NumPy

Arrays and functions

Watch by Tuesday, October 20, 2020 | Lesson #5

OCEAN 215 | Autumn 2020 Ethan Campbell and Katy Christensen

What we'll cover in this lesson

1. Functions and arguments

- 2. NumPy arrays arithmetic, logical operations, indexing
- 3. NumPy functions and constants

Functions and arguments

Function name An "argument" or "parameter"

The function "returns" or "evaluates to" the integer 4

Some functions act on a target

The "target" of the function Function name

The function returns 'PYTHON'

Values returned by functions can be stored in a variable

The "target" of the function Function name 'python'.upper() new_string = 'python'.upper()

Some functions don't return anything

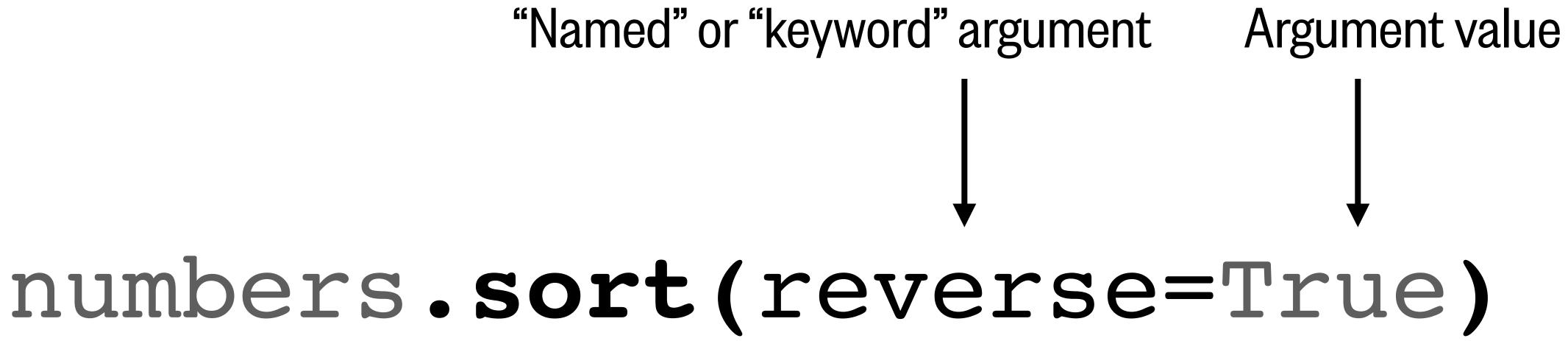
numbers.sort()

This function returns nothing at all! It simply modifies numbers "in-place," which becomes [5,6,7,8].

numbers = [6, 8, 7, 5]

Some functions have named arguments

Now, numbers will be sorted in reverse order: [8,7,6,5].



Named arguments always have a default value

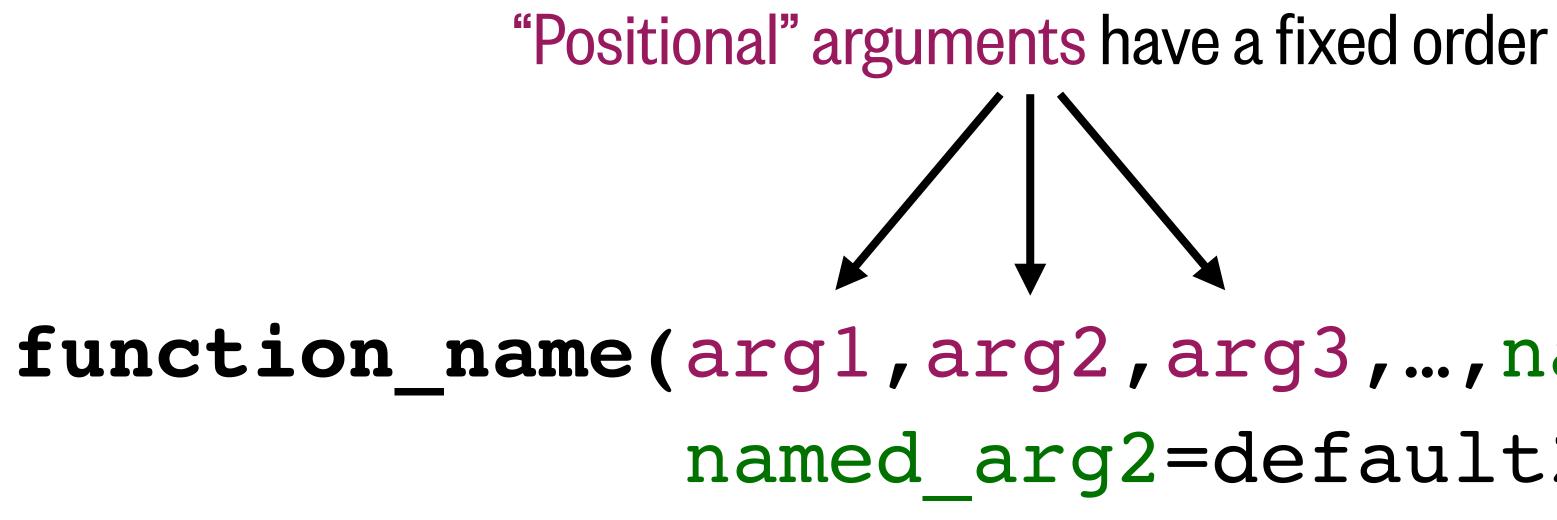
The "default" value of reverse numbers.sort(reverse=False)

