Práce s kontejnery

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMI/ZPP1 Základy programování v Pythonu 1

Test přítomnosti prvku

Příklad

- Napíšeme funkci, která rozhodne, zda je hodnota prvkem kontejneru.

```
def je_v (prvek,kontejner):
    vysledek = False
    for i in kontejner:
        if i == prvek :
            vysledek = True
            break

    return vysledek
```

- Vyzkoušíme:

```
k1="Python"
print(je_v ("y",k1))
k2=[1,8,5,6,7]
print(je_v (6,k2))
k3={1,8,5,6,7}
print(je_v (6,k3))
k4={"Jablko": "Apple", "Knoflík": "Button", "Myš": "Mouse"}
print(je_v ("Jablko",k4))
print(je_v ("Button",k4.values()))
k5=(1,8,5,6,7)
print(je_v (6,k5))
```

Příklad

- Lze prvky ve funkci iterovat i pomocí indexu?

```
def je_v (prvek,kontejner):
    vysledek = False
    i=0
    pocet=len(kontejner)
    while vysledek==False and i<pocet:
        if kontejner[i] == prvek:
            vysledek = True
        i+=1
    return vysledek
```

- Vyzkoušíme:

```
k3={1,8,5,6,7}
print(je_v (6,k3))
k4={"Jablko": "Apple", "Knoflík": "Button", "Myš": "Mouse"}
print(je_v ("Jablko",k4))
print(je_v ("Button",k4.values()))
```

Použití operátorů in a not in

- Použití:

```
prvek in kontejner
prvek not in kontejner
```

```
k1="Python"
print("y" in k1)
k2=[1,8,5,6,7]
print(6 not in k2)
k3={1,8,5,6,7}
print(6 in k3)
k4={"Jablko": "Apple", "Knoflík": "Button", "Myš": "Mouse"}
print("Jablko" in k4)
print("Button" in k4.values())
k5=(1,8,5,6,7)
print((6 not in k5))
```

Řezy

Příklad

- Vezmeme-li seznam, například [1, 2, 3], můžeme indexovat mezery mezi prvky.
- Mezera před prvním prvkem bude mít index nula, mezera mezi prvním a druhým prvkem bude mít index jedna, a tak dále. Délka seznamu bude indexem mezery za posledním prvkem.
- Podseznam l2 seznamu l1 můžeme určit dvojicí indexu mezer i a j seznamu l1. Podseznam l2 budou tvořit právě ty prvky mezi mezerami i a j.
- Napíšeme funkci rez, která by brala seznam a dva indexy mezer a vracela by jimi urcený podseznam:

```
>>> rez ([1 , 2, 3], 1, 3)
[2, 3]
>>> rez ([1 , 2, 3], 0, 2)
[1, 2]
>>> rez ([1 , 2, 3], 0, 0)
[]
```

Příklad

```
def rez (kontejner,od,do):
    vysledek = []
    for i in range(do-od):
        prvek = kontejner [i + od ]
        vysledek += [prvek]
    return vysledek

k1="Python"
print(rez(k1,0,2))
k2=[1,8,5,6,7]
print(rez(k2,0,2))
k3={1,8,5,6,7}
print(rez(k3,0,2))
k4={"Jablko": "Apple", "Knoflík": "Button", "Myš": "Mouse"}
print(rez(k4,0,2))
print(rez(k4.values(),0,2))
k5=(1,8,5,6,7)
print(rez(k5,0,2))
```

Řezy

- Podkontejner kontejneru lze získat jednodušeji pomocí takzvaných řezů.
- Pokud v, i a j jsou výrazy, pak

```
v[i:j]
```

- je výraz řezu (anglicky slice).
- Hodnota výrazu v je kontejner.
- Hodnoty výrazu i a j jsou indexy mezer seznamu v.
- Řezy lze používat pouze na sekvence - posloupnosti.

```
k1="Python"
print(k1[0:2])
k2=[1,8,5,6,7]
print(k2[0:2])
k3={1,8,5,6,7}
print(k3[0:2])
k4={"Jablko": "Apple", "Knoflík": "Button", "Myš": "Mouse"}
print(k4[0:2])
k5=(1,8,5,6,7)
print(k5[0:2])
```

