Introduction to Python

Python Basics

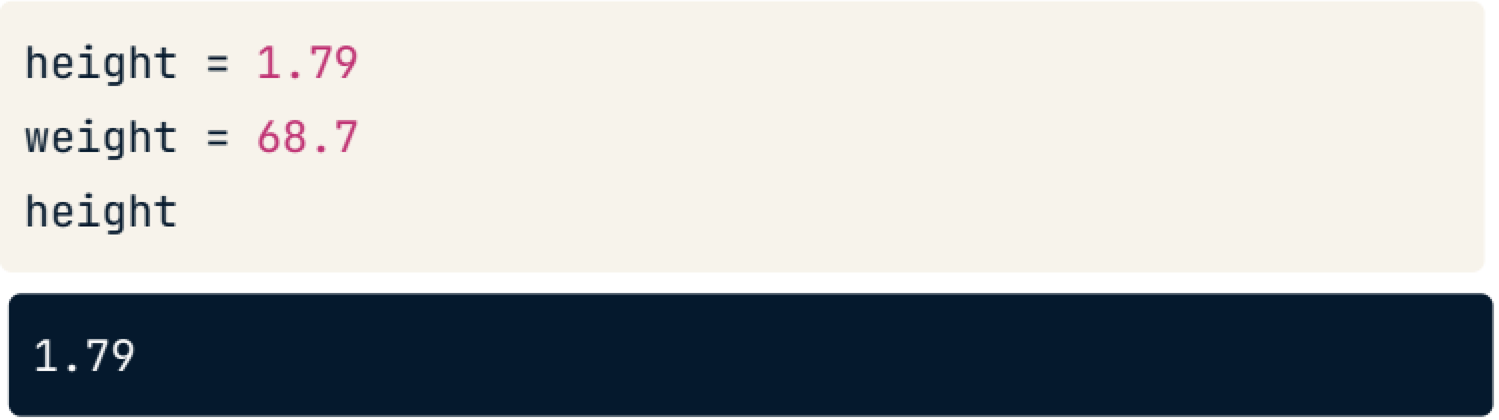
Variables and Calculators

Python as a calculator

Row	Sign	Description
1	+	Addition
2	-	Subtraction
3	*	Multiplication
4	/	Division
5	//	Floordiv
6	%	Modulo
7	**	Exponentiation

Variable

- ✓ Specifc, case-sensitive name
- ✓ Call up value through variable name
- ✓ 1.79 m - 68.7 kg



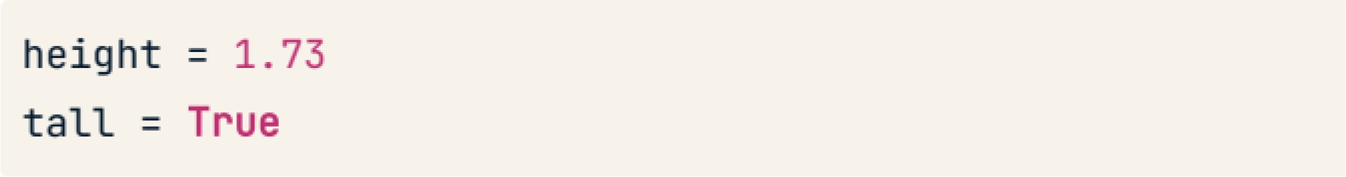
```python
height = 1.79
weight = 68.7
height
```

```
1.79
```

Python Lists

Python Data Types

- **int,** or integer: a number without a fractional part.
- **float,** or floating point: a number that has both an integer and fractional part, separated by a point.
- **str,** or string: a type to represent text. You can use single or double quotes to build a string.
- **bool,** or boolean: a type to represent logical values. Can only be True or False (the capitalization is important!).

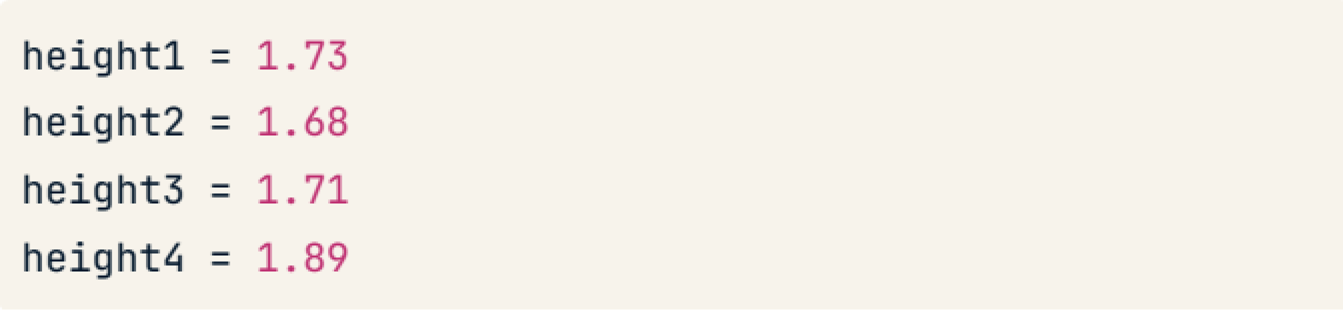


```
height = 1.73
tall = True
```

- ✓ Each variable represents single value

Problem

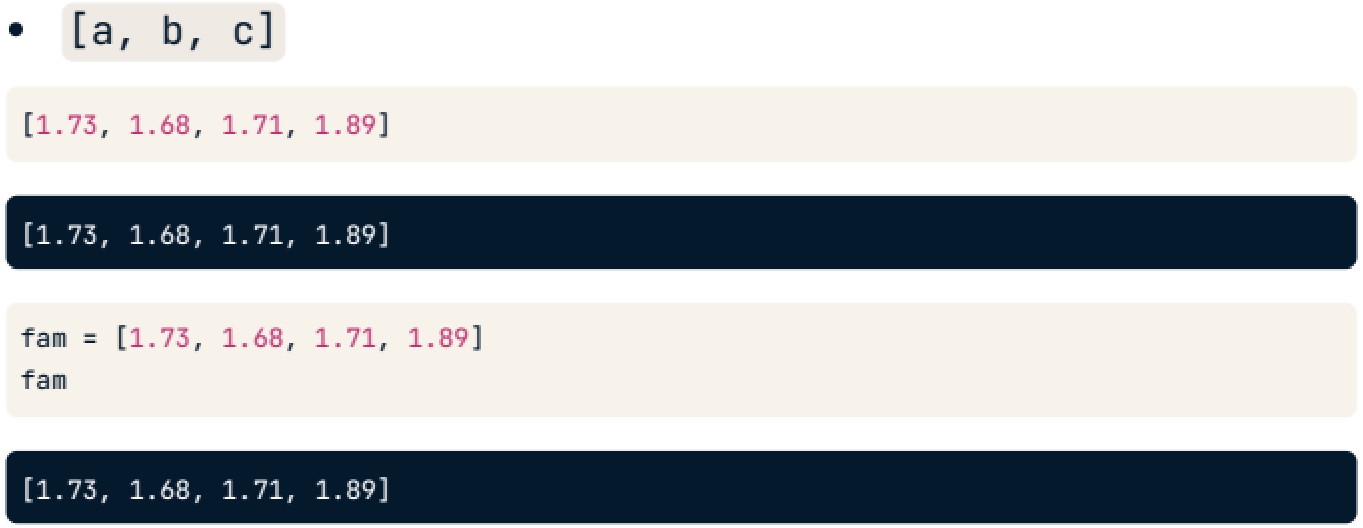
- ✓Data Science: many data points
- ✓Height of entire family



```
height1 = 1.73
height2 = 1.68
height3 = 1.71
height4 = 1.89
```

- ✓ Inconvenient

Python List



- `[a, b, c]`

```python
[1.73, 1.68, 1.71, 1.89]
```

```
[1.73, 1.68, 1.71, 1.89]
```

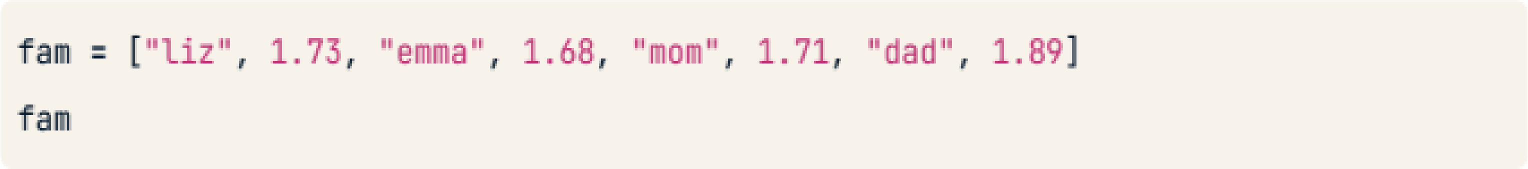
```python
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

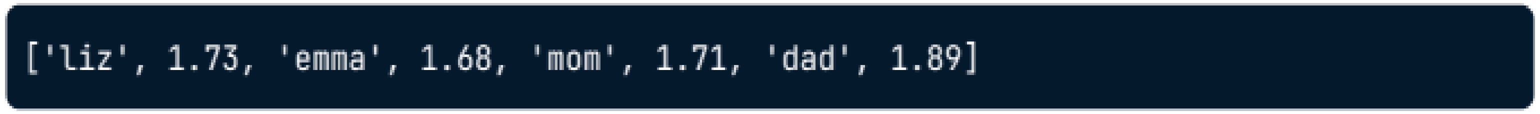
- ➢Name a collection of values
- ➢Contain any type
- ➢Contain different types

Python List

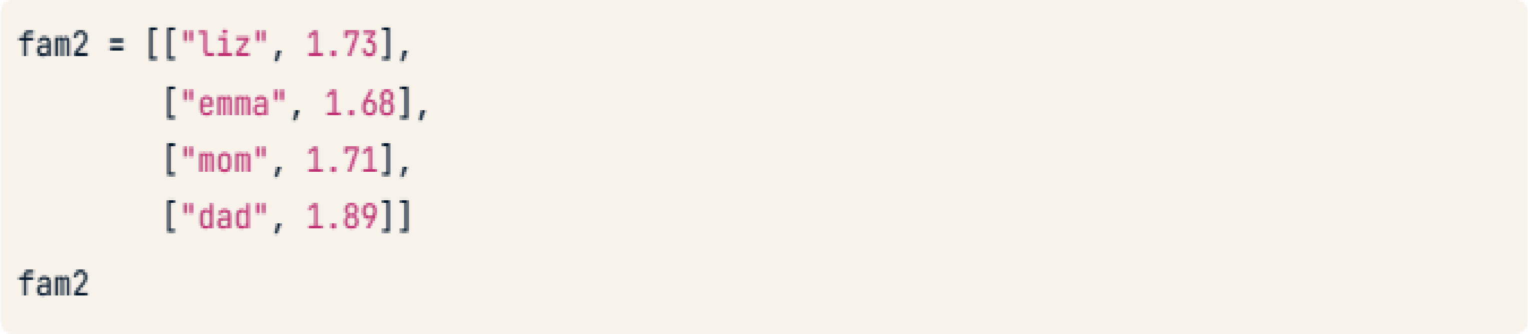
- `[a, b, c]`



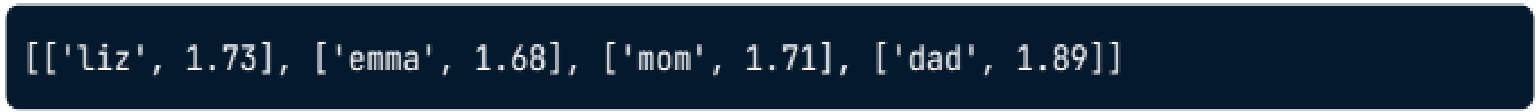
```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```



```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

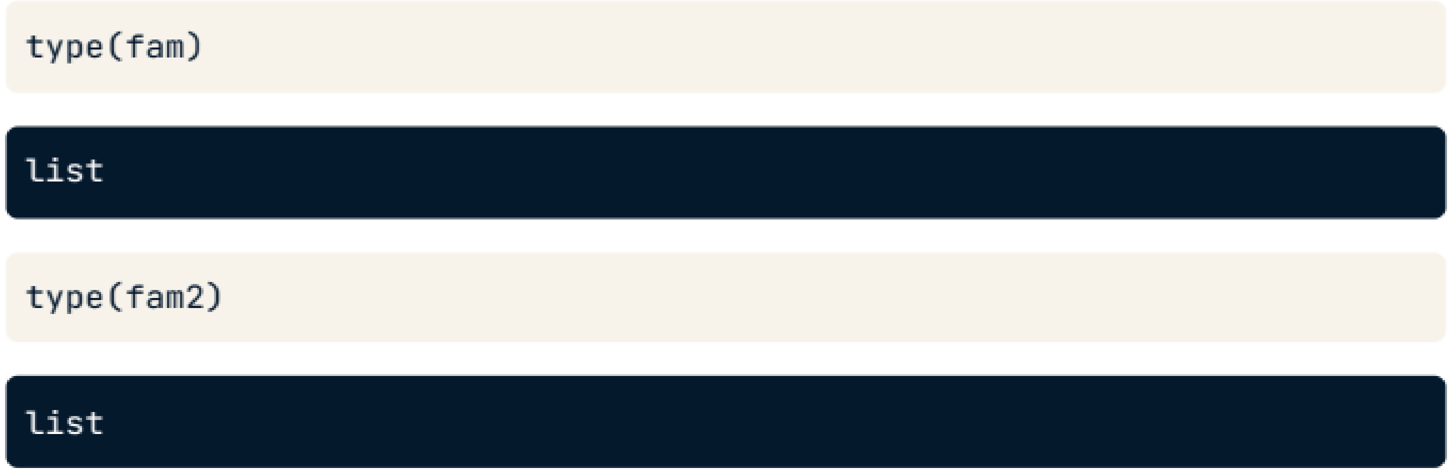


```
fam2 = [["liz", 1.73],
        ["emma", 1.68],
        ["mom", 1.71],
        ["dad", 1.89]]
fam2
```



