Python Fibonacci Sequence

Summary: in this tutorial, you'll learn how to define a custom Sequence type in Python and how to implement the Fibonacci sequence using a custom Sequence type.

Introduction to the custom Sequence type in Python

Sometimes, it's useful to implement a custom sequence type (https://www.pythontutorial.net/advanced-python/python-sequences/) that has functions similar to the built-in sequence type like tuples (https://www.pythontutorial.net/python-basics/python-tuples/) and lists (https://www.pythontutorial.net/python-basics/python-list/) .

As you've learned so far, a sequence can be mutable or immutable (https://www.pythontutorial.net/advanced-python/python-mutable-and-immutable/) . In this tutorial, you'll focus on defining a custom immutable sequence type.

Basically, an immutable sequence type should support two main functions:

- Return the number of elements of the sequence. Technically, this requirement is not necessary.
- Return an element at a given index or raise an `IndexError` if the index is out of bounds.

If an object can fullfil the above requirements, then you can:

- Use the square brackets `[]` syntax to retrieve an element by an index.
- Iterate over the elements of the sequence using the for loop, comprehension, etc.

Technically, a custom sequence type needs to implement the following methods:

- `__getitem__` – returns an element at a given index.
- `__len__` – returns the length of the sequence.

1) The __getitem__ method

The `__getitem__` method has the `index` argument which is an integer. The `___getitem__` should return an element from the sequence based on the specified `index` .

The range of the `index` should be from `zero` to `length - 1` . If the `index` is out of bounds, the `__getitem__` method should raise an `IndexError` exception.

Also, the `__getitem__` method can accept a slice object to support slicing.

2) The __len__ method

If a custom sequence has the `__len__` method, you can use the built-in `len` function to get the number of elements from the sequence.

Introduction to the Fibonacci sequence

The Fibonacci sequence was first discovered by Leonardo Fibonacci, who is an Italian mathematician, around A.D. 1170.

In the Fibonacci sequence, each number is the sum of two numbers that precede it. For example:

```
1, 1, 2, 3, 5, 8 , 13, 21, ...
```

The following formula describes the Fibonacci sequence:

```
f(1) = 1
f(2) = 1
f(n) = f(n-1) + f(n-2) if n > 2
```

Some sources state that the Fibonacci sequence starts at zero, not 1 like this:

```
0, 1, 1, 2, 3, 5, 8 , 13, 21, ...
```

But we'll stick with the original Fibonacci sequence that starts at one.

To calculate a Fibonacci number in Python, you define a recursive function (https://www.pythontutorial.net/python-basics/python-recursive-functions/) as follows:

```
def fib(n):
    if n < 2:
        return 1
    return fib(n-2) + fib(n-1)
```

In this recursive function, the `fib(1)` and `fib(2)` always returns 1. And when n is greater than 2, the fib(n) = fib(n-2) – fib(n-1)

The following adds a statement at the beginning of the `fib` function for the logging debugging purpose:

```
def fib(n):
    print(f'Calculate Fibonacci of {n}')
    if n < 2:
        return 1
    return fib(n-2) + fib(n-1)
```

And this shows output of the `fib` function when calculating the Fibonacci of 6:

```
Calculate the Fibonacci of 6
Calculate the Fibonacci of 4
Calculate the Fibonacci of 2
Calculate the Fibonacci of 0
Calculate the Fibonacci of 1
Calculate the Fibonacci of 3
Calculate the Fibonacci of 1
Calculate the Fibonacci of 2
Calculate the Fibonacci of 0
Calculate the Fibonacci of 1
Calculate the Fibonacci of 5
Calculate the Fibonacci of 3
Calculate the Fibonacci of 1
Calculate the Fibonacci of 2
Calculate the Fibonacci of 0
Calculate the Fibonacci of 1
Calculate the Fibonacci of 4
Calculate the Fibonacci of 2
Calculate the Fibonacci of 0
Calculate the Fibonacci of 1
Calculate the Fibonacci of 3
Calculate the Fibonacci of 1
Calculate the Fibonacci of 2
Calculate the Fibonacci of 0
Calculate the Fibonacci of 1
```

As shown clearly from the output, the `fib` function has many repetitions.

For example, it has to calculate the Fibonacci of 3 three times. This is not efficient.

To solve this, Python provides a decorator called `lru_cache` from the `functools` module.

The `lru_cache` allows you to cache the result of a function. When you pass the same argument to the function, the function just gets the result from the cache instead of recalculating it.

The following shows how to use the `lru_cache` decorator to speed up the `fib` function:

```
from functools import lru_cache


@lru_cache
def fib(n):
    print(f'Calculate the Fibonacci of {n}')
    if n < 2:
        return 1
    return fib(n-2) + fib(n-1)


fib(6)
```

Output:

```
Calculate the Fibonacci of 6
Calculate the Fibonacci of 4
Calculate the Fibonacci of 2
Calculate the Fibonacci of 0
Calculate the Fibonacci of 1
Calculate the Fibonacci of 3
Calculate the Fibonacci of 5
```

As shown clearly from the output, the number of calculations is reduced significantly.

