

Wstęp do języka Python. Część 5

Dzięki poszerzeniu wiedzy na temat napisów omówimy dziś takie operacje jak wybieranie (po angielsku nazywa się to „slicing”). Ponadto poznamy operatory przynależności oraz zapoznamy się z wyrażeniem warunkowym, które pozwala na podejmowanie decyzji o tym, czy dany fragment kodu powinien być uruchomiony w określonych warunkach, czy nie.

WYBIERANIE ELEMENTÓW NAPISU

Jak już wiemy – na napis możemy spojrzeć jak na pewną uporządkowaną kolekcję znaków. Umiemy już określać pozycję danego znaku, licząc od początku napisu. Wiemy, że te liczenie rozpoczyna się od 0, a każdy kolejny znak ma pozycję o jeden większą od poprzedniego. Co ciekawe, w Pythonie możemy także liczyć od końca napisu. Wtedy posługujemy się liczbami ujemnymi. W Tabeli 1 umieszczono napis (wiersz 1), a także indeksy znaków liczone od jego początku (wiersz 2) oraz od końca (wiersz 3).

P	r	o	g	r	a	m	i	s	t	a
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Tabela 1. Znak i indeksy liczone od początku i końca

Jeśli chcemy wybrać jakiś znak spod konkretnego indeksu, to możemy do tego użyć nawiasów kwadratowych [], w których podamy numer tego indeksu – tak jak w Listingu 1.

Listing 1. Wybieranie znaków poprzez ich indeksy

```
In [1]: napis = "Programista"
In [2]: napis[0]
Out[2]: 'P'
In [3]: napis[-11]
Out[3]: 'P'
In [4]: napis[10]
Out[4]: 'a'
In [5]: napis[-1]
Out[5]: 'a'
```

Wyobraźmy sobie teraz pewną wskazówkę – taki patyczek zakończony strzałką. Kiedy ja byłem w waszym wieku, widziałem taką w telewizji w programach Adama Słodowego (na przykład tu: <https://www.youtube.com/watch?v=bdkuYLlJSmg>). Może to być też po prostu długopis czy ołówek. Zamiast sobie wyobrażać możemy też po prostu takiej wskazówki użyć. Jeśli wybieramy pojedynczy znak, to wskazujemy taką wskazówką na początek komórki, w której ten znak się znajduje. Indeks będzie oznaczał, o ile znaków powinna się ona przesunąć w prawo (jeśli liczę od początku) lub w lewo (licząc od końca napisu), by jej koniec znalazł się z lewej strony znaku w komórce zawierającej nasz znak. By wziąć znak 'P', licząc od początku, nie muszę robić ruchu. Stąd wartość 0. Z kolei licząc od końca (czyli z prawej strony ostatniego 'a'), muszę przesunąć się 11 razy. Stąd -11.

Python oferuje także możliwość wybrania za jednym razem większej ilości znaków. Powiedzmy, że chcemy wybrać napis 'ogram' z napisu 'Programista'. Ponownie możemy użyć nawiasów kwadratowych, tym razem podając dwie liczby, które będą oznaczać początek i koniec, rozdzielimy je znakiem dwukropka – [początek:koniec]. Wybór znaków zobrazowano na Ilustracji 1.

W tym przypadku potrzebny byłby następujący kod:

Listing 2. Wybieranie znaków poprzez ich indeksy

```
In [6]: napis[2:7]
Out[6]: 'ogram'
```

Dlaczego [2:7], a nie [2:6]? Możemy pomóc sobie naszą wskazówką. By zacząć, muszę przesunąć się o dwie komórki. Wskazówka jest przed znakiem 'o'. Teraz by wziąć kolejny znak, muszę przesunąć się za niego, a tym samym będę wskazywał lewą stronę jakiegoś innego


znaku. Muszę zatrzymać wskazówkę za ostatnim znakiem, który chcę wybrać. Będzie to jednocześnie miejsce przed jakimś kolejnym znakiem – w tym przypadku 'i', którego indeks wynosi 7. Jeśli nie zrobię takiego przesunięcia – na przykład jak w Listingu 3, to dostanę pusty napis:

Listing 3. Jeśli indeks początku i końca będzie taki sam, to dostaniemy pusty napis

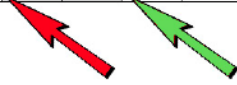
```
In [7]: napis[2:2]
Out[7]: ''
```

W ramach ćwiczenia spróbuj wybrać napis 'ogram' z napisu 'Programista', posługując się indeksami liczonymi od końca. Odpowiedź znajdziesz na końcu tego artykułu (1). Najpierw jednak spróbuj samodzielnie – posługując się wskazówką, a następnie sprawdź w interpreterze Pythona.