Volba kroku řezu

- V řezech sekvence můžeme zvolit délku kroku.
- Pokud s; i; j; k jsou výrazy, pak

```
s[i:j:k]
```

- je řez s volbou kroku. Hodnotou s je sekvence, hodnoty i a j jsou indexy mezer sekvence s a hodnota k je přirozené číslo.
- Hodnota řezu s volbou kroku je sekvence (stejného typu jako s) každé k-té položky od indexu i po index j.
- Výchozí hodnotou výrazu k je jedna. Například:

```
>>> l = list ( range (10))
>>> l [0:10:2]
[0, 2, 4, 6, 8]
>>> l [1:10:2]
[1, 3, 5, 7, 9]
>>> l [::3]
[0, 3, 6, 9]
>>> l [::]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

„Řezy“ ve slovníku

- Napíšeme funkci pro vytváření „řezů“ ve slovníku:

```
def rez(slovnik,klic_start,klic_konec):
    vysledek = {}
    pridavat=False
    for i in slovnik:
        if i==klic_start:
            pridavat=True
        if i==klic_konec:
            pridavat=False
        if pridavat==True:
            vysledek[i]=slovnik[i]
    return vysledek

ascii={chr(x):x for x in range(128)}
velka_pismena=rez(ascii,"A","[")
print(velka_pismena)
```

- Má to smysl?

Změna řezu

- Na levé straně od příkazu = můžeme použít i řez.
- Například nahrazení druhého a třetího prvku lze provést následovné:

```
l = [1, 2, 3, 4, 5]
l[1:3] = [6,7]
print(l)
>>>[1, 6, 7, 4, 5]
```

- Délka řezu se nemusí rovna délce seznamu, který chceme místo řezu vložit:

```
l = [1, 2, 3, 4, 5]
l [1:3] = [6, 7, 8]
print(l)
>>>[1, 6, 7, 8, 4, 5]
```

Změna řezu

- Dokonce můžeme i řez ze seznamu smazat:

```
l = [1, 2, 3, 4, 5]
l[1:3] = []
print(l)
>>>[1, 4, 5]
```

- Pokud použijeme řez s krokem, musí se rovnat délka řezu délce náhrady:

```
l = [0, 1, 2, 3, 4, 5, 6]
l [::2] = [7, 8, 9, 10]
print(l)
>>>[7, 1, 8, 3, 9, 5, 10]
l[::2] = [7, 8, 9, 10, 11]
>>>ValueError : attempt to assign sequence of size 5 to extended slice of
size 4
```

Změna řezu

- Na pravé straně od příkazu přirazení muže být libovolná sekvence:

```
l = list(range (10))
l [2:4] = range(3)
print(l)
>>>[0, 1, 0, 1, 2, 4, 5, 6, 7, 8, 9]
l [4:4] = 'abc '
print(l)
>>>[0, 1, 0, 1, 'a', 'b', 'c', 2, 4, 5, 6, 7, 8, 9]
```

- Pokud se dolní i horní mez řezu rovnají, dojde k vložení sekvence do seznamu.

Záporný krok u řezu

- V řezech můžeme použít i záporný krok. Zde se však indexy mezer posunují o jedna doprava. Proto:

```
l = list ( range (10))
print(l)
>>>[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(l [10:0: -1])
>>>[9, 8, 7, 6, 5, 4, 3, 2, 1]
print(l [10:0: -2])
>>>[9, 7, 5, 3, 1]
print(l[3:0: -1])
>>>[3, 2, 1]
l[10:0: -2] = range (5)
print(l)
>>>[0, 4, 2, 3, 4, 2, 6, 1, 8, 0]
```

Záporný krok

- Pokud chceme do řezu se záporným krokem uvést i první prvek, musíme druhou mez vynechat:

```
l = list ( range (10))
print(l [5:: -1])
>>>[5, 4, 3, 2, 1, 0]
print(l[:: -1])
>>>[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Příklad

- Chceme získat řetězec pozpátku:

```
def obracene(text):
    vysledek=""
    for i in text:
        vysledek=i+vysledek
    return výsledek
```

- Stejný úkol provedeme za použití řezů:

```
def obracene(text):
    return text[::-1]
```

