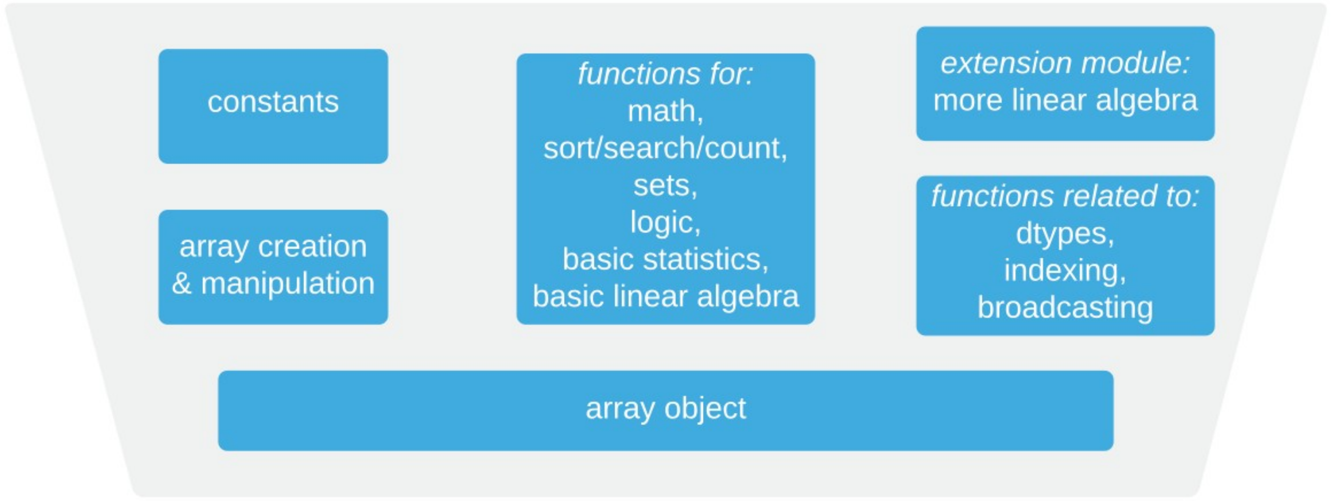
Python Ragged array library development: a final report

Oleksii Hrechykha

Mentors:
Jim Pivarski
Ianna Osborne

Array API



Out of scope

Behavior and functions/objects that aren't common or cannot be fully specified are out of scope.

This doesn't mean libraries cannot implement them, only that they're not part of the standard.

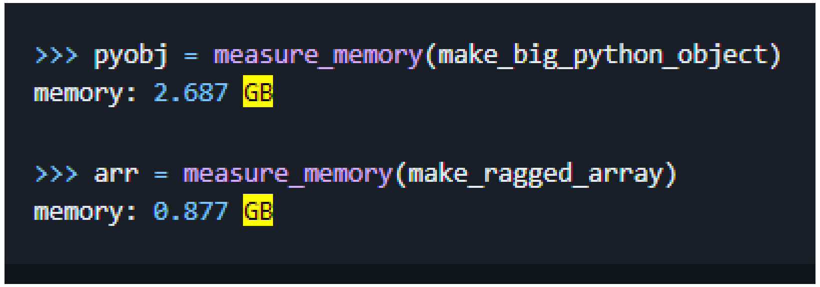
- *I/O*
- *Polynomials*
- *Error handling*
- *Testing & building*
- *CFFI/ctypes*
- *Datetime dtypes*
- *String dtype*