Python Fibonacci sequence example

First, define a class (https://www.pythontutorial.net/python-oop/python-class/) that implements the Fibonacci sequence:

```
class Fibonacci:
    def __init__(self, n):
        self.n = n
```

The `__init__` method accepts an integer `n` that specifies the length of the sequence.

Second, define a static method (https://www.pythontutorial.net/python-oop/python-static-methods/) that calculates a Fibonacci number of an integer:

```
@staticmethod
@lru_cache(2**16)
def fib(n):
    if n < 2:
        return 1
    return Fibonacci.fib(n-2) + Fibonacci.fib(n-1)
```

Third, implement the `__len__` method so that you can use the built-in `len` function to get the number of elements from the Fibonacci sequence:

```
def __len__(self):
    return self.n
```

Finally, implement the `__getitem__` method to support indexing through the square brackets syntax:

```
def __getitem__(self, index):
    if isinstance(index, int):
        if index < 0 or index > self.n - 1:
            raise IndexError

        return Fibonacci.fib(index)
```

The `__getitem__` method accepts an `index` which is an integer. The `__getitem__` checks if the `index` is integer by using the `isinstance` function.

The `__getitem__` method raises the `IndexError` exception if the `index` is out of bounds. Otherwise, it returns the Fibonacci number of the `index`.

Putting it all together.

```
from functools import lru_cache


class Fibonacci:
    def __init__(self, n):
        self.n = n

    def __len__(self):
        return self.n

    def __getitem__(self, index):
        if isinstance(index, int):
            if index < 0 or index > self.n - 1:
                raise IndexError

            return Fibonacci.fib(index)

    @staticmethod
    @lru_cache(2**16)
    def fib(n):
        if n < 2:
            return 1
        return Fibonacci.fib(n-2) + Fibonacci.fib(n-1)
```

No, you can save the Fibonacci class in the `fibonacci.py` module and use it in a new script.

Using Fibonacci sequence

The following shows how to use the Fibonacci sequence from the `fibonacci` module:

```
from fibonacci import Fibonacci

fibonacci = Fibonacci(10)

# access elements via indices
print('Accessing Fibonacci sequence using []:')
print(fibonacci[0])
print(fibonacci[1])
print(fibonacci[2])

print('Accessing Fibonacci sequence using for loop:')
# using for loop
for f in fibonacci:
    print(f)
```

Output:

```
Accessing Fibonacci sequence using []:
1
1
2
Accessing Fibonacci sequence using for loop:
1
1
2
3
5
8
13
21
34
55
```

How it works.

- First, create a new instance of the `Fibonacci` sequence that contains 10 elements.
- Second, access the Fibonacci sequence's elements using the square brackets `[]` .
- Third, use the Fibonacci sequence in a for loop (https://www.pythontutorial.net/python-basics/python-for-range/) .

Adding slicing support

To support slicing (https://www.pythontutorial.net/advanced-python/python-slicing/) like this:

```
fibonacci[1:5]
```

...you need to add a logic to handle the `slice` object.

In the `fibonacci[1:5]` , the `index` argument of the `__getitem__` method is a `slice` object whose `start` is 1 and `stop` is 5.

And you can use the `indices` method of the `slice` object to get the indices of elements to return from the sequence:

```
indices = index.indices(self.n)
```

The `self.n` is the length of the sequence that will be sliced. In this case, it's the number of elements in the Fibonacci sequence.

To returns a list of Fibonacci from the slice, you can pass the indices to `range` function and use the list comprehension (https://www.pythontutorial.net/python-basics/python-list-comprehensions/) as follows:

```
[Fibonacci.fib(k) for k in range(*indices)]
```

The following shows the new version of the Fibonacci sequence that supports slicing:

```
from functools import lru_cache


class Fibonacci:
    def __init__(self, n):
        self.n = n

    def __len__(self):
        return self.n

    def __getitem__(self, index):
        if isinstance(index, int):
            if index < 0 or index > self.n - 1:
                raise IndexError

            return Fibonacci.fib(index)
        else:
            indices = index.indices(self.n)
            return [Fibonacci.fib(k) for k in range(*indices)]

    @staticmethod
    @lru_cache
    def fib(n):
        if n < 2:
            return 1
        return Fibonacci.fib(n-2) + Fibonacci.fib(n-1)
```

Now, you can slice the Fibonacci sequence as follows:

```
from fibonacci import Fibonacci

fibonacci = Fibonacci(10)
print(fibonacci[1:5])
```

Output:

```
[1, 2, 3, 5]
```

Summary

- Implement the `__len__` and `__getitem__` method to define a custom sequence.
- The `__getitem__` method need to returns an element based on a given index or raise an IndexError if the index is out of bounds.