Nahrazení prvku v kontejneru

Příklad

- Chceme napsat funkci, která v seznamu nahradí jeden prvek druhým.

```
def nahrad(seznam,co, cim):
    for i in range(len(seznam)):
        if seznam[i]==co:
            seznam[i]=cim


k2=[1,8,5,6,7]
nahrad(k2,10,4)
print(k2)
```

Příklad

- Chceme napsat funkci, která v množině nahradí jeden prvek druhým.

```
def nahrad(mnozina,co, cim):
    if co in mnozina:
        mnozina.remove(co)
        mnozina.add(cim)

k3={1,8,5,6,7}
nahrad(k3,8,4)
print(k3)
```

Příklad

- Chceme napsat funkci, která v slovníku nahradí jeden prvek druhým.

```
def nahrad(slovnik,co, cim):
    if list(co.keys())[0] in slovnik and
slovnik[list(co.keys())[0]]==list(co.values())[0]:
     del slovnik[list(co.keys())[0]]
     slovnik[list(cim.keys())[0]]=list(cim.values())[0]

k4={"Jablko": "Apple", "Knoflík": "Button", "Myš": "Mouse"}
nahrad(k4,{"Jablko": "Apple"},{"Papír": "Paper"})
print(k4)
```

Příklad

- U řetězců nemůžeme nahrazovat, ale můžeme vytvořit nový řetězec, kde bude znak nahrazen jiným.

```
def nahrad(retezec,co, cim):
    vysledek=""
    for i in retezec:
        if i==co:
            vysledek+=cim
        else:
            vysledek+=i
    return vysledek


k5="abeceda"
k6=nahrad(k5,"a","f")
print(k6)
```

Příklad

- Nahradíme podřetězec řetězcem:

```
def nahrad(retezec,co, cim):
    vysledek=""
    i=0
    while i< len(retezec):
        if retezec[i:i+len(co)]==co:
            vysledek+=cim
            i+=len(co)
        else:
            vysledek+=retezec[i]
            i+=1
    return vysledek


k5="ababcedab"
k6=nahrad(k5,"ab","Python")
print(k6)
```

Příklad

- Totéž umí metoda replace:

```
k5="ababcedab"
k6=k5.replace("ab","Python")
print(k6)
```

Přidávání prvků do kontejnerů

Příklad

- Pro přidání prvku do kontejneru napíšeme funkci:

```
def pridej(kontejner,prvek):
     kontejner+=[prvek]
```

- Tato funkce však funguje pouze pro seznamy. Pokud chceme napsat univerzální funkci pro seznamy, množiny i slovníky, musíme testovat typ kontejneru:

```
def pridej(kontejner,prvek):
    if (type(kontejner)==list):
     kontejner+=[prvek]
    if (type(kontejner)==set):
     kontejner.add(prvek)
    if (type(kontejner)==dict):
     kontejner[list(prvek.keys())[0]]=list(prvek.values())[0]
```

Přidání prvku na konec kontejneru

- Není univerzální metoda či operátor pro všechny kontejnery.
- Seznam:

```
l += v
```

- Množina:

```
m.add(v)
```

- Slovník:

```
s[k]=v
```

Úkol

- S využitím řezů napište tyto funkce:
  - `def do_stredu(text)` – funkce pro řetězec text vrátí řetězec, který bude sestaven tak, že 1. znak řetězce text bude 1. znakem výsledného řetězce, 2. znak řetězce text bude posledním znakem výsledného řetězce, 3. znak řetězce text bude 2. znakem výsledného řetězce, ...
    - Příklad:

      `print(do_stredu("Python"))` vytiskne: `Ptonhy`

  - `def vlozeni_do_textu(text,vloz)` – funkce pro řetězce text a vloz vrátí řetězec, který bude sestaven tak, že za každým znakem z řetězce text bude vložen celý řetezec vloz.
    - Příklad:

      `print(vlozeni_do_textu("Python","abc"))` vytiskne: `Pabcyabctabchabcoabcnabc`

  - `def eratosthenovo_sito(n)` – funkce pro celé číslo n vrátí seznam prvočísel <=n, který bude sestaven podle algoritmu Erasthotenova síta.
    - Příklad:

      `print(eratosthenovo_sito(100))` vytiskne: `[1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]`