Documentation literacy as a metacognitive skill in computer programming

Dominic Bordelon, Research Data Librarian
University of Pittsburgh Library System

October 6, 2022

Quick terminology note

I will say “function” often in this presentation, e.g., “documentation tells us how a function works.” For the purposes of this argument, it could be interchanged with “method” and “class.”

The (anecdotal) search behavior of novice programmers

Scenario: learner recognizes an information need, because of 1) not knowing *how* to do something (which function/method to use?)

- Straight to Google
- Blog posts
  - Unpleasant and distracting UX; crucial version info may be unclear or missing (e.g., Python 2.x vs. 3.x)
- Tutorials
  - Same problems as blog posts; info presented in a dribble and may not go deep enough to answer the user's question
- (if they're lucky) Stack Overflow posts
  - large forum with trusted results

They don't know *how* to google for their programming information needs.

What they typically *don't* do: go the Help menu, press F1, search the official docs

As instructors, we often model mature ways of reasoning and effective strategies for *writing* code. But we should also model how to effectively formulate and answer questions about coding using API documentation.

What should we do instead?

Teach learners how to read and use API documentation and use help systems.

Why does this matter?

A poor information pathway contributes to extraneous cognitive load, which is already substantial while programming.

How to read documentation

Features of (good) API documentation:

- names of arguments and their default values
- data types of arguments → what kind of object does the function expect as input?
- return value → what kind of object will the object return to us?
- parameterized configurations → polymorphic behavior
- description of how the function works
- code examples
- references to other functions → expansion of mental model; encouragement to explore

But. . .

- users have to know it exists
- how to interpret API documentation is not immediately obvious (e.g., function signature)
- documentation is written in a technical style
- applicability to current situation is not always apparent

Example Python API documentation

https://docs.python.org/3/library/stdtypes.html#str.format

`str.`**`format`**`(*args, **kwargs)`

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces `{}`. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

```
>>> "The sum of 1 + 2 is {0}".format(1+2)
'The sum of 1 + 2 is 3'
```

See Format String Syntax for a description of the various formatting options that can be specified in format strings.

Note: When formatting a number (`int`, `float`, `complex`, `decimal.Decimal` and subclasses) with the `n` type (ex: `'{:n}'.format(1234)`), the function temporarily sets the `LC_CTYPE` locale to the `LC_NUMERIC` locale to decode `decimal_point` and `thousands_sep` fields of `localeconv()` if they are non-ASCII or longer than 1 byte, and the `LC_NUMERIC` locale is different than the `LC_CTYPE` locale. This temporary change affects other threads.

Changed in version 3.7: When formatting a number with the `n` type, the function sets temporarily the `LC_CTYPE` locale to the `LC_NUMERIC` locale in some cases.

Fig. 1. The entry for str.format() shows us the method's arguments, a concise description of how it works, a simple example, an important note, and version-specific information. Note that a novice will need help interpreting elements such as the arguments and determining which information is relevant to their situation.

Example R API documentation

https://www.rdocumentation.org/packages/ggplot2/versions/0.9.1/topics/geom_histogram

ggplot2 (version 0.9.1)

geom_histogram: Histogram

Description

`geom_histogram` is an alias for `geom_bar` plus `stat_bin` so you will need to look at the documentation for those objects to get more information about the parameters.

Usage

```
geom_histogram(mapping = NULL, data = NULL, stat = "bin",
    position = "stack", ...)
```

Arguments

mapping	The aesthetic mapping, usually constructed with `aes` or `aes_string`. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

Details

By default, `stat_bin` uses 30 bins - this is not a good default, but the idea is to get you experimenting with different binwidths. You may need to look at a

Fig. 2. The API documentation for geom_histogram (in the ggplot2 package for R) illustrates a different formatting as well as named arguments with default values.

How to use help systems

We are inured to [what we imagine to be] unhelpful help systems, and they feel particularly obsolete with the Internet.

But

- running `help()` on your function might provide the answer you need more quickly than Google
- alt-tabbing to your browser and searching the Web adds extraneous cognitive load
  - Googling requires the user to 1) switch applications, 2) run their search, 3) assess results with varying authorship and format, and then within a web page to 4) isolate content

