Python #7: Pętla for

Znamy zmienne, umiemy pobrać dane od użytkownika, wyświetlić wynik operacji, a także wykonywać różne instrukcje w zależności od warunku, ale cały czas brakuje czegoś naszym programom. Przychodzi ten moment, gdy chcemy by w naszym kodzie dana część wykonywała się kilkukrotnie. Takie powtarzanie konkretnego bloku kodu nazywamy pętlą, zresztą na pewno znamy słowo zapętlić i to będziemy dzisiaj z naszym kodem robić.

Pętla FOR

W prostym przypadku chcemy, by dana czynność wykonała się z góry określiną liczbę razy. Tutaj przyda nam się **część składni, która występuje w każdym języku programowania – pętla FOR**.

Załóżmy, że chcemy zapytać 3 użytkowników o imię, a następnie przywitać każdego po imieniu. Możemy zrobić to tak:

```
name = input("Jak masz na imię?")
print("Cześć", name)
name = input("Jak masz na imię?")
print("Cześć", name)
name = input("Jak masz na imię?")
print("Cześć", name)
```

ale wygodniej byłoby jednak tak:

```
for user in range(0, 3):
    name = input("Jak masz na imię?")
    print("Cześć", name)
```

Możemy wywoływać każdego pokolei i kopiować ten sam kod, ale nie musimy. Pętle w językach programowania będą nam służyć do radzenia sobie w prosty sposób z takimi powtarzającymi się instrukcjami.

Gdyby osób było 5 nasza pętla zostałaby zmodyfikowana `for user in range(0, 5)`.

Funkcja `range()` może być używana na 3 sposoby.

Najbardziej podstawowy i najczęściej używany `range(0, end)` utworzy sekwencję od 0 do podanej liczby.

Podobnie `range(start, end)` pozwala określić początek i koniec zakresu, a `range(start, end, step)` dodaje do tego co jaki krok ma się wykonać pętla.

Utworzona sekwencja nigdy nie zawiera końca zakresu(!).

Oznacza to, że `range(0, 5)` przeczytamy: *jako wygeneruj zakres od 0 do 5,*
ale na 5 przestań się wykonywać.

Sprawdź:

```
for i in range(0, 5):
   print("krok: ", i)
```

Powyższy kod wyświetli:
0, 1, 2, 3, 4 (wyświetli 5 cyfr, ale bez ostatniej będącej końcem zakresu tj. cyfry 5).

Porównaj ten kod z

```
for i in range(2, 5):
   print("krok: ", i)
```

oraz

```
for i in range(0, 10, 2):
   print("krok: ", i)
```

Może wydawać się, że funkcja `range()` tworzy nam niewidoczną listę i dla każdego jej elementu wyświetla wartości.

Nic bardziej mylnego!

Funkcja `range()` **nie tworzy bezpośrednio listy, zamiast tego zwraca generator**.

Generatory generują elementy sekwencji po jednym na raz, tym samym unikamy niepotrzebnego przechowywania pełnej sekwencji w pamięci.

Iterowanie po liście

Oczywiście, nic nie stoi na przeszkodzie, by w pętli wywoływać elementy z listy.

Możemy mieć z góry znaną listę imion użytkowników.

Np. nasi goście to: Ania, Kasia, Jan, Piotr, Paweł.

Ich też da się łatwo powitać w pętli:

```
names = ["Ania", "Kasia", "Jan", "Piotr", "Paweł"]
for i in range(0, 5):
   print("Cześć", names[i])
```

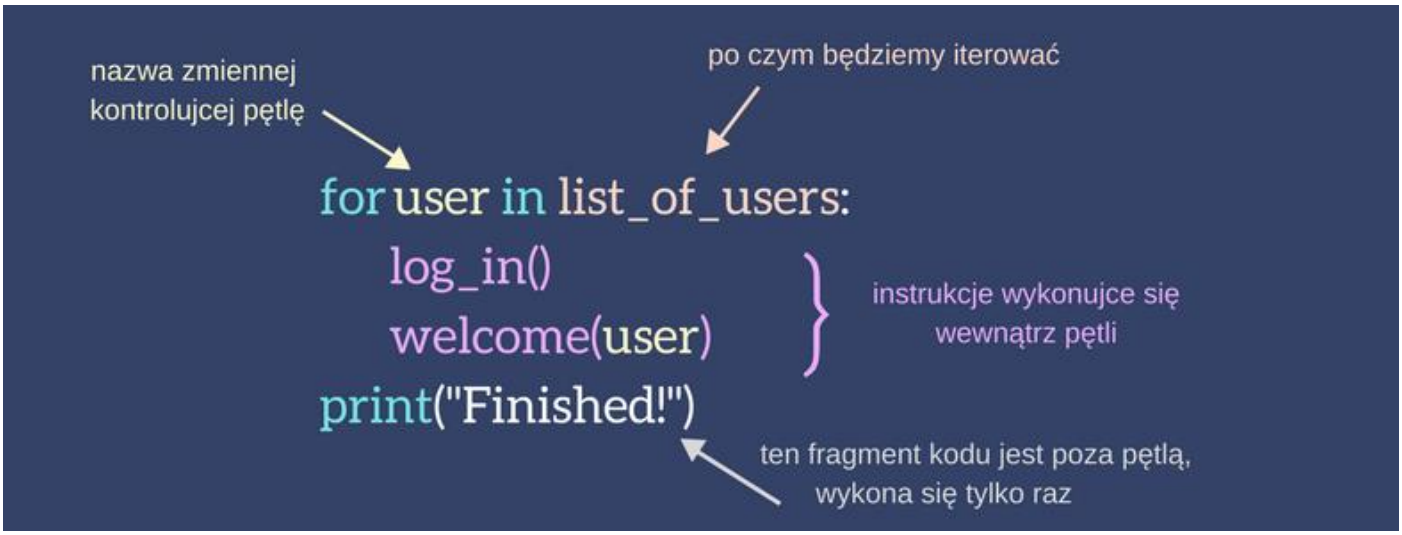
ale znacznie czytelniej będzie to zrobić tak:

```
names = ["Ania", "Kasia", "Jan", "Piotr", "Paweł"]
for name in names:
   print("Cześć", name)
```

