print(row_rank1, row_rank1.shape)
```

```
## [21 22 23 24] (4,)
```

```
row_rank2 = an_array[1:2, :]
print(row_rank2, row_rank2.shape)
```

```
## [[21 22 23 24]] (1, 4)
```


NumPy: Array Indexing

Both Integer Indexing and Slice Indexing

```
col_rank1 = an_array[:, 1]
print(col_rank1,col_rank1.shape)
```

```
## [12 22 32] (3,)
```

```
col_rank2 = an_array[:, 1:2]
print(col_rank2,col_rank2.shape)
```

```
## [[12]
##  [22]
##  [32]] (3, 1)
```


NumPy: Array Indexing

Array Indexing for Changing Elements

```
an_array = np.array([[11,12,13],[21,22,23],[31,32,33],[41,42,4
print('Original array:')
```

```
## Original array:
```

```
print(an_array)
```

```
## [[11 12 13]
##  [21 22 23]
##  [31 32 33]
##  [41 42 43]]
```


NumPy: Array Indexing

Array Indexing for Changing Elements

```
col_indices = np.array([0,1,2,0])
print('Col indices picked : ', col_indices)
```

```
## Col indices picked :  [0 1 2 0]
```

```
row_indices = np.arange(4)
print('Rows indices picked : ', row_indices)
```

```
## Rows indices picked :  [0 1 2 3]
```

```
for row,col in zip(row_indices,col_indices):
  print(row,', ',col)
```

```
## 0 ,  0
## 1 ,  1
## 2 ,  2
## 3 ,  0
```

NumPy: Array Indexing

Array Indexing for Changing Elements

```
print('Values in the array at those indices: ', \
an_array[row_indices,col_indices])
```

```
## Values in the array at those indices:  [11 22 33 41]
```

```
an_array[row_indices,col_indices] += 100000
print('Changed array:\n', an_array)
```

```
## Changed array:
##  [[100011     12     13]
##  [    21 100022     23]
##  [    31     32 100033]
##  [100041     42     43]]
```


NumPy: Boolean Indexing

Array Indexing for Changing Elements

```
an_array = np.array([[11,12],[21,22],[31,32]])
print(an_array)
```

```
## [[11 12]
##  [21 22]
##  [31 32]]
```

```
filter = an_array > 15
filter
```

```
## array([[False, False],
##        [ True,  True],
##        [ True,  True]])
```

```
print(an_array[filter])
```

```
## [21 22 31 32]
```


NumPy: Boolean Indexing

Array Indexing for Changing Elements

```
an_array[an_array > 15]
```

```
## array([21, 22, 31, 32])
```

```
an_array[(an_array > 20) & (an_array < 30)]
```

```
## array([21, 22])
```

```
an_array[an_array % 2 == 0]
```

```
## array([12, 22, 32])
```

```
an_array[an_array % 2 == 0] += 100
print(an_array)
```

```
## [[ 11 112]
##  [ 21 122]
##  [ 31 132]]
```


NumPy: Data Types and Operations

Data Types

```
ex1 = np.array([11,12])
print(ex1.dtype)
```

```
## int64
```

```
ex2 = np.array([11.0,12.0])
print(ex2.dtype)
```

```
## float64
```

```
ex3 = np.array([11,21], dtype=np.int64)
print(ex3.dtype)
```

```
## int64
```

NumPy: Data Types and Operations

Data Types

```
ex4 = np.array([11.1,12.7],dtype=np.int64)
print(ex4.dtype)
## int64
print(ex4)
## [11 12]
ex5 = np.array([11,21], dtype=np.float64)
print(ex5.dtype)
## float64
print(ex5)
## [11. 21.]
```


NumPy: Data Types and Operations

Arithmetic Array Operations

```
x = np.array([[111,112],[121,122]], dtype=np.int)
print(x)
## [[111 112]
##  [121 122]]
y = np.array([[211.1,212.1], [221.1, 222.1]], dtype=np.float64
print(y)
## [[211.1 212.1]
##  [221.1 222.1]]
```


NumPy: Data Types and Operations

Arithmetic Array Operations: Addition and Subtraction

