Факултет за електротехника и
информарички технологии - Скопје

Вовед во PYTHON

http://docs.python.org/tut/tut.html

Билјана Милева - Бошкоска

Скопје, 08.10.2007

```
>>> # ova e prv komentar
SPAM = 1          # ova e prv komentar
                  # ... I ova komentar!
STRING = "# Ova ne e komentar "
```

```
>>> print STRING;
# Ova ne e komentar.
>>> print SPAM;
1
```

```
>>> hello = "Ova e prilichno dolg string koj sodrzhi\n\
Nekolku linii na tekst isto kako vo programskiot jazik C.\n\
    Zabelezhete deka praznoto mesto na pochetokot od linijata e\
 znachajno."
```

```
>>> print hello;
Ova e prilichno dolg string koj sodrzhi
Nekolku linii na tekst isto kako vo programskiot
jazik C.
    Zabelezhete deka praznoto mesto na
pochetokot od linijata e znachajno.
```

```
>> word = 'Pomosh' + 'A'
>>> word;
 ' PomoshA '
>>> '<' + word*5 + '>'
'<PomoshAPomoshAPomoshAPomoshAPomoshA>'
```

Стринговите може да се индексираат; како во C, првиот карактер на стригот има индекс 0. Може да се испишуваат и само делови од стрингови

```
>>> word='masinska inteligencija';
>>> word[4];
'n'
>>>


>>> word[0:2];
'ma'
>>> word[0:2]; #gi dava prvite dva karakteri
'ma'
>>> word[2:]; # gi dava site karakteri osven prvite dva
'sinska inteligencija‘ (znachi pochnuva od tretiot karatker)
```

10.05.2006

За разлика од C стринговите, во Python стринговите не може да се менуваат.
Доделување на карактер на некоја позивија во струингот предизвикува грешка:

```
>>> word[0]='x';

Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    word[0]='x';
TypeError: 'str' object does not support item assignment
>>>
```

Но сепак, креирање на нов стринг преку комбинирање на содржината е лесно и ефикасно:

```
>>> 'x'+word[1:];
'xasinska inteligencija‘
```

Што ќе се добие со следната команда:

```
>>> word[:2] + word[2:]
```

'masinska inteligencija‘

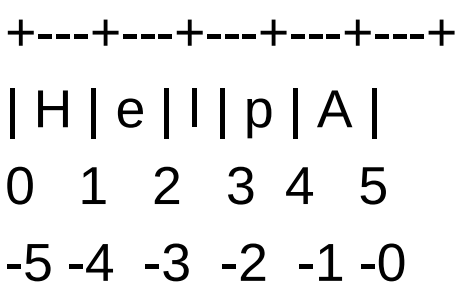
Лошите парчиња на индекси се справуваат одлично: индексот кој е предолг се заменува со должината на стрингот

```
>>> word[1:100];
'asinska inteligencija'
```

Индексите може да бидат и негативни броеви, при што се започнува со броење наназад, од десно кон лево. На пример:

```
>>>word[-1] # Posledniot karakter
>>> word[-1]     # Posledniot karakter
'a'
>>> word[-2]
'j'
>>> word[-2:]
'ja'
>>> word[:-2]
'masinska inteligenci'
>>>
```

Најлесен начин да се запамти како работат деловите е да се гледаат индексите како да покажуваат меѓу карактерите, при што левиот ќош на првиот карактер е означен со 0. Потоа десниот ќош на последниот карактер на стрингот со *n* карактери има индекс *n*, на пример:



Листи

Во Python има неколку *мешани* типови на податоци, кои се користат за групирање на други вредности. Најсестрани се листите, кои можат да се запишат како листа на вредниости одвоени со запирка помеѓу квадратни загради. Членовите на листите не секогаш се од ист тип.

```
>>> a = ['spam', 'jajce', 100, 1234];
>>> a;
['spam', 'jajce', 100, 1234]
>>> a[0];
'spam'

>>> a[-3];
'jajce'

>>> a[1:-1]
['jajce', 100]

>>> a
['spam', 'jajce', 100, 1234]

>>> a[:2]+[' slanina ', 2*2]
['spam', 'jajce', ‘slanina', 4]
>>>
```

10.05.2006

За разлика од стринговите кои се непроменливи, можно е да се сменат одредени елементи на листите

```
>>> a;
['spam', 'jajce', 100, 1234]
>>> a[2] = a[2] + 23;
 >>> a;
['spam', 'jajce', 123, 1234]
```