Czym się różnią powyższe przykłady?
Sposobem dostawania do elementów na liście. Pierwszy przykład wykorzystuje fakt, że każdy element na liście ma swój indeks. Jak wiecie z poprzednich odcinków kursu komputery nie liczą jak ludzie. Zamiast liczyć *jeden, dwa, trzy …*, komputer zawsze (o ile programista nie zada inaczej) będzie numerował rzeczy od 0 – *zero, jeden, dwa….* Kolejne elementy tablicy z imionami mają indesy 0, 1, 2, 3, 4. Dzięki temu możemy wyświetlić pierwszego mężczyznę jako element `names[2]` i pojawi nam się Jan.

Drugi, ładniejszy sposób to wywołanie bezpośrednio imienia z listy `for name in names` oznacza tyle co *dla każdego imienia z listy imion wykonaj instrukcje*.

Jeżeli wszystko do tej pory było zrozumiałe, czas to podsumować:



Pętla for zawiera:

- słowo kluczowe for
- nazwa zmiennej, która odpowiada kolejnym elementom
- słowo in
- wartości, po których będziemy iterować
- blok kodu, który będzie wykonywał się w pętli (oznaczony wcięciem)

W przedostatnim podpunkcie celowo jest **wartości**, a nie lista po której będziemy iterować.

Te wartości to może być lista, generator, krotka, string czy słownik.

Iterowanie po krotce:

```
for i in (1,2,3,4):
   print("krok: ", i)
```

Iterowanie po stringu:

```
for i in "ala ma kota":
   print("krok: ", i)
```

Pętla w pętli

Narysowanie trójkąta w konsoli już po pierwszej lekcji i było dla was problemem:

```
print("#")
print("##")
print("###")
print("####")
print("#####")
```

Teraz powinno być to jeszcze prostsze:

```
for i in range(1, 6):
    print(i * "#")
```

Gdybyśmy chcieli zbudować 3 takie trójkąty?
Można to zrobić powielając kod odpowiedzialny za pojedynczy trójkąt w ten sposób

```
for i in range(0, 3):
    print("#")
    print("##")
    print("###")
    print("####")
    print("#####")
```

albo stworzyć pętle w pętli:

```
for i in range(0, 3):
    for j in range(1, 6):
        print(j * "#")
```

Proste?

Zadanie 1
Napisz program, który dla 10 kolejnych liczb naturalnych wyświetli sumę poprzedników.
Oczekiwany wynik: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55

Zadanie 2
Napisz program, który dla 10 kolejnych liczb naturalnych wyświetli ich wartość do sześcianu.
Szybka przypominajka:

Liczba	Sześcian
1	$1^3 = 1$
2	$2^3 = 8$
3	$3^3 = 27$
4	$4^3 = 64$
5	$5^3 = 125$

Zadanie 3
Pozwól użytkownikowi wprowadzić dowolną liczbę imion ciągiem (np. jako jeden string rozdzielonych przecinkiem lub białym znakiem). Następnie powitaj każdą osobę na liście.

Zadanie 4
Napisz prosty program, który wykona się zadaną przez użytkownika liczbę razy. Z każdym uruchomieniem pętli wyświetli informacje:
– czy liczba jest wielokrotnością 3
– czy liczba jest wielokrotnością 4
– wyświetli „hurra” jeżeli liczba dzieli się zarówno przez 3 jak i 4
– wyświetli gwiazdkę, jeśli żaden z powyższych warunków nie jest spełniony

Zadanie 5
Spróbuj wyświetlić choinkę z trójkątów w taki sposób, aby każdy poziom choinki był o 1 wiersz dłuższy:

```
#
##
#
##
###
#
##
###
####
```

Zadanie 6
Wyświetl w konsoli klasyczną tabliczkę mnożenia. W razie wątpliwości jak sformatować, by w konsoli pojawiła się tabela warto sobie przypomnieć lekcje 3.