Exercises

1) Create an iterator that returns the first n powers of 10.
2) Create an iterator that returns the first n multiples of a given number.
3) Create an iterator that returns the first n multiples of 2.
4) Create an iterator that returns the first n multiples of 3.
5) Create an iterator that returns the first n multiples of 10.
6) Create an iterator that returns all the divisors of a given number.
7) Create an iterator that returns the common divisors of two given numbers.
8) Create an iterator that returns the factors of a given number in descending order.
9) Create an iterator that returns the squares of even numbers from 0 to n.
10) Create an iterator that returns the cubes of odd numbers from 0 to n.

Exercises and solution

1) Create an iterator that returns the first n powers of 10.

```
class PowersOfTen:
    def __init__(self, n):
        self.n = n
        self.current = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.n:
            result = 10 ** self.current
            self.current += 1
            return result
        else:
            raise StopIteration
```

2) Create an iterator that returns the first n multiples of a given number.

```
class Multiples:
    def __init__(self, num, n):
        self.num = num
        self.n = n
        self.current = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.current <= self.n:
            result = self.num * self.current
            self.current += 1
```

```
            return result
        else:
            raise StopIteration
```

3) Create an iterator that returns the first n multiples of 2.

```
class MultiplesOfTwo:
    def __init__(self, n):
        self.n = n
        self.current = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.current <= self.n:
            result = 2 * self.current
            self.current += 1
            return result
        else:
            raise StopIteration
```

4) Create an iterator that returns the first n multiples of 3.

```
class MultiplesOfThree:
    def __init__(self, n):
        self.n = n
        self.current = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.current <= self.n:
            result = 3 * self.current
            self.current += 1
            return result
```

```
        else:
            raise StopIteration
```

5) Create an iterator that returns the first n multiples of 10.

```
class MultiplesOfTen:
    def __init__(self, n):
        self.n = n
        self.current = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.current <= self.n:
            result = 10 * self.current
            self.current += 1
            return result
        else:
            raise StopIteration
```

6) Create an iterator that returns all the divisors of a given number.

```
class Divisors:
    def __init__(self, num):
        self.num = num
        self.current = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.current <= self.num:
            if self.num % self.current == 0:
                result = self.current
                self.current += 1
```

```
            return result
        else:
            self.current += 1
            return self.__next__()
    else:
        raise StopIteration
```

7) Create an iterator that returns the common divisors of two given numbers.

```
class CommonDivisors:
    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2
        self.current_divisor = 1

    def __iter__(self):
        return self

    def __next__(self):
        while self.current_divisor <= min(self.num1, self.num2):
            if self.num1 % self.current_divisor == 0 and self.num2
% self.current_divisor == 0:
                divisor = self.current_divisor
                self.current_divisor += 1
                return divisor
            else:
                self.current_divisor += 1
        raise StopIteration

Example usage:

common_divisors = CommonDivisors(12, 18)
for divisor in common_divisors:
    print(divisor)
```

Output :

```
1
2
3
6
```

8) Create an iterator that returns the factors of a given number in descending order.

```
class DescendingFactors:
    def __init__(self, num):
        self.num = num
        self.current_factor = num

    def __iter__(self):
        return self

    def __next__(self):
        while self.current_factor > 0:
            if self.num % self.current_factor == 0:
                factor = self.current_factor
                self.current_factor -= 1
                return factor
            else:
                self.current_factor -= 1
        raise StopIteration
```

Example usage:

```
descending_factors = DescendingFactors(12)
for factor in descending_factors:
   print(factor)
```

Output :

```
12
6
4
3
2
1
```

9) Create an iterator that returns the squares of even numbers from 0 to n.

```
class EvenSquares:
   def __init__(self, n):
      self.n = n
      self.current_num = 0

   def __iter__(self):
      return self

   def __next__(self):
      while self.current_num <= self.n:
         if self.current_num % 2 == 0:
            square = self.current_num ** 2
            self.current_num += 2
            return square
         else:
            self.current_num += 1
      raise StopIteration
```

Example usage:

```
even_squares = EvenSquares(10)
for square in even_squares:
    print(square)
```

Output :

```
0
4
16
36
64
```

10) Create an iterator that returns the cubes of odd numbers from 0 to n.

```
class OddCubes:
    def __init__(self, n):
        self.n = n
        self.current_num = 1

    def __iter__(self):
        return self

    def __next__(self):
        while self.current_num <= self.n:
            if self.current_num % 2 != 0:
                cube = self.current_num ** 3
                self.current_num += 2
                return cube
            else:
                self.current_num += 1
        raise StopIteration
```

Example usage:

```
odd_cubes = OddCubes(10)
for cube in odd_cubes:
    print(cube)
```

Output :

```
1
27
125
343
```