

# C2184 Úvod do programování v Pythonu (2021)

## 4. Podmínky a cykly

### Opakování: Logické výrazy

- Vyhodnocením logických výrazů získáme vždy výsledek typu bool
  - True
  - False
- == je rovno, != není rovno
- > je větší, >= je větší rovno
- < je menší, <= je menší rovno

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

```
[1]: True
```

```
[2]: 9 >= 9
```

```
[2]: True
```

```
[3]: x = 5  
     y = 10  
     x < y
```

```
[3]: True
```

- in / not in - je / není součástí
- Metody startswith, endswith, isalpha ...

```
[4]: 'abc' not in 'abcdef'
```

```
[4]: False
```

```
[5]: 'abc'.startswith('a')
```

```
[5]: True
```

```
[6]: 'Hello'.isalpha()
```

[6]: True

- and – “a” (logický součin)
- or – “nebo” (logický součet)
- not – “neplatí, že” (negace)

Pozor, nezaměňovat za & |

```
[7]: 2 + 2 == 5 and 9 > 1
```

[7]: False

```
[8]: False or False
```

[8]: False

```
[9]: not False
```

[9]: True

### Priorita operací

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

```
[10]: 9 + 3 > 11 and 5 != 6 or not True
```

[10]: 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*

```
[ ]: 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')

```

- Odsazení celého bloku musí být stejné, jinak čekaete `IndentationError`
- Doporučené odsazení jsou 4 mezery, VS Code automaticky nahrazuje tabulátor za 4 mezery
- Odsazení více označených řádků lze zvětšit klávesou Tab, zmenšit Shift+Tab

```

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

```

```

File "/tmp/ipykernel_36935/1006172414.py", line 3
    print('I am confused')
    ^

```

`IndentationError: unexpected indent`

```

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

```

```

File "<tokenize>", line 3
    print('This is the end')
    ^

```

`IndentationError: unindent does not match any outer indentation level`

- Bloky nesmí být prázdné

```

[14]: if 1 == 2:
        else:

```

```
File "/tmp/ipykernel_36935/1300489546.py", line 2
else:
^
IndentationError: expected an indented block
```

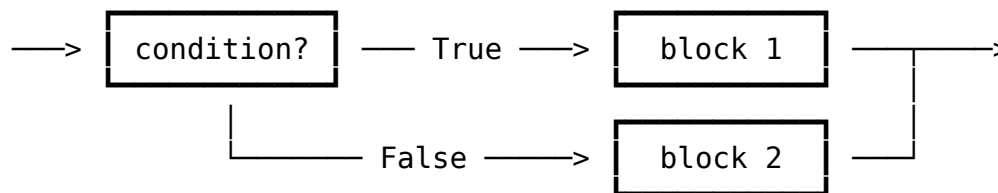
- Když chceme blok, ve kterém se nedělá nic, použijeme příkaz `pass`

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

## Podmíněné příkazy (*conditional statements*)

- Na základě výsledku logického výrazu se rozhodne, které bloky provedou a které ne.

### Podmíněný příkaz `if ... else`



- Syntaxe:

```
if condition:
    block1

else:
    block2
```

```
[16]: x = 8
      if x > 0:
          print(f'{x} je kladné.')
      else:
          print(f'{x} je záporné nebo nula.')
      print('Konec')
```

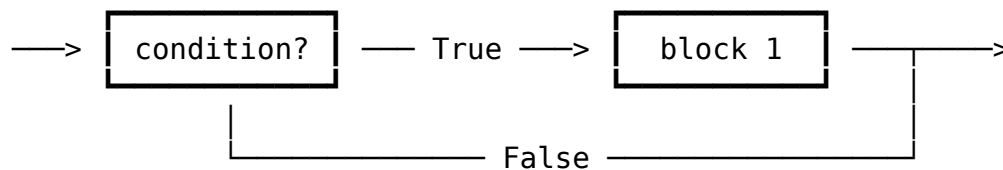
8 je kladné.  
Konec

```
[17]: x = -3
      if x > 0:
          print(f'{x} je kladné.')
      else:
          print(f'{x} je záporné nebo nula.')
      print('Konec')
```

-3 je záporné nebo nula.  
Konec

## Podmíněný příkaz if

- Můžeme vynechat blok else



- Syntaxe:

```
if condition:
```

```
    block1
```

- Význam je přesně stejný jako:

```
if condition1:
```

```
    block1
```

```
else:
```

```
    pass
```

```
[18]: x = 8
      if x > 0:
          print(f'{x} je kladné.')
      print('Konec')
```

8 je kladné.  
Konec

```
[19]: x = -3
      if x > 0:
          print(f'{x} je kladné.')
      print('Konec')
```

