C2184 Úvod do programování v Pythonu

4. Podmínky a cykly

Opakování:

Logické výrazy a logické operátory

Vyhodnocením logických výrazů získáme pouze výsledek `True` nebo `False`.

Logické operátory I.

- == je rovno, != není rovno
- > je větší, >= je vetší rovno
- < je menší, <= je menší rovno

In [1]:
```
8 < 10
```

Out[1]:
```
True
```

In [3]:
```
9 >= 9
```

Out[3]:
```
True
```

In [4]:
```
x = 5
y = 10
x < y
```

Out[4]:
```
True
```

Logické operátory II.

- `and` – "a" (logický součin)
- `or` – "nebo" (logický součet)
- `not` – negace

In [5]:
```
'abc'.startswith('a') and 9 > 1
```

Out[5]:
```
True
```

In [6]:
```
False or False
```

Out[6]:
```
False
```

In [7]:
```
not False
```

Out[7]:
```
True
```

Logické operátory III.

- `in`/`not in` – je/není součástí
- Metody `startswith, endswith, isalpha` …

In [8]:
```
'abc' not in 'abcdef'
```

Out[8]:
```
False
```

In [1]:
```
'Hello'.isalpha()
```

Out[1]:
```
True
```

Pořadí operací

1. Matematické: `**, *, /, +, -`
2. Porovnávací: `<, >, <=, >=, ==, !=, is, is not, in, not in`
3. `not`
4. `and`
5. `or`

In [11]:
```
9 + 3 > 11 and 5 != 6 or not True
```

Out[11]:
```
True
```

Bloky

- Jsou to části kódu, které se provádí v konkrétních situacích
- Začínají na předchozím řádku dvojtečkou
- Jsou odsazeny určeným počtem mezer/tabulátorů
- Bloky mohou být i vnořené, tj. *vnitřní blok* je součástí *vnějšího bloku*

In [12]:
```
if 1 == 2:
    print('1 == 2')                          # Blok 1
    print('This is strange')                 # Blok 1
elif 1 > 2:
    print('1 > 2')                           # Blok 2
    print('This is even stranger')           # Blok 2
    while 1 > 2:                             # Blok 2
        print('It is strange indeed')        # Blok 2, vnořený blok 3
        x = 5                                # Blok 2, vnořený blok 3
        y = x + 2                            # Blok 2, vnořený blok 3
    print('Blablabla')                       # Blok 2
else:
    pass                                     # Blok 4
print('This is the end')
```

```
This is the end
```

- Odsazení celého bloku musí být stejné, jinak čekejte `IndentationError`
- Doporučené odsazení jsou 4 mezery, VS Code automaticky nahrazuje tabulátor za 4 mezery

In [13]:
```
if 1 == 2:
    print('Hmmm...')
     print('I am confused')
```

```
  File "<ipython-input-13-d8dc889750bf>", line 3
    print('I am confused')
    ^
IndentationError: unexpected indent
```

In [14]:
```
if 1 == 2:
    print('Hmmm...')
 print('This is the end')
```

```
  File "<ipython-input-14-55048d5a040f>", line 3
    print('This is the end')
                           ^
IndentationError: unindent does not match any outer indentation level
```

- Bloky nesmí být prázdné

In [15]:
```
if 1 == 2:
else:
```

```
  File "<ipython-input-15-c11bbeb9214a>", line 2
    else:
       ^
IndentationError: expected an indented block
```

