Programmēšanas valoda

iesācējiem

4. daļa. Programmēšanas prakse

Saturs

Ievads....3
Algoritmi....4
Uzdevumi....6
1. uzdevums. Olu kastes....6
2. uzdevums. Lielie cipari....6
3. uzdevums. Zīmuļi....7
4. uzdevums. Fibonači skaitļi....7
5. uzdevums. Garais gads....7
6. uzdevums. Cietais rieksts....8
7. uzdevums. Pirmreizinātāji....8
8. uzdevums. Eglīte....8
9. uzdevums. Līzings....9
10. uzdevums. Binārie skaitļi....9
...un to risinājumi....10
1. uzdevums. Olu kastes....10
2. uzdevums. Lielie cipari....12
3. uzdevums. Zīmuļu rūpnīca....14
4. uzdevums. Fibonači skaitļi....15
5. uzdevums. Garais gads....16
6. uzdevums. Cietais rieksts....17
7. uzdevums. Pirmreizinātāji....21
8. uzdevums. Eglīte....23
9. uzdevums. Līzings....25
10. uzdevums. Binārie skaitļi....26
Nobeigumam....28

Ievads

Grāmata "Programmēšanas valoda `Python` iesācējiem" ir tematisku pamācību sērija, kuras uzdevums ir sniegt iesācējiem ieskatu programmēšanas valodā `Python`. Šī ir grāmatas ceturtā daļa, kurā aplūkoti dažādi programmēšanas uzdevumi un to realizācija šajā lieliskajā programmēšanas valodā. Šajā daļā esmu nedaudz lauzis tradīciju aplūkot `Python` valodas konstrukcijas un bibliotēkas, tā vietā izmantojot lasītāja pirmajās trīs daļās iegūtās zināšanas, lai risinātu tipiskus programmēšanas un algoritmu izstrādes uzdevumus, tā kā patiesībā šai būtu jābūt 3 ½ daļai. Kādēļ tā?

Pēc trešās daļas laišanas klajā es sapratu, ka līdz šim esmu "spiedis" lasītāju apgūt `Python` valodas konstrukcijas un bibliotēkās iekļauto funkcionalitāti, demonstrējot to ar samērā triviāliem piemēriem. Ja lasītājs ir iesācējs programmēšanā, pieredzes trūkuma dēļ var rasties zināmas problēmas izstrādāt sarežģītākas programmas, tādēļ kļuva skaidrs, ka ir nepieciešams aplūkot tipiskas programmēšanas problēmu situācijas un šo problēmu risinājumus (paldies Madaram Virzam, Matīsam Sīlim un arī visiem citiem par ieteikumiem!).

Lai neizgudrotu velosipēdu, programmēšanas uzdevumi tiek risināti, izmantojot pirmajās trīs grāmatas daļās aplūkotās `Python` konstrukcijas, līdz ar ko tās netiek papildus paskaidrotas. Ja lasītājam rodas priekšstats, ka ir grūtības izprast tādas vai citādas programmas konstrukcijas darbību, varu vien ieteikt aplūkot pirmās trīs grāmatas daļas. Tās iespējams lejuplādēt autora mājaslapā - `http://alvils.latvietis.com/`.

Algoritmi

Lasītājs, kurš nekad iepriekš nav mēģinājis apgūt programmēšanu, varētu jautāt - "kas šis par svešvārdu"? Pēc Vikipēdijas (`http://www.wikipedia.org`) ieraksta varam noskaidrot, ka

Algoritms ir procedūra (galīga detalizētu instrukciju kopa) kāda uzdevuma veikšanai, pie kam, uzsākot procedūras izpildi noteiktā sākumstāvoklī, tās izpilde tiks pabeigta kādā noteiktā beigu stāvoklī.

Ar algoritmiem mēs ikdienā saskaramies nepārtraukti. Tipisks piemērs ir kafijas vārīšana. Lai arī, šķiet, katram būs skaidrs, kā jāvāra kafiju, mēģināsim detalizēti aprakstīt, kā to jāveic.

```
0. Sākumstāvoklī mūsu rīcībā ir tukša krūzīte, karotīte, kafija, cukurtrauks
   ar cukuru, tējkanna un ūdenskrāns. Pārejam pie 1. punkta.
1. Izmantojot karotīti, ieberam krūzītē nepieciešamo daudzumu kafijas. Pārejam
   pie 2. punkta.
2. No ūdenskrāna piepildam tējkannu un liekam to vārīties. Pārejam pie 3.
   punkta.
3. Vai ūdens ir uzvārījies? Ja ir, pārejam pie 4. punkta. Ja nē, pārejam pie
   3. punkta (t.i., atkārtojam tajā aprakstītās darbības).
4. Vai vēlamies kafiju ar cukuru? Ja vēlamies, pārejam pie 5. punkta. Ja nē,
   pārejam pie 6. punkta.
5. Izmantojot karotīti, ieberam krūzītē nepieciešamo daudzumu cukura. Pārejam
   pie 6. punkta.
6. Pielejam krūzīti ar karstu ūdeni no tējkannas. Pārejam pie 7. punkta.
7. Izmantojot karotīti, izmaisam kafiju krūzītē. Pārejam pie 8. punkta.
8. Beigu stāvoklis. Kafija sagatavota!
```

Protams, tik detalizētas instrukcijas varbūt ir nepieciešamas cilvēkam, kurš kafiju vāra pirmo reizi un nekad neko līdzīgu nav darījis. Nākamajā reizē šādas instrukcijas vairs nebūs nepieciešamas, jo cilvēks šo darbību veiks intuitīvi. Diemžēl dators nespēj "mācīties" un tādēļ tam katru reizi, izpildot uzdevumu, ir nepieciešamas šādas instrukcijas.

Datoram "saprotamā" valodā pierakstītu algoritmu sauc par `programmu`.

Tātad - lai varētu izveidot programmu, kas risina kādu problēmu, ir jāprot izveidot šīs problēmas risināšanas algoritmu, pie tam tik detalizētu, lai to būtu iespējams pierakstīt ar izvēlētās programmēšanas valodas līdzekļiem. Lai labāk ilustrētu algoritmu veidošanu, šajā grāmatas daļā aplūkosim dažādus programmēšanas uzdevumus, katram izveidojot algoritmu un to realizējot ar `Python` līdzekļiem.

Algoritmi tiek veidoti, izmantojot "melnās kastes" principu, kas ļauj sākt ar vispārīgu algoritma aprakstu un sadalīt tā atsevišķās komponentes līdz līmenim, kas ļauj to pierakstīt `Python`. Piemēram, "kafijas vārīšanas" algoritmu ir iespējams veidot vairākos "piegājienos":

Vispārīgais

```
1. Krūzītē ieberam kafiju. Pārejam pie 2. soļa.
2. Ja nepieciešams, pieberam cukuru. Pārejam pie 3. soļa.
3. Krūzītē ielejam karstu ūdeni. Pārejam pie 4. soļa.
```

```
4. Izmaisām kafiju. Pārejam pie 5. soļa.
5. Beigu stāvoklis. Kafija sagatavota!
```

Detalizācija

Piemēram, 2. punktu sadalot "sīkāk", tiek iegūti pirms tam norādītā algoritma 4. un 5. punkti.

Uzdevumi...

Lai būtu interesantāk, vispirms aplūkosim pašus uzdevumus. Mēģiniet tos realizēt patstāvīgi, tomēr, ja tas neizdosies (sākums vienmēr ir grūts!), nākamajā nodaļā varēsit atrast to risinājumus.

1. uzdevums. Olu kastes.

