Python akademie - lekce 1 - 17.10.2024

# 01_04 🥬 Úvod k sekvencím

(k některým kontejnerovým datovým typům ~sequencies)

---

## Zajímavé odkazy z této lekce:

- Oficiální dokumentace pro `list`, `tuple`

---

Doposud jsme si ukázali, jak pracovat s proměnnou, která obsahuje **jedno číslo** ( `int`, `float` ), nebo **jeden řetězec** textových znaků ( `str` ).

Pojďme si nyní ukázat, že Python umí pracovat i s údaji, které obsahují více různých informací jako několik čísel, nebo několik textových hodnot.

Takové hodnoty potom budeme označovat jako tzv. **sekvenční datové typy** (tedy v jedné proměnné bude několik oddělených údajů). Obecně Python nabízí tyto tři základní sekvenční typy:

1. `list` (z angl. *list*, česky *seznam*),
2. `tuple` (z angl. *tuple*, česky *n-tice*),
3. `range` (z angl. *range*, česky *rozsah*) - na něj přijde řada později.

## Seznam (~list)

---

*List* je opravdu datový typ, který je doslova tvořen seznamem údajů. Tyto údaje jsou oddělené datovým oddělovačem čárkou:

```
muj_seznam = ["Matous", "Marek", "Lukas", "Jan"]
```

V příkladu si můžeme všimnout některých **charakteristických rysů** pro `list`:

1. **Hranaté závorky** na začátku a na konci listu,
2. **stringy**, které náš list obsahuje,
3. **čárky**, které oddělují jednotlivé hodnoty,
4. **proměnná**, do které si nově napsaný list schovám (`muj_seznam`).

Opět si můžeš pomocí funkce `type` ověřit datový typ. Není nutné chápat celkový význam výstupu funkce `type`. Stačí si povšimnout výrazu `list` ve výstupu.

```
print(type(["Matous", "Marek", "Lukas", "Jan"]))
```

```
<class 'list'>
```

## Jak vytvořit list

Nejprve si ukážeme možnosti, jak **vytvořit prázdný list**, kam si budeš moct v budoucnu ukládat svoje hodnoty:

1. Možnost, pomocí **prázdných hranatých závorek**,
2. Možnost, pomocí **zabudované funkce** `list`.

```
prvni_seznam = []
```

```
druhy_seznam = list()
```

```
print(type(prvni_seznam))
```

```
<class 'list'>
```

```
print(type(druhy_seznam))
```

```
<class 'list'>
```

Opět použijeme funkci `type` pro ověření, že výsledné hodnoty jsou skutečně typu `list`.

Pokud potřebuješ vytvořit neprázdný list, můžeš údaje zapsat přímo do hranaté závorky (jako první úkazka v této kapitole):

```
treti_seznam = [2, 4, 6, 8, 10]

ctvrty_seznam = [1.0, 3.0, 5.0, 7.0, 9.0]

print(type(treti_seznam))
```

```
<class 'list'>
```

```
print(type(ctvrty_seznam))
```

```
<class 'list'>
```

## Jak pracovat s listem

Hodnoty, které `list` obsahuje, můžeš zpřístupnit pomocí jejich **pořadí**, tedy indexů. Tento princip funguje stejně jako jsme si ukázali u stringů.

```
muj_seznam = ["Matous", "Marek", "Lukas", "Jan"]

print(muj_seznam[0])
```

```
Matous
```

```
print(muj_seznam[1])
```

```
Marek
```

```
print(muj_seznam[-1])
```

```
Jan
```

```
print(muj_seznam[1:3])
```

```
['Marek', 'Lukas']
```

```
print('indexování'[-1:-5:-1])
```

```
ínáv
```

```
print('indexování'[-5:-1])
```

```
ován
```

Tedy index `0` představuje **první hodnotu** a index `-1` **poslední hodnotu**.

## Tuple (~n-tice)

---

Tuple je na první pohled velice podobný **listu** (seznamu):

```
muj_tupl = ("Matous", "Marek", "Lukas", "Jan")
```

Pro srovnání s `list`:

```
muj_seznam = ["Matous", "Marek", "Lukas", "Jan"]
```

V příkladu si můžeme všimnout některých **charakteristických rysů** pro `tuple`:

1. **Kulaté závorky** na začátku a na konci tuplu,
2. **stringy**, které naše sekvence obsahuje,
3. **čárky**, které oddělují jednotlivé hodnoty,
4. **proměnná**, do které si nově napsaný tupl schováme (`muj_tupl`).

Proč je nutné mít jak `list`, tak `tuple`, když jsou tak podobné. Hlavním rozdílem je **změnitelnost**.

| Sekvenční typ | Změnitelnost | Vysvětlení |
|---|---|---|
| `list` (~seznam) | *mutable* (~změnitelný) | Můžeš přidávat a odebírat hodnoty |
| `tuple` (~n-tice) | *immutable* (~nezměnitelný) | Jakmile jej vytvoříš, nelze změnit |

Z tabulky uvedené výše vyplývá, že pokud chceš pracovat se sekvencí, u které budeš v průběhu **měnit její obsah**, použiješ `list` (~seznam).

Naopak pokud budeš chtít jako programátor napsat takovou sekvenci, kterou si **nepřeješ změnit** (a dát to na uvědoměnou sobě nebo ostatním programátorům), použiješ `tuple`. Podívej se na ukázku níže:

```
nejvetsi_mesta = ("Praha", "Brno", "Ostrava", "Plzen", "Liberec", "Olomouc")
```

V tuplu `nejvetsi_mesta` jsou všechna města v České republice, která mají více než 100 000 obyvatel. Pro nás je toto zásadní hodnota a nechceme, aby do této proměnné kdokoliv přidal nějaký další údaj. Na základě této potřeby jsme vybrali `tuple`.

## Jak vytvořit tuple

Vzhledem k faktu, že je `tuple` nezměnitelný, není vytvoření prázdného tuplu moc výhodné. Nicméně, pokud bychom to vážně potřebovali, postupujeme jako u listu:

1. Pomocí **prázdných kulatých závorek**,
2. pomocí **zabudované funkce** `tuple`.

```
prvni_tupl = ()

druhy_tupl = tuple()

print(type(prvni_tupl))

print(type(druhy_tupl))
```

Ovšem takové datové struktury nejsou příliš užitečné (obvykle vyžadujeme různé hodnoty), a proto se zaměříme na vytvoření **neprázdných tuplů**:

```
treti_tupl = ("Praha", "Berlin", "Varsava", "Bratislava", "Viden")

ctvrty_tupl = 1.3, 3.6, 1.8, 0.4, 1.9

print(type(treti_tupl))
```

```
print(type(ctvrty_tupl))
```

Ačkoliv varianta bez kulatých závorek není zcela běžná, můžeš se s ní setkat.

## Jak pracovat s tuplem

Stejně jako `list` můžeš i `tuple` *indexovat*, *rozkrájet* (slicing), *přeskakovat* (~striding):

```
treti_tupl = ("Praha", "Berlin", "Varsava", "Bratislava", "Viden")

print(treti_tupl[0])

print(treti_tupl[-1])

print(treti_tupl[0:2])

print(treti_tupl[-2])
```