**O‘ZBEKISTON ALOQA VA AXBOROTLASHTIRISH,
TELEKOMMUNIATSIYA TEXNOLOGIYALARI DAVLAT QO‘MITASI
TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI
QARSHI FILIALI**

***Kompyuter injiniring* fakulteti**

Dasturiy injiniring
kafedrasi

Sh.A.Murtozaev

« Ruby dasturlash tili asoslari »

O’quv uslubiy qo’llanma

**“5521900- Informatika va axborot texnologiyalari “
Ta’lim yunalishi uchun
(2-qism)**

Qarshi-2015

Tuzuvchilar: Dasturiy injiniring kafedrasi assistenti Sh.A.Murtozaev
Taqrizchilar: QarMII dotsenti B.Maxmadiev, TATU KF AT kafedrasi mudiri B.Nosirov

Ushbu o’quv uslubiy qo’llanmada Ruby dasturlarsh tilining mohiyati, sintaksisi va semantikasi, ruby dastur kodining dinamik aspekti, qatorlarni qayta ishlash va dastur tuzish keltirilgan. Ruby dasturlash tili hozir dinamik va interaktiv tarkibga ega bo‘lgan Web sahifalarni yaratilishda, katta miqyosdagi tadbirkorlik uchun kompyuter dasturlarini yaratishda, WWW axborot tizimi funksional xizmatlarini kuchaytirishda foydalanuvchining uyali telefonlari, peyjerlari va shaxsiy raqamli yordamchilari kabi qurilmalarini dasturlar bilan ta'minlashda qo‘llaniladi.

Qo’llanma ruby dasturlash tilini o’rganuvchi bakalavr va magistrlar uchun mo‘ljallangan.

Uslubiy ko‘rsatma TATU Qarshi filiali "Axborot texnologiyalari" kafedrasi (bayon №___, ___ _____2014y) va filial uslubiy kengashi (bayon №___, ___ _____2014y) da ko‘rib chiqilib, o‘quv jarayonida foydalanishga va kichik adadda chop etilishga ruxsat etilgan.

Mundarija

Kirish

Bugunki kunda mustaqil yurtimizning har bir fuqarosi mustaqillikni mustahkamlash, jamiyatni rivojlantirish uchun nimalardir qilishi zarur va shart. Mana shu nimalardir har bir insonning o‘z faoliyati va o‘z kasbiga fidoyiligi bilan belgilanadi. Ozod va obod Vatan, demokratik va huquqiy davlat qurish, yurtimizni rivojlangan demokratik davlatlar qatoriga olib chiqishdek buyuk maqsadlar shu aziz yurtda yashayotgan har bir insonning ma'naviy dunyosi, ma'rifiy darajasiga ham bog‘liqdir. Shu sababli ma'naviy pok avlodni tarbiyalash, ma'rifan yuksak kadrlarni tayyorlash asosan ta'lim tarbiya sifatiga bog‘liqdir va qolaversa ta'lim sifati - hayot sifatini belgilovchi omillardan biri bo‘lib hisoblanadi. Shu sababli «Kadrlar tayyorlash milliy dasturi»ning hozirgi, uchinchi bosqichining asosiy vazifasi ta'lim tizimining barcha bo‘g‘inlarida ta'lim sifatini oshirishga qaratilgandir.

Ushbu ma’ruzalar matnida bir qancha dasturlash tillari c++, java, Delphi va shuningdek web saxifa yaratuvchi tillarni urganib chiqqanimiz holda sintaksis jihatdan sodda va urganilishi bir muncha osonroq va JSP, ASP va PHP dan qolishmaydigan Ruby dasturlash tilining mazmunini ochib berishga harakat qildik. Bizningcha ikkita bir-biri bilan bog’langan aspektlar bor: Rubyning falsafiy qurilishi va falsafiy qullanilishi, dizayin va qullanilishi bir-bir bilan chambarchas bog’liq, bu apparat ta’minoti bo’ladimi yoki program ta’minoti bo’ladimi. Agarda biz qurilmani biror ruchka bilan ishga keltirsak, shu ruchkadan foydalanmoqchi bo’lgan yana bir kishi chiqib qoladi deb o’ylaymiz. Ruby tiliga qaraganimizda, fizikadan jismlar haqidagi masalani eslatib turuvchi kirishuvchanlik qiyinchiliklari va har hil proyekt maqsadlari o’rtasidagi muvozanatni ко’ramiz. Bu shunchaki ilm va texnologiya tili emas, balki butun bir madaniyatli programmistlar uchun juda yaxshi til bo’ladi. Biz kompyuter qoidalarida qandaydir ruhiy aspekt bor ! – deyish albatta biz uchun g’alati. Insoniyatning chiroyli va foydali dasturlarni yaratishga bo’lgan ehtiyojiga Ruby manbadir. Ruby da yozilgan har bir programma ana shunday ilhomlantiruvchi manbaidan o’qib o’rganishlari kerak.

1- mavzu. Ruby dasturlarsh tilining mohiyati

reja:

1.1 ob'ektga yunaltirilgan dasturlash haqida
1.2 vorislik
1.3 polimorfizm

Til – bu bizning fikrlash uslubimizni aks ettiruvchi omil va biz nimani fikrlasak shuni aniqlab qoladi.

Benjamin Di Uorf

Shuni eslatish joizki, programmalashtirishda, yangi yaralgan tilda panasiya seziladi, ayniqsa uni moslashuvida. Ammo hech qaysi bir til boshqa bir tilning o'rnini to'liq egallay olmaydi.

Biror bir masala yechimini topishda to'laqonli qo'llaniladigan xech bir uskuna yo'q xali. Muhimi – fikrlash masalalarini yechishning har xil yo'llari mavjud. Ko'p tillar borki – ularni himoya qiluvchi hamda tanqid qiluvchi insonlar ko'p bo'ladi. Xullas, "til jangi"ning oxiri yo'q.

Shunga qaramasdan takomillashgan yozuv programmasi sistemasini topishdagi to'xtovsiz izlanishlar davomida tug'ulib qolgan yangi fikrlar bizni bir muncha qiynab qo'yadi.

Algoldan Pascal ko'p belgilar olgani, yoki C dan Java rivojlangani singari, har bir til oldingi avlodlarda ma'lum qirralarni o'ziga singdiradi.

Til o'z vaqtida uskunalar terilmasi va o'yin maydonidir. Uning malakaviy tomoni bor, ammo shu bilan birga u yangi fikrlar sinov maydonida ham xizmat qiladi, qaysini programmistlar tomonidan ma'qullangan yoki rad etilgan tillar.

Eng keng tarqalgan fikrlardan biri –ob'ektga yo'naltirilgan dasturlash konsepsiyasidir (OOP). Kupchilik OOP ni revalutsion xarakterdan ko'ra ko'proq evalutsion xarakterga ega deyishadi, lekin uni ishlab chiqarishda juda katta ta'sir etganini xech kim rad qila olmaydi. Kup yillar ilgari OOP kupincha akademik qiziqishlar asosida aks etardi, bugun esa universal qabul qilingan paradigma. 1998 yillarda Rodjer King o'zining "Agarda siz biror

kompyuter dasturchisiga mushugingizni sotmoqchi bo'lsangiz, unga mushuk ob'ektga yunaltirilgan deng''. Bu gapida u OOP aslida uzida nima aks etirish haqida fikr yuritgan, terminalogiya bilan bog'liq fikrlar tuqnashuvi xattoki ummumiy nuqtai nazarni ajrata oluvchilar urtasida bo'lib turadi. Bizning maqsadimiz bunday tuqnashuvlarda ishtirok etish emas. OOPni-masalalar yechishda foydali uskuna va qimmatbaxo metodlardir-fikriga albatta qo'shilamiz.

OOPnining asl tabiatiga kelsak, albatta biz sevimli muloxazalarga va terminlarga egamiz, ammo biz faqatgina foydali muommolarda ulardan foydalanamiz.

1.1 *Ob'ektga yo'naltirilgan dasturlash haqida*

Ruby tili haqida suhbatni boshlashdan oldin, obyektiv – mo'ljallangan dasturlar haqida mulohaza yuritsak yaxshi bo'lar edi.

Ob'ektga yunaltirilgan dasturlashda ob'ekt- bu fundamental tushuncha. Ob'ekt bu – ma'lumotlar konteyneriga xizmat qiluvchi va ana shu kiruvchi ma'lumotlarni boshqaruvchi manbadir. Bundan tashqari ob'ekt interfeys va ob'ektning funksional imkoniyatlarini o'zida aks ettirishadi. Bu funksiya – metod (yoki usub) deb nomlanadi.

Shuni aytish joizki, har bir ob'ektiv mo'ljallangan til o'zida inkapsulatsiya mexanizmini o'zida aks ettiradi.

Ob'ekt aslida ob'ekt sinfi namunasi hisoblanadi. Odatda bu sinf (class) deb nomlanadi. Klassni biz chizma yoki namuna deb tasavvur qilsak, ob'ektni mana shu chizma asosida yaratilgan biror predmet deb tasavvur qilsak bo'ladi. Shu bilan birga ob'ektni bu umumlashtirib ramzlar qatorida deb ataymiz.

Ob'ekt yaratishni (clacc namunasi) – instansivlik deb ataymiz. Bazi tillarda ob'ektni yo'q qilishda(destructor) va instalizasiya uchun zarur bo'lgan(konstructor) harakatlarni amalga oshiruvchi funksiyalar –

konstruktorlar va destructorlar mavjud. Bazida shunday holatlar tug'ulib qoladi: berilgan ma'lumot juda katta sig'mga ega. Ya'ni bitta ob'ekt bilan cheklanib qolmaydigan. Bunday holda har bir ob'ektga har bir na'muna joylashtirish notug'ri. Keling My dogs va uning uchta ob'ektlar misolida ko'rib chiqamiz: fido, rover va spot. Har bir it yoshi va vaksinatsiya sanasiga ko'ra shunday atributlar bo'ladi. Ularni albatta har bir ob'ektga joylashtirish mumkin, lekin bu xotirani bekorga sarf qilishdir. Masalan shuni bilish kerakki ba'zi klasslarga nechta ob'ekt yaratilganini har bir ob'ektni yaratishda "klass"ni 1 gacha kattalashtirib yoki uni 0 gacha inisializatsiya qilishda vaqtinchalik klassni yaratsa ham bo'ladi. Bu vaqtinchalik oyna "klass"bilan bog'liq. "Klass" atributlarini oddiy atributlardan ajratib olish uchun oddiy atributlarni ko'pincha ob'ekt atributlari (yoki namuna atributlari) deb nomlashadi.

Yana bir narsani aytish joizki, ba'zi ma'noda hamma uslublar "klass" uslublari hisoblanadi. 100 ta ob'ekt yaratsak yuzlab uslub kodini ko'chirmasini yaratding deb o'ylash noto'g'ri albatta. Ammo ko'rinish chegaralari qoidasi amalda mavjud bo'lib, har bir ob'ekt uslubi faqatgina qaysi so'ralgan bo'lsa shu ob'ekt ma'lumotlarini boshqaradi. Shuning uchun bizda quyidagi mavxumlik paydo bo'ladi: ob'ektlar uslubi ob'ektlarning o'zi bilan uzviy bog'liq

1.2 Vorislik

Endi biz OOP ning asosiy kuchli tomoniga to'xtalsak izdoshlik yangi imkoniyatlarni qo'shish yo'li orqali yuqorida ma'lum bo'lgan biror jismni kengaytira oladigan mexanizmdir! Qisqa qilib aytganda yunalish bu koddan foydalanishning takroriy usulidir. Odatda yo'nalish klass miqiyosida o'rganiladi. Bizga birorta klass kerak bo'lib qolsa, amalda bizda faqat

umumiy ko’rinish bo’lsa bunday hollarda kerakli klassni avvalgilarning xatti harakatlariga qarab aniqlaymiz.

Masalan qavariq ko’p burchakli tasvirlovchi Polygon klassi bor. Bu holda ko’p burchakni ifodalagan Restangle Polygonda bemalol merosxo’r bo’la oladi va shu bilan birga Restangle Polydonning uslub va atributlariga bemalol egalik qila oladi.

Agar b klass a klassning merosi bulsa, biz A klassni-superklass, B klassni tobe klass deb ataymiz. Ko’p yunalishlarni cheklovchi Al’ternativ xam mavjud. Bu uslub kupincha Ruby va Javada qo’llaniladi. Shu bilan birga biz polimorfizm haqida ham unutmaylik.

1.3 Polimorfizm

Agar inkapsulyatsiya va vorislik OOPning foydali vositalari sifatida qarash mumkin bo’lsa, polimorfizm - eng universal va radikal vositadir. Polimorfizm inkapsulyatsiyalash va vorislik bilan chambarchas bog’liq, boz ustiga polimorfizmsiz OOP samarali bo’lolmaydi. Polimorfizm - OOP paradigmasida markaziy tushunchadir. Polimorfizmni egallamay turib OOP dan samarali foydalanish mumkin emas. Ruby ko’pincha interfeysli polimorfizmni qo’llab-quvvatlaydi. U hozirdagi klasslarga qo’llanuvchi metodlar modulini aniqlashga yordam beradi. Odatda modullardan bunday foydalanilmaydi. Modul metod va konstantdan iborat bo’lib, undan shunday foydalanish mumkinki, xuddi ular klass va ob’ektni ajralmas qismidek. Qachonki biz modulni include bilan bog’lasak, biz yo’nalishning cheklangan shakliga ega bo’lamiz.

Ruby bunday sintaksis (konstruksiya) qurilmani ma’qullamaydi. Ba’zi tillar OOPning “toza” modelini qo’llashadi. Bu esa har qanday borliq tilda ob’ekt bo’la olishni anglatadi. Lekin Java, C++ va Eyffel tillarida boshqacha tus oladi.

C++ singari tillarda “mavhum klass” tushunchasi bor. Bunday klassga me’roslik ruxsat beriladi, lekin kuchirmasini yaratish mumkin emas. Ko’p karrali bo’lishi Ruby da bunday tushuncha yo’q, lekin agar dasturchi hoxlasa uni modullashtirish mumkin.

C++ tili asoschisi Bern Straustrup shu bilan birga “aniq tur” tushunchasini ham yaratdi. Bu klass faqat qulayliklar uchun foydalaniladi. Bu klassda hech qanday klass izdosh bula olmaydi. Bunda noaniq tiplar ob’ekt bula olmaydi.

Nazorat savollari:

1. Asosiy tiplarni ko‘rsating.
2. Ko‘rsatkich ta'rifini keltiring.
3. Xamma amallarni ko‘rsating.
4. Ifodalarda tiplar qanday avtomatik keltiriladi?

2-mavzu: Rubyning sintaksis va semantikasi.

Reja:

2.1. Kalit so'zlar va idenfikatorlar.

2.2 Izohlar

Yuqorida aytib o'tkanimizdek Ruby-bu dinamik Ob'ektga yunaltirilgan tildir. Bu tilning sintaksis va semantikasiga o'tishdan oldin, uning boshqa ba'zi moxiyatlarini eslatib o'tsak.

Ruby-progmatik til (adile). U o'zgaruvchan bo'lib qiyinchiliksiz amalga oshuvchi qayta dasturlash qobig'ini kengaytiradi. Albatta kelajakda ishlab chiqarish rivojlanib Rubyning komplyatori xam paydo bulishi mumkin, ammo interpritatorning ko'p qirralari bor. U nafaqat protokollarni tez kashf qilishga balki qayta ishlash tsiklini xam qisqartirishga yordam beradi.

Ruby-tasvirga moslashgan. Tasvir yetarli bulsa, gaplar yozishga ne xojat? Bu shuni anglatadiki-dastur yanada qulaylashadi.

Ruby-yuqori darajali tildir (VHLL). Buning prinsiplaridan biri bu kompyuter insonlarga xizmat qilishi kerak, inson kompyuterga emas.

Bosh dastur (main funksiyasi) mavjud emas, amallar temaning pastga qarab bajariladi. Yanada murakkab dasturlarda tekst boshida bir qancha aniqlovchilar aks ettirilishi mumkin, bosh dasturga yuldosh bo'luvchi.

2.1. Kalit so'zlar va idenfikatorlar.

Rubyda odatda kalit so'zlar biror bir maqsadni amalga oshirishi uchun foydalanilmaydi. Quyidagilar ularning to'liq aksi:

BEGIN END alias and begin
break case class def defined?
do else elsif end ensure
false for if in module
next nil not or redo

rescue retry return self super
then true undef unless until
when while yield

O’zgaruvchi yoki boshqa bir idenfikatorlar nomlari odatda harf yoki maxsus modifikatorlar bilan boshlanadi.

Asosiy qoidalar:

- Lokal o’zgaruvchi nomlar (self yoki nil kabi) qator harfi yoki tagida chizish “__” belgisi bilan;
- Global o’zgaruvchilar $ belgisi bilan boshlanadi;
- O’zgaruvchi kuchirmalari “kuchikcha” @ bilan;
- O’zgaruvchi sinf nomlari @@lar bilan;
- Konstant nomi yozma katta harflar bilan ;
  - Idenfikatorlar nomida tagiga chizish “__” belgisi qator harf bilan baravar ishlatiladi ;
  - Maxsus o’zgaruvchi nomlar ($ belgisi bilan belgilanuvchila) bu yerda qurib chiqilmaydi.

    Misollar :
  - Lokal o’zgaruvchilar **alpha, __ iden, some __ var**
  - Psevdoo’zgaruvchilar **self, nil,__ File __;**
  - Konstantlar **k6chip, length, LENGTH**.
  - O’zgaruvchan ko’chirma **@ foobar, @ Hix 1138, @ not_const ;**
  - O’zgaruvchi klass **@@ phydeaux, @@ my-var, @@_const;**
  - Global o’zgaruvchilar**. $ beta, $ B2 vitamin, $ not-const.**

2.2 Izohlar

Rubyda izohlar. (#) panjara belgisi bilan boshlanadi :

- X = y+5 # bu izohdir.

- # bu ham izohdir.

print “#” lekin bu izoh emas.

Qurilgan hujjatlashtirish dasturdan qandaydir tashqi uskuna yordamida yo’qotiladi deb taxmin qilinadi.

= begin va =end belgisi yordamida boshlanuvchi qatordagi har qanday matn, interpritator tomonidan boshlanadi.

Nazorat savollari:

1. Izox qanday ko‘rsatiladi?
2. Qo‘shma operator ta'rifini keltiring.
3. Kiritish qanday amalga oshiriladi?

3-mavzu: konstantalar, asosiy turlar va o’zgaruvchilar.

Reja:

3.1 massivlar.

3.2 Operatorlar va prioritetlar.

3.3 Ruby dasturiga misol.

Rubyda o’zgaruvchilar ma’lum bir turga ega bo’lmaydi, o’zgaruvchilar yuboriladigan ob’ektlar turlarga qarab tanlaydi(PHP singari). Oddiy tiplar – bu simvollar, raqamlar va qatorlardir. Raqamli konstantlar ancha tushunarli va qator bilan teng. Umumiy qilib aytganda, qo’shtirnoq (“”) bilan tugallangan gaplar izohni interpolyatsiya qilishga ruxsat beradi, a bittalik qo’shtirnoq (‘) bilan tugagan gaplar va deyarli birdaniga unda ekranli qayta ko’ndalang chiziqlar paydo bo’ladi. Qo’shtirnoq bilan tugagan gapning qatordagi o’zgaruvchilar va izoxlarning interpolyatsiyasi quyidagicha ko’rsatiladi :

```
a = 3
b = 79
puts "#{a} ko’paytirilgan #{b} = #{a*b}" # 3 ko’paytirilgan 79=237
```

Eslatib o’tish joizki, sal kattaroq dasturlarni birlashtirishdagi unchalik katta bo’lmagan senariylar uchun qator xilma-xilligi katta axamiyatga ega. Dasturdan ajratib olingan qator buyruqlarni ijro etish uning operatsiya sistemasiga junatiladi. Oddiy xollarda qator teskari qushtirnoq bilan belgilanadi. Murakkab xollarda sintaksis konstruksiya foydalaniladi. % x:

```
`whoami`
`ls -l`
%x[grep -i meta *.html | wc -l]
```

Rubyning muntazam izoxlari ramziy qatorlarga o’xshaydi, lekin boshqacha foydalaniladi. Odatda chegaralovchi vazifasini kundalang

chiziqlar boshqaradi. Ruby va Perldagi muntazam sintaksis izoxlari ko’p umumiy jixatlarga ega.

