Exercises

1) Create an iterator that returns numbers from 0 to 9.
2) Create an iterator that returns the first n Fibonacci numbers.
3) Create an iterator that returns the squares of numbers from 0 to n.
4) Create an iterator that returns the cubes of numbers from 0 to n.
5) Create an iterator that returns only the even numbers from 0 to n.
6) Create an iterator that returns only the odd numbers from 0 to n.
7) Create an iterator that returns the prime numbers from 0 to n.
8) Create an iterator that returns all the factors of a given number.
9) Create an iterator that returns the first n powers of 2.
10) Create an iterator that returns the first n powers of 3.

Exercises and solution

1) Iterator that returns numbers from 0 to 9:

```
class NumbersIterator:
    def __init__(self):
        self.current = 0

    def __iter__(self):
        return self

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

```
            raise StopIteration

    # Usage:
    numbers_iter = NumbersIterator()
    for number in numbers_iter:
        print(number)
```

2) Iterator that returns the first n Fibonacci numbers:

```
    class FibonacciIterator:
        def __init__(self, n):
            self.current = 0
            self.next = 1
            self.count = 0
            self.max_count = n

        def __iter__(self):
            return self

        def __next__(self):
            if self.count < self.max_count:
                result = self.current
                self.current, self.next = self.next, self.current + self.next
                self.count += 1
                return result
            else:
                raise StopIteration

    # Usage:
    fib_iter = FibonacciIterator(10)
    for number in fib_iter:
        print(number)
```

3) Iterator that returns the squares of numbers from 0 to n:

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

    def __iter__(self):
        return self

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

# Usage:
squares_iter = SquaresIterator(10)
for number in squares_iter:
    print(number)
```

4) Iterator that returns the cubes of numbers from 0 to n:

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

    def __iter__(self):
        return self

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

```
            return result
        else:
            raise StopIteration

# Usage:
cubes_iter = CubesIterator(10)
for number in cubes_iter:
    print(number)
```

5) Iterator that returns only the even numbers from 0 to n:

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

    def __iter__(self):
        return self

    def __next__(self):
        while self.current <= self.max:
            result = self.current
            self.current += 2
            if result % 2 == 0:
                return result
        raise StopIteration

# Usage:
even_iter = EvenNumbersIterator(10)
for number in even_iter:
    print(number)
```

6) Iterator that returns only the odd numbers from 0 to n:

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

    def __iter__(self):
        return self

    def __next__(self):
        while self.current <= self.max:
            result = self.current
            self.current += 2
            if result % 2 == 1:
                return result
        raise StopIteration

# Usage:
odd_iter = OddNumbersIterator(10)
for number in odd_iter:
    print(number)
```

7) Create an iterator that returns the prime numbers from 0 to n.

```
class PrimeIterator:
    def __init__(self, n):
        self.n = n
        self.current = 2

    def __iter__(self):
        return self

    def __next__(self):
        while self.current < self.n:
            is_prime = True
```

```
            for i in range(2, self.current):
                if self.current % i == 0:
                    is_prime = False
                    break
            if is_prime:
                result = self.current
                self.current += 1
                return result
            else:
                self.current += 1
        raise StopIteration

n = 20
prime_iterator = PrimeIterator(n)

for num in prime_iterator:
    print(num)
```

8) Create an iterator that returns all the factors of a given number.

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

    def __iter__(self):
        return self

    def __next__(self):
        if self.current > self.n:
            raise StopIteration
        while self.current <= self.n:
            if self.n % self.current == 0:
                result = self.current
                self.current += 1
                return result
            else:
```

```
            self.current += 1

n = 12
factor_iterator = FactorIterator(n)

for num in factor_iterator:
    print(num)
```

9) Create an iterator that returns the first n powers of 2.

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

    def __iter__(self):
        return self

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

n = 5
power_iterator = PowerIterator(n)

for num in power_iterator:
    print(num)
```

10) Create an iterator that returns the first n powers of 3.

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

    def __iter__(self):
        return self

    def __next__(self):
        if self.current >= self.n:
            raise StopIteration
        else:
            result = 3 ** self.current
            self.current += 1
            return result

n = 5
power_iterator = PowerIterator(n)

for num in power_iterator:
    print(num)
```