Capitolul 2. Manipularea sirurilor de caractere

2.1 Manipularea sirurilor de caractere, metode

2.1.1 Manipularea sirurilor de caractere, *metode*

In Capitolul 1 am discutat despre o serie de operatiuni cu siruri de caractere: concatenarea, repetitia, secvente de evadare, despartirea prin virgula, utilizarea ghilimelelor, introducerea unui text de la tastatura, etc.

Un string este ***o secventa de caractere*** si poate include litere, cifre si caractere speciale. Intotdeauna, un string este reprezentat prin incadrarea intre ghilimele sau apostroafe.

In Python, tipurile de date, cum este si string, sunt asimilate cu obiectele. Cand o sa discutam despre programarea orientata pe obiecte o sa detaliem acest lucru. Fiecare obiect dispune de anumite ***metode*** specifice (functii) care i se pot aplica. Cu ajutorul acestor metode putem prelucra acel obiect, in cazul de fata sirurile de caractere.

Sintaxa pentru aplicarea metodelor este urmatoarea:

```
>>>"string".metoda()
```

String poate fi atat un string, cat si o variabila care contine un string. Metoda este numele metodei. Parantezele sunt specifice functiilor si metodelor si vor fi mentionate.

Pentru a vedea lista cu metodele aplicabile stringurilor vom apela functia **dir()** cu parametru un string sau o variabila care contiine un string, astfel:

```
>>>dir('string')          # string poate sa fie un string sau o variabila care contine un string
'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'
```

De regula, putem aplica o singura metoda. In unele conditii putem combina aplicarea mai multor metode simultan, aceluiasi sir.

In continuare, vom discuta cele mai uzuale metode de manipulare a stringurilor.

2.2 Variabila temporara in Python

2.2.1 Variabila temporara

Orice rezultat al unei expresii, care nu este stocat intr-o variabila, nu poate fi utilizat ulterior. Ca exceptie, rezultatul ultimei expresiiva fi disponibil intr-o ***variabila temporara.*** Acesta poate fi apelata doar in modul interactiv, in IDE si se noteaza cu "_" (underscore). Valoarea este disponibila in aceasta variabila locala pana la rescrierea ei si poate fi utilizata ca orice alta variabila.

Exemplu:

```
>>>input("Introduceti un numar: ")
Introduceti un numar: 100        # solicitarea introducerii si introducerea numarului
'100'                            # afisarea automata a numarului introdus
>>>print ( _ )                   # utilizarea variabilei
100
>>>print ( int( _ ) / 5    )     # utilizarea variabilei
20
```

2.3 Utilizarea metodelor

2.3.1 Utilizarea metodelor *upper si lower*

Metodele upper si lower sunt utilizate pentru transformarea in litere mari (majuscule) si respectiv in litere mici a intregului sir de caractere.

Sintaxa este de forma: **str.upper()** si respectiv **str.lower()**.

```
>>>var1 = 'Astazi e ziua ta';
>>>print ( var1.upper()  )
ASTAZI E ZIUA TA
>>>print ( var1.lower()  )
astazi e ziua ta
```

Atentie, variabila var1 ramane nemodificata oricate metode am aplica. Daca vrem s-o modificam va trebui sa-i atribuim o alta valoare (Ex: ***var1 = var1.upper()***). Pe de alta parte, rezultatul aplicarii metodelor poate fi asignat unei alte variabile.

2.3.2 Utilizarea metodelor *title si capitalize*

Cu ajutor metodei ***title***, prima litera din fiecare cuvant va fi transformata in litera mare.

Sintaxa este de forma: **str.title()**.

```
>>>print ( var1.title( ) )
Astazi E Ziua Ta
```

Metoda ***capitalize*** va modifica doar prima litera a textului in majuscula.

Sintaxa este de forma: **str.capitalize()**.

```
>>> print ( var1.capitalize() )
Astazi e ziua ta
```

2.3.3 Utilizarea metodelor *rjust, ljust si center*

Cu ajutorul acestor metode aliniem textul la dreapta, la stanga sau la centru. Mai mult, completam textul (padding) cu anumite caractere (default spatiu) pana ajunge la lungimea dorita, mentionata ca prim argument intre paranteze. Putem modifica acest caracter default, mentionandu-l ca al doilea argument.

