Python Lists

- **No support** for vectorized operations
- **No fixed** type elements
- For loops **not efficient**

NumPy Arrays

- **Supports** vectorized operations (addition, multiplications)
- **Fixed** data type
- More **efficient**

Cleaner with More Functions

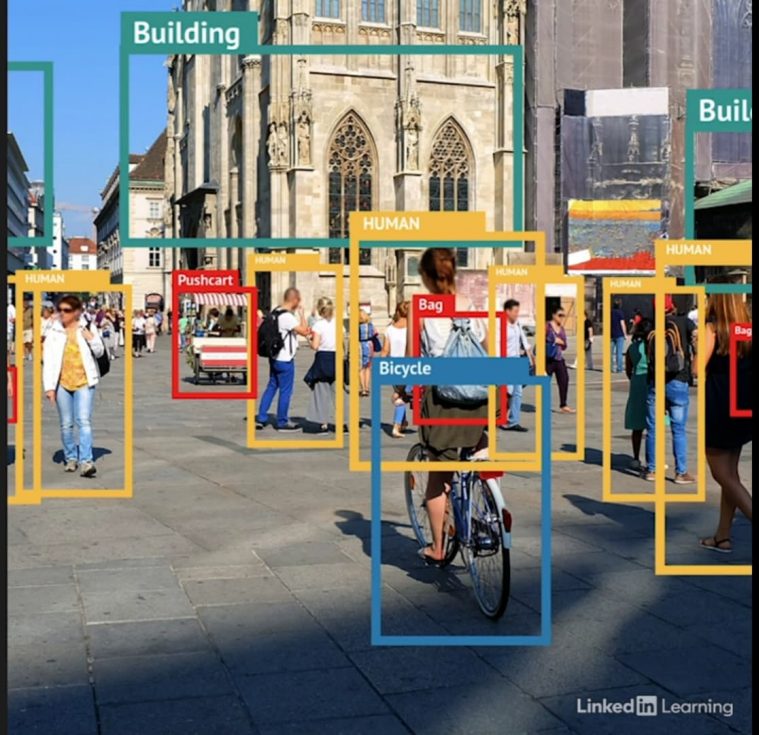
Code is cleaner and has more advanced built-in functions.

Cleaner Code

- Same tasks with fewer lines
- Fewer loops
- Advanced mathematical functions

Applications

- Machine learning and computational packages
- Essential library for scientific projects



Python List vs. NumPy Array

Python Lists

```
list = [1,2,3]
```

NumPy Arrays

```
import numpy as np

array = np.array([1,2,3])
```

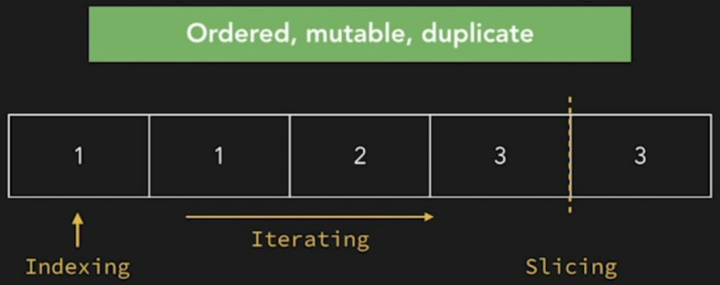
Performance/memory

Python Lists

```
list = [1,1,2,3,3]
```

NumPy Arrays

```
import numpy as np

array = np.array([1,1,2,3,3])
```



Python Lists

```
list = ["Apple",123]
```

[str,int]

```
list.append("Banana")
-> ["Apple",123, "Banana"]
```

NumPy Arrays

```
import numpy as np
```

```
array = np.array([1,1,2,3,3])
```

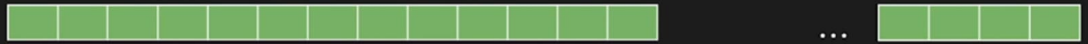
[int,int,...]

NumPy Arrays Consume Less Memory

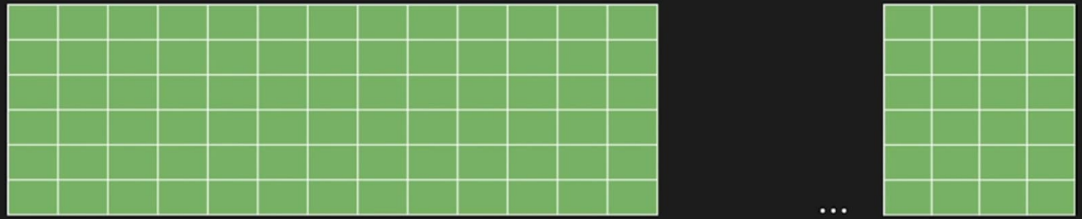
1,000 Elements

NumPy Arrays

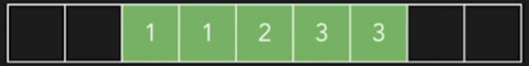
Six times less memory



Python Lists



NumPy Arrays `np.array([1,1,2,3,3])`



Python Lists `list = [1, "Apple"]`

NumPy Arrays Are Faster

Python Lists

```
list = [1,2,3,....]
```

```
[i*5 for i in list]
```

1 ms

NumPy Arrays

```
import numpy as np
```

```
array = np.array([1,2,3,…])
```

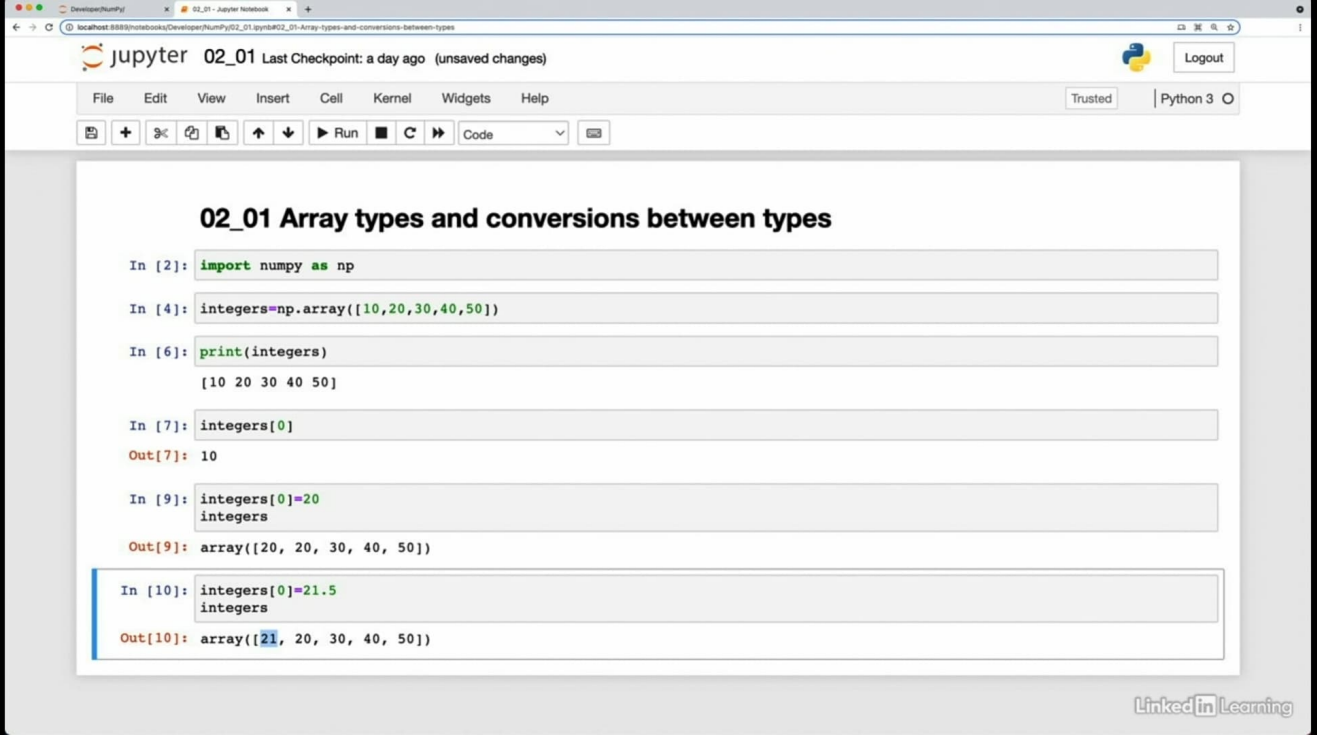
```
array * 5
```

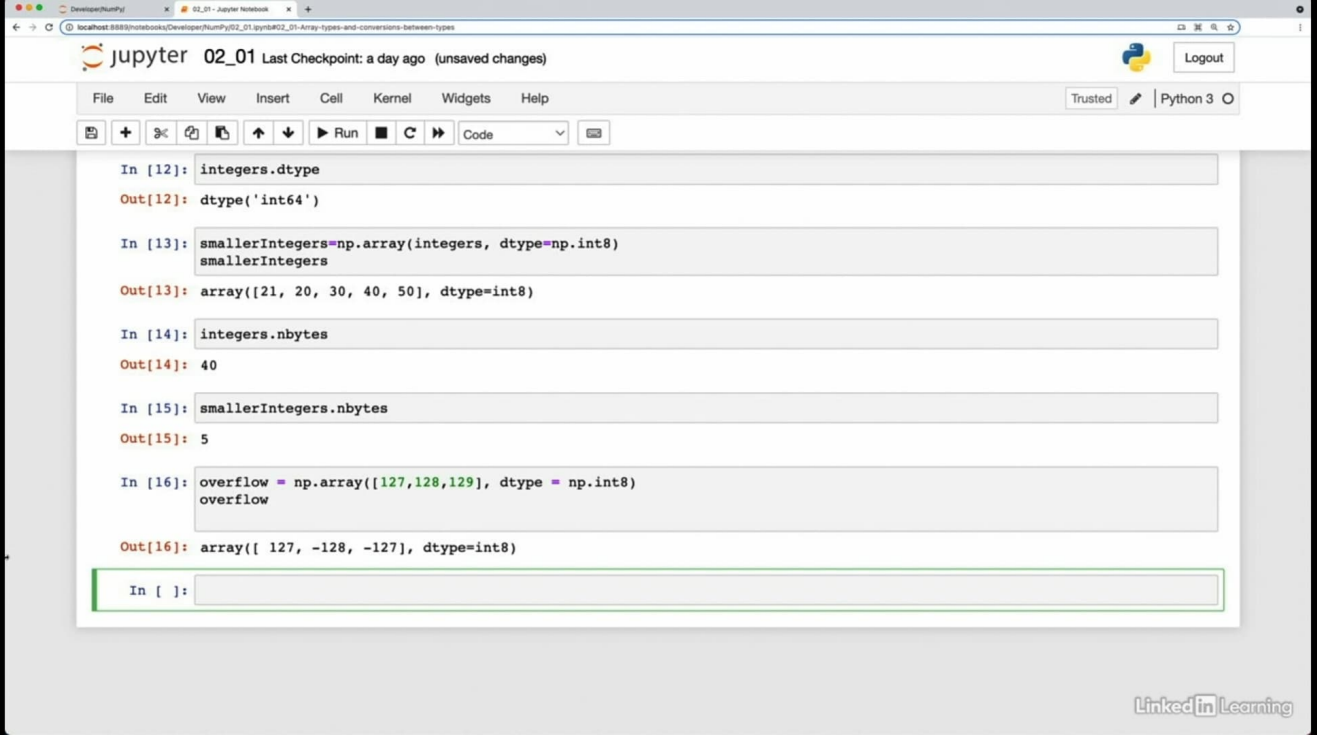
1/100 ms

100 times faster

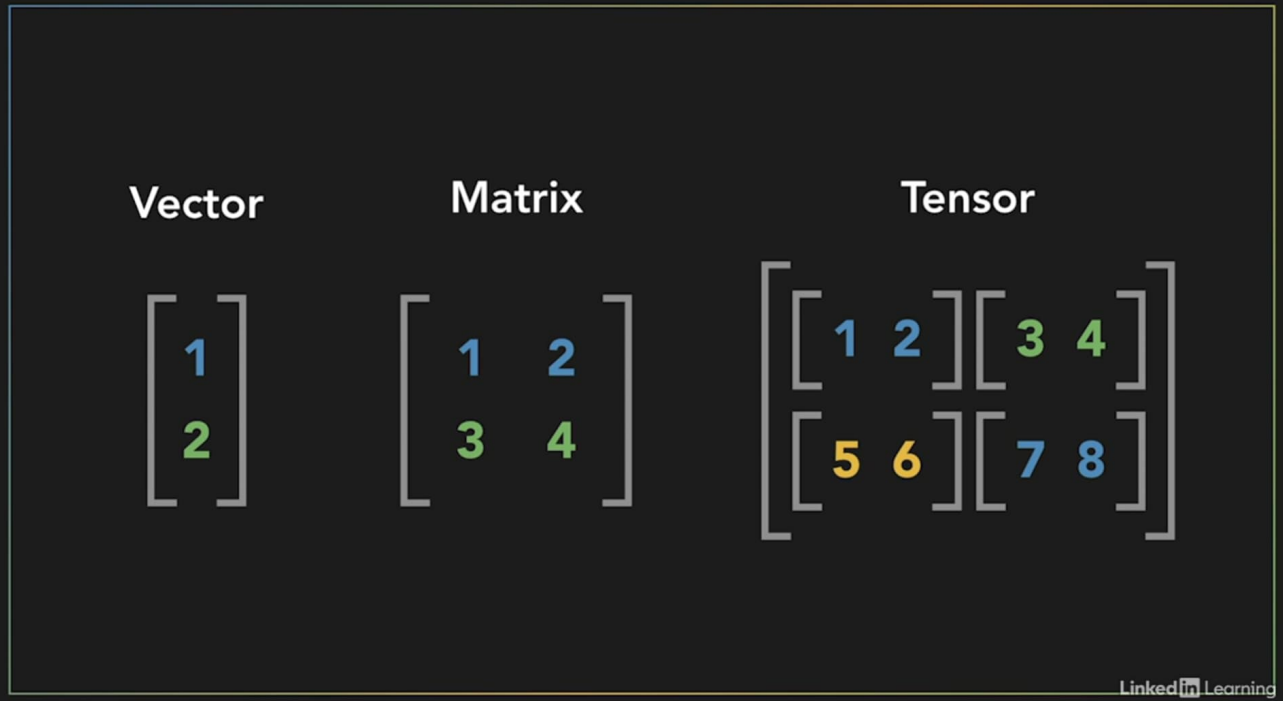
NumPy Arrays Are Convenient to Use

Obvious choice for use in data analytics and data science


Developer/NumPy/
02_01 - Jupyter Notebook
localhost:8889/notebooks/Developer/NumPy/02_01.ipynb#02_01-Array-types-and-conversions-between-types
jupyter 02_01 Last Checkpoint: a day ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
02_01 Array types and conversions between types
In [2]: import numpy as np
In [4]: integers=np.array([10,20,30,40,50])
In [6]: print(integers)
[10 20 30 40 50]
In [7]: integers[0]
Out[7]: 10
In [9]: integers[0]=20
integers
Out[9]: array([20, 20, 30, 40, 50])
In [10]: integers[0]=21.5
integers
Out[10]: array([21, 20, 30, 40, 50])
LinkedIn Learning


Developer/NumPy/
02_01 - Jupyter Notebook
localhost:8889/notebooks/Developer/NumPy/02_01.ipynb#02_01-Array-types-and-conversions-between-types
jupyter 02_01 Last Checkpoint: a day ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [12]: integers.dtype
Out[12]: dtype('int64')
In [13]: smallerIntegers=np.array(integers, dtype=np.int8)
smallerIntegers
Out[13]: array([21, 20, 30, 40, 50], dtype=int8)
In [14]: integers.nbytes
Out[14]: 40
In [15]: smallerIntegers.nbytes
Out[15]: 5
In [16]: overflow = np.array([127,128,129], dtype = np.int8)
overflow
Out[16]: array([127, -128, -127], dtype=int8)
In [ ]:
LinkedIn Learning

Multidimensional Arrays


Vector
Matrix
Tensor
1
2
1 2
3 4
1 2 3 4
5 6 7 8
LinkedIn Learning

jupyter 02_02 Last Checkpoint: 6 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run | Code

Multidimensional arrays

In [1]:
```
import numpy as np
```

In [2]:
```
nums=np.array([[1,2,3,4,5],[6,7,8,9,10]])
nums
```

Out[2]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [3]:
```
nums[0,0]
```

Out[3]:
```
1
```

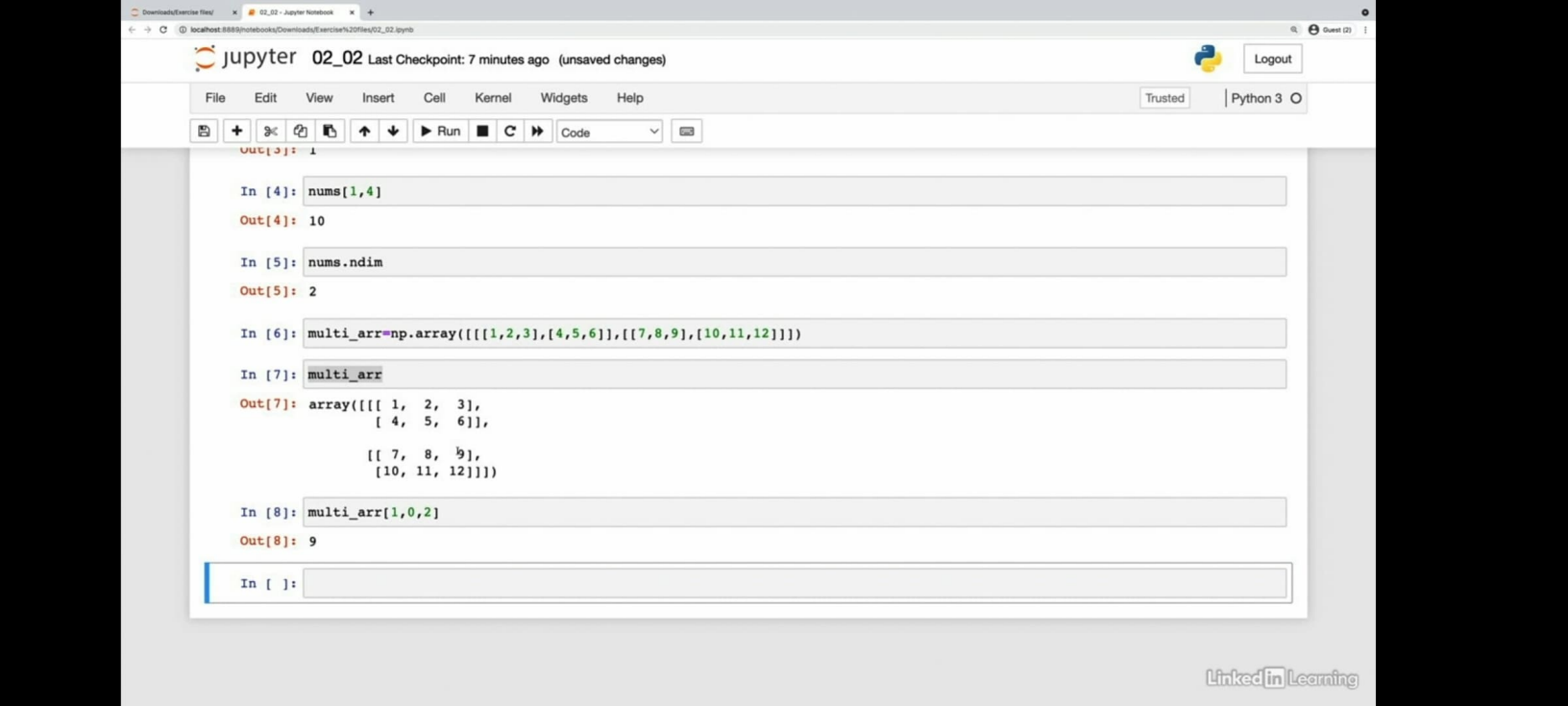
In [4]:
```
nums[1,4]
```

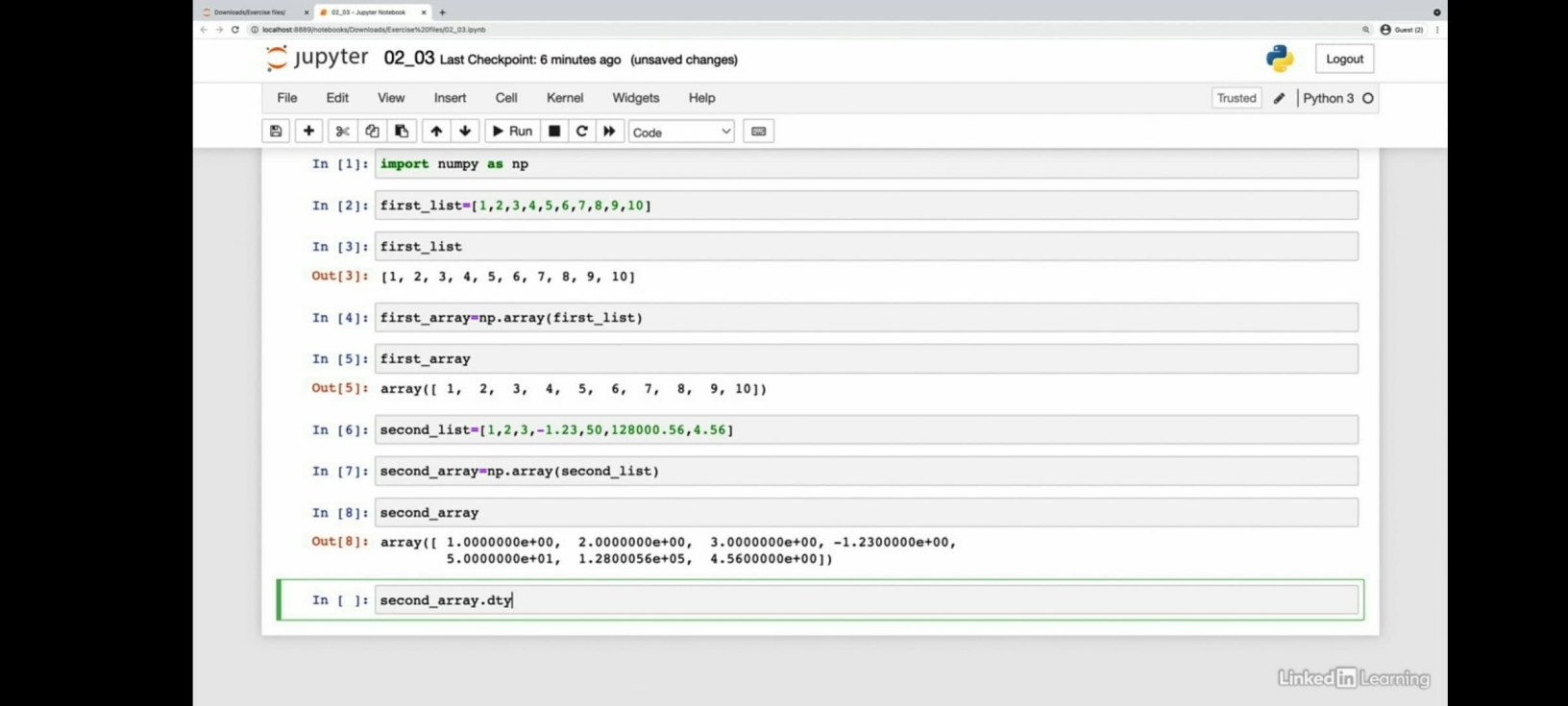
Out[4]:
```
10
```

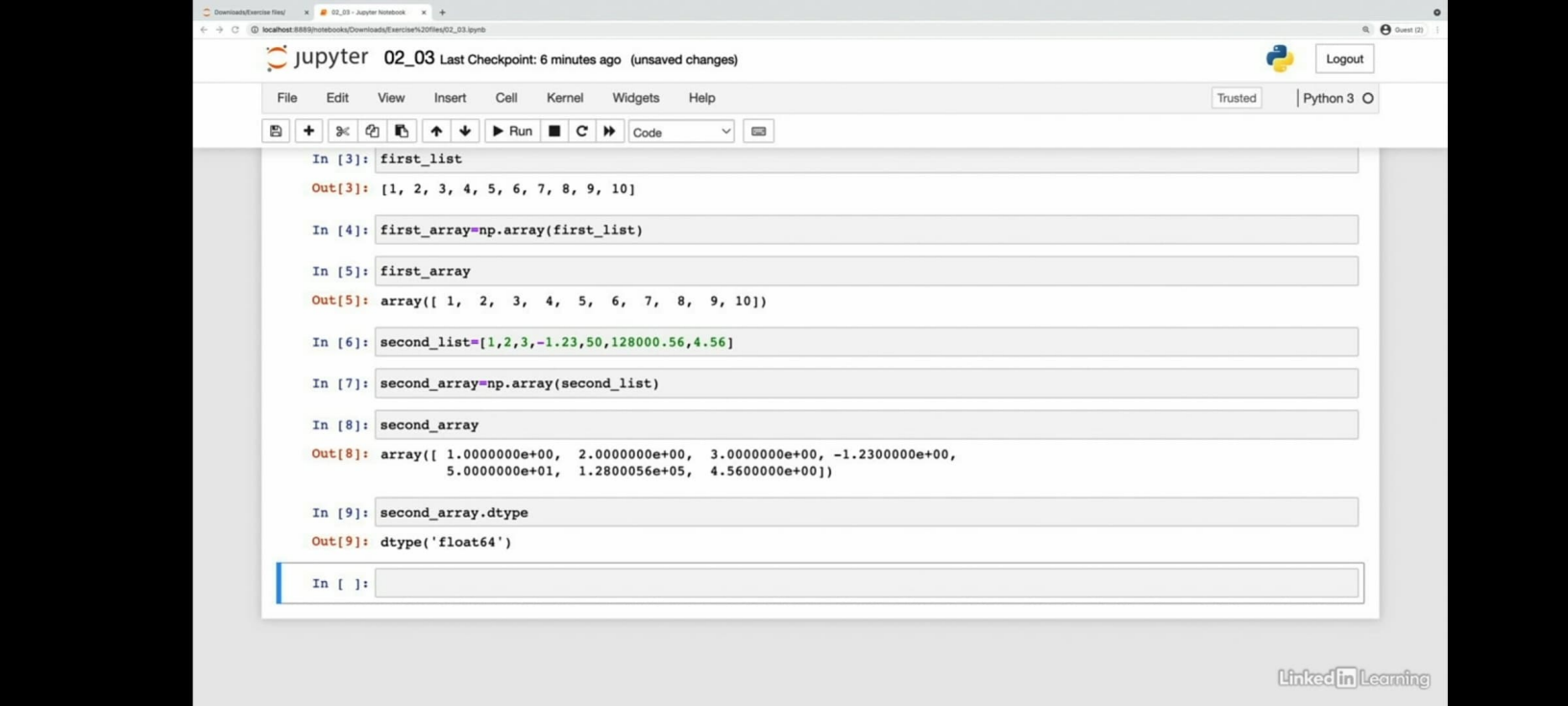
In [5]:
```
nums.ndim
```

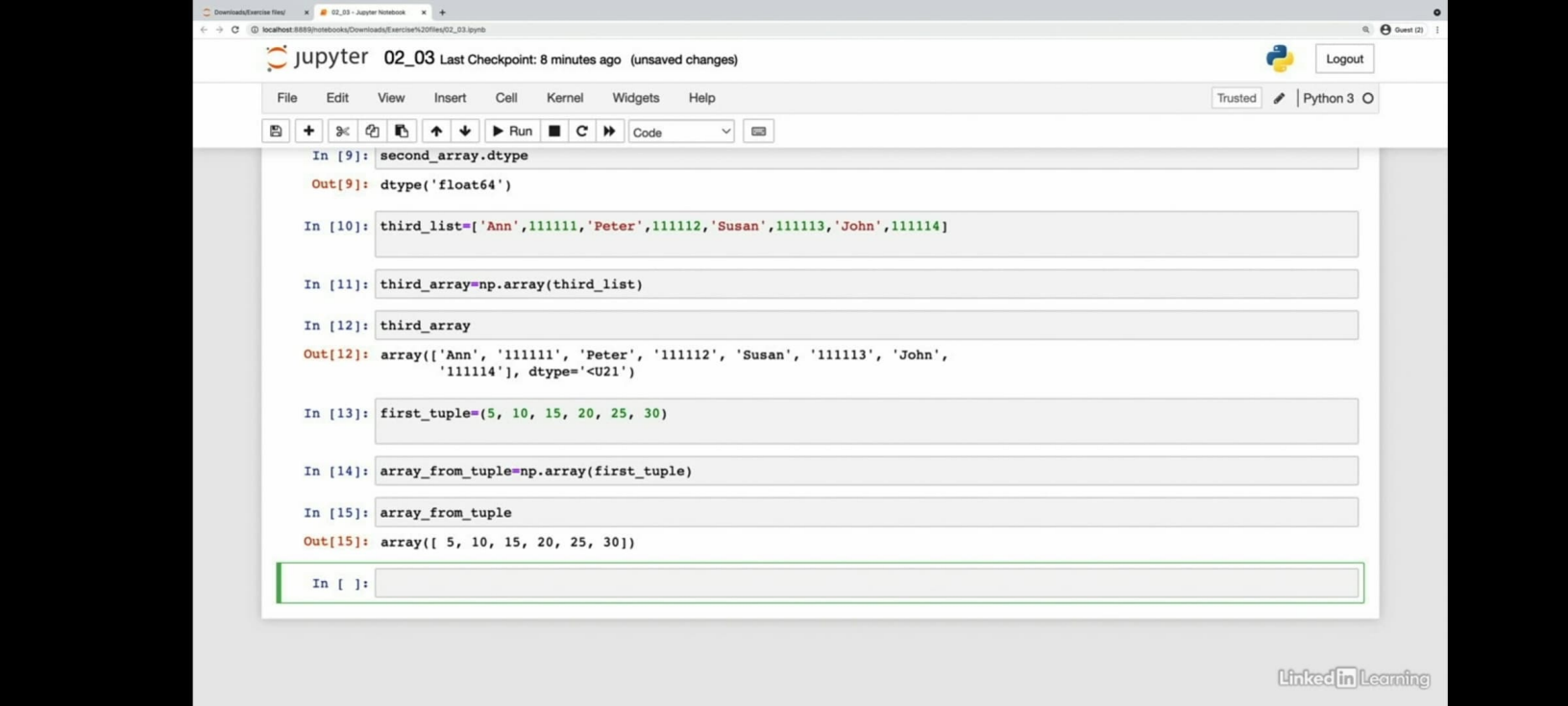
Out[5]:
```
2
```

In [ ]:


