

در روبی if دستور



دستور if در روبی



در زبان برنامه‌نویسی روبی، دستور `if` برای کنترل جریان برنامه و اجرای کد بر اساس یک شرط استفاده می‌شود. ساختار کلی این دستور به شکل زیر است:

ruby

```

شرط if
# کدی که در صورت درست بودن شرط اجرا می‌شود
شرط دیگر elsif
# کدی که در صورت درست بودن شرط دوم اجرا می‌شود
else
# کدی که در صورت نادرست بودن تمام شرایط اجرا می‌شود
end
  
```

مثال ساده:

ruby

```

x = 10

if x > 5
  puts "بزرگ‌تر از 5 است x"
elsif x == 5
  puts "برابر با 5 است x"
else
  puts "کوچک‌تر از 5 است x"
end
  
```

خروجی:

بزرگ‌تر از 5 است x

دستورات کوتاه‌تر (Inline If):

می‌توانید شرط `if` را در یک خط بنویسید:

ruby

```
puts "بزرگتر از 5 است x" if x > 5
```

استفاده از Unless:

برای شرایطی که می‌خواهید وقتی شرط نادرست است کدی اجرا شود، از `unless` استفاده می‌شود:

ruby

```
puts "کوچکتر یا برابر با 5 است x" unless x > 5
```

دستور Ternary (سه‌گانه):

برای ساده کردن شرط‌های `if-else` کوتاه، از دستور سه‌گانه استفاده می‌شود:

ruby

```
result = x > 5 ? "بزرگتر از 5" : "کوچکتر یا برابر با 5"
puts result
```

خروجی:

```
بزرگتر از 5
```

نکات:

- بلوک‌های کد در روبی با **indentation** مشخص نمی‌شوند، بلکه از کلمات کلیدی مانند `end` برای بستن بلوک استفاده می‌شود.
- روبی مقدار `nil` و `false` را نادرست می‌داند و بقیه مقادیر را درست.



دستور unless در روبی



دستور `unless` در روبی شبیه به دستور `if` است، با این تفاوت که زمانی کد را اجرا می‌کند که شرط نادرست باشد. به بیان دیگر، `unless` به معنای "اگر نه" است.

ساختار کلی:

ruby

```

unless شرط
  # کدی که در صورت نادرست بودن شرط اجرا میشود
else
  # کدی که در صورت درست بودن شرط اجرا میشود
end

```

مثال ساده:

ruby

```

x = 10

unless x > 15
  puts "کوچکتر یا برابر با 15 است x"
else
  puts "بزرگتر از 15 است x"
end

```

خروجی:

```

کوچکتر یا برابر با 15 است x

```

استفاده از `unless` به صورت یک خطی:

برای شرایط ساده، می‌توانید از فرم تکخطی استفاده کنید:

ruby

```

puts "کوچکتر یا برابر با 15 است x" unless x > 15

```

استفاده از `unless` با مقداردهی:

ruby

```

x = 10
message = "کوچکتر یا برابر با 15 است x" unless x > 15
puts message

```

نکات مهم درباره `unless`:

- دستور `unless` برای شرایط منفی بسیار خوانا است، اما اگر شرط پیچیده شود، بهتر است از `if` با منطق معکوس (مثل `if ! (شرط)`) استفاده کنید تا کد خواناتر شود.

ruby

```
unless x > 15
  puts "کوچکتر یا برابر با 15 است x"
end
```

معادل:

ruby

```
if !(x > 15)
  puts "کوچکتر یا برابر با 15 است x"
end
```

2. از `unless` در ترکیب با `else` استفاده نکنید مگر اینکه واقعا نیاز باشد، زیرا ممکن است کد گیج‌کننده شود. به جای آن، از `if` استفاده کنید.

ترکیب `unless` با روش‌های کاربردی:

گاهی می‌توانید `unless` را با متدها ترکیب کنید تا کد مختصر شود:

ruby

```
x = nil
puts "مقداری تعیین نشده است" unless x
```

این دستور زمانی اجرا می‌شود که `x` مقدار `nil` یا `false` داشته باشد.