- Když chceme blok, ve kterém se nedělá nic, použijeme příkaz `pass`

In [16]:
```
if 1 == 2:
    pass
else:
    pass
```

Podmínky

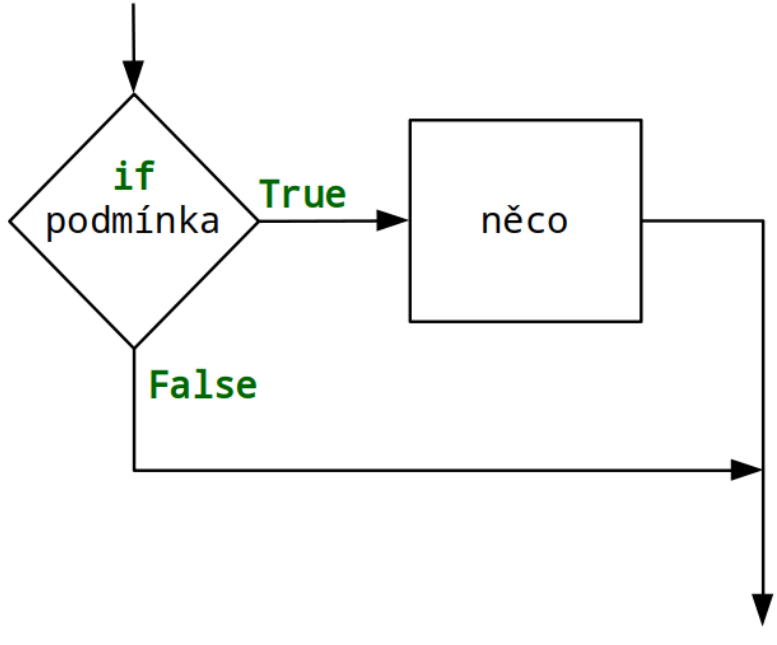
Na základě vyhodnocení logického výrazu se rozhodne, jestli se daný blok provede nebo nikoliv.

Podmínka `if`

- Syntaxe:

```
if podminka:
    udelej_neco
```

- Význam:



In [17]:
```
x = 15
if x > 10:
    print(f'{x} je větší než 10.')
print('Konec')
```

```
15 je větší než 10.
Konec
```

In [18]:
```
x = 3
if x > 10:
    print(f'{x} je větší než 10.')
print('Konec')
```

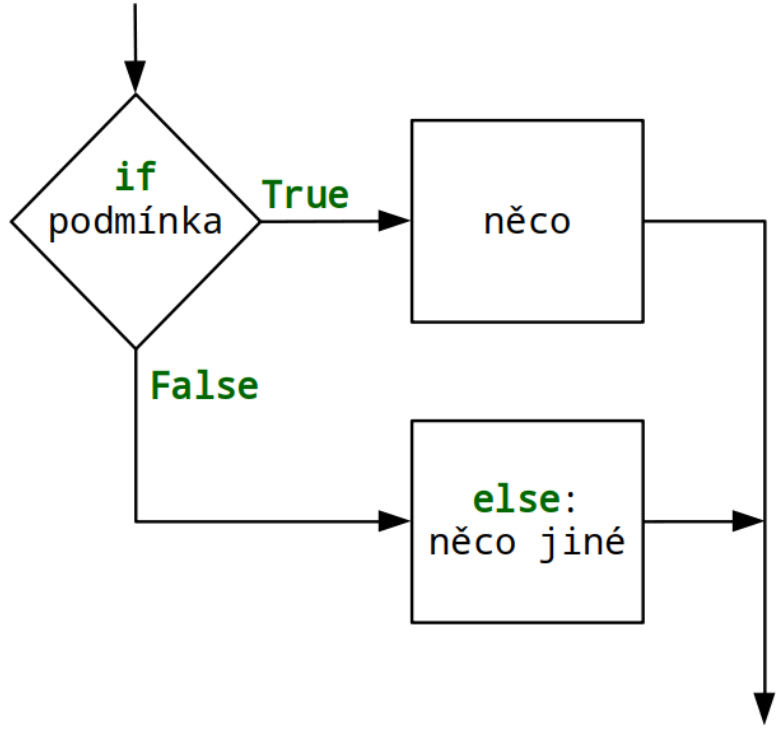
```
Konec
```

Podmínka `if ... else`

- Syntaxe:

```
if podminka:
    udelej_neco
else:
    udelej_neco_jine
```

- Význam:



In [19]:
```
x = 15
if x > 10:
    print(f'{x} je větší než 10.')
else:
    print(f'{x} je menší nebo rovno 10.')
print('Konec')
```

```
15 je větší než 10.
Konec
```

In [20]:
```
x = 3
if x > 10:
    print(f'{x} je větší než 10.')
else:
    print(f'{x} je menší nebo rovno 10.')
print('Konec')
```

```
3 je menší nebo rovno 10.
Konec
```

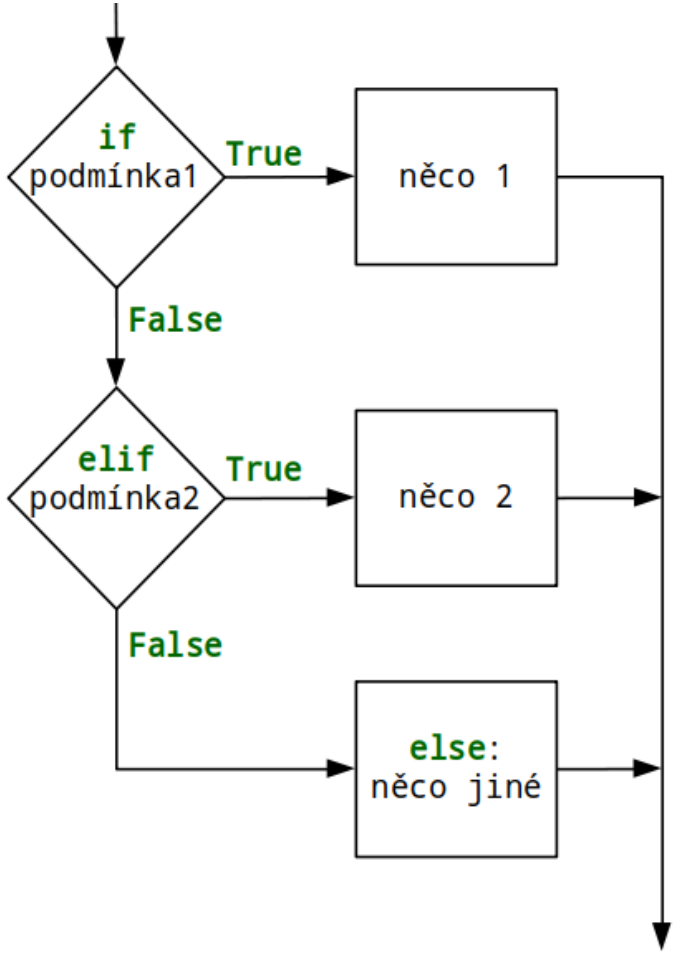
Podmínka `if ... elif ... else`

- Syntaxe:

```
if podminka1:
    udelej_neco_1
elif podminka2:
    udelej_neco_2
else:
    udelej_neco_jine
```

- Význam je stejný jako:

```
if podminka1:
    udelej_neco_1
else:
    if podminka2:
        udelej_neco_2
    else:
        udelej_neco_jine
```



- Můžeme mít i více podmínek s `elif`.
- Vždy se provede pouze jeden z těchto bloků – první, u kterého je splněna podmínka!

In [21]:
```
x = 15
if x > 10:
    print(f'{x} je větší než 10.')
elif x > 5:
    print(f'{x} je větší než 5.')
else:
    print(f'{x} je menší nebo rovno 5.')
print('Konec')
```

```
15 je větší než 10.
Konec
```

In [22]:
```
x = 8
if x > 10:
    print(f'{x} je větší než 10.')
elif x > 5:
    print(f'{x} je větší než 5.')
else:
    print(f'{x} je menší nebo rovno 5.')
print('Konec')
```

```
8 je větší než 5.
Konec
```

In [23]:
```
x = 3
if x > 10:
    print(f'{x} je větší než 10.')
elif x > 5:
    print(f'{x} je větší než 5.')
else:
    print(f'{x} je menší nebo rovno 5.')
print('Konec')
```

```
3 je menší nebo rovno 5.
Konec
```

Pozor, záleží na pořadí

- Provede se vždy pouze první blok, u kterého je splňena podmínka!