P	r	o	g	r	a	m	i	s	t	a
0	1	2	3	4	5	6	7	8	9	10



P	r	o	g	r	a	m	i	s	t	a
0	1	2	3	4	5	6	7	8	9	10



P	r	o	g	r	a	m	i	s	t	a
0	1	2	3	4	5	6	7	8	9	10



Ilustracja 1. Kolejne kroki wybierania napisu „ogram”. Strzałka czerwona wskazuje na początek przedziału, a strzałka zielona na jego koniec. Szarym tłem oznaczono wybrane znaki

Nawiasy kwadratowe pozwalają nam na użycie jeszcze jednej liczby. Będzie ona oznaczała krok, czyli wskaże, co który znak powinien zostać wybrany. Pełna funkcjonalność wygląda następująco: [start:stop:krok], gdzie zamiast słów są odpowiednie liczby. Przećwiczmy to w Listingu 4.

Listing 4. Wybieranie z krokiem

```
In [1]: napis = "Konstantynopolitańczyeczka"
In [2]: napis[2:20:2]
Out[2]: 'ntnyooiac'
```

Jeśli chcielibyśmy powiedzieć, że wybór powinien iść od jakiegoś znaku do końca, to w tym zapisie drugi parametr pomijamy – czyli za dwukropkiem nic nie wpisujemy. Dlaczego? Spójrzmy na kolejny przykład. Weźmy napis "tata". Jeśli zechcemy wziąć co drugi znak, licząc od początku (czyli od indeksu 0), to otrzymamy "tt". Jeśli poprosiłbym cię o wybranie co drugiego znaku, zaczynając od drugiego (czyli tego z indeksem 1), to pewnie odpowiedzią byłoby: "aa". Zobaczmy, jak mogłoby to wyglądać w kodzie. Celowo na początku spróbuję kilku niepoprawnych wersji.

Listing 5. Koniec przedziału powinien być pomijany, jeśli to ma być koniec napisu

```
In [1]: text = "tata"
In [2]: text[1:3:2]
Out[2]: 'a'
In [3]: text[1:-1:2]
Out[3]: 'a'
In [4]: text[1:-0:2]
Out[4]: ''
In [5]: text[1::2]
Out[5]: 'aa'
```

Krok – co ciekawe – może być też liczbą ujemną. Wtedy kolejne znaki brane są od końca – jak w Listingu 6. Co ważne, jeśli chcemy, by brany był pod uwagę napis od początku do końca, to musimy pominąć także 0, w przeciwnym wypadku możemy otrzymać zaskakujące wyniki. Prześledź Listing 6 i postaraj się znaleźć odpowiedź na to, co tam się dzieje? Odpowiedź prześlij do autora na adres programistajr.python@gmail.com (w temacie wiadomości koniecznie wpisz „Konkurs Python”). Postaraj się w kilku słowach wyjaśnić, skąd takie, a nie inne wyniki w poszczególnych sekcjach. Pierwsza poprawna odpowiedź zostanie nagrodzona roczną prenumeratą Programisty Junior 😊.

Listing 6. Krok może być ujemny, ale mogą się tu też wydarzyć dziwne rzeczy. Warto pamiętać, by poprawnie wskazać początek i koniec. Co tu się stało?

```
In [1]: napis = "Programista"
In [2]: napis[::-2]
Out[2]: 'asmroP'
In [3]: napis[2::-1]
Out[3]: 'orP'
In [4]: napis[2:6:-1]
Out[4]: ''
In [5]: napis[-2:-6:-1]
Out[5]: 'tsim'
In [6]: napis[6:2:-1]
Out[6]: 'marg'
```

Spójrzmy na kolejne ćwiczenie, które powinno być teraz całkiem proste. W jaki sposób możemy odwrócić napis? Czyli jak z napisu 'Programista' zrobić 'atsimargorP'. Odpowiedź znajdziesz na końcu artykułu (2).

OPERATOR „IN” ORAZ „NOT IN”

Często chcielibyśmy wiedzieć, czy jakaś litera, znak, a może nawet cały ciąg znaków znajduje się w innym napisie. Jak sobie z tym możemy poradzić? Znamy już metodę count dla napisu.

Listing 7. Zastosowanie metody „count” do sprawdzenia, czy coś jest w napisie

```
In [1]: napis = "Programista Junior"
In [2]: napis.count("a")
Out[2]: 2
In [3]: napis.count("z")
Out[3]: 0
In [4]: napis.count("Junior")
Out[4]: 1
```

W Pythonie możemy to zrobić również w inny sposób. Warto go poznać, ponieważ przyda nam się także przy innego rodzaju kontenerach takich jak tak zwane krotki (ang. tuple), listy, zbiory czy słowniki. Poznamy je już

niedługo i zobaczymy, że będziemy mogli sprawdzać dla nich przynależność tak samo jak dla napisów. Nasz nowy operator to właściwie para: in oraz not in. Zerknijmy w Listing 8, by zobaczyć, jak działają:

Listing 8. Zastosowanie operatorów „in” oraz „not in” do sprawdzania przynależności

```
In [5]: "a" in napis
Out[5]: True
In [6]: "z" in napis
Out[6]: False
In [7]: "z" not in napis
Out[7]: True
In [8]: "Junior" in napis
Out[8]: True
```

Jak widzimy, tym razem wartość takiego wyrażenia jest typu bool i przyjmuje wartości True/False. Wartości takie świetnie przydają się przy podejmowaniu różnych decyzji w czasie wykonywania programu, na przykład w wyrażeniach warunkowych.

