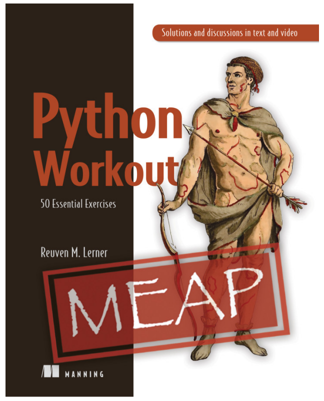
How to sort anything

Reuven M. Lerner • Euro Python 2020

reuven@lerner.co.il • @reuvenmlerner

I teach Python

- Corporate training
- Video courses about Python + Git
- Weekly Python Exercise
  - More info at https://lerner.co.il/
- "Python Workout" — published by Manning
- https://PythonWorkout.com
- "Better developers" — free, weekly newsletter about Python
  - https://BetterDevelopersWeekly.com/



Sorting is important!

Why sort?

- Display data nicely
- Make messy data (slightly) less messy
- Find the largest (or smallest) value in a collection
- See which products sold best (or worst)
- Which supplier's proposal will cost you the most?
- Find the closest gas station to your current location
- Find the a similar films to the one you've just watched
- Find the most similar products to the one you're looking at

Python makes sorting easy

- If you have a list, then you can use the “sort” method:

```
mylist = [10, 5, -3, 7, -2, 4]
print(f'Before, {mylist=}')
mylist.sort()
print(f'After, {mylist=}’)
```

```
Before, mylist=[10, 5, -3, 7, -2, 4]
After, mylist=[-3, -2, 4, 5, 7, 10]
```

About list.sort

- It's a list method, so it only works on lists
- It sorts from smallest to largest (by default)
- It changes the list object itself!

```
mylist = [10, 5, -3, 7, -2, 4]
also_mylist = mylist

print(f'Before, {also_mylist=}')
mylist.sort()
print(f'After, {also_mylist=}')
```

```
Before, also_mylist=[10, 5, -3, 7, -2, 4]
After, also_mylist=[-3, -2, 4, 5, 7, 10]
```

list.sort returns None

```
mylist = [10, 5, -3, 7, -2, 4]
print(f'Before, {mylist=}')
mylist = mylist.sort()
print(f'After, {mylist=}')
```

```
Before, mylist=[10, 5, -3, 7, -2, 4]
After, mylist=None
```

Better than list.sort: sorted

- A builtin function (not a method)
- Works with all iterables — not just lists!
- Always returns a list, sorted lowest to highest (by default)
- Doesn’t modify the source data at all

Using sorted

```python
mylist = [10, 5, -3, 7, -2, 4]

print(sorted(mylist))
print(f'After, {mylist=}')
```

```
[-3, -2, 4, 5, 7, 10]
After, mylist=[10, 5, -3, 7, -2, 4]
```

How is this all being sorted?

- What sort algorithm is being used here?
- Hint: It was invented by Tim Peters.
- That's right: Timsort!
- Timsort assumes that real-world data contains "natural runs"
  - Given some runs, Timsort merges them
  - If there aren't any runs, then it uses insertion sort to add them
- In this way, Timsort is a mix of merge and insertion sorts

Comparing items

- Given items A and B, we'll thus need to know which is true:
  - A < B
  - A > B
  - A == B
- When merging or inserting, Timsort will rely on this comparison
- If we have a sequence of numbers, then we can just use Python's <, >, and == operators. And indeed, we saw that earlier!

Sorting a list of strings

```
words = 'this is a bunch of words'.split()

print(sorted(words))

['a', 'bunch', 'is', 'of', 'this', 'words']
```

How does this work?

- One-character strings can be compared with <
    - The comparison is based on the Unicode code point for the one-character string (i.e., character)
- To compare multi-character strings, we compare the characters at index 0.
    - Does word1[0] < word2[0]? Then word1 comes first.
    - Does word1[0] > word2[0]? Then word2 comes first.
- If they're the same, then try again with index 1, continuing until you work your way through the string.
- If they're equal, then return word1.
- If one is a substring of the other, then return the shorter string.

