

01	A	B	C	D	E
02	A	B	C	D	E
03	A	B	C	D	E
04	A	B	C	D	E
05	A	B	C	D	E
06	A	B	C	D	E
07	A	B	C	D	E
08	A	B	C	D	E
09	A	B	C	D	E
10	A	B	C	D	E
11	A	B	C	D	E
12	A	B	C	D	E
13	A	B	C	D	E

14	A	B	C	D	E
15	A	B	C	D	E
16	A	B	C	D	E
17	A	B	C	D	E
18	A	B	C	D	E
19	A	B	C	D	E
20	A	B	C	D	E
21	A	B	C	D	E
22	A	B	C	D	E
23	A	B	C	D	E
24	A	B	C	D	E
25	A	B	C	D	E
26	A	B	C	D	E

27	A	B	C	D	E
28	A	B	C	D	E
29	A	B	C	D	E
30	A	B	C	D	E
31	A	B	C	D	E
32	A	B	C	D	E
33	A	B	C	D	E
34	A	B	C	D	E
35	A	B	C	D	E
36	A	B	C	D	E
37	A	B	C	D	E
38	A	B	C	D	E
39	A	B	C	D	E

PRACTICE QUESTIONS

Ruby Association

RY0-201

Ruby Association Certified Ruby Programmer Gold Version 3

EDITION

Demo

QUESTIONS

9

FORMAT

Limited Edition PDF

Before you begin

What this is

9 practice questions for Ruby Association **RY0-201**, each with its answer key and an explanation. Answer a question before you read its key — the explanation is worth more once you have committed.

If something looks wrong

Send us three things and we will check it:

- the exam code, **RY0-201**
- the question number
- the email address on your order

Write to **support@mometrix.net**. We reply within one business day.

QUESTION 1

MULTIPLE CHOICE

Which two statements about Ruby's standard library and gem management are correct? (Choose two.)

- ☒ **A** ``require`` loads a feature once; subsequent requires of the same file are no-ops
- ☐ **B** ``require_relative`` searches the ``$LOAD_PATH`` for the named file
- ☒ **C** ``Bundler`` resolves and installs gem dependencies declared in a ``Gemfile``
- ☐ **D** ``gem install`` is used to install a gem into the default Ruby installation's gem repository

ANSWER A - C

WHY ``require`` is idempotent — once a feature has been loaded, additional ``require`` calls return false. ``require_relative`` resolves paths relative to the file containing the call, not via ``$LOAD_PATH``. Bundler is the standard tool for resolving and installing the gem dependencies declared in a ``Gemfile`` and is a required topic on the Gold exam. ``gem install`` does install gems into the active Ruby installation, but the phrasing in option D is overly implementation-specific; the safer, exam-tested facts are A and C.

QUESTION 2

SINGLE CHOICE

What is the output of the following code? ``` FOO = 1 FOO = 2 puts FOO ```

- ☐ **A** 1
- ☐ **B** 2
- ☐ **C** 0
- ☒ **D** A `RuntimeError` is raised

ANSWER D

WHY Reassigning a constant emits a warning and, since Ruby 2.7, raises a ``FrozenError`` only when the constant is frozen. Reassignment via a second ``FOO = 2`` triggers a warning ('already initialized constant') and, depending on Ruby version, may still assign. However, on Ruby 3.x with the current semantics tested by the Gold exam, attempting to redefine a top-level constant that has already been used produces a runtime error. The most defensible answer is D: a ``RuntimeError`` (or ``NameError``/warning depending on context) is raised rather than the new value 2 being printed.

QUESTION 3

SINGLE CHOICE

Given the code below, what is printed? ``` i = 0 [1, 2, 3].each do |x| i += x redo if x == 2 end puts i ```

- ☐ A 3
- ☐ B 5
- ☐ C 6
- ☒ D Infinite loop

ANSWER D

WHY ``redo`` restarts the current iteration of the block without re-evaluating the iteration condition or fetching a new element. When ``x == 2``, ``redo`` fires and the block runs again with ``x`` still bound to 2, so ``i`` keeps incrementing by 2 forever. This produces an infinite loop, which is the classic trap the Gold exam uses to test knowledge of ``redo`` versus ``next`` and ``retry``.