Vistu saimniecībā olas tiek iepakotas kastēs pa 10 olām katrā. Izveidot programmu, kura pieprasa ievadīt olu skaitu un izvada atbildi, kurā sniedz informāciju par piepildītajām olu kastēm un, ja pēdējā kaste nav aizpildīta līdz galam, par olu skaitu tajā.

Piemērs:

```
Ievadiet olu skaitu (0 - beigt darbu):
12
1 pilna kaste un 1 kaste ar 2 olam.
Ievadiet olu skaitu (0 - beigt darbu):
23
2 pilnas kastes un 1 kaste ar 3 olam.
Ievadiet olu skaitu (0 - beigt darbu):
41
4 pilnas kastes un 1 kaste ar 1 olu.
```

2. uzdevums. Lielie cipari.

Izveidot programmu, kura pieprasa ievadīt trīsciparu skaitli un izvada to konsolē ar ciprariem, kuru augstums ir 5 rindiņas un platums - 3 simboli. Cipari tiek veidoti, izmantojot simbolu "*". Skaitļus, kas mazāki par 100 (t.i., kuros ir mazāk par trim cipariem), jāizvada, izmantojot nulles simbolus pirms tiem.

Piemērs:

```
Ievadiet skaitli (0 - beigt darbu):
123
  * *** ***
  *   *   *
  * *** ***
  * *     *
  * *** ***
Ievadiet skaitli (0 - beigt darbu):
26
*** *** ***
* *   * *
* * *** ***
* * *   * *
*** *** ***
Ievadiet skaitli (0 - beigt darbu):
1
*** ***   *
* * * *   *
* * * *   *
* * * *   *
*** ***   *
```

3. uzdevums. Zīmuļi.

Rūpnīcā tiek ražoti zīmuļi. Uz konveijera lentes nonāk zīmulis, kurš tiek nokrāsots un noasināts (sk. attēlu). Diemžēl rūpnīcas iekārtas ir sabojājušās un katrs 5. zīmulis netiek nokrāsots un katrs 6. zīmulis, to asinot, tiek nolauzts. Izveidot programmu, kura pieprasa ievadīt apstrādājamo zīmuļu skaitu un sniedz informāciju par skaitu tiem zīmuļiem, kuri nav nokrāsoti, kuri ir nolauzti, kā arī par tiem, kuri nav nokrāsoti un ir nolauzti.

```
Piemērs:
Ievadiet zimulu skaitu (0 - beigt darbu):
35
Nenokrasoti: 6
Nolauzti: 4
Nenokrasoti un nolauzti: 1
```

4. uzdevums. Fibonači skaitļi.

Fibonači skaitļu virkne tiek veidota pēc šādiem nosacījumiem:
1. virknes elements ir 1
2. virknes elements ir 1
Katrs nākamais virknes elements tiek veidots kā divu iepriekšējo elementu summa.
Piemērs:
(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...)

Izveidot programmu, kas pieprasa ievadīt Fibonači skaitļu virknes elementa kārtas numuru un izvada tā vērtību.

```
Piemērs:
Ievadiet virknes elementa numuru (0 - beigt darbu):
8
Fib(8) ir 21.
```

5. uzdevums. Garais gads.

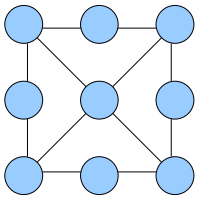
Gads tiek uzskatīts par "garo gadu" un tā februāra mēnesī ir 29 dienas, ja gads dalās ar 4, izņemot gadījumu, ja tas ir "gadsimta" gads (beidzas ar 00). Tādā gadījumā tam jādalās ar 400.

Izveidot programmu, kas pieprasa ievadīt gadu un sniedz informāciju, vai tas ir garais gads vai nē.

```
Piemērs:
Ievadiet gadu (0 - beigt darbu):
2004
Garais gads
Ievadiet gadu (0 - beigt darbu):
2005
Nav garais gads
```

6. uzdevums. Cietais rieksts.

Aplūkosim šādu figūru:



Ievietojot aplīšos ciparus no 1 līdz 9 (katru atļauts izmantot tieši vienu reizi), jāatrod tādu ciparu izvietojumu, lai visās savienoto aplīšu trijnieku kombinācijās šo ciparu summa būtu vienāda.

Lai nelauzītu galvu, jāizveido programmu, kura nosaka visas ciparu izvietojuma kombinācijas, kuras apmierina uzdevuma nosacījumu.

7. uzdevums. Pirmreizinātāji.

Izveidot programmu, kura pieprasa ievadīt skaitli un sadala to pirmreizinātājos. Ja skaitlis ir pirmskaitlis (un attiecīgi to sadalīt pirmreizinātājos nav iespējams), izvadīt attiecīgu paziņojumu.

Piemērs:

```
Ievadiet skaitli (0 - beigt darbu):
12
12 = 2 * 3 * 3
Ievadiet skaitli (0 - beigt darbu):
3
3 ir pirmskaitlis!
```

8. uzdevums. Eglīte.

Izveidot programmu, kura pieprasa ievadīt nepāra skaitli (jāveic atbilstošas pārbaudes!), kas lielāks par 3, un konsolē izvada simbolisku eglītes attēlu, pie kam eglītes apakšējās daļas "garums" ir vienāds ar ievadīto skaitli. Eglītes "apakšā" jāizveido arī 3 simbolus "platu" "stumbru". Attēla veidošanai jāizmanto simbolu "*".

Piemērs:

```
Ievadiet nepara skaitli (>3) (0 - beigt darbu):
6
Ievadits para skaitlis!
Ievadiet nepara skaitli (>3) (0 - beigt darbu):
3
Skaitlim jabut >3!
Ievadiet nepara skaitli (>3) (0 - beigt darbu):
7
```

```
   *
  ***
 *****
*******
  ***
  ***
```

9. uzdevums. Līzings.

Līzinga uzņēmums piedāvā patērētājiem iegādāties preces uz nomaksu. Preces atmaksai tiek piemērota mēneša procentu likme, kura tiek attiecināta uz atlikušo summu. Izveidot programmu, kura pieprasa ievadīt aizņēmuma summu, laika posmu (mēnešos) un mēneša procentu likmi, un izvada maksājumu grafiku katram mēnesim, kā arī grafika beigās aprēķina kopējo procentos nomaksāto summu.

```
Piemērs:
Ievadiet aiznemuma summu:
1000
Ievadiet menesu skaitu:
5
Ievadiet procentu likmi:
10
Maksajumu grafiks:
1. menesis: 300
2. menesis: 280
3. menesis: 260
4. menesis: 240
5. menesis: 220
Kopa procentos jamaksa: 300
Vai atkartot? (j/n):
```

10. uzdevums. Binārie skaitļi.

Izveidot programmu, kura pieprasa ievadīt skaitli un izvada to binārajā skaitīšanas sistēmā.

```
Piemērs:
Ievadiet skaitli (0 - beigt darbu):
10
1010
Ievadiet skaitli (0 - beigt darbu):
255
11111111
```

...un to risinājumi

1. uzdevums. Olu kastes.

Lai atrisinātu šo uzdevumu, mēģināsim izveidot ļoti vispārīgu algoritmu.

```
  0. Sākumstāvoklis. Pieprasām ievadīt datus un veicam attiecīgas pārbaudes
(vai skaitlis >0).
  1. Aprēķinām kastu skaitu un "pāri palikušo" olu skaitu.
  2. Izvadām rezultātu.
```

Redzam, ka svarīgākā daļa ir 1. algoritma punkts. Mēģināsim to aprakstīt detalizēti. Ja kastē iespējams ievietot 10 olas un olu skaits ir `n`, tad top skaidrs, ka "pāri paliks" tik olas, cik ir `n` dalījuma ar 10 atlikums. Vienosimies, ka to apzīmēsim ar `Atlik`.

