Programování: základní konstrukce

IB111 Úvod do programování skrze Python

2015

Rozcvička

1	2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17	18	19

Základní konstrukce

- proměnné, operace
- řízení toku výpočtu:
  - podmiňovací příkaz (if-else)
  - cykly (for, while)
- funkce

O přednášce

- důraz na princip použití (k čemu to je), ilustrace použití, přemýšlení o problému
- ilustrace na příkladech
- syntax (zápis) jen zběžně, zdaleka ne vše z jazyka Python
- syntax je však potřeba také umět!
  - cvičení
  - samostudium, např. `howto.py.cz`

Příklad

Problém: vypočítat výšku mostu na základě času pádu koule

vstup: čas
výstup: výška

```
t = input()
h = 0.5 * 10 * t * t
print h
```

Proměnné

- udržují hodnotu
- udržovaná hodnota se může měnit – proto *proměnné*
- typy:
  - číselné: int, float, ...
  - pravdivostní hodnota (bool)
  - řetězec (string)
  - seznam / pole
  - slovník
  - ...

Výrazy a operace

- výrazy: kombinace proměnných a konstant pomocí operátorů
- operace:
  - aritmetické: sčítání, násobení, ...
  - logické: and, or, not, ...
  - zřetězení řetězců
  - ...
- preference operátorů, pořadí vyhodnocování

Proměnné a výrazy: příklady

```
x = 13
y = x % 4     #  dělení se zbytkem
y = y + 1
y += 1

a = (x==3) and (y==2)
b = a or not a

s = "petr"
t = "klic"
u = s + t

z = x + s     # chyba: nelze sčítat int a string
```

Zápis v Pythonu

- přiřazení $=$
- test na rovnost $==$
- většina operací „intuitivní“: $+$, -, *, $/$, and, or, ...
- umocňování: **
- dělení se zbytkem: %
- zkrácený zápis: „y $+=$ 5“ znamená „y $=$ y $+$ 5“

Typy v Pythonu

- „deklarace“ proměnné: první přiřazení hodnoty
- dynamické implicitní typování
  - typ se určuje automaticky
  - typ proměnné se může měnit
  - rozdíl oproti statickému explicitnímu typování (většina kompilovaných jazyků jako C, Pascal): `int x;`
- zjištění typu: funkce `type`

Explicitní přetypování, dělení

- explicitní přetypování (`x = float(3); s = str(158)`)
- významné např. při dělení
  - $3 \,/\, 2 = 1$
  - $\text{float}(3) \,/\, 2 = 1.5$
  - $3.0 \,/\, 2 = 1.5$