Downloads/Exercise files/
02_02 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/02_02.ipynb
Guest (2)
jupyter 02_02 Last Checkpoint: 7 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Out[3]: 1
In [4]: nums[1,4]
Out[4]: 10
In [5]: nums.ndim
Out[5]: 2
In [6]: multi_arr=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
In [7]: multi_arr
Out[7]: array([[[1, 2, 3],
[4, 5, 6]],
[[7, 8, 9],
[10, 11, 12]]])
In [8]: multi_arr[1,0,2]
Out[8]: 9
In [ ]:
LinkedIn Learning


Downloads/Exercise files/
02_03 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20Files/02_03.ipynb
Guest (2)
jupyter 02_03 Last Checkpoint: 6 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [1]: import numpy as np
In [2]: first_list=[1,2,3,4,5,6,7,8,9,10]
In [3]: first_list
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In [4]: first_array=np.array(first_list)
In [5]: first_array
Out[5]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
In [6]: second_list=[1,2,3,-1.23,50,128000.56,4.56]
In [7]: second_array=np.array(second_list)
In [8]: second_array
Out[8]: array([1.0000000e+00, 2.0000000e+00, 3.0000000e+00, -1.2300000e+00,
5.0000000e+01, 1.2800056e+05, 4.5600000e+00])
In [ ]: second_array.dty
LinkedIn Learning


Downloads/Exercise files/
02_03 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20Files/02_03.ipynb
Guest (2)
jupyter 02_03 Last Checkpoint: 6 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [3]: first_list
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In [4]: first_array=np.array(first_list)
In [5]: first_array
Out[5]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
In [6]: second_list=[1,2,3,-1.23,50,128000.56,4.56]
In [7]: second_array=np.array(second_list)
In [8]: second_array
Out[8]: array([1.0000000e+00, 2.0000000e+00, 3.0000000e+00, -1.2300000e+00,
5.0000000e+01, 1.2800056e+05, 4.5600000e+00])
In [9]: second_array.dtype
Out[9]: dtype('float64')
In [ ]:
LinkedIn Learning


Downloads/Exercise files/
02_03 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/02_03.ipynb
Guest (2)
jupyter
02_03 Last Checkpoint: 8 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [9]: second_array.dtype
Out[9]: dtype('float64')
In [10]: third_list=['Ann',111111,'Peter',111112,'Susan',111113,'John',111114]
In [11]: third_array=np.array(third_list)
In [12]: third_array
Out[12]: array(['Ann', '111111', 'Peter', '111112', 'Susan', '111113', 'John',
'111114'], dtype='<U21')
In [13]: first_tuple=(5, 10, 15, 20, 25, 30)
In [14]: array_from_tuple=np.array(first_tuple)
In [15]: array_from_tuple
Out[15]: array([5, 10, 15, 20, 25, 30])
In [ ]:
LinkedIn Learning

Intrisic NumPy array creation

In [1]:
```
import numpy as np
```

In [2]:
```
integers_array=np.arange(10)
integers_array
```

Out[2]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [3]:
```
integers_second_array=np.arange(100,130)
integers_second_array
```

Out[3]:
```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
       113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
       126, 127, 128, 129])
```

In [4]:
```
integers_third_array=np.arange(100,151,2)
integers_third_array
```

Out[4]:
```
array([100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124,
       126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150])
```

In [ ]:

Out[3]:
```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
       113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
       126, 127, 128, 129])
```

In [4]:
```
integers_third_array=np.arange(100,151,2)
integers_third_array
```

Out[4]:
```
array([100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124,
       126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150])
```

In [5]:
```
first_floats_arr=np.linspace(10,20)
first_floats_arr
```

Out[5]:
```
array([10.        , 10.20408163, 10.40816327, 10.6122449 , 10.81632653,
       11.02040816, 11.2244898 , 11.42857143, 11.63265306, 11.83673469,
       12.04081633, 12.24489796, 12.44897959, 12.65306122, 12.85714286,
       13.06122449, 13.26530612, 13.46938776, 13.67346939, 13.87755102,
       14.08163265, 14.28571429, 14.48979592, 14.69387755, 14.89795918,
       15.10204082, 15.30612245, 15.51020408, 15.71428571, 15.91836735,
       16.12244898, 16.32653061, 16.53061224, 16.73469388, 16.93877551,
       17.14285714, 17.34693878, 17.55102041, 17.75510204, 17.95918367,
       18.16326531, 18.36734694, 18.57142857, 18.7755102 , 18.97959184,
       19.18367347, 19.3877551 , 19.59183673, 19.79591837, 20.        ])
```

In [ ]:

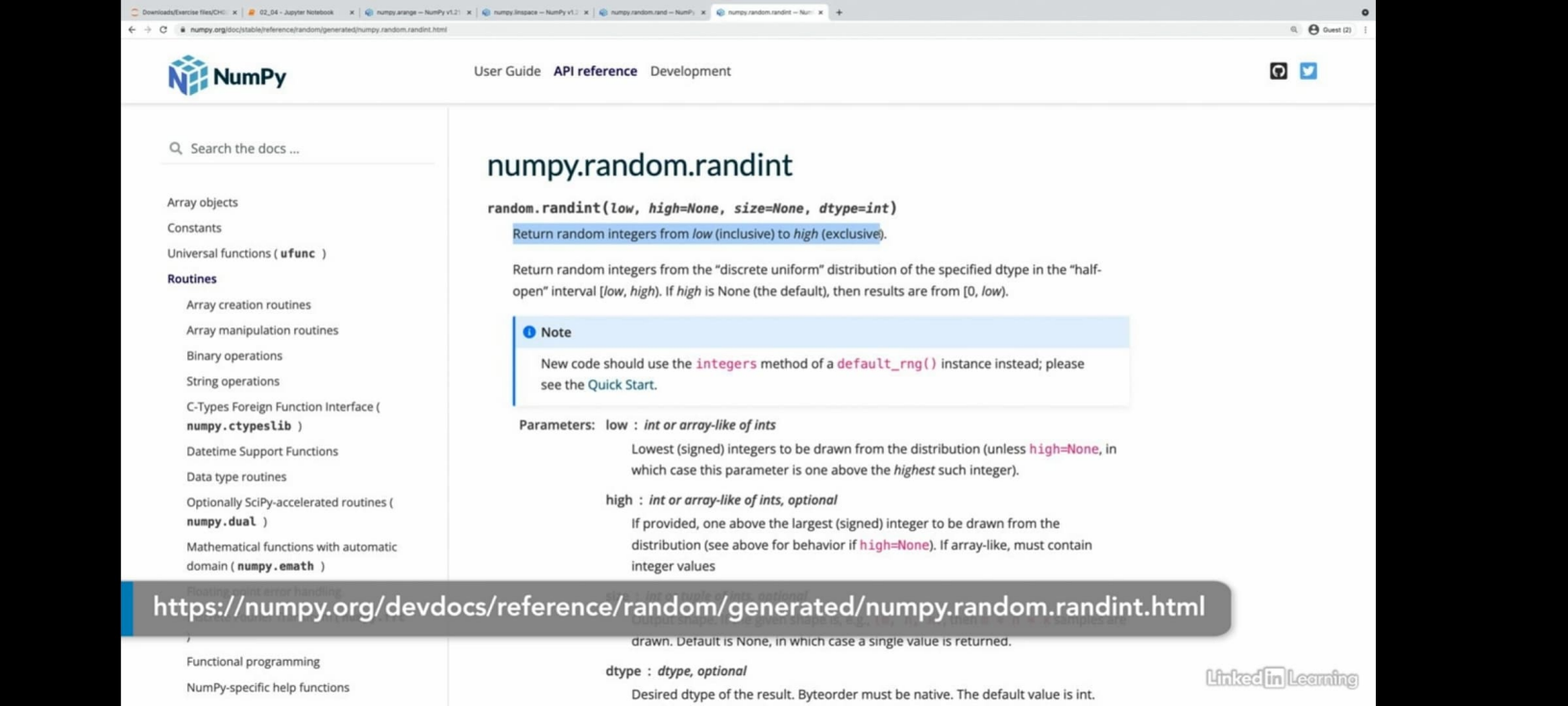
Problem Statement

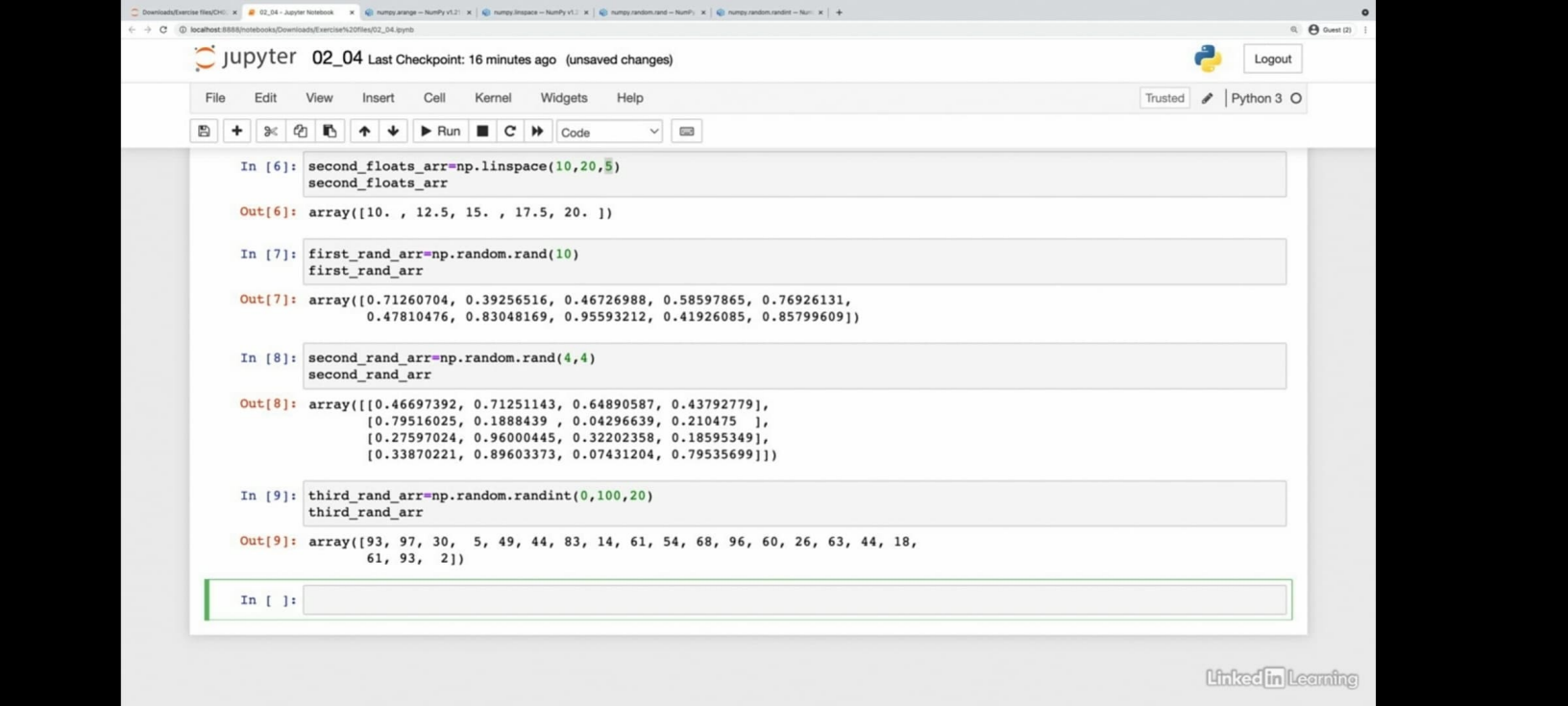
Problem 1

- DIFFERENTLY ABLED INDIVIDUALS WITH VISUAL IMPAIRMENT, HEARING IMPAIRMENT , SPEECH IMPAIRMENT AND PHYSICALLY CHALLENGED, FACE SIGNIFICANT BARRIERS IN ACCESSING ESSENTIAL INFORMATION, RESOURCES, AND OPPORTUNITIES TAILORED TO THEIR SPECIFIC NEEDS.

Problem 2

- THERE IS A LACK OF A COMPREHENSIVE AND USER-FRIENDLY PLATFORM THAT CONSOLIDATES RELEVANT INFORMATION, SERVICES, AND SUPPORT FOR THE DIFFERENTLY ABLED COMMUNITY.


numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
Constants
Universal functions (ufunc)
Routines
Array creation routines
Array manipulation routines
Binary operations
String operations
C-Types Foreign Function Interface (numpy.ctypeslib)
Datetime Support Functions
Data type routines
Optionally SciPy-accelerated routines (numpy.dual)
Mathematical functions with automatic domain (numpy.emath)
Functional programming
NumPy-specific help functions
numpy.random.randint
random.randint(low, high=None, size=None, dtype=int)
Return random integers from low (inclusive) to high (exclusive).
Return random integers from the "discrete uniform" distribution of the specified dtype in the "half-open" interval [low, high). If high is None (the default), then results are from [0, low).
Note
New code should use the integers method of a default_rng() instance instead; please see the Quick Start.
Parameters:
low : int or array-like of ints
Lowest (signed) integers to be drawn from the distribution (unless high=None, in which case this parameter is one above the highest such integer).
high : int or array-like of ints, optional
If provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if high=None). If array-like, must contain integer values
drawn. Default is None, in which case a single value is returned.
dtype : dtype, optional
Desired dtype of the result. Byteorder must be native. The default value is int.
https://numpy.org/devdocs/reference/random/generated/numpy.random.randint.html
Linked in Learning


02_04 Last Checkpoint: 16 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [6]:
second_floats_arr=np.linspace(10,20,5)
second_floats_arr
Out[6]:
array([10. , 12.5, 15. , 17.5, 20.])
In [7]:
first_rand_arr=np.random.rand(10)
first_rand_arr
Out[7]:
array([0.71260704, 0.39256516, 0.46726988, 0.58597865, 0.76926131,
0.47810476, 0.83048169, 0.95593212, 0.41926085, 0.85799609])
In [8]:
second_rand_arr=np.random.rand(4,4)
second_rand_arr
Out[8]:
array([[0.46697392, 0.71251143, 0.64890587, 0.43792779],
[0.79516025, 0.1888439 , 0.04296639, 0.210475],
[0.27597024, 0.96000445, 0.32202358, 0.18595349],
[0.33870221, 0.89603373, 0.07431204, 0.79535699]])
In [9]:
third_rand_arr=np.random.randint(0,100,20)
third_rand_arr
Out[9]:
array([93, 97, 30, 5, 49, 44, 83, 14, 61, 54, 68, 96, 60, 26, 63, 44, 18,
61, 93, 2])
In [ ]:

Solution 1

SafeZone is a mobile application that aims to create a safer and more inclusive environment for individuals with special abilities. Our vision is to enhance their quality of life by leveraging cutting-edge technologies such as Machine Learning (ML), Artificial Intelligence (AI), and Blockchain.

The Utopia

How SafeZone will proposes to solve them.

Solution 2

The app will serve as a digital hub, connecting disabled individuals, caregivers, volunteers, and organizations. It will provide a range of features to address their unique needs and challenges

The "SAFEZONE" app is a comprehensive and user-friendly platform designed to empower individuals with special abilities by providing access to essential information, resources, and opportunities. The app will feature the following key components:

- Opportunities section: A curated list of job openings, internships, and educational opportunities specifically tailored for individuals with special abilities.
- Schemes section: A comprehensive database of government scholarships, financial assistance programs, and support initiatives for the specially abled community.
- Nearby Facilities section: Information on disability-friendly restaurants, recreational facilities, and hangout spots, ensuring accessibility and inclusivity.
- Read Aloud features: A text-to-speech functionality that converts written content into audio, benefiting visually impaired users.
- Sign Language Converter: A tool that translates spoken or written language into sign language videos or animations, facilitating communication for hearing-impaired individuals.
- Obstacle Detection: A feature that utilizes augmented reality or computer vision to detect and alert users to potential obstacles in their surroundings, enhancing mobility and safety for the visually impaired.

Revenue Model

1. Primary Subscription Revenue: This will be our main source of income. We can offer different tiers of subscriptions to users, such as:

- **Basic:** Access to core features.
- **Premium:** Access to additional features, such as advanced analytics or additional storage.
- **Enterprise**: Custom solutions for larger organizations. Pricing can be set on a monthly or annual basis, with incentives for longer commitments.

2. Secondary Subscription Revenue: This involves partnerships with government bodies and NGOs. They can subscribe to use our platform to:

- Store and track data for disabled individuals they support.
- Access reports and analytics to understand the impact of their programs.
- Collaborate on creating better solutions for the disabled community.

For this, we can also have a separate pricing model based on the volume of data, number of users, and level of access required.

3. Additional Revenue Streams:

- **Data Insights**: Provide paid subscription based data access to research institutions or policy makers interested in understanding disability trends.
- **Consulting Services**: Offer our expertise to help organizations implement and use our platform effectively.
- **Training and Support:** Provide training sessions and dedicated support for a fee.

4. Cost Structure:

- **Fixed Costs:** Server costs, salaries, office space, etc.
- **Variable Costs:** Payment processing fees, customer support per ticket, etc.

5. Financial Projections:

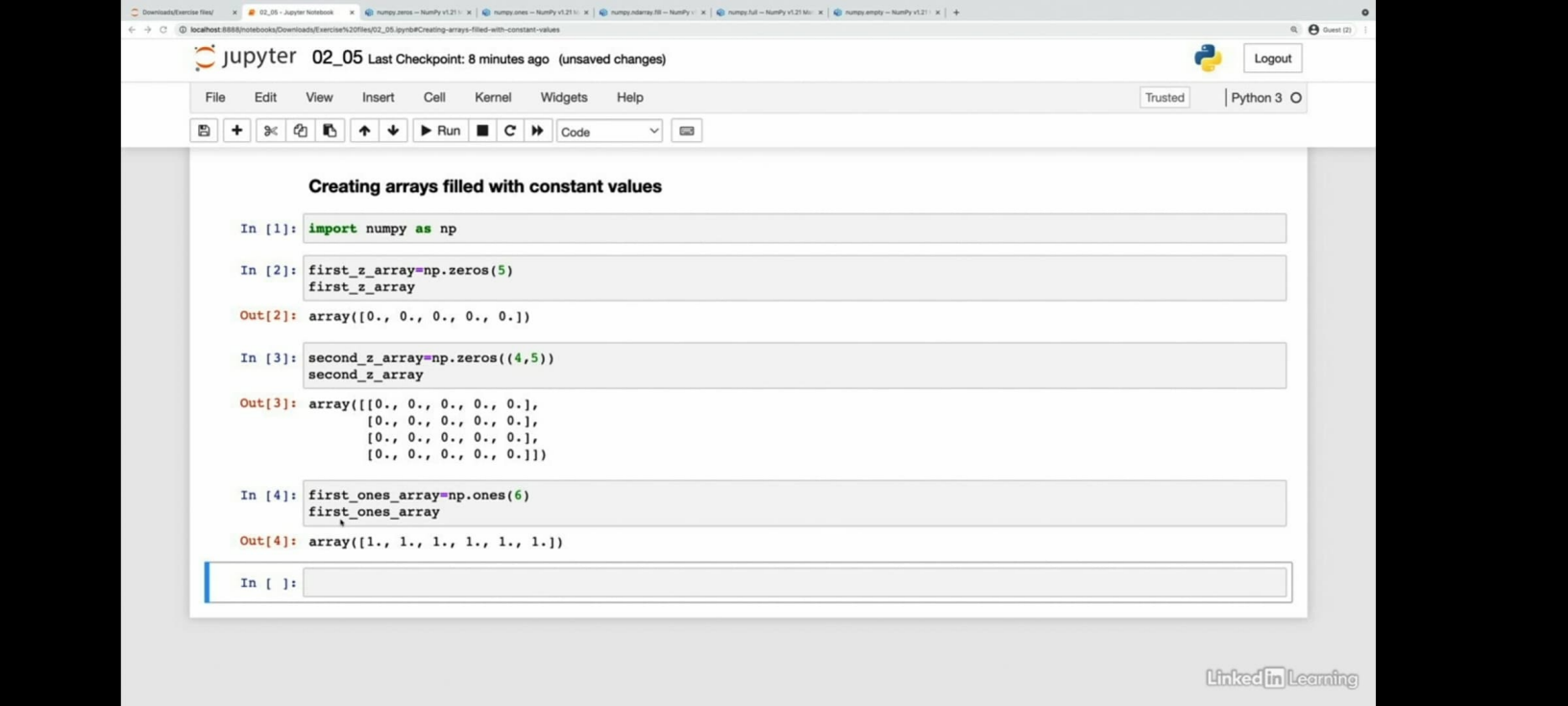
- Estimate the number of subscribers for each tier and calculate the monthly recurring revenue (MRR).
- Project growth rates based on market research and your marketing strategy.
- Calculate the break-even point and when you expect to become profitable.

6. Government and NGO Engagement:

- Research government schemes and NGO grants that support disability solutions[12].
- Tailor our secondary subscription offerings to align with the objectives of these schemes.
- Ensure compliance with data protection regulations when dealing with sensitive information.

7. Key Performance Indicators (KPIs):

- **Churn Rate:** The percentage of subscribers who cancel their subscription.
- **Customer Acquisition Cost (CAC):** The cost of acquiring a new customer.
- **Lifetime Value (LTV):** The total revenue we can expect from a customer over the duration of their subscription.


jupyter 02_05 Last Checkpoint: 8 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Creating arrays filled with constant values
In [1]: import numpy as np
In [2]: first_z_array=np.zeros(5)
first_z_array
Out[2]: array([0., 0., 0., 0., 0.])
In [3]: second_z_array=np.zeros((4,5))
second_z_array
Out[3]: array([[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.]])
In [4]: first_ones_array=np.ones(6)
first_ones_array
Out[4]: array([1., 1., 1., 1., 1., 1.])
In [ ]:

jupyter 02_05 Last Checkpoint: 8 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Out[2]:
```
array([0., 0., 0., 0., 0.])
```

In [3]:
```
second_z_array=np.zeros((4,5))
second_z_array
```

Out[3]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [4]:
```
first_ones_array=np.ones(6)
first_ones_array
```

Out[4]:
```
array([1., 1., 1., 1., 1., 1.])
```

In [5]:
```
second_ones_array=np.ones((7,8))
second_ones_array
```

Out[5]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.]])
```

In [ ]:

```
first_ones_array
```

Out[4]:
```
array([1., 1., 1., 1., 1., 1.])
```

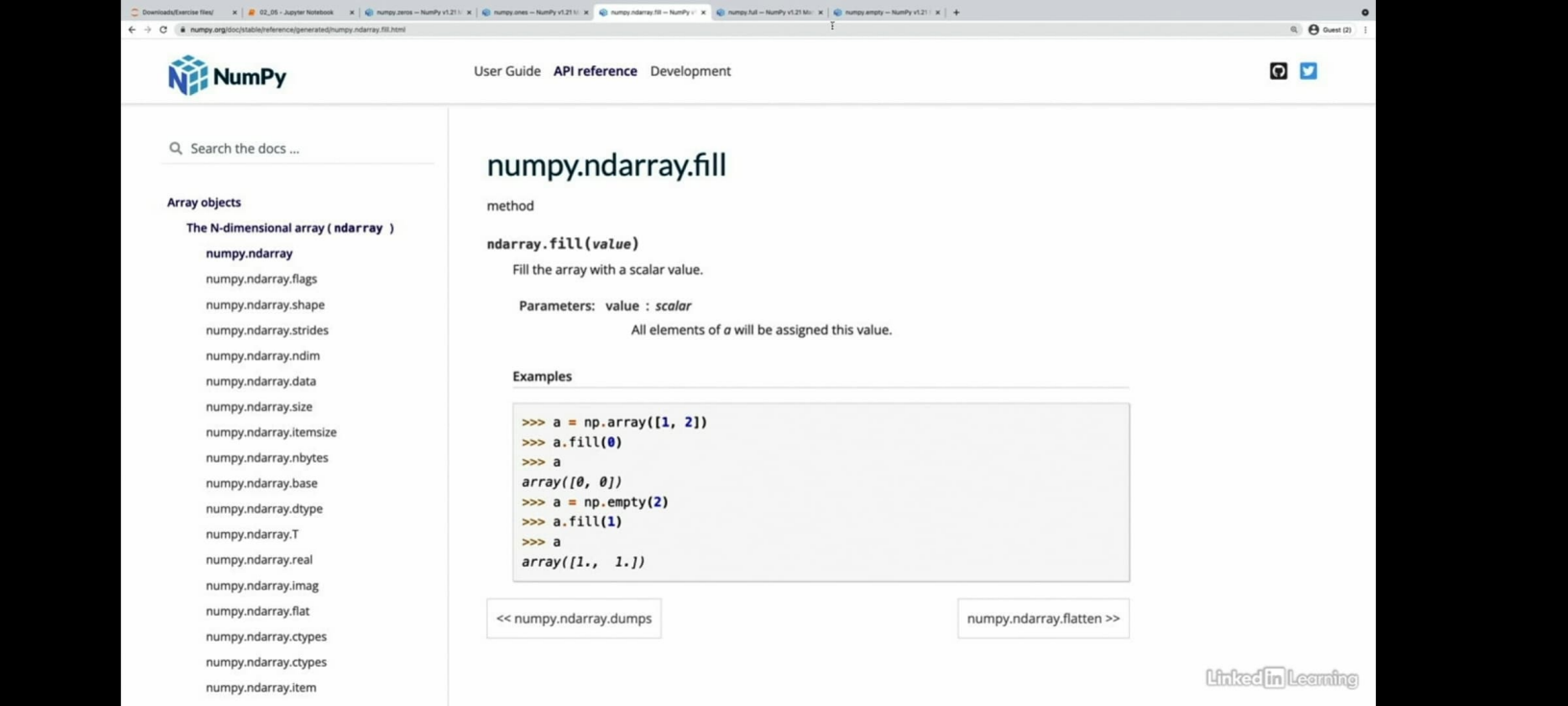
In [5]:
```
second_ones_array=np.ones((7,8))
second_ones_array
```

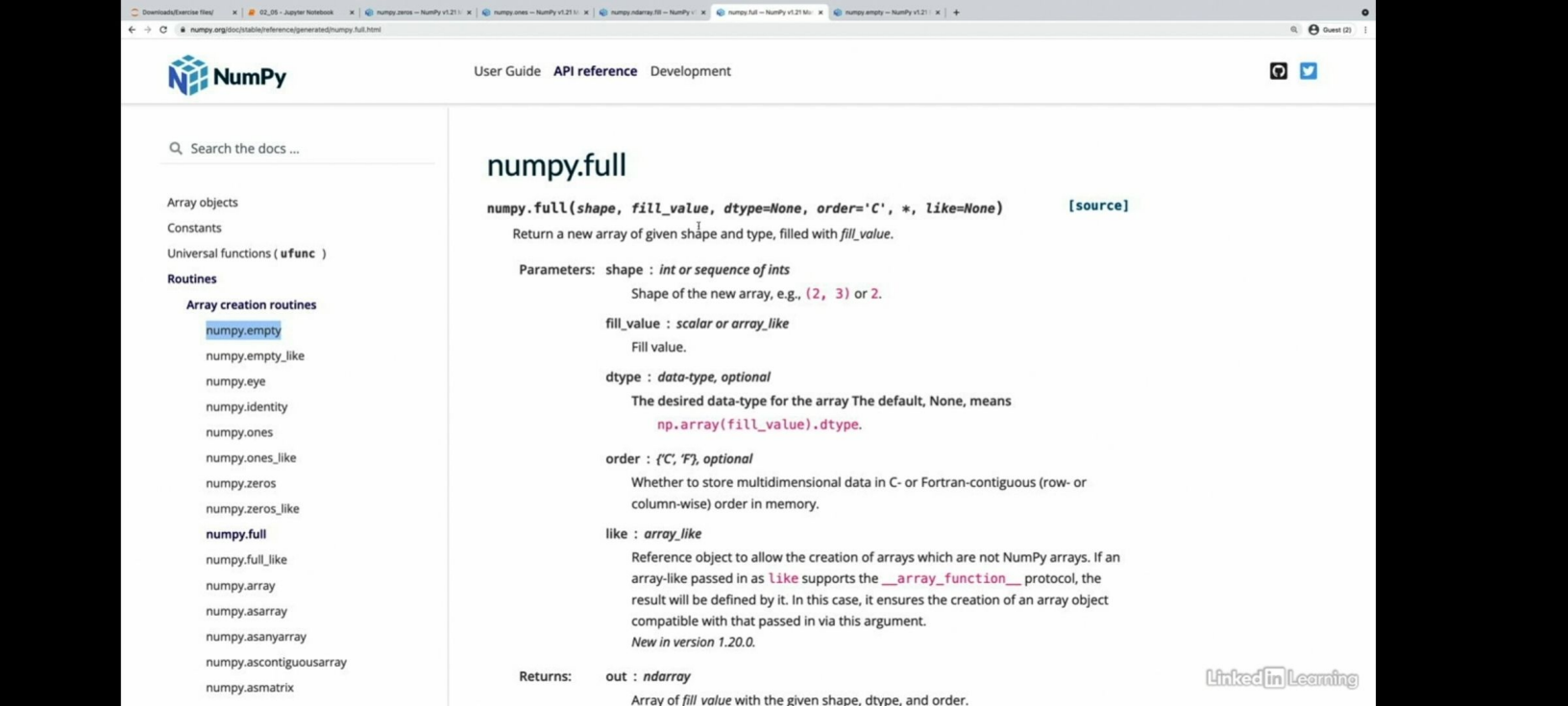
Out[5]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.]])
```

