

## Metódusok

1

## Metódusok

- minden alprogram metódus
  - nincsenek függvények/eljárások
- nem kötelező osztályba ágyaznunk a főprogramot
  - a Ruby automatikusan megteszi ezt
  - a főprogram is objektum (Object típusú)
- a főprogramban definiált metódusok erre az objektumra hajtódnak végre
- transzparens
  - teljesen objektum-orientált
  - de továbbra is természetesen használható

2

## Metódusok definiálása

- `def method_name(arg1, arg2, arg3)`
  - ...
  - `end`
    - metódusok nevét kisbetűvel kezdjük
      - konvenció
      - Ruby engedné a nagybetűt is
- nevesített argumentumok
  - paraméterátadás nem egy tömbön keresztül történik
- `def method_name`
  - ...
  - `end`
- nem kötelező kitenni a zárójelot, ha nincs paraméter

3

methods.rb

## Metódusok definiálása

- metódus neve
  - végződhet ? karakterrel
  - végződhet ! karakterrel
- visszatérési érték az utolsó kifejezés értéke
  - `return` kulcsszóval explicit jelezhető
- a formális paraméterekhez a metódus törzsére lokális változó tartozik
  - nem érhető el kívülről
  - élettartama kicsit hosszabb is lehet
    - mark-and-sweep gc
- a Ruby futási időben ellenőrzi az aktuális és a formális paraméterek számát
  - hiba, ha ezek nem egyenlőek

4

default-args.rb  
default-args-log.rb

## Metódusok definiálása

- `def method_name(arg1 = "def", arg2 = "def2")`
  - ...
  - `end`
- default értékek megadhatók minden paraméter számára
- ha egy paraméternek default értéket adunk, akkor az összes utána következőnek is kötelező
  - ellenkező esetben nem lehetne az elhagyott paramétereket kezelni
- default argumentum értéke nem csak konstans lehet
  - lehet kifejezés
  - lehet egy másik argumentum értéke

5

modulo.rb

## Változó argumentumszám

- ha nem fix a paraméterek száma
- `def method_name(arg1, arg2, *rest)`
  - ...
  - `end`
- a \* jelöli, hogy a maradék argumentumok kerüljenek összegyűjtésre
  - rest egy tömb lesz (Array)
- csak egy lehet
  - az első mindenképpen benyelni az összes elemet
  - szintaktikai hiba is

6

## Iterátorok implementálása

- iterátor
  - olyan Ruby metódus, amely képes blokkot fogadni
  - valamilyen (absztrakt) adatszerkezet elemeinek
    - ... felsorolására
    - ... feldolgozására
  - egyéb esetek is elképzelhetők
    - pl. File.open
- egy metódus legfeljebb egy blokkot kaphat paraméterként
  - alternatív lehetőségekkel ez megkerülhető, de blokk csak egy lehet

7

## Iterátorok implementálása

- az iterátor megtudhatja, hogy kapott-e blokkot
  - block\_given?
- meghívhatja a blokkot
  - yield
  - paraméterül csak a blokknak szánt paramétereit kell adni
  - argumentumszám-ellenőrzés futási időben
    - nem egyezés nem vezet hibához
    - ha kevesebb, a maradék nil lesz
    - ha több
      - 1 argumentuma van a blokknak, akkor a blokk egy Array objektumot kap + egy warning
      - 0 vagy több, mint egy argumentuma van a blokknak, akkor a többi argumentum figyelmen kívül marad

8

## Iterátorok implementálása

- ```
def my_iterator(arg1, arg2)
  yield(arg1, arg2)
  ...
end
```
- az argumentumlistában nem jelenik meg a blokk
  - block\_given?
- a yield paramétereit kapja meg a blokk paraméterként

9

## Proc objektumok

- kód blokkokat kezelhetünk objektumként
  - closure
- Proc.new
  - új Proc objektum létrehozása
  - a paraméterül adott blokkot alakítja Proc objektummá
  - ha nincs paraméterül adott blokk, akkor a hívás helyén az aktuális blokk
- később a Proc objektum meghívható
  - prc.call(...)
  - prc[...]
- megtudható az aritása
  - prc.arity
- Kernel#proc