Ja no `n` atņemam iegūto "pāri palikušo" olu skaitu, iegūstam skaitli, kurš noteikti dalās ar 10. Veicot šo dalīšanu, iegūsim skaitli, kas vienāds ar "pilnajām" kastēm. To apzīmēsim ar `Kastes`.

Tātad, vispārīgā gadījumā aprēķinu varam veikt šādi:

```
  0. Sākumstāvoklis. n - olu skaits.
  1. Atlik - olu skaits, kuras "paliek pāri". Atlik = n % 10. Šeit
atcerēsimies, ka darbība % valodā Python aprēķina atlikumu no dalīšanas.
  2. Kastes - "pilno" kastu skaits. Kastes = (n-Atlik)/10.
  3. Beigu stāvoklis. Nepieciešamie lielumi aprēķināti.
```

Arī 0. stāvokli ir jāapraksta detalizētāk.

```
  0. Sākumstāvoklis.
  1. Pieprasām ievadīt n.
  2. Vai n<0? Ja jā, pārejam uz 3. punktu. Ja nē, pārejam uz 4.punktu.
  3. Izvadām kļūdas paziņojumu. Pārejam uz 1. punktu.
  4. Beigu stāvoklis. n ievadīts.
```

Šķiet, 2. stāvoklis ir visvienkāršākais? Nebūt ne. Uzdevumā ir prasīts informāciju izvadīt "saprotamā" veidā kā teikumu. Tomēr, ja kastu skaits ir 1, 11, 21, ..., tad teikumā redzēsim vārdu "pilna kaste", pretējā gadījumā redzēsim vārdu "pilnas kastes". Līdzīgi ir, izvadot informāciju par "pāri palikušo" olu skaitu. Ja pāri palikusi 1 ola, tad jāizvada vārdu "olu", ja vairāk - "olam". Tas nozīmē, ka mums jāsagatavo divus, patiesībā gandrīz vienādus apakšalgoritmus, kuri veiks šo situāciju "apstrādi".

```
  0. Sākumstāvoklis. Kastes - kastu skaits.
  1. Vai (Kastes % 10) == 1? Ja jā, pārejam pie 2. punkta. Ja nē, pārejam pie
3. punkta.
  2. Rezultāts ir "pilna kaste". Pārejam pie 4. punkta.
  3. Rezultāts ir "pilnas kastes". Pārejam pie 4. punkta.
  4. Beigu stāvoklis.
```

```
  0. Sākumstāvoklis. Olas - "pāri palikušo" olu skaits.
  1. Vai (Olas % 10) == 1? Ja jā, pārejam pie 2. punkta. Ja nē, pārejam pie
3. punkta.
  2. Rezultāts ir "olu". Pārejam pie 4. punkta.
  3. Rezultāts ir "olam". Pārejam pie 4. punkta.
  4. Beigu stāvoklis.
```

Ja apakšalgoritmu, kas nosaka vārda "kastes" "izskatu", nosaucam par `VardsKaste` un apakšalgoritmu, kas nosaka vārda "olas" "izskatu" - par `VardsOlas`, tad rezultāta izvade "izskatās" šādi:

```
Kastes + " " + VardsKaste + " un 1 kaste ar " + Atlik + " " + VardsOlas + "."
```

Vēl kāda uzdevuma prasība ir atkārtot algoritmu, līdz tiek ievadīts skaitlis "0". To iespējams veikt, papildinot "galveno" algoritmu.

```
  0. Sākumstāvoklis. Pieprasām ievadīt datus un veicam attiecīgas pārbaudes
(vai skaitlis >0).
  1. Vai ievadītais skaitlis ir 0? Ja jā, pārejam uz 4. punktu. Ja nē,
pārejam uz 2. punktu.
  2. Aprēķinām kastu skaitu un "pāri palikušo" olu skaitu.
  3. Izvadām rezultātu.
  4. Pārejam uz 0. stāvokli.
  5. Beigu stāvoklis.
```

Tagad mēģināsim izstrādāto algoritmu realizēt valodā `Python`. Tā kā apakšalgoritmi `VardsKaste` un `VardsOlas` "neattiecas" uz pamatuzdevumu, tos realizēsim kā `Python` funkcijas. Funkcijām tiks nodots viens arguments - `n`, kas apzīmēs attiecīgi kastu vai olu skaitu.

```
  def VardsKaste(n):
      if n % 10 == 1:
            return "pilna kaste"
      return "pilnas kastes"
  def VardsOlas(n):
      if n % 10 == 1:
            return "olu"
      return "olam"
```

Tagad realizēsim algoritma 0. punktu pēc sastādītā apakšalgoritma.

```
  while 1:
      print "Ievadiet olu skaitu (0 - beigt darbu):"
      n = input()
      if n >= 0:
            break
      print "Olu skaitam jabut > 0!"
```

Algoritma 1. punktu (vai pēc papildināšanas - 2.), kurš veic aprēķinu, realizēsim šādi:

```
  Atlik = n % 10
  Kastes = (n - Atlik) / 10
```

Rezultātu izvadi veiksim šādi:

```
  print str(Kastes) + " " + VardsKaste(Kastes) + " un 1 kaste ar " +
str(Atlik) + " " + VardsOlas(Atlik) + "."
```

Visbeidzot, mēģināsim to visu "salikt kopā", vienlaikus realizējot arī programmas darbu ciklā un darba pārtraukšanu pēc "0" ievades.

```
  # Funkcija VardsKaste
  def VardsKaste(n):
      if n % 10 == 1:
            return "pilna kaste"
      return "pilnas kastes"
  # Funkcija VardsKastes
  def VardsOlas(n):
      if n % 10 == 1:
            return "olu"
      return "olam"
  # Programmas cikls
  while 1:
      # Olu skaita ievades cikls
      while 1:
            print "Ievadiet olu skaitu (0 - beigt darbu):"
            n = input()
            # Vai olu skaits >= 0?
            if n >= 0:
                # Partraucam ievades ciklu
                  break
            print "Olu skaitam jabut > 0!"
      # Vai olu skaits == 0?
      if n == 0:
          # Partraucam programmas ciklu
            break;
      # Aprekinam kastu skaitu un atlikumu
      Atlik = n % 10
      Kastes = (n - Atlik) / 10
      # Izvadam rezultatu
      print str(Kastes) + " " + VardsKaste(Kastes) + " un 1 kaste ar " +
str(Atlik) + " " + VardsOlas(Atlik) + "."
```

2. uzdevums. Lielie cipari.

Vispirms vienosimies, kā tad "izskatīsies" izvadāmie cipari. Piemēram, tie varētu izskatīties šādi:

```
  ***   * *** *** * * *** *** *** *** ***
  * *   *   *   * * * *   *     * * * * *
  * *   * *** *** *** *** ***   * *** ***
  * *   * *     *   *   * * *  *  * *   *
  ***   * *** ***   * *** ***  *  *** ***
```

Tālāk atcerēsimies uzdevuma nosacījumu - "Skaitļus, kas mazāki par 100 (t.i., kuros ir mazāk par trim cipariem), jāizvada, izmantojot nulles simbolus pirms tiem". Tas nozīmē, ka pirms ciparu virknītes jāveic pārbaudi - ja skaitlis ir mazāks par 100, tad tam "priekšā" jāpievieno nulles simbolu, bet, ja tas ir mazāks par 10 - tad vēl vienu nulles simbolu.

