2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)

Instrument Designs for Validating Cross-Language Behavioral Differences

Nischal Shrestha
NC State University
Raleigh, NC, USA
nshrest@ncsu.edu

Chris Parnin
NC State University
Raleigh, NC, USA
cjparnin@ncsu.edu

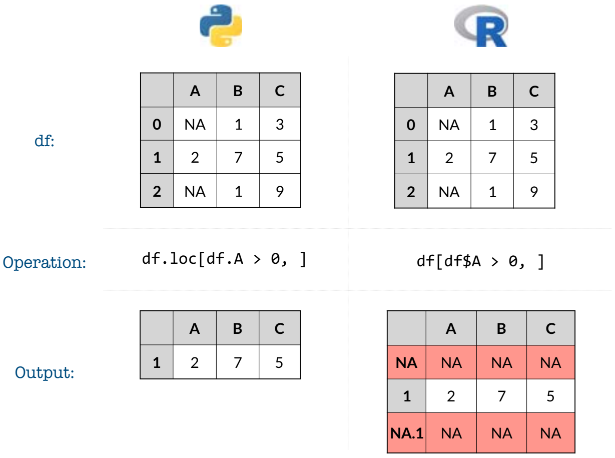
***Abstract*—Programmers are expected to use multiple programming languages frequently. Studies have found that programmers try to reuse existing knowledge from their previous languages. However, this strategy can result in misconceptions from previous languages. Current learning resources that support the strategy are limited because there is no systematic way to produce or validate the material.**

We designed three instruments that can help identify and validate meaningful behavior differences between two languages to pinpoint potential misconceptions. To validate the instruments, we examined how Python programmers predict behavior in a less familiar language like R, and whether they expect various R semantics. We found that the instruments are effective in validating differences between Python and R which were linked to misconceptions. We discuss design trade-offs between the three instruments and provide guidelines for researchers and educators in systematically validating programming misconceptions when switching to a new language.

I. Introduction

Modern programmers typically work on systems built with a cocktail of multiple programming languages [1]. A recent survey found that professional software developers have a mean of seven different programming languages inside their industrial software projects [2] and open-source software projects frequently have between 2–5 programming languages [3], [4]. Programmers are also expected to continue learning multiple programming languages on a daily basis. To learn a new programming language, studies have shown that programmers attempt to use a *cross-language* learning strategy by reusing knowledge from the previous language [5], [6], [7]. This can be a useful bootstrapping strategy but it can result in misconceptions being carried to the new language [6] or a negative transfer [8] of knowledge.

Learning resources like "X for Y Programmers" books, migration cheat sheets and online guides support cross-language learning. For example, consider Tidynomicon [9], a guide for learning R from the perspective of Python. The lessons for the guide were designed using a method called "backward design" [10] where the designer brainstorms topics and potential misconceptions they would like to cover. Brainstorming is a useful exercise for identifying misconceptions, but the resulting material may not cover the relevant misconceptions Python programmers actually hold. Programmers need training that can prevent negative transfer, but there are no systematic ways to produce or assess this material.


df:

	A	B	C
0	NA	1	3
1	2	7	5
2	NA	1	9

	A	B	C
0	NA	1	3
1	2	7	5
2	NA	1	9

Operation:

```
df.loc[df.A > 0, ]
```

```
df[df$A > 0, ]
```

Output:

	A	B	C
1	2	7	5

	A	B	C
NA	NA	NA	NA
1	2	7	5
NA.1	NA	NA	NA

Fig. 1. Subsetting using logical indexing in R can result in unexpected rows when NAs (missing values) are present.