جمع‌بندی:

- `unless` وقتی مناسب است که بخواهید کدی بنویسید که تنها در صورت نادرست بودن شرط اجرا شود.
- از فرم تک‌خطی برای شرایط ساده استفاده کنید.
- در شرایط پیچیده از `if` همراه با منطق منفی استفاده کنید تا خوانایی افزایش یابد.

دستور `while` در روبی

در زبان برنامه‌نویسی روبی، دستور `while` برای اجرای یک بلوک کد تا زمانی که شرط مشخصی درست (**true**) باشد، استفاده می‌شود. این دستور یکی از روش‌های متداول برای ایجاد حلقه‌ها (loops) است.

ساختار کلی:

ruby

```

while شرط
  # کدی که در صورت درست بودن شرط اجرا میشود
end

```

مثال ساده:

ruby

```

i = 0
while i < 5
  puts "عدد i برابر است با #{i}"
  i += 1
end

```

خروجی:

CSS

```

عدد i 0 برابر است با
عدد i 1 برابر است با
عدد i 2 برابر است با
عدد i 3 برابر است با
عدد i 4 برابر است با

```

حالت تک خطی:

در روبی می‌توانید حلقه `while` را در یک خط بنویسید:

ruby

```

i = 0
puts i += 1 while i < 5

```

خروجی:

```

1
2
3
4
5

```

حلقه بی‌نهایت با `while`:

اگر شرط همیشه درست باشد، حلقه بی‌نهایت اجرا می‌شود. برای جلوگیری از قفل شدن برنامه، می‌توانید با دستور `break` از حلقه خارج شوید:

ruby

```
i = 0

while true
  puts "i: #{i}"
  i += 1
  break if i >= 5
end
```

خروجی:

makefile

```
i: 0
i: 1
i: 2
i: 3
i: 4
```

استفاده از `until` به جای `while`:

در روبی می‌توانید از `until` استفاده کنید، که برعکس `while` عمل می‌کند و تا زمانی که شرط نادرست (**false**) است، حلقه را اجرا می‌کند:

ruby

```
i = 0
puts i += 1 until i >= 5
```

دستورات کنترلی در `while`:

- `break`: برای خروج از حلقه.
- `next`: برای رفتن به اجرای بعدی حلقه، بدون ادامه اجرای باقی‌مانده کدها.
- `redo`: برای اجرای مجدد بلوک بدون ارزیابی دوباره شرط.

مثال:

ruby

```
i = 0

while i < 5
  i += 1
  next if i == 3 # حلقه از عدد 3 عبور می‌کند
  puts i
end
```

خروجی:

```
1
2
4
5
```

نکات مهم:

1. همیشه مراقب باشید که شرط در حلقه `while` به درستی تغییر کند تا از ایجاد حلقه بی‌نهایت جلوگیری شود.
2. برای خوانایی بهتر کد، از فرم تکخطی فقط در شرایط ساده استفاده کنید.
3. از روش‌های مدرن‌تر مثل `each` یا `for` در صورت امکان برای کار با لیست‌ها و شمارنده‌ها استفاده کنید.



در روبی for



در زبان برنامه‌نویسی روبی، حلقه `for` برای تکرار روی عناصر یک مجموعه (مانند آرایه، محدوده `range`)، یا هاش) استفاده می‌شود. این نوع حلقه در روبی نسبت به متدهای مدرن مانند `each` کمتر استفاده می‌شود، اما همچنان در مواردی کاربرد دارد.

ساختار کلی:

ruby

```
for مجموعه in متغیر
  # کدی که باید در هر تکرار اجرا شود
end
```

مثال با محدوده (Range):

ruby

```
for i in 1..5
  puts "عدد: #{i}"
end
```

خروجی:

makefile

```
1 : عدد
2 : عدد
3 : عدد
4 : عدد
5 : عدد
```

مثال با آرایه:

ruby

```
names = ["علی", "سارا", "حسین"]

for name in names
  puts "سلام، #{name}!"
end
```

خروجی:

```
!سلام، علی
!سلام، سارا
!سلام، حسین
```

استفاده با هاش (Hash):

ruby

```
student_scores = { علی: 90, سارا: 85, حسین: 95 }

for name, score in student_scores
  puts "#{name} امتیاز: #{score}"
end
```

خروجی:

```
90 امتیاز: علی
85 امتیاز: سارا
95 امتیاز: حسین
```

حالت‌های کنترلی:

در حلقه `for`، می‌توانید از دستورات کنترلی استفاده کنید:

- `break``: برای خروج از حلقه.
- `next``: برای رد کردن اجرای فعلی و رفتن به تکرار بعدی.
- `redo``: برای اجرای مجدد همان تکرار.