Доделување на делови од листи е исто така можно, и ова може дури да ја смени големината на листите или пак сосема да ја избрише:

```
>>> # Menuvanje na nekoi elementi na listata:
... a[0:2] = [1, 12];

>>> a;
[1, 12, 123, 1234]
>>> # Otstranuvanje na nekoi elementi na listata :
... a[0:2] = []
>>> a
[123, 1234]
>>> # Vnes na elementi na listata :
... a[1:1] = ['shef', 'xyzzy']
>>> a
[123, ' shef, 'xyzzy', 1234]
>>> # Insertiranje na  (kopija) namata niza vo sebe na pochetokot
>>> a[:0] = a
>>> a
[123, 'bletch', 'xyzzy', 1234, 123, 'bletch', 'xyzzy', 1234]
>>> # Chistenje na listata: da se zamenat site elementi od listata so prazno mesto, t.e. da se dobie
      prazna lista
>>> a[:] = []
```

Вградената функција len() исто така се однесува на листите, и ја дава нивната должина:

```
>>> len(a)
8
```

Секако дека може да го користиме Python за многу покомплицирани задачи од собирање на два и два. На пример може да напишеме за почеток една *Fibonacci* низа:

```
>>> # Fibonacci niza: ... # sumata na dvata prethodni elementi go definira sledniot element ...
>>> a, b = 0, 1
>>> a,b=0,1;
>>> while b<10:
        print b
    a,b=b,a+b;

1
1
2
3
5
8
>>>
```

OVOJ SLAJD E SAMO ZA MENE!!!`

Овој пример воведува неколку нови особини.

Првата линија содржи мултипно доделување: променливите a и b симултано добиваат нови вредности 0 и 1. Во последната линија, ова е повторно искористено, демонстрирајќи дека изразот на десната страна се извршува прв, пред да се изврши некој од другите зададени задачи на програмот. (The right-hand side expressions are evaluated from the left to the right.)

Циклусот while се извршува се додека е исполнет условот (овде: b < 10). Во Python, како и во C, секоја не-нулта интеџер вредност е true; додека само нулата е false. Улсовот може да биоде и стринг или листа, или било која секвенца; се што има должина различна од нула е true, празната секвенца е false. (The test used in the example is a simple comparison. The standard comparison operators are written the same as in C: < (less than), > (greater than), == (equal to), <= (less than or equal to), >= (greater than or equal to) and != (not equal to)).

Телото на циклусот е повлечено . Повлекувањето е начин во Python за групирање на искази. Python се уште не овозможува интелигентен начин за внезс на командите, па мора да се користи таб или space за секоја повлечена линија. (In practice you will prepare more complicated input for Python with a text editor; most text editors have an auto-indent facility. When a compound statement is entered interactively, it must be followed by a blank line to indicate completion (since the parser cannot guess when you have typed the last line). Note that each line within a basic block must be indented by the same amount.)

Исказот print врши испишување на бараниот израз. It differs from just writing the expression you want to write in the way it handles multiple expressions and strings. Strings are printed without quotes, and a space is inserted between items, so you can format things nicely, like this:

```
>>> i = 256*256
>>> print 'The value of i is', i
The value of i is 65536
```

10.05.2006

Исказот if

```
>>>x=0
>>> if x < 0:
    print 'Negativen';
elif x==0:
    print 'Nula';
elif x == 1:
    print 'Eden';
else:
    print 'Ne nulev';

Zero-
```

Може да има нула или повеќе elif делови, додека делот else е опционален. Клучниот збор `elif' е кратенка за `else if'. Секвенцата if ... elif ... elif ... Е замена за switch или case исказите кои постојат во останатите јазици.

