Rozwiązania

Zadanie 1: **Prosty iterator**

```
class Counter:
    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
        value = self.current
        self.current += 1
        return value
```

Zadanie 2: **Generator liczb parzystych**

```
def even_numbers(n):
    for i in range(0, n + 1, 2):
        yield i
```

Zadanie 3: **Co się stanie?**

Wynik działania:

```
0
1
[4, 9, 16]
```

Wyjaśnienie: generator wylicza po kolei wartości ‘x**2‘ dla ‘x‘ w zakresie od 0 do 4. Pierwsze dwa ‘next()‘ zwracają 0 i 1, potem ‘list(gen)‘ konsumuje pozostałe elementy: $2^2 = 4$, $3^2 = 9$, $4^2 = 16$.

Zadanie 4: **Generator słów**

```
def word_by_word(text):
    for word in text.split():
        yield word
```

Zadanie 5: **Kalendarz**

```
class DaysIterator:
    def __init__(self):
        self.days = ["pon.", "wt.", " r  .", "czw.", "pt.", "sob.",
        self.index = 0

    def __iter__(self):
        return self
```

```
    def __next__(self):
        day = self.days[self.index % len(self.days)]
        self.index += 1
        return day
```

Zadanie 6: **Generator nieskończony**

```
def fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b
```

Zadanie 7: **Plik jako iterator**

```
class FileLineIterator:
    def __init__(self, filename):
        self.file = open(filename, 'r')

    def __iter__(self):
        return self

    def __next__(self):
        line = self.file.readline()
        if line == '':
            self.file.close()
            raise StopIteration
        return line.strip()
```

Zadanie 8: **Stronicowanie**

```
def paginate(items, size):
    for i in range(0, len(items), size):
        yield items[i:i + size]
```

Zadanie 9: **Własny range**

```
class MyRange:
    def __init__(self, start, stop=None, step=1):
        if stop is None:
            self.start, self.stop = 0, start
        else:
            self.start, self.stop = start, stop
        self.step = step
        self.current = self.start

    def __iter__(self):
        return self
```

```
def __next__(self):
    if (self.step > 0 and self.current >= self.stop) or \
       (self.step < 0 and self.current <= self.stop):
        raise StopIteration
    value = self.current
    self.current += self.step
    return value
```

Zadanie 10: **Generator z `send()`**

```
def accumulator():
    total = 0
    while True:
        value = yield total
        if value is not None:
            total += value
```