is equivalent to:

numbers.sort()



Functions can have **both** positional and named arguments



"Named" arguments can be provided in any order, but they must follow any positional arguments

function_name(arg1,arg2,arg3,...,named arg1=default1, named_arg2=default2,...) /



What we'll cover in this lesson

- 1. Functions and arguments
- 3. NumPy functions and constants

2. NumPy arrays – arithmetic, logical operations, indexing

Loading NumPy ("Numeric Python")

Makes this package available to Python import numpy

Package names are usually all lowercase

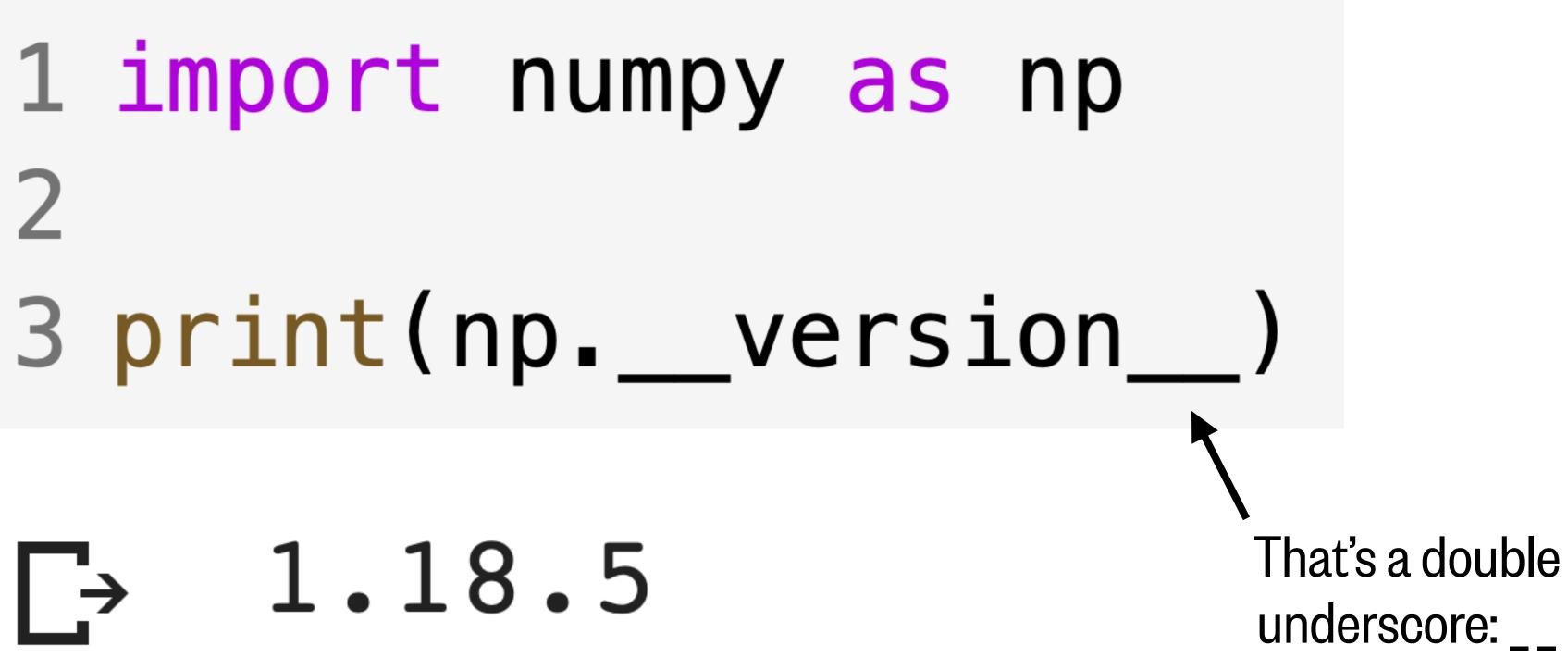
This is a shortcut; you can choose any name but np is most common as np This part is technically optional