Наредбата for

Наредбата for во Python се разликува малку од онаа нејзина функција во C или Pascal. Наместо секогаш да врши итерација помеѓу аритметичка прогресија од броеви (како во Pascal), или давајќи му на корисникот можност да дефинира и чекор на итерација и услов за запирање (како во C), Pythono-виот for исказ врши итерација над елементите во една секвенца (листа или стринг), по ред како што тие се појавуваат во секвенцата. На пример :

```
>>> a=['1','macka','string']
>>> for x in a:
    print x, len(x);

1 1
macka 5
string 6
```

Функцијата range()

Ако имате потреба за итерација помеѓу некоја секвенца на броеви, вградената функција range() е многу згодна. Таа генерира листи кои содржат аритметички прогресии:

```
>>> range(10);
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5,10);
[5, 6, 7, 8, 9]
>>> range(5,10,2);
[5, 7, 9]
>>>  a = [‘Meri', ‘ima', ‘edno', ‘malo', ‘mache']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Meri
1 ima
2 edno
3 malo
4 mache
```

Исказите break и continue

```
>>> for n in range(2,10):
        for x in range (2,n):
                if n % x == 0:
                                print n, 'e ednakov na', x, '*', n/x
                                break
                else:
                                print n, 'e primaren broj'
```

```
3 e primaren broj
4 e ednakov na 2 * 2
5 e primaren broj
5 e primaren broj
5 e primaren broj
6 e ednakov na 2 * 3
7 e primaren broj
7 e primaren broj
7 e primaren broj
7 e primaren broj
7 e primaren broj
8 e ednakov na 2 * 4
9 e primaren broj
9 e ednakov na 3 * 3
```

Исказот pass

Исказот pass не прави ништо. Може да се користи кога исказот е потребен синтаксички , додека пак програмата не прави ништо. На пример:

```
>>> while True:
    pass #chekaj za interupt od tastatura
```

Дефинирање на функции

```
>>> def fib(n):
    """Ispechati ja Fibonachievata niza do n"""
    a, b = 0,1
    while b < n:
        print b,
        a, b = b, a+b


>>> fib(2000);
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

def –се воведува дефиниција на функција. По неа следи името на функцијата и нејзините параметри во мали загради. Исказите кои го сочинуваат телото на функцијата започнуваат во следниот ред, и мора да се вовлечени со TAB. Првиот исказ во функцијата може да биде string; така наречен *docstring*.

Извршувањето на функцијата воведува нова симбол табела која ја користат локалните варијабли на функцијата. Поточно, доделувањето на сите варијабли на функцијата се чуваат во локалната симбол табела; додека пак референците кон променливите прво се бараат во лакалната симбол табела, потоа во глобалната симбол табела, а потоа во табелата на вградени имиња (built-in names). Значи на глобалните варијабли не може директно да им се додели вредност во рамките на функцијата (освен во глобален исказ), иако кон нив може да се референцираме.

Моменталните параметри (аргументи) при повик на функцијата се внесуваат во локалната симбол табела на повиканата функција; значи аргументите се предаваат по вредност (using *call by value)* (каде вредноста е секогаш референца кон објект, а не вредноста на објектот). Кога функцијата повикува друга функција, се креира нова симбол табела, за новиот повик.

Дефинирањето на функција воведува име на функцијата во моменталната симбол табела. Вредноста на името има тип кој може да го препознае интерпретерот како кориснички дефинирана функција. Оваа вредност потоа може да се додели на друго име кое потоа може да се користи како функција. Ова служи како општ механизам за доделување на нови имиња (renaming mechanism).

Исто така можно е да се дефинира функција во променлив број на аргументи

Default Argument Values

Најкорисна форма е да се специфицира на номинална вредност за еден или повеќе аргументи. Ова овозможува да се повика функцијата со помалку аргументи од колку што е специфицирано. На пример:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return True
        if ok in ('n', 'no', 'nop', 'nope'): return False
        retries = retries - 1
        if retries < 0: raise IOError, 'refusenik user'
        print complaint
```

Оваа функција може да се повика вака: ask_ok('Do you really want to quit?') or like this: ask_ok('OK to overwrite the file?', 2).

Овој пример ја воведува и функцијата **in**. Таа проиверува дали определена секвенца содржи определена вредност.

The default values are evaluated at the point of function definition in the *defining* scope, so that

```
i = 5
def f(arg=i):
    print arg i = 6
f()
```

will print 5.