https://data-apis.org/array-api/latest/purpose_and_scope.html

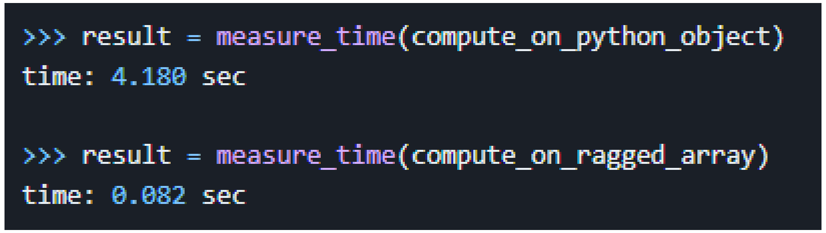
Awkward and ragged array libraries

- are time and memory-efficient
- take up less storage

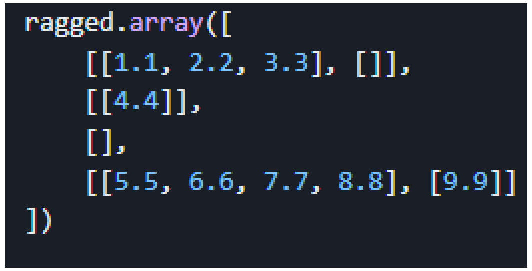
- Awkward uses numpy and has a similar effect on performance
- but it allows for arbitrary data (i.e. supports numbers, dates, strings, record structures)
- ragged uses Awkward but is API-complacent



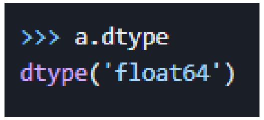
```
>>> pyobj = measure_memory(make_big_python_object)
memory: 2.687 GB

>>> arr = measure_memory(make_ragged_array)
memory: 0.877 GB
```



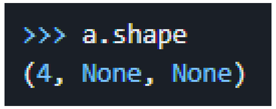
```
>>> result = measure_time(compute_on_python_object)
time: 4.180 sec

>>> result = measure_time(compute_on_ragged_array)
time: 0.082 sec
```



```
ragged.array([
    [[1.1, 2.2, 3.3], []],
    [[4.4]],
    [],
    [[5.5, 6.6, 7.7, 8.8], [9.9]]
])
```