Sound familiar?

- If you've ever looked up words in a dictionary, then you've used a version of this algorithm.
- It turns out that this works on all Python sequences!
    - Lists of strings
    - Lists of lists
    - Lists of tuples
- Lists and tuples implement < in the same way!

Comparing lists

```
list1 = [10, 20, 30]
list2 = [10, 20, 15]

print(list1 < list2)    False
print(list1 > list2)    True
```

Lists containing different types

```
mylist = [20, 'b', 'a', 10, 30]
print(sorted(mylist))

Traceback (most recent call last):
  File "./slide7.py", line 3, in <module>
    print(sorted(mylist))

TypeError: '<' not supported between
instances of 'str' and ‘int'
```

Reversing the direction

```
mylist = [20, 30, 10]
print(sorted(mylist, reverse=True))
```

```
[30, 20, 10]
```

Sorting by word length

- What if we want to sort a list of words… but by their lengths?
- We no longer want Timsort to compare this:

```
word1 < word2
```

- Rather, we want Timsort to compare this:

```
len(word1) < len(word2)
```

- Note: We don't want to sort the lengths! We want to use the lengths to sort the words.

The "key" parameter

- Given a function "f", if we want to compare

```
f(A) < f(B)
```

- We can call "sorted" with "key=f"
- Because we want to sort the words by length, we can call "sorted" with "key=len"

Using “key”

```
words = 'this is a bunch of words'.split()
print(sorted(words, key=len))
```

```
['a', 'is', 'of', 'this', 'bunch','words']
```

What can be a key function?

- Any function that takes a single argument, and returns a value that can be compared with <.
- Examples:
  - sorted(words, key=len): Sort words by length
  - sorted(numbers, key=abs): Sort numbers by absolute value
  - sorted(words, key=str.lower): Sort words, ignoring case
- Notice that we can pass a method by passing it as a class attribute.

Don’t execute the key function!

- It’s a common mistake to use parentheses after the key function’s name.
- Bad:

```
sorted(numbers, key=abs())
```

- Good:

```
sorted(numbers, key=abs)
```

- That’s because we have to pass a callable (function or class) to “key”. “abs” is a function, but the result is an int.. not that it’ll work this way…

Sorting lists of lists

- What if I have a list of lists (or a list of tuples), and want to sort them by length?
  - Just use "key=len"
  - (Yes, just like with strings)
- What if I want to sort them by the sum of numbers?
  - Use "key=sum"

Custom key functions

- We can pass our own functions to "key"!
- The function takes one argument, an element in what we're sorting
- The function's return value is how that element will be sorted
  - This value must be sortable
  - This value doesn't need to be of the same type as the input

Example: Sort integers by the number of digits

```
numbers = [500, 2000, 100,
           1, 30, 1000, 40]

def by_digit_count(n):
    return len(str(n))

print(sorted(numbers,
             key=by_digit_count))
```

```
[1, 30, 40, 500, 100, 2000, 1000]
```

Sorting sublists by their means

```
numbers = [[5, 7, 3, 4], [2, 4, 6, 7],
           [1, 3, 5], [10, 1, 1, 1]]


def by_mean(one_list):
    return sum(one_list) / len(one_list)

print(sorted(numbers, key=by_mean))
```

```
[[1, 3, 5], [10, 1, 1, 1],
 [5, 7, 3, 4], [2, 4, 6, 7]]
```

Sorting by vowels per word

```
words = 'this here is a fascinating,
scintillating test'.split()

def by_vowel_count(word):
    print(f'Checking {word}')
    total = 0
    for one_letter in word.lower():
        if one_letter in 'aeiou':
            total += 1

    return total

print(sorted(words, key=by_vowel_count))
```

```
Checking this
Checking here
Checking is
Checking a
Checking fascinating,
Checking scintillating
Checking test
['this', 'is', 'a', 'test', 'here',
'fascinating,', 'scintillating']
```

