

Przetwarzanie tekstu 3-4

Ruby

Ruby

Istotne klasy

- ▶ String
- ▶ Regexp
- ▶ Array
- ▶ Hash
- ▶ File
- ▶ IO
- ▶ Enumerable (Mixin)

Ruby

Styl programowania

Ruby jest językiem wyrażeniowym. Każdy poprawny fragment programu zwraca wartość.

Ruby

```
if 0<1 then a="ala" else a=10 end
```



Ruby

```
a = if 0<1 then "ala" else 10 end
```

Ruby

Styl programowania

Ruby ma wiele zapożyczeń z języków funkcyjnych. Pozwalają one zwięźlej formułować kod: np. stosować funkcje wyższego rzędu w miejsce pętli.

Ruby

```
t = ["ala", "ola", "ula"]  
for i in 0...(t.length)  
  puts t[i]  
end
```

Ruby

```
t = ["ala", "ola", "ula"]  
for s in t  
  puts s  
end
```



Ruby

```
t = ["ala", "ola", "ula"]  
t.each{|s| puts s}
```

Ruby

Styl programowania

Ruby ma wiele zapożyczeń z języków funkcyjnych. Pozwalają one zwięźlej formułować kod: np. stosować funkcje wyższego rzędu w miejsce pętli.

Ruby

```
s = "Ala ma 33 koty, 8 psów i 6 rybek."  
  
# ile zwierząt ma Ala?  
n = s.scan(/\d+/).map{|s| s.to_i}.reduce(0){|a,b| a+b}
```

Ruby

String: pojedyncze cytowanie

- ▶ W napisach pojedynczo cytowanych wszelkie sekwencje/wyrażenia specjalne są traktowane dosłownie – nie są ewaluowane.

Ruby

```
n = 10  
puts 'Ala ma #{n} kotów.\nAla ma #{n-2} psów.\nAla ma #{n+4} rybek.'
```

output

```
Ala ma #{n} kotów.\nAla ma #n-2 psów.\nAla ma #{n+4} rybek.
```

Ruby

String: podwójne cytowanie

- ▶ W napisach podwójnie cytowanych wszelkie sekwencje/wyrażenia specjalne są ewaluowane i zastępowane ich wartością.

Ruby

```
n = 10
puts "Ala ma #{n} kotów.\nAla ma #{n-2} psów.\nAla ma #{n+4} rybek."
```

output

```
Ala ma 10 kotów.
Ala ma 8 psów.
Ala ma 6 rybek.
```

Ruby

String: formatowanie

► operator %

Ruby

```
a=123.0; b=456  
puts "%.2f / %.2f = %.2f" % [a,b,a/b]
```

output

```
123.00 / 456.00 = 0.27
```


Ruby

String: łączenie

- ▶ operatory: +, *, metoda join

irb

```
>> 'Ala' + 'ma' + 'kota'  
=> "Alamakota"  
>> 'Ala' + ' ' + 'ma' + ' ' + 'kota'  
=> "Ala ma kota"  
>> ["Ala", "ma", "kota"].join  
=> "Ala ma kota"  
>> ["Ala", "ma", "kota"].join(' ')  
=> "Ala ma kota"  
>> ["Ala", "ma", "kota"] * ' '  
=> "Ala ma kota"
```

Ruby

String: dzielenie

- ▶ operatory: metoda **split**

irb

```
>> "Ala ma kota".split  
=> ["Ala", "ma", "kota"]  
>> "Ala ma kota".split(' ')  
=> ["Ala", "ma", "kota"]  
>> "Ala ma kota".split('')  
=> ["A", "l", "a", " ", "m", "a", " ", "k", "o", "t", "a"]  
>> "Ala ma kota".split('o')  
=> ["Ala ma k", "ta"]
```

Ruby

String: dopisywanie

► operator <<

irb

```
>> a = 'Ala'  
=> "Ala"  
>> a << ' ma'  
=> "Ala ma"  
>> a << ' kota'  
=> "Ala ma kota"
```

Wartość po lewej stronie operatora << jest modyfikowana, nie powstaje nowa wartość typu String.

