Watch by Thursday, October 22, 2020 | **Lesson #6**

Multi-dimensional NumPy arrays, dictionaries, and datetime

OCEAN 215 | Autumn 2020

Ethan Campbell and **Katy Christensen**

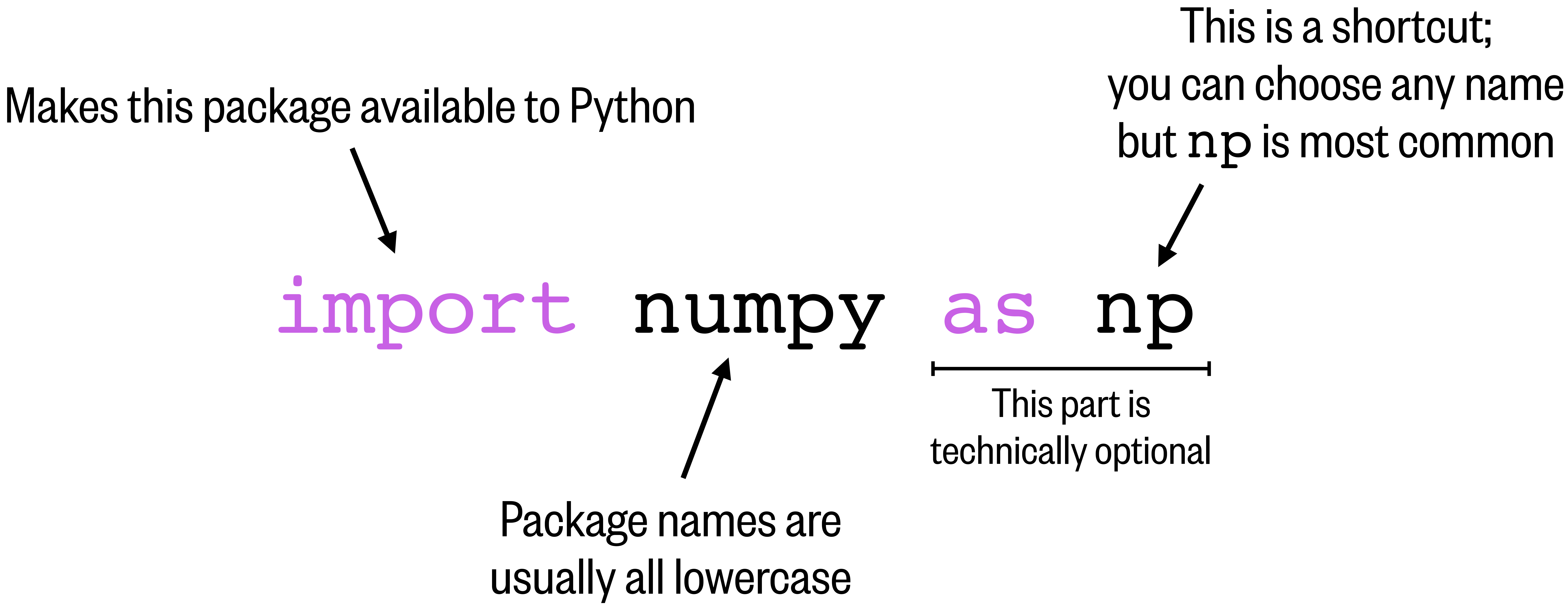
What we’ll cover in this lesson

1. Multi-dimensional arrays
2. More array functions
3. Dictionaries
4. Datetime

What we’ll cover in this lesson

1. **Multi-dimensional arrays**
2. More array functions
3. Dictionaries
4. Datetime

Loading NumPy ("Numeric Python")



```
import numpy as np
```

Makes this package available to Python

This is a shortcut; you can choose any name but `np` is most common

Package names are usually all lowercase

This part is technically optional

The NumPy array (`ndarray`)

"N-dimensional array" (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

```
np.array([5,6,7,8])
```

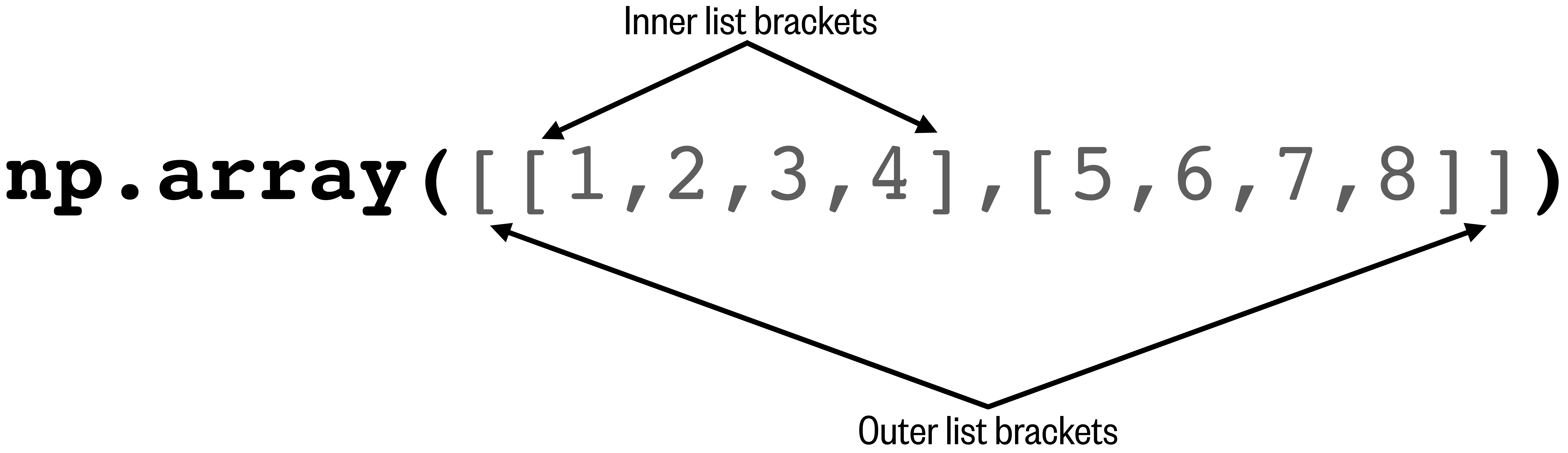
The NumPy array (`ndarray`)

“N-dimensional array” (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

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

The NumPy array (`ndarray`)

"N-dimensional array" (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

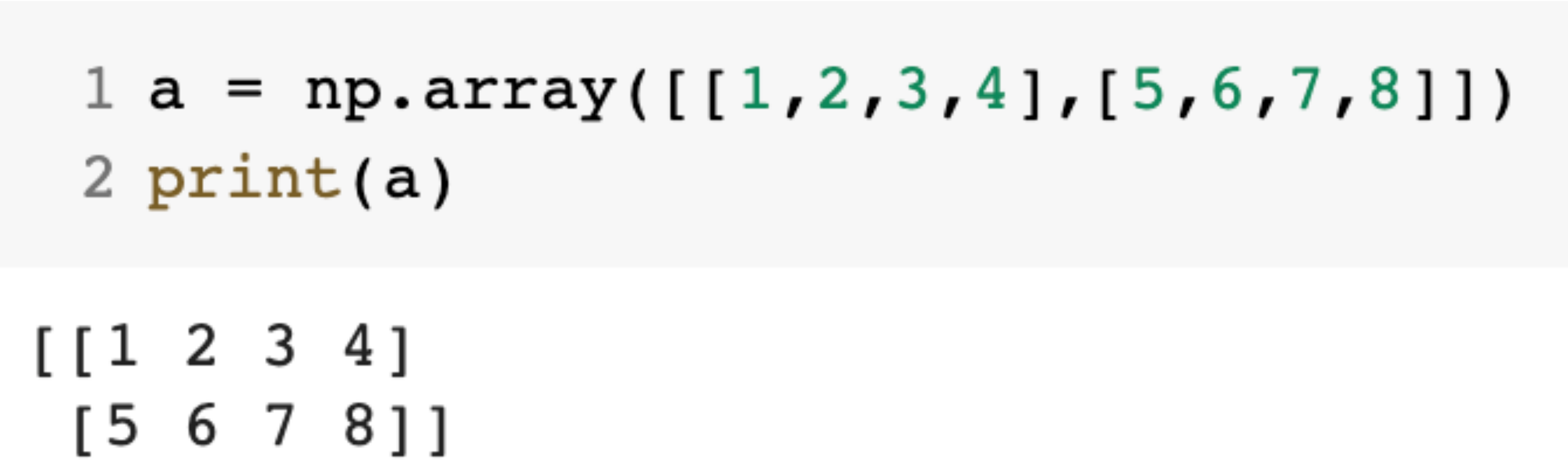


Inner list brackets

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

Outer list brackets

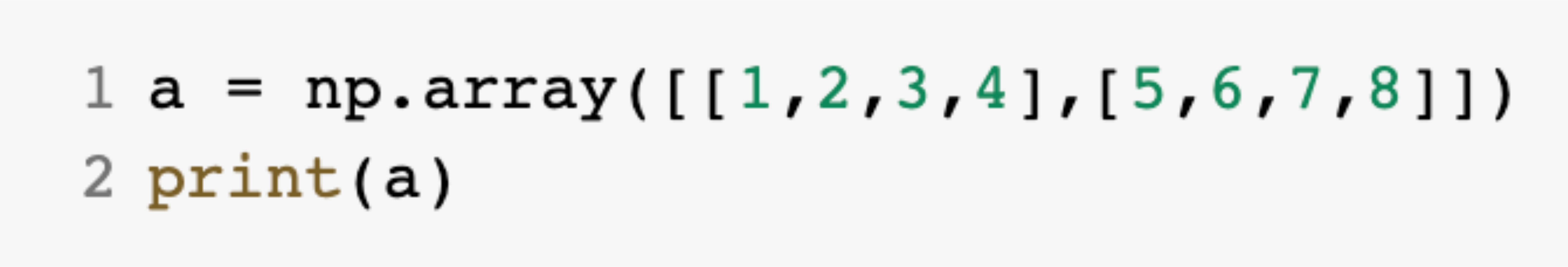
Slicing and indexing N-dimensional arrays



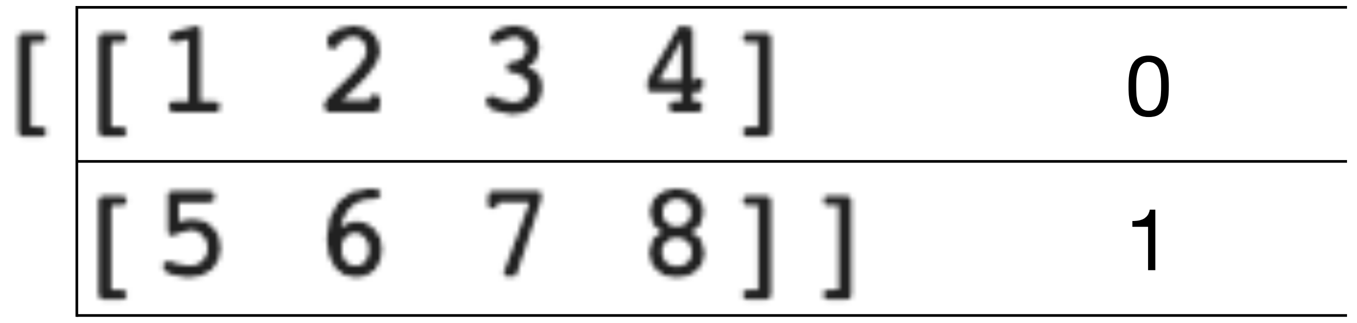
```python
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

Slicing and indexing N-dimensional arrays

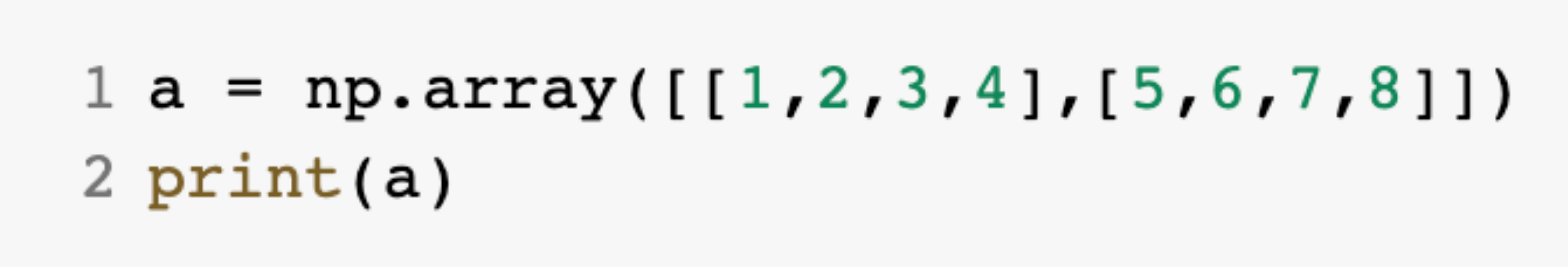


```python
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```

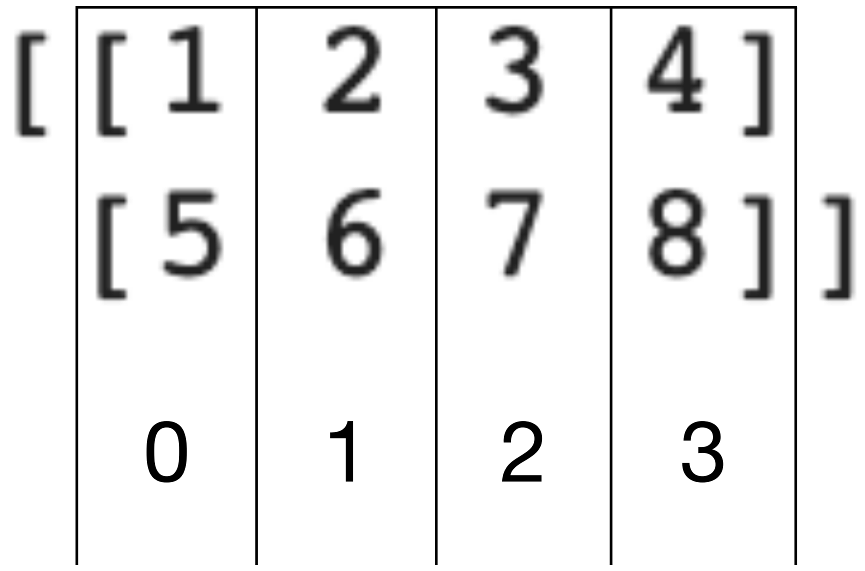


```
[[1 2 3 4]
 [5 6 7 8]]
```

Slicing and indexing N-dimensional arrays

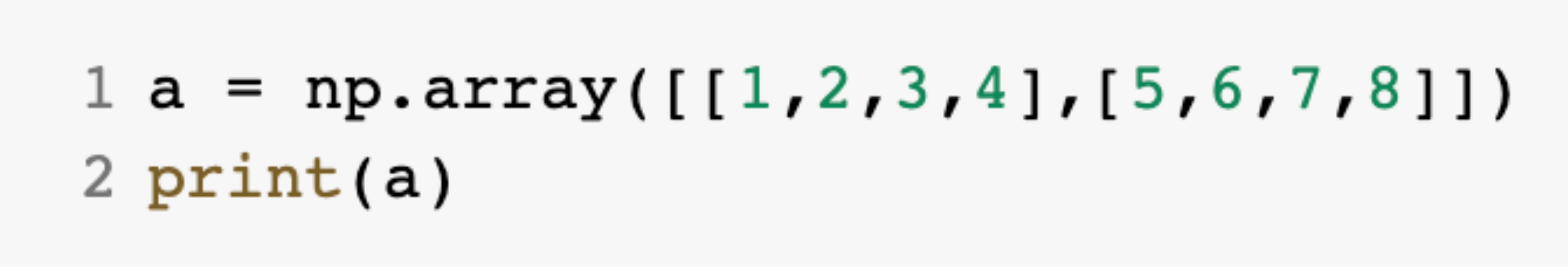


```python
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```



```
[[1 2 3 4]
 [5 6 7 8]]
```

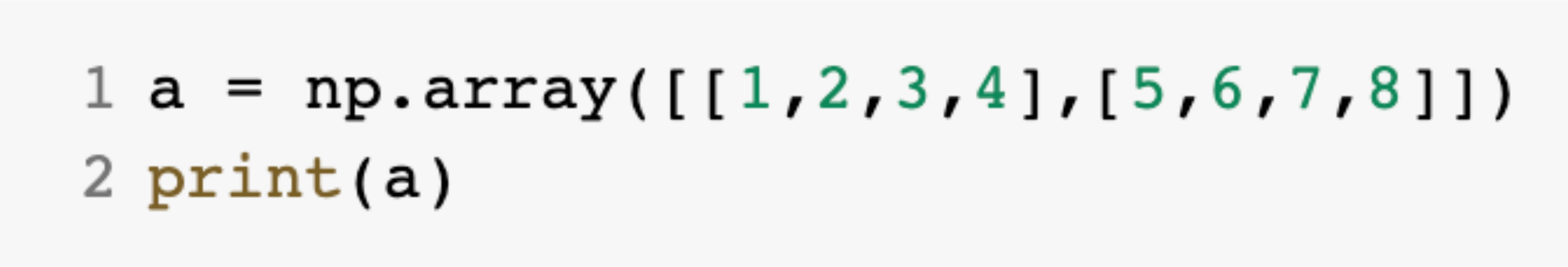
Slicing and indexing N-dimensional arrays



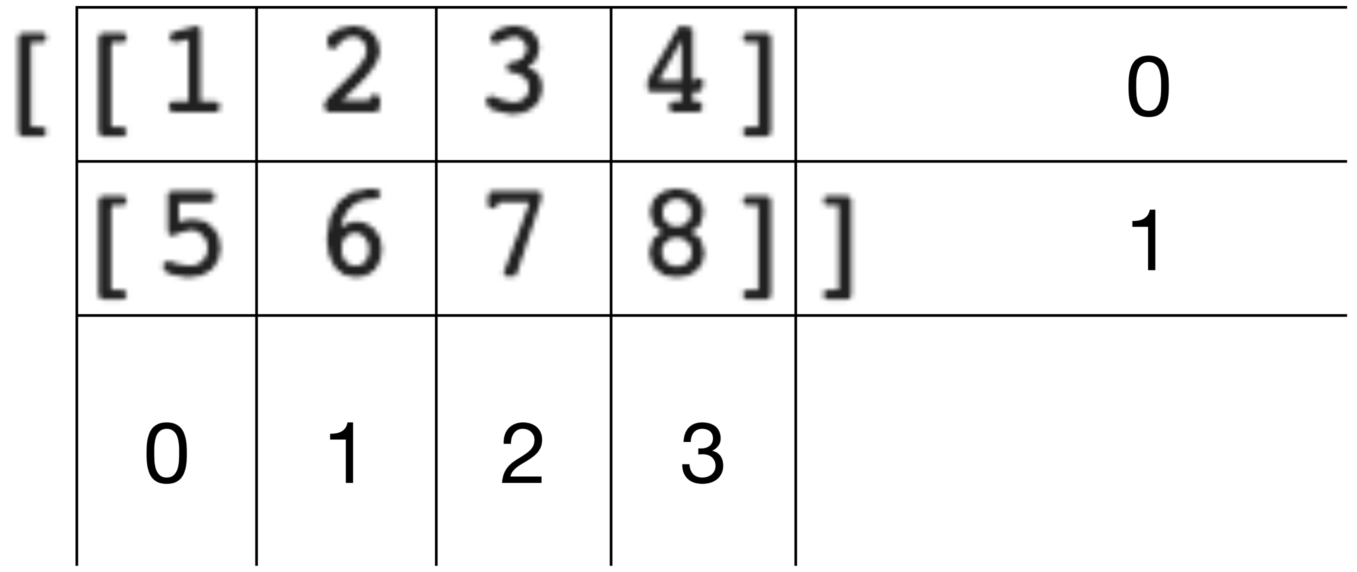
```python
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```

[	[1	2	3	4]	0
	[5	6	7	8]]	1
	0	1	2	3	

Slicing and indexing N-dimensional arrays

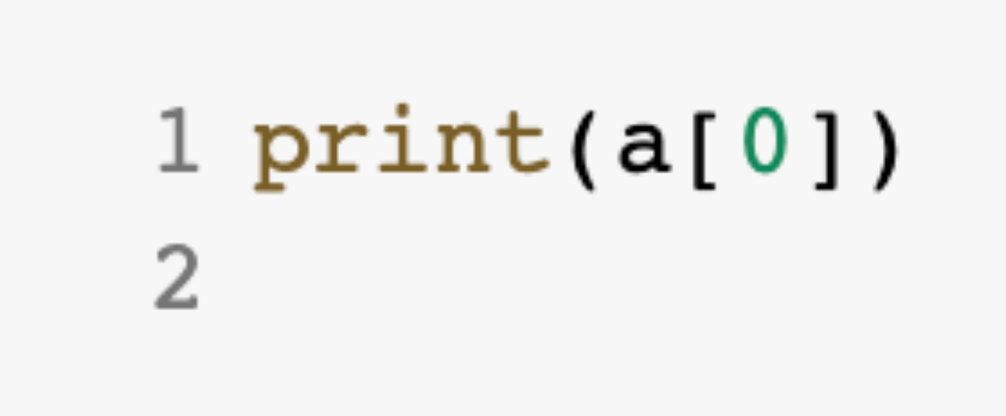


```
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```

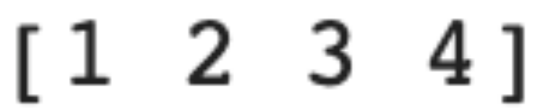


[[1	2	3	4]	0
[5	6	7	8]]	1
0	1	2	3	

A single number index gives the **items** of the outer list.

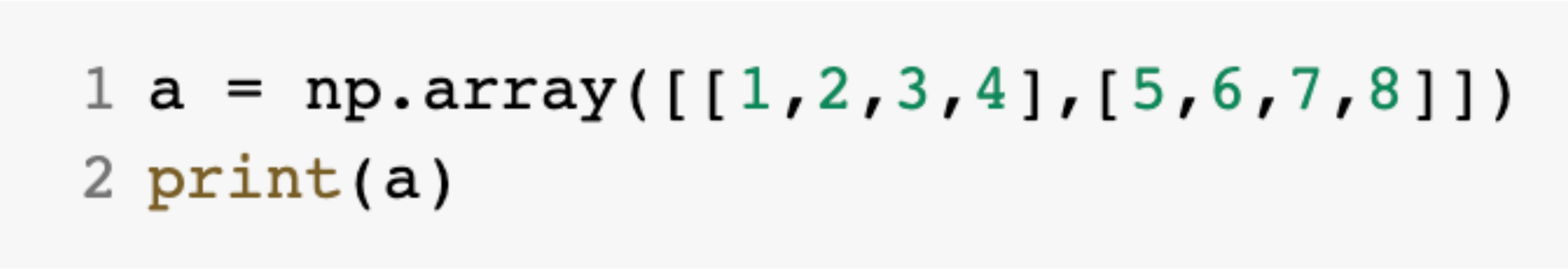


```
print(a[0])

```

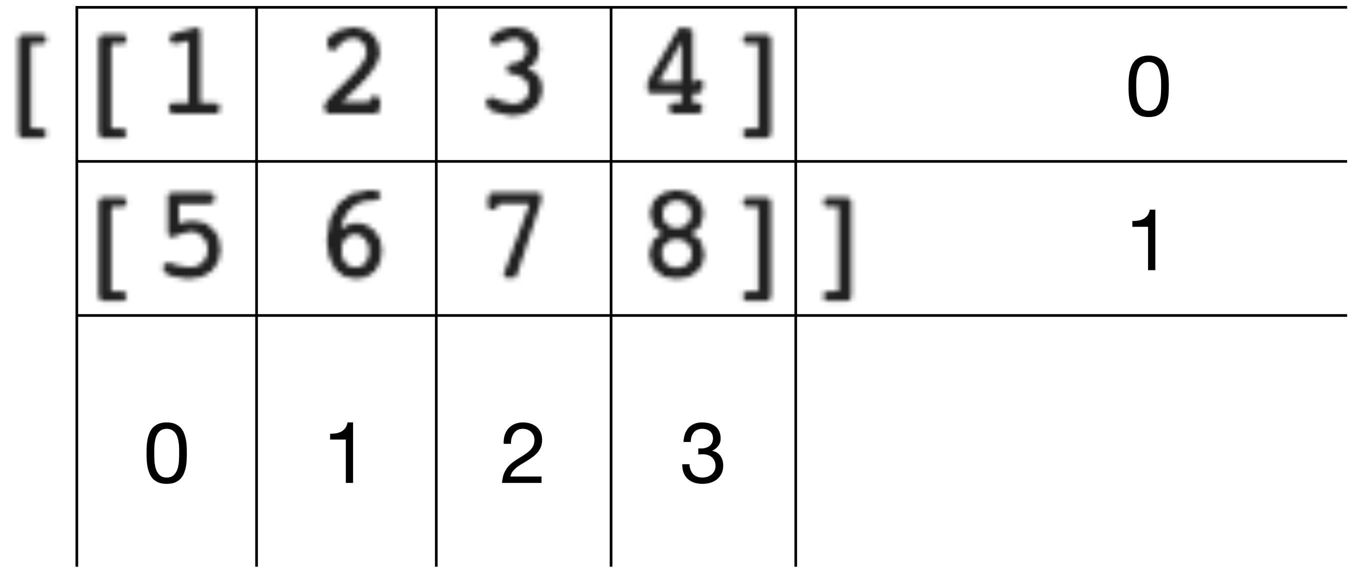


```
[1 2 3 4]
```

Slicing and indexing N-dimensional arrays

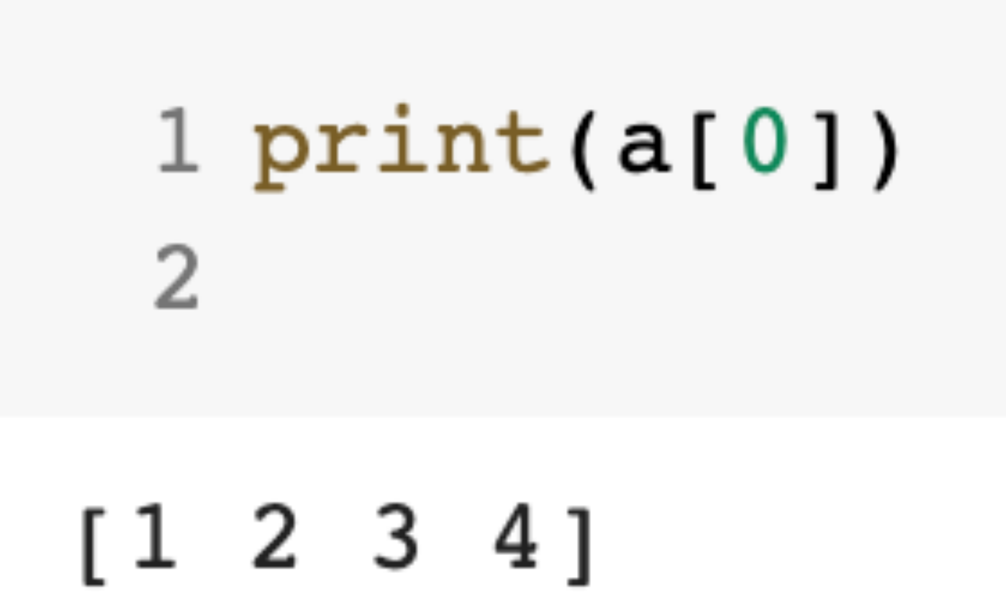


```python
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```



```
[[1 2 3 4]
 [5 6 7 8]]
```