10

## Blokk paraméterek

- blokk paraméterek kezelhetők
  - Proc objektummá konvertálhatóak
- ennek két módja van
  - 1 Proc.new hívás anélkül, hogy paraméterül adnánk blokkot
  - 2 utolsó paraméter elé & jelet kell tenni a formális paraméterek listájában
    - def method\_name(arg1, &block\_arg)
    - ...
    - end
    - ha nincs blokk, akkor nil

11

## Metódushívások

- receiver.method\_name(arg1, arg2) { ... }
  - receiver: az üzenetet fogadó objektum
  - a metódus neve
  - argumentumok
  - opcionális blokk paraméter
- zárójel az argumentumok körül elhagyható
  - legtöbb esetben célszerű kitenni
- receiver
  - lehet osztály, vagy modul is
    - File.size("filename")
    - Process.fork
- receiver
  - ha elmarad, akkor a self lesz

12

## Osztályok

13

## Osztályok

- Ruby-ban minden objektum, mindennek van típusa
- az osztály is objektum
  - típusa: Class
  - MetaClass nincs
- a new az osztály, mint objektum metódusa
  - nem kulcsszó
  - MyClass.new(args)
- nincs többszörös öröklődés
  - helyette: modulok
    - jobb, mint a Java interface-k
- egy osztály definíciója mindig nyitott marad

14

## Osztályok definíciója

- osztályok neve konstans
  - nagybetűvel kell kezdődnie
  - ha nem, akkor a fordító hibát jelez
- class MyClass
- ...
- end
- osztály törzsében
  - metódus definíciók
  - attribútum definíciók
  - alias definíciók

15

## Konstruktorok

- Ruby a new metódusban helyet foglal, inicializál
  - ezt nem célszerű felüldefinálni
    - Class#new
  - meghívja az osztály initialize metódusát, ha definiált
    - ezt kell definiálni
    - példánymetódus
- class MyClass
- def initialize(arg1, arg2)
- ...
- end
- end
- nem szükséges visszatérési értéket megadni

16

## Destruktorok

- Ruby nem tartalmazza ezt a feature-t
  - bonyolítaná és lassítaná a gc működését
- ObjectSpace.define\_finalizer
  - egy objektumhoz definiálható egy finalizer
  - a finalizerben az objektum már nem elérhető, csak egy object ID-t kap
  - szándékos a definíció furcsa helye
- ObjectSpace.define\_finalizer(object) { |id|
- ...
- }

17

file.open.rb

## Destruktorok

- helyette az iterátorok nyújtotta előnyök használhatóak ki
- például a File.open
- osztálymetódus
- a megnyitott file reprezentáló objektum csak a paraméterül adott blokkban létezik
- File.open { |file|
- ...
- }
- - megnyitja a file-t
- - a file objektumot átadja a blokknak
- - kezeli, ha fellép valamilyen kivétel
- - végül mindenképpen lezárja a file-t

18

## Objektum változók

- az objektumhoz tartozó változók neve @ jellel kezdődik
  - @attribute
- minden metódusban használt egyéb változó lokális
- ```
def my_method  
  @attr = 2  
end
```
- nem inicializált értékű objektum változóra való hivatkozás nem fatális hiba
  - értéke nil
  - warning!
- az osztályon kívülről nem érhető el sehogya

19

## Osztály változók

- osztály változók @@ jellel kezdődnek
  - @@my\_class\_variable
- inicializálatlan osztály változóra való hivatkozás hiba
  - futási idejű hiba
- az osztályon kívülről nem elérhető

20

## Osztály konstansok

- láthatóak az osztályon kívülről is
- :: minősítővel érhetőek el
- ```
class MyClass  
  MyConstant = 1  
end
```