Pořadí vyhodnocování

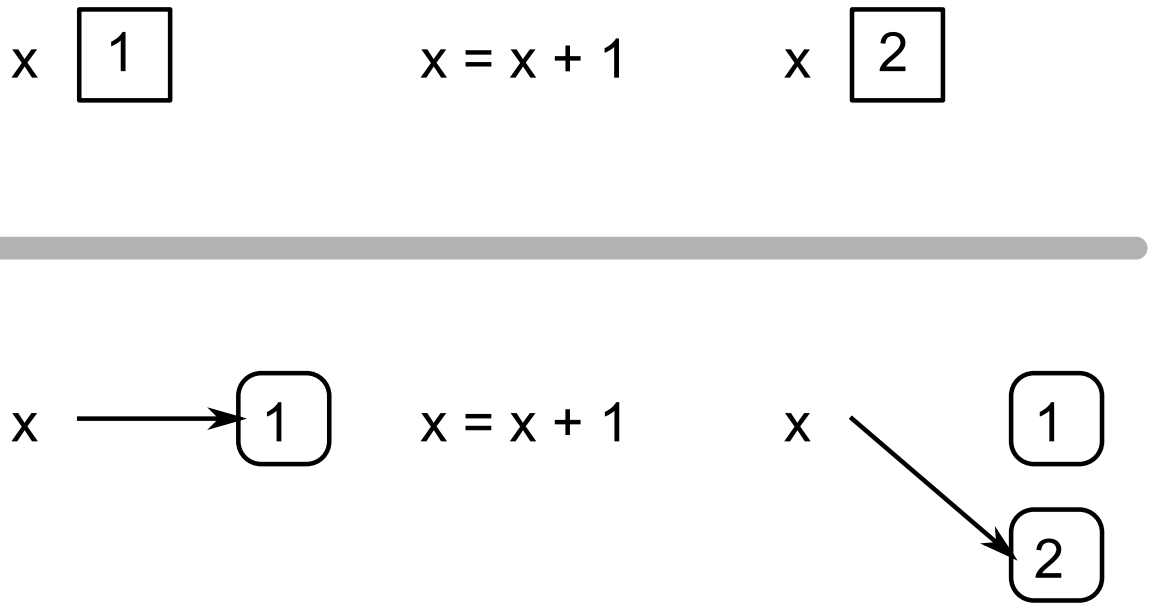
```
3 + 2**3 < 15 or "pes" == "kos"
```

- pořadí vyhodnocování vesměs intuitivní
- pokud na pochybách:
  - konzultujte dokumentaci
  - závorkujte
- zkrácené vyhodnocování: `1+1 == 2 or x == 3`

Operace, pořadí vyhodnocování

Operator	Description
`lambda`	Lambda expression
`if – else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in, not in, is, is not, <, <=, >, >=, <>, !=, ==`	Comparisons, including membership tests and identity tests,
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<, >>`	Shifts
`+, -`	Addition and subtraction
`*, /, //, %`	Multiplication, division, remainder [8]
`+x, -x, ~x`	Positive, negative, bitwise NOT
`**`	Exponentiation [9]
`x[index], x[index:index], x(arguments...), x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...), [expressions...], {key:datum...}, `expressions...``	Binding or tuple display, list display, dictionary display, string conversion

Proměnné a paměť



Výstup

- základní výpis: `print x`
- bez odřádkování: `print x,`
- další možnosti: `sys.write`, `.format`, `.rjust` ...
- (rozdíl oproti Python 3)

Podmínky: příklad

Příklad: počítání vstupného

vstup: věk
výstup: cena vstupenky

```
vek = input()
if vek < 18:
    cena = 50
else:
    cena = 100
print cena
```

Podmíněný příkaz

```
if <podmínka>: příkaz1
else: příkaz2
```

- podle toho, zda platí podmínka, se provede jedna z větví
- podmínka – typicky výraz nad proměnnými
- else větev nepovinná
- vícenásobné větvení: if - elif - ... - else
  (switch v jiných jazycích)

Blok kódu

- co když chci provést v podmíněné větvi více příkazů?
- blok kódu
  - Python: vyznačeno odsazením
  - jiné jazyky: složené závorky { }, begin-end

Podmíněný příkaz: příklad

```
if x < 0:
    x = 0
    print 'Zaporne vynulovano'
elif x == 0:
    print 'Nula'
elif x == 1:
    print 'Jedna'
else:
    y = x * x
    print 'Moc'
```

Cykly: příklady

- vstupné za celou rodinu
- výpis posloupnosti čísel
- výpočet faktoriálu
- převod čísla na binární zápis

Cyklus

- opakované provádění sekvence příkazů
- známý počet opakování cyklu:
  - příkaz `for`
- neznámý počet opakování cyklu:
  - příkaz `while`
  - opakuj dokud není splněna podmínka

For, range

```
for x in range(n):
    příkazy
```

- provede příkazy pro všechny hodnoty `x` ze zadaného intervalu
- `range(a, b)` – interval od `a` do `b-1`
- `range(n)` – interval od `0` do `n-1` (tj. `n` opakování)
- `for`/`range` lze použít i obecněji (nejen intervaly) – viz později/samostudium

Faktoriál pomocí for cyklu

- Co to faktoriál? K čemu se používá?
- Kolik je „`5!`“?
- Jak vypočítat „`n!`“ (`n` je vstup od uživatele)?

Faktoriál pomocí for cyklu

```
n = input()
f = 1

for i in range(1,n+1):
    f = f * i

print f
```

První posloupnost z úvodní přednášky

```
1 0 0 2 8 22 52 114 240 494
```

```
for i in range(n):
        print 2**i - 2*i,
```

Zanořování

- řídicí struktury můžeme zanořovat, např.:
  - podmínka uvnitř cyklu
  - cyklus uvnitř cyklu
  - ...
- libovolný počet zanoření (ale ...)

