Python #3: Formatowanie napisów

Wynik BMI wyświetlany w poprzednim zadaniu jest w raczej niezbyt przyjaznej formie. Bardzo dużo miejsc po przecinku sprawia, że wynik jest nieczytelny.

```
Twoje bmi wynosi: 20.703124999999996
```

Zaokrąglenie liczb w Pythonie możemy wykonać na kilka sposobów

Rzućmy okiem na to:

```
>>> print("Twoje BMI wynosi %f." % BMI)
Twoje BMI wynosi 20.703125
>>> print("Twoje BMI wynosi {:f}".format(BMI))
Twoje BMI wynosi 20.703125
>>> print("Twoje BMI wynosi", round(BMI,6))
Twoje BMI wynosi 20.703125
```

Dwa pierwsze przykłady przedstawiają zaokrąglenie za pomocą **formatowania łańcuchów znaków** (stringów), natomiast trzeci wykorzystuje funkcję **round()**, aby zaokrąglić zmienną BMI do 6 miejsc.

Dla dociekliwych: dlaczego 6 miejsc po przecinku?

Jeśli Python nie jest Twoim pierwszym język programowania, to prawdopodobnie kojarzysz więcej typów liczbowych – np. int, long, float, double.

Jeśli nie, to w skrócie: long – oznacza dużą liczbę całkowitą, a double – liczbę o większej dokładności „po przecinku”.

Nie ma tutaj całej teorii jak liczby są przechowywane w pamięci komputera (tabelkę z wielkością i zakresami znajdziesz tutaj), ale ta teoria nie jest programiście zbyt często w życiu potrzebna.

Z tego założenia wyszli też twórcy Pythona – interpreter martwi się o to za Ciebie. Jeśli Twój int będzie za duży automatycznie wejdzie w zakres liczby long, natomiast liczby zmiennoprzecinkowe mają domyślną precyzję typu double.

BMI zostało zwrócone, aż do 15 miejsca po przecinku, dopiero Ty nakazujesz Pythonowi „weź to pokaż jako float”. Float „ma” na liczby po przecinku przeznaczone 6 miejsc. Oczywiście, możemy to ładniej przedstawić i o tym za chwilę

Formatowanie stringów w Pythonie

Wyświetlanie różnych wartości wewnątrz napisów to całkiem przydatna sprawa, jednak dokumentacja zarówno starego (wywodzącego się z Pythona 2.x), jak i nowego (od Pythona 3) stylu formatowania jest dość ciężka w lekturze, dlatego najlepiej zaprezentować ją w przykładach.

Co to znaczy formatowanie stringów / napisów / łańcuchów znaków?

To nic innego jak umieszczenie wewnątrz napisu treści, którą podamy jako listę argumentów (konkretnie jako krotkę, ale o krotkach będzie trochę później).

W przykładzie z BMI podajemy do funkcji print() wartość liczbową, a jednocześnie informujemy przez **%f** czy też **{:f}** – „w to miejsce wstaw liczbę float".

Odpalmy konsolę i sprawdzimy inne przykłady – starym i nowym sposobem:

Stary:

```
>>> print("1 po angielsku: %s \n2 po angielsku: %s" % ('one', 'two'))
```

Nowy:

```
>>> print("1 po angielsku: {} \n2 po angielsku: {}".format('one', 'two'))
```

Wynik w obu wypadkach otrzymujemy taki sam:

```
1 po angielsku: one
2 po angielsku: two
```

Powyższe wyrażenia mogą wydawać się skomplikowane, ale zaraz wszystko będzie oczywiste. Tekst jest bardzo prosty, chcemy wyświetlić 1 i 2 po angielsku, rozdzielamy je za pomocą **\n** – znaku nowej linii (znaki specjalne znamy z poprzedniej części).

