Rafał Korzeniewski

Wstęp do języka Python. Część 4

Trudno wyobrazić sobie współczesne programowanie bez napisów. Pełnią one nieocenioną rolę w komunikacji programów z użytkownikami. W artykule dowiemy się trochę o tym typie, nauczymy się też, w jaki sposób w Pythonie tworzyć, przekształcać i łączyć ze sobą różne teksty.

Jednym z pierwszych programów, jakie programiści piszą w różnych językach, jest program wypisujący na ekranie krótki tekst „hello, world". Zatrzymajmy się tu na chwilę. W programowaniu napisy określamy często angielskim słowem *string*, lub jakąś nazwą czy skrótem wywodzącym się od tego słowa. Po polsku moglibyśmy określić to jako łańcuch – czyli pewna całość stworzona z ogniw – w tym przypadku znaków. W Pythonie typ napisowy określamy jako `str`. Określamy tak typ, w którym przechowujemy ciągi znaków (czyli te łańcuchy). Taki napis z reguły ograniczony jest znakami " lub '. W Pythonie oba te sposoby są możliwe. Dodatkowo można też tworzyć wielolinijkowe napisy, używając potrójnych znaków ''' lub """. Napisem będzie więc zarówno `"Programista Junior"`, jak i `'Programista Junior'` czy `'''Programista Junior'''`.

Napis może składać się z różnych znaków, które możemy wprowadzić z poziomu klawiatury. Znaki te (ang. *characters*) możemy pogrupować na różne sposoby. I tak na przykład możemy wyróżnić:

- cyfry (*digits*):
  - `0123456789`
- litery małe (*lowercase*):
  - `abcdefghijklmnopqrstuvwxyz`
- litery duże (*uppercase*):
  - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- znaki interpunkcyjne i specjalne:
  - `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`

Oprócz nich mamy też grupę białych znaków, takich jak znak nowej linii, spacji czy tabulacji. Takie znaki widzimy na ekranie czy na kartce papieru jako odstępy między wyrazami, puste miejsca. Znaki takie możemy oznaczyć w naszym napisie także za pomocą specjalnych kodów. Zaczynają się one od tzw. znaku ucieczki \, po którym następuje kolejny znak. Na przykład znak nowej linii oznaczamy jako `\n`, znak tabulacji jako `\t`. W edytorach tekstowych albo procesorach tekstu możemy odpowiednimi kombinacjami klawiszy oznaczać np. nowy akapit, nową stronę itd. Te znaki też są tam pod spodem zapisywane jako specjalne instrukcje, a program, w którym je piszemy, dba o to, by odpowiednio taki tekst nam zaprezentować. Te szczególne znaki są potrzebne zwłaszcza nam, ludziom. Python, czy ogólniej mówiąc komputer, nie potrzebuje tych białych przestrzeni, by lepiej rozumieć tekst. By to lepiej zobrazować, zerknijmy w Listing 1.

Listing 1. Napis, jego reprezentacja oraz wyświetlenie przy użyciu funkcji „print"

```
In [1]: text = "a\tb\n1\t2"

In [2]: text
Out[2]: 'a\tb\n1\t2'

In [3]: repr(text)
Out[3]: "'a\\tb\\n1\\t2'"

In [4]: str(text)
Out[4]: 'a\tb\n1\t2'

In [5]: print(text)
a    b
1    2
```

W `In[1]` przypisujemy do zmiennej `text` napis. W `In[2]` odwołujemy się do naszej zmiennej. Wynikiem tego jest zwrócona wartość, która kryje się w zmiennej. Do tej pory działa to tak samo jak w przypadku typów numerycznych. Ale co się dzieje w `In[5]`? Nagle nasz napis zaczyna wyglądać inaczej. Sekwencja `\t` odpowiada za poziomy odstęp. Co ważne - my widzimy to jako dwa znaki. Z punktu widzenia Pythona jest to jednak jeden znak.

