

Téma: Ako možno naprogramovať štandardné funkcie na prácu s reťazcami

Symbolika: text v zložených zátvorkách {} je v príkaze nepovinný.

Zopakuj si:

- príkazy if, for, while a break, generátor range (študijný text Python - príkazy)
- vlastné funkcie (študijný text Python - funkcie)
- reťazce, štandardné funkcie na prácu s reťazcami (študijný text Python - reťazce)
- výnimky (študijný text Python - výnimky) - nie nevyhnutné pre pochopenie ďalšieho textu

Príkaz if má tvar

if podmienka_1:

 blok_príkazov, ktorý sa vykoná, ak je splnená podmienka1

{ nasledujúcu časť možno opakovane použiť viackrát, ale aj vynechať

elif podmienka_i:

 blok_príkazov, ktorý sa vykoná, ak je splnená podmienka_i

}

{ nasledujúcu časť možno v príkaze if vynechať

else:

 blok_príkazov, ktorý sa vykoná, ak nie je splnená ani jedna z podmienok vyššie

}

Príkaz while má tvar

while podmienka:

 blok_príkazov, ktorý sa má opakovane vykonávať, pokiaľ je splnená podmienka

{ nasledujúca časť sa vo while-cykle používa len zriedka

else:

 blok_príkazov, ktorý sa vykoná po nesplnení podmienky (ak je „else“ použité)

}

Príkaz for s generátorom range má tvar

for prem_cyklu in range(od, po, krok):

 blok_príkazov, ktorý sa má opakovane vykonávať, pokiaľ hodnota prem_cyklu je z intervalu <od; po),

 pričom po každom vykonaní bloku príkazov sa hodnota prem_cyklu zmení o hodnotu krok (od, po a krok

 musia byť celé čísla)

{ nasledujúca časť sa vo for-cykle používa len zriedka

else:

 blok_príkazov, ktorý sa vykoná po ukončení príkazu for (ak je „else“ použité)

}

Definícia funkcie má tvar

def meno_funkcie ({formálny_parameter1, ..., formálny_parameterN}):

 blok_príkazov, ktoré sa majú vykonať, keď sa zavolá funkcia

Formálne parametre nemusia byť použité; používame ich, len ak chceme do funkcie doviest' nejaké hodnoty potrebné k výpočtu vo funkcii. Z funkcie sa nemusí vyvieš' žiadna hodnota. Ak sa má z funkcie vyvieš' hodnota, musí blok príkazov aspoň raz obsahovať príkaz **return** výraz. Vyhodnotením výrazu vznikne hodnota, ktorá bude vyvezená z funkcie, t.j. dosadená na miesto, kde sa uvedie meno_funkcie ({skutočný_parameter1,..., skutočný_parameterN}) - volanie funkcie. Skutočné parametre musíme použiť, ak sme použili formálne parametre pri písaní funkcie. So skutočnými parametrami sa uskutoční vykonanie bloku príkazov vo volanej funkcii.

Štandardné funkcie na prácu s reťazcami (výklad na vyučovacej hodine ☺):

Funkcia len(ret) - vráti počet znakov - dĺžku reťazca ret:

„Špecialita“ pythona:

```
def mlen(s):
    dlzka = 0
    for znak in s:
        dlzka += 1
    return dlzka
```

Použitie nárazníka:

```
def mlen1(s):
    dlzka = 0
    s += '\0' # vloženie nárazníka
    while s[dlzka] != '\0':
        dlzka += 1
    return dlzka
```

```
def mlen2(s):
    dlzka = 0
    try:
        while s[dlzka]:
            dlzka += 1
    except IndexError:
        return dlzka
```

Použitie:

```
print("Počet znakov:", mlen(ret))
pre ret = "horela hora" vráti
Počet znakov: 11
pre ret = "" vráti
Počet znakov: 0
```

Rezací operátor reťazec [*začiatok* : *koniec* : *krok*] má mnoho variant, naprogramujme si len niektoré, všetky s kladným krokom:

- a) reťazec [*začiatok*] - vracia podreťazec zo zadaného reťazca so znakmi od znaku s indexom *začiatok* až po posledný znak pôvodného reťazca. Našu funkciu si nazvime `mcopyEndStr`.

```
def mcopyEndStr(s,od):
    novy = ""
    for i in range(od, len(s)):
        novy += s[i]
    return novy
```

Použitie:

```
ret = "horela hora"
print(ret+"[7:]:", mcopyEndStr(ret,7)) vráti horela hora[7:]: hora
```

- b) reťazec [: *koniec*] - vracia podreťazec zo zadaného reťazca so znakmi od znaku s indexom 0 až pred znak na indexe *koniec*. Našu funkciu si nazvime `mcopyBeginStr`.

```
def mcopyBeginStr(s,po):
    novy = ""
    for i in range(0,po):
        novy += s[i]
    return novy
```

Použitie:

```
ret = "horela hora"
print(ret+"[:5]:", mcopyBeginStr(ret,5)) vráti horela hora[:5]: horel
```

- c) reťazec [*začiatok* : *koniec*] - vracia podreťazec zo zadaného reťazca so znakmi od znaku s indexom *začiatok* až pred znak na indexe *koniec*. Našu funkciu si nazvime `mcopyStr`.

```
def mcopyStr(s,od,po):
    novy = ""
    for i in range(od,po):
        novy += s[i]
    return novy
```

Použitie:

```
ret = "horela hora"
print(ret+"[3:10]:", mcopyStr(ret,3,10)) vráti horela hora[3:10]: ela hor
```

- d) reťazec [*začiatok* : *koniec* : *krok*] - vracia podreťazec zo zadaného reťazca so znakmi od znaku s indexom *začiatok* až pred znak na indexe *koniec*; *krok* udáva, znak s akým prírastkom indexu sa má preniesť do podreťazca. Našu funkciu si nazvime `mcopyStepStr`.

```
def mcopyStepStr(s,od,po,krok):
    novy = ""
    for i in range(od,po,krok):
        novy += s[i]
    return novy
```

Použitie:

```
ret = "horela hora"
print(ret+"[3:10:2]:", mcopyStepStr(ret,3,10,2)) vráti horela hora[3:10:2]: eahr
```

Funkcia `reťazec.count` (*hľadať, začiatok, koniec*) - vráti počet výskytov reťazca *hľadať* v reze reťazca *reťazec*. Naprogramovanie tejto funkcie si zjednodušíme, budeme hľadať počet výskytov len jednoznakového reťazca *hľadať* v podreťazci. Našu funkciu si nazvime `mcount`.

```
def mcount(s, hľadat, od, po):
    pocet = 0
    for i in range(od, po):
        if s[i] == hľadat:
            pocet += 1
    return pocet
```

Použitie:

```
ret = "horela hora"
znak = 'a'
print("Počet výskytov '"+znak+"' v podreťazci '"+ret[:]+"' :",
      mcount(ret, znak, 0, len(ret)))
```

vráti výpis: Počet výskytov 'a' v podreťazci 'horela hora': 2

```
print("Počet výskytov '"+znak+"' v podreťazci '"+ret[5:5]+"' :", mcount(ret, znak, 5, 5))
```

vráti výpis: Počet výskytov 'a' v podreťazci '': 0

Funkcia `reťazec.find` (*hľadať, začiatok, koniec*) - vráti index prvého výskytu zľava reťazca *hľadať* v reze reťazca *reťazec*. Ak sa hľadaný reťazec v reze reťazca nenachádza, funkcia vráti číslo -1. Naprogramovanie tejto funkcie si zjednodušíme, budeme hľadať prvý výskyt len jednoznakového reťazca *hľadať* od začiatku podreťazca. Našu funkciu si nazvime `mfind`.

```
def mfind(s, hľadat, od, po):
    for i in range(od, po):
        if s[i] == hľadat:
            return i
    return -1
```

Použitie:

```
ret = "horela hora"
znak = 'a'
print("Index prvého výskytu zľava '"+znak+"' v podreťazci '"+ret[:]+"' :",
      mfind(ret, znak, 0, len(ret)))
```

vráti výpis: Index prvého výskytu zľava 'a' v podreťazci 'horela hora': 5

Pre znak = 'x' vráti Index prvého výskytu zľava 'x' v podreťazci 'horela hora': -1