Spójrzmy – w pierwszym wypadku wystąpienie **%s** wymusza, by to miejsce zastąpić innym napisem (s = string). Mamy dwa wystąpienia %s i dwa argumenty „jeden" i „dwa", które umieszczamy po kolei wewnątrz napisu. Nasz napis i argumenty łączymy za pomocą znaku **%**.

W nowym formatowaniu miejsca oznaczamy przez klamry {},
a napis i argumenty łączymy przez **.format()**.

Liczby

– *całkowite:*

Stary:

```
>>> print("Cyfra %d poprzedza %d" % (7, 8))
Cyfra 7 poprzedza 8
```

Nowy:

```
>>> print("Cyfra {} poprzedza {}".format(7,8))
Cyfra 7 poprzedza 8
```

– zmiennoprzecinkowe:

Stary:

```
>>> print("Rekord świata na 100m to %f ustanowił go %s" % (9.58, 'Usain Bolt'))
Rekord świata na 100m to 9.580000 ustanowił go Usain Bolt
```

Nowy:

```
>>> print("Rekord świata na 100m to {} ustanowił go {}".format(9.58, 'Usain Bolt'))
Rekord świata na 100m to 9.580000 ustanowił go Usain Bolt
```

Niespodzianka ! Przecież jeszcze przed chwilą użyliśmy **{:f}**, a teraz wystarczą same klamry **{}**, aby za pomocą nowej metody wyświetlić zawartość float'a!

Jest to całkiem słuszne spostrzeżenie, jak również to, że w formacie podajemy czas 9.58, a Python uparcie wyświetla 9.580000 – nadal 6 miejsc po przecinku.

Przy wyświetlaniu wg nowego formatowanie, nie jest nam konieczne **:f** wewnątrz nawiasów {}, ale jest to informacja, dzięki której płynnie przejdziemy do zaokrąglania:

Stary:

```
>>> print("Rekord świata na 100m to %.2f ustanowił go %s" % (9.58, 'Usain Bolt'))
Rekord świata na 100m to 9.58 ustanowił go Usain Bolt
```

Nowy:

```
>>> print("Rekord świata na 100m to {:.2f} ustanowił go {}".format(9.58, 'Usain Bolt'))
Rekord świata na 100m to 9.58 ustanowił go Usain Bolt
```

Gdy przed **f**, wstawiamy modyfikator **.2** nakazujemy Pythonowi zaokrąglić wynik do dwóch miejsc po przecinku.

Analogicznie zadziała zaokrąglenie do 1 miejsca:

Stary:

```
>>> print("Rekord świata na 100m padł %s i wynosi poniżej %.1f sekund - wow!" % ('16 sierpnia 2009', 9.58))
Rekord świata na 100m padł 16 sierpnia 2009 i wynosi poniżej 9.6 sekund – wow!
```

Nowy:

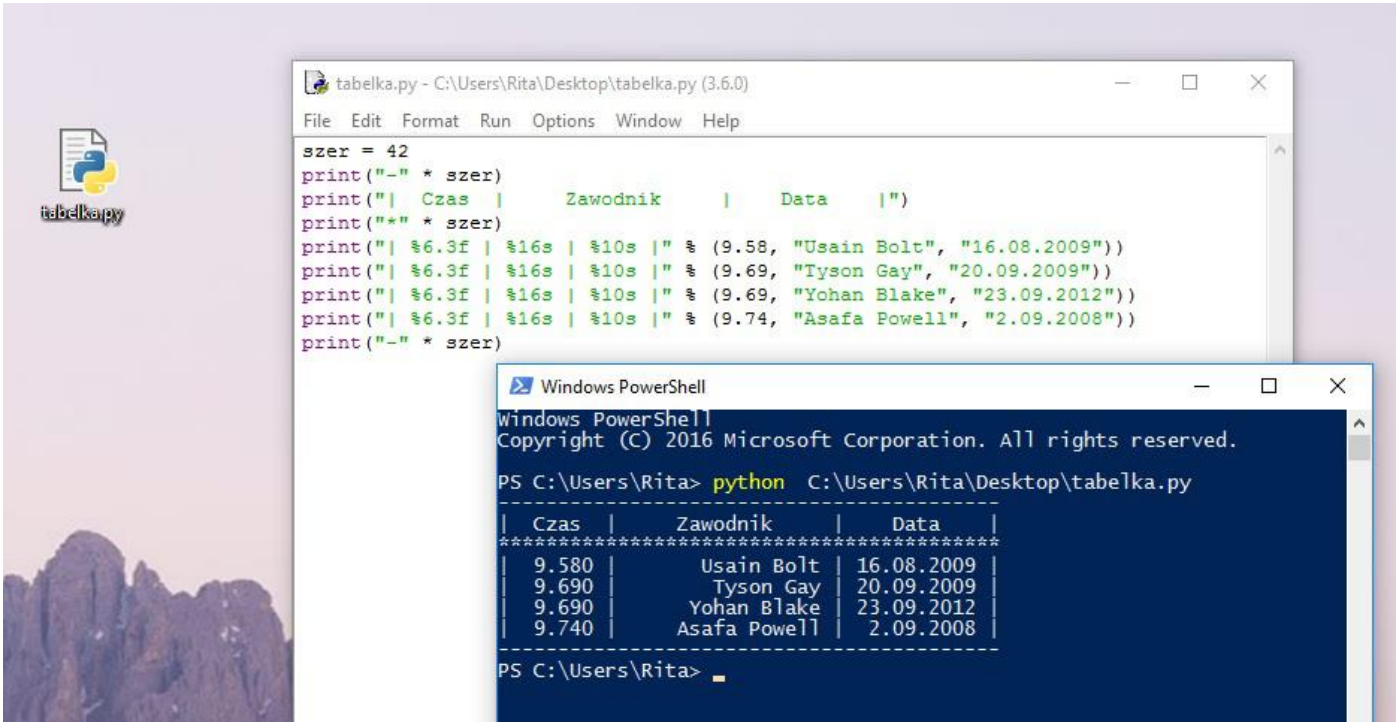
```
>>> print("Rekord świata na 100m padł {} i wynosi poniżej {:.1f} sekund - wow!".format('16 sierpnia 2009', 9.58))
Rekord świata na 100m padł 16 sierpnia 2009 i wynosi poniżej 9.6 sekund – wow!
```

Co więcej możemy przewidzieć **ile znaków na zapisanie wyrażenia chcemy przeznaczyć**.

Utwórz plik, skopiuj i uruchom poniższy skrypt (jeśli korzystasz z IDLE Pythona wystarczy, że wybierzesz Run lub wciśniesz F5, możesz uruchomić z konsoli, jak na poniższym screenie)

```
szer = 42
print("-" * szer)
print("|  Czas  |     Zawodnik     |    Data    |")
print("*" * szer)
print("| %6.3f | %16s | %10s |" % (9.58, "Usain Bolt", "16.08.2009"))
print("| %6.3f | %16s | %10s |" % (9.69, "Tyson Gay", "20.09.2009"))
print("| %6.3f | %16s | %10s |" % (9.69, "Yohan Blake", "23.09.2012"))
print("| %6.3f | %16s | %10s |" % (9.74, "Asafa Powell", "2.09.2008"))
print("-" * szer)
```