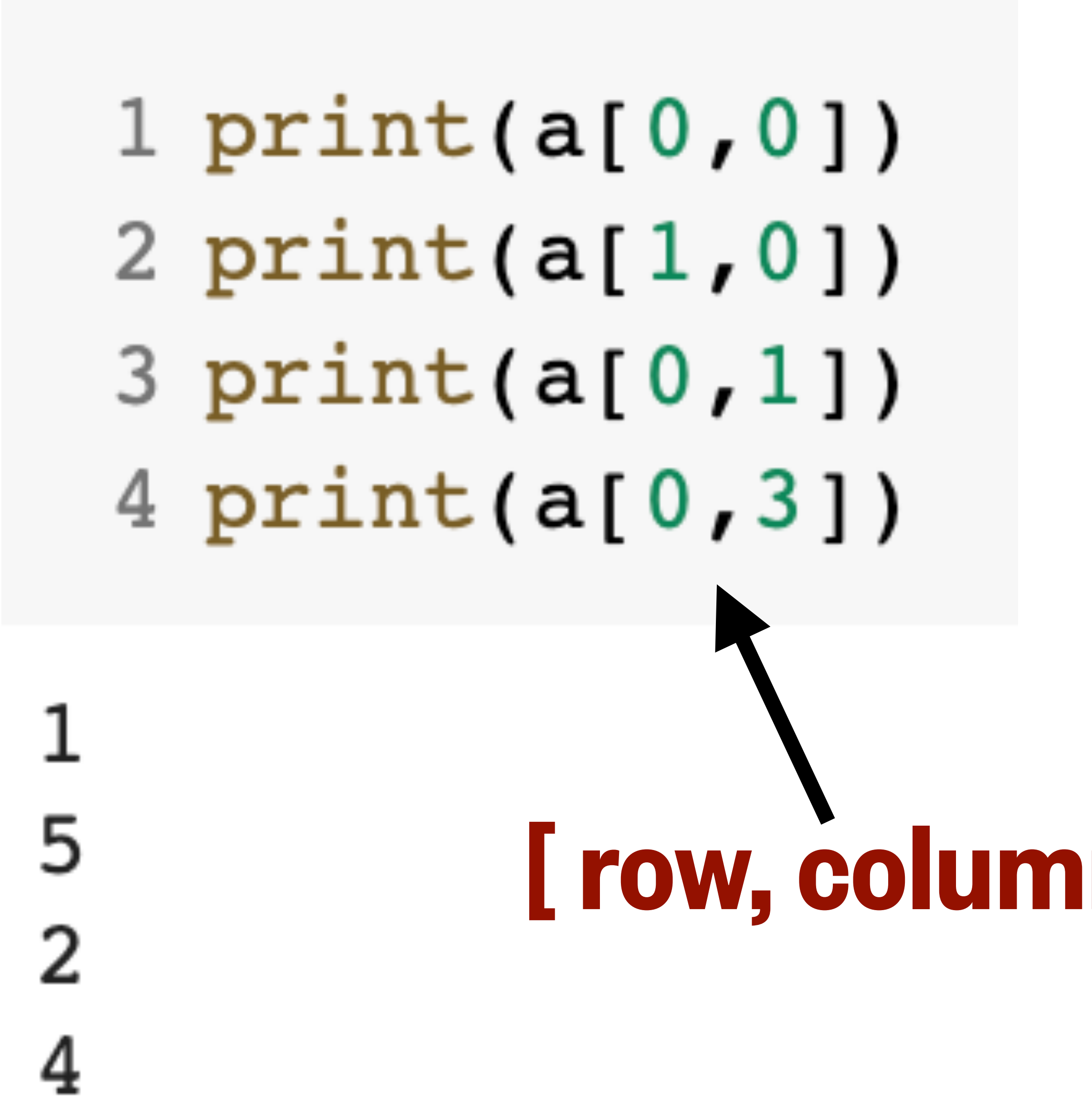
A single number index gives the **items** of the outer list.



```python
print(a[0])
```

```
[1 2 3 4]
```

Select specific items using the row and column index values

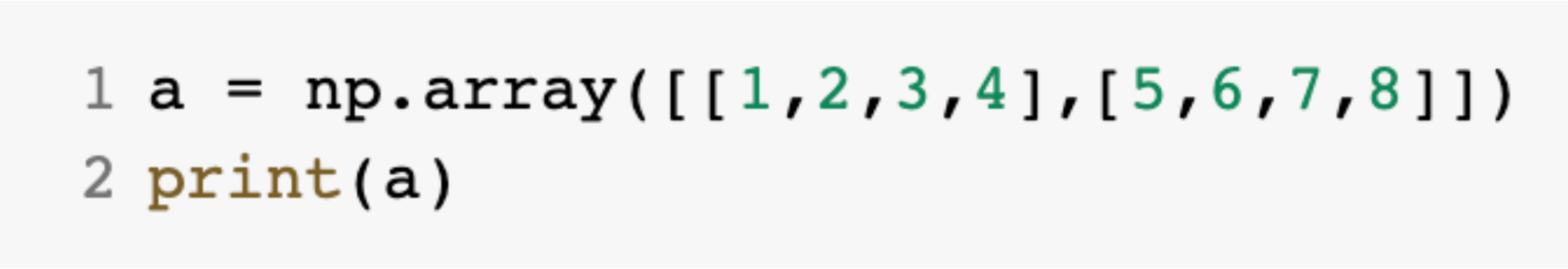


```python
print(a[0,0])
print(a[1,0])
print(a[0,1])
print(a[0,3])
```

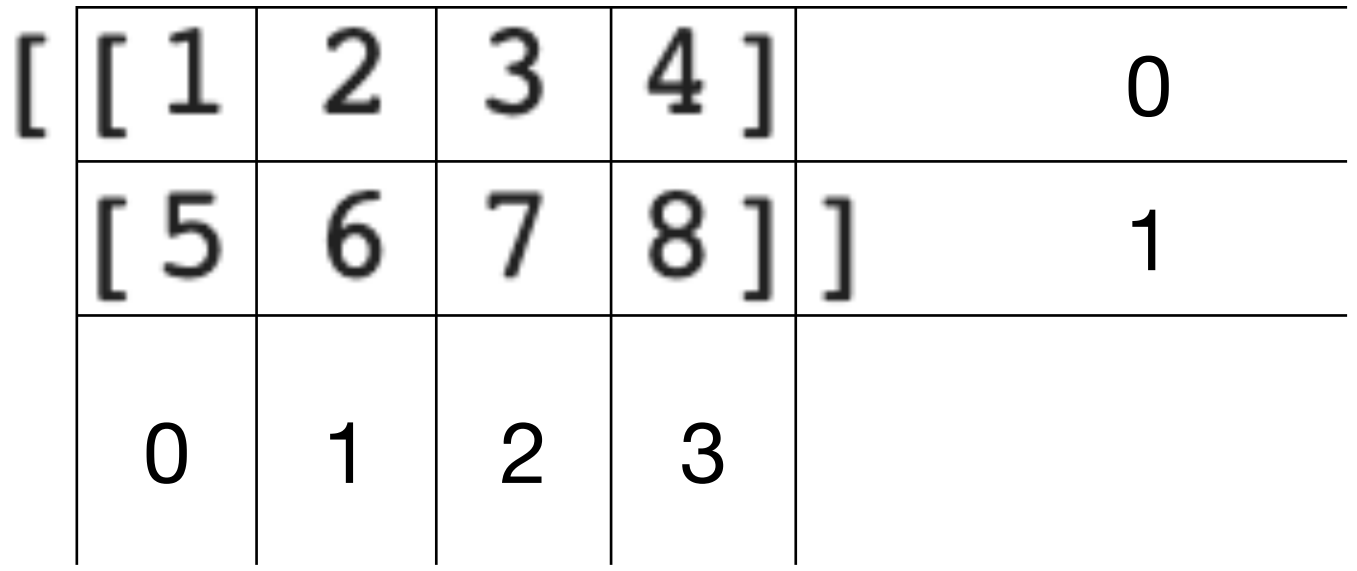
```
1
5
2
4
```

[row, column]

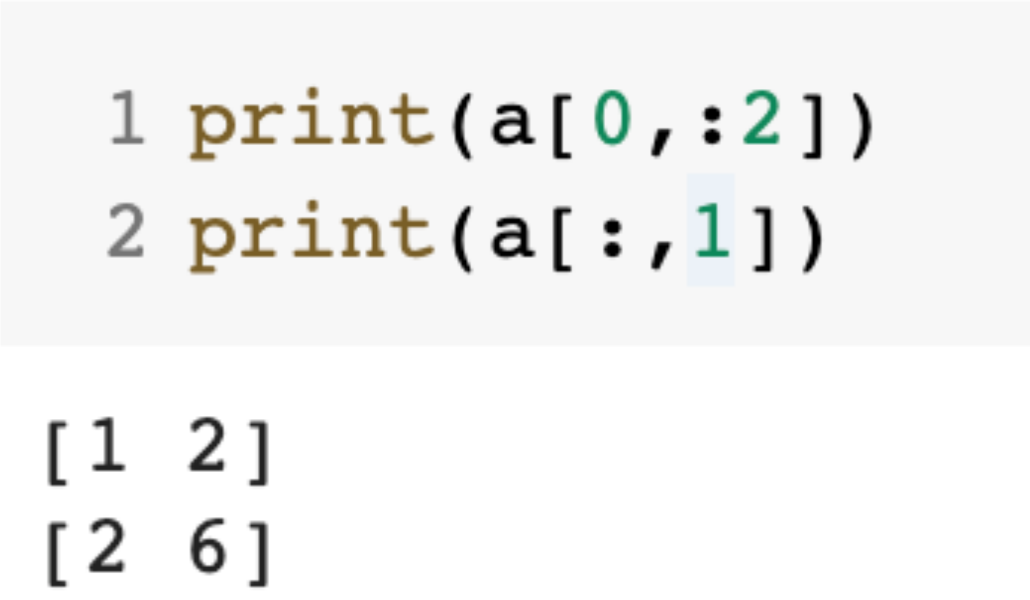
Slicing and indexing N-dimensional arrays



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



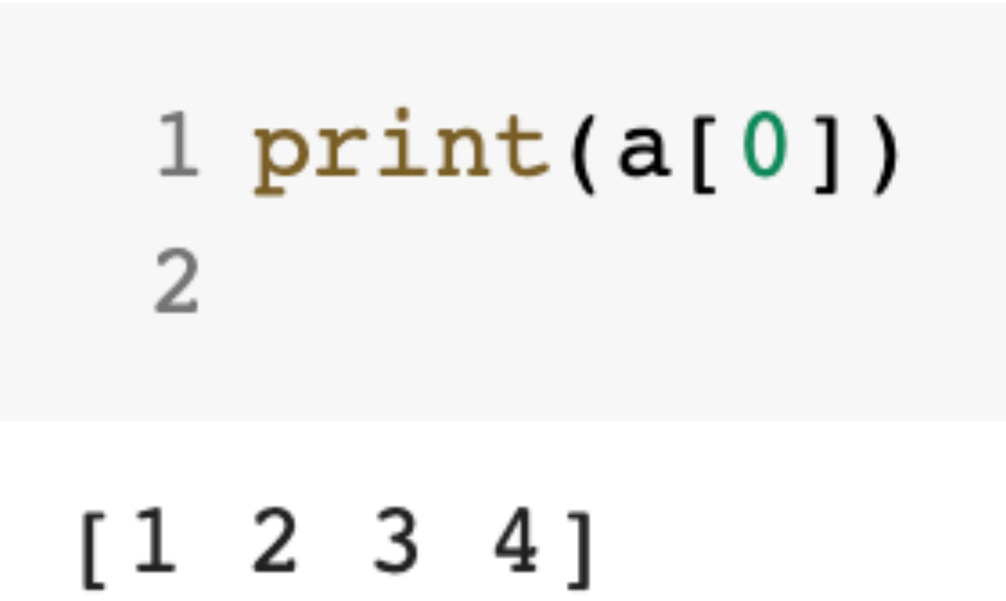
Arrays can also be sliced with rows and columns



```
print(a[0,:2])
print(a[:,1])
```

```
[1 2]
[2 6]
```

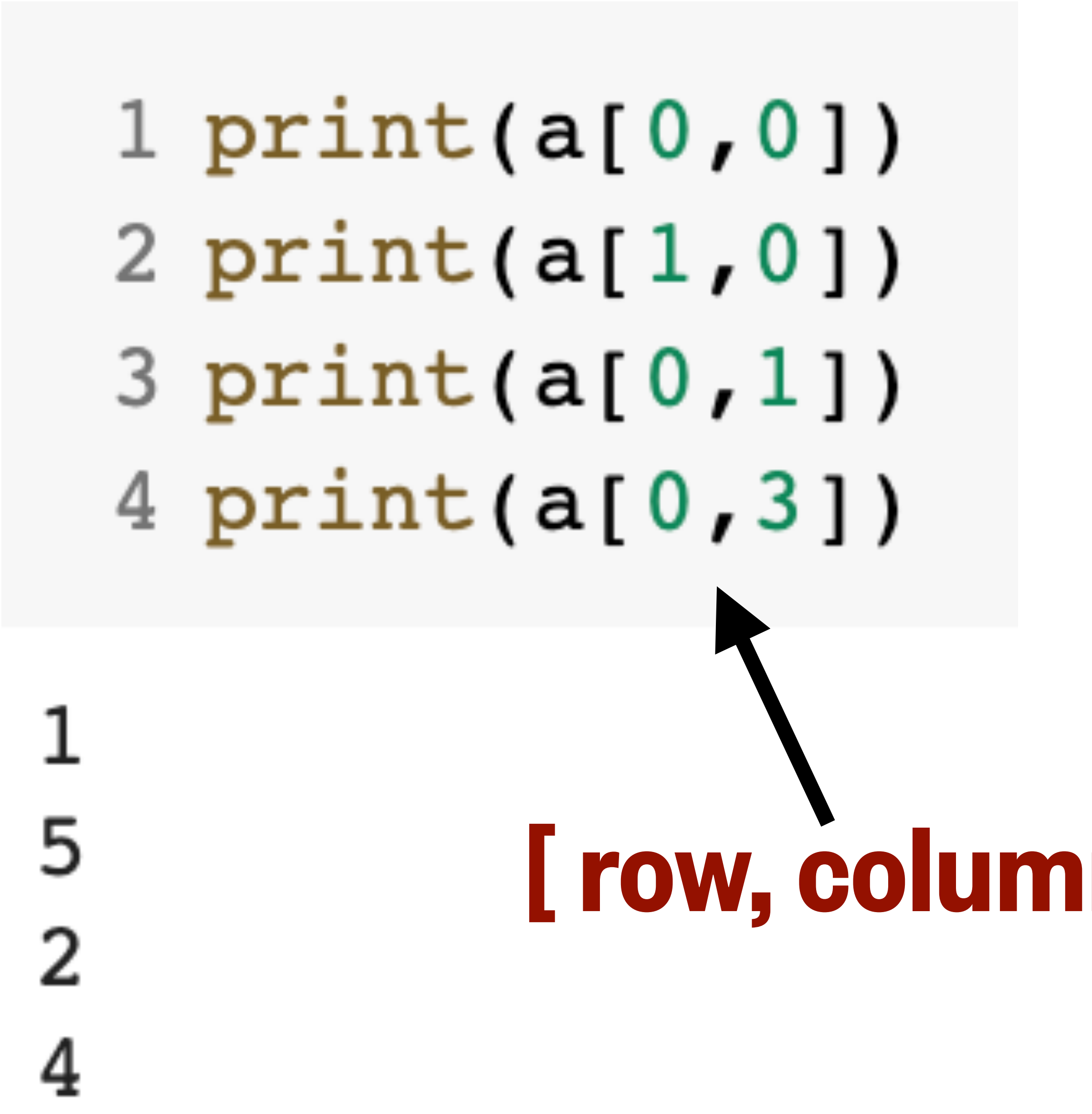
A single number index gives the **items** of the outer list.



```
print(a[0])
```

```
[1 2 3 4]
```

Select specific items using the row and column index values



```
print(a[0,0])
print(a[1,0])
print(a[0,1])
print(a[0,3])
```

```
1
5
2
4
```

[row, column]

The NumPy array (`ndarray`)

“N-dimensional array” (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

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

The NumPy array (`ndarray`)

“N-dimensional array” (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

```
np.array([[[1,2,3,4],[5,6,7,8]],
 [[9,10,11,12],[13,14,15,16]]])
```

The NumPy array (`ndarray`)

"N-dimensional array" (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

```
np.array([[[1,2,3,4],[5,6,7,8]],
 [[9,10,11,12],[13,14,15,16]]])
```

Outer brackets

Middle brackets

Inner brackets

The NumPy array (`ndarray`)

"N-dimensional array" (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

```
np.array([[[1,2,3,4],[5,6,7,8]],
 [[9,10,11,12],[13,14,15,16]]])
```

Outer brackets

Middle brackets

Inner brackets

The NumPy array (`ndarray`)

“N-dimensional array” (e.g. 1-D, 2-D, 3-D, 4-D, etc.)

```
np.array([[[1,2,3,4],[5,6,7,8]],
 [[9,10,11,12],[13,14,15,16]]])
```

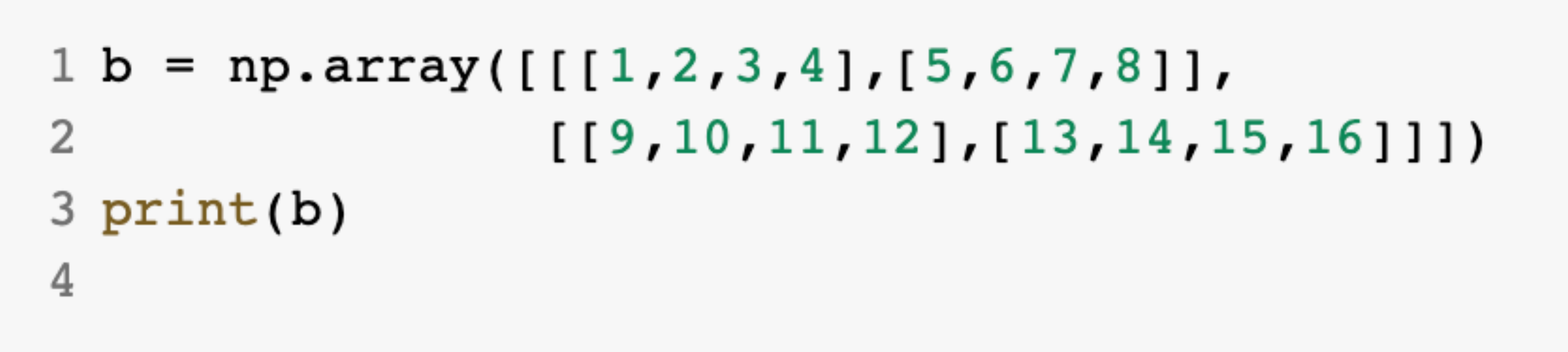
Outer brackets

Middle brackets

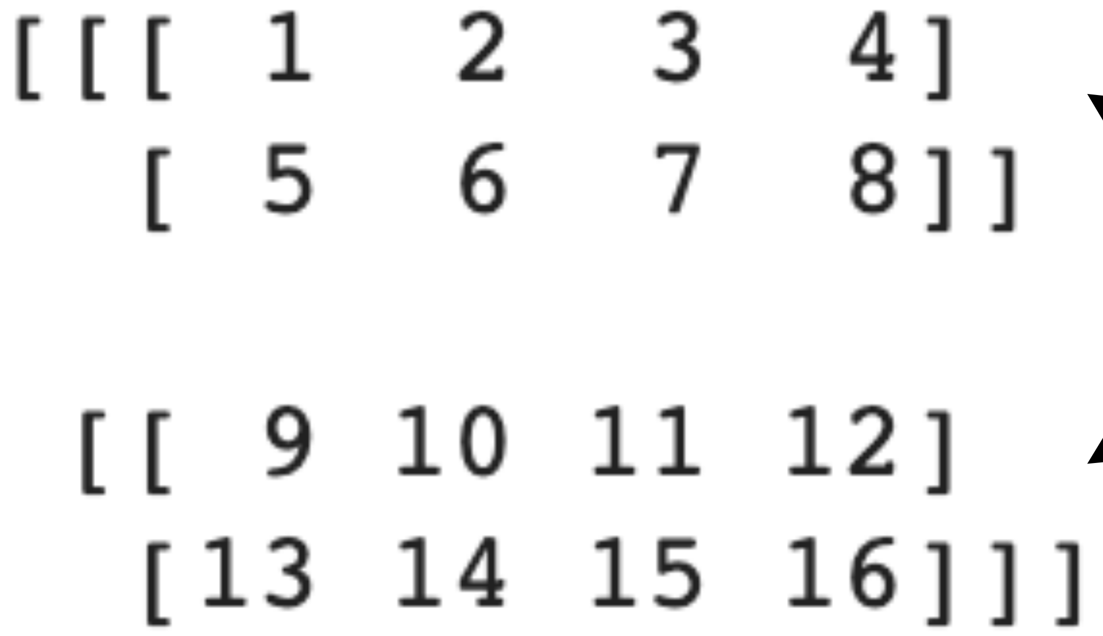
Inner brackets

The NumPy array (`ndarray`)

"N-dimensional array" (e.g. 1-D, 2-D, 3-D, 4-D, etc.)



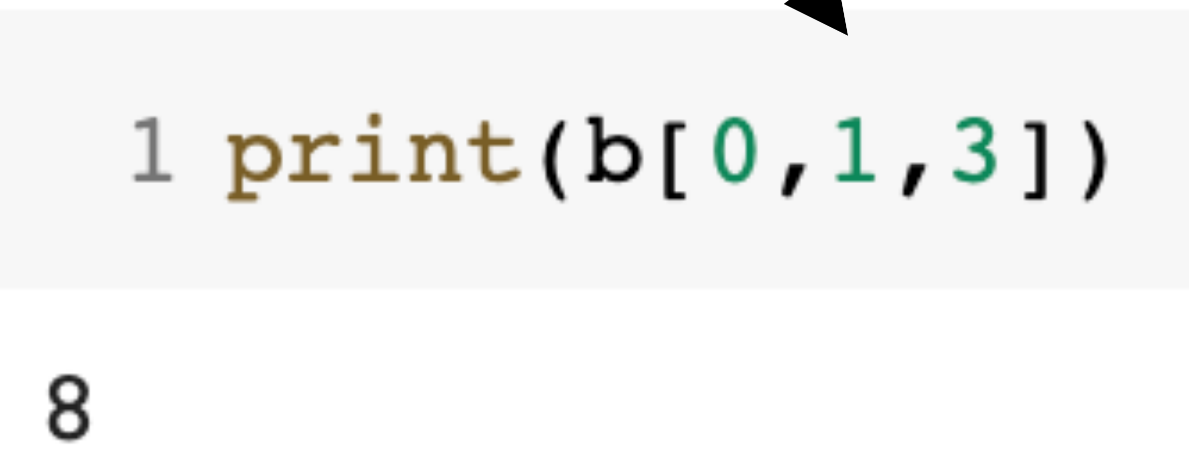
```python
b = np.array([[[1,2,3,4],[5,6,7,8]],
              [[9,10,11,12],[13,14,15,16]]])
print(b)

```



```
[[[ 1  2  3  4]
  [ 5  6  7  8]]

 [[ 9 10 11 12]
  [13 14 15 16]]]
```

2 layers with 2 rows and 4 columns each

[layer, row, column]



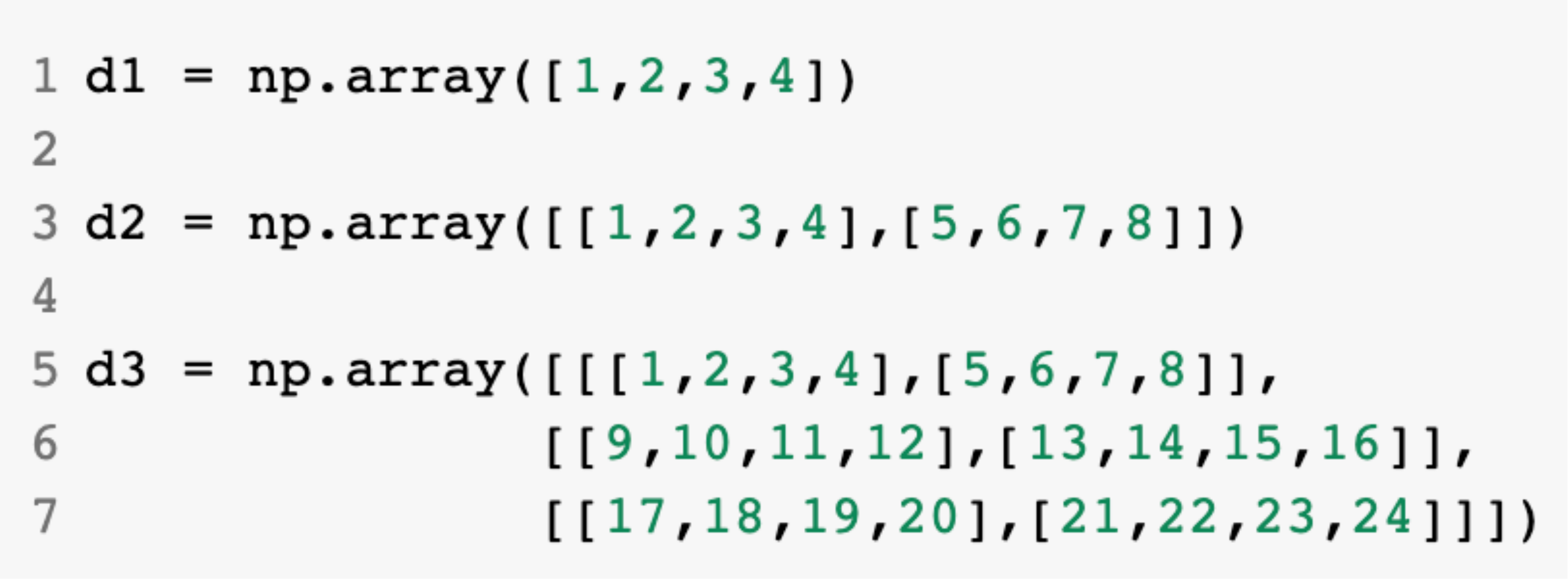
```python
print(b[0,1,3])
```

```
8
```

What we'll cover in this lesson

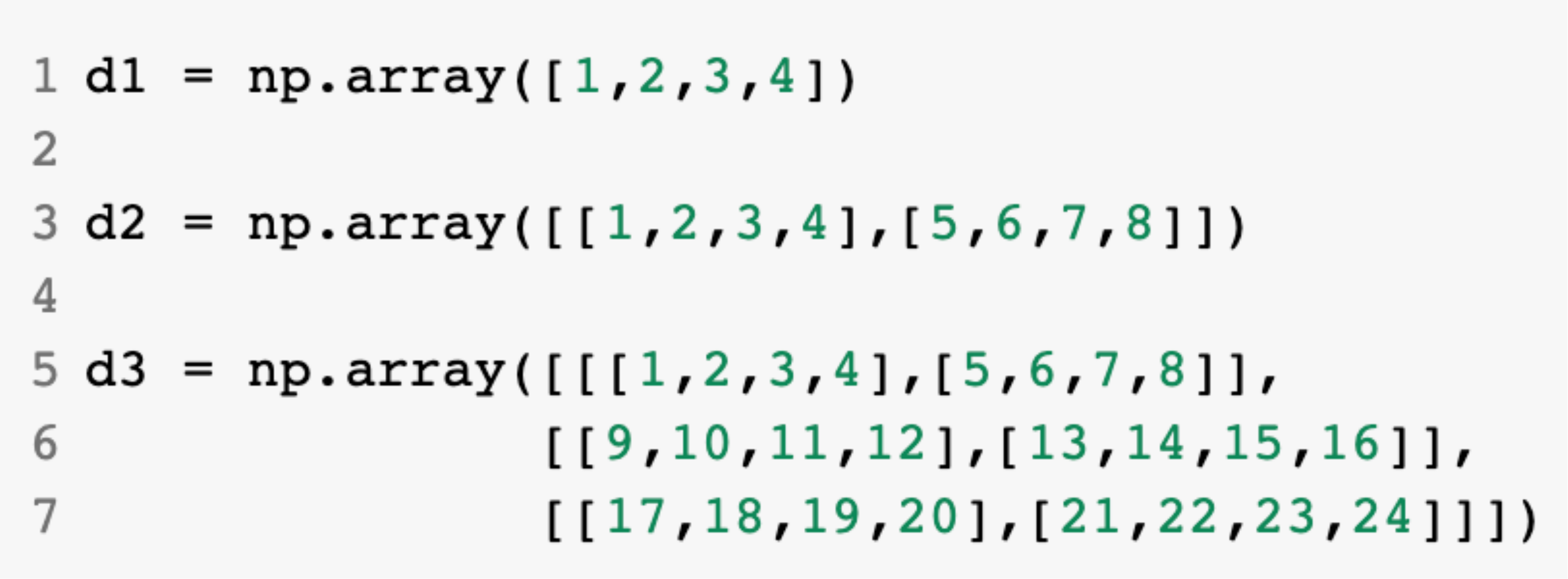
1. Multi-dimensional arrays
2. **More array functions**
3. Dictionaries
4. Datetime

Multi-dimensional NumPy arrays have more than just a length



```python
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
              [[9,10,11,12],[13,14,15,16]],
              [[17,18,19,20],[21,22,23,24]]])
```