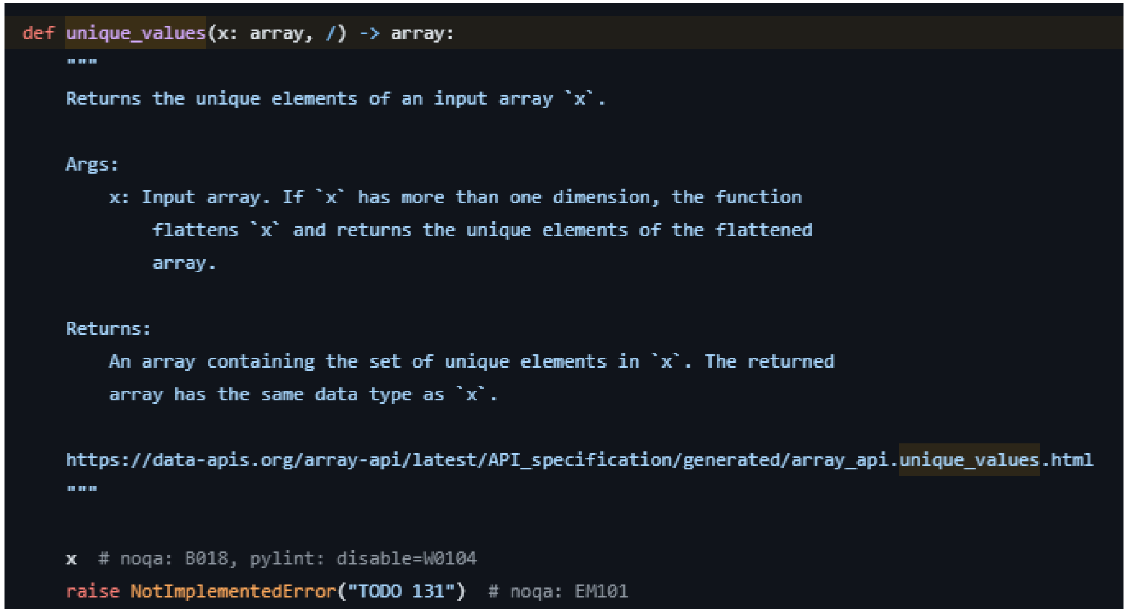
```
>>> a.dtype
dtype('float64')
```



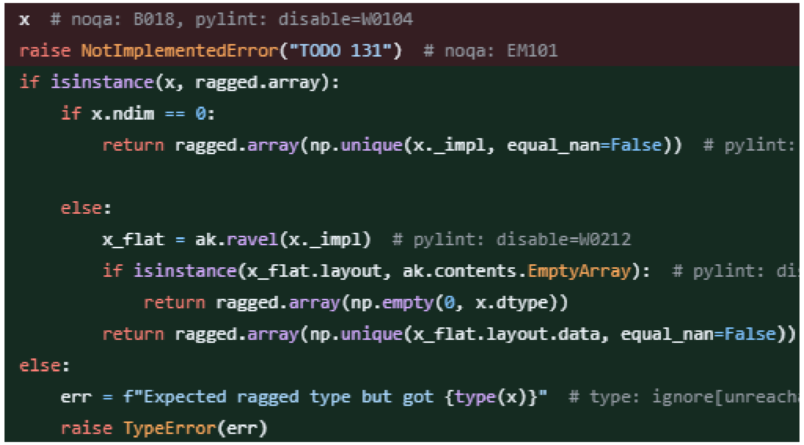
```
>>> a.shape
(4, None, None)
```

https://github.com/scikit-hep/ragged/blob/fff53544be6f9ded884e112d41ba476751b84085/README.md

Example: unique_values function



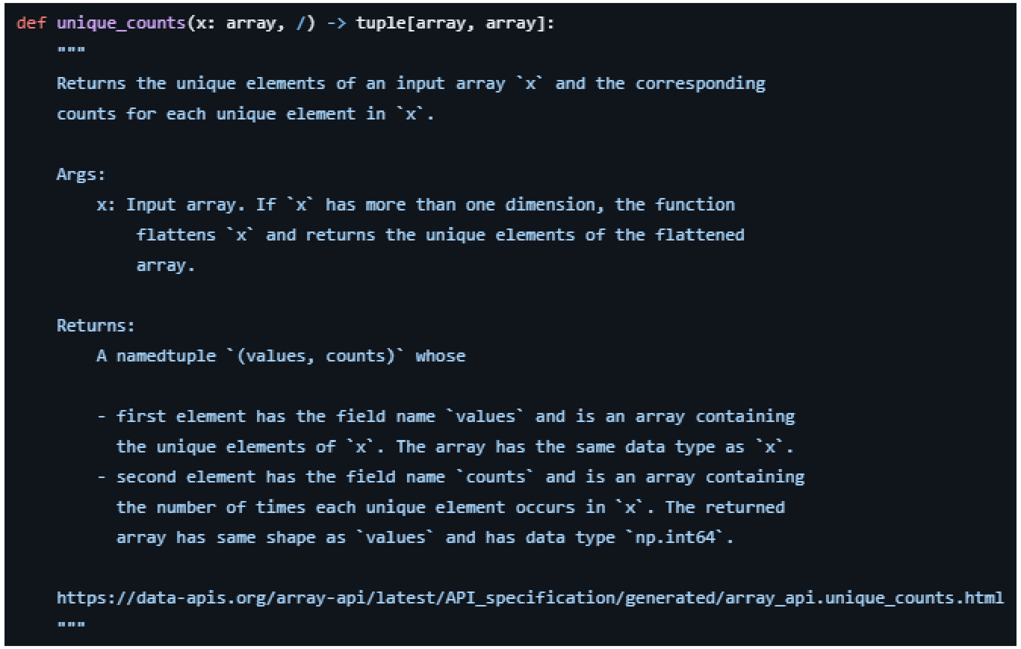
```
def unique_values(x: array, /) -> array:
    """
    Returns the unique elements of an input array `x`.

    Args:
        x: Input array. If `x` has more than one dimension, the function
            flattens `x` and returns the unique elements of the flattened
            array.

    Returns:
        An array containing the set of unique elements in `x`. The returned
        array has the same data type as `x`.

    https://data-apis.org/array-api/latest/API_specification/generated/array_api.unique_values.html
    """

    x  # noqa: B018, pylint: disable=W0104
    raise NotImplementedError("TODO 131")  # noqa: EM101
```



