16.6. 🧑‍💻 When to use a Lambda Expression

Though you can often use a lambda expression or a named function interchangeably when sorting, it's generally best to use lambda expressions until the process is too complicated, and then a function should be used. For example, in the following examples, we'll be sorting a dictionary's keys by properties of its values. Each key is a state name and each value is a list of city names.

For our first sort order, we want to sort the states in order by the length of the first city name. Here, it's pretty easy to compute that property. `states[state]` is the list of cities associated with a particular state. So If `state` is a list of city strings, `len(states[state][0])` is the length of the first city name. Thus, we can use a lambda expression:

Save & Run Original - 1 of 1 Show in CodeLens

```python
states = {"Minnesota": ["St. Paul", "Minneapolis", "Saint Cloud", "Stillwater"],
          "Michigan": ["Ann Arbor", "Traverse City", "Lansing", "Kalamazoo"],
          "Washington": ["Seattle", "Tacoma", "Olympia", "Vancouver"]}

print(sorted(states, key=lambda state: len(states[state][0])))
```

```
['Washington', 'Minnesota', 'Michigan']
```

Activity: 1 -- ActiveCode (ac18_6_1)

That's already pushing the limits of complex a lambda expression can be before it's reall hard to read (or debug).

For our second sort order, the property we want to sort by is the number of cities that begin with the letter 'S'. The function defining this property is harder to express, requiring a filter and count accumulation pattern. So we are better off defining a separate, named function. Here, we've chosen to make a lambda expression that looks up the value associated with the particular state and pass that value to the named function s_cities_count. We could have passed just the key, but then the function would have to look up the value, and it would be a little confusing, from the code, to figure out what dictionary the key is supposed to be looked up in. Here, we've done the lookup right in the lambda expression, which makes it a little bit clearer that we're just sorting the keys of the states dictionary based on a property of their values. It also makes it easier to reuse the counting function on other city lists, even if they aren't embedded in that particular states dictionary.

Save & Run Original - 1 of 1 Show in CodeLens

```python
def s_cities_count(city_list):
    ct = 0
    for city in city_list:
        if city[0] == "S":
            ct += 1
    return ct

states = {"Minnesota": ["St. Paul", "Minneapolis", "Saint Cloud", "Stillwater"],
          "Michigan": ["Ann Arbor", "Traverse City", "Lansing", "Kalamazoo"],
          "Washington": ["Seattle", "Tacoma", "Olympia", "Vancouver"]}

print(sorted(states, key=lambda state: s_cities_count(states[state])))
```

```
['Michigan', 'Washington', 'Minnesota']
```

Activity: 2 -- ActiveCode (ac18_6_2)

At this point in the course, we don't even know how to do such a filter and accumulation as part of a lambda expression. There is a way, using something called list comprehensions, but we haven't covered that yet.

There will be other situations that are even more complicated than this. In some cases, they may be too complicated to solve with a lambda expression at all! You can always fall back on writing a named function when a lambda expression will be too complicated.

You have attempted 3 of 2 activities on this page

✔ Completed. Well Done!