مثال:

ruby

```
for i in 1..5
  next if i == 3 # از عدد 3 عبور می‌کند
  break if i == 5 # حلقه را در عدد 5 متوقف می‌کند
  puts i
end
```

خروجی:

```
1
2
4
```

مقایسه با `each`:

در روبی، متد `each` معمولاً جایگزین `for` می‌شود زیرا بیشتر با سبک برنامه‌نویسی روبی سازگار است و بلوک‌ها را بهتر مدیریت می‌کند.

مثال معادل با `each`:

ruby

```
(1..5).each do |i|
  puts "عدد: #{i}"
end
```

چرا `each` بهتر است؟

- ایزوله‌تر عمل می‌کند و به متغیرهای خارج از بلوک تأثیر نمی‌گذارد.
- قابلیت انعطاف بیشتری برای ترکیب با سایر متدها دارد.

نکات:

1. استفاده از `for` همچنان ساده و خواناست و در مواردی که به ایزوله بودن متغیر نیازی نیست، قابل استفاده است.
2. برای پروژه‌های مدرن، `each` و دیگر متدهای پیمایشی (مانند `map` و `select`) ترجیح داده می‌شوند.



12 مثال فوق العاده ساده در روب



در اینجا 12 مثال بسیار ساده و قابل فهم از کدهای روبی آورده شده است که برای یادگیری اولیه مناسب هستند:

1. چاپ یک پیام ساده:

ruby

```
puts "سلام دنیا"
```

خروجی:

```
سلام دنیا
```

2. جمع دو عدد:

ruby

```
a = 5  
b = 10  
puts a + b
```

خروجی:

```
15
```

3. چک کردن شرط با `if`:

ruby

```
x = 7  
  
if x > 5  
  puts "بزرگتر از 5 است x"  
end
```

خروجی:

```
بزرگتر از 5 است x
```

4. حلقه با `while`:

ruby

```
i = 0

while i < 3
  puts "عدد: #{i}"
  i += 1
end
```

خروجی:

makefile

```
عدد: 0
عدد: 1
عدد: 2
```

5. حلقه با `for`:

ruby

```
for i in 1..3
  puts "تکرار #{i}"
end
```

خروجی:

```
تکرار 1
تکرار 2
تکرار 3
```

6. تعریف و استفاده از متد:

ruby

```
def greet(name)
  puts "سلام، #{name}!"
end

greet("سارا")
```

خروجی:

```
!سلام، سارا
```

7. کار با آرایه:

ruby

```
numbers = [1, 2, 3, 4, 5]

numbers.each do |num|
  puts "عدد: #{num}"
end
```

خروجی:

makefile

```
1 : عدد
2 : عدد
3 : عدد
4 : عدد
5 : عدد
```

8. کار با هش (Hash):

ruby

```
person = { name: "علی", age: 25 }

puts "نام: #{person[:name]}"
puts "سن: #{person[:age]}"
```

خروجی:

makefile

```
نام: علی
سن: 25
```

9. خواندن ورودی از کاربر:

ruby

```
puts "نام شما چیست؟"
name = gets.chomp
puts "سلام، #{name}!"
```

ورودی:

سارا

خروجی:

سلام، سارا

10. شرط با `unless`:

ruby

```
x = 10
puts "کوچکتر از 5 نیست x" unless x < 5
```

خروجی:

کوچکتر از 5 نیست x

11. عملگر سه‌گانه (Ternary):

ruby

```
x = 8
message = x > 5 ? "کوچکتر یا برابر با 5" : "بزرگتر از 5"
puts message
```

خروجی:

بزرگتر از 5

12. تعریف کلاس ساده:

ruby

```
class Person
  def initialize(name)
    @name = name
  end
end
```

```
def say_hello
  puts "هستم #{@name} سلام، من"
end

person = Person.new("علی")
person.say_hello
```

خروجی:

سلام، من علی هستم

این مثال‌ها پایه و اساس زبان روبی را به سادگی و جذابیت نمایش می‌دهند و برای شروع یادگیری مناسب هستند.