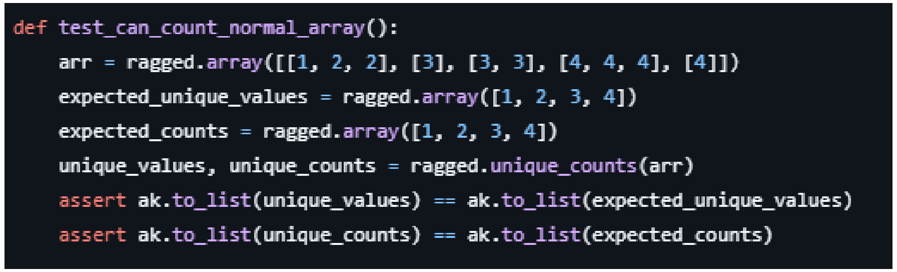
```
x  # noqa: B018, pylint: disable=W0104
raise NotImplementedError("TODO 131")  # noqa: EM101
if isinstance(x, ragged.array):
    if x.ndim == 0:
        return ragged.array(np.unique(x._impl, equal_nan=False))  # pylint:

    else:
        x_flat = ak.ravel(x._impl)  # pylint: disable=W0212
        if isinstance(x_flat.layout, ak.contents.EmptyArray):  # pylint: di
            return ragged.array(np.empty(0, x.dtype))
        return ragged.array(np.unique(x_flat.layout.data, equal_nan=False))
else:
    err = f"Expected ragged type but got {type(x)}"  # type: ignore[unreach
    raise TypeError(err)
```

- A function that returns the unique elements of an input array, the first occurring indices for each unique element in this array, the indices from the set of unique elements that reconstruct it, and the corresponding `counts` for each unique element ("TODO 128")
- A function that returns the unique elements of an input array and the corresponding counts for each unique element in this array ("TODO 129").
- A function that returns the unique elements of an input array and the indices from the set of unique elements that reconstruct it ("TODO 130")
- A function that returns the unique elements of an input array ("TODO 131").

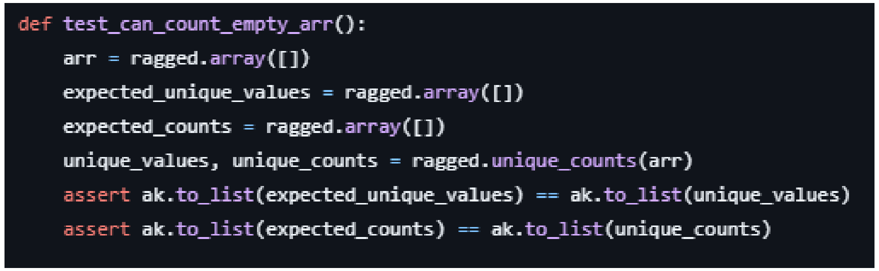
unique_counts function and tests



```python
def unique_counts(x: array, /) -> tuple[array, array]:
    """
    Returns the unique elements of an input array `x` and the corresponding
    counts for each unique element in `x`.

    Args:
        x: Input array. If `x` has more than one dimension, the function
            flattens `x` and returns the unique elements of the flattened
            array.

    Returns:
        A namedtuple `(values, counts)` whose

        - first element has the field name `values` and is an array containing
          the unique elements of `x`. The array has the same data type as `x`.
        - second element has the field name `counts` and is an array containing
          the number of times each unique element occurs in `x`. The returned
          array has same shape as `values` and has data type `np.int64`.

    https://data-apis.org/array-api/latest/API_specification/generated/array_api.unique_counts.html
    """
```