Checking a package's version

1 import numpy as np

 $\square 1.18.5$



The NumPy array (ndarray)

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

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



Similarities between lists and NumPy 1-D arrays

Both are **mutable** (they can be changed)

```
1 \text{ numbers} = np.array([5,6,7,8])
```

- 2 numbers[1] = 13
- 3 print(numbers)
- [→ [5 13 7 8]

Both are **iterable**

- 1 for num in numbers:
- print(num)
- 5 C→ 13 7 8

Both are compatible with **indexing** and **slicing**

- 1 print(numbers[-3:])
- [→ [13 7 8]

Find length using **len()**

1 print(len(numbers))

C→ 4

Check membership using **in** and **not in**

1 print(13 in numbers) 2 print(14 in numbers)

[→ True False



Differences between lists and NumPy 1-D arrays

Lists

• Lists can contain a mix of object types (integers, strings, sub-lists, etc.)

 Lists are computationally inefficient (avoid using to store large data sets)

NumPy 1-D arrays

- Arrays can contain only a single object type (check using .dtype, change using .astype())
 - 1 numbers = np.array([5,6,7,8])
 2 print(numbers.dtype)
 3 print(numbers.astype(str))

• Arrays are fast for computation and small in memory (great for big data)



Differences between lists and NumPy 1-D arrays

Lists

• Lists don't preserve scientific notation in floating-point numbers

1 print([3.5e9,1.4e-3])

[350000000.0, 0.0014] **□**→

- Use Python's in-place append() or extend(), insert(), del, reverse(), remove(), pop()
 - 1 numbers = [5, 6, 7, 8]
 - 2 numbers.append([9,10])
 - 3 print(numbers)
 - $[\rightarrow [5, 6, 7, 8, [9, 10]]$

NumPy 1-D arrays

- Arrays preserve scientific notation
 - 1 print(np.array([3.5e9,1.4e-3]))

[3.5e+09 1.4e-03] **Γ**→

• NumPy's append(), insert(), delete(), flip() functions are **not in-place**; note the different syntax; no functions to remove, pop

1 numbers = np.array([5,6,7,8])2 numbers = np.append(numbers, [9,10]) 3 print(numbers)

[5 6 7 8 9 10]



Differences between lists and NumPy 1-D arrays

Lists

- Convert from list \rightarrow array using:
 - $1 \text{ my_list} = [5, 6, 7, 8]$ 2 my_array = np.array(my_list)
- Adding lists **concatenates** (joins) **them**:

1 a =
$$[1,2,3,4]$$

2 b = $[5,6,7,8]$
3 print(a + b)

 $[] \rightarrow [1, 2, 3, 4, 5, 6, 7, 8]$

NumPy 1-D arrays

• Convert from array \rightarrow list using:

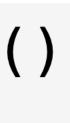
1 my_list1 = my_array.tolist() $2 \text{ my_list2} = \text{list(my_array)}$

Adding arrays actually adds them!*

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

8 10 12] [6

* Note that NumPy also has a concatenate() function.