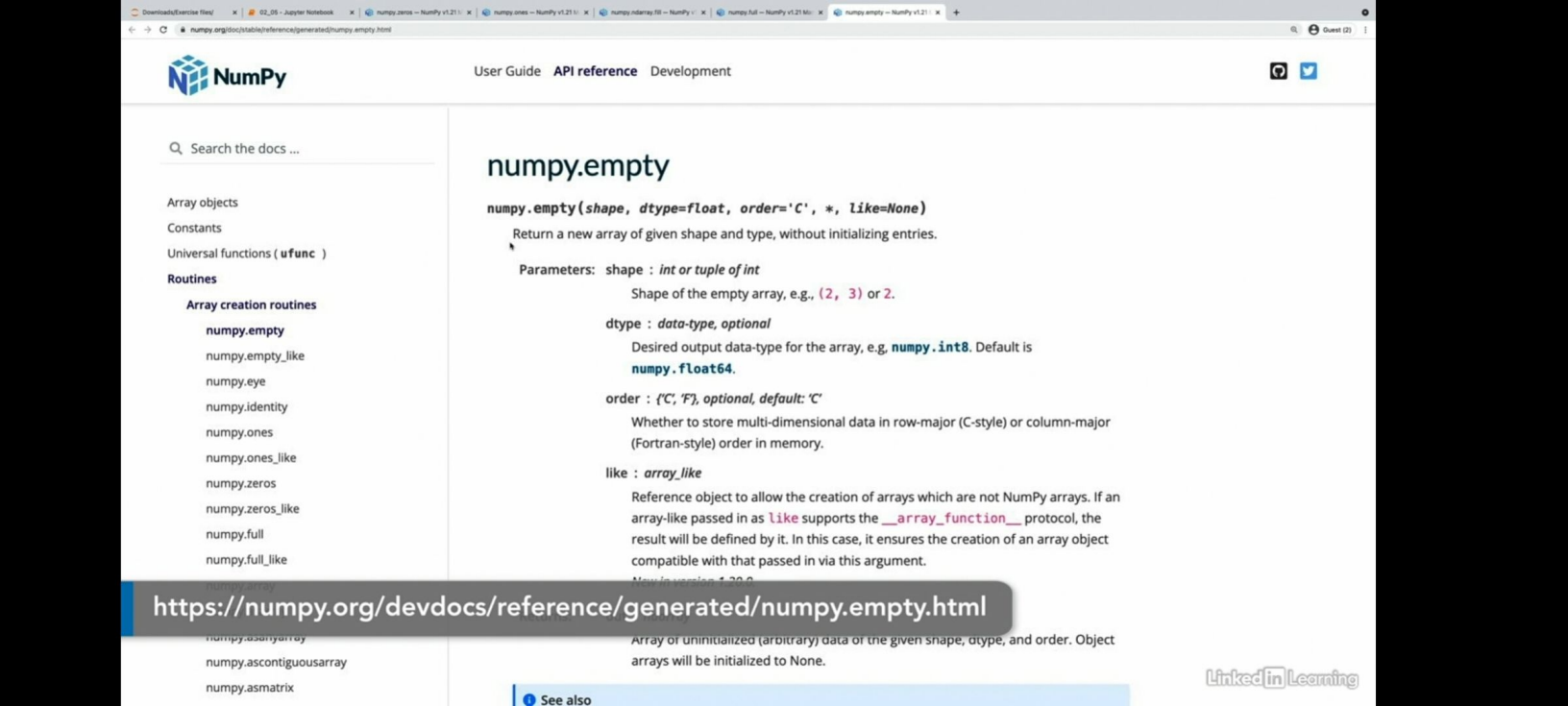
In [6]:
```
third_ones_array=np.ones((4,5),dtype=int)
third_ones_array
```

Out[6]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

In [ ]:


numpy.org/doc/stable/reference/generated/numpy.ndarray.fill.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
The N-dimensional array (ndarray)
numpy.ndarray
numpy.ndarray.flags
numpy.ndarray.shape
numpy.ndarray.strides
numpy.ndarray.ndim
numpy.ndarray.data
numpy.ndarray.size
numpy.ndarray.itemsize
numpy.ndarray.nbytes
numpy.ndarray.base
numpy.ndarray.dtype
numpy.ndarray.T
numpy.ndarray.real
numpy.ndarray.imag
numpy.ndarray.flat
numpy.ndarray.ctypes
numpy.ndarray.ctypes
numpy.ndarray.item
numpy.ndarray.fill
method
ndarray.fill(value)
Fill the array with a scalar value.
Parameters: value : scalar
All elements of a will be assigned this value.
Examples
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([1., 1.])
<< numpy.ndarray.dumps
numpy.ndarray.flatten >>
LinkedIn Learning


numpy.org/doc/stable/reference/generated/numpy.full.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
Constants
Universal functions (ufunc)
Routines
Array creation routines
numpy.empty
numpy.empty_like
numpy.eye
numpy.identity
numpy.ones
numpy.ones_like
numpy.zeros
numpy.zeros_like
numpy.full
numpy.full_like
numpy.array
numpy.asarray
numpy.asanyarray
numpy.ascontiguousarray
numpy.asmatrix
numpy.full
numpy.full(shape, fill_value, dtype=None, order='C', *, like=None)
[source]
Return a new array of given shape and type, filled with fill_value.
Parameters:
shape : int or sequence of ints
Shape of the new array, e.g., (2, 3) or 2.
fill_value : scalar or array_like
Fill value.
dtype : data-type, optional
The desired data-type for the array The default, None, means
np.array(fill_value).dtype.
order : {'C', 'F'}, optional
Whether to store multidimensional data in C- or Fortran-contiguous (row- or column-wise) order in memory.
like : array_like
Reference object to allow the creation of arrays which are not NumPy arrays. If an array-like passed in as like supports the __array_function__ protocol, the result will be defined by it. In this case, it ensures the creation of an array object compatible with that passed in via this argument.
New in version 1.20.0.
Returns:
out : ndarray
Array of fill value with the given shape, dtype, and order.
LinkedIn Learning


numpy.org/doc/stable/reference/generated/numpy.empty.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
Constants
Universal functions (ufunc)
Routines
Array creation routines
numpy.empty
numpy.empty_like
numpy.eye
numpy.identity
numpy.ones
numpy.ones_like
numpy.zeros
numpy.zeros_like
numpy.full
numpy.full_like
numpy.ascontiguousarray
numpy.asmatrix
numpy.empty
numpy.empty(shape, dtype=float, order='C', *, like=None)
Return a new array of given shape and type, without initializing entries.
Parameters:
shape : int or tuple of int
Shape of the empty array, e.g., (2, 3) or 2.
dtype : data-type, optional
Desired output data-type for the array, e.g, numpy.int8. Default is numpy.float64.
order : {'C', 'F'}, optional, default: 'C'
Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.
like : array_like
Reference object to allow the creation of arrays which are not NumPy arrays. If an array-like passed in as like supports the __array_function__ protocol, the result will be defined by it. In this case, it ensures the creation of an array object compatible with that passed in via this argument.
Array of uninitialized (arbitrary) data of the given shape, dtype, and order. Object arrays will be initialized to None.
See also
https://numpy.org/devdocs/reference/generated/numpy.empty.html
LinkedIn Learning

In [6]:
```
third_ones_array=np.ones((4,5),dtype=int)
third_ones_array
```

Out[6]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

In [7]:
```
first_fill_array=np.empty(10,dtype=int)
first_fill_array.fill(12)
first_fill_array
```

Out[7]:
```
array([12, 12, 12, 12, 12, 12, 12, 12, 12, 12])
```

In [8]:
```
first_full_array=np.full(5,10)
first_full_array
```

Out[8]:
```
array([10, 10, 10, 10, 10])
```

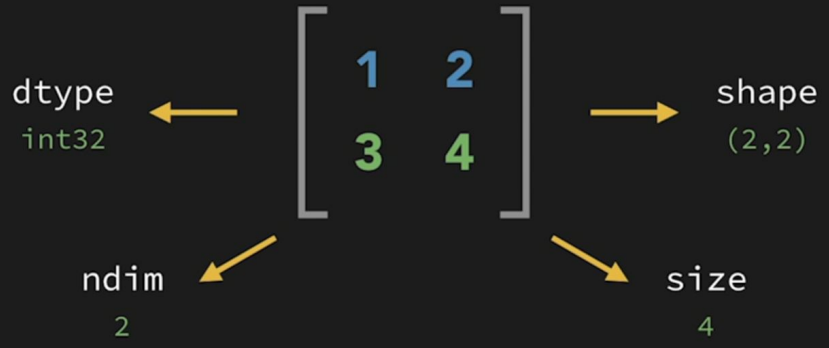
In [9]:
```
second_full_array=np.full((4,5),8)
second_full_array
```

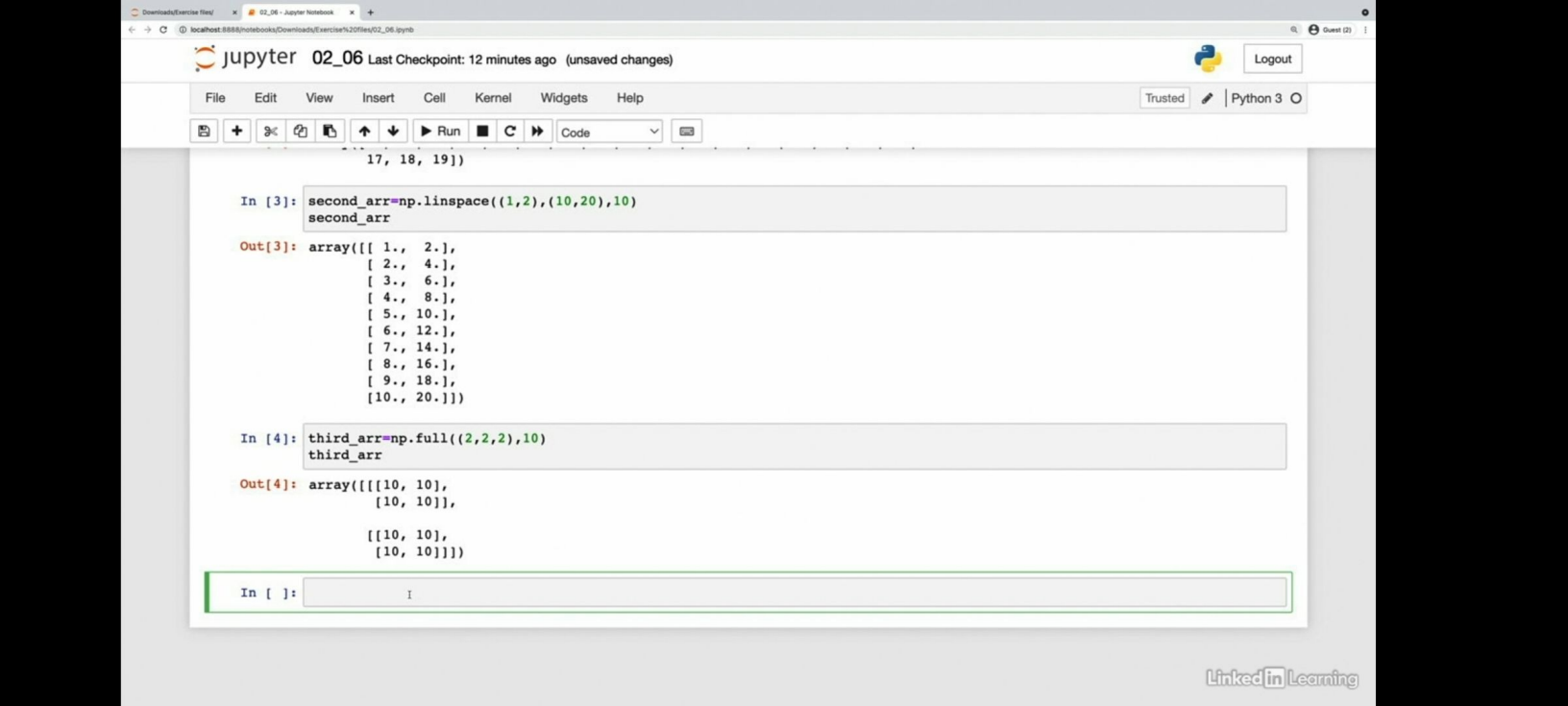
Out[9]:
```
array([[8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

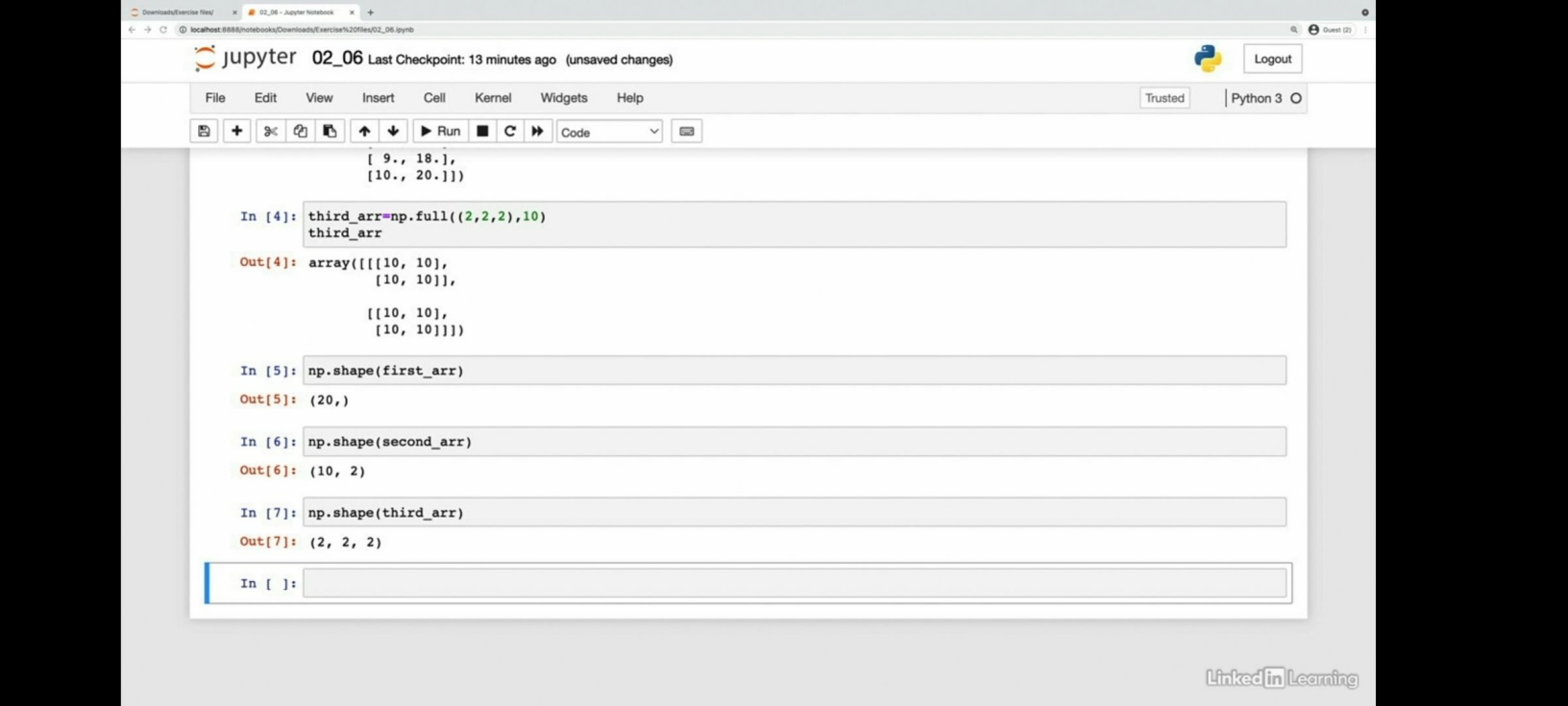
In [ ]:

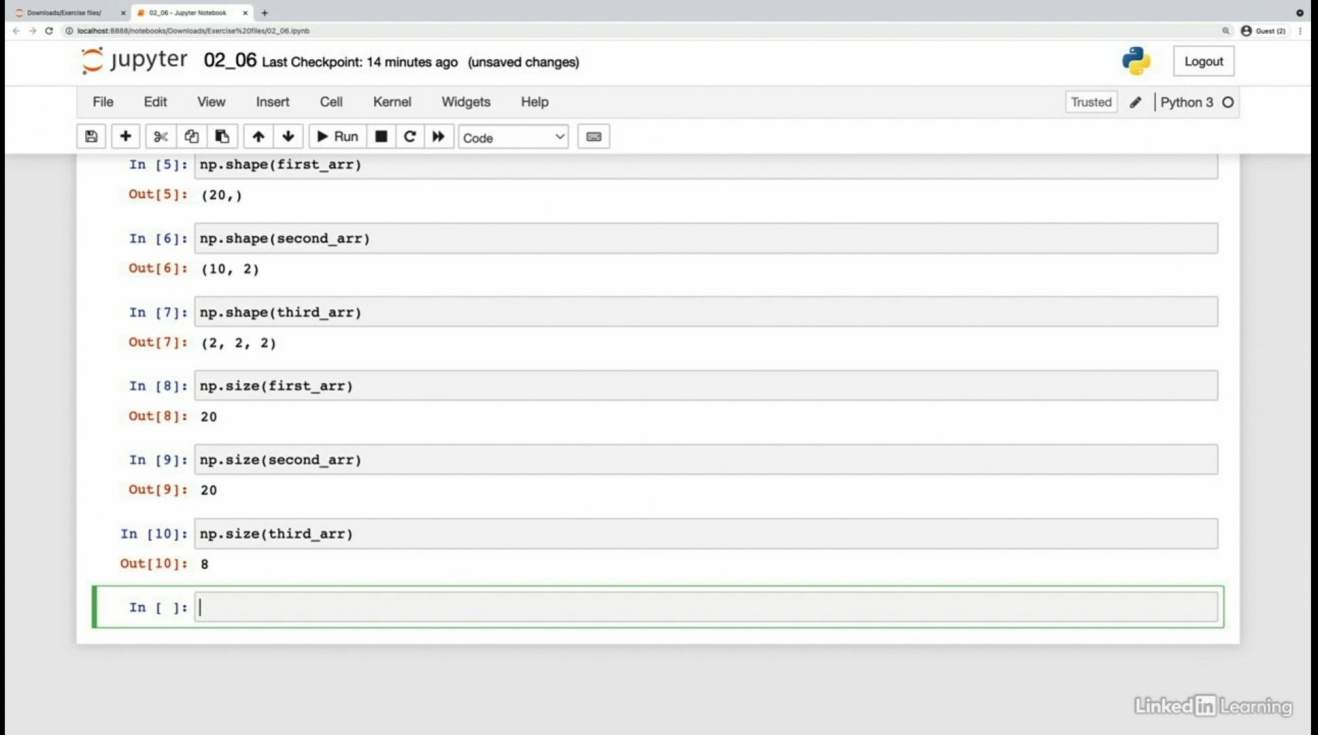
Finding the Shape and the Size of an Array

```
Integers = np.array([[1,2],[3,4]])
```




Downloads/Exercise files/
02_06 - Jupyter Notebook
localhost:8888/notebooks/Downloads/Exercise%20files/02_06.ipynb
Guest (2)
jupyter
02_06 Last Checkpoint: 12 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
17, 18, 19])
In [3]:
second_arr=np.linspace((1,2),(10,20),10)
second_arr
Out[3]:
array([[1., 2.],
[2., 4.],
[3., 6.],
[4., 8.],
[5., 10.],
[6., 12.],
[7., 14.],
[8., 16.],
[9., 18.],
[10., 20.]])
In [4]:
third_arr=np.full((2,2,2),10)
third_arr
Out[4]:
array([[[10, 10],
[10, 10]],
[[10, 10],
[10, 10]]])
In [ ]:
LinkedIn Learning


Downloads/Exercise files/
02_06 - Jupyter Notebook
localhost:8888/notebooks/Downloads/Exercise%20files/02_06.ipynb
Guest (2)
jupyter
02_06 Last Checkpoint: 13 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
[9., 18.],
[10., 20.]])
In [4]:
third_arr=np.full((2,2,2),10)
third_arr
Out[4]:
array([[[10, 10],
[10, 10]],
[[10, 10],
[10, 10]]])
In [5]:
np.shape(first_arr)
Out[5]:
(20,)
In [6]:
np.shape(second_arr)
Out[6]:
(10, 2)
In [7]:
np.shape(third_arr)
Out[7]:
(2, 2, 2)
In [ ]:
LinkedIn Learning


Downloads/Exercise files/
02_06 - Jupyter Notebook
localhost:8888/notebooks/Downloads/Exercise%20files/02_06.ipynb
Guest (2)
jupyter 02_06 Last Checkpoint: 14 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [5]: np.shape(first_arr)
Out[5]: (20,)
In [6]: np.shape(second_arr)
Out[6]: (10, 2)
In [7]: np.shape(third_arr)
Out[7]: (2, 2, 2)
In [8]: np.size(first_arr)
Out[8]: 20
In [9]: np.size(second_arr)
Out[9]: 20
In [10]: np.size(third_arr)
Out[10]: 8
In [ ]:

jupyter 03_01 Last Checkpoint: an hour ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run | Code

Adding, removing and sorting elements

In [1]:
```
import numpy as np
```

In [2]:
```
first_arr=np.array([1, 2, 3, 5])
first_arr
```

Out[2]:
```
array([1, 2, 3, 5])
```

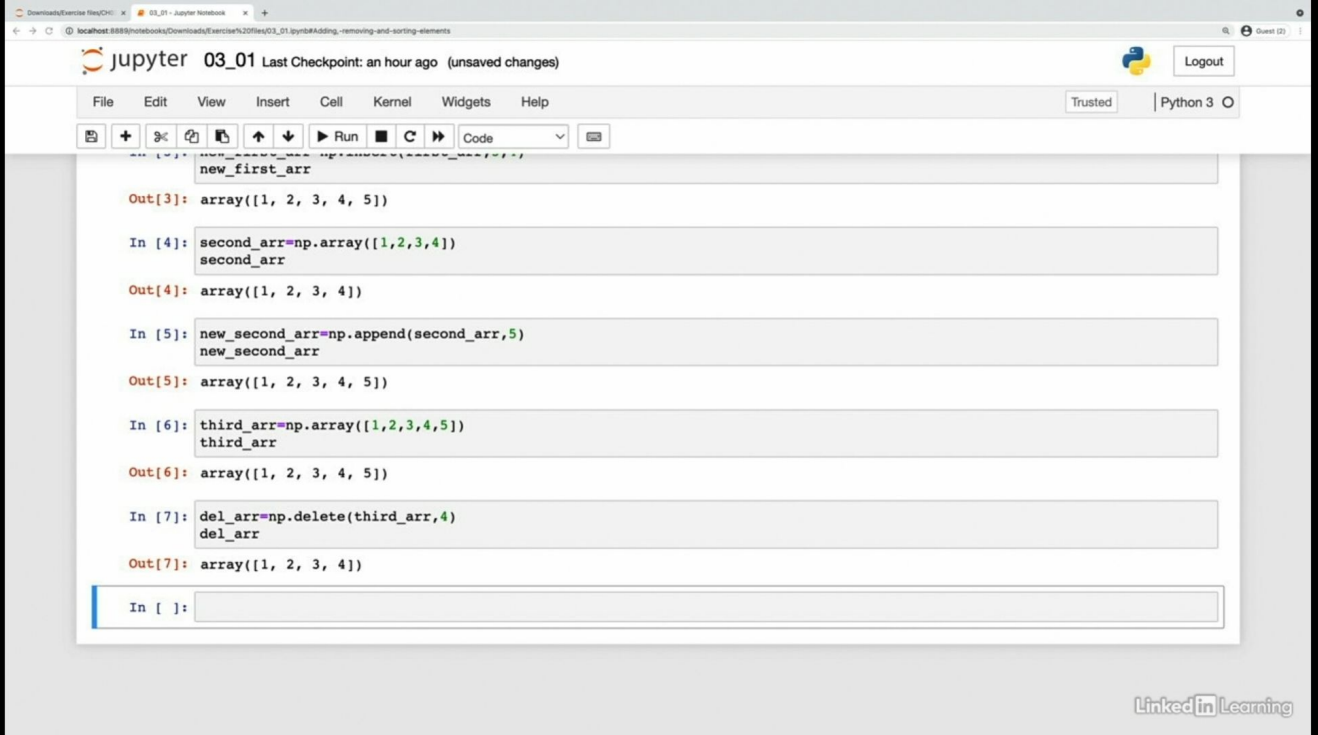
In [3]:
```
new_first_arr=np.insert(first_arr,3,4)
new_first_arr
```

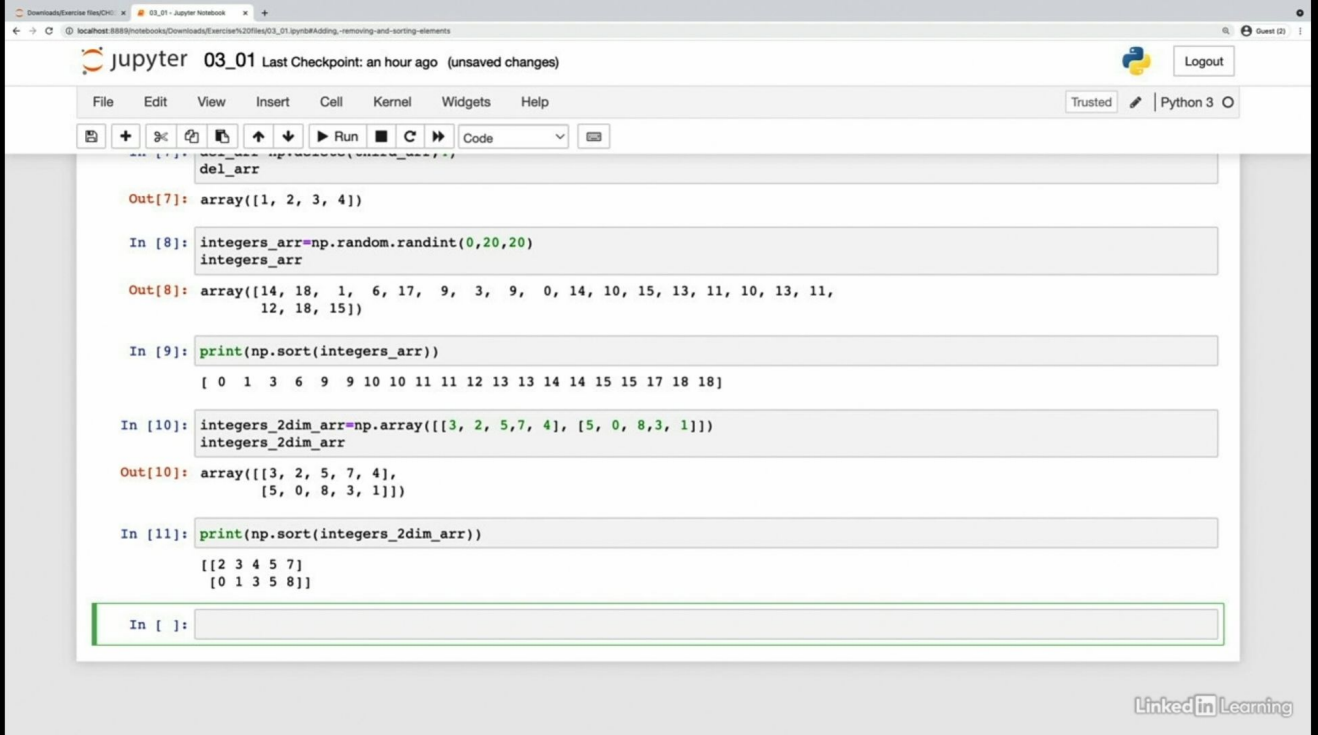
Out[3]:
```
array([1, 2, 3, 4, 5])
```

In [4]:
```
second_arr=np.array([1,2,3,4])
second_arr
```

Out[4]:
```
array([1, 2, 3, 4])
```

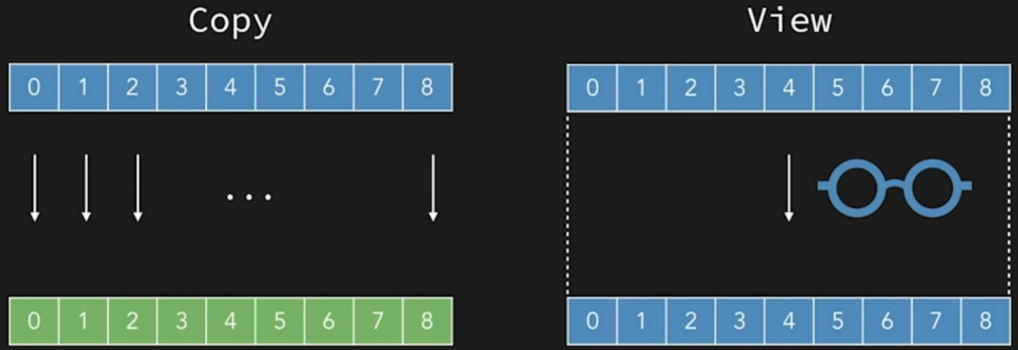
In [ ]:


jupyter 03_01 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
new_first_arr
Out[3]: array([1, 2, 3, 4, 5])
In [4]: second_arr=np.array([1,2,3,4])
second_arr
Out[4]: array([1, 2, 3, 4])
In [5]: new_second_arr=np.append(second_arr,5)
new_second_arr
Out[5]: array([1, 2, 3, 4, 5])
In [6]: third_arr=np.array([1,2,3,4,5])
third_arr
Out[6]: array([1, 2, 3, 4, 5])
In [7]: del_arr=np.delete(third_arr,4)
del_arr
Out[7]: array([1, 2, 3, 4])
In [ ]:


03_01 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/03_01.ipynb#Adding,-removing-and-sorting-elements
jupyter 03_01 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
del_arr
Out[7]: array([1, 2, 3, 4])
In [8]: integers_arr=np.random.randint(0,20,20)
integers_arr
Out[8]: array([14, 18, 1, 6, 17, 9, 3, 9, 0, 14, 10, 15, 13, 11, 10, 13, 11,
12, 18, 15])
In [9]: print(np.sort(integers_arr))
[0 1 3 6 9 9 10 10 11 11 12 13 13 14 14 15 15 17 18 18]
In [10]: integers_2dim_arr=np.array([[3, 2, 5,7, 4], [5, 0, 8,3, 1]])
integers_2dim_arr
Out[10]: array([[3, 2, 5, 7, 4],
[5, 0, 8, 3, 1]])
In [11]: print(np.sort(integers_2dim_arr))
[[2 3 4 5 7]
[0 1 3 5 8]]
In [ ]:

Copies and Views

