Pokazana w przykładzie funkcja `sum` działa na sekwencji liczb, natomiast `min` i `max` przyjmują albo sekwencję, albo pojedyncze argumenty. Istnieje kilka sposobów na opuszczenie części ułamkowej z liczb zmiennoprzecinkowych. Wcześniej spotkaliśmy się z odcinaniem i zaokrąglaniem do najbliższej mniejszej liczby całkowitej. Możemy również zaokrąglać liczby, zarówno liczbowo, jak i na potrzeby wyświetlania:

```
>>> math.floor(2.567), math.floor(-2.567)      # Zaokrąglenie do najbliższej mniejszej liczby całkowitej
(2, -3)

>>> math.trunc(2.567), math.trunc( 2.567)      # Odcięcie (pominięcie części ułamkowej)
(2, -2)

>>> int(2.567), int( 2.567)                    # Odcięcie (konwersja na liczbę całkowitą)
(2, -2)

>>> round(2.567), round(2.467), round(2.567, 2) # Zaokrąglenie (Python 3.0)
(3, 2, 2.5699999999999998)

>>> '%.1f' % 2.567, '{0:.2f}'.format(2.567)    # Zaokrąglenie na potrzeby wyświetlania (rozdział 7.)
('2.6', '2.57')
```

Jak widzieliśmy wcześniej, ostatni przykład zwraca łańcuchy znaków, które normalnie byśmy wyświetlili, a także obsługuje różne opcje formatowania. Jak już wspomnieliśmy, przedostatni przykład zwróciłby `(3, 2, 2.57)`, gdybyśmy umieścili go w wywołaniu `print` w celu uzyskania zapisu nieco bardziej przyjaznego dla użytkownika. Mimo to te dwa przykłady różnią się od siebie — `round` zaokrągla liczbę zmiennoprzecinkową, jednak przechowuje ją w pamięci, natomiast formatowanie z użyciem łańcucha znaków zwraca łańcuch znaków i nie przechowuje zmodyfikowanej liczby:

```
>>> (1 / 3), round(1 / 3, 2), ('%.2f' % (1 / 3))
(0.33333333333333331, 0.33000000000000002, '0.33')
```

Co ciekawe, w Pythonie istnieją trzy sposoby obliczenia *pierwiastka kwadratowego* — za pomocą funkcji modułu, wyrażenia lub funkcji wbudowanej (osoby zainteresowane wydajnością tych rozwiązań odsyłam do ćwiczenia i jego rozwiązania na końcu czwartej części książki, gdzie sprawdzimy, które z nich działa szybciej).

```
>>> import math
>>> math.sqrt(144)                             # Moduł
12.0
>>> 144 ** .5                                  # Wyrażenie
12.0
>>> pow(144, .5)                               # Funkcja wbudowana
12.0

>>> math.sqrt(12345678900)                     # Większe liczby
35136.418286444619
>>> 12345678900 ** .5
35136.418286444619
>>> pow(12345678900, .5)
35136.418286444619
```

Warto zauważyć, że moduły biblioteki standardowej, takie jak `math`, trzeba importować, jednak funkcje wbudowane, takie jak `abs` czy `round`, dostępne są zawsze bez importowania. Innymi słowy, moduły to komponenty zewnętrzne, natomiast funkcje wbudowane znajdują się w domniemanej przestrzeni nazw, którą Python automatycznie przeszukuje w celu odnalezienia nazw użytych w programie. Ta przestrzeń nazw odpowiada modułowi o nazwie `builtins` w Pythonie 3.0 (`__builtin__` w wersji 2.6). Więcej informacji na temat rozwijania nazw można znaleźć w częściach książki poświęconych funkcjom i modułom. Kiedy jednak słyszymy słowo „moduł", od razu powinno nam przyjść do głowy słowo „importowanie".