A Common Framework for Scripting Tutorials

Erik Hjelmås and Ivar Farup

Abstract

A scripting tutorial should focus on teaching the characteristics of scripting languages, avoiding the pitfall of teaching only syntax conversion from system programming languages. Examples should be selected to emphasize features of scripting languages in general (e.g. weak typing, associate arrays, "gluing") and the scripting language at hand. Students might benefit from using the same framework (structure of programming elements and examples) for learning multiple scripting languages. Their existing programming skills should be reused while newly introduced scripting concepts should be reinforced in subsequent examples.

1 Introduction

Scripting languages are popular for solving problems in many different domains [8]. Some are general scripting languages such as Perl, Python and Ruby, while others are slightly domain-specific (platform-specific: Bash, PowerShell) or very domain-specific (application-specific: Cfengine, Matlab). They can be characterized as high-level and weakly typed, as opposed to system programming languages such as Java and C++ [11]. In higher education, scripting languages are used both for introductory programming courses [14] and for lab exercises in later courses. It is in this last context the present work seeks to contribute to a better learning experience for students. Although much effort has been invested in researching ways of teaching introductory programming [12], the problem of how to convert existing programming skills to a new language seems slightly overlooked. The way this is solved in practice is by either providing a short tutorial style introduction to the new language, or referring the student to an online tutorial resource. A tutorial can be described as [16]:

> A tutorial is one method of transferring knowledge and may be used as a part of a learning process. More interactive and specific than a book or a lecture; a tutorial seeks to teach by example and supply the information to complete a certain task. Depending on the context a tutorial can take one of many forms, ranging from a set of instructions to complete a task to an interactive problem solving session (usually in academia).

Students with some programming background should be able to quickly get going in a new language. Quickly meaning half-a-day (four teaching hours, 45 minutes each) tutorial session. Student's programming skills depend on the number of programming courses they have completed, on how much time he or she has invested in actual

This paper was presented at the NIK-2011 conference; see http://www.nik.no/.

programming practice and how many programming languages he or she has encountered. The prerequisites of a scripting tutorial session in the second half of a study program must require that students already have aquired programming skills. The transition to a new programming syntax in a short tutorial will be too challenging if the student has not passed the hurdle of feeling comfortable with basic programming constructs. The learning outcome of a half-a-day scripting tutorial can be formulated as:

> The student should be able to apply the programming syntax of the new scripting language to existing programming skills. The student should be able to describe the basic characteristics of the new scripting language and its typical application domain/domain of use. The student should be able to find the relevant help and documentation to broaden his/her knowledge of the scripting language further.

There are a plethora of excellent scripting tutorials available on the Internet, but we have not found tutorials that cover multiple languages based on the same structure and examples. In this paper we propose such a structure, including discussion of issues and concrete recommendations.

In the following section we will describe the characteristics of some of the existing scripting tutorials and form the basis of our framework. In sections 3 and 4 we present the selection process of the included programming elements and examples. In section 5 the current implementations of the framework are described which is further discussed in section 6, and with concluding remarks in section 7.

2 Survey of scripting tutorials

We have selected five scripting tutorials, one for each of the three most popular general scripting languages according to `tiobe.com`: PHP [3], Python [9] and Perl [5], and two platform-specific ones: Bash [7] and PowerShell [13]. The selection criteria was threefold: the tutorial should require exisiting programming skills, most of the content should be possible to cover in less than a day, and the content should be quite general (not domain specific). In addition, the tutorials should appear to be quite popular (meaning they should appear on the first couple of pages of a Google search).

Each tutorials presentation sequence of classical programming topics is shown in table 1. In addition, the characteristics of each tutorial can be described as:

PHP [3] There is a separate part early on assignment, arithmetic, comparison and logical operators. In other words, operators are addressed as a separate topic. Associative arrays (and arrays in general) are not treated together with variables, addressed later in the tutorial.

Python [9] A top-down approach is applied, starting with two examples first demonstrating many Python (and scripting in general) features. Conditionals are not introduced. It includes a discussion on Pythons treatment of strings vs numbers.

Perl [5] This tutorial introduces variables and containers first, but leaves associative arrays to after control structures. The final chapter is devoted to regular expressions.

Bash [7] Associative arrays where not introduced in Bash before 2009 so it is not included in this tutorial. Similar to the Python tutorial it starts out with a simple example demonstrating syntax followed by an example demonstration scripting features (in this case, “gluing”).

Programming element	PHP	Python	Perl	Bash	PowerShell
Intro/Syntax	1	1	1	1	1
Variables/Containers	2	3	2	2	2
Conditionals	3		3	4	3
Loops	4	2	4	5	4
Input/Output		5		3	
Subroutines/Functions/Procedures	5	4	6	5	5
Classes/Objects		6			
Libraries/Modules		7			
Regular expressions			5		

Table 1: The sequence of programming elements in a few selected tutorials.

PowerShell [13] Since command line shells are less of an issue on Windows platforms compared to Unix/Linux platforms, much material in the start of the tutorial is devoted to the use of the PowerShell, before scripting is addressed. Associative arrays are called hash tables and introduced early, followed by control structures and functions, before quickly moving on to windows specific topics.

Naturally, these tutorials differ in coverage and sequence based on both the preference of the author and the intended audience. However they all cover some topics that are characteristic of scripting languages: weak typing and associative arrays (except the Bash tutorial since it is a quite new feature of Bash).