I

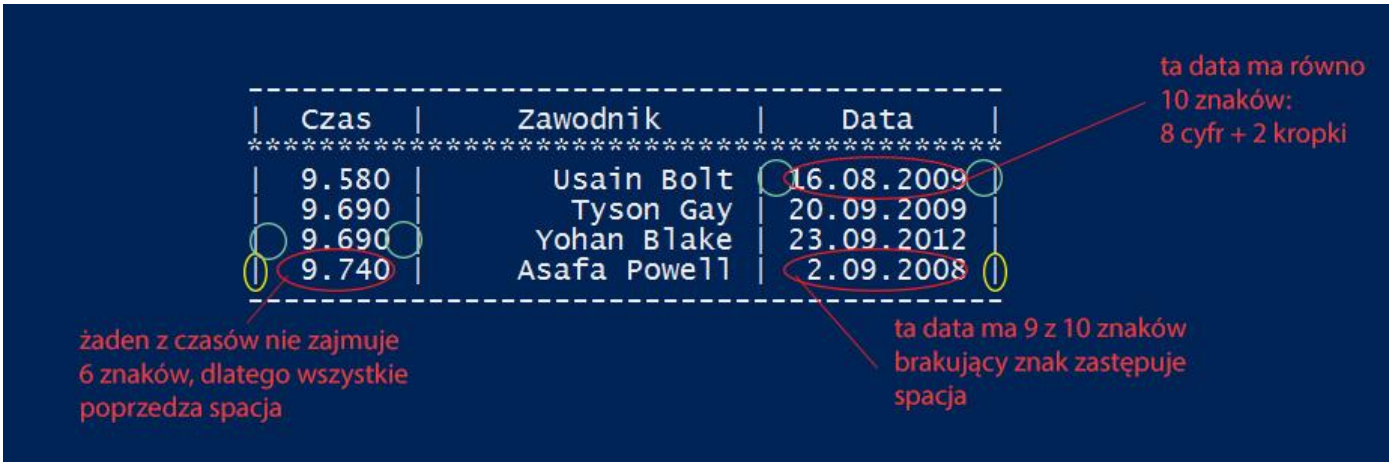


Ten zapis „mówi” Pythonowi:

%6.3f – przeznacz 6 znaków na zapisanie liczby zmiennoprzecinkowej, z czego 3 będą po kropce
%16s – przeznacz 16 znaków na zapisanie string’a
%10s – przeznacz 10 znaków na zapisanie string’a

szer – całkowita szerokość mojej tabeli to 42 znaki:

- 2 znaki | – budujące tabelę,
- po 2 znaki spacji z każdej strony wyświetlanej wartości
- znaki przeznaczone na wyświetlanie wartości,
  które wynikają z powyższego zapisu (6 + 16 + 10).

