Installing and using Python packages

How to install and update Python packages: Use the terminal and a package manager

- A package manager helps to install new packages and update them.
- Popular alternatives are **pip** and **conda**.
  - When using Google Colab you can use **pip** to install a Python Package.
  - When using a local Anaconda installation of Python, you can use **conda** to install a Python Package.
- **pip** and **conda** are a shell commands NOT Python commands, thus you have to include a **!** (“exclamation mark”) in front of the command or execute them directly in a shell:

```
pip install pandas
pip update pandas
```

```
conda install pandas
conda update pandas
```

Add an exclamation mark if you wan to execute this from a Python script.

How to install and update Python packages: Use the terminal and a package manager

- A package manager helps to install new packages and update them.
- Popular alternatives are **pip** and **conda**.
  - When using Google Colab you can use **pip** to install an Python Package.
  - When using a local Anaconda installation of Python, you can use **conda** to install an Python Package.
- **pip** and **conda** are a shell commands NOT Python commands, thus you have to include a **!** (“exclamation mark”) in front of the command or execute them directly in a shell/terminal/prompt window:

```
pip install pandas
pip update pandas
```

```
conda install pandas
conda update pandas
```

Add an exclamation mark if you wan to execute this from a Python script.

How to load/activate a package

In your code editor type:

```
import numpy as np
import pandas
```

1. You can specify an alias (with "`as aliasname`") pointing at the package

2. You have to reload a package in every session you are using it

Every time you use a function of the package, you have to rewrite the alias or the original name:

```
np.sum(...)
pandas.DataFrame(...)
```

You can also explicitly import modules (certain functions/classes) of a package:

```
from numpy import sum
```

How to load/activate a package

In your code editor type:

```
import numpy as np
import pandas
```

1. You can specify an alias (with "`as aliasname`") pointing at the package
2. You have to reload a package in every session you are using it

Every time you use a function of the package, you have to rewrite the alias or the original name:

```
np.sum(...)
pandas.DataFrame(...)
```

3. Use the alias
4. Use the full name

You can also explicitly import modules (certain functions/classes) of a package:

```
from numpy import sum
```

How to load/activate a package

In your code environment type:

```
import numpy as np
import pandas
```

1. You can specify an alias (with "`as aliasname`") pointing at the package
2. You have to reload a package in every session you are using it

Every time you use a function of the package, you have to rewrite the alias or the original name:

```
np.sum(...)
pandas.DataFrame(...)
```

3. Use the alias
4. Use the full name

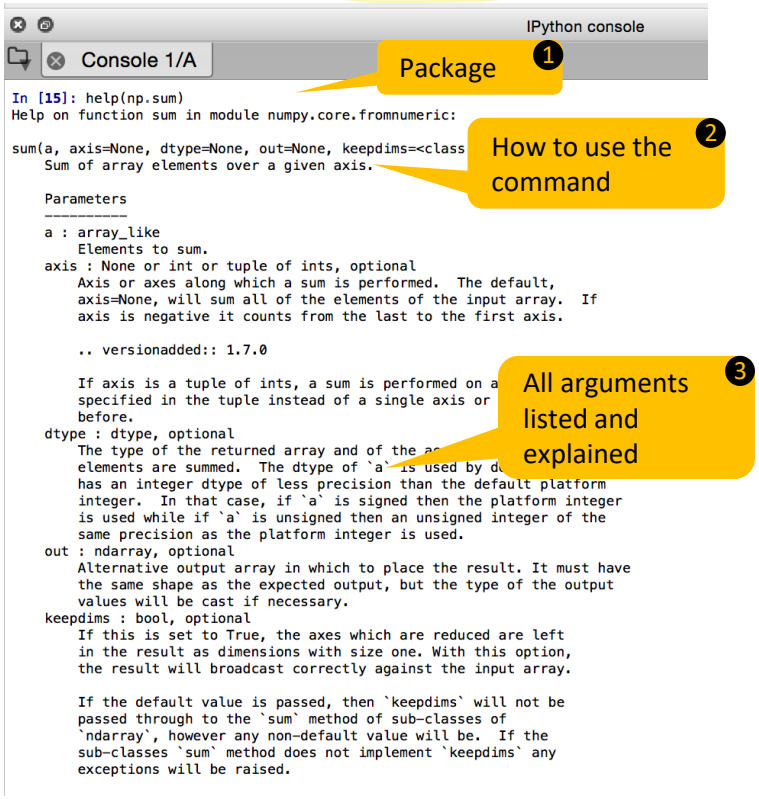
You can also explicitly import modules (certain functions/classes) of a package:

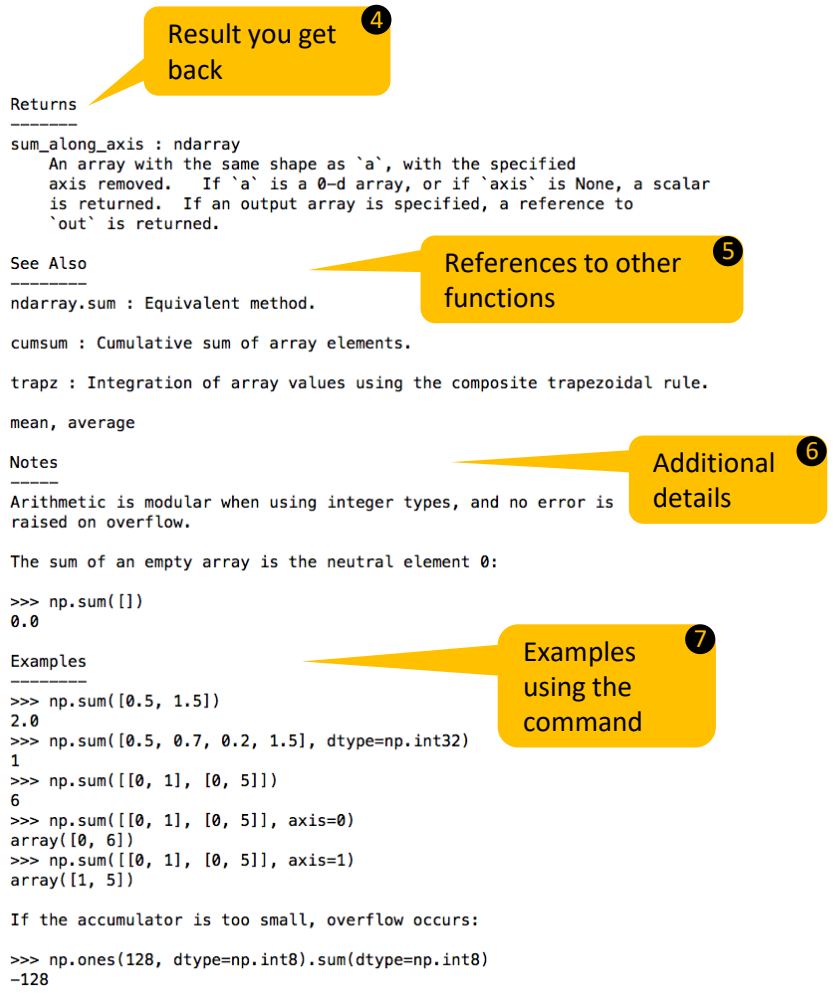
```
from numpy import sum
```

5. Instead of calling `numpy.sum()` you can just call `sum()` now
6. Caution: When importing your functions this way, the entire module "numpy" will be loaded though. Thus, you cannot save any time.