Ruby

String: czyszczenie nadmiarowych odstępów

► metody `squeeze`, `strip`

irb

```
>> "      Ala      ma      kota      ".squeeze  
=> " Ala ma kota "  
>> "      Ala      ma      kota      ".strip  
=> "Ala      ma      kota"  
>> "      Ala      ma      kota      ".squeeze.strip  
=> "Ala ma kota"
```

Ruby

Regexp



`https://www.tutorialspoint.com/ruby/ruby_regular_expressions.htm`

- ▶ dokumentacja klasy Regexp

Ruby

Regexp: operator dopasowania

- ▶ operator `=~`
- ▶ wartością jest pozycja początku dopasowania lub `nil`

irb

```
>> /a+b+/ =~ 'acbaabc'  
=> 3  
>> /a+b+/ =~ 'acbaadc'  
=> nil
```

Ruby

Regexp: zapamiętywanie podnapisów i referencje wsteczne

- ▶ operator `(...)` wewnątrz wyrażenia regularnego zapamiętuje część napisu dopasowaną do wzorca w nawiasie
- ▶ odwołanie wewnątrz wyrażenia: `\1`, `\2`, `\3`, etc.
- ▶ odwołanie poza wyrażeniem: `$1`, `$2`, `$3`, etc.

irb

```
>> /(a+) (b+) \1/ =~ 'abcaaabaac'  
=> 4  
>> $1  
=> "aa"  
>> $2  
=> "b"
```

Ruby

zastępowanie podnapisów

► metoda **sub**, **gsub**

irb

```
>> "abba".sub('a','A')  
=> "Abba"  
>> "abba".gsub('a','A')  
=> "AbbA"
```

irb

```
>> "abba".sub('a','A')  
=> "Abba"  
>> "abba".gsub('a','A')  
=> "AbbA"
```


Ruby

zastępowanie podnapisów

irb

```
>> s = "Ala ma kota i psa"
>> s.sub(/^(?:\w+ )+(\w+ )+ i (\w+)/, '\1\2 i \1\3')
=> "Ala ma kota i Ala ma psa"
```

irb

```
>> s = "Ala ma 2 koty, 2 psy i 8 ryb"
>> s.gsub(/\d+/){|n| {'2'=>'dwa', '8'=>'osiem'}[n] }
=> "Ala ma dwa koty, dwa psy i osiem ryb"
```

irb

```
>> cap = /\b[[:upper:]][[:lower:]]+\b/
=> /\b[[:upper:]][[:lower:]]+\b/
>> "Jan Kowal i Adam Nowak".gsub(/({cap}) (?=#{cap})/){|i| i[0]+'.'}
=> "J.Kowalski i A.Nowak"
```

Ruby

selekcja napisów

irb

```
>> ["ala", "ola", "ela", "ula"].select{|s| ["ala", "ola"].member? s }  
=> ["ala", "ola"]  
>> ["ala", "ola", "ela", "ula"].reject{|s| ["ala", "ola"].member? s }  
=> ["ela", "ula"]
```

Ruby

czytanie i zapisywanie do pliku

► czytanie

- `read` - czytanie całości do stringa
- `readlines` - czytanie całości do tablicy linii
- `gets` - czytanie jednej linii

Metody te, użyte bez obiektu określającego plik, czytają ze standardowego wejścia o ile tablica ARGV jest pusta, jeśli nie - kolejne elementy tablicy ARGV są traktowane jako nazwy plików z których program ma kolejno czytać

► zapisywanie

- `puts` - wypisanie linii
- `print` - wypisanie argumentu w bieżącej linii

Metody te wypisują na standardowe wejście.

Tem mechanizm pozwala łatwo pisać programy zachowujące się jak typowe filtry.

Ruby

typowy szkieleł programu

```
#!/usr/bin/env ruby
require 'getoptlong'
opts = GetoptLong.new(
  [ '--help',      '-h',    GetoptLong::NO_ARGUMENT ],
  [ '--graph',     '-g',    GetoptLong::REQUIRED_ARGUMENT ],
  [ '--tree',      '-s',    GetoptLong::REQUIRED_ARGUMENT ])

HELPMESSAGE = <<END
Command:      comptree [options]
Options:
--help        -h      Print help (this text) and exit.
...
END
$graph=nil; $tree=nil
opts.each do |opt, arg|
  case opt
  when '--help' then print HELPMESSAGE; exit 0
  when '--graph' then $graph = arg
  when '--tree' then $tree = arg
  else print "Unknown option. Ignored."
  end
end

def f1 ... ; ... end
def f2 ... ; ... end
...
def main ... ; ... end

main
```

Ruby

iteracja linia po linii

Ruby

```
słowa = {}  
słowa.default = 0  
while l=gets  
  l.split.each{|w| słowa[w] += 1}  
end  
puts słowa.inspect
```

wejście

```
ala  
ola  
ela  
ala
```

wyjście

```
{"ala"=>2, "ola"=>1, "ela"=>1}
```

Ruby

