Jak pokazałem w poprzednim przykładzie, operator + (plus) zastosowany na sach „skleja" je ze sobą, **tworząc nowy obiekt**. Na obiektach można wywoływać pewne funkcje zwane metodami i jest to cecha wspólna wszelkich realizacji obiektowych — omówimy je w kolejnym rozdziale. Co ciekawe, możesz także mnożyć napis przez liczbę całkowitą, np. wyrażenie `3*"Cześć!"` utworzy napis „Cześć!Cześć!Cześć!".

Napisy w konsoli tekstowej możemy wypisywać, używając znanej nam już funkcji `print()`. Możesz w nich stosować parametry formatujące napis wyświetlany na ekranie:

- `\t` wstawia odpowiednio odstęp (tabulator);
- `\n` wstawia tzw. znak nowej linii (jest to znak specjalny, niedrukowalny, ale wywołujący określone efekty na konsoli).

Przykład:

```
print("*\t\tNapis poprzedzony podwójnym tabulatorem (kod \'\\t\')*")
```

wyświetli:

```
*                Napis poprzedzony podwójnym tabulatorem (kod '\t')*
```

W Pythonie musisz ponadto stosować specjalny zapis, aby wyświetlić znaki ' (apostrof zwykły, niesformatowany) i " (cudzysłów zwykły, niesformatowany):

- `\\` wyświetla pojedynczy znak \;
- `\'` wyświetla pojedynczy znak ';
- `\"` wyświetla pojedynczy znak ".

W przykładowych programach na FTP znajdziesz rozbudowany program w Pythonie nie *napisy.py*, pokazujący kilka popularnych metod formatowania napisów wanych w funkcji `print()` (nie jest to wyczerpujący opis; na stronach tej ksi przedstawiłem kilka innych sposobów, które będę wprowadzał wraz z now wianym materiałem).

Konwersje napisów na liczby (i odwrotnie)

Python nie operuje jawnie pojęciem typu, jednak dość skutecznie „domyśla naszych intencji, analizując deklaracje zmiennych i danych, którymi je inicjuj W praktyce bardzo często operujemy ciągami znaków, z których chcemy wyłu wać wartości liczbowe w określonym formacie. Wtedy na pomoc przychodz funkcje konwertujące:

- `int(s)` — konwertuje ciąg znaków s na liczbę całkowitą (np. napis „3" na liczbę 3). Uwaga, wywołanie w stylu mieszanym (znaki i liczby) `int("kod23")` zwróci błąd i skrypt się zatrzyma!
- `float(s)` — konwertuje ciąg znaków s na liczbę zmiennoprzecinkową (np. napis „2.54" na liczbę 2.54).

Czasami potrzebne są konwersje w drugim kierunku, z liczby na obiekty typu `str`, i wówczas możesz zastosować takie przykładowe instrukcje używające funkcji `str()`:

```
moje_pi=3.14159265              # Deklaracja zmiennej moje_pi
pi_jako_napis=str(moje_pi)      # Deklaracja zmiennej pi_jako_napis
print("Długość napisu moje_pi to: ", len(pi_jako_napis)) # Długość napisu moje_pi to: 10
print("Wartość pi wynosi: ", moje_pi ) # Wartość pi wynosi: 3.14159265 # (*)
print("Wartość pi wynosi: " + moje_pi ) # (**)
print("Wartość pi wynosi: " + str( moje_pi)  )
```

Przedostatnia linijka, oznaczona (**), jest błędna i Python zgłosi błąd: `TypeError: can only concatenate str (not "float") to str`.

Ostatnia linijka jest już poprawna, przed użyciem operatora + (konkatenacji napisów) dokonujemy konwersji zmiennej `float` na `str` i dopiero wówczas dodawanie będzie miało sens!

Linijka oznaczona (*) też jest poprawna, nie musimy konwertować zmiennej na napis, bo jest ona bezpośrednio dekodowana przez funkcję `print()` jako jeden z podanych argumentów, oddzielonych przecinkami.

Formatowanie wyników z użyciem notacji f"

Przeanalizuj przykład:

```
x=45
p=79011306001
print("Wiek=", x, "PESEL=", p) # (*), wypisze: Wiek= 45 PESEL= 79011306001
print(f"Wiek={x}, PESEL= {p}") # (**), wypisze: Wiek=45, PESEL= 79011306001
print("Wiek=" + str(x) + ", PESEL= " + str(p)) # (***), wypisze: Wiek=45,
                                               # PESEL= 79011306001
```

Wszystkie wywołania funkcji `print()` wypiszą dokładnie ten sam napis, ale różnią się formą implementacji:

- W wersji oznaczonej (*) wypisujemy pewną serię napisów i wartości zmiennych w funkcji `print()`, która samodzielnie formatuje napis wynikowy[2].
- W wersji oznaczonej (**) niejako *wtrącamy* do istniejącego napisu wartości zmiennych (tutaj: `{x}` oraz `{p}`— stąd wynika otoczenie nazwy zmiennej nawiasami klamrowymi `{}` oraz poprzedzenie napisu literą `f`.
- W wersji oznaczonej (***) po prostu *sklejamy* kolejne fragmenty napisu, używając operatora + (plus). Jest to konwencja znana z innych języków programowania.