Multi-dimensional NumPy arrays have more than just a length

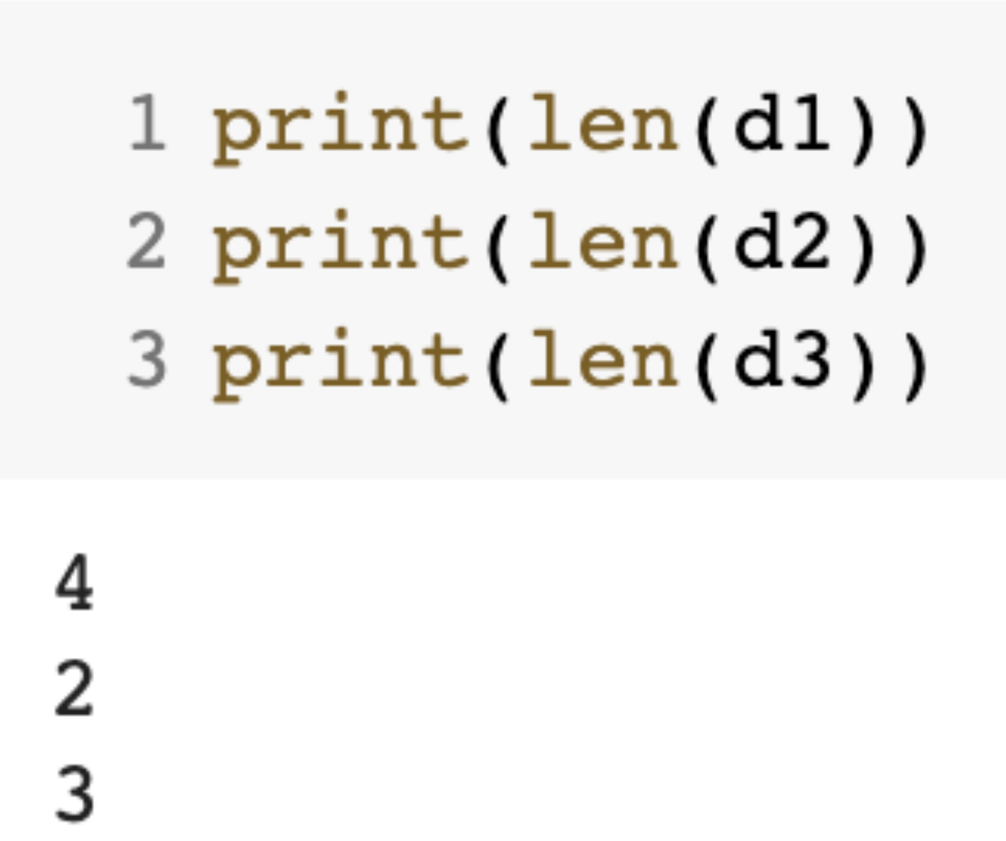


```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

len( )

Gives the number of items in the outer list dimension

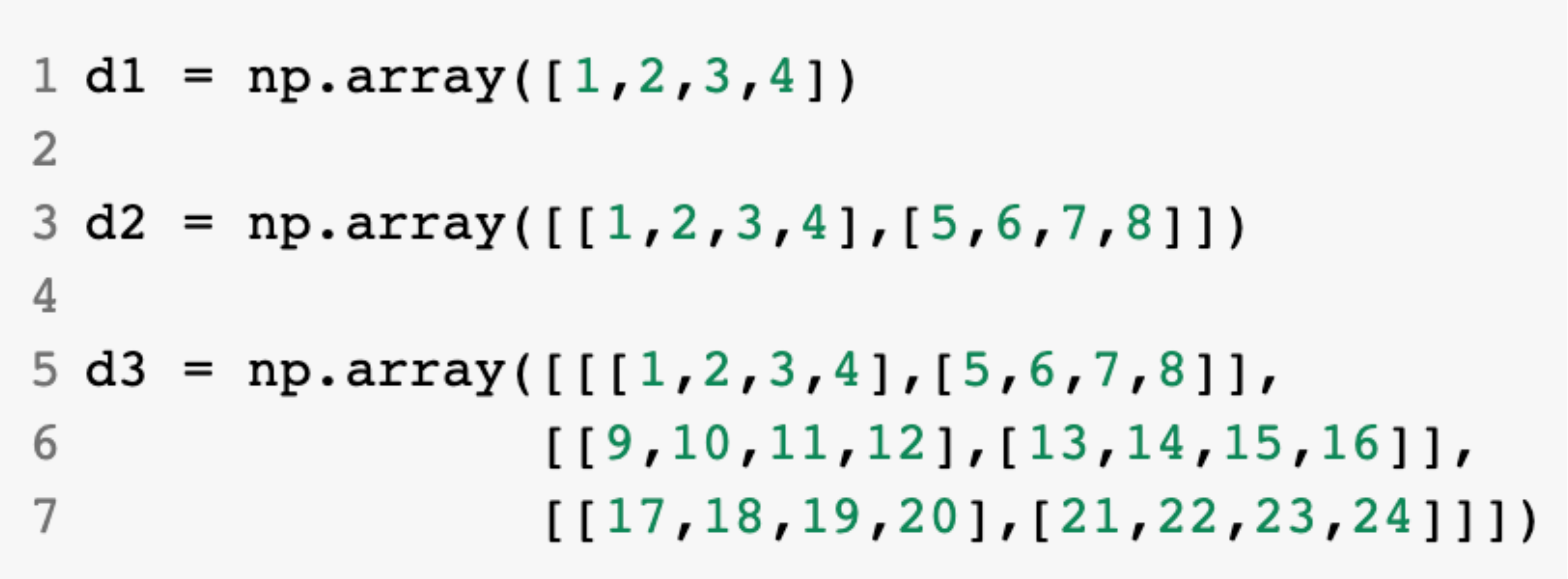
Example



```
print(len(d1))
print(len(d2))
print(len(d3))
```

```
4
2
3
```

Multi-dimensional NumPy arrays have more than just a length



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

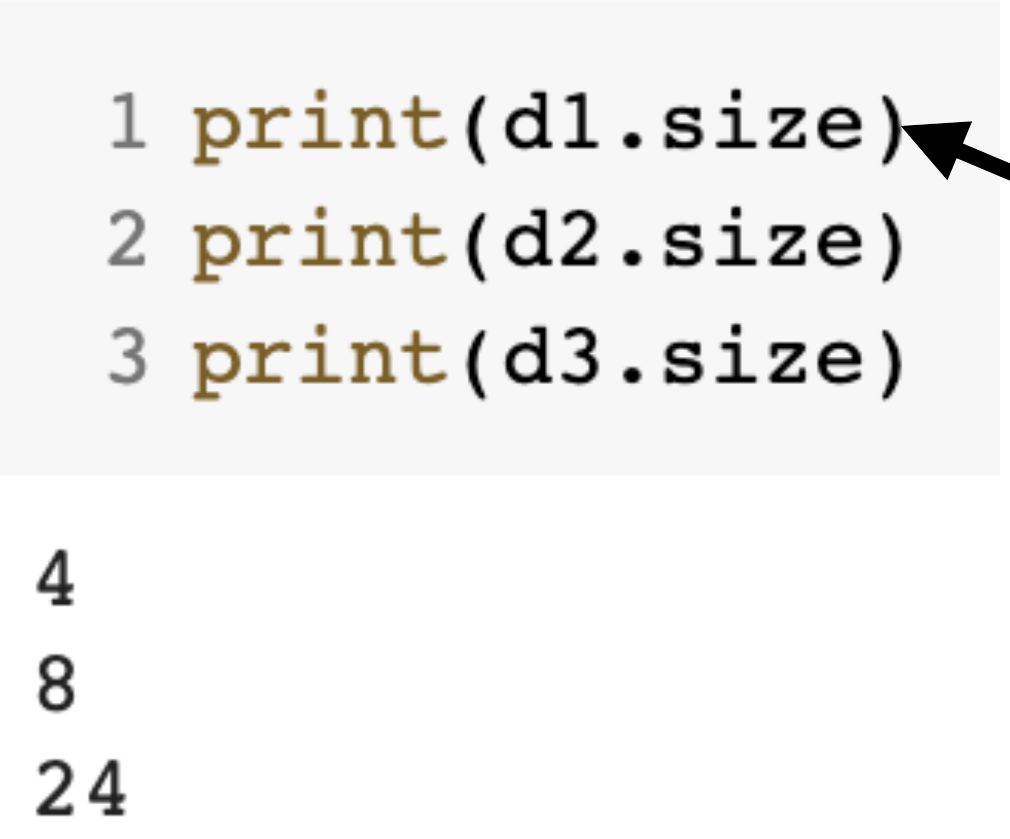
len( )

Gives the number of items in the outer list dimension

size

Gives the total number of items in the array

Example

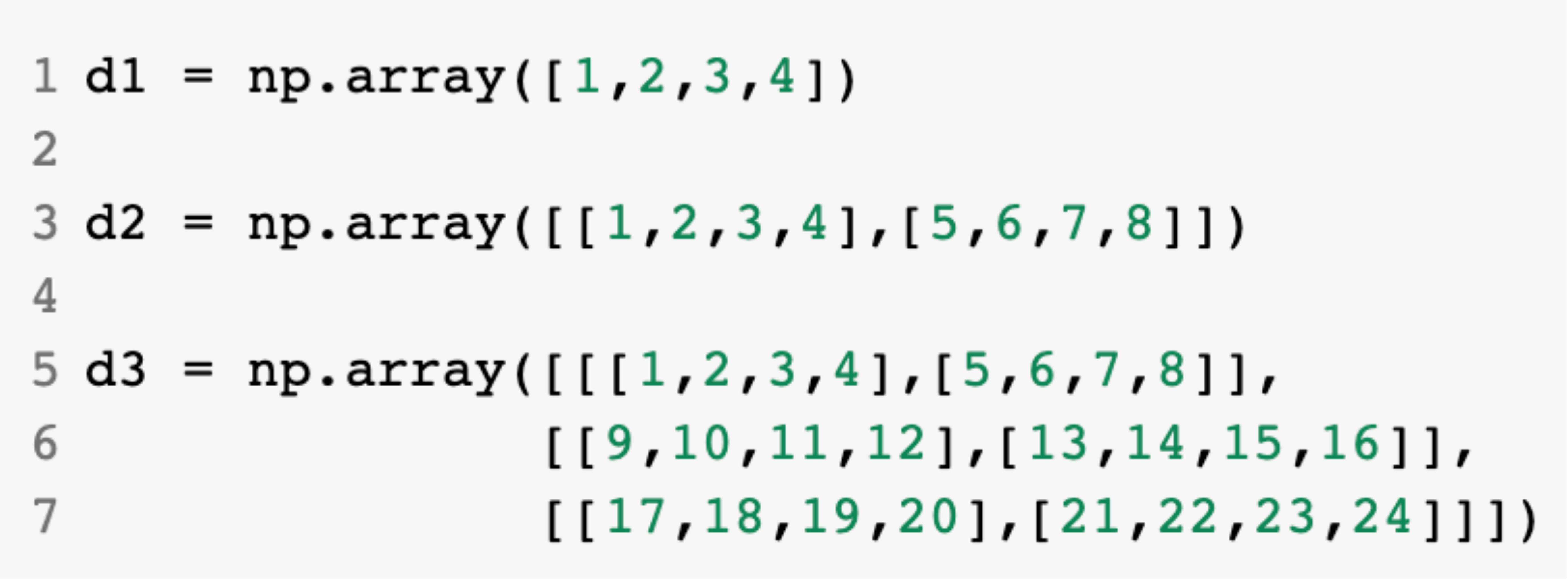


```
print(d1.size)
print(d2.size)
print(d3.size)
```

```
4
8
24
```

Notice there are no parentheses at the end of this. This is because size is not a function, but an attribute of the array.

Multi-dimensional NumPy arrays have more than just a length



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

len( )

Gives the number of items in the outer list dimension

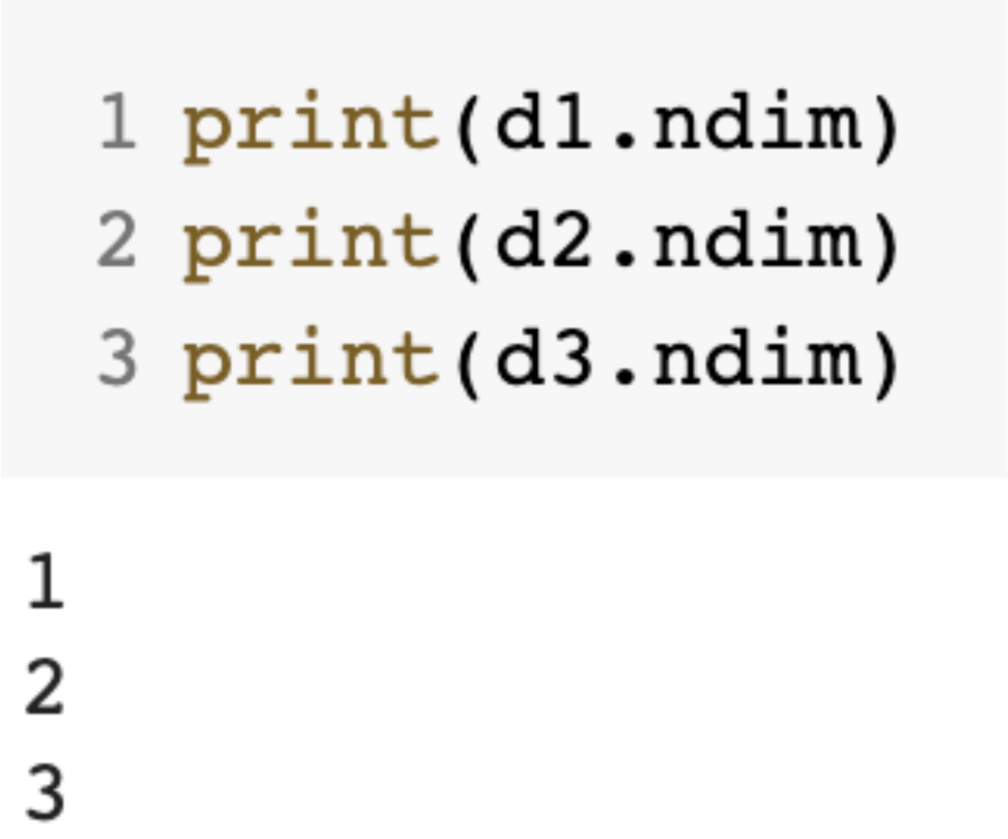
size

Gives the total number of items in the array

ndim

Gives the number of dimensions in an array

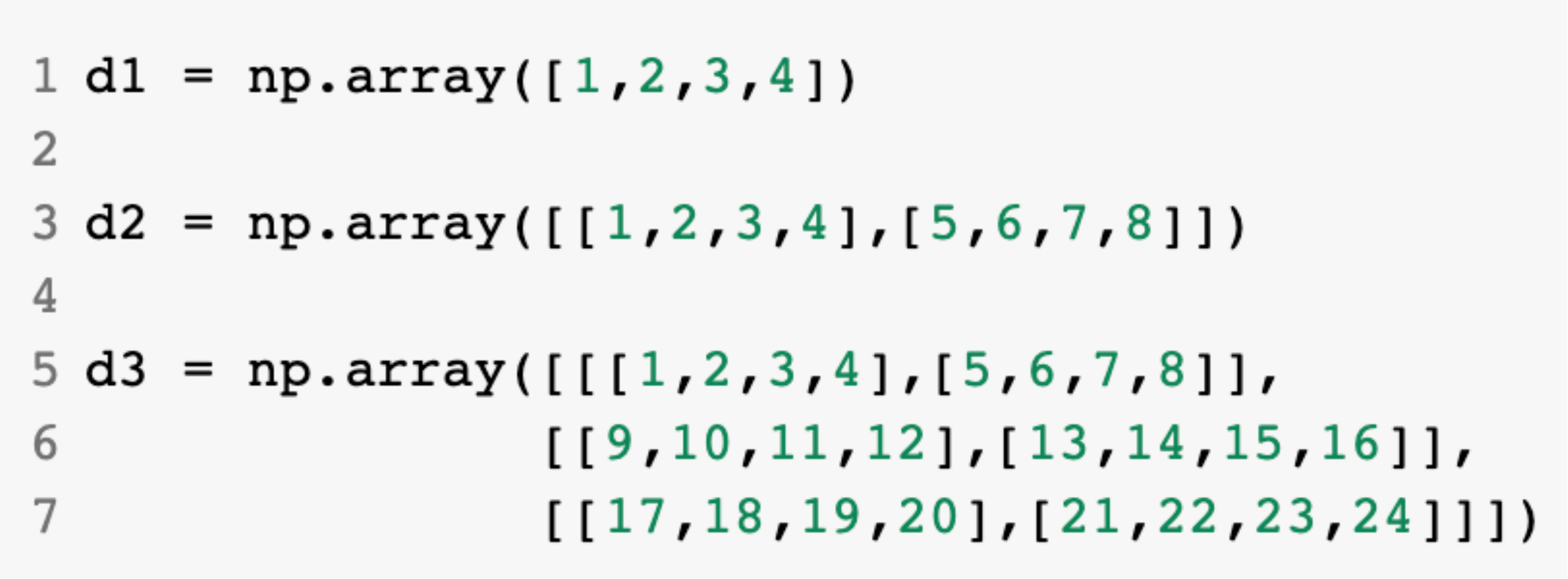
Example



```
print(d1.ndim)
print(d2.ndim)
print(d3.ndim)
```

```
1
2
3
```

Multi-dimensional NumPy arrays have more than just a length



```python
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
              [[9,10,11,12],[13,14,15,16]],
              [[17,18,19,20],[21,22,23,24]]])
```

len( )

Gives the number of items in the outer list dimension

size

Gives the total number of items in the array

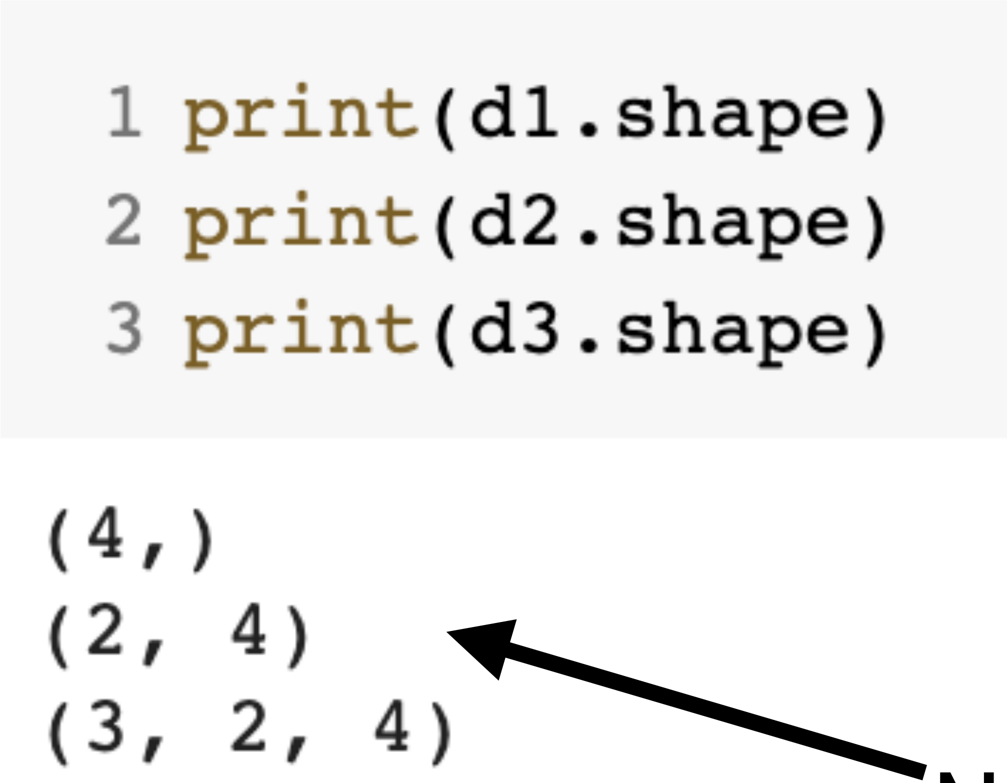
ndim

Gives the number of dimensions in an array

shape

Gives the number of items in each dimension of an array

Example

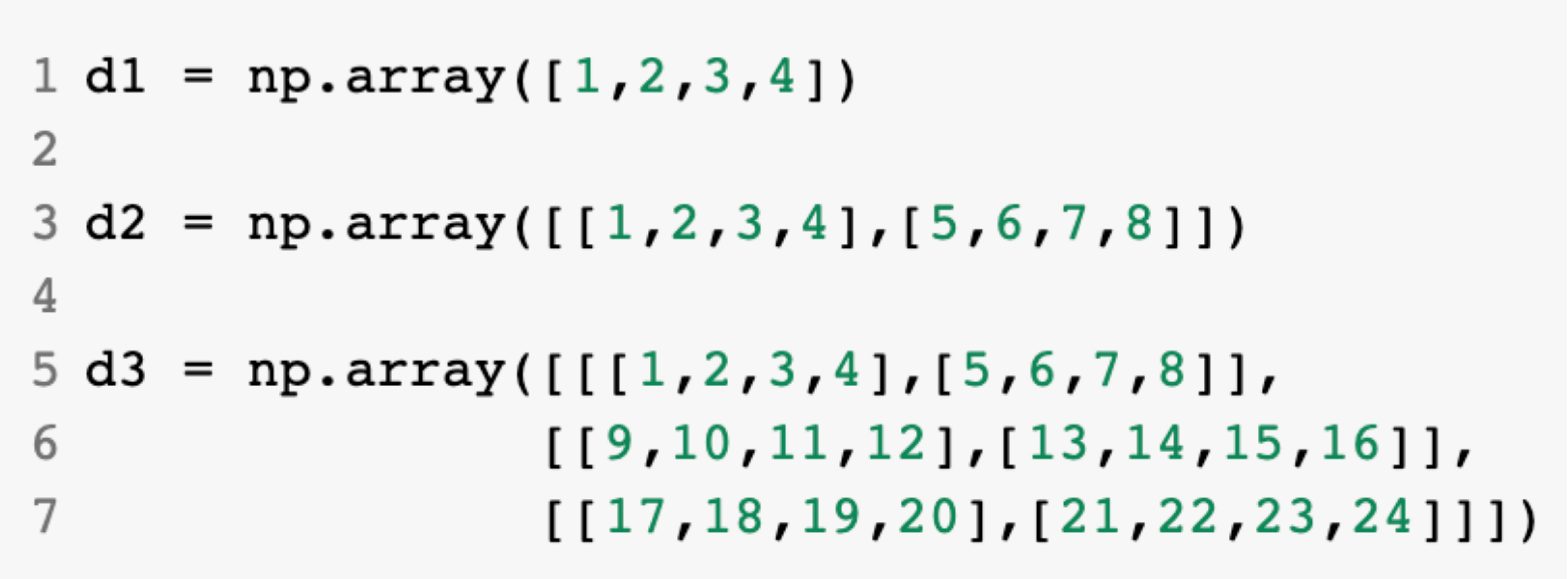


```python
print(d1.shape)
print(d2.shape)
print(d3.shape)
```

```
(4,)
(2, 4)
(3, 2, 4)
```

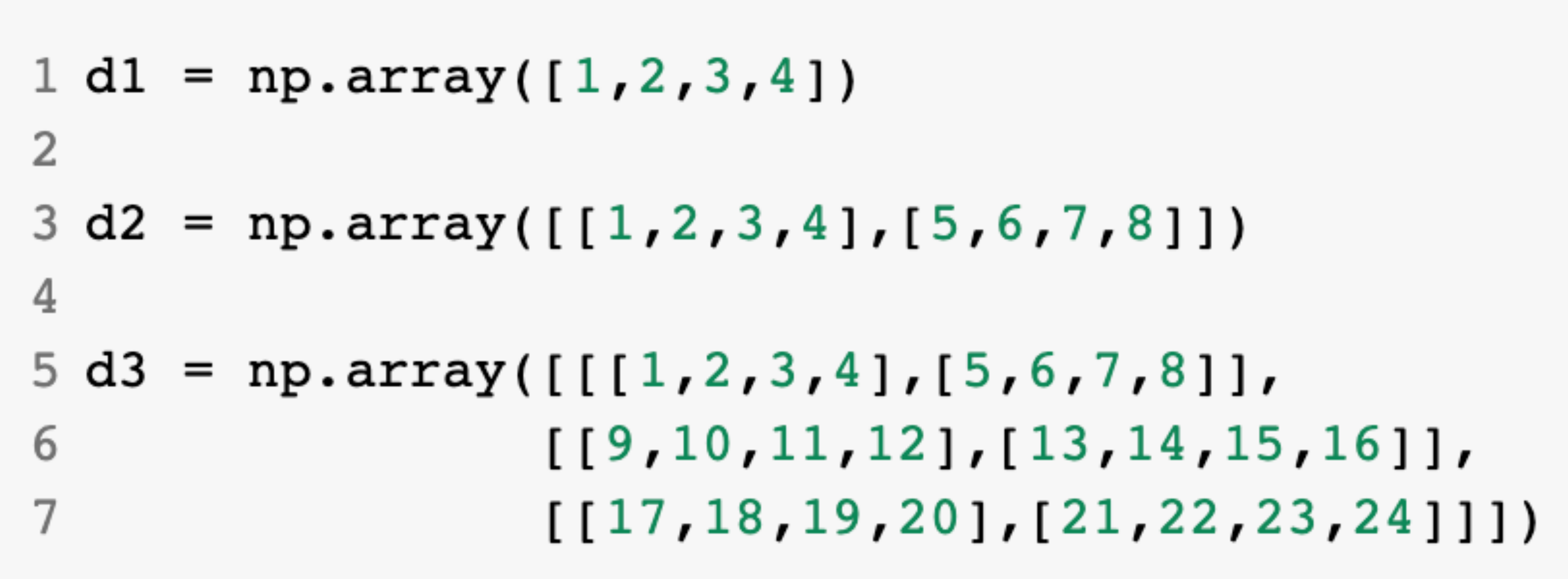
Notice these are given as tuples

You can change the shape of a NumPy array



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

You can change the shape of a NumPy array

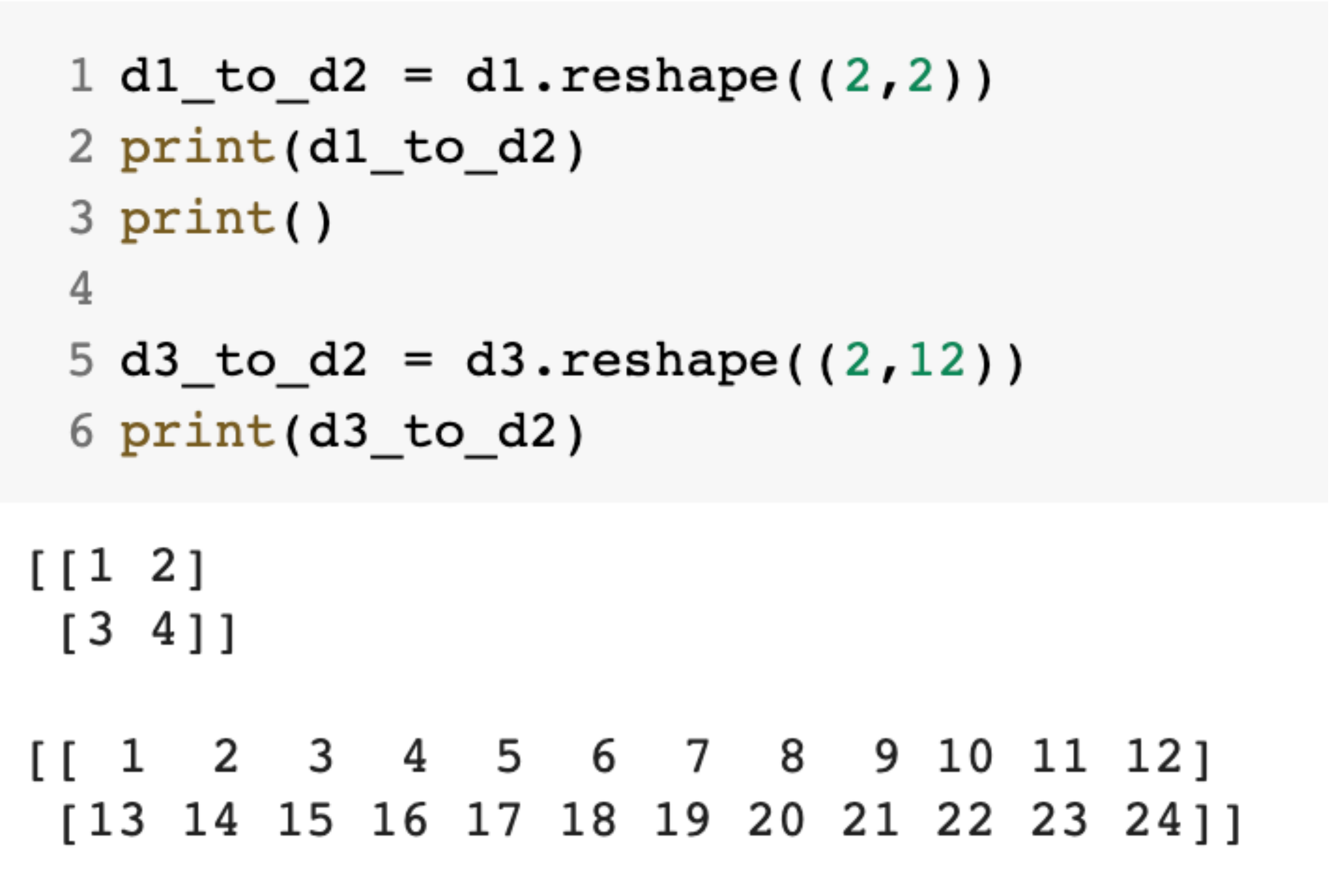