```
Numbers = np.array([0,1,2,3,4,5,6,7,8])
```



Copies and views

In [1]:
```
import numpy as np
```

In [2]:
```
students_ids_number=np.array([1111,1212,1313,1414,1515,1616,1717,1818])
students_ids_number
```

Out[2]:
```
array([1111, 1212, 1313, 1414, 1515, 1616, 1717, 1818])
```

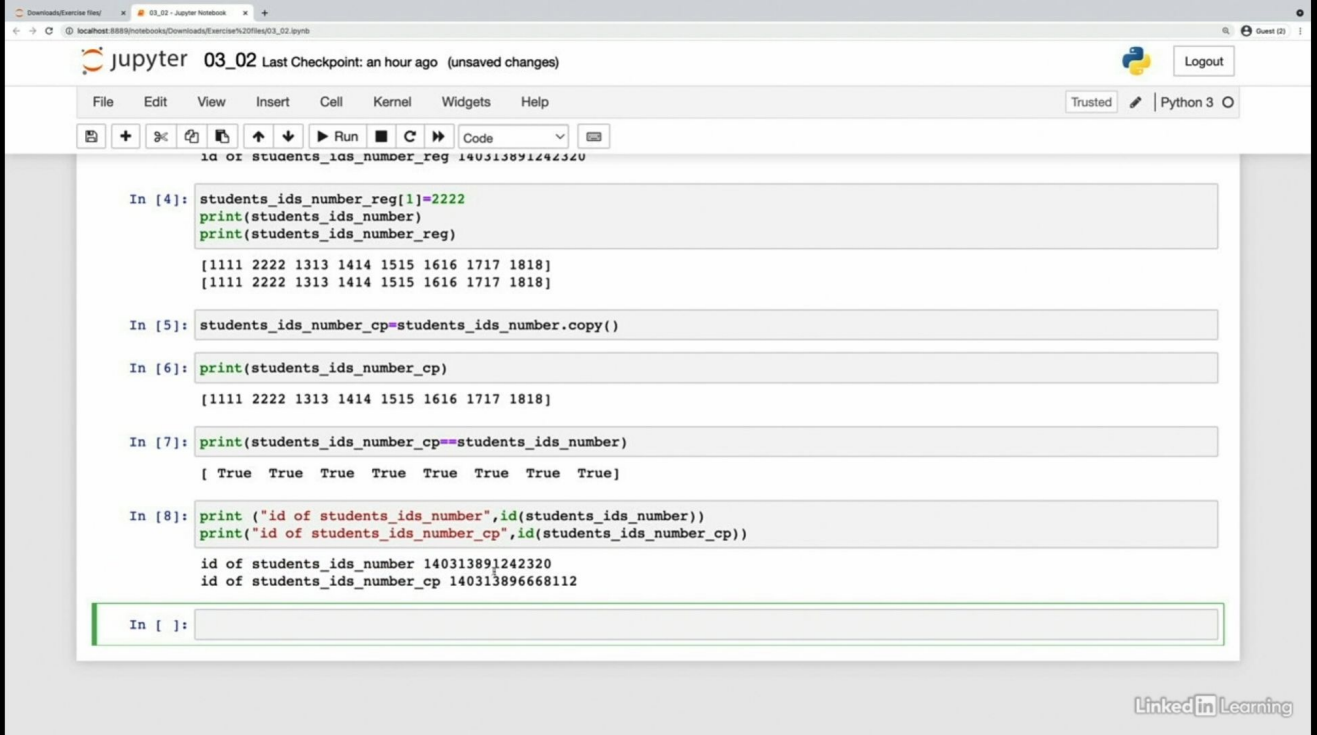
In [3]:
```
students_ids_number_reg=students_ids_number
print("id of students_ids_number",id(students_ids_number))
print("id of students_ids_number_reg",id(students_ids_number_reg))
```

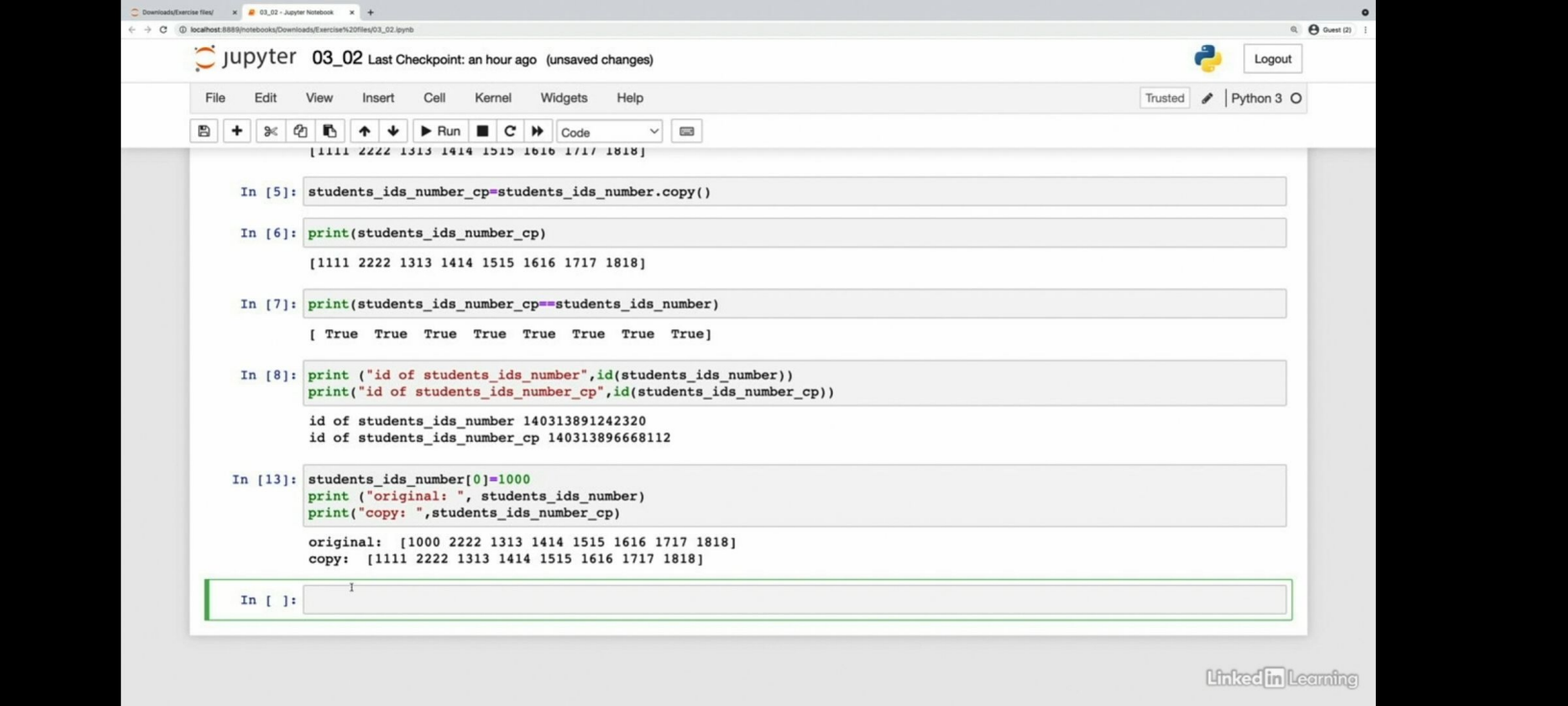
```
id of students_ids_number 140313891242320
id of students_ids_number_reg 140313891242320
```

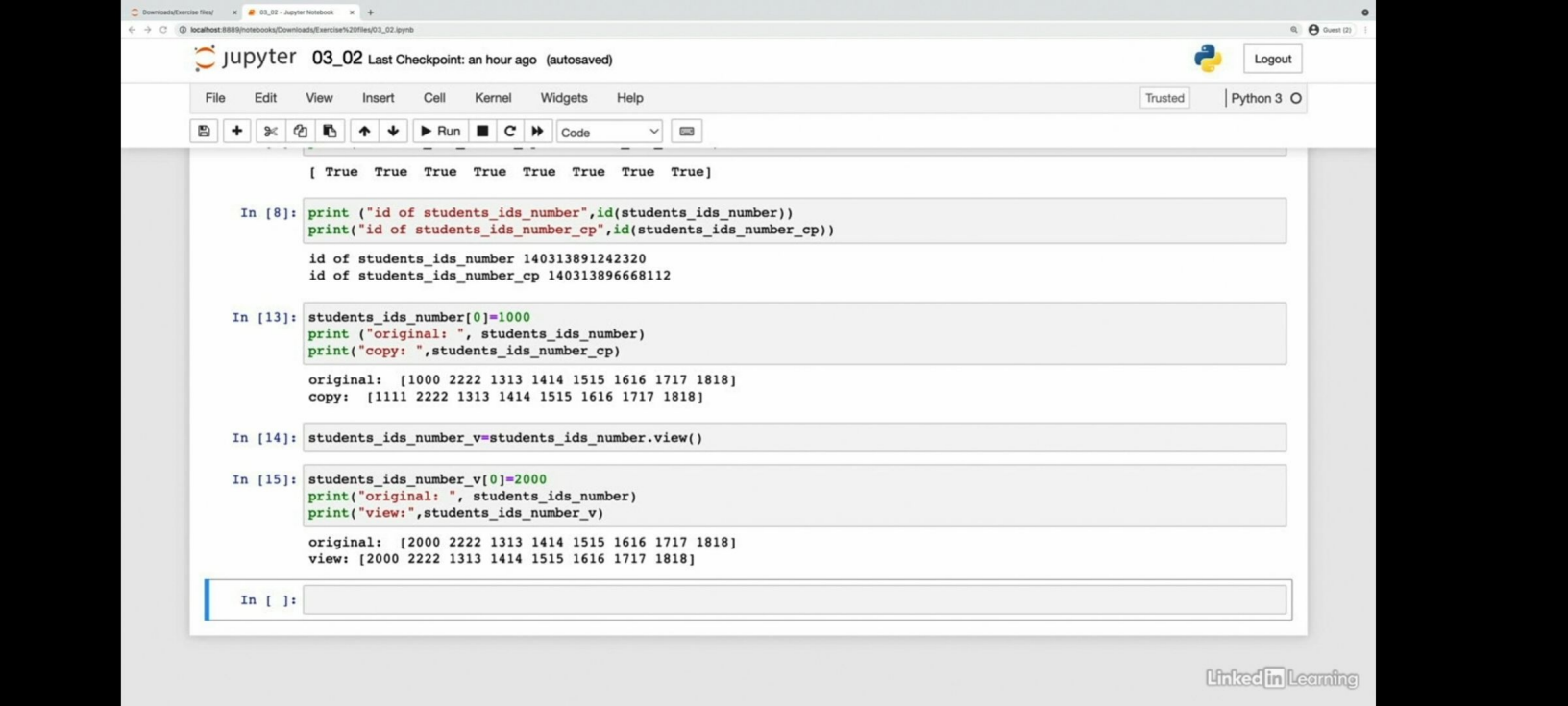
In [4]:
```
students_ids_number_reg[1]=2222
print(students_ids_number)
print(students_ids_number_reg)
```

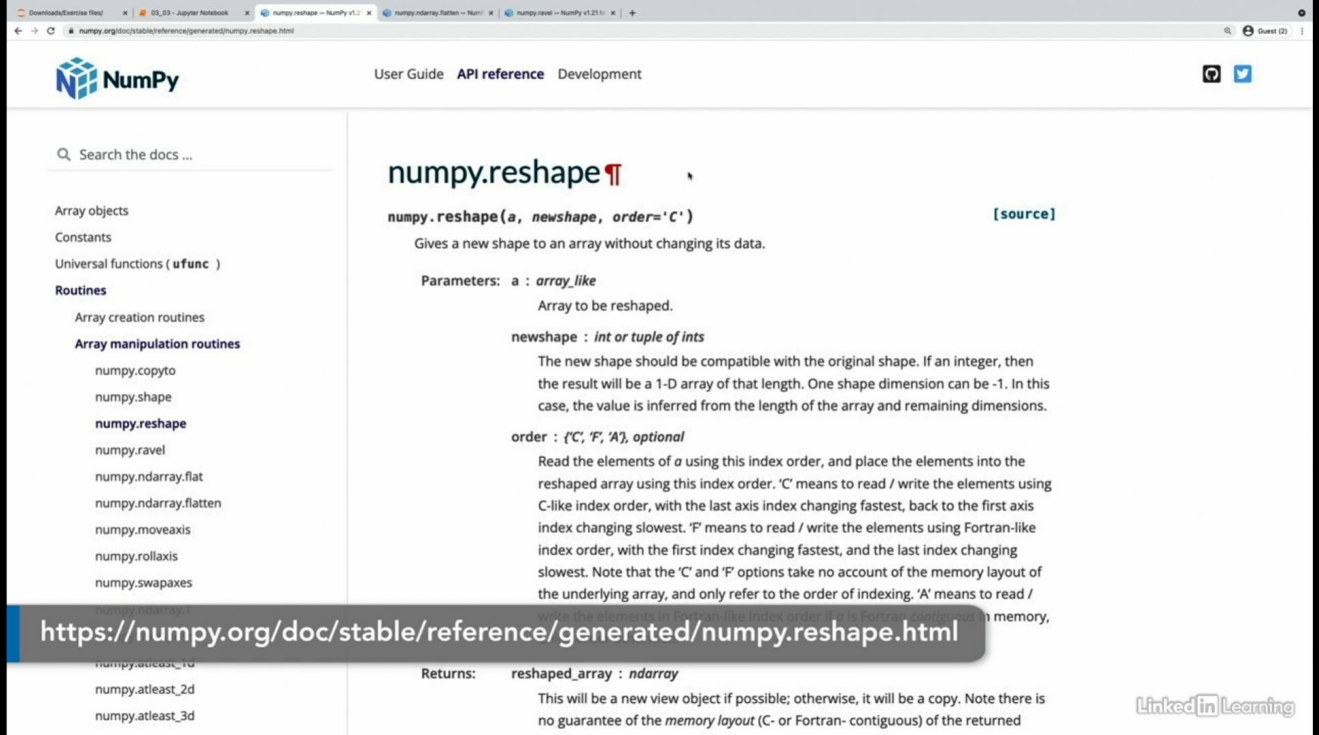
```
[1111 2222 1313 1414 1515 1616 1717 1818]
[1111 2222 1313 1414 1515 1616 1717 1818]
```

In [ ]:

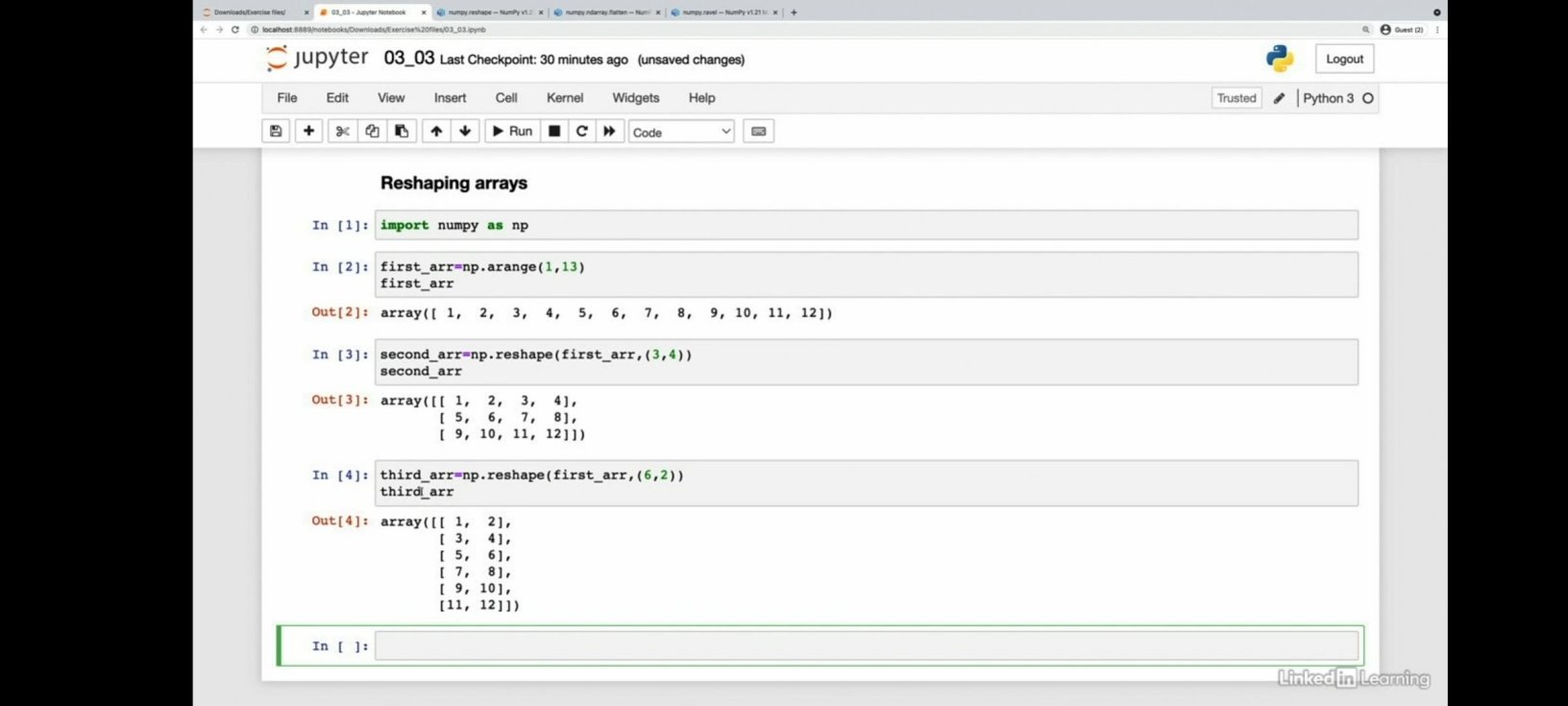

Downloads/Exercise files/
03_02 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/03_02.ipynb
Guest (2)
jupyter 03_02 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
id of students_ids_number_reg 140313891242320
In [4]:
students_ids_number_reg[1]=2222
print(students_ids_number)
print(students_ids_number_reg)
[1111 2222 1313 1414 1515 1616 1717 1818]
[1111 2222 1313 1414 1515 1616 1717 1818]
In [5]:
students_ids_number_cp=students_ids_number.copy()
In [6]:
print(students_ids_number_cp)
[1111 2222 1313 1414 1515 1616 1717 1818]
In [7]:
print(students_ids_number_cp==students_ids_number)
[True True True True True True True True]
In [8]:
print ("id of students_ids_number",id(students_ids_number))
print("id of students_ids_number_cp",id(students_ids_number_cp))
id of students_ids_number 140313891242320
id of students_ids_number_cp 140313896668112
In [ ]:
LinkedIn Learning

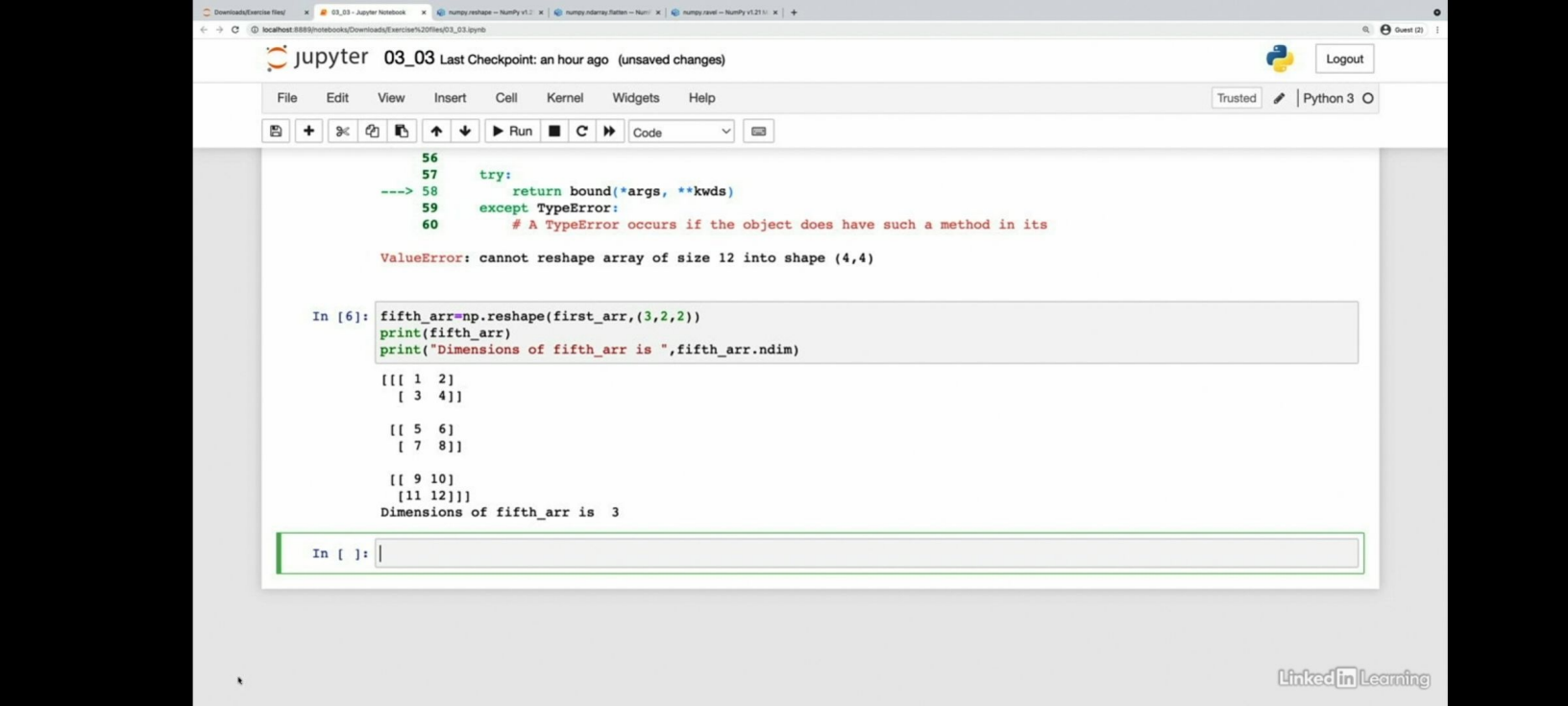

Downloads/Exercise files/
03_02 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/03_02.ipynb
Guest (2)
jupyter 03_02 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
[1111 2222 1313 1414 1515 1616 1717 1818]
In [5]:
students_ids_number_cp=students_ids_number.copy()
In [6]:
print(students_ids_number_cp)
[1111 2222 1313 1414 1515 1616 1717 1818]
In [7]:
print(students_ids_number_cp==students_ids_number)
[True True True True True True True True]
In [8]:
print ("id of students_ids_number",id(students_ids_number))
print("id of students_ids_number_cp",id(students_ids_number_cp))
id of students_ids_number 140313891242320
id of students_ids_number_cp 140313896668112
In [13]:
students_ids_number[0]=1000
print ("original: ", students_ids_number)
print("copy: ",students_ids_number_cp)
original: [1000 2222 1313 1414 1515 1616 1717 1818]
copy: [1111 2222 1313 1414 1515 1616 1717 1818]
In [ ]:
LinkedIn Learning

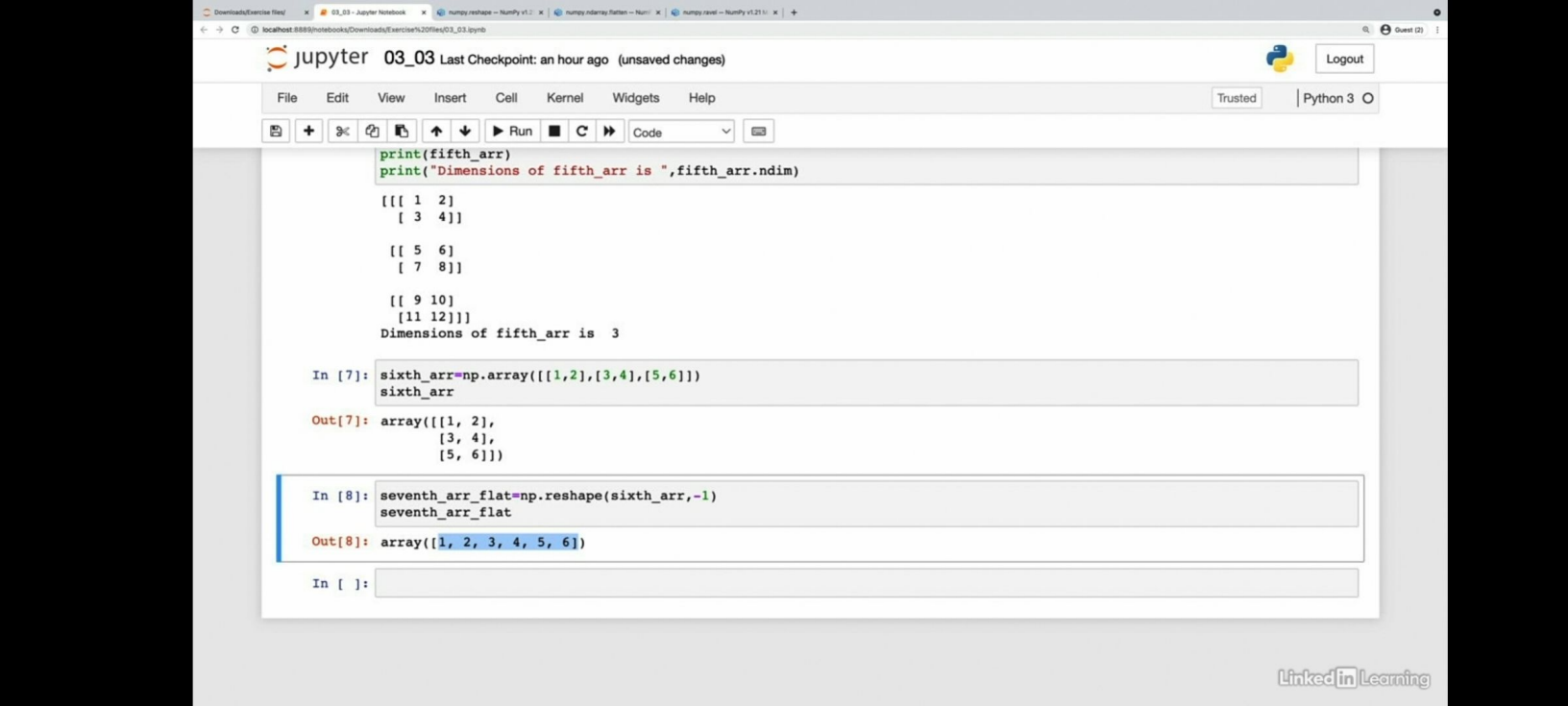

jupyter 03_02 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
[True True True True True True True True]
In [8]:
print ("id of students_ids_number",id(students_ids_number))
print("id of students_ids_number_cp",id(students_ids_number_cp))
id of students_ids_number 140313891242320
id of students_ids_number_cp 140313896668112
In [13]:
students_ids_number[0]=1000
print ("original: ", students_ids_number)
print("copy: ",students_ids_number_cp)
original: [1000 2222 1313 1414 1515 1616 1717 1818]
copy: [1111 2222 1313 1414 1515 1616 1717 1818]
In [14]:
students_ids_number_v=students_ids_number.view()
In [15]:
students_ids_number_v[0]=2000
print("original: ", students_ids_number)
print("view:",students_ids_number_v)
original: [2000 2222 1313 1414 1515 1616 1717 1818]
view: [2000 2222 1313 1414 1515 1616 1717 1818]
In [ ]:

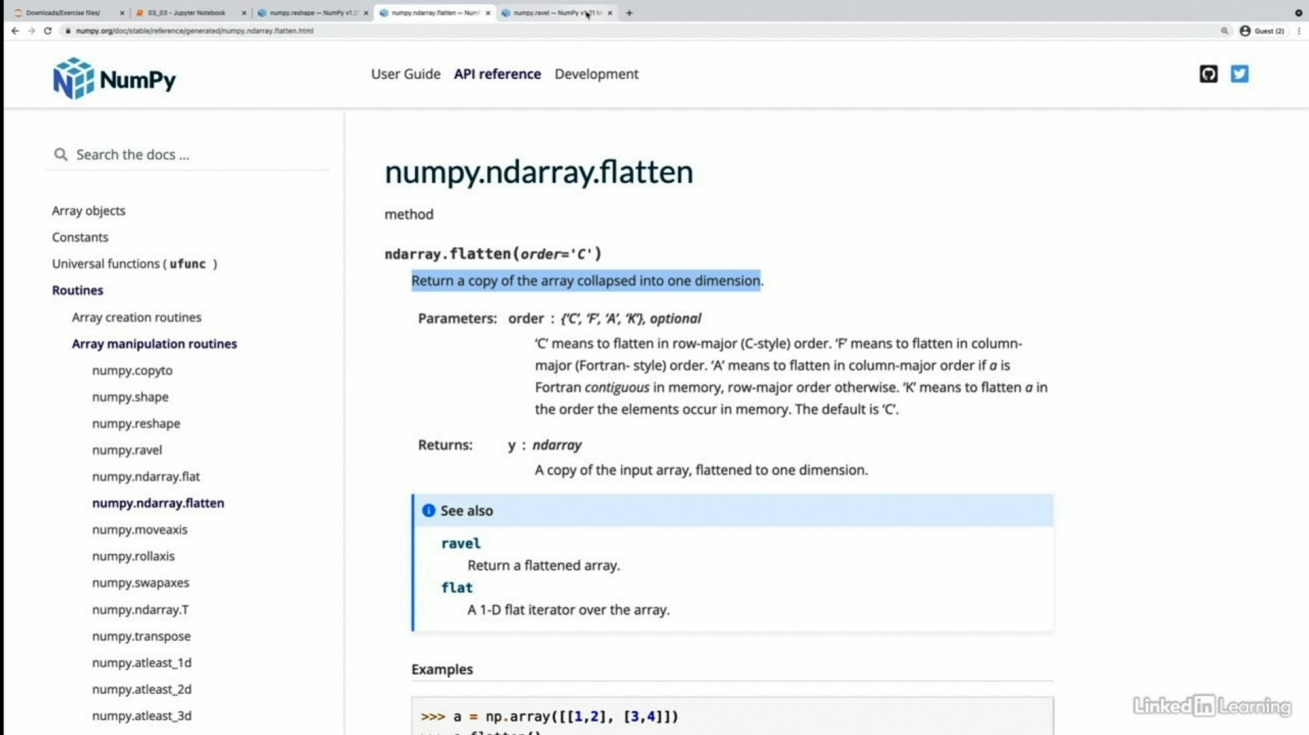


https://numpy.org/doc/stable/reference/generated/numpy.reshape.html


jupyter 03_03 Last Checkpoint: 30 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Reshaping arrays
In [1]: import numpy as np
In [2]: first_arr=np.arange(1,13)
first_arr
Out[2]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
In [3]: second_arr=np.reshape(first_arr,(3,4))
second_arr
Out[3]: array([[1, 2, 3, 4],
[5, 6, 7, 8],
[9, 10, 11, 12]])
In [4]: third_arr=np.reshape(first_arr,(6,2))
third_arr
Out[4]: array([[1, 2],
[3, 4],
[5, 6],
[7, 8],
[9, 10],
[11, 12]])
In [ ]:


jupyter 03_03 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
56
57 try:
---> 58 return bound(*args, **kwds)
59 except TypeError:
60 # A TypeError occurs if the object does have such a method in its
ValueError: cannot reshape array of size 12 into shape (4,4)
In [6]:
fifth_arr=np.reshape(first_arr,(3,2,2))
print(fifth_arr)
print("Dimensions of fifth_arr is ",fifth_arr.ndim)
[[[1 2]
[3 4]]
[[5 6]
[7 8]]
[[9 10]
[11 12]]]
Dimensions of fifth_arr is 3
In [ ]:


jupyter 03_03 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
print(fifth_arr)
print("Dimensions of fifth_arr is ",fifth_arr.ndim)
[[[1 2]
[3 4]]
[[5 6]
[7 8]]
[[9 10]
[11 12]]]
Dimensions of fifth_arr is 3
In [7]:
sixth_arr=np.array([[1,2],[3,4],[5,6]])
sixth_arr
Out[7]:
array([[1, 2],
[3, 4],
[5, 6]])
In [8]:
seventh_arr_flat=np.reshape(sixth_arr,-1)
seventh_arr_flat
Out[8]:
array([1, 2, 3, 4, 5, 6])
In [ ]:



Downloads/Exercise files/ | 03_03 - Jupyter Notebook | numpy.reshape — NumPy v1.2 | numpy.ndarray.flatten — Num | numpy.ravel — NumPy v1.21

numpy.org/doc/stable/reference/generated/numpy.ndarray.flatten.html

NumPy

User Guide **API reference** Development

Search the docs ...

Array objects
Constants
Universal functions (**ufunc**)
Routines
Array creation routines
Array manipulation routines
numpy.copyto
numpy.shape
numpy.reshape
numpy.ravel
numpy.ndarray.flat
numpy.ndarray.flatten
numpy.moveaxis
numpy.rollaxis
numpy.swapaxes
numpy.ndarray.T
numpy.transpose
numpy.atleast_1d
numpy.atleast_2d
numpy.atleast_3d

numpy.ndarray.flatten

method

ndarray.flatten(*order='C'*)

Return a copy of the array collapsed into one dimension.

Parameters: **order** : *{'C', 'F', 'A', 'K'}, optional*
'C' means to flatten in row-major (C-style) order. 'F' means to flatten in column-major (Fortran- style) order. 'A' means to flatten in column-major order if *a* is Fortran *contiguous* in memory, row-major order otherwise. 'K' means to flatten *a* in the order the elements occur in memory. The default is 'C'.

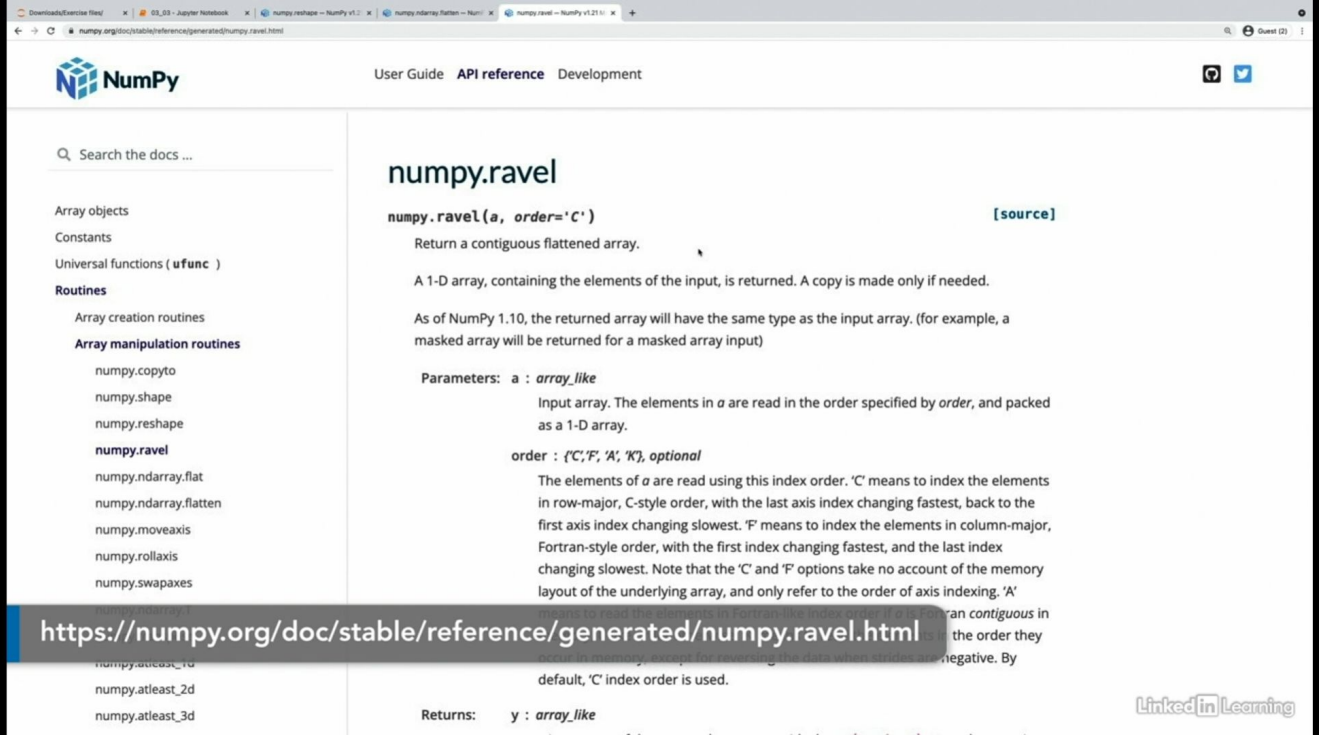
Returns: **y** : *ndarray*
A copy of the input array, flattened to one dimension.

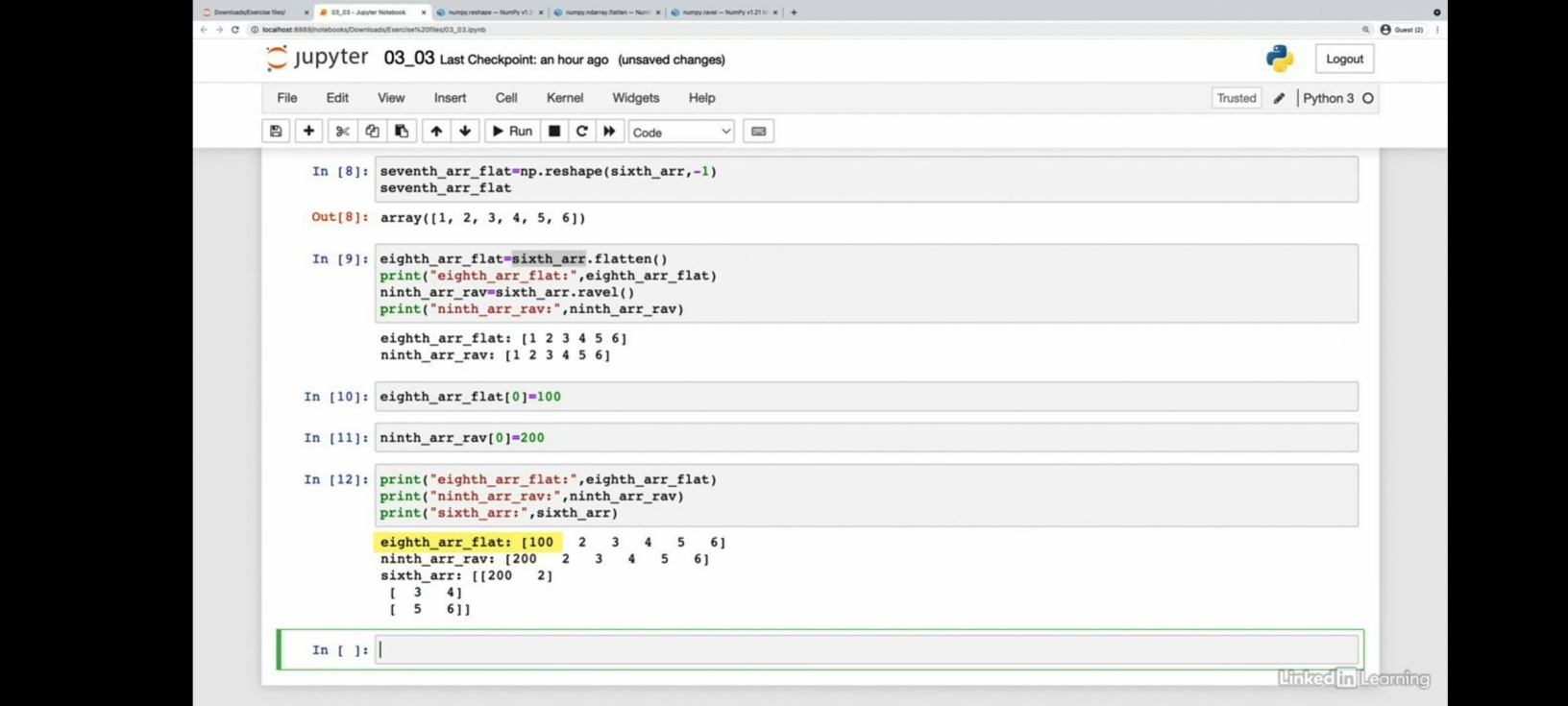
See also

ravel
Return a flattened array.
flat
A 1-D flat iterator over the array.

Examples

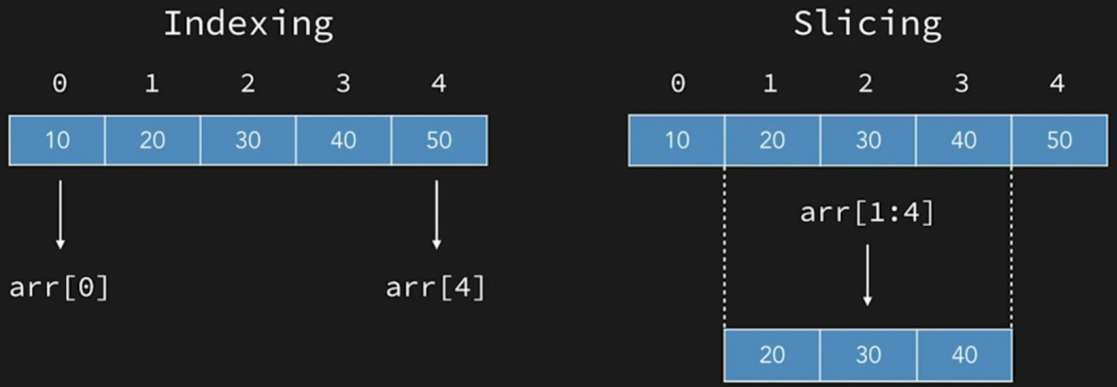
```
>>> a = np.array([[1,2], [3,4]])
```

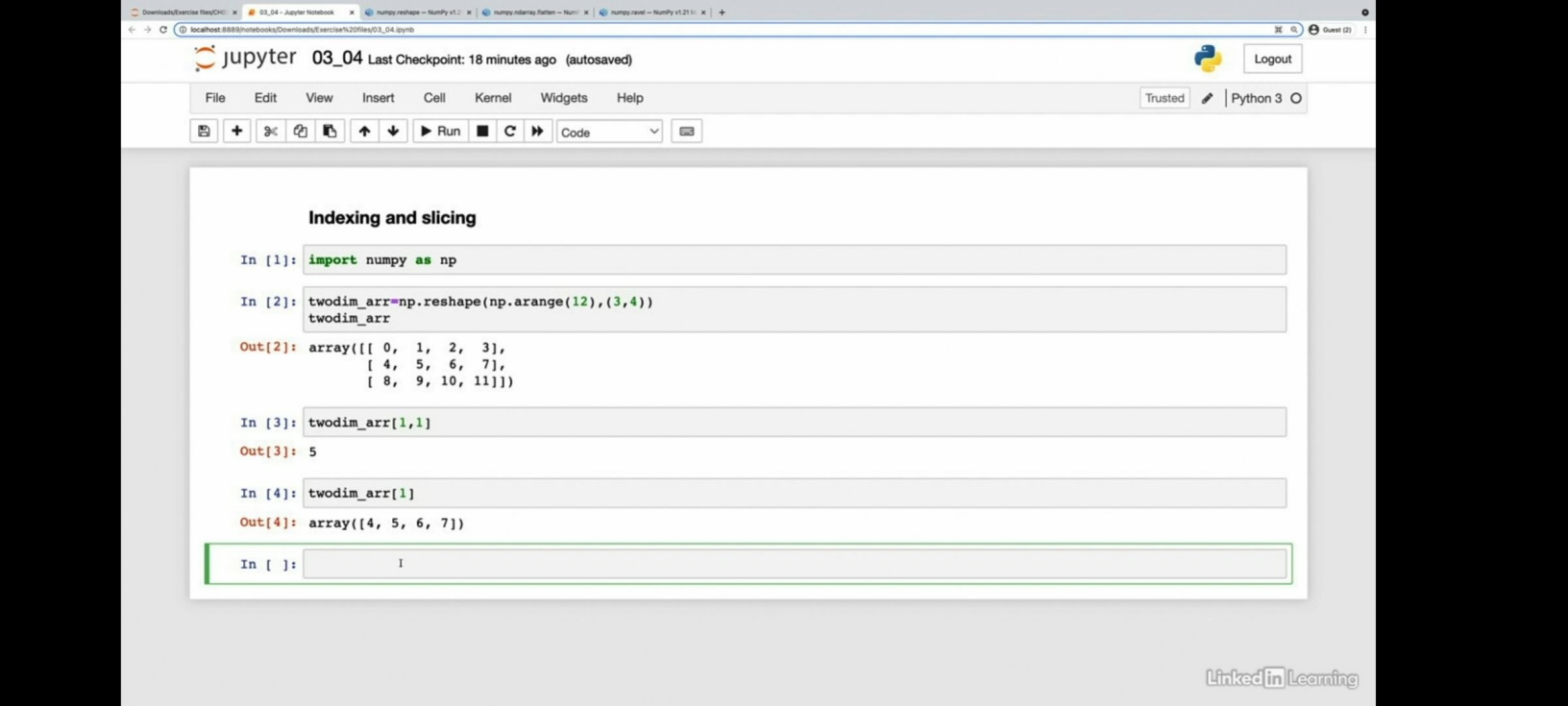

numpy.org/doc/stable/reference/generated/numpy.ravel.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
Constants
Universal functions (ufunc)
Routines
Array creation routines
Array manipulation routines
numpy.copyto
numpy.shape
numpy.reshape
numpy.ravel
numpy.ndarray.flat
numpy.ndarray.flatten
numpy.moveaxis
numpy.rollaxis
numpy.swapaxes
numpy.atleast_2d
numpy.atleast_3d
numpy.ravel
numpy.ravel(a, order='C')
[source]
Return a contiguous flattened array.
A 1-D array, containing the elements of the input, is returned. A copy is made only if needed.
As of NumPy 1.10, the returned array will have the same type as the input array. (for example, a masked array will be returned for a masked array input)
Parameters: a : array_like
Input array. The elements in a are read in the order specified by order, and packed as a 1-D array.
order : {'C','F', 'A', 'K'}, optional
The elements of a are read using this index order. 'C' means to index the elements in row-major, C-style order, with the last axis index changing fastest, back to the first axis index changing slowest. 'F' means to index the elements in column-major, Fortran-style order, with the first index changing fastest, and the last index changing slowest. Note that the 'C' and 'F' options take no account of the memory layout of the underlying array, and only refer to the order of axis indexing. 'A'
contiguous in
the order they
default, 'C' index order is used.
Returns: y : array_like
https://numpy.org/doc/stable/reference/generated/numpy.ravel.html

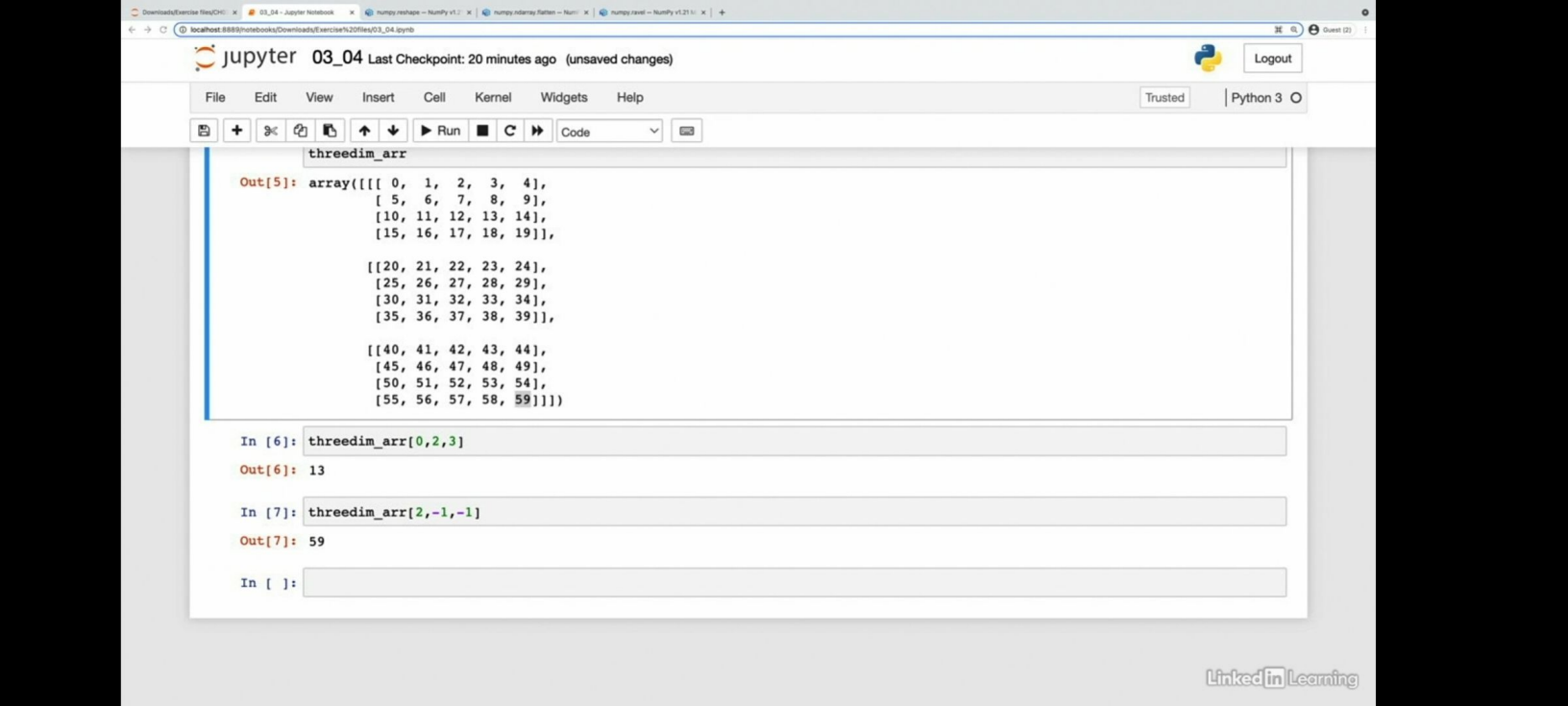

Downloads/Exercise files/
03_03 - Jupyter Notebook
numpy.reshape — NumPy v1.2
numpy.ndarray.flatten — Num
numpy.ravel — NumPy v1.21
localhost:8889/notebooks/Downloads/Exercise%20files/03_03.ipynb
Guest (2)
jupyter 03_03 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [8]:
seventh_arr_flat=np.reshape(sixth_arr,-1)
seventh_arr_flat
Out[8]:
array([1, 2, 3, 4, 5, 6])
In [9]:
eighth_arr_flat=sixth_arr.flatten()
print("eighth_arr_flat:",eighth_arr_flat)
ninth_arr_rav=sixth_arr.ravel()
print("ninth_arr_rav:",ninth_arr_rav)
eighth_arr_flat: [1 2 3 4 5 6]
ninth_arr_rav: [1 2 3 4 5 6]
In [10]:
eighth_arr_flat[0]=100
In [11]:
ninth_arr_rav[0]=200
In [12]:
print("eighth_arr_flat:",eighth_arr_flat)
print("ninth_arr_rav:",ninth_arr_rav)
print("sixth_arr:",sixth_arr)
eighth_arr_flat: [100 2 3 4 5 6]
ninth_arr_rav: [200 2 3 4 5 6]
sixth_arr: [[200 2]
[3 4]
[5 6]]
In [ ]:
Linked in Learning

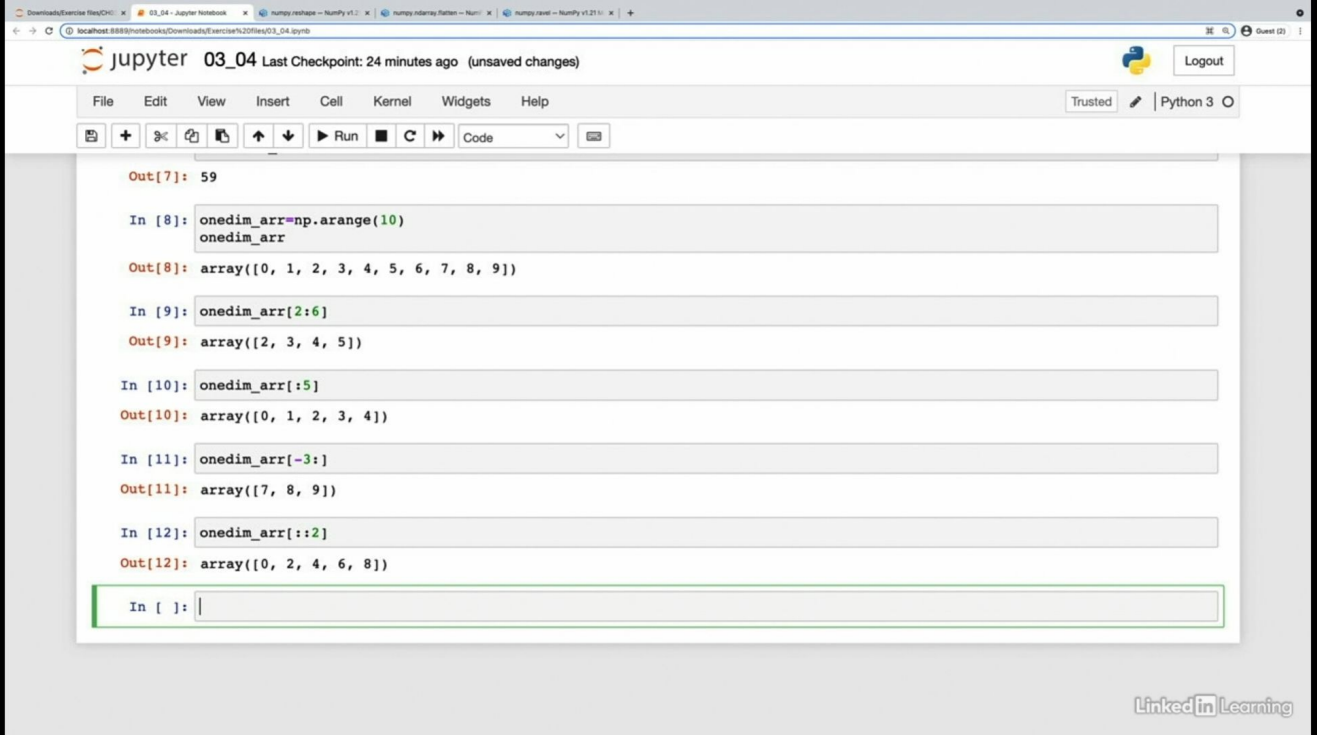
Indexing and Slicing