[2] Funkcja `print()` akceptuje zmienną liczbę argumentów.

Używaj takiej metody formatowania napisów, jaka Ci pasuje, przy czym stosowanie modyfikatora f" wydaje się jednak bardziej czytelne. „Wtrącane" wartości zmiennych otaczasz nawiasami klamrowymi, co pozwala je szybko wyłuskać z kodu i umożliwia jego zaawansowane formatowanie, co zaraz pokażę.

Wyrażenie `print(f"Wiek={x},PESEL={p}")` wyświetli napis: „Wiek=45, PESEL=79011306001"... pod warunkiem, że wcześniej zmienne x i p zostaną zainicjowane wartościami 45 i 79011306001.

Jest to niewątpliwie bardzo wygodna notacja, która ułatwia pisanie czytelnego kodu, bez obawy, że gdzieś zapomnimy domknąć napis cudzysłowem lub pomylimy nam się sekwencja zmiennych.

Okazuje się, że notacja f" oferuje więcej możliwości formatowania napisów wyjściowych. Zanim wyjaśnię zasady ogólne modyfikowania formatu wyświetlanych informacji, przyjrzyj się następującemu przykładowi (*formatowanie.py*):

```
moje_pi = 3.141592653589793
odchylka = 0.23
x_dec= 146
wielka_liczba=2344456612435.567
roznica=0.46
print(f"Liczba pi to {moje_pi}, co można skrócić do >>>{moje_pi:.4f}<<<")
# Zwykłe zaokrąglenie do 4 cyfr po przecinku:
print(f"Odchyłka pomiarowa wynosi >>>{odchylka:20.4f}<<<")
# Szerokość 20 znaków + rozszerzenie po przecinku:
print(f"Odchyłka pomiarowa wynosi >>>{odchylka:<20.4f}<<<")
# Jak wyżej + wyrównanie do lewej:
# Ustawienie przecinka jako separatora kolejnych 3 cyfr dużej liczby:
print(f"Wielka liczba: {wielka_liczba:}, wielka liczba z separatorem: {wielka_liczba:,}")
print(f"Różnica w cenie to: {roznica:.0%}") # Zamiana ułamka na procenty
print(f"Liczba w systemie dziesiętnym: {x_dec}, ta sama liczba binarnie: "
      f"{x_dec:b} oraz heksadecymalnie: {x_dec:X}")
```

Program po uruchomieniu wyświetli interesujące rezultaty (celowo wprowadziłem znaki >>> oraz <<<, aby pokazać szerokość pola zajmowanego przez liczbę):

```
$python3 formatowanie.py
Liczba pi to 3.141592653589793, co można skrócić do >>>3.1416<<<
Odchyłka pomiarowa wynosi >>>              0.2300<<<
Odchyłka pomiarowa wynosi >>>0.2300                <<<
Wielka liczba: 2344456612435.567, wielka liczba z separatorem: 2,344,456,612,435.567
Różnica w cenie to: 46%
Liczba w systemie dziesiętnym: 146, ta sama liczba binarnie: 10010010 oraz
➥heksadecymalnie: 92
```

W programie można zauważyć, że w nawiasach klamrowych zastosowana jest składnia: {*nazwa_zmiennej*: *kod_formatowania*}.

Co to jest nazwa zmiennej, nie trzeba tłumaczyć, ale przenalizuj te specyficzne zapisy zawarte *po dwukropku*, np.:

TABELA 5.1. Wybrane kody formatowanie zawartości pól w notacji f"