Konec

## Podmíněné příkazy lze do sebe libovolně vnořovat

```
[20]: x = 5
      if x > 0:
          if x % 2 == 0:
              print(f'{x} je kladné sudé číslo.')
          else:
              print(f'{x} je kladné liché číslo.')
          print('Každopádně je kladné.')
      else:
          print(f'{x} je záporné číslo.')
          if x == 0:
              print(f'Ups, vlastně není kladné ani záporné.')
      print('Konec')
```

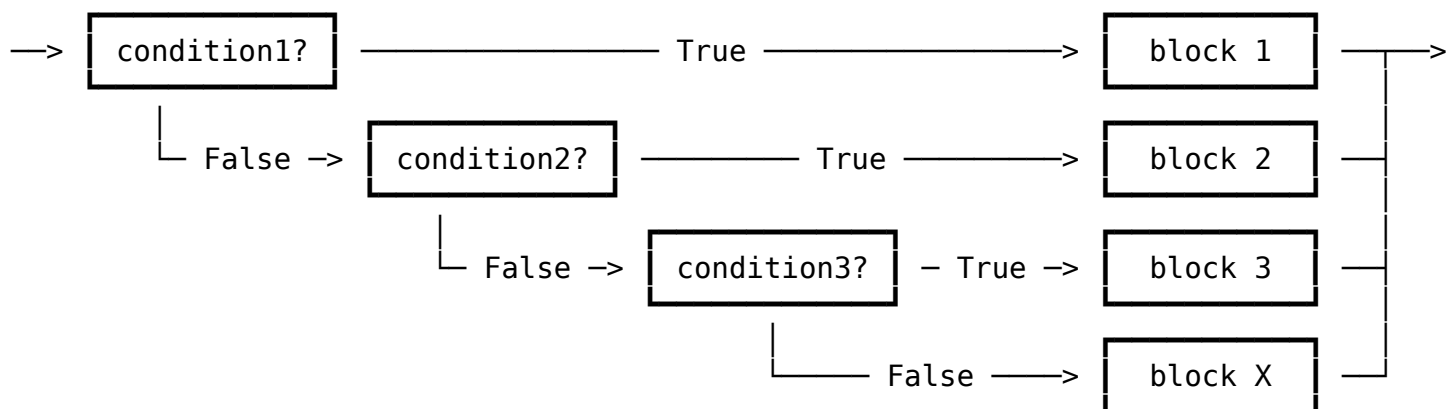
5 je kladné liché číslo.

Každopádně je kladné.

Konec

## Podmíněný příkaz `if ... elif ... else`

- Pokud neplatí první podmínka, testujeme další



- Syntaxe:

```
if condition1:
    block1
elif condition2:
    block2
elif condition3:
    block3
...
else:
    blockX
```

- Význam je přesně stejný jako:

```

if condition1:
    block1
else:
    if condition2:
        block2
    else:
        if condition3:
            block3
            ...
        else:
            blockX

```

- První blok je vždycky if.
- Pak můžeme mít libovolný počet bloků elif.
- Na konci může a nemusí být blok else.

### Pamatujte:

- Za if a elif je vždy podmínka, else je vždy bez podmínky.
- Vždy se provede POUZE JEDEN z bloků – první, u kterého je splněna podmínka.

```

[21]: x = 8
      if x > 0:
          print(f'{x} je kladné.')
      elif x < 0:
          print(f'{x} je záporné.')
      else:
          print(f'{x} není kladné ani záporné.')
      print('Konec')

```

8 je kladné.  
Konec

```

[22]: x = -3
      if x > 0:
          print(f'{x} je kladné.')
      elif x < 0:
          print(f'{x} je záporné.')
      else:
          print(f'{x} není kladné ani záporné.')
      print('Konec')

```

-3 je záporné.  
Konec

```
[23]: x = 0
      if x > 0:
          print(f'{x} je kladné.')
      elif x < 0:
          print(f'{x} je záporné.')
      else:
          print(f'{x} není kladné ani záporné.')
      print('Konec')
```

0 není kladné ani záporné.  
Konec

### Pozor, záleží na pořadí

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

```
[24]: x = 15
      if x > 0:
          print(f'{x} je kladné.')
      elif x > 10:
          print(f'{x} je větší než 10.')
      else:
          print(f'{x} je záporné.')
      print('Konec')
```

15 je kladné.  
Konec

### Další if - další nezávislý podmíněný příkaz

```
[25]: x = 15
      if x > 0:
          print(f'{x} je kladné.')
      if x > 10:
          print(f'{x} je větší než 10.')
      else:
          print(f'{x} je menší než 10.')
      print('Konec')
```

15 je kladné.  
15 je větší než 10.  
Konec

### Podmíněné výrazy (*conditional expressions*)

result = value1 **if** condition **else** value2

je zkratka pro



```
if condition:
    result = value1
else:
    result = value2
```

```
[26]: print('Yep' if 2 + 2 == 4 else 'Nope')
```

Yep

```
[27]: print('Yep' if 2 + 2 == 5 else 'Nope')
```

Nope

### Otázky:

Která z čísel 1-6 vypíše následující program?