```
[['liz', 1.73], ['emma', 1.68], ['mom', 1.71], ['dad', 1.89]]
```

List type



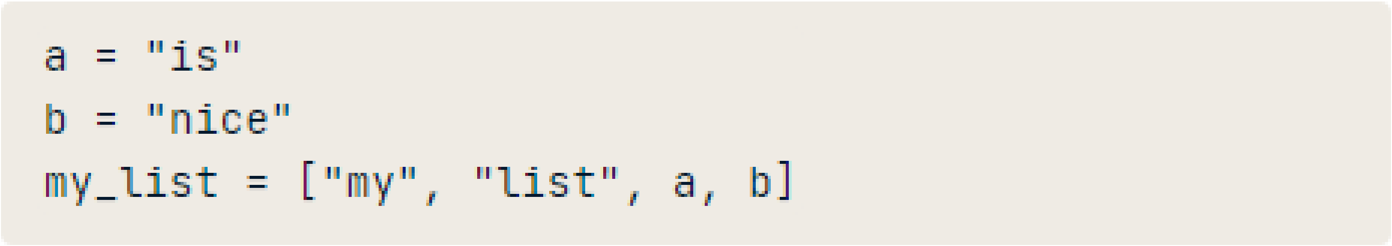
```
type(fam)
```

```
list
```

```
type(fam2)
```

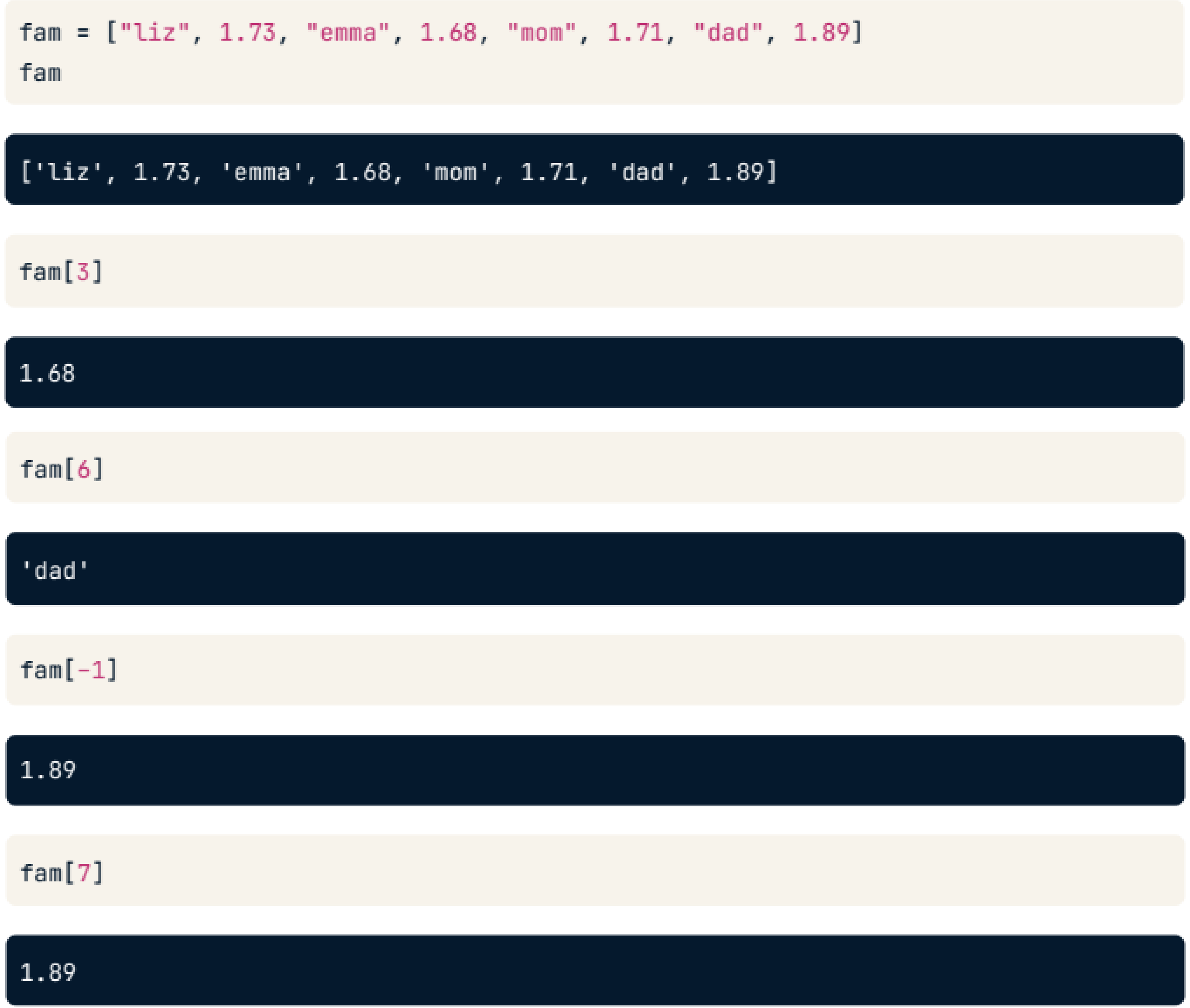
```
list
```

✓ As opposed to int, bool etc., a list is a compound data type; you can group values together:



```
a = "is"
b = "nice"
my_list = ["my", "list", a, b]
```

Sub setting Lists



```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3]
```

```
1.68
```

```
fam[6]
```

```
'dad'
```

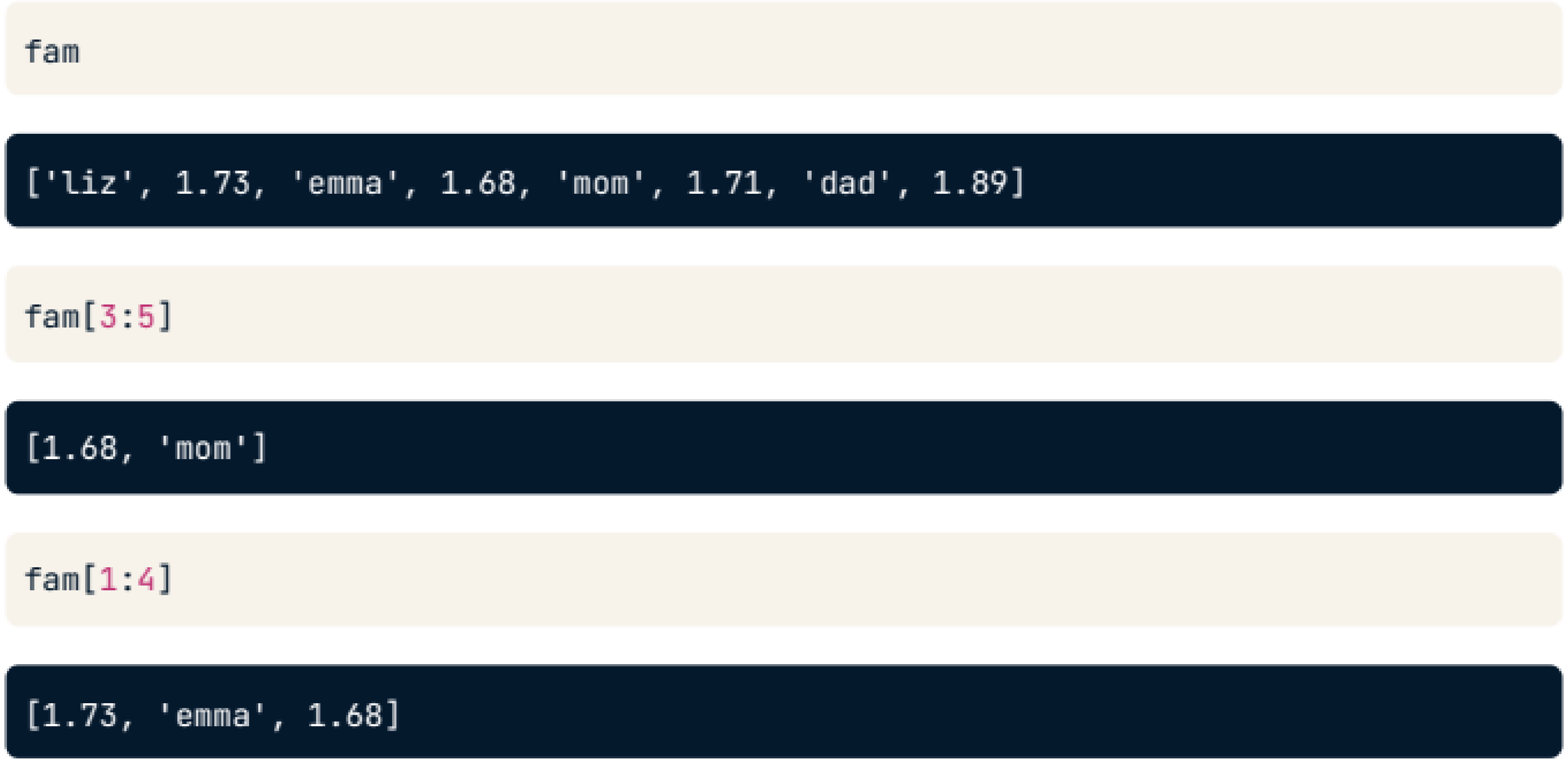
```
fam[-1]
```

```
1.89
```

```
fam[7]
```

```
1.89
```

List slicing



```
fam
```

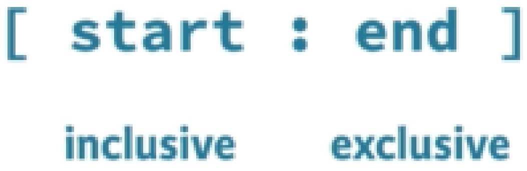
```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

```
[1.68, 'mom']
```

```
fam[1:4]
```

```
[1.73, 'emma', 1.68]
```

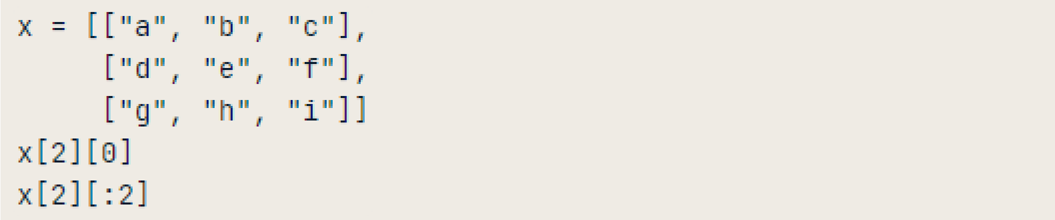


```
[ start : end ]
```

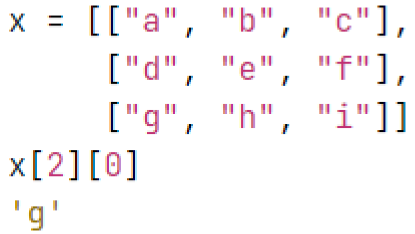
inclusive exclusive

Subsetting lists of lists

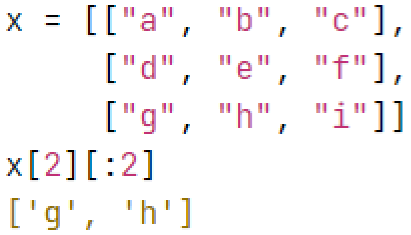
- You saw before that a Python list can contain practically anything; even other lists!
- To subset lists of lists, you can use the same technique as before: square brackets.
- Try out the commands in the following code sample in the IPython Shell:



```
x = [["a", "b", "c"],
     ["d", "e", "f"],
     ["g", "h", "i"]]
x[2][0]
x[2][:2]
```



```
x = [["a", "b", "c"],
     ["d", "e", "f"],
     ["g", "h", "i"]]
x[2][0]
'g'
```



```
x = [["a", "b", "c"],
     ["d", "e", "f"],
     ["g", "h", "i"]]
x[2][:2]
['g', 'h']
```

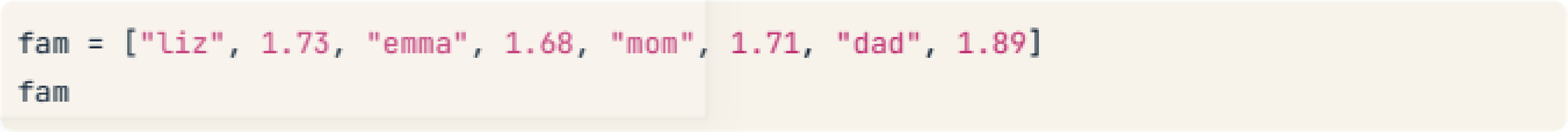
List Manipulation

✓Change list elements

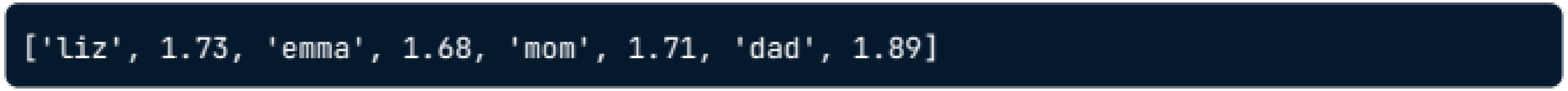
✓Add list elements

✓Remove list elements

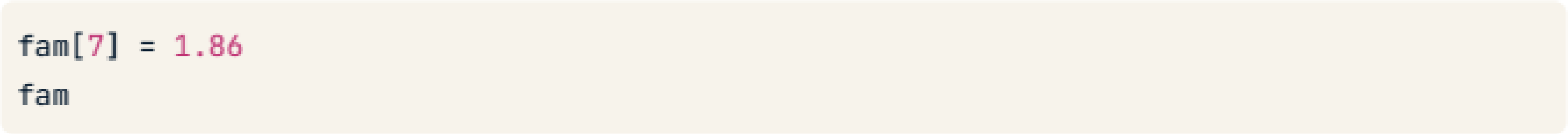
Changing list elements



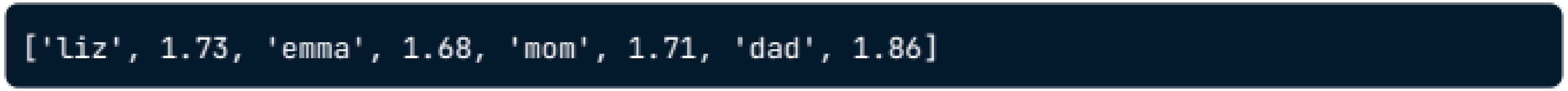
```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```



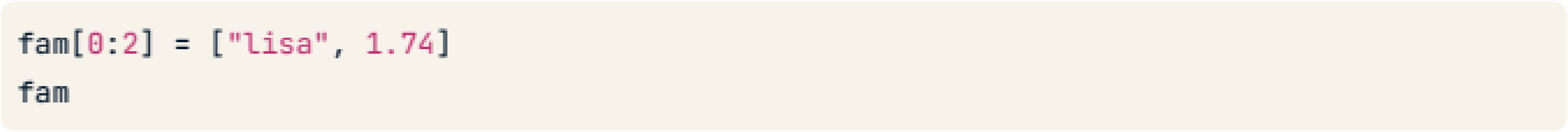
```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```