W `In[3]` widzimy użycie specjalnej funkcji `repr`. To za jej pomocą interpreter np. w `Out[2]` wyświetla tekst.

Jak widzimy, w tym, co zwraca `repr \t`, poprzedzony jest jeszcze jednym znakiem ucieczki. Taka sekwencja `\\` oznacza, że chcemy znak \ po prostu wyświetlić.

W maszynach do pisania dostępny był mechanizm, który umożliwiał przeskok. Z czasem nawet pojawił się do tego specjalny klawisz oznaczony właśnie jako Tab. Jak to działało w takiej maszynie, można zobaczyć w filmiku: https://www.youtube.com/watch?v=rsSG0L0Ts2E.

Wnikliwy obserwator zauważy, że w maszynie naciśnięcie klawisza Return powodowało przesunięcie głowicy w lewo, przy jednoczesnym przewinięciu karty w górę. W innym momencie jednak Return naciśnięty z innym klawiszem pozwalał na przesunięcie głowicy do początku linii bez przewijania kartki. Czy takie coś jest możliwe w Pythonie? Jak najbardziej. Służy do tego specjalny znak `\r`. Jak działa – zobaczmy w Listingu 2.

Listing 2. Różnica między napisem a sposobem jego wypisywania

```
In [4]: text = "Programista\rJunior"

In [5]: text
Out[5]: 'Programista\rJunior'

In [6]: print(text)
Juniormista
```

Ponownie w `In[4]` definiujemy zmienną, w kolejnej linii odwołujemy się do niej i zwracana jest wartość tego wyrażenia. Niespodzianka pojawia się w po `In[6]`. Tekst tym razem jest mocno zmieniony. W przypadku maszyny do pisania mielibyśmy na kartce nałożone na siebie dwa znaki. Mogłoby to zadziałać podobnie, gdybyśmy zlecili wydrukowanie takiego tekstu w jakimś historycznym modelu drukarek – na przykład w drukarce mozaikowej. Na ekranie kursor wrócił na początek linii i zastąpił pierwsze 6 znaków słowa „Programista" – wydrukował tam nowe znaki. W zmiennej `text` nie jest jednak przechowywane słowo `Juniormista`. Wartość w zmiennej może się więc mocno różnić od tego, jak jest prezentowana na ekranie.

To nie koniec niespodzianek. Włącz głośniki i wpisz w interpreterze następujący kod:

Listing 3. Czy drukowanie na monitorze może wydawać dźwięki? Przeczytaj o tym w „Ciekawostce"

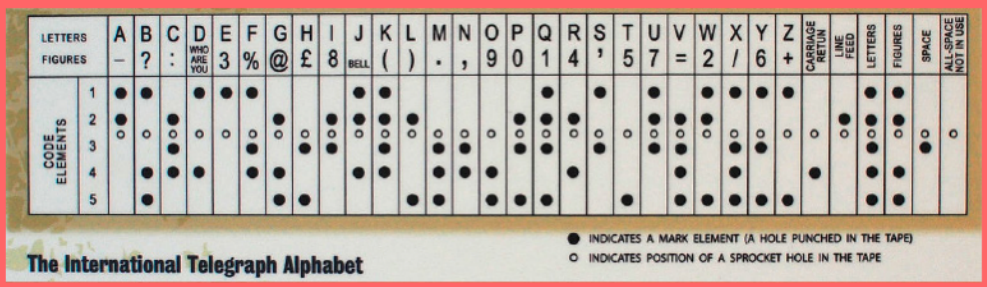
```
In [7]: print("\a")

In [8]:
```

Instrukcja ta najczęściej nie drukuje żadnego znaku, ale ma efekt uboczny w postaci dźwięku czy migającego terminala. Takich specjalnych sekwencji jest więcej. Zawsze zaczynają się od znaku ucieczki (ang. *escape character*) \.