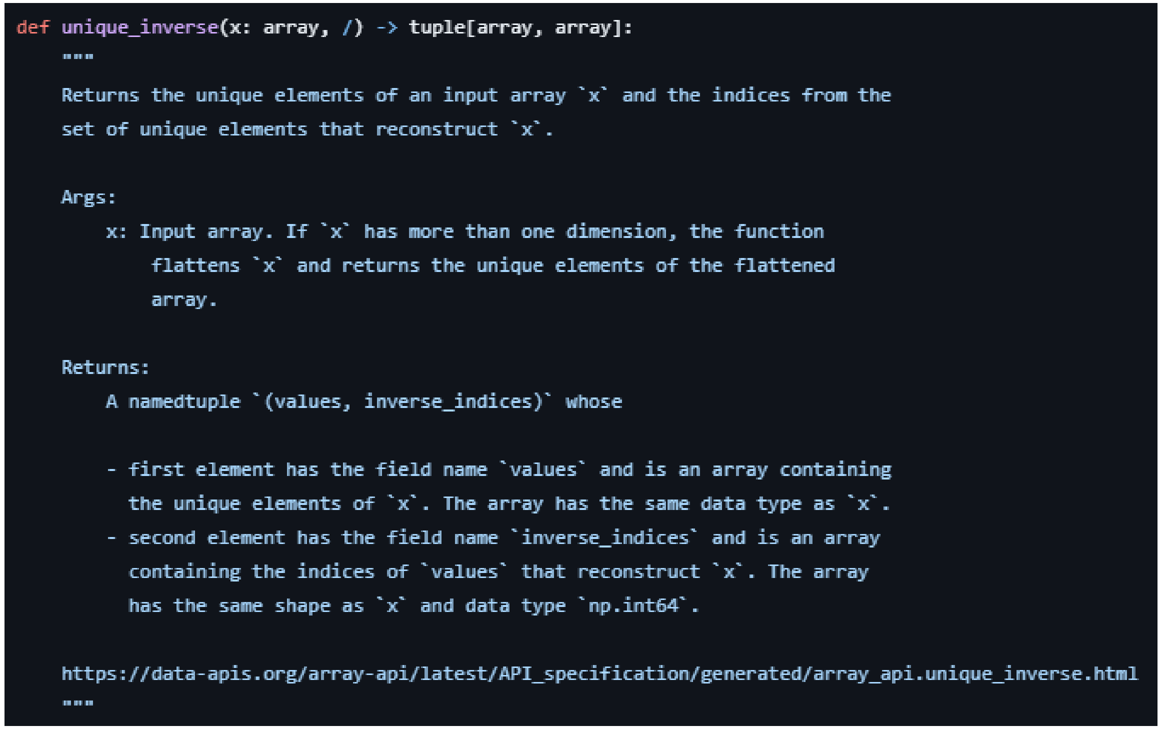


```python
def test_can_count_normal_array():
    arr = ragged.array([[1, 2, 2], [3], [3, 3], [4, 4, 4], [4]])
    expected_unique_values = ragged.array([1, 2, 3, 4])
    expected_counts = ragged.array([1, 2, 3, 4])
    unique_values, unique_counts = ragged.unique_counts(arr)
    assert ak.to_list(unique_values) == ak.to_list(expected_unique_values)
    assert ak.to_list(unique_counts) == ak.to_list(expected_counts)
```