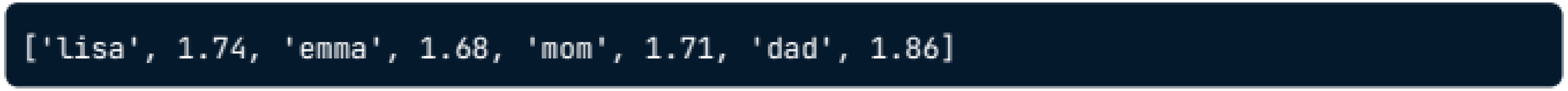
```
fam[7] = 1.86
fam
```



```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

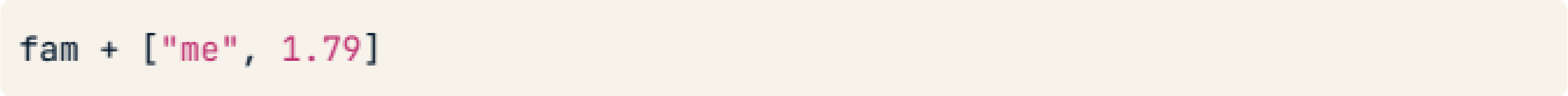


```
fam[0:2] = ["lisa", 1.74]
fam
```

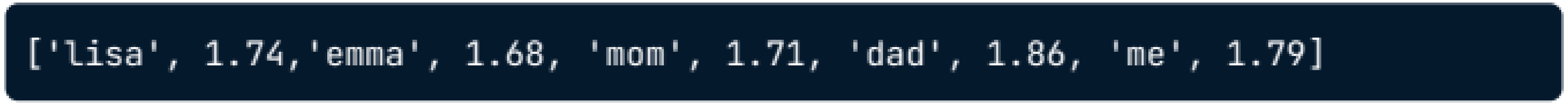


```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

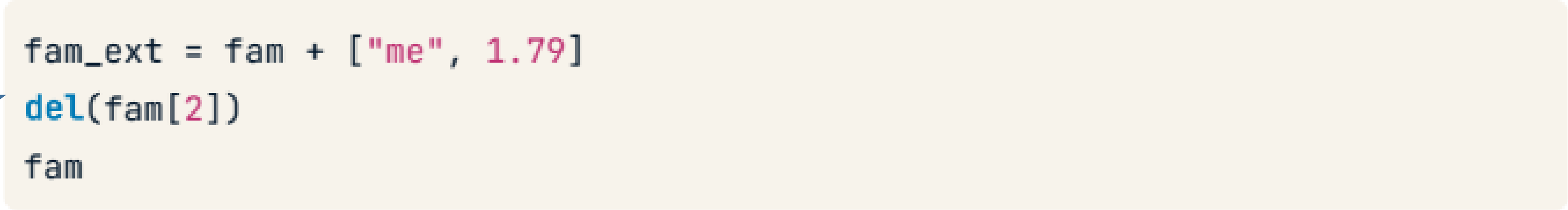
Adding and removing elements



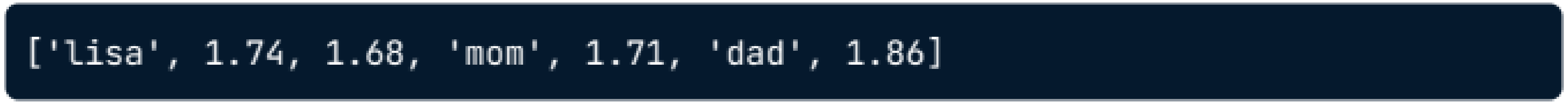
```python
fam + ["me", 1.79]
```



```
['lisa', 1.74,'emma', 1.68, 'mom', 1.71, 'dad', 1.86, 'me', 1.79]
```



```python
fam_ext = fam + ["me", 1.79]
del(fam[2])
fam
```



```
['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
```

Functions

- ✓ Nothing new!
- ✓ type() : To find out the type of a value or a variable that refers to that value, you can use the **type()** function.
- ✓ Functions such as **str()**, **int()**, **float()**, **list()** and **bool()** will help you convert Python values into any type.
- ✓ Piece of reusable code
- ✓ Solves particular task
- ✓ Call function instead of writing code yourself

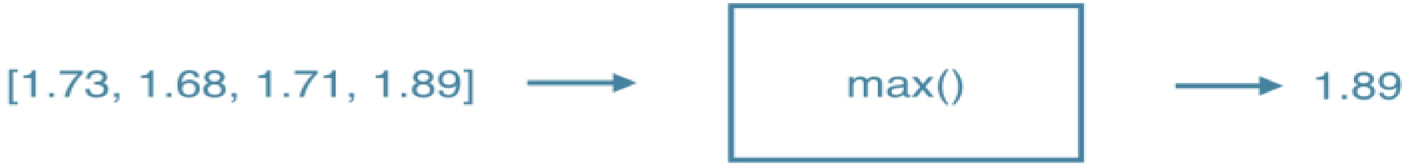
Example

```
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

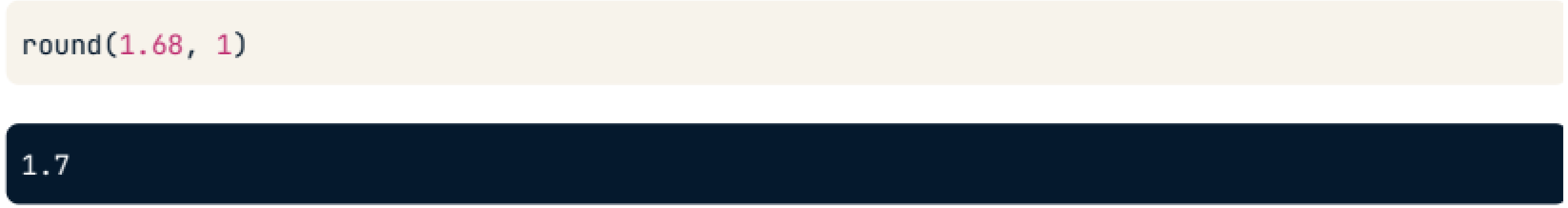
```
1.89
```



```
tallest = max(fam)
tallest
```

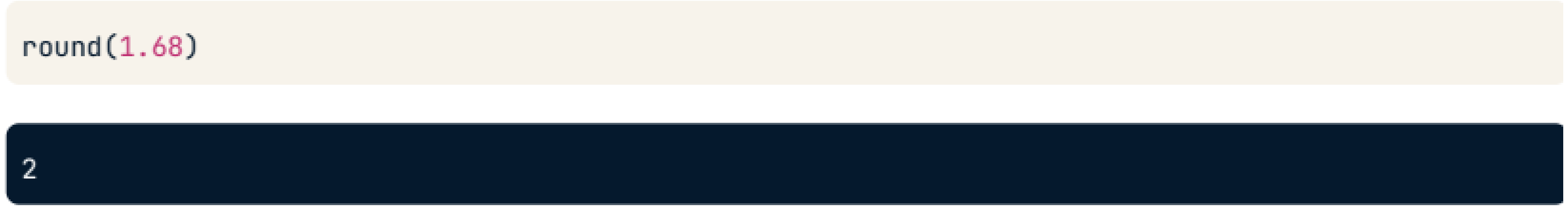
```
1.89
```

round()



```
round(1.68, 1)
```

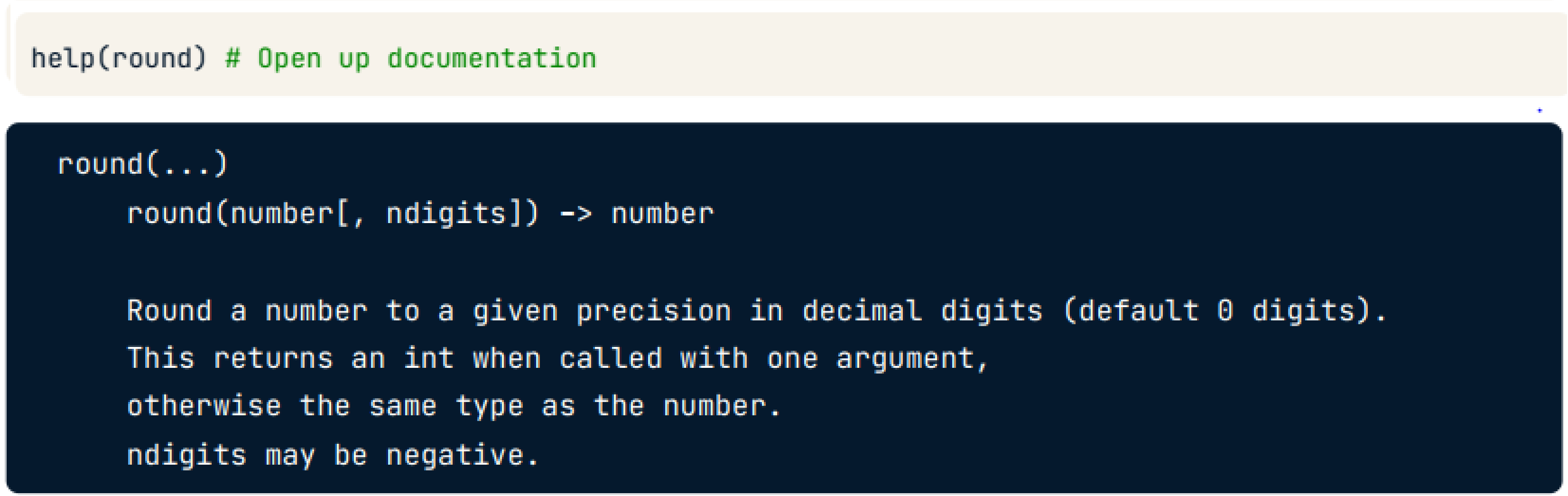
```
1.7
```



```
round(1.68)
```

```
2
```

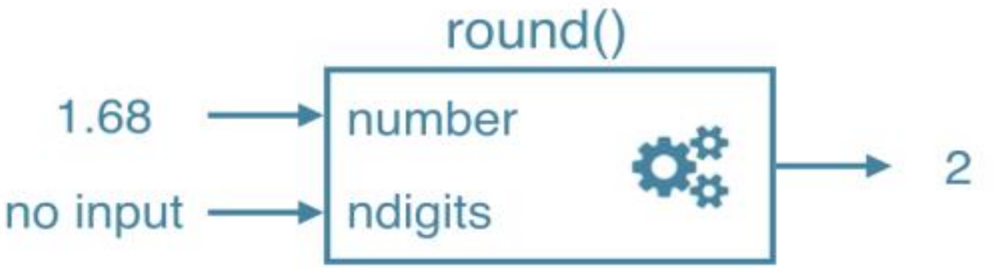
round()



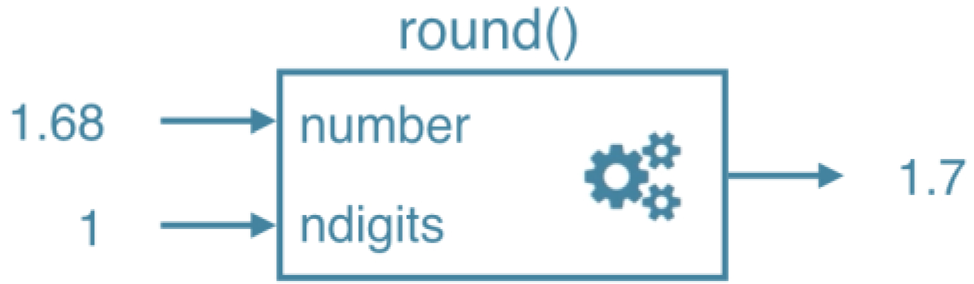
```
help(round) # Open up documentation
```

```
round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument,
    otherwise the same type as the number.
    ndigits may be negative.
```

round(1.68)



round(1.68, 1)



Find functions

- How to know?
- Standard task -> probably function exists!
- The internet is your friend

w3schools

datacamp

TutorialsTeacher

matplotlib
Version 3.4.2

Built-in Functions

✓Maximum of list: max()

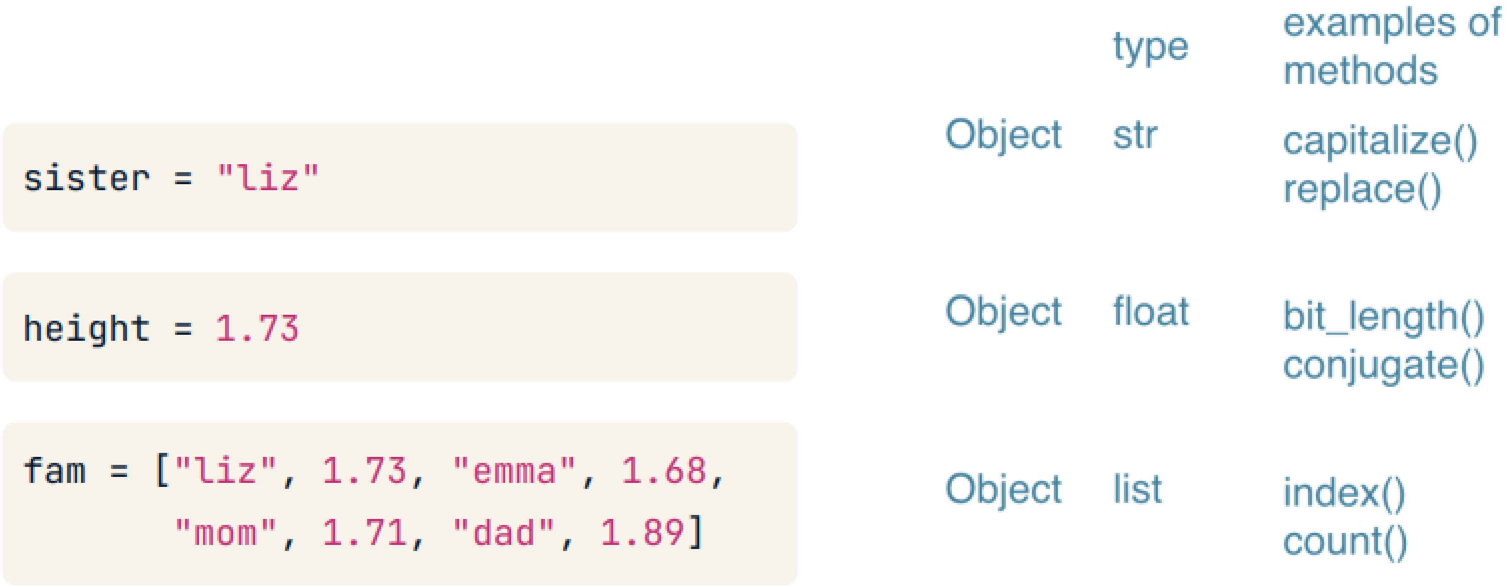
✓Length of list or string: len()

✓Get index in list: index()

✓Reversing a list: reverse

Methods

Methods



