
KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
https://www.inf.upol.cz/lide/jan-lastovicka
jan.lastovicka@upol.cz
17. listopadu 12, 771 46 Olomouc

Úvod do programovacích stylů ◇ poznámky k přednášce

8. Iterátory

verze z 22. listopadu 2023

1 Iterátory

Na hodnoty, které se skládají z prvků, můžeme pohlížet jako na **posloupnosti**. Například pole `[1, 2, 3]` můžeme chápat jako posloupnost hodnot 1, 2 a 3. Posloupnosti se také nazývají **iterovatelné hodnoty**.

Průchod posloupností reprezentujeme hodnotou, která se nazývá **iterátor**. Iterátor k iterovatelné hodnotě *`iterable`* získáme zavoláním vestavěné funkce `iter`:

```
iter(iterable) => iterator
```

Například:

```
>>> i = iter([1, 2, 3])
```

Voláním vestavěné funkce `next` na iterátor postupně získáváme prvky posloupnosti:

```
next(iterator) => element
```

Vyzkoušíme:

```
>>> next(i)
1
>>> next(i)
2
>>> next(i)
3
```

Pokud další prvek posloupnosti neexistuje, funkce `next` vyvolá výjimku `StopIteration`. Tedy:

```
 >>> next(i)
StopIteration
```

Přirozeně pro jednu iterovatelnou hodnotu můžeme vytvořit více iterátorů:

```
>>> array = [1, 2, 3]
>>> i1 = iter(array)
>>> i2 = iter(array)
>>> next(i1)
1
>>> next(i2)
1
>>> next(i1)
2
>>> next(i1)
3
>>> next(i1)
StopIteration
>>> next(i2)
2
>>> next(i2)
3
>>> next(i2)
StopIteration
```

Iterátor sám je iterovatelná hodnota představující zbytek dosud neprojité posloupnosti:

```
>>> i1 = iter([1, 2, 3])
>>> next(i1)
1
>>> i2 = iter(i1)
>>> next(i2)
2
```

Pokračování v průchodu `i1` mění průchod `i2`:

```
>>> next(i1)
3
>>> next(i2)
StopIteration
```

Iterátory nelze použít ve funkcionálním programovacím stylu a to z toho důvodu, že při každém zavoláním funkce `next` dojde ke změně iterátoru. Jak víme, změny hodnot jsou ve funkcionálním stylu zakázané.

Řetězec je iterovatelná hodnota. Na řetězec se můžeme dívat jako na posloupnost znaků, které jej tvoří. Například:

```
>>> i = iter("py")
```

```
>>> next(i)
'p'
>>> next(i)
'y'
>>> next(i)
StopIteration
```

Vestavěná funkce:

```
range(start, stop)
```

vrací iterovatelnou hodnotu, která představuje posloupnost čísel větších nebo rovno než *start* a menších než *stop*. Volání

```
range(stop)
```

je ekvivalentní s

```
range(0, stop)
```

Například máme:

```
>>> i = iter(range(1, 3))
>>> next(i)
1
>>> next(i)
2
>>> next(i)
StopIteration
```

Příkaz `for` má tvar:

```
for variable in iterable:
    block
```

kde *variable* je proměnná, *iterable* iterovatelná hodnota a *block* blok kódu. Příkaz se vykoná následovně:

1. Získá se iterátor *iterator* iterovatelné hodnoty *iterable*.
2. Dokud volání `next(`*iterator*`)` nevyvolá výjimku `StopIteration`, tak
   (a) se nastaví proměnná *variable* na návratovou hodnotu funkce `next`
   (b) a vykoná se *block*.

Například:

```
>>> for i in range(3):
        print(i)
0
1
2
```

2 Generátory

Příkaz produkující prvek posloupnosti nabývá tvaru:

```
yield element
```

kde *element* je další prvek posloupnosti. Příkaz může být použit pouze v těle funkce.