Skaitļa izvadi teksta konsolē apgrūtina fakts, ka mēs nevaram "zīmēt" attēlu patvaļīgā vietā uz ekrāna, līdz ar to pa rindiņai vien jāizvada visus trīs ciparus vienlaicīgi.

Ņemot vērā šos nosacījumus, mēģināsim sastādīt algoritmu.

```
  0. Sākumstāvoklis. Pieprasām ievadīt skaitli. Pārejam pie 1.soļa.
  1. Ja skaitlis ir lielāks vai vienāds ar 100, to pārveidojam teksta
rindiņā, kas sastāv no trim simboliem. Pārejam pie 2.soļa.
```

```
  2. Ja skaitlis ir mazāks par 100, bet lielāks vai vienāds ar 10, to
pārveidojam teksta rindiņā, tās "priekšā" pievienojot ciparu "0". Pārejam pie
3.soļa.
  3. Ja skaitlis ir mazāks par 10, to pārveidojam teksta rindiņā, tās
"priekšā" pievienojot divus ciparus - "00". Pārejam pie 4.soļa.
  4. Ciklā (no 1 līdz 5) izvadām katru trīs ciparu virknītes rindiņu n.
```

Šajā algoritmā "melnā kaste" ir 4.solis, kurā tiek veikta ciparu virknītes rindiņas izvade. Pieņemot, ka mums ir funkcija `Cipars(n,m)`, kura teksta konsolē izvada cipara `n` "attēla" rindiņu `m`, šī "melnā kaste" cipariem 1,2 un 3 izskatīsies šādi:

```
  print Cipars(1, m)+" "+Cipars(2, m)+" "+Cipars(3,m)
```

Tas nozīmē, ka patiesā "melnā kaste" ir funkcija `Cipars()`. Lai to izveidotu, jāvienojas, kā tiks glabāts katra cipara "attēls". `Python` valodā to ērti realizēt, izmantojot sarakstu, kurā katrs elements ir visu ciparu vienas rindiņas "attēlu" saraksts:

```
Cipari = [["***","  *","***","***","* *","***","***","***","***","***"],
          ["* *","  *","  *","  *","* *","*  ","*  ","  *","* *","* *"],
          ["* *","  *","***","***","***","***","***","  *","***","***"],
          ["* *","  *","*  ","  *","  *","  *","* *","* * ","* *","  *"],
          ["***","  *","***","***","  *","***","***","* * ","***","***"]]
```

Tas nozīmē, ka funkcijas `Cipars` izpildes rezultāts ir gluži vienkārši saraksta `Cipari` noteiktā elementa (`m`) elements (`n`).

Proti:

```
  def Cipars(n,m):
     Cipari = [["***","  *","***","***","* *","***","***","***","***","***"],
               ["* *","  *","  *","  *","* *","*  ","*  ","  *","* *","* *"],
               ["* *","  *","***","***","***","***","***","  *","***","***"],
               ["* *","  *","*  ","  *","  *","  *","* *","* * ","* *","  *"],
               ["***","  *","***","***","  *","***","***","* * ","***","***"]]
      return Cipari[m][n]
```

Tā kā pati svarīgākā risinājuma daļa ir izveidota, realizēsim arī pašu programmu:

```
while 1:
    # Algoritma 0.solis
    print "Ievadiet skaitli (0-beigt darbu):"
    Skaitlis = input()
    Virkne = ""
    # Ja ievadits 0, partraucam darbu
    if Skaitlis == 0:
      break
    # Algoritma 1.solis
    if Skaitlis >= 100:
      Virkne = str(Skaitlis)
    # Algoritma 2.solis
    if Skaitlis < 100 and Skaitlis >= 10:
      Virkne = "0" + str(Skaitlis)
    # Algoritma 3.solis
    if Skaitlis < 10:
      Virkne = "00" + str(Skaitlis)
    # Algoritma 4.solis
    for Rinda in xrange(0,5):
       print Cipars(int(Virkne[0]), Rinda)+" "+Cipars(int(Virkne[1]), Rinda)+"
"+Cipars(int(Virkne[2]), Rinda)
```

Ja ir vēlēšanās, pamēģiniet pārveidot šo programmu, lai tā izvadītu 4 ciparu

virknītes. Papildus var mēģināt pārveidot programmu tā, lai būtu iespējams izvadīt patvaļīga teksta virknīti, izmantojot angļu alfabēta burtus.

3. uzdevums. Zīmuļu rūpnīca.

Aplūkojot uzdevumu un pieņemot, ka katram zīmulim tiek piešķirts kārtas skaitlis n, kļūst skaidrs, ka:

1. Ja n dalot ar 5, atlikums ir 0, tad zīmulis nav nokrāsots.

2. Ja n dalot ar 6, atlikums ir 0, tad zīmulis ir nolauzts.

3. Ja izpildās abi iepriekšminētie nosacījumi, tad zīmulis nav nokrāsots un ir nolauzts.

Mēģinot situāciju pētīt tālāk un pieņemot, ka zīmuļu skaits ir `m`, redzam, ka:

1. Nenokrāsoto zīmuļu skaits ir vienāds ar `m` dalījumu ar 5.

2. Nolauzto zīmuļu skaits ir vienāds ar `m` dalījumu ar 6.

Tas nozīmē, ka jāizveido metodi, pēc kuras iespējams noteikt arī nenokrāsoto un nolauzto zīmuļu skaitu `x`, kuru pēc tam jāatņem no nenokrāsoto zīmuļu skaita, kā arī no nolauzto zīmuļu skaita. To iegūstam, dalot `m` ar 5 un 6 reizinājumu - 30.

Tagad varam izveidot arī algoritmu.

```
  0. Sākumstāvoklis. Pieprasām ievadīt zīmuļu skaitu m. Pārejam pie 1. soļa.
  1. Aprēķinām to brāķu skaitu B1, kuri nav nokrāsoti.
  2. Aprēķinām to brāķu skaitu B2, kuri ir nolauzti.
  3. Aprēķinām to brāķu skaitu B3, kuri nav nokrāsoti un ir nolauzti.
  4. Precizējam B1 skaitu, no tā atņemot B3 - jo B3 ir pieskaitīti arī B1
piederošie zīmuļi.
  5. Precizējam B2 skaitu, no tā atņemot B3 - jo B3 ir pieskaitīti arī B2
piederošie zīmuļi.
  6. Izvadām rezultātu.
```

Tagad varam sastādīt programmu:

```
while 1:
  # Algoritma 0.solis
  print "Ievadiet zimulu skaitu (0-beigt darbu):"
  m = input()
  # Ja ievadits 0, partraucam darbu
  if m == 0:
    break
  # Aprekinam B1
  B1 = m / 5
  # Aprekinam B2
  B2 = m / 6
  # Aprekinam B3
  B3 = m / 30
  # Precizejam B1 un B2
  B1 = B1 - B3
  B2 = B2 - B3
  # Izvadam rezultatu
  print "Nenokrasoti:", B1
  print "Nolauzti:", B2
  print "Nenokrasoti un nolauzti:", B3
```

4. uzdevums. Fibonači skaitļi.

Kā redzams no šī uzdevuma nosacījumiem, katru nākamo Fibonači skaitļu virknes elementu varam aprēķināt, saskaitot divus iepriekšējos elementus. Tas nozīmē, ka katra skaitļa aprēķināšanas brīdī jāzina šāda informācija:

Virknes elementa numurs `n`

Skaitļa `Fib(n-1)` vērtība

Skaitļa `Fib(n-2)` vērtība

Tad virknes elementu `Fib(n)` varam aprēķināt šādi:

```
Fib(n) = Fib(n-1)+Fib(n-2)
```

Pie tam jāatceras, ka pirmais un otrais virknes elements vienmēr ir 1.

