hw1

January 16, 2023

1 Programming Tasks

1.1 Report/Explanation

1. Basically a copy of what was written in the assignment, no change necessary.
2. In the second block, created the Fibonacci class as specified. Calls the superclass's constructor to create the array.
3. Grows the array as necessary in accordance with a fibonnaci sequence when called.
4. Added a **len** function to Sequence class. Wrote it as a generator for simplicity (don't need to keep track of index).
5. The Prime class has 47 prime numbers already "cached" within it. If the length is greater than the cached amount, it will find more… hopefully my algorithm is correct, haven't tested it much.
6. Implemented the **gt** function as specified in assignment. Also do a type check for good measure.

1.2 Implementation and Output

[19]:
```
class Sequence:
    def __init__(self, array):
        self.array = array

    def __len__(self):
        return len(self.array)

    def __iter__(self):
        for item in self.array:
            yield item

    def __gt__(self, other):
        if not isinstance(other, Sequence):
            raise TypeError(f"{other} is not of type Sequence")
        if len(other) != len(self):
            raise ValueError(f"The sequences being compared do not have the␣
 ↪same length: {len(self)} vs {len(other)}")
        num_gt = 0
        for item1, item2 in zip(self, other):
            if item1 > item2:
```

```
                num_gt += 1
        return num_gt
```

[20]:
```
class Fibonacci(Sequence):
    def __init__(self, first_value, second_value):
        super().__init__([first_value, second_value])

    def __call__(self, length):
        if length > len(self.array):
            for i in range(length-len(self.array)):
                self.array.append(self.array[-1] + self.array[-2])
        else:
            self.array = self.array[:length]
        print(self.array)

FS = Fibonacci(1 ,2)
FS(length=5)

print(len(FS))
print([n for n in FS])
```

```
[1, 2, 3, 5, 8]
5
[1, 2, 3, 5, 8]
```

[25]:
```
class Prime(Sequence):
    primes = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,
              47,53,59,61,67,71,73,79,83,89,97,101,103,
              107,109,113,127,131,137,139,147,151,157,
              163,167,173,179,181,191,193,197,199]
    def __init__(self):
        super().__init__(list())

    def _is_prime(self, num):
        def gcd(num1, num2):
            if num2 == 1:
                return True
            elif num2 == 0:
                return False
            return gcd(num2, num1 % num2)
        for i in range(1, num//2, 2):
            if not gcd(num, i):
                return False
        return True

    def __call__(self, length):
        if length <= len(self.primes):
```

```
            self.array = self.primes[:length]
        else:
            n = self.primes[-1]//6 + 1
            while len(self.primes) < length:
                if self._is_prime(6*n-1):
                    self.primes.append(6*n-1)
                if self._is_prime(6*n+1):
                    self.primes.append(6*n+1)
                n += 1
            self.array = self.primes[:length]
        print(self.array)

PS = Prime()
PS(length=8)
print(len(PS))
print([n for n in PS])
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
8
[2, 3, 5, 7, 11, 13, 17, 19]
```

[23]:
```
FS = Fibonacci(first_value=1, second_value=2)
FS(length=8)
PS = Prime()
PS(length=8)
print(FS > PS)
```

```
[1, 2, 3, 5, 8, 13, 21, 34]
[2, 3, 5, 7, 11, 13, 17, 19]
2
```