In [24]:
```
x = 15
if x > 5:
    print(f'{x} je větší než 5.')
elif x > 10:
    print(f'{x} je větší než 10.')
else:
    print(f'{x} je menší nebo rovno 5.')
print('Konec')
```

```
15 je větší než 5.
Konec
```

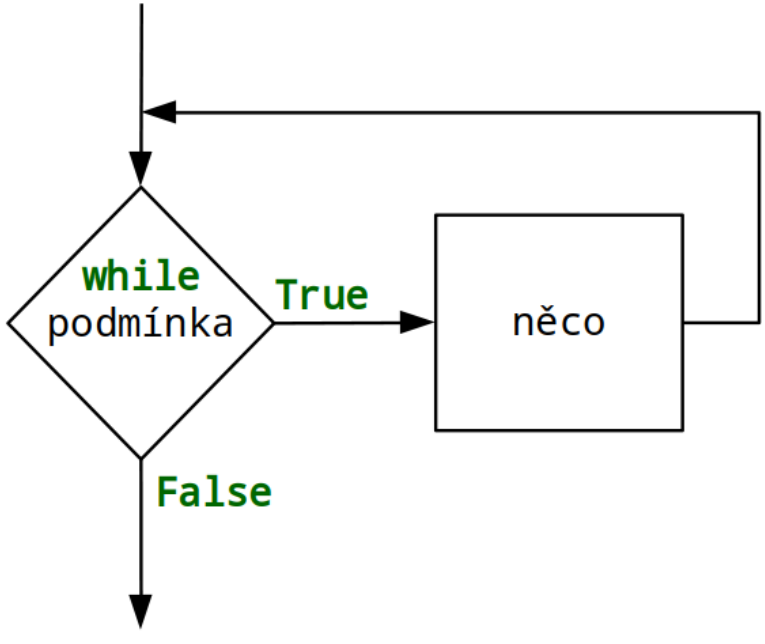
Cykly

Podobně jako podmínky se řídí logickým výrazem, který rozhoduje o spuštění příslušného bloku. Tyto bloky běží stále dokola, dokud je splněna podmínka.

Cyklus `while`

- Syntaxe:

```
while podminka:
    udelej_neco
```



- `udelej_neco` se bude opakovat až dokud nepřestane platit `podminka`.
- Jedno provedení `udelej_neco` se nazývá jedna *iterace*.

In [25]:
```
slovo = 'ozvěna'
while len(slovo) > 0:
    print(slovo)
    slovo = slovo[1:]
print('Konec')
```

```
ozvěna
zvěna
věna
ěna
na
a
Konec
```

- Někdy se neprovede ani jedna iterace (když podmínka už na začátku neplatí)

In [26]:
```
i = 0
while i == 3:
    print(i)
    i += 1
print('Konec')
```

```
Konec
```

- Pozor na zacyklení

```
i = 5
while i > 0:
    print(i)
print('Tento řádek se nikdy nevypíše')
```

Iterovatelné objekty (*iterables*)

- Objekty, které lze procházet po prvcích (iterovat)
- Příklady:
  - Řetězce – lze procházet znak po znaku

```
'Ahoj'  --->  'A', 'h', 'o', 'j'
```

  - Seznamy – lze procházet prvek po prvku

```
'Dobrý den všem'.split()  --->  'Dobrý', 'den', 'všem'
```

