Collection_protocols

May 7, 2021

1 Collection Protocols

Protocol	Implementing Collections
Container	str, list, range, tuple, set, bytes, dict
Sized	str, list, range, tuple, set, bytes, dict
Iterable	str, list, range, tuple, set, bytes, dict
Sequence	str, list, range, tuple, bytes
Set	set
Mutable Sequence	list
Mutable Set	set
Mutable Mapping	dict

1.1 Container

The `Container` protocol allow us to determine if a given value is part of the collection. This is done by using the `in` and `not in` operators.

It requires the collection to implement the `__contains__(item)` method, however, **it falls back to the `Iterable` protocol if implemented**.

1.2 Sized

The `Sized` protocol allow us to determine the length of the collection. This is done by passing the collection to the `len()` build-in function.

It requires the collection to implement the `__len__()` method. **It must not consume or modify the collection**

1.3 Iterable

The `Iterable` protocol allow us to iterate over the collection. This is done by using the `iter(iterable)` function.

It requires the collection to implement the `__iter__()` method.

1.4 Sequence

The `Sequence` protocol allow us to do the following: * Retrieve an item from the sequence by using the square brakets `item = seq[index]` * Retrieve a slice of items form the sequence by using the

square brakets `items = seq[start:stop]` * Produce a reversed sequence by calling the build-in function reversed `r = reversed(seq)` * Find the position of a given item in the sequence `position = seq.index(item)` * Count the amount of items that are the same as the one provided `num = seq.count(item)` * Concatenate sequences by using the `+` operator `new_seq = seq1 + seq2` * Repete sequences by using the `*` operator `new_seq = seq1 * 100`

In order to provide all this functionality stated above the collection first needs to implement the `Container`, the `Sized` and the `Iterable` protocols. In addition to that, the collection needs to implement the following extra methods: * `__getitem__(item)` allows the collection to retrieve items and slices by using square brakets. * `__reversed__()` allows the collection to produce a reversed sequence. **Uses `__getitem__()` and `__len__()` as fallback**. * **No extra methods are required to find the index of an item or to count items**. However, the methods available in the base class might not be the most performant `O(n)`. **If there is a faster way to obtain this information the methods to be implemented are `index` and `count` respectively**. * `__add__()` allows the collection to concatenate sequences. **It must produce a new sequence**. * `__mul__()` and `__rmul__()` allows the collection to repete sequences.

1.5 Set

The `Set` protocol allow us to do the following set operations:

1.5.1 Relational operators

special method	infix operator	set method	meaning
`__len__()`	<=	issubset()	subset
`__lt__()`	<		proper subset
`__eq__()`	==		equal
`__ne__()`	!=		not equal
`__gt__()`	>		proper superset
`__ge__()`	>=	issuperset()	superset

1.5.2 Algebraic operators

special method	infix operator	set method
`__and__()`	&	intersection()
`__or__()`	|	union()
`__xor__()`	^	symmetric_difference()
`__sub__()`	-	difference()