Example Python help system

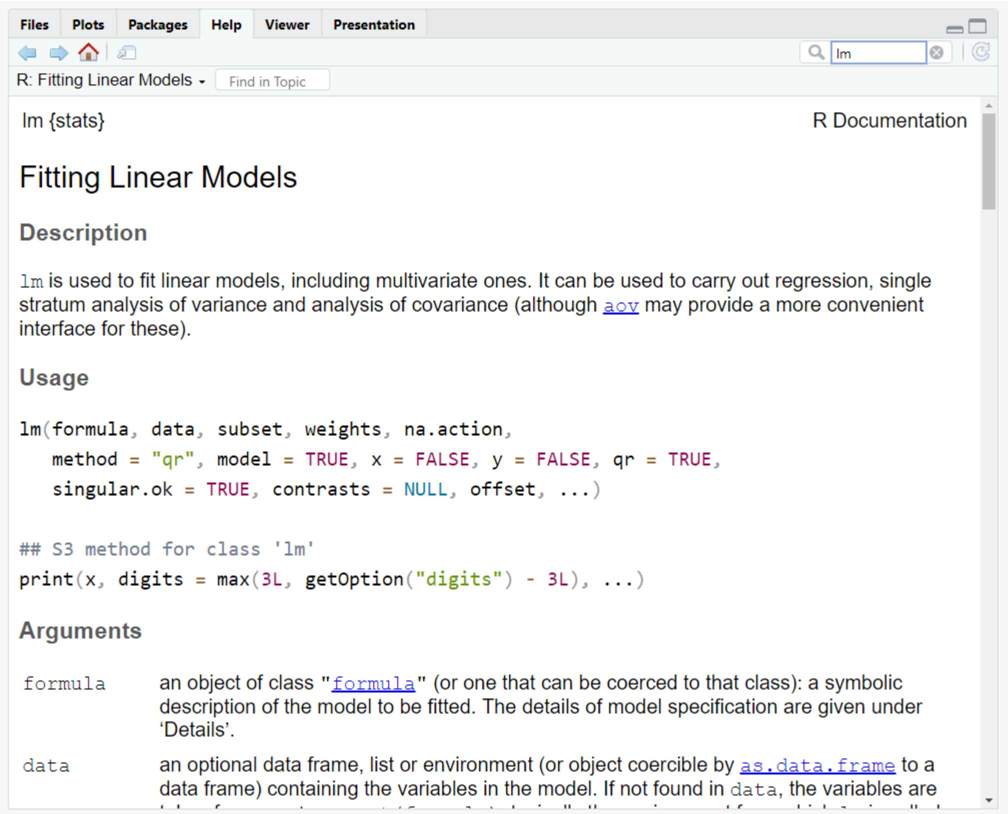
help(pandas.DataFrame) as run in a Jupyter notebook 6.4.12:

```
In [2]: help(pd.DataFrame)

Help on class DataFrame in module pandas.core.frame:

class DataFrame(pandas.core.generic.NDFrame, pandas.core.arraylike.OpsMixin)
 |  DataFrame(data=None, index: 'Axes | None' = None, columns: 'Axes | None' = None, dtype: 'Dtype | None' = None, copy: 'boo
l | None' = None)
 |
 |  Two-dimensional, size-mutable, potentially heterogeneous tabular data.
 |
 |  Data structure also contains labeled axes (rows and columns).
 |  Arithmetic operations align on both row and column labels. Can be
 |  thought of as a dict-like container for Series objects. The primary
 |  pandas data structure.
 |
 |  Parameters
 |  ----------
 |  data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
 |      Dict can contain Series, arrays, constants, dataclass or list-like objects. If
 |      data is a dict, column order follows insertion-order. If a dict contains Series
 |      which have an index defined, it is aligned by its index.
```

Fig. 3. The Python help system entry for pandas.DataFrame. Note that plain text is returned which is compatible with command-line interfaces.

Example R help system

[?lm](?lm) as run in RStudio 2022.07:



lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models, including multivariate ones. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)

## S3 method for class 'lm'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`formula` an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.

`data` an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are

Fig. 4. R's lm() documentation entry as viewed in RStudio's help pane.

Documentation literacy

- Scherer, Siddiq, and Sánchez Viveros (2020) categorize existing literature of teaching and learning computer programming as dealing with:
  - effectiveness of programming interventions *per se* (e.g., effects of learning programming on math or problem-solving)
  - effectiveness of visualization or physicality (e.g., Scratch, Arduino)
  - effectiveness of instructional approaches (e.g., pair programming, learner reflection)
- "Teaching programming through metacognition seems effective, and the metacognitive skills acquired during instruction may ultimately impact students' problem-solving performance and success." Scherer, Siddiq, and Sánchez Viveros (2020)
- Rum and Zolkepli (2018) applied metacognitive strategies such as planning and organizing, making a project timeline, troubleshooting issues, linking learning to prior knowledge in discussion, and self-reflection and self-assessment, and found a correlation with student success in teaching and learning computer programming.
- Documentation literacy is another metacognitive strategy or skill we could impart to students to support their learning and doing of computer programming.
- We can also think of documentation literacy as a specific kind of information literacy from a library science perspective.

Teaching example: Learner exercise

A handy function is `help()`, which queries R's documentation system. Most commonly, you'll look up functions. You can search by running `help(topic)` or `?topic`, e.g., `?sqrt`. Notice that the result will appear in the Help pane.

1. There is confusion among some R users what the "c" in the function `c()` stands for. Using the help system, what does `c()` do? What do you think "c" stands for?

c {base} R Documentation

Combine Values into a Vector or List

Description

This is a generic function which combines its arguments.

The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed.

