

IT 資格 道 場

Ruby技術者認定試験 Gold 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

Rubyアソシエーション「Ruby技術者認定試験 Gold」出題範囲(Ruby 3.1 対象)準拠
2026年10月版

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

目次

この教材群の使い方(3点セット)

この試験が問うもの

第1章 実行環境 [1.1]

組み込み変数と組み込み定数 [1.1]

サンプルはここまでです

SAMPLE

この教材群の使い方(3点セット)

種類	役割
① 学習用テキスト	これだけで問題が解けるようになる本編
② 頻出度別ノート	テキストを読んだら次はこれ。頻出順に絞った問題と解説で「解ける」に橋渡し。試験直前の最終チェックにも
③ 暗記用ノート	覚えるべき要点だけを一問一答に凝縮。空き時間の反復と用語の再確認用

使い方: ① 学習用を読む → ② 頻出度別で解き方を掴む → ③ 問題演習で腕試し → ④ 頻出度別に戻って再確認(試験直前もここ)。暗記用は空き時間や用語の再確認に。

この試験が問うもの

Ruby 技術者認定試験 Gold は、Ruby の文法と標準ライブラリを深く理解しているかを問う試験です。対象は Ruby 3.1 系です。出題範囲は実行環境、文法、オブジェクト指向、組み込みライブラリ、標準添付ライブラリの 5 分野あり、このテキストの章立ても同じ順に並べました。

項目	内容
運営	一般財団法人 Ruby アソシエーション
対象バージョン	Ruby 3.1.x
問題数	50問(選択式。一部は複数選択)
試験時間	90分
合格ライン	正答率75%
受け方	コンピュータ試験 (CBT)
認定	Gold の認定には Silver と Gold の両方の合格が必要

本番はコードを読んで実行結果を選ぶ問題が中心です。複数選択は「2つ選択」の形で出ます。例のコードは、手元の Ruby 3.1 で実行して出力を確かめるまで、読み終えたことにはなりません。バージョンが違くと結果が変わる所があります。

第1章 実行環境 [1.1]

組み込み変数と組み込み定数 [1.1]

`$` で始まる特別なグローバル変数と、`ARGV` や `ENV` などの定数は、宣言なしで使えるもので、値が何を指すか、いつ書き換わるかが問われます。

標準出力と標準エラー出力

`puts` と `print` と `p` は `$stdout` に書き、`warn` は `$stderr` に書きます。出力先を決めているのは定数ではなく、この2つのグローバル変数です。

どちらにも別のオブジェクトを代入できますが、`write` に応答しないものは受け付けません。`1` を代入すると `TypeError` です。元の入出力は定数 `STDOUT` と `STDERR` に残っているので、いつでも戻せます。

```
require "stringio"

old = $stdout
$stdout = StringIO.new

puts "hidden"

print "x", "y"

buf = $stdout.string

$stdout = old

p buf # => "hidden\nxy"

$stderr = StringIO.new

warn "w"

err = $stderr.string

$stderr = STDERR

p err # => "w\n"

begin
  $stdout = 1
rescue TypeError => e
  p e.class # => TypeError
end
```

スクリプト名の変数

`$0` は起動したスクリプトの名前で、`$PROGRAM_NAME` は同じ値の別名です。
`__FILE__` は、その式が書かれているファイルの名前です。`require` で読み込まれたファイルの中では、2つは等しくなりません。`ruby main.rb` で直接動かしたファイルでは、等しくなります。

そのため `if __FILE__ == $0` は、直接実行された時だけ動く部分を書く定番の形になります。`$0` には代入できますが、`__FILE__` は変数ではないので、代

入すると `SyntaxError` です。`__dir__` は、`__FILE__` があるディレクトリの絶対パスを返すメソッドで、変数ではありません。

```
if __FILE__ == $0
  puts "main script" # => main script
end

$0 = "renamed"

p $PROGRAM_NAME # => "renamed"

p __FILE__ == $0 # => false
```

直接実行した時だけ動かす条件として、次の3つの式は働きません。

式	結果
<code>\$0 == \$PROGRAM_NAME</code>	同じ値の別名どうしの比較で、読み込まれた場合でも常に真
<code>__FILE__ == ARGV.first</code>	引数を渡さなければ <code>nil</code> との比較で、直接実行しても偽
<code>__FILE__ == __method__.to_s</code>	メソッドの外では <code>__method__</code> が <code>nil</code> で、空文字列との比較になり常に偽

次のファイルを別のファイルから `require_relative` すると、読み込まれた側の `__FILE__ == $0` は偽になります。

```
puts "loaded"

p __FILE__ == $0 # => false

p $0 == $PROGRAM_NAME # => true
```