Arithmetic operations with arrays

Arithme	Element-w	
+	Addition	1 a = np.a 2 b = np.a
	Subtraction	3 4 print('a
*	Multiplication	5 print('a 6 print('a
/	Division	Element-w
* *	Exponential	<pre>1 print('a 2 print('1</pre>
8	Remainder	<pre>2 print('1 3 print('a 4 print('a)</pre>
//	Floor	

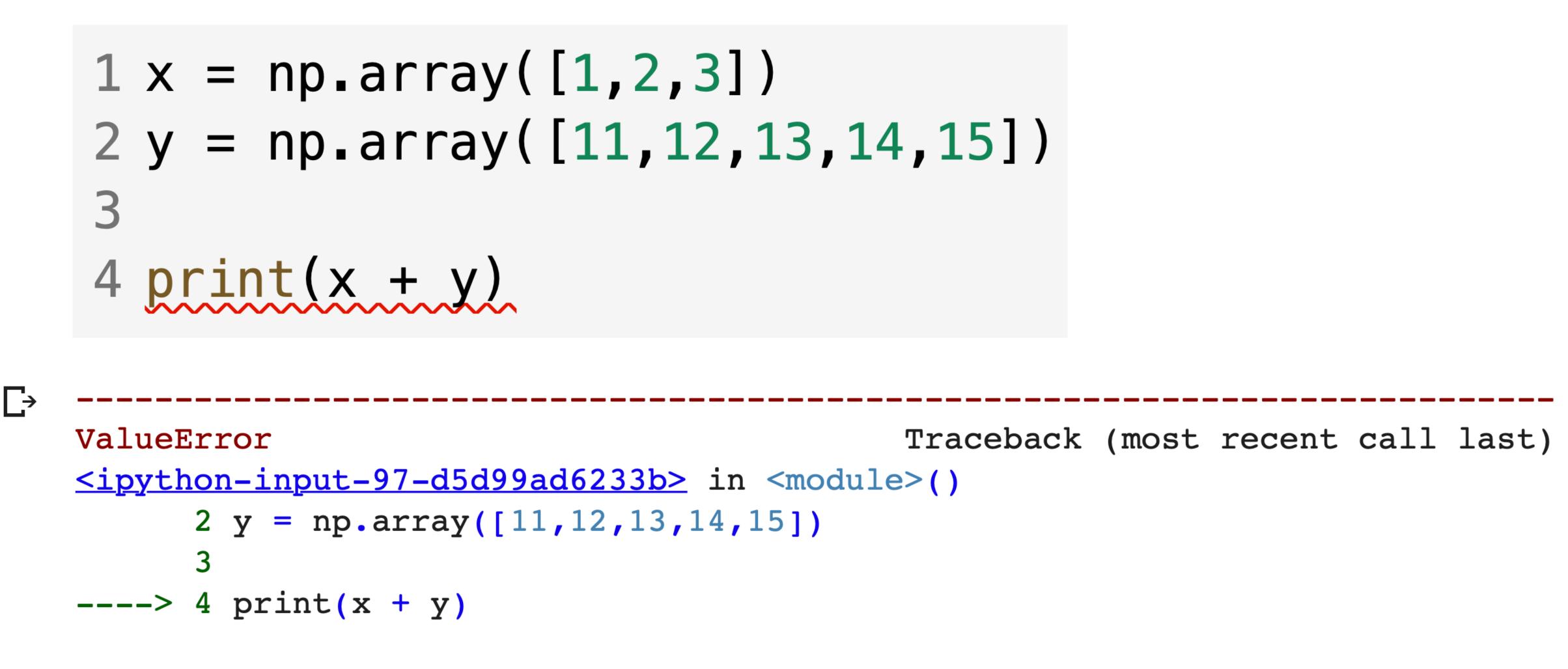
- vise arithmetic between two or more arrays
- array([1,2,3,4]) array([5,6,7,8])
- a + b = ', a + b a + b = [6 8 10 12]a - b = ', a - b) a - b = [-4 - 4 - 4]a * b = ',a * b) a * b = [5 12 21 32]
- ise arithmetic with an array and a number

$$a + 10 = [11 \ 12 \ 13 \ 14]$$

 $10 * a = [10 \ 20 \ 30 \ 40]$
 $a / 10 = [0.1 \ 0.2 \ 0.3]$
 $a**2 = [1 \ 4 \ 9 \ 16]$



Element-wise operations require arrays to be the same dimensions



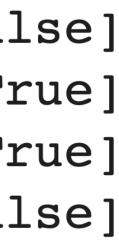
ValueError: operands could not be broadcast together with shapes (3,) (5,)

