UMpy: Lambdas

Goals

1. Awareness/Literacy: write anonymous lambda functions.
2. Awareness/Literacy: Sort sequences using `list.sort()` and built-in function `sorted()` and lambdas as the key function for each.
3. Review: Write list comprehensions.
4. Awareness: Use `operator.itemgetter()` as a key function and alternative to a lambda.
5. Awareness: import a custom module named `umpy_utils`.

Glossary

Source: https://docs.python.org/3/glossary.html, https://docs.python.org/3/library/functools.html.

1. **Higher order function**: a function that acts on or returns other functions. Built-in functions such as `map()`, `filter()`, and `sorted()` are consider higher-order functions as are the method `list.sort()` and the function `functools.reduce()`.

2. **key function**: A key function or collation function is a callable that returns a value used for sorting or ordering. Both `list.sort()` and the built-in function `sorted()` can be passed a key function such as a lambda as an optional argument in order to specify how a sequence is to be ordered.

3. **lambda**: An anonymous inline function consisting of a single expression which is evaluated when the function is called.

Lambda functions

A Python *lambda* function is an anonymous function (i.e., a function defined without recourse to the `def` keyword) that specifies one or more parameters and a single expression that acts on the arguments provided it. The syntax to create a lambda function is

```
lambda < parameter or comma-separated set of parameters >: < expression >
```

❗ Lambda syntax *does not* include a `return` statement. A Lambda function returns the expression defined for it *not* the anticipated value. A lambda can be passed to another expression and can be assigned to a variable.

You can experiment with lambdas by starting a terminal session and running the Python interactive console (a.k.a Python shell). Then create and call the following lambda functions.

```
python3   <- or python for Win users

>>> (lambda x: x * 10)(5) # expression returned first followed by a value
passed to it
50
```

```
>>> y = lambda x: x % 2 # expression assigned to the variable y

>>> y(30) # call y passing the value 30 to it
0

>>> y(17)
1

>>> addup = lambda x: sum(x) # sum takes an iterable

>>> addup([6, 50, 100, 150, 200])
506

>>> splitter = lambda x: x.split() # lambdas are not limited to processing
numeric values

>>> words = splitter("You don't need a weatherman to know which way the
wind blows.")

>>> print(words)
['You', "don't", 'need', 'a', 'weatherman', 'to', 'know', 'which', 'way',
'the', 'wind', 'blows.']

>>> dylan = sorted(words, key=lambda x: len(x), reverse=True) # key
function determines sort order

>>> print(dylan)
['weatherman', 'blows.', "don't", 'which', 'need', 'know', 'wind', 'You',
'way', 'the', 'to', 'a']
```

Sorting with list.sort() and sorted()

Both `list.sort()` and the built-in function `sorted()` define a `key` parameter that permits a key function such as a lambda or other callable to be passed as an argument in order to specify a custom sort order.

❗ Recall that `list.sort()` performs an *in-place* operation that mutates the current list while implicitly returning `None`. The built-in function `sorted()` exhibits different behavior. It returns a *new* list based on the list passed to it and sorted according to the key function (if any) defined for it.

`sorted()`

Signature: https://docs.python.org/3/library/functions.html#sorted

```
sorted(iterable, *, key=None, reverse=False)
```

```
countries = [
    ('Lesotho', 'LSO'),
```

```
    ('South Africa', 'ZAF'),
    ('Botswana', 'BWA'),
    ('Eswatini', 'SWZ'),
    ('Namibia', 'NAM')
    ]

southern_africa = sorted(countries)
print(f"\nWarm up: southern_africa, default sort = {southern_africa}")

southern_africa = sorted(countries, key=lambda x: x[1])

print(f"\nWarm up: southern_africa, ISO Alpha3 code sort =
{southern_africa}")

southern_africa = sorted(countries, key=lambda x: x[1], reverse=True)

print(f"\nWarm up: southern_africa, ISO Alpha3 code sort (reversed) =
{southern_africa}")
```

`list.sort()`

Signature: https://docs.python.org/3/tutorial/datastructures.html

```
list.sort(*, key=None, reverse=False)
```

❗ `list.sort()` returns `None` and *not* a new list.

```
central_america = [
    ('Nicaragua', 'NIC', 6.105),
    ('El Salvador', 'SLV', 6.253),
    ('Honduras', 'HND', 5.860),
    ('Costa Rica', 'CRI', 7.167),
    ('Mexico', 'MEX', 6.595),
    ('Belize', 'BLZ', 0.0),
    ('Panama', 'PAN', 6.321),
    ('Guatemala', 'GTM', 6.436)
]

central_america.sort(key=lambda x: x[-1], reverse=True)
print(f"\nWARM UP: Central America, happiness score (reversed) =
{central_america}")
```