WYRAŻENIE WARUNKOWE

W programowaniu, podobnie jak w życiu, często chcemy wykonać jakieś czynności tylko w określonej sytuacji. Innymi słowy, chcemy, by jakiś fragment kodu powinien wykonać się tylko wtedy, gdy są spełnione określone warunki. Moglibyśmy chcieć na przykład sprawdzić, czy jakiś znak jest w napisie – i jeśli jest, to wypisać jego indeks.

Listing 9. Wyrażenie warunkowe

```
In [9]: if "z" in napis:
...:     print("Znalazłem 'z'")
```

Jeśli operacja "z" in napis zwróci True, to print się wykona. Jeśli nie – nic się nie zadzieje. Jest to przykład najprostszego użycia wyrażenia warunkowego.

Myślenie programistów często jest przedmiotem zabawnych żartów. Jeden z nich obrazuje użycie wyrażenia warunkowego:

Żona mówi do programisty:

» Idź do sklepu i kup parówki. Jeśli będą jajka, to kup 10.

Programista przychodzi do sklepu i pyta:

- » Czy są parówki?
- » Tak – odpowiada sprzedawca.
- » To poproszę 10 jajek.

Komputery są trochę jak ten programista. Nie są zbyt domyślne. Dlatego tworząc wyrażenia warunkowe, musimy być bardzo precyzyjni.

Zastanów się, jak ta żona powinna poinstruować swojego męża, by mieć pewność, że kupi zarówno parówki, jak i 10 jajek.

Zanim pokażemy bardziej złożone przykłady, zauważmy jedną niezwykle istotną rzecz. Po pierwszej linii mamy dwukropek, a w kolejnej linii kod przesunięty jest wizualnie w prawo. Przyjęło się, że powinien być przesunięty o 4 spacje. Ale można się też spotkać z tym, że wstawiany jest tu znak tabulacji czy jakaś inna liczba spacji. Takie przesunięcie oznacza, że znajdujemy się wewnątrz naszego wyrażenia warunkowego. Instrukcje znajdujące się wewnątrz takiego wyrażenia nazywamy blokiem kodu. Może to być wiele linii, ale dopóki mają być wykonywane tylko wtedy, gdy warunek jest spełniony, muszą być przesunięte w prawo, a dodatkowo – musi to być takie same przesunięcie. Zobaczmy, jak to działa w Listingu 10:

Listing 10. Zastosowanie wcięć – indentacja.
W „In [12]” zastosowano niekonsekwentnie różne wcięcia wewnątrz bloku. To spowodowało rzucenie wyjątku

```
In [10]: warunek = True
...: if warunek:
...:     print("jestem w if")
...:     print("też jestem w if")
...:     print("Jestem poza if")
jestem w if
też jestem w if
Jestem poza if

In [11]: warunek = False
...: if warunek:
...:     print("jestem w if")
...:     print("też jestem w if")
...:     print("Jestem poza if")
Jestem poza if
```

```
In [12]: warunek = False
...: if warunek:
...:     print("jestem w if")
...:     print("też jestem w if")
...:     print("Jestem poza if")
File "<tokenize>", line 4
    print("też jestem w if")
    ^
```

IndentationError: unindent does not match any outer indentation level

W innych językach na ogół nie występuje wymóg stosowanie wcięć, ale często jest to dobra praktyka. Bloki kodu wyznacza się za pomocą różnego rodzaju nawiasów, słów kluczowych. Instrukcje często oddziela się od siebie średnikami.

W Pythonie z dobrej praktyki zrobiono obowiązujący wszystkich standard. Blok kodu wyznaczamy poprzez odpowiednie przesunięcie, czyli wcięcie. Nastąpić ono powinno zawsze po instrukcji kończącej się dwukropkiem. Jedną z takich instrukcji jest właśnie `if`. Instrukcje wewnątrz bloku oddziela się od siebie znakiem nowej linii.

Może to być początkowo trochę uciążliwe, ale z czasem docenisz ten wymóg, bo wymusza on czytelność kodu. Efekt spotęgowany jest tym, że każdy programista Pythona musi się do tych reguł stosować. Zostawia to mniej dowolności, ale znacznie łatwiej jest odnaleźć się w kodzie napisanym przez kogoś innego. Takie podejście wynika z założenia, że kod znacznie częściej się czyta niż pisze.



WARTO WIEDZIEĆ

Wcięcia pojawiły się w programowaniu dość wcześnie – w takich językach jak FORTRAN czy COBOL. Wiązały się one jednak wtedy z budową kart, na których robiono dziurki w odpowiednich kolumnach. Współcześnie ta idea wykorzystywana jest także w innych językach programowania. Ciekawy artykuł (w języku angielskim) na ten temat można znaleźć pod linkiem: <https://codelani.com/posts/which-programming-languages-use-indentation.html>

Niezależnie jednak od tego, czy jest to wymagane czy nie, konsekwentne stosowanie wcięć jest bardzo dobrą i zalecaną praktyką.