Rozcvička

1	2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17	18	19

Rozcvička programem

```
n = 10
soucet = 0

for i in range(1,n+1):
    for j in range(n):
        print i+j,
        soucet += i+j
    print

print "Soucet je", soucet
```

Rozcvička programem – hezčí formátování

```
for i in range(1,n+1):
    for j in range(n):
        print str(i+j).rjust(2),
    print
```

While cyklus

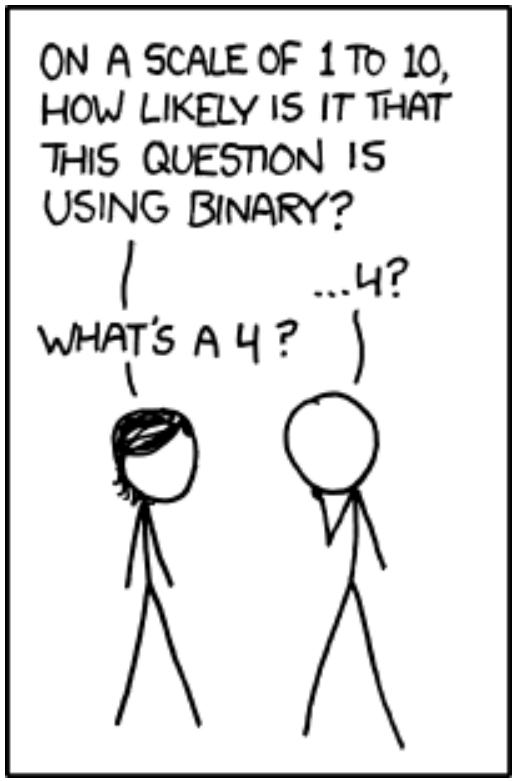
```
while <podmínka>:
    příkazy
```

- provádí `příkazy` dokud platí podmínka
- může se stát:
  - neprovede `příkazy` ani jednou
  - provádí `příkazy` do nekonečna (nikdy neskončí) – to většinou znamená chybu v programu
- napište výpočet faktoriálu pomocí while cyklu

Faktoriál pomocí while cyklu

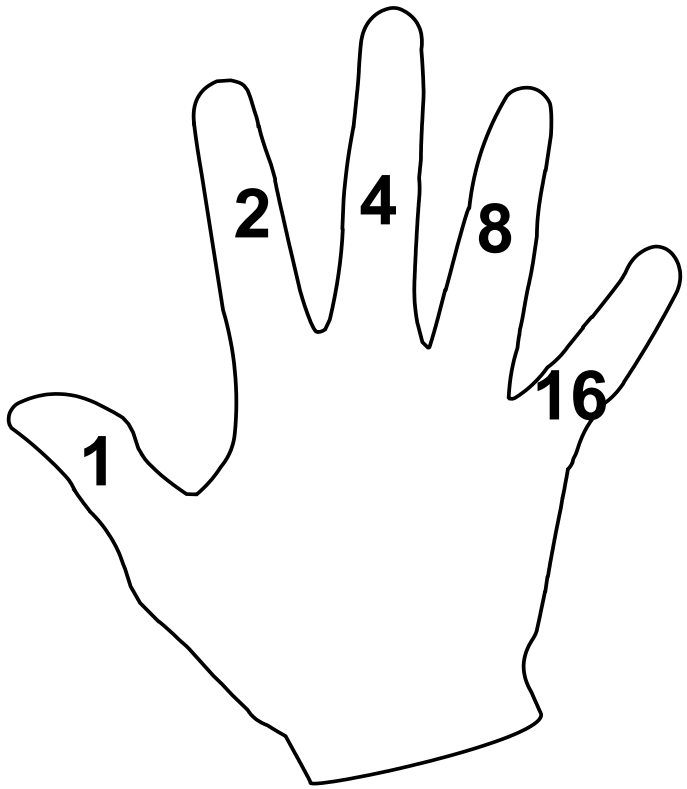
```
n = input()
f = 1
while n > 0:
    f = f * n
    n = n - 1

print f
```

Binární soustava



https://xkcd.com/953/

Binární soustava: počítání na prstech



http://www.khanacademy.org/math/vi-hart/v/binary-hand-dance

Příklad: převod na binární zápis

Problém: převodník z desítkové na binární soustavu
vstup: číslo v desítkové soustavě
výstup: číslo v binární soustavě

- Jak převedeme „22“ na binární zápis?
- Jak převedeme obecné číslo na binární zápis?