Rubydagi massivlar juda kuchli konstruksiya ; ular xoxlagan tipdagi ma’lumotlarni o’zida saqlay olmaydi. Bundan tashqari bitta massivda xar xil tipdagi ma’lumotlar saqlanishi mumkin.

Massiv-konstanta to’rtburchak qavslar orqali chegaralanadi:

```
[1, 2, 3]
[1, 2, "hello world!"]
[1, 2, [3,4], 5]
["alpha", "beta", "gamma", "delta"]
```

– misolda massiv ifodalangan (butun bir son 1 va qator bilan) 3 – misolda qo’yilgan massiv ifodalangan, a 4 – misolda qator massiv ifoda etilgan.

Oxirgi misoldagi “gamma” elementi 2 indeksiga ega hamma massivlar dinamik bo’lib, yaratishda o’lcham so’rovini hojati yo’q. Qator massivlarni juda ko’p uchraganligi sababli (yanayam ularni terish qiyin), ular uchun maxsus sintaksis yaratilgan :

```
%w[alpha beta gamma delta]
%w(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec)
%w/am is are was were be being been/
```

Bu yerda qavslar ham, vergullarni ham keragi yo’q. Elementlar ochiq qoldirishi bilan ifodalanadi.

Aniq massiv elementiga kirish uchun to’rtburchak qavslardan foydalaniladi :

```
val = myarray[0]
print stats[j]
x[i] = x[i+1]
```

Rubydagi yana bir sexrli konstruksiyalarning bu XESH dir. Shuningdek uni “ lug’at ” deb ham atashadi.

U odatda so’rov jadvali sifatida foydalaniladi. Yoki umumiy massiv singari massiv. Hamma Xeshlar Hash sinflar ekzemplyardir.

XESH – konstanta ham, shaklli qavslar bilan ifodalaniladi, kalitlar esa => belgisi bilan.

Kalitni kirishda indeks deb hisoblasa ham bo’ladi. Tip kalitlari va izohi uchun chegaralash shart emas. Misollar : {1=>1, 2=>4, 3=>9, 4=>16, 5 = >25, 6=>36}

{"cat"=>"cats", "ox"=>"oxen", "bacterium"=>"bacteria"}

{"vodorod"=>1, "geley"=>2, "uglerod"=>12}

{"toq"=>[1,3,5,7], "juft"=>[2,4,6,8]}

{"foo"=>123, [4,5,6]=>"my array", "867-5309"=>"Jenny"}

print phone_numbers["Jenny"]

plurals["octopus"] = "octopi"

Quyidagi xeshdan uzgaruvchi ruxsat xuddi massivdagi kabi katta to’rt burchak qavslar bilan ifodalaniladi : ammo shuni aytish joizki, massiv va xeshlarda ko’p uslublar bo’lib, aynan shular konteynerlarni foydaliligi uchun xizmat qilishadi.

3.1 Operatorlar va prioritetlar.

Ma’lumotlarning asosiy tiplari bilan tanishib, Ruby tili operatorlariga o’tsak. Quyidagi berilgan ro’yxatda ular ahamiyati ketma ketligi ko’rsatilgan :

: : - ko’rinish qismiga ruxsat.

[ ] – olingan indekslar.

** - daraja

+-!~ unar musbat / manfiy inkor

* / % butun va qoldiq bo’lish.

<< >> mantiqiy (xarakat) surilish.

& - mantiqiy va

||^ mantiqiy ИЛИ (yoki)

. . . . . – diapozon operatori.

? : - ternar tanlov.

not – mantiqiy inkor.

and or – mantiqiy и,или. (va, yoki)

berilgan belgilarning birnechtasi birvaqtning o'zida bir qancha vazifani bajaradi. Masalan, operator <<- chap tomonga surulishni, shu bilan birga tayyor xujjatda yug'on chiziq vazifasini bajarib kelyapti.

Mavhum + belgisi sonlar tartibi va qator konpatenasiyasini anglatadi. Ko'p operatorlar – bular shunchaki chaqiriq uslubining qisqa yozuvidir.

Ruby dasturiga misol.

Istalgan dasturlash tili o'rganila boshlaganilganda dastlab Hello world suzini konsol oynasiga chiqarish dasturi bilan boshlanadi. Quyidagilar misolida yani bu unchalik katta bo'lmagan interaktiv konsol dastur bo'lib, u xarakatni Farengeyta shkalasidan selsiy shkalasiga o'tkazishda foydalandik :

```
print "harorat va shkalani kiriting (C or F): "
str = gets
exit if str.nil? or str.empty?
str.chomp!
temp, scale = str.split(" ")

abort "#{temp} aniq son emas." if temp !~ /-?\d+/

temp = temp.to_f case scale
 when "C", "c"
  f = 1.8*temp + 32
 when "F", "f"
  c = (5.0/9.0)*(temp-32)
```

```
 else
  abort "olingan malumot C yoki F."
end
if f.nil?
 print "#{c} gradus C\n"
else
 print "#{f} gradus F\n"
end
```

Quyida esa mana shu dastur teskarisi berilgan. Shu bilan birga noto'g'ri ifodalangan sonni qayta to'g'irlash usuli ham berigan :

Harorat va shkalani kiriting (C orF) : 98 . 6F 37.0 harorat

Harorat va shkalani kiriting (C orF) : 100 C 212.0 harorat F

Harorat va shkalani kiriting (C orF) : junk F junk – bu mumkin bo'lmagan son.

Endi bu dastur qanday ishlashini ko'rib chiqamiz. Hammasi print so'zi x bilan boshlanadi. kernel moduldagi print singari. Berilgan uslub standart chiqishda bosmachilikni amalga oshiradi. Bu qator oxirida pursorni qoldirishning eng oddiy uslubidir.

Keyinchalik biz gets uslubini chaqiramiz, qabul qilingan o'zgaruvchan izoh str dan foydalanib, yangi qatordan dumli ramzlarni o'chirish un champ ! uslubidan foydalanamiz.

Diqqat qiling, "erkin" funksiyalardek ko'ringan print va gets – Objekt sinfining uslublaridir.

Xuddi shu kabi chomp ! ham str ob'ekti nomidan chaqirilgan uslubdir. Rubyda uslublar chaqiruvi odatda qavslarni tushurishi mumkin : prin "foo" va print ("foo"). O'zgaruvchi Strda ramziy qatorlar saqlanadi, lekin istalgan singari ma'lumotlar saqlanishi ham mumkin edi. Rubyda ma'lumotlar tipga ega lekin uzgaruvchanlar ega emas. Interpretorlar qabul qilganlaridan keying

o’zgaruvchilar paydo bo’lishadi ; ungacha hech qanaqa taxminiy e’lonlarga ega bo’lmaydi.

Exit uslubi dasturni tugatadi. Shu qatorda yana biz “modifikator if” deb nomlanuvchi boshqaruvchi qatorni ham kurishimiz mumkin. if modifikatori uchun elsedan foydalanilmaydi, va u yechishni ham talab qilmaydi. Shartga kelsak biz ikki narsani aniqlashtiryapmiz : uzgaruvchi str biror ma’noga egami yoki bo’sh qatormi.

Bu taxminni yana boshqacha qilib ifodalasak ham bular edi :

Exit if not str or not str [0]

Rubyda o’zgaruvchilar nil, ahil ma’noga ega bo’lib, mantiqiy konteksdan “yolg’on” deb ajraladi, shu sababli yuqoridagilarning shuni aniqlashda ishda foydalaniladi. Aslida “yolg’on” nil yoki False ko’rinishida ko’rinadi, qolganlari esa “haqiqat” sifatida bo’ladi. Bu bo’sh qator ”” va raqami O- “yolg’on” emasligini anglatadi. Keying qatorda chomp ! operasiyasi amalga oshiriladi. So’z oxiridagi ! belgisi operatsiyasi qatordan o’zining ma’nosini o’zgartirishidan, yangisini qaytarmasligidan dalolat beradi. ! belgisi bunga o’xshash ko’p vaziyatlarda ishlatiladi, u shu bilan birga u dasturchiga uslubda nojo’ya ta’sirlar borli haqida, u “xavfli” ekanligini ko’rsatib turadi. Keyingi gapda biz ko’p aniqliklarni ko’ramiz. Spilt uslubi qatorni bo’sh oraliqlarga bo’lib tashlaydi va massivni qaytaradi. Operatorning chap tarafidagi ikkita o’zgaruvchi izohi o’ng qismidagi ikkita elementni o’zlashtiradi. Keying chapda ko’p o’zlashtirishlarga misollarni ko’ramiz.

Keying gapda oddiygina doimiy ibora bo’lmish if yordamida ruxsat etilgan raqam kiritilganmi yoki yo’qligi aniqlanadi. Agar namuna zarur bo’lmagan manfiy belgilardan tashkil topgan bo’lib qator bilan to’g’ri kelmasa, bu holda raqam ruxsat etilmagan hisoblanib, dastur tugatiladi. If gapimiz kalit so’z end bilan tugaydi. Biz enddan avval elseni qo’shsak ham

bo’ladi. Then kalit so’zi shart emas ; bu kitobda biz uni ishlatmaslikga harakat qilamiz.

to – f uslubi qatorni sonlar va suzuvchi nuqtalar ko’rinishiga o’zgartiradi. Bu raqam xuddi o’sha o’zgaruvchan tempga yozib olinadi yani avval sasiangan qator.

Aytkanga, Rubyda case misolida ko’rsatilgandan ham ko’proq xususiyatga ega. Ma’lumotlar tipida hech qanaqa chegaralash yo’q, lekin xamla iboralar qiyin bo’lishi mumkin.

Hisobning o’zida qiziqarli hech narsa yo’q. Lekin bir narsaga etibor bering, o’zgaruvchan e va f birinchi marta case shoxlarida to’qnashayapti. Rubyda hech qanaqa e’lonlar yo’q, o’zgaruvchan faqat o’zlashtirishlaridan keyin paydo bo’ladi. Bu degani casedan chiqilgandan keyin faqat bitta o’zgaruvchi elif o’z ma’nosiga ega bo’ladi.

Biz 04 faktdan foydalanishimizdan maqsad; qaysi shox amalga oshirilganini tushunish va bundan kelib chiqib u yoki bu xabarni yozishdir.

f , c , nil larni solishtirishdan, biz o’zgaruvchilarning o’z ma’noli ahamiyati bor – yo’qligini tushunib olamiz.

Bu usul imkoniyatlarni namoyish qilishda qo’llaniladi; istakga qarab bosman case ni ichiga ham joylashtirsa bo’lar edi. Etiborli o’quvchi biz faqat “lokal” o’zgaruvchilarning foydalanganimizni birdaniga sezadi. Bu g’alati tuyulishi mumkin, bir qarashda, ularning ko’rinishi zonasi butun dasturdir. Aslida ular dasturning yuqoridagi darajasiga nisbatan “lokal” dirlar.

Bu oddiy dasturda past darajadan tashqari konteks bo’lmagani uchun ular global bo’lib ko’rinishlari mumkin. Lekin bir qandaydir sinf yoki uslublarni ma’lum qilganimizda edi, bu xolda yuqori darajadagi o’zgaruvchilar ko’rinmagan bo’lardi.

3.1.1 Takrorlanish(For) va tarmoqlanish(IF)

Endi boshqaruvchilar tuzulishiga ozgina e’tibor bersak. Biz oddiy if taklifi va if modifikatorini ko’rib chiqdik ular uchun unless kaliy so’zining foydalaniladi. Ularning hammasi 1.1. jadvalida berilgan.

1.1. Shartli gaplar.

If ning formasi	**Unless ning formasi**
if x < 5 then statement1 end	unless x >= 5 then statement1 end
if x < 5 then statement1 else statement2 end	unless x < 5 then statement2 else statement1 end
statement1 if y == 3	statement1 unless y != 3
x = if a>0 then b else c end	x = unless a<=0 then c else b end

Bu yerda kalit so’zlar bo’lmish if va unless 1 qatorda joylashgan bo’lib, deyarli bir xil aniqlikda funksiyalarni bajaradilar. E’tibor bering, then so’zini har xil holatlarda ishlatishimiz mumkin. Yana e’tibor qiling, modifikatorlarda else shoxi bo’lishi mumkin emas.

Rubyda case boshqa tillarga nisbatan ko’p vazifalarni bajarishi mumkin. Uning shoxlarida ko’p shartlarni tekshirish mumkin (nafaqat solishtirishi yoki tenglashtirish). Misolni ko’rib chiqaylik :

```
Case “bu belgilar qatori”.
When “bitta ma’no”.
Puts “shox  1”
When “boshqa ma’no”.
Puts “shox 2”
When / ramz /
Puts “shox 3”
Else
Puts “shox 4”
End.
```

Bu pod shox 3 ni nashr qiladi. Nega Deysizmi ?

Avvalo, tekshiriladigan izoh ikkita qator bilan solishtirib ko'riladi : "bitta ma'no" va "boshqa ma'no" .

Bu tekshiruv omadsiz yakunlanadi va shuning uchun biz 3chi shoxga o'tamiz. U yerda ma'nolarni taqqoslovchi namuna bor. Ularning na'munaga to'g'ri kelishiga qarab, print amalga oshiriladi. Oldingi tekshiruvlarning hech qaysi amalga oshirilmasagina, else shaklini obrabotka (tozalaymiz) qilamiz.

Agar tekshirilayotgan izoh butun 1 raqam bo'lsa, u xolda uni butun sonli diapazonga tushayaptimi yo'qmi deb.

Rubyda – juda boy skill tuzulma to'plamlariga ega. Masalan, while va until – taxminiy tekshirish sikllari bo'lib, ikkalasi ham odatiy hollarda ishlaydi ; 1- holda izoh sikli sharti beriladi, a 2- holda – tugatish sharti. Shu bilan birga ularning modifikatorli shakli ham bor, if uchun hamda unless uchun. Bundan tashqari kernel modulida loop uslubi bor va ba'zi sinflarda interatorlar tashkil qilingan.

1.2 Modulida qayerdadir listdan massivi aniqlangani taxmin qilinadi :

list = %w[alpha bravo charlie delta echo];

Siklda bu massivni har bir elementi nashr qilinadi.

1.2. Jadval. Sikllar.

# Tsikl 1 (while)	**# Tsikl 2 (until)**
i=0 while i < list.size do print "#{list[i]} " i += 1 end	i=0 until i == list.size do print "#{list[i]} " i += 1 end
# Tsikl 3 (for)	**# Tsikl 4 (итератор 'each')**
for x in list do print "#{x} " end	list.each do |x| print "#{x} " end
# Tsikl 5 (метод 'loop')	**# Tsikl 6 (метод 'loop')**
i = 0 n=list.size-1 loop do print "#{list[i]} " i += 1 break if i > n	i=0 n=list.size-1 loop do print "#{list[i]} " i += 1 break unless i

end	<= n end
# Tsikl 7 (итератор 'times')	**# Tsikl 8 (итератор 'upto')**
n=list.size n.times do |i| print "#{list[i]} " end	n=list.size-1 0.upto(n) do |i| print "#{list[i]} " end
# Tsikl 9 (for)	**# Tsikl 10 ('each_index')**
n=list.size-1 for i in 0..n do print "#{list[i]} " end	list.each_index do |x| print "#{list[x]} " end

Bu misollarning yanada chuqurroq ko'rib chiqamiz. Sikllar 1 va 2 standart shaklli white va untilar bo'lib; ular o'zini deyarli bir xil tutishadi, faqat bitta shartda qarama qarshi. 3 va 4 sikllari – avvalgilarning variant bo'lib, ular oxirida tekshirish shartlari bor bo'lib, faqat interasiya boshida emas.

Eslatib o'tamiz bu yerda begin va end so'zlaridan foydalanish – bu shunchaki samarasiz urunishdir. Menimcha 3 va 4 kostruksiya – siklni kodlashda eng "to'g'ri" usuldir. Ular boshqalariga nisbatan sezilarli darajada soda; birorta ham aniq inisializasiya, aniq tekshiruvi yoki inkrementasi yo'q. Bu balkim massiv o'z "o'lchamini" bilgani uchundir, standart iterator each bunday detallarni avtomatik tozalab tashlaydi. Aslida 5 siklda bu interpritatorlarga nisbatan noaniq murojaat ishlab chiqiladi. For sikli esa istalgan iterator bilan ishlay oladi. For sikli - each chaqiruvining qisqa yozuvidir.

Ko'pincha bunday qisqartmalar "sintaksis glazur" deb nomlanadi.

5 - 6 sikllarda loop konstruksiyasidan foydalaniladi.

7 – 8 sikllarda – massivning raqamli indeksi bor – faktidan foydalaniladi. Times iterator berilgan raqamni 1 marta amalga oshiradi, upto iteratori o'z parametrlarini (o'lchamlarini) berilgan izohgacha o'zgartiradi. Unisil bunisiyam yaxshi emas.

9 – sikl – diapazon ko'rsatkichi yordamida indeks izohi bilan ishlash uchun mo'ljallangan for sikli variantidir.

Avvalgi misollarda while va loop sikllariga yetarlicha e’tibor qaratmadik. Ular qisqaliklari uchun ancha tez tez ishlatiladi. Quyidagi 2ta misollarda bir xil xolat amalga oshiriladi.

Perform – task () until finished.

Perform – task () white not finished.

Shu bilan birga 1.2 jadvalida sikllarga har doim boshdan oxirigacha bajarmasliklari noaniq qoldi. Siklni boshqarishda qo’shimcha vositalar ham zarur.

Ularning 1chisi – kalit so’z breakdir, u 5 - 6 sikllarda uchraydi.
U sikldan muddatidan oldin chiqishga yordam beradi. Qistirilgan sikllarda chiqish eng tubga amalga oshiriladi. Dasturchilar uchun bu oddiy holdek tushunarli.

Retry – kalit so’zi 2 xil vaziyatlarda qo’llaniladi : iterator konteksi va begin – end bloki konteksida. Interator qismida (yoki for siklida) u (yangi retry) interatorni qayta inisializasiyani amalga oshirishga majbur etadi. Eslatib o’tamiz bu umumiy ko’rinish sikliga ta’luqli emas.

Redo kalit so’zi – bu umumiy ko’rinishi siklida retryni umumlashtirishdir. U xuddi iteratorlarda retry retry ishlagani kabi, white va until sikllarida ishlaydi.

Next kalit so’zi – eng tubdagi siklning oxiriga o’tishni amalga oshiradi va xuddi shu nuqtadan amalga oshirishni yangilaydi. Xohlagan sikl va interatorda ishlaydi.

Hozirgina ko’rganimizdek, iterator – Rubyda muhim tushunchadir. Shuni aytish joizki, til – foydalanuvchi interatorlarni aniqlashda foydali, nafaqat ko’rishganlar bilan chegaralanib qolmasdan.

Istalgan ob’ektlar uchun standart iterator each deb nomlanadi. Lekin iteratorlarga boshqa nom berish ham mumkin va xar hil maqsadlarda foydalanish mumkin. Misol qilib qo’sh maqsadli iteratorni olamiz :

```
j=0
repeat (j >= 10) do
 j += 1
 puts j
end
```

Bu misollar yield blok chaqiruviga xizmat qiladi. Yield kalit so'zi yordamida parametrlarni ham jo'natish mumkin (parameter bloki taxlangan ro'yxatlar) keyingi suniy misolda iterator shunchaki 1 – 10 bo'lgan raqamlarni boshqaradi, chaqiriq interatori bu sonlarni kubik darajasini aniqlaydi def

```
my_sequence
 for i in 1..10 do
  yield i
 end
end

my_sequence {|x| puts x**3 }
```

Eslatish joizki, blok tugagan shaklli qavs o'rniga kalit so'z do va enddan foydalanish mumkin.

3.1.2 Istisnolarni qayta ishlash

Xuddi boshqa tillar kabi Ruby ham istisnolarni qo'llab quvatlaydi. Istisno – bu xatolarga ishlov berish mexanizmi bo'lib, avvali qulayliklardan afzalligi bilan ajralib turadi. Bizga xatolar kodini takrorlanishidan va ularning analizidagi chalkashib ketgan mantiqidan qutulishiga erishdik ; xatoni aniqlangan kod unga ishlov bergan koddan ajralib turadi.

Raise istisno xosil qiladi. Eslatib o'tamiz raise – bu zaxiradagi so'z emas, kernel moduli uslubidir.

