```
>>> 5 / -2, 5 // -2                          # Zaokrąglenie w dół w wersji 2.6
( 3, -3)
>>> math.trunc(5 / float( 2))                # Odcięcie reszty w wersji 2.6
 2
```

Dlaczego odcinanie ma znaczenie?

Jeśli korzystamy z Pythona 3.0, oto krótkie podsumowanie sposobu działania operatorów dzielenia:

```
>>> (5 / 2), (5 / 2.0), (5 / -2.0), (5 / -2)          # Prawdziwe dzielenie w 3.0
(2.5, 2.5, -2.5, -2.5)

>>> (5 // 2), (5 // 2.0), (5 // -2.0), (5 // -2)     # Dzielenie bez reszty w 3.0
(2, 2.0, -3.0, -3)

>>> (9 / 3), (9.0 / 3), (9 // 3), (9 // 3.0)          # Oba
(3.0, 3.0, 3, 3.0)
```

U osób korzystających z wersji 2.6 dzielenie działa następująco:

```
>>> (5 / 2), (5 / 2.0), (5 / -2.0), (5 / -2)          # Dzielenie klasyczne w 2.6
(2, 2.5, -2.5, -3)

>>> (5 // 2), (5 // 2.0), (5 // -2.0), (5 // -2)     # Dzielenie bez reszty w 2.6 (to samo)
(2, 2.0, -3.0, -3)

>>> (9 / 3), (9.0 / 3), (9 // 3), (9 // 3.0)          # Oba
(3, 3.0, 3, 3.0)
```

Choć o wynikach dopiero się przekonamy, możliwe, że działanie operatora `/` bez odcinania reszty w Pythonie 3.0 spowoduje problemy w funkcjonowaniu sporej liczby programów. Być może ze względu na korzenie w składni języka C wielu programistów polega na dzieleniu z odcinaniem reszty dla liczb całkowitych. Będą oni musieli się nauczyć stosować w tym kontekście operator `//`. Przykład kodu, na jaki powyższa zmiana operatora `/` może mieć wpływ, znajduje się w zastosowaniu pętli `while` na liczbach pierwszych z rozdziału 13. oraz odpowiadającym mu ćwiczeniu umieszczonym na końcu części czwartej książki. Dodatkowo więcej informacji na temat przedstawionej wyżej specjalnej formy instrukcji `import` będzie można znaleźć w rozdziale 24.

Precyzja liczb całkowitych

Operacja dzielenia może działać różnie w poszczególnych wersjach Pythona, jednak jest elementem dość standardowym. A teraz coś bardziej egzotycznego. Jak wspomniano wcześniej, Python 3.0 obsługuje liczby całkowite o nieograniczonej wielkości:

```
>>> 99999999999999999999999999999 + 1
1000000000000000000000000000000
```

Python 2.6 ma osobny typ dla długich liczb całkowitych, jednak wszystkie liczby, które są zbyt duże, by można je było przechować w zwykłej liczbie całkowitej, przekształca na ten typ automatycznie. Tym samym by korzystać z długich liczb całkowitych, nie trzeba używać żadnej specjalnej składni — jedyną oznaką tego, że w Pythonie 2.6 korzystamy z długiej liczby całkowitej, jest wyświetlenie jej ze znakiem `L` na końcu:

```
>>> 99999999999999999999999999999 + 1
1000000000000000000000000000000L
```