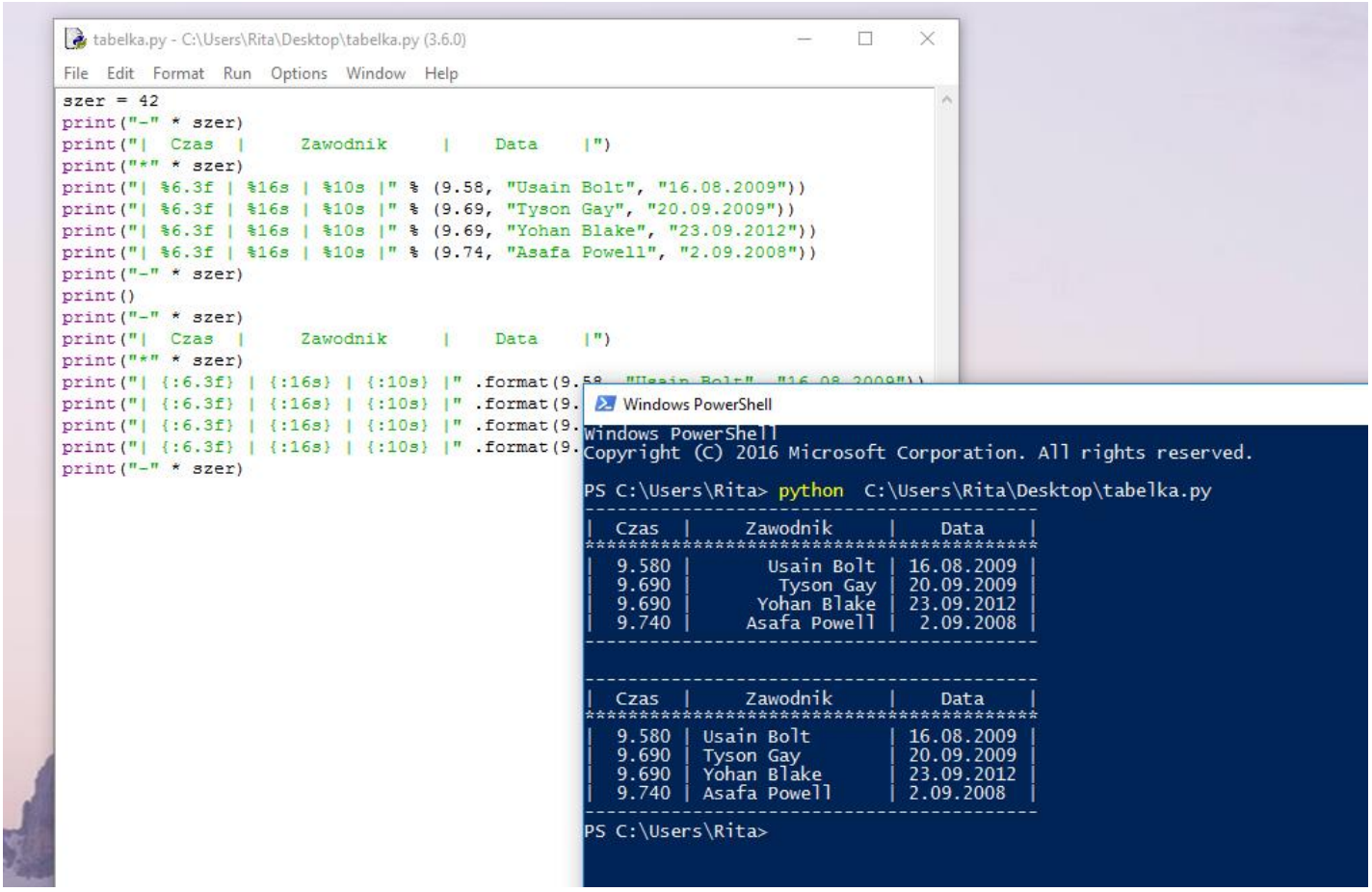


A jak ten skrypt będzie wyglądał dla nowego formatowania stringów?

```
szer = 42
print("-" * szer)
print("|  Czas  |     Zawodnik     |    Data    |")
print("*" * szer)
print("| {:6.3f} | {:16s} | {:10s} |" .format(9.58, "Usain Bolt", "16.08.2009"))
print("| {:6.3f} | {:16s} | {:10s} |" .format(9.69, "Tyson Gay", "20.09.2009"))
print("| {:6.3f} | {:16s} | {:10s} |" .format(9.69, "Yohan Blake", "23.09.2012"))
print("| {:6.3f} | {:16s} | {:10s} |" .format(9.74, "Asafa Powell", "2.09.2008"))
print("-" * szer)
```

Formatowanie, działa analogicznie, z drobnym wyjątkiem.

CZY WIDZIMY RÓŻNICĘ?



W **starym formatowaniu**, gdy mamy nadmiar wolnych znaków w stringu, spacje wstawiane są od lewej strony, czyli „przed” napisem.

W **nowym formatowaniu** puste znaki trafiają na koniec, czyli w sposób bardziej dla nas intuicyjny.