Raise # Misol

Raise “xato bo’ldi” # Misol 2

raise ArgumentError,”boshqa son”# Misol 3

Rraise ArgumentError.new “notug’ri ma’lumotlar ” # Misol 4 nm

raise ArgumentError, " notug’ri ma’lumotlar ", caller[0] # Misol 5

Birinchi misolda oxirgi uchragan istisno qayta paydo bo’ladi. 2 – misolda Runtime istisnosi qaratiladi, qaysiki, “xato bo’ldi” xabari jo’natilgan 3 misolda tip istisnosi Argumentdan Eror harakatga keltiriladi. 4 misolda esa xuddi shunaqa istisno bo’lib, lekin “noto’g’ri ma’lumotlar” xabari keladi. 5 misol – shunchaki 4 misolning boshqacha yozilishi. Va oxirgi 6- misolda yo’naltirilgan ma’lumot “file name : line” yoki “file name : line : in I metod” ko’rinishida qo’shiladi.

Rubyda istisnoga qanday ishlov beriladi ? Bu maqsad uchun begin – end bloki xizmat qiladi. Sodda qilib aytganda uning ichida hech narsa yo’q, koddan tashqari :

begin

 # foydali hech narsa

 # . . . . .

end .

Shunchaki xatolarni topadi. Lekin blokning bir nechta ishlovchi mutaxasislari bo’ladi. Rescue . Agar dasturda begin va rescue orasida biror xato bo’lsa boshqaruv darxol mos keluvchi biror 1 ishlov beruvchiga beriladi

```
rescue :
begin
 x = Math.sqrt(y/z)
 # ...
rescue ArgumentError
 puts "kvadrat ildizda hato."
```

```
rescue ZeroDivisionError
 puts "nolga bulish buldi"
end
begin
 x = Math.sqrt(y/z)
 # ...
rescue => err
 puts err
end
```

Bu yerda o’zgaruvchi errda ob’ekt – istisno saqlanadi uni bosmaga chiqarishdan oldin, ob’ekt mantiqiy ramzlar qatorida qayta shakl beriladi. Eslatamiz, tip xatosi berilmaganda, bu rescue ishlov beruvchisi hamma istisnolarni ushlab qoladi, Standart Error sinfida ishlab chiqarligan, Rescue konstruksiyasida => variable, => ramzi oldidan qushimcha tip xatosini aniqlash mumkin.

Agar tip xatosi berilgan bo’lsa, haqiqiy paydo bo’lgan tip istisnosi ularning hech biri bilan to’g’ri kelmasligi mumkin. Bunday hollarda hamma ishlov beruvchilarning rescue dan keyin else ni joylashtirishga ruxsat beriladi.

```
begin
 # yozilga kod va unda uchraydigan error...
rescue Type1
 # ...
rescue Type2
 # ...
else
 #Прочие исключения...
end
```

Biz ko’pincha xatolarning keyin tez tiklanishini xohlaymiz. Bunday xollarda bizga kalit so’z ***retry*** yordam beradi. U bizga begin blokka qayta kirishga va operasiyasini yana 1 marta amalga oshirishga yordam beradi :

```
begin
 # kod, va undagi hatoning bulishi mumkin qismi...
rescue
 # qayta tiklash...
 retry # yana bir marta takrorlash.
end
```

Va nixoyat, ba’zida begin – end bloki amalida keyin “tozalaydi” kodi zarurdir. Bunday vaziyatda ensure qismini qo’yish mumkin :

```
 begin
 # KoD
rescue
 # Istisnoni qayta ishlash.
ensure
 # bu kod barcha jarayonda ishlatiladi
end
```

Ensurening ichki qismiga aralashib ketgan kod begin – end blokidan chiqishda istalgan usulda amalga oshiriladi – istisno bo’lgan yoki bo’lmaganiga qaramasdan.

Istisno yana ikki usulda amalga oshirish mumkin 1 chida modifikator ko’rinishida rescue shakli bor : x = a/b rescue puts (“nolga bo’lish”).

Bundan tashqari, aniqlash uslubi jismi o’zi bilan tushunarsiz begin – end blokini yuzaga keltiradi; begin so’zi tushgan, a uslubning qolgan jismi istisno ishlov berishga tayyorlanadi, va end so’zi bilan tugatiladi : x = a/b rescue puts("nolga bo’lish bo’ldi!")

Bunda biz istisnoga ishlov berishni muhokamadek tugatamiz, shuningdek sintaksis va semantikani 1 butun kurib chiqqandek.

Rubyning ko'p qirrali nuqtai nazari bo'lib, biz ularga hali til tekkizganimiz yo'q. Bobning qolgan qismi tilning imkoniyatlariga bag'ishlanadi.

3.2 Rubyda OOP.

Ruby tilida ob'ektga – mo'ljallangan tillar bilan bog'lovchi ko'p elementlar bor; Ular : inkapsulyasiyali ob'ektlar va ma'lumotlarni yashiruvchi ob'ektlar, polimorfizmli uslub, sinflari izdoshlari. Lekin klass yaratishda chegaralangan imkoniyatlarni qo'shib Ruby olg'a ketaveradi.

1.3.1. Ob'ektlar

Rubyda hamma raqam , qator , massiv , doimiy izohlar va ko'pgina boshqa borliqlar aniq ob'ektlar chaqirig'idan tashkil topgan. 3.succ

```
# 4
"abc".upcase          # "ABC"
[2,1,5,3,4].sort      # [1,2,3,4,5]
someObject.someMethod # birorta rezultat
```

Rubyda har bir ob'ekt o'zi bilan qaysidir sinf nusxasini namoyish etadi. Sinf uslubni amalga oshirishni o'z ichiga oladi :

```
 "abc".class       # String
"abc".class.class # Class
```

Shaxsiy atributlar inkapsulyasiyasi va ob'ekt operasiyasidan tashqari. Ruby, ajoyib identifikatorga ega : "abc".object_id # 53744407

Bu ob'ekt idenfikatori odatda dasturchi uchun qiziq emas.

ichki Sinflar.

Rubyda 30 dan oshiq ichki sinflar tuzulgan. Xuddi boshqa ob'ektga yunaltirilgan tillar kabi, Rubyda ham ko'p izlanishlarga yo'l qo'yilmaydi,

lekin bu til kam ma’noli bo’lib qoladi degani emas. Zamonaviy tillar ko’pincha yolg’iz izlanish modeliga asoslanib quriladi. Ruby keying bobda muxokama qiladigan modullar va sinflar aralashmasini qo’llab quvvatlaydi. Shu bilan birga ob’ektlar identifikatori ishlab chiqarilgan :

Mavjud sinflarda ob’ektlar yaratishda new uslubi qo’llaniladi:

myFile = File.new("textfile.txt","w")

myString = String.new(“bu qator ob’ekti ")

Lekin uni har doim ham chaqirish shart emas. Strning ob’ektni yaratishda qisman mumkin, va bu uslubni eslatmaslik kerak :

yourString = " bu qator ob’ekti "

aNumber =5 # bu yerda new metodi shart emas

Ob’ektlar havolalarda saqlanadi. Yuqorida aytgan edik, o’zgaruvchilar tipga ega emaslar va ob’ekt bo’la olishmaydi – ular shunchaki ob’ektlarga yuboriladi. Bu x = “abc” qoidada istisno bor : ba’zi sinflardagi unchalik katta bo’lmagan o’zgarmas ob’ektlar (masalan Fixnim) ya’ni ular yuborilgan o’zgaruvchilarda to’g’ridan to’g’ri ko’chirma olinadi. Bunday vaziyatda o’zlashtirish vaqtida ob’ekt ko’chirma olinadi : ob’ektlarda o’zgaruvchi ssilkalarni o’zlashtirishda umumlashtiradi :

y = "abc"

x = y

x # "abc"

O’zlashtirishni amalga oshirishdan keyin x = y va x, y bir xil ob’ektga yuboriladi : **x.object_id # 53732208**

y.object_id # 53732208

Agar ob’ekt o’zgaruvchan bo’lsa, bitta o’zgaruvchi uchun ishlatilgan modifikator boshqasi qiyofasida ko’rinadi : x x.gsub!(/a/, "x")

y # "xbc

Lekin bu o’zgaruvchilardagi o’zlashtirish bir biriga o’xshamaydi :

xozirgi misol davomi.

X = abc

Y = avvalgidek “xbc” ga teng.

O’zgaruvchi ob’ektni freerc uslubi yordamida o’zgarmas ob’ektga aylantirish mumkin: x freesze // - // - Rubyga simvollarni uzgaruvchiga nomlari bilan yuborilmaydi. Ko’p hollarda ular umuman identefikatorlarga yuborilmaydi. Simvol **to-s** yordamida qator ko’rinishiga o’tishlari mumkin.

Hearts = : Hearts # Bu o’zlashtirishning yana bir usuli.

Clubs = : Clubs # konstantadagi ajoyib izox.

Diamonds = : Diamonds # qandaydir utqazish analogi.

Spades = : Spades # Pascal yoki C tillarida.

Put s Hearts. Yu-s # “Hearts” deb yoziladi.

Yuqorida ko’rsatilgan “o’tkazish” bilan bo’lgan sehr Ruby da rivojlanishning erta bosqichlarida ko’zda tutilgan , hali Symbol sinfi mavjud emas edi.

3.2.1 Modullar va sinflar – aralashmasi .

Ko’p ko’rilgan metodlar avvalgi sinf-vorislaridir. Ayniqsa Object super sinfiga aralashib ketgan kernel uslub moduli. Object sinfiga hamma joyda mumkin bo’lib, unga qushilgan kernel uslublari ham hamma joyda mumkin. Bu uslublar Ruby da muhim ahamiyatga ega.

“Modul” va “aralashma” terminlari – deyarli bir-biriga sinonimdir. Modul – uslub va konstantlar to’plamlarini Ruby da namoyish etadi. Uni shunchaki nomlari xilma-xilligini boshqarishi uchun ishlatish mumkin, lekin modullardan foydalanish “aralashma” bilan bog’liq(#include direrktivasi yordami bilan).

Bunday xollarda undan sinf - aralashmasi sifatida foydalaniladi. Bu termin Piton tili bilan ham bog’liq. EHMda o’tiladigan “modul” terminini

bizning foydalanish “moduli” bilan aralashtirmang. Ruby da modul – dastlabki tashqi tekst yoki ikkilangan fayl emas. Bu ob’ektga yunaltirilgan, yana qandaydiram sinfga o’xshash abstraktlikdir.

Nomlar boshqaruv maydonidan foydalanish uchun Math moduli misol uchun xizmat qiladi. & # 960 raqamini aniqlash uchun. Math modulida include yordamida foydalanmasa ham bo’ladi, ***Math :: PI*** deb yozsa kifoya. Aralashma uning muammolarini qiyinlashtirmaydigan ko’p me’roslardan foyda ko’rish uslubini beradi. Bu aralashmaning ko’p me’rosdan chegaralangan shakli deb hisoblash mumkin, lekin Mats yaratuvchisi buni xollari 1 ta me’rosni har-hil rivojlangan shaklidir deydi. Eslatish joizki include faqat ko’rsatilgan maydondagi nomlardan quyish mumkin.

Extend uslubi faqat modul ichidan funksiyalarni qo’shadi. ***Include*** dan foydalanish hollarida modul uslublari huddi nusxa uslubidek bo’ladi, a extenda esa – sinf uslublari singari bo’ladi.

Load va require operatsiyalaridan modullar bilan xech qanday umumiy jihatlari yuq: Ular chiquvchi va ikkilanuvchi fayllari bilan bo’g’liq. Load operatsiyasi faylni o’qiydi va uni chiquvchi tekst nuqtasiga quyadi, shuning uchun tashqi fayldagilar huddi shu nuqtadan boshlab hamma aniqliklar ko’rina boshlaydi

1.3.4. Sinflar bilan ishlash.

Rubyda ko’plab tuzilgan ichki sinflar bor, va siz o’zingiz yangisini aniqlashingiz mumkin. Yangi sinfni aniqlash uchun quyidagi konstruksiya qo’llaniladi:

```
class ClassName
# ...
end
```

Sinf nomi – o’zi bir global konstantadir, shuning uchun u bosh (ya’ni katta) harf bilan boshlanishi kerak. Ma’lum bir sinflar o’zida konstantalarni

o’zgaruvchan sinflarni, sinf uslublarini, o’zgaruvchan nusxalarni va nusxa uslublarini saqlay oladi. Berilgan sinflar darajasi o’sha klassning hamma ob’ektlariga tegishlidir, bu holda berilgan nusxa darajasi faqatgina bitta ob’ektga tegishlidir.

Yo’l-yo’lakay eslatma : jiddiy qilib aytganda Rubyda sinflar o’z nomlariga ega bo’lmaydi.

Sinf “nomi” bu shunchaki ob’ektga jo’natilgan class nomi - konstantadir. Bitta sinfga bir nechta konstantalar yuborilishi mumkin, va bu konstantalarning xuddi biz boshqa ob’ektlarni o’zgartirganimiz singari, o’zgaruvchanlarga o’zlashtirish mumkin. Oddiy sinf quyidagicha aniqlanadi :

```
class Friend
 @@myname = "eshmat"          # klas o’zgaruvchisi

 def initialize(name, sex, phone)
  @name, @sex, @phone = name, sex, phone
  # bu o’zgatuvchi nushasi
 end

 def hello # metod nushasi
  puts "salom, men #{@name}."
 end

 def Friend.our_common_friend # klass metodi
  puts "biz hammamiz do’stmiz #{@@myname}."
 end
end

f1 = Friend.new("toshmat","F","555-0123")
```

f2 = Friend.new("Xoji","M","555-4567")

f1.hello ***# salom men, toshmat.***
f2.hello ***#salom men, xoji.***
Friend.our_common_friend ***# biz hammamiz dustmiz eshmat.***

Darxaqiqat ushbu berilgan sinflar darajasi boshqa hamma sinflarga ham tegishli bo'lib ularni klasslarni aniqlash vaqtida inisializasiya qilish mumkin.

Agar initialize nomi bilan uslub aniqlangan bo'lsa, uni ob'ekt uchun xotira ajratilgandanoq chaqirilishi kafolatlanadi. Bu metod an'anaviy konstruktorga o'xshaydi, ammo xotirani tanlashni amalga oshirilmaydi. Xotira new uslubi bilan ajratiladi.

class My class uslublariga e'tibor bering.
Name = "class Name" # konstanta klassa

@@count = o # o'zgaruvchi klassni inisializasiya qilish.
def initialize # ob'ekt uchun xotira ajratilgandan so'ng chaqiriladi.
@@count + = 1
@myvar = 10
end

def My Class. getcount # klass uslubi.
@@ count # klass o'zgaruvchisi.
end

def getcount # nusxa o'zgaruvchi klassni qaytaradi.
@@ count # klass o'zgaruvchisi.
end

```
def getmyvar # nusxa uslubi.
@myvar # nusxa o'zgaruvchisi.
end

def setmyvar (val) # nusxa uslubi @ myvar ni o'rgaradi.
@myvar = val
end

foo = My class.new # a) myvar teng 10.
foo. setmyvar 20 # a) myvar teng 20
foo. myvar = 30 # a) myvar teng 30
```

Bunda biz getmyvar @ myvar o'zgaruvchi ma'nosini qaytarib, setmyvar uni o'rnatadi. (Ko'pchilik dasturchilar o'rnatish va o'qitish uslublari haqida gapirishadi)

Hammasi ishlaydi, ammo Ruby da harakat vositasi bo'lib kelmaydi. Myvar uslubi ko'p yuklangan o'zlashtiruvchi operator kabidir, setmyvar omadli alternativa, lekin undanam yaxshiroq vositalar bordir.

Mobile sinf o'z ichiga *att, attr_accesor, attr_reaber va attr_writer*_uslublarini oladi. Ulardan foydalanish mumkin. (parametrlar sifatida belgilarni (simvol) uzatish mumkin). Berilgan nusxalarga (ekzemplyar) ruxsatni (dastur) boshqaruv avtomatizasiyasi uchun.

Misol, 3ta usul : getmyvar, setmyvar, myvarlarni bitta misra bilan almashtirish mumkin attr_accesor : myvar.

Shuningdek, @ myvar ma'nosini qaytarib beradigan myvar uslubi yaratiladi. attr_reder va attr_writer uslublari o'qish va o'zgartirish uchun bo'lgan atributlarni ruxsat (dastur) usullarini yaratadi.

Sinflarda aniqlangan nusxalar ichki uslublarda zarur vaqtda self o’zgaruvchisidan foydalanish mumkin.

Nusxa (ekzemplyar) usuli nomidan chaqirilgan ob’ektga yuborilgan oddiy chaqiruvdir (ссылка).

Private, protectecl va public modifikatorlaridan foydalanib, sinf usullari ko’rinishlaridan foydalanish mumkin.

O’zgaruvchi nusxalar doim yopiq, ruxsat (доступ) usullari yordamida sinflarga murojaat etish mumkin.

Har bir modifikator parametr belgisi (символ) sifatida qabul qiladi, masalan : foo , ammo agar u tushirilgan bo’lsa, modifikator harakati keyingi sinflarga tarqaydi (yechiladi). Masalan : class My Class

class MyClass

```
def method1
 # ...
end

def method2
 # ...
end

def method3
 # ...
end

private :method1
public
```

```
:method2
protected :method3

private
def my_method
 # ...
end

def another_method
 # ...
end

end
```

Bu sinfdagi method 1 usuli yopiq, method 2 ochiq, method 3 usuli esa himoyalangan. Keyinchalik private usuli chaqirilganda, my_method va another method usullari yopiq bo'ladi. Public ruxsat (доступ) darajasini tushuntirish shart emas, chunki u usul ko'rinishini va ruxsatini hech qanday chegaralanmaydi. Private darajasi usulni mutloq (guruhlarini) (подклассы) sinf ichida yoki uning sinf bolinmalarini bildiradi, self nomidan " funksional shaklda " chaqiradi, chaqirilayotgan ob'ekt aniq yoki noaniq ko'rsatilishi mumkin. Protected darajasi faqat sinf ichida chaqiriladi, lekin yopiq usuldan farq qilgan holda self nomidan chaqirilish shart emas.

По умолчанию aniqlangan hamma sinfdagi usullar ochiqdir.

Initialize gina faqat bundan hashdir (mutloq) yuqori darajasidagi dasturlar aniqlangan usullar ham по умолчанию ochiqdir.

Agarda ular yopiq deb e'lon qilinsa, faqatgina funksional shaklda chaqiriladi. (Object sinfdagi usullardek).

Ruby sinflari o’zlari ob’ekt bo’lib, class meta sinf nusxalari (экземплярлари) bo’lib keladi. Bu tildagi sinflar doim aniq (конкрет), abstrakt sinflar esa mavjud emas.

Biroq abstrakt sinflarni ham nazariy tadbiq (реализасия) etish mumkin, agarda sizga zarur bo’lib qolsa. Object sinf ierarxiyaning (tag-tubi) tomiri bo’lib, Kernel modulidagi o’rnatilgan hamma usullarini ko’rsatib beradi.

Agarda sinfni yaratmoqchi bo’lsangiz, qutidagi amallarni bajarish kerak :

```
class My Class other Class
# …..
end.
```

O’rgatilgan usullardan foydalanishdan tashqari o’zining usulini aniqlash mumkin. Agarda aniqlanilayotgan usul mavjud bo’lgan usulning nomiga ega bo’lsa, eski usul o’zgartiriladi. Yangi usul “avvalgi” usulga murojaat etsa (bu boshlang’ich holat tez – tez bo’lib turadi), u holda super asosiy (kalit) so’zdan foydalanish mumkin.

Operator misridan ko’proq yuklatilsa (перегрузка), ООП ajralmas qismi bo’lib kelmaydi. Ammo bu mexanizm C++ da ba’zi boshqa tillarda dasturchilarga belgi bo’lib keladi.

Rubyda ko’pchilik operatorlar usullar bo’lib keladi va foydalanuvchi sinflarda aniqlik kiritish mumkin.

Operatorning semantikasini mavjud bo’lgan sinfda qayta aniqlik kiritish to’g’ri kelmaydi (ma’noga ega emas), biroq yangi sinflarda operatorlarga aniqlik kirgizish oddiy holat.

Usullarning sinonimini yaratish mumkin. Buning uchun sinfning ichki aniqligi shunday sintaksis bilan ko’rsatish mumkin alias newname oldname.

