A1: (a) and (b)

In Ruby, all objects have a logical value for use in conditional statements.

The objects `false` and `nil` are treated as logically false, all other objects are treated as logically true.

A2: (c) and (e)

The complete list of reserved words as of Ruby 3.1 is listed below:

BEGIN	class	ensure	nil	self	when
END	def	false	not	super	while
alias	defined?	for	or	then	yield
and	do	if	redo	true	__LINE__
begin	else	in	rescue		__FILE__
break	elsif	module	retry	unless	__ENCODING__
case	end	next	return	until	

A3: (c)

Ruby variable names must begin with a lowercase letter or underscore, and may contain only letters, numbers, underscore, and non-ASCII characters.

Variable names must not conflict with keywords (e.g. you cannot have a variable called `class`), but unambiguous names that contain reserved words are acceptable (i.e. both `classy` and `_class` are valid Ruby variable names)

A4: (a) and (b)

Single quoted string literals are simple, and are meant to represent raw sequences of characters.

Double quoted string literals are more complex, but offer extra features such as string interpolation (`#{...}`), where entire Ruby expressions can be evaluated and inserted into a string.

As a shortcut, `#$` is usable for inserting the contents of a global variable into a string. (Similarly, `#@` can be used with instance variables). This shortcut variant is less commonly used than the more general `#{...}` form.

A5: (c)

A leading zero in an integer literal indicates 'octal-mode' in Ruby, i.e. a number in base 8 format. However, all print functions in Ruby will output numeric values in base 10 by default.

Should you need to output numbers in something other than base 10, there are many different functions in Ruby for formatted numeric output (e.g. `String#%`, `Numeric#to_s(base)`, `Kernel#sprintf`)

A6: (b)

The ternary operator (`cond ? expr1 : expr2`) is a compact form of `if/else` which will return `expr1` if `cond` is true, otherwise will return `expr2`. It is most suitable for short statements that easily fit on a single line.

A7: (a)

Ruby `case` statements will select the first branch to match its `when` condition.

Because Ruby's two-dot range literal is an inclusive range, the end value is included as part of the range.

So although both `1..120` and `120..170` include `120`, the `when 1..120` branch is matched because it appears first in the case statement.

A8: (d)

Although local variables from the surrounding scope are accessible within blocks, block parameters themselves are always block-local variables. This means that when a block parameter has the same name as a local variable from the surrounding scope, within the block any references will refer to the block-local variable. This prevents accidental modification of variables from the outside scope due to naming collisions.

Defining block parameters with the same name as a local variable from the surrounding scope is considered an antipattern and may be a sign of an accidental programming error. To catch this problem, run `ruby` with the `-w` flag, and you will see warnings like `warning: shadowing outer local variable - item` wherever this problem occurs.

A9: (c)

The `Integer#times` method yields values starting at zero, up to one less than the specified integer.

Although block variables with the same name of local variables from the surrounding scope are shadowed (see `A8`), other local variables are accessible and can be modified. This is because Ruby blocks are *closures*.

A10: (d)

String#each_char returns an Enumerator which yields each character in the receiver. Note that characters are represented by single character strings.

Enumerable#map returns an array of objects returned by the given block.

`string * integer` returns a string containing `integer` copies of `string`.

Therefore, `s.each_char.map { |i| i * 2}` evaluates to `["aa", "bb", "cc", "dd", "ee"]`.

A11: (c)

String#chars returns an array of characters in the receiver. Note that characters are represented by single character strings.

Enumerable#tally returns a hash containing the counts of equal elements.

Therefore, `"cocoa".chars.tally` evaluates to `{"c"=>2, "o"=>2, "a"=>1}`.

A12: (b)

String#gsub returns a copy of the receiver with all occurrences of the given pattern replaced.

String#sub returns a copy of the receiver with only the first occurrence of the given pattern replaced.

String#replace replaces the contents of the receiver with the contents of the given string.

String does not have `#replace_all` method.

A13: (b)

In an `if/elsif/else` conditional statement, the first matching `if` or `elsif` branch will be executed. If none match, then the `else` branch will be run.

A14: (b)

String#slice returns the first matching substring found in the receiver, or `nil` if none found. `&.` is called safe nagivation operator. `x&.foo` invokes `foo` on `x` if `x` is not `nil`. If `x` is `nil`, `x&.foo` does not invoke `foo` and evaluates to `nil`.

A15: (b)

If any keyword arguments are not given, the default value from the method definition will be used.

A16: (b)

`**` turns a Hash into keyword arguments.

A17: (a) and (c)

Character classes (`[...]`) match any single character from within the brackets.

Alternatives (`...|...`) are used to match any one of many possible subexpressions.

The `\A` anchor matches the beginning of a string, and the `\z` anchor matches the end of a string.

