Reťazce, zoznamy, n-tice, množiny a slovníky

1 Usporiadané postupnosti a rezy

Reťazec `str`, zoznam `list` a n-tica `tuple` sú usporiadané postupnosti. Zachovávajú poradie prvkov, môžu obsahovať opakované hodnoty a možno ich prechádzať cyklom `for`. Prvý prvok má index `0`. Záporné indexy počítajú od konca: `-1` označuje posledný prvok. Funkcia `len()` vráti počet prvkov a operátor `in` overí, či sa hodnota v postupnosti nachádza. Rez `postupnost[start:stop:step]` vytvorí novú postupnosť. Hodnota na indexe `start` sa zahrnie, hodnota na indexe `stop` nie. Vynechaný začiatok znamená začiatok postupnosti, vynechaný koniec jej koniec; predvolený krok je `1`.

```
s = "Dnes je velmi pekne"
print(len(s))      # 19
print(s[0], s[-1]) # D e
print(s[8:13])     # velmi
print(s[-6:-3])    #  pe
print(s[:4], s[8:], s[::2])
print(s[::-1])
```

Pri dĺžke `19` označuje index `-6` index `13` a index `-3` index `16`. Preto rez `s[-6:-3]` vyberie znaky s indexmi `13`, `14` a `15`.

```
l = [1, 2, 3, 4]
print(l[2], l[-2])
print(l[1:3], l[2:], l[:3])
print(len(l)); print(3 in l)
```

Prístup k neexistujúcemu indexu vyvolá `IndexError`. Rez reťazca, zoznamu alebo n-tice je reťazec, zoznam alebo n-tica.

2 Reťazec str

Reťazec je nemenná postupnosť znakov Unicode. Jednotlivý znak nemožno priradením zmeniť; treba vytvoriť nový reťazec. Operátor `+` reťazce spojí a `*` ich zopakuje.

```
prazdny = "" # alternativa: prazdny = str()
text = "Dnes" + " je pekne"
print(text)
print("pekne" in text)
print("ha " * 3 + "!")
```

Reťazce sa porovnávajú lexikograficky podľa kódových bodov Unicode. Rozlišujú sa veľké a malé písmená. Funkcia `ord(znak)` vráti číselný kód znaku a `chr(kod)` vykoná opačný prevod.

```
print(ord("A"), chr(65))
print("Adam" < "Boris")
print("a" == "A")
```

Funkcie `count()` a `find()` spočítajú alebo nájdu podreťazec; Funkcia `find()` pri neúspechu vráti `-1`. Funkcie `lower()` a `upper()` menia veľkosť písmen v novom reťazci. Funkcia `strip()` odstráni “biele” znaky na oboch koncoch.

```
text = "  Python je pekny jazyk  "
cisty = text.strip()
print(cisty.count("y"), cisty.find("jazyk"))
print(cisty.lower(), cisty.upper())
slova = cisty.split()
print(slova)
print("-".join(slova))
```

`Funkcia split()` rozdelí reťazec na zoznam častí. Zapis `oddelovac.join(postupnost)` spojí postupnosť reťazcov so zvoleným oddeľovačom. Funkcia `join()` nevie priamo spájať čísla; tie treba najskôr previesť pomocou `str()`.

3 Zoznam list

Zoznam je usporiadaná mutovateľná postupnosť. Zapisuje sa hranatými zátvorkami. Môže obsahovať rôzne typy, ale zoznam rovnakých druhov hodnôt sa spracúva jednoduchšie. Prvok možno zmeniť priradením `l[index] = hodnota`. Operátory `+` a `*` vytvoria nový zoznam; rozšírené priradenia `+=` a `*=` menia (mutujú) pôvodný zoznam.

```
prazdny = [] # alternativa: prazdny = list()
l = [4, 1, 3, 1, 2]
print(l[1:3], l[2:], l[:3]); print(type(l)); print(l.count(1)); print(l.index(1)); print(3 in l)
print((1, 2) * 3)
print((1, 2) + (3, 4))
```

Funkcia `append()` pridá prvok na koniec a `insert()` na zadanú pozíciu. Funkcia `remove()` odstráni prvý výskyt hodnoty a `pop()` odstráni a vráti prvok.

```
l.append(5); l.insert(1, 0); l[0] = 5
print(l)
print("odstránené:", l.pop())
l.remove(0)
print(l)
```

Funkcia `clear()` odstráni všetky prvky, funkcia `count()` počíta výskyty a `index()` vráti index prvého výskytu.

```
lk = l.copy()
lk.sort()
l.reverse()
print(l, lk)
```

Funkcia `sort()` usporiada zoznam a nič nevráti. Funkcia `sorted()` vráti nový usporiadaný zoznam. Priradenie `b = l` nevytvorí kópiu; obe mená odkazujú na ten istý zoznam. Jednoduchú kópiu vytvorí `l.copy()` alebo `l[:]`.