Parametr soni o’zgarmaydi, eski nomiday, sinonim – usul ham shunday chaqiriladi. Vergul yo’qligiga ahamiyat bering ; alias – usul nomi emas, kalit so’zdir. alias_method usuli ham bor bo’lib, o’zini shunga o’xshash holatda

tutadi, ammo uni ishlatilganda (qo'llanilganda) parametrlar vergul bilan ajratilishi shart, boshqa usullar kabi.

1.3.5. Usullar va atributlar

Ko'rganimizdek bu usullar oddiy nusxali (ekzemplyarlar) sinflar va o'zgaruvchi bilan birgalikda foydalaniladi. Chaqirilayotgan ob'ekt usul nomidan nuqta bilan ajratiladi (receiver. method). Agar usul nomi tinish belgisi bo'lsa, nuqta qo'yilmaydi. Usullar argumenti bo'lishi mumkin :

```
Time.mktime(2000, "Aug", 24, 16, 0).
```

Har bir ifoda ma'noni qaytaradi,bunda usullar chaqiruvchi bir biri bilan ulanadi:

```
3.succ.to_s
/(x.z).*?(x.z).*?/.match("x1z_1a3_x2z_1b3_").to_a[1..3]
3+2.succ
```

Darvoqe, agar ifoda, birlashuv birikuv (сцепление) natijasi bo'lib, aniq usulga ega bo'lmasa, muammo paydo bo'ladi.

Aniqrog'i, aniq sharoitda (xolatda) ba'zi usullar nilga qaytadi, bu ob'ektdan chaqirilgan har bir usul xatoga olib keladi. (albatta, nil – to'liq ob'ekt, lekin massivday usullarga ega emas). Ba'zi usullarga bloklarni uzatish mumkin. Foydalanuvchi aniqlangan yoki o'rnatilgan hamma iteratorlar uchun. Blok odatda operator qavslarga do-end yoki figurali qavslarga yopiladi. Lekin ular ulardan oldingi parametrlardan ko'rilmaydi.

Chaqiruv uslub File. open misolida :

```
my_array.each do |x|
 some_action
end
File.open(filename) { |f| some_action }
```

Ruby versiyasida nomlangan parametrlar ko'rsatiladi, Python tilida ular kalit argumentlar (dalillar), Ada tilida aytiladi.

Usullar o’zgaruvchan argument sonni qabul qilishi mumkin :

receiver. method (arg 1, * more_args)

Berilgan xolatda chaqirilgan usul more_args deb nomlanib, turli massivlarday qo’llaniladi.

Darhaqiqat, formal parametrdagi ro’yxatdagi yulduzcha (oxirgi bo’lmagan yoki yagona parametr) massivga parametr ketma-ketligini “свернуть” yopishi mumkin.

```
def mymethod(a, b, *c)
 print a, b
 c.each do |x| print x end
end
mymethod(1,2,3,4,5,6,7)
# a=1, b=2, c=[3,4,5,6,7]
```

Sinf emas ob’ekt darajasida Rubyda usullarni aniqlash imkoniyatiga ega. Bu usullar sitletli deb ataladi ; Ular yagona ob’ektga mansub bo’lib, na sinflarga,na uning super – sinflariga ta’sir qiladi. Bunday imkoniyat foydalanuvchining grafik interfeys ishlanmasida foydalanish mumkin. Knopkani (tugmacha) harakatini aniqlash uchun, berilgan tugmachaga singlet usulini qo’llaysiz ;

Qator ob’ekt uchun singlet usuli aniqlash :

```
str = "Hello, world!"
str2 = "Goodbye!"
def str.spell
 self.split(/./).join("-")
end
str.spell # "H-e-l-l-o-,- -w-o-r-l-d-!"
str2.spell # hatoooooo!
```

Shunisi ahamiyatli.

O’zgaruvchi uchun emas, bu usul ob’ekt uchun qo’llaniladi. Nazariy singlet usuli yordami bilan o’xshash hollarda ob’ektlar tizimini (sistemasini) yaratish mumkin.

Sinfsiz OOP shakli. Undagi asosiy struktura mexanizmi avval qo’llanilgan ob’ektga qarab yangi ob’ektni konstruksiya qilish mumkin. Bunday sistema Rubyning imkoniyatlarini to’liq ochish mumkin.

4-mavzu: *Rubyning dinamik* aspekti

Reja:

4.1 Jarayon vaqtida kodlash

4.2 Aks (akslanish)

4.3 Mavjud bo'lmagan usullar.

Ruby – dinamik til, uni ob'ektlarda va sinflarda ishni bajarish vaqtida o'zgartirish mumkin. Ruby statistik yozilgan dasturni bajarilayotgan vaqtda konstruksiya yoki kod qismlarini internritasiya qilishga imkon beradi. Unda ayer API aks etilib, uning yordamida dastur o'zi haqida ma'lumot oladi.

Bu esa отладчик, профилировицик va boshqa asboblarni yaratishda va тривиал bo'lmagan kodlanish vositalarini qo'llaydi.

Ruby dasturini o'rganmoqchi bo'lgan dasturchilarga eng og'ir mavzudir.

1.4.1. Jarayon vaqtida kodlash

Load va reguile директивилар usullarini parametrday yoki o'zgaruvchiday chaqirish mumkin, shartli ravishda ham. Компилясия vaqtida ishlovida include direktivasi bilan C va C ++ tillarida taqqoslang.

Kodni ko'rish va qismlarni interpritasiya qilish mumkin. Calculate usuli va uni chaqiradigan kod :

```
Def calculate (op 1, operator, op 2)
string = op 1.to _S + operator + op 2. to_S
# operator – jumla ; uzun jumla operatordan va operandlardan iborat
eval (string)      # hisoblab chiqamiz va qaytaramiz ma'nosini
end
@   alpha = 25
@   beta = 12

puts calculate (2,"+",2) # 4 pechatlaniyapti (chop etiliyapti)
```

puts calculate (5,”*”,”@ alpha”) # 125 chop etiliyapti.

puts calculate (“@ beta,””**”, 3) # 1728 chop etiliyapti.

Dastur foydalanuvchidan usul va kodni bir jumlasini so’raydi, so’ng bu usul aniqlanadi va chaqiriladi :

```
puts “usul nomi !”
meth_name = gets
puts “Kod jumlasi !”
code = gets.

String = % [def # {meth_name} \ n # {code}\n end] # Jumla ko’ramiz eval
eval (string)              # Usulini aniqlaymiz.
eval (meth_ name)          # Usulni chaqiramiz .
```

Turli sharoitda va turli platformalarda ishlaydigan dasturni yozish zarur, shu bilan birga matnlarni saqlab qolish kerak.

C tilida # ifdef direktivasi qo’llaniladi, Rubyda “Kapilyasiya pog’onasi” tushunchasi yo’q, hamma konstruksiyalar dinamik, statistik emas. Shuning uchun bajarish jarayoni vaqtida biz shartlarni hisoblab chiqishimiz mumkin :

```
if platform – Windows
      action 1
      elsif platform = hinux
      action 2
      alse
      default_action
      end
```

Albatta, bunday kodlashda ishlab chiqarish pasayish mumkin, chunki shart bir necha marotaba hisoblab chiqiladi.

Bir misolni ko’rib chiqamiz, platformadan nomi bog’liq emas, hamma platformali qarashli (tobe, qarashli) kod bir usulga joylashan.

```
ef platform == Windows
def my_action
action 1
end
elsif platform = =linux
def my_action
action 2
end
else
def my_action
defaaul_action
end
end
```

Bunday vosita bilan biz kutilgan natijaga erishamiz, lekin shart bir martagina hisoblanadi. Dastur my_action usulini chaqirganida, u to’g’ri aniqlangan bo’ladi.

1.4.2. Aks (akslanish)

Refleksiv dasturda g’oya Smalltalk, LISP va Java tillarida (turli darajada) tadbiq etilishi mumkin – faol muhim ob’ekt strukturasini so’roqlab va kengaytirib yoki bajarilish vaqtida модификасия qilishi mumkin.

Ruby tilida akslanishni quvvatlovchi bo’lib, smalltalk dagi konstruksiyani boshqaradigan ham ob’ekt bo’lib keladi, Ruby tilida bunday emas, akslanishni qo’llab keladi (поддержка). Rubyda konstruksiyani

boshqaradigan va bloklar ob’ektlar bo’lib kelmaydi. (Proc ob’ektini blokday ob’ekt ko’rinishida foydalanish mumkin, ammo konstruksiya boshqarmasi ob’ekt bo’lib kelmaydi hech qachon).

Berilgan nom bilan identifikator foydalanilishini aniqlash uchun defined ? kalit so’zi ishlatiladi (so’zning oxirida so’roq belgisiga ahamiyat bering).

```
if defined ? some var
puts “some_var = # {some_var}”
else
puts “o’zgaruvchan some_var noma’lum”
end.
```

Ko’rsatilgan uslub chaqirig’iga ob’ekt javob bera olishini o’xshash repond_to ? usuli aniqlaydi.(ya’ni berilgan uslub berilgan ob’ektga to’g’irlash) Object sinfda respond_to ? usuli aniqlangan.

Rubyda ma’lumot so’rovi bajarilayogan vaqtda to’liq (поддержка) qo’llab-quvatlanadi bo’ladi. Ob’ektning turi yoki sinfi type usuli bilan aniqlash mumkin.

is_a ? usuli qaysi sinfga ob’ekt mansubligini ma’lum qiladi (super – sinflarni o’z ichiga olgan holda) kind_ob ? nomi sinonimi bo’lib keladi. Masalan : puts “abc : class”# String deb pechatlanadi. puts 345. class # Fixnum deb pechatlanadi.

```
rover = Dog . new
print rover . class # Dog  deb pechatlanadi.
if rover . is_a ? Dog
puts “Albatta, bo’lib keladi”.
end
if rover . kind_of Dog
puts “Ha, xaliyam quruq.” end.
if rover. is_a ? Animal
```

puts “Ha, u yana xayvon ham”.

end

Berilgan ob’ektga chaqiriladigan hamma usullarning to’liq ro’yxatini olish mumkin. Object sinfidan methods usuli to’g’ri keladi. Uning variantlari ham bor private_instance_methods, public_instance_methods va h.z.

Shunga o’xshashlarni berilgan ob’ektdan o’zgaruvchan sinflarni va nusxalarni (ekzemplyarlarni) bilib olish mumkin. O’z tabiatida ООП usullar va o’zgaruvchilar ro’yxatiga sinfdagi va super – sinfdagi ob’ektlar aniqlanadi.

Module sinfida constants usuli bo’lib, modulda aniqlangan hamma konstant ro’yxatini olish mumkin. Module sinfida ancestors usuli bo’lib, u berilgan modulda modul ro’yxatini qaytarib beradi.

Bu ro’yxatga modulni o’zi ham kiradi, ya’ni Mod elementi bor Mod . ancestors chaqiruvi bilan ro’yxat qaytariladi. Bu ro’yxatga boshlang’ich “ota - onalar” moduli kiradi.

Object sinfida superclass usuli bo’lib, u nil yoki ob’ektning super – sinfni qaytarib beradi. Object sinfigina super – sinfga ega emas, nil uning uchungina qaytarib beriladi.

Object Spase moduli har xil “tirik” ob’ektlarga dastur (ruxsatga) ega.

_idtoref moduli ob’ekt identifikatoriga ссылка (chaqiruv) yuboradi ; bu operasiya nomning boshida 2 nuqta qo’yiladigan operasiyaning teskarisi.

Object Spase modulida each_object iteratori mavjud bo’lib, shu vaqtdagi hamma bor ob’ektlarni va tanib bo’lmaydigan ob’ektlarni va tanib bo’lmaydigan ob’ektlarni ham ko’zdan o’tkazadi (qarab chiqadi).(Eslatma, Fixnum, NilClass, ThueCkasse va False Class sinflarga mansub bo’lgan o’zgarmas ob’ektlar optimizasiya tufayli saqlanmaydi)

1.4.3. Mavjud bo’lmagan usullar.

Ruby (myobject, mymethod) usulini chaqirganda quyidagi tartibga nomlangan usullarni qidiradi :

1. myobject ob’ekti uchun singlet usuli.
2. myobject ob’ekti aniqlangan sinflar usullari.
3. myobject ob’ekti aniqlangan avvalgi sinflar usullari.

Agar mymethod usulini topish imkoni bo’lmasa, Ruby method_missing nomli usulni qidiradi.

Agar aniqlansa, unga yo’q bo’lgan (ishtirok etmagan) usul nomi (belgi tarzda) va hamma uzatilgan parametrlar beriladi. Bu mexanizm noma’lum ma’lumotni dinamik qayta ishlashda ish vaqtida qo’llaniladi.

Ruby yaratayotgan dinamik muhitda past darajada xotirani boshqarib bo’lmaydi, bu xatolarga yo’l qo’yadi. Chiqindi yig’ish mexanizmi – katta imkoniyat. C + + tilida ajratilgan va xotirani tozalash uchun dasturchi javob beradi. Kechki tillarda, misol Javaday, xotira chiqindi yig’uvchisi bilan tozalanadi (ob’ekt ko’rinish darajasida bo’lmaganda). Xotira bilan boshqarishda 2ta xatoga yo’l qo’yish mumkin.

Havola bor ob’ektni xotirasini tozalayotganda – bunda keying dasturda (ruxsatda) ob’ekt qarama – qarshi (yoki teskari) holatda bo’lib qolishi mumkin. “Osilib turadigan ko’rsatkichlar” xato qilingan joydan uzoqda bo’ladi. Hech kim ссылка yubormayotgan ob’ektni tozalashda xotiraning утечкаси (istamagan holda ketib - qolish) .

Bu holda dastur ko’p xotirani “eydi” ishlatadi va avariya tugaydi : bunday xatolarni izlash nihoyatda qiyin.

Ruby tilida ishlatilmagan (foydalanilmagan) ob’ekt va u bilan band bo’lgan xotirada chiqindi yig’ish mexanizmi qo’llaniladi. Ruby yo’q qilish va belgilash algoritmidan foydalaniladi, ссылкаlarni hisoblash bilan emas.

Chiqindi yig’ish o’zi bilan ishlab chiqarishni pasayishiga olib keladi.

C C moduli cheklangan boshqaruv vositalarini ko'rsatib, dasturchi ishini aniq dasturga yo'naltiradi. Ob'ektni tozalovchisi (finalizer) ham aniqlash mumkin.

1.5. O'z instruksiyangiz ustida mashq qiling : nimalarni eslab qolish kerak.

"**Hamma narsani tushunganingdan so'ng, hamma narsa ravshan bo'ladi".**

Dasturchilar turli tajribalarga ega : C va Smalltalk foydalanuvchilari Rubyni har xil qabul qilishlari mumkin. Ruby sintaksis analizatori murakkab, lekin ko'p narsani "kechiradi" Ruby dasturchini nima demoqchi bo'lganini tushunishga xarakat qiladi, o'zining qoidalariga bo'ysuntirmay. Ruby sintaksisini bilish uchun quyidagi ro'yxatni bilish kerak :

Usulni chaqirishda qavslarni tushurib qoldirishi mumkin.

```
foobar
foobar ( )
foobar (a,b,c)
foobar  a,b,c
```

Qavs zarur bo'lmaganda x y z ? yozuvi nimani bildiradi. "Y – usulni, chaqirishi, Z – parametrni uzatish, x – natijani esa x usuliga parametr ko'rinishda uzatish".

x (y(x)) Bu holat keyinchalik o'zgaradi.

X E SH usulini uzatamiz : my_method {a = >1, b = >2}

Bu sintaksis xatoga olib keladi, chunki chap figurali qavs blokning boshi deb qabul qilinadi. Bu xolda qavslar zarur :

my_method ({a = >1, b = >2})

Taxmin qilaylik, xesh – yakka (yoki oxirgi) uslub parametri.

Ruby figurali qavslarni tushirib qoldirishga ruxsat beradi:

my_method (a = >1, b = >2)

Kimdir bu yerda nomlangan parametrli usul (вузов) chaqirishni ko'radi. Bu yolg'on ta'surot lekin bunday konstruksiyani hech kim man etmagan.

Boshqa holatlarda ham qavslarni tushurib qoldirish ham boshqa ma'noga ega.

Masalan, to'rtta ifoda ham bir qarashda bir xilday :

x = y + z ; x = y + z ; x = y + z ; x = y + z :

Birinchi uchtalik ekvivalentli, to'rtinchisida analizator *y* usuli *+z* parametr bilan chaqirilgan deb hisoblaydi va xato haqida ma'lumot beradi, chunki bunday nomli usul yo'q. Muammolar bilan dono foydalaning.

- O'xshash *x = y*z* – bu ko'paytiruv *y* ni *z* ga, shunda *x = y*z* - *y* chaqiruv usuli *z* massiv kengligi parameter sifatida uzatiladi.
- Identifikator nomlarida " - " belgisi zarur xarf hisoblanadi.

Bu holda, identifikator nomi shu belgidan boshlanishi mumkin, lekin bunday identifikator konstanta bo'lib kelmaydi, agar keyingi harfi bosh harf bo'lsa ham.

- Ketma – ketlikda if da kalit so'z elsif qo'llaniladi, boshqa tildagi else if yoki elif so'zlari emas.
- Rubyda kalit so'zlar usul nomi bilan to'g'ri kelib qolishi mumkin, bu holda extiyot bo'lish kerak, chunki bu dasturni odamlar o'qiydi, buni unutish kerak emas.
- Kalit so'z then (gapda if va case) zarur emas. Agar dastur bu so'z bilan tushinarli bo'lsa kodga kiriting.

Bu narsa do so'zini while va until ga ham ta'luqli.

So'roq va undov belgilari identifikator qismi bo'lib kelmaydi, modifikasiyalashadi, ularni suffiks (so'z yasovchi qo'shimcha) deb qaraladi. Shunday qilib, chop va chop ! identifikatorlari turli deb hisoblansa, ! belgisi nomlanishda ruxsat berilmaydi boshqa joyda. Bunday o'xshash holatlar Rubyda defined ? konstruksiyasida, lekin kalit so'zi – defined bo'lib keladi.

-Jumla ichidagi # belgisi – ifodaning boshlanishi. Demak, ba’zi xolatlarda {,$ yoki @ belgilari rishotka # dan keyin kelsa.

-Identifikator oxirida so’roq belgisini qo’shish mumkin bo’lgan hollarda ternarli operatorda пробел quyish mumkin.

- Misol uchun o’zgaruvchan my_flag, true yoki false ma’nosini qabul qilishi mumkin. Bu xolda birinchisi to’g’ri, ikkinchisi sintaksis xatoga ega : *x = my_flag ? 23 : 45* # to’g’ri

x = my_flag ? 23 : 45 # sintaksis xato

Marker oxirini kiritilgan xujjat uchun leksema deb xisoblanmaydi. U misrani hammasini belgilaydi, shuning uchun misradagi belgilar dastur matn qismi bo’lib kelmaydi, kiritilgan xujjatga qarashlidir.

Rubyda ozod bloklar yo’q, C day hohlagan joyda blokni boshlash mumkin emas. Bloklarga ruxsat faqat kerakli joylardagina ruxsat etilgan, masalan, iteratorga qo’shilishi mumkin. Blok begin – end bundan mustasno, bu blok hamma joyda qo’llaniladi.

Eslatma BEGIN va END kalit so’zlari begin va end so’zlari bilan hech qanday o’xshashlik tomoni yo’q statistik konkatepasiya misrasida konkatenasiya pasroq chaqiruv usulidan.

Misol :

str = “Birinchi” ‘second ! center (20) # Misol 1 and 2

str = “Ikkinchi” + ‘second ! center (20) # bir xil

str = “Birinchi”+” ikkinchi”. Center (20) # Misol 3 and 4

str = ("birinchi " + ikkinchi).center(20) # bir hil

Rubyda bir necha yolg’on o’zgaruvchilar (псевдопеременный) lokal o’zgaruvchiga o’xshash bo’lib, ayrim maqsadlar uchun qo’llaniladi. Bular self, nil, true, false,

_FILE_u_LINE_.