Fig. 5. API documentation entry for c() in R.

2. Create a vector of arbitrary patient ages and store it as an object called `ages`.

   ```
   # answer code goes here
   ```

3. What do you estimate is the mean age? Calculate it using `mean()`.

   ```
   # answer code goes here
   ```

4. What is the mean value of the `Size` variable in `tg`? How about `numAge` in `cvdr`?

   ```
   # answer code goes here
   ```

Teaching example: Problem set section about missing values

In R, a missing value (equivalent to an empty cell in Excel—NOT to zero) is represented with `NA` (not available). You can't use it in calculations because its uncertainty taints any numbers it interacts with. Many times, the presence of `NA` is expected and fine; some variables are empty sometimes.

Try the code below:

```
my_values <- c(1, 0, 3, 4)
# predict: what will sum() and mean() of my_values be?
# calculate them below:


some_values <- c(1, NA, 3, 4)
# predict: what will sum() and mean() of some_values be?
# calculate them below:



# why does this happen?
# how can we fix this problem?
# (hint: run ?sum or ?mean and look in the Arguments section)

# can you fix the sum() and mean() function calls for some_values?
# (hint: if you're unsure of the syntax, run ?sum and check the Examples)
```

```
## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical evaluating to `TRUE` or `FALSE` indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Fig. 6. mean() function signature and arguments.

Examples

Run examples

```r
## Pass a vector to sum, and it will add the elements together.
sum(1:5)

## Pass several numbers to sum, and it also adds the elements.
sum(1, 2, 3, 4, 5)

## In fact, you can pass vectors into several arguments, and everything gets added.
sum(1:2, 3:5)

## If there are missing values, the sum is unknown, i.e., also missing, ....
sum(1:5, NA)
## ... unless  we exclude missing values explicitly:
sum(1:5, NA, na.rm = TRUE)
```

Fig. 7. Examples section of sum() documentation.

How to check whether a vector has any `NA`s? The `anyNA()` function:

```
anyNA(my_values)
[1] FALSE
anyNA(some_values)
[1] TRUE
```

There is a help file for missing values: `?NA`

Limitations of this approach

- These ideas are empirically based, but no hypotheses have been tested
  - I have a hunch this can be effective, but is it?
- API documentation is most usable when the reader knows already the name of the function they want to look up.
- API documentation is not always well written or up to date. (But we should still show learners how to use the manual, even if we don't think it's an ideal manual.)

API documentation might be better for:	Web search might be better for:
"What order do the arguments take?" "What are the names of the arguments?" "What is the default value of this argument? What is the default behavior of this function?" "Can `str.join()` only be used with lists, or also other kinds of iterables?"	"How do you turn a list of items into a single string?" (A novice will not think to search for `str.join()` and search functionality "What does this error mean?" (copy/paste it) "Functions A and B appear to do the same thing. Is that right? If yes, is there any advantage to one or the other?" "Is recent development x a known issue with function A?"

Table 1. Each approach can be advantageous for different information-need cases.

Conclusions

- Let’s emphasize the API documentation when we’re teaching programming languages and tools (people work hard on it!)
- As part of our computer programming instruction, let’s model effective web search practices—query formation, assessment of results, navigation within a document—and favor official API documentation where appropriate (e.g., once determining the name of the needed function).
- By walking novices through our own thought processes and strategies, which have formed from experience as practitioners, we can hope to transfer some of our skills and knowledge to them.

References

- The pandas development team. (2022). *pandas-dev/pandas: Pandas (v1.5.0)*. Zenodo. https://doi.org/10.5281/zenodo.7093122.
- R Core Team (2022). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.
- Rossum, Guido van, and Fred L. Drake. *The Python Language Reference*. Release 3.0.1 [Repr.]. Python Documentation Manual / Guido van Rossum; Fred L. Drake [Ed.], Pt. 2. Hampton, NH: Python Software Foundation, 2010.
- RStudio Team (2020). *RStudio: Integrated Development for R*. RStudio, PBC, Boston, MA URL http://www.rstudio.com/.
- Rum, Siti Nurulain Mohd, and Maslina Zolkepli. “Metacognitive Strategies in Teaching and Learning Computer Programming.” *International Journal of Engineering & Technology* 7, no. 4.38 (December 3, 2018): 788–94. https://doi.org/10.14419/ijet.v7i4.38.27546.
- Scherer, Ronny, Fazilat Siddiq, and Bárbara Sánchez Viveros. “A Meta-Analysis of Teaching and Learning Computer Programming: Effective Instructional Approaches and Conditions.” *Computers in Human Behavior* 109 (August 2020): 106349. https://doi.org/10.1016/j.chb.2020.106349.
- Sweller, John, Paul Ayres, and Slava Kalyuga. *Cognitive Load Theory*. 1st ed. Explorations in the Learning Sciences, Instructional Systems and Performance Technologies. New York Dordrecht Heidelberg London: Springer, 2011.
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.