WARTO WIEDZIEĆ

Kod dzwonka (ang. *bell code*), albo znak dzwonka (ang. *bell character*), wykorzystywany był w dalekopisach, czyli urządzeniach służących do przekazywania wiadomości tekstowych. Urządzenia te powstały pod koniec XIX wieku i zastąpiły telegrafy. Składały się z kodera, modulatora oraz klawiatury, a znaki w nich kodowane były za pomocą specjalnych systemów. Koder urządzenia miał 5 wyjść, na których mogło pojawić się dodatnie lub ujemne napięcie. Taka ilość wyjść daje 2^5, czyli 32 możliwości zakodowania różnych znaków. Jednym z kodów opisujących, jak te kombinacje przekładają się na znaki, był kod Baudota, który powstał około 1874 roku. Dzwonek w takich maszynach informował operatora o tym, że przyszła nowa wiadomość. Co ciekawe, w Polsce dalekopisy zakończyły swój oficjalny żywot w 2007 roku. Na świecie najdłużej były użytkowane w Indiach – aż do 2013 roku.



Ilustracja 2. Kod Baudota wykorzystywany w dalekopisach. Jednym ze znaków jest BELL (źródło: https://en.wikipedia.org/wiki/Baudot_code#/media/File:International_Telegraph_Alphabet_2_brightened.jpg)

W Listingu 4 znajdziesz więcej przykładów takich sekwencji.

Listing 4. Różne przykłady sekwencji ucieczki (escape sequences) stosowanych w Pythonie

```
In [1]: # Pojedynczy \ pozwala na złamanie linii.
   ...: # Co ważne, może po nim wystąpić jedynie
   ...: # przejście do nowej linii,
   ...: # inaczej będzie błąd.
   ...: # W praktyce wygodniej skorzystać z
   ...: # tego, że jeśli kod jest w nawiasach,
   ...: # to tego ukośnika nie musi być:
   ...: print("line1 \
   ...: line2 \
   ...: line3")
line1 line2 line3
 In [1]:# W praktyce wygodniej umieścić
   ...: # tekst w nawiasach,
   ...: # nie musimy się wtedy przejmować
   ...: # tym, by po \ nie było jakiejś spacji:
   ...: print("line1 "
   ...: "line2 "
   ...: "line3 ")
line1 line2 line3

In [2]: # Nie da się jednak wypisać tego znaku:
   ...: print("\")
  File "<ipython-input-2-840ba3bbe5f7>", line 2
    print("\")
               ^
SyntaxError: EOL while scanning string literal

In [3]: # Tak natomiast się uda:
   ...: print("\\")
\

In [4]: # Podobnie dla ":
   ...: print("\"")
"

In [5]: # Choć można go też umieścić
   ...: # pomiędzy '':
   ...: print('"')
"

In [6]: # Analogicznie w drugą stronę:
   ...: print('\'', "'")
''

In [7]: # \b to backspace, czyli
   ...: # skasowanie znaku po lewej:
   ...: print("Programista \bJunior")
ProgramistaJunior

In [8]: # \f to ASCII Formfeed.
   ...: # Ten znak w drukarkach
   ...: # wysunie kartkę:
   ...: print("Programista\fJunior")
Programista
           Junior

In [9]: # \n to nowa linia.
   ...: # Kursor przeniesie się
   ...: # na początek nowej linii:
   ...: print("Programista\nJunior")
Programista
Junior

In [10]: # \r powrót karetki
    ...: # przesunie kursor na początek linii,
    ...: # nie przechodząc do nowej:
    ...: print("Programista\rJunior")
Juniormista

In [11]: # \t ASCII Horizontal Tab (TAB),
    ...: # zrobi wcięcie w danej linii:
    ...: print("Programista\tJunior")
Programista   Junior

In [12]: # \v ASCII Vertical Tab (VT)
    ...: # to wertykalne wcięcie,
    ...: # dziś już bardzo rzadko stosowane:
    ...: print("Programista\vJunior")
Programista
           Junior

In [13]: # Można też przy pomocy \
    ...: # podawać kody znaków
    ...: # w systemie ósemkowym (octal):
    ...: "\110\145\154\154\157\40\127\
    ...: \157\162\154\144\41"
Out[13]: 'Hello World!'
```

```
In [14]: # Lub szesnastkowym.
    ...: # Tu i powyżej
    ...: # dodatkowo złamałem linię,
    ...: # by zachować odpowiednią szerokość
    ...: # listingu:
    ...: "\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\
    ...: \x6c\x64\x21"
Out[14]: 'Hello World!'

In [15]: # \N{} pozwala na podanie nazwy znaku.
    ...: # Obecnie niemal każdy znak z różnych
    ...: # języków z całego świata
    ...: # ujęty jest w specjalnym systemie
    ...: # zwanym Unicode. Znaki mają swoje
    ...: # identyfikatory i mogą mieć nazwy
    ...: # np. litera ą ma nazwę:
    ...: "\N{Latin Small Letter A with Ogonek}"
Out[15]: 'ą'

In [16]: # Jej identyfikator to 4 cyfry.
    ...: # Są to liczby szesnastkowe,
    ...: # czyli typu hex:
     ...: "\u0105"
Out[16]: 'ą'

In [17]: # Może być też podany w dłuższej formie
    ...: # 8 cyfr:
    ...: "\U00000105"
Out[17]: 'ą'
```