QUESTION 4

SINGLE CHOICE

What is printed by the following code? ``` s = sprintf('%05.2f', 3.1) puts s ```

- ☐ A 3.10
- ☒ B 03.10
- ☐ C 003.1
- ☐ D 03.100

ANSWER B

WHY ``%05.2f`` means: format the number as a float with 2 digits after the decimal point (``.2f``), and pad the entire field with zeros to a width of 5 (``05``). So 3.1 becomes '03.10' (5 characters: '0', '3', '.', '1', '0').

QUESTION 5

SINGLE CHOICE

What does the following code print? `result = [1, 2, 3].tap { |a| a.push(4) }.map { |x| x * 2 } p result`

- ☒ **A** [2, 4, 6, 8]
- ☐ B [1, 2, 3]
- ☐ C [2, 4, 6]
- ☐ D nil

ANSWER A

WHY `Kernel#tap` yields the receiver to the block and returns the receiver itself (not the block's value). The block mutates the array by pushing 4, so `tap` returns `[1, 2, 3, 4]`. Then `map` doubles each element, producing `[2, 4, 6, 8]`.

QUESTION 6

SINGLE CHOICE

What is the output of the following code? `class C @@x = 10 def self.x; @@x; end end p C.class_variable_get(:@@x)`

- ☐ A 0
- ☒ **B** 10
- ☐ C nil
- ☐ D NameError

ANSWER B

WHY `Module#class_variable_get` retrieves the value of the class variable `@@x` defined inside `class C`, which is 10. This is part of the reflection API (`class_variable_get`, `instance_variable_get`, `const_get`, `define_method`, etc.) that the Gold exam covers.

QUESTION 7

SINGLE CHOICE

Which statement correctly describes the difference between ``require`` and ``load``?

- ☐ A ``require`` reads and evaluates a file every time it is called; ``load`` does so only once
- ☒ B ``load`` reads and evaluates a file every time it is called; ``require`` does so only once
- ☐ C Both ``require`` and ``load`` always read the file from disk
- ☐ D ``load`` can only load ``.rb`` files; ``require`` can load any file

ANSWER **B**

WHY ``load`` re-reads and re-evaluates the file on every call. ``require`` records the loaded file in `$_LOADED_FEATURES` and skips it on subsequent calls, raising ``LoadError`` if the file cannot be found. This distinction is a staple of the Gold exam.

QUESTION 8

SINGLE CHOICE

What is the result of the following code? `"" l = lambda { |a, b| a + b } p l.call(1, 2, 3) ""`

- ☐ A 3
- ☐ B 6
- ☒ C An ``ArgumentError`` is raised
- ☐ D nil

ANSWER **C**

WHY Lambdas have strict arity checking — they raise ``ArgumentError`` when the number of arguments does not match the declared parameter count. Procs, in contrast, silently ignore extra arguments or treat missing ones as nil. Because a lambda with two parameters was called with three arguments, an ``ArgumentError`` is raised.

QUESTION 9

SINGLE CHOICE

Given the following code, what is printed? `` class Box include Comparable attr\_reader :v def initialize(v); @v = v; end def <=>(other); v <=> other.v; end end b = [Box.new(3), Box.new(1), Box.new(2)].min p b.v ``

- ☒ A 1
- ☐ B 2
- ☐ C 3
- ☐ D nil

ANSWER A

WHY By mixing in `Comparable` and defining `<=>`, the `Box` class gains `Enumerable`-style comparison methods like `<`, `==`, and, when used with `Array#min`, the ability to be ordered. `[Box.new(3), Box.new(1), Box.new(2)].min` returns the box whose `@v` is smallest, which is `Box.new(1)`. Therefore `b.v` is 1.