To illustrate, imagine we wanted to add another example to Tidynomicon—how could we decide which misconception to warn Python programmers about? For example, an expression like `df[['A']]` references column A from dataframe `df` in both Python and R, a positive transfer. However, the return type of this expression is a vector instead of a dataframe in R, a negative transfer. This might surprise some programmers, but it is a minor mismatch of data type. Now consider the expression shown in Fig. 1, which returns rows for a dataframe where the column A value is positive. If the dataframe has rows where a column A is missing (`NA`), R will keep those rows while also converting the entire row to `NA`. This can be quite unexpected behavior for a Python programmer who is accustomed to rows with `NA` being gracefully handled. In both examples, the behavior of the expressions differ across languages, but perhaps some differences are more meaningful than others? While the topic of subsetting with `[` is covered in Tidynomicon, the unexpected behavior when using logical indexing is not mentioned. The second expression seems a more worthy addition as it is much more surprising for a Python programmer: *"what on earth is happening?"* (P10)

978-1-7281-0810-0/19/$31.00 ©2019 IEEE

In this paper, we present instruments that identify and validate meaningful behavior differences between programming languages, which can be linked to misconceptions or negative transfer between languages. The instruments evaluate when programmers make incorrect predictions about a target language or have unexpected reactions to the behavior of a code snippet. We explore three different instruments designs:

- Multiple Choice: Use misconceptions as the distractors
- Simple Yes/No: Confirm a misconception by asking if some code behavior is true
- Surprise: Ask if programmers *expect* code behavior

We conducted a small study with the three instruments and found some promising results that show that they are effective in capturing Python misconceptions when examining R code snippets. Based on the preliminary results, we identified design trade-offs between them and suggest future guidelines for researchers and educators on how to design effective instruments to systematically validate cross-language misconceptions.

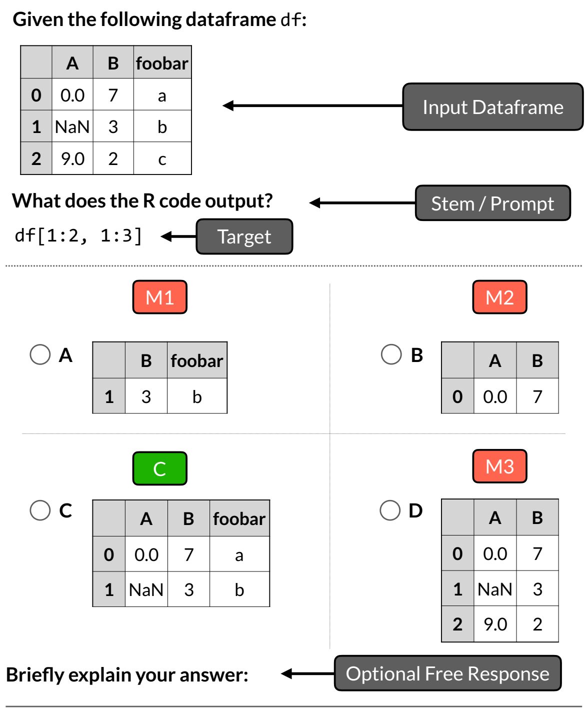
II. Methodology

A. Language Choice

We decided to ask questions about a subset of R from the perspective of Python for the following reasons: 1) Python's popularity is rapidly growing [11] and most programmers are familiar with it compared to R. 2) In the data science context, the code snippets for the two languages are terse yet incorporate numerous important concepts like indexing and filtering data. 3) Visualizing behavioral differences between Python and R code behavior is simple as the operations typically produce a dataframe output. In our instruments, we used output differences between Python and R code in addition to the syntactical differences to reduce "hidden state" [12] and ensure "perceptible information" [13]. Dataframes also provide a way to reduce an information barrier [14] as programmers can immediately check their prediction or expectation of R semantics. Although we picked Python and R for our study, it should be noted that the instruments we present can be reformulated for other language pairs such as static versus dynamic [15].

B. Population

We recruited 32 participants from a graduate Computer Science course at our University, sampling for participants with experience in Python, but less experience in R. Participants reported their experience with Python with a median of 4, on a 5-point Likert scale ranging from "Unfamiliar" to "Familiar", a median of 3 with the Pandas library, and a median of 3 with R. The participants took our first survey which involved the Multiple Choice and Simple Yes/No instruments. Then, to conduct a pilot study with the Surprise instrument, we recruited 13 Computer Science graduate students who met the same criteria and did not take the first survey. These participants reported their experience with Python with a median of 5, a median of 4 with Pandas and a median of 3 with R. The surveys are available online[1].