1.5.2. Dasturlashning kelajagi.

Bugungi kunda Rubyni o’rganayotgan inson avval boshqa tillarni o’rgangandir yoki foydalangandir. Bu yaxshi, chunki Rubyni o’rganish ancha yengil bo’ladi, chunki boshqa tillarda ham vositalar bir - biriga o’xshashdir, lekin Rubyni tanish konstruksiyalarini tanish ko’ringan dasturchilar extiyot bo’lishlari shart, chunki noto’g’ri xulosaga kelish mumkin, o’tmishdagi tajribani, ya’ni “ekspert yukini” olgan holda bu holatni ko’rib chiqamiz. Ko’pchilik mutaxasislar Rubyga Smalltalk, Perl, C/C + + va boshqa tillardan utishadi. Adashmaslik uchun misollarni ko’rib chiqamiz :

- Rubyda belgi butun son hisoblanadi. Bu Pascalday mustaqil tur va ekvivalent misra uzunligi 1 emas.

```
x = "Hello"
y = ?A
puts "x[0] = #{x[0]}" # chop etish  x[0] = 72
puts "y = #{y}"       # chop etish  y = 65
if y == "A"           # chop etish  no
 puts "yes"
else
 puts "no"
end
```

- bool turi yo’q. TrueClass va FalseClass – turli sinflardir, true va false ob’ektlari ularning yagona nusxasi (ekzemplyari)
- Rubydagi operatorlar C tilidagi operatorlarni eslatishadi.

Ikkita mustasno qoida – increment va decrement (+ + va - -) Operatorlari. Ular Rubydagi yo’q “пост” va “перед” shaklida ham.

- Ma’lumki, (салыш) musbat sonlar uchun bo’lish modul buyicha operator turli tillarda turlicha ishlaydi.

```
puts (5 % 3) # pechatlanadi 2
puts (-5 % 3) # pechatlanadi 1
puts (5 % -3) # pechatlanadi -1
puts (-5 % -3) # pechatlanadi -2
```

- Ko'plar "yolg'on" so'zini "O", bo'sh misra deb tasavvur etishadi. Rubyda esa bu "haqiqat"ga teng. Haqiqat hamma narsa, false va nil ob'ektlardan tashqari.
- Rubyda o'zgaruvchilar hech qanday sinfga mansub emas.
- Rubyda o'zgaruvchilar e'lon qilinmaydi, biroq nil boshlang'ich ma'nosini o'zlariga o'zgaruvchilar o'zlashtiradi.

Analizator biladi, berilgan nom usulga emas, o'zgaruvchiga xosdir.

- ARGV[0] – buyruq misrasidagi birinchi argument . Ular "0"- noldan boshlab raqamlanadi. Bu C tilidagi argv[0] yoki fayl nomi emas.
- Ko'pchilik operatorlar Rubyda usullar bo'lib kelmaydi ; "tinish ishoralari" belgilari qulaylik uchun foydalaniladi.

Birinchi mustasno qoida – o'zlashtirish operator nabori (yig'indisi) (+ =, - = va h.k.). Ikkinchi mustasno qoida – operatorlar =,..,…,!, net, &&, and, ||, or, ! = ,! .

- Zamonaviy dastur tilida bulev operatsiyasi tugaydi.

Operatsiya ketma – ketligida or musbat tugaydi, true birinchi ma'no olinganda, and ketma – ketlik operasiyasida – false birinchi ma'no olinadi.

- @@ prefix o'zgaruvchi sinflar uchun qo'llaniladi.
- loop – kalit so'z emas. Bu kernel modulining uslubi,

konstruksiyani boshqaruvchisi emas.

- Unless – else sintaksisi kalitdagi shubhali ko'rinar, chunki

unless – ifga qarama – qarshi, lekin else tarmog'i shart haqiqiy (to'g'ri) bo'lsa bajariladi.

- Fixnum oddiy tur uslub ichida o'zgartirilmaydi, true, false va

nil va ham ta'luqli.

- && va || operatorini & va I operatori bilan adashtirmang .

Unisi ham bunisi ham C tilida foydalaniladi ; birinchi ikkitalik mantiq (mantiqiy) operatsiyalar uchun, oxirgi ikkitasi – razryadlar uchun – and va or operatorlari && va || ga nisbatan past :

a = true

b = false

c = true

d = true

al = a && b or c && d # [&& operasiyalari birinchi bajariladi

a2 = a && (b or c) && d # [or operasiyasi birinchi bajariladi.

puts al # false pechatlanadi.

puts a 2 # true pechatlanadi

- Yoddan chiqmasin, o'zlashtirish "operatori" and va or

operatorlardan ancha ustun (bu tarkibiy o'zlashtirib oluvchi operatorlarga : + =,- = va h.k.ga mansub).

Masalan, kod *x* = *y* or *z* oddiy o'zlashtiruvchi gap (taklif), lekin (ekvivalent (*x*=*y*) or *z*) ajratib ko'rsatilgan ifoda.

Dasturchi *x* = *(y* or *z)* deb olgan.

y = false

z = true

x = *y* or *z* # operator = or dan ertaroq bajariladi.

puts *x* # pechatlanadi false

(*x* = *y*) or *z* # 5 jumla : yuqorigiday

x = (y or z) # or operator boshidan hisoblaydi.

puts *x* # pechatlanadi true.

Ob'ekt atributlarni o'zgaruvchi lokal bilan adashtirmang. Agar siz C+ + yoki javaga o'rganib qolgan bo'lsangiz,bu haqda esingizdan chiqaring.

a) my-var o'zgaruvchi sinf matnida bu ekzimplyar

o'zgaruvchisi (yoki atribut) ammo my_var shu matnda – lokal o'zgaruvchi.

- Ko'pchilik tillarda, Ruby da ham "bor" sikli bor. Indeks

o'zgaruvchisini modifikatsiyalantirish mumkinmi degan savol tug'iladi. Ba'zi tillarda bu boshqaruv o'zgaruvchisini qat'iy o'zgartirish man etiladi (ish bajarilishi vaqtida yoki kampilatsiya pog'onasida (davrida) xato bo'lgani haqida ogohlantiruvchi ma'lumot kiritiladi). Boshqalarda bu holatga ruxsat beriladi, siklni o'zgarishiga qaramay. Rubyda uchinchi yondashuv bor. for o'zgaruvchan,boshqaruv sikli, oddiy o'zgaruvchi deb hisoblanadi,istagan vaqtda o'zgartirish mumkin, bu o'zgarish sikliga ta'sir qilmaydi. for sikli bu o'zgaruvchilarni o'ziga qabul qiladi.

Masalan: keying sikl 10 marta bajariladi va 1dan 10gacha sonlarni pechatlaydi.

```
for var in 1…10
puts " var=# {var} "
if var>5
var=var+2
 end
 end
```

- O'zgaruvchilar nomlarini usul nomlaridan ajratish oson

emas. Buni analizator bajaradi,agar idintifikatorga uni ishlatish oldidan ma'noni o'zlashtirishini analizator ko'rsa bu holda u o'zgaruvchi hisoblanadi.

Ruby zid bo’lmagan til qilib loyihalashtirildi. Lekin u shu bilan birga murakkab til, o’zining ideomalari bilan.

- alias kalit so’zi yordamida o’zgaruvchi va usullarga alternativ nomlar (sinonimlar) berish mumkin.
- $ 1, $ 2, $ 3 va boshqa nomerlangan global o’zgaruvchilar

sinonimga ega bo’lolmaydi.

$ =,$_,$| ga o’xshash “maxsus o’zgaruvchilarni” ishlatishga maslahat bermaymiz. Ba’zida ular kompakt kod yozishga ruxsat beradi, ammo tushunarsiz bo’lib keladi.

- Diapazon operatorini adashtirmang, birinchisi yuqori

chegarani yoqadi, ikkinchisi - yo’q. 5…10 diapazon sonni yoqadi, 5…10 – diapazoni yo’q.

- Agar m…n diapazon berilsa, end usuli n oxirgi nuqtasini

qaytaradi, uning last sinonimiday. Bunday usullar n ma’noni m…n diapazoniga qaytaradi.

Bu ikki vaziyatni farqlash uchun end_excluded ? usuli beriladi.

- Diapazonlarni massivlar bilan adashtirmang, ular har xildir :

x = 1…5

x = {1,2,3,4,5}

Biroq qulay to a usuli massivda diapazoni o’zgartirish uchun bor.

- nil ma’nosi “o’zlashtirilmagan” o’zgaruvchisida bo’lib, bu

masala shunday echiladi : $x = x||5$ yoki qisqartirilib X||=5.
false ma’nosi va nil ma’nosi qayta yoziladi.

- Ko’pchilik tillarda ikki o’zgaruvchining ma’nosini (значение)

o’zgartirish uchun qo’shimcha vaqtinchali o’zgaruvchi kerak. Rubyda ko’p o’zlashtirish mexanizmi unga ortiqchalik qiladi : *x, y = y,x* ifoda x va *y* ma’noni almashtiradi.

- Nusxani (ekzemplyarni) sinfdan aniq operating. Masalan

@@ foobar o’zgaruvchi butun sinf ko’rinadi, @ foobar o’zgaruvchi nusxada har bir sinf ob’ektiga qayta yaratiladi.

- Sinf usuli aniqlangan sinf bilan o’xshash, u hech qanday

aniq ob’ektga qarashli emas va ob’ekt nomidan chaqirilmaydi. Sinf usulini chaqirilganda sinf nomi ko’rsatiladi, (ekzemplyar) nusxa usulida esa – ob’ekt nomi ko’rsatiladi.

- Ruby haqidagi chop etilgan materiallarda nusxa usulini

belgilash uchun reshotka # potasiyasi qo’llaniladi. Masalan, biz File. chmod deb yozamiz, File sinfidagi chmod usulini belgilah uchun, va File # chmod shu nomli nusxa usulini belgilash uchun. Bu potasiya Ruby sintaksisini qismi bo’lib kelmaydi.

- Ruby konstantlar o’zgartirilmaydi. Ularni nusxa usullarda

o’zgartirib bo’lmaydi, ammo boshqa joylarda mumkin.

- ChU tilidan yield kalit so’zi kelgan va ba’zi dasturchilar

uchun bu tushunarli emas.

Iterator chaqirilganda, shu iterator ichida blokka boshqarishni uzatishda foydalaniladi.

- O’zlashtirishning tarkibiy operatorlari + =, - = va boshqalar

-bu usullar emas. Bu uzun yozuvning qisqartirilgan shakli. Shuning uchun *x + = y* o’xshaydi *x = x + y* ga. Agar + operator ko’p yuklatilgan bo’lsa, + = operatori “avtomatik” ravishda ya’ni semantikani qo’llaydi.

- Inisializasiya o’zgaruvchilari uchun o’zlashtirish tarkibiy operatordan foydalanish mumkin emas. Agar birinchi murojaat o’zgaruvchi *x* *x* + = *1* day ko’rinsa, xato kelib chiqadi.
- Bu holat bo’lmasligi uchun, nil ob’ekti uchun operatorlarni aniqlash mumkin. Boshlang’ich ma’no nil o’zgaruvchisiga teng bo’lsa, kerakli natija olamiz. nil.+, usuli Fixmum yoki string ob’ektlarni inisializasiya qilishda other аргумент ni (dalilni) orqaga qaytarish kerak. Shunday qilib, nil + other teng otherga.

```
def nil.+ (other)
other
end
```

- Class bu ob’ekt, Object – bu sinf.
- Ba’zi operatorlarni ko’p yuklab bo’lmaydi, chunki ular tilni

o’ziga joylashtirilgan (o’rnatilgan) =,….,…., and, or, not, &&, ||, !,= va ! . Tarkibiy o’zlashtirish operatorlarni ham (+=,-= va h.z.) ko’p yuklatish mumkin emas.

- O’zlashtirish operatorini ko’p yuklatish mumkin bo’lmagan

holda ham foo = (*x*.foo=5) nomli nusxa usulini yozish mumkin. = tenglik ishorasini suffiksday ko’rishingiz mumkin.

- fail – raise sinonimi.
- for (for *x* in *a*) konstruksiyasi each iteratorni chaqiradi.
- Eslatma, Kernel moduliga yuqori darajada aniqlangan usul

qo’shiladi va Object sinfi a’zosi bo’ladi.

- O’rnatish usullari (masalan foo=) ob’ekt nomidan chaqirilishi shart, bo’lmasa analizator shu nomli o’zlashtirish o’zgaruvchisi haqida deb noto’g’ri tushunchaga ega bo’ladi.

- Eslatma, retry kalit so’zi iteratorlarda foydalanish mumkin. Matlarda u hamma parametrlarni qayta inisializasiya qilib, iterasiyani qayta qayta boshidan tiklaydi.
- retry kalit so’zi mustasnolarni ham ishlov berishda qo’llaniladi.

1.6. Ruby jargonlari

Rubyni bilish uchun ingliz tilini qayta yodlash zarur emas. Lekin ba’zi jargonlarni bilish shart. Ba’zilarni kompyuter dunyosida ishlatiladigan so’zlardan boshqa ma’noda keladi.

- Rubyda “atribut” termini norasmiy xarakterga ega.

Atributni o’zgaruvchi nusxa deb olib, attr usullari yordamida ochiladi.

foo va foo= usullari @ foo o’zgaruvchisiga mos kelmasligi mumkin.

Ruby atributlarini (reader) o’qish usuli va (writer) o’rnatish usuliga bo’lish mumkin. Aksessor – ruxsat (доступ) usuli (accessor) bo’lib bir vaqtning o’zida ham o’qish va o’rnatish usullari bo’lib keladi. attr_accessor usuli nomi bilan keladi.

- Faqat Rubyda = = = operatori bor. (case equality operator)

“tarmoqli tenglik” operatori, (relation ship operator) “munosabat operatori”, (threequal operator) “uchlik tenglik operatori”, (spaceship operator) “kodlik operator” yoki “o’xshatish operatori”. <=> (uchar tarelkani eslatadi) – termin “te’riy rejimi” (poetry mode)da leksima va tinish belgilarni tushurib qoldirish mumkin.

some_method (1,2,3) # ortiqcha qavs

some_method 1,2,3 # “yahshi rejim”

Ko’pchilik xolatlarda jumla oxirida qavs, nuqta vergul bilan qo’ymaslik, then kalit so’zini if va case gaplarda tushirib qoldirish ruxsat beriladi.

def my_method (a,b,c) # bunday ham mumkin: def my_method a,b,c

#

```
end
```

- Eslatma Ruby grammatikasining murakkabligi analizatorni ishlatmay qo’yadi (to’xtatib qo’yadi).

```
def alpha (x)
x * 2
end
def beta (y)
y * 3
end
gamma = 5
delta = alpha beta gamma # 30 - - alpha (beta(gamma)) day
# ogohlantirish beriladi
# warning : parenthesize argument(s)
For future version
# ogohlantirish : argumentlarni qavs ichiga qo’ying
# keying versiyalar bilan
```

- Termin duck typing (“o’rdak tipizasiyasi”) Deyv Tomas (Dave Thomas)ga mansub. “Agar kimdir o’rdakka o’xshasa, o’rdakday yursa, ovoz chiqarsa – unda u o’rdak” degan ekan.

Rubyda biz is_a ? yoki kind_of usulidan foydalanamiz, ko’pincha respond_to ? usulini qo’llaymiz.

1.7. Xulosa.

Shu bilan ob’ekt – ob’ektga yunaltirilgan dasturlash va Ruby tili haqida gapimiz tugadi.

Endi qolgan bo’limlar boshlang’ich va o’rta darajali va tajribali dasturchilar Rubyda o’zi uchun yangi foydali narsalarni tanishadi deb o’ylaymiz.

Nazorat savollari:

1 Jarayon vaqtida kodlash nima?

2 Aks (akslanish) nima maqsadda ishlatiladi?

3 Mavjud bo’lmagan usullar?

5-mavzu. Qatorlarni qayta ishlash va dastur tuzish.

Reja:

1. Oddiy qator haqida tushuncha.
2. Qatorlarni Uzunligini olish.
3. Stringga ishlov berish. (Qator obrabotkasi)

> Qachonlardir dunyoni yaratilishida elementar ,,g'isht" -atomlar, keyinchalik protonlar, kvarklar hisoblangan. Endilikda qatorlar deb hisoblanadi.
>
> Devid Gross, nazariy fizika professori, Prinston universiteti.

1980 – yillari boshida bir informatika fani professori uzining birinchi ma'ruzasida ,,ma'lumotlar strukturasi"da uzi tanishtirmadi ham, kursni nomini ham e'lon qilmay, kursning dasturi, darsliklarni aytmay, birdaniga talabalar o'rtasiga ,,qanday ma'lumotlar turi eng asosiy?" degan savolni o'rtaga tashladi. Bir necha javob variantlari aytildi. ,,Ko'rsatgichlar" degan so'zni eshitganda, professor ,,To'g'ri, lekin eng asosiysi belgidir" dedi.

Kompyuterlar insonga xizmat qilishi uchun yartilgan, xujayin bo'lishi uchun emas. Chunki faqat insonga ma'lumotlar belgisi tushunarli. Demak, belgilar, ya'ni qatorlar inson kompyuter bilan muloqot qilishiga imkon beradi.

Boshqa tillarga o'xshab Rubydagi qatorlar – oddiy belgilar ketma-ketligidir. Turli ma'lumotlar, shu jumladan matnlarni biz qator ko'rinishida kodirovka qilamiz. Jumlalar to'liq ob'ektlar hisoblanadi. Dasturlarda jumlalar ustidan turli operatsiyalar bajariladi: konkaterlash, leksemani ajratish, analiz qilmoq, qidirish va o'rnini almashtirish va h.z. Ruby tili bu operatsiyalarni oson hal etadi.

2.1. Oddiy qator haqida tushuncha.

Ruby da qator – 5 bitli bayt ketma-ketligi. C dagiday nol belgisi bilan turmaydi. OxFF kodli belgilar jumlada bo'lishi mumkin, lekin (kodirovka) belgilar yig'indisi aniqlangan (yoki tanlangan) bo'lsa, jumlalar ma'noga ega bo'ladi.

Rubydagi oddiy jumla yakka qavsga olinadi. (\') – ekranli yakka qavs va (\\) ekranli yotiq chiziqli belgilargina belgilar boshqaruvi bilan aniqlanadi.

S 1 = "bu qator" # Bu qator

S 2 = "Xonim 0\' Liri' # Xonim o'Liri.

S 3 = 'Qara C : \\ TEMP' # Qara C : |TEMP

2 qavsli qatorlar katta egiluvchanlikka ega. Ularda boshqarish ketma – ketliklar : Ishni to'xtab qolish belgilari, tabulyasiyalashtirish, karetkani qaytarib berish, qatorlarni tarjima qilishdek.

Sakkizta raqamli belgilar :

S 1 = "Bu tabulyasiya belgisi : (\t)"

S 2 = "Ishning to'xtab qolishining bir necha belgisi : xyz|b|b|b"

S 3 = "Bu ham tabulyasiya belgisi:\011"

S 4 = "Bu ovoz berish signalining belgisi: |a|007"

2.4. Qatorlarni Uzunligini olish.

length usuli tizimlarni uzunligini olish uchun xizmat qiladi. Uning size sinonimi bor.

```
str1 = "eshmat"
x = str1.length # 6
str2 = "ali"
x = str2.size # 3
```

2.5. Stringga ishlov berish. (Qator obrabotkasi)

Rubydagi qator yangi qator belgilariga ega bo'lishi mumkin. Masalan, faylni xotirada o'qib bir qator ko'rinishida saqlash mumkin.

Iterator each ayrim qatorlarni tanlaydi :

```
str = “Qachonlardir \n qadim – qadimlarda.…\ n tugadi\n”
num = 0
str. each do |line|
num + = 1
print “Qator # {num}: # {line}”
end
```

Bu kodni bajarishda qutidagi natijalarga erishish mumkin :

Qator 1 : qachonlardir

Qator 2 : qadim – qadimda

Qator 3 : tugadi (tamom)

each_writh_index usuli bilan alternativ foydalanish mumkin.

2.6. Baytli ishlov berish.

Iterator each_byte bilan belgi (ramz)ni qayta ishlash ketma – ketligi uchun foydalanishning.

```
str = “ABC”
str.each_byte {|char|print char,””}
# Natija : 65 66 67
```

Ruby versiyasida scan usuli yordamida bir ramzli (belgili) qatorlar massivida qatorni yaratish mumkin.

```
str = “ABC”
chars = str.scan (|.|)
chars. each {|char|print char,””}
# Natija : ABC
```

2.7. Qatorlarni Maxsus taqqoslash.

Ruby tilida qatorlarni taqqoslash mexanizmi kiritilmagan : qatorlar leksikogradik tartibda taqqoslanadi.