```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
              [[9,10,11,12],[13,14,15,16]],
              [[17,18,19,20],[21,22,23,24]]])
```

reshape( )

Changes the shape of the array into a given shape

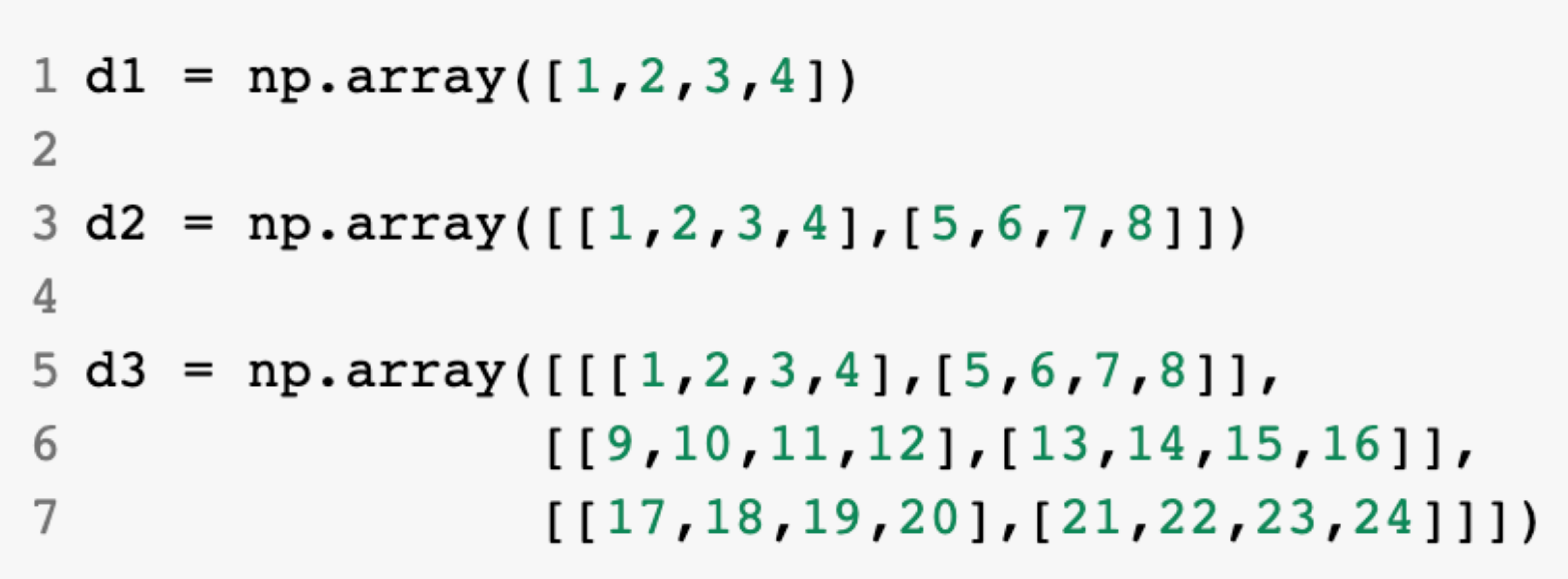
Example



```
d1_to_d2 = d1.reshape((2,2))
print(d1_to_d2)
print()

d3_to_d2 = d3.reshape((2,12))
print(d3_to_d2)
```

```
[[1 2]
 [3 4]]

[[ 1  2  3  4  5  6  7  8  9 10 11 12]
 [13 14 15 16 17 18 19 20 21 22 23 24]]
```

You can change the shape of a NumPy array



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

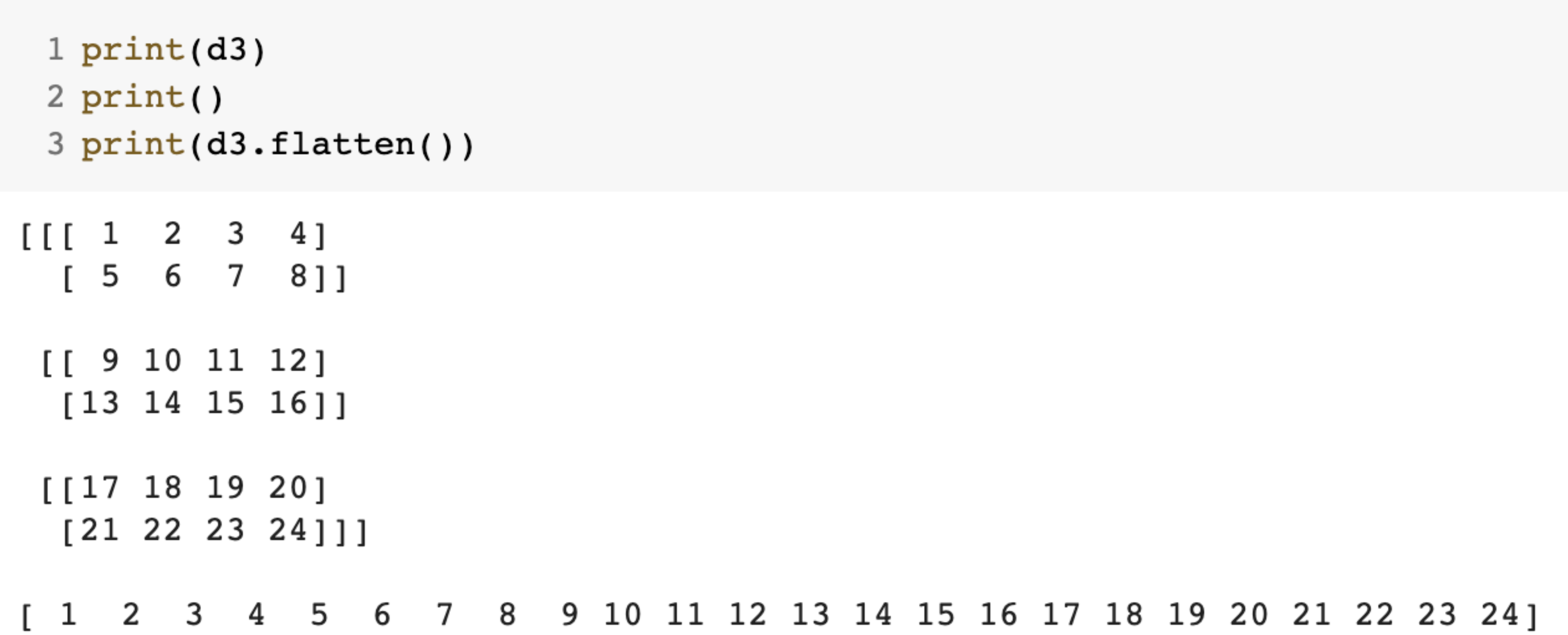
reshape( )

Changes the shape of the array into a given shape

flatten( )

Creates a copy of an array as a 1-D array

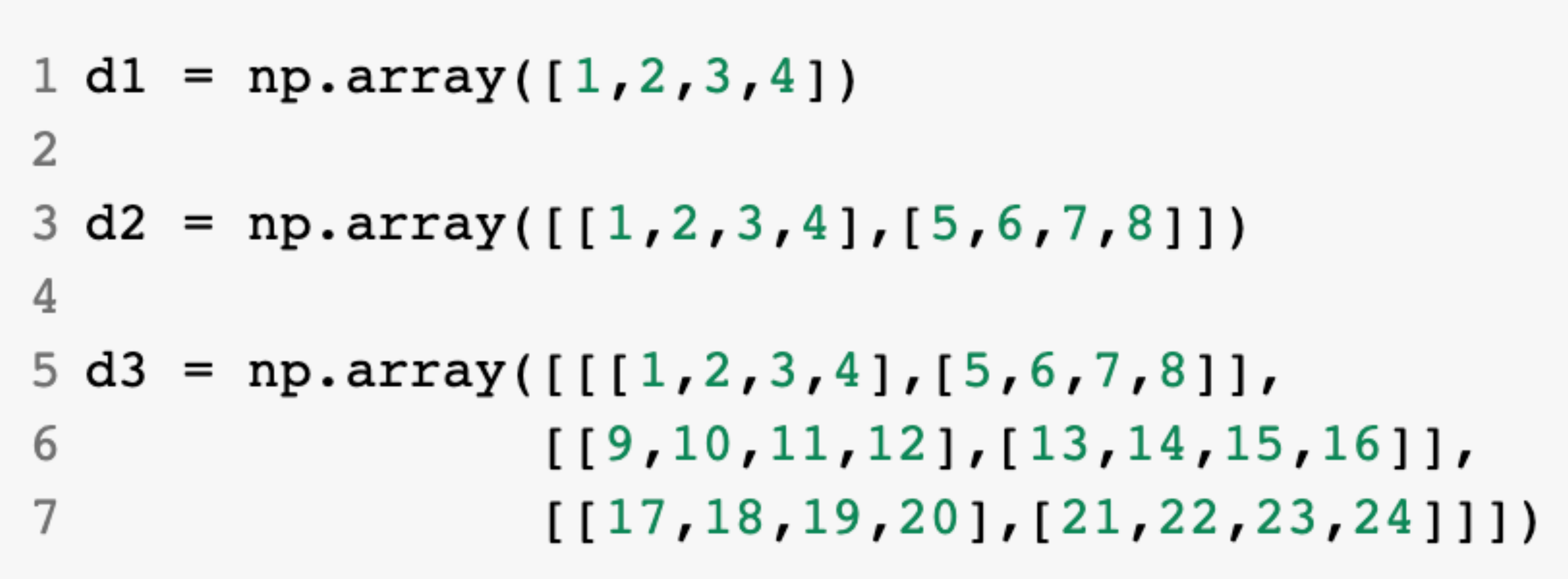
Example



```
print(d3)
print()
print(d3.flatten())
```

```
[[[ 1  2  3  4]
  [ 5  6  7  8]]

 [[ 9 10 11 12]
  [13 14 15 16]]

 [[17 18 19 20]
  [21 22 23 24]]]

[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
```

You can change the shape of a NumPy array



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

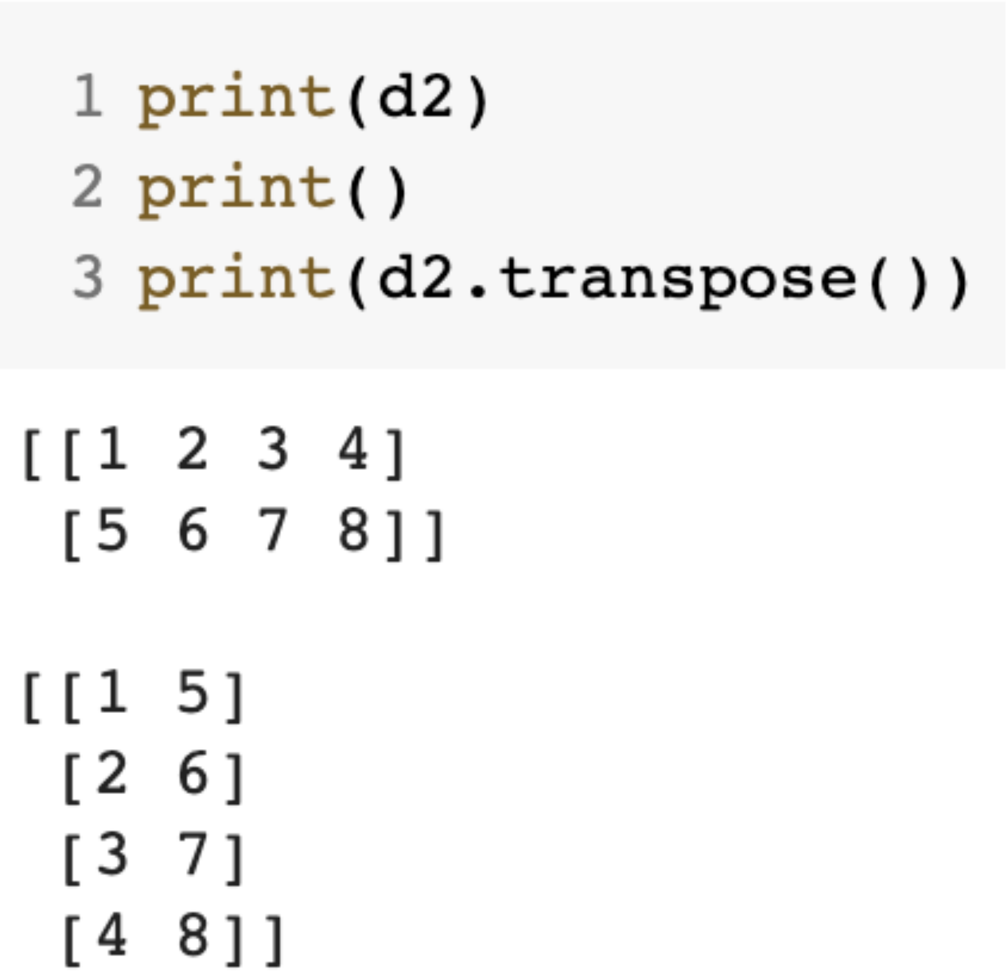
reshape( )

Changes the shape of the array into a given shape

flatten( )

Creates a copy of an array as a 1-D array

Example



```
print(d2)
print()
print(d2.transpose())
```

```
[[1 2 3 4]
 [5 6 7 8]]

[[1 5]
 [2 6]
 [3 7]
 [4 8]]
```

transpose( )

Permutes (e.g. rotates) the axes of an array

Arithmetic operations with arrays

Arithmetic operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponential
%	Remainder
//	Floor

Element-wise arithmetic between two or more arrays

```python
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])

print('a + b =',a + b)
print('a - b =',a - b)
print('a * b =',a * b)
```

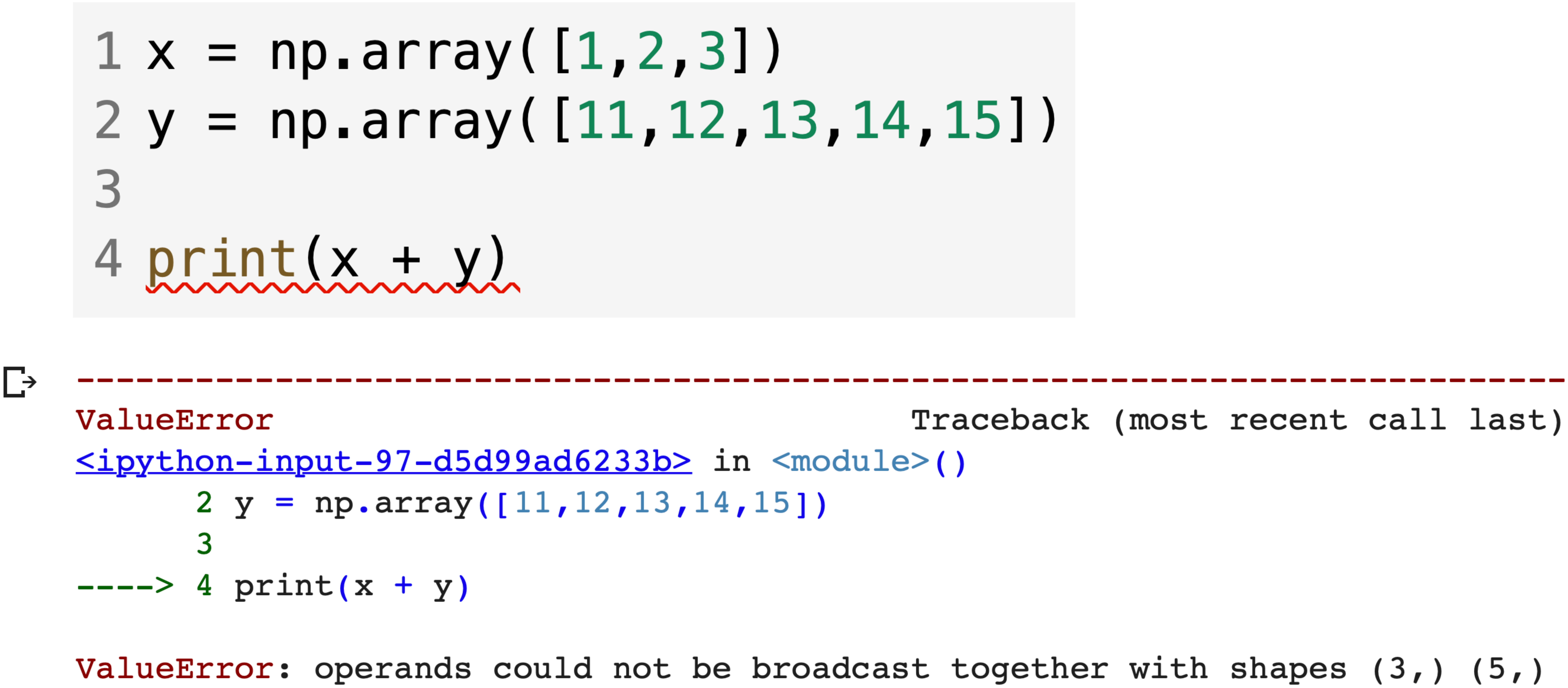
```
a + b = [ 6  8 10 12]
a - b = [-4 -4 -4 -4]
a * b = [ 5 12 21 32]
```

Element-wise arithmetic with an array and a number

```python
print('a + 10 =',a + 10)
print('10 * a =',10 * a)
print('a / 10 =',a / 10)
print('a**2 =',a**2)
```

```
a + 10 = [11 12 13 14]
10 * a = [10 20 30 40]
a / 10 = [0.1 0.2 0.3 0.4]
a**2 = [ 1  4  9 16]
```

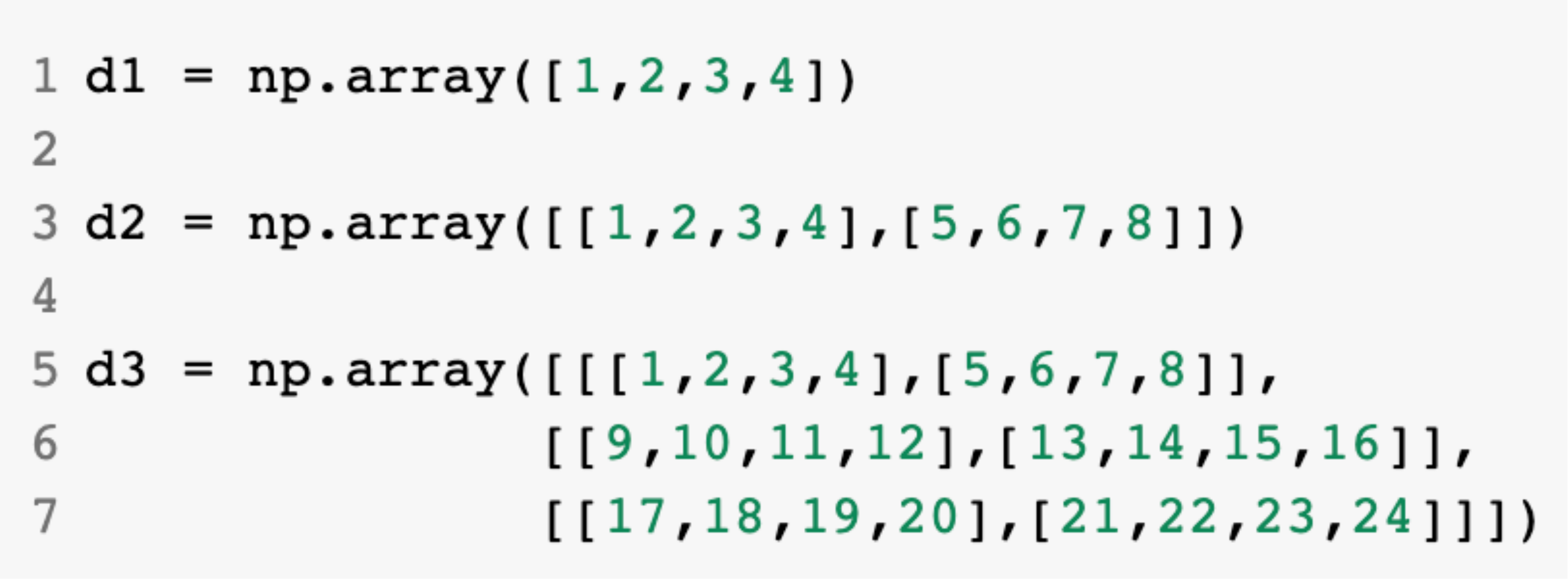
Element-wise operations require arrays to be the same dimensions



```python
x = np.array([1,2,3])
y = np.array([11,12,13,14,15])

print(x + y)
```

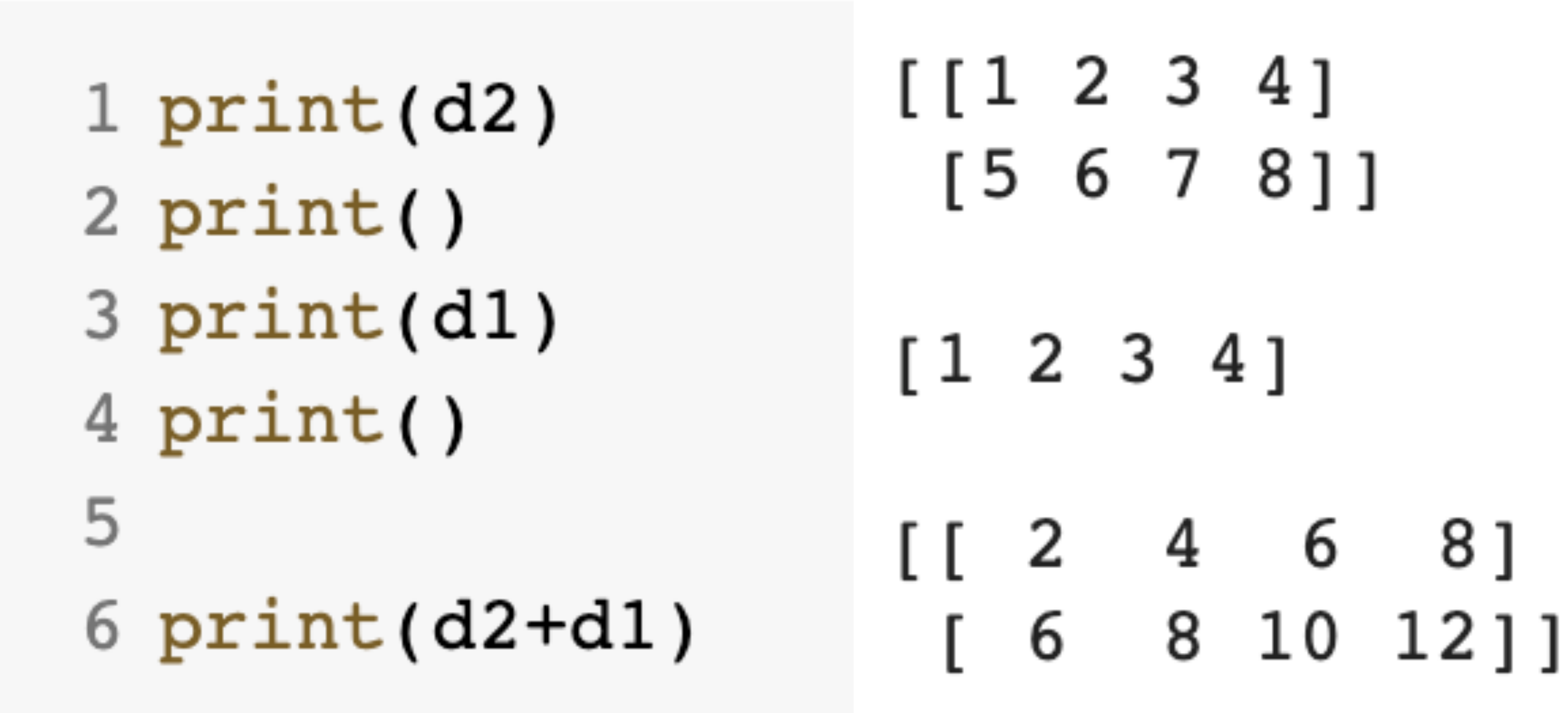
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-97-d5d99ad6233b> in <module>()
      2 y = np.array([11,12,13,14,15])
      3 
----> 4 print(x + y)

ValueError: operands could not be broadcast together with shapes (3,) (5,)
```

Element-wise operations require arrays to broadcast to the same dimensions



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

Example



```
print(d2)
print()
print(d1)
print()

print(d2+d1)
```

```
[[1 2 3 4]
 [5 6 7 8]]

[1 2 3 4]

[[ 2  4  6  8]
 [ 6  8 10 12]]
```

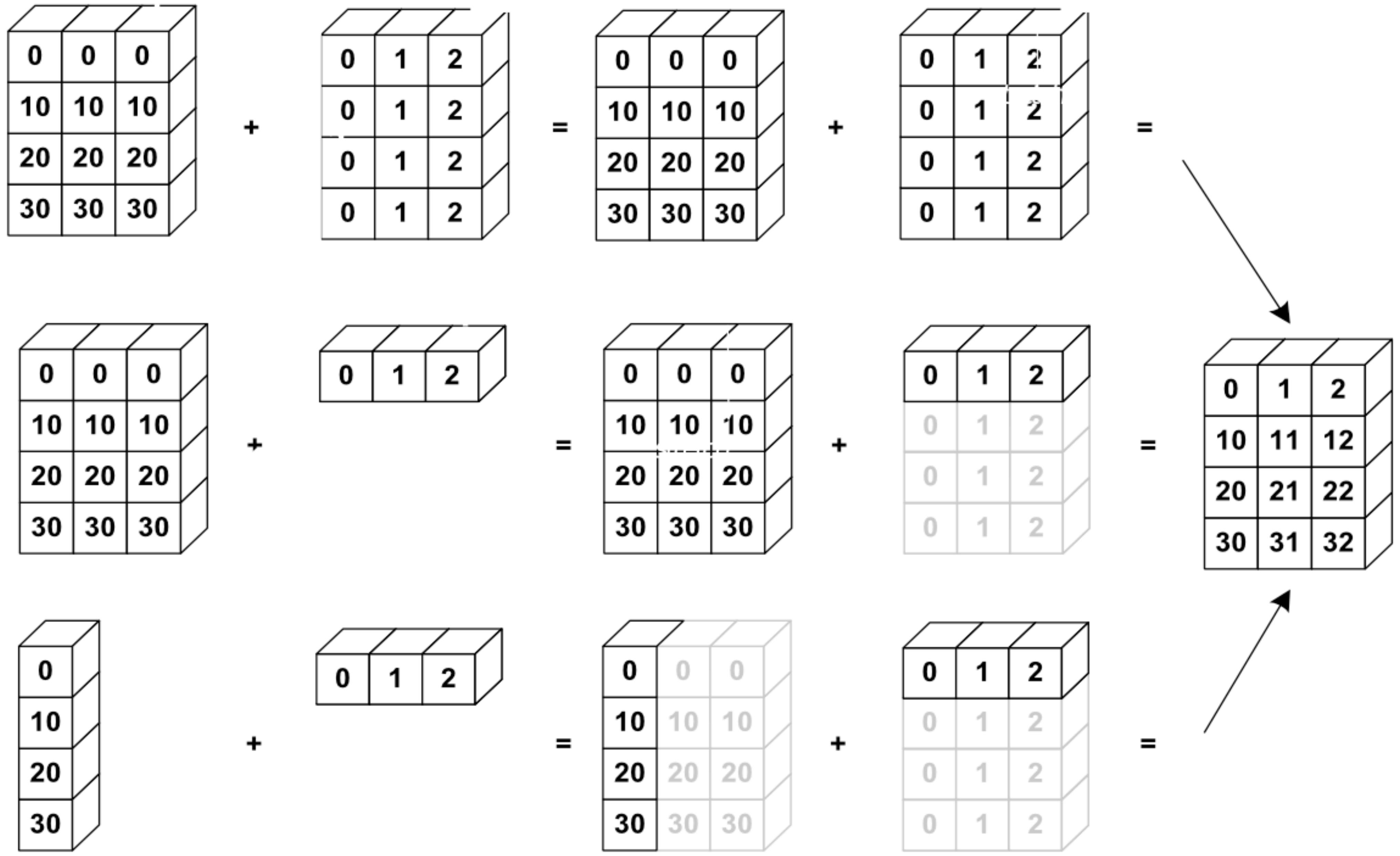
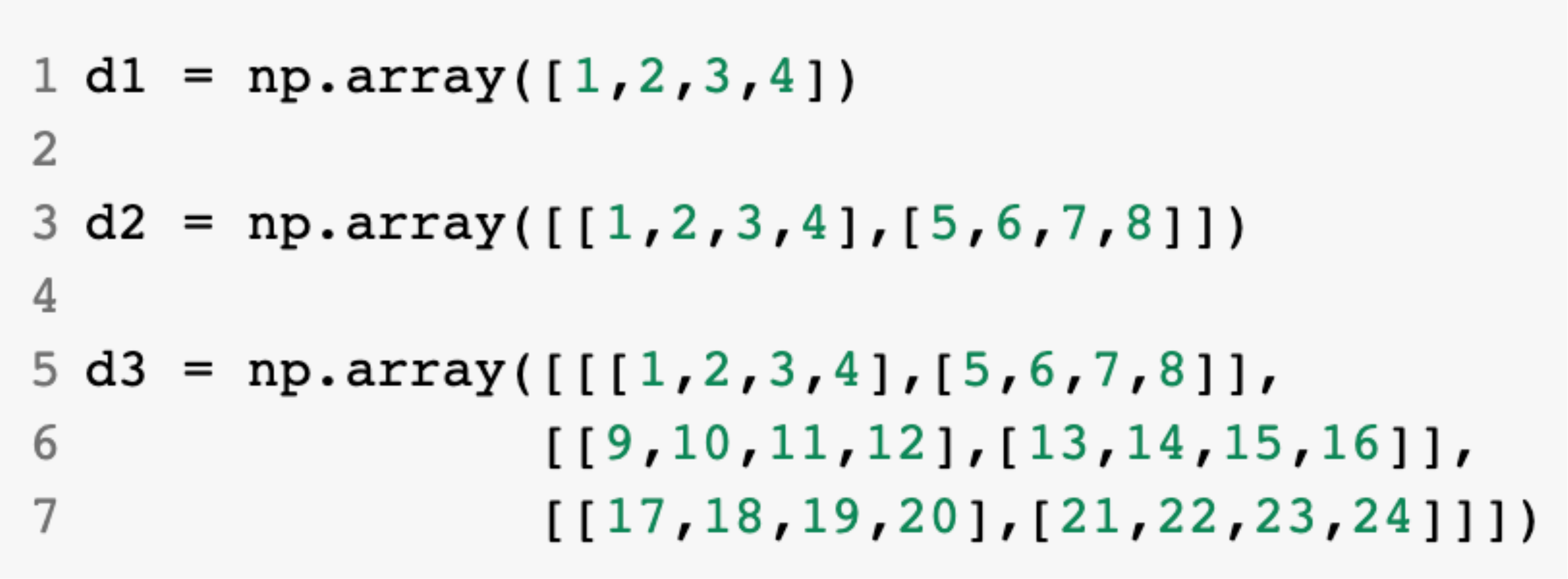