W powyższym listingu odwołano się do Unicode. Jest to standard opisujący w spójny sposób kodowanie różnych symboli. O Unicode i kodowaniu znaków przeczytać możesz więcej w naszym ebooku „Niezbędnik Młodego Programisty", dostępnym do pobrania ze strony: https://programistajr.pl/download/

Do tego momentu poznaliśmy napisy trochę od innej strony niż robi się to zazwyczaj. Nie musisz tego wszystkiego zapamiętywać, ale chciałem wyrobić w tobie ogólne pojęcie o tym, co to jest napis, z czego może być zbudowany. Jak widać, jest tam naprawdę dużo możliwości i gdyby chcieć zagłębić się w szczegóły, to sprawa może okazać się bardzo skomplikowana. Ot, choćby dlatego, że samo pismo chińskie zawiera około 50000 symboli i ciągle powstają nowe. W codziennej pracy z napisami na szczęście jednak nie musimy się tak bardzo tym przejmować. Szczególnie w takich prostszych, codziennych zastosowaniach.

Trochę problemów może pojawić się, gdy będziemy odczytywać teksty z plików. Mogą się tam pojawić różne problemy z kodowaniem znaków – wrócimy do tego, gdy zajmiemy się plikami.

Skoro mniej więcej wiemy już, czym napisy są, to musimy się jeszcze nauczyć coś z nimi robić. Częstą potrzebą jest na przykład to, by napisy ze sobą łączyć. Można to robić w Pythonie na kilka sposobów. Na przykład przy użyciu znaku +, jak w Listingu 5.

Listing 5. Łączenie napisów przy użyciu znaku +

```
In [1]: text1 = "Programista"

In [2]: text2 = "Junior"

In [3]: text1 + text2
Out[3]: 'ProgramistaJunior'

In [4]: text1 + " " + text2
Out[4]: 'Programista Junior'
```

Widzimy tutaj, że napisy są ze sobą sklejane. Znak + dokleja do końca napisu po swojej lewej stronie napis po prawej. Można wielokrotnie używać tego sposobu.

Innym sposobem jest metoda `format`. Jest ona dostępna dla każdego napisu, ale trzeba ten napis na nią specjalnie przygotować. Działa ona tak, że szuka w napisie specjalnych miejsc, w które ma wstawić inne napisy. Przykład w Listingu 6.

Listing 6. Użycie metody „format"

```
In [1]: text1 = "Programista"

In [2]: text2 = "Junior"

In [3]: szablon = "{} {}"

In [4]: szablon.format(text1, text2)
Out[4]: 'Programista Junior'
```

Jak widać, w zmiennej `szablon` pojawiły się `{}`, które są zastępowane napisami podanymi jako argumenty funkcji `format`. Ważne jest to, żeby ilość tych przygotowanych miejsc zgadzała się z ilością argumentów. Pola `{}` są wypełniane w tej kolejności, w jakiej podano argumenty. Metody `format` można używać jeszcze trochę inaczej – spójrzmy w Listing 7.

Listing 7. Użycie „format" z nazwanymi argumentami

```
In [1]: imie = "Clark"

In [2]: nazwisko = "Kent"

In [3]: "{a} {b}".format(a=imie, b=nazwisko)
Out[3]: 'Clark Kent'

In [4]: "{imie} {nazwisko}".format(
   ...:    imie=imie, nazwisko=nazwisko
   ...: )
Out[4]: 'Clark Kent'

In [5]: "{imie} {nazwisko}".format(
   ...:   nazwisko="Parker", imie="Peter"
   ...: )
Out[5]: 'Peter Parker'
```

Tym razem, jak widzimy, w nawiasach podajemy nazwy. Te nazwy muszą pojawić się jako argumenty funkcji `format`. Przypisujemy im tam też w tym miejscu wartości. Mogą to być wartości z już istniejących zmiennych – jak w `In[4]`, albo nowe – jak w `In[5]`. Widzimy też, że tym razem kolejność nie jest już istotna. Ważne jest to, by zgadzały się nazwy. Spróbuj we własnym zakresie przekonać się, co się stanie, jeśli tak nie będzie.

Od Pythona w wersji 3.6 pojawił się jeszcze jeden bardzo wygodny sposób na łączenie napisów. Są to tak zwane f-stringi. Jak działają, możemy zobaczyć w Listingu 8, w którym dodatkowo pokazałem, jak wygodnie można tworzyć wielolinijkowe napisy bez konieczności wstawiania `\n`. Wystarczy umieścić tekst pomiędzy potrójnymi cudzysłowami lub apostrofami.

Listing 8. Użycie f-string oraz wielolinijkowego napisu

```
In [1]: imie = "Sarah"

In [2]: nazwisko = "Connor"

In [3]: opis = f"""Postać
   ...: ======
   ...: imie:\t\t{imie}
   ...: nazwisko:\t{nazwisko}
   ...: """
In [12]: print(opis)
Postać
======
imie:        Sarah
nazwisko:    Connors
```

Wygląda to dość podobnie do tego drugiego użycia metody `format`, ale tym razem wartości brane są z już z samych istniejących zmiennych – dlatego ważne jest, by takie zmienne istniały. I co bardzo ważne – przed napisem trzeba dodać literkę `f`. Musi się ona znaleźć przed otwierającym napis apostrofem czy cudzysłowem. Każdy z tych sposobów ma swoje zastosowania. W praktyce najczęściej wykorzystuje się metodę `format` lub f-stringa. Jest to po prostu najwygodniejsze i daje ponadto wiele możliwości dodatkowego formatowania takich napisów. Opowiemy sobie o tym jeszcze w przyszłości.

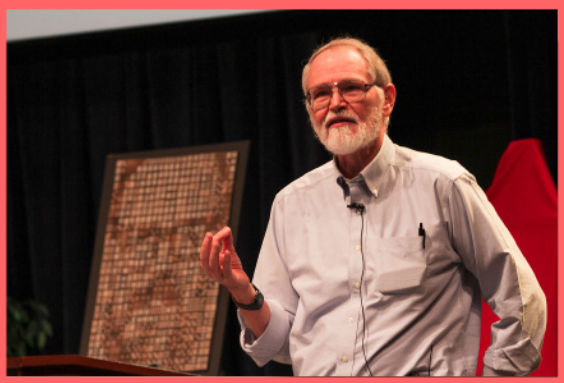
Czasem – szczególnie w starszym kodzie – spotkać się jeszcze można z formatowaniem przy użyciu `%`. Warto więc dać i taki przykład:

Listing 9. Formatowanie napisy przy użyciu „%"

```
In [5]: "%s %s" % (imie, nazwisko)
Out[5]: 'Sarah Connor'
```