Uzining qoidalarini kiritish ham mumkin. Masalan : a, an, the ingliz tilidagi artikllarni tushurib qoldirish qator boshida, yoki tinish belgilarga etibor

bermaslik. Buning uchun <=> kiritilgan usulni (<,<=,> va >=) qayta aniqlash kerak usullardan chaqiriladi.

Qatorlarni maxsus taqqoslash.

```
class String
 alias old_compare <=>
 def <=>(other)
  a = self.dup
  b = other.dup
  # ohirgi simvollarni uchradi
  a.gsub!(/[\,\.\?\!\:\;]/, "")
  b.gsubshidagi artiklni uchirish!(/[\,\.\?\!\:\;]/, "")
  # qator b.
  a.gsub!(/^(a |an | the )/i, "")
  b.gsub!(/^(a |an | the )/i, "")
  # probellarni yuq qilish
  a.strip!
  b.strip!
  # asosiy metodga muroojaat <=>.
  # a.old_compare(b)
 end
end
title1 = "Calling All Cars"
title2 = "The Call of the Wild"
# chp qil "yes".
if title1 < title2
 puts "yes"
else
 puts "no" #endi bulsa  "no".
```

end

- Ikkinchi parametr qaytarilgan maydon soni bilan chegaralanadi, quyidagi qoidaga rioya qilgan holda :
- Agar parametr tushurilgan bo'lsa, oxirida bo'sh maydon olib

tashlanadi.

- Agar parametr – ijobiy (musbat +) son bo'lsa, ko'rsatilgan

maydon soni qaytarib beriladi. Bo'sh maydonlar oxirida saqlanadi.

- Agar parametr – salbiy (nisbiy) son bo'lsa, orqaga

Qaytarilayotgan maydonlar soni chegaralanmagan bo'lib, oxirida bo'sh maydonlar saqlanadi.

Quyidagi misollar :

```
str = "alpha,beta,gamma,,"
list1 = str.split(",")    # ["alpha","beta","gamma"]
list2 = str.split(",",2)  # ["alpha", "beta,gamma,,"]
list3 = str.split(",",4)  # ["alpha", "beta", "gamma", ","]
list4 = str.split(",",8)  # ["alpha", "beta", "gamma", "", ""]
list5 = str.split(",",-1) # ["alpha", "beta", "gamma", "", ""]
```

Scan : uslubi boshqa qatorda yoki qatorlarni taqqoslashda xizmat qiladi.

```
str = "I am a leaf on the urind…."
# Qator so'zma – so'z interpritasiya qilingan, doimiy ifoda emas.
arr = str. scan (/\ w +/) # ["I", "am", "a", "leaf", "on", "the", "urind"] # blokni qo'yish mumkin. str. scan (/\ w +/) {|x| puts x}
```

String Scanner sinf standart kutubxonasidan farq qiladi, skanerlashni saqlab qoladi, bir vaqtning o'zida hammasini bajaradi. Reguire ' strscan'

```
str = "Qara, men qanday uchyapman !"
SS = String Scanner. new (str)
```

Word = SS. scan (/\ w +/) # bittadan so’z olish break if word. nil? sep = SS. scan (/\ w +/) # keying fragmentni olish # so’z bo’lmay keladi, break if sep. nil ? end

Nazorat savollari:

1. Oddiy qator nima maqsadda ishlatidi.
2. Qatorlarni Uzunligini olish.
3. Stringga ishlov berish. (Qator obrabotkasi)

6- mavzu. Belgilarni ASCII kodiga kiritish va qaytarish.

Reja:

1. qatorlarni sinflash.
2. Doimiy ifodalar sintaksisi.

Rubyda belgi butun son hisoblanadi. Kelgusida belgilarni bir belgili qator ko'rinishida saqlanadi.

```
Str = Martin
print Str [0] # 77
```

Agar Fixnum turdagi ob'ekt jumla oxirida yozilsa, u oldindan belgiga aylanadi.

```
Str 2 = str << 11 # "Martino"
```

6.1 Qatorlarni shifrlash.

Ba'zida qatorlarni yengil topish to'g'ri kelmaydi. Masalan, parolni ochiq qoldirish mumkin emas, faylga dastur (ruxsat) chegaralangan bo'lsa ham crypt standart uslubda DES algoritmi bo'yicha qatorni shifrlash uchun shu nomli standart funksiya qo'llaniladi.

U "затравку" parametr sifatida qabul qilinadi. Platformalarda, UNIXdan parametr boshqa bo'lishi mumkin.

Talkienni yaxshi ko'ruvchilar biladigan parol so'rovchi trivial qo'shimcha quyidagicha :

```
coded = "hfCqhHIE5LAM".
puts "Gapir, do'st, va Enterni bos !"
print "Parol :" password = qets. shop
if password. crypt ("hf") = = coded
puts "Xush kelibsiz !"
else
puts "Sen kim, ork ?"
end.
```

Server Web – prilojeniyalarda bunday shifrovkaga ishonmaslik kerak, chunki maydon shakliga kirgizilgan parol tarmoq bo'yicha ochiq ko'rinishda uzatiladi. Bu holda SSL protokoldan (Secure Sockets Layer) foydalanish mumkin. Serverda shifrdan foydalanishni hech kim ta'qiqlamagan, lekin boshqa sabablarga ko'ra tarmoq bo'ylab uzatish vaqtida emas, omborxonada parolni himoya qilish uchun.

Doimiy ifodalar.

Dasturlash quroli (инструменти) qilib doimiy ifodalar (регулярное варажение) kuchi olinishi ko'pincha e'tiborga olinmaydi. Birinchi nazariy tadqiqot bu mavzu bo'yicha o'tgan asrning 40 – yillariga to'g'ri keladi, 1960 – yillarida esa hisoblash sistemalari kiritildi. Keyinchalik UNIX Operasion sistemadagi turli instrumental vositalar kirib keldi.

1990 – yillari esa Perl tili mashhur bo'ldi. Doimiy ifodalar eski kutubxonasi yangi oniguruma nomi ostida yangilana boshladi.

6.2. Doimiy ifodalar sintaksisi.

Odatda doimiy ifodalar 2 tomonlama yotiq chiziq belgisi bilan belgilanadi. Yana %r shakli ham qo'llaniladi.

Jadval oddiy doimiy ifodalar

Doimiy Ifodalar	Izoh
|Ruby|	Rubydagi bitta so'zga mos keladi
|[Rr]uby|	Rubyga yoki rubyga mos
|^abc|	Bosh qatordagi abc ga mos
%r (*xyz*$)	Qatorning oxirida *xyz* ga mos
%r|[0 - 9]*|	Noldan yoki ko'proq sonlar ketma – ketligiga mos

Jadval Doimiy ifodalar modifikatorlari.

Modifikatorlar	Nimaga mo'ljallangan

I	Registrni ignor (e’tiborga olmaslik) qilish
O	Ifodani bir marotaba bajarish
M	Ko’p muddatli rejim (nuqta yangi qatordagi ramz bilan taqqoslash)
X	Doimiy umumiy ifoda (izoh va пробел (ochiq joy) mumkin)

Jadval Doimiy ifodalarda umumiy ishlatiladigan belgilar

Belgi	Izoh
^	(line) Matn qatorsining boshi yoki (string) qatorlar ramzi
$	Matn qatorsining yakuni (oxiri) yoki ramzlar qatorsi
○	Yangi qator ramzidan tashqari turli ramzlar (agar ko’p muddatli rejim o’rnatilmagan bo’lsa)
/W	Ramz – so’z qismi (raqam, harf yoki chiziq belgisi)
\W	So’z qismi bo’lmagan ramz
\s	Пропуск (пробел(bo’sh joy)), tabulyasiya belgisi, yangi qator ramzi
\S	Пропуск bo’lmagan ramz
\d	Raqam ([0 – 9 day])
\D	Raqam emas
\A	(string) ramzlar qatorsining boshlanishi
\Z	Ramzlar qatorsining tugashi yoki ramz oxirining yangi qatorsi pozisiyasi
\z	(string) ramzlar qatorsining tugashi
\b	So’z chegarasi (kvadrat qavsdan tashqari)
\B	So’z chegarasi emas
\b	Забой (faqat kvadrat qavs ichida [ ])
[ ]	Ramzlarni yig’indisi

*	O yoki oldingi ifodalar takrorlanishi
*?	O yoki oldingi ifodalar takrorlanishi (ochko’z bo’lmagan algoritm)
+	Bir yoki oldingi ifodaning takrorlanishi
+?	Bir yoki oldingi ifodalarni takrorlanishi (ochko’z bo’lmagan algoritm)
{m,n}	m dan n gacha oldingi ifodalar kirishi
{m,n}?	m dan n gacha oldingi ifodalar kirishi (ochko’z bo’lmagan algoritm)
?	0 yoki 1 oldingi ifodalar takrorlanishi
|	Alternativlar
(? =)	Oldinga qarash pozitivi
(? !)	Oldinga qarash negative
( )	Ifodalar guruhlari
(? >)	Kiritilgan ifodalar
(? :)	Saqlanadigan ifodalar guruhi
(?imx - imx)	Shu joydan boshlanadigan yoqish / o’chirish rejimi
(?imx – imx : expr)	Bu ifodalar uchun yoqish / o’chirish rejimi
(? #)	Izoh

Zamonaviy dasturchi uchun doimiy ifodalar bilan ishlay olish katta plyusdir. D J. Fridl “Регулярные выражения, (Питер, 2003 – yil) Zeffrey Friedl, Mastering Regular Expressions” kitoblarini o’qib chiqishni maslahat beramiz.

Doimiy ifodalarning kompilyasiyasi

Regexp. compile (Regexp. new sinonimi) uslubi doimiy ifodalar kompilyasiyasi qo'llaniladi. Birinchi parametr zarur, u qator yoki doimiy ifoda bo'lib keladi.

pat 1 = Regexp. compile ("^foo.*"|#^foo*)

pat 2 = Regexp. compile (/bar $/i) # bar/ (i o'tkazilmaydi)

2 chi parametr berildi, unda quyidagi konstantalar ishlatiladi :

Regexp : : EXTENDED,

Regexp : : IGNORECASE

Regexp : : MULTILINE

nil ifodasidan doimiy ifodalar registrlarni tanlab ololmaydi.

2 chi parametrni tushurib qoldiriladi.

options = Regexp : : MULTILNEII Regexp : : IGNORECASE

pat 3 = Regexp. compile ("^foo", options)

pat 4 = Regexp. compile (|bar|, Regexp : : IGNORECASE)

3 chi parametr ko'p baytli belgilarni yoqadi. U to'rt ifodadan 1 tasini qabul qiladi :

"N" yoki "n" – (поддержка) tayanch yo'qligini bildiradi.

"E" yoki "e" – EUC ni bildiradi.

"S" yoki "s" – Shift - /IS ni bildiradi.

"U" yoki "u" – UTF - & ni bildiradi.

Adabiy doimiy ifodani new yoki compile uslubini qo'llamay ishlatish mumkin. Uni chegaralanish belgilari (yotiq chiziq) bilan belgilash mumkin.

pat 1 = /^foo.*/

pat 2 = /bar $/i

Maxsus belgilarni ekranlashtirish.

Regexp. escape sinf usuli hamma maxsus belgilarni ekranlashtiradi.

Ularga yulduzcha, so'roq belgisi va kvadrat qavslar kiradi.

str 1 = "[* ?]"

str 2 = Regexp. escape (str 1) # “ \ [\ * \ ? \ ] ”

Regexp. quote usuli sinonimi bo’lib keladi.

Yakorlar.

Yakor – maxsus ifoda, qatordagi pozisiyaga mos, bu aniq belgi yoki belgilar ketma – ketligi emas.

$ va ^ - oddiy yakor bo’lib, belgilar qatorsi boshi va oxiriga mos.

Sting = “abc x def X ghi”

/def/ = ~ string # 4

/abc/ = ~ string # 0

/ghi/ = ~ string # 8

/^def/ = ~ string # nil

/def $/ = ~ string # nil

/^abc/ = ~ string # 0

/ghi $/ = ~ string # 8

Bu yakorlar (string) jumlalar belgilarga emas, (line) matn jumlasiga mosdir.

Bu namunalarni jumlaga qo’llasak, ichki yangi jumlalar belgisi o’zgaradi :

string = “abc\ndef\nghi” /def $/ = ~ string # 4,

/def/ = ~ string # 4, /^abc/ = ~ string # 0, /abc/ = ~ string # 0,

/ghi $/ = ~ string # 8, /ghi/ = ~ string # 8, /^def/ = ~ string # 4.

Biroq |A va|Z yakorlari jumla belgilarining boshi va oxiriga mos keladi.

/\ A def / = ~ string # nil / ghi \ Z \ = ~ string # 8

/\ def \ Z / = ~ string # nil

/\ A abc / = ~ string # 0

| z yakor | Z yakoridan farq qiladi. | Z yangi jumlada oxirgi belgini mosligini o’rnatadi, birinchisi esa |z mos tushishi aniq.

string = “abc|ndef|nghi” str 2 << “|n”

/ ghi \ Z / = ~ string # 8 |ghi \ Z | = ~ string # 8

/ ghi \ Z / = ~ str 2 # nil /\ A abc / = ~ str 2 # 8

|b yakori yordamida yoki pozisiyadan so’z chegarasida moslik o’rnatish mumkin, (|B| so’z chegarasida bo’lmasa.

gsub usuli yordamida yakor ishini ko’rsatish mumkin.

str = “this is a test”

str. gsub (/\ b /, “/”) # “|this| |is| |a| |test|”

str. gsub (/\ B /, “_”) # “t-h-i-s i-s a t-e-s-t”

Kvantorlar.

Doimiy ifodani apparatlar qismi zarur bo’lmagan elementlar va takrorlanish обработкаси (qayta ishlash) bilan bo’liq. So’roq belgisi bilan keladigan element zarur emas, u bo’lishi ham bo’lmasligi ham mumkin (Bu kvantor faqat nol bo’lmagan ifodalarga ishlatish mumkin, yakorlarga emas).

patern = / ax ? b /

pat 2 = / a[xy] ?? b /

pattern = ~ “ab” # 0

pattern = ~ “acb” # nil

pattern = ~ “axb” # 0

pat 2 = ~ “ayb” # 0

pat 2 = ~ “acb” # nil

Elementlar tez – tez noaniq raqamlar bilan takrorlanib keladi (kvantor + formulirovka uchun ishlatiladi).

Masalan : berilgan namuna musbat + ijobiy sonlarga to’g’ri keladi.

patterm = / [0 – 9] + /

patterm = ~ “1” # 0

patterm = ~ “2 3 4 5 6 7 8” # 0

Kvantor + U ? yordamida ifodalash mumkin.

Huzzah jumlasidan keyin nol yoki undov belgisi qo’yilishi aytiladi :

pattern = / Huzzah (!+) ? / # qavs zarur

pattern = ~ “Huzzah” # 0

pattern = ~ “Huzzah !!!!” # 0

Boshqa yaxshiroq usullar ham bor. Ular kvant * bilan belgilanadi.

pattern = /Huzzah ! */ # *

faqat ! belgisigagina qo’llaniladi.

pattern = ~ “Huzzah” # 0

pattern = ~ “Huzzah !!!!” # 0

Amerika ijtimoiy sug’urta raqamini qanday bilish mumkin ?

Quyidagi namuna yordami bo’yicha :

ssn = “987 – 65 - 4320”

pattern = /\ d \ d \ d - \ d \ d - \ d \ d \ d \ d \

pattern = ~ ssn # 0

Lekin bu uncha tushunarli emas. Har bir guruhda qancha raqam borligini aniq aytish kerak. Buni figurali qavsda takrorlanib kelayotgan sonlarni ko’rsatish kerak :

pattern = /\ d { 3 } - \ d { 2 } - \ d { 4 } /

Dastur o’qiydiganlarga tushunarli bo’ladi. Yana vergul bilan ajratilgan diapazon, chegaralardan foydalanish mumkin. Masalan, Elboniyadagi telefon 2 qismli raqamlardan iborat : 1- qismda 3 dan 5 gacha raqam, 2 chi qismda – 3 dan 7 gacha ; albonian_phone = /\ d { 3,5 } - \ d { 3,7} /

Pastgi va yuqorigi diapazon chegaralari zarur emas.

/ x {5} / # 5x mos

/ x {5,7} / # 5 – 7x ga mos

/ x {,8} / # 8 dan ko’p bo’lmaydigan moslik

/ x {3,} / # 3 ga mos.

Kvantor ?, + u * shunday yozish mumkin :

/ x ?/ # / x {0,1} / day

/ x */ # / x {0,} / day

/ x + / # / x {1,}/ day

Doimiy ifodalarni tasvirlashda ravshan terminlar bilan frazeologiyalar (iboralar) qo'llaniladi :

greedy – ochko'z

reluctant – faol emas

lary – dangasa

possessive – xususiy (shaxsiy).

Kod fragmentini (qismini, bo'lagini) ko'rib chiqamiz :

"Where the" jumlasi "Where the sea meets the" uzun jumla bilan taqqoslaymiz.

str = "Where the sea meets the moon – blanch d land",

match = /.*the /. match (str)

p match [0] # moslik # "Where the sea meets the".

Sababi operator * ochko'z moslikni bajaryapti, bunday bo'lmasligi uchun so'roq belgisi qo'yiladi :

str = "Where the sea meets the moon – blanch d land",

match = /.*? the/. Match (str)

p match [0] # moslik # "Where the". Bu kvantor + u {m,n} va kvantor ? larga ham xosdir.

"Eshitdim-unutdim, kurdim-bildim, bajardim-tushundim."

Belgi turi eng asosiydir. Belgi ma'lumotlari nima ? Qanday belgilar ? Qanday alifbodan ? Qanday har bir millatning o'ziga xos madaniyat belgilari ?

Avvallari xisoblash texnikasiga va informatikaga mutloq ingliz tili qo'llanilgan. Bu an'ana Charlz Bebbidju vaqtlari kiritilgan . Alifbo 26 harfdan iborat bo'lib, diakritik belgilar bo'lmagan.

Tabiiyki, har bir inson Web – sahifani, elektron pochtasini, va boshqa ma'lumotlarni o'z ona tilida o'qigisi keladi. Ba'zi alifbolar (alfavitlar) oddiy harflardan, ba'zilari bir necha ming belgilardan iborat bo'lib,

piktogrammalardan kelib chiqishini ko’rsatadi. Ba’zi tillarda bitta alfavit emas, bir nechta. Ba’zi tillarda yuqoridan pastga harflarni yoziladi, ba’zilarida esa o’ngdan –chapga.

Ba’zi tillarda harflar nuqtalar, chiziqlar, aylanalar, shtrixlar bilan bezatilgan.

25 yil ichida biz uzun yo’l bosib o’tdik, yomonmi yaxshimi xaotik tartibdagi tillarni va belgilarni o’zlashtirib oldik.

Turli til muhitiga ishlash uchun berilgan loyiha dasturi bilan ishlagan bo’lsangiz, internasionalizasiya nimaligini bilasiz. Bu bitta tabiiy tilni ushlab turish dasturning qobilyatidir.

Internasionalizasiya bilan multi – tillar va lokalizasiya uzviy bog’liq. Nima uchundir so’zlarni qisqartirib, o’rta harflarni yo’q qilib, o’rniga son qo’yiladi. O’chirilgan harflar nechta bo’lsa, raqamlar soni ham shuncha qo’yiladi.

```
def shorten (str)
(str [0…0] + str [1….- 2]. Length. to s + str [ - 1…- 1]). up case.
end
shorten (“internationalization”) # I 18 N
shorten (“multilingualization”) # M 17 N
shorten (“localization”) # h 10 N
```

I 18 N va M 17 N terminlari – sinonimlar, “globalizasiyalashtirish” termini boshqa ma’noga keladi.

h 10 N termini keng ma’noli (tushuncha) bo’lib, joydagi madanyati va kelishilgan holda, masalan valgota belgisi, vaqt va sana formatlash turlari, o’nlik sonidan kasr yoki butun sondan vergul, nuqtadan foydalanish kabilar qo’llab – quvvatlanadi.

3.1. Terminalogiya va tarixiy ma’lumotlar.

Kompyuter texnologiyalari kelib chiqish davrida perfokartlar va behisob belgilardan foydalanishardi.