Note that the reason that (b) is not a correct answer is because its subexpressions are `\ARuby` and `ruby\z`, allowing matches for things like `Ruby123`

A18: (d)

Constants can be redefined, but because this is usually a bad practice, a warning is displayed.

Because Ruby also uses constants for referencing module and class names, the constant redefinition warning can also help catch accidental naming collisions.

A19: (b)

No warning is shown because the constant is not being redefined; instead the object it references is being modified.

By convention, objects referenced by constants are usually treated as immutable. But there are certain rare cases where that convention would not apply.

A20: (b) and (e)

Some notes on Ruby variable naming rules:

- Global variables start with `$`.
- Class variables start with `@@`.
- Instance variables start with `@`.
- Local variables must begin with a lowercase letter or an underscore.
- The remaining characters in any variable type are limited to letters, numbers, underscores, and non-ASCII characters.

A21: (c)

In this example, both the `x` and `y` variables reference the same array object.

Because `Array#reject!` modifies its receiver, this means that it modifies the single array that is referenced by both variables.

A22: (c)

Some notes on `Array` operations:

- `shift` removes the first element of an array and returns its value.
- `pop` removes the last element of an array and returns its value
- `push` adds the specified element to the end of an array.

A23: (b)

The logical `||` and `&&` operators short-circuit, only executing the right side of the expression if necessary.

The special `|` and `&` operators provided on Ruby's boolean objects do not short circuit, so the right side of the expression is always evaluated.

Note that all Ruby objects support the `||` and `&&` operators, but not all objects implement `|` and `&`.

A24: (c)

Although the `or` operator short circuits and the `n = true` expression is never executed, the `n` local variable is still statically declared. Therefore, the variable is present but its value is `nil`.

A25: (a) and (d)

This question illustrates two different ways of indexing a sub-array.

One approach is to use two integers, i.e. `x[1,3]` -- This means "get a subarray of length `3` starting at index `1`.

Another approach is to use a range, which generates a subarray based the index values within that range.

The simple form of using a range is something like `x[1..3]` which would give you a subarray starting at index `1` and ending at index `3`.

But Ruby also allows *negative* indexing, which define indexes relative to the end of an array rather than the beginning.

Thus, `x[-4..-2]` is referring to the subarray starting from the 4th to the last position in the array, and continuing to the second-to-last position.

To clarify, here is a list of the index values for each position in the array from this question:

```
  x  [ 9,   7,   5,    3,    1]
  i    0    1    2     3     4
 -i   -5   -4   -3    -2    -1
```

A26: (b) and (d)

`Array#select` and `Array#filter` return elements for which the given block returns a truthy value.

A27: (a)

`String#to_i` attempts to parse an integer from a string starting from its first character, and continuing until the end of a valid number in a particular base. If a string does not begin with a valid number, `0` is returned.

By default, numbers are assumed to be in base 10, but other bases (from `2` to `36`) can be specified via a parameter.

Note that while `"42A7".to_i` returns `42` because `A` is not a valid part of a base 10 number, `"42A7".to_i(16)` would extract the hexadecimal value `0x42A7`, which when converted to decimal would be equal to `17063`.

A28. (b)

`has_key?`, `include?`, `key?`, `member?` are all aliases for a single method which returns `true` if the given key is present in the hash, and returns `false` otherwise.

The `contain?` method is not defined by `Hash`.

A29: (a) and (e)

Some notes on array processing methods:

- In addition to `reject!` there is also `reject`, which returns a new array rather than modifying the original.
- Because there isn't a non-destructive form of `delete_if`, there is no `delete_if!` method. By convention Ruby only uses `!` at the end of the method when there are two features that work similarly but with one being more dangerous than the other.
- The `Array#slice` method works similarly to `Array#[]`, and is used for retrieving a specific value or subarray by index rather than filtering based on a condition. It does not accept a block.

A30: (c)

The `|` operator is equivalent to a set union. It returns a new array that is built by joining two arrays together, eliminating any duplicates while preserving order.

A31: (d)

`%i(...)` is an non-interpolated array of symbols, separated by whitespace.

See documentation of percent literals for details.

A32: (b)

In a `begin/rescue/end` block... the first matched `rescue` clause will be executed.

Because `SomeOtherError` is a subclass of `SomeError`, it matches the `rescue SomeError` clause, and so that branch is what gets run.

In a real application, it is usually a good practice to attempt to match more specific errors before the more general errors that they inherit from (e.g. `rescue StandardError` would usually come last).

A33: (c)

Dividing by zero raises a `ZeroDivisionError` exception.

That exception is rescued, and a message is printed out. Then `exit(1)` tells Ruby to exit with an error code.

But because the `begin...end` expression has an `ensure` section, it is run before the interpreter exits.

The `ensure` clause is useful because it can be used to do cleanup even when some code raises an exception or tells Ruby to exit. It is often used for things like closing file handles, database connections, etc.

A34: (e)

By default, all classes inherit from the `Object` class, whether or not they are explicit subclasses of some other class.