Funkce, která obsahuje v těle příkaz `yield` se nazývá **generator**. Příkaz návratu (`return`) v těle generátoru smí vracet pouze hodnotu `None`.

Příklad generátoru:

```
def get_numbers():
    i = 0
    while i < 3:
        yield i
        i = i + 1
```

Poté, co se zavolá generátor, vykonávání těla generátoru se pozastaví před vykonáním prvního příkazu, a volání vrátí iterátor. Tedy:

```
>>> i = get_numbers()
```

Při vyžádání dalšího prvku funkcí `next` se začne vykonávat tělo generátoru od pozastaveného místa až po příkaz `yield`. Dalším prvkem posloupnosti bude hodnota určená příkazem `yield`. Vykonávání těla generátoru se pozastaví na následujícím příkazu. Tedy:

```
>>> next(i)
0
```

Vykonávání těla generátoru je pozastaveno na řádku:

```
i = i + 1
```

Popsaným způsobem získáme další prvky posloupnosti:

```
>>> next(i)
1
>>> next(i)
2
```

Skončení vykonávání těla generátoru povede k vyvolání výjimky `StopIteration`:

```
>>> next(i)
StopIteration
```

3 Generující výraz

Iterátor lze vytvořit pomocí **generujícího výrazu**, který má tvar:

```
(element clauses)
```

kde *element* je výraz určující prvky iterátoru a část *clauses* určuje hodnoty proměnných použitých v *element*.

Část *clauses* obsahuje za sebe napsané klauzule if a for. První klauzule je vždy for klauzule.

For klauzule má tvar:

```
for variable in iterable
```

kde *variable* je proměnná a *iterable* iterovatelná hodnota.

If klauzule má tvar:

```
if condition
```

kde *condition* je podmínka.

Například:

```
(x + 1 for x in [1, 4, 2, 3] if x % 2 == 0)
```

Hodnotou generujícího výrazu je iterátor, kde prvky jsou určeny následovně:

Vyhodnocení for klauzule:

```
for variable in iterable
```

probíhá tak, že se pro každý prvek iterovatelné hodnoty *iterable* vyhodnotí zbývající klauzule, kde proměnná *variable* bude aktuálním prvkem posloupnosti.

Například klauzule:

```
for x in [1, 4, 2, 3]
```

Vyhodnotí zbytek klauzulí, kde proměnná `x` bude postupně nabývat hodnot 1, 4, 2 a 3.

Při vyhodnocování if klauzule se vyhodnotí její podmínka. Pokud je podmínka splněna, pokračuje se následující klauzulí. Jinak se přeruší postupné vyhodnocování klauzulí způsobené naposledy vyhodnocovanou for klauzulí.

Pokud se podaří vyhodnotit všechny klauzule až do konce, pak další hodnota iterátoru je hodnota výrazu *`element`*.

Proto:

```
>>> i = (x + 1 for x in [1, 4, 2, 3] if x % 2 == 0)
>>> next(i)
5
>>> next(i)
3
>>> next(i)
StopIteration
```

Pokud je výraz generátoru použit jako argument volání funkce, můžeme vynechat vnější kulaté závorky:

```
>>> list(x ** 2 for x in range(5))
[0, 1, 4, 9, 16]
```

Klauzule for může záviset na proměnných uvedených dříve zapsanou for klauzulí:

```
>>> list([x, y] for x in range(4) for y in range(x))
[[1, 0], [2, 0], [2, 1], [3, 0], [3, 1], [3, 2]]
```

Uzavřením popisu generování do hranatých místo kulatých závorek rovnou vytvoříme seznam:

```
>>> [[x, y] for x in range(4) for y in range(x)]
[[1, 0], [2, 0], [2, 1], [3, 0], [3, 1], [3, 2]]
```

Poslední příklad:

```
>>> [[x, y] for x in range(1, 10) for y in range(2, x) if x % y == 0]
[[4, 2], [6, 2], [6, 3], [8, 2], [8, 4], [9, 3]]
```