Ponieważ pętle `for` zazwyczaj działają szybciej od pętli z licznikami opartych na `while`, użycie `for` w połączeniu z odpowiednimi narzędziami, takimi jak powyższe, zawsze będzie działało na naszą korzyść. Przyjrzymy się kolejno tym wbudowanym funkcjom.

Pętle liczników — while i range

Funkcja `range` jest tak naprawdę narzędziem, które można zastosować w wielu różnych kontekstach. Choć najczęściej wykorzystywana jest do generowania indeksów dla pętli `for`, można jej użyć w każdym miejscu, w którym potrzebna jest nam lista liczb całkowitych. W Pythonie 3.0 `range` jest *iteratorem* generującym elementy na żądanie, dlatego musimy opakować tę funkcję w wywołanie `list` w celu wyświetlenia wszystkich wyników naraz (więcej informacji na temat iteratorów znajduje się w rozdziale 14.).

```
>>> list(range(5)), list(range(2, 5)), list(range(0, 10, 2))
([0, 1, 2, 3, 4], [2, 3, 4], [0, 2, 4, 6, 8])
```

Z jednym argumentem funkcja `range` generuje listę liczb całkowitych od zera do wartości argumentu (ale bez niej samej). Jeśli przekażemy jej dwa argumenty, pierwszy uznawany jest za dolną granicę. Opcjonalny trzeci argument jest *krokiem*. Jeśli się go użyje, Python dodaje krok do każdej kolejnej liczby całkowitej wyniku (krok ma wartość domyślną `1`). Zakresy mogą również być niedodatnie i nierosnące, o ile jest nam to do czegoś potrzebne.

```
>>> list(range( 5, 5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]

>>> list(range(5, -5, -1))
[5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
```

Choć takie wyniki funkcji `range` mogą same w sobie być użyteczne, najbardziej przydają się wewnątrz pętli `for`. Po pierwsze, umożliwiają łatwe powtórzenie działania określoną liczbę razy. By na przykład wyświetlić trzy wiersze, wystarczy wykorzystać `range` do wygenerowania odpowiedniej liczby liczb całkowitych. Pętle `for` automatycznie wymuszają wyniki z `range` w Pythonie 3.0, dlatego nie musimy tutaj używać wywołania `list`.

```
>>> for i in range(3):
...     print(i, 'Python')
...
0 Python
1 Python
2 Python
```

Funkcja `range` jest często wykorzystywana do pośredniej iteracji po sekwencji. Najłatwiejszym i najszybszym sposobem wyczerpującego przejścia sekwencji jest zawsze proste `for`, gdyż Python większość szczegółów zrobi za nas.

```
>>> X = 'mielonka'
>>> for item in X: print(item, end=' ')       # Prosta iteracja
...
m i e l o n k a
```

Wewnętrznie pętla `for` obsługuje szczegóły iteracji automatycznie, kiedy użyje się jej w ten sposób. Jeśli naprawdę musimy wziąć logikę indeksowania w swoje ręce, możemy to zrobić za pomocą pętli `while`.

```
>>> i = 0
>>> while i < len(X):                        # Iteracja z pętlą while
...     print(X[i], end=' ')
```