To create class hierarchies that do not inherit from `Object`, it is possible to explictly inherit from `BasicObject` instead, which has very few features built into it. But the use cases for doing so are uncommon.

A35: (c)

Class definitions can be re-opened and updated at any time, including Ruby core classes like `Object`.

Because all Ruby core objects (except `BasicObject`) inherit from the `Object` class, adding new methods to `Object` will make them available on all objects.

A36: (c)

Whenever the `new` method is called on a class, a new instance of that class is allocated and then the `initialize` method is called on that instance. This allows some setup code to be run as soon as the object is instantiated.

A37: (d)

The `new` method (defined by `Class`) is used to create new object instances.

A38: (a)

The `super` keyword invokes a method with the same name higher up the ancestry chain.

In this particular example, calling `Bar.new` causes `Bar#initialize` to run, which sets `@var = "banana"`. But then immediately after that, `super` is called, causing `Foo#initialize` to run. That method sets `@var = "apple"`, which explains the final result.

A39: (d)

String#delete_prefix returns a copy of the receiver with a given prefix deleted.

`"$foo$".delete("$")` evaluates to `"foo"` because String#delete returns a copy of the receiver with all characters specified by its arguments deleted.

`sub("$")` and `chop("$")` raise ArgumentError because String#sub takes two arguments and String#chop takes no argument.

A40: (c)

The `to_a` method uses the common naming convention for converting an object into an array, and is found throughout Ruby's collection classes.

Some objects also implement `to_ary`, which is used for implicit conversions. For example, `Array#flatten` will attempt to call `to_ary` on the elements within an array if it is present. But these use cases are uncommon.

A41: (b)

The `#find` method is defined by the `Enumerable` module. It returns the first element of the collection for which the block's result is not `false` or `nil`.

Note that `Enumerable#find` is also aliased as `Enumerable#detect`.

A42: (a) and (c)

The `sort_by` method maps the elements in a collection to a set of values via a block, and then sorts the elements of the collection in ascending order based on those values.

The `sort` method (when called without a block) sorts an array in ascending order directly based on the values of its elements. There is also a block form of `sort` which allows for element-by-element comparison.

Both `sort_by` and `sort` rely on the `<=>` operator to be defined in order to make comparisons between objects. Ruby's `Numeric` classes all implement this operator, but you can also define it for your own objects.

A43: (b)

When called with a block, `sort` will attempt to put elements in order based on the block's result.

The block must implement a comparison between two elements, and is expected to return a negative integer when the first element should appear before the second in the sorted array, `0` if the two elements have an equal sort order, and a postive integer when the first element should appear after the second in the sorted array.

Ruby's numeric objects implement `<=>`, which provides this behavior automatically:

```
>> 3 <=> 1
=> 1
>> 3 <=> 3
=> 0
>> 3 <=> 5
=> -1
```

The `<=>` (spaceship operator) can be implemented by any object that has a meaningful sort order.

A44: (b)

The `seek` method is used to move to a specific byte offset in an I/O stream. Offsets are zero-based, so `seek(5)` sets the position in the stream to just *after* the fifth byte.

The `gets` method reads from the current position in the stream to the end of a line.

A45: (a)

The `"r"` open mode means "read only, starting from the beginning of the file."

This is both the safest default option and the most common use case.

A46: (d) and (e)

The following I/O open modes are supported by Ruby:

```
"r"  Read-only, starts at beginning of file  (default mode).

"r+" Read-write, starts at beginning of file.

"w"  Write-only, truncates existing file
     to zero length or creates a new file for writing.

"w+" Read-write, truncates existing file to zero length
     or creates a new file for reading and writing.

"a"  Write-only, each write call appends data at end of file.
     Creates a new file for writing if file does not exist.

"a+" Read-write, each write call appends data at end of file.
     Creates a new file for reading and writing if file does
     not exist.
```

A47: (b) and (c)

Some additional notes:

`FileUtils.mv` from the *fileutils* stdlib can be used to rename a directory.

`File.basename` is used for getting the last part of a file name from a path string. (e.g. `File.basename("long/path/to/something") #=> "something"`)

A48: (b)

Similar to the syntax for indexing subarrays (Q25), it is possible to index a substring by providing a starting position and length.

A49: (b)

Note that the replacement string does not need to be the same length as the original string. For example:

```
>> str = "boat"
=> "boat"
>> str[1,2] = "uil"
=> "uil"
```

```
>> str
=> "built"
```

A50: (b)

Ruby's numeric objects define a method called `coerce` which attempts to convert objects into the same type for arithmetic operations. This method is not implemented by the `String` class, so a `TypeError` is raised.

Note that if the order was reversed (i.e. `"hi" * 5`), then the result would be `"hihihihihi"`. This is because `String` does define its own `*` operator, which is used when the string appears on the left hand side of the expression.