KOD FORMATU/KONWENCJA ZAPISU	PRZYKŁAD, UWAGI
Liczba podana bez symbolu kropki — cała szerokość pola przeznaczonego na wyświetlenie wartości.	Ustalamy szerokość na 6 znaków (ograniczniki > oraz < są dodane sztucznie, aby ułatwić analizę): `x=23` `print(f">{x:6}<")` # Wynik: > 23<
Liczba podana *po symbolu kropki* lub innego symbolu użytego do rozdzielenia części dziesiętnej od ułamkowej — liczba cyfr następująca po tym symbolu (zauważ, że możemy mieć do czynienia ze skróceniem i rozszerzeniem matematycznym!).	`x=23.346` `print(f"{x:.2f}")` # Wynik: 23.35 `print(f"{x:.4f}")` # Wynik: 23.3460
Liczba podana przed symbolem kropki lub innego symbolu użytego do rozdzielenia części dziesiętnej od ułamkowej — szerokość pola, ale całego, łącznie ze znakiem rozdzielającym!	Ustalamy szerokość na 8 znaków i 2 znaki po przecinku: `x=23.346` `print(f">{x:8.2f}<")` # Wynik: > 23.35<
< Wyrównanie zawartości do lewej strony w ramach dostępnej lub ustalonej szerokości pola.	`x=23.346` `print(f">{x:<8.2f}<")` # Wynik: >23.35 <
> Wyrównanie zawartości do prawej strony w ramach dostępnej lub ustalonej szerokości pola.	Uwaga: dla liczb to wyrównanie jest domyślne!
^ Wyśrodkowanie zawartości w ramach dostępnej lub ustalonej szerokości pola.	`x=23` `print(f">{x:^8}<")` # Wynik: > 23 <
= Przesuwa znak – (dla liczb ujemnych) skrajnie na lewo.	`x=-23.346` `print(f">{x:=8.2f}<")` # Wynik: >- 23.35<
+ Zawsze wyświetla symbol + lub –, aby pokazać, czy liczba jest dodatnia, czy ujemna.	`x=23.346` `print(f">{x:+8.2f}<")` # Wynik: > +23.35<
, Użyj przecinka jako separatora kolejnych 3 cyfr (wielokrotności tysięcy) w przypadku dużych liczb.	`x=234001343.23` `print(f">{x:,}<")` # Wynik: >234,001,343.23<
b Liczba w systemie dwójkowym (binarnym).	`z=65345` `print(f"{z:b}")` # Wynik: 1111111101000001
o Liczba w systemie ósemkowym.	`z=65345` `print(f"{z:o}")` # Wynik: 177501
x Liczba w systemie szesnastkowym.	`z=65345` `print(f"{z:x}")` # Wynik: ff41
d Liczba w systemie dziesiętnym.	`h=0x1f` `print(f"{h:d}")` # Wynik: 31

TABELA 5.1. Wybrane kody formatowanie zawartości pól w notacji f" — *ciąg dalszy*

KOD FORMATU/KONWENCJA ZAPISU	PRZYKŁAD, UWAGI

% Prezentacja ułamka w formacie procentowym.	`a=2.5888`
	`b=0.23`
	`print(f"{a:.1%}")` # Wynik: 258.9%
	`print(f"{b:.0%}")` *# Wynik: 23%*

e Liczba w systemie „naukowym" (tzw. notacja wykładnicza opisywana jako $a \cdot 10^n$, na ekranie wyświetlana w formacie *ae+n*, w której *a* mnożymy przez 10^n.	`a=2588.92` `print(f"{a:e}")` # Wynik: 2.588920e+03
f Zapis stałopozycyjny liczby ułamkowej (ang. *fixed point*).	Patrz *https://pl.wikipedia.org/wiki/Kod_stałopozycyjny.*
g Format ogólny (ang. *general*).	Tryb domyślny (wyrównanie do prawej).

Trochę się rozpisałem na tematy niealgorytmiczne, ale ufam, że ta wiedza okaże się przydatna podczas pisania programów prezentujących wyniki w czytelnej, estetycznej formie!

Tablice (nie całkiem) klasyczne

Tablice pomagają rozwiązywać wiele interesujących problemów algorytmicznych, np. wyszukiwanie i sortowanie danych, modelowanie grafów. Jak się przekonasz, programy, w których użyto tablic, będą nam towarzyszyły przez całą tę książkę, zwłaszcza że algorytmy z typami tablicowymi są wyjątkowo zwięzłe i łatwe do zrozumienia.

Według klasycznej definicji tablice są wektorami elementów określonego typu prostego lub złożonego i służą do przechowywania wartości tego typu (tak jak przegródki w szafce, do której możesz wkładać np. tylko piłki tenisowe i nic ponadto, gdyż z tyłu stoi trener i pilnuje porządku w szatni). Dostęp do poszczególnych wartości wymaga podania tzw. indeksu w tablicy i można wręcz uznać, że ten typ zastępuje przy użyciu jednej nazwy wiele zmiennych — np. pewna tablica o nazwie *T* zawierająca pięć wartości skutecznie zastępuje pięć zmiennych (moglibyśmy przecież zdefiniować zmienne `t0`, `t1`, `t2`, `t3`, `t4`). Tablice upraszczają programowanie wielu zagadnień, gdyż dzięki pojedynczej nazwie (etykiecie) uzyskujesz dostęp do wszystkich zapisanych w niej elementów, używając prostego mechanizmu indeksowania. Aby dotrzeć do elementu tablicy, podajesz jego pozycję, licząc od zera, np.: `t[0]`, `t[1]`, ..., `t[4]`.

Ograniczenie dotyczące tego samego typu dla przechowywanych wartości nie występuje co prawda w Pythonie, ale warto mieć je na uwadze, gdyż w innych językach może mieć ono zastosowanie!