10.05.2006

Функциите може да се повикаат со користење на аргументи во форма "*клучен збор* = *вредност*". На пример, следната функција:

```
def parrot(voltage, state='raspolozen', action='leta', type='Toa e norveski plav'):
    print "—ovoj papagal ne saka da", action,
    print "ako stavis", voltage, "volti niz nego."
    print "—Ubavi perja, ", type
    print "—Toj e", state, "!"
```

може да се повикаат на следниот начин:

```
parrot(1000)
parrot(action = 'kolva', voltage = 1000000)
parrot('a thousand', state = 'brka peprutki')
parrot('a million', 'ubav', 'skoka')
```

но следниот тип на повикување не е валиден:

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument following keyword
parrot(110, voltage=220)     # duplicate value for argument
parrot(actor='John Cleese')  # unknown keyword
```

Во општ случај, листата со аргументи мора да ги има сите зададени аргументи на функцијата, каде клучните зботрови мора да се изберат од формалните параметарски имиња. Не е важно дали формалниот параметар има номинална вредност или не. Ниту еден аргумент не смее да добие вредност повеќе од еднаш -- формалните имиња на параметрите кои одговараат ма позиционите аргументине може да се користат како клучни зборови во ист повик. Еве еден пример за оваа рестрикција:

```
>>> def function(a):...
pass...
>>> function(0, a=0)
Traceback (most recent call last):
File "<stdin>", line 1, in ?
TypeError: function() got multiple values for keyword argument 'a'
```

Конечно, најретко користена опција е да се специфицира да функцијата се повикува со произволен број н парметри. Овие аргумети се ставаат во n - торки. Пред променливата за број на аргументи, може да се појави нула или други номинални аргументи.

```
def fprintf(file, format, *args):
    file.write(format % args)
```

Еве ги сите методи кои се користат за објектите - листи:

append(*x*)Додади член на крајот од листата; еквивалентно на a[len(a):] = [*x*].

extend(*L*)Продолжи ја листата со додавање на членовите во дадената листа; еквивалентно на a[len(a):] = *L*.

insert(*i, x*)Додај член на дадена позиција. Првиот аргумент е индексот на елементот пред кој треба да се додаде нов член, па a.insert(0, *x*) додава на почетокот на силтата, a.insert(len(a), *x*) е еквивалентно на a.append(*x*).

remove(*x*)Го брише првиот елемент од листата чија вредност е *x*. Дава грешка ако не постои таков елемент.

pop([*i*])Отстранување на елемент од дадена позиција во листата и и негово враќање (на екран на пример). Ако не е специфициран ниту еден индекс, a.pop() го отстранува и враќа последнит елемент од листата. (Квадратните загради означуваат дека аргументот е опционен.

index(*x*)Го враќа индексот на елементот во листата на првиот елемент чија вреднот е *x*. Враќа грешка ако не постои таков елемент.

count(*x*)Враќа колку пати се појавува елементот *x* во листата.

sort()Сортирање на листата.

reverse()Ги превртува елементите на листата.

Пример кој користи најголем дел од излистаните методи:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Користење на листите како стекови

Методите на листи овозможуваат лесно користење на истите како стекови, каде последниот додаден елемент е прв кои излегува (``last-in, first-out''). За додавање на елемент на врвот од стекот, се користи append(). За вадење на елемент од врвот на стекот се користи pop()без експлицитен индекс. На пример:

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack[3, 4]
```

Листите може да се користат и како редови, каде првиот додаден елемент е прв изваден од листата (``first-in, first-out''). За да се додаде елемент на крајот од редот се користи append(). За вадење на елемент од врвот на стекот се користи pop() со индекс 0:

```
>>> queue = ["Eric", "John", "Michael"]
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.pop(0)
'Eric‘
>>> queue.pop(0)
'John‘
>>> queue['Michael', 'Terry', 'Graham']
```

Има три вградени функции кои се многу корисни кога се користат со листи: filter(), map(), and reduce().

"filter(*function*, *sequence*)" враќа секвенца која се состои од оние елементи од секвенцата за кои *function*(*item*) е точна. Ако *sequence* е стринг или торка, резултатот ќе биде од ист тип; инаку, секогаш друг пат е листа. На пример:

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
```

"map(*function*, *sequence*)" ја повикува *function*(*item*) за секој елемент на секвенцата и враќа листа од повратни вредности. На пример,:

```
>>> def cube(x): return x*x*x...
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

"reduce(*function*, *sequence*)" враќа единечна вредност конструирана со повикување на бинарната функција *function* на првите два елементи од секвенцата, потоа, на резултатот и на следниот елемент, итн.На пример, да се пресмета сумата на броевите од 1 до 10:

```
>>> def add(x,y): return x+y
>>> reduce(add, range(1, 11))
55
```

Ако во секвенцата има само еден број, се враќа тој број, инаку, ако секвенцата е празна се фрла exception.