Sorting filenames by file length

```
import glob
import os

def by_file_length(filename):
    return os.stat(filename).st_size

print(sorted(glob.glob('*.txt'),
             key=by_file_length))
```

```
['nums.txt', ‘wcfile.txt',
 'shoe-data.txt', 'linux-etc-passwd.txt',
'mini-access-log.txt']
```

Sorting filenames by the file's vowel count

```
import glob

def by_vowel_count(filename):
    total = 0
    for one_line in open(filename):
        for one_character in one_line.lower():
            if one_character in 'aeiou':
                total += 1

    return total


print(sorted(glob.glob('*.txt'),
             key=by_vowel_count))
```

What about a list of dicts?

```
people =
  [{'first': 'Atara', 'last': 'Lerner-Friedman', 'age': 19},
   {'first': 'Shikma', 'last': 'Lerner-Friedman', 'age': 17},
   {'first': 'Amotz', 'last': 'Lerner-Friedman', 'age': 14},
   {'first': 'Reuven', 'last': 'Lerner', 'age': 50}
  ]
```

- Can I sort this list of dicts?

Nope.

```
Traceback (most recent call last):
  File "./slide16.py", line 8, in <module>
    print(sorted(people))
TypeError: '<' not supported between instances of
'dict' and 'dict'
```

Solution: A key function!

- For example, we can sort by age:

```
def by_age(d):
    return d['age']

print(sorted(people, key=by_age))
```

```
[
 {‘first': 'Amotz', 'last': 'Lerner-Friedman',
'age': 14},
 {'first': 'Shikma', 'last': 'Lerner-Friedman',
           'age': 17},
 {'first': 'Atara', 'last': 'Lerner-Friedman',
'age': 19},
 {'first': 'Reuven', 'last': 'Lerner', 'age': 50}
]
```

Sorting by multiple keys

- What if we want to sort by last name, and then first name?
- Solution: Have the key function return a tuple!
    - Python knows how to sort tuples, after all

```
def by_last_first(d):
    return d['last'], d['first']


print(sorted(people, key=by_last_first))

[
 {'first': 'Reuven', 'last': 'Lerner', 'age': 50},
 {‘first': 'Amotz', 'last': 'Lerner-Friedman', 'age': 14},
 {'first': 'Atara', 'last': 'Lerner-Friedman', 'age': 19},
 {'first': 'Shikma', 'last': 'Lerner-Friedman', 'age': 17}
]
```

Why write a key function?

- We're defining it, and then using it once.
- We could do better with *lambda,* which returns a function object
- Functions defined with lambda consist of a single expression
- So instead of "key=by_last_first", we could use:

```
print(sorted(people,
             key=lambda d: (d['last'],
                            d['first']))
```

Using operator.itemgetter

- An even more Pythonic approach: operator.itemgetter
- It's a function that returns a function!
- To sort "people" by age, we can say:

```
import operator

print(sorted(people,
       key=operator.itemgetter('age')))
```

Multiple items

- But we can call it with multiple arguments, too
- So instead of “key=by_last_first”, we can say:

```
import operator

print(sorted(people,
        key=operator.itemgetter('last',
                                'first')))
```

How can we sort objects?

```
class Person:
    def __init__(self, first, last, age):
        self.first = first
        self.last = last
        self.age = age


people = [Person('Reuven','Lerner', 50),
          Person('Atara', 'Lerner-Friedman', 19),
          Person('Shikma','Lerner-Friedman', 17),
          Person('Amotz', 'Lerner-Friedman', 14)]

print(sorted(people))
```

Doesn't work

```
Traceback (most recent call last):
  File "./slide22.py", line 15, in <module>
    print(sorted(people))
TypeError: '<' not supported between instances of
'Person' and 'Person'
```

Let’s implement < !

```
class Person:
    def __init__(self, first, last, age):
        self.first = first
        self.last = last
        self.age = age

    def __lt__(self, other):
        return self.age < other.age

    def __repr__(self):
        return f'Person, {vars(self)}'


people = [Person('Reuven', 'Lerner', 50),
          Person('Atara', 'Lerner-Friedman', 19),
          Person('Shikma', 'Lerner-Friedman', 17),
          Person('Amotz', 'Lerner-Friedman', 14)]

print(sorted(people))
```

