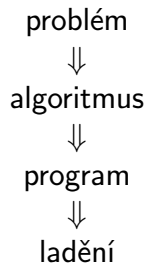


Programování: základní konstrukce

IB111 Programování a algoritmizace

2010

Základní přístup



Citát (varování)

Ladění je dvakrát tak náročné, jak psaní vlastního kódu. Takže pokud napíšete program tak chytrě, 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
- 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
- typy:
 - číselné: int, float, ...
 - řetězec
 - seznam (pole)
 - složené
 - ...

Výrazy a operace

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

Proměnné a výrazy: příklady

```
x = 13
y = x % 4      # deleni 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
```

Proměnné a výrazy: Python

- přiřazení =
- test na rovnost ==
- „deklarace“ proměnné: první přiřazení hodnoty
- typ se určuje automaticky
- explicitní přetypování (`x = float(3)`), význam např. při dělení
 - $3/2 = 1$
 - $\text{float}(3)/2 = 1.5$

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 – např. 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 'Negative changed to zero'
elif x == 0:
    print 'Zero'
elif x == 1:
    print 'Single'
else:
    y = x * x
    print 'More'
```

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 a 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

```
n = input()
f = 1

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

print f
```

While cyklus

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

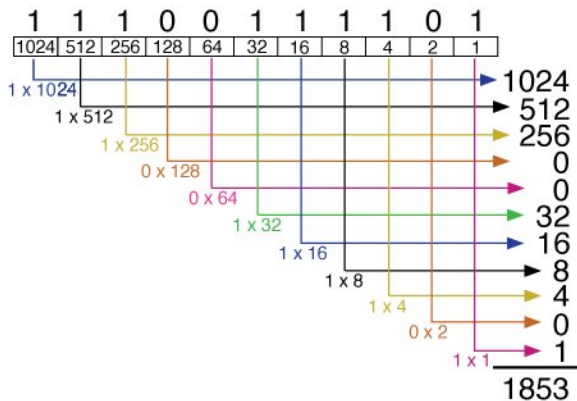
- provádí příkazy dokud platí podmínka
- může se stát:
 - neprovede příkazy ani jednou
 - provádí příkazy do nekonečna (nikdy neskončí) – to většinou znamená chybu v programu
- napište výpočet faktoriálu pomocí while cyklu

Faktoriál pomocí while cyklu

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

print f
```

Příklad: převod na binární zápis



Příklad: převod na binární zápis

Problém: převodník z desítkové na binární soustavu

vstup: číslo v desítkové soustavě

výstup: číslo v binární soustavě

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
```

programy nepíšeme jako jeden dlouhý „štrůdl“, 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

- 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
```

Myšlení o/ve funkcích

- vstupně-výstupní chování
- vedlejší efekty: změny nezahrnuté ve „vstupně-výstupním chování“, spíše nepoužívat
- rekurze: volání sebe sama (podrobněji později)

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

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?