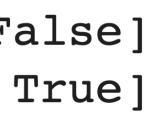


Logical operations with arrays

Comparison operators		Element-wise comparisons two arrays or an array and a				
==	Equal	1 u = np.array([1,2,3,4])				
! =	Not equal	<pre>2 v = np.array([0,2,4,6]) 3 4 print(u == v)</pre>	[False	'alse True False Fal		
>	Greater than	<pre>5 print(u < v) 6 print(v != 0) 7 print(v <= 4)</pre>	[False	[False False Tru [False True Tru [True True Tru	True	e Tr
>=	Greater than or equal to	Instead of comparing Boole	L			
<	Less than	<pre>use np.logical_and() and np.logical_or() 1 bool1 = np.array([True,False,True]) 2 bool2 = np.array([True,False,False]) 3 4 print(np.logical_and(bool1,bool2)) [True False False]</pre>				
<=	Less than or equal to				False	Fal

5 print(np.logical_or(bool1,bool2)) True False





New indexing options with arrays

When you want to access certain value(s) in an array:

```
1 v = np.array([10, 11, 12, 13])
2
3 print(v[3])
4
5 print(v[[2,3]])
6
7 print(v[v >= 12])
8
9 print(v[[False,False,True,True]])
```

When you want the indices of certain values in an array:

- np.where() gives the indices at which 1 print(np.where(v >= 12)) (array([2, 3]),) a Boolean condition is satisfied... 2
- 3 print(np.where(v >= 12)[0]) [2 3]

Python prints:

- Traditional list-style **single index** 13
- [12 13] **Multiple indices** retrieves multiple elements
- [12 13] Logical conditions also work...
- $[12 \ 13]$... because they evaluate to **Boolean arrays**

... but you have to index into the result using [0]



What we'll cover in this lesson

- 1. Functions and arguments
- 2. NumPy arrays arithmetic, logical operations, indexing
- **3. NumPy functions and constants**

Most functions acting on NumPy arrays can be called two ways

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

np. Sum(X) ------ Evaluates to: 46

X. **Sum**() \leftarrow Evaluates to: 46



NumPy functions can also be applied to lists

x = [10, 11, 12, 13]



np. Sum(X) ----- Evaluates to: 46

X. **Sum**() \leftarrow Evaluates to: 46

Mathematical reductions (array \rightarrow number)

x = np.array([10, 11, 12, 13])**Function: Purpose:** np.sum(x) Sum np.mean(x) Mean (average) np.median(x) Median np.max(x) Maximum value np.min(x) Minimum value np.std(x)Standard deviation

Evaluates to:

- 46
- 11.5
- 11.5
- 13
- 10
- 1.11803...

Mathematical constants (each return a float)

Constant value:	Pur
np.pi	Π(
np.e	<i>e</i> (E
np.inf	Pos
np.nan	"Not (used

Note:

1 print(5 * np.inf)2 print(5 * np.nan)

- rpose:
- (pi)
- Euler's number)
- sitive infinity
- t a Number" as a placeholder for missing data)
 - inf **Γ**→ nan

- **Evaluates to:**
- 3.14159...
- 2.71828...
- inf
- nan



Element-wise functions (number \rightarrow number, or array \rightarrow array)

Function:

- np.absolute([-2, -1]) np.round([5.23,5.29],1)
- np.sqrt([4,9,16])
- np.exp([0,1,2])
- np.sin([0,np.pi/2])
- np.cos([np.pi,2*np.pi]) Cosine

Purpose:

- Absolute value
- Round to a certain decimal place
- Square root (same as **0.5)
- Exponential (same as np.e**)
- Sine (from radians)

Evaluates to arrays:

- [2,1]
- [5.2, 5.3]
- [2.,3.,4.]
- [1.,2.718...,7.389...]
- [0., 1.]
- [-1., 1.]





Functions to create new arrays

Function: Purpose: Array of given length np.zeros(4) filled with zeros Array of given length np.ones(4) filled with ones Array of given length np.full(4,2)filled with given value np.arange(4) Same as range ()... np.arange(0, 1, 0.25)increments are allowed np.linspace(0, 1, 5)

- ...except floats and fractional
- Returns the given number of evenly spaced values from start to end (both are inclusive)

Evaluates to arrays:

- [0., 0., 0., 0.]
- [1., 1., 1., 1.]
- [2, 2, 2, 2]
- [0, 1, 2, 3]
- [0., 0.25, 0.5, 0.75]
- [0., 0.25, 0.5, 0.75, 1.]

