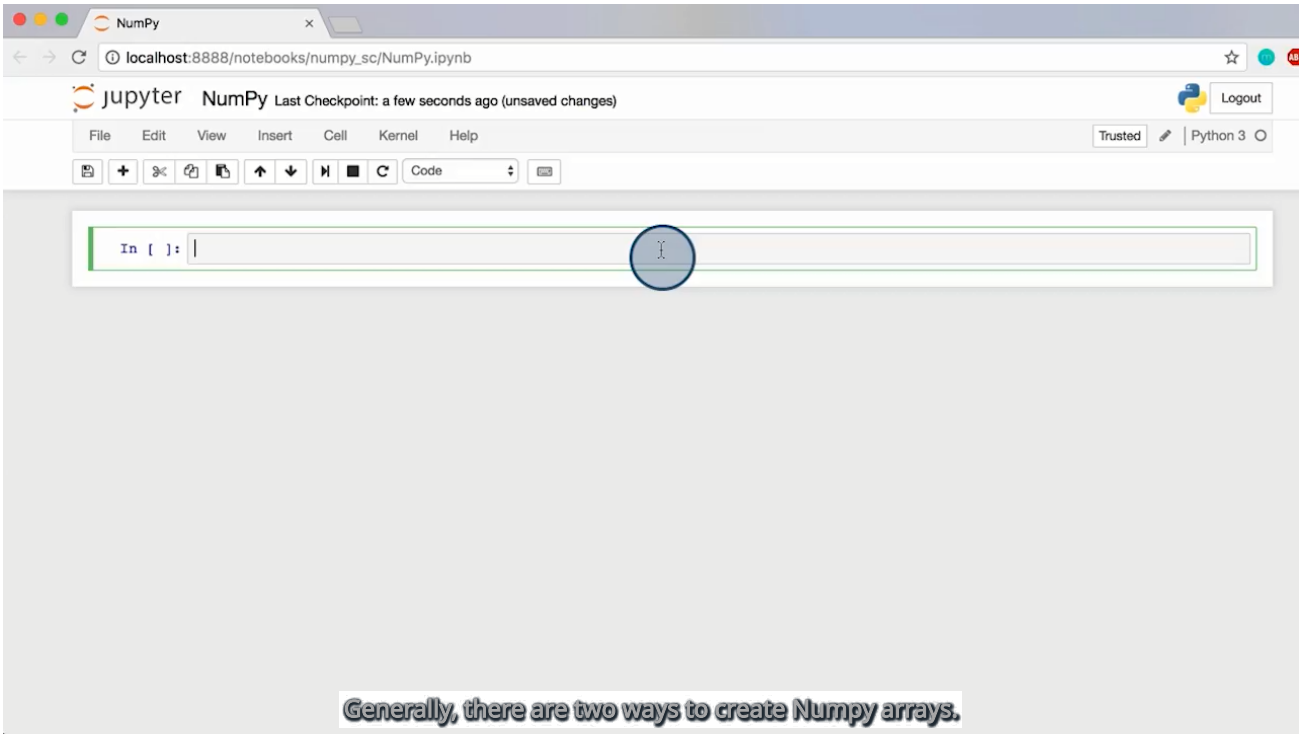
# 创建 NumPy ndarray




00:00 / 05:44 1x CC

NumPy 的核心是 **ndarray**，其中 *nd* 表示 n 维。ndarray 是一个多维数组，其中的所有元素类型都一样。换句话说，ndarray 是一个形状可以多样，并且可以存储数字或字符串的网格。在很多机器学习问题中，你通常都会发现需要以多种不同的方式使用 ndarray。例如，你可能会使用 ndarray 存储一个图像的像素值，然后将该图像馈送到神经网络中以进行图像分类。

但是在深入讲解 NumPy 并开始使用 NumPy 创建 ndarray 之前，我们需要在 Python 中导入 NumPy。我们可以使用 `import` 命令在 Python 中导入软件包。通常，我们使用 `np` 导入 NumPy。因此，你可以在 Jupyter notebook 中输入以下命令，导入 NumPy：

```
import numpy as np
```