```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]
```

	type	examples of methods

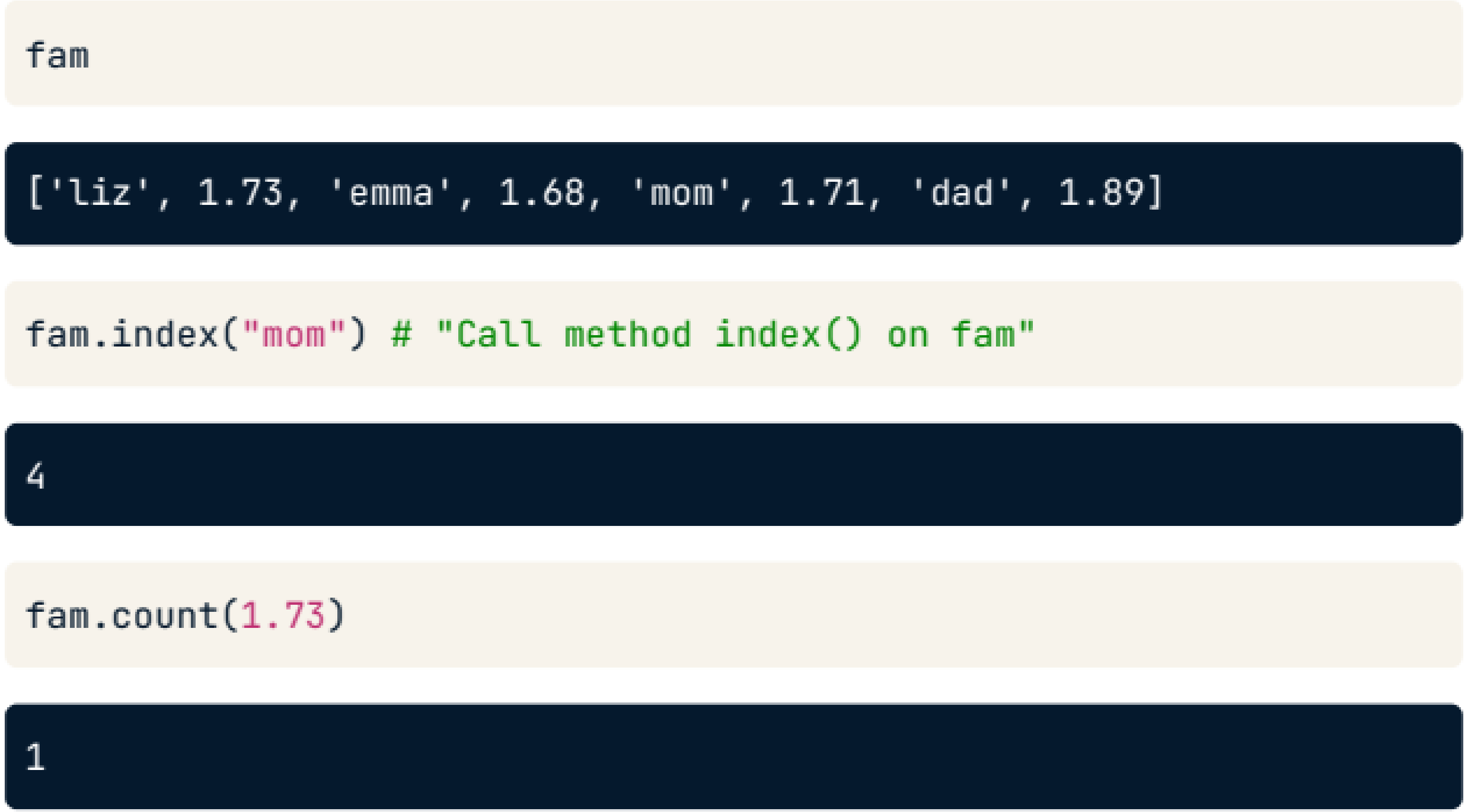
Object	str	capitalize()
		replace()

Object	float	bit_length()
		conjugate()

Object	list	index()
		count()

Methods: Functions that belong to objects

list methods



```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

```
4
```

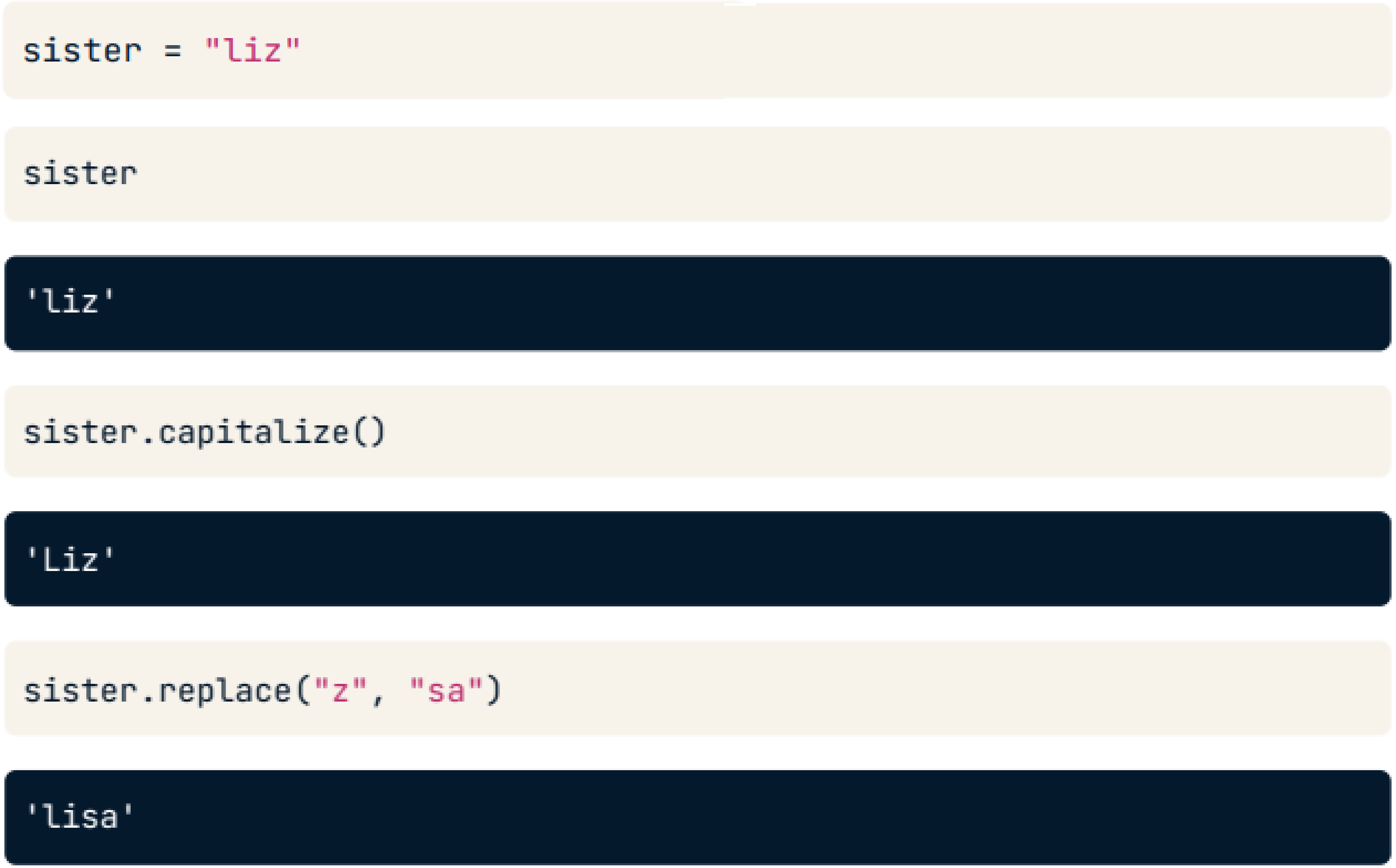
```
fam.count(1.73)
```

```
1
```

Syntax

list.Method(*value*)

str methods



```
sister = "liz"
```

```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

```
sister.replace("z", "sa")
```

```
'lisa'
```

Syntax

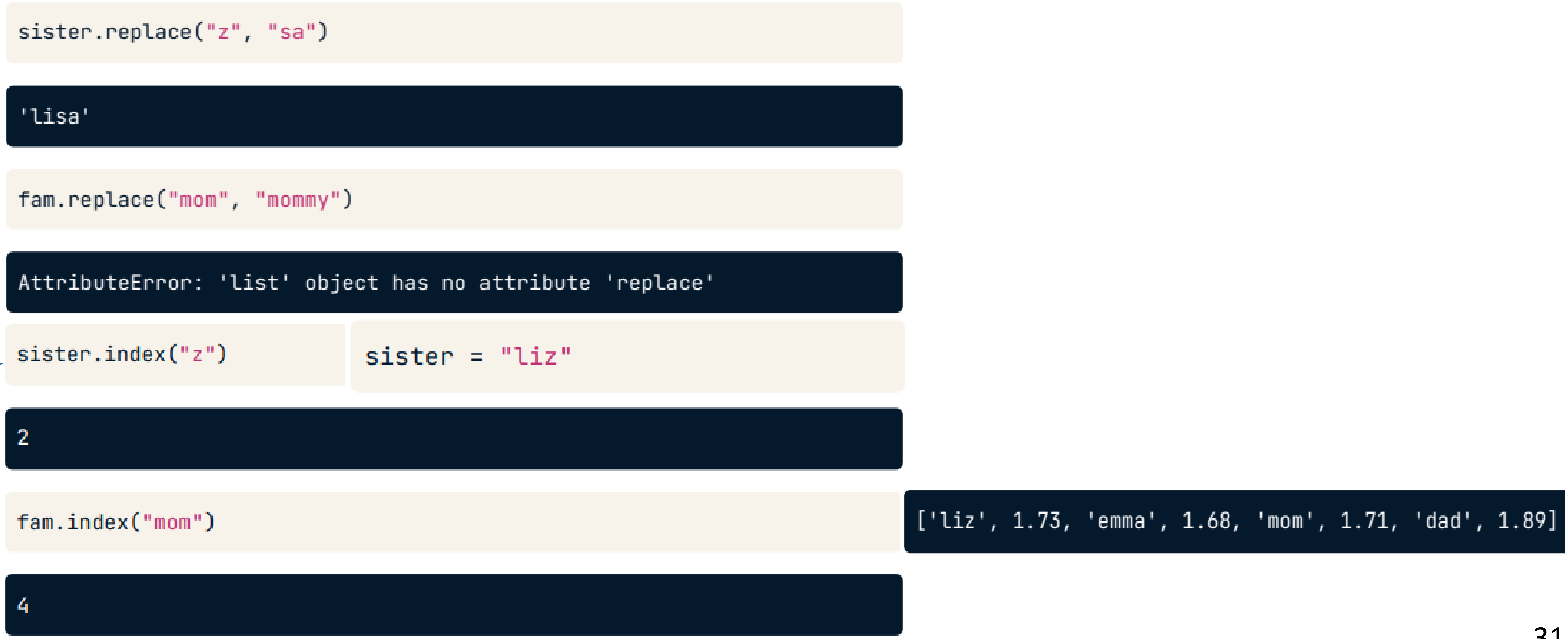
string.capitalize()

Syntax

string.replace(*oldvalue*, *newvalue*, *count*)

Methods

- Everything = object
- Object have methods associated, depending on type



```
sister.replace("z", "sa")
```

```
'lisa'
```

```
fam.replace("mom", "mommy")
```

```
AttributeError: 'list' object has no attribute 'replace'
```

```
sister.index("z")
```

```
sister = "liz"
```

```
2
```

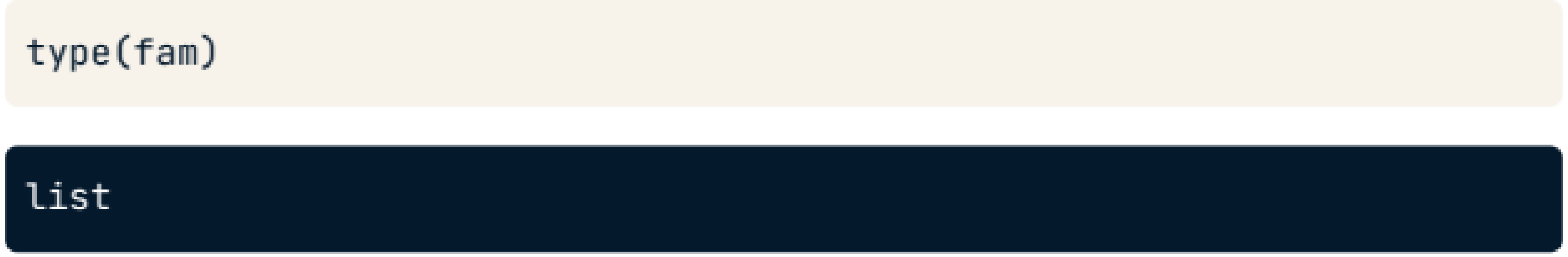
```
fam.index("mom")
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
4
```

Summary

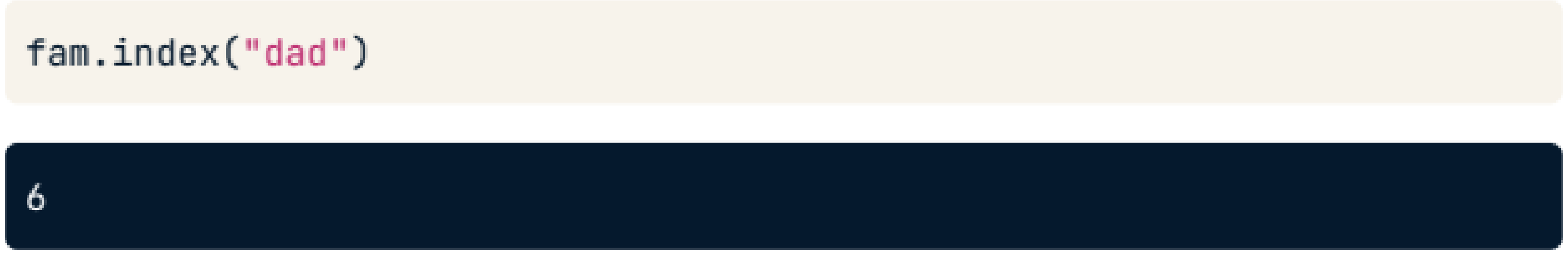
✓Functions