Given the following dataframe `df`:

	A	B	foobar
0	0.0	7	a
1	NaN	3	b
2	9.0	2	c

What does the R code output?

`df[1:2, 1:3]`

○ A

	B	foobar
1	3	b

○ B

	A	B
0	0.0	7

○ C

	A	B	foobar
0	0.0	7	a
1	NaN	3	b

○ D

	A	B
0	0.0	7
1	NaN	3
2	9.0	2

Briefly explain your answer:

Fig. 2. A Multiple Choice question with one correct (C) and three incorrect (M1-M3) answers which are possible misconceptions from Python. There is an optional free response to provide explanations for the selected answer.

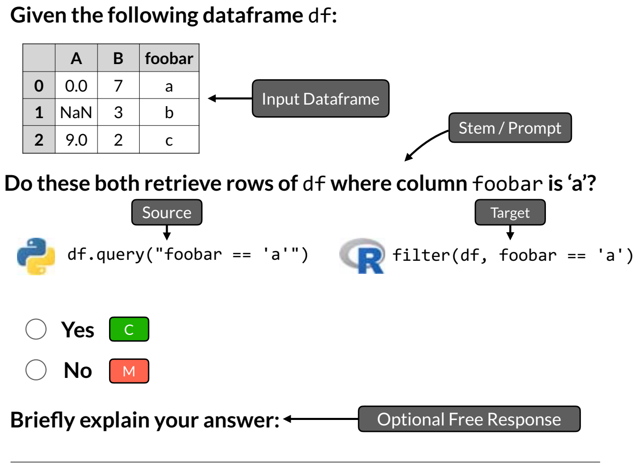
C. Multiple Choice and Simple Yes/No Survey

For our first survey, we wanted to test the designs of the Multiple Choice and the Simple Yes/No instruments. We created 10 total questions, 5 Multiple Choice and 5 Simple Yes/No. We interleaved the two formats on a single survey to prevent a potential tendency for the test-taker to answer all questions in the same manner. We reveal answers after programmers complete the survey which can help them understand or gain awareness of these differences.

1) Multiple Choice Design: Consider the example Multiple Choice question shown in Fig. 2. The use of `[` to index or subset rows of a dataframe is common to both Python and R, a positive transfer. However, R uses 1-indexing and the end index is inclusive when using `start:end` to define a range, a negative transfer. Given these differences, a number of misconceptions might occur: 1) R uses 0-indexing 2) The end index is exclusive for the `:` operator 3) The left side of the comma is column selection and the right side is row selection. However, it is not clear which of these misconceptions are the most problematic.

For the stem or question prompt, we present the input dataframe and ask the programmer to predict the R code output. There is one correct choice (C) with 3 distractors (M1-M3) representing misconceptions. Prior work has used concept inventory questions to reveal misconceptions for introductory programming [16]; we were interested in using a similar format to identify misconceptions when switching languages.

[1] https://github.com/alt-code/Research/tree/master/Misconceptions



Given the following dataframe `df`:

	A	B	foobar
0	0.0	7	a
1	NaN	3	b
2	9.0	2	c

Do these both retrieve rows of `df` where column `foobar` is ‘a’?

```
df.query("foobar == 'a'")
```

```
filter(df, foobar == 'a')
```

○ Yes C

○ No M

Briefly explain your answer:

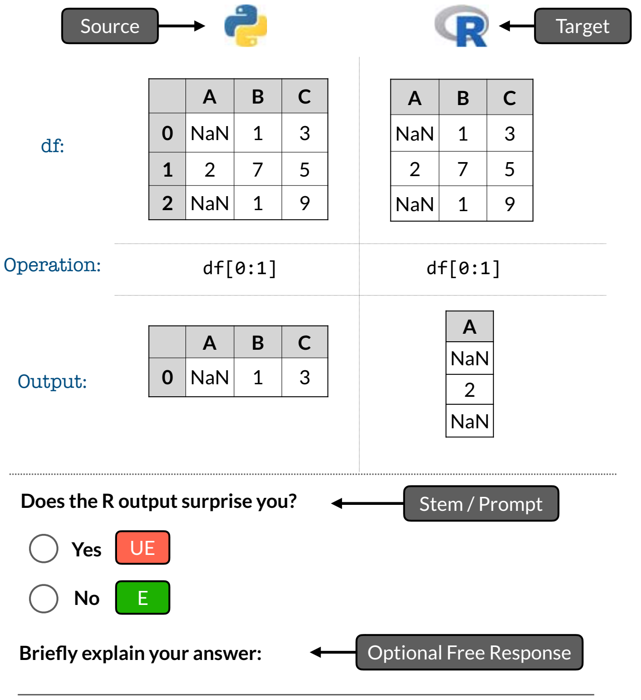
Fig. 3. A Simple Yes/No question. The equivalent code snippet for Python (Source) and R (Target) are shown. There is only one misconception M for this format. There is an optional free response to provide explanations for the selected answer.

Each distractor is the output that would be produced based on the misconception. For example, choice A (M1) represents the combination of the first and second misconception, that R is 0-indexed and the end index is exclusive for : operator. Choice B (M2) represents the second misconception while choice D (M3) represents the third misconception, flipping the row and column selection. The frequency with which programmers choose one of M1-M3 indicates which misconceptions are the most problematic for the particular operation. We also allow the option to provide an explanation behind the answer choice.

2) Simple Yes/No Design: The Simple Yes/No question is a binary version of Multiple Choice, designed to target smaller differences between Python and R. Consider the example Simple Yes/No question shown in Fig. 3 where we ask the programmer whether or not the code snippets that filter the dataframe on column A is equivalent between the two languages. Unlike the Multiple Choice, there are only two choices: 1) “Yes” indicates the programmer is able to map Python’s `query` function to R’s `filter` 2) “No” indicates confusion between the two. There is also the option to explain their answer choice which can provide insight into their confusion regarding smaller differences.

D. Surprise Survey

For our second survey, we designed and validated a third instrument called Surprise to assess whether or not a Python programmer expected certain R behavior. Past research [17] has shown that the level of surprise a person experiences is related to the difficulty of trying to integrate new information with the old. We wanted to investigate this notion of surprise in the context of programming language differences. We created 12 total Surprise questions and conducted a small pilot study with 13 graduate students. Some of the questions were reformatted from the first survey. We did this to examine whether or not different instrument designs on the same behavioral differences between Python and R can change the way programmers respond. Just like the previous two instruments, we reveal answers after programmers complete the survey.



df:

	A	B	C
0	NaN	1	3
1	2	7	5
2	NaN	1	9

A	B	C
NaN	1	3
2	7	5
NaN	1	9

Operation: `df[0:1]` `df[0:1]`

Output:

	A	B	C
0	NaN	1	3

A
NaN
2
NaN

Does the R output surprise you?

○ Yes UE

○ No E

Briefly explain your answer:

Fig. 4. A Surprise question. The Python (Source) and R (Target) code snippets and outputs are shown. There are two choices to indicate surprise: Expected (E) and Unexpected (UE). There is an optional free response to provide explanations for the selected answer.

Consider the Surprise question shown in Fig. 4. In this question, we ask the programmer whether or not they are surprised by the output of `df[0:1]` in R (Target), given the equivalent code and output in Python (Source). Unlike the previous two instruments, we present the correct dataframe output for Python and R. Here, we might expect the R output to be unexpected (UE) as there is no difference syntactically yet both the rows and columns don’t match Python’s output since R subsets columns by default. Additionally, there is an optional open response to further understand programmers’ expectations. Unlike the other two instruments, we do not reveal answers as the instrument is only designed to assess their reaction.