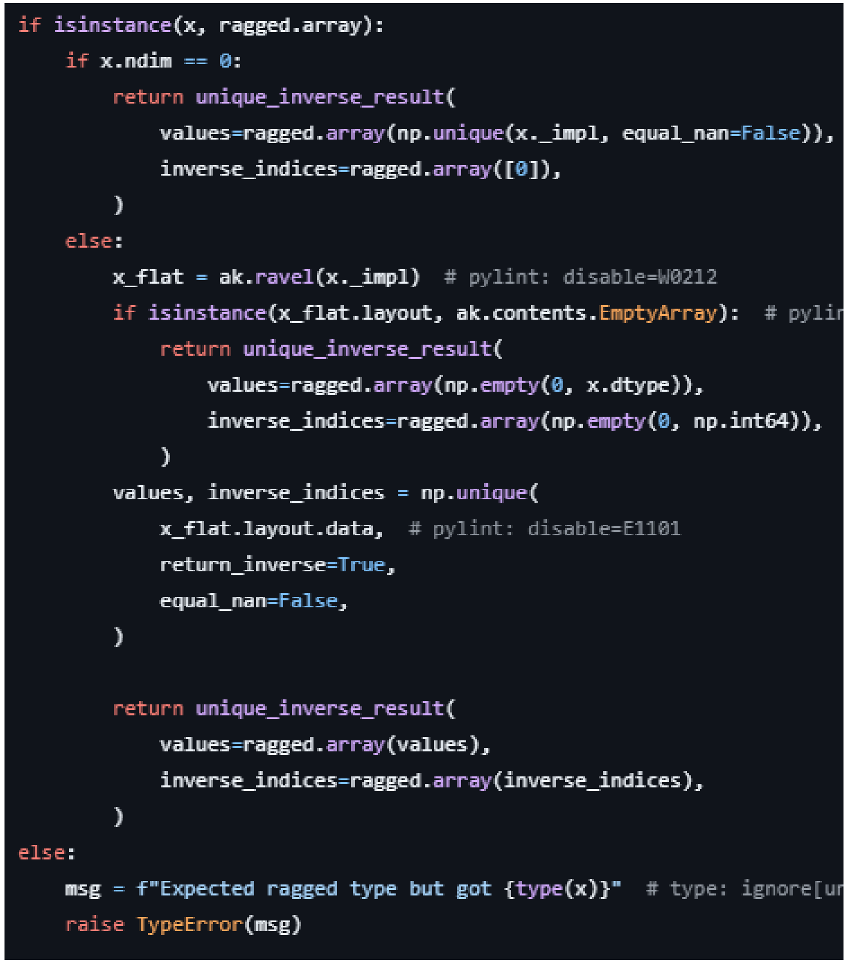


```python
def test_can_count_empty_arr():
    arr = ragged.array([])
    expected_unique_values = ragged.array([])
    expected_counts = ragged.array([])
    unique_values, unique_counts = ragged.unique_counts(arr)
    assert ak.to_list(expected_unique_values) == ak.to_list(unique_values)
    assert ak.to_list(expected_counts) == ak.to_list(unique_counts)
```