```
type(fam)
```

```
list
```

✓Methods: call functions on objects



```
fam.index("dad")
```

```
6
```

List Methods

- ✓**index()**, to get the index of the first element of a list that matches its input and
- ✓**count()**, to get the number of times an element appears in a list.
- ✓**append()**, that adds an element to the list it is called on,
- ✓**remove()**, that removes the first element of a list that matches the input, and
- ✓**reverse()**, that reverses the order of the elements in the list it is called on.

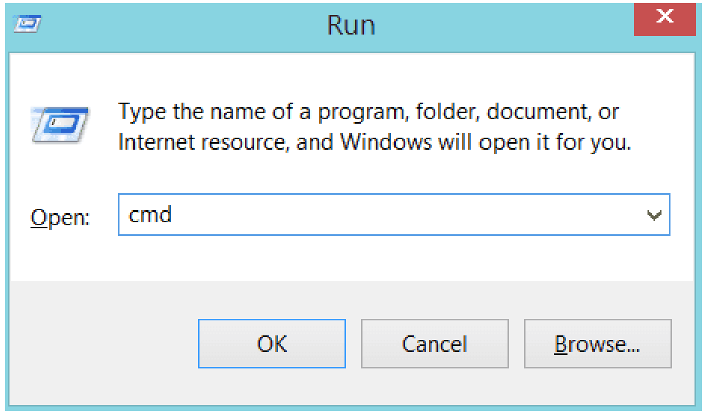
Packages

Packages

- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available
  - ✓ Numpy
  - ✓ Matplotlib
  - ✓ Scikit-learn

Install package

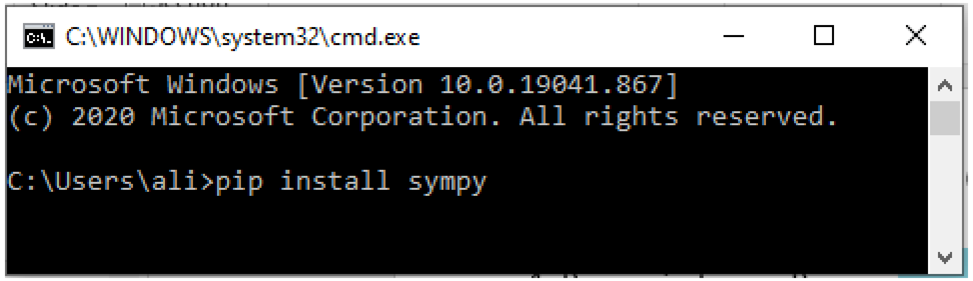
1. Press windows + R



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

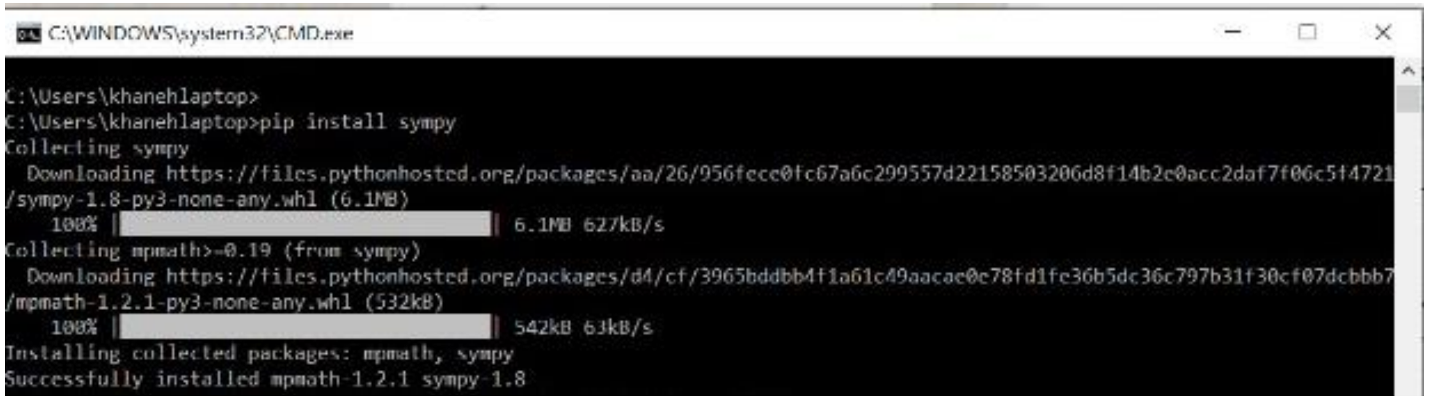
Open: cmd

2. Type “pip install **Package Name**” and press Enter



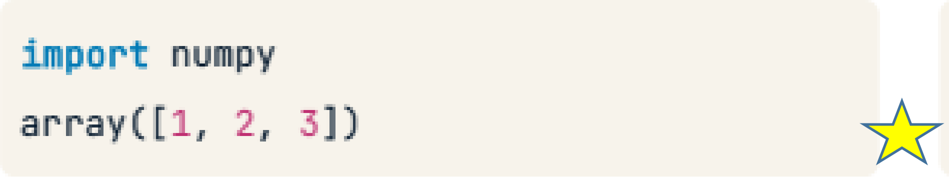
```
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\ali>pip install sympy
```

3. Wait to see the message “**successfully install**”

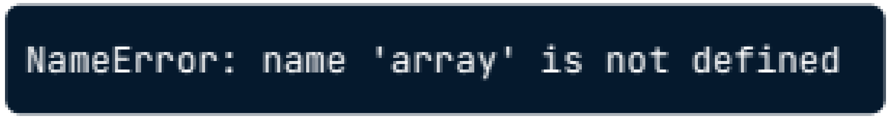


```
C:\Users\khanehlaptop>
C:\Users\khanehlaptop>pip install sympy
Collecting sympy
  Downloading https://files.pythonhosted.org/packages/aa/26/956fece0fc67a6c299557d22158503206d8f14b2e0acc2daf7f06c5f4721
/sympy-1.8-py3-none-any.whl (6.1MB)
    100% |████████████████████████████████| 6.1MB 627kB/s
Collecting mpmath>=0.19 (from sympy)
  Downloading https://files.pythonhosted.org/packages/d4/cf/3965bddbb4f1a61c49aacae0e78fd1fe36b5dc36c797b31f30cf07dcbbb7
/mpmath-1.2.1-py3-none-any.whl (532kB)
    100% |████████████████████████████████| 542kB 63kB/s
Installing collected packages: mpmath, sympy
Successfully installed mpmath-1.2.1 sympy-1.8
```

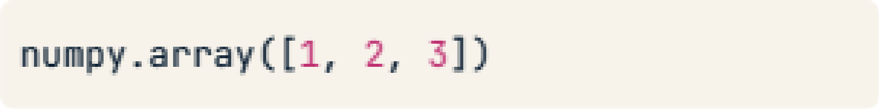
Import package



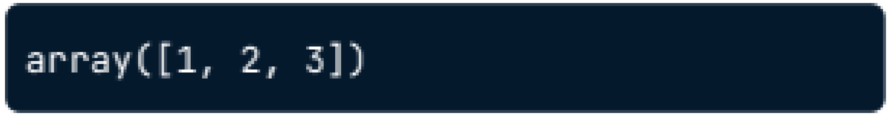
```python
import numpy
array([1, 2, 3])
```



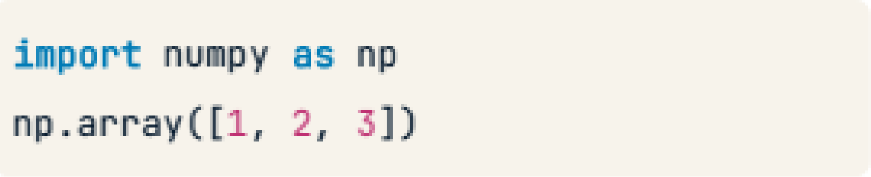
```
NameError: name 'array' is not defined
```



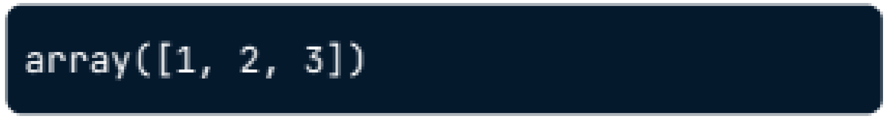
```python
numpy.array([1, 2, 3])
```



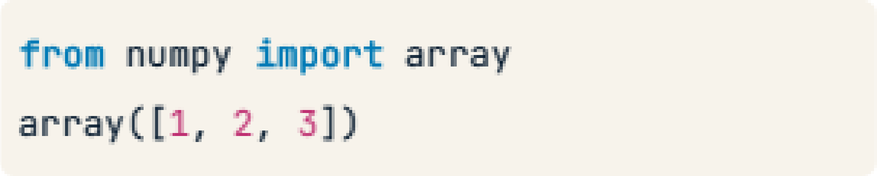
```
array([1, 2, 3])
```



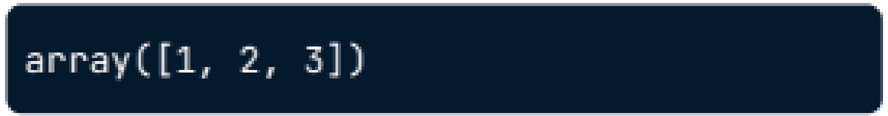
```python
import numpy as np
np.array([1, 2, 3])
```



```
array([1, 2, 3])
```



```python
from numpy import array
array([1, 2, 3])
```



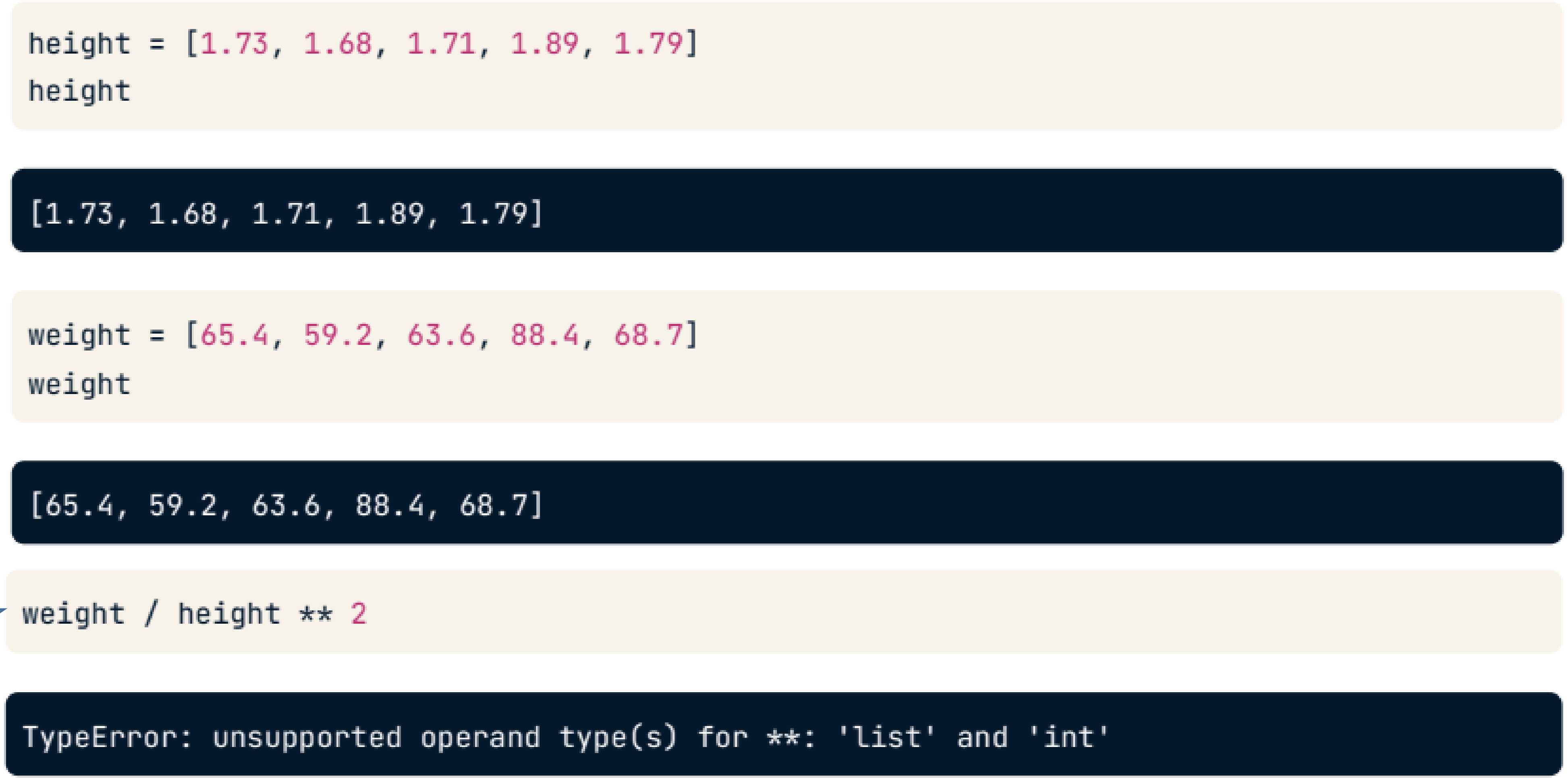
```
array([1, 2, 3])
```

Numpy

Lists Recap

- Powerful
- Collection of values
- Hold different types
- Change, add, remove
- Need for Data Science
    - ✓ Mathematical operations over collections
    - ✓ Speed

Illustration



```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
height
```

```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

```
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight
```

```
[65.4, 59.2, 63.6, 88.4, 68.7]
```

