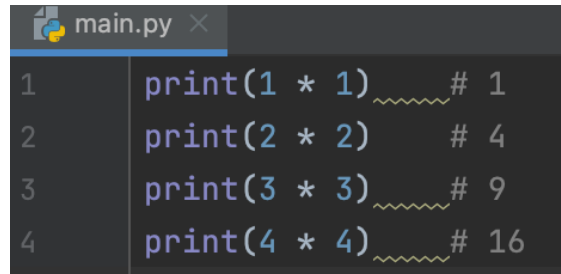


Instrukcja for

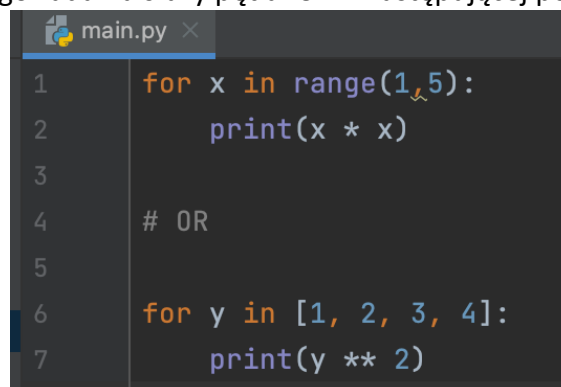
Załóżmy, że chcemy obliczyć kwadraty wszystkich liczb od 1 do 4. Zgodnie z dotychczasową wiedzą, w tym celu musimy wykonać 4 działania:



```
main.py x
1 print(1 * 1) # 1
2 print(2 * 2) # 4
3 print(3 * 3) # 9
4 print(4 * 4) # 16
```

Widzimy jednak, że te działania są bardzo podobne i chciałoby się je wykonać „za jednym zamachem”. Do wykonywania wielokrotnie tego samego (lub podobnego) kodu służą pętle. Najprostszym rodzajem pętli jest pętla **for**, która dla danego zakresu, danej listy, pliku czy też obiektu wykona operację po kolei na każdym elemencie.

Do wykonania powyższego zadania służy pętla **for** w następującej postaci:

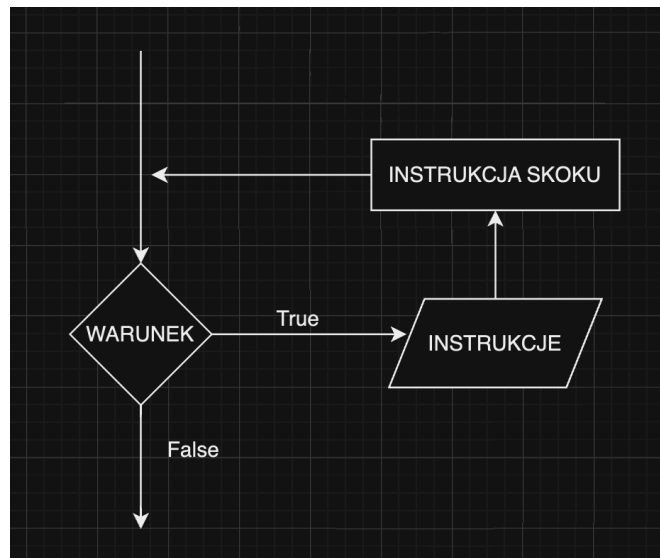


```
main.py x
1 for x in range(1,5):
2     print(x * x)
3
4 # OR
5
6 for y in [1, 2, 3, 4]:
7     print(y ** 2)
```

Zauważ, że wewnątrz pętli jest wyznaczone w sposób analogiczny do wnętrza instrukcji warunkowej:

- rozpoczyna się od dwukropka kończącego pierwszą linię,
- kolejne linijki są wcięte, tzn. rozpoczynają się od spacji, kilku spacji lub znaku tabulacji,
- jeżeli w ramach pętli chcielibyśmy wykonać kilka instrukcji muszą one mieć taki sam poziom wcięcia,
- „wewnątrz” pętli kończymy wracając do takiego samego poziomu wcięcia na jakim ją rozpoczęliśmy (takiego wcięcia jakie miała linijka z słowem kluczowym for),
- pętle możemy zagnieżdżać jedna w drugiej - blok wewnętrznej pętli musi być „bardziej” wcięty,
- powrót do poziomu wcięcia zewnętrznej pętli oznacza zakończenie pętli wewnętrznej i kontynuowanie zewnętrznej.

Pętla for



Instrukcja **for** (**for ... in**) jest zaliczana do tak zwanych pętli – czyli powtarzania działań tak długo, aż pojawią się warunki zakończenia pętli (na przykład nie ma więcej danych do przetworzenia). Chcąc przedstawić coś w miarę prostego w skomplikowany sposób, można napisać, że pętla **for** definiuje iteracje. Chodzi wyłącznie o to, że kolejne dane są pobierane do przetwarzania w określonym porządku (w przykładzie podana została potęgowanie kolejnych liczb w liście), a działania zostaną wykonane na wszystkich danych (chyba, że nastąpi błąd przerywający program).

Funkcja range

Wbudowana funkcja **range** działa jak generator kolejnych liczb (od 1 do 4). Możemy też wygenerować ciąg arytmetyczny inny niż złożony z kolejnych liczb. Funkcja **range** akceptuje bowiem trzeci parametr określający krok, co ile zwiększamy kolejne liczby. Na przykład, `range(1,5,2)` daje tylko dwie liczby [1,3]. Pamiętaj, że drugi parametr (zakres) określa górne ograniczenie otwarte, a nie ilość liczb! Ważną kwestią jest to, że **range()** jest generatorem zwracającym kolejne liczby pojedynczo, a nie wszystkie liczby naraz. Nie można więc zapisać: `lista=range(1,5)`. Jeśli chcemy uzyskać listę pięciu kolejnych numerów, należy użyć konstrukcji `lista=list(range(1,5))`. Funkcja **range** może być wywołana z jednym parametrem. Wtedy przyjmuje się, że dolne ograniczenie wynosi 0. Zapis `list(range(5))` daje nam listę [0, 1, 2, 3, 4].

Przykład:

```
main.py x
1  li = ['a', 'b', 'e']
2  for s in li:
3      print(s)
```