Mēģināsim sastādīt vispārīgu algoritmu, ņemot talkā "melno kasti" - funkciju, kura aprēķina `n`-to Fibonači skaitli, ja `n`>2.

```
  0. Sākumstāvoklis. Pieprasām ievadīt aprēķināmā Fibonači skaitļu virknes
elementa numuru n. Pārejam pie 1.soļa.
  1. Aprēķinām n-to Fibonači skaitļu virknes elementu. Pārejam pie 2.soļa.
  2. Izvadām rezultātu.
```

Lai pabeigtu uzdevuma risināšanu, jāizveido arī algoritmu 1. punktā izmantotajai "melnajai kastei". To varam uzskatīt par funkciju `Fib(n)`, kurai tiek norādīts viens arguments - `n`, kas ir Fibonači skaitļu virknes elementa kārtas numurs, un tās rezultāts ir atbilstošais Fibonači skaitlis.

```
  1. Ja n=1 vai n=2, tad Fib(n)=1. Atgriežam rezultātu.
  2. Noteiksim, ka n1 un n2 ir divi "iepriekšējie" Fibonači skaitļu virknes
skaitļi un t ir tekošā aplūkotā virknes elementa kārtas numurs. n1=1; n2=1.
  3. Izpildām ciklu no 3 līdz n.
  3.1. n1=n1+n2
  3.2. n2=n1-n2
  4. Fib(n) vērtība ir n2 pēc cikla izpildes beigām.
```

Ievērojiet, kāda "viltība" tika izmantota šī algoritma punktos 3.1 un 3.2! Tā kā `n1` satur `Fib(n-1)` vērtību un `n2` - `Fib(n-2)` vērtību un pēc šo darbību izpildes `n1` jāsatur "iepriekšējo" `n1` un `n2` summu, bet `n2` - "iepriekšējā" `n1` vērtību, tad pirmajā darbībā pie `n1` pieskaitām `n2`, tā iegūstot `n1` un `n2` summu, bet otrajā - `n2` piešķiram starpību starp "jauno" `n1` un `n2`, kas patiesībā ir "vecā" `n1` vērtība. Patiešām, ja `n1` ir 2 un `n2` ir 1, tad:

```
  n1 = 2 + 1 = 3
  n2 = 3 - 1 = 2
```

Tagad realizēsim funkciju `Fib(n)` un arī pašu pamatprogrammu.

```
def Fib(n):
   # Ja n=1 vai n=2, tad Fib(n)=1
   if n==1 or n==2:
      return 1
   # Sakotnejas n1 un n2 vertibas
   n1=1
   n2=1
   # Cikls
   for c in xrange(3,n+1):
      n1=n1+n2
      n2=n1-n2
   # Atgriezam rezultatu
   return n1
while 1:
  # Algoritma 0.solis
  print "Ievadiet virknes elementa numuru (0-beigt darbu):"
  n = input()
  # Ja ievadits 0, partraucam darbu
  if n == 0:
    break
  # Aprekinam un izdrukajam rezultatu
  print "Fib(",n,") =",Fib(n)
```

5. uzdevums. Garais gads.

Vispirms vienosimies par ļoti vispārīgu algoritmu, uzskatot, ka garā gada noteikšana ir "melnā kaste".

```
0. Sākumstāvoklis. Pieprasām ievadīt gadskaitli.
1. "Melnā kaste". Nosakām, vai tas ir garais gads.
2. Izvadām rezultātu.
```

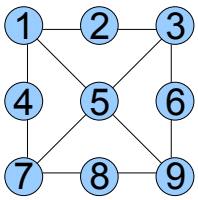
Tagad mēģināsim saprast, kādam jābūt algoritmam, kas realizē "melnās kastes" funkciju. Vispirms jāaplūko divas situācijas - situācija, kurā tiek "apstrādāds" "gadsimta" gads, un situācija, kurā tiek "apstrādāti" visi pārējie gadi. Pirmajā gadījumā jāpārbauda, vai gadskaitlis dalās ar 400, otrajā - vai tas dalās ar 4. Kā noteikt, vai gadskaitlis ir "gadsimta" gads? Ļoti vienkārši - to dalot ar 100, atlikumam no dalījuma jābūt 0. Valodā `Python` atlikuma aprēķins tiek realizēts ar operatoru `%`. Tieši tāpat nosakām, vai gadskaitlis dalās ar 400 un 4 - atlikumam no atbilstošā dalījuma jābūt 0. Tagad sastādīsim programmu:

```
def GaraisGads(Gads):
  # Vai gads ir "gadsimta" gads?
  if Gads % 100 == 0:
    # Vai tas dalas ar 400?
    if Gads % 400 == 0:
      return "Garais gads"
  # Gads nav "gadsimta" gads...
  else:
    # Vai tas dalas ar 4?
    if Gads % 4 == 0:
      return "Garais gads"
  # Gads nav garais gads
  return "Nav garais gads"

# Galvenais programmas cikls
while 1:
  # Algoritma 0.solis
  print "Ievadiet gadu (0-beigt darbu):"
  Gads = input()
  # Ja ievadits 0, partraucam darbu
  if Gads == 0:
    break
  # Aprekinam un izdrukajam rezultatu
  print GaraisGads(Gads)
```

6. uzdevums. Cietais rieksts.

Lai būtu ērtāk izprast uzdevumu, apzīmēsim katru aplīti ar skaitli no 1 līdz 9:



Skaitļu izvietojums aplīšos ir "pareizs", ja atbilstošo aplīšu summas ir vienādas. Mēģināsim izveidot algoritmu, kas veiks šo summu pārbaudi katram iespējamam skaitlim katrā aplītī:

```
  0. Sākumstāvoklis. Pārejam pie 1.soļa.
  1. Aplūkojam visus N1 no 1 līdz 9:
  1.1. Aplūkojam visus N2 no 1 līdz 9:
  1.1.1. Aplūkojam visus N3 no 1 līdz 9:
  1.1.1.1. Aplūkojam visus N4 no 1 līdz 9:
  1.1.1.1.1. Aplūkojam visus N5 no 1 līdz 9:
  1.1.1.1.1.1. Aplūkojam visus N6 no 1 līdz 9:
  1.1.1.1.1.1.1. Aplūkojam visus N7 no 1 līdz 9:
  1.1.1.1.1.1.1.1. Aplūkojam visus N8 no 1 līdz 9:
  1.1.1.1.1.1.1.1.1. Aplūkojam visus N9 no 1 līdz 9:
  1.1.1.1.1.1.1.1.1.1. Ja N1...N9 ir dažādi skaitļi (jo tie nedrīkst
atkārtoties), tad:
  1.1.1.1.1.1.1.1.1.1.1. Ja aplīšu summas ir vienādas, tad atrasts viens no
rezultātiem. Izvadām to.
  2. Beigu stāvoklis.
```

Nav diez ko skaisti, ko? Vispirms noteikti nepatīkama ir pārbaude "Vai `N1`...`N9` ir dažādi skaitļi". Bez tam, kopā programmai būs jāapskata 9^9 skaitļu kombinācijas, t.i., 387420489 kombinācijas. Tas var prasīt visnotaļ ilgu laiku. Vai šeit iespējams ko uzlabot? Ir gan!