ASCII kodi 1970 – yillarda paydo bo'ldi. ASCII abbreviaturasi (qisqartmasi) American Standart Code for Information Interchange (ma'lumot almashadigan Amerika standart kodi) deb o'qiladi. Kalit so'zi "amerikancha" bo'lgani uchun, Yevropa va Osiyo tillari loyihaga kirgizilmagan edi lekin bu kodda bir necha xatoliklarga yo'l qo'yilgan. ASCII belgilar yig'indisi 128 belgidan iborat edi (7 - razryadli). ASCII ni kengaytirish g'oyasi paydo bo'lib, 128 dan 255 kodlardan foydalanib, boshqa maqsadlarga yo'naltirildi.

Lekin IBM kompaniya va boshqalar bilan bu g'oya bir necha marotaba va har xil qilib hayotga tadbiq etilib, pand yedi. Chunki kelishilmovchilik yuzaga keldi. Masalan, 221 kod qaysi belgiga mosligi ?

Agar "oluvchi" va "yuboruvchi" foydalaniladigan belgi haqida kelishib olgan taqdirda ham, ular bir necha tilda muloqot qilisholmasdi, hammaga birdan belgi yetishmasdi.

Masalan, nems tilida yozilayotgan matnga, grek yoki ingliz tilida sitata qo'shmoqchi bo'lsangiz, muammo bo'lardi. Bunday sxema umuman Osiyo tillarga : xitoy, yapon, koreya tillariga umuman to'g'ri kelmasdi.

Bu masalani yechishda ikki yo'l bor edi. Birinchisi – keng belgilar, masalan har bir belgiga 16 bitdan foydalanish, ikkinchi yo'l – o'zgaruvchi uzunligidagi ko'p baytli kodirovkaga murojaat qilishdir.

Bu sxemada ba'zi belgilar yagona baytlar bilan, boshqalari 2 yoki 3 dan ortiq baytlar bilan ko'rsatiladi.

Bu holda ko'p savollar paydo bo'ladi. Xoxlagan jumla bir xil dekodirlanish kerak. Ko'p baytli belgi birinchi baytda maxsus sinfga mansub bo'lib, qo'shimcha bayt kutilardi.

Ammo ikkinchi va hokazo baytlar – chi ? Bir baytni belgilar yig'indisi bilan ruxsat beriladimi ?

Aniqlangan belgilar ikkinchi va uchinchi baytlar rolida chiqishga ruxsat etiladimi yoki man etiladimi ?

Jumlaning o'rtasiga o'ta olamizmi adashmay yo yo'q-mi ?

Jumlani (oxiridan oldinga) teskari yo'nalishda ko'rib chiqa olamiz-mi ? Turli kodirovkalar uchun turli loyiha yechimlari qabul qilindi. Shunday qilib Unicode kodirovka g'oyasi paydo bo'ldi. Buni "Butun dunyo belgi yig'indisi" deb hisoblang.

Afsus, amaliyotda bu oson emas Unicode 65536 ta belgi bilan chegaralanadi (turli kombinatsiyalar uchun 16 bit) loyihalashda bunday chegaralanish Unicodeda kiritilmagan edi. Bu ko'p baytli sxema edi. Unicodeda belgilar deyarli chegarasizdir, bu juda yaxshi, chunki 65 000 belgi butun dunyodagi tillarga hech ham yetmasdi.

Ichki jumla – baytlar ketma – ketligidir. Ko'z oldingizga keltiring, ASCII kodirovkasida mashina xotirasida 1 bayt saqlangan. "A lotin katta harf" bo'lsa, real 65 soni saqlanadi.

Nima uchun biz 65 ni "A" deb o'ylaymiz ? Chunki bu ifodani interpritasiyadan foydalanamiz.

Agar biz bu sonni boshqa son bilan qo'shsak, unda interpritasiyalanadi (foydalanadi) son bo'lib.

Agarda uni terminal bo'yicha ketma – ketlik aloqa liniyasidan yuborsak – demak ASCII belgi deb interpritasiyalaymiz.

Kodirovka – bu belgilar va ikkilash sonlari orasidagi moslikdir. Ruby Yaponiyada paydo bo'lib, ikkita turli yapon kodirovkasi bilan ajoyib foydalanadi (ASCII).

Eng mashhur kodirovka bu Unicodedir. Terminlar bilan tanishib o'tamiz. Narsalarni foydali nomlash – donolikning asoslaridan biridir !

Bayt – 8 bit. Baytni bir belgiga mos keladi deb ko'plar hisoblashadi. [18 N kontekstida bunday emas.

Kod pozisiyasi – jadvalning elementlaridan biri. Uning yordamida belgi yig'indisi ko'rsatiladi.

Glif (pechatlash belgisi) – kodli pozisiyaning vizual tasavvuri. Lekin belgi va vizual tasavvur turli narsalardir.

(Men tekst (matn) redaktorini ochib, turli shriftga "A" xarfini yozishim mumkin, lekin u baribir belgi "A" bo'lib keladi.)

Grafemalar – glifga yaqin tushuncha bo'lib, lekin grafemalar haqida biz konteksdagi (matndagi) tilda gapiramiz, dasturiy ta'minotda emas.

Grafema ikki yoki undan ortiq gliflar kombinatsiyasidan iborat. Foydalanuvchi konteksdagi (matnda) belgilarni o'z ona tilisida qabul qiladi. Farqi juda nozikdir, ko'pchilik dasturchilar bu haqda o'ylashmaydilar ham.

Belgi nimadir ? Unicode dunyosida ham aniq tushuncha bu predmet haqida yo'qdir. Chunki turli tillar o'zini turlicha tutadi, dasturchilar esa boshqa insonlardek o'ylashmaydi, fikrlashmaydi, ularning o'zlarini fikrlari bor.

Shunday qilib, ramz – belgi yozuvining abstraksiyasidir, vizual bir yoki bir necha vositalar bilan ko'rsatilgandir.

Endi potasiyaga o'tamiz. Unicodening ananali kodli pozisiyasi U + deb, keyinchalik yuqori registrda 4 yoki 16 dan ko'proq raqam bilan yoziladi. Lotin tilidagi "A" U + 0041 ko'rinishida bo'lishi mumkin. endi & # 233 harfini olamiz (e akut bilan). Unicodeda ikki usul bilan ko'rsatish mumkin. Birinchidan, bitta kodli pozisiya – U + 00E9 (Lotin E akut bilan). Ikkinchidan, 2 kodli pozisiya yig'indisi : jumlani e + diakrit akut belgi – U + 0065 va U + 0301. Ikkita shakl ham to'g'ri. Eng qisqasi – monolit shakl (precomposed) deb nomlanadi.

Lekin monolit varianti har bir tilga ham to'g'ri kelavermaydi, shuning uchun har doim ham bunday ramz bitta kodli pozisiyaga to'g'ri kelmaydi.

3.2. Listing. Web – saxifani UTF – 8 kodirovkasiga perekodirovka qilish.

```
require ` open – uri `
require ` iconv `
def get_web_page_as_utf 8 (url)
open (url) do |io|
source = io. read
type, * parameters = io. conten_type_parse # qayta kodlanmaydi, agar (x) HTML emas unless type = ~ %r !^(? : text/ html/ application/ x html + xml) $ ! return sourse end.
# Avval bosh sahifani tekshiramiz, server yuborgan :
if pair = parameters. assoc (‘charset’)
encoding = pair. last
# Keyin analiz qilamiz HTML :
elsif source = ~ ? \]*? charse = ([^|S””]+)|i
encoding = $ 1
# Agar aniqlash bo’lmasa, kodlash
# HTTP standartdagi
else
encoding = ‘ISO – 8859 - 1’
end
converter = I conv. new (‘UTF – 8 II IGNORE’,encoding)
retirn converter. Iconv (soource)
end
end
```

Taxmin qilaylik, Ruby o’rnatilgan operasion sistemada (tizimda) local aniqlangan, UTF – 8 dan farqli yoki UTF – 8 da emas Ruby OC bilan muloqotda (Win 32 uchun distributivda). Unda qo’shimcha muammolar paydo bo’ladi. Masalan, Windows fayl nomlarida Unicode qo’llab – quvatlaydi va Unicodeda mutloq tizim darajasida ishlaydi. Ammo hozirgi

vaqtda Ruby Windows bilan eskirib (eski bo'lgan) ketgan kodli sahifalar yordamida o'zaro harakatda. Ingliz tilli va ko'pchilik boshqa G'arbiy chop etilgan bu sahifalar 1252 (yoki WINDOWS -1252).

Dastur ichida UTF – 8 kodirovkasi bilan foydalanish mumkin, ammo fayldagi hamma nomlarni kodli sahifa bergan kodirovkaga o'tkazish kerak.

Buni I son v qilishga yordam beradi, eng asosiysi esdan chiqarish kerak emas, Unicodedagi hamma ramzlarni faqat ozginasini kodli sahifa yozishga qodir. Bundan tashqari, Ruby Windows uchun hozirchalik kodli sahifa yordami bilan nomlarni yozolmaydigan fayllarni ocholmasligini bildiradi.

Bu chegaralanish Mac OSX, linux va boshqa UTF – 8 lokalik sistemalarga (tizimlarga) qarashli emas.

3.3. Ma'lumotlar so'rovnomasi .

Ma'lumotlar справочниги – bir tildagi ma'lumotlar yig'indisi. Bu tushuncha (L 10 N) lokalizasiya konsepsiyasining ajralmas qismi.

G'oya shundan iboratki, tilga bog'liq jumlalarni boshqa dasturlardan ajratib olish. Eng yaxshi vosita qilib, Ruby g'oyani hayotga tadbiq qilish uchun Ruby – Gettext – Package kutubxonasidan foydalanadi. Uni oddiy qilib gettext nomlaymiz (uni fayli ham shu nomda) (gettext ! uchtasi bilan adashtirmang). Bu ajoyib kutubxonani Masao Muto (Masao Mutoh) yozgan. Uning rasmiy sayti http : // gettext, rubyforge, org /da, GNU ulitlarni sayti http : // www.gnu.org / Software / gettext /. da toppish mumkin. Gettext kutubxonasi bir nechta kutubxonalardan iborat. Asosiy funksiyalar доступ (ruxsat) uchun require ' gettext ' ni, qo'shimcha vositalar uchun esa require ' gettext / utils ' qo'shish kerak. Ma'lumotlar справочник gidan yuborilgan сообщение (ma'lumotni, xabarni) boshqa tillarga o'girish uchun foydalanamiz. Uning yordamida birlik va ko'plikni (bir fayl, ikki fayl) farqlash uchun ham kerak.

Bu qoidalar aniq tilga juda qattiq bog'liq. Odatda har bir kutubxona va qo'shimchalar o'zining shaxsiy axborot справочник iga ega.

Distributivga turli tillarga o'girilgan справочник lar yig'indisini kirish mumkin.

LANG va GETTEXT_PATH o'zgaruvchilar ham hisobga olinadi. Справочник uchun ikkita asosiy operasiya bor (ular bizning dasturimizdan tashqari) : boshlang'ich справочник shakllanishi uchun сообщение (axborot, ma'lumotni) Ruby – dasturning bosh matnidan chiqaramiz va mavjud справочника bosh matndan yangi сообщенияни kirgizamiz. Bu ichidan olish (ajratish) va qo'shish (birlashtirish) operasiyasi.

3.3.2. Xabar spravochniki bilan ishlash.

Sizning kompyuteringizda gettext kutubxonasi o'rnatilgan bo'lmasa, gem install gettext komandasini (buyrug'ini) bajaring. Buning uchun GNU utilitlar kerak bo'ladi.

Agar UNIX sistemasida (tizimida) ishlayotgan bo'lsangiz, ular kirgizilgan (o'rnatilgan) bo'ladi. Win 32 platformalariga Windows uchun Clade / GTK + ni o'rnatish mumkin, bir yo'la GNV ulitalarni olishingiz mumkin. Ular bajarish vaqtida emas, faqat разработка jarayonida zarur . race dasturi bo'lmasa, gem – paketdan uni o'rnating. Bu qo'shimcha qulaylik. Справочник bilan ishlash oldidan, terminalogiya bilan tanishaylik :

PO – fayl – bu ko'chiriladigan ob'ektli fayl. Bu matnli (insonga tushunarli) сообщение справочниги. har bir faylda turli lokal variantlari uchun bor.

POT – fayl – bu shablon MO – fayl – bu ko'chiriladigan справочник ning ikkilik fayli. U PO – fayldan yaratiladi. Ruby uchun kutubxona MO – fayllardagina o'qiy oladi, PO – fayllarni emas. Matnli domen – MO – faylning bazaviy nomi.

Sonli uslublar, usullar.

> Parlament a’zolari menga ikki marta shunday savol berishdi : “Ayting-chi, mister Bebbidj, agar mashinangizga noto’g’ri sonlarni kiritsangiz, to’g’ri javob ololasiz-mi ?”. Bunday axmoqona savollarni beradigan kishining fikrlash qobilyati qanday darajada ekanligini tasavvur ham qilolmayman.
>
> Charlz Bebbidj.

Sonlar – har bir kompyuter uchun eng birinchi ma’lumotlar turi. Son bo’lmagan bilim sohasini topish qiyin. Hisobchi, havoga uchadigan apparat konstruktori bo’lasizmi, sonsiz ishlolmaysiz.

Zamonaviy tillarday, Ruby ham turli sonlar bilan ishlashni biladi. Uning ichida to’liq matematik operatorlar va funksiyalar, Bignum, Big Decimal va Rational sinflari bor.

Tizimli va standart kutubxonalardagi raqam maninulasiyasi uchun bo’lgan vositalar va trigonometriya, matematik analiz, statistika ko’rib chiqamiz (kitobning boshqa bo’limlarida).

Nazorat savollari

1. Oddiy qator haqida tushuncha nimalardan iborat?
2. Qatorlarni Uzunligini olish qayday amalga oshiriladi?
3. Stringga ishlov berishdan maqsad nima?

7-mavzu. **Ruby dasturlash tilida Android operatsion tizimi uchun dasturlash texnologiyasi**

Ruby(inglizcharuby – yoqut, la'l) - dinamik,interpretatorli yuqori darjadagi dasturlash tilidir. Rubytezkorvaqulayobyektgayo'naltirilgandasturlashhisoblanadi. U yuqori aniqlikda operatsion tizimga murojat qila olish imkoniyatiga ega, kuchli dinamik tuzulishga ega, dasturda uchraydigan keraksiz kodlarni yig'uvchi modul va boshqa imkoniyatlarga ega til. RubytiliPerlvaEiffel tillariga sintaktik jixatdan yaqindir, obyektga yo'naltirilganli jixatdan esa Smalltalk tiga o'xshash. Bazi bir jixatlari Python, Lisp, Dylan tillariga yaqin.

Ruby logotipi

Mazkur til istalgan operatsion tizimda ishlash xususiyati bilan qolgan tillardan ustunlikka ega.

Ruby asoschisi — YukixiroMatsumoto tomonidan yaratilgan. Rubyni yaratish 1993yil 23 fevralda boshlangan va 1995 yilda muvafaqiyatlar ko'ringan. U Perl tiliga o'xshagani uchun pearl — «marvarid» ruby — «yoqut» deb nomlangan.

Rubyda aniq kod yozilish uchun ketma ketlik talab etilmaydi va u dasturchi tomonidan doim o'zgartirib borilishiga ruxsat beriladi.

Rubyda dastur yozish uchun quyida misol keltirilgan

bilanqatornioxirigaqadarizoxyoziladi

= belgisi tenglash tizish belsi hisoblanadi

«"» - belgisi manipulyatsiya qiluvchi belgi hisoblanadi

str = "Salom Ruby" – str degan o’zgaruvchi elon qilindi va qiymat berildi

def - funksiyani elon qiluvchi kalit so’zi

( ) – ichida funksiya parametrlari elon qilinadi

end – rubyning qarcha amallarini yakunlaydi

Misol:

```
str = "Salom Ruby"
def str.bye
“Xayr”
end
puts str.bye
```

Rubyda barcha funksiyalar qiymat qaytaradi.

Shart operatori sintaksisi

```
puts( if 5 > 3 then "ha" else "yo’q" end )
```

Rubyning ustunligining yana biri massiv o’lchami har qanday tip bo’lishidan qatiy nazar avtomatik o’zgartiriladi. Masivni elon qilish

```
a = [1, 'hi', 3.14, 1, 2, [4, 5] * 3]
```

Massiv indeksiga murojat

```
a[2]
```

massivni bir xil qiymatlardan tozalash

```
a.flatten.uniq
```

qiymati 5 ga teng bo’lgan element indeksini qidirish

`a.index(6)` agar topilmasa nil(Javadagi null) qaytariladi.

O’zgaruvchi nomi ham o’zgarishi mumkin uni o’zgarmas deb elon qilish uchun !bilan yoziladi

```
a.flatten!
```

kodning 2 xil blok usuli bor

```
{ puts "Hello, World!" }
do puts "Hello, World!" end
```

tilning ko’plab ichki metodlari bor

```
File.open('file.txt', 'w') {|file|
```

File.txt ni yozish uchun ochish

```
file.puts 'Wrote some text.'}
```

Bu konstruksiya fayl yozpiq bo'lsa ham malumotni yoza oladi

Rubyda dastur kodi qisqa lekin samarali bo'ladi. Misol uchun 1 dan 10 gacha sonlar kvadratini olib insdekslari bo'yicha saralash kodi:

```
(0..10).collect{ |v| v ** 2 }.select{ rand(2).zero? }.map.with_index { |*v| v }
```

Klaslar, opertorlarni qayta yuklanishi:

```
class Person < Object     # Person klasi elon qilindi va object klasidan nasl oldi
include Comparable        # klasdan nusxa olindi
mix MyModel               # o'zgaruvchi nomidan nusxa olindi va
extend MyModel# klas metodiga <, <=, ==, >=, > , between amallari kiritildi
@variable                 # o'zgaruvchi olindi
@@count_obj = 0           # o'zgaruvchi olindi va qiymat berildi
def initialize(name, age) # name, age –qiymatlarini qabul qiluvchi funksiya elon qilindi
@name, @age = name, age# @ kalit belgisi bilan o'zgaruvchilar elon qilindi
@@count_obj += 1   # hisoblagich kiritildi
end # funksiya tugatildi
def<=>(person)    # <=> yordamida funksiya qayta yuklandi
@age <=> person.age #oxirgi elon qilingan qiymatga tenglashtirildi
end
def to_s      # puts uchun chop etish usuli aniqlandi
"#{@name} (#{@age})"          #"#{x}"konstruksiyasi 2langan qavsni ifodaalaydi
end
def inspect
"<#{@@count_obj}:#{to_s}>"
end
attr_reader :name, :age
end
# massiz obyektlarini elon qilish
```

```
group = [ Person.new("karim", 20),
Person.new("sobir", 63),
Person.new("Abbos", 16) ]
# => [<3:karim (20)>, <3:sobir (63)>, <3:Abbos (16)>]
                        # masivni saralash: teskari tartibda
puts group.sort.reverse # chop etiladi:
                           # karim (63)
                           # sobir (20)
                           # Abbos (16)
# agar qiymat kiritlgan bo'lsa funksiyaga murojat

group[0].between?(group[2], group[1]) # => true
```

Rubyni qo'llovchi dasturlar.