Start working with a Python package by looking at its help files

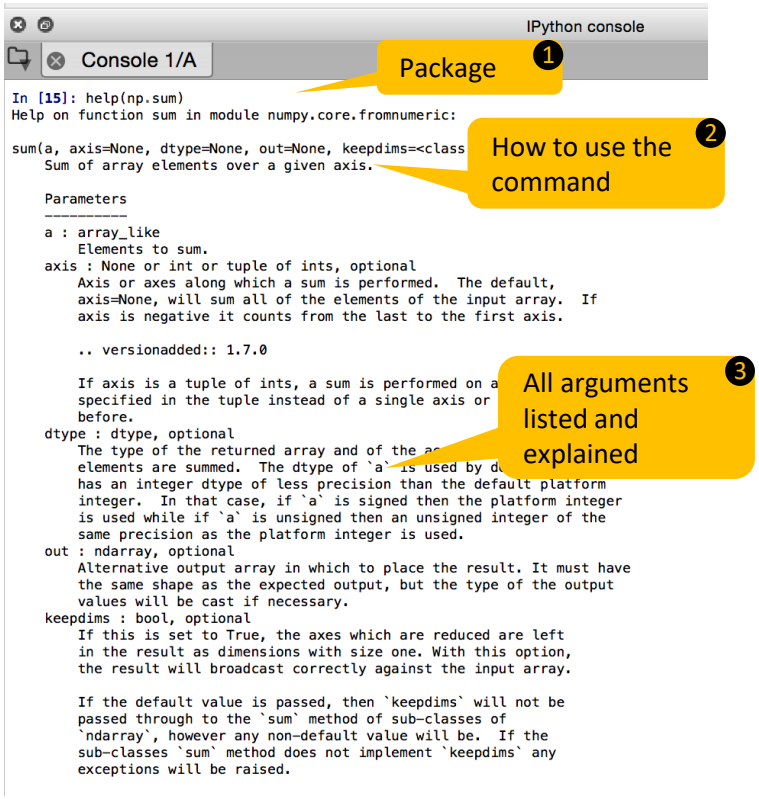
Open the help file: `help(np.sum)`

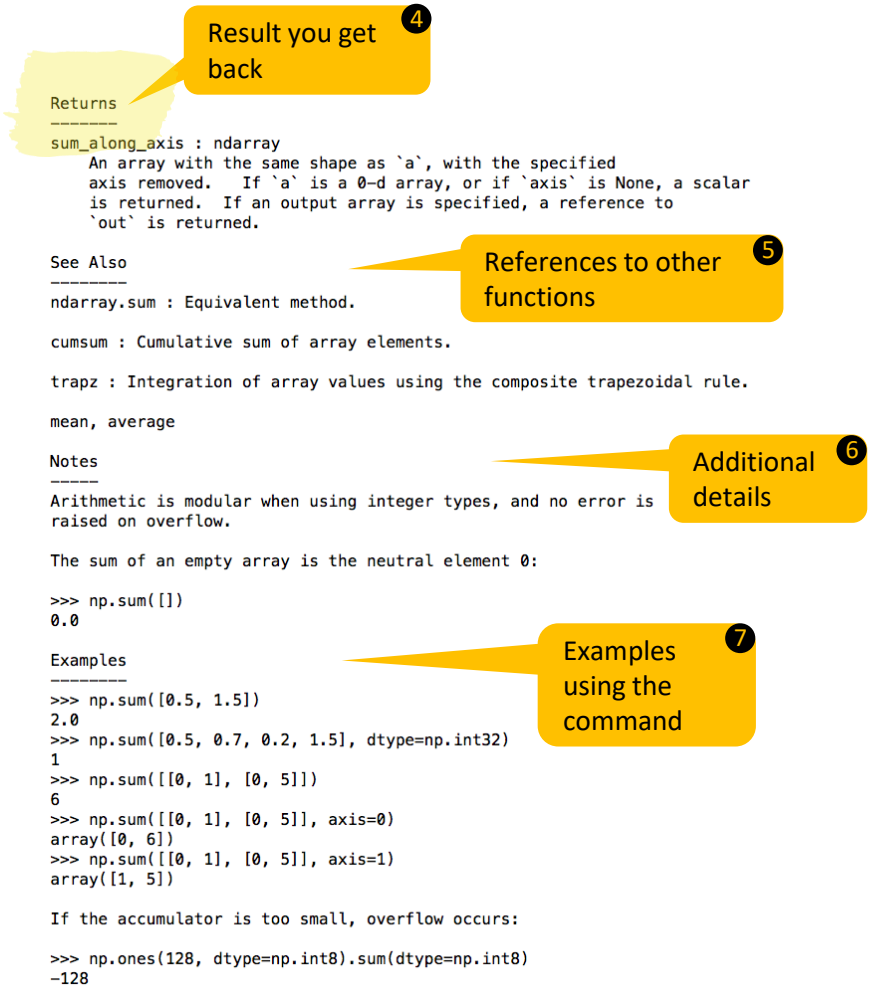




Start working with a Python package by looking at its help files

Open the help file: **`help(np.sum)`**





Start working with a Python package by looking at its help files

Open the help file: **`help(np.sum)`**

```
In [15]: help(np.sum)
Help on function sum in module numpy.core.fromnumeric:

sum(a, axis=None, dtype=None, out=None, keepdims=<class
    Sum of array elements over a given axis.

    Parameters
    ----------
    a : array_like
        Elements to sum.
    axis : None or int or tuple of ints, optional
        Axis or axes along which a sum is performed.  The default,
        axis=None, will sum all of the elements of the input array.  If
        axis is negative it counts from the last to the first axis.

        .. versionadded:: 1.7.0

        If axis is a tuple of ints, a sum is performed on a
        specified in the tuple instead of a single axis or
        before.
    dtype : dtype, optional
        The type of the returned array and of the ac
        elements are summed.  The dtype of `a` is used by d
        has an integer dtype of less precision than the default platform
        integer.  In that case, if `a` is signed then the platform integer
        is used while if `a` is unsigned then an unsigned integer of the
        same precision as the platform integer is used.
    out : ndarray, optional
        Alternative output array in which to place the result. It must have
        the same shape as the expected output, but the type of the output
        values will be cast if necessary.
    keepdims : bool, optional
        If this is set to True, the axes which are reduced are left