  - MyClass::MyConstant

21

## Osztály metódusok

- egy osztály definícióján belül osztály metódusok is megadhatók
- ```
class MyClass  
  def my_instance_method  
    ...  
  end  
  
  def MyClass.my_class_method  
    ...  
  end  
end
```
- Class.new
- File.delete(filename)

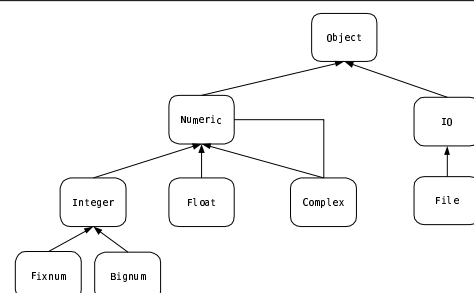
22

## Öröklődés

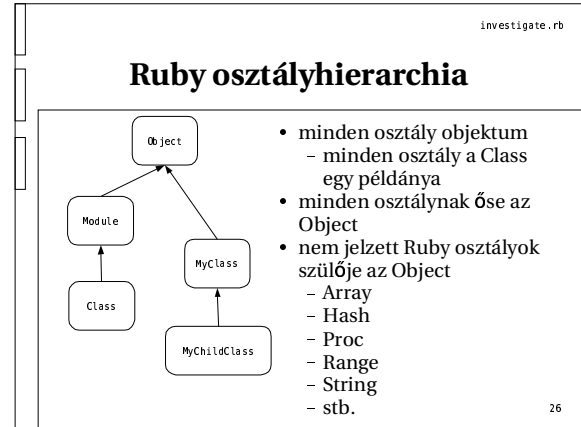
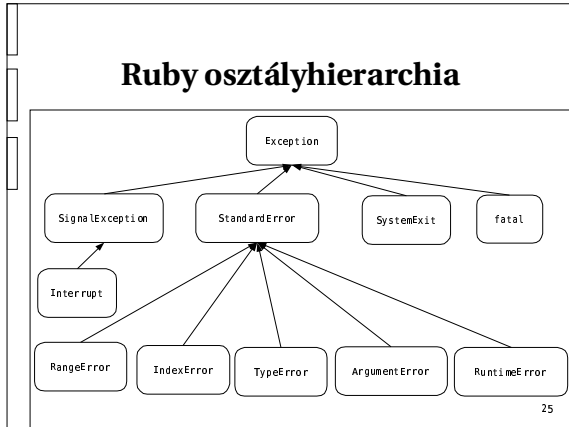
- ```
class MyChildClass < MyParentClass  
  ...  
end
```
- nincs többszörös öröklődés
  - helyette modulok
- attribútumok is öröklődnek
  - elérhetőek a leszármazott osztályokban
- nem szabályozható az öröklődés
  - nincs public, private, stb. öröklődés
- MyChildClass.superclass
  - megadja a szülő osztályt
    - Class
  - csak osztályra működik, a Class osztály metódusa

23

## Ruby osztályhierarchia



24



## super

- super(...)
  - a szülőben lévő metódus meghívása
  - nem csak konstruktorban
  - a hívás helyének megfelelő metódus meghívása
- super
  - ha nem adunk meg paramétert, akkor a megkapott paramétereket adja tovább

27

## Attribútumok

string-holder.rb  
string-holder-attr.rb

- az objektum belső állapotát attribútumok írják le
- Ruby attribútumok neve @ jellel kezdődik
  - az osztályon kívülről nem érhetőek el
- getter metódus kell, ha kívülről lekérdezhetővé akarjuk tenni
  - minden attribútumra külön
- attr\_reader :attribute
  - getter metódus automatikus generálása
  - kompakt, olvasható
- attr\_reader :attr1, :attr2
  - több is felsorolható

28

## Attribútumok

- attribútumok kívülről történő módosításához setter metódust kell definiálni
- def setAttr(newValue)
  - ...
  - end
  - kevésbé konzisztens
- def attr=(newValue)
  - ...
  - end
  - ezután használható:
    - object.attr = newValue
- attr\_writer :attribute
  - a setter metódus automatikus generálása
  - nem definiálja a gettert automatikusan