CIEKAWOSTKA

„hello, world" zrobił ogromną karierę w świecie programowania. Po raz pierwszy taki przykład został użyty przez Briana Kernighan'a w podręczniku „Programming Language B", który wydano w 1973 roku. Podręcznik ten obecnie dostępny jest jako strona WWW: *https://www.bell-labs.com/usr/dmr/www/btut.html*. Brian nie pamięta dokładnie, kiedy i dlaczego zdecydował się na te dwa słowa. W jednym z wywiadów powiedział, że przypomina sobie kreskówkę, w której było jajko i pisklę. I to właśnie te pisklę miało wypowiedzieć te słowa.



Ilustracja 1. Autor słów „hello, world! (źródło: https://pl.wikipedia.org/wiki/Brian_Kernighan#/media/Plik:Brian_Kernighan_in_2012_at_Bell_Labs_1.jpg)

Do możliwości formatowania wrócimy jeszcze w kolejnych artykułach.

Teraz spójrzmy na kolejny aspekt pracy z napisami. Przede wszystkim musimy pamiętać, że to jest osobny typ. Jak sprawdzić, jaki to typ? Na pewno już pamiętasz. Możemy użyć funkcji `type`. Możemy też sprawdzić, jakie metody są dostępne dla napisu. Czyli co możemy z nim zrobić. Do tego służy funkcja `dir` – jak w Listingu 10.

Listing 10. Sprawdzenie typu oraz metod dostępnych dla napisu. Wynik funkcji „dir" skrócono i sformatowano na potrzeby druku

```
In [1]: napis = "PJ"

In [2]: type(napis)
Out[2]: str

In [3]: dir(napis)
Out[3]:
[ ...
'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum',
'isalpha', 'isascii', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric',
'isprintable',
'isspace', 'istitle', 'isupper', 'join',
'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper',
'zfill']
```

Mamy tu szereg metod. Niektóre coś sprawdzają i zwracają wartość logiczną: `True` lub `False`. Na przykład tak jak w Listingu 11.

Listing 11. Wybrane metody napisów zwracające „True" lub „False"

```
In [1]: # Czy małe litery?
   ...: "Python".islower()
Out[1]: False

In [2]: "python".islower()
Out[2]: True

In [3]: # Czy zaczyna się od fragmentu?
   ...: "python".startswith("Py")
Out[3]: False

In [4]: "python".startswith("py")
Out[4]: True
```

Do innej grupy moglibyśmy zaliczyć takie metody jak `count`, `index`, `split` czy `join`.

Listing 12. Użycie metod „count", „index", „split", „join"

```
In [5]: "Panama".count("a")
Out[5]: 3

In [6]: "Panama".index("m")
Out[6]: 4

In [7]: "a b c d".split()
Out[7]: ['a', 'b', 'c', 'd']

In [8]: ".".join("litery")
Out[8]: 'l.i.t.e.r.y'

In [9]: "panama".split("a")
Out[9]: ['p', 'n', 'm', '']
```

- `count` zlicza wystąpienia danego znaku. Jeśli go nie ma, zwraca 0.
- `index` – sprawdza indeks pierwszego wystąpienia danego znaku lub ciągu znaków (o tym za chwilę). Jeśli znaku nie ma, to zostanie rzucony wyjątek.
- `split` – dzieli napis – domyślnie po białych znakach, ale jeśli podamy tam jako argument jakiś napis, to podzieli nasz oryginalny napis w miejscach jego wystąpienia. Zwracana jest tu lista. Zaprzyjaźnimy się z nimi w kolejnych częściach z serii „Wstęp do Pythona".
- `join` – tworzy nowy napis. Wywołany dla kropki połączył tą kropką wszystkie znaki z napisu podanego jako argument tej metody. Tak naprawdę mogą to być też kolekcje napisów, ale o tym dowiemy się później.