Převod na binární zápis

```
n = input()
vystup = ""
while n > 0:
    if n % 2 == 0:
        vystup = "0" + vystup
    else:
        vystup = "1" + vystup
    n = n / 2
print vystup
```

Převod na binární zápis – průběh výpočtu

```
n =  22    vystup =
n =  11    vystup =  0
n =  5     vystup =  10
n =  2     vystup =  110
n =  1     vystup =  0110
n =  0     vystup =  10110
```

Funkce

Programy nepíšeme jako jeden dlouhý „štrúdl“, ale dělíme je do funkcí.

Proč?

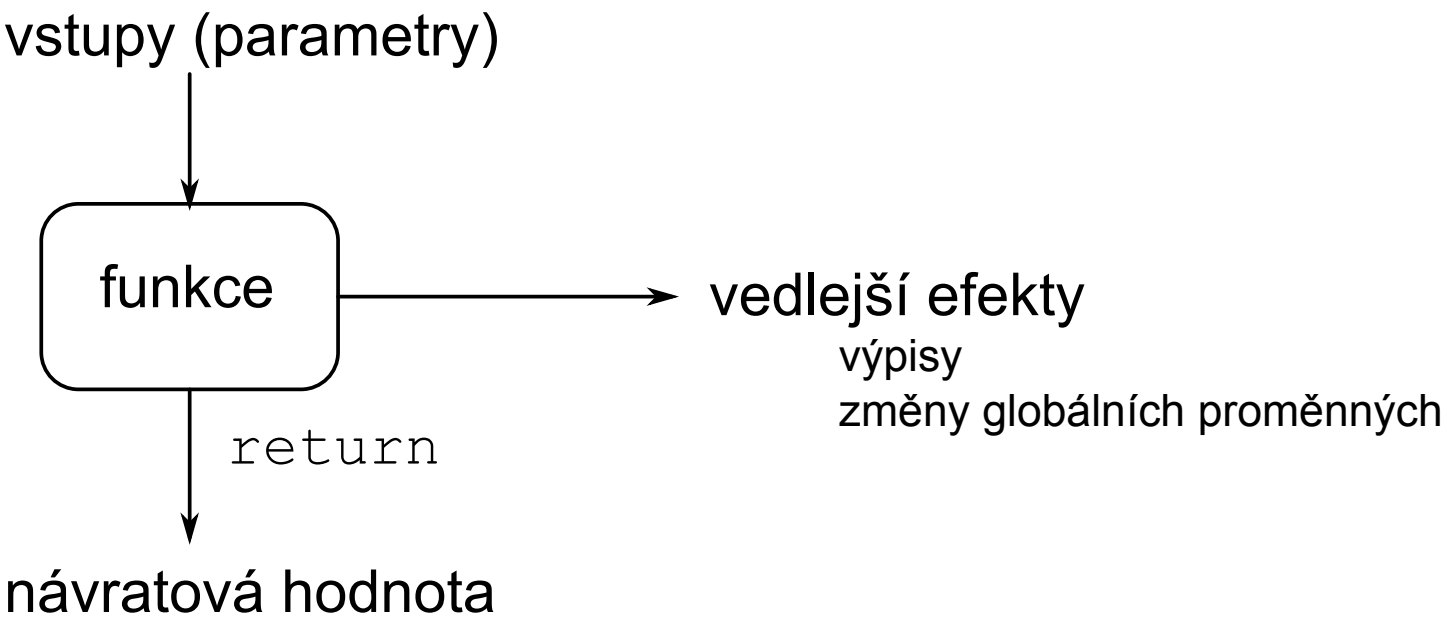
Funkce

Programy nepíšeme jako jeden dlouhý „štrúdl“, ale dělíme je do funkcí.

Proč?