`Python` valodā eksistē kāda interesanta konstrukcija. Pamēģināsim to izpildīt:

```
  a = [1,2,3,4]
  print a
  b = [1,4]
  for c in b:
      a.remove(c)
  print a
```

Kā redzams, no saraksta `a` izdzēsti elementi "`1`" un "`4`". Šo `Python` valodas iespēju varam izmantot, lai samazinātu darbību skaitu un arī novērstu nepieciešamību pēc pārbaudes, vai `N1`...`N9` ir dažādi skaitļi:

```
  0. Sākumstāvoklis. Sagatavojam sarakstu Sagatave ar vērtībām no 0 līdz 8.
  1. Sagatavojam sarakstu S1, to "nokopējot" no Sagatave.
  2. Aplūkojam visus N1 no saraksta S1:
  2.1. Sagatavojam sarakstu S2, to "nokopējot" no Sagatave un dzēšot no tā
vērtību N1.
  2.2. Aplūkojam visus N2 no saraksta S2:
  2.2.1. Sagatavojam sarakstu S3, to "nokopējot" no Sagatave un dzēšot no tā
vērtības N1 un N2.
  2.2.2. Aplūkojam visus N3 no saraksta S3:
  2.2.2.1. Sagatavojam sarakstu S4, to "nokopējot" no Sagatave un dzēšot no
tā vērtības N1...N3.
  2.2.2.2. Aplūkojam visus N4 no saraksta S4:
  2.2.2.2.1. Sagatavojam sarakstu S5, to "nokopējot" no Sagatave un dzēšot no
tā vērtības N1...N4.
  2.2.2.2.2. Aplūkojam visus N5 no saraksta S5:
  2.2.2.2.2.1. Sagatavojam sarakstu S6, to "nokopējot" no Sagatave un dzēšot
no tā vērtības N1...N5.
```

```
  2.2.2.2.2.2. Aplūkojam visus N6 no saraksta S6:
  2.2.2.2.2.2.1. Sagatavojam sarakstu S7, to "nokopējot" no Sagatave un
dzēšot no tā vērtības N1...N6.
  2.2.2.2.2.2.2. Aplūkojam visus N7 no saraksta S7:
  2.2.2.2.2.2.2.1. Sagatavojam sarakstu S8, to "nokopējot" no Sagatave un
dzēšot no tā vērtības N1...N7.
  2.2.2.2.2.2.2.2. Aplūkojam visus N8 no saraksta S8:
  2.2.2.2.2.2.2.2.1. Sagatavojam sarakstu S9, to "nokopējot" no Sagatave un
dzēšot no tā vērtības N1...N8.
  2.2.2.2.2.2.2.2.2. Aplūkojam visus N9 no saraksta S9:
  2.2.2.2.2.2.2.2.2.1. Ja aplīšu summas ir vienādas, tad atrasts viens no
rezultātiem. Izvadām to.
  3. Beigu stāvoklis.
```

Izpildot šo algoritmu, nav jāveic pārbaudi, vai `N1`...`N9` ir dažādi skaitļi. Papildus tam, izpildāmo darbību skaits ir samazinājies līdz `9!` jeb 362880, t.i., vairāk kā 1000 reižu! Uzskatīsim šo par gana optimālu algoritmu un izveidosim `Python` programmu:

```
  # 1. solis
  S1 = [1,2,3,4,5,6,7,8,9]
  # 2. solis un talak...
  for N1 in S1:
      S2 = [1,2,3,4,5,6,7,8,9]
      S2.remove(N1)
      for N2 in S2:
            S3 = [1,2,3,4,5,6,7,8,9]
            S3.remove(N1)
            S3.remove(N2)
            for N3 in S3:
                  S4 = [1,2,3,4,5,6,7,8,9]
                  S4.remove(N1)
            S4.remove(N2)
                  S4.remove(N3)
                  for N4 in S4:
                        S5 = [1,2,3,4,5,6,7,8,9]
                        S5.remove(N1)
                  S5.remove(N2)
                        S5.remove(N3)
                        S5.remove(N4)
                        for N5 in S5:
                              S6 = [1,2,3,4,5,6,7,8,9]
                              S6.remove(N1)
                       S6.remove(N2)
                          S6.remove(N3)
                          S6.remove(N4)
                                S6.remove(N5)
                                for N6 in S6:
                                      S7 = [1,2,3,4,5,6,7,8,9]
                                      S7.remove(N1)
                            S7.remove(N2)
                                S7.remove(N3)
                                S7.remove(N4)
                                      S7.remove(N5)
                                      S7.remove(N6)
                                      for N7 in S7:
                                            S8 = [1,2,3,4,5,6,7,8,9]
                                            S8.remove(N1)
                                S8.remove(N2)
                                S8.remove(N3)
                                S8.remove(N4)
                                      S8.remove(N5)
                                      S8.remove(N6)
                                            S8.remove(N7)
                                            for N8 in S8:
                                                  S9 = [1,2,3,4,5,6,7,8,9]
                                                  S9.remove(N1)
                                    S9.remove(N2)
                                      S9.remove(N3)
                                      S9.remove(N4)
                                      S9.remove(N5)
                                            S9.remove(N6)
                                            S9.remove(N7)
                                                  S9.remove(N8)
                                                  # Seit vairs nav jaciklejas,
jo S9 ir tikai viens elements
                                                  N9 = S9[0]
                                                  #  Parbaude,  vai  summas  ir
pareizas:
                                                  Sum1 = N1+N2+N3
```

```
                                                Sum2 = N4+N5+N6
                                                Sum3 = N7+N8+N9
                                                Sum4 = N1+N5+N9
                                                Sum5 = N3+N5+N7
                                                Sum6 = N1+N4+N7
                                                Sum7 = N3+N6+N9
                                                if      (Sum1==Sum2)      and
(Sum1==Sum3)   and   (Sum1==Sum4)   and   (Sum1==Sum5)   and   (Sum1==Sum6)   and
(Sum1==Sum7):
                                                        print
"****************"
                                                        print N1, N2, N3
                                                        print N4, N5, N6
                                                        print N7, N8, N9

  # 3.solis
  print "Visi rezultati atrasti."
  raw_input()
```

Protams, elementu dzēšanu no sarakstiem `S2`...`S9`, iespējams organizēt, izmantojot iepriekš aplūkoto konstrukciju `for...in`. Pamēģiniet patstāvīgi pārveidot šo programmu, lai izmantotu šādu metodi!

7. uzdevums. Pirmreizinātāji.

Vispārīgā gadījumā algoritms izskatās šādi:

```
  0. Sākumstāvoklis. Pieprasām ievadīt skaitli un pārejam pie 1.soļa.
  1. Ja skaitlis ir pirmskaitlis (t.i., nedalās ne ar vienu citu pirmskaitli,
kas mazāks par to), izvadām kļūdas paziņojumu un pārejam pie 0.soļa. Pretējā
gadījumā pārejam pie 2.soļa.
  2. Mēģinām dalīt šo skaitli ar pirmskaitļiem, kas mazāki par šo skaitli,
sākot ar lielāko no tiem. Atrodot pirmskaitli, ar kuru mūsu skaitlis dalās, to
uzskatām par pirmreizinātāju. Atkārtojam šo darbību ar dalījumu, līdz dalījums
ir vienāds ar 2.
  3. Izvadām rezultātu.
```