Inne funkcje biorą nasz oryginalny napis i jakoś go przekształcają. Mogą zamienić go na przykład na małe litery (`lower`) albo na same duże (`upper`). Mogą otoczyć taki napis spacjami z lewej i prawej strony – tak by wyrównać w zadanej szerokości (`center`). W Listingu 13 znajdziesz takie przykłady.

Listing 13. Funkcje zwracające przekształconą kopię napisu

```
In [1]: napis = "Veni, vidi, vici"

In [2]: napis.upper()
Out[2]: 'VENI, VIDI, VICI'

In [3]: napis.lower()
Out[3]: 'veni, vidi, vici'

In [4]: napis.title()
Out[4]: 'Veni, Vidi, Vici'

In [5]: napis.center(44)
Out[5]: '              Veni, vidi, vici              '

In [14]: "  białe znaki znikną \t ".strip()
Out[14]: 'białe znaki znikną'
```

To, co jest tu ważne, to to, że żadna z tych metod nie zmienia oryginalnego napisu. Napis jest typem niezmienialnym, niemutowalnym. Możemy natomiast – bazując na istniejących – tworzyć nowe napisy, czy to przekształcając je metodami, czy łącząc z innymi. I możemy oczywiście przypisywać takie teksty do zmiennej o tej samej nazwie. Nazwa pozostaje, ale pod spodem będzie to inny obiekt. Przykład powiększania napisu wraz z pokazaniem, że po każdym przypisaniu wartości do zmiennej tworzony jest nowy obiekt, znajdziesz w Listingu 14.

Listing 14. Łączenie tekstów a tworzenie nowych obiektów

```
In [1]: text = ""

In [2]: id(text)
Out[2]: 140080785163184

In [3]: text += "a"

In [4]: id(text)
Out[4]: 140080783969328

In [5]: text += "b"

In [6]: id(text)
Out[6]: 140080744801392

In [7]: text
Out[7]: 'ab'
```

I to w zasadzie prawie wszystko. Na koniec wspomnę tylko, że napisy możemy też potraktować jak kolekcję. Kolekcję znaków. Jeśli taka kolekcja jest uporządkowana, to możemy mówić o tym, że każdy znak ma swój indeks, swoją pozycję. Te pozycje liczymy od 0. Na przykład:

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

Tabela. 1. Indeksy znaków

Właśnie te pozycje sprawdza metoda `index` (Listing 15).

Listing 15. Pozycje znaków w napisie

```
In [9]: "Programista".index("P")
Out[9]: 0

In [10]: "Programista".index("m")
Out[10]: 6

In [11]: "Programista".index("a")
Out[11]: 5
```

Jak widać, dane zgadzają się z Tabelą 1. W przypadku litery „a” zwracane jest pierwsze wystąpienie tego znaku. Możemy też sprawdzać od prawej strony. Wtedy pierwsze wystąpienie „a” będzie miało indeks 10. Służy do tego metoda `rindex`.

Do indeksów, kolekcji wrócimy już w następnym artykule, w którym poznamy pierwsze struktury danych, takie jak tak zwane tuple (krotki) czy listy. A teraz zachęcam cię do własnych eksperymentów z napisami. Pamiętaj o tym, że zawsze możesz skorzystać z metody `help` czy dokumentacji języka na stronie https://python.org, by dowiedzieć się lepiej, jak działają wybrane metody. Nic jednak nie zastąpi praktyki. Teraz mamy sporo czasu na takie eksperymenty, warto go więc dobrze wykorzystać. Życzę wam dużo zdrowia i cierpliwości. Dbajcie o siebie i swoich bliskich. Do usłyszenia!

Rafał Korzeniewski

Z wykształcenia muzyk-puzonista i fizyk. Z zamiłowania i zawodu Pythonista. Trener Pythona w ALX, współorganizator PyWaw (http://pywaw.org) – warszawskiego meetupu poświęconego Pythonowi. W wolnych chwilach uczy się gry na nowych instrumentach, udziela się społecznie i dużo czyta.

KORZENIEWSKI@GMAIL.COM