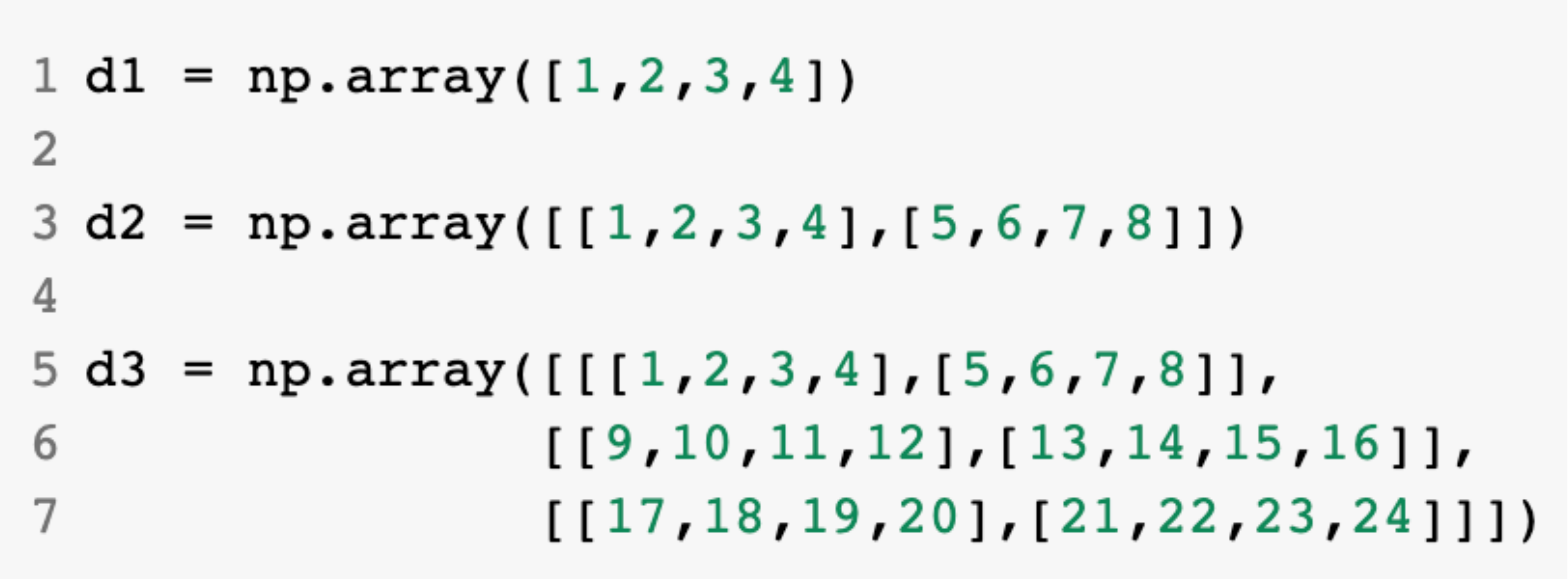
Image: http://scipy-lectures.org/_images/numpy_broadcasting.png

You can combine NumPy arrays



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

You can combine NumPy arrays

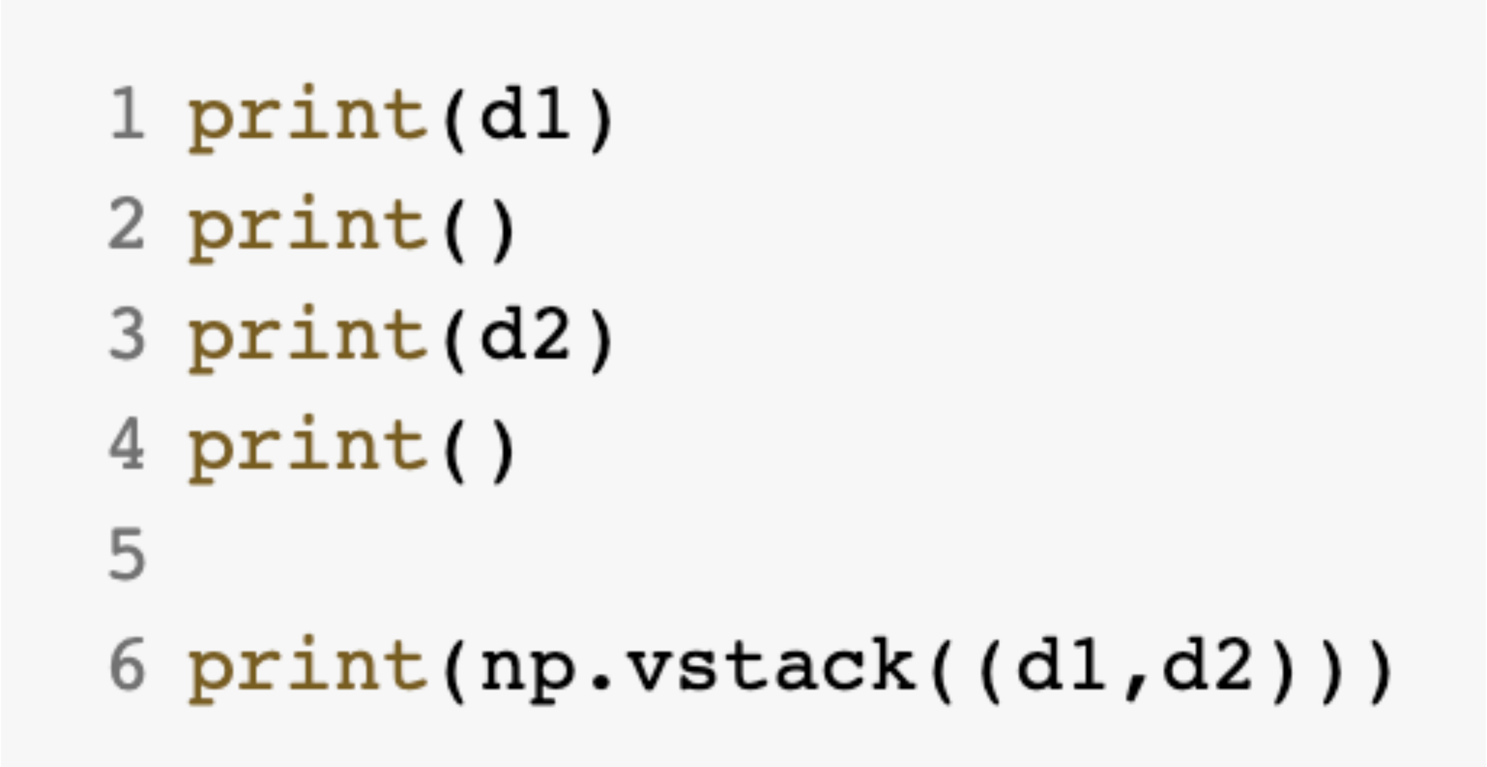


```python
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

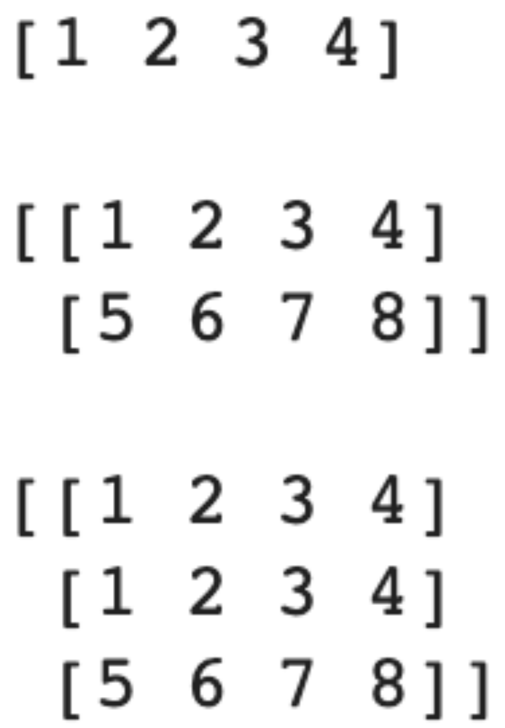
vstack( )

Stacks arrays on top of each other vertically

Example

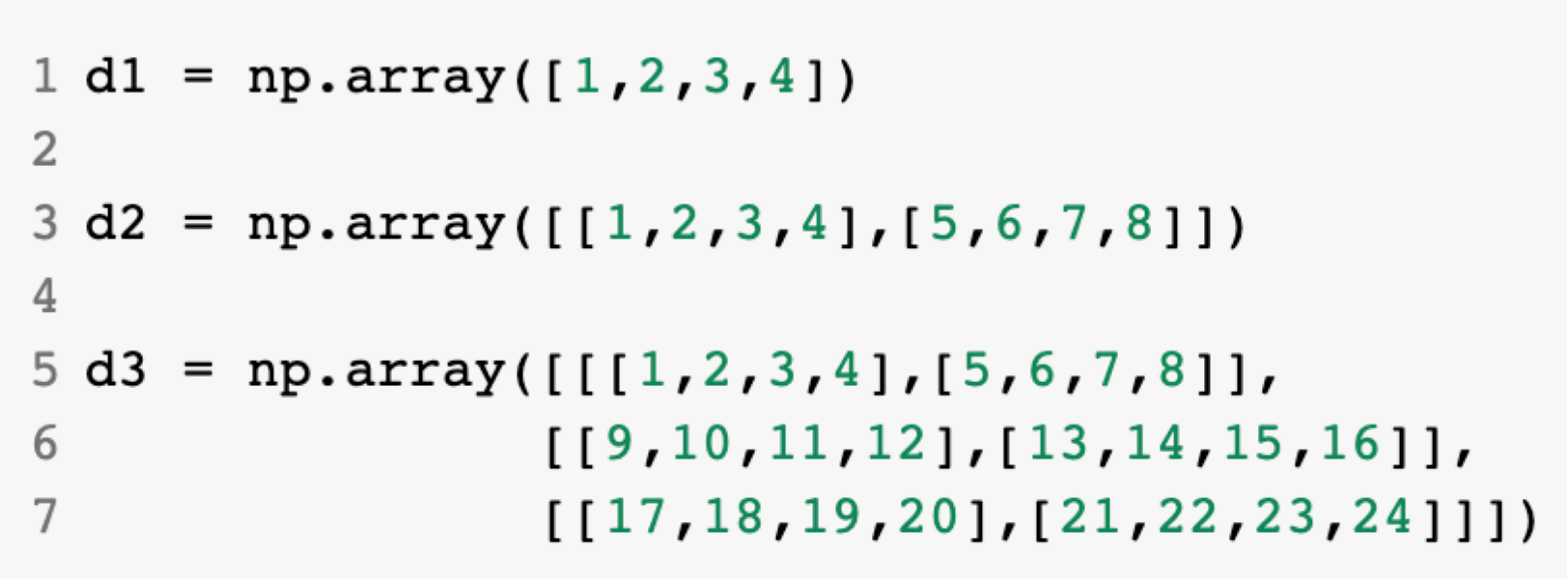


```python
print(d1)
print()
print(d2)
print()

print(np.vstack((d1,d2)))
```



```
[1 2 3 4]

[[1 2 3 4]
 [5 6 7 8]]

[[1 2 3 4]
 [1 2 3 4]
 [5 6 7 8]]
```

You can combine NumPy arrays



```
d1 = np.array([1,2,3,4])

d2 = np.array([[1,2,3,4],[5,6,7,8]])

d3 = np.array([[[1,2,3,4],[5,6,7,8]],
               [[9,10,11,12],[13,14,15,16]],
               [[17,18,19,20],[21,22,23,24]]])
```

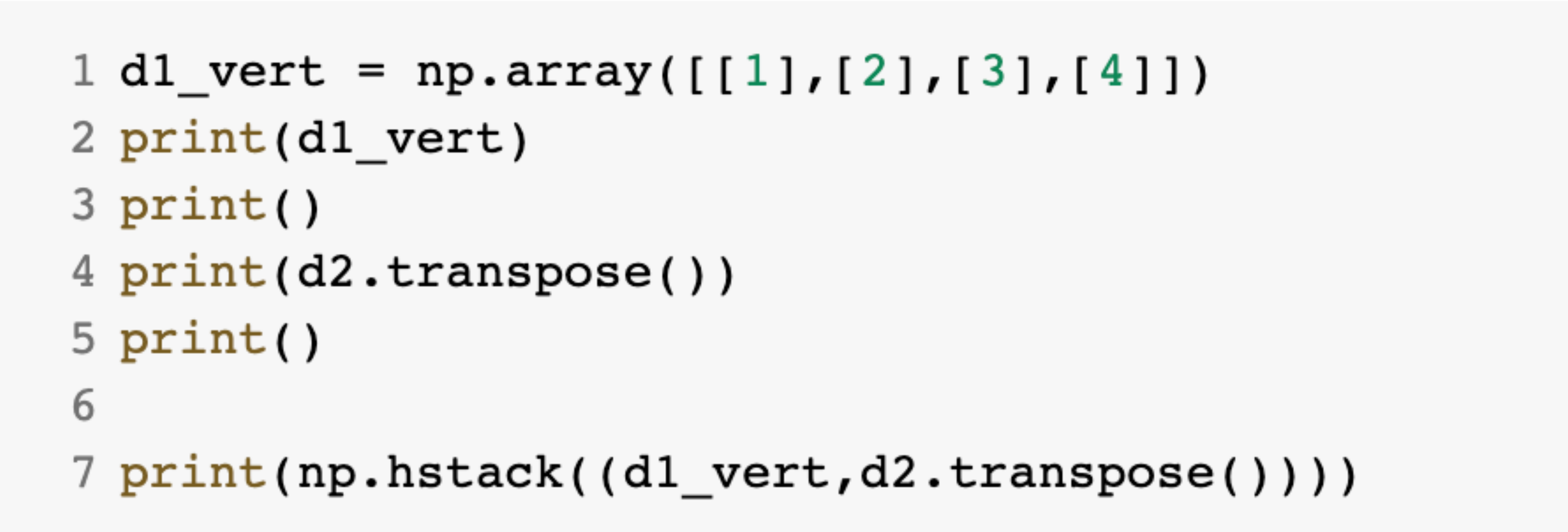
vstack( )

Stacks arrays on top of each other vertically

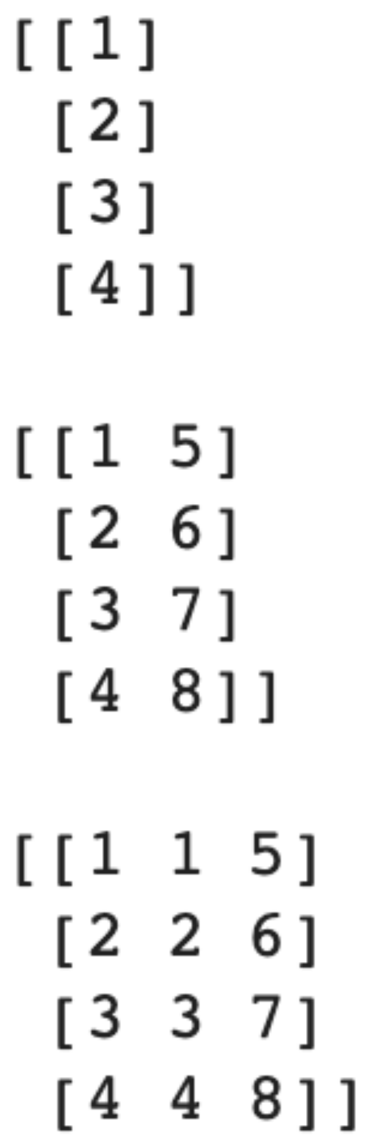
hstack( )

Stacks arrays horizontally

Example



```
d1_vert = np.array([[1],[2],[3],[4]])
print(d1_vert)
print()
print(d2.transpose())
print()

print(np.hstack((d1_vert,d2.transpose())))
```



```
[[1]
 [2]
 [3]
 [4]]

[[1 5]
 [2 6]
 [3 7]
 [4 8]]

[[1 1 5]
 [2 2 6]
 [3 3 7]
 [4 4 8]]
```

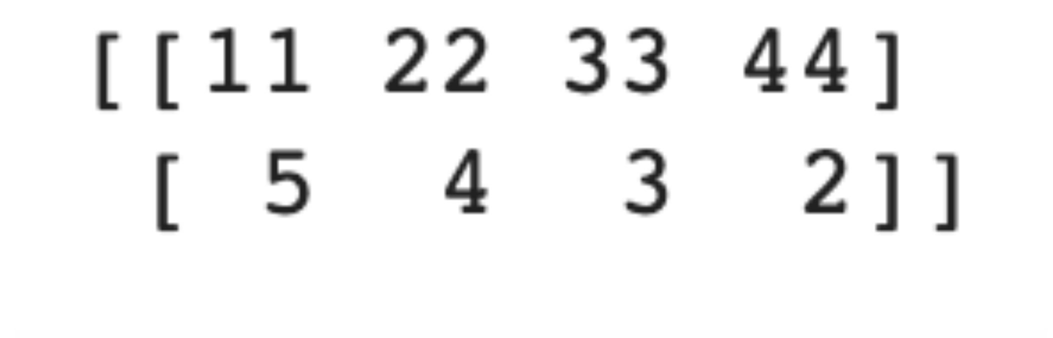
Mathematical reductions (array → number)

```
x = np.array([10,11,12,13])
```

Function:	Purpose:	Evaluates to:
`np.sum(x)`	Sum	46
`np.mean(x)`	Mean (average)	11.5
`np.median(x)`	Median	11.5
`np.max(x)`	Maximum value	13
`np.min(x)`	Minimum value	10
`np.std(x)`	Standard deviation	1.11803...

Mathematical reductions (array → number)

```
x = np.array([[11,22,33,44],[5,4,3,2]])
```

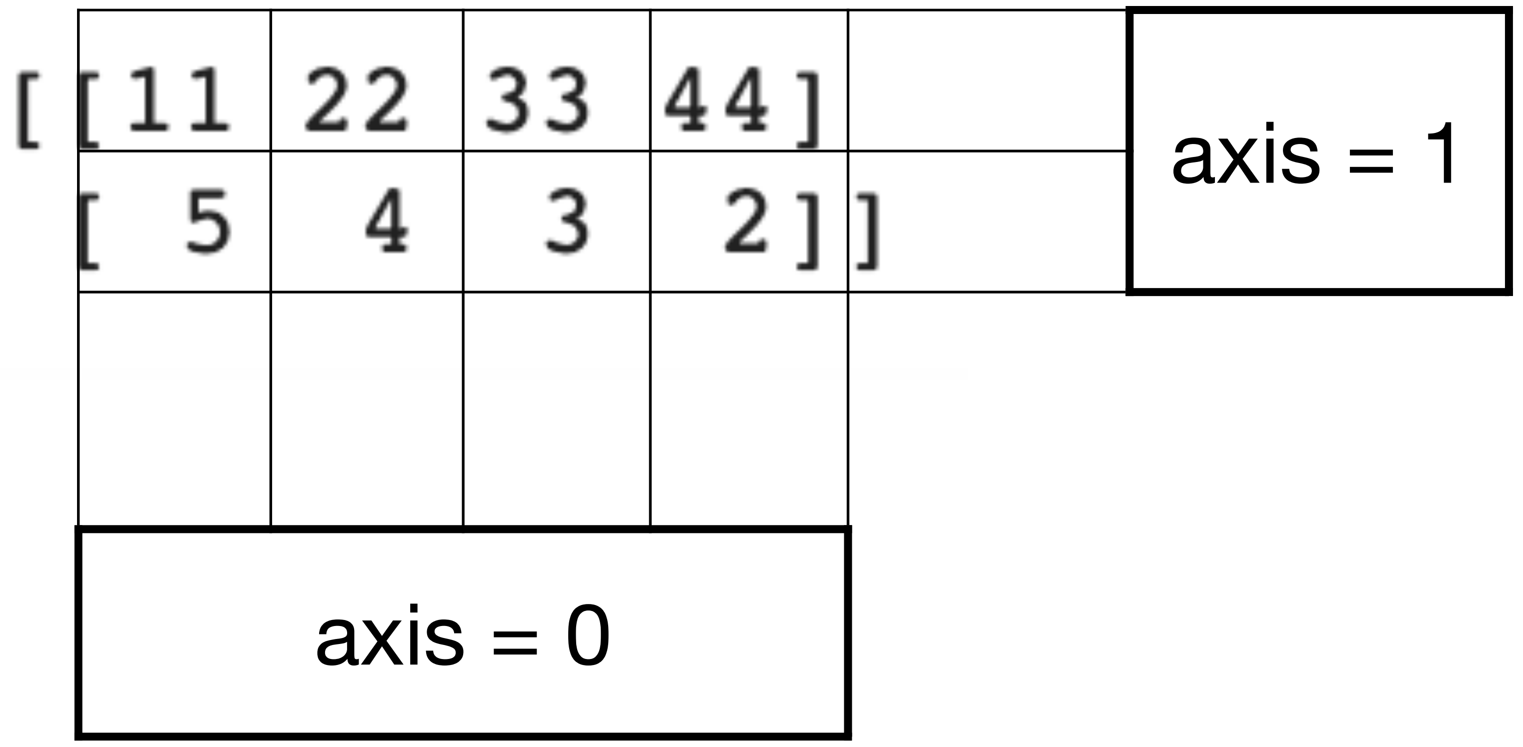


```
[[11 22 33 44]
 [ 5  4  3  2]]
```

Function:	Evaluates to:
`np.sum(x,axis=0)`	`[16 26 36 46]`
`np.mean(x,axis=1)`	`[27.5  3.5]`
`np.median(x,axis=0)`	`[ 8. 13. 18. 23.]`
`np.max(x)`	`44`
`np.min(x,axis=1)`	`[11  2]`
`np.std(x)`	`14.84082…`

Mathematical reductions (array → number)

```
x = np.array([[11,22,33,44],[5,4,3,2]])
```



```
[[11 22 33 44]
 [ 5  4  3  2]]
```

Function:	Evaluates to:
`np.sum(x,axis=0)`	`[16 26 36 46]`
`np.mean(x,axis=1)`	`[27.5  3.5]`
`np.median(x,axis=0)`	`[ 8. 13. 18. 23.]`
`np.max(x)`	`44`
`np.min(x,axis=1)`	`[11  2]`
`np.std(x)`	`14.84082…`

Functions to create new arrays

Function:	Purpose:	Evaluates to arrays:
`np.zeros(4)`	Array of given length filled with zeros	`[0.,0.,0.,0.]`
`np.ones(4)`	Array of given length filled with ones	`[1.,1.,1.,1.]`
`np.full(4,2)`	Array of given length filled with given value	`[2,2,2,2]`
`np.arange(4)`	Same as `range()`…	`[0,1,2,3]`
`np.arange(0,1,0.25)`	…except floats and fractional increments are allowed	`[0.,0.25,0.5,0.75]`
`np.linspace(0,1,5)`	Returns the given number of evenly spaced values from start to end (both are inclusive)	`[0.,0.25,0.5,0.75,1.]`

Functions to create new arrays

Function:	Purpose:	Evaluates to arrays:

`np.zeros((4,3))`	Array of given length filled with zeros	`[[0. 0. 0.]`
		`[0. 0. 0.]`
		`[0. 0. 0.]`
		`[0. 0. 0.]]`

`np.ones((4,))`	Array of given length filled with ones	`[1. 1. 1. 1.]`

`np.full((2,3,4),2)`	Array of given length filled with given value	`[[[2 2 2 2]`
		`[2 2 2 2]`
		`[2 2 2 2]]`
		
		`[[2 2 2 2]`
		`[2 2 2 2]`
		`[2 2 2 2]]]`

What we'll cover in this lesson

1. Multi-dimensional arrays

2. More array functions

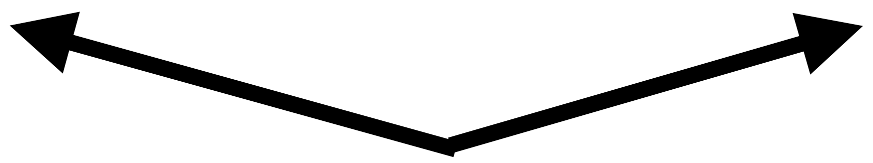
3. **Dictionaries**

4. Datetime

Python dictionaries

A dictionary is a collection of items, each with a key/value pair.

```
my_dict = { ‘Name’ : ‘Katy’, ‘Email’ : ‘katyc4@uw.edu',
                ‘Office’ : ‘OSB331A’ }
```

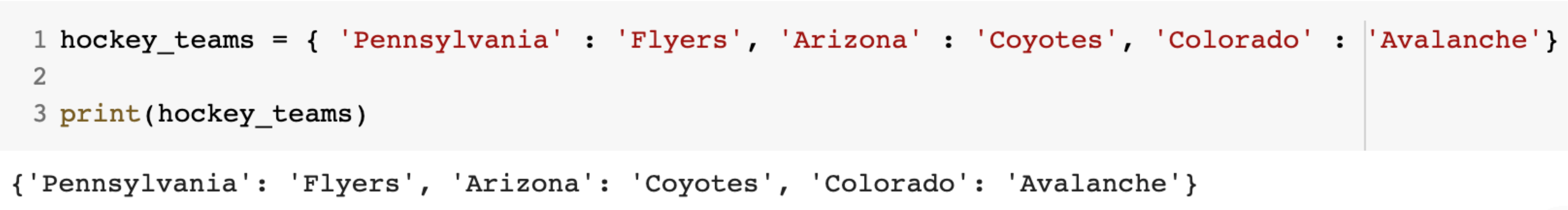


```
{ Key : Value }
```

To call a dictionary item, use its key!