29

## Attribútumok

- attr :attribute, writeable
  - writeable logikai
  - ha hamis, akkor csak getter definiálódik
  - ha igaz, akkor getter és setter is
- attr\_accessor :attribute
  - attr :attribute, true
  - definiál getter és setter metódust is
- Module osztály privát példánymetódusai!
  - attr\_reader
  - attr\_writer
  - attr
  - attr\_accessor

30

## Hozzáférés szabályozás

- metódusokhoz való hozzáférés szabályozható
  - attribútumok soha nem elérhetőek
    - public, protected, private
      - eltér a szokásostól
- public
  - mindenki által hozzáférhető kívülről is
- protected
  - csak a definiáló osztály illetve a leszármazottak *objektumai* hívhatják
- private
  - csak a definiáló osztály illetve a leszármazottak hívhatják

31

## Hozzáférés szabályozás

- protected
  - csak az osztály metódusain belül hívható
  - lehet receiver-e
- private
  - nem adható meg receiver
  - emiatt csak az adott objektumon belül hívható

32

## Hozzáférés szabályozás

- alapértelmezés szerint public
- class MyClass
  - def method\_this
  - end
  
  - def method\_that
  - end
  - protected :method\_this, :method\_that
  - end- fel kell sorolni a metódusokat szimbólumokkal

33

access-control.rb

## Hozzáférés szabályozás

- class MyClass
  - protected
  - def method\_this
  - end
  
  - def method\_that
  - end
- mindkét metódus protected lesz
  - minden következő metódus definíció hozzáférhetőségének alapértelmezett értékét módosítja
- Module osztály privát példánymetódusai

34

## Nyílt osztálydefiníciók

- Ruby nyelven minden osztály definíciója nyílt
- bármikor hozzáadhatok új szolgáltatásokat
- több definíció tartalma merge-elődik
- class MyClass
  - def method\_1
  - end
  - end
  - ...
  - class MyClass
  - def method\_2
  - end
  - end

35

extend-array.rb  
nil.rb

## Nyílt osztálydefiníciók

- öröklődéssel ez nem lenne teljesen megvalósítható
- már meglévő osztályok módosítása nagy lehetőség
  - példa a Set osztály
  - ha használjuk, akkor módosítja például az Array osztályt
    - hozzáad egy to\_set metódust

36

## Fagyasztás

- freeze metódus
  - Object#freeze
- egy objektum állapota befagyasztható
  - visszavonhatatlan
  - az objektum attribútumai nem módosíthatók
  - ha osztály, akkor a metódusai nem módosíthatók, nem definiálható új, és nem törölhető meglévő
- s = "dog"  
s.freeze  
s[0] = "f"
  - in `[]=': can't modify frozen string (TypeError)
- vizsgálható egy objektum fagyasztott állapota
  - s.frozen?

37

## Alias

- alias newName oldName
  - alias hozható létre
  - paraméterek lehetnek nevek, vagy szimbólumok
    - alias new\_method old\_method
    - alias :new\_method :old\_method
- alias készíthető
  - metódusokra
    - operátorokra
  - globális változókra
  - reguláris kifejezések változóira
    - \$&, \$`, \$', \$+

38

## Alias

- gyakran csak egy funkció több néven nevezéséhez használt
  - Enumerable#map, Enumerable#collect
  - Hash
    - each\_pair visszaad minden elemet (kulcs, érték) párok formájában
    - Enumerable miatt szükséges egy each metódus
    - alias each each\_pair
- Kombinálva azzal, hogy minden osztály definíciója nyílt, nagy lehetőségeket rejt
  - felüldefiniálható a Class.new, alias-val megőrizve az eredeti new-t
  - stb.