Sintaxa este de forma: **str.rjust(), str.ljust(), str.center()**.

```
>>> var1.rjust(20)          # alinierea la dreapta
'    Astazi e ziua ta'
>>> var1.ljust(20)          # alinierea la stanga
'Astazi e ziua ta    '
>>> var1.center(20)         # centrarea
'  Astazi e ziua ta  '
>>> var1.center(22, '*')    # centrarea, padding cu un alt caracter decat spatiul
```

'***Astazi e ziua ta***'

2.3.4 Utilizarea metodelor *isdigit, isalpha, isalnum, isnumeric()*

In functie de tipurile de caractere continute un string putem lua anumite decizii. Metodele ***isdigit, isalpha, isalnum*** verifica daca sirul de caractere contine exclusiv caracterele mentionate (numere, litere, caractere alfanumerice). Va returna ***True*** sau ***False***. Daca sirul contine spatii sau alte caractere speciale va returna ***False***.

Sintaxa este de forma: **str.isdigit(), str.isalpha(), str.isalnum() si respectiv str.isnumeric()**

```
>>> '100'.isdigit()          # contine doar cifre
True
>>> '10x'.isdigit()          # contine si litere
False
>>> '1 0'.isdigit()          # contine si spatii
False
>>> var1.isalpha()           # contine si spatii
False
>>> 'Metafizica'.isalpha()
True
>>> '10x'.isalnum()
True
>>> '10,x'.isalnum()         # contine si alte caractere
False
>>> '333'.isnumeric()        # contine doar cifre
True
```

2.3.5 Utilizarea metodelor *isspace, isupper, islower*

Metodele ***isspace, isupper, islower*** testeaza daca sirul de caractere contine doar spatii si respectiv doar litere mari sau mici. Va returna ***True*** sau ***False***.

Sintaxa este de forma: **str.isspace(), str.isupper() si str.islower()**.

```
>>> '    '.isspace()
True
>>> var1.isupper()
False
>>> 'MAJUSCULE'. isupper()
True
>>> 'abc'.islower()
True
```

2.3.6 Utilizarea metodei *replace*

Cu ajutorul metodei ***replace*** inlocuim un subsir_vechi cu un subsir_nou. Daca vom avea doar doi parametri inlocuirea va fi efectuata pentru toate subsirurile vechi gasite. Al treilea parametru este optional si va duce la inlocuirea a doar atatea aparitii cate mentioneaza acest parametru (primele aparitii).

Sintaxa este de forma: **str.replace(subsir_vechi, subsir_nou[, numar_de_aparitii de_inlocuit]).**

```
>>> 'In fiecare zi merg in vacanta. In fiecare zi este Craciunul. In fiecare zi ma plimb'.replace('zi', 'an')        # inlocuim toate aparitiile
'In fiecare an merg in vacanta. In fiecare an este Craciunul. In fiecare an ma plimb'
>>> 'In fiecare zi merg in vacanta. In fiecare zi este Craciunul. In fiecare zi ma plimb'.replace('zi', 'an', 2)      # inlocuim doar primele 2 aparitii
'In fiecare an merg in vacanta. In fiecare an este Craciunul. In fiecare zi ma plimb'
```

Putem incerca si aplicarea unor metode combinate:

>>> **var1.replace(' ', '').isalpha()** # eliminam spatiile si testam daca are doar litere

True

2.2.7 Utilizarea metodei *join*

Cu ajutorul metodei ***join*** putem face doua feluri de concatenari: includerea unui sir de n-1 ori intre caracterele date (n este numarul de caractere din sirul primit ca argumet) si respectiv despartirea unor siruri primite ca lista de argumente, avand drept separator caracterul dat.

Sintaxa este de forma: **str.join('sir')** si respectiv **'caracter'.join(['sir 1', 'sir 2', '...', 'sir n'])**.

>>> **print (var1.join** (**'*-*-*'**) **)** *# scrie sirul intre caracterele date*
* Astazi e ziua ta-Astazi e ziua ta*Astazi e ziua ta-Astazi e ziua ta*

>>> **print (** ' '**.join** (['**Astazi**', '**va**', '**fi**', '**soare**']) **)** *# scrie elementele listei despartite de sirul dat*

Astazi va fi soare

2.3.8 Utilizarea metodei *format* – notiuni de baza

Exista situatii in care dorim sa facem diferite combinatii de date de intrare pentru a obtine anumite rezultate dorite. Formatarea stringurilor poate fi foarte utila in astfel de cazuri.

Pentru a face formatarea avem nevoie de cateva notiuni elementare de indexare in Python. Intr-un sir de caractere putem identifica in mod unic fiecare caracter al unui sir, prin intermediul unui numar asociat. Acest lucru este foarte util atunci cand cautam in interiorul sirului diferite caractere sau subsiruri.