Kompyterlarda

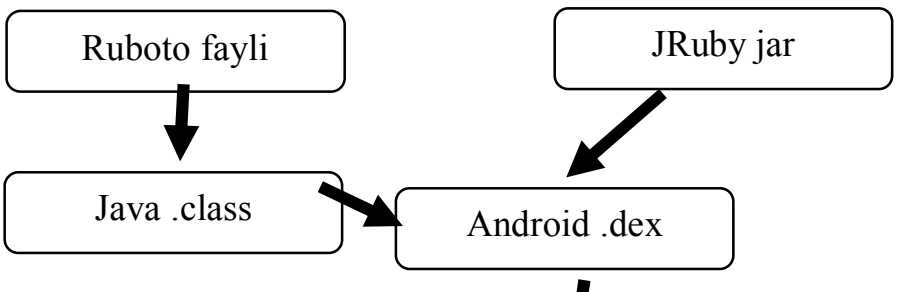
- Ruby on Rails — veb saxifalar yaratuvchi freymwork
- RubyGems — Ruby paketlar menedjeri
- IronRuby — Microsoft .NET uchun Rubyni qo'llovchi dastur

Mobil qurilmalarda

- Titanium Studio - HMTL5, CSS3, Javascript, Ruby, Rails, Python, PHP larni mobilda qo'llovchi dastur
- Ruboto — Ruby yordamida Android uchun dasturlash
- RubyMotion — Ruby yordamida IOS uchun dasturlash
- MobiRuby — Ruby yordamida Android va IOS uchun dasturlash

Ruboto . Ruboto (Ruby on Android)- android operatsion sistemasi uchun dastushda ruby tilidan va kutubxonalaridan foydalanuvchi dasturiy vosita. Ruboto yordamida Android API , Java, Ruby dan foydalanib Ruby tilida dasturlash amalga oshiriladi

Rubotoning ishlashi printspi quyidagicha:



Rubotoni kompyuterga o’rnatish va foydalanish uchun kerakli dasturlar:

- Java JDK(http://www.oracle.com/technetwork/java/javase/downloads/index.html)
- Android SDK(http://developer.android.com/sdk/index.html)
- JRuby(http://jruby.org/download)
- Apache Ant(http://ant.apache.org/bindownload.cgi)

Ishni birinchi navbatda Java dasturini o’rnatishdan boshlash kerak. Hozirgi misol Windows OT uchun ko’rib chiqilganva internet tarmog’iga kamida 128Mbit/s tezlikda ulanilgan bo’lish lozim. Dasturlash uchun Javaning kamida 6 versiyasi kerak bo’ladi. Javani o’rnatamiz va JRuby paketi ishlashi uchun komandalar qatoridan PATH ni kiritamiz

```
path=%path%;d:\Java\jdk1.7.0_51\bin;
classpath=%classpath%;.;
JAVA_HOME=d:\Java\jdk1.7.0_51
```

JRuby paketini `http://jruby.org/download` dan yuklaymiz. Yuklash yakunlangandan so’ng o’rnatamiz. O’rnatish paytida standart sozlama tanlash lozim. Agar oldindan dastur ishlash papkasi yaratilgan bo’lsa sozlamani shu papka bo’yicha o’zgartiramiz. O’rnatish to’g’ri amalga oshirilganini bilish uchun komandalar qatoridan

```
jruby –v
```

buyrug’ini berish kerak. Agar xarakat to’g’ri amalga oshgan bo’lsa JRuby versiyasi haqida malumot qaytariladi.

Rubotoni o’rnatish uchun komandalar qatoridan

```
jruby -S gem install ruboto
```

buyrug’i beriladi.

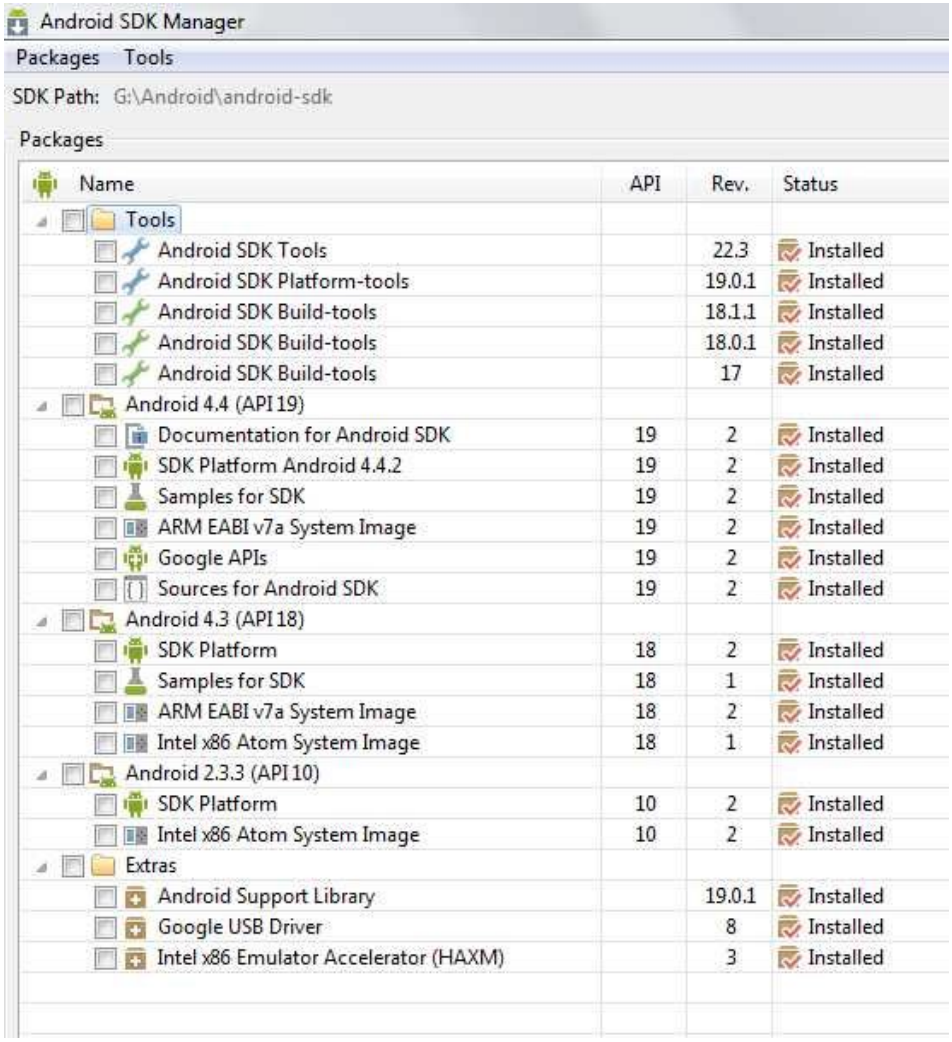
Ruboto imkoniyatlari:

- Ruby va Android papkalari fayllarini hosil qiladi
- Ruboto dastur kodlari va qo’shimchalarini kompilyatsiya qiladi
- Ruboto va dasturiy qismlarini sozlaydi va versiyani yangisini o’rnatadi
- JRuby componentalarini qo’llaydi
- Emulyatorni ishga tuzshiradi va sozlaydi

Apache Ant dasturini http://ant.apache.org/bindownload.cgi manzildan yuklaymiz va arxivdan kerakli papkaga bo’shatamiz. Papkani PATH deb elon qilamiz

```
path=d:\apache-ant-1.9.3\bin;
```

Android SDK nihttp://developer.android.com/sdk/index.html manzilidan yuklaymiz va o’rnatib, ishga tushiramiz. Ishchi oynada SDK ning kerakli paketlarini yuklab o’rnatamiz.



Android o’rnatilgan papkaning tools, build-tools, platform-tools ni PATH ga kiritamiz. Android o’rnatilgan papkaning o’zini ANDROID_HOME ga kiritamiz.

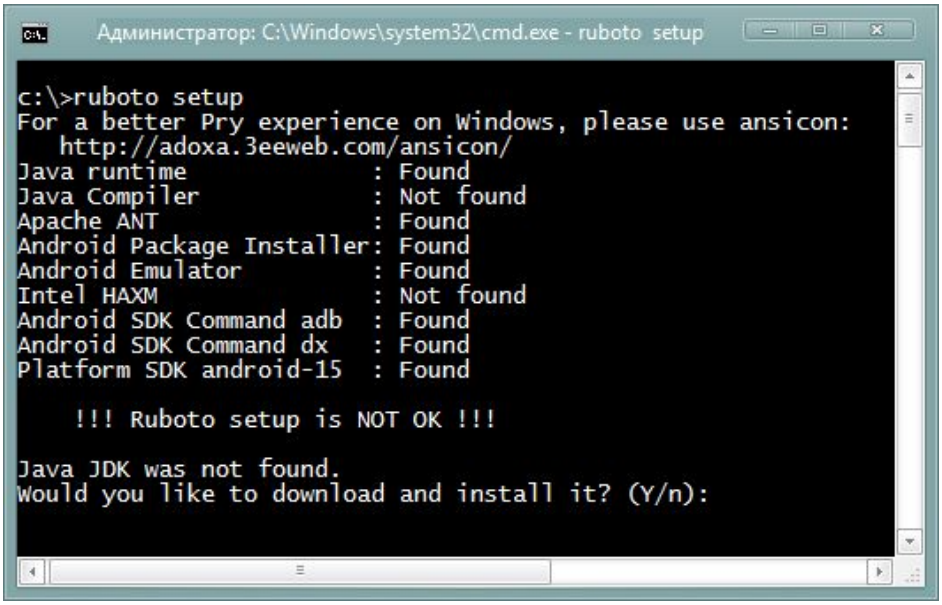
path=D:\Android\android-sdk\platform-tools;D:\Android\android-sdk\tools;D:\Android\android-sdk\build-tools\18.1.1;

```
ANDROID_HOME= D:\Android\android-sdk;
```

Kompyuterni qaytayuklab barchasini tekshirish uchun ruboto sozlamasini tekshiramiz

```
ruboto setup
```

Dastur kamchiliklari aniqlansa buyruq davom etadi va uni avtomatik bartaraf etadi. Kamchiliklar yo’q bo’lsa maxsus xabar chop etiladi.



Yoziladigan dasturni emulyator orqali test qilish uchun emulyatorni hosil qilish talab etiladi. Emulyator yaratish mumkin bo’lgan versiyalarni bilish uchunandroid list target buyrug’i beriladi. Yangi emulyatorni hosil qilish

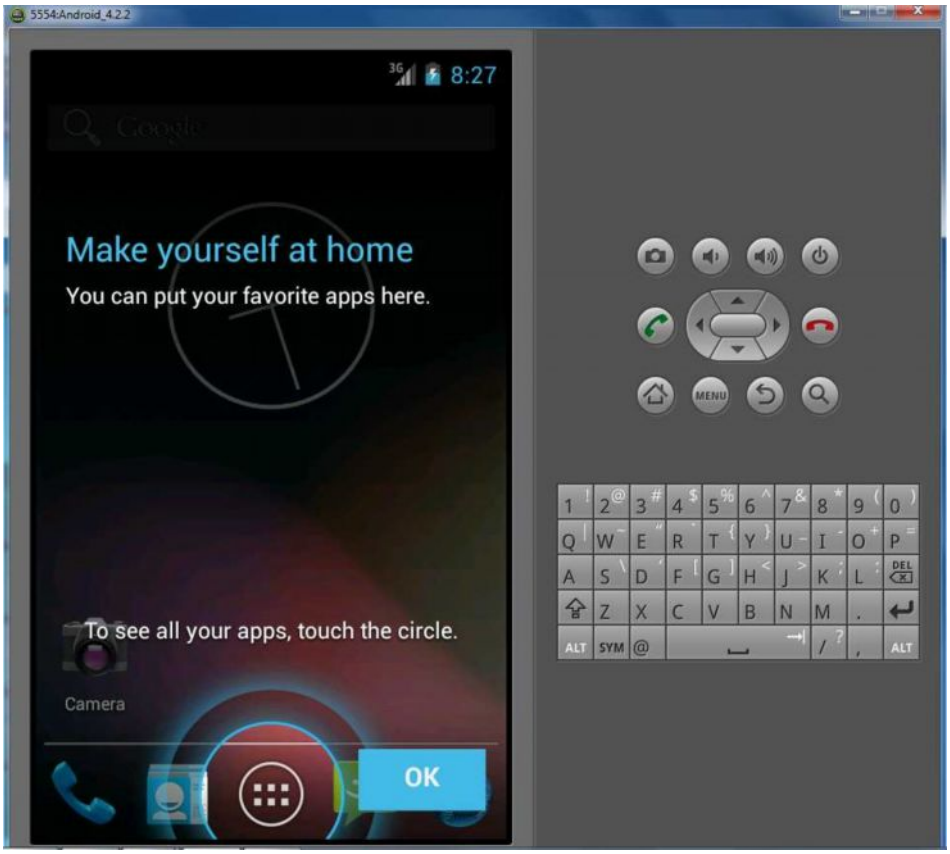
```
android -s create avd -f -n Android_4.3 -c 1000M -t android-18 --abi x86
Android 4.3 is a basic Android platform.
Do you wish to create a custom hardware profile [no]no
Created AVD 'Android_4.3' based on Android 4.3, Intel Atom (x86) processor,
with the following hardware config:
hw.lcd.density=240
vm.heapSize=48
```

```
hw.ramSize=512
```

birozvaqtdan so’ng yangi emulyator xosil qilinadi, uni ishga tushirish uchun

ruboto emulator -t android-18

buyrug’i beriladi. Dastur ishga tushgandagi oyna:



Yangi dasturni yaratish uchun maxsus papka ochiladi. Masalan C:\\ruboto . Ruboto yangi dasturni xosil qilishi uchun JRuby, Android SDK , Java dasturlari xatosiz ishlab turishi talab qilinadi. Yangi dasturni yaratish uchun o’sha papkaga kirish lozim. Yangi standart dasturni yaratish buyrug’i

```
c:\ruboto>ruboto gen app --package org.ruboto.example.quick_start --target android-18
```

gen app yangi dastur laratilishi haqida buyruq

--package android paketi manzili. Paket C:\ruboto papkasida yaratiladi

--target kompilyatsiya qilinuvchi android versiyasi

Yangi dastur kodi`C:\ruboto\quick_start\src` manzilda joylashadi, asosiy dastur fayli dastur nomi bilan bir xil bo’ladi. Asosiy dastur fayli quyidagicha: nomi - fayquick_start_activity.rb

```
require 'ruboto/widget'
```

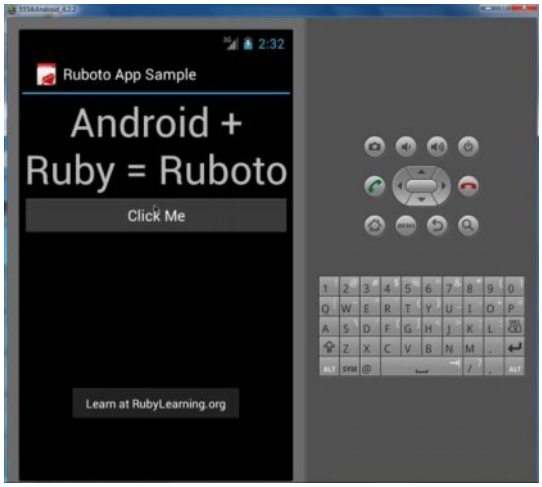
```
require 'ruboto/util/toast'
ruboto_import_widgets :Button, :LinearLayout, :TextView
# http://xkcd.com/378/
class QuickStartActivity
def onCreate(bundle)
super
 set_title 'Ruboto app sample!'
    self.content_view =
linear_layout :orientation => :vertical do
          @text_view = text_view :text => 'Ruby on Android
rocks!', :id => 42, :width => :match_parent,:gravity => :center,
:text_size => 48.0
          button :text => 'Click me', :width => :match_parent,
:id => 43, :on_click_listener => proc { butterfly }
end
rescue
puts "Exception creating activity: #{$!}"
puts $!.backtrace.join("\n")
end
private
def butterfly
    @text_view.text = 'Android + Ruby = Ruboto'
toast 'Learn at RubyLearning.org'
end
end
```

Dasturni kompilyatsiya qilish uchun `rake`o’rnatish uchun `install`ishga tushirish uchun `start`buyruqlaridan foydalaniladi. Biroz vaqtdan keyin kompilyatsiya bo’lganli haqidagi habar chiqadi.

```
BUILD SUCCESSFUL
Total time: 8 seconds
adb shell date -s 20131120.102437
Wed Nov 20 10:24:37 EST 2013
Installing package org.ruboto.example.quick_start
pkg: /data/local/tmp/QuickStart-debug.apk
Success
```

```
526 KB/s (74474 bytes in 0.138s)
```

Natija :



Rubotoda SQLite malumotlar ombori bilan ishlash

Androidda SQLite malumotlar bazidan foydalaniladi. Bu imkoniyatlar foydalanish uchun Ruby dasturlash tili foydalanishi uchun SQLDroid dasturi talab qilinadi. Dasturni https://github.com/SQLDroid/SQLDroid manzilidan yuklab olish o'rnatish kerak.

Dasturni hosil qilish buyrug'i

```
ruboto gen app -t 15 --package org.ruboto.example.sqliteda_ishlash
```

Dastur SQLiteni qo'llashi uchun quyidagi buyruq berilishi kerak

```
source "http://rubygems.org"
gem 'activerecord', '<4.0.0'
gem 'activerecord-jdbc-adapter', '<1.3.0'
gem 'activerecord-jdbcsqlite3-adapter', '<1.3.0'
gem 'sqldroid'
```

Dastur papkasida yangi fayl yaratamiz `src/android_logger.rb`

```
class AndroidLogger
def self.debug(*args)
    # Java log yaratildi::android.util.Log.d *args
puts *args
end
def self.debug?
true # funksiya qayta yuklandi
end
```

```
    def self.info(*args)
        # Java info log yaratildi::android.util.Log.i *args
    puts *args end
    def self.warn(*args)
        # Java elon log yaratildi ::android.util.Log.w *args
    puts *args
    end
    def self.error(*args)
        # Java xato log yaratildi::android.util.Log.e *args
    args.each do |m|
    if m.is_a? Exception
    puts m.message
    puts m.backtrace.join("\n")
    end
    puts m
    end
    end
    end
```

Malumotlar bazasini hosil qilish

src/database.rb

```
require 'active_record'
require 'android_logger'
class Database
def self.setup(context)
    ActiveRecord::Base.logger = AndroidLogger
    db_dir = "#{context.files_dir.path}/my_db.sqlite"
    connection_options = {
:adapter => 'jdbcsqlite3',
:driver => 'org.sqldroid.SQLDroidDriver',
:url => "jdbc:sqldroid:#{db_dir}",
:database => db_dir,
:pool => 30,
:timeout => 25000,}
    ActiveRecord::Base.configurations = {
:production => connection_options }
```

```
ActiveRecord::Base.establish_connection(ActiveRecord::
Base.configurations[:production])
end
def self.migrate(context)src_dir = 'file:' +
context.package_manager.getApplicationInfo($package_name, 0).
sourceDir + '!/'
    migration_path = File.expand_path("#{src_dir}/migrate")
puts "Looking for migration scripts in #{migration_path}"
migrator = ActiveRecord::Migrator.new(:up, migration_path)
if migrator.pending_migrations.size > 0
puts "Found #{migrator.pending_migrations.size} migrations."
      migrator.migrate
end
end
end
```

Yuqoridagi kodda dasturning ishchi papkasida `my_db.sql` malumotlar fayli hosil qilinadi.

```
db_dir = "#{context.files_dir.path}/my_db.sqlite"
```

Malumotlar bazasida yangi jadval hosil qiluvchi fayl yaratish.

```
src/migrate/001_create_people.rb
class CreatePeople < ActiveRecord::Migration
def self.up
create_table :people do |t|
t.string :name, :limit => 32, :null => false
t.date :birthdate
      t.timestamps
end
add_index :people, :name, :unique => true
end
def self.down
drop_table :people
end
end
```

Yuqoridagi kodlar yordamida people jadvali hosil qilindi.

```
create_table :people do |t|
```

Jadvalda name ustuni elon qilingan va null qiymat qabul qilmaslik 32 kenglikkacha qiymap qabul qilish parametrlari kiritilgan.

```
t.string :name, :limit => 32, :null => false
```

name qiymatlarni bir xil nom bilan qabul qilmaslik sharti kiritlgan.

```
add_index :people, :name, :unique => true
```

Dasturning asosiy kodlari src\sqliteda_ishlashfaylidan joy olgan.

Foydalanilgan adabiyotlar

1. Х.Фултон. Программирование на язике Ruby: Москва - 2007.684с.
2. Подбельский В.В. Язык Си++: Учеб. пособие. - 4-е изд.- М.: Финансы и статистика,
3. Uzakov Z. The text of lectures on the subject " Java Programming ". Karshi, 2002, 56 pages
4. Д.Родли., Создание Java апплетов.: ДиаСофт.,Киев-1996. 380с
К.Джамса, С.Лалани, С.Уикли., Программирование в Web для профессионалов.:Попурри.,Минск.- 1997.630с.
5. Шарма В., Шарма Р. Разработка Web-серверов для электронной коммерции. - 2001., 350с.
6. Кубенский А. Создание и обработка структур данных на Java.: BHV., СПб - 2000.,322с.
7. Майкл Эферган **Java: справочник**.- QUE Corporation, 1997,
Издательство "Питер Ком", 1998

Internet tarmog'i ma'lumotlari

1. **Ruby on Rails http://rubyonrails.org/**
2. **http://www.mondrian-ide.com/**
2. **http://www.intuit.ru**
3. **http://www.ziyonet.uz**