- opakované provádění stejného (velmi podobného) kódu na různých místech algoritmu
- modularita (viz Lego kostky), znovupoužitelnost
- snazší uvažování o problému, dělba práce

Funkce


vstupy (parametry)
funkce
return
návratová hodnota
vedlejší efekty
výpisy
změny globálních proměnných

Funkce

- vstup: parametry funkce
- výstup: návratová hodnota
- proměnné v rámci funkce:
  - lokální: dosažitelné pouze v rámci funkce
  - globální: dosažitelné všude, minimalizovat použití

Funkce pro převod na binární zápis

```
def binarni_zapis(n):
    vystup = ""
    while n > 0:
        if n % 2 == 0:
            vystup = "0" + vystup
        else:
            vystup = "1" + vystup
        n = n / 2
    return vystup
```

Vnořené volání funkcí

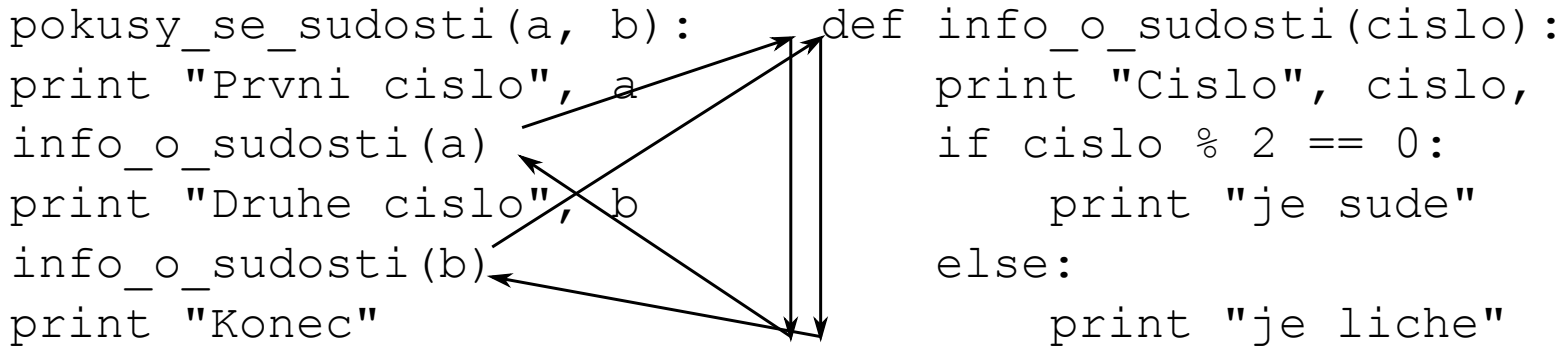
- funkce mohou volat další funkce
- po dokončení vnořené funkce se interpret vrací a pokračuje
- rekurze: volání sebe sama, cyklické volání funkcí (podrobněji později)

Vnořené volání: jednoduchý příklad

```
def info_o_sudosti(cislo):
    print "Cislo", cislo,
    if cislo % 2 == 0:
        print "je sude"
    else:
        print "je liche"

def pokusy_se_sudosti(a, b):
    print "Prvni cislo", a
    info_o_sudosti(a)
    print "Druhe cislo", b
    info_o_sudosti(b)
    print "Konec"

pokusy_se_sudosti(3, 18)
```

Vnořené volání – ilustrace

```
pokusy_se_sudosti(3,18)
```



```
def pokusy_se_sudosti(a, b):
    print "Prvni cislo", a
    info_o_sudosti(a)
    print "Druhe cislo", b
    info_o_sudosti(b)
    print "Konec"
```

```
def info_o_sudosti(cislo):
    print "Cislo", cislo,
    if cislo % 2 == 0:
        print "je sude"
    else:
        print "je liche"
```

Rozcvička programem II

Experimentální otestování hypotézy o třetí mocnině

```
def tabulka_soucet(n):
    soucet = 0
    for i in range(1,n+1):
        for j in range(n):
            soucet += i+j
    return soucet

def vypis_souctu(kolik):
    for n in range(1, kolik+1):
        print n, n**3, tabulka_soucet(n)
```

Druhá posloupnost z úvodní přednášky

0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4 1 2 2 3 2 3 3 4 2
3 3 4 3 4 ...

