Po co nam kolejny mechanizm formatujący?

Skoro już poświęciłem sporo miejsca na porównanie dwóch dostępnych technik formatowania łańcuchów znaków, powinienem wyjaśnić, w jakim właściwie celu programista miałby stosować jeden lub drugi wariant formatowania? Mimo że metoda `format` wymaga czasem napisania większej ilości kodu, posiada kilka zalet:

- ma kilka możliwości, których nie mają wyrażenia formatujące z operatorem `%`,
- odwołania do podstawianych argumentów są bardziej jawne i czytelne,
- zamiast symbolu operatora używamy czytelnej nazwy metody,
- nie wymaga stosowania osobnej składni dla pojedynczych i wielokrotnych podstawień.

We współczesnych wersjach Pythona mamy dostęp do obydwu technik formatowania. Wprawdzie wyrażenia formatujące są bardzo powszechnie stosowane, ale, jak się wydaje, metoda `format` z czasem przejmie dominację w tym zakresie. Wybór jednak należy do programistów. Przeanalizujmy zatem podstawowe różnice między tymi technikami.

Dodatkowe możliwości

Metoda `format` obsługuje kilka możliwości nieobsługiwanych przez wyrażenia formatujące, jak formatowanie liczb w notacji dwójkowej oraz (dostępne od Pythona 3.1) separator tysięcy. Dodatkowo bezpośrednio z szablonu formatującego dostępne są atrybuty i wartości słowników po kluczu. Jednak, jak mieliśmy okazję się przekonać, w wyrażeniach formatujących możemy użyć tych możliwości w nieco inny sposób:

```
>>> '{0:b}'.format((2 ** 16)-1)
'1111111111111111'

>>> '%b' % ((2 ** 16)-1)
ValueError: unsupported format character 'b' (0x62) at index 1

>>> bin((2 ** 16)-1)
'0b1111111111111111'

>>> '%s' % bin((2 ** 16)-1)[2:]
'1111111111111111'
```

W poprzednim punkcie można znaleźć przykład formatowania z użyciem słowników w wyrażeniach formatujących oraz jego porównanie z odwołaniami do kluczy słowników i atrybutów obiektów w metodzie `format`. W praktyce obydwa te podejścia wydają się jedynie różnymi wariacjami na ten sam temat.

Jawne odwołania do wartości

Jednym z przypadków, w których metoda `format` wydaje się doskonalsza, jest podstawianie większej liczby wartości do szablonu formatującego. Przedstawiony w rozdziale 30. przykład *lister.py* wykorzystuje podstawianie sześciu elementów do jednego szablonu i w tym przypadku odwołanie pozycyjne typu `{i}` wydaje się dawać czytelniejszy kod w porównaniu do podstawiania z użyciem `%s`:

```
'\n%s<Klasa %s, adres %s:\n%s%s%s>\n' % (...)             # Wyrażenie
'\n{0}<Klasa {1}, adres {2}:\n{3}{4}{5}>\n'.format(...)   # Metoda
```