Ruby master - Bug #18931
Inconsistent handling of invalid codepoints in String#lstrip and String#rstrip

07/20/2022 04:25 PM - nirvdrum (Kevin Menard)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
<tr>
<td>ruby -v:</td>
<td>ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [arm64-darwin21]</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN</td>
</tr>
</tbody>
</table>

**Description**

When attempting to strip a string, there are three basic options when an invalid code point is encountered:

1. Ignore the code point
2. Strip the code point
3. Raise an exception

For background, Ruby does not consider the string's code range for lstrip or rstrip. It permits stripping strings with an ENC_CODERANGE_BROKEN so long as any invalid code points are not encountered while performing the loop to remove whitespace. What it does when such a code point is encountered, however, is not consistent between lstrip and rstrip.

**String#lstrip will unconditionally raise an invalid byte sequence error:**

```
> ruby -v
ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [arm64-darwin21]
> ruby -e 'p " \x80abc".lstrip'
-e:1:in `lstrip': invalid byte sequence in UTF-8 (ArgumentError)
from -e:1:in `<main>'
> ruby -e 'p " \x80 abc".lstrip'
-e:1:in `lstrip': invalid byte sequence in UTF-8 (ArgumentError)
from -e:1:in `<main>'
> ruby -e 'p "\x80 abc".lstrip'
-e:1:in `lstrip': invalid byte sequence in UTF-8 (ArgumentError)
from -e:1:in `<main>'
> ruby -e 'p "\x80\n abc".lstrip'
-e:1:in `lstrip': invalid byte sequence in UTF-8 (ArgumentError)
from -e:1:in `<main>'
> ruby -e 'p "\x80\n\.lstrip''
-e:1:in `lstrip': invalid byte sequence in UTF-8 (ArgumentError)
from -e:1:in `<main>'
> ruby -e 'p " a\x80bc".lstrip'
*a\x80bc*  "# This one is okay because the broken code point appears after a non-whitespace code point.
```

Things get a lot messier with String#rstrip, however. Depending on context, rstrip may raise an exception, treat the broken code point as a non-whitespace boundary and stop processing, or treat the broken code point as if it were whitespace and remove it.

**String#rstrip will ignore the invalid code point if it immediately follows a non-whitespace code point:**

```
> ruby -v
ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [arm64-darwin21]
> ruby -e 'p "abc\x80 ".rstrip'
*abc\x80*  
> ruby -e 'p "abc\x80 " .rstrip'
*abc\x80*  
```

**String#rstrip will remove the invalid code point if it is surround by whitespace:**

```
> ruby -v
ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [arm64-darwin21]
> ruby -e 'p "abc\x80\n ".rstrip'
*abc\x80\n*  
> ruby -e 'p "abc\x80\n\n .rstrip'
*abc\x80\n\n*  
```
String#rstrip will raise an exception if no valid, non-whitespace code points appear before it:

```
> ruby -v
ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [arm64-darwin21]
> ruby -e 'p "abc \x80".rstrip'
"abc"
> ruby -e 'p "abc \x80 \".rstrip'
"abc"
> ruby -e 'p " \x80 \".rstrip'
""
```

It looks to me like the current behavior is a byproduct of the functions chosen for finding code point boundaries, rather than something deliberately chosen. E.g., rb_str_lstrip will call rb_enc_codepoint_len, which raises on invalid code points, while rb_str_rstrip calls rb_enc_prev_char, which doesn't perform the same code point validation. I think it'd make for a better user experience if lstrip and rstrip behaved consistently with each other, which would then unify the behavior in rstrip. What that behavior should be needs to be decided and I'm hoping to reach consensus on the semantics in this issue.

**History**

#1 - 07/20/2022 04:38 PM - nirvdrum (Kevin Menard)

My own take on three options, with no significance to the order, are:

**Ignore the code point**

The documentation for lstrip is "Returns a copy of the receiver with leading whitespace removed." It seems fairly straightforward and there's no mention of string validation; raising an exception might violate user expectations.

Treating broken code points the same as any other non-whitespace code point would be logically consistent. An additional benefit is the method could be implemented more efficiently as the whitespace check can be done without calculating code point boundaries. Only ASCII whitespace code points are stripped and those, by definition, are only one byte wide. However, if lstrip and rstrip ever evolve to handle non-ASCII whitespace we'll be back to calculating code point boundaries.

**Strip the code point**

Despite rstrip doing it in some cases, I don't think removing the invalid code points is what an end user would expect and runs counter to the method's documentation.

**Raise an exception**

Given lstrip's behavior, raising in all cases would be the most backward-compatible and is consistent with equivalent expressions (e.g., " \x80 abc".sub(/\s+/", ")) will raise an error on the invalid byte sequence). While the documentation makes no mention of string validation, encountering an invalid code point is arguably an exceptional condition.