# Awkward Array

Jim Pivarski

Princeton University – IRIS-HEP

July 14, 2021

# Awkward Array: NumPy-like idioms on JSON-like data

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

# Awkward Array: NumPy-like idioms on JSON-like data

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

NumPy-like expression

```
output = np.square(array["y", ..., 1:])
```

# Awkward Array: NumPy-like idioms on JSON-like data

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

NumPy-like expression

```
output = np.square(array["y", ..., 1:])
```

```
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

# Awkward Array: NumPy-like idioms on JSON-like data

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

## NumPy-like expression

```
output = np.square(array["y", ..., 1:])
```

```
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

## equivalent Python

```
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(np.square(number))
        tmp1.append(tmp2)
    output.append(tmp1)
```

# Awkward Array: NumPy-like idioms on JSON-like data

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

NumPy-like expression

```
output = np.square(array["y", ..., 1:])
```

```
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

**4.6 seconds to run (2 GB footprint)**

equivalent Python

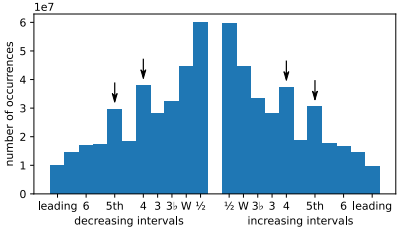
```
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(np.square(number))
        tmp1.append(tmp2)
    output.append(tmp1)
```

**138 seconds to run (22 GB footprint)**

(single-threaded on a 2.2 GHz processor with a dataset 10 million times larger than the one shown)

# Extended example: histogram of intervals in Million Song Dataset

```
import awkward as ak, numpy as np

# read an 80 GB Parquet dataset as a lazy array (through pyarrow)
millionsongs = ak.from_parquet("s3://pivarski-princeton/millionsongs/", lazy=True)

# volume of each half-step pitch (0-11) in each quarter-second segment of each song
pitches = millionsongs.analysis.segments.pitches
# <Array [[[0.294, 0.158, ... 0.083, 1, 0.078]]] type='1000000 * var * var * float64'>

# loudest pitch in each segment (axis=-1 applies to the deepest level of nesting)
loudest_pitches = ak.argmax(pitches, axis=-1)

# change in loudest-pitch from each segment to the next (list lengths vary with song)
intervals = loudest_pitches[:, 1:] - loudest_pitches[:, :-1]

# not including zero change (extremely variable)
nonzero_intervals = intervals[intervals != 0]

# histogram them
np.histogram(
    np.asarray(ak.flatten(nonzero_intervals)),
    bins=np.arange(-11.5, 12.5),
)
```



# Extended example: same thing in Numba (single pass)

```
import awkward as ak, numpy as np, numba as nb

@nb.jit                                          # Numba will JIT-compile this function
def collect_pitch_intervals(millionsongs):
    pitch_intervals = np.zeros(23, np.int64)     # histogram as an array to fill

    for song in millionsongs:                    # iteration over variable-length songs
        previous = None
        for segment in song.analysis.segments:
                                                 # cast as NumPy to use NumPy functions
            loudest_pitch = np.argmax(np.asarray(segment.pitches))

            if previous is not None:
                interval = loudest_pitch - previous
                if interval != 0:                # manually histogram non-zero changes
                    pitch_intervals[interval + 11] += 1

            previous = loudest_pitch             # remember previous for differences

    return pitch_intervals

millionsongs = ak.from_parquet("s3://pivarski-princeton/millionsongs/", lazy=True)
collect_pitch_intervals(millionsongs)            # run JIT-compiled function
```

# Actively extending beyond its original domain (particle physics)

**Variable-length, nested data is a common problem,** so Awkward Array is a general-purpose library for working with nested lists, records, and missing data.

# Actively extending beyond its original domain (particle physics)

**Variable-length, nested data is a common problem,** so Awkward Array is a general-purpose library for working with nested lists, records, and missing data.

But scientists in other fields have revealed gaps in scope:

- Radio astronomers asked for complex numbers: #392, #421, #652, #857, #858
- Data scientists asked for date-times: #913, #911, #909, #835, #367
- ...

# Actively extending beyond its original domain (particle physics)

**Variable-length, nested data is a common problem,** so Awkward Array is a general-purpose library for working with nested lists, records, and missing data.

But scientists in other fields have revealed gaps in scope:

- Radio astronomers asked for complex numbers: #392, #421, #652, #857, #858
- Data scientists asked for date-times: #913, #911, #909, #835, #367
- ...

Starting a 3-year project (NSF OAC-2103945) to ensure that Awkward Array works for use-cases across the sciences.

# Actively extending beyond its original domain (particle physics)

**Variable-length, nested data is a common problem,** so Awkward Array is a general-purpose library for working with nested lists, records, and missing data.

But scientists in other fields have revealed gaps in scope:

- Radio astronomers asked for complex numbers: #392, #421, #652, #857, #858
- Data scientists asked for date-times: #913, #911, #909, #835, #367
- ...

Starting a 3-year project (NSF OAC-2103945) to ensure that Awkward Array works for use-cases across the sciences.

> We're actively seeking to collaborate with scientists who have "awkward" data problems, to ensure that Awkward Array addresses them.

# Actively extending beyond its original domain (particle physics)

**Variable-length, nested data is a common problem,** so Awkward Array is a general-purpose library for working with nested lists, records, and missing data.

But scientists in other fields have revealed gaps in scope:

- Radio astronomers asked for complex numbers: #392, #421, #652, #857, #858
- Data scientists asked for date-times: #913, #911, #909, #835, #367
- ...

Starting a 3-year project (NSF OAC-2103945) to ensure that Awkward Array works for use-cases across the sciences.

> We're actively seeking to collaborate with scientists who have "awkward" data problems, to ensure that Awkward Array addresses them.

... especially if it would lead to better integration with other scientific Python libraries and GPUs. (**Dask integration** and full **GPU support** are major parts of this project.)

# Documentation and contacts

**Install:** `pip install awkward` or
`conda install -c conda-forge awkward`

**Docs:** https://awkward-array.org

**GitHub:** https://github.com/scikit-hep/awkward-1.0

**Gitter:** https://gitter.im/Scikit-HEP/awkward-array

**Contact:** pivarski@princeton.edu

**Postdoc opportunity:** https://puwebp.princeton.edu/AcadHire/apply/application.xhtml?listingId=21021