مثال‌های ساده برای شروع روبی را تا 44 مثال افزایش بده



در ادامه، 44 مثال ساده و مفید برای شروع یادگیری زبان برنامه‌نویسی روبی آورده شده است که به تدریج شما را با مفاهیم این زبان آشنا می‌کند:

1-12: مثال‌های ساده اولیه

(این‌ها همان مثال‌های قبلی هستند. برای دیدن جزئیات به پیام قبلی مراجعه کنید.)

13. جمع اعداد در آرایه:

ruby

```
numbers = [1, 2, 3, 4, 5]
sum = numbers.sum
puts "مجموع: #{sum}"
```

خروجی:

makefile

مجموع: 15

14. بررسی عدد زوج یا فرد:

ruby

```
num = 7

if num.even?
  puts "#{num} است زوج"
else
  puts "#{num} است فرد"
end
```

15. ایجاد یک محدوده (Range):

ruby

```
range = (1..5).to_a
puts range.inspect
```

خروجی:

csharp

```
[1, 2, 3, 4, 5]
```

16. چاپ مقادیر یک آرایه معکوس:

ruby

```
arr = [1, 2, 3, 4, 5]
puts arr.reverse.inspect
```

خروجی:

csharp

```
[5, 4, 3, 2, 1]
```

17. فاکتوریل یک عدد:

ruby

```
def factorial(n)
  (1..n).inject(:*) || 1
end
```

```
puts factorial(5) # خروجی: 120
```

18. تبدیل رشته به حروف کوچک و بزرگ:

ruby

```
text = "Ruby Programming"
puts text.upcase
puts text.downcase
```

19. چک کردن مقدار `nil`:

ruby

```
x = nil
puts x.nil? ? "مقدار نال نیست" : "مقدار نال است"
```

20. کار با تاریخ و زمان:

ruby

```
require 'date'

today = Date.today
puts "تاریخ امروز: #{today}"
```

21. پیدا کردن طول یک آرایه:

ruby

```
arr = [1, 2, 3, 4, 5]
puts "تعداد عناصر: #{arr.length}"
```

22. ادغام دو آرایه:

ruby

```
arr1 = [1, 2, 3]
arr2 = [4, 5, 6]
merged = arr1 + arr2
puts merged.inspect
```


ruby

```
text = "Hello, Ruby!"  
puts text[7, 4] # "Ruby"
```

24. حذف فضاهای خالی اضافی:

ruby

```
text = "  Ruby  "  
puts text.strip
```

25. تکرار رشته:

ruby

```
puts "Ruby " * 3
```

26. مقادیر پیش فرض در متدها:

ruby

```
def greet(name = "مهمان")  
  puts "سلام، #{name}!"  
end  
  
greet  
greet("علی")
```

27. تقسیم یک رشته به آرایه:

ruby

```
text = "Ruby is fun"  
puts text.split.inspect
```

28. تبدیل آرایه به رشته:

ruby

```
arr = ["Ruby", "is", "fun"]  
puts arr.join(" ")
```

29. حذف مقدار از آرایه:

ruby

```
arr = [1, 2, 3, 4, 5]  
arr.delete(3)  
puts arr.inspect
```

30. مقایسه اعداد:

ruby

```
a = 10  
b = 20  
  
puts a > b ? "a بزرگتر است" : "b بزرگتر است"
```

31. تبدیل عدد به رشته:

ruby

```
num = 123  
puts num.to_s
```

32. تبدیل رشته به عدد:

ruby

```
str = "456"  
puts str.to_i
```

33. تعریف یک ثابت:

ruby

```
PI = 3.14
```

```
puts "#{PI} عدد پی"
```

34. ایجاد یک متد بازگشتی (Recursive):

ruby

```
def countdown(n)
  return if n < 0
  puts n
  countdown(n - 1)
end

countdown(5)
```

