Effective Programming Practices for Economists

Scientific Computing

Creating arrays

Janoś Gabler and Hans-Martin von Gaudecker

Creating arrays from lists

Flat list

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

Nested list

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

Mixed dtypes in list

```
>>> np.array([[1, 2], [3.1, 4])
array([[1. , 2. ],
       [3.1, 4. ]])
```

- Can use flat or (deeply) nested lists
- Lists length cannot be rugged
- List might have mixed dtypes, arrays will not!

Constructors

```
>>> np.ones(3)
array([1., 1., 1.])

>>> np.ones((2, 2))
array([[1., 1.],
       [1., 1.]])

>>> np.arange(3)
array([0, 1, 2])

>>> np.eye(2)
array([[1., 0.],
       [0., 1.]])
```

- `np.ones_like`
- `np.zeros` and `np.zeros_like`
- `np.empty`
- `np.linspace`
- `np.full`
- More in the (documentation)
- Learn them like vocabulary!

Reshaping

```
>>> a = np.array([1, 2, 3, 4, 5, 6])
>>> b = a.reshape((2, 3))
>>> b
array([[1, 2, 3],
       [4, 5, 6]])

>>> b.shape
(2, 3)
```

- Reshape can transform arrays from one shape to another
- Shapes are denoted as in matrices, i.e. (`n_rows`, `n_cols`)
- Number of elements must not change
- Elements aranged in row-major order

Repeating arrays

```
>>> a = np.array([1, 2, 3])
>>> a.repeat(2)

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

>>> b = np.array([[1, 2], [3, 4]])
>>> b.repeat(2)

array([1, 1, 2, 2, 3, 3, 4, 4])

>>> b.repeat(2, axis=1)

array([[1, 1, 2, 2],
       [3, 3, 4, 4]])

>>> a.reshape(-1, 1).repeat(2, axis=1)

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

- `repeat` duplicates elements n times
- Without axis, result is flattened
- Versatile together with reshaping
- Tip for complex cases:
  - First introduce new dimensions via reshaping
  - Then repeat along these axes
- Always prefer broadcasting over repetition!