Exemplu:

0	1	2	3	4	5
c	u	v	a	n	t
-6	-5	-4	-3	-2	-1

Putem obtine un anumit caracter din sir cu ajutorul constructiei:

```
>>> 'cuvant'[3]
'a'
```

In primul rand, pozitiile unde dorim sa includem textul formatat, vor fi indicate prin acolade {0} care vor contine si un numar incepand cu zero. In versiunea 2.x numarul poate sa lipseasca, fiind identificate prin pozitia in care apar acoladele in cadrul textului.

Sintaxa este de forma **str.format(expresia0, expresia1, ..., expresian)**:

```
>>> print "Acest produs costa {0} lei si se afla pe raftul {1}". format('10', '007')
Acest produs costa 10 lei si se afla pe raftul 007
```

Prin formatare vom indica sursa datelor care vor fi incluse in locul {0} si {1}.

{0} si {1} pot fi asimilate aici cu variabile locale, care vor fi date de parametrii metodei format.

Este important de stiut ca numarul acestor "variabile" trebuie sa fie egal cu numarul parametrilor metodei format. De asemenea, tipurile de date trebuie sa fie corespunzatoare.

Parametrii format se indexeaza, incepnd de la 0(zero) primul parametru si care va da valoarea pentru {0} si pana la n-1(n fiind numarul total de parametri).

Putem specifica formatul datelor utilizand metacaractere (cu semnificatii specifice):

- **s** - pentru string. String este formatul default si poate sa lipseasca;
- **b** - pentru binary, numere in baza 2;
- **o** - pentru octal, numere in baza 8;
- **x** - pentru hexazecimal, numere in baza 16, litere mici;
- **X** - pentru hexazecimal, numere in baza 16, litere mari;
- **d** - pentru decimal, numere in baza 10;
- **n** - similar cu decimal cu separatoare grupe de cifre;
- **e** - pentru numere in notatie stiintifica, default precizie de 6 caractere;
- **f** - pentru numere rationale, default 6 zecimale. Se poate seta numarul dorit de zecimale;
- **%** - multiplica numarul cu 100 si-l returneaza ca numar rational.

```
>>> print ( '{0}, {1}, {2}'.format('x', 'y', 'z') )
                                        # x pentru {0}, y pentru {1}, z pentru {2}
x, y, z
>>> print ( '{2}, {1}, {0}'.format('x', 'y', 'z') )
z, y, x
>>> print ( '{2}, {1}, {0}'.format(*'xyz')  )      # face split la sir
```

```
z, y, x
>>> print ( '{0}{1}{1}'.format('Tra', ', la') )    # putem repeta sirul
Tra, la, la
```

Putem accesa "variabilele" dupa numele acestora, avand o mapare exacta dupa denumirea acestora:

```
>>> print ( 'Produsul: {prod}, cantitatea: {cant}, pretul: {pret}'

.format(prod = 'cirese', cant = 100, pret = 5)

Produsul: cirese, cantitate: 100, pret: 5
```

Putem alininia un text (similar cu rjust, ljust, center) completandu-l cu spatii ca sa ajunga la dimensiunea dorita. In plus, in locul spatiilor putem pune alte caractere:

```
>>> print (  '{:<20}'.format('stanga')  )  # il face de 20 de caractere aliniat la stanga
stanga
>>> print (  '{:>20}'.format('dreapta')) # il face de 20 de caractere aliniat la dreapta
          dreapta
>>> print ( '{:^19}'.format('centrat') ) # il face de 19 de caractere centrat
     centrat
>>> print (  '{:*^19}'.format('centrat') )
                                          # centrare si inlocuirea spatiilor cu caracterul *
******centrat******
```

Putem formata textul de tip float, implicit cu 6 zecimale. Numarul de zecimale poate fi diferit, stabilit de programator.

```
>>> print ( '{: f}'.format(15) )
15.000000
>>> print (  '{: .2f}'.format(15) )      # " .2 " reduce numarul de zecimale la 2
15.00
```

Putem formata numere in diferite baze:

Cat reprezinta numele primite ca parameetri de format in bazele de numeratie corespunzatoare?

```
>>> print (  "int: {0:d};  hex: {1:x};  oct: {2:o};  bin: {3:b}"

.format(100, 255, 63, 15) )

int: 100;  hex: ff;  oct: 77;  bin: 1111
```