III. Preliminary Results

A. Multiple Choice and Simple Yes/No Survey

1) Simple Yes/No Questions: For the Simple Yes/No questions we wanted to validate smaller differences between equivalent Python and R snippets which we expected most participants to guess correctly. An average score of $\bar{x} = 78\%$ confirms the instrument was effective in validating small differences. The most problematic question was Q8 (69%) which asked about equivalent syntax to select columns in

Python and R: `df[['A', 'B']]` and `select(df, 'A', 'B')`. In the open responses, P15 expressed *"that R looks a bit goofy"* and another observed *"in R, it's not specified that 'A' is column or not"* (P19). We believe participants were confused because the Python code uses a bracket notation whereas R uses a function call.

2) Multiple Choice Questions: We expected most participants to perform worse on the Multiple Choice questions as they were designed to validate the larger meaningful differences between Python and R. The average score ($\bar{x} = 64\%$) was lower compared to the Simple Yes/No questions, validating larger meaningful differences. The distractors for each question also helped pinpoint the most problematic misconceptions. For example, most (61%) participants chose one of two misconceptions when asked about the output of `df[0:1]` on Q1: 1) The default subset operation selects rows 2) The end index is exclusive when using `:` to define a range. These two were more common than the third misconception, that the default subset operation selects columns but start and end are inclusive. The other most frequently missed question was Q7 which was similar to the NA example in Section I, where many (41%) participants chose the Python output.

B. Surprise Survey

We expected most participants to be less surprised on questions presenting smaller differences between Python and R compared to questions presenting larger differences. For the 5 questions with smaller differences, most (78%) participants were not surprised with the R code output. For the 5 questions on larger differences, most were surprised but less than expected (62%). We did not expect the lack of surprise (39%) on Q10 which compared `df.drop([A], axis=1))` and `select(df, -A)`, dropping a column in Python and R respectively. We inspected the open responses for those who weren't surprised despite a large difference in syntax. P5 explained that the *"- notation is often seen for negation, so I guess this makes sense after seeing it"* and P12 expressed that *"it's interesting how -A will be understood as removing A column in the df."* Based on these responses, we believe that participants made sense of the code behavior by observing that the output were the same.

IV. Discussion

A. Question formats affect the way programmers respond

Based on the results, we found that presenting the same code snippets in different formats can yield different results. As mentioned in Section III-A1, the lowest scoring question for the Simple Yes/No instrument was Q8 (69%) which contradicts the low surprise (16%) when it was reformatted as a Surprise question (Q1). We examined the open responses to Q1 and discovered that participants rationalized their answer choice when provided both Python and R code and output. For example, P3 said, *"Looks like the output is doing as the code specified"* and P6 expressed that the code snippets *"should work the same way (and it does)"*. On the other hand, most participants (78%) were able to correctly guess the R output for `df[1:2, 1:3]` on Q5 in the Multiple Choice, but only 53% of participants expected the output for the same snippet in a Surprise (Q3) format. We examined the open responses for those who were surprised and found them saying *"I would say the Python one surprised me more... Python is the language I use the least of the two"* (P4) and *"Pandas locators are magic and always surprise me when they work"* (P5). Presenting both the Python and R code and output can sometimes lead to further confusion, even regarding the source language. P4's statement suggests that it's also important to consider which language is actively used by the programmer if they are familiar with both, which can affect their response.

B. Instrument design trade-offs and guidelines

1) Multiple Choice and Simple Yes/No: The Multiple Choice and Simple Yes/No instruments are effective in assessing whether programmers can *predict* behavior in the target language. The Multiple Choice is useful for testing larger differences where multiple misconceptions are possible for a given operation. Multiple iterations of testing might be required to pinpoint common misconceptions and use them as distractors on subsequent tests. To test smaller differences, the Simple Yes/No is useful; however, a follow-up with some participants revealed that the questions were perceived as "trick questions", which might not warrant honest answers. The lowest scoring questions for both instruments indicate the most problematic behavioral differences between the two languages.