4 N-tica tuple

N-tica je usporiadaná nemutovateľná postupnosť. Zapisuje sa pomocou `(,)`. Rozhodujúca je čiarka: jednoprvková n-tica je `(1,)`, zatiaľ čo `(1)` je iba celé číslo.

```
prazdna = () # alternativa: prazdna = tuple()
t = (4, 1, 3, 1, 2)
print(t[1:3], t[2:], t[:3]); print(type(t)); print(t.count(2)); print(t.index(3)); print(1 in t)
print((1, 2) * 3)
print((1, 2) + (3, 4))
```

N-tica podporuje indexovanie, rezy, spájanie, opakovanie, `count()` a `index()`, **nemá** `append()` ani priradenie do prvku. Hodí sa na údaje, ktoré tvoria jeden celok a nemajú sa meniť.

```
bod = (3.0, 2.0)
x, y = bod
print(x, y)
usecka = ((0.0, 0.0), (4.0, 3.0))
(x1, y1), (x2, y2) = usecka
print(x1, y1, x2, y2)
```

Zápis `x, y = bod` rozbalí prvky n-tice do premenných. Počet mien naľavo sa musí zhodovať s počtom hodnôt vpravo.

5 Množina set

Množina je meniteľná kolekcia jedinečných prvkov bez indexov. Opakované hodnoty sa pri vytvorení odstránia. Poradie prvkov nie je určené a nemožno sa naň spoliehať. Neprázdna množina sa zapisuje pomocou `{,}`. Prázdna množina je `set()`. Množina **nepodporuje** indexovanie, `+` ani `*`, ako reťazec či list.

```
prazdna = set() # nema alternativu
s = {4, 1, 3, 1, 2}
print(s), print(type(s)); print(len(s)); print(3 in s)
s.add(0)
s.remove(9) # alternativa: s.discard(9)
print(set([1, 2, 3, 2, 1]))
print(set("trava")) # mnozina nema usporiadanie
```

Funkcia `add()` pridá prvok. Funkcia `remove()` pri chýbajúcom prvku vyvolá `KeyError`, kým `discard()` neurobí nič. Funkcia `clear()` množinu vyprázdni.

```
s1 = {0, 2, 4, 6}
s2 = {1, 3, 5, 7}
print(s1 | s2)              # zjednotenie
print(s1 & s2)              # prienik
print(s1 - s2)              # rozdiel
print({0, 2} <= s1)         # je podmnožina ?
print(s1.union(s2))
print(s1.intersection(s2))
print(s1.difference(s2))
```

Operátory `|`, `&` a `-` zodpovedajú funkciám `union()`, `intersection()` a `difference()`. Operátor `<=` alebo funkcia `issubset()` testuje podmnožinu. Tieto operácie nemenia pôvodné množiny.

6 Slovník dict

Slovník uchováva dvojice `kľúč: hodnota`. Ku hodnote sa pristupuje podľa kľúča, nie podľa poradového indexu. Kľúče sú jedinečné; bežne sú nimi reťazce alebo čísla. Hodnoty môžu mať ľubovoľný typ.

```
prazdny = {} # alternativa: prazdny = dict()
d = {"x": 1, "y": 2}
print(d["x"]), print(type(d))
d["x"] = 11; d["z"] = 3
print(d); print(d.keys()); print(d.values()); print(d.items())
```

Priradenie k existujúcemu kľúču hodnotu zmení; nový kľúč pridá novú dvojicu. Výraz `kluc in d` testuje prítomnosť kľúča. Funkcia `get()` umožní určiť predvolenú hodnotu a nevyvolá chybu pri chýbajúcom kľúči.

```
auto = {"znacka": "Ford", "model": "Mustang", "rok": 1984}
print(auto["model"]); print(auto.get("farba", "nezadaná"))
for kluc, hodnota in auto.items():
    print(kluc, hodnota)
auto.pop("rok")
print(auto)
```

Zápis `dict(meno="Janko", priezvisko="Hrasko")` vytvorí slovník s reťazcovými kľúčmi. Funkcie `keys()`, `values()` a `items()` poskytujú pohľady na kľúče, hodnoty a dvojice; `pop()` odstráni položku.

7 Konverzie medzi kolekciami

Funkcie `list()`, `tuple()` a `set()` vytvoria kolekciu z iterovateľného objektu. Konverzia na množinu odstráni opakovania a nezachová použiteľné poradie. Spájať operátorom `+` možno iba rovnaké typy postupností.

```
print(list("Python"))
print(tuple([1, 2, 3]))
print(set((1, 2, 3, 2, 1)))
cisla = [1, 2, 3, 4]
text = ", ".join([str(cislo) for cislo in cisla])
print(text)
```

8 Jednoduchý list comprehension

List comprehension vytvorí zoznam stručným zápisom cyklu. Základný tvar je `[výraz for premenná in postupnosť]`. Na konci môže byť jednoduchá podmienka `if`.

```
znaky = [znak for znak in "Python"]
stvorce = [hodnota * hodnota for hodnota in range(10)]
parne = [cislo for cislo in range(10) if cislo % 2 == 0]
kody = [ord(znak) for znak in "ABC"]
print(znaky); print(stvorce); print(parne); print(kody)
```

9 Generátorový výraz

Generátorový výraz s použitím cyklu a podmienky

```
",".join(str(i) for i in range(0, 10))
",".join(str(i) for i in range(10) if i % 2 == 0)
" ".join("parne" if i % 2 == 0 else "neparne" for i in range(5))

"".join(chr((ord(c) - ord('a') + 1) % (ord('z') - ord('a') + 1) + ord('a')) for c in "python")
```