```
print(x + y)
## [[322.1 324.1]
##  [342.1 344.1]]
print(np.add(x, y))
## [[322.1 324.1]
##  [342.1 344.1]]
print(x - y)
## [[-100.1 -100.1]
##  [-100.1 -100.1]]
print(np.subtract(x, y))
## [[-100.1 -100.1]
##  [-100.1 -100.1]]
```


NumPy: Data Types and Operations

Arithmetic Array Operations: Multiplication and Division

```
print(x * y)
## [[23432.1 23755.2]
##  [26753.1 27096.2]]
print(np.multiply(x, y))
## [[23432.1 23755.2]
##  [26753.1 27096.2]]
print(x / y)
## [[0.52581715 0.52805281]
##  [0.54726368 0.54930212]]
print(np.divide(x, y))
## [[0.52581715 0.52805281]
##  [0.54726368 0.54930212]]
```

NumPy: Data Types and Operations

Arithmetic Array Operations: Square Root and Exponentiation

```
print(np.sqrt(x))
```

```
## [[10.53565375 10.58300524]
##  [11.         11.04536102]]
```

```
print(np.sqrt(y))
```

```
## [[14.52928078 14.56365339]
##  [14.86943173 14.90301983]]
```

```
print(np.exp(x))
```

```
## [[1.60948707e+48 4.37503945e+48]
##  [3.54513118e+52 9.63666567e+52]]
```

```
print(np.exp(y))
```

```
## [[4.78151068e+91 1.29974936e+92]
##  [1.05319781e+96 2.86288848e+96]]
```


NumPy: Data Types and Operations

Statistical Operations

```
arr = 10 * np.random.randn(2,4)
print(arr)
```

```
## [[ 20.46587678  -3.27484766 -13.25934005  -6.57582682]
##  [ -3.80967172 -27.37168851 -11.61494363  -1.84470507]]
```

```
print(arr.mean())
```

```
## -5.9106433349440035
```

```
print(arr.mean(axis=1)) #compute the means by row
```

```
## [ -0.66103444 -11.16025223]
```

```
print(arr.mean(axis=0)) #compute the means by column
```

```
## [  8.32810253 -15.32326809 -12.43714184  -4.21026594]
```


NumPy: Data Types and Operations

Statistical Operations

```
print(arr)
```

```
## [[ 20.46587678  -3.27484766 -13.25934005  -6.57582682]
##  [ -3.80967172 -27.37168851 -11.61494363  -1.84470507]]
```

```
print(arr.sum())
```

```
## -47.28514667955203
```

```
print(np.median(arr, axis=1)) #compute the medians by row
```

```
## [-4.92533724 -7.71230768]
```

```
print(np.median(arr, axis=0)) #compute the medians by column
```

```
## [  8.32810253 -15.32326809 -12.43714184  -4.21026594]
```


NumPy: Data Types and Operations

Statistical Operations

```
print(arr)
```

```
## [[ 20.46587678  -3.27484766 -13.25934005  -6.57582682]
##  [ -3.80967172 -27.37168851 -11.61494363  -1.84470507]]
```

```
print(np.min(arr, axis=0))
```

```
## [ -3.80967172 -27.37168851 -13.25934005  -6.57582682]
```

```
print(np.max(arr, axis=1))
```

```
## [20.46587678 -1.84470507]
```

NumPy: Data Types and Operations

Sorting Operations

```
unsorted = np.random.randn(3, 4)
print(unsorted)
```

```
## [[ 0.76859203 -1.83239443  1.66864301  0.38893783]
##  [ 0.49874272 -1.02167097  0.65179828  0.28803875]
##  [ 0.61714062 -1.56626636  0.22030067 -0.80298711]]
```

```
sorted = np.array(unsorted) #copy created not to lose the orig
sorted.sort()
print(sorted)
```

```
## [[-1.83239443  0.38893783  0.76859203  1.66864301]
##  [-1.02167097  0.28803875  0.49874272  0.65179828]
##  [-1.56626636 -0.80298711  0.22030067  0.61714062]]
```


NumPy: Data Types and Operations

Finding Unique Elements

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

```
print(np.unique(array))
```

```
## [1 2 4]
```

Set Operations with np.array Data Type

```
team1 = np.array(['Ronaldo', 'Ronaldinho', 'Zidane'])
print(team1)
```

```
## ['Ronaldo' 'Ronaldinho' 'Zidane']
```

```
team2 = np.array(['Zlatan','Ronaldo','Messi'])
print(team2)
```

```
## ['Zlatan' 'Ronaldo' 'Messi']
```


NumPy: Data Types and Operations

Set Operations with np.array Data Type