Sau reprezentarea specifica acestor baze de numeratie:

```
>>> print ( "int: {0:d};  hex: {1:#x};  oct: {2:#o};  bin: {3:#b}"
.format(100, 127,15, 7) )
int: 100;  hex: 0x7f;  oct: 0o17;  bin: 0b111
```

Putem delimita grupe de cate 3 cifre pentru partea intreaga a unui numar:

```
>>> print ( "{:,}".format(35735735735) )
35,735,735,735
```

Putem calcula valori procentuale astfel:

```
>>> raspunsuri_corecte = 17
>>> intrebari_total = 21
>>> print ( 'Nota : {:.2%}'.format( raspunsuri_corecte / intrebari_total ) )
Nota: 80.95%
```

Putem completa spatii sau zerouri inaintea unui numar intreg:

```
>>> print ( '{:5d}'.format(15) )      # completarea cu spatii, in stanga, pana la
                                      lungimea de 5 caractere
  15
>>> print ( '{:05d}'.format(15) )     # completarea cu zerouri, in stanga, pana la
                                      lungimea de 5 caractere
00015
```

Mai exista o metoda de formatare, utilizand metacaracterul "%" in locul acoladelor.

Sintaxa este de forma: **'Nota de trecere este %.2f. Nota ta este %.2f.'** % (75, 75.01).

2.3.9 Utilizarea metodei *find*

Metoda ***find*** este utilizata pentru a returna pozitia unui subsir in sirul dat (mai precis pozitia primului caracter al subsirului, pozitie data de indexul acestui caracter).

Sintaxa este de forma: **str.find(subsir[, poz_inceput, poz_terminat]**. Pozitia se determina in functie de indexul unui string. Primul parametru este obligatoriu, fiind subsirul cautat, daca ceilalti doi parametri lipsesc va fi returnata pozitia ***primei aparitii*** a subsirului. In cazul existentei parametrilor optionali va cauta intre prima limita si sfarsitul sirului (daca cel de-al treilea paramentru lipseste) sau intre cele doua limite.

```
>>> print ( var1.find ( 'ta' ) )          # cauta prima aparitie a subsirului 'ta'
2
>>> print ( var1.find ( 'ta' ), 7 )       # cauta prima aparitie a subsirului 'ta' dupa
                                          caracterul 7
14
```

2.3.10 Utilizarea metodei *split*

Metoda ***split*** imparte un text in functie de un delimitator (subsir). Daca acesta nu este specificat, implicit este spatiul.

Sintaxa este de forma: **str.split([subsir])**. Va fi returnata o lista cu toate sirurile rezultate in urma splitului.

```
>>> print ( var1.split () )
['Astazi', 'e', 'ziua', 'ta']
>>> print ( var1.split ( 'e ' ) )
['Astazi ', 'ziua ta']
```

2.3.11 Utilizarea metodei *count*

Cu ajutorul metodei ***count*** numaram de cate ori apare un subsir in sirul dat.

Sintaxa este de forma: **str.count(subsir)**.

```
>>> print ( var1.count ( 'a' ) )
3
```

2.3.12 Utilizarea metodelor *startswith si endswith*

Cu ajutorul metodelor ***startswith si endswith*** testam daca un sir incepe si respectiv se termina cu subsirul primit ca argument.

Sintaxa este de forma: **str.startswith(subsir) si respectiv str.endswith(subsir)**.

```
>>> print ( var1.startswith ( 'Astazi' ) )
True
>>> print ( var1.endswith( 'mea' ) )
False
```

2.4 Recapitulare

2.4.1 Recapitulare

Principalele metode de manipulare a sirurilor de caractere sunt urmatoarele:

upper si lower

title si capitalize

rjust, ljust si center

isdigit, isalpha, isalnum, isnumeric

isspace, isupper, islower

replace

join

format

find

split

count

startswith si endswith

Metodele se aplica prin constructia:

'sir de caractere'.nume_metoda([parametri])

Atentie, aceste metode sunt specifice doar sirurilor de caractere si nu se pot aplica altor tipuri de date (numerelor).

Cu ajutorul functiei dir, care primeste ca parametru un sir sau o variabila care contine un sir, putem vizualiza metodele aplicabile unui sir de caractere.

Un sir de caractere este indexat. Fiecarui caracter ii este atribuit un numar, de la stanga catre dreapta, incepand de la zero - primul caracter - si pana la n minus 1, n fiind numarul de caractere.

Variabila temporara se noteaza cu "_", stocheaza temporar valoarea unei expresii nestocate intr-o variabila, doar pana cand o alta situatie asemanatoare se intampla.