Programování: základní konstrukce

IB111 Programování a algoritmizace

2011

Základní přístup

problém

$\Downarrow$

algoritmus

$\Downarrow$

program

$\Downarrow$

ladění

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:
  - podmiňovací 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
- udržovaná hodnota se může měnit – proto *proměnné*
- typy:
  - číselné: int, float, ...
  - řetězec (string)
  - 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     #  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
```

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
  - $3/2 = 1$
  - 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)

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 '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

Cyklus

- 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

- Co to faktoriál?
- 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
```

While cyklus

```
while <podmínka>:
    příkazy
```

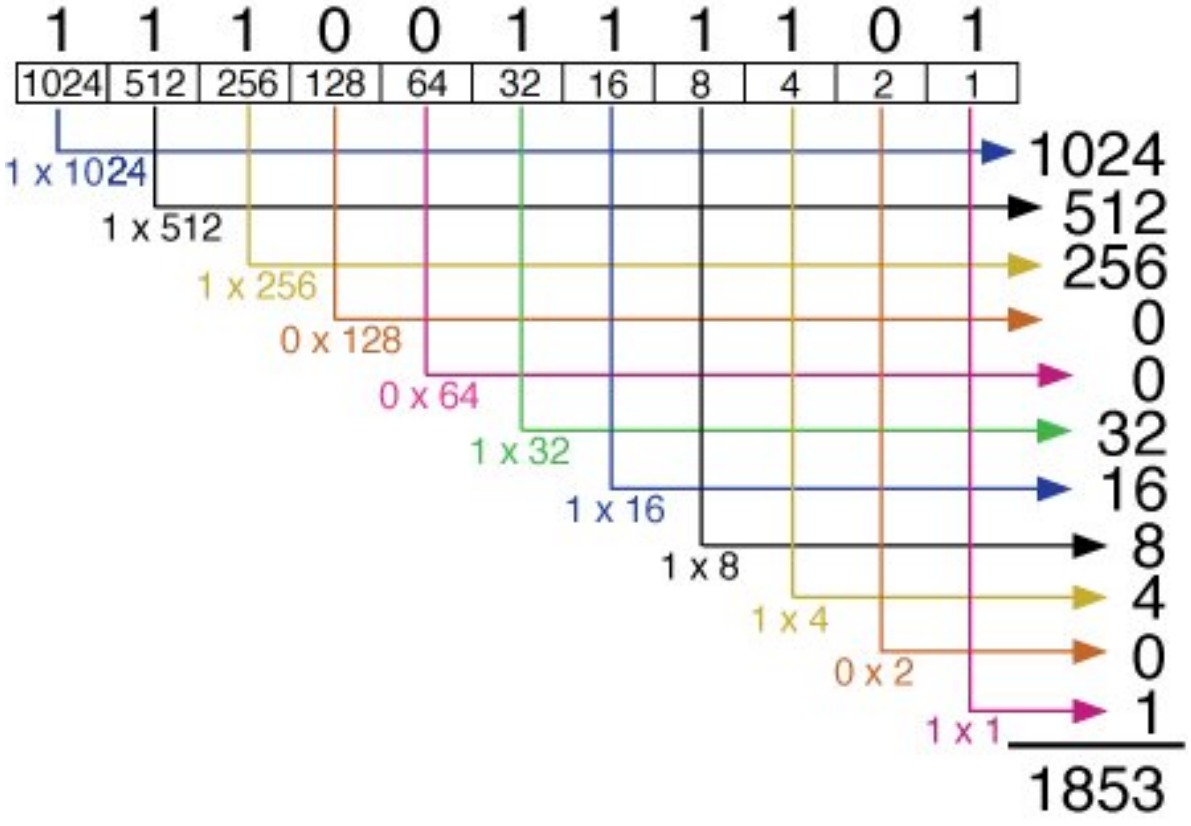
- provádí `příkazy` dokud platí podmínka
- může se stát:
  - neprovede `příkazy` ani jednou
  - provádí `příkazy` do nekonečna (nikdy neskončí) – to většinou znamená chybu v programu
- napište výpočet faktoriálu pomocí while cyklu

Faktoriál pomocí while cyklu

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

print f
```

Binární soustava



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
```

Funkce

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

Proč?

Funkce

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: Jaké mají být vstupní argumenty? Co má být výstupem?
- 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
- neopakovat se, nepoužívat „cut&paste kód“

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?