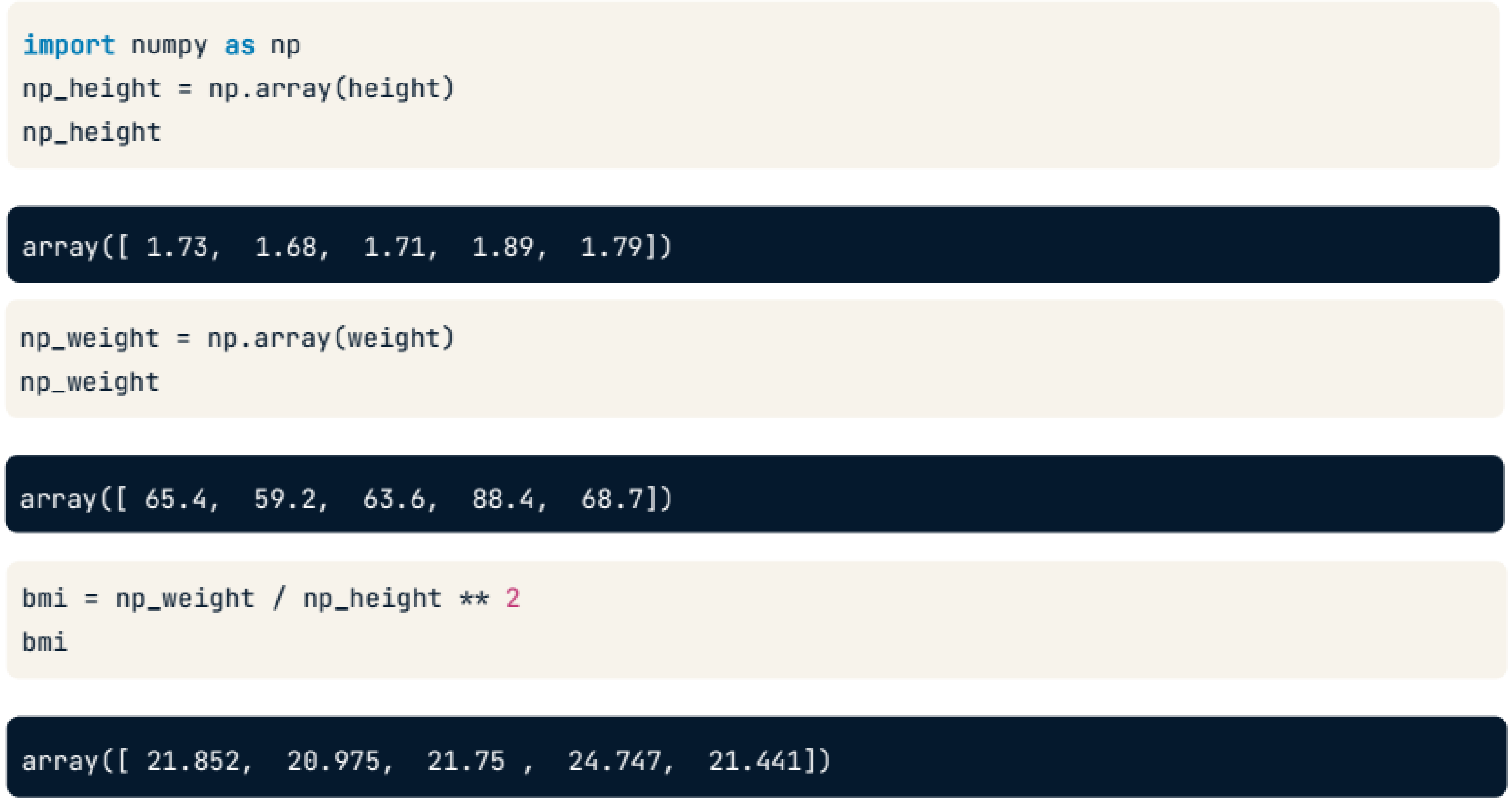
```
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for **: 'list' and 'int'
```

Solution: Numpy

- Numeric Python
- Alternative to Python List: Numpy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
  - ✓ In the terminal: pip3 install numpy

Numpy



```
import numpy as np
np_height = np.array(height)
np_height
```

```
array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
np_weight = np.array(weight)
np_weight
```

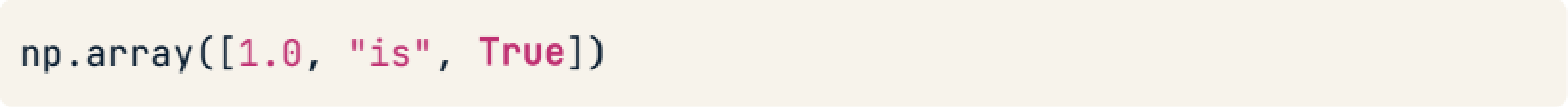
```
array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

☆

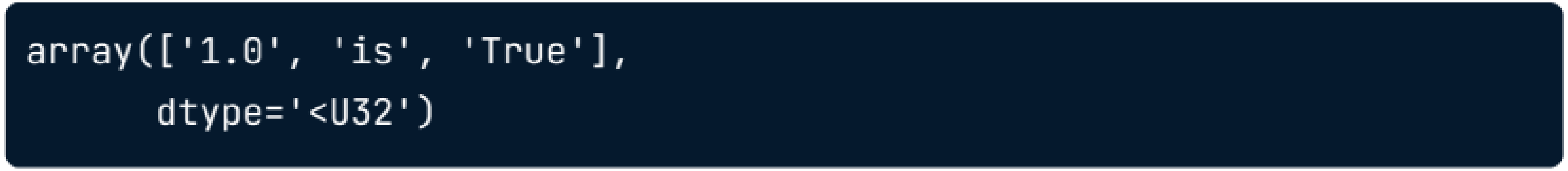
```
bmi = np_weight / np_height ** 2
bmi
```

```
array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

Numpy: remarks



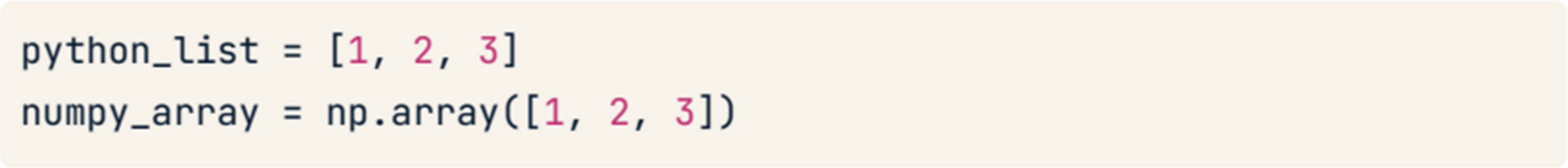
```
np.array([1.0, "is", True])
```



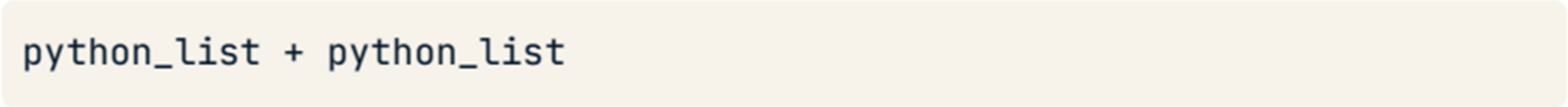
```
array(['1.0', 'is', 'True'],
      dtype='<U32')
```

- Numpy arrays: contain only one type

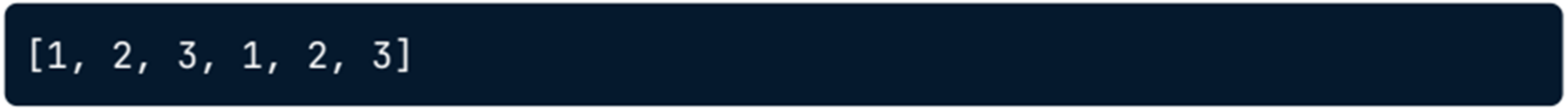
Numpy: remarks



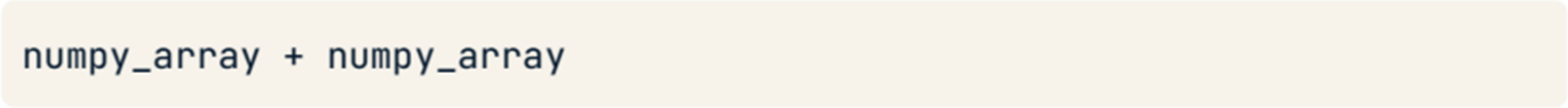
```
python_list = [1, 2, 3]
numpy_array = np.array([1, 2, 3])
```



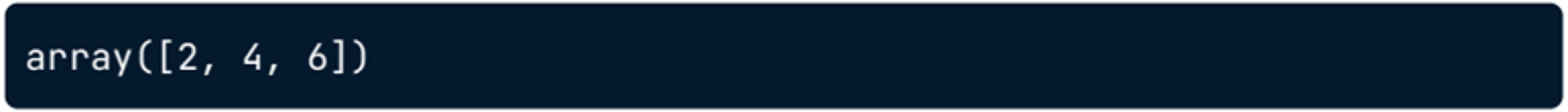
```
python_list + python_list
```



```
[1, 2, 3, 1, 2, 3]
```



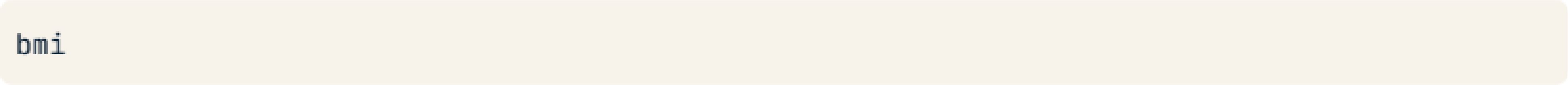
```
numpy_array + numpy_array
```



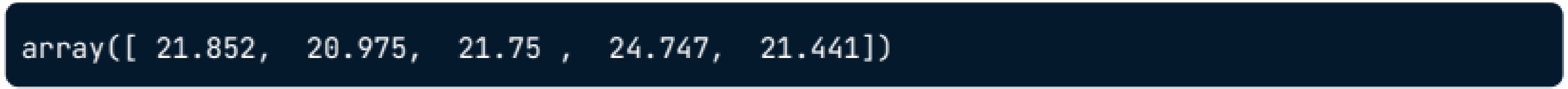
```
array([2, 4, 6])
```

- Different types: different behavior!

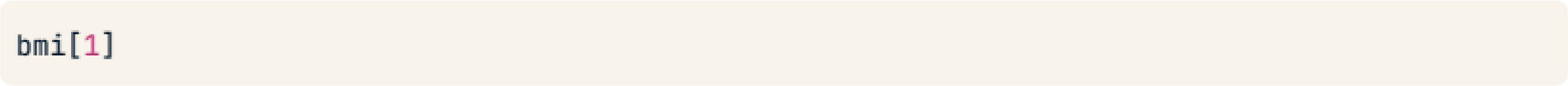
Numpy Subsetting



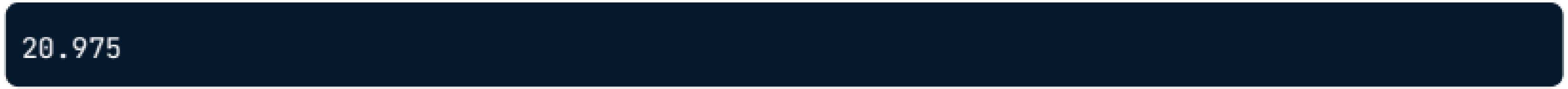
```
bmi
```



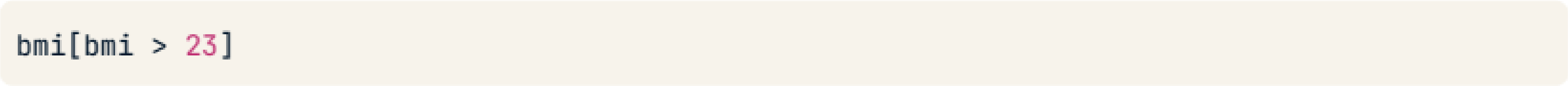
```
array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```



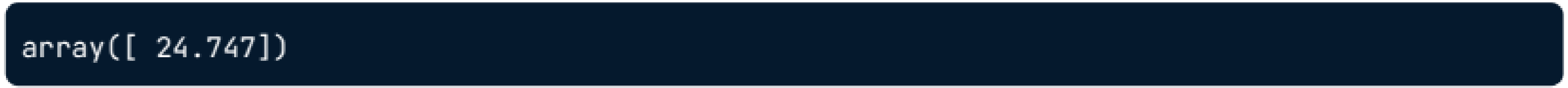
```
bmi[1]
```



```
20.975
```



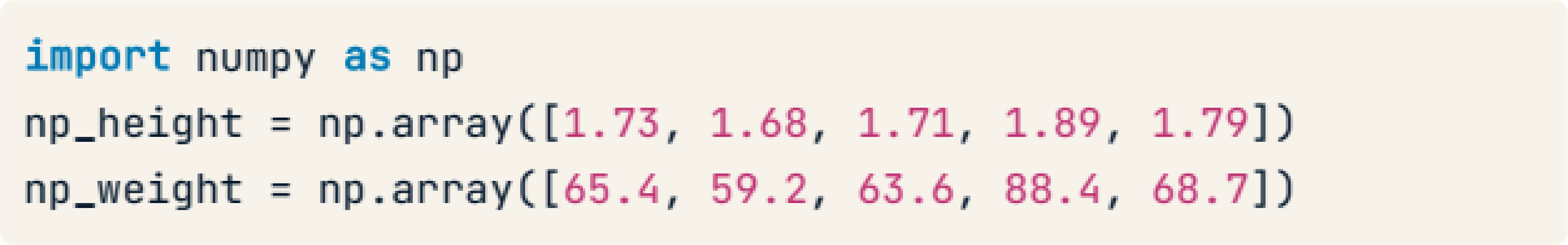
```
bmi[bmi > 23]
```



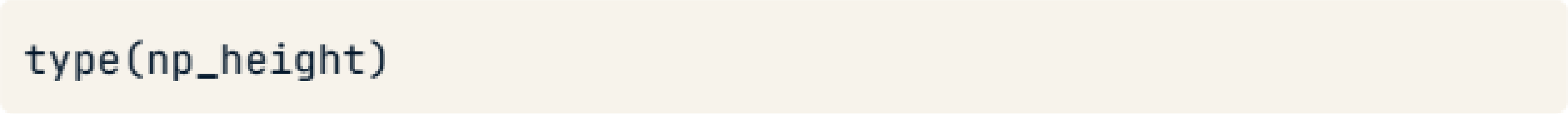
```
array([ 24.747])
```

2D Numpy Arrays

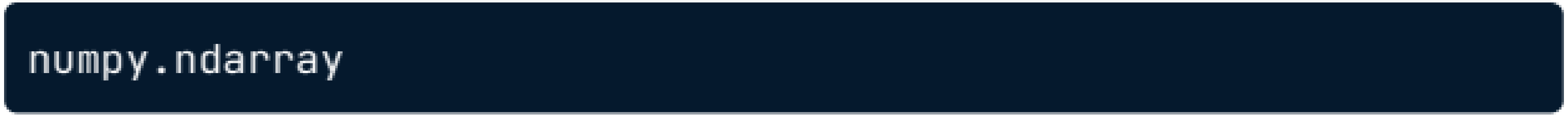
Type of Numpy Arrays



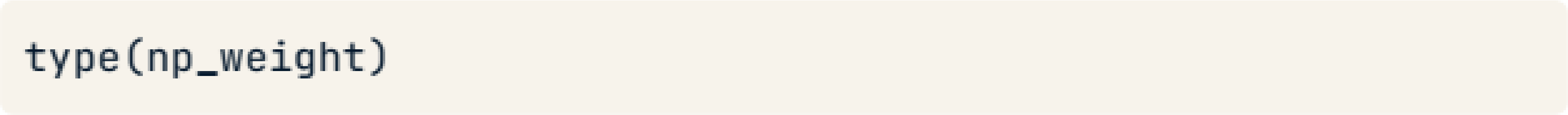
```python
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
```



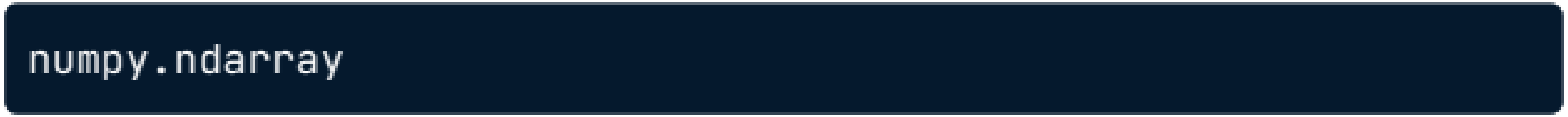
```python
type(np_height)
```