```
print(my_dict[‘Name’])          ‘Katy’
```

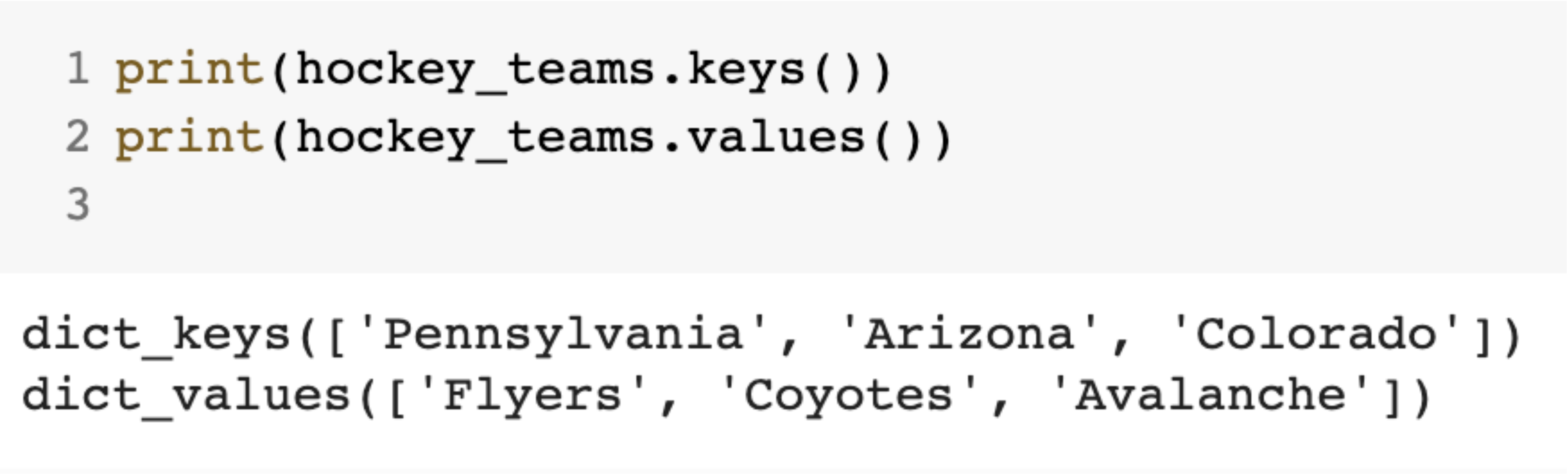
Python dictionaries



```python
hockey_teams = { 'Pennsylvania' : 'Flyers', 'Arizona' : 'Coyotes', 'Colorado' : 'Avalanche'}

print(hockey_teams)
```

```
{'Pennsylvania': 'Flyers', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

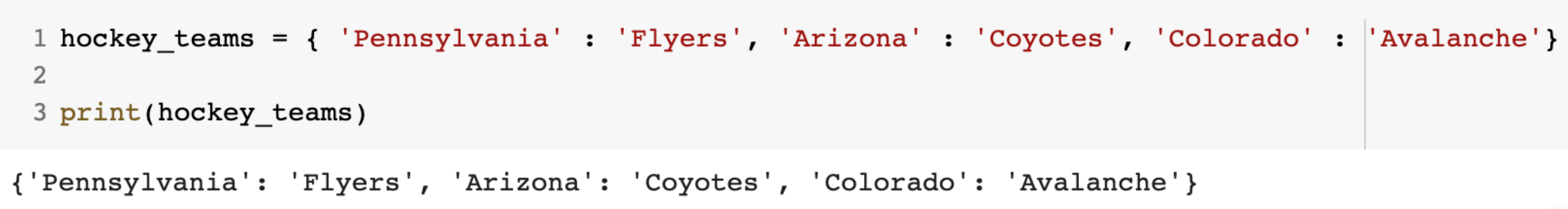
Get the keys and values from a dictionary...



```python
print(hockey_teams.keys())
print(hockey_teams.values())

```

```
dict_keys(['Pennsylvania', 'Arizona', 'Colorado'])
dict_values(['Flyers', 'Coyotes', 'Avalanche'])
```

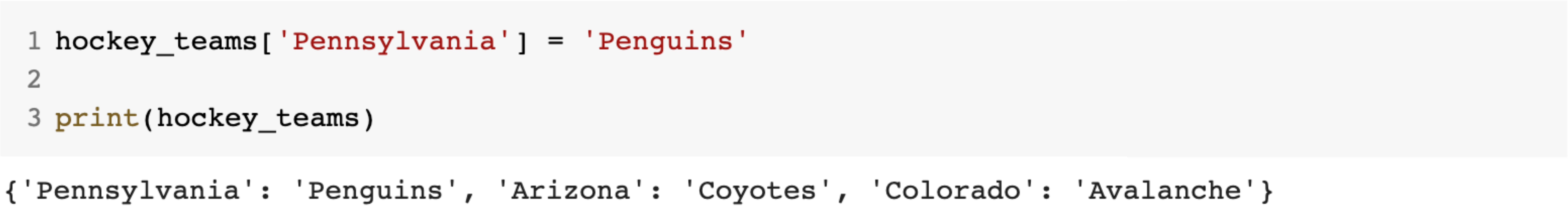
Python dictionaries



```python
hockey_teams = { 'Pennsylvania' : 'Flyers', 'Arizona' : 'Coyotes', 'Colorado' : 'Avalanche'}

print(hockey_teams)
```

```
{'Pennsylvania': 'Flyers', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

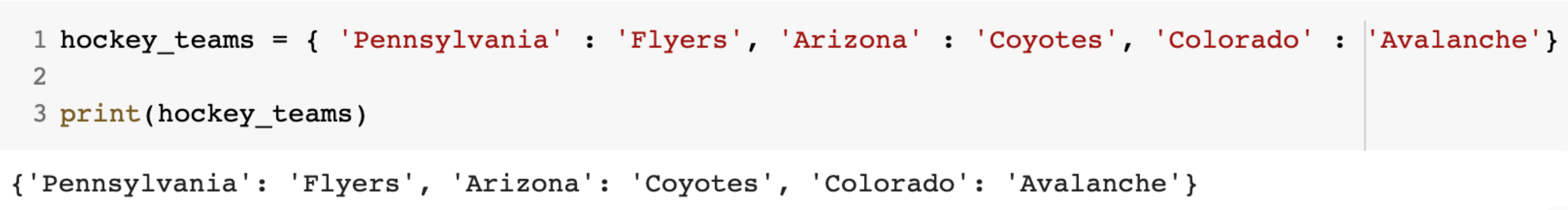
Replace values using the key name.



```python
hockey_teams['Pennsylvania'] = 'Penguins'

print(hockey_teams)
```

```
{'Pennsylvania': 'Penguins', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

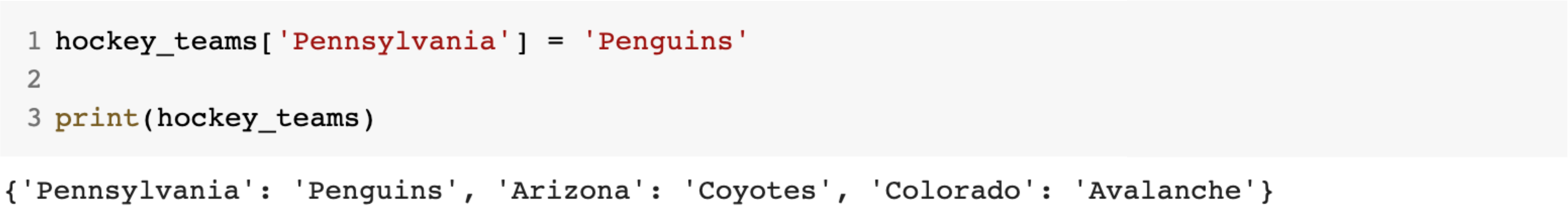
Python dictionaries



```python
hockey_teams = { 'Pennsylvania' : 'Flyers', 'Arizona' : 'Coyotes', 'Colorado' : 'Avalanche'}

print(hockey_teams)
```

```
{'Pennsylvania': 'Flyers', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

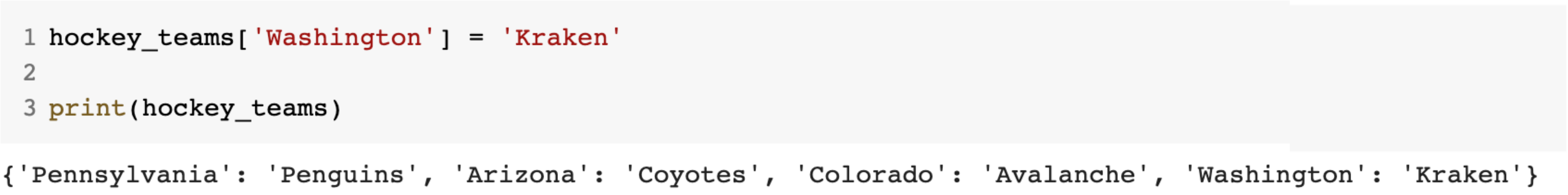
Replace values using the key name.



```python
hockey_teams['Pennsylvania'] = 'Penguins'

print(hockey_teams)
```

```
{'Pennsylvania': 'Penguins', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

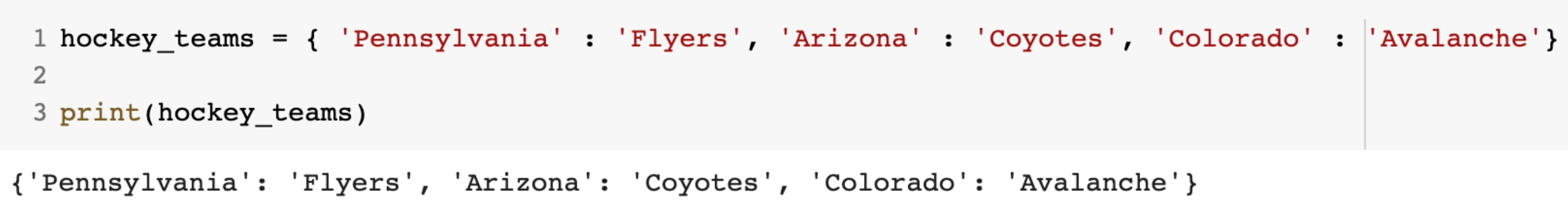
Create new keys/values by using square brackets to append them.



```python
hockey_teams['Washington'] = 'Kraken'

print(hockey_teams)
```

```
{'Pennsylvania': 'Penguins', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche', 'Washington': 'Kraken'}
```

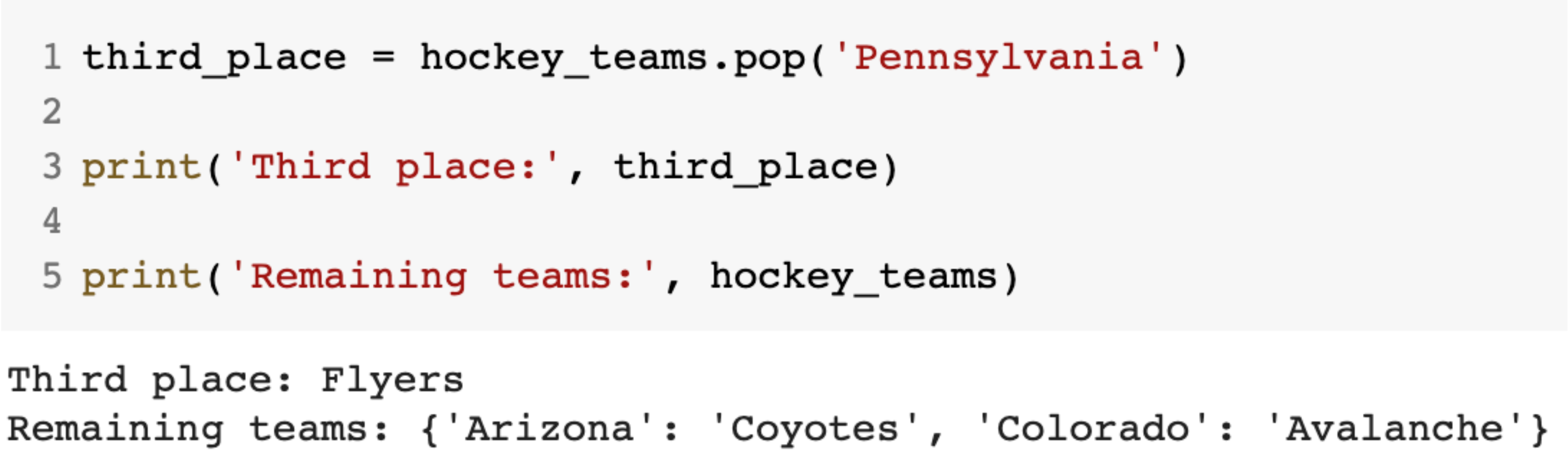
Python dictionaries



```python
hockey_teams = { 'Pennsylvania' : 'Flyers', 'Arizona' : 'Coyotes', 'Colorado' : 'Avalanche'}

print(hockey_teams)
```

```
{'Pennsylvania': 'Flyers', 'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

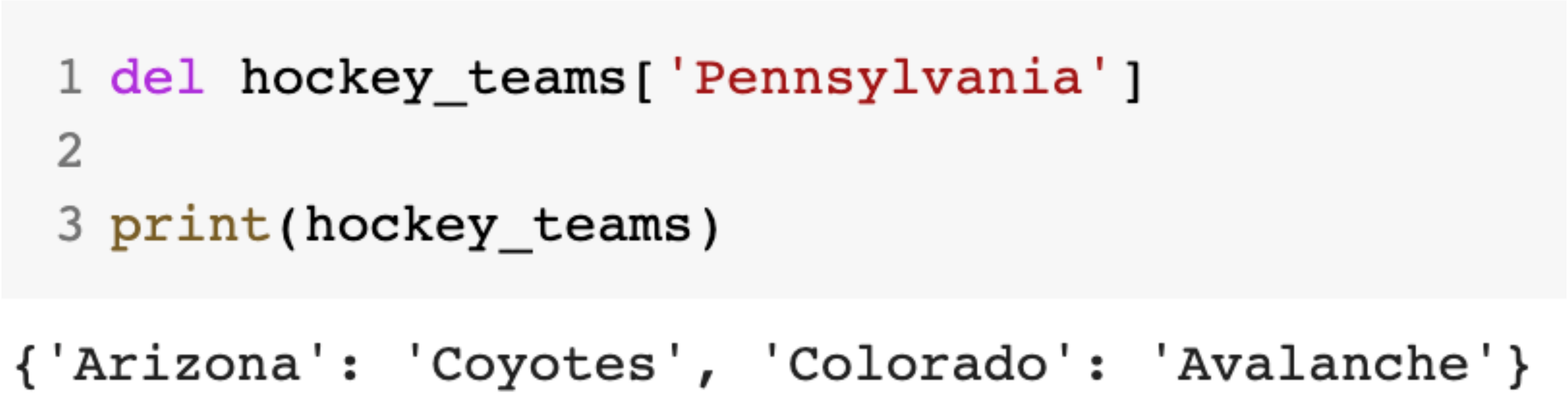
Remove keys (and their values) using pop( ).



```python
third_place = hockey_teams.pop('Pennsylvania')

print('Third place:', third_place)

print('Remaining teams:', hockey_teams)
```

```
Third place: Flyers
Remaining teams: {'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

Or del



```python
del hockey_teams['Pennsylvania']

print(hockey_teams)
```

```
{'Arizona': 'Coyotes', 'Colorado': 'Avalanche'}
```

What we'll cover in this lesson

1. Multi-dimensional arrays

2. More array functions

3. Dictionaries

4. **Datetime**

STANDARD TIME ZONES OF THE WORLD

-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
+1
+2
+3
+4
+5
+6
+7
+8
+9
+10
+11
+12
-12
ARCTIC OCEAN
NORTH PACIFIC OCEAN
NORTH ATLANTIC OCEAN
SOUTH PACIFIC OCEAN
SOUTH ATLANTIC OCEAN
INDIAN OCEAN
Greenland (DENMARK)
CANADA
UNITED STATES
MEXICO
BRAZIL
ARGENTINA
CHILE
PERU
BOLIVIA
COLOMBIA
VENEZUELA
RUSSIA
CHINA
MONGOLIA
KAZAKHSTAN
INDIA
IRAN
SAUDI ARABIA
ALGERIA
LIBYA
EGYPT
NIGER
MALI
CHAD
SUDAN
ETHIOPIA
ANGOLA
SOUTH AFRICA
INDONESIA
AUSTRALIA
JAPAN
PHILIPPINES
NEW ZEALAND
French Polynesia (FRANCE)
KIRIBATI
International Date Line
Eastward across Date Line subtract 24 hours
Westward across Date Line add 24 hours
Coordinated Universal Time (UTC)
formerly Greenwich Mean Time (GMT)
(number indicates standard time in zone when it is 12 noon, UTC)
Sun 1:00 -11
2:00 -10
3:00 -9
4:00 -8
5:00 -7
6:00 -6
7:00 -5
8:00 -4
9:00 -3
10:00 -2
11:00 -1
Sun 12:00 0
13:00 +1
14:00 +2
15:00 +3
16:00 +4
17:00 +5
18:00 +6
19:00 +7
20:00 +8
21:00 +9
22:00 +10
23:00 +11
Sun 24:00 +12
Sun 00:00 -12
Sun 1:00 -11

Boundary representation is not necessarily authoritative.

Add time zone number to local time to obtain UTC.
Subtract time zone number from UTC to obtain local time.

WEST

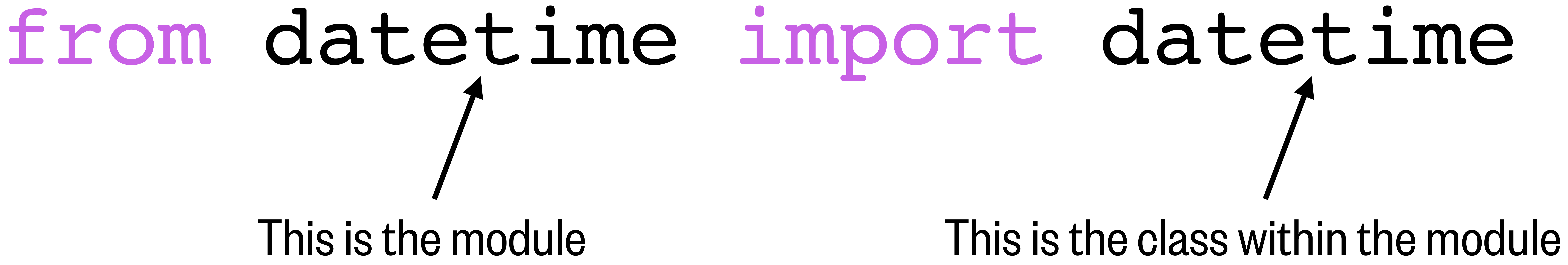
EAST

Subtract time zone number from local time to obtain UTC.
Add time zone number to UTC to obtain local time.

18-5386 5-18

https://upload.wikimedia.org/wikipedia/commons/thumb/8/88/World_Time_Zones_Map.png/1599px-World_Time_Zones_Map.png

Loading datetime



```
from datetime import datetime
```

This is the module

This is the class within the module

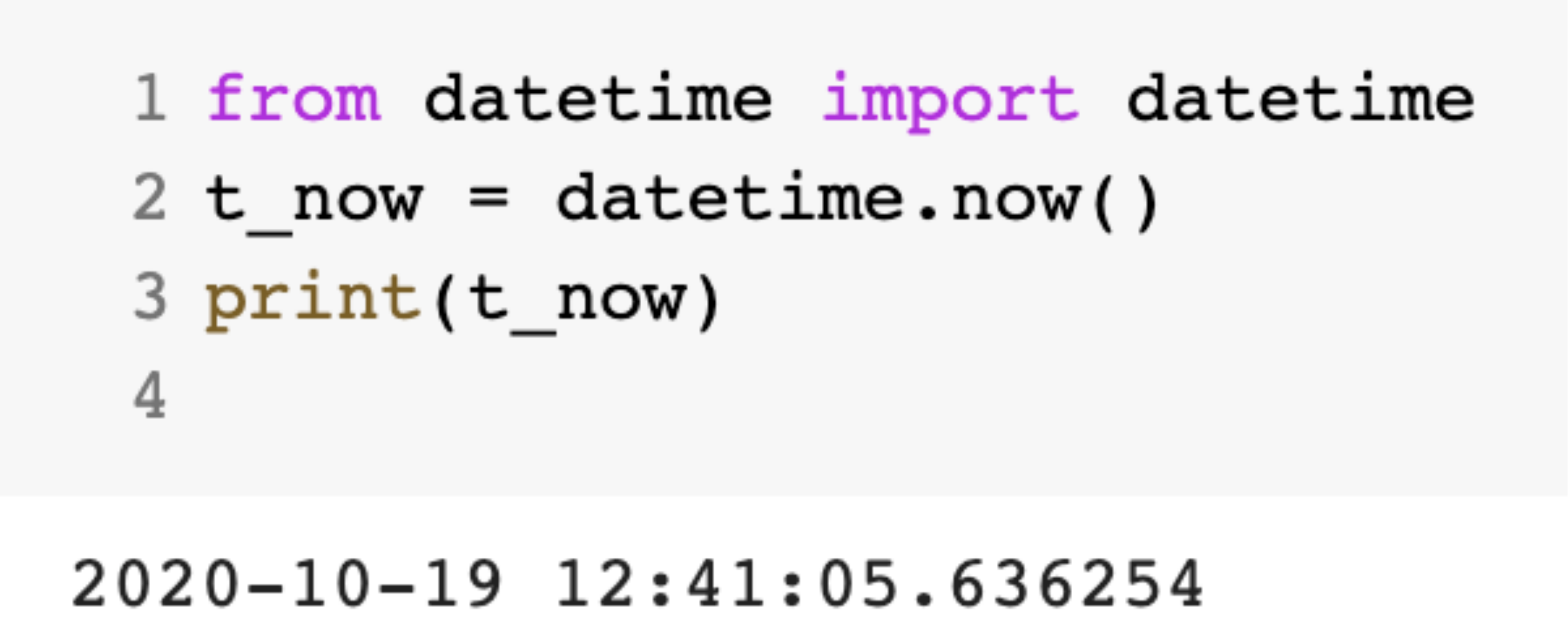
Pseudocode:

From the datetime module, I am importing the datetime class
which will allow me to use the functions stored there.

Class datetime objects

Get the current date and time

`datetime.now()`



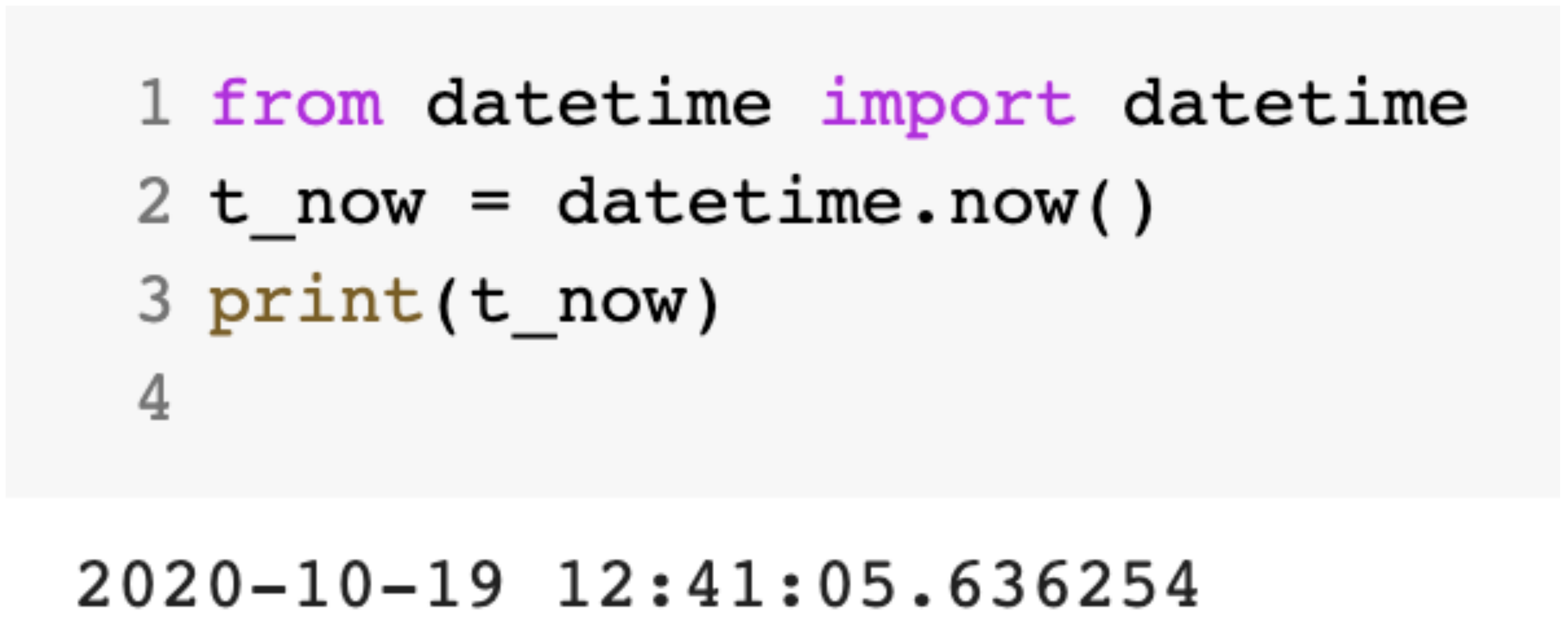
```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

Class datetime objects

Get the current date and time

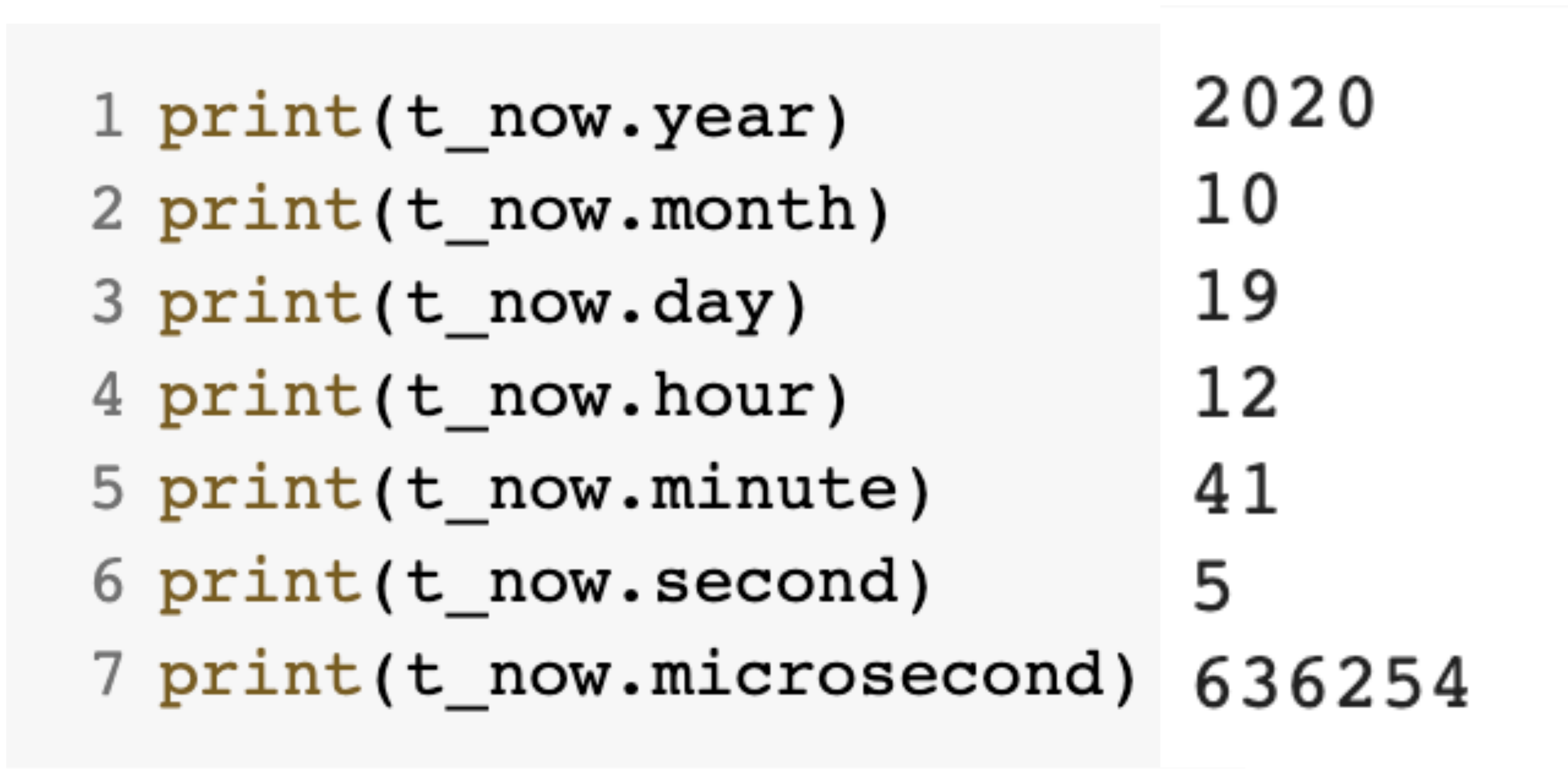
`datetime.now()`



