NumPy More

CAC 350

Pyyytthhooonnn

https://www.nytimes.com/2014/01/28/science/the-sloths-busy-inner-life.html

- Example from book

  import bumpy as np
  np.random.seed(0)

Remind me why we seed

```
def compute_reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0.values[i]
    return output

big_array = np.random.randint(1,100,size=1000000)
%timeit compute_reciprocals(big_array

1 loop, best of 3: 2.91 s per loop
```

Why so slow?

Universal Functions

- Need statically typed, compiled routine => vectorized operation

Definition:

In the context of high-level languages like Python, Matlab, and R, the term **vectorization** describes the use of optimized, pre-compiled code written in a low-level language (e.g. C) to perform mathematical operations over a sequence of data. This is done in place of an explicit iteration written in the native language code (e.g. a “for-loop” written in Python).

https://www.pythonlikeyoumeanit.com/Module3_IntroducingNumpy/VectorizedOperations.html

- ufuncs offer the vectorization we need - primary job is to quickly execute repeated operations on values in NumPy arrays

- unary ufuncs and binary ufuncs

Available ufuncs

Operator	ufunc	Description
+	np.add	Addition
-	np.substract	Substraction
-	np.negative	Unary negative
/	np.divide	Division
//	np.floor_divide	Floor division/Integer
**	np.power	Exponentiation
%	np.mod	Modulus/remainder
abs(x)	np.absolute(x)/np.abs(x)	Absolute value
sin/cos/tan	np.sin/np.cos/np.tan	Trig functions
arcsin/arccos/arctan	np.arcsin/np.arccos/ np.arctan	Trig functions
Exponent	np.exp(x)/np.exp2(x)/ np.power(3,x)	e^x / 2^x / 3^x
Logs	np.log(x)/np.log2(x)/ np.log10(x)	ln(x)/log2(x)/log10(x)

Mapping

- myList = [1,2,3,4,5]
- myList5 = map(lambda x: x*5, myList)
- myList5 = [5,10,15,20,25]
- myList5.sum()
- myList5.min()

Reduce

- Reduce is a function (can be used in general Python) that repeatedly applies a given operation to the elements in an array until only one single result remains
- Example:
  x = np.arange(1,6)
  np.add.reduce(x)

- 15

- What would accumulate do? Let’s see…

Aggregates

- We want to use the ufunc for aggregates as there is a time efficiency difference…let's test multiplication, then, let's test an aggregate function

- The ufunc aggregate differs from the general aggregate… how? why?

  - Speed is one way as we have seen

  - Different arguments - particularly, multidimensional

Aggregates

NaN-Not a Number
Good for missing data

Function Name	NaN-Safe	Description
np.sum	np.nansum	Compute sum
np.prod	np.nanprod	Compute product
np.mean	np.nanmean	Compute mean
np.std	np.nanstd	Compute standard dev
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum
np.max	np.nanmax	Find maximum
np.argmin	np.nanargmin	Find index of min
np.argmax	np.nanargmax	Find index of max
np.median	np.nanmedian	Compute median
np.percentile	np.nanpercentile	Compute rank-based stats
np.any	N/A	Any elements True
np.all	N/A	All elements True

Activity Tips

- Use pandas to import your data:
  import pandas as pd
  data = pd.read_csv(‘file.csv’)
  magnitude = np.array(data[‘magnitude’])

- Use a dataframe to get one column based on another:
  df = pd.DataFrame(data)
  limited = df.loc[df['colName'] == 'val', ‘colName’].to_numpy()

- To perform a reverse geocode, you’ll need to install the reverse_geocode package
  - If using pip: pip install reverse_geocode
  - You may need to close Jupyter Notebook and restart
  - Article about installing packages: https://jakevdp.github.io/blog/2017/12/05/installing-python-packages-from-jupyter/

Activity Tips

In [1]:
```
import reverse_geocode
coordinates = (-37.81, 144.96), (31.76, 35.21)
```

In [2]:
```
print(coordinates)
```

```
((-37.81, 144.96), (31.76, 35.21))
```

In [3]:
```
reverse_geocode.search(coordinates)
```

Out[3]:
```
[{'country_code': 'AU', 'city': 'Melbourne', 'country': 'Australia'},
 {'country_code': 'IL', 'city': 'Jerusalem', 'country': 'Israel'}]
```

In [7]:
```
coordinates = (19.18816757,-155.4616699),(19.18816757,-155.4616699)
reverse_geocode.search(coordinates)
```

Out[7]:
```
[{'country_code': 'US', 'city': 'Pāhala', 'country': 'United States'},
 {'country_code': 'US', 'city': 'Pāhala', 'country': 'United States'}]
```