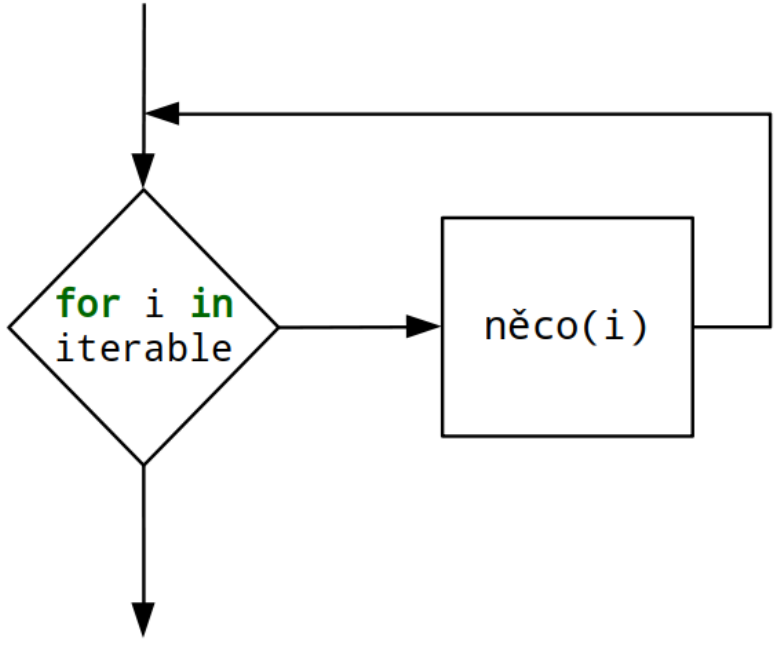
- Objekty, které nelze iterovat: `5, True, 3.14, print` …

Cyklus `for`

- Určen pro procházení iterovatelných objektů po prvcích
- Syntaxe:

```
for i in iterable:
    udelej_neco(i)
```

- Všechny prvky z objektu `iterable` se postupně dosadí do proměnné `i` a s každým se provede `udelej_neco(i)`


for i in
iterable
něco(i)

In [27]:
```
for znak in 'abcd':
    print(znak + '!')
```

```
a!
b!
c!
d!
```

In [28]:
```
for slovo in 'Toto je hezká věta'.split():
    print(slovo, len(slovo))
```

```
Toto 4
je 2
hezká 5
věta 4
```

Rozsah (*range*)

- Je objekt typu `range`, který reprezentuje posloupnost čísel
- Vytváří se pomocí funkce `range`
- Je to iterovatelný objekt, lze ho procházet číslo po čísle (pomocí cyklu `for`)

```
range(5)  --->  0, 1, 2, 3, 4
```

In [29]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Použití cyklu `for` s rozsahem

- Pokud chceme něco udělat přesně daný počet krát

In [30]:
```
for i in range(5):
    print('Ahoj')
```

```
Ahoj
Ahoj
Ahoj
Ahoj
Ahoj
```

- Pokud chceme něco udělat s každým prvkem číselné posloupnosti

In [31]:
```
for i in range(5):
    print(i, i * 'ha')
```

```
0
1 ha
2 haha
3 hahaha
4 hahahaha
```

Proměnné řídící běh cyklu se často pojmenovávají `i`, `j`, k..., pokud nemají jiný význam.

Funkce `range` – tři způsoby volání

- S jedním argumentem `end`

```
range(10)  --->  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

- Se dvěma argumenty `start`, `end`

```
range(3, 10)  --->   3, 4, 5, 6, 7, 8, 9
```

- Se třemi argumenty `start`, `end`, `step`

```
range(3, 10, 2)  --->  3, 5, 7, 9
```

- Všechny argumenty musí být typu `int`
- `start` je zahrnut v rozsahu, `end` nikoliv (podobně jako u indexování řetězců)

In [32]:
```
print(range(-5, 0))
```

```
range(-5, 0)
```

In [33]:
```
for i in range(-5, 0):
    print(i, end=' ')
```

