While terms like strftime and strptime are ubiquitous throughout the history of computer science, I feel that the terms are very dense. If you are not already in-the-know, they are gibberish. If you are in the know, they are still a bit clunky. While discussing ways to improve the Time and Date formatting APIs for humanity, I thought a quick and easy improvement would be removing the need to use the method #strftime. #format is already reserved as a private method, but how do people feel about allowing a format string as an argument for #to_s?

I'm not comfortable writing C, but the relevant code is here

time_to_s(VALUE time) // add format arg
{
  struct time_object *tobj;

  GetTimeval(time, tobj);
  if (TZMODE_UTC_P(tobj))
    return strftimev("%Y-%m-%d %H:%M:%S UTC", time, rb_usascii_encoding()); // format || "%Y-%m-%d %H:%M:%S UTC"
  else
    return strftimev("%Y-%m-%d %H:%M:%S %z", time, rb_usascii_encoding()); // format || "%Y-%m-%d %H:%M:%S %z"
}

This would allow an API that feels a bit more intuitive. You still have to know the formatting symbols, but it creates a much more expressive statement:

# The current time, to string. What kind of string? A Y-m-d string.
Time.now.to_s('%Y-%m-%d')

(As an aside for discussion, I feel this way about formatting things like Floats and other numbers also. That API is equally confusing, and a holdover from history in comp-sci.)

History

#1 - 02/17/2020 04:43 PM - ttilberg (Tim Tilberg)

Gerald Bauer pointed out that this concept already exists in Rails' ActiveSupport:
https://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/time/conversions.rb

Is this a case where we might consider integrating this idea from Rails? I feel like it's very much in the spirit of core Ruby, with attention to developer happiness. It always surprises me how much I have to look up when I'm printing date/time as strings.

#2 - 02/17/2020 04:54 PM - shevegen (Robert A. Heiler)

Agreed. I can't remember them offhand either, I just copy/paste from my local knowledgebase. ;)
(Though I do happen to have a bad memory at any rate.)

To the proposal, though:

Time.now.to_s('%Y-%m-%d')

02/21/2020
I am not sure if this is a good suggestion though, largely because .to_s already having a distinct meaning, e.g. "to string" (or to a string representation).

People also typically associate .to_s, if there is an argument, with something like this:

```ruby
37.to_s(2).rjust(8, "0") # => "00100101"
```

So I think this should be considered as well, since the trade-off here is that .to_s would become a bit more complex.

(As an aside for discussion, I feel this way about formatting things like Floats and other numbers also. That API is equally confusing, and a holdover from history in comp-sci.)

I do not disagree with you here on the premise - I think it may be inspired a lot by C, and C may have been inspired by ... I don't know. Often people just typing less on UNIX I guess (e.g. /usr versus /users or /users and so forth, but you may wonder why it is called /usr/include/ rather than /usr/inc/ ... a lot of these things are not very logical or consistent. See also the explanation of how the /usr/sbin/ versus /sbin/ distinction came about, and the FHS not really making a whole lot of sense ... but I digress.)

I am just not entirely sure if .to_s should be modified.

I have no real strong preference either way though, just the trade-offs have to be considered.

but how do people feel about allowing a format string as an argument for #to_s?

Ultimately you have to convince matz. :)

I think it may be worth to consider whether it would/could be another method, other than .to_s, IF is to be considered that .to_s should not be changed. Then there could simply be an alias that may be easier to remember. Note that even in the .to_s example, people may not always remember the various format parameters without having to look them up. Perhaps it may be time for something simpler to remember altogether ... but I have no good suggestion for this either.

I'll just keep on copy/pasting from my local knowledgebase. :D

[... this concept already exists in Rails' ActiveSupport:](https://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/time/conversions.rb)

The active* ecosystem is very large, though. I am not sure if applying it as primary means of reasoning should be the primary explanation. Some time ago I suggested some alias to be added to core ruby, which matz approved; at a later time I found out that rails did have the same alias (I did not know that before).

Again, though, it is not me saying yes or no to that - just wanting to keep the discussion more open. :)

Is this a case where we might consider integrating this idea from Rails?

I suppose if it makes sense from a core ruby perspective, e.g. may benefit the ruby ecosystem. (Ruby is also used in non-rails areas.)

I feel like it's very much in the spirit of core Ruby, with attention to developer happiness.

Well, this is a bit difficult because you can make a language very complex, and keep on thinking that it makes developers happy. So we end up with C++ :D

It always surprises me how much I have to look up when I'm printing date/time as strings.

Yes, this part I agree with. I am just not automatically sure that the proposal of changing .to_s should be the one that addresses this, or the issue. (On a side note, for a similar reason I did not like the old global variables inspired
from perl; I always have to look them up and can never remember them. I do not use them much at all these days though.)

#3 - 02/17/2020 07:56 PM - sawa (Tsuyoshi Sawada)
Since the receiver is a Time object or Time, at least the time-part in strftime and strptime is redundant. That is for sure. Perhaps aliases like Time#strf, Time.strp may make sense.

#4 - 02/17/2020 09:46 PM - ttilberg (Tim Tilberg)
shevegen (Robert A. Heiler) wrote in #note-2:

I am not sure if this is a good suggestion though, largely because .to_s already having a distinct meaning, e. g. “to string” (or to a string representation).

People also typically associate .to_s, if there is an argument, with something like this:

37.to_s(2).rjust(8, "0") # => "00100101"

I feel that this directly supports the proposal in that when you call #to_s on a Time or Date object, you are casting this object to a string representation, which could be formatted whichever way. The fact that the current #to_s already includes a default strftime format further suggests that this feels like the right place. Finally, I feel that casting an integer to string, and being able to specify that it should be represented as binary further supports this cause.

I should note, I have no desire to introduce something that doesn't slot in naturally, but I think since this method previously did not take any args, it should be able to change without breaking anything, right? If there are edge cases where this may not be true, than I do not want to support #to_s(format) and instead would prefer an alternate name. However #format is already taken and marked private.

sawa (Tsuyoshi Sawada) wrote in #note-3:

at least the time-part in strftime and strptime is redundant. That is for sure. Perhaps aliases like Time#strf, Time.strp may make sense.

Oh my. Yes, this seems even more clear that the notion of using strftime (and strf) is unnatural (with exception to the fact that languages have used these terms forever. However, this is Ruby! We are people, not machines!). I find #strf equally unintuitive.

Personal anecdote: Whenever I see strftime I always think “string from time” first, and strptime as “string print time”.

Thank you for your comments so far, folks.