Druhá posloupnost z úvodní přednášky

0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4 1 2 2 3 2 3 3 4 2
3 3 4 3 4 ...

Počet jedniček v bitovém zápisu.

Jak vypsat programem?

Druhá posloupnost z úvodní přednášky

```
def pocet_jednicek(n):
    pocet = 0
    while n > 0:
        if n % 2 == 1: pocet += 1
        n = n / 2
    return pocet

def posloupnost2(n):
    for i in range(n):
        print pocet_jednicek(i),
```

Funkce: Python speciality

```
def test(x, y = 3):
    print "X =", x
    print "Y =", y
```

- defaultní hodnoty proměnných
- volání pomocí jmen proměnných
- `test` můžeme volat např.:
  - `test(2,8)`
  - `test(1)`
  - `test(y=5, x=4)`
- (dále též libovolný počet argumentů a další speciality)

Programátorská kultura

- psát smysluplné komentáře, dokumentační řetězce (viz později)
- dávat proměnným a funkcím smysluplná jména
- funkce by měly být krátké:
  - „jedna myšlenka“
  - max na jednu obrazovku
  - jen pár úrovní zanoření
- příliš dlouhá funkce – rozdělit na menší
- neopakovat se, nepoužívat „copy&paste kód“

Další důležité programátorské konstrukce

- složitější datové typy (seznamy, řetězce), objekty
- vstup/výstup (input/output, IO):
  - standardní IO
  - soubory
- dělení projektu do více souborů (packages), použití knihoven

viz další přednášky, cvičení, samostudium

Příklad: výpis šachovnice

```
# . # . # . # .
. # . # . # . #
# . # . # . # .
. # . # . # . #
# . # . # . # .
. # . # . # . #
# . # . # . # .
. # . # . # . #
```

Řešení funkční, ne úplně vhodné

```
def sachovnice(n):
    for i in range(n):
        if (i % 2 == 0): sudy_radek(n)
        else: lichy_radek(n)

def sudy_radek(n):
    for j in range(n):
        if (j % 2 == 0): print "#",
        else: print ".",
    print

def lichy_radek(n):
    for j in range(n):
        if (j % 2 == 1): print "#",
        else: print ".",
    print
```

Lepší řešení

```
def sachovnice(n):
    for i in range(n):
        radek(n, i % 2)

def radek(n, parita):
    for j in range(n):
        if (j % 2 == parita): print "#",
        else: print ".",
    print
```

Jiný zápis

```
def sachovnice(n):
    for i in range(n):
        for j in range(n):
            if ((i+j) % 2 == 0):
                print "#",
            else:
                print ".",
        print
```

Příklad: Hádanka hlavy a nohy

Farmář chová prasata a slepice. Celkem je na dvoře 20 hlav a 56 noh. Kolik má slepic a kolik prasat?

- jak vyřešit pro konkrétní zadání?
- jak vyřešit pro obecné zadání (H hlav a N noh)?
- co když farmář chová ještě osminohé pavouky?

Hlavy, nohy: řešení

dva možné přístupy:

1. „inteligentně“: řešení systému lineárních rovnic
2. „hrubou silou“:
   - „vyzkoušej všechny možnosti“
   - cyklus přes všechny možné počty prasat

Hlavy, nohy: program

```
def hledej_reseni(hlavy, nohy):
  for prasata in range(0, hlavy+1):
      slepice = hlavy - prasata
      if prasata * 4 + slepice * 2 == nohy:
          print "prasata =", prasata,\
                "slepice =", slepice
```

Jak bych musel program změnit, kdybych řešil úlohu i s pavouky?

Ukázka nevhodného programu: ciferný součet

```
if n % 10 == 0:
        f = 0 + f
  elif n % 10 == 1:
        f = 1 + f
  elif n % 10 == 2:
        f = 2 + f
  elif n % 10 == 3:
        f = 3 + f
  elif n % 10 == 4:
        f = 4 + f
  ...
```

Ukázka nevhodného programu: čtverec

```
def ctverec(n):
    for i in range(1, n):
        print "*" * i
        if i == n-1:
            p = 1
            while n+1 > p:
                print "*" * n
                n = n - 1
```