区切りの変数 `$;` と `$,`

`$;` は、引数なしの `split` が使う区切りです。`$,` は、引数なしの `join` が使う区切りであり、`print` が引数の間に入れる文字でもあります。初期値はどちらも `nil` で、`$;` が `nil` の `split` は空白で区切ります。

Ruby 3.1 では、どちらも非推奨の変数で、`nil` 以外を代入して `-w` を付けて実行すると、`warning: `$,,' is deprecated` のような警告が標準エラーに出るだけで、代入した値は無視されません。

```
p "a b,c".split # => ["a", "b,c"]
$; = ","
p "a b,c".split # => ["a b", "c"]
$; = nil
$; = "-"
puts [1, 2, 3].join # => 1-2-3
puts [1, 2, 3].join("+") # => 1+2+3
print "x", "y" # => x-y
```

例外の変数 `$!` と `$@`

`$!` は、いま処理している例外で、`rescue` 節の中だけで値を持ち、例外が起きていない所では `nil` です。節を抜けると、その節に入る前の値へ戻ります。

入れ子の `rescue` では、内側の節に入ると `$!` は内側の例外になります。内側の節を抜けた直後は、外側の例外に戻ります。外側の節も抜けるまで、`nil` には戻りません。最後に起きた例外が残り続けることはありません。`$@` は `$!.backtrace` と同じ配列で、`$!` が `nil` なら `nil` になります。

```
begin
  raise "outer"
rescue
  begin
    raise "inner"
  rescue
    puts $!.message # => inner
  end
  puts $!.message # => outer
end
p $! # => nil
p $@ # => nil
begin
  raise ArgumentError, "bad"
rescue
  p $@ == $!.backtrace # => true
end
```

照合結果の変数 `$~` と `$1`

`=~` や `match` が成功すると、結果の `MatchData` が `$~` に入ります。関連する変数は、次の表の5つです。

変数	中身
<code>\$~</code>	直近の照合の <code>MatchData</code>
<code>\$1</code> 、 <code>\$2</code> など	1番目、2番目のグループの文字列
<code>\$&</code>	一致した全体

変数	中身
<code>\$`</code>	一致の前の部分
<code>\$'</code>	一致の後の部分

照合に失敗すると `$~` も `$1` も `nil` になり、`Regexp.last_match` は `$~` と同じ値を返します。

`$~` は、プログラム全体で1つの変数ではありません。メソッドの呼び出しごとに別々に持たれるので、メソッドの中で照合しても、呼び出し元の `$~` と `$1` は変わりません。ブロックは、それを囲むメソッドと同じ `$~` を共有するので、ブロックの中の照合は外から見えます。`case` の `when` に置いた正規表現や、`sub` と `gsub` のブロックの中でも、`$~` は更新されます。`match?` だけは更新しません。正規表現そのものは、第4章の正規表現の節で扱います。

```

def inner
  "xy" =~ /(x)/
  $1
end

"ab" =~ /(b)/

p inner # => "x"
p $1   # => "b"

[1].each { "q" =~ /(q)/ }

p $1     # => "q"

"cd".match?(/(c)/)

p $1     # => "q"

"zzz" =~ /(a)/

p $~     # => nil

p $1     # => nil

```

__method__ と __callee__

どちらも、実行中のメソッド名をシンボルで返し、メソッドの外では `nil` になります。`__method__` は定義された時の名前を返し、`__callee__` は呼び出しに使った名前を返します。違いが出るのは、`alias` で別名を付けた時だけです。メソッド内のブロックの中では、囲んでいるメソッドの名前を返し、ブロック専用の名前にはなりません。

呼び出し	<code>__method__</code>	<code>__callee__</code>
<code>foo</code>	<code>:foo</code>	<code>:foo</code>
<code>bar (alias bar foo)</code>	<code>:foo</code>	<code>:bar</code>

```
def foo
  [__method__, __callee__]
end

alias bar foo

p foo # => [:foo, :foo]
p bar # => [:foo, :bar]
p __method__ # => nil

def outer = [1].map { __method__ }
p outer # => [:outer]
```

ARGV と ENV

ARGV は、スクリプト名の後ろに並べたコマンドライン引数の配列です。引数を渡さなければ空の配列で、要素はすべて文字列です。数として使いたい時は、`Integer(ARGV[0])` のように自分で変換します。