```
numpy.ndarray
```

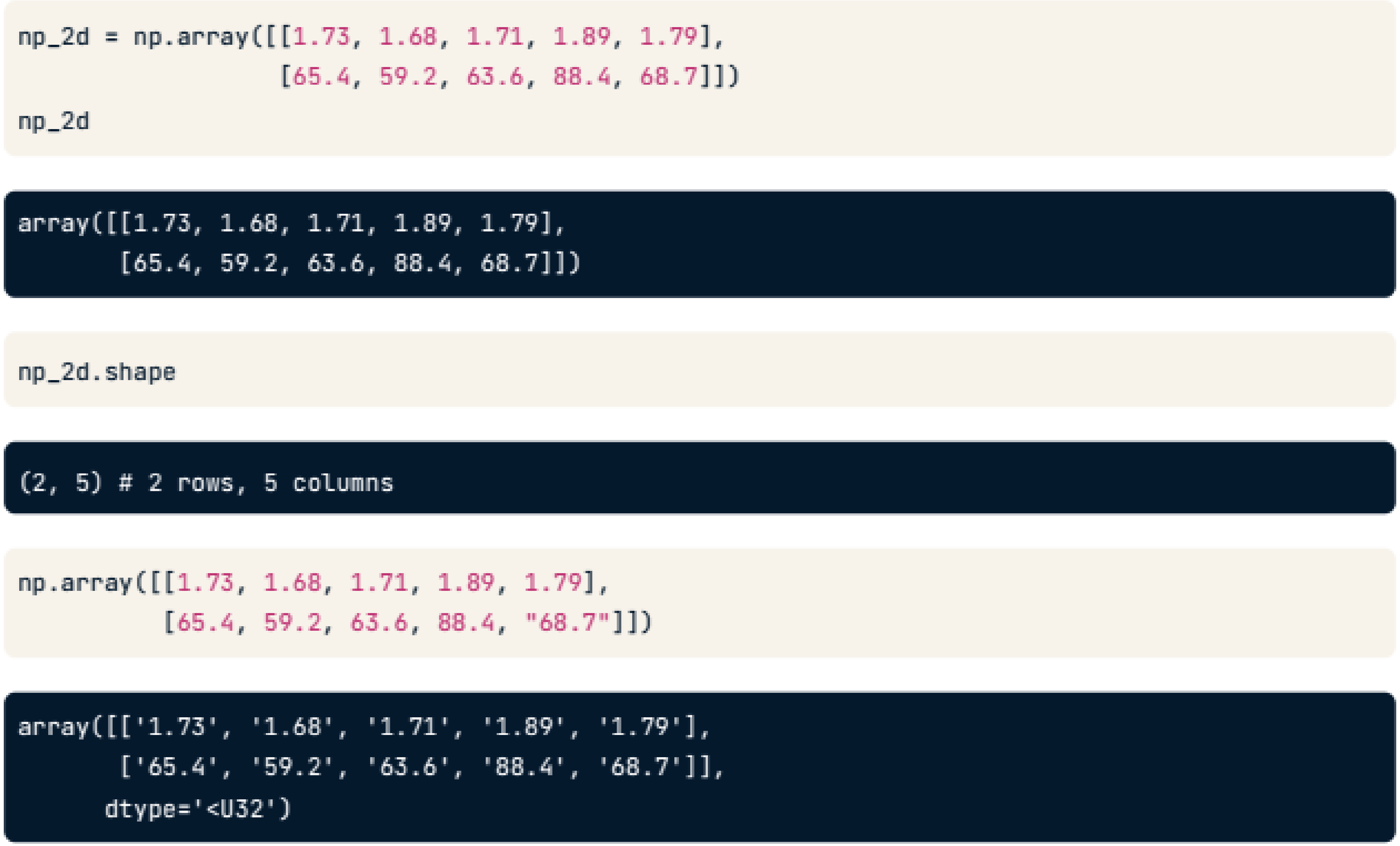


```python
type(np_weight)
```



```
numpy.ndarray
```

2D Numpy Arrays



```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, 68.7]])
np_2d
```

```
array([[1.73, 1.68, 1.71, 1.89, 1.79],
       [65.4, 59.2, 63.6, 88.4, 68.7]])
```

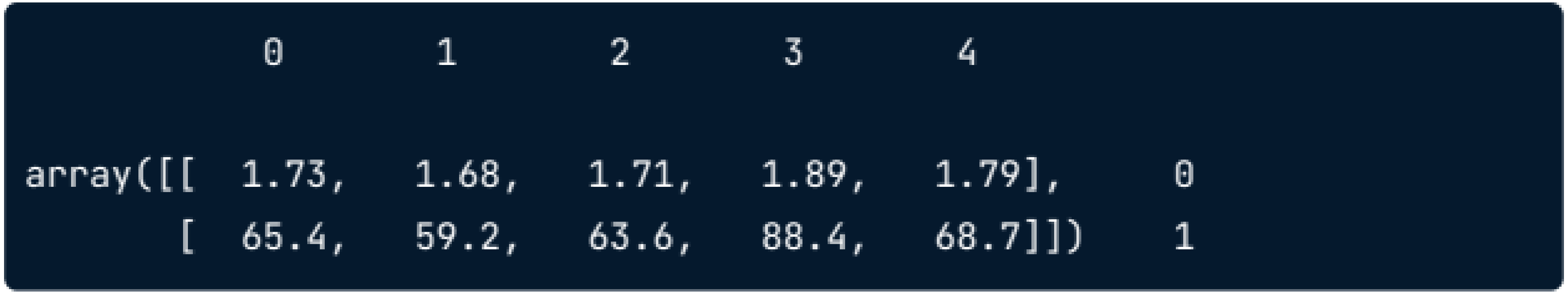
```
np_2d.shape
```

```
(2, 5) # 2 rows, 5 columns
```

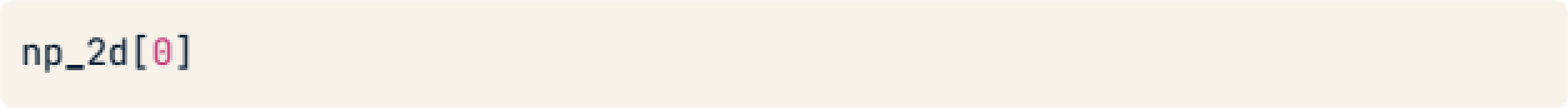
```
np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
          [65.4, 59.2, 63.6, 88.4, "68.7"]])
```

```
array([['1.73', '1.68', '1.71', '1.89', '1.79'],
       ['65.4', '59.2', '63.6', '88.4', '68.7']],
      dtype='<U32')
```

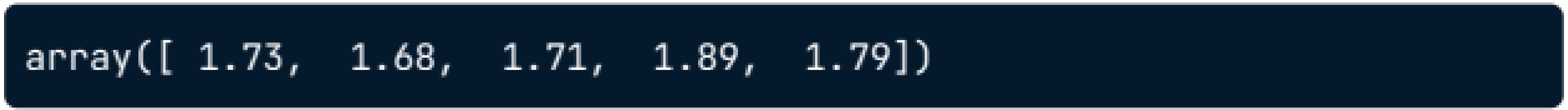
Sub setting



```
               0         1         2         3         4


array([[   1.73,     1.68,     1.71,     1.89,     1.79],       0
        [  65.4,     59.2,     63.6,     88.4,     68.7]])      1
```

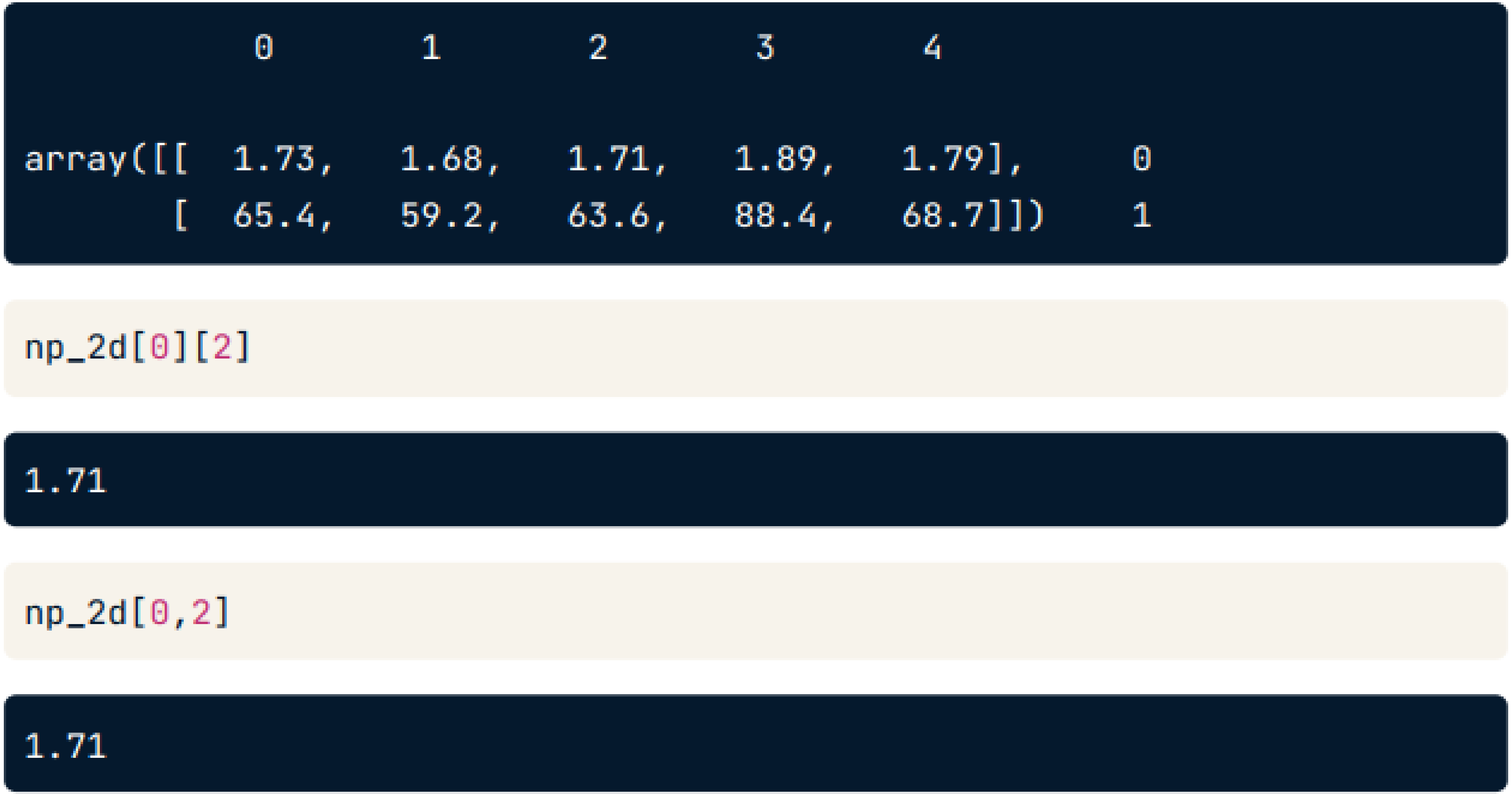


```
np_2d[0]
```



```
array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

Sub setting



```
              0          1          2          3          4


array([[   1.73,      1.68,      1.71,      1.89,      1.79],       0
        [  65.4,      59.2,      63.6,      88.4,      68.7]])      1
```