35. پیدا کردن بزرگترین عدد در آرایه:

ruby

```
arr = [3, 7, 2, 8, 1]
puts arr.max
```

36. پیدا کردن کوچکترین عدد در آرایه:

ruby

```
arr = [3, 7, 2, 8, 1]
puts arr.min
```

37. چک کردن وجود مقدار در آرایه:

ruby

```
arr = [1, 2, 3, 4, 5]
puts arr.include?(3) # خروجی: true
```

38. مرتب‌سازی آرایه:

ruby

```
arr = [5, 3, 8, 1, 2]
puts arr.sort.inspect
```

39. تبدیل آرایه به مجموعه (Set):

ruby

```
require 'set'

arr = [1, 2, 3, 2, 1]
unique_set = arr.to_set
puts unique_set.inspect
```

40. تکرار روی کلیدها و مقادیر هاش:

ruby

```
hash = { a: 1, b: 2, c: 3 }

hash.each do |key, value|
  puts "#{key}: #{value}"
end
```

41. تبدیل کلیدهای هاش به آرایه:

ruby

```
hash = { a: 1, b: 2, c: 3 }
puts hash.keys.inspect
```

42. تبدیل مقادیر هاش به آرایه:

ruby

```
hash = { a: 1, b: 2, c: 3 }
puts hash.values.inspect
```

43. چاپ اعداد فرد در محدوده:

ruby

```
(1..10).each do |num|
  puts num if num.odd?
end
```

44. استفاده از متد `map` برای تغییر عناصر:

```
arr = [1, 2, 3, 4, 5]
squared = arr.map { |num| num**2 }
puts squared.inspect
```

این 44 مثال به تدریج شما را با مفاهیم اولیه روبی، از ساده‌ترین موارد تا مفاهیم کمی پیشرفته‌تر، آشنا می‌کند.



چکار می‌کند `.inspect`



در زبان روبی، متد `inspect` یک نمایش متنی از یک شیء را ارائه می‌دهد که برای اشکال‌زدایی (**debugging**) و مشاهده ساختار داخلی شیء مفید است. این متد معمولاً اطلاعات بیشتری نسبت به متد `to_s` (که برای تبدیل به رشته استفاده می‌شود) ارائه می‌دهد و بیشتر برای توسعه‌دهندگان طراحی شده است.

ویژگی‌های کلیدی `inspect`:

1. ساختار داخلی شیء را نشان می‌دهد:

○ برای آرایه‌ها، هش‌ها، رشته‌ها و دیگر ساختارهای داده، `inspect` ساختار دقیق و قابل درک را نمایش می‌دهد.

2. مناسب برای اشکال‌زدایی:

○ این متد برای زمانی که می‌خواهید داده‌ها را بررسی کنید، مناسب است.

3. تفاوت با `to_s`:

○ `to_s` خروجی قابل خواندن برای کاربران فراهم می‌کند، در حالی که `inspect` برای توسعه‌دهندگان طراحی شده است.

مثال‌ها:

1. روی یک آرایه:

```
arr = [1, 2, 3, 4]
puts arr.inspect
```

خروجی:

```
[1, 2, 3, 4]
```

2. روی یک هش:

ruby

```
hash = { name: "علی", age: 25 }
puts hash.inspect
```

خروجی:

ruby

```
{:name=>"علی", :age=>25}
```

3. تفاوت `inspect` و `to_s`:

ruby

```
str = "دنیا\nسلام"
puts str.to_s # خروجی کاربر پسند
puts str.inspect # خروجی دقیق برای اشکال زدایی
```

خروجی:

swift

```
سلام
دنیا
"دنیا\nسلام"
```

- `to_s` فقط متن را چاپ می‌کند.
- `inspect` کاراکترهای کنترلی (مثل `\n`) و علائم اضافی مثل نقل قول‌ها را نیز نشان می‌دهد.

4. روی اشیاء سفارشی:

اگر کلاس سفارشی تعریف کنید، می‌توانید `inspect` را بازنویسی کنید تا اطلاعات دلخواهی نمایش دهد:

ruby

```
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def inspect
    "شخص: #{@name}, سن: #{@age}"
  end
end
```

```
person = Person.new("25", "علی")
puts person.inspect
```

خروجی:

makefile

```
شخص: علی, سن: 25
```

نکته:

- متد `inspect` بیشتر برای نمایش داده‌های اشیاء در حالت خام یا دقیق استفاده می‌شود.
- در بسیاری از موارد به‌طور خودکار در ابزارهایی مثل کنسول روبی (`irb`) استفاده می‌شود.



colorize a simple text in ruby



برای رنگ‌آمیزی متن در روبی می‌توانید از جواهر (gem) `colorize` استفاده کنید. این کتابخانه به شما امکان می‌دهد متون را به رنگ‌های مختلف نمایش دهید و همچنین برخی از استایل‌ها مانند پررنگ (**bold**) یا زیرخط (underline) را اعمال کنید.

