

Programování: základní konstrukce

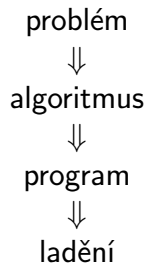
IB111 Úvod do programování skrze Python

2012

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 přístup



Poznámka o ladění

- laděním se nebudeme v tomto kurzu explicitně příliš zabývat
- to ale neznamená, že není důležité...

Ladění je dvakrát tak náročné, jak psaní vlastního kódu. Takže pokud napíšete program tak chytře, jak jen umíte, nebudete schopni jej odladit. (Brian W. Kernighan)

Základní konstrukce

- proměnné, operace
- řízení toku výpočtu:
 - podmínovací 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

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, ...
 - 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
- typ se určuje automaticky (dynamické typování)
- typ proměnné se může měnit
- zjištění typu: funkce `type`
- explicitní přetypování (`x = float(3)`), význam 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"`

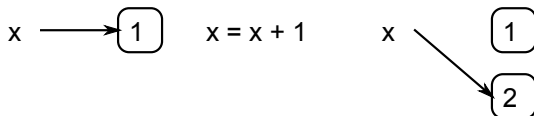
- pořadí vyhodnocování vesměs intuitivní
- pokud na pochybách:
 - konzultujte dokumentaci
 - závorkujte
- zkrácené vyhodnocování: `1+1 == 2 or x == 3`

Operace, pořadí vyhodnocování

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

Proměnné a paměť

x 1 $x = x + 1$ x 2



- 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
```

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)

- 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
- výpočet faktoriálu
- převod čísla na binární zápis

- 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
```

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
```

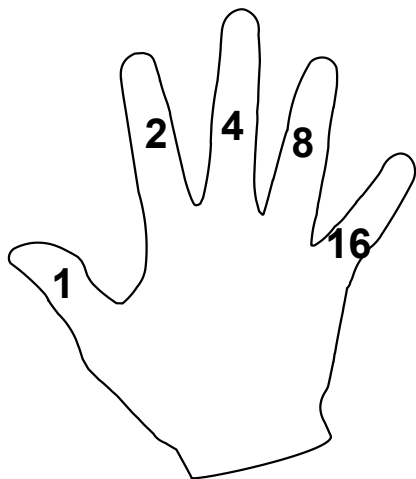
- provádí příkazy dokud platí podmínka
- může se stát:
 - neprovede příkazy ani jednou
 - provádí příkazy do nekonečna (nikdy neskončí) – to většinou znamená chybu v programu
- napište výpočet faktoriálu pomocí while cyklu

Faktoriál pomocí while cyklu

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

print f
```

Binární soustava: 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.

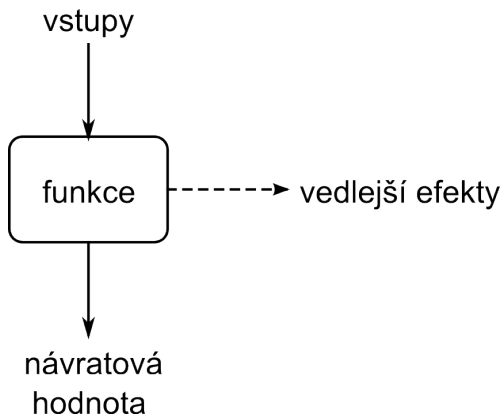
Proč?

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



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"
```

A diagram illustrating nested function calls. It shows two function definitions side-by-side. The left one is `def pokusy_se_sudosti(a, b):` and the right one is `def info_o_sudosti(cislo):`. Arrows indicate the call flow: from the `info_o_sudosti(a)` call in the first function to the start of the second function; from the `info_o_sudosti(b)` call in the first function to the start of the second function; and from the `info_o_sudosti(cislo)` call in the second function to the `print "Konec"` statement in the first function.

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)
```

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**
- dávat proměnným a funkcím **smysluplná jména**
- funkce by měly být krátké:
 - max na jednu obrazovku
 - jen pár úrovní zanoření
- příliš dlouhá funkce – rozdělit na menší
- neopakovat se, nepoužívat „cut&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

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

```


Nevhodné řešení

```
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
```