unique_inverse function

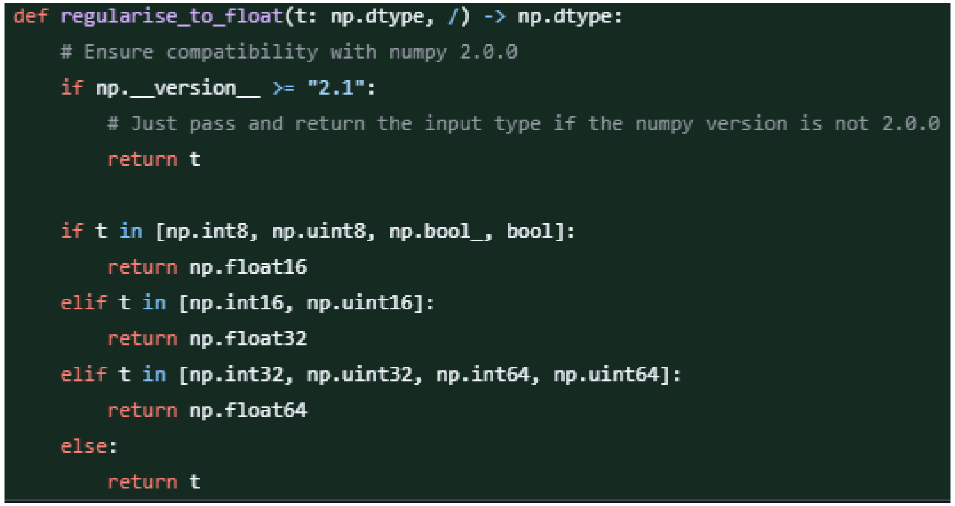


```python
def unique_inverse(x: array, /) -> tuple[array, array]:
    """
    Returns the unique elements of an input array `x` and the indices from the
    set of unique elements that reconstruct `x`.

    Args:
        x: Input array. If `x` has more than one dimension, the function
            flattens `x` and returns the unique elements of the flattened
            array.

    Returns:
        A namedtuple `(values, inverse_indices)` whose

        - first element has the field name `values` and is an array containing
          the unique elements of `x`. The array has the same data type as `x`.
        - second element has the field name `inverse_indices` and is an array
          containing the indices of `values` that reconstruct `x`. The array
          has the same shape as `x` and data type `np.int64`.

    https://data-apis.org/array-api/latest/API_specification/generated/array_api.unique_inverse.html
    """
```

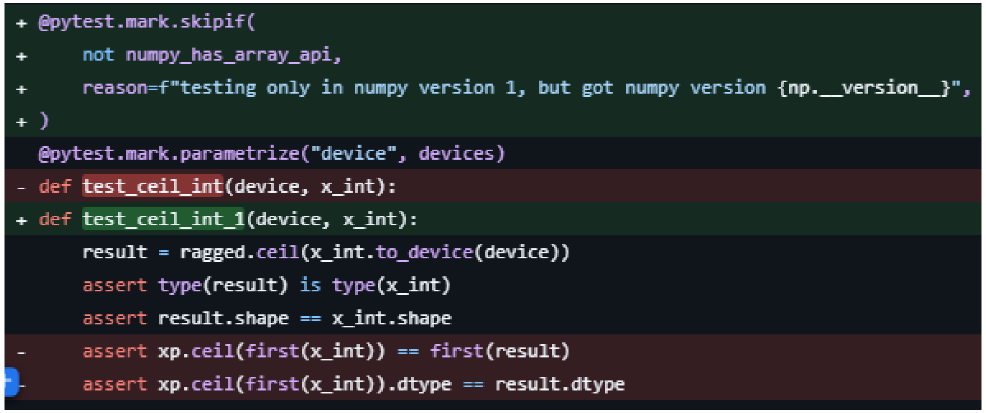


```python
if isinstance(x, ragged.array):
    if x.ndim == 0:
        return unique_inverse_result(
            values=ragged.array(np.unique(x._impl, equal_nan=False)),
            inverse_indices=ragged.array([0]),
        )
    else:
        x_flat = ak.ravel(x._impl)  # pylint: disable=W0212
        if isinstance(x_flat.layout, ak.contents.EmptyArray):  # pylir
            return unique_inverse_result(
                values=ragged.array(np.empty(0, x.dtype)),
                inverse_indices=ragged.array(np.empty(0, np.int64)),
            )
        values, inverse_indices = np.unique(
            x_flat.layout.data,  # pylint: disable=E1101
            return_inverse=True,
            equal_nan=False,
        )

        return unique_inverse_result(
            values=ragged.array(values),
            inverse_indices=ragged.array(inverse_indices),
        )
else:
    msg = f"Expected ragged type but got {type(x)}"  # type: ignore[ur
    raise TypeError(msg)
```