Úloha: Naprogramujte funkciu `mfindEnd`, ktorá vráti index posledného výskytu hľadaného znaku v reze reťazca *reťazec* alebo -1, ak sa hľadaný znak v reze reťazca nevyskytuje. Jedno z možných riešení:

```
def mfindEnd(s, hľadat, od, po):
    posledny = -1
    for i in range(od, po):
        if s[i] == hľadat:
            posledny = i
    return posledny
# pri každom nájdení hľadat aktualizuje hodnotu posledny
```

Použitie:

```
znak = 'h'
print("Index posledného výskytu znaku '"+znak+"' v podreťazci '"+ret[:]+"' :",
      mfindEnd(ret, znak, 0, len(ret)))
```

Iné riešenie (hľadá zľava prvý výskyt v obrátenom reťazci):

```
pom = ret[::-1] # pom je opačný reťazec k reťazcu ret
print("Kontrola:", len(pom) - mfind(pom, znak, 0, len(pom)) - 1) # výpočet indexu s mfind
print("Kontrola:", len(pom) - pom.find(znak) - 1) # výpočet indexu s find
```

Funkcia `reťazec.index` (*hľadať, začiatok, koniec*) - vráti index prvého výskytu zľava reťazca *hľadať* v reze reťazca *reťazec* alebo vyvolá výnimku (ak sa reťazec *hľadať* v reze nevyskytuje). Je „slabšia“ ako funkcia `find`, tak sa ju pokúsme naprogramovať len pre zaujímavosť. Naprogramovanie aj tejto funkcie si zjednodušíme, budeme hľadať

prvý výskyt len jednoznakového reťazca *hľadať* od začiatku podreťazca a ak sa *hľadať* v podreťazci nevyskytuje, stačí nám hlásenie "ValueError: substring not found". Našu funkciu si nazvime mindex.

```
def mindex(s, hľadat, od, po):
    s = s[od:po]
    i = 0
    try:
        while s[i] != hľadat:
            i += 1
        return i
    except IndexError:
        return "ValueError: substring not found"
```

Použitie:

```
ret = "horela hora"
znak = 'x'
print("Index prvého výskytu zľava '"+znak+"' v podreťazci '"+ret[:]+"':",
mindex(ret, znak, 0, len(ret)))
```

vráti výpis: Index prvého výskytu zľava 'x' v podreťazci 'horela hora': ValueError: substring not found

Funkcia *reťazec.isalpha()* vráti True, ak je *reťazec* neprázdny a každý znak je písmenom (aj národnej abecedy). Našu funkciu si nazvime misalpha.

Funkcia *reťazec.isdigit()* vráti True, ak je *reťazec* neprázdny a každý jeho znak je číslicou (od 0 po 9). Našu funkciu si nazvime misdigit.

Funkcia *reťazec.isalnum()* vráti True, ak je *reťazec* neprázdny a každý znak v reťazci je písmenom (aj národnej abecedy) alebo číslicou (od 0 po 9). Našu funkciu si nazvime misalnum.

Naprogramovanie týchto funkcií si opäť zjednodušíme. Za písmená budeme považovať len písmená anglickej abecedy (teda znaky a až z alebo A až Z).

```
def misalpha(s):
    if len(s) > 0:
        for znak in s:
            if not ('A' <= znak <= 'Z' or 'a' <= znak <= 'z'):
                # if znak not in "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz":
                    return False
        return True
    else:
        return False

def misdigit(s):
    if len(s) > 0:
        for znak in s:
            if not ('0' <= znak <= '9'):
                # alebo if znak not in "0123456789":
                    return False
        return True
    else:
        return False

def misalnum(s):
    if len(s) > 0:
        for znak in s:
            if not (misalpha(znak) or misdigit(znak)):
                return False
        return True
    else:
        return False
```

Použitie:

Príkazy

```
print("misalpha:", misalpha(ret))
print("Kontrola:", ret.isalpha())
print("misdigit: ", misdigit(ret))
print("Kontrola:", ret.isdigit())
print("misalnum: ", misalnum(ret))
print("Kontrola:", ret.isalnum())
```

vrátia výpisy:

Retazec: "1234" misalpha: False Retazec: "1234" Kontrola: False misdigit: True Kontrola: True misalnum: True Kontrola: True	Retazec: "u1234" misalpha: False Retazec: "u1234" Kontrola: False misdigit: False Kontrola: False misalnum: True Kontrola: True	Retazec: "u 1234" misalpha: False Retazec: "u 1234" Kontrola: False misdigit: False Kontrola: False misalnum: False Kontrola: False	Retazec: "Jánošík" misalpha: False Retazec: "Jánošík" Kontrola: True misdigit: False Kontrola: False misalnum: False Kontrola: True
--	--	--	--

Funkcia `oddelovač.join(reťazce)` vráti reťazec spojených reťazcov vytvorený z postupnosti *reťazce*, oddelených reťazcom *oddelovač*. Našu funkciu si nazvime `mjoin`.

```
def mjoin(oddel_cim, zoz_ret):
    novy = zoz_ret[0]
    for i in range(1, len(zoz_ret)):
        novy += oddel_cim + zoz_ret[i]
    return novy
```

Použitie:

```
retazce = ("Adam", "Fero", "Dušan", "Zora", "Zuzana") # pozri Python - údajové typy (N-tica, tuple)
print("mjoin:", mjoin(" ", retazce))
print("Kontrola:", " ".join(retazce))
vypíše
mjoin: Adam Fero Dušan Zora Zuzana
Kontrola: Adam Fero Dušan Zora Zuzana
```

Funkcia `reťazec.lower()` vráti kópiu reťazca *reťazec*, veľké písmená zmenené na malé. Našu funkciu si nazvime `mlower` a predpokladajme, že v reťazci boli použité len písmená anglickej abecedy.

```
def mlower(s):
    POSUN = ord('a') - ord('A')
    novy = ""
    for znak in s:
        if 'A' <= znak <= 'Z': # alebo if znak in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
            novy += chr(ord(znak) + POSUN)
        else:
            novy += znak
    return novy
```

Použitie:

```
ret = ret.upper()
print("mlower:", mlower(ret))
print("Kontrola:", ret.lower())
vypíše
mlower: horela hora
Kontrola: horela hora
pre ret = "JÁNOŠÍK" vypíše
mlower: jánošík
Kontrola: jánošík
```

Funkcia `reťazec.replace(nahrad_čo, nahrad_čím, n-krát)` vráti kópiu reťazca *reťazec*, v ktorej je každý reťazec *nahrad_čo* nahradený reťazcom *nahrad_čím* najviac *n-krát*. V našej verzii `mreplace` nahradíme všetky výskyty

jednoznakového reťazca, vo verzii mreplaceN nahradíme najviac prvých n výskytov jednoznakového reťazca v reťazci s.

```
def mreplace(s, nahrad_co, nahrad_cim):
    novy = ""
    for znak in s:
        if znak == nahrad_co:
            novy += nahrad_cim
        else:
            novy += znak
    return novy
```

Použitie:

```
print("mreplace:", mreplace(ret, "h", "m"))
print("Kontrola:", ret.replace("h", "m"))
```

vypíše pre reťazec "horela hora"

```
mreplace: morela mora
Kontrola: morela mora
```

```
def mreplaceN(s, nahrad_co, nahrad_cim, n_krat):
    novy = ""
    pocet_nahradeni = 0
    for znak in s:
        if znak == nahrad_co and pocet_nahradeni < n_krat:
            novy += nahrad_cim
            pocet_nahradeni += 1
        else:
            novy += znak
    return novy
```

Použitie:

```
print("mreplaceN:", mreplaceN(ret, "h", "m", 1))
print("Kontrola:", ret.replace("h", "m", 1))
```

vypíše pre reťazec "horela hora"

```
mreplaceN: morela hora
Kontrola: morela hora
```