```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

Retrieve the individual values from the datetime object



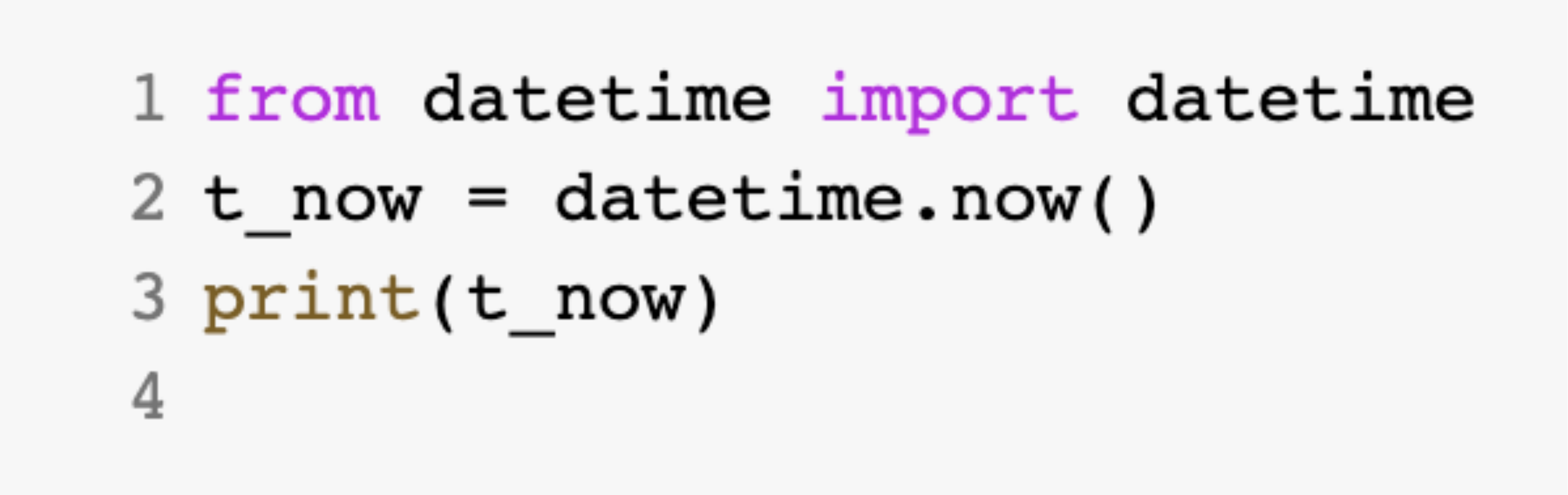
```python
print(t_now.year)
print(t_now.month)
print(t_now.day)
print(t_now.hour)
print(t_now.minute)
print(t_now.second)
print(t_now.microsecond)
```

```
2020
10
19
12
41
5
636254
```

Class datetime objects

Get the current date and time

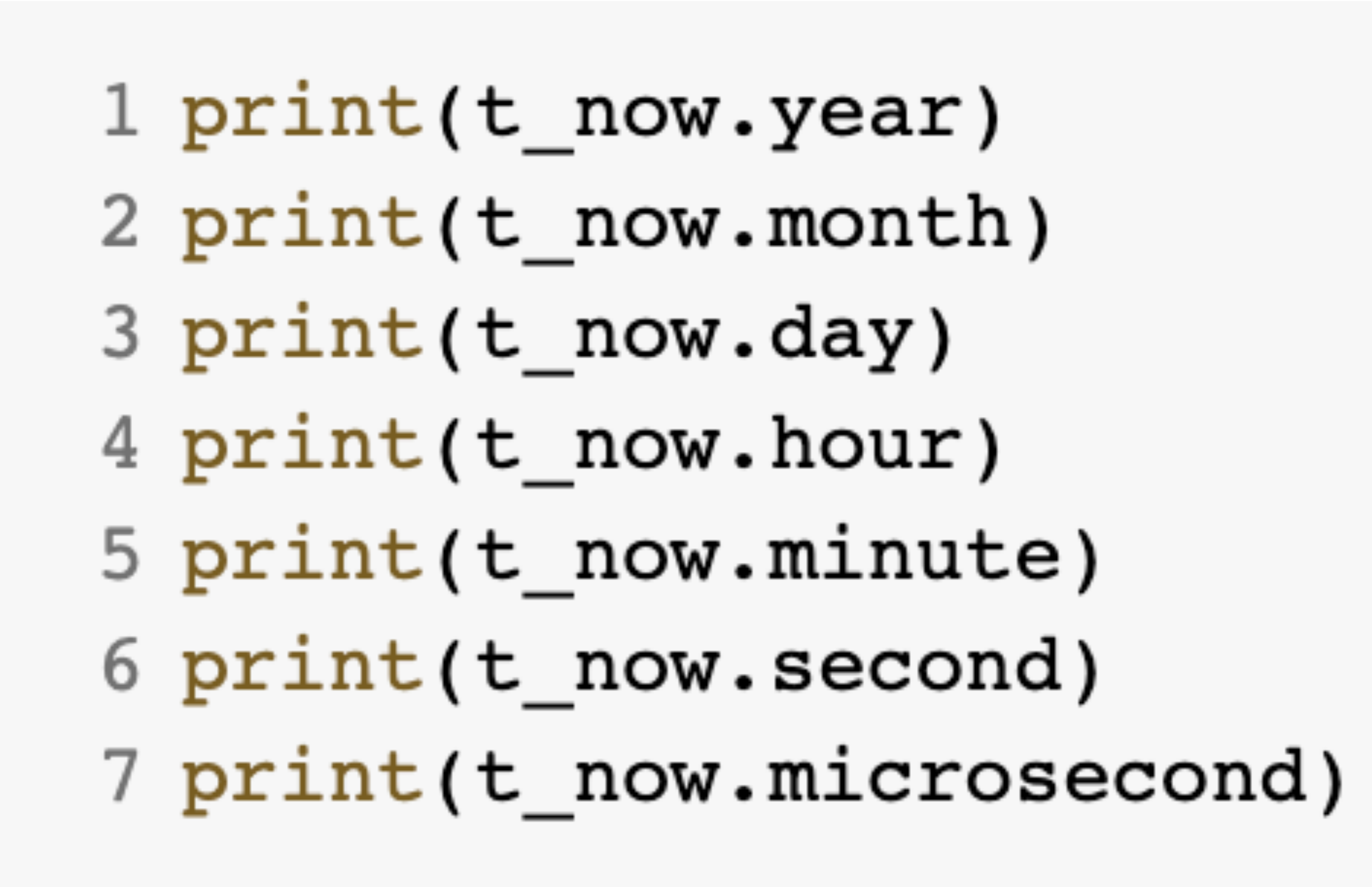
`datetime.now()`



```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

Retrieve the individual values from the datetime object



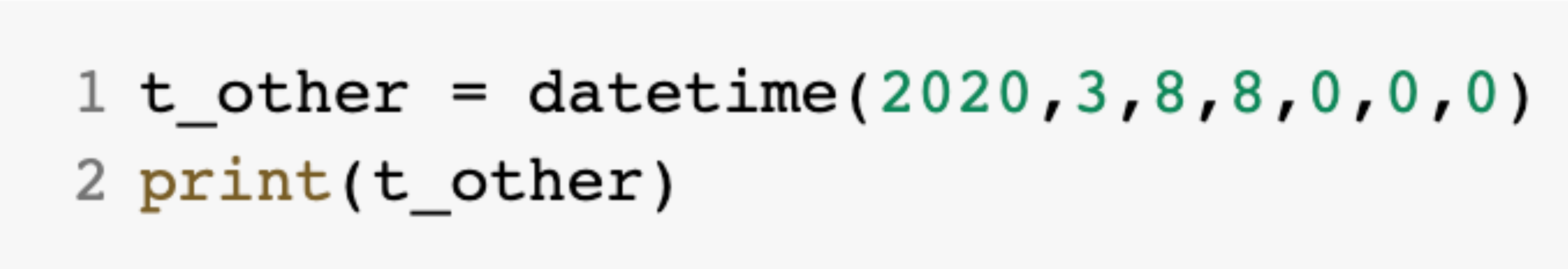
```python
print(t_now.year)
print(t_now.month)
print(t_now.day)
print(t_now.hour)
print(t_now.minute)
print(t_now.second)
print(t_now.microsecond)
```

```
2020
10
19
12
41
5
636254
```

Create a datetime object for any time

`datetime(year, month, day, hour, minute, second, microsecond)`

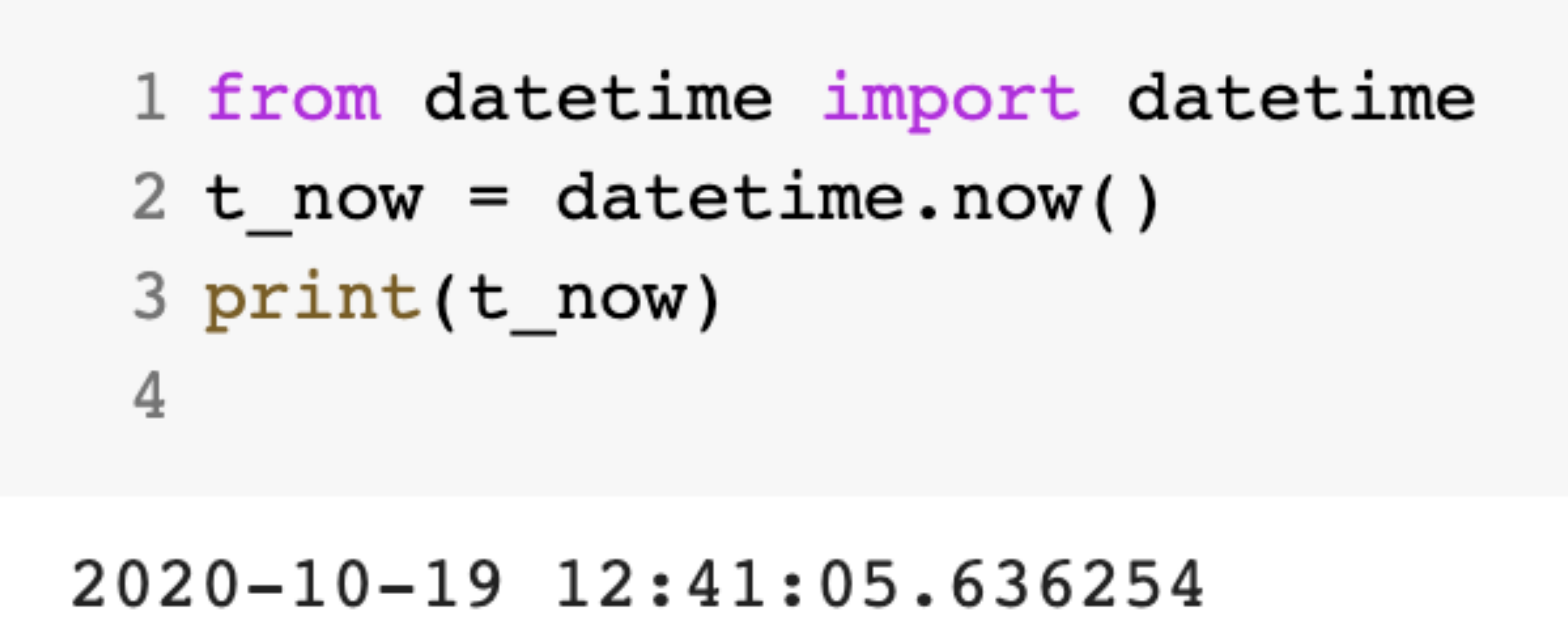
This part is optional



```python
t_other = datetime(2020,3,8,8,0,0,0)
print(t_other)
```

```
2020-03-08 08:00:00
```

Datetime objects to/from strings

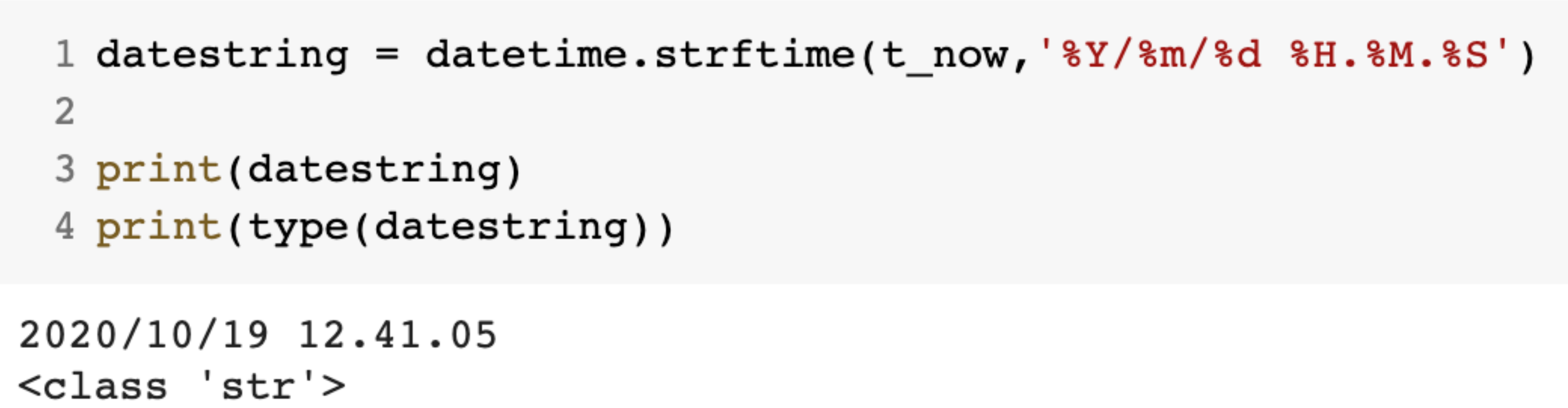


```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

Change into string

`datetime.strftime()`

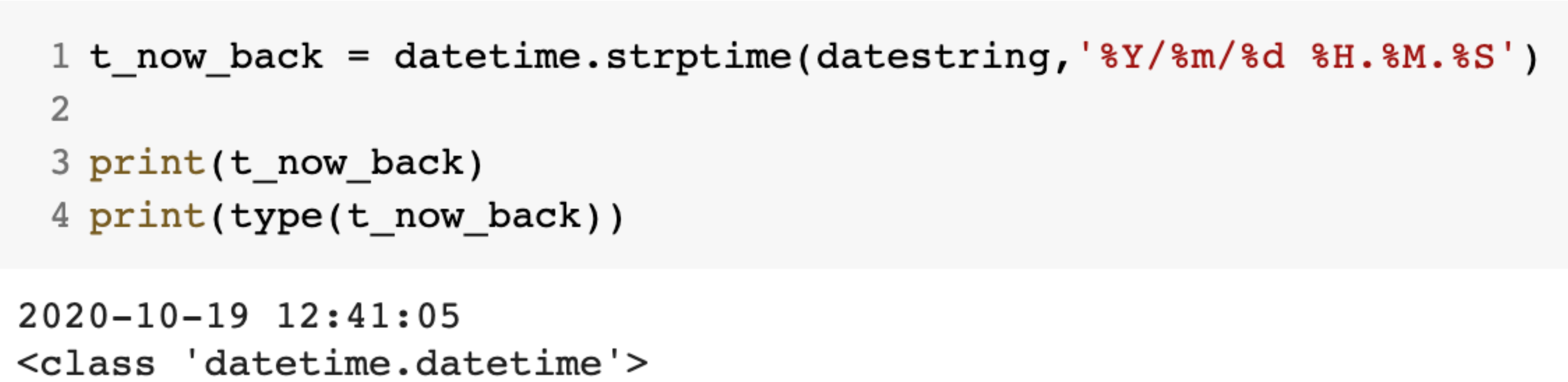


```python
datestring = datetime.strftime(t_now,'%Y/%m/%d %H.%M.%S')

print(datestring)
print(type(datestring))
```

```
2020/10/19 12.41.05
<class 'str'>
```

Change from a string

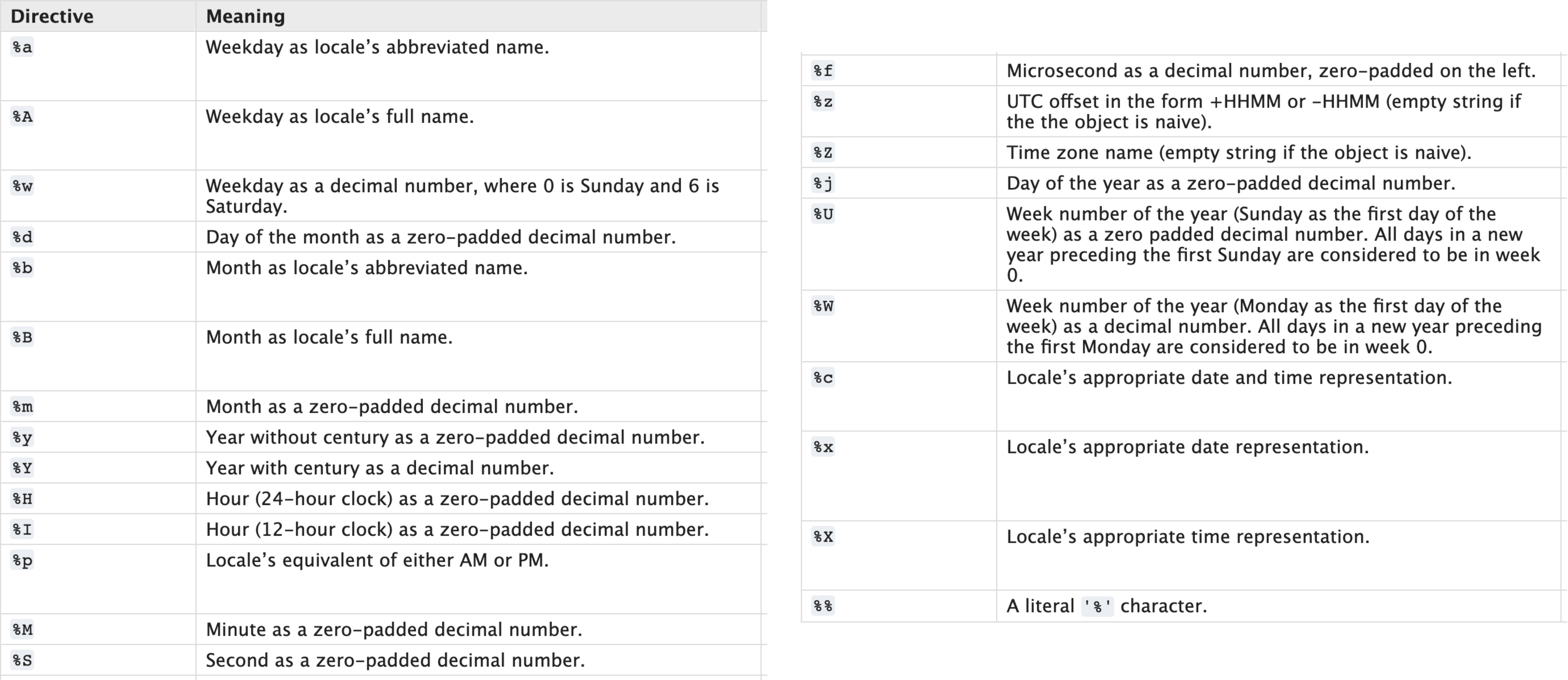
`datetime.strptime()`



```python
t_now_back = datetime.strptime(datestring,'%Y/%m/%d %H.%M.%S')

print(t_now_back)
print(type(t_now_back))
```

```
2020-10-19 12:41:05
<class 'datetime.datetime'>
```

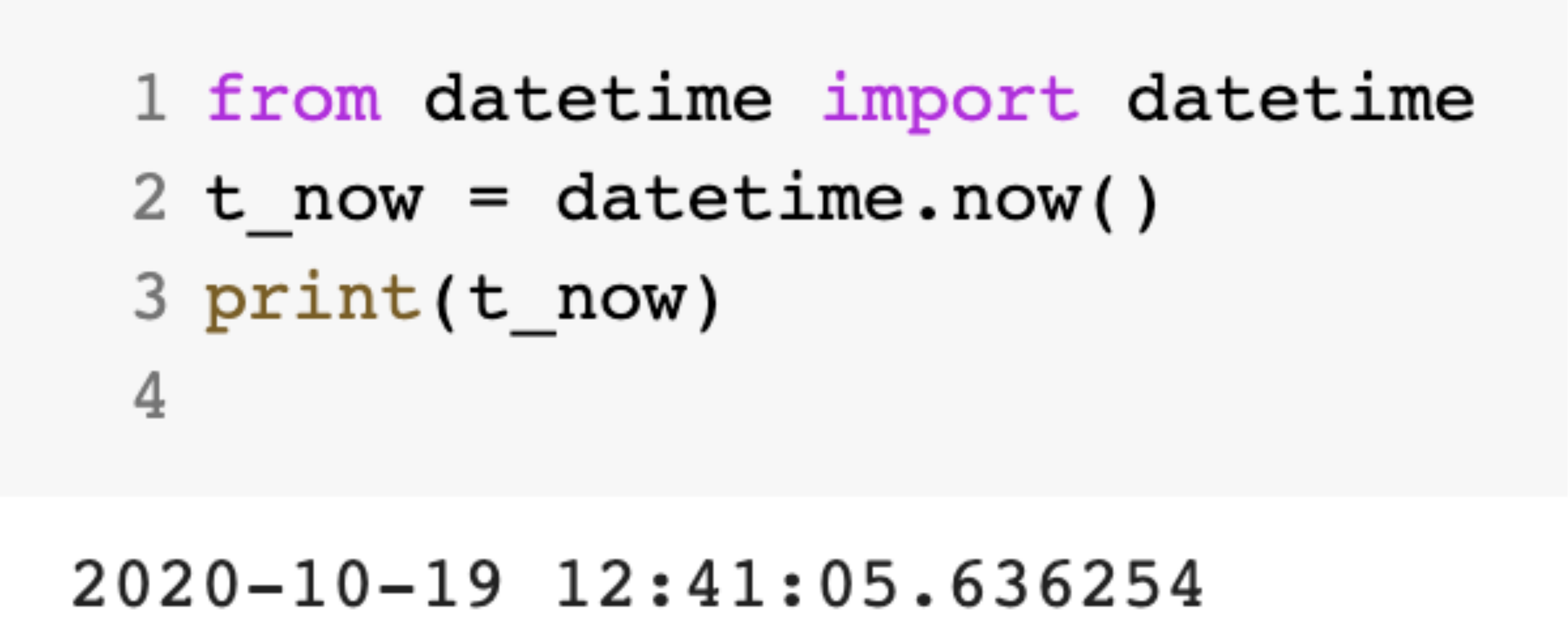
String datetime formatting



Directive	Meaning
`%a`	Weekday as locale’s abbreviated name.
`%A`	Weekday as locale’s full name.
`%w`	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.
`%d`	Day of the month as a zero-padded decimal number.
`%b`	Month as locale’s abbreviated name.
`%B`	Month as locale’s full name.
`%m`	Month as a zero-padded decimal number.
`%y`	Year without century as a zero-padded decimal number.
`%Y`	Year with century as a decimal number.
`%H`	Hour (24-hour clock) as a zero-padded decimal number.
`%I`	Hour (12-hour clock) as a zero-padded decimal number.
`%p`	Locale’s equivalent of either AM or PM.
`%M`	Minute as a zero-padded decimal number.
`%S`	Second as a zero-padded decimal number.
`%f`	Microsecond as a decimal number, zero-padded on the left.
`%z`	UTC offset in the form +HHMM or -HHMM (empty string if the the object is naive).
`%Z`	Time zone name (empty string if the object is naive).
`%j`	Day of the year as a zero-padded decimal number.
`%U`	Week number of the year (Sunday as the first day of the week) as a zero padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0.
`%W`	Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0.
`%c`	Locale’s appropriate date and time representation.
`%x`	Locale’s appropriate date representation.
`%X`	Locale’s appropriate time representation.
`%%`	A literal `'%'` character.

How to deal with time passing

datetime objects are snapshots of a specific time

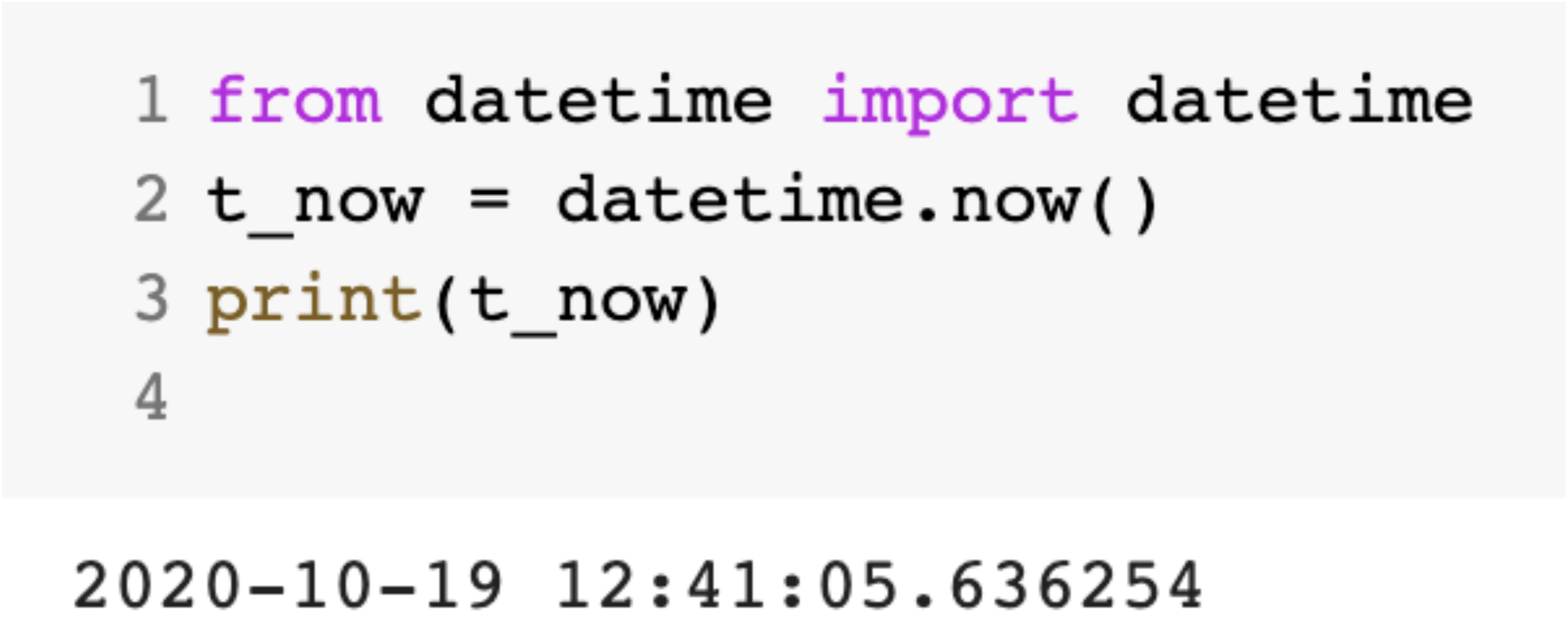


```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

How to deal with time passing

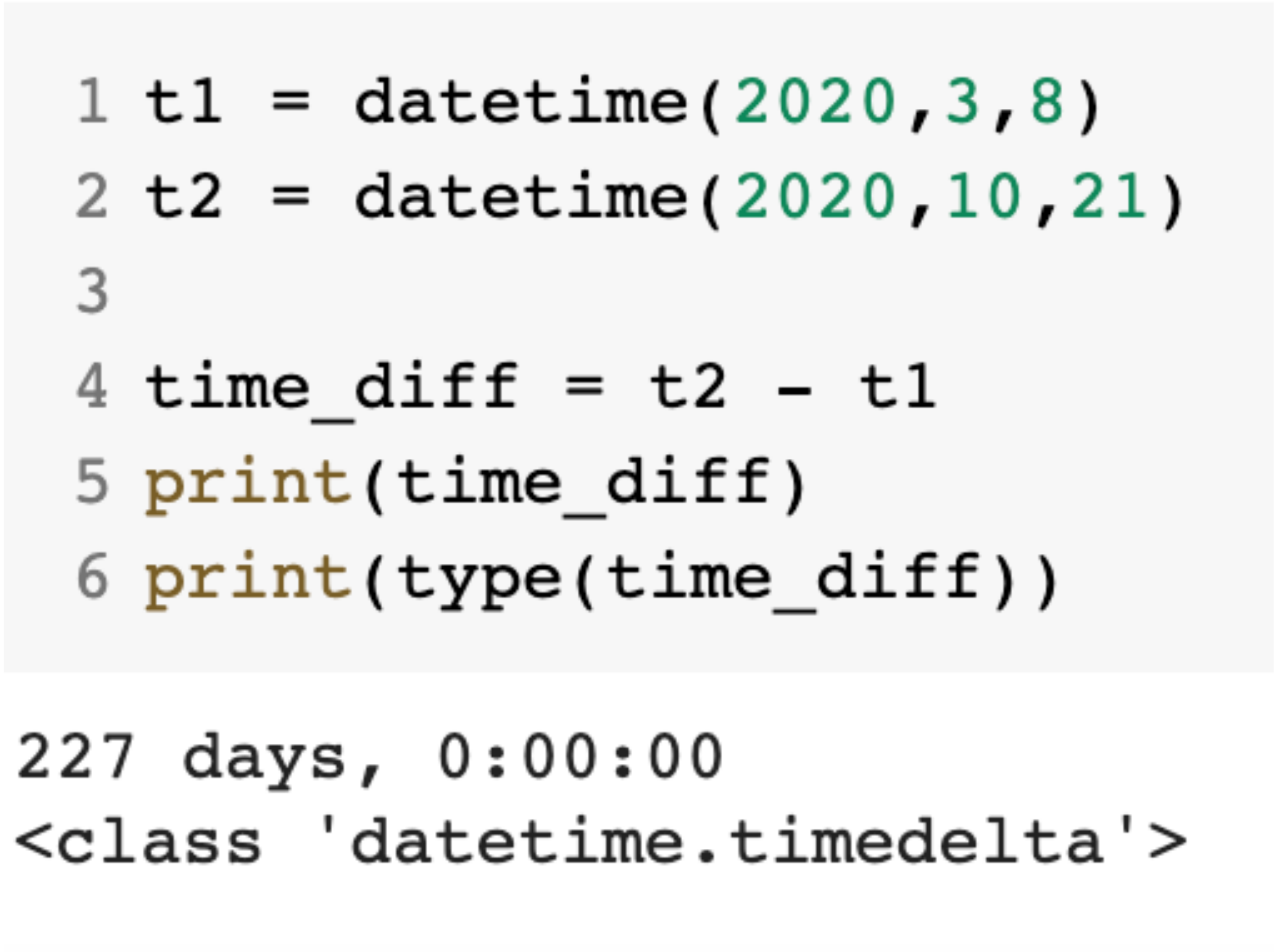
datetime objects are snapshots of a specific time