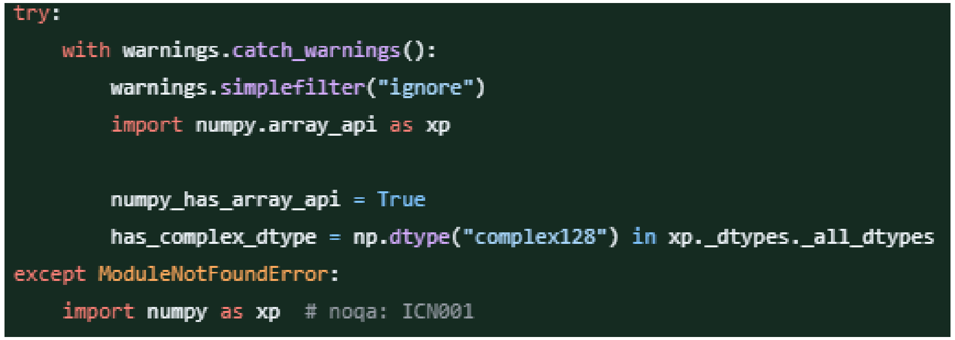
Side project: port to Numpy 2



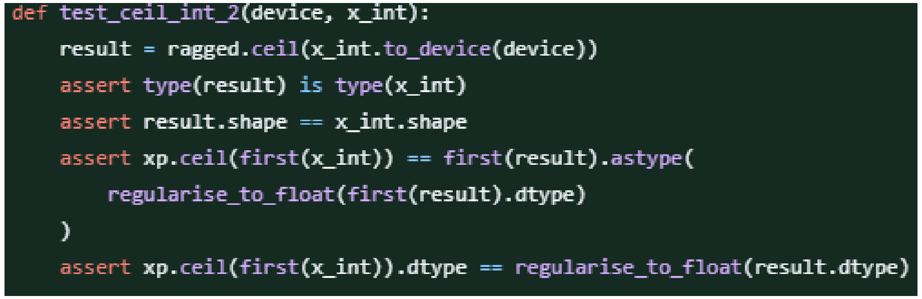
```python
def regularise_to_float(t: np.dtype, /) -> np.dtype:
    # Ensure compatibility with numpy 2.0.0
    if np.__version__ >= "2.1":
        # Just pass and return the input type if the numpy version is not 2.0.0
        return t

    if t in [np.int8, np.uint8, np.bool_, bool]:
        return np.float16
    elif t in [np.int16, np.uint16]:
        return np.float32
    elif t in [np.int32, np.uint32, np.int64, np.uint64]:
        return np.float64
    else:
        return t
```



```diff
+ @pytest.mark.skipif(
+     not numpy_has_array_api,
+     reason=f"testing only in numpy version 1, but got numpy version {np.__version__}",
+ )
  @pytest.mark.parametrize("device", devices)
- def test_ceil_int(device, x_int):
+ def test_ceil_int_1(device, x_int):
      result = ragged.ceil(x_int.to_device(device))
      assert type(result) is type(x_int)
      assert result.shape == x_int.shape
-     assert xp.ceil(first(x_int)) == first(result)
-     assert xp.ceil(first(x_int)).dtype == result.dtype
```



```python
try:
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        import numpy.array_api as xp

        numpy_has_array_api = True
        has_complex_dtype = np.dtype("complex128") in xp._dtypes._all_dtypes
except ModuleNotFoundError:
    import numpy as xp  # noqa: ICN001
```



```python
def test_ceil_int_2(device, x_int):
    result = ragged.ceil(x_int.to_device(device))
    assert type(result) is type(x_int)
    assert result.shape == x_int.shape
    assert xp.ceil(first(x_int)) == first(result).astype(
        regularise_to_float(first(result).dtype)
    )
    assert xp.ceil(first(x_int)).dtype == regularise_to_float(result.dtype)
```

Thank you for your attention!