我们可以通过多种方式在 NumPy 中创建 ndarray。在下面的课程中，我们将学习创建 ndarray 的两种方式：

1. 使用普通的 Python 列表
2. 使用内置 NumPy 函数

返回 ndarray 的函数。要阐明的是，这些课程中用到的示例都将使用简单的小型 ndarray。我们开始创建一维 ndarray 吧。

```
# We import NumPy into Python
import numpy as np

# We create a 1D ndarray that contains only integers
x = np.array([1, 2, 3, 4, 5])

# Let's print the ndarray we just created using the print() command
print('x = ', x)
```

```
x = [1 2 3 4 5]
```

我们先暂停一下，了解一些实用的术语。我们将一维数组称之为*秩*为 1 的数组。通常，*N* 维数组的秩为 *N*。因此，二维数组称为秩为 2 的数组。数组的另一个重要特性是*形状*。数组的形状是指每个维度的大小。例如，秩为 2 的数组的形状对应于数组的*行数*和*列数*。你将发现，NumPy ndarray 具有特殊的*属性*，使我们能够非常直观地获取关于 ndarray 的信息。例如，可以通过 `.shape` 属性获取 ndarray 的形状。shape 属性返回一个由 n 个正整数（用于指定每个维度的大小）组成的元组。在下面的示例中，我们将创建一个秩为 1 的数组，并了解如何获取其形状、类型和元素数据类型 (*dtype*)。

```
# We create a 1D ndarray that contains only integers
x = np.array([1, 2, 3, 4, 5])

# We print x
print()
print('x = ', x)
print()

# We print information about x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = [1 2 3 4 5]
```

```
x has dimensions: (5,)
x is an object of type: class 'numpy.ndarray'
The elements in x are of type: int64
```

并且有 5 个元素。 `type()` 函数告诉我们 `x` 的确是 NumPy ndarray。最后， `.dtype` 属性告诉我们 `x` 的元素作为*有符号 64 位整数*存储在内存中。NumPy 的另一个重要优势是能够处理的数据类型比 Python 列表要多。你可以在以下链接中查看 NumPy 支持的所有不同数据类型：

### NumPy 数据类型

正如之前提到的，ndarray 还可以存储字符串。我们来看看如何按照之前的相同方式创建一个秩为 1 的字符串 ndarray：向 `np.array()` 函数提供 Python 字符串列表。

```
# We create a rank 1 ndarray that only contains strings
x = np.array(['Hello', 'World'])

# We print x
print()
print('x = ', x)
print()

# We print information about x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = ['Hello' 'World']
```

```
x has dimensions: (2,)
x is an object of type: class 'numpy.ndarray' The elements in x are of type: U5
```

可以看出，shape 属性告诉我们 `x` 现在只有 2 个元素，虽然 `x` 现在存储的是字符串，但是 `type()` 函数告诉我们 `x` 依然像之前一样是 ndarray。但是， `.dtype` 属性告诉我们 `x` 中的元素作为*具有 5 个字符的 Unicode 字符串*存储在内存中。

请务必注意，Python 列表和 ndarray 之间的最大区别是：与 Python 列表不同的是，ndarray 的所有元素都必须类型相同。因此，虽然我们可以同时使用整数和字符串创建 Python 列表，但是无法在 ndarray 中同时使用这两种类型。如果向 `np.array()` 函数提供同时具有整数和字符串的 Python 列表，NumPy 会将所有元素解析为字符串。我们可以在下面的示例中见到这种情况：

```
x = np.array([1, 2, 'World'])

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

> x = ['1' '2' 'World']

> x has dimensions: (3,)
> x is an object of type: 'numpy.ndarray' 类 The elements in x are of type: U21

可以看出，虽然 Python 列表具有不同的数据类型，但是 `x` 中的元素类型都一样，即*具有 21 个字符的 Unicode 字符串*。在 NumPy 简介的剩余部分，我们将不使用存储字符串的 ndarray，但是请注意，ndarray 也可以存储字符串。

现在看看如何利用嵌套 Python 列表创建秩为 2 的 ndarray。