نصب `colorize`:

1. ابتدا gem را نصب کنید:

bash

```
gem install colorize
```

2. سپس در فایل روبی خود آن را وارد کنید:

ruby

```
require 'colorize'
```

یک مثال ساده:

ruby

```
require 'colorize'

puts "این متن قرمز است".colorize(:red)
```

```
puts "این متن سبز است".colorize(:green)
puts "این متن آبی است".colorize(:blue)
```

خروجی (در ترمینال): متن‌ها به رنگ‌های مشخص شده نمایش داده می‌شوند.

تغییر پس‌زمینه:

برای تغییر رنگ پس‌زمینه، می‌توانید از گزینه‌های پس‌زمینه استفاده کنید:

ruby

```
require 'colorize'

puts "متن با پس‌زمینه زرد".colorize(:background => :yellow)
puts "متن قرمز با پس‌زمینه آبی".colorize(:color => :red, :background => :blue)
```

افزودن استایل:

`colorize` از استایل‌هایی مانند **bold** و underline نیز پشتیبانی می‌کند:

ruby

```
require 'colorize'

puts "این متن پررنگ است".colorize(:red).bold
puts "این متن زیرخط دارد".underline
```

لیست رنگ‌ها:

این کتابخانه رنگ‌های زیر را پشتیبانی می‌کند:

- متن: `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `cyan`, `white`
- پس‌زمینه: مشابه متن‌ها، اما به صورت `background => color`

نکته:

- برخی از ترمینال‌ها ممکن است نیاز به پشتیبانی از ANSI داشته باشند تا رنگ‌ها به درستی نمایش داده شوند.
- برای آزمایش سریع، از کنسول روبی (IRB) یا اسکریپت در ترمینال استفاده کنید.

این ابزار ساده و کارآمد برای زیباتر کردن خروجی‌های ترمینال در پروژه‌های کوچک و آموزشی است!



برای کلاس درس کارگاه می‌تونی 14 مثال از روبی بدی و 7 نکته و 6 ابزار و ... به بچه‌ها بگم



14 مثال کاربردی روبی برای کلاس کارگاه:

1. چاپ ساده متن:

```
ruby

puts "سلام به کلاس کارگاه"
```

2. جمع دو عدد:

```
ruby

num1 = 5
num2 = 10
puts "مجموع: #{num1 + num2}"
```

3. حلقه `for`:

```
ruby

for i in 1..5
  puts "شماره: #{i}"
end
```

4. استفاده از `if` و شرط:

```
ruby

grade = 18
if grade >= 10
  puts "قبولی!"
else
  puts "مردودی!"
end
```

5. تعریف متد ساده:

```
ruby

def greet(name)
  puts "سلام، #{name}!"
end
greet("علی")
```

6. کار با آرایه‌ها:

```
ruby

students = ["علی", "سارا", "زهرا"]
```

```
students.each { |student| puts student }
```

7. کار با هش (Hash):

```
ruby
```

```
grades = { "علی" => 18, "سارا" => 20, "زهرا" => 15 }
grades.each { |name, grade| puts "#{name}: #{grade}" }
```

8. جمع مقادیر آرایه:

```
ruby
```

```
numbers = [1, 2, 3, 4, 5]
puts "مجموع: #{numbers.sum}"
```

9. حذف مقدار از آرایه:

```
ruby
```

```
arr = [1, 2, 3, 4, 5]
arr.delete(3)
puts arr.inspect
```

10. تبدیل رشته به حروف کوچک و بزرگ:

```
ruby
```

```
text = "Ruby Is Fun"
puts text.upcase
puts text.downcase
```

11. حذف فضاهاى اضافی:

```
ruby
```

```
text = " !سلام رویی "
puts text.strip
```