```
arr = np.array([10,20,30,40,50])
```

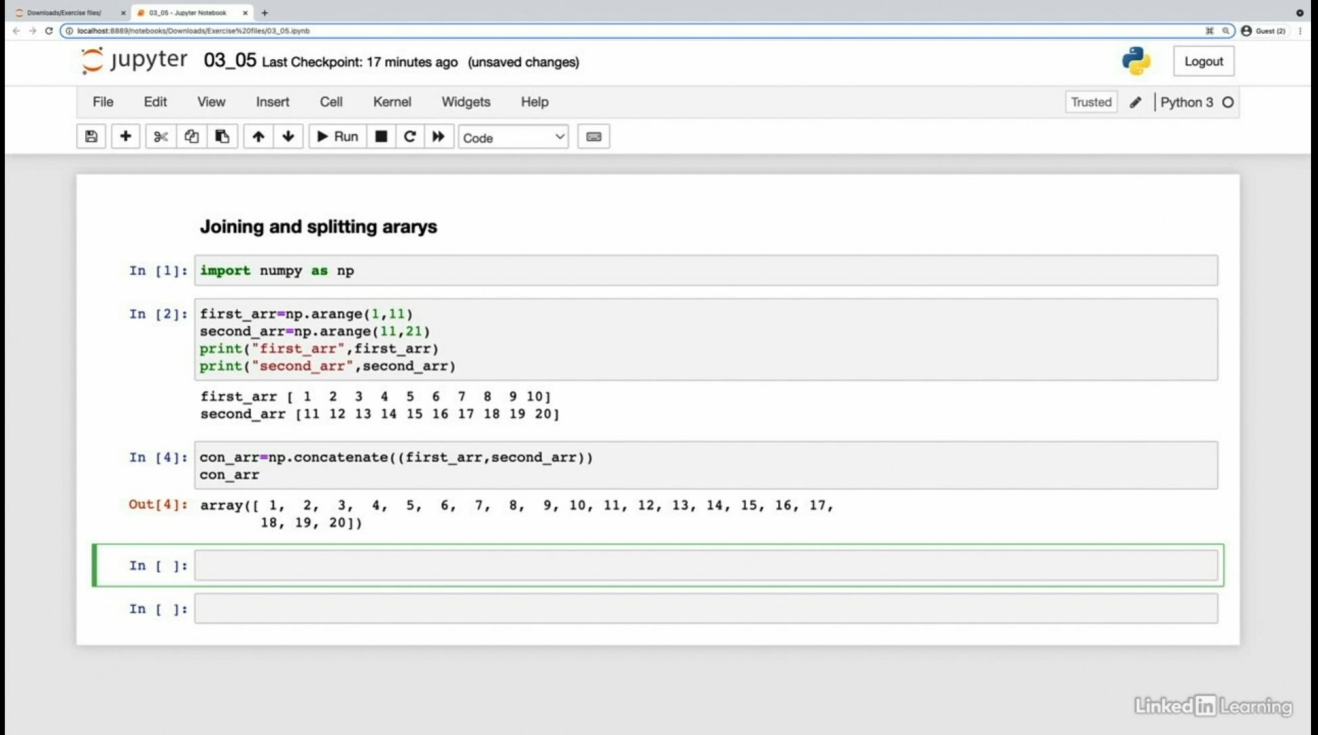


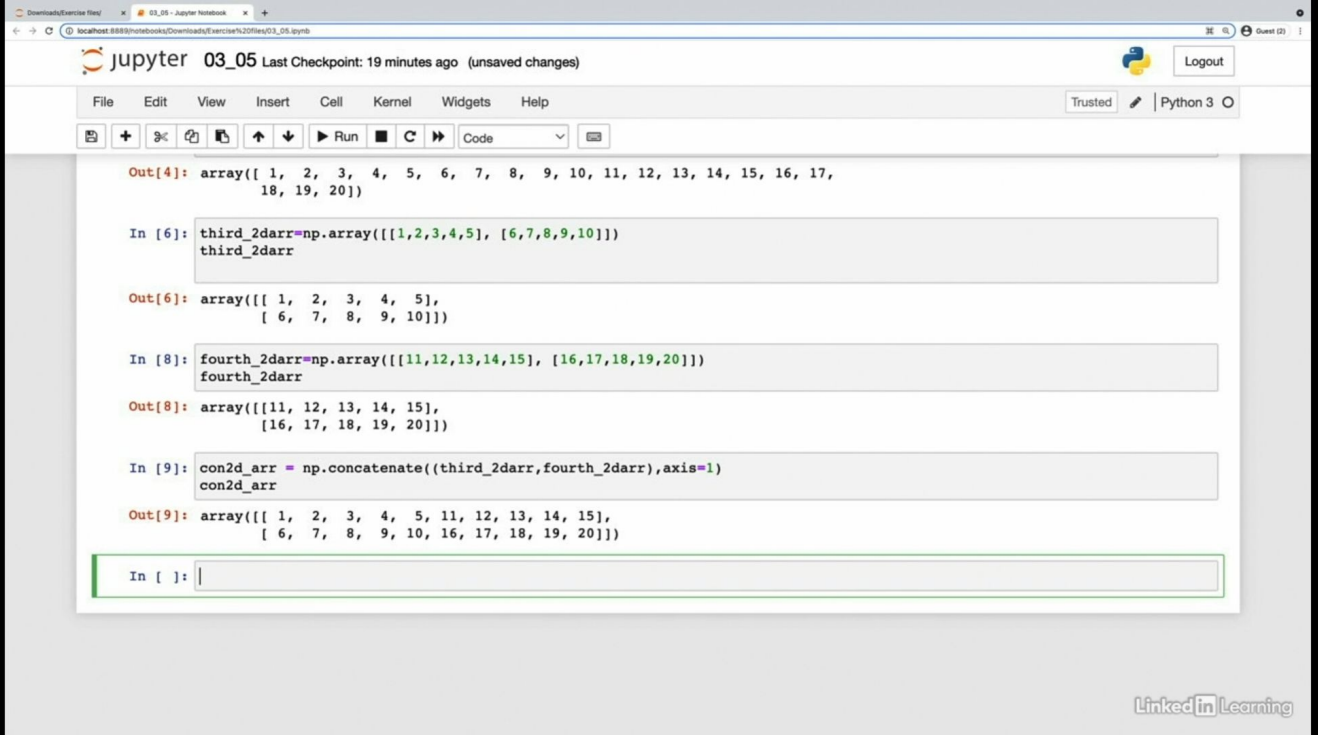

Downloads/Exercise files/CH0
03_04 - Jupyter Notebook
numpy.reshape -- NumPy v1.2
numpy.ndarray.flatten -- Num
numpy.ravel -- NumPy v1.21
localhost:8889/notebooks/Downloads/Exercise%20files/03_04.ipynb
Guest (2)
jupyter 03_04 Last Checkpoint: 18 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Indexing and slicing
In [1]:
import numpy as np
In [2]:
twodim_arr=np.reshape(np.arange(12),(3,4))
twodim_arr
Out[2]:
array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]])
In [3]:
twodim_arr[1,1]
Out[3]:
5
In [4]:
twodim_arr[1]
Out[4]:
array([4, 5, 6, 7])
In [ ]:

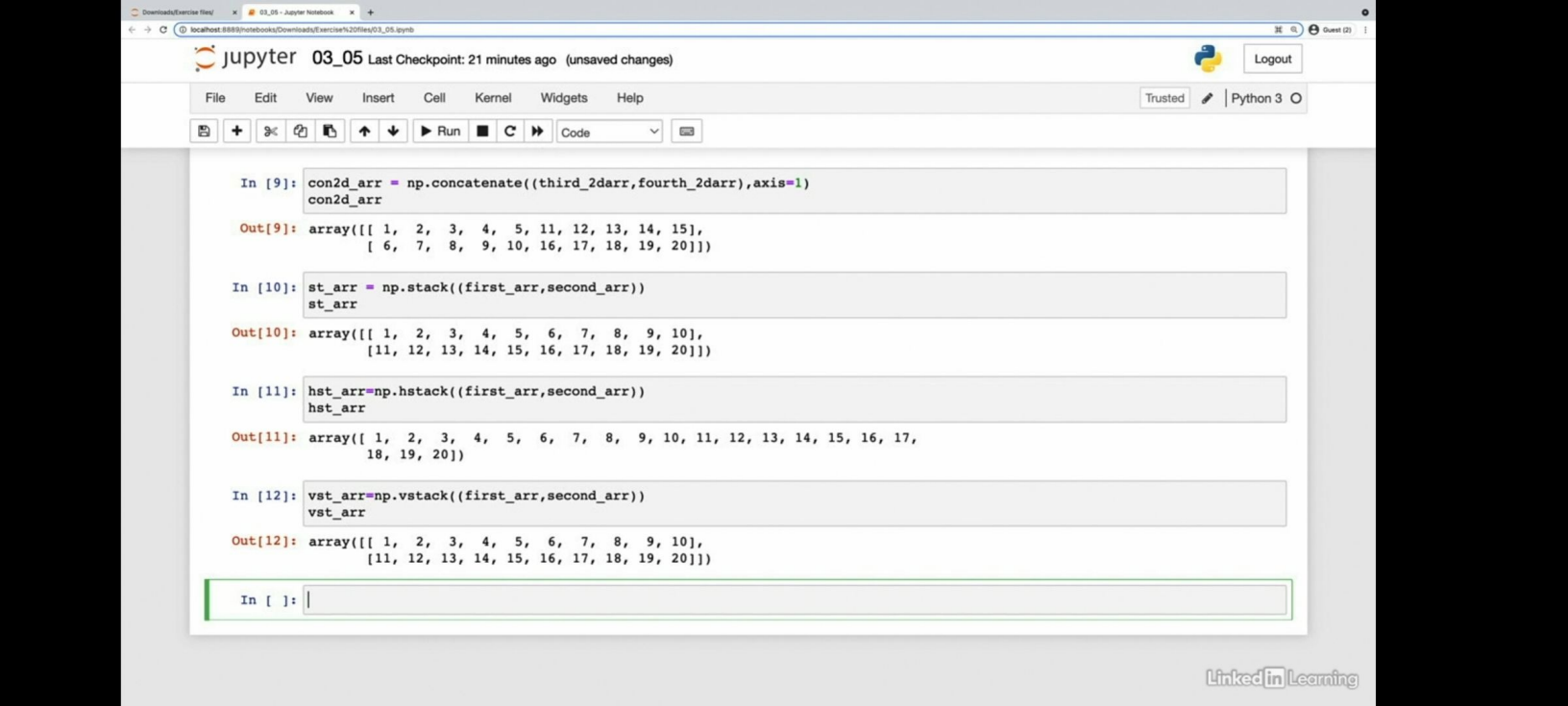

jupyter 03_04 Last Checkpoint: 20 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
threedim_arr
Out[5]: array([[[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9],
[10, 11, 12, 13, 14],
[15, 16, 17, 18, 19]],
[[20, 21, 22, 23, 24],
[25, 26, 27, 28, 29],
[30, 31, 32, 33, 34],
[35, 36, 37, 38, 39]],
[[40, 41, 42, 43, 44],
[45, 46, 47, 48, 49],
[50, 51, 52, 53, 54],
[55, 56, 57, 58, 59]]])
In [6]: threedim_arr[0,2,3]
Out[6]: 13
In [7]: threedim_arr[2,-1,-1]
Out[7]: 59
In [ ]:


jupyter 03_04 Last Checkpoint: 24 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
Out[7]: 59
In [8]: onedim_arr=np.arange(10)
onedim_arr
Out[8]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
In [9]: onedim_arr[2:6]
Out[9]: array([2, 3, 4, 5])
In [10]: onedim_arr[:5]
Out[10]: array([0, 1, 2, 3, 4])
In [11]: onedim_arr[-3:]
Out[11]: array([7, 8, 9])
In [12]: onedim_arr[::2]
Out[12]: array([0, 2, 4, 6, 8])
In [ ]:

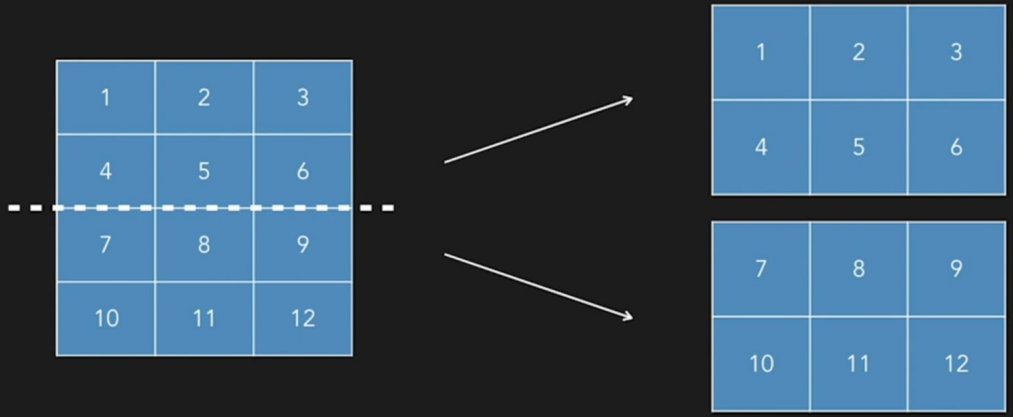
Joining and Splitting Arrays

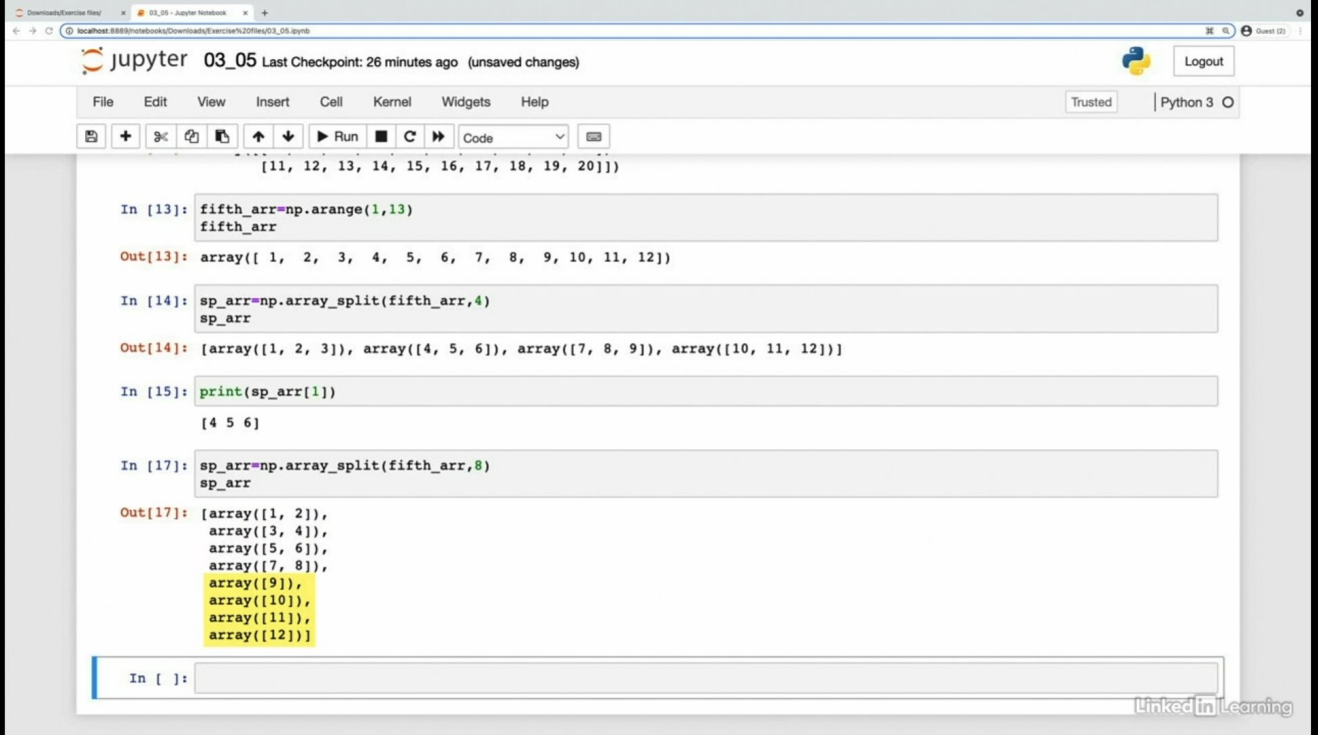

Downloads/Exercise files/
03_05 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/03_05.ipynb
Guest (2)
jupyter 03_05 Last Checkpoint: 17 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Joining and splitting ararys
In [1]:
import numpy as np
In [2]:
first_arr=np.arange(1,11)
second_arr=np.arange(11,21)
print("first_arr",first_arr)
print("second_arr",second_arr)
first_arr [1 2 3 4 5 6 7 8 9 10]
second_arr [11 12 13 14 15 16 17 18 19 20]
In [4]:
con_arr=np.concatenate((first_arr,second_arr))
con_arr
Out[4]:
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20])
In [ ]:
In [ ]:
LinkedIn Learning




Downloads/Exercise files/
03_05 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/03_05.ipynb
Guest (2)
jupyter
03_05 Last Checkpoint: 21 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [9]:
con2d_arr = np.concatenate((third_2darr,fourth_2darr),axis=1)
con2d_arr
Out[9]:
array([[1, 2, 3, 4, 5, 11, 12, 13, 14, 15],
[6, 7, 8, 9, 10, 16, 17, 18, 19, 20]])
In [10]:
st_arr = np.stack((first_arr,second_arr))
st_arr
Out[10]:
array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])
In [11]:
hst_arr=np.hstack((first_arr,second_arr))
hst_arr
Out[11]:
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20])
In [12]:
vst_arr=np.vstack((first_arr,second_arr))
vst_arr
Out[12]:
array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])
In [ ]:
LinkedIn Learning

Splitting Arrays




Downloads/Exercise files/
03_05 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/03_05.ipynb
Guest (2)
jupyter
03_05 Last Checkpoint: 26 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])
In [13]:
fifth_arr=np.arange(1,13)
fifth_arr
Out[13]:
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
In [14]:
sp_arr=np.array_split(fifth_arr,4)
sp_arr
Out[14]:
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9]), array([10, 11, 12])]
In [15]:
print(sp_arr[1])
[4 5 6]
In [17]:
sp_arr=np.array_split(fifth_arr,8)
sp_arr
Out[17]:
[array([1, 2]),
array([3, 4]),
array([5, 6]),
array([7, 8]),
array([9]),
array([10]),
array([11]),
array([12])]
In [ ]:
LinkedIn Learning

Arithmetic Operations and Functions

Vectorization

Operation can be executed in parallel on multiple elements of the array.

Benefits

- Higher performing code
- Less verbose code
- Better maintainability

Arithmetic Operations and Functions

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation
- Specific functions to perform them

jupyter 04_01 Last Checkpoint: 6 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Arithmetic operations

In [1]:
```
import numpy as np
```

In [2]:
```
a=np.arange(1,11)
b=np.arange(21,31)
print("a",a)
print("b",b)
```