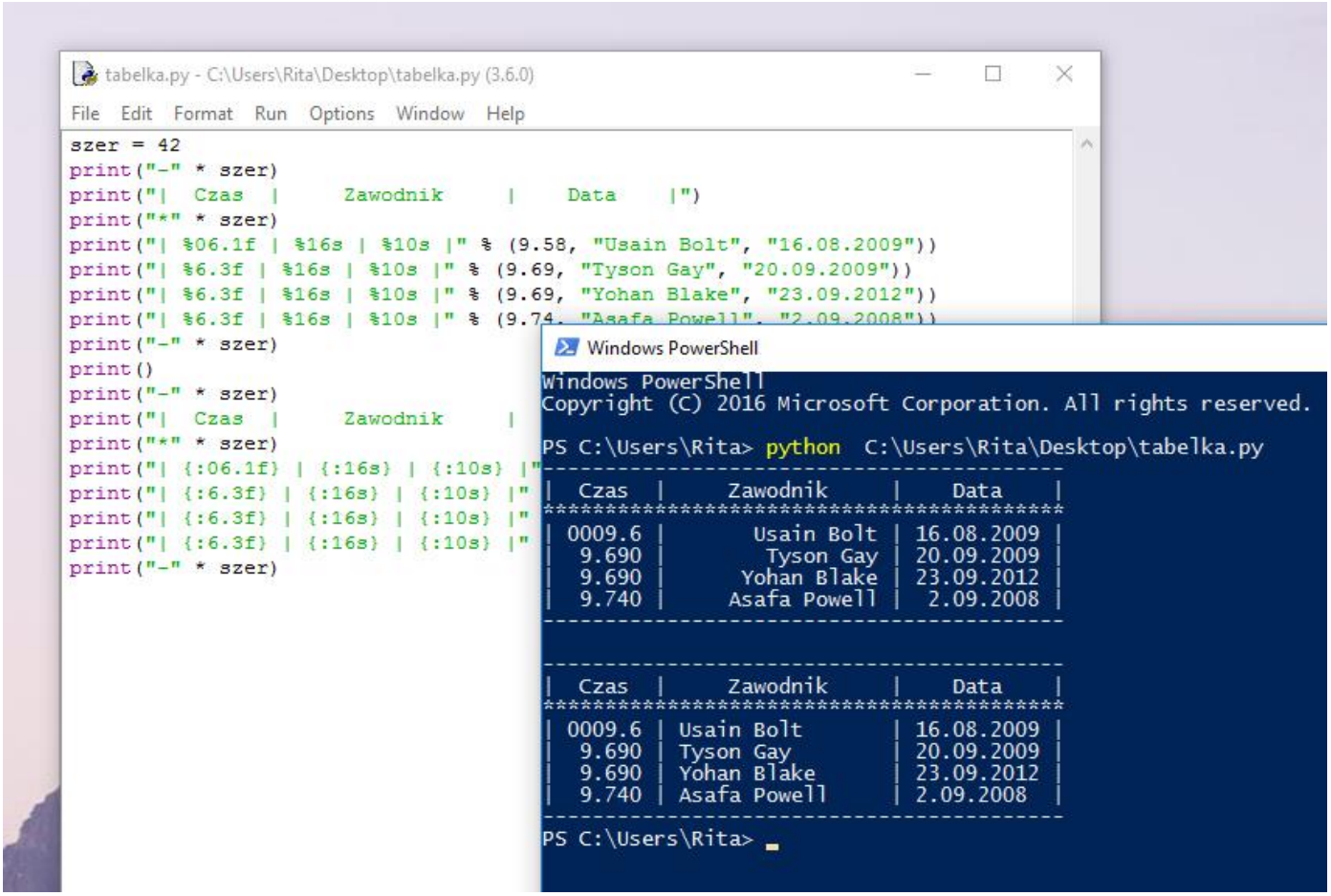
Oczywiście w starym formatowaniu, problem ten można rozwiązać. Spróbuj podmienić tylko jedną linijkę w pierwszej tabeli i zobacz co się stanie:

```
print("| %6.3f | %-16s | %-10s |" % (9.58, "Usain Bolt", "16.08.2009"))
```

Mimo, że napisy wyświetlają się inaczej w starej i nowej metodzie, to **%6.3f** i **{:6.3f}** dają dokładnie ten sam efekt.
Żeby to zrozumieć zmień **%6.3f** na **%06.3f**, a w drugim wypadku **{:6.3f}** na **{:6.3f}**.