ENV は環境変数にアクセスする、ハッシュに似たオブジェクトです。ただし `Hash` ではありません。クラスは `Object` で、`to_h` で本物のハッシュに変えられます。`ENV["名前"]` は、無ければ `nil` を返します。`fetch` は、無いと `KeyError` になり、2番目の引数を渡せばその値を返します。値に代入できるのは文字列だけで、整数を入れると `TypeError` です。

```
p ARGV # => ["a", "b"]
ARGV.shift

p ARGV # => ["b"]
ENV["RG_TEST"] = "1"
p ENV["RG_TEST"] # => "1"
p ENV["RG_NONE"] # => nil

begin
  ENV.fetch("RG_NONE")
rescue KeyError => e
  p e.class # => KeyError
end

p ENV.fetch("RG_NONE", "dflt") # => "dflt"

begin
  ENV["RG_X"] = 1
rescue TypeError => e
  p e.class # => TypeError
end

p ENV.class # => Object
p ENV.to_h.class # => Hash
```

バージョンと実行環境の定数

`RUBY_VERSION` は `"3.1.7"` のような文字列で、数値ではありません。文字列として比較されるため、`"3.10" < "3.9"` は `true` になり、桁数が違うバージョンの比較には使えません。

```
p RUBY_VERSION.class # => String
p RUBY_VERSION >= "3.1" # => true
p RUBY_VERSION.split(".").first(2) # => ["3", "1"]
p "3.10" < "3.9" # => true
p RUBY_ENGINE # => "ruby"
```

そのほかの特別な変数は、次の5つです。

変数	中身
<code>\$\$</code>	実行中のプロセス ID (整数)
<code>\$/</code>	入力の行区切り。初期値は <code>"\n"</code>
<code>\$\</code>	<code>print</code> が最後に付ける文字。初期値は <code>nil</code>
<code>\$(\$LOAD_PATH)</code>	<code>require</code> が探すディレクトリの配列
<code>\$" (\$LOADED_FEATURES)</code>	読み込み済みのファイルの配列

DATA と __END__

最初に動かしたスクリプトに `__END__` の行があると、定数 `DATA` が定義されます。`DATA` は `File` オブジェクトで、読み位置は `__END__` の次の行の先頭から始まります。`__END__` の行そのものは読めません。`require` で読み込んだファイルに `__END__` があっても、`DATA` は定義されません。

読み位置は1つだけで、`gets`・`read`・`readlines` が共有します。`gets` が1行読めば、次の `read` はその続きから残りを返し、先頭へは戻りません。読み切ると `eof?` は `true` になり、`gets` は `nil`、`read` は空文字列を返します。`DATA`

を別のオブジェクトに渡した場合も、読み位置は共有されたままで、新しく読み直すことはありません。

```
class Reader
  def initialize(io) = @io = io
  def first = @io.gets
  def rest = @io.read
end

r = Reader.new(DATA)
p [r.first, r.rest] # => ["head\n", "x\ny\n"]
p DATA.eof?      # => true
p DATA.gets      # => nil
p DATA.read      # => ""
p DATA.class     # => File
__END__
head
x
y
```

`DATA` が指す `File` は、スクリプトのファイル全体です。`pos` が返すのはファイルの先頭からの位置で、初めは `__END__` の次の行の先頭にあたります。読み位置を戻すには、`pos` をあらかじめ控えておく必要があり、控えなければ `__END__` の次の行には戻れません。

呼び出し	動く先
<code>seek(start)</code>	控えた位置。次の <code>gets</code> は <code>__END__</code> の次の行

呼び出し	動く先
<code>seek(0, IO::SEEK_SET)</code>	ファイルの先頭。次の <code>gets</code> はスクリプトの1行目
<code>seek(0, IO::SEEK_END)</code>	末尾。次の <code>gets</code> は <code>nil</code>
<code>seek(n, IO::SEEK_CUR)</code>	今の位置から <code>n</code> バイト先。末尾を越えると <code>gets</code> は <code>nil</code>
<code>rewind</code>	ファイルの先頭

```
start = DATA.pos
DATA.read
DATA.seek(start)
p DATA.gets # => "one\n"
DATA.seek(0, IO::SEEK_SET)
p DATA.gets # => "start = DATA.pos\n"
DATA.seek(0, IO::SEEK_END)
p DATA.gets # => nil
DATA.seek(start)
DATA.gets
DATA.seek(-2, IO::SEEK_CUR)
p DATA.gets # => "e\n"
__END__
one
two
```

サンプルはここまでです

続き(第2章～巻末)は、Ruby技術者認定試験 Gold 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

Ruby技術者認定試験 Gold 学習用テキスト(無料サンプル)

版

2026年10月版(Rubyアソシエーション「Ruby技術者認定試験 Gold」出題範囲 (Ruby 3.1 対象) 準拠)

発行

IT資格道場(<https://www.shikaku-doujou.com>)

お問い合わせ

info@shikaku-doujou.com

Ruby は Ruby アソシエーションおよび関係者の商標または登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式の出題範囲と Ruby 3.1 のリファレンスの内容を基に作成したもので、一般財団法人 Rubyアソシエーションとの提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。