```
a [ 1  2  3  4  5  6  7  8  9 10]
b [21 22 23 24 25 26 27 28 29 30]
```

In [3]:
```
a+b
```

Out[3]:
```
array([22, 24, 26, 28, 30, 32, 34, 36, 38, 40])
```

In [5]:
```
b-a
```

Out[5]:
```
array([20, 20, 20, 20, 20, 20, 20, 20, 20, 20])
```

In [6]:
```
a*b
```

Out[6]:
```
array([ 21,  44,  69,  96, 125, 156, 189, 224, 261, 300])
```

In [7]:
```
b/a
```

Out[7]:
```
array([21.        , 11.        ,  7.66666667,  6.        ,  5.        ,
        4.33333333,  3.85714286,  3.5       ,  3.22222222,  3.        ])
```

In [30]:
```
c=np.arange(2,12)
print("c",c)
```

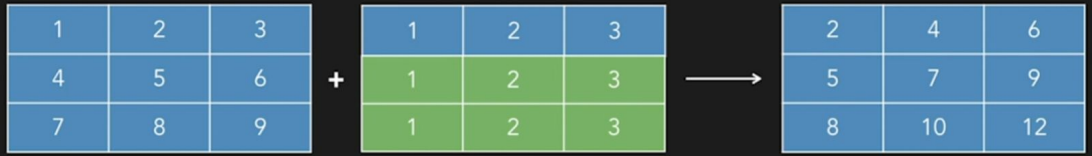
```
c [ 2  3  4  5  6  7  8  9 10 11]
```

In [ ]:

```
In [32]: a*2
Out[32]: array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])

In [33]: np.add(a,b)
Out[33]: array([22, 24, 26, 28, 30, 32, 34, 36, 38, 40])

In [34]: np.subtract(b,a)
Out[34]: array([20, 20, 20, 20, 20, 20, 20, 20, 20, 20])

In [36]: np.multiply(a,b)
Out[36]: array([ 21,  44,  69,  96, 125, 156, 189, 224, 261, 300])

In [37]: np.divide(b,a)
Out[37]: array([21.        , 11.        ,  7.66666667,  6.        ,  5.        ,
                4.33333333,  3.85714286,  3.5       ,  3.22222222,  3.        ])

In [39]: np.mod(b,a)
Out[39]: array([0, 0, 2, 0, 0, 2, 6, 4, 2, 0])

In [40]: np.power(a,c)
Out[40]: array([           1,            8,           81,         1024,
                   15625,       279936,      5764801,    134217728,
                3486784401, 100000000000])

In [41]: np.sqrt(a)
Out[41]: array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
                2.44948974, 2.64575131, 2.82842712, 3.        , 3.16227766])

In [ ]:
```

Broadcasting

Broadcasting



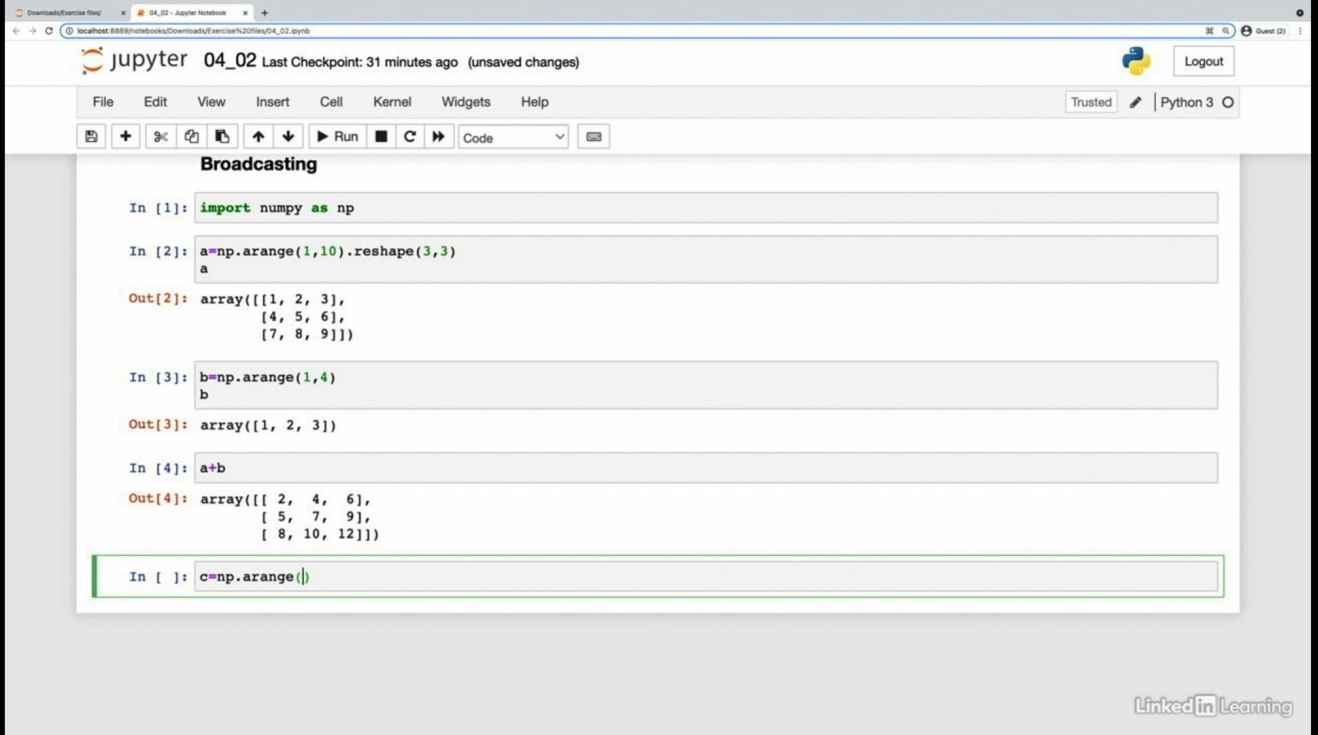
Broadcasting

The term **broadcasting** describes how NumPy treats arrays with different shapes during arithmetic operations.

Subject to certain constraints, the smaller array is **broadcast** across the larger array so that they have compatible shapes.

Dimensions Are Compatible

If their axes on a one-by-one basis, they have either
the same length or **a length of one**


Downloads/Exercise files/
04_02 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_02.ipynb
Guest (2)
jupyter
04_02 Last Checkpoint: 31 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Broadcasting
In [1]:
import numpy as np
In [2]:
a=np.arange(1,10).reshape(3,3)
a
Out[2]:
array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
In [3]:
b=np.arange(1,4)
b
Out[3]:
array([1, 2, 3])
In [4]:
a+b
Out[4]:
array([[2, 4, 6],
[5, 7, 9],
[8, 10, 12]])
In [ ]:
c=np.arange()

Aggregate functions

In [1]:
```
import numpy as np
```

In [2]:
```
first_arr=np.arange(10,110,10)
first_arr
```

Out[2]:
```
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

In [3]:
```
second_arr=np.arange(10,100,10).reshape(3,3)
second_arr
```

Out[3]:
```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

In [4]:
```
third_arr=np.arange(10,110,10).reshape(2,5)
third_arr
```

Out[4]:
```
array([[ 10,  20,  30,  40,  50],
       [ 60,  70,  80,  90, 100]])
```

In [ ]:

```
In [3]: second_arr
```

```
Out[3]: array([[10, 20, 30],
               [40, 50, 60],
               [70, 80, 90]])
```

```
In [4]: third_arr=np.arange(10,110,10).reshape(2,5)
        third_arr
```

```
Out[4]: array([[ 10,  20,  30,  40,  50],
               [ 60,  70,  80,  90, 100]])
```

```
In [5]: first_arr.sum()
```

```
Out[5]: 550
```

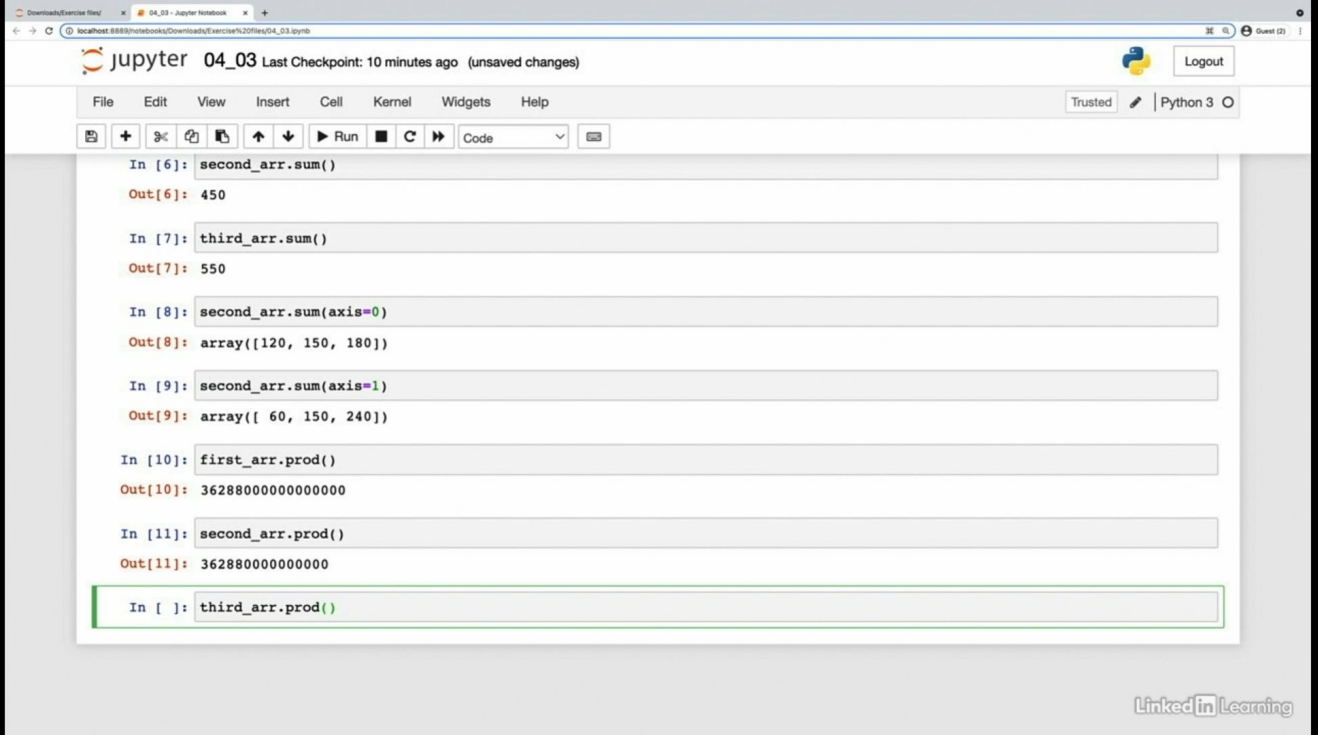
```
In [6]: second_arr.sum()
```

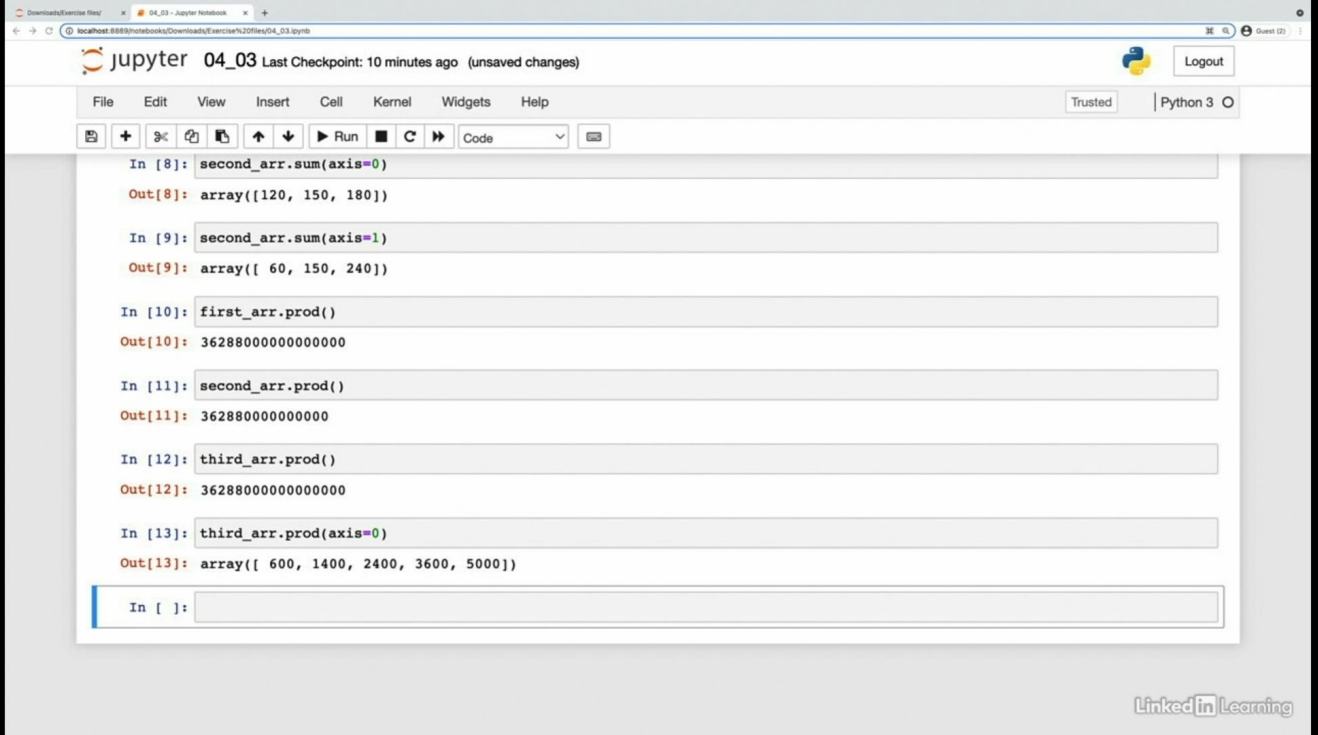
```
Out[6]: 450
```