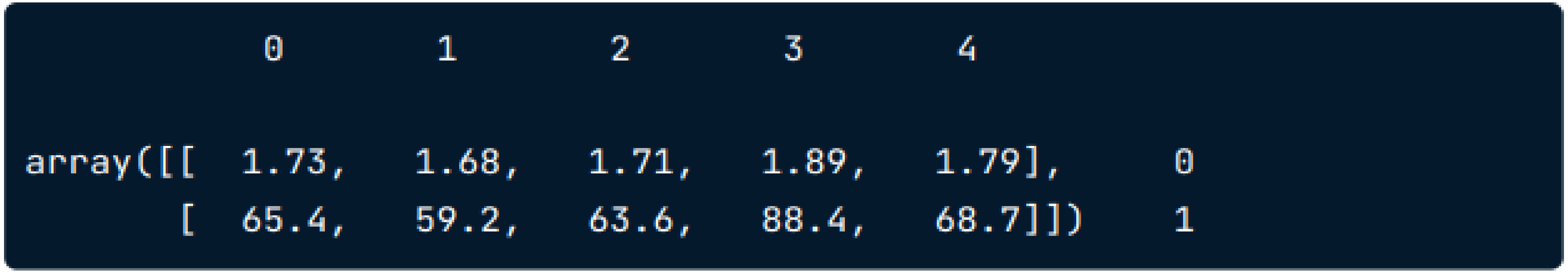
```
np_2d[0][2]
```

```
1.71
```

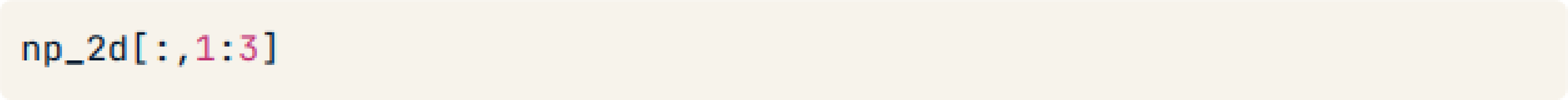
```
np_2d[0,2]
```

```
1.71
```

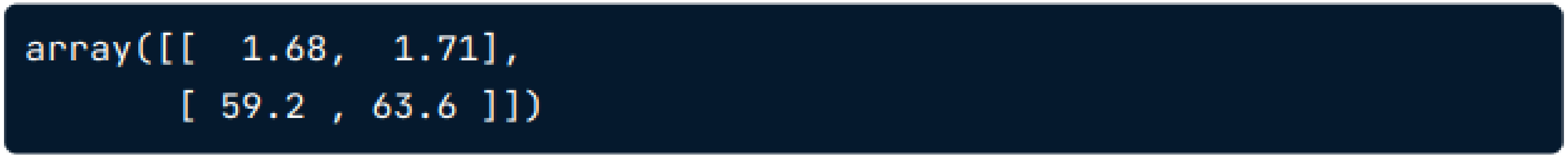
Sub setting



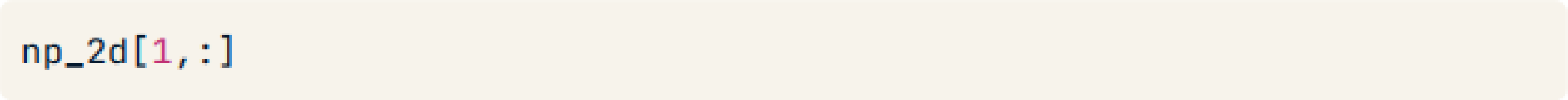
```
             0        1        2        3        4

array([[  1.73,    1.68,    1.71,    1.89,    1.79],      0
       [  65.4,    59.2,    63.6,    88.4,    68.7]])     1
```



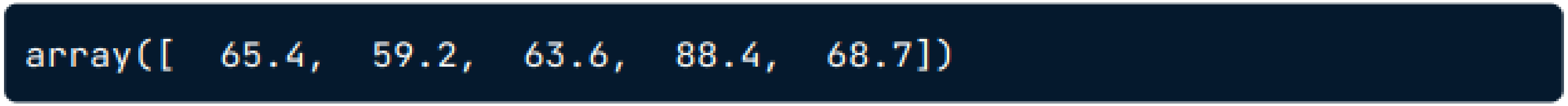
```
np_2d[:,1:3]
```



```
array([[  1.68,   1.71],
       [ 59.2 ,  63.6 ]])
```



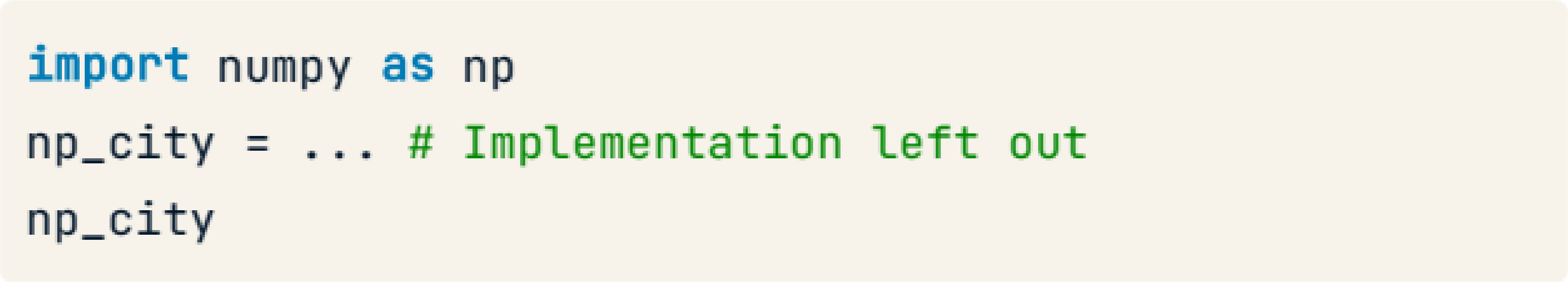
```
np_2d[1,:]
```



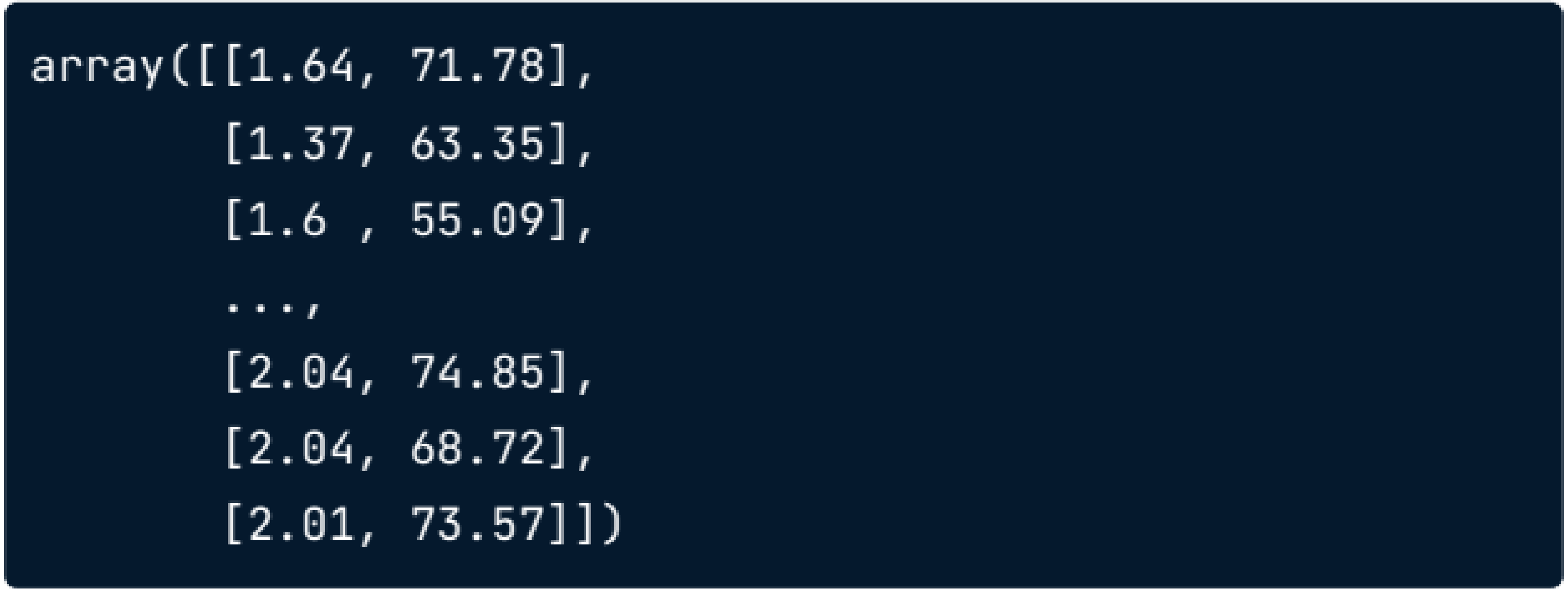
```
array([  65.4,   59.2,   63.6,   88.4,   68.7])
```

Numpy:
Basic Statistics

City-wide survey



```python
import numpy as np
np_city = ... # Implementation left out
np_city
```

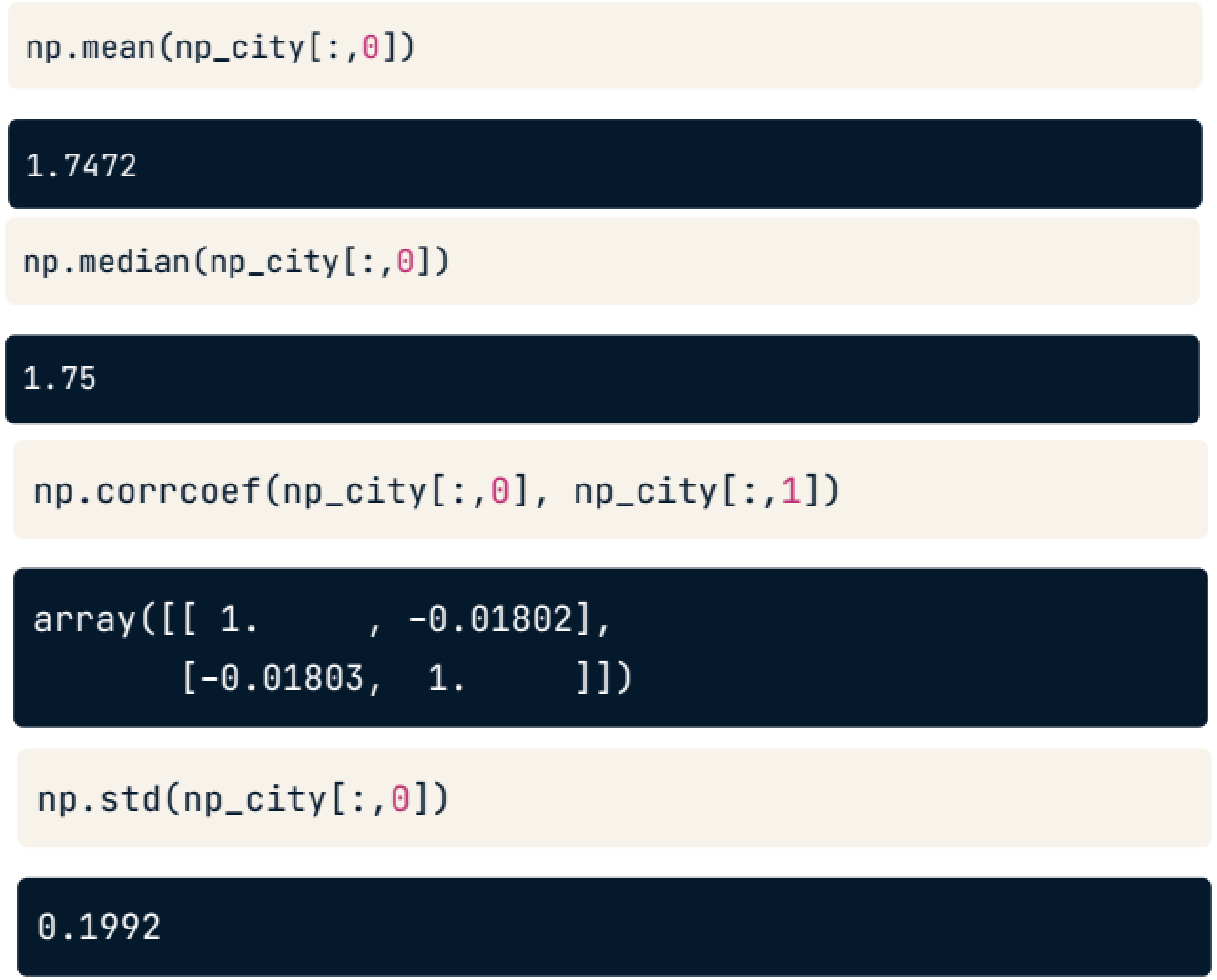


```
array([[1.64, 71.78],
       [1.37, 63.35],
       [1.6 , 55.09],
       ...,
       [2.04, 74.85],
       [2.04, 68.72],
       [2.01, 73.57]])
```

Numpy

✓ mean, median, corrcoef, std, sum(), sort(), ...

✓ Enforce single data type: speed!



```
np.mean(np_city[:,0])
```

```
1.7472
```

```
np.median(np_city[:,0])
```

```
1.75
```

```
np.corrcoef(np_city[:,0], np_city[:,1])
```

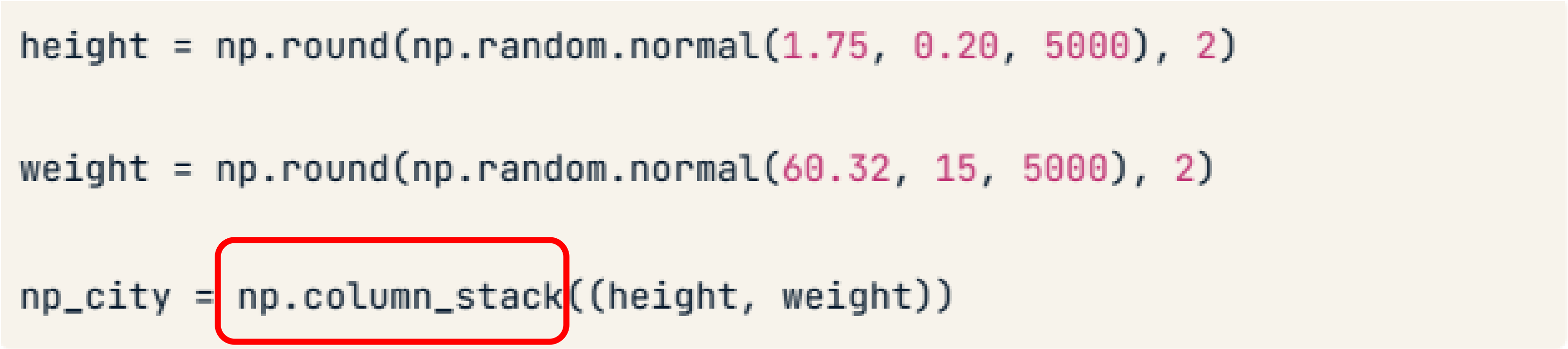
```
array([[ 1.      , -0.01802],
       [-0.01803,  1.      ]])
```

```
np.std(np_city[:,0])
```

```
0.1992
```

Generate data

- Arguments for np.random.normal()
  - ✓ Distribution mean
  - ✓ Distribution standard deviation
  - ✓ Number of samples



```
height = np.round(np.random.normal(1.75, 0.20, 5000), 2)

weight = np.round(np.random.normal(60.32, 15, 5000), 2)

np_city = np.column_stack((height, weight))
```

Let's practice!