Kā redzams, šajā algoritmā ir vairākas "melnās kastes". Pirmkārt, ir nepieciešams to pirmskaitļu saraksts, kuri ir mazāki par aplūkojamo skaitli. Otrkārt, ir nepieciešama funkcija, kas spēj noteikt, vai skaitlis ir pirmskaitlis (pretējā gadījumā nevaram izveidot pirmskaitļu sarakstu un arī nevaram izpildīt algoritma 1.soli. Sāksim ar to, ka izveidosim funkciju, kura noteiks, vai skaitlis ir pirmskaitlis. Tas nozīmē, ka šis skaitlis nedrīkst dalīties ar skaitļiem, kas mazāki par to. Patiesībā kā dalītāji skaitlim `N` jāaplūko tikai skaitļi no 2 līdz (`N`-2), jo jebkuri skaitļi dalās ar 1 (tātad to aplūkot nav vērts), un pirmskaitļi nedalās ar (`N`-1), jo (`N`-1) noteikti ir pāra skaitlis, ja `N` ir pirmskaitlis. Savukārt, ja `N` ir pāra skaitlis un tas ir lielāks par 2, tad `N` nav pirmskaitlis. Mēģināsim sastādīt funkciju `Pirmsk(N)`, kuras rezultāts būs `True`, ja `N` ir pirmskaitlis, vai `False` - ja nav.

```
  def Pirmsk(N):
      for c in xrange(2,N):
            if N % c == 0:
                  return False
      return True
```

Tālāk izveidosim funkciju `PirmskaitluAprekins(N)`, kas atgriež sarakstu ar

visu to pirmskaitļu vērtībām, kas mazāki par `N`.

```
def PirmskaitluAprekins(N):
    Pirmskaitli = []
    for c in xrange(2,N):
          if Pirmsk(c):
                Pirmskaitli.append(c)
    return Pirmskaitli
```

Kā pēdējo izveidosim funkciju `Aprekins`, kura atgriež sarakstu ar skaitļa `N` pirmreizinātāju vērtībām.

```
def Aprekins(N):
    Pirmreiz = []
    while N > 2:
          Pirmskaitli = PirmskaitluAprekins(N+1)
          # Sakartojam sarakstu Pirmskaitli apgriezta seciba
          Pirmskaitli.sort(reverse=True)
          for s in Pirmskaitli:
                if N % s == 0:
                      # Atrasts pirmreizinatajs
                      Pirmreiz.append(s)
                      N = N / s
                      break
    # Pievienojam pirmreizinataju sarakstam 2, ja "atlikusais" N ir 2
    if N==2:
          Pirmreiz.append(2)
    # Atgriezam rezultatu
    return Pirmreiz
```

Kad visas šīs funkcijas ir izveidotas, varam izveidot "galveno" programmu, kura veiks mūsu izveidoto funkciju izsaukumus uzdevuma risināšanai, kā arī izvadīs rezultātu.

```
  def Pirmsk(N):
      for c in xrange(2,N):
            if N % c == 0:
                  return False
      return True
  def PirmskaitluAprekins(N):
      Pirmskaitli = []
      for c in xrange(2,N):
            if Pirmsk(c):
                  Pirmskaitli.append(c)
      return Pirmskaitli
  def Aprekins(N):
      Pirmreiz = []
      while N > 2:
            Pirmskaitli = PirmskaitluAprekins(N+1)
            # Sakartojam sarakstu Pirmskaitli apgriezta seciba
            Pirmskaitli.sort(reverse=True)
            for s in Pirmskaitli:
                  if N % s == 0:
                        # Atrasts pirmreizinatajs
                        Pirmreiz.append(s)
                        N = N / s
                        break
      # Pievienojam pirmreizinataju sarakstam 2, ja "atlikusais" N ir 2
      if N==2:
            Pirmreiz.append(2)
      # Atgriezam rezultatu
      return Pirmreiz
  while 1:
    # Algoritma 0.solis
    print "Ievadiet skaitli (0-beigt darbu):"
    Skaitlis = input()
    # Ja ievadits 0, partraucam darbu
    if Skaitlis == 0:
      break
    # Ja ievadits pirmskaitlis, izvadam pazinojumu...
    if Pirmsk(Skaitlis):
      print Skaitlis, "ir pirmskaitlis!"
      #  ...  preteja  gadijuma  sadalam  skaitli  pirmreizinatajos  un  izvadam
rezultatu.
    else:
      Pirmreiz = Aprekins(Skaitlis)
      # Sarakstu izvadisim teksta rinda
      IzvadesRinda = ""
      for s in Pirmreiz:
            IzvadesRinda = IzvadesRinda + str(s) + " * "
      IzvadesRinda = IzvadesRinda[:len(IzvadesRinda)-2]
      print Skaitlis, "=", IzvadesRinda
```

8. uzdevums. Eglīte.

Faktiski programmu varam sadalīt divās daļās:

1) Skaitļa ievade un pārbaude

2) Eglītes izvade

Skaitļa ievadi realizēsim šādi:

```
  while 1:
```

```
    # Skaitla ievade
    print "Ievadiet nepara skaitli (>3) (0 - beigt darbu):"
    Skaitlis = input()
    if Skaitlis == 0:
          break
    if Skaitlis >3:
          if Skaitlis % 2 == 1:
                ZimetEgliti(Skaitlis)
          else:
                print "Ievadits para skaitlis!"
    else:
          print "Skaitlim jabut >3!"
```

Kā redzam, eglītes "zīmēšanu" veic funkcija `ZimetEgliti`, kurai kā arguments tiek nodots nepāra skaitlis, kas lielāks par 3. Visos citos gadījumos (ievadīts pāra skaitlis, vai arī skaitlis nav lielāks par 3) tiek izvadīts atbilstošs kļūdas paziņojums un tiek piedāvāts atkārtoti ievadīt skaitli.

Lai eglītes "zīmēšanu" padarītu uzskatāmāku, izveidosim funkciju `ZimetRindu`, kura izvadīs teksta rindu, kura saturēs norādītu skaitu simbolu `*` ar "pareizu" atkāpi. Atkāpes simbolu skaitu `A` varam aprēķināt šādi (`N` ir rindas garums, bet `Z` - `*` simbolu skaits):

```
A = (N-Z)/2
```

Patiešām, ja rindas garums ir 7 simboli un tajā jābūt 3 `*` simboliem, tad atkāpe būs (7-3)/2 = 2 simboli.

Lai iegūtu teksta rindu, kuras sākumā būtu `A` atstarpju simboli, izmantosim `Python` valodas īpatnību - teksta rindu reizinot ar skaitli, tiek iegūta teksta rinda, kurā atbilstošu skaitu reižu atkārtojas sākotnējā teksta rinda. Piemēram, `"A"*5` rezultāts būs `"AAAAA"` (sk. grāmatas "Programmēšanas valoda `Python` iesācējiem" pirmo daļu).

```
def ZimetRindu(N,Z):
    # N - rindas garums, Z - * simbolu skaits
    # Aprekinam "atstarpi"
    A = (N-Z)/2
    # Izvadam rindu
    print " " * A + "*" * Z
    return
```

Funkcijas `ZimetEgliti` "uzdevums", savukārt, ir atbilstoši parametram `N` (rindas garums) izsaukt funkciju `ZimetRindu` visiem nepāra skaitļiem no 1 līdz `N`, pēc kā izvadīt "stumbru", divas reizes izsaucot `ZimetRindu` un norādot `*` simbolu skaitu kā 3:

```
def ZimetEgliti(N):
    # N - rindas garums
    for Skaits in xrange(1, N+1):
          if Skaits % 2 == 1:
                ZimetRindu(N, Skaits)
    ZimetRindu(N, 3)
    ZimetRindu(N, 3)
    return
```

Tas arī būtu viss. Vēl tikai "saliksim kopā" visu programmu:

```
def ZimetRindu(N,Z):
    # N - rindas garums, Z - * simbolu skaits
    # Aprekinam "atstarpi"
    A = (N-Z)/2
    # Izvadam rindu
    print " " * A + "*" * Z
    return
def ZimetEgliti(N):
    # N - rindas garums
    for Skaits in xrange(1, N+1):
          if Skaits % 2 == 1:
                ZimetRindu(N, Skaits)
    ZimetRindu(N, 3)
    ZimetRindu(N, 3)
    return
while 1:
    # Skaitla ievade
    print "Ievadiet nepara skaitli (>3) (0 - beigt darbu):"
    Skaitlis = input()
    if Skaitlis == 0:
          break
    if Skaitlis >3:
          if Skaitlis % 2 == 1:
                ZimetEgliti(Skaitlis)
          else:
                print "Ievadits para skaitlis!"
    else:
          print "Skaitlim jabut >3!"
```

9. uzdevums. Līzings.