- A) 1 3 4
- B) 1 3
- C) 1 4
- D) 4

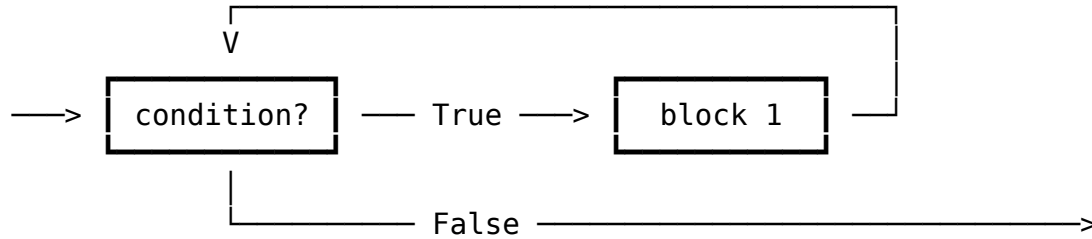
```
[ ]: word = 'ukazováček'
if len(word) >= 10:
    if 'ukaz' in word:
        print(1)
    else:
        print(2)
    print(3)
elif word.startswith('ukaz'):
    if word.isalpha():
        print(4)
elif word.endswith('ukaz'):
    print(5)
else:
    print(6)
```

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

### Cykly (loops)

- Podobně jako podmíněné příkazy se řídí logickým výrazem, který rozhoduje o spuštění příslušného bloku.
- Tento blok ale běží stále dokola, dokud je splněna podmínka.

## Cyklus while



- Syntaxe:

```
while condition:  
    block1
```

- Jedno provedení bloku se nazývá jedna *iterace*.

```
[28]: word = 'ozvěna'  
while len(word) > 0:  
    print(word)  
    word = word[1:]  
print('Konec')
```

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

```
[29]: x = 1  
while x < 200:  
    x *= 2  
    print(x)
```

```
2  
4  
8  
16  
32  
64  
128  
256
```


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

```
[30]: i = 10
while i < 5:
    print(i)
    i += 1
print('Konec')
```

Konec

- Pozor na zacyklení

```
[ ]: a = 5
while a > 0:    # Tento cyklus se bude opakovat donekonečna!
    b = a
print('Konec') # Tento řádek se nikdy nevypíše.
```

- Jak zastavit zacyklený program?
  - V terminálu: Ctrl+C
  - V Jupyter Notebooku: tlačítko  (Interrupt Kernel / Stop Cell Execution)

## Otázky

Co vypíše následující program?

- A) 0 0
- B) 4 5
- C) 0 120
- D) 0 3125

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

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

print(x, y)
```

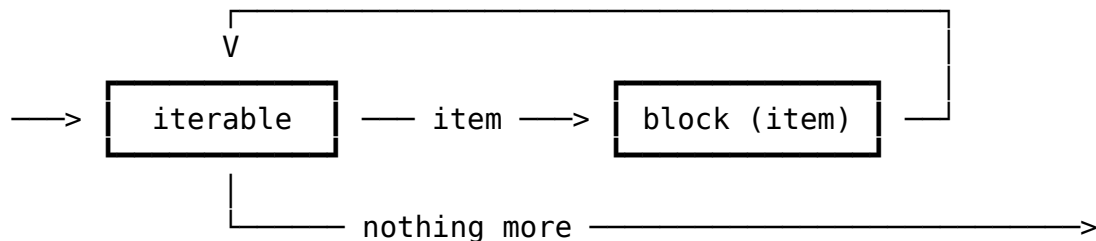
## Iterovatelné objekty (*iterables*)

- Objekty, které lze procházet po prvcích (iterovat)
- Iterovatelné objekty:
  - Řetězce - lze procházet znak po znaku
    - 'Ahoj' —> 'A', 'h', 'o', 'j'
    - 'x' —> 'x'

- Seznamy - lze procházet prvek po prvku  
   'Dobrý den všem'.split() —> 'Dobrý', 'den', 'všem'
- Rozsahy - lze procházet číslo po čísle  
   range(0, 10) —> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9  
   ...
- Objekty, které nelze iterovat:
  - 5, True, 3.14, print, None ...

## Cyklus for

- Určen pro procházení iterovatelných objektů po prvcích



- Všechny prvky z objektu iterable se postupně dosadí do proměnné item a s každým se provede blok
- Syntaxe:

```
for item in iterable:
    block
```

```
[31]: for char in 'abcd':
      print(char + '!')
```

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

```
[32]: for word in 'Toto je hezká věta'.split():
      print(word, len(word))
```