12. شرط با `unless`:

```
ruby
```

```
x = 8
puts "کوچکتر از 5 نیست" unless x < 5
```

13. تعریف کلاس:

ruby

```
class Student
  def initialize(name, grade)
    @name = name
    @grade = grade
  end

  def info
    puts "#{@name} : نمره #{@grade}"
  end
end

student = Student.new("علی", "18")
student.info
```

14. کار با تاریخ:

ruby

```
require 'date'
puts "امروز: #{Date.today}"
```

7 نکته مهم برای یادگیری روبی:

1. سادگی روبی را جدی بگیرید: روبی زبانی ساده و روان است، با استفاده از امکانات زیاد آن کدهای خود را بهینه بنویسید.
2. از `irb` برای آزمایش سریع کدها استفاده کنید: در کنسول روبی (`irb`)، می‌توانید خطوط کد خود را سریع تست کنید.
3. استفاده از `puts` و `inspect`: متد `puts` برای چاپ خروجی مناسب کاربران است، اما `inspect` برای مشاهده ساختار داخلی اشیاء و اشکال‌زدایی استفاده می‌شود.
4. آرایه‌ها و هش‌ها را به خوبی درک کنید: بسیاری از داده‌ها در روبی در قالب آرایه یا هش مدیریت می‌شوند.
5. متدها را ساده تعریف کنید: متدها بهترین ابزار برای سازمان‌دهی کدهای شما هستند. متدهای کوچک و قابل خواندن بنویسید.
6. از کتابخانه‌ها (Gem) استفاده کنید: روبی دارای کتابخانه‌های بسیار زیادی برای انجام کارهای پیچیده است.
7. تکرار، کلید یادگیری است: برای یادگیری سریع‌تر روبی، کدهای مختلف بنویسید و خطاهای خود را برطرف کنید.

6 ابزار کاربردی برای کار با روبی:

1. `irb`: کنسول تعاملی روبی برای تست سریع کدها.

bash

irb

2. `pry`: ابزاری قوی‌تر از `irb` برای اشکال‌زدایی و تست.

bash

gem install pry

3. `rubocop``: برای بررسی و بهبود سبک کدنویسی رویی.

bash

gem install rubocop

4. `rspec``: برای تست خودکار کدهای رویی.

bash

gem install rspec

5. `rails``: یک فریم‌ورک قدرتمند برای توسعه وب.

bash

gem install rails

6. `bundler``: برای مدیریت کتابخانه‌های پروژه.

bash

gem install bundler

5 تمرین پیشنهادی برای دانشجویان:

1. کد چاپ عدد فرد و زوج: برنامه‌ای بنویسید که اعداد فرد و زوج را در محدوده‌ای مشخص چاپ کند.
2. مدیریت آرایه: آرایه‌ای از نام دانشجویان بسازید و آن را به ترتیب حروف الفبا مرتب کنید.
3. محاسبه میانگین: برنامه‌ای بنویسید که میانگین نمرات دانشجویان را محاسبه کند.
4. ساخت کلاس دانشجو: کلاس ساده‌ای برای ذخیره اطلاعات دانشجویان بنویسید و اطلاعات آن‌ها را چاپ کنید.
5. کار با تاریخ: برنامه‌ای بنویسید که تعداد روزهای باقی‌مانده تا پایان سال را نشان دهد.

3 پروژه کوچک برای انجام در کلاس:

1. دفترچه نمرات: برنامه‌ای بنویسید که اطلاعات دانشجویان و نمرات آن‌ها را در یک هش ذخیره کند و امکانات زیر را ارائه دهد:
 - اضافه کردن دانشجو
 - جستجوی نمره یک دانشجو
 - نمایش همه دانشجویان

2. مدیریت کتابخانه: برنامه‌ای ساده برای مدیریت کتاب‌های یک کتابخانه:

- ذخیره نام کتاب‌ها
- جستجوی کتاب
- حذف کتاب

3. ماشین حساب: ساخت یک ماشین حساب ساده با عملیات جمع، تفریق، ضرب و تقسیم.

این موارد می‌توانند کلاس شما را به‌صورت جذاب و تعاملی پیش ببرند و دانشجویان را درگیر یادگیری کنند!