```
In [7]: third_arr.sum()
```

```
Out[7]: 550
```

```
In [ ]:
```


Downloads/Exercise files/
04_03 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_03.ipynb
Guest (2)
jupyter
04_03 Last Checkpoint: 10 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [6]: second_arr.sum()
Out[6]: 450
In [7]: third_arr.sum()
Out[7]: 550
In [8]: second_arr.sum(axis=0)
Out[8]: array([120, 150, 180])
In [9]: second_arr.sum(axis=1)
Out[9]: array([60, 150, 240])
In [10]: first_arr.prod()
Out[10]: 36288000000000000
In [11]: second_arr.prod()
Out[11]: 362880000000000
In [ ]: third_arr.prod()


Downloads/Exercise files/
04_03 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_03.ipynb
Guest (2)
jupyter
04_03 Last Checkpoint: 10 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [8]: second_arr.sum(axis=0)
Out[8]: array([120, 150, 180])
In [9]: second_arr.sum(axis=1)
Out[9]: array([60, 150, 240])
In [10]: first_arr.prod()
Out[10]: 36288000000000000
In [11]: second_arr.prod()
Out[11]: 362880000000000
In [12]: third_arr.prod()
Out[12]: 36288000000000000
In [13]: third_arr.prod(axis=0)
Out[13]: array([600, 1400, 2400, 3600, 5000])
In [ ]:
LinkedIn Learning

In [11]:
```
second_arr.prod()
```
Out[11]: 362880000000000

In [12]:
```
third_arr.prod()
```
Out[12]: 36288000000000000

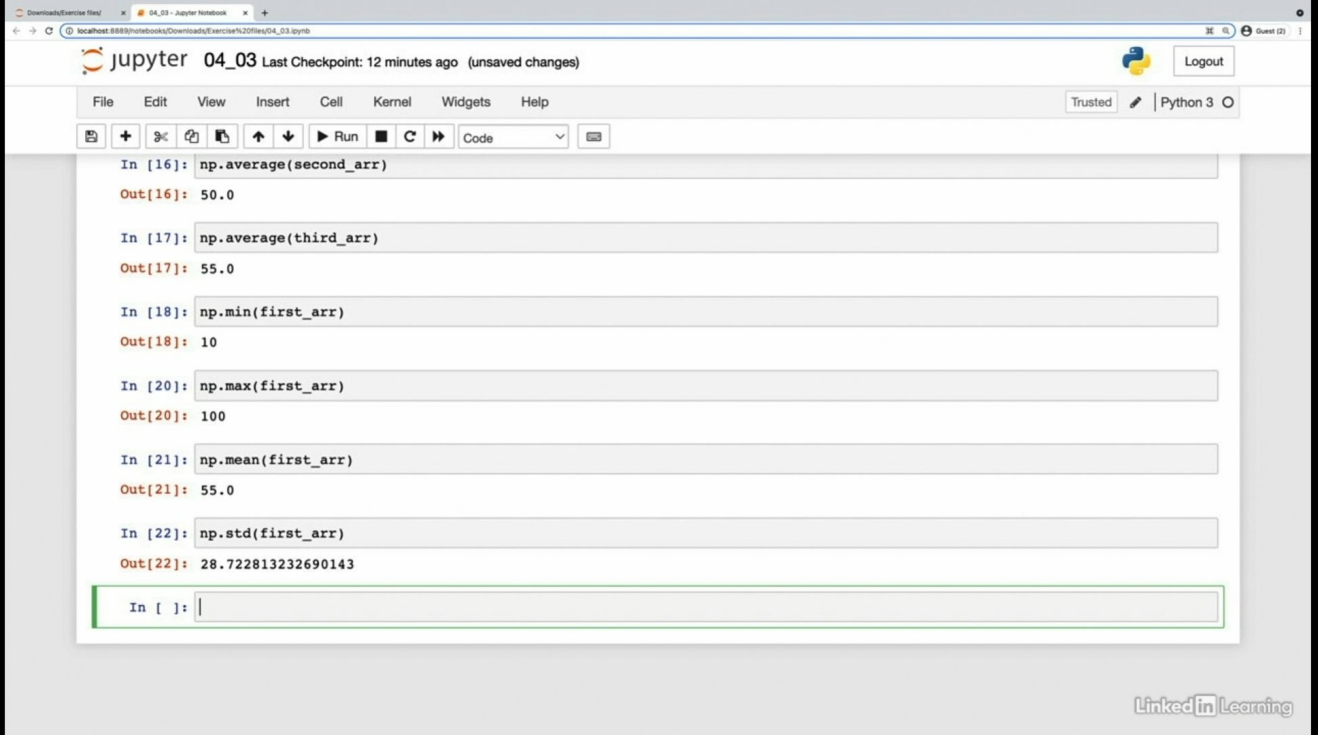
In [13]:
```
third_arr.prod(axis=0)
```
Out[13]: array([600, 1400, 2400, 3600, 5000])

In [15]:
```
np.average(first_arr)
```
Out[15]: 55.0

In [16]:
```
np.average(second_arr)
```
Out[16]: 50.0

In [17]:
```
np.average(third_arr)
```
Out[17]: 55.0

In [ ]:
```

```


Downloads/Exercise files/
04_03 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_03.ipynb
Guest (2)
jupyter
04_03 Last Checkpoint: 12 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [16]: np.average(second_arr)
Out[16]: 50.0
In [17]: np.average(third_arr)
Out[17]: 55.0
In [18]: np.min(first_arr)
Out[18]: 10
In [20]: np.max(first_arr)
Out[20]: 100
In [21]: np.mean(first_arr)
Out[21]: 55.0
In [22]: np.std(first_arr)
Out[22]: 28.722813232690143
In [ ]:
LinkedIn Learning

How to get unique items and counts

In [1]:
```
import numpy as np
```

In [2]:
```
first_arr=np.array([1,2,3,4,5,6,1,2,7,2,1,10,7,8])
```

In [3]:
```
np.unique(first_arr)
```

Out[3]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8, 10])
```

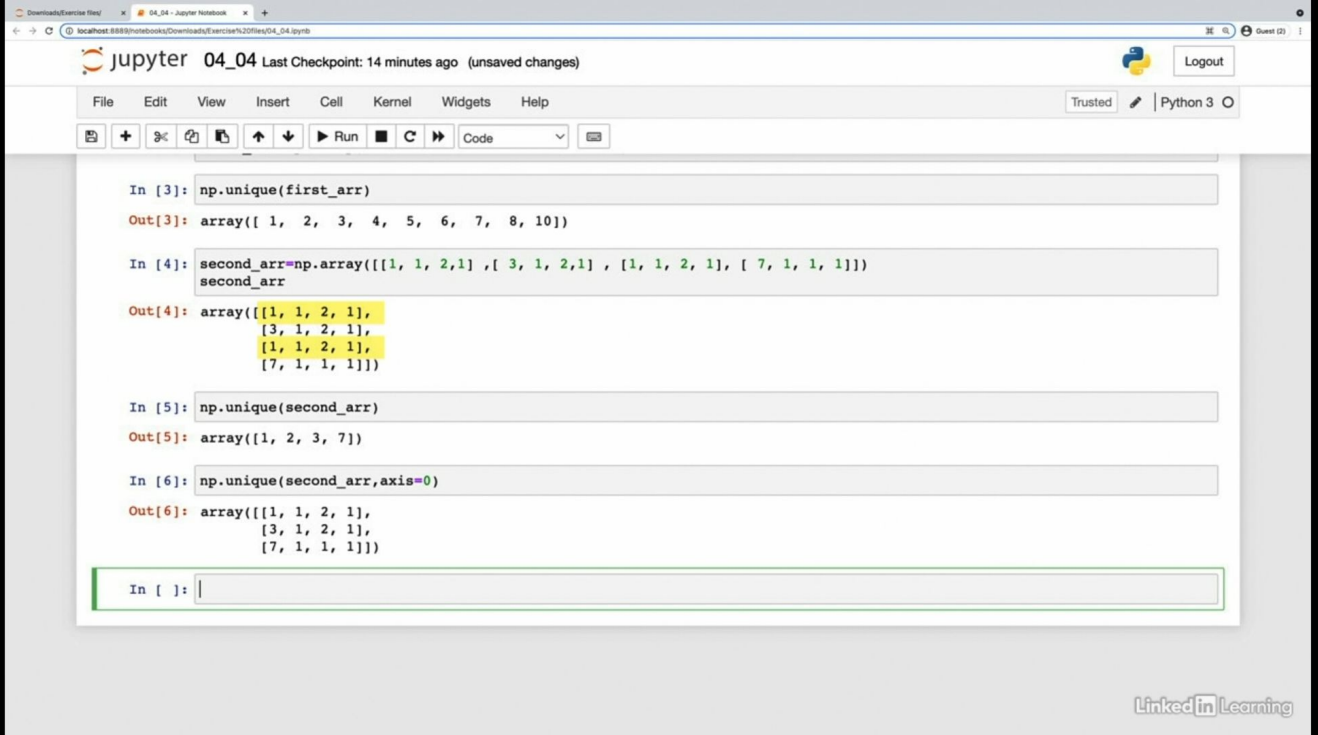
In [4]:
```
second_arr=np.array([[1, 1, 2,1] ,[ 3, 1, 2,1] , [1, 1, 2, 1], [ 7, 1, 1, 1]])
second_arr
```

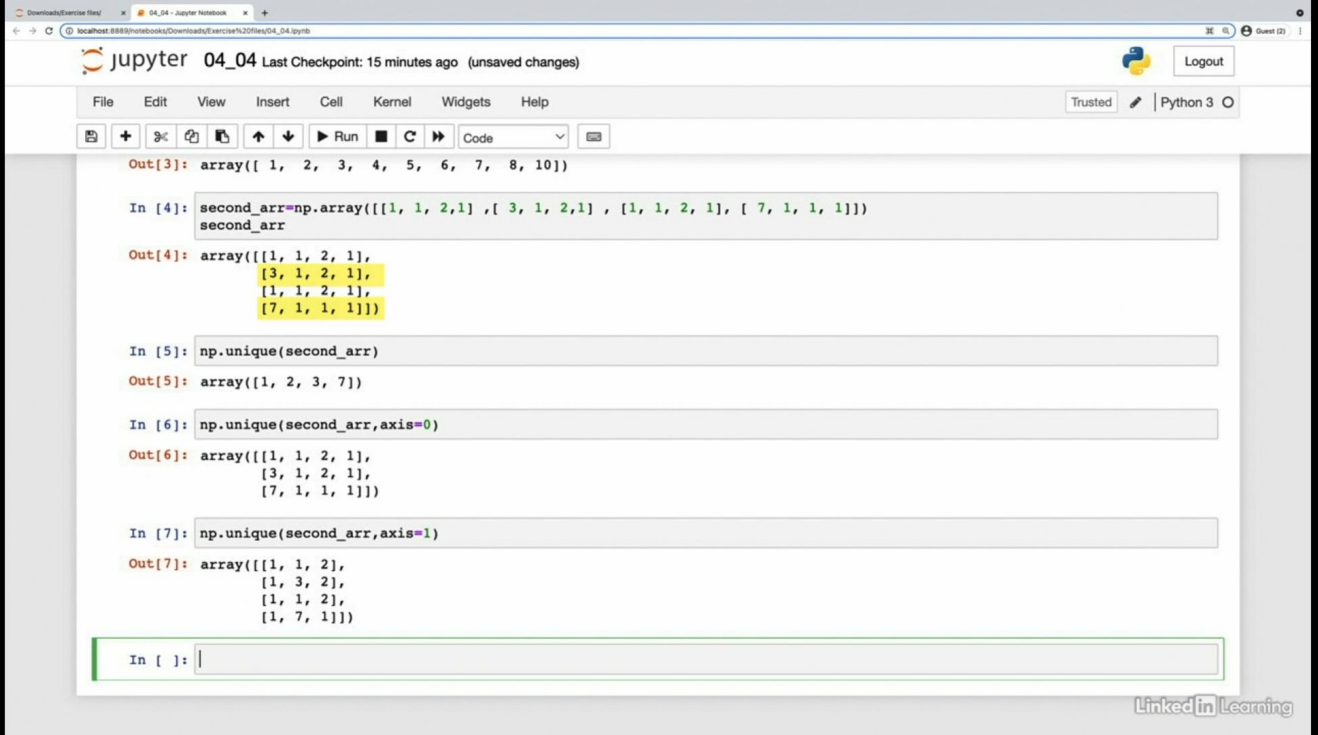
Out[4]:
```
array([[1, 1, 2, 1],
       [3, 1, 2, 1],
       [1, 1, 2, 1],
       [7, 1, 1, 1]])
```

In [5]:
```
np.unique(second_arr)
```

Out[5]:
```
array([1, 2, 3, 7])
```

In [ ]:




Downloads/Exercise files/
04_04 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_04.ipynb
Guest (2)
jupyter 04_04 Last Checkpoint: 15 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Out[3]: array([1, 2, 3, 4, 5, 6, 7, 8, 10])
In [4]: second_arr=np.array([[1, 1, 2,1] ,[3, 1, 2,1] , [1, 1, 2, 1], [7, 1, 1, 1]])
second_arr
Out[4]: array([[1, 1, 2, 1],
[3, 1, 2, 1],
[1, 1, 2, 1],
[7, 1, 1, 1]])
In [5]: np.unique(second_arr)
Out[5]: array([1, 2, 3, 7])
In [6]: np.unique(second_arr,axis=0)
Out[6]: array([[1, 1, 2, 1],
[3, 1, 2, 1],
[7, 1, 1, 1]])
In [7]: np.unique(second_arr,axis=1)
Out[7]: array([[1, 1, 2],
[1, 3, 2],
[1, 1, 2],
[1, 7, 1]])
In [ ]:

```
                [1, 1, 2, 1],
                [7, 1, 1, 1]])
```

In [5]:
```
np.unique(second_arr)
```

Out[5]:
```
array([1, 2, 3, 7])
```

In [6]:
```
np.unique(second_arr,axis=0)
```

Out[6]:
```
array([[1, 1, 2, 1],
       [3, 1, 2, 1],
       [7, 1, 1, 1]])
```

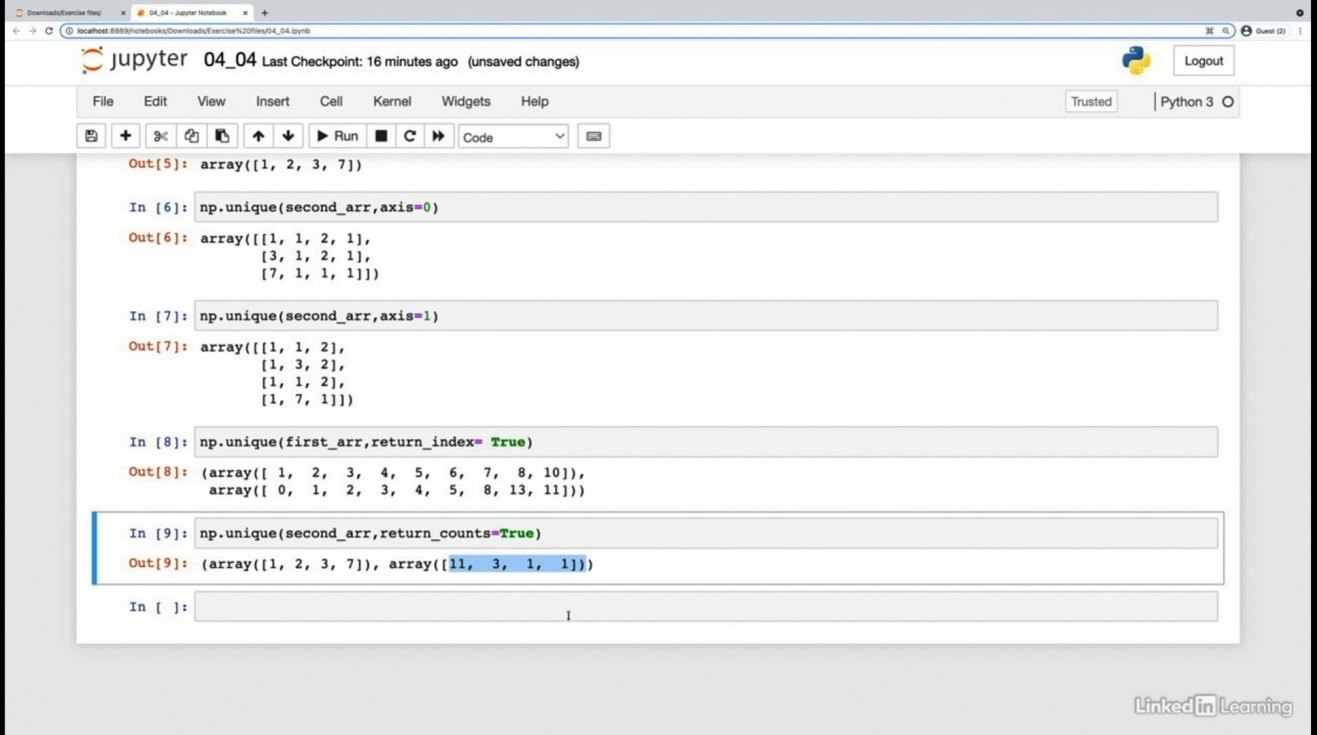
In [7]:
```
np.unique(second_arr,axis=1)
```

Out[7]:
```
array([[1, 1, 2],
       [1, 3, 2],
       [1, 1, 2],
       [1, 7, 1]])
```

In [8]:
```
np.unique(first_arr,return_index= True)
```

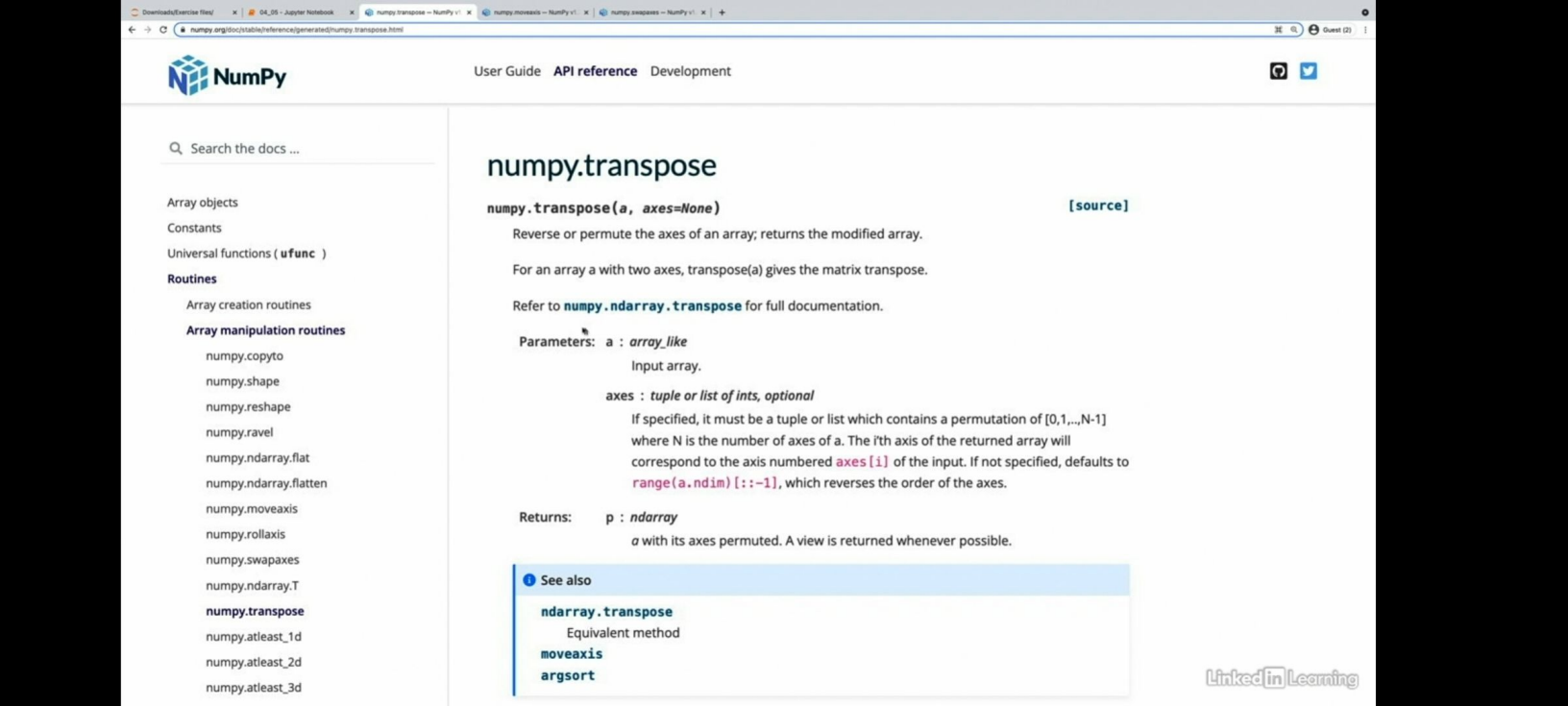
Out[8]:
```
(array([ 1,  2,  3,  4,  5,  6,  7,  8, 10]),
 array([ 0,  1,  2,  3,  4,  5,  8, 13, 11]))
```

In [ ]:



Downloads/Exercise files/
04_04 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_04.ipynb
Guest (2)
jupyter 04_04 Last Checkpoint: 16 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
Out[5]: array([1, 2, 3, 7])

In [6]: np.unique(second_arr,axis=0)

Out[6]: array([[1, 1, 2, 1],
               [3, 1, 2, 1],
               [7, 1, 1, 1]])

In [7]: np.unique(second_arr,axis=1)

Out[7]: array([[1, 1, 2],
               [1, 3, 2],
               [1, 1, 2],
               [1, 7, 1]])

In [8]: np.unique(first_arr,return_index= True)

Out[8]: (array([ 1,  2,  3,  4,  5,  6,  7,  8, 10]),
         array([ 0,  1,  2,  3,  4,  5,  8, 13, 11]))

In [9]: np.unique(second_arr,return_counts=True)

Out[9]: (array([1, 2, 3, 7]), array([11,  3,  1,  1]))

In [ ]:
```



Downloads/Exercise files/ | 04_05 - Jupyter Notebook | numpy.transpose — NumPy v1 | numpy.moveaxis — NumPy v1 | numpy.swapaxes — NumPy v1

numpy.org/doc/stable/reference/generated/numpy.transpose.html

Guest (2)

NumPy

User Guide **API reference** Development

Search the docs ...

Array objects
Constants
Universal functions (`ufunc`)
Routines
Array creation routines
Array manipulation routines
numpy.copyto
numpy.shape
numpy.reshape
numpy.ravel
numpy.ndarray.flat
numpy.ndarray.flatten
numpy.moveaxis
numpy.rollaxis
numpy.swapaxes
numpy.ndarray.T
numpy.transpose
numpy.atleast_1d
numpy.atleast_2d
numpy.atleast_3d

numpy.transpose

`numpy.transpose(a, axes=None)` [source]

Reverse or permute the axes of an array; returns the modified array.

For an array a with two axes, transpose(a) gives the matrix transpose.

Refer to **`numpy.ndarray.transpose`** for full documentation.

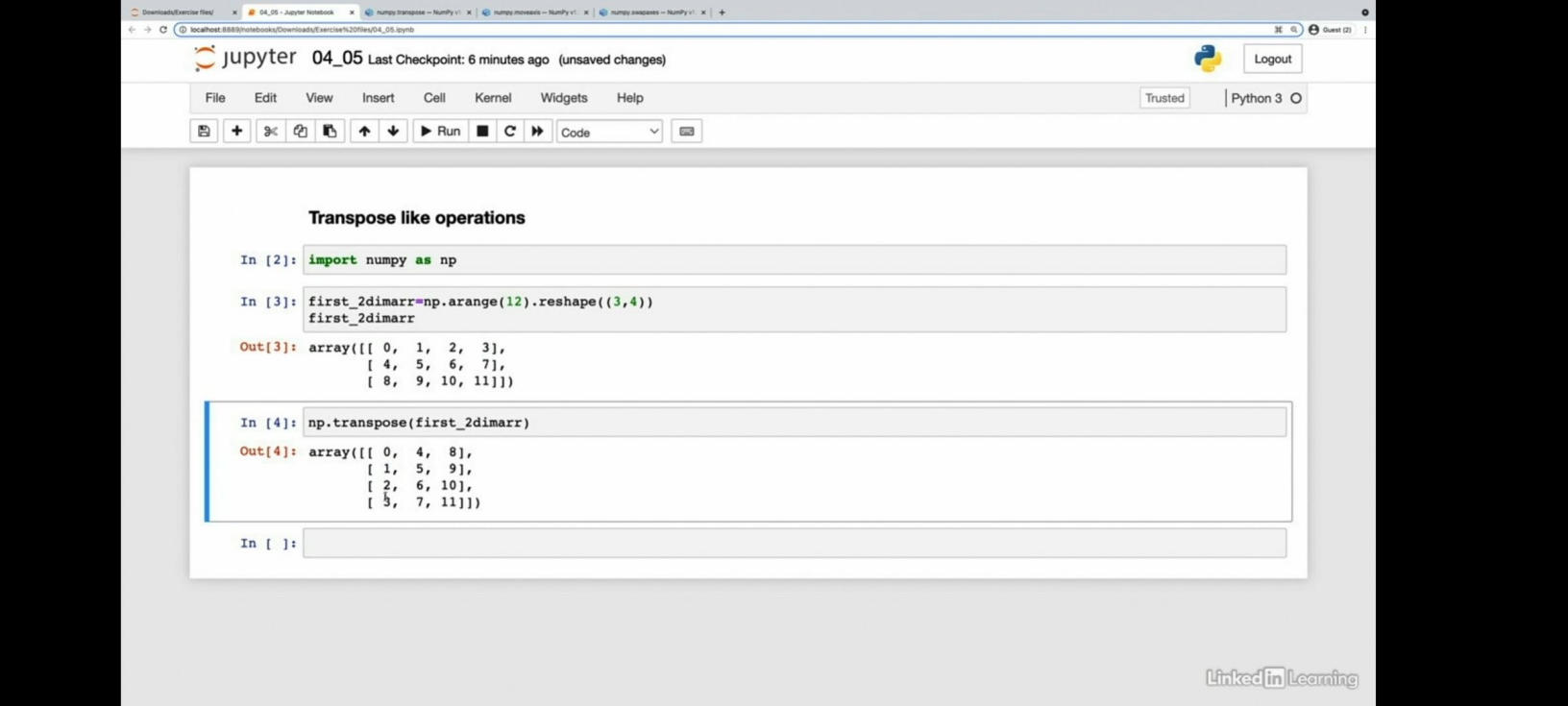
Parameters: a : *array_like*
Input array.

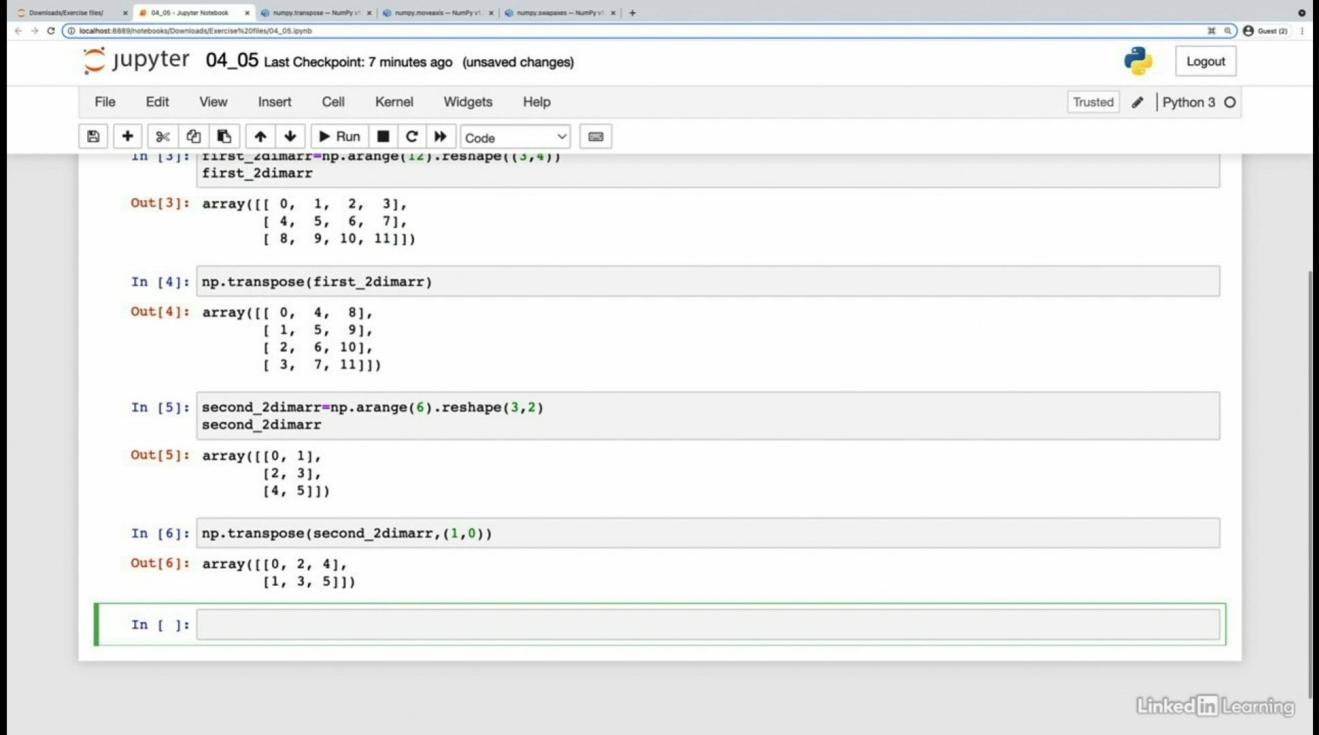
axes : *tuple or list of ints, optional*
If specified, it must be a tuple or list which contains a permutation of [0,1,..,N-1] where N is the number of axes of a. The *i*'th axis of the returned array will correspond to the axis numbered `axes[i]` of the input. If not specified, defaults to `range(a.ndim)[::-1]`, which reverses the order of the axes.

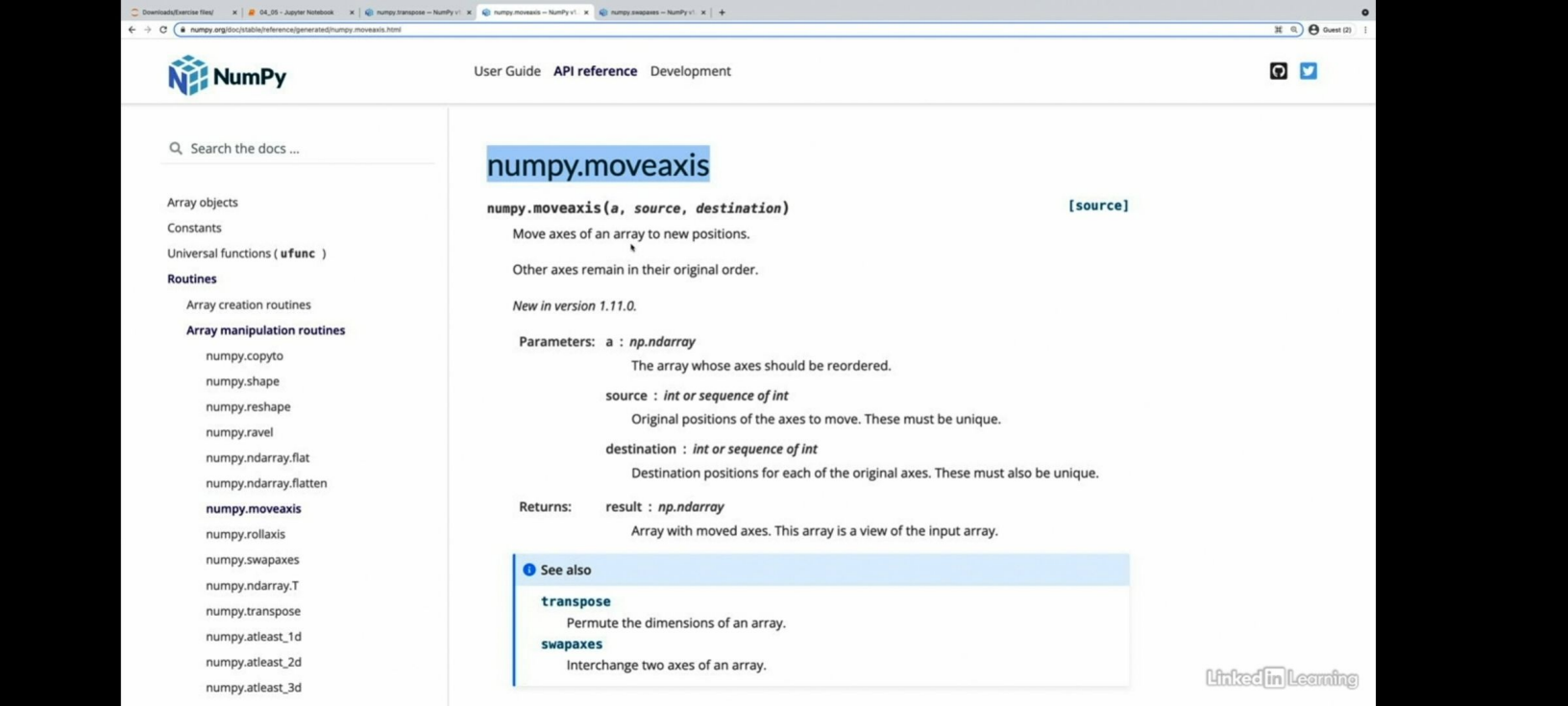
Returns: p : *ndarray*
a with its axes permuted. A view is returned whenever possible.

See also

`ndarray.transpose`
Equivalent method
`moveaxis`
`argsort`


Downloads/Exercise files/
04_05 - Jupyter Notebook
numpy.transpose -- NumPy v1
numpy.moveaxis -- NumPy v1
numpy.swapaxes -- NumPy v1
localhost:8889/notebooks/Downloads/Exercise%20files/04_05.ipynb
Guest (2)
jupyter
04_05 Last Checkpoint: 6 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Transpose like operations
In [2]:
import numpy as np
In [3]:
first_2dimarr=np.arange(12).reshape((3,4))
first_2dimarr
Out[3]:
array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]])
In [4]:
np.transpose(first_2dimarr)
Out[4]:
array([[0, 4, 8],
[1, 5, 9],
[2, 6, 10],
[3, 7, 11]])
In [ ]:


jupyter 04_05 Last Checkpoint: 7 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
In [3]:
first_2dimarr=np.arange(12).reshape((3,4))
first_2dimarr
Out[3]:
array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]])
In [4]:
np.transpose(first_2dimarr)
Out[4]:
array([[0, 4, 8],
[1, 5, 9],
[2, 6, 10],
[3, 7, 11]])
In [5]:
second_2dimarr=np.arange(6).reshape(3,2)
second_2dimarr
Out[5]:
array([[0, 1],
[2, 3],
[4, 5]])
In [6]:
np.transpose(second_2dimarr,(1,0))
Out[6]:
array([[0, 2, 4],
[1, 3, 5]])
In [ ]:


numpy.org/doc/stable/reference/generated/numpy.moveaxis.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
Constants
Universal functions (ufunc)
Routines
Array creation routines
Array manipulation routines
numpy.copyto
numpy.shape
numpy.reshape
numpy.ravel
numpy.ndarray.flat
numpy.ndarray.flatten
numpy.moveaxis
numpy.rollaxis
numpy.swapaxes
numpy.ndarray.T
numpy.transpose
numpy.atleast_1d
numpy.atleast_2d
numpy.atleast_3d
numpy.moveaxis
numpy.moveaxis(a, source, destination)
[source]
Move axes of an array to new positions.
Other axes remain in their original order.
New in version 1.11.0.
Parameters:
a : np.ndarray
The array whose axes should be reordered.
source : int or sequence of int
Original positions of the axes to move. These must be unique.
destination : int or sequence of int
Destination positions for each of the original axes. These must also be unique.
Returns:
result : np.ndarray
Array with moved axes. This array is a view of the input array.
See also
transpose
Permute the dimensions of an array.
swapaxes
Interchange two axes of an array.
LinkedIn Learning