```
Toto 4
je 2
hezká 5
věta 4
```

## Rozsah (*range*)

- Rozsah 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(0, 5)` —> 0, 1, 2, 3, 4

```
[33]: print(range(0, 5))
```

`range(0, 5)`

```
[34]: for i in range(0, 5):  
      print(i)
```

0  
1  
2  
3  
4

## 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)

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

-5 -4 -3 -2 -1

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

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

10 9 8 7 6 5 4 3 2 1



```
[42]: letters = 'ABC'
      for letter in letters:
          print(letter)
```

A  
B  
C

- Když potřebujeme i index i prvek, použijeme funkci enumerate:

```
[43]: letters = 'ABC'
      for i, letter in enumerate(letters):
          print(f'{i}. {letter}')
```

0. A  
1. B  
2. C

```
[44]: letters = 'ABC'
      for i, letter in enumerate(letters, start=1):
          print(f'{i}. {letter}')
```

1. A  
2. B  
3. C

- Pythoní cyklus for spíše odpovídá cyklu foreach v některých jiných jazycích.

## Otázky

Co vypíše následující program?

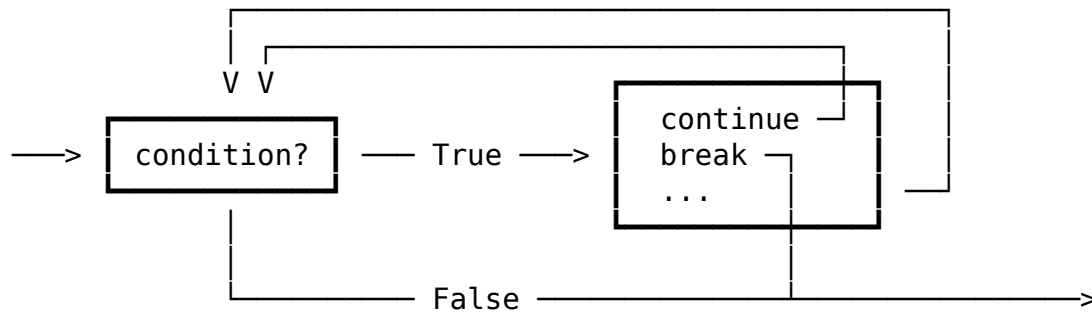
- A) 5
- B) 0
- C) 10
- D) 45

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

## Řízení běhu cyklu pomocí continue a break

- continue ukončí vykonávání aktuální iterace a spustí následující iteraci

- break ukončí vykonávání celého cyklu



- Syntaxe:

**while** condition:

```
...
continue
...
break
...
```

**for** item **in** iterable:

```
...
continue
...
break
...
```

```
[45]: x = 1
while x < 200:
    x *= 2
    print(x)
```

```
2
4
8
16
32
64
128
256
```

```
[46]: x = 1
while x < 200:
    x *= 2
    if x == 32:
        print('----')
        continue
    print(x)
```

```
2
4
8
16
----
64
```

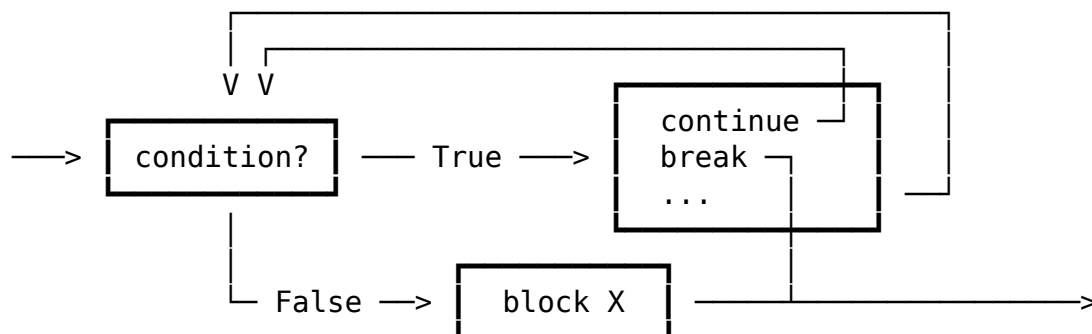


128  
256

```
[47]: x = 1
while x < 200:
    x *= 2
    if x == 32:
        print('----')
        break
    print(x)
```

2  
4  
8  
16  
----

### Řízení cyklu pomocí break a else



- Syntaxe:

```
while condition:
    ...
    break
    ...
```

else:

blockX

```
for item in iterable:
    ...
    break
    ...
```

else:

blockX

- Pouze pokud cyklus proběhl celý (nebyl ukončen pomocí break), spustí se blok else
- Klíčová slova continue, break a else fungují stejně pro cyklus while a cyklus for

```
[48]: 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
```

```
[49]: 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.
```