39

## Modulok

40

## Modulok

- mit nyújtanak?
  - névtér elkülönítés
  - többszörös öröklődés hiányának enyhítése
- nem osztály
  - nem példányosítható
  - funkcionalitást hordozhat
- metódus definíciókat tartalmaz
  - tartalmazhat hivatkozásokat más metódusokra
  - fordítási időben nem ellenőrzött
    - megj: Ruby-ban máshol sem
  - tartalmazhat osztálydefiníciókat
    - ilyenkor a névtér funkciója érdekes

41

## Modulok

- egy osztályba keverhető a modul tartalma
  - module mixin
- a modul hívhat olyan metódusokat, amiket a modul nem definiál
  - ezek azok a metódusok, amiket a modul vár a kliens osztálytól
  - futási idejű hibát okoz ezek hiánya
- module kulcsszóval definiálható egy modul
- include kulcsszóval lehet egy osztályt "mixelni"
  - ezt egy modul is megteheti!

42

## Modulok

- `module MyModule`
  - `def module_method`
    - `method_in_the_class`
  - `end`
- ...
- `class MyClass`
  - `include MyModule`
  - `def method_in_the_class`
    - `end`
  - `end`

43

## Nyílt moduldefiníciók

- modulok definíciója is nyílt
- hozzáadhatok új metódusokat
  - jelentkezni fog azokban az osztályokban, amelyek használják a modult
  - teljesen dinamikusan

44

## Modul attribútumok

- egy modul kezelhet attribútumokat
  - ezeknek csak egy osztályon belül van értelme!
- `module MyModule`
  - `attr_reader :attr`
  - `def method`
    - `@attr = 1`
  - `end`
- ekkor az osztálynak is lesz `attr` attribútuma, és hozzá egy getter metódus
- nem szerencsés
  - a modul alapvetően funkcionális tartalmazzon, adatokat ne

45

## Ruby standard modulok

- Ruby számos standard modult tartalmaz
- `mixin`-re szánt modulok
  - `Comparable`
  - `Enumerable`
- `funkcionális csoportosító` modulok
  - névtérként funkcionál
  - `FileTest`
    - `File` osztály `include`-olja
  - `GC`
  - `Kernel`
  - `Marshal`
  - `Math`
  - `ObjectSpace`
  - `Process`

46

## Comparable

- függ a `<=>` metódustól
  - az osztály, ami `include`-olja, definiál egy ilyen metódust
- `a <=>` operátor segítségével definiál
  - `<`, `>`, `<=`, `>=`, `==`
  - `between?`
- emlékeztető:
  - Ruby nem definiál `!=` metódust
    - `a != b` mindig `!(a==b)` formára alakul
  - `==` metódus definiálásával a fenti is használható lesz
- használja például:
  - `String`
  - `Numeric`

47

## Enumerable

- szükséges egy `each` metódus
  - iterátor, amely felsorolja az elemeket
- szolgáltatásai
  - `collect`, `map`
  - `detect`, `find`
  - `each_with_index`
  - `select`, `find_all`
  - `grep`
  - `include?`, `member?`
  - `reject`
  - `to_a`, `entries`
- használja például:
  - `Array`
  - `Hash`

48



## Enumerable

- collect, map
  - transzformáció elvégzése a bemeneti adatokon
  - blokkot vár paraméterként
- grep
  - reguláris kifejezés alapján szűr
  - ha van blokk paraméter, akkor minden, a regexpre illeszkedő elemre meghívja azt
- select, find\_all
  - blokkot vár paraméterként
  - ha a blokk igazat ad vissza, akkor az elem megfelel a szűrésnek
- reject
  - !select
- stb.

49

## Enumerable

- ha a kliens osztály definiál <=> operátort, akkor használható továbbá
  - min
  - max
  - sort

50

## Kivételek

51

## Kivételek

- Ruby-ban a kivételek kezelése is OO
- kivételkezelés során objektumokat dobunk
  - mi mást is dobhatnánk?
- begin...end blokk közötti kód által dobott kivételeket figyeljük
- rescue
  - kivételkezelő ágak
- else
  - ha nem történt kivétel
- ensure
  - mindenképp lefut
  - más nyelvekben általában: finally
- nincs kivételkezelési kényszer, mint pl. Java-ban
  - nem specifikálható, hogy mit dobhat