Funkcia reťazec.split(oddelené_čím, n-krát) vráti zoznam reťazcov oddelených najviac *n-krát* reťazcom *oddelené_čím*; ak nie je zadané *oddelené_čím*, reťazec sa rozdelí podľa medzier; *n-krát* je nepovinný parameter. Naša funkcia msplit bude vracáť nie zoznam, ale n-ticu (tuple), parameter *n-krát* nepoužijeme a *oddelené_čím*, ak nebude zadaný, nahradí medzerou.

```
def msplit(s, oddelene_cim = " "):
    entica = ()
    prvok = ""
    for znak in s:
        if znak != oddelene_cim:
            prvok += znak
        else:
            entica += (prvok,)
            prvok = ""
    entica += (prvok,)
    return entica
```

Použitie:

```
print("msplit:", msplit(ret))
ntica = msplit(ret)
print("Kontrola:", ret.split())
```

vypíše pre reťazec "horela hora"

```
msplit: ('horela', 'hora')
Kontrola: ['horela', 'hora']
```

alebo

```
print("msplit:", msplit(ret,"r"))
ntica = msplit(ret,"r")
print("Kontrola:", ret.split("r"))
vypíše pre reťazec "horela hora"
msplit: ('ho', 'ela ho', 'a')
Kontrola: ['ho', 'ela ho', 'a']
```

Ďalšie zaujímavé funkcie:

Funkcia `prevedStrNaInt(reťazec)` simuluje konverziu `int(reťazec)`

```
def prevedStrNaInt(s):
    cislo = 0
    for znak in s:
        cifra = ord(znak) - ord('0')
        cislo = cislo*10 + cifra
    return cislo
```

Použitie:

```
print(2*prevedStrNaInt("150"))
```

vypíše 300

Funkcia `prevedIntNaStr(celé_číslo)` simuluje konverziu `str(celé_číslo)`

```
def prevedIntNaStr(cislo):
    if cislo == 0: s = '0'
    else:
        s = ''
        while cislo>0:
            cifra = cislo % 10
            s = chr(ord('0')+cifra) + s
            cislo //= 10
    return s
```

Použitie:

```
print(2*prevedIntNaStr(150))
```

vypíše 150150

Funkcia `vratVsetkyVyskyty(reťazec, hľadat_znak, od, po)` vráti reťazec obsahujúci indexy výskytov hľadaného znaku v reze `<od, po)` reťazca `reťazec` oddelené medzerami, alebo prázdny reťazec, ak sa hľadaný znak v reťazci ani raz nevyskytol.

```
def vratVsetkyVyskyty(s, hladat, od, po):
    indexy = ""
    for i in range(od, po):
        if s[i] == hladat:
            indexy += " " + str(i)
    return indexy
```

Použitie:

```
ret = "horela hora"
```

```
hladat = "a"
```

```
vsetkyVyskyty = vratVsetkyVyskyty(ret, hladat, 1, len(ret))
```

```
if vsetkyVyskyty == "":
    print("'+hladat+'" sa v reze reťazca nevyskytlo.")
else:
    print("'+hladat+'" sa v reze vyskytlo na indexoch:" + vsetkyVyskyty)
```

vypíše:

```
'a' sa v reze vyskytlo na indexoch: 5 10
```


Funkcia `vratZoznamIndexov` (*reťazec*, *hladat_znak*, *od*, *po*) vráti zoznam (pole, list) obsahujúci indexy výskytov hľadaného znaku v reze <od, po) reťazca *reťazec*, alebo prázdny zoznam ([]), ak sa hľadaný znak v reťazci ani raz nevyskytol.

```
def vratZoznamIndexov(s, hladat, od, po):  
    zoz_indexov = []  
    for i in range(od, po):  
        if s[i] == hladat:  
            zoz_indexov.append(i)  
    return zoz_indexov
```

Použitie:

```
ret = "horela hora, v tej hore hrala kapela"  
hladat = "a"  
vsetkyInx = vratZoznamIndexov(ret, hladat, 1, len(ret))  
print(vsetkyInx)
```

vypíše:

```
[5, 10, 26, 28, 31, 35]
```