```
# We create a rank 2 ndarray that only contains integers
Y = np.array([[1,2,3],[4,5,6],[7,8,9], [10,11,12]])

# We print Y
print()
print('Y = \n', Y)
print()

# We print information about Y
print('Y has dimensions:', Y.shape)
print('Y has a total of', Y.size, 'elements')
print('Y is an object of type:', type(Y))
print('The elements in Y are of type:', Y.dtype)
```

> Y =
> [[ 1 2 3]
> [ 4 5 6]
> [ 7 8 9]
> [10 11 12]]

> Y has a total of 12 elements Y is an object of type: class 'numpy.ndarray'
> The elements in Y are of type: int64

可以看出，现在 shape 属性返回元组 `(4,3)`，告诉我们 `Y` 的秩为 2，有 4 行 3 列。`.size` 属性告诉我们 `Y` 共有 12 个元素。

注意，当 NumPy 创建 ndarray 时，它会自动根据用于创建 ndarray 的元素的类型为其分配 *dtype*。到目前为止，我们只创建了包含整数和字符串的 ndarray。我们发现，当我们创建只有整数的 ndarray 时，NumPy 将自动为其元素分配 dtype int64。我们来看看当我们创建具有浮点数和整数的 ndarray 时，会发生什么。

```
# We create a rank 1 ndarray that contains integers
x = np.array([1,2,3])

# We create a rank 1 ndarray that contains floats
y = np.array([1.0,2.0,3.0])

# We create a rank 1 ndarray that contains integers and floats
z = np.array([1, 2.5, 4])

# We print the dtype of each ndarray
print('The elements in x are of type:', x.dtype)
print('The elements in y are of type:', y.dtype)
print('The elements in z are of type:', z.dtype)
```

> The elements in x are of type: int64
> The elements in y are of type: float64
> The elements in z are of type: float64

可以看出，当我们创建只有浮点数的 ndarray 时，NumPy 将元素当做 *64 位浮点数 (float64)* 存储在内存中。但是，当我们创建同时包含浮点数和整数的 ndarray 时，就像上面的 `z` ndarray，NumPy 也会为其元素分配 *float64* dtype。这叫做*向上转型*。因为 ndarray 的所有元素都必须类型相同，因此在这种情况下，NumPy 将 `z` 中的整数向上转型为浮点数，避免在进行数学计算时丢失精度。

虽然 NumPy 自动为 ndarray 选择 dtype，但是 NumPy 也允许你指定要为 ndarray 的元素分配的特定 dtype。当你在 `np.array()` 函数中创建 ndarray 时，可以使用关键字 `dtype` 指定 dtype。我们来看一个示例：

```
# We print x
print()
print('x = ', x)
print()

# We print the dtype x
print('The elements in x are of type:', x.dtype)
```

> x = [1 2 3 4 5]

> The elements in x are of type: int64

可以看出，虽然用浮点数创建了 ndarray，但是通过将 dtype 指定为 int64，NumPy 通过去除小数将浮点数转换成了整数。如果你不希望 NumPy 意外地选择错误的数据类型，或者你只希望达到一定的计算精度，从而节省内存，则指定 ndarray 的数据类型很有用。

创建 ndarray 后，你可能需要将其保存到文件中，以便以后读取该文件或供另一个程序使用。NumPy 提供了一种将数组保存到文件中以供日后使用的方式。我们来看看操作方式。

```
# We create a rank 1 ndarray
x = np.array([1, 2, 3, 4, 5])

# We save x into the current directory as
np.save('my_array', x)
```

上述代码将 `x` ndarray 保存到叫做 `my_array.npy` 的文件中。你可以使用 `load()` 函数将保存的 ndarray *加载*到变量中。

```
# We load the saved array from our current directory into variable y
y = np.load('my_array.npy')

# We print y
print()
print('y = ', y)
print()

# We print information about the ndarray we loaded
print('y is an object of type:', type(y))
print('The elements in y are of type:', y.dtype)
```

> y is an object of type: class 'numpy.ndarray' The elements in y are of type: int64

从文件中加载数组时，确保包含文件名和扩展名 `.npy` ，否则将出错。

下一项