52

## Kivételek

- az elkapott kivétel a \$! változóban található
- begin
  - ...
  - rescue
  - puts "ERROR CAUGHT: #!"
  - end
    - ha nincs megadva semmi, akkor a rescue elkap minden kivételt
- begin
  - ...
  - rescue => e
  - puts "ERROR CAUGHT: #{e}"
  - end
    - elkap mindent, és a kivétel az 'e' változóba kerül

53

## Kivételek

- begin
  - ...
  - rescue RuntimeError
  - puts "ERROR CAUGHT: #!"
  - end
    - RuntimeError és leszármazottait kezeli
- begin
  - ...
  - rescue RuntimeError => e
  - puts "ERROR CAUGHT: #{e}"
  - end
    - RuntimeError és leszármazottai, hiba az e változóban

54

## Kivételek

- begin  
...  
rescue RuntimeError, NameError => e  
...  
end  
- RuntimeError, illetve NameError leszármazottai
- begin  
...  
rescue  
  puts "ERROR"  
else  
  puts "All OK"  
end

55

## Kivételek

- begin  
  file = File.open(filename)  
  ...  
rescue  
  ...  
ensure  
  file.close() unless file.closed?  
end  
- mindenképp végrehajtdik

56

## Kivételek

- retry utasítás  
- Eiffel-ből ismerős  
- a kivételvédtett blokk újratekzdése  
- csak rescue ágban hívható  
- ensure ág csak egyszer hajtódik végre
- begin  
...  
rescue  
...  
  retry  
end

57

## Kivételek

- raise  
- kivételt dobhatunk mi is  
- nem dobható minden  
- Exception-t kell dobni
- raise "Something's wrong"  
- RuntimeError típusú kivétel lesz ebből
- raise ArgumentError, "Parameter must be String"  
- ArgumentError típusú kivétel keletkezik,  
  beállítható hozzá egy üzenet
- raise ArgumentError, "No param", caller  
- stack backtrace is megadható

58

## Reguláris kifejezések

- természetesen ez is teljesen OO
- minden reguláris kifejezés a Regexp osztály egy példánya
- minden mintaillesztés eredménye egy MatchData objektum
- re = /\d{2}/  
- Regexp típusú objektum
- re.match("hello")  
- ha nem illeszkedik, akkor a match visszatérési értéke nil  
- ha illeszkedik, akkor egy MatchData objektum

60

## Reguláris kifejezések

- `/\d+/`
  - az elhatároló módosítható
- `%r{/usr/bin/(.*)}`
  - `%r{}`
  - elhatároló sokféle lehet
- `Regexp.compile`
  - `Regexp.new("/usr/bin/(.*)")`
    - `/\usr\bin\/(.*)/`
    - `Regexp.compile` egy szinoníma
- módosítók
  - `i`: case insensitive
    - `/Hello/i`
    - `/Hello/i.match("hello")`
      - `#<MatchData:0x403108e8>`

61

## Reguláris kifejezések

- `/alma$/`.`match("alma\nfa")`
  - `$` mindenképp illeszkedik a sor végére
  - `^` mindenképp illeszkedik a sor végére
  - Perl-ben ez a szemantika csak az 'm' módosítóval él
- `m`: multiline
  - `.` illeszkedik az újsor karakterre is
  - Perl-ben ezt a funkciót az 's' módosító adja!
  - `/alma.narancs/`.`match("alma\nnarancs")`
    - `nil`
  - `/alma.narancs/m`.`match("alma\nnarancs")`
    - `#<MatchData:0x402d12f0>`

62

## Reguláris kifejezések

- `x`: extended legibility
- `re = /`
  - `(\w+)` # hónap
  - `\`
  - `(\d{2})` # hónap napja
  - `\`
  - `(\d{2})` # óra
  - `:`
  - `(\d{2})` # perc
- `/x`
- `re.match("Sat Apr 10 11:18:06 CEST 2004")`
  - `#<MatchData:0x402e10b0>`