        in the result as dimensions with size one. With this option,
        the result will broadcast correctly against the input array.

        If the default value is passed, then `keepdims` will not be
        passed through to the `sum` method of sub-classes of
        `ndarray`, however any non-default value will be.  If the
        sub-classes `sum` method does not implement `keepdims` any
        exceptions will be raised.
```

1. Package
2. How to use the command
3. All arguments listed and explained

```
Returns
-------
sum_along_axis : ndarray
    An array with the same shape as `a`, with the specified
    axis removed.   If `a` is a 0-d array, or if `axis` is None, a scalar
    is returned.  If an output array is specified, a reference to
    `out` is returned.

See Also
--------
ndarray.sum : Equivalent method.

cumsum : Cumulative sum of array elements.

trapz : Integration of array values using the composite trapezoidal rule.

mean, average

Notes
-----
Arithmetic is modular when using integer types, and no error is
raised on overflow.

The sum of an empty array is the neutral element 0:

>>> np.sum([])
0.0

Examples
--------
>>> np.sum([0.5, 1.5])
2.0
>>> np.sum([0.5, 0.7, 0.2, 1.5], dtype=np.int32)
1
>>> np.sum([[0, 1], [0, 5]])
6
>>> np.sum([[0, 1], [0, 5]], axis=0)
array([0, 6])
>>> np.sum([[0, 1], [0, 5]], axis=1)
array([1, 5])

If the accumulator is too small, overflow occurs:

>>> np.ones(128, dtype=np.int8).sum(dtype=np.int8)
-128
```

4. Result you get back
5. References to other functions
6. Additional details
7. Examples using the command

Remove a Python package

```
pip uninstall numpy
```

If requested, confirm process with "y"

```
conda remove numpy
```

If you used `conda` for the installation

Installing and using Python packages