```
Out[5]: array([[0, 1],
               [2, 3],
               [4, 5]])
```

```
In [6]: np.transpose(second_2dimarr,(1,0))
```

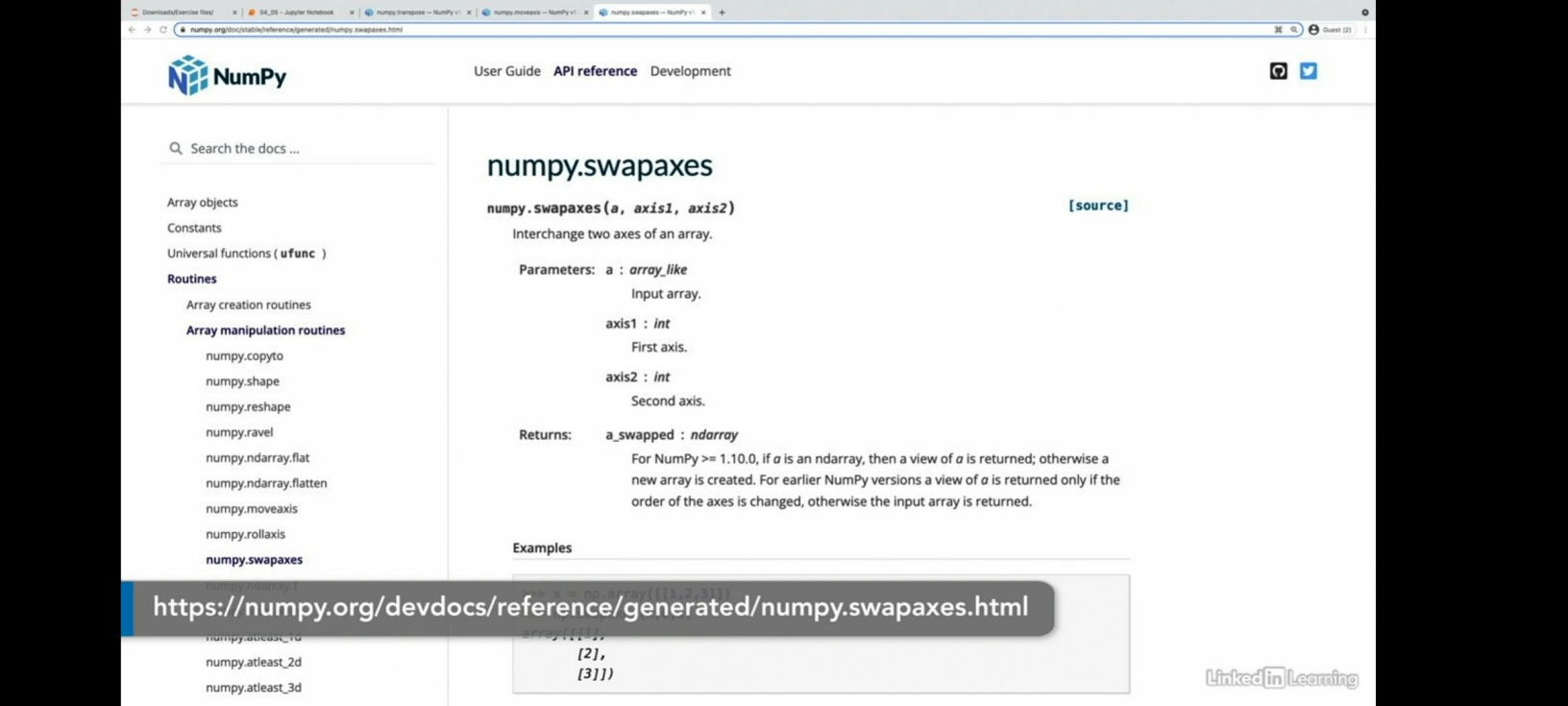
```
Out[6]: array([[0, 2, 4],
               [1, 3, 5]])
```

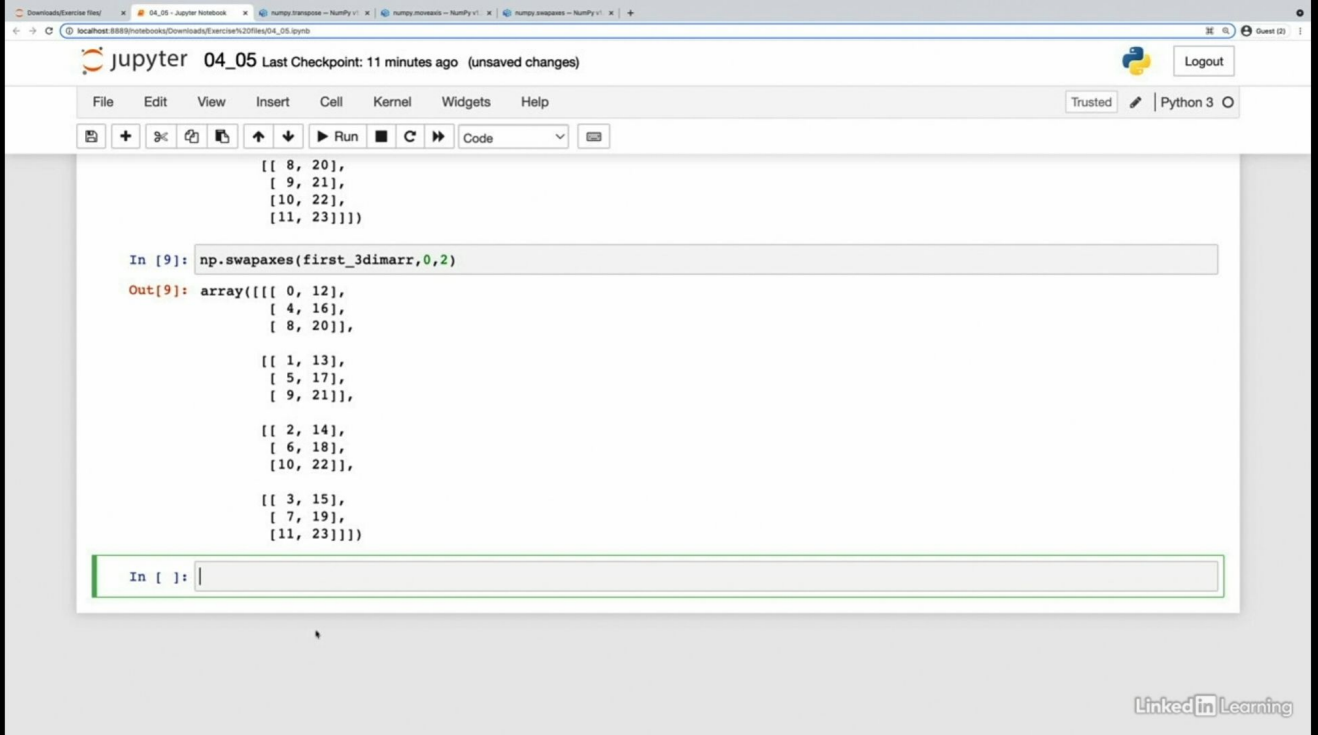
```
In [7]: first_3dimarr=np.arange(24).reshape(2,3,4)
        first_3dimarr
```

```
Out[7]: array([[[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]],

               [[12, 13, 14, 15],
                [16, 17, 18, 19],
                [20, 21, 22, 23]]])
```

```
In [8]: np.moveaxis(first_3dimarr,0,-1)
```

```
Out[8]: array([[[ 0, 12],
                [ 1, 13],
                [ 2, 14],
                [ 3, 15]],

               [[ 4, 16],
                [ 5, 17],
                [ 6, 18],
                [ 7, 19]],
```


numpy.org/doc/stable/reference/generated/numpy.swapaxes.html
NumPy
User Guide
API reference
Development
Search the docs ...
Array objects
Constants
Universal functions (ufunc)
Routines
Array creation routines
Array manipulation routines
numpy.copyto
numpy.shape
numpy.reshape
numpy.ravel
numpy.ndarray.flat
numpy.ndarray.flatten
numpy.moveaxis
numpy.rollaxis
numpy.swapaxes
numpy.atleast_2d
numpy.atleast_3d
numpy.swapaxes
numpy.swapaxes(a, axis1, axis2)
[source]
Interchange two axes of an array.
Parameters:
a : array_like
Input array.
axis1 : int
First axis.
axis2 : int
Second axis.
Returns:
a_swapped : ndarray
For NumPy >= 1.10.0, if a is an ndarray, then a view of a is returned; otherwise a new array is created. For earlier NumPy versions a view of a is returned only if the order of the axes is changed, otherwise the input array is returned.
Examples
[2],
[3]])
https://numpy.org/devdocs/reference/generated/numpy.swapaxes.html
LinkedIn Learning


Downloads/Exercise files/
04_05 - Jupyter Notebook
numpy.transpose -- NumPy v1
numpy.moveaxis -- NumPy v1
numpy.swapaxes -- NumPy v1
localhost:8889/notebooks/Downloads/Exercise%20files/04_05.ipynb
Guest (2)
jupyter
04_05 Last Checkpoint: 11 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
[[8, 20],
[9, 21],
[10, 22],
[11, 23]]])
In [9]: np.swapaxes(first_3dimarr,0,2)
Out[9]: array([[[0, 12],
[4, 16],
[8, 20]],
[[1, 13],
[5, 17],
[9, 21]],
[[2, 14],
[6, 18],
[10, 22]],
[[3, 15],
[7, 19],
[11, 23]]])
In [ ]:

Reversing an array

In [1]:
```
import numpy as np
```

In [2]:
```
arr_1dim=[10,1,9,2,8,3,7,4,6,5]
arr_1dim
```

Out[2]:
```
[10, 1, 9, 2, 8, 3, 7, 4, 6, 5]
```

In [3]:
```
arr_1dim[::-1]
```

Out[3]:
```
[5, 6, 4, 7, 3, 8, 2, 9, 1, 10]
```

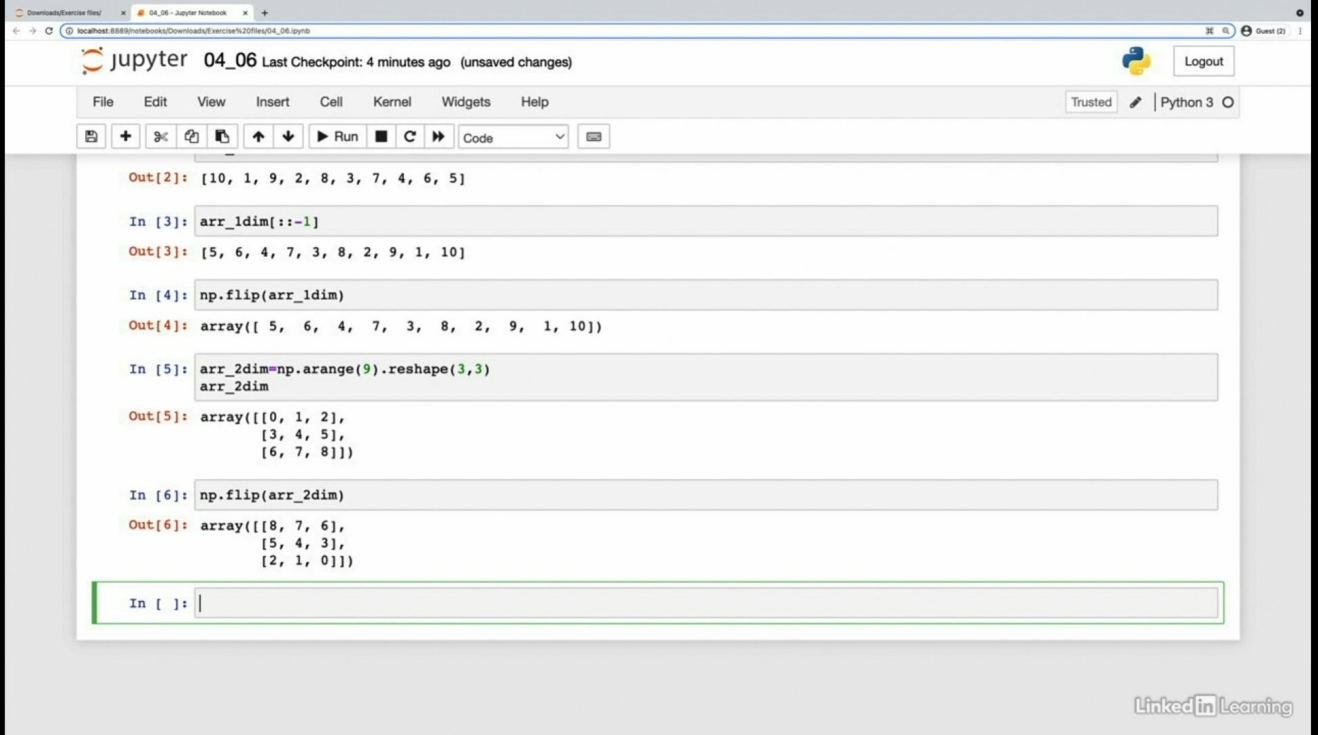
In [4]:
```
np.flip(arr_1dim)
```

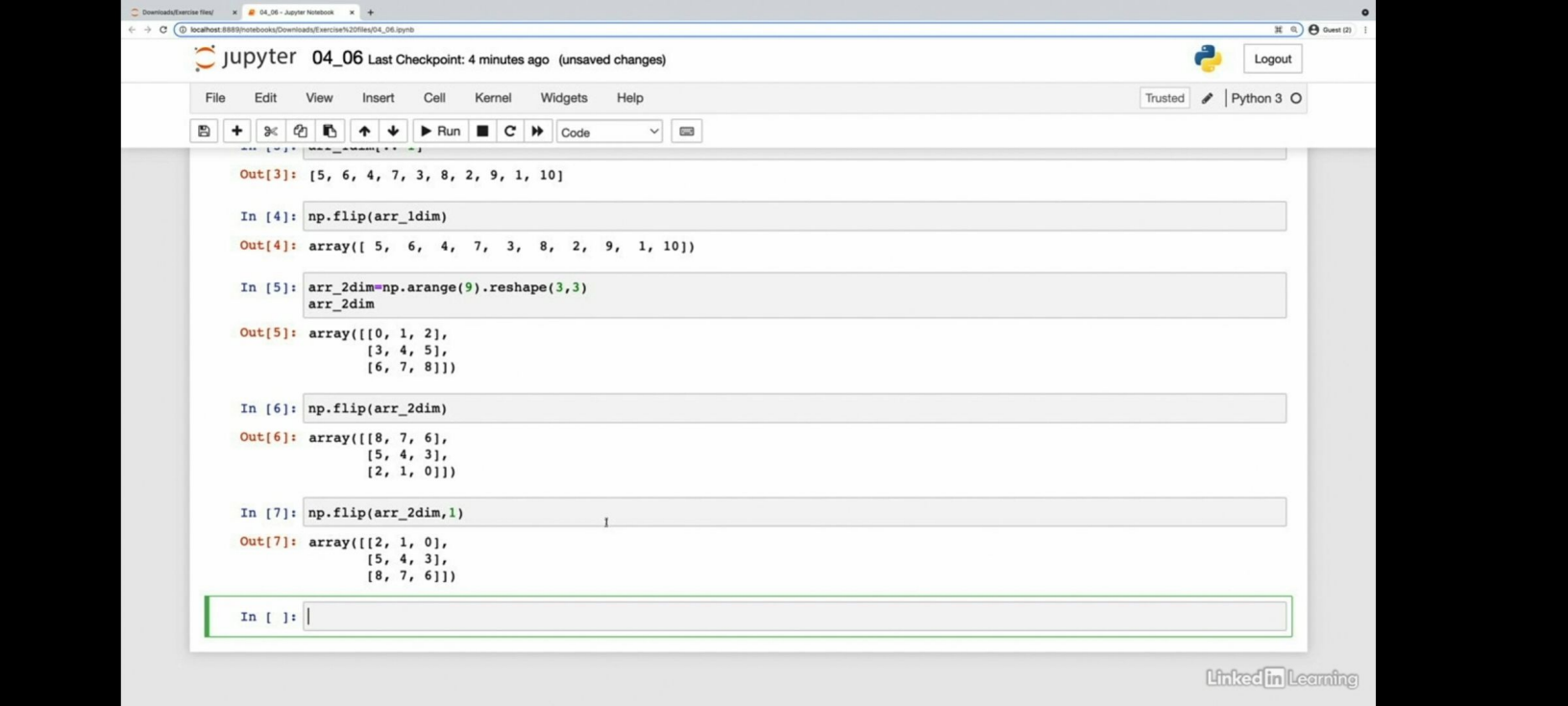
Out[4]:
```
array([ 5,  6,  4,  7,  3,  8,  2,  9,  1, 10])
```

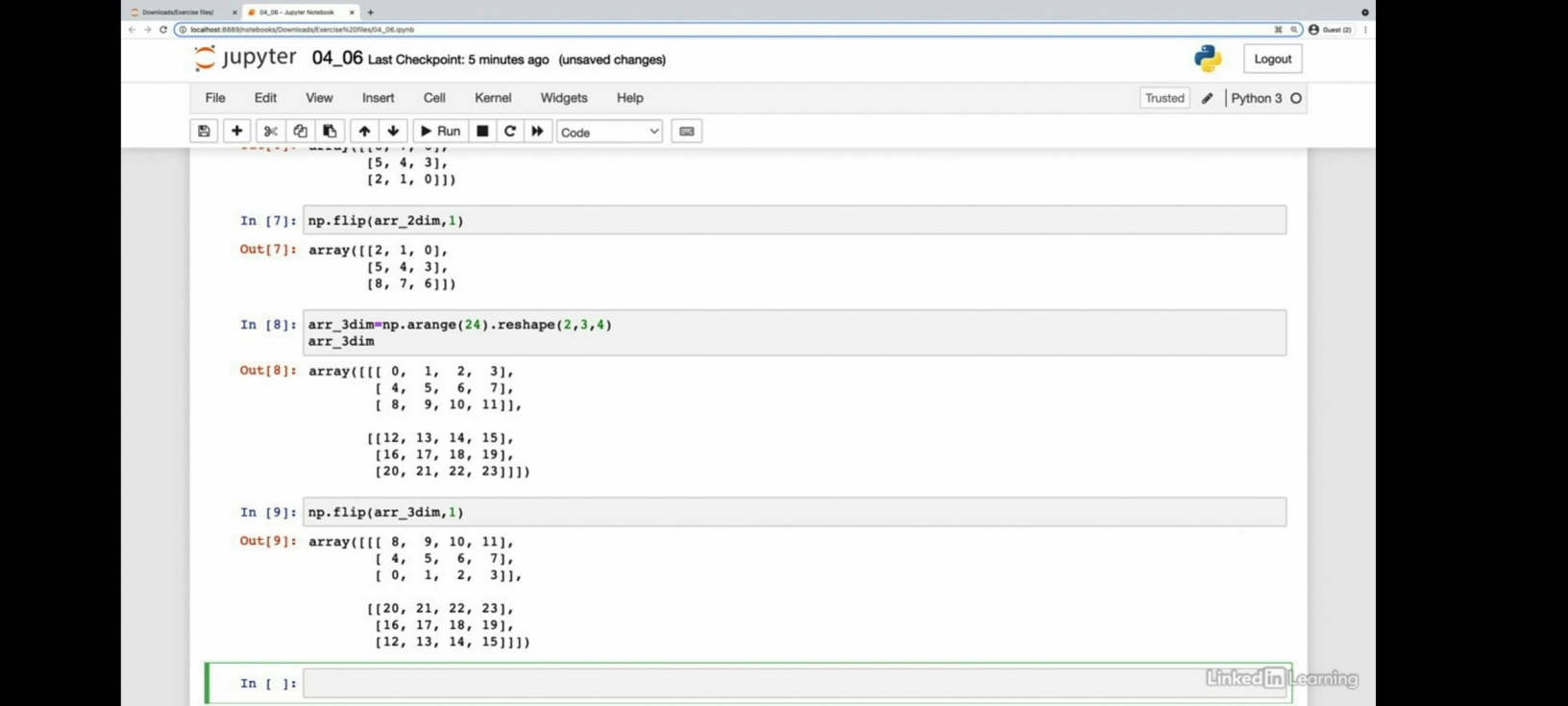
In [5]:
```
arr_2dim=np.arange(9).reshape(3,3)
arr_2dim
```

Out[5]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [ ]:




Downloads/Exercise files/
04_06 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_06.ipynb
Guest (2)
jupyter
04_06 Last Checkpoint: 4 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Out[3]: [5, 6, 4, 7, 3, 8, 2, 9, 1, 10]
In [4]: np.flip(arr_1dim)
Out[4]: array([5, 6, 4, 7, 3, 8, 2, 9, 1, 10])
In [5]: arr_2dim=np.arange(9).reshape(3,3)
arr_2dim
Out[5]: array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
In [6]: np.flip(arr_2dim)
Out[6]: array([[8, 7, 6],
[5, 4, 3],
[2, 1, 0]])
In [7]: np.flip(arr_2dim,1)
Out[7]: array([[2, 1, 0],
[5, 4, 3],
[8, 7, 6]])
In [ ]:
LinkedIn Learning


Downloads/Exercise files/
04_06 - Jupyter Notebook
localhost:8889/notebooks/Downloads/Exercise%20files/04_06.ipynb
Guest (2)
jupyter
04_06 Last Checkpoint: 5 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
[5, 4, 3],
[2, 1, 0]])
In [7]:
np.flip(arr_2dim,1)
Out[7]:
array([[2, 1, 0],
[5, 4, 3],
[8, 7, 6]])
In [8]:
arr_3dim=np.arange(24).reshape(2,3,4)
arr_3dim
Out[8]:
array([[[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]],
[[12, 13, 14, 15],
[16, 17, 18, 19],
[20, 21, 22, 23]]])
In [9]:
np.flip(arr_3dim,1)
Out[9]:
array([[[8, 9, 10, 11],
[4, 5, 6, 7],
[0, 1, 2, 3]],
[[20, 21, 22, 23],
[16, 17, 18, 19],
[12, 13, 14, 15]]])
In [ ]:
LinkedIn Learning

```
                [16, 17, 18, 19],
                [20, 21, 22, 23]]])
```

In [9]:
```
np.flip(arr_3dim,1)
```

Out[9]:
```
array([[[ 8,  9, 10, 11],
        [ 4,  5,  6,  7],
        [ 0,  1,  2,  3]],

       [[20, 21, 22, 23],
        [16, 17, 18, 19],
        [12, 13, 14, 15]]])
```

In [10]:
```
np.flip(arr_3dim,2)
```

Out[10]:
```
array([[[ 3,  2,  1,  0],
        [ 7,  6,  5,  4],
        [11, 10,  9,  8]],

       [[15, 14, 13, 12],
        [19, 18, 17, 16],
        [23, 22, 21, 20]]])
```

In [ ]: