Szczecin, rok akademicki 2019

Wydział Informatyki

Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
WYDZIAŁ INFORMATYKI

Praca zaliczeniowa z przedmiotu “Projektowanie systemów wieloagentowych”
Prowadzący: prof.dr hab.inż. Walery Rogoza

Autor
Marcin Ziajkowski

Przetwarzanie danych tekstowych z wykorzystanie języka Python

Python jest językiem interpretowanym wysokiego poziomu ogólnego przeznaczenia. Interpretery Pythona są dostępne na każdą platformę więc jest też między platformowy. Jego największa zaletą jest ogromna liczba dobrze udokumentowanych bibliotek typu open source.

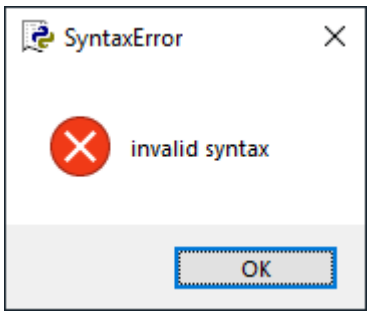
Python wykorzystywany jest główne to przeprowadzania obciążających procesor obliczeń (np. głębokie uczenie (eng. Deep learning)), obróbki tekstu lub analizy Big Data. Skupię się na obróbce tekstu. Do tego niezbędna jest wiedza z zakresu edytowania danych typu string.

W pythonie zmienne stringowe są obiektami które mają swoje id. Inaczej niż w innych popularnych dynamicznych językach nie wolno tworzyć zmiennych typowanych. To znaczy że taki zapis jest błędny:

```
int   zmienna1 = 1;
float zmienna2 = 2.2;
str   zmienna3 = „333”;
```

Przykładowa zła implementacja zmiennych w Pythonie.

Taki zapis skutkuje błędem:



Przykładowy błąd uruchomienia błędnego kodu (podanie typu zmiennej).

Podstawowe operacje tekstu

Typy danych są rozpoznawane jako sugestia (w parametrach funkcji) lub funkcja kastująca.
Aby wyświetlić zawartość zmiennej należy użyć funkcji
print*(object(s)*, separator=*separator*, end=*end*, file=*file*, flush=*flush*)
gdzie:
object(s) oznacza obiekty które zostaną przekonwertowane do stringa i wyświetlone. Można wprowadzać je po przecinku.
separator (sep dla pythona 3.x) oznacza sposób w jaki poszczególne elementy są oddzielane.
end wyznacza co zostanie wstawione na końcu linii. Domyślnie jest to „\n” (znak nowej linii).
file obiekt do którego dane są wpisywane. Domyślnie jest to sys.stdout.
flush parametr typu boolean. Precyzuje czy dane z funkcji trafią na wyjście od razu czy najpierw trafia do bufora. Domyślnie ustawione jest False czyli dane są buforowane.

Input:

```
zmiennal = "lll"
zmienna2 = "222"
zmienna3 = "333"

print(zmiennal,zmienna2,zmienna3,"444",sep=" + ",end=" $$ ")
```

Output:

```
lll + 222 + 333 + 444 $$
>>>
```

Jeżeli zmienne nie będą typu string zostaną niejawnie kastowane do str. Jednak możemy to zrobić jawnie:

Input:

```
# zmienna typu string
zmiennal = "lll"
# zmienna typu int
zmienna2 = 222
# zmienna typu float
zmienna3 = 3.3
# zmienna typu lista
zmienna4 = ["lista", 4, 5.5]

print(zmiennal,    zmienna2,     zmienna3,     zmienna4, "444",sep=" + ",end=" $$ \n")
print(zmiennal,str(zmienna2),str(zmienna3),str(zmienna4),"444",sep=" + ",end=" $$ \n")
```

Output:

```
lll + 222 + 3.3 + ['lista', 4, 5.5] + 444 $$
lll + 222 + 3.3 + ['lista', 4, 5.5] + 444 $$
>>>
```

Analogicznie można kastować string na liczbę. Przykładowo dodawanie kastowanej zmiennej typu string „111” do zmiennej typu int 222, da wynik liczby całkowitej.

Input:

```
zmienna5 = int(zmienna1) + zmienna2

print("string plus int:",zmienna5,sep="\n")
```

Output:

```
string plus int:
333
>>>
```

Cały kod znajduje się w skrypcie nr 1.

Specjalne operatory stringów

Na typach liczbowych można wykonywać operacje arytmetyczne takie jak dodawanie czy mnożenie. Python umożliwia operacje arytmetyczne na typach stringowych.

Input:

```
zmienna6 = "Hello "
print("operator +",zmienna6 + "world",                                    sep=" : ",end=" $$ \n")
print("operator *",zmienna6 * 3,                                          sep=" : ",end=" $$ \n")
print("operator []",zmienna6[0],                                          sep=" : ",end=" $$ \n")
print("operator []",zmienna6[-1],                                         sep=" : ",end=" $$ \n")
print("operator [:]",zmienna6[1:-2],                                      sep=" : ",end=" $$ \n")
print("operator in","e" in zmienna6,                                      sep=" : ",end=" $$ \n")
print("operator not in","k" not in zmienna6,                              sep=" : ",end=" $$ \n")
print("operator r",r"\n\t",                                               sep=" : ",end=" $$ \n")
print("operator %"," slowo %s, liczba %d, float %f" %(zmienna6,13,2.14),  sep=" : ",end=" $$ \n")
```

Output:

```
operator + : Hello world $$
operator * : Hello Hello Hello  $$
operator [] : H $$
operator [] :   $$
operator [:] : ell $$
operator in : True $$
operator not in : True $$
operator r : \n\t $$
operator % :  slowo Hello , liczba 13, float 2.140000 $$
>>>
```

Nie jest dozwolone wykonywanie rzeczywistych operacji na stringach, takich jak dodawanie czy dzielenie liczb wewnątrz zdania. Natomiast podobny efekt można osiągnąć w inny sposób. Można użyć funkcji
eval(expression, globals=None, locals=None)
Gdzie:
Expresion oznacza wyrażenie matematyczne które chcemy wykonać zapisane jako string.
globals to słownik zmiennych.
locals to zmapowane oiekty.
Skupię się tylko na *expresion*.

Input:

```
zmienna7 = "1024 + 1024"
zmienna8 = "1024"
zmienna9 =  eval(zmienna7)
zmienna10 = eval("1024 * 2")
zmienna11 = eval(zmienna8 + " / " + "4")
print("zmienne 9,10,11 są nie jawnie kastowane do stringa")
print(zmienna9,zmienna10,zmienna11,sep="\n")
```

Output:

```
zmienne 9,10,11 są nie jawnie kastowane do stringa
2048
2048
256.0
>>>
```

Dzięki jawnemu kastowaniu wyników możemy zwiększyć czytelność dla użytkownika. W tym celu dodaj odpowiednią linię kodu:
print("dodawanie: "+str(zmienna9),"mnożenie: "+str(zmienna10),"dzielenie: "+str(zmienna11),sep="\n")

Taki zabieg sformatuje nasz output:

```
dodawanie: 2048
mnożenie: 2048
dzielenie: 256.0
>>>
```

Maksymalna liczba dodatnia jaką możemy zapisać w systemie 64 bitowym to prawie 19 MMM (M zastępuje 6 zer). Dzięki funkcji eval() można wykonywać operacje na liczbach znacznie większych niż pozwalają na to inne języki programowania. Dodatkowo jeżeli dodamy do skryptu bibliotekę math (przechowująca funkcje matematyczne), będziemy mogli wykonywać zaawansowane obliczenia na ogromnych liczbach w formacie string.

Python jest tak potężnym narzędziem że radzi sobie z ogromnymi liczbami nawet bez funkcji eval(), dlatego porównaj wyniki dla obliczeń na zmiennych liczbowych i tekstowych.

Input:

```
# zaawansowane używanie funkcji eval()
from math import *
M = "000000"
# liczba dużo większa niż możemy zapisać w systemie 64 bitowym
zmienna12 = "200"+M+M+M+M+M
print("wyniki dla wyrażenia:\nsqrt(("+zmienna12+"*99999999199999999)/1234567890987654321)\n")
print(eval("sqrt(("+zmienna12+"*99999999199999999)/1234567890987654321)",{"sqrt" : sqrt}))
print(sqrt((int(zmienna12)*99999999199999999)/1234567890987654321))
```

Output:

```
wyniki dla wyrażenia:
sqrt((200000000000000000000000000000000*99999999199999999)/1234567890987654321)

4024922360103360.0
4024922360103360.0
>>>
```

Jak widać wyniki są identyczne.

Cały kod znajduje się w skrypcie nr 1.

Edycja zmiennych tekstowych

W Pythonie nie są rozróżniane zmienne typu character i string. To znaczy że jeżeli zmienna przechowuje jeden znak to zawsze będzie stringiem. Nie mniej jednak istnieje możliwość pracy z pojedynczymi znakami. Do tego użyję funkcji chr() i ord(). Zgodnie z tabelą ASCII funkcja chr(numer) przyjmuje wartość liczbową i zwraca znak. Odwrotnie działa funkcja ord(znak) która przyjmuje jako parametr string z jednym znakiem i zwraca odpowiednio numer z tabeli ASCII.

Input:

```
zmienna01 = 50
zmienna02 = 'A'
zmienna03 = chr(zmienna01)
zmienna04 = ord(zmienna02)
finish = " w tabeli ASCII\n"
print(str(zmienna01)+": "+zmienna03,zmienna02+": "+str(zmienna04),sep=finish,end=finish)
```

Output:

```
50: 2 w tabeli ASCII
A: 65 w tabeli ASCII
```

Operacje wykonujemy zwykle na bardzo długich tekstach. W Pythonie zapis zmiennej tekstowej możemy wprowadzić zapisując tekst między apostrofami lub cudzysłowami. Bardzo wygodną formą zapisu tekstu jest użycie trzech apostrofów ‘’’. Wyjątkową cechą takiego zapisu jest możliwość wprowadzania tekstu nawet z nowymi liniami.

Input:

```
zmienna05 = 'któtki tekst między apostrofami'
zmienna06 = "któtki tekst między cudzysłowami"
zmienna07 = '''Bardzo długi tekst,
w którym dozwolone są znaki końca linii,
bez błędu kompilacji.
'''

print(zmienna05,zmienna06,zmienna07,sep="\n------------------------------\n")
```

Output:

```
któtki tekst między apostrofami
------------------------------
któtki tekst między cudzysłowami
------------------------------
Bardzo długi tekst,
w którym dozwolone są znaki końca linii,
bez błędu kompilacji.
```

Najczęściej używane metody do edycji takich tekstów to:
strip() - usuwa spacje z początku I końca tekstu.
capitalize() - powiększa pierwszą literę tekstu.
count(str, beg= 0,end=len(string)) - liczy wystąpienia wyrazu w tekscie.
find(str, beg=0 end=len(string)) - zwraca pierwsze wystąpienie wyrazu w tekscie.
istitle() - sprawdza czy tekst ma wszystkie wyracy pisane z dużych liter.
len(string) - zwraca długość tekstu.
join(seq) - łączy element sekwensji wyrażeniem stringowym.
upper () - zmniejsza wszystkie znaki w tekscie.
replace(old, new [, max]) - podmienia jedno wyrażenie na drugie w całym tekscie.
split(str="", num=string.count(str)) - zwraca listę elementów zawartych w tekscie.
title() - powiększa pierwsze litery każdego sowa w tekscie.
lower() - powiększa wszytkie znaki w tekscie.

Input:

```
commands =  ['strip()','capitalize()','count()','find()','istitle()','len()','join()','upper()']
commands += ['replace()','split()','title()','lower()']
zmienna08 = '''       __tekst__       '''
print(commands[0],"'"+zmienna08+"'","'"+zmienna08.strip()+"'",sep=" -> ")
zmienna08 = '''abc'''
print(commands[1],"'"+zmienna08+"'","'"+zmienna08.capitalize()+"'",sep=" -> ")
zmienna08 = '''Ala ma kota, ale altanka altruista'''
print(commands[2],"'"+zmienna08+"'",str(zmienna08.count("al"))+"x'al'",sep=" -> ")
print(commands[3],"'"+zmienna08+"'",str(zmienna08.find("al"))+" index",sep=" -> ")
zmienna08 = '''Book Real Title'''
print(commands[4],"'"+zmienna08+"'",str(zmienna08.istitle()),sep=" -> ")
zmienna08 = zmienna08.lower()
print(commands[4],"'"+zmienna08+"'",str(zmienna08.istitle()),sep=" -> ")
print(commands[5],"'"+zmienna08+"'",str(len(zmienna08)),sep=" -> ")
print(commands[6],"'"+zmienna08+"'","+".join(zmienna08),sep=" -> ")
print(commands[7],"'"+zmienna08+"'",zmienna08.upper(),sep=" -> ")
zmienna08 = '''Ala ma kota'''
print(commands[8],"'"+zmienna08+"'",zmienna08.replace("kota","małą myszkę"),sep=" -> ")
print(commands[9],"'"+zmienna08+"'",zmienna08.split(),sep=" -> ")
zmienna08 = '''Za DuŻo DużyCh ZnAkÓw...Za MaŁo MaŁyCH'''
print(commands[10],"'"+zmienna08+"'",zmienna08.title(),sep=" -> ")
print(commands[11],"'"+zmienna08+"'",zmienna08.lower(),sep=" -> ")
```

Output:

```
strip() -> '       __tekst__       ' -> '__tekst__'
capitalize() -> 'abc' -> 'Abc'
count() -> 'Ala ma kota, ale altanka altruista' -> 3x'al'
find() -> 'Ala ma kota, ale altanka altruista' -> 13 index
istitle() -> 'Book Real Title' -> True
istitle() -> 'book real title' -> False
len() -> 'book real title' -> 15
join() -> 'book real title' -> b+o+o+k+ +r+e+a+l+ +t+i+t+l+e
upper() -> 'book real title' -> BOOK REAL TITLE
replace() -> 'Ala ma kota' -> Ala ma małą myszkę
split() -> 'Ala ma kota' -> ['Ala', 'ma', 'kota']
title() -> 'Za DuŻo DużyCh ZnAkÓw...Za MaŁo MaŁyCH' -> Za Dużo Dużych Znaków...Za Mało Małych
lower() -> 'Za DuŻo DużyCh ZnAkÓw...Za MaŁo MaŁyCH' -> za dużo dużych znaków...za mało małych
>>>
```

Cały kod znajduje się w skrypcie nr 2.

Usuwanie i podmiana stringów

Ponieważ zmienne typu string w Python są trwałe (eng. immutable), nie można zmieniać lub usuwać pojedyńczych znaków wewnątrz zmiennej. Aby zmienić zawartość zmiennej trzeba ją całkowicie podmienić lub usunąć.

Input:

```
zmienna09 = "inicjalizacyjny napis"

print(zmienna09,"ID obiektu: ",id(zmienna09))
zmienna09 = "nowystring"
print(zmienna09,"ID obiektu: ",id(zmienna09))
del zmienna09
# użycie zmiennej zmienna09 jest nie legalne po usunięciu, dlatego kod niżej wywoła błąd.
#print(zmienna09,"ID obiektu: ",id(zmienna09))
```

Output:

```
inicjalizacyjny napis ID obiektu:  50699488
nowystring ID obiektu:  50693888
>>>
```

Cały kod znajduje się w skrypcie nr 2.

Sekwencje specjalne

Aby używać znaków specjalnych w tekscie używam lini pochylonych w lewo \ (backshlash).
Przykładowe znaki sekwencyjne to:
\n nowa linia
\" znak cudzysłów
\' znak apostrof
\t tabulacja
\\ znak '\'
Dodatkowo używając znaku '\' połączonego z 'x' można wyświetlać znaki z tablicy ASCII wpisując je szesnastkowo. Aby wyświetlić znaki specjalne bez dodatkowego edytowanie znakiem '\' można przed string wstawić literę 'r'. Jest to skrót od raw. Taki string jest nazywany surowym stringiem. Taki string nie rozpoznaje sekwencji specjalnych. Każdy znak trakuje jako prawdziwy znak.

Input:

```
zmienna10 = "standardowy tekst"
print(zmienna10)
zmienna10 = "\\ \\ \\ \"standardowy\t tekst\"\n\x49\x50\x51 \'\t\'"
print(zmienna10)
zmienna10 = r"\\ \\ \\ \"standardowy\t tekst\"\n\x49\x50\x51\n"
print(zmienna10)
```

Output:

```
standardowy tekst
\ \ \ "standardowy       tekst"
IPQ '    '
\\ \\ \\ \"standardowy\t tekst\"\n\x49\x50\x51\n
>>>
```

Cały kod znajduje się w skrypcie nr 2.

Formatowanie stringów

Python udostępnia specjalny sposób formatowania danych tekstowych metodą format().

Input:

```
# Standardowy porządek
zmiennall = "{} {} {} {}".format('Python', 'Obsluga', 'Danych', 'Tekstowych')
print("String wyświetlany normalnie:",zmiennall,sep="\n")
# Edycja pozycji
Stringl = "{2} {3} {0} {1}".format('Python', 'Obsluga', 'Danych', 'Tekstowych')
print("String wyświetlany w innej kolejności:",zmiennall,sep="\n")
# Formatowanie liczb całkowitych
zmiennall = "{0:b}".format(16)
print("Liczba całkowita biniarnie:",zmiennall,sep="\n")
# Formatowanie liczb zmiennoprzecinkowych
zmiennall = "{0:e}".format(165.6458)
print("Liczba 165.6458 zmiennoprzecinkowa:",zmiennall,sep="\n")
# Zaokrąglanie liczb
zmiennall = "{0:.2f}".format(1/6)
print("Liczba 1/6 zaokrąglona:",zmiennall,sep="\n")
# Justowanie i wyrównywanie tekstu
zmiennall = "|{:<12}|{:^10}|{:^10}|{:>12}|".format('Python', 'Obsluga', 'Danych', 'Tekstowych')
print("Wyrównany tekst:",zmiennall,sep="\n")
```

Output:

```
String wyświetlany normalnie:
Python Obsluga Danych Tekstowych
String wyświetlany w innej kolejności:
Python Obsluga Danych Tekstowych
Liczba całkowita biniarnie:
10000
Liczba 165.6458 zmiennoprzecinkowa:
1.656458e+02
Liczba 1/6 zaokrąglona:
0.17
Wyrównany tekst:
|Python      | Obsluga   |   Danych   |   Tekstowych|
>>>
```

Cały kod znajduje się w skrypcie nr 2.

Zadania

Aby jak najlepiej przyswoić właśnie zdobytą wiedzę wykonam przykładową pracę na tekście.

1. Zadanie pierwsze polega na wydrukowaniu tablicy ASCII w konsoli Pythona.
2. Wczytajmy przykładowy tekst z pliku (dołączam plik tekstowy plik.txt) i policzmy ile razy wystąpi każdy znak.
3. Zamieńmy dowolne słowo na inne i wyświetlmy zmieniony string.

Zadania znajdują się w plikach „zadanie 1.py”, „zadanie 2.py” i „zadane 3.py”

Źródła:

https://www.geeksforgeeks.org/python-strings/?fbclid=IwAR1UsDYLLObeQ1Clg7CS_r1nQo-Rg_VaVuxg5N_Vqbs2WZVgm0U6ldz7lLw

https://www.tutorialspoint.com/python/python_strings.htm?fbclid=IwAR3H3Z8-lfsEEamEXmOW4o4j333sFN1W6nUdZZVYfB8Ff5hFbjR_Su5nyes