Let’s create ==, as well

```
class Person:
    def __init__(self, first, last, age):
        self.first = first
        self.last = last
        self.age = age

    def __lt__(self, other):
        return self.age < other.age

    def __eq__(self, other):
        return self.age == other.age

    def __repr__(self):
        return f'Person, {vars(self)}'


people = [Person('Reuven', 'Lerner', 50),
          Person('Atara', 'Lerner-Friedman', 19),
          Person('Shikma', 'Lerner-Friedman', 17),
          Person('Amotz', 'Lerner-Friedman', 14)]

print(sorted(people))
```

Sorting by last + first names

```
class Person:
    def __init__(self, first, last, age):
        self.first = first
        self.last = last
        self.age = age

    def __lt__(self, other):
        return [self.last, self.first] < [other.last, other.first]

    def __eq__(self, other):
        return [self.last, self.first] == [other.last, other.first]

    def __repr__(self):
        return f'Person, {vars(self)}'


people = [Person('Reuven', 'Lerner', 50),
          Person('Atara', 'Lerner-Friedman', 19),
          Person('Shikma', 'Lerner-Friedman', 17),
          Person('Amotz', 'Lerner-Friedman', 14)]

print(sorted(people))
```

What about other comparisons?

```
print(people[1] >= people[0])
```

```
TypeError: '>=' not supported between instances of
'Person' and ‘Person'
```

Using total_ordering

```
import functools

@functools.total_ordering
class Person:
    def __init__(self, first, last, age):
        self.first = first
        self.last = last
        self.age = age

    def __lt__(self, other):
        return [self.last, self.first] < [other.last, other.first]

    def __eq__(self, other):
        return [self.last, self.first] == [other.last, other.first]

    def __repr__(self):
        return f'Person, {vars(self)}'

print(people[1] >= people[0])
```

```
True
```

Why work so hard?

- As of Python 3.7, we can use “dataclasses”
- These reduce our code by a *lot*.
- And they can handle sorting, also!

Re-implemented

```
from dataclasses import dataclass

@dataclass(order=True)
class Person:
    first: str
    last: str
    age: int
```

```
people = [Person('Reuven', 'Lerner', 50),
          Person('Atara', 'Lerner-Friedman', 19),
          Person('Shikma', 'Lerner-Friedman', 17),
          Person('Amotz', 'Lerner-Friedman', 14)]

print(sorted(people))
```

```
[Person(first='Amotz', last='Lerner-Friedman', age=14),
Person(first='Atara', last='Lerner-Friedman', age=19),
Person(first='Reuven', last='Lerner', age=50),
Person(first='Shikma', last='Lerner-Friedman', age=17)]
```

Wait a second!

- We didn't get an error…
- …but how are things being sorted?
- Answer: In the order that attributes were declared!
- Want to modify the sort order? Change your declarations.

Sort by last, first

```
from dataclasses import dataclass


@dataclass(order=True)
class Person:
    last: str
    first: str
    age: int

people = [Person('Lerner', 'Reuven', 50),
          Person('Lerner-Friedman', 'Atara', 19),
          Person('Lerner-Friedman', 'Shikma', 17),
          Person('Lerner-Friedman', 'Amotz', 14)]

print(sorted(people))
```

```
[Person(last=‘Lerner', first='Reuven', age=50),
Person(last='Lerner-Friedman', first='Amotz', age=14),
Person(last='Lerner-Friedman', first='Atara', age=19),
Person(last='Lerner-Friedman', first='Shikma', age=17)]
```

Summary

- Python's "sorted" function lets you sort *anything*
- The key to this working is the "key function"
  - Any function or method that takes a single input and returns a sortable output
  - Write your own function
  - Use lambda
  - Use operator.itemgetter with one or more arguments
- Classes are sortable if they define some magic methods
- Dataclasses make it even easier!

Questions or comments?

- E-mail me: reuven@lerner.co.il
- Follow me on Twitter: @reuvenmlerner

- See you next year… in person, I hope!