```
print(np.intersect1d(team1, team2))
```

```
## ['Ronaldo']
```

```
print(np.union1d(team1, team2))
```

```
## ['Messi' 'Ronaldinho' 'Ronaldo' 'Zidane' 'Zlatan']
```

```
print(np.setdiff1d(team1, team2))
```

```
## ['Ronaldinho' 'Zidane']
```

```
print(np.in1d(team1, team2))
```

```
## [ True False False]
```


NumPy: Broadcasting

Broadcasting

```
start = np.zeros((4,3))
print(start)
```

```
## [[0. 0. 0.]
##  [0. 0. 0.]
##  [0. 0. 0.]
##  [0. 0. 0.]]
```

```
add_rows = np.array([1,0,2])
print(add_rows)
```

```
## [1 0 2]
```

Broadcasting

```
y = start + add_rows
print(y)
```

```
## [[1. 0. 2.]
##  [1. 0. 2.]
##  [1. 0. 2.]
##  [1. 0. 2.]]
```

```
add_cols = np.array([[0,1,2,3]])
add_cols = add_cols.T
print(add_cols)
```

```
## [[0]
##  [1]
##  [2]
##  [3]]
```


Broadcasting

```
y = start + add_cols
print(y)
```

```
## [[0. 0. 0.]
##  [1. 1. 1.]
##  [2. 2. 2.]
##  [3. 3. 3.]]
```

```
add_scalar = np.array([1])
print(start+add_scalar)
```

```
## [[1. 1. 1.]
##  [1. 1. 1.]
##  [1. 1. 1.]
##  [1. 1. 1.]]
```


Question 3 - Broadcasting

```
a = np.array([[0,0],[0,0]])
b1 = np.array([1,1])
b2 = 1
```

Do $a + b1$ and $a + b2$ result in the same matrix?

- Yes, it is true.
- No, it is false.

Solution 3 - Broadcasting

```
a = np.array([[0,0],[0,0]])
b1 = np.array([1,1])
b2 = 1
(a + b1) == (a + b2)
```

```
## array([[ True,  True],
##        [ True,  True]])
```

```
print(a + b1)
```

```
## [[1 1]
##  [1 1]]
```

```
print(a + b2)
```

```
## [[1 1]
##  [1 1]]
```

NumPy: Speed Test

ndarray vs Lists

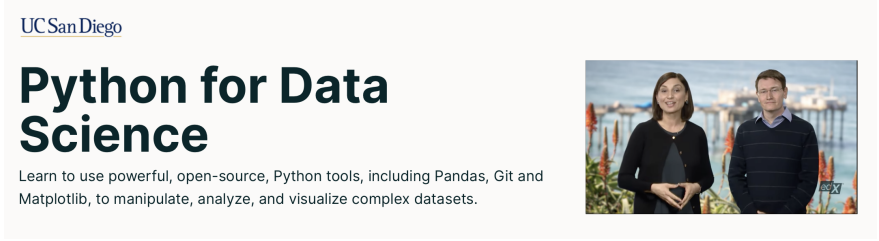
```
from numpy import arange
from timeit import Timer
size    = 1000000
timeits = 1000

# create the ndarray with values 0,1,2...,size-1
nd_array = arange(size)
print(type(nd_array))

## <class 'numpy.ndarray'>

# create the list with values 0,1,2...,size-1
a_list = list(range(size))
print(type(a_list))

## <class 'list'>
```


NumPy: Speed Test

ndarray vs Lists

```
numpy = Timer('nd_array.sum()', 'from __main__ import nd_array
print('Time ndarray: %.4f seconds' \
% (numpy.timeit(timeits)/timeits))

## Time ndarray: 0.0003 seconds

list = Timer('sum(a_list)', 'from __main__ import a_list')
print("Time list: %.4f seconds" \
% (list.timeit(timeits)/timeits))

## Time list: 0.0061 seconds

print('list/numpy: %.2f' % ((list.timeit(timeits)/timeits) \
/ (numpy.timeit(timeits)/timeits)))

## list/numpy: 19.51
```


References



Aforementioned contents are adapted from the course:

- 'Python for Data Science' (Access Link) taught on edX by Dr İlkay Altıntaş and Dr Leo Porter from the University of California at San Diego.