2) Surprise: The Surprise instrument is effective in measuring programmer's *surprise* regarding code behavior in the target language without the pressure of answering correctly. However, as discussed in Section IV-A, it might not provide the most meaningful responses due to potential interference between the languages [18]. The most problematic differences can be identified by examining questions which have the highest percentage of surprise.

C. Automatic identification of cross-language differences

The behavioral differences between Python and R were hand-picked which is a time-consuming process. We have started investigating ways to automatically detect inconsistencies between the languages. Using a Kaggle competition [19] as the corpus, we found discrepancies in behavior between Python and R code by finding differences and partial matches of dataframes, given various inputs. We think this technique is promising for efficiently identifying discrepancies which can be validated using one of our instruments.

V. Conclusion

In this paper, we presented three instruments that can help identify and validate meaningful behavior differences between two languages to pinpoint potential misconceptions. We conducted a small study and found promising results in validating differences between Python and R, which were linked to misconceptions. We discussed some design trade-offs for each instrument and suggest guidelines for future researchers and computer science educators.

ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Grant No. 1755762.

REFERENCES

[1] H.-C. Fjeldberg, "Polyglot programming," Ph.D. dissertation, Master thesis, Norwegian University of Science and Technology, Trondheim/Norway, 2008.

[2] P. Mayer, M. Kirsch, and M. A. Le, "On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers," *Journal of Software Engineering Research and Development*, vol. 5, no. 1, p. 1, Apr 2017. [Online]. Available: https://doi.org/10.1186/s40411-017-0035-z

[3] F. Tomassetti and M. Torchiano, "An empirical assessment of polyglot-ism in github," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14. New York, NY, USA: ACM, 2014, pp. 17:1–17:4. [Online]. Available: http://doi.acm.org/10.1145/2601248.2601269

[4] P. Mayer and A. Bauer, "An empirical analysis of the utilization of multiple programming languages in open source projects," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:10. [Online]. Available: http://doi.acm.org/10.1145/2745802.2745805

[5] Q. Wu and J. R. Anderson, "Problem-solving transfer among programming languages," Carnegie Mellon University, Tech. Rep., 1990.

[6] J. Scholtz and S. Wiedenbeck, "Learning second and subsequent programming languages: A problem of transfer," *International Journal of Human–Computer Interaction*, vol. 2, no. 1, pp. 51–72, 1990.

[7] N. Shrestha, T. Barik, and C. Parnin, "It's Like Python But: Towards Supporting Transfer of Programming Language Knowledge," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2018, pp. 177–185.

[8] D. N. Perkins, G. Salomon, and P. Press, "Transfer of learning," in *International Encyclopedia of Education*. Pergamon Press, 1992.

[9] G. Wilson. (2018) The Tidynomicon: A Brief Introduction to R for Python Programmers. [Online]. Available: https://gvwilson.github.io/tidynomicon/

[10] G. Wilson. (2018) A Lesson Design Process. [Online]. Available: http://teachtogether.tech/#s:process

[11] P. Guo, "Python is now the most popular introductory teaching language at top us universities," *BLOG@ CACM, July*, vol. 47, 2014.

[12] B. Victor. (2012) Learnable programming. [Online]. Available: http://worrydream.com/LearnableProgramming/

[13] M. F. Story, J. L. Mueller, and R. L. Mace, "The universal design file: Designing for people of all ages and abilities," 1998.

[14] Andrew J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *2004 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sep. 2004, pp. 199–206.

[15] A. Pang, C. Anslow, and J. Noble, "What programming languages do developers use? a theory of static vs dynamic language choice," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2018, pp. 239–247.

[16] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Computer Science Education (SIGCSE)*, 2010, pp. 107–111.

[17] R. Maguire, P. Maguire, and M. T. Keane, "Making sense of surprise: An investigation of the factors influencing surprise judgments." *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 37, no. 1, p. 176, 2011.

[18] R. E. Slavin and N. Davis, "Educational psychology: Theory and practice," 2006.

[19] "Titanic: Machine Learning from Disaster," https://www.kaggle.com/c/titanic.