63

substitute-once.rb

## Reguláris kifejezések

- `o`: substitute once
  - a helyettesítéseket csak egyszer végzi el a regexpben
- `/hello #{name}/`
  - minden illesztés alkalmával elvégzi a helyettesítést
  - `name` változó aktuális értékével
- `/hello #{name}/o`
  - csak az első illesztés alkalmával cserél
- olyan alkalom számít, ahol a regexp, mint literál szerepel, és új Regexp objektumot generál

64

## MatchData

- `s = "Sat Apr 10 11:18:06 CEST 2004"`
- `md = /(\w+) (\d{2}) (\d{2}):(\d{2})/`.`match(s)`
- `md[0]`
  - a teljes illesztett sztring
  - `"Apr 10 11:18"`
- `md[i]`, ha `i > 0`
  - az `i`. illesztett csoport
  - `md[1] == "Apr"`
  - `md[2] == "10"`
  - `md[3] == "11"`
  - `md[4] == "18"`

65

## MatchData

- `md.begin(1)`
  - a *sztring* hányadik karakterén kezdődik az 1. számú csoport?
  - 4
- `md.end(1)`
  - a *sztring* hányadik karaktere az, amely az első csoportot követi?
  - 7
- `md.to_a`
  - `md[i]` elemek egy tömbben
  - `["Apr 10 11:18", "Apr", "10", "11", "18"]`

66

## Perl-szerű szintaxis

- a Perl inspirálta szintaxis használható Ruby-ban
  - a regexpek kezelése továbbra is OO marad
  - wrapper réteg az OO réteg fölé
- =~
  - Regexp# =~
  - String# =~
- "Hello world!" =~ /hello/i
  - String# =~
  - visszatérési érték az illeszkedés pozíciója
    - 0
- /hello/i =~ "Hello world!"
  - Regexp# =~
  - visszatérési érték az illeszkedés pozíciója
    - 0

67

## Perl-szerű szintaxis

- mintaillesztés eredménye
  - \$~
    - MatchData
    - utolsó illesztés eredménye
  - \$1, \$2, ..., \$9
    - ezek a \$- változótól függenek
  - \$&
    - match: az illeszkedő sztringrészlet
  - \$', \$'
    - pre-, postmatch
  - \$+
    - az utolsó illesztett csoport

68

## Cserék

- String#sub
  - egyszeri cserék
- String#sub(pattern, replacement)
  - pattern isa Regexp
  - replacement
    - egy sztring, amire cserélni kell
    - a mintában lévő csoportok a \1, \2, ..., \9 hivatkozásokkal érthetők el
  - "hello 21".sub(/(\d+)/, '\1:')
    - "hello :21:"
    - "\1:" mást jelent!
- String#sub!: a fogadót (receiver) módosítja

69

## Cserék

- String#sub(pattern) { |s| ... }
  - a blokk paraméterként kapja az illeszkedő részsstringet
  - a blokk visszatérési értékével helyettesíti az illesztett részsstringet
  - \$1, \$2, ..., \$9 megfelelően be van állítva
- "hello#21".sub(/#(\d+)/) { |s|  
a = "  
a[0] = \$1.hex  
a  
}  
- "hello!"

70

## Cserék

- String#gsub(pattern, replacement)
  - minden előfordulást helyettesít
- String#gsub!(pattern, replacement)
  - minden előfordulást helyettesít és módosítja a fogadó (receiver) objektumot
- String#sub(pattern) { |o| ... }
  - a blokk visszatérési értéke lesz a helyettesített sztring
- String#sub!(pattern) { |o| ... }
  - a blokk visszatérési értéke lesz a helyettesített sztring, és a fogadó (receiver) módosul

71

## Ami kimaradt

- I/O
- catch and throw
- szálak
- RDoc
- Kernel modul metódusai
- osztályokról információ lekérése
  - példánymetódusok listája
  - osztálymetódusok listája
  - konstansok listája
- példányhoz tartozó metódusok (singleton methods)

72

**Itt a vége fuss el véle...**