```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

To reflect time passing, use timedelta objects

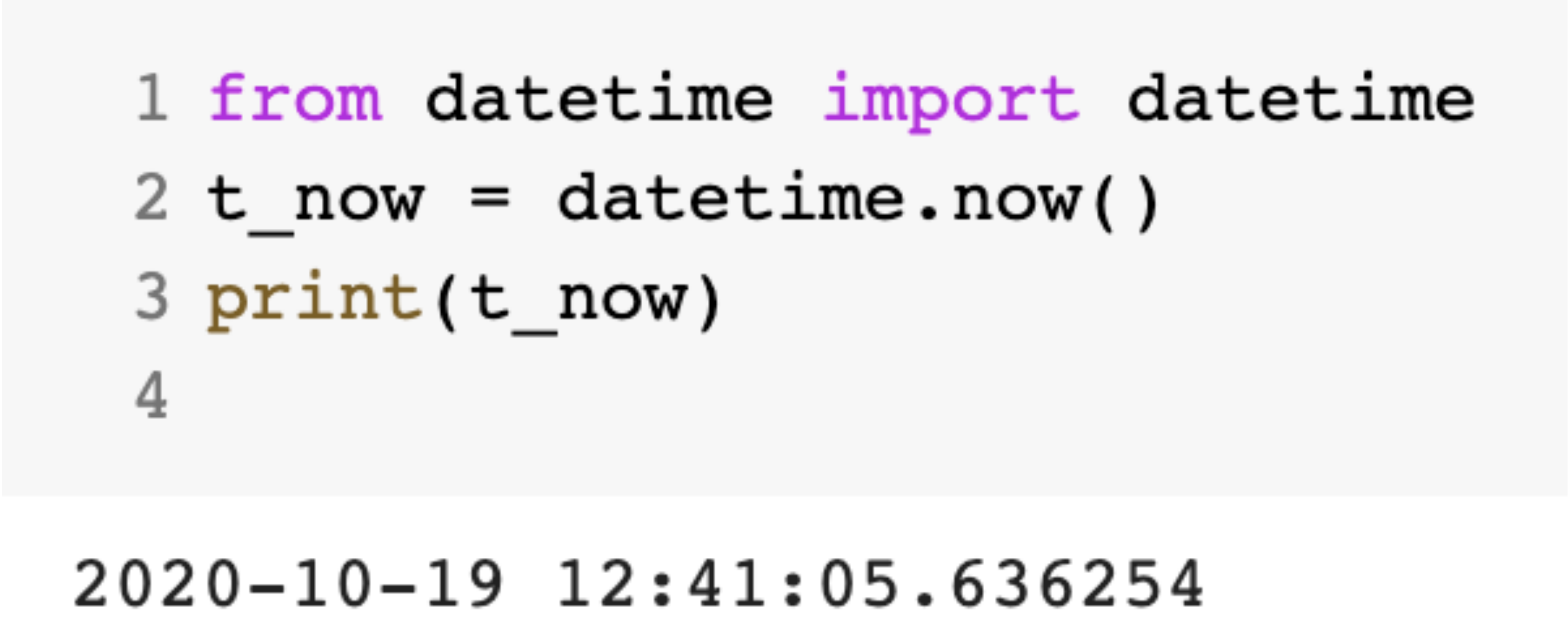


```python
t1 = datetime(2020,3,8)
t2 = datetime(2020,10,21)

time_diff = t2 - t1
print(time_diff)
print(type(time_diff))
```

```
227 days, 0:00:00
<class 'datetime.timedelta'>
```

How to deal with time passing

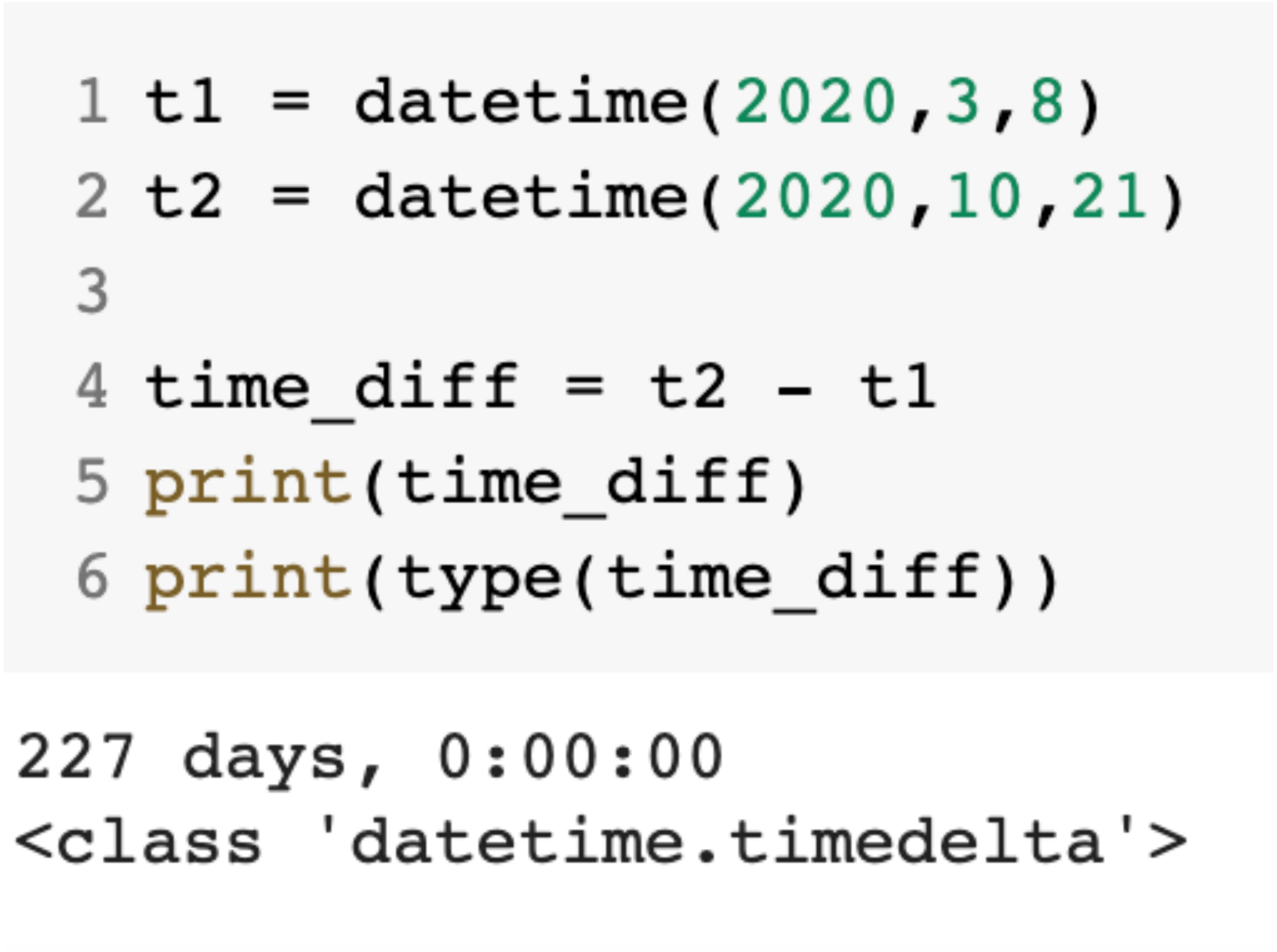
datetime objects are snapshots of a specific time



```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

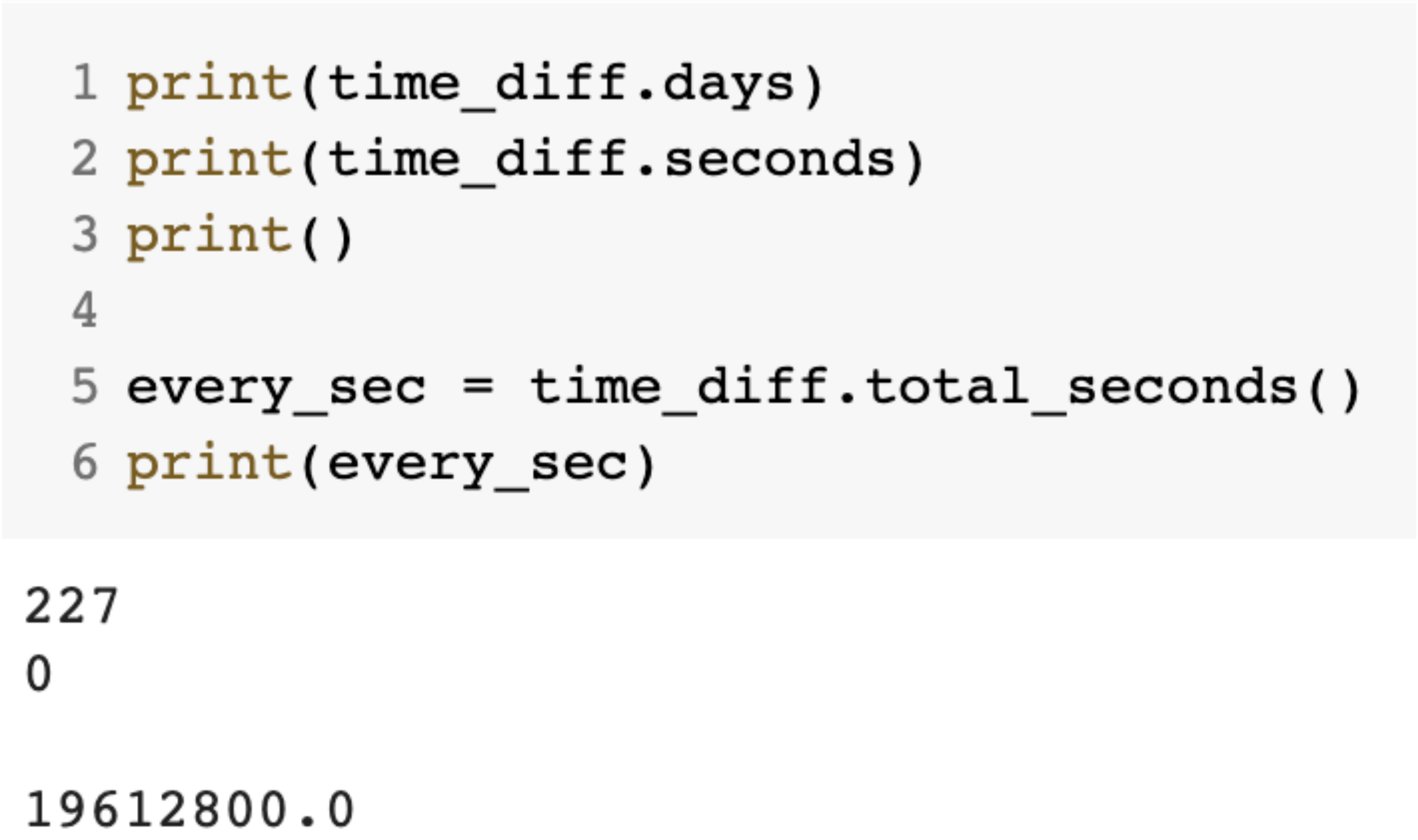
To reflect time passing, use timedelta objects



```python
t1 = datetime(2020,3,8)
t2 = datetime(2020,10,21)

time_diff = t2 - t1
print(time_diff)
print(type(time_diff))
```

```
227 days, 0:00:00
<class 'datetime.timedelta'>
```

Retrieve the individual values from the timedelta object

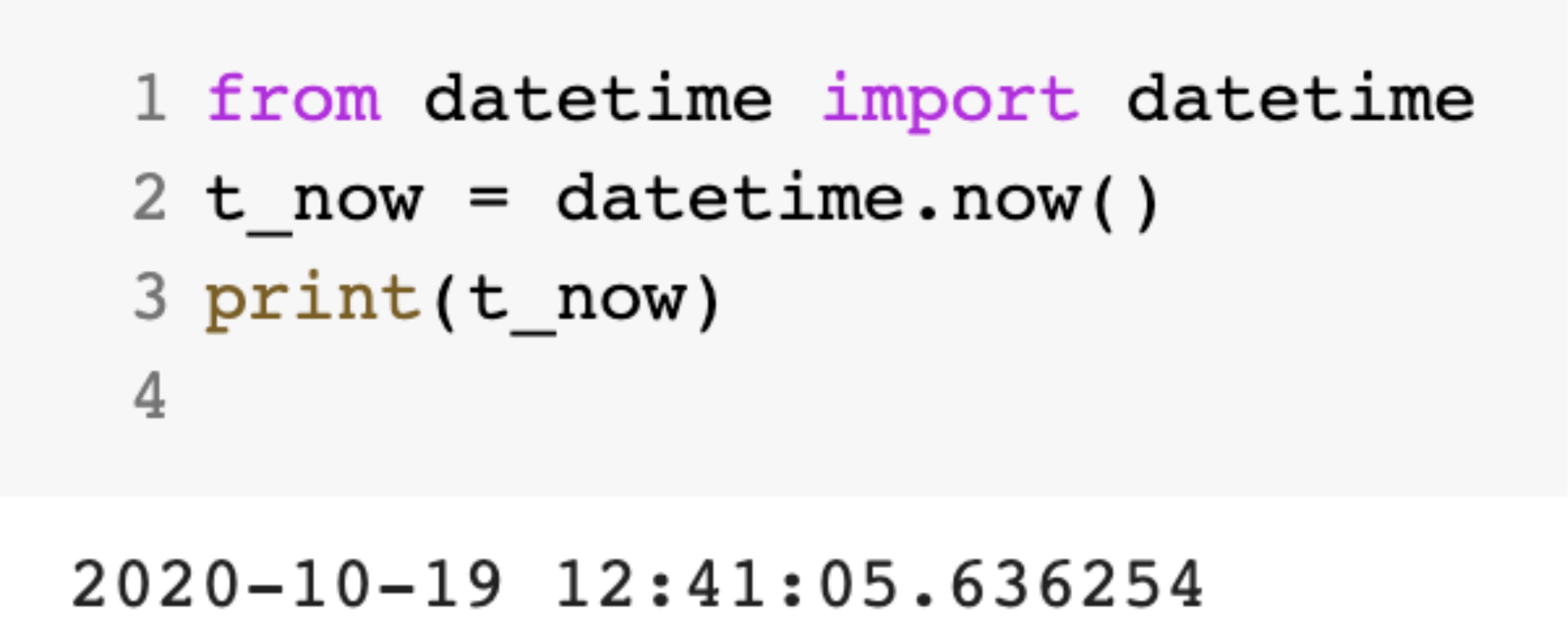


```python
print(time_diff.days)
print(time_diff.seconds)
print()

every_sec = time_diff.total_seconds()
print(every_sec)
```

```
227
0

19612800.0
```

How to deal with time passing

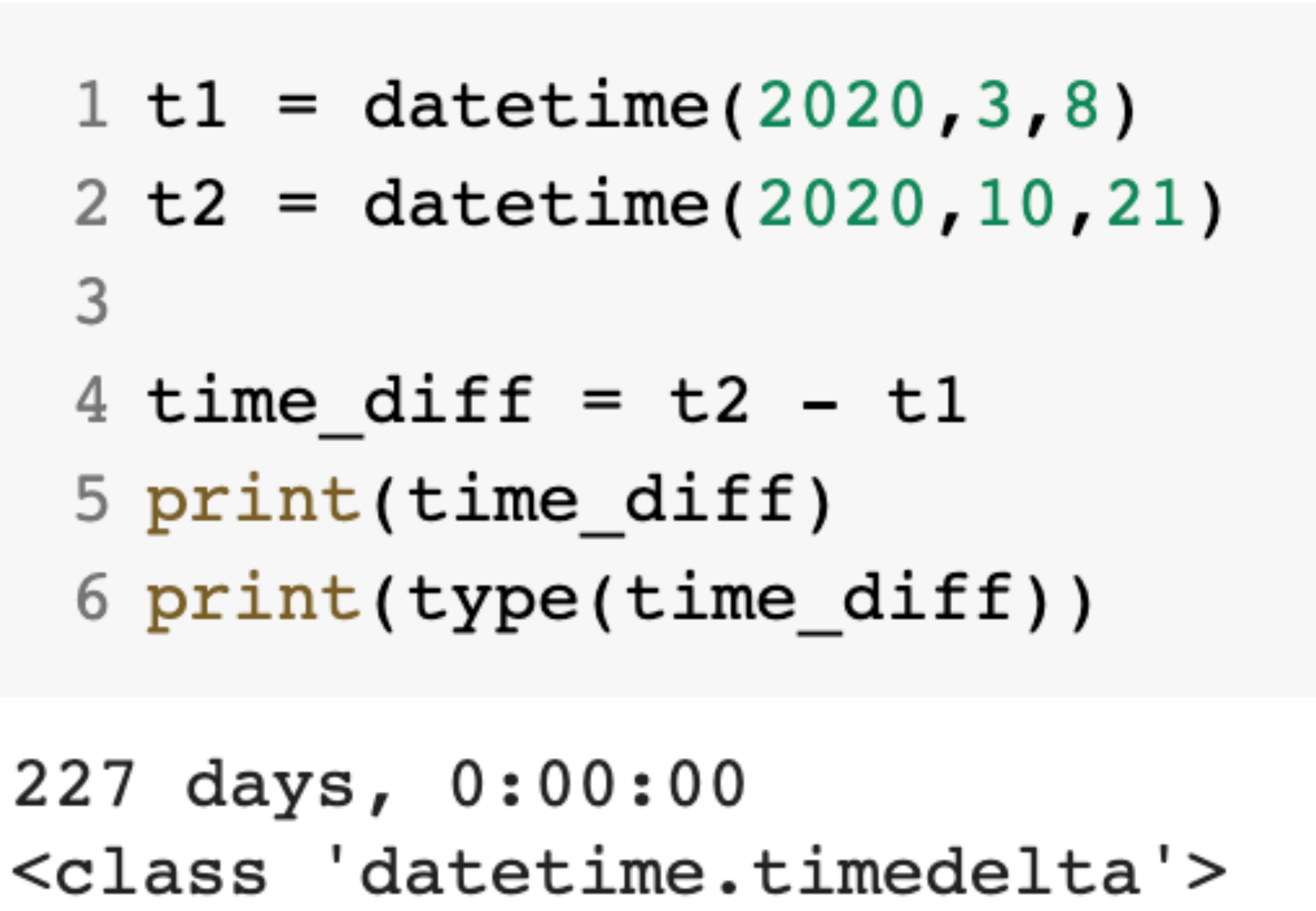
datetime objects are snapshots of a specific time



```python
from datetime import datetime
t_now = datetime.now()
print(t_now)

```

```
2020-10-19 12:41:05.636254
```

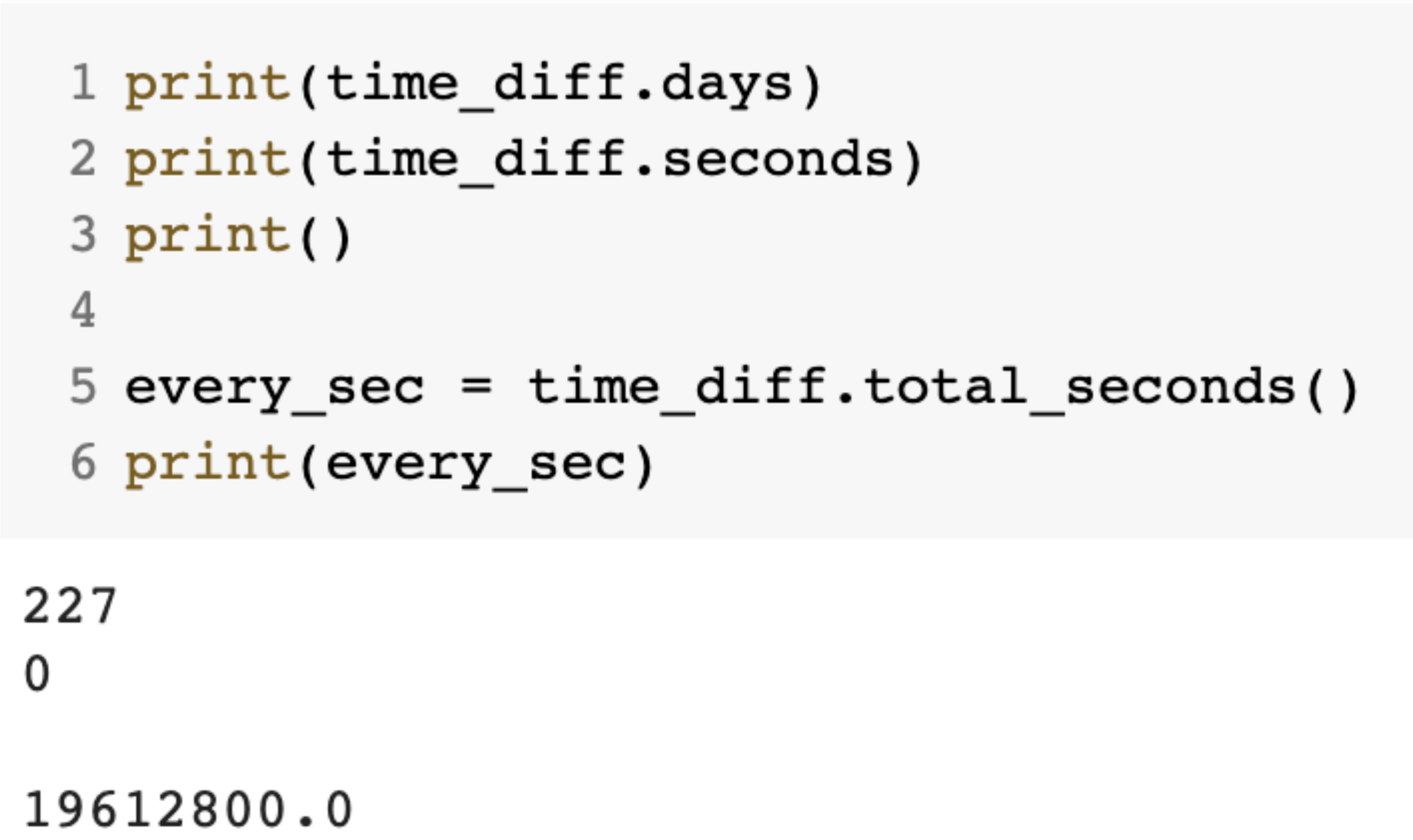
To reflect time passing, use timedelta objects



```python
t1 = datetime(2020,3,8)
t2 = datetime(2020,10,21)

time_diff = t2 - t1
print(time_diff)
print(type(time_diff))
```

```
227 days, 0:00:00
<class 'datetime.timedelta'>
```

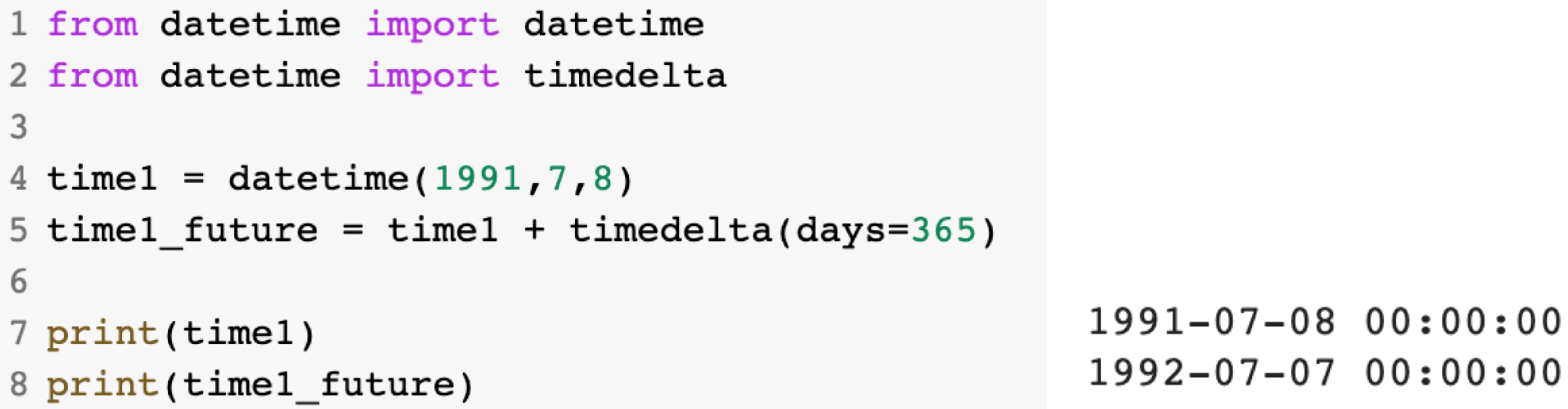
Retrieve the individual values from the timedelta object



```python
print(time_diff.days)
print(time_diff.seconds)
print()

every_sec = time_diff.total_seconds()
print(every_sec)
```

```
227
0

19612800.0
```

A timedelta object can alter a datetime object

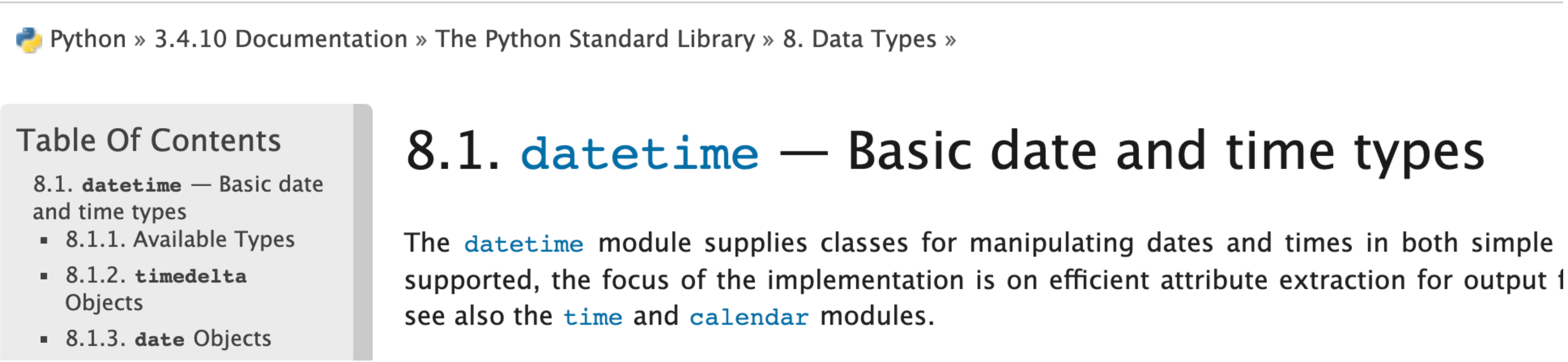


```python
from datetime import datetime
from datetime import timedelta

time1 = datetime(1991,7,8)
time1_future = time1 + timedelta(days=365)

print(time1)
print(time1_future)
```

```
1991-07-08 00:00:00
1992-07-07 00:00:00
```

Datetime resource

https://docs.python.org/3.4/library/datetime.html



Python » 3.4.10 Documentation » The Python Standard Library » 8. Data Types »

Table Of Contents

8.1. `datetime` — Basic date and time types

The `datetime` module supplies classes for manipulating dates and times in both simple
supported, the focus of the implementation is on efficient attribute extraction for output f
see also the `time` and `calendar` modules.