```
-5 -4 -3 -2 -1
```

In [34]:
```
for i in range(10, 0):
    print(i, end=' ')
```

In [35]:
```
for i in range(10, 0, -1):
    print(i, end=' ')
```

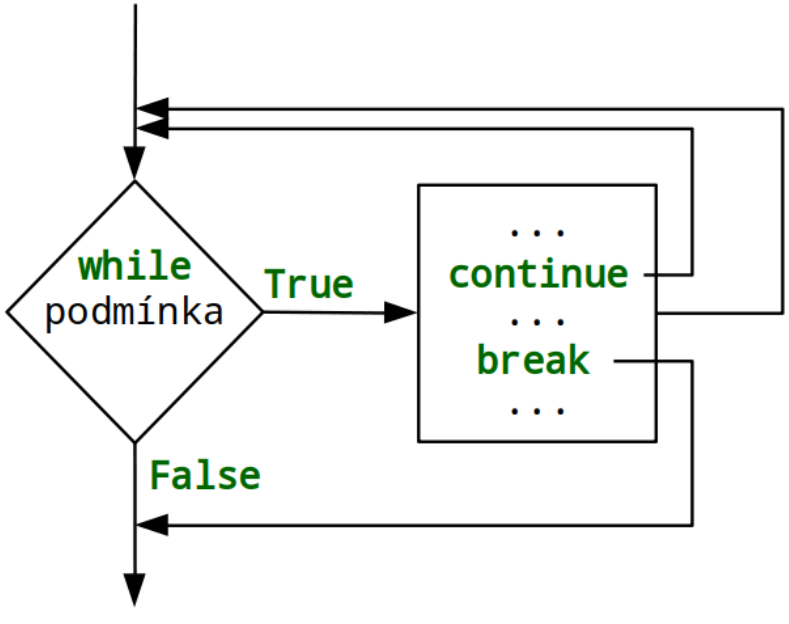
```
10 9 8 7 6 5 4 3 2 1
```

In [36]:
```
for i in range(0, -10, -1):
    print(i, end=' ')
```

```
0 -1 -2 -3 -4 -5 -6 -7 -8 -9
```

Řízení běhu cyklu pomocí `break` a `continue`

- `continue` ukončí vykonávání aktuální iterace a spustí následující iteraci
- `break` ukončí vykonávání celého cyklu



In [37]:
```
for i in range(6):
    if i % 2 == 1:
        continue
    print(i)
print('Konec')
```

```
0
2
4
Konec
```

In [38]:
```
for i in range(6):
    if i % 2 == 1:
        break
    print(i)
print('Konec')
```

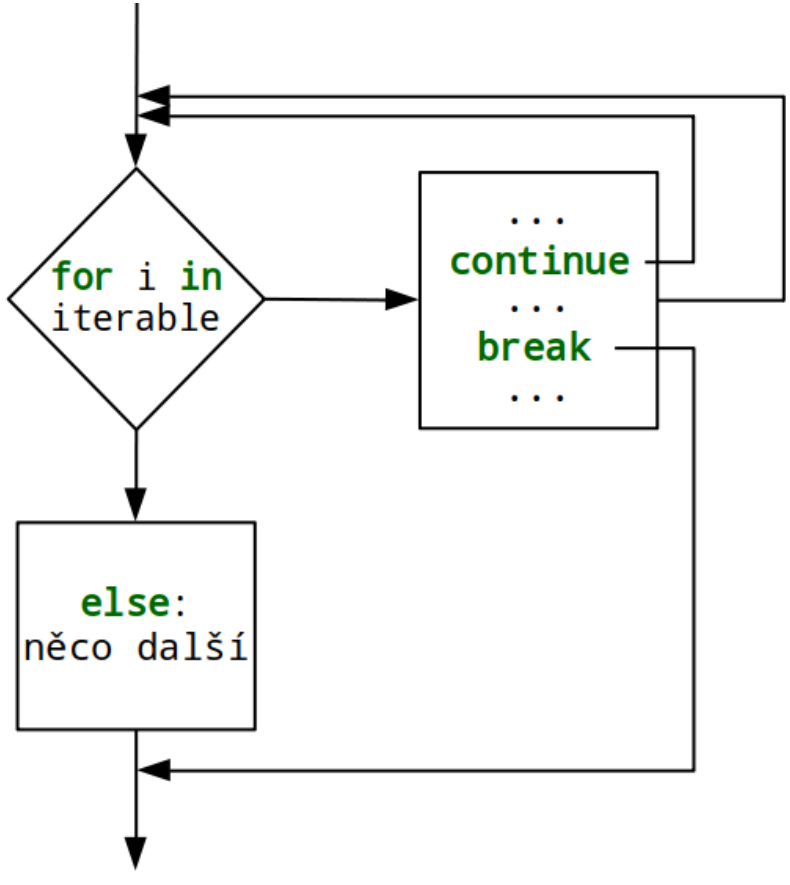
```
0
Konec
```

Řízení cyklu `for` pomocí `break` a `else`

- Syntaxe:

```
for prvek in iterable:
    delej_neco ...
    break
    ...
else:
    udelej_neco_dalsi
```

- Pokud cyklus přeběhl celý (nebyl ukončen pomocí `break`), spustí se blok `else`


for i in
iterable
...
continue
...
break
...
else:
něco další

In [39]:
```
for i in range(30, 35):
    print(i)
    if i % 11 == 0:
        print(f'Nalezen násobek jedenácti: {i}')
        break
else:
    print('Žádné číslo dělitelné 11 nenalezeno.')
```

```
30
31
32
33
Nalezen násobek jedenácti: 33
```

In [40]:
```
for i in range(60, 65):
    print(i)
    if i % 11 == 0:
        print(f'Nalezen násobek jedenácti: {i}')
        break
else:
    print('Žádné číslo dělitelné 11 nenalezeno.')
```

```
60
61
62
63
64
Žádné číslo dělitelné 11 nenalezeno.
```

Cvičení 1

- Co vypíše tento program?

In [ ]:
```
slovo = 'ukazováček'
if len(slovo) >= 10:
    if 'ukaz' in slovo:
        print(1)
    else:
        print(2)
    print(3)
elif slovo.startswith('ukaz'):
    if slovo.isalpha():
        print(4)
elif slovo.endswith('ukaz'):
    print(5)
else:
    print(6)
```

- Řešení:

In [2]:
```
slovo = 'ukazováček'
if len(slovo) >= 10:
    if 'ukaz' in slovo:
        print(1)
    else:
        print(2)
    print(3)
elif slovo.startswith('ukaz'):
    if slovo.isalpha():
        print(4)
elif slovo.endswith('ukaz'):
    print(5)
else:
    print(6)
```

```
1
3
```

- Co by se vypsalo, kdyby v proměnné `slovo` bylo 'ukazatel', 'palec', 'poukaz', nebo 'spisovatel'?

Cvičení 2

- Co vypíše tento program?

In [ ]:
```
x = 0

for i in range(10):
    if i % 2 == 0:
        x += 2
    else:
        x -= 1

print(x)
```

- Řešení:
  - Za každé sudé *i* (0, 2, 4, 6, 8) se k *x* přičítá 2, za každé liché *i* (1, 3, 5, 7, 9) se od *x* odečítá 1.

In [3]:
```
x = 0

for i in range(10):
    if i % 2 == 0:
        x += 2
    else:
        x -= 1

print(x)
```

```
5
```

Cvičení 3

- Co vypíše tento program?

In [ ]:
```
x = 5
y = 1

while x > 0:
    y *= x
    x -= 1

print(x, y)
```

- Řešení:
  - *y* se postupně vynásobí 5, 4, 3, 2 a 1, tj. program nám spočítá hodnotu 5 faktoriál (5!).

In [4]:
```
x = 5
y = 1

while x > 0:
    y *= x
    x -= 1

print(x, y)
```

```
0 120
```