Patiesībā šī uzdevuma risinājums ir ļoti vienkāršs. Kad ievadīti sākotnējie dati (aizņēmuma summa `A`, mēnešu skaits `M` un procentu likme `P`), jāveic šādas darbības:

1. Jāaprēķina mēneša pamatsummas atmaksas apjomu (nosauksim to par `PS`):

```
PS = A/M
```

2. Jāveic ciklu ar `M` iterācijām, kura ietvaros jāaprēķina atmaksājamo procentu apjoms un, to summējot ar pamatsummas atmaksas apjomu, jāizvada katras iterācijas rezultāts. Vienlaikus atmaksājamo pamatsummu jāsamazina un kopējo procentu maksājumu apjomu attiecīgi jāpalielina.

3. Beigās jāizvada kopējo procentu maksājumu apjomu.

```
while 1:
    print "Ievadiet aiznemuma summu:"
    A = input()
    print "Ievadiet menesu skaitu:"
    M = input()
    print "Ievadiet procentu likmi:"
    P = input()
    # Aprekinam menesa pamatsummas apjomu
    PS = A/M
    # Nosakam, ka sakuma atmaksato procentu apjoms ir 0
    ProcentiKopa = 0
    print "Maksajumu grafiks:"
    for Menesis in xrange(1,M+1):
          # Kadi somenes procenti? :)
          Procenti = A * (P/100.0)
          print str(Menesis)+". menesis:", int(PS+Procenti)
          # Samazinam atlikuso pamatsummu
          A = A - PS
          # Palielinam jau atmaksato procentu apjomu
          ProcentiKopa += Procenti
    # Izvadam, cik kopa jamaksa procentos
    print "Kopa procentos jamaksa:", int(ProcentiKopa)
    print "Vai atkartot? (j/n):"
    Darbiba = raw_input()
    if Darbiba == "n":
          break
```

Šeit jāievēro funkciju `str` un `int` lietojums. Tā kā `Python` var uzskatīt, ka ievadītā procentu likme ir vesels skaitlis (`integer`), tad, lai iegūtu daļskaitļa tipa rezultātu, `P` jādala nevis ar `100`, bet ar `100.0`. Pēc tam rezultātu jāizvada kā veselu skaitli, izmantojot funkciju `int`. Tāpat, lai pēc mēneša numura netiktu ievietota atstarpe, to vispirms pārveidojam par teksta rindu un to savienojam ar rindu ". `menesis`".

10. uzdevums. Binārie skaitļi.

Domājams, ka daudziem lasītājiem varētu rasties jautājums - kas tad īsti ir binārā skaitīšanas sistēma? Vienkāršos vārdos - tā ir skaitīšanas sistēma, kurā, atšķirībā no decimālās, tiek izmantoti tikai divi cipari - 0 un 1 (kā zināms, decimālajā skaitīšanas sistēmā tiek izmantoti cipari no 0 līdz 9).

Decimālajā skaitīšanas sistēmā katra nākamā pozīcija ir atbilstošā skaitļa 10 pakāpe. Pakāpes tiek skaitītas no kreisās puses, sākot ar 0. Tā piemēram, 12 ir vienāds ar $1*10^1+2*10^0$. Līdzīgi ir arī binārajā skaitīšanas sistēmā, tomēr tajā katra pozīcija ir atbilstošā skaitļa 2 pakāpe. Piemēram, 1010 ir $1*2^3+0*2^2+1*2^1+0*2^0 = 10$.

Līdz ar to, ja mums ir dots skaitlis decimālajā pierakstā, jārīkojas šādi:

```
1. Jānosaka lielāko skaitļa 2 pakāpi, kas mazāka par doto skaitli.
2. Tālāk iteratīvi "jādodas uz leju" līdz pakāpei 0, veicot šādas darbības:
2.1. Ja tekošā skaitļa 2 pakāpe ir mazāka par aplūkojamo skaitli, tad
atņemam tekošo pakāpi no mūsu skaitļa un pierakstam ciparu 1. Pretējā gadījumā
neko neatņemam un pierakstam ciparu 0.
2.2. Samazinām aplūkojamo pakāpi par 1 un atkārtojam punktu 2.1, ja pakāpe
ir >= 0.
```

Mēģināsim šo algoritmu pierakstīt valodā `Python`.

```
  import math
  while 1:
      print "Ievadiet skaitli (0 - beigt darbu):"
      Skaitlis = input()
      if Skaitlis == 0:
            break
      # Noteiksim lielako skaitla 2 pakapi, kas mazaka par Skaitlis
      Pakape = 0
      while int(math.pow(2, Pakape))<=Skaitlis:
            Pakape += 1
      #  Patiesiba esam ieguvusi mazako skaitla 2 pakapi, kas lielaka par
Skaitlis
      # Tapec atnemsim no tas 1, lai iegutu to, ko esam patiesiba meklejusi
      Pakape -= 1
      # Sagatavosim "binara pieraksta" teksta rindu
      Binarskaitlis = ""
      # Cikls
      for Cikls in xrange(Pakape, -1, -1):
            # Tekosaja pozicija ir binarcipars 1...
            if int(math.pow(2, Cikls)) <= Skaitlis:
                  Binarskaitlis += "1"
                  Skaitlis -= int(math.pow(2, Cikls))
            # ...un 0
            else:
                  Binarskaitlis += "0"
      # Izdrukajam rezultatu
      print Binarskaitlis
```

Nobeigumam

Protams, ka šajā grāmatas daļā aplūkotie 10 uzdevumi ir stipri par maz, lai varētu iemācīties programmēt. Arī to risinājumi ne vienmēr tika optimizēti, par svarīgāku uzskatot labāku izprotamību. Un tomēr - šie ir reālā dzīvē sastopami uzdevumi, kuru risinājumi ir sastādīti valodā `Python`, tādējādi dodot lasītājam iespēju izprast, kā iepriekšējās grāmatas daļās iegūto teorētisko pamatu likt lietā, risinot praktiskus uzdevumus.

Kā vienmēr - atsauksmes (izņemot nekonstruktīvu kritiku) lūdzu sūtīt uz e-pastu `alvilsb@parks.lv`. Īpaši tiks gaidītas norādes uz kļūdām un neprecizitātēm, jo darbu pie šīs grāmatas daļas pabeidzu nedēļu pirms atvaļinājuma, tādēļ jūtos ārkārtīgi noguris. Un, kā zināms, nogurums ir precizitātes ienaidnieks...

Šo un arī citas grāmatas daļas ir iespējams lejuplādēt manā mājaslapā - `http://alvils.latvietis.com/` sadaļā "Rakstu darbi".

Vairāk informācijas par programmēšanas valodu `Python` ir iespējams iegūt `Python` projekta mājaslapā - `http://www.python.org/`