Challenges

For today's challenges you will work with country data for year 2019 drawn from The World Happiness Report and sourced from Kaggle. The data was enriched with the regional affiliation assigned to each country per the United Nations M49 standard. The accompanying `./input/happiness-shuffled-`

`unranked-2019.csv` file
contains the following columns:

1. Country
2. Region
3. Score
4. GDP per capita
5. Social support
6. Healthy life expectancy
7. Freedom to make life choices
8. Generosity
9. Perceptions of corruption

Challenge 01

1. Call the function `read_csv` from the imported module `umpy_utils` and use it to retrieve the data contained in `happiness-shuffled-unranked-2019.csv`. Assign the list retrieved to a variable named `data`.
2. Access the header row from `data` and assign it to a variable named `headers`.
3. Access the country elements in `data` and assign them to a variable named `countries`.

Challenge 02

1. Perform an in-place sort of `countries` (default lexicographic order).
2. Write a list comprehension that returns a new list of tuples comprising the country name and score `(< country >, < happiness score >)`. Limit the number of elements returned to the first ten (10) countries in the `countries` list. Assign the new list to a variable named `printable`.
3. Call the built-in function `print()` and pass to it `printable` (uncomment code provided).

Challenge 03

1. Repoint the `countries` label to the original country `data` (i.e., perform a variable reassignment).
2. Sort `countries` in-place based on the happiness score in ascending order. Write a lambda function to effect the custom happiness score sort. Convert the happiness score to a `float` in the lambda expression.
3. Write a list comprehension that returns a new list of tuples comprising the country name and score `(< country >, < happiness score >)`. Limit the number of elements returned to the first ten (10) countries in the `countries` list. Assign the new list to a variable named `printable`.
4. Call the built-in function `print()` and pass to it `printable` (uncomment code provided).

Challenge 04

Repeat Challenge 03 steps above with a single variation. For step two (2) sort the countries in-place based on the happiness score but do so in *descending* order.

 An alternative approach using `operator.itemgetter()` will also be shared.

Challenge 05

1. Repoint the `countries` label to the original country `data` (i.e., perform a variable reassignment).
2. Employ the built-in function `sorted()` and a lambda function to sort the `countries` based on the social support score in descending order. Convert the social support score to a `float` in the lambda expression.
3. Write a list comprehension that returns a new list of tuples comprising the country name and score `(< country >, < soc support score >)`. Limit the number of elements returned to the first ten (10) countries in the `countries` list. Assign the new list to a variable named `printable`.
4. Call the built-in function `print()` and pass to it `printable` (uncomment code provided).

 An alternative approach using `operator.itemgetter()` will also be shared.

Challenge 06

1. Repoint the `countries` label to the original country `data` (i.e., perform a variable reassignment).
2. Employ the built-in function `sorted()` and a lambda function to sort the `countries` based on region, happiness score, and country in descending order. Convert the happiness score to a `float` in the lambda expression.
3. Write a list comprehension that returns a new list of tuples comprising the region, country name and score `(<region>, < country >, < happiness score >)` for countries with 'Europe' in the `region` name. Assign the new list to a variable named `writable`.
4. Write the new list to a file employing the path `'./output/europe-happiness.csv'`. Call `write_csv` from the `umpy_utils` module.

Challenge 07

Repeat Challenge 06 steps above with a couple of variations. For step two (2) sort by region, happiness score, and country but sort the score in *descending* order. For step 3 write a list comprehension that filters on countries in `Sub-Saharan Africa`.

Write the new list to a file employing the path `'./output/southern_africa-happiness.csv'`. Call `write_csv` from the `umpy_utils` module.

 An alternative approach using a custom function will also be shared.

Challenge 08 (Bonus)

1. Repoint the `countries` label to the original country `data` (i.e., perform a variable reassignment).
2. Employ the built-in function `sorted()` and a lambda function to sort the `countries` based on happiness score in descending order. Convert the happiness score to a `float` in the lambda expression.
3. Create a new list based on `countries` that adds a "Rank" column for each country based on their happiness score. Create the new list with a `for` loop or a list comprehension and assign it to a variable named `rankings`.
4. Write the new list to a file employing the path `'./output/world_rank-happiness-loop.csv'`. Call `write_csv` from the `umpy_utils` module.

Recommended reading

Andrew Dalke and Raymond Hettinger, "Sorting HOW TO" (Python official documentation, n.d.).

David Fundakowski, "How to Use sorted() and sort() in Python" (Real Python, n.d.).

Priyankur Sarkar, "How to Use Python Lambda Functions" (knowlege hut, Aug 2019).