Python #5: Modyfikujemy napisy

Napisy jako typ sekwencyjnym

Typ sekwencyjny (złożony) oznacza, że w jednej zmiennej możemy przechować wiele wartości (w przeciwieństwie do typów prostych, które mogą przechowywać tylko jedną wartość np. int).
Jeden napis (łańcuch znaków) może zawierać wiele znaków.

```
>>> napis1="ciasto czekoladowe"
>>> napis2="babeczki truskawkowe"
```

Aby sprawdzić ile znaków zawiera łańcuch możemy „zmierzyć" jego długość funkcją **len()**.

```
>>> len(napis1)
18
>>> len(napis2)
20
>>> len(napis1+napis2)
38
>>> len(napis1)+len(napis2)
38
```

Jaka różnica jest między przedostatnim i ostatnim sprawdzeniem.

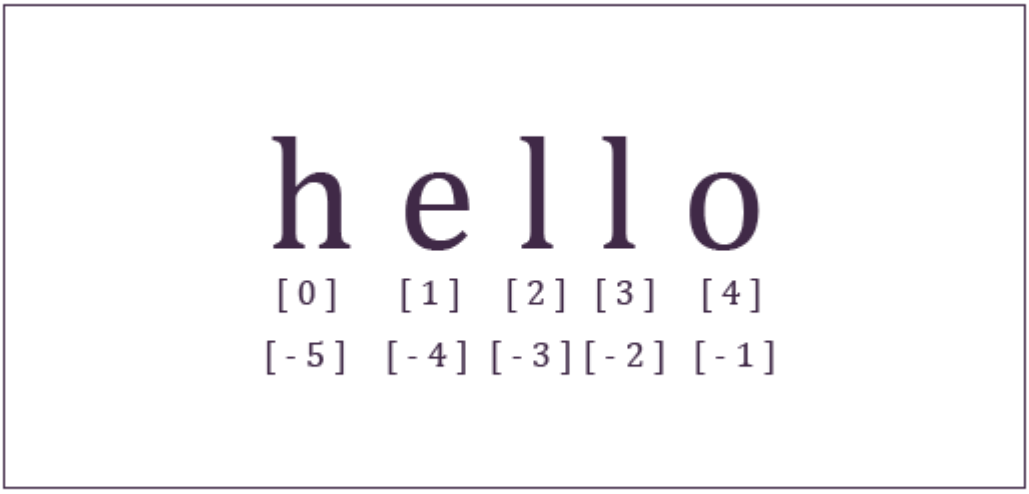
Typ sekwencyjny pozwala na dostęp do każdego swojego elementu z osobna – analogicznie jak w poznanych do tej pory listach i krotkach.

Kolejne elementy – znaki numerowane są od 0.

```
>>> txt="hello"
>>> txt[0]
h
>>> txt[1]
e
>>> txt[2]+txt[3]
ll
```

Jednocześnie możemy cofać się po elementach napisu. Elementem o indeksie [-1] będzie ostatni znak w łańcuchu.

```
>>> txt[-1]
o
>>> txt[-2]
l
>>> txt[-3]
l
```



Co się stanie jeśli spróbujemy dostać się **do elementu, który nie istnieje** np. `txt[5]` czy `txt[-6]`?

IndexError: string index out of range

Python wyrzuca nam prosty komunikat o wyjściu poza zakres.

Załóżmy, że w wyrazie hello, chcemy zamienić `e` na literę `a`.
Może wystarczy zapisać `txt[1] = 'a'`? Nie, to nie zadziała.
Stringi w Pythonie są niemutowalne (niezmienne)!

Rozwiązać, można to np. `txt = txt[0] + 'a' + txt[2:]`.

To nie koniec rzeczy, które możemy zrobić dzięki temu, że napis jest sekwencją znaków. **String slicing** pozwala na wycięcie części napisu i zwrócenie jako nowy napis. Do tej pory każda pojedynczo wyciągnięta litera stawała się nowym 1 elementowym napisem, teraz możemy wyciąć kawałek dowolnej wielkości.

```
>>> txt[1:4]
ell
>>> napis1[4:10]
to cze
```

Przyjrzyj się uważnie jak zostały wycięte znaki. Od pozycji [1] do pozycji 4, ale bez uwzględnienia znaku na miejscu [4]. Tak samo dla napisu drugiego, zostały wycięte znaki od [4] (czyli znaku piątego, gdyż liczymy od zera) do znaku 10, ale bez znaku na miejscu [10].

```
>>> txt[1:] #zwraca od pos 1 do końca
ello
>>> napis1[:10] #zwraca od początku do pos 10
ciasto cze
```

Metody klasy String

Python posiada zbiór metod, za pomocą, których możecie łatwo modyfikować napisy. Podzielone są tu na tabele wg tego co zwracają, dzięki czemu nauka powinna być łatwiejsza, a w razie potrzeby szybko znajdziecie potrzebną wam metodę (a gwarantuję jeszcze nie raz się przydadzą).
Metod używamy **na stringu (napisie)** według wzoru `string.metoda()`, z wyjątkiem metod – len, max, min, które string przymują wewnątrz metody.

Metody zwracające nowy napis (string):

Metoda	Znaczenie
`lower()`	Zmienia wszystkie duże litery na małe w stringu
`upper()`	Zmienia wszystkie małe litery na duże w stringu
`swapcase()`	Odwraca rodzaj każdej litery – małe na duże, duże na małe
`capitalize()`	Zmienia pierwszą literę w ciągu na dużą
`title()`	Zwraca string – tytuł, w którym wszystkie wyrazy zaczynają się dużą literą, a reszta jest małymi lub są to znaki nieliterowe
`join(seq)`	Łączenie (konkatenacja) wyrazów w napisie `seq` w jeden napis, według separatora/stringu na jakim wywołujemy metodę
`lstrip()`	Usuwa białe znaki z początku napisu – zwraca kopię pozbawioną białych znaków od lewej strony
`rstrip()`	Usuwa białe znaki z końca napisu – zwraca kopię pozbawioną białych znaków od prawej strony
`strip([chars])`	Usuwa białe znaki lub znak podanny jako `char` z początku i końca napisu – wykonuje lstrip() i rstrip() na napisie.
`max(string)`	Zwraca literę znajdującą się najdalej w alfabecie od A
`min(string)`	Zwraca literę znajdującą się najbliżej w alfabecie od A

Metoda	Znaczenie
`split(str="", num=string.count(str))`	Dzieli łańcuch według separatora `str` (spacja jeśli nie podano) i zwraca podciągi jako listę lub podzieli na co najwyżej liczbę podciągów, jeśli podano `num`
`splitlines( num=string.count('n'))`	Dzieli cały łańcuch (lub wg zadanej liczby `num`) na osobne linie wg znaku nowej linii'\n' i zwraca je jako tablicę
`replace(old, new [, max])`	Zamienia wszystkie wystąpienia ciagu `old` na ciąg `new` lub jeśli jest podane max – podmiana zostanie wykonana o wskazaną liczbę wystąpień

Przykłady użycia:

```
>>> text = "ala ma morsa"
ala ma kota
>>> text.upper()
ALA MA MORSA
>>> "*".join("abrakadabra")
a*b*r*a*k*a*d*a*b*r*a
>>> "abrakadabra".strip("a")
brakadabr
>>> text.replace(" ","-")
ala-ma-morsa
```

Metody zwracające wartość liczbową:

Metoda	Znaczenie
`len(string)`	Zwraca długość ciągu znaków
`count(str, beg=0,end=len(string))`	Zlicza ile razy zadany ciąg znaków(str) wystąpił w ciągu znaków lub wewnątrz podciągu, który zaczyna się od indeksu `beg` i kończy indeksem `end`
`find(str, beg=0 end=len(string))`	Sprawdza gdzie ciąg str występuje w napisie lub podciągu tego napisu jeśli podamy index`beg` i indeks końcowy `end`. Zwraca indeks początkowy lub -1 jeśli ciąg str nie znajduje się w napisie
`rfind(str, beg=0,end=len(string))`	Działa jak find(), ale wyszukiwanie od końca ciągu znaków

Metoda	Znaczenie
`index(str, beg=0, end=len(string))`	Działa jak find(), ale zwraca wyjątek jeśli ciąg str nie zostanie znaleziony
`rindex( str, beg=0, end=len(string))`	Działa jak index(), ale wyszukiwanie od końca ciągu znaków

Przykłady użycia:

```
>>> text = "ala ma morsa"
ala ma kota
>>> text.count('a')
4
>>> text.count('a', 3)
2
>>> text.count('a', 3,10)
1
>>> text.find("a ")
2
>>> text.rfind("a ")
5
```

Metody zwracające `true`/`false`:

Metoda	Znaczenie
`isalnum()`	Zwraca `true` jeśli wszystkie znaki w ciągu są alfanumeryczne (litery lub cyfry)
`isalpha()`	Zwraca `true` jeśli wszystkie znaki w ciągu są literami
`isdigit()`	Zwraca `true` jeśli wszystkie znaki w ciągu są cyframi
`islower()`	Zwraca `true` jeśli wszystkie znaki w ciągu są małymi literami.
`isspace()`	Zwraca `true` jeśli wszystkie znaki w ciągu są białymi znakami (spacja, tabulacja, przejście do nowej linii itp)
`istitle()`	Zwraca `true` jeśli ciąg spełnia warunek tytułu (każdy wyraz napisu musi zaczynać się dużą literą i składać wyłącznie z małych liter lub znaków nieliterowych)

Metoda	Znaczenie
`isupper()`	Zwraca `true` jeśli ciąg ma co najmniej jedną dużą literę.
`startswith(str, beg=0,end=len(string))`	Zwraca wynik sprawdzenia, czy napis jest rozpoczęty ciągiem str. Przy podaniu indeksu `beg`, sprawdzenie rozpoczyna się od tego znaku. Przy wystąpieniu argumentu `end` sprawdzenie zakończy się na tym znaku
`endswith(str, beg=0, end=len(string))`	Zwraca wynik sprawdzenia, czy napis jest zakończony ciągiem str. Przy podaniu indeksu `beg`, sprawdzenie rozpoczyna się od tego znaku. Przy wystąpieniu argumentu `end` sprawdzenie zakończy się na tym znaku

Przykład użycia:

```
>>> text = "ala ma morsa"
ala ma kota
>>> text.isalnum()
True
>>> text.isupper()
False
>>> "Python Jest Fajny".istitle()
True
```

Polecam przetestować każdą z metod w konsoli, aby przekonać się, że opisy ich działania są zrozumiałe!

Zadanie 1 – rozgrzewka

Podpunktów nie trzeba wykonywać po kolei, jeśli czegoś nie pamiętasz – idź dalej. Możesz przeczytać wpis ponownie i wrócić do pozostawionego zadania

Do zmiennej `sentence` przypisz zdanie: „Kurs Pythona jest prosty i przyjemny.”, a następnie:

1. Policz wszystkie znaki w napisie
2. Nie modyfikując zmiennej `sentence` wyświetl słowo „prosty”
3. Wyświetl znak o indeksie (czy za każdym razem rozumiesz co się dzieje?):
    - 7
    - 12
    - -4
    - 37
4. Wprowadź do zdania 2 błędy ortograficzne

Zadanie 2

Utwórz skrypt, który zapyta użytkownika o imię, nazwisko i numer telefonu, a następnie:

1. Sprawdź czy imię i nazwisko składają się tylko z liter, a nr tel składa się wyłącznie z cyfr (wyświetl tę informację jako true/false)
2. Użytkownicy bywają leniwi. Nie zawsze zapisują imię czy nazwisko z dużej litery – popraw ich
3. Niektórzy podają numer telefonu z myślnikami lub z spacjami, usuń zbędne znaki z numeru
4. Zakładając, że twoi użytkownicy noszą polskie imiona, sprawdź czy użytkownik jest kobietą
5. Połącz dane w jeden ciąg `personal` za pomocą spacji
6. Policz liczbę wszystkich znaków w napisie `personal`
7. Podaj liczbę tylko liter w napisie `personal`

Podpowiedź – weź pod uwagę, że numery telefonów w Polsce są 9-cyfrowe

Zadanie 3*

Łatwiejsze niż się wydaje;

Wyobraź sobie, że jesteś bioinformatykiem i otrzymujesz kod genetyczny do analizy.

Kod DNA składa się z 4 zasad azotowych: adeniny(A), cytozyny(D), guaniny(G), tyminy(T). Idealny kod DNA wygląda następująco:
TGCACGATCATGTCTACTATCCTCTCTATGGTGGGGTT.

Zdarza się, jednak, że kod zawiera małe jak i duże litery. Kolejny problem to maszyny sekwencjonujące nie są wolne od błędów. W zależności od maszyny błędy sekwencjonowania mogą zostać zamienione na znak – czy literę N.

- W jaki sposób łatwo rozpoznasz, że otrzymany kod DNA zawiera błędy?

Dotarł do ciebie następujący kod genetyczny:

```
ACTGTGCTGACTCCCGGTGCTGCCGCTGCCATAGCTAAAGCCCGGGTCCTGGTAGGCAGGCGGGAAGCAG
GGTGGGGGTCCCGGGTACTGGTAGGGGTAGCCCTGACCCAGAGGCGGGGGGGCAGCCGGGTGGGGCAGCG
GGGCCAGCGTGTCCTGAA-CGAAGTCCCACTGGAGCCACTGTTGAGGTTCAGGGTGGCGAGATCTGGCGG
NNNAGGGTAGGTGAGGGCCGCGGAGGGGCCTCCGGCGTTCCCCTCCCCCCCGCCCTGAAACCCGAAGCCC
CCACTCACTGCTGCAGAGATCCCCTGAAAACGTAGTAGCACTGCTCgagacAGGTGATCTGTTGACCTGA
AACCGCAGGAAGCCGTGCTTCAGCAAGCTGCTGGCGTACTTCCGGGCCT---GCCGCTCCTTGAAGCCCT
CCACGTGTGTGTACAGCCAGTCCACCACGTCCGCCCCTGGCCGGCACCAGCGGTCAGCCCGCAGCCTCGA
GGCAAGCAGCCCTGCCNNTGGCACTATCCGC-CGCGGGGACGGCCACTCACCGATGACGGCATNNGCGAT
GGTGATCTTGAGCCACATGCGGTCGCGGATCTCCAGTCCCGAG---GGCAGCTGCATGACCCGGACGACG
GCGCTCATGTCACtcaccgtcagcggcgcctcttccagCCAGCTCTGCAAAGCACAGACAGCCCCGCTTC
GCCCCAGCATCTGAAAGCGGGGGACTCggcAcgCTGCACCCCCAGGGGAGCCTCTGGGCAGAGCCTGCGC
CAGGGCGCAAGCTGGACGGTGCGTGACAGCAGGGCCCCGGCCCACTGCAGGATGCACCCCCGTGAGGCTG
GGGCGTGAGCAGGGGGGTTGGACAtttAGTCTCCCACTTCTACAGACACTTTTCATCAGGATCCTAGGCA
CAAACTGGGCTGAAACCCCACCCTGCAGACCAGGAAGTAATGAGAACAGGGCAGGCCCCTTCCCCTCNNC
GCATGCC-CACCCGAGAGCGCAGGCTGTTAGTCGTGTTAATGGCAGGAAGCAGAATGGAGACCTGGCCCC
TGCCTCTGAA-CCGTGGGTGCTCaactggctaGCCCTACGTACATCCCCTGTTCcggCCAACACACAGAC
ATGAGCAGGATGGGCTGCACAAGGTGGGCACGGGTGCCTGTGCACACGTCTGTGCAGGGAGTTGGGGACA
GGCAACACACACGTGTCACAGCCCCATGACGGggcaattgcGCCATGCTGGCTGAATGGCAGAGACGCCC
CTCCAAGCCTCGGTTTCTGCTGGGGCCCTCAGGAGCTGCCACTTACGTGGAGCACCAGGCACGGAGCTGG
TTAGTGAGGAGGAGCTGGTGCGCGTGACGGCGCTGGAGCAGGGACTCGTACCGTAGCGGGGCAGGGCNNN
TGTCAGTGCCGCCGTGTGGtcagcggcgatCGGCG-GGTCGATGGGCCGCACCGGGTCAGCTGGGTGNAG
ACACGTGGCGATGACAGGCGGACAGATGGACAGGGTGGGAGGGCAGGGTGCAGGGCACAGAGGAGAGAGG
CCTTCAGGCTAGGTAGGCGCCCCCTCCCCATCCCGccccGTGTGCCCCGAGGGCCACTCACCCCGTGGGA
CGGTGAAGTAGCTTCG-GGCGTTGGGTCCAGCACTTGGCCACAGTGAGGCTGNAAATGGCTGCAGGAACG
GTGGTCCCCCCGCAAGGCCCCCATGGTCCCACCTCCCTGCCTGGCCCCTCCCGCTCCAGCGCCNCCAGCC
```

- Skopiuj kod genetyczny do swojego skryptu i zapisz jako `DNA = ACTG...`
- Policz ile razy występuje w kodzie każda zasada azotowa – adenina, cytozyna, guanina, tymina.

Na pewno zauważysz błędy sekwencjonowania – znaki, które nie są żadną z 4 zasad.
W panice szukasz pliku z dokumentacją. Ufff... znalazł się!
Co więcej, pojawił się zapis, że *gdy jakość sekwencji nie pozwala dokładnie odczytać rodzaju zasady azotowej wstawiany jest znak „-" Natomiast, gdy laser sczytujący ześlizgnie się wstawiane są litery „N", jednocześnie ostatnia wartość zasady jest ponownie zaczytywana bez ubytku zasady w tym miejscu.* Co za przydatna informacja!

- Policz wystąpienia sekwencji GAGA i zamień je na AGAG
- Znajdź miejsce (indeks) w łańcuchu, gdzie występuje 7 guanin pod rząd
- Znajdź miejsce (indeks) , gdzie od końca łańcucha występuje 6 cytozyn
- Policz ile razy w kodzie pojawiła się sekwencja CTGAAA
- W sekwencji CTGAAA czasem mutuje ostania litera A, wówczas jakość ostatniej litery może być wątpliwa. Ile sekwencji znajdziesz, jeśli weźmiesz pod uwagę wątpliwą, ostatnią adeninę?
- Oczyść DNA z wszystkich błędów. Na podstawie czystej nici utwórz odpowiadającą jej nić RNA (nić RNA w miejscu tyminy będzie mieć uracyl (U))