Ciągi znaków

Rozdział 6

UMSI

Python dla wszystkich
www.py4e.pl

Typ danych: ciąg znaków

- Ciąg znaków to sekwencja znaków
- Literał ciągu zapisuje się w apostrofach 'Witaj' lub cudzysłowach "Witaj"
- W przypadku ciągów + oznacza "konkatenację"
- Ciąg zawierający cyfry nadal jest ciągiem znaków
- Można skonwertować go do liczby, korzystając z int()

```
>>> str1 = "Hej"
>>> str2 = 'tam'
>>> bob = str1 + str2
>>> print(bob)
Hejtam
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

Wczytywanie i konwersja

- Najlepiej wczytywać dane jako ciągi znaków, a następnie parsować i konwertować według potrzeb
- Daje nam to więcej kontroli w przypadku błędów lub złego typu danych
- Liczby otrzymujemy, konwertując ciągi znaków

```
>>> name = input('Wpisz: ')
Wpisz: Chuck
>>> print(name)
Chuck
>>> apple = input('Wpisz: ')
Wpisz: 100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```

Przyjrzyjmy się ciągom

- Możemy wybrać dowolny pojedynczy znak z ciągu za pomocą indeksu określonego w nawiasach kwadratowych
- Wartość indeksu musi być liczbą całkowitą od zera wzwyż
- Wartość indeksu może być określona wyrażeniem matematycznym

b	a	n	a	n
0	1	2	3	4

```
>>> fruit = 'banan'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

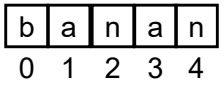
O jeden znak za daleko

- Python zwróci błąd, jeśli spróbujesz indeksu wskazującego poza koniec ciągu
- Uważaj, tworząc wartości indeksów i wycinków

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```

Ciągi mają długość

Wbudowana funkcja len podaje nam długość ciągu znaków

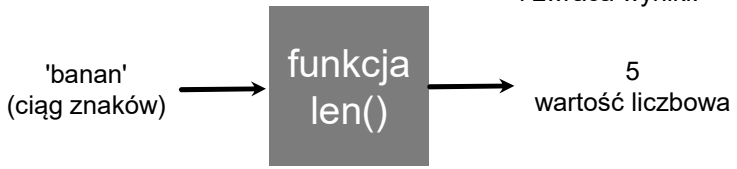



```
>>> fruit = 'banan'
>>> print(len(fruit))
5
```

Funkcja len()

```
>>> fruit = 'banan'
>>> x = len(fruit)
>>> print(x)
5
```

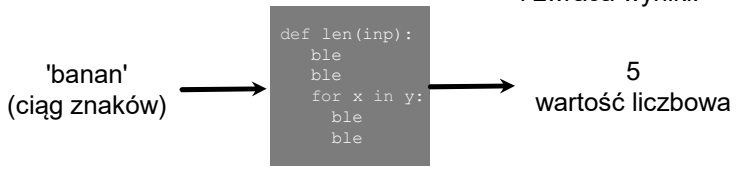
Funkcja to zapisany kod, z którego korzystamy. Funkcja przyjmuje wartości wejściowe i zwraca wyniki.



Funkcja len()

```
>>> fruit = 'banan'
>>> x = len(fruit)
>>> print(x)
5
```

Funkcja to zapisany kod, z którego korzystamy. Funkcja przyjmuje wartości wejściowe i zwraca wyniki.



Przechodzenie pętlą przez ciąg

Korzystając z instrukcji while, zmiennej sterującej, i funkcji len(), możemy stworzyć pętlę, która przyjrzy się każdej literze z osobna

```
fruit = 'banan'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

```
0 b
1 a
2 n
3 a
4 n
```

Przechodzenie pętlą przez ciąg

- Skończona pętla korzystająca z instrukcji for jest bardziej elegancka
- Zmienną sterującą zajmie się za nas pętla for

```
fruit = 'banan'
for letter in fruit:
    print(letter)
```

```
b
a
n
a
n
```

Przechodzenie pętlą przez ciąg

- Skończona pętla korzystająca z instrukcji for jest bardziej elegancka
- Zmienną sterującą zajmie się za nas pętla for

```
fruit = 'banan'
for letter in fruit :
    print(letter)


index = 0
while index < len(fruit) :
    letter = fruit[index]
    print(letter)
    index = index + 1
```

```
b
a
n
a
n
```

Pętle i liczenie

Oto prosta pętla, która przechodzi przez każdą literę w ciągu i liczy, ile razy pętla napotkała znak 'a'

```
word = 'banan'
count = 0
for letter in word :
    if letter == 'a' :
       count = count + 1
print(count)
```

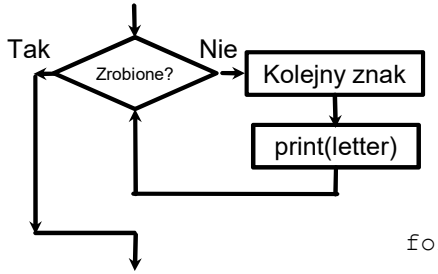
Bliższe spojrzenie na in

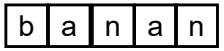
- Zmienna sterująca "przechodzi" przez sekwencję (uporządkowany zbiór)
- Blok (ciało) kodu jest wykonywany jeden raz dla każdego elementu w sekwencji
- Zmienna sterująca przechodzi przez wszystkie elementy w sekwencji

Zmienna sterująca

pięcioznakowy ciąg

```
for letter in 'banan' :
    print(letter)
```






```
for letter in 'banan' :
    print(letter)
```

Zmienna sterująca "przechodzi" przez ciąg i blok (ciało) kodu jest wykonywany jeden raz dla każdego elementu w sekwencji

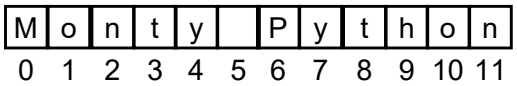
Więcej operacji na ciągach

Wycinki ciągów

- Możemy też spojrzeć na kilka kolejnych znaków w ciągu, używając operatora dwukropka
- Druga wartość jest o jeden większa niż koniec wycinka – "do, ale nie razem z"
- Jeśli druga wartość jest większa niż koniec ciągu, to zatrzymujemy się na nim

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```

Wycinki ciągów



Jeśli opuścimy pierwszą (lub drugą) wartość określającą wycinek, to Python zacznie od początku (albo końca) ciągu.

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

GD dodatek


```
lancuch[start:stop:step]
```

GD dodatek


```
lancuch[start:stop:step]
```

Domyślnie równe 1
(patrz poprzednie slajdy)

GD dodatek


```
lancuch[start:stop:step]
lancuch[::2]
```

Co to robi?

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Jak wyświetlić tylko ostatni znak łańcucha?

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Jak wyświetlić tylko ostatni znak łańcucha?
1. Policzyć znaki w łańcuchu od zera i znaleźć indeks ostatniego znaku

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
print(lancuch[11])
```

Jak wyświetlić tylko ostatni znak łańcucha?
1. Policzyć znaki w łańcuchu od zera i znaleźć indeks ostatniego znaku

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Jak wyświetlić tylko ostatni znak łańcucha?
2. Wykorzystać len(lancuch)

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

`len(lancuch) = 12`

```
print(lancuch[len(lancuch) - 1])
```

Jak wyświetlić tylko ostatni znak łańcucha?
2. Wykorzystać len(lancuch)

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Jak wyświetlić tylko ostatni znak łańcucha?
3. Poszukać, poczytać, zastosować

GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Jak wyświetlić tylko ostatni znak łańcucha?
3. Poszukać, poczytać, zastosować

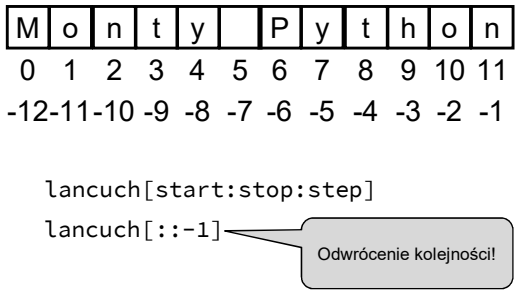
GD dodatek

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
print(lancuch[-1])
```

Jak wyświetlić tylko ostatni znak łańcucha?
3. Poszukać, poczytać, zastosować

GD dodatek



Konkatenacja ciągów znaków

Zastosowanie operatora + do ciągów oznacza "konkatenację", czyli łączenie

```
>>> a = 'Hej'
>>> b = a + 'tam'
>>> print(b)
Hejtam
>>> c = a + ' ' + 'tam'
>>> print(c)
Hej tam
>>>
```

Użycie in jako operatora logicznego

- Słowo kluczowe in może też służyć do sprawdzenia, czy jeden ciąg jest "w" innym ciągu
- in jest wyrażeniem logicznym, które zwraca wartość True lub False i może być używane w instrukcji if

```
>>> fruit = 'banan'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Jest!')
...
Jest!
>>>
```

Porównywanie ciągów znaków

```
if word == 'banan':
    print('Okej, to banan.')

if word < 'banan':
    print('Twoje słowo,' + word + ', jest przed bananem.')
elif word > 'banan':
    print('Twoje słowo,' + word + ', jest po bananie.')
else:
    print('Okej, to banan.')
```

Biblioteka ciągów znaków

- Python ma wiele funkcji zapisanych w bibliotece ciągów znaków
- Są to funkcje wbudowane w każdy ciąg znaków – wywołujemy je, dodając funkcję do zmiennej ciągu
- Te funkcje nie modyfikują oryginalnego ciągu, zwracają nowy, zmodyfikowany ciąg znaków

```
>>> greet = 'Witaj Bob'
>>> zap = greet.lower()
>>> print(zap)
witaj bob
>>> print(greet)
Witaj Bob
>>> print('Hej Tam'.lower())
hej tam
>>>
```

```
>>> stuff = 'Witaj świecie'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', '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']
```

https://docs.python.org/3/library/stdtypes.html#string-methods

`str.`**`replace`**(*old*, *new*[, *count*])
Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.`**`rfind`**(*sub*[, *start*[, *end*]])
Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.`**`rindex`**(*sub*[, *start*[, *end*]])
Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.`**`rjust`**(*width*[, *fillchar*])
Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.`**`rpartition`**(*sep*)
Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

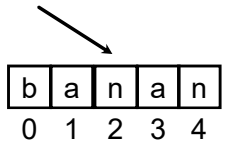
`str.`**`rsplit`**(*sep=None*, *maxsplit=-1*)
Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

Biblioteka ciągów znaków

```
str.capitalize()
str.center(width[, fillchar])
str.endswith(suffix[, start[, end]])
str.find(sub[, start[, end]])
str.lstrip([chars])
str.replace(old, new[, count])
str.lower()
str.lstrip([chars])
str.strip([chars])
str.upper()
```

Wyszukiwanie w ciągu znaków

- Funkcji find() używamy do szukania jednego podciągu ciągu w innym
- find() odnajduje pierwsze wystąpienie podciągu
- Jeśli podciąg nie został znaleziony, find() zwraca -1
- Pamiętaj, że pozycje w ciągach zaczynają się od zera




```
>>> fruit = 'banan'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

Wszystko DUŻYMI LITERAMI

- Możesz stworzyć kopię ciągu pisaną małymi albo dużymi literami
- Często gdy przeszukujemy ciąg, używając find(), najpierw konwertujemy go na małe litery, żeby wyszukiwać bez względu na wielkość znaków

```
>>> greet = 'Witaj Bob'
>>> nnn = greet.upper()
>>> print(nnn)
WITAJ BOB
>>> www = greet.lower()
>>> print(www)
witaj bob
>>>
```

Znajdź i zamień

- Funkcja replace() jest podobna do operacji “znajdź i zamień” w edytorze tekstu
- Zamienia wszystkie wystąpienia wyszukiwanego ciągu na nowy

```
>>> greet = 'HaloBob'
>>> nstr = greet.replace('Bob','Jane')
>>> print(nstr)
Halo Jane
>>> nstr = greet.replace('o','X')
>>> print(nstr)
HalX BXb
>>>
```

Usuwanie białych znaków

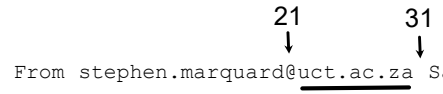
- Czasami chcemy usunąć białe znaki z początku/końca ciągu
- lstrip() i rstrip() usuwają białe znaki z lewej i prawej strony
- strip() usuwa białe znaki z początku i końca

```
>>> greet = '   Halo Bob  '
>>> greet.lstrip()
'Halo Bob  '
>>> greet.rstrip()
'   Halo Bob'
>>> greet.strip()
'Halo Bob'
>>>
```

Prefiksy

```
>>> line = 'Życzę miłego dnia'
>>> line.startswith('Życzę')
True
>>> line.startswith('ż')
False
```

Parsowanie i wyodrębnianie




```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```

Dwa rodzaje ciągów znaków

```
Python 2
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

```
Python 3
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

W Pythonie 3 wszystkie ciągi stosują Unicode

Podsumowanie

- Typ danych: ciąg znaków
- Czytanie/Konwersja
- Indeksowanie ciągów []
- Wycinki ciągów [2:4]
- Przechodzenie przez ciąg pętlą for i while
- Konkatenacja ciągów opertorem +
- Operacje na ciągach znaków
- Biblioteka ciągów znaków
- Porównywanie ciągów znaków
- Wyszukiwanie w ciągu znaków
- Zamienianie tekstu
- Usuwanie białych znaków

Podziękowania dla współpracowników

Autorstwo pierwszej wersji: Charles Severance, University of Michigan School of Information

Polska wersja powstała z inicjatywy Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu

Tłumaczenie: Agata i Krzysztof Wierzbiccy, EnglishT.eu

... wstaw tu nowych współpracowników i tłumaczy

...