Możesz też sprawdzić jak się zmieni wyświetlanie, gdy zamiast zaokrąglać do 3 miejsc po przecinku będziesz zaokrąglać do 2 lub tylko 1:

```
print("| %06.1f | %16s | %10s |" % (9.58, "Usain Bolt", "16.08.2009"))
```



Jeśli kopiując i zmieniając te napisy myślimy „jakie to nudneeeee!”…

ALE:
– warto wiedzieć, że coś takiego istnieje
– właściwie nie musimy umieć tego na pamięć
– szczerze nie pamiętam kiedy tego w życiu użyłem ☺

Przyjrzyjmy się jeszcze poniższemu skryptowi:

```
waluta = "dolar"
us = 1
pln = 4.08234915
print("Aktualnie %d %s kosztuje %.2f zł" % (us, waluta, pln))
```

Czy umiesz przepisać go za pomocą nowego formatowania?

Zamień ostatnią linijkę mojego skryptu na:

```
print("Aktualnie %r %r kosztuje %r zł" % (us, waluta, pln))
```

Teraz w miejscu %r wyświetla się dokładnie to samo co zawiera zmienna, nieważne co to jest.

Zamiana miejscami

Jeszcze jedna rzecz, której **nie ma w starej metodzie**, a może Ci się spodobać.

Sprawdźmy w konsoli:

```
>>> "{} ma {}".format("Ala", "kota")
Ala ma kota
>>> "{1} ma {0}".format("Ala", "kota")
kota ma Ala
```

Elementy są numerowane od 0, dlatego Ala jest domyślnie na pozycji 0 a kot na pozycji 1 – podając pozycje w nawiasach możesz dowolnie zmieniać kolejność wyświetlania.

```
>>> "{} {} {} {}".format("truskawka", "13", "1.5","herbata")
truskawka 13 1.5 herbata
>>> "{3} {2} {1} {0}".format("truskawka", "13", "1.5","herbata")
herbata 1.5 13 truskawka
>>> "{1} {3} {3} {2} {0}".format("truskawka", "13", "1.5","herbata")
13 herbata herbata 1.5 truskawka
```

PODSUMOWANIE:

Dla starego formatowania wystąpienie, które ma zostać zastąpione przez:

- napis oznaczamy: **%s** (s – string)
- liczbę całkowitą oznaczamy: **%d** (d – digit)
- liczbę zmiennoprzecinkową: **%f** (f – float)
- dowolną wartość: **%r** (r – raw)

np. `"Moja %s o wadze %dg ma ok. %d kcal i %.1f węglowodanów" % ("czekolada", 100, 545, 58.5)`

Używając nowego formatowania, nie musimy wiedzieć, co będzie wyświetlane wystarczy umieścić **nawiasy klamrowe {}**.

np. `"Moja {} o wadze {}g ma ok. {} kcal i {:.1f} węglowodanów".format("czekolada", 100, 545, 58.5)`

Więcej o formatowaniu napisów, bardzo przystępnie pokazanych znajdziesz tutaj:pyFormat.info

Zadanie 1 – konwerter jednostek
Utwórz nowy plik np. jednostki.py, który po podaniu przez użytkownika długości w cm będzie wyświetlał wynik w metrach i calach zaokrąglając do 4 miejsc po przecinku.

Podobny skrypt możesz wykonać dla zamiany kg na funty.

Wynik wyświetl używając dowolnego sposobu formatowania tekstu.

Zadanie 2 – lokata
Napisz skrypt, który, który obliczy stan konta za kilka lat. Program pyta użytkownika o:

- stan początkowy konta,
- stopę oprocentowania rocznego (zwróć uwagę, że odsetki podlegają miesięcznej kapitalizacji)
- liczbę lat na lokacie.

Wynik wyświetl jako zdanie używając dowolnego sposobu formatowania tekstu.

Wypisz np. takie zdanie:

Twoje *stan_początkowy* zł przez *czas* lata na lokacie *oprocentowanie* % urośnie do *wynik*.