یک بار دیگه آخرین متن به انگلیسی هم مینویسی



14 Practical Ruby Examples for a Workshop Class:

1. Simple text output:

```
ruby

puts "Welcome to the workshop class!"
```

2. Sum of two numbers:

```
ruby

num1 = 5
num2 = 10
puts "Sum: #{num1 + num2}"
```

3. Using `for` loop:

```
ruby

for i in 1..5
  puts "Number: #{i}"
end
```

4. `if` condition example:

```
ruby

grade = 18
if grade >= 10
  puts "Pass!"
else
  puts "Fail!"
end
```

5. Defining a simple method:

```
ruby

def greet(name)
  puts "Hello, #{name}!"
end
greet("Ali")
```

6. Working with arrays:

```
ruby

students = ["Ali", "Sara", "Zahra"]
students.each { |student| puts student }
```

7. Using hashes:

```
ruby

grades = { "Ali" => 18, "Sara" => 20, "Zahra" => 15 }
grades.each { |name, grade| puts "#{name}: #{grade}" }
```

8. Sum of array values:

```
ruby

numbers = [1, 2, 3, 4, 5]
puts "Sum: #{numbers.sum}"
```

9. Deleting an element from an array:

```
ruby

arr = [1, 2, 3, 4, 5]
arr.delete(3)
puts arr.inspect
```

10. String case transformation:

```
ruby

text = "Ruby Is Fun"
puts text.upcase
puts text.downcase
```

11. Removing extra spaces:

```
ruby

text = " Hello Ruby! "
puts text.strip
```

12. `unless` condition:

```
ruby

x = 8
puts "Not less than 5" unless x < 5
```

13. Defining a class:

```
ruby

class Student
  def initialize(name, grade)
    @name = name
    @grade = grade
  end

  def info
    puts "#{@name}, Grade: #{@grade}"
  end
end

student = Student.new("Ali", 18)
student.info
```

14. Working with dates:

```
ruby

require 'date'
puts "Today: #{Date.today}"
```

7 Key Tips for Learning Ruby:

1. Leverage Ruby's simplicity:

Ruby is easy and intuitive. Use its features to write concise and clean code.

2. Use `irb` for quick testing:

Ruby's interactive shell (`irb`) is great for trying out code snippets.

3. Understand `puts` vs `inspect`:

`puts` is for user-friendly output, while `inspect` shows detailed object structure for debugging.

4. Master arrays and hashes:

Most data in Ruby is managed using arrays or hashes, so mastering them is essential.

5. Write modular code with methods:

Use methods to keep your code organized and readable.

6. Explore gems (libraries):

Ruby has a vast ecosystem of gems to simplify complex tasks.

7. Practice regularly:

Write code daily and debug your mistakes to build confidence.

6 Essential Tools for Ruby:

1. ``irb``:

An interactive Ruby shell for quick code testing.

```
bash

irb
```

2. ``pry``:

A more advanced tool than ``irb`` for debugging and exploring code.

```
bash

gem install pry
```

3. ``rubocop``:

A static code analyzer to enforce Ruby coding standards.

```
bash

gem install rubocop
```

4. ``rspec``:

A popular testing framework for writing and running tests.

```
bash

gem install rspec
```

5. ``rails``:

A powerful framework for web development with Ruby.

```
bash

gem install rails
```

6. ``bundler``:

A tool to manage project dependencies.


```
bash
```

```
gem install bundler
```

5 Suggested Exercises for Students:

1. Print even and odd numbers:

Write a program to print even and odd numbers within a specified range.

2. Array manipulation:

Create an array of student names and sort them alphabetically.

3. Calculate average:

Write a program to calculate the average of student grades.

4. Create a `Student` class:

Define a class to store student information and print their details.

5. Date operations:

Write a program to calculate the number of days remaining until the end of the year.

3 Mini Projects for Class:

1. Gradebook System:

Create a program to store student grades in a hash with features to:

- Add students
- Search for a student's grade
- Display all students and grades

2. Library Manager:

Build a simple system to manage books in a library:

- Add books
- Search for books
- Remove books

3. Simple Calculator:

Create a calculator to perform basic operations like addition, subtraction, multiplication, and division.

These examples, tips, and projects will keep your workshop engaging and help students effectively learn Ruby basics!