3 Characteristics of scripting languages

Scripting languages are high-level languages where focus is on high programmer productivity rather than fast program execution time [8]. Programmers can do more with less in scripting languages, but scripting languages should be considered supplementary to system programming languages since they are commonly used for "gluing" together components written in system programming languages. Scripting languages have weak typing, meaning whether variables are declared before use or not is optional, and common practice is to not declare variables: leaving it to the interpreter (dynamic typing). This encourages reuse of components since strongly typed interfaces are avoided [11]. Some scripting languages (e.g. Perl) also have operators that behave differently based on context. E.g. they treat variables as different types dependent on the values used in an assignment construct.

The scripting languages that are "Shell-based" such as Bash and PowerShell are integrated with command-line usage, making use of environment variables and piping. This makes input/output very well accommodated, since piping is an easy way of creating communication channels between components ("glued together") and also makes reading or writing to files easy.

Some programming constructs are more widely used in scripting languages than in system programming languages. Among containers, the associate array has always been a key element for script programmers. This is due to the application domain, with scripting languages being used for text processing and parsing of object collections. It is also a consequence of weakly typed languages since weak typing makes mixing types in the same array easy. With widespread use of associate arrays, the foreach loop is also a typical feature of scripting languages, where looping an object collection is done without

any test condition for loop termination. Being a text/object parsing and gluing language, a scripting languages also has support for *here documents* (also called heredoc, here-strings, hereis, here-script) [15].

4 Examples

Examples play a slightly different role in scripting tutorials than in introductory programming courses. While focus on learning to program is the most important in programming courses, focus is on new syntax and demonstration of scripting specific features in scripting tutorials. However, some characteristics of good introductory examples also apply to scripting tutorials. Börstler [2] defined ten quality factors for examples in introductory textbooks on object-oriented programming. Five of these can also be applied to scripting examples (the remaining five are specific to learning object-oriented programming):

T1 - Correctness and completeness The code is bug free and the example is sufficiently complete.

T2 - Readability and style The code is easy to read and follows a consistent formatting and style.

D1 - Sense of purpose Students can relate to the example's domain and computer programming seems like a reasonable way to solve the problem.

D2 - Process An appropriate programming process is followed/described.

D3 - Well balanced cognitive load Explanations and supporting materials promote comprehension; they are neither simplistic nor do they impose extraneous cognitive load.

T1 and T2 are related to "technical quality", and D1, D2 and D3 are related to "didactic quality". In addition, we define the following quality factors specific to examples in scripting tutorials:

S1 - Efficiency Since scripting languages are interpreted, sometimes with frequent use of sub processes, examples should be as efficient as possible.

S2 - Scripting related Examples should focus on the characteristics of scripting languages mentioned in section 3.

S3 - Reinforcement Focus in examples should be on showing differences to what is already know, while reinforcing everything that is new. New programming constructs are always challenging to understand. Repeating these new programming constructs (reinforcement learning) in multiple examples improves the learning process.

S4 - Security This is not specific to scripting, but should be a characteristic of any piece of code.

Examples should be selected with these criteria in mind.

The earliest examples in a tutorial introduce variables/containers (compound variables). Section 2 showed that the associative array is sometimes introduced together with variables in the early part of a tutorial and sometimes delayed to after control

```
email = {'Erik': 'erikh@hig.no',          # create dictionary
         'Frode': 'frodeh@hig.no'}        # with two items
email['Ivar'] = 'ivarf@hig.no'            # expand dictionary
print len(email)                          # prints "3"
email.pop('Frode')                        # remove one item
print len(email)                          # prints "2"
```

Figure 1: Python example showing the use of an associative array (called dictionary in Python) with dynamic expansion and deletion.

```
infile = open('emails.txt', 'r')          # open file for reading
for i in range(10):                       # loop with fixed range
    line = infile.readline()              # read (10 first lines)
    line = line.strip('\n').split(' ')    # parse
    email[line[0]] = line[1]              # expand dictionary
print len(email)                          # prints "12"
for k, v in email.iteritems():            # print entire dictionary
    print k, v                            # (no particular order)
```

Figure 2: Python example showing a loop with a fixed range. The scrip reads the ten first lines of the file (without checking), parses them and expands the array already created in Figure 1. Finally, the entire assosiative array is printed.

structures. Figure 1 introduces the associative array (called dictionary in Python) and demonstrates three characteristics of scripting languages (D1, S2):

- No declarations are needed (reinforcement of dynamic typing – S3).
- Automatic array expansion (reinforcement of automatic memory management – S3).
- Easy use of high-level data structures such as the associative array (the new concept – S2).

Figure 2 shows how a loop can be implemented (in Python) without testing a condition on every iteration. The example script reads the ten first lines of a formatted file with names and corresponding email addresses, parses the line, and add the content to the assosiative array created in the previous example. Finally, the entire dictionary is printed using a construct similar to the foreach loop. The script demonstrates the following characteristics (D1):

- Automatic expansion of dynamic arrays (reinforcement – S3)
- No testing in loop (new concept – S2)
- File operations: opening and reading (new concept – S2)
- String operations: srtripping and splitting (new concept – S2)
- The foreach loop (new concept – S2)

Figure 3 is an example of “gluing” (D1, S2) by using the existing powerful and widely used `wget` program to build a small program which is typically executed from

the command line like this:

```
$ ./mywebsearch.bash '(SSD|ssd)' < urls.txt
```

This program takes a regular expression as a command line argument, downloads web pages from urls in a file fed from STDIN and search the web pages for lines using the regular expression. This example program uses characteristic scripting languages features: easy input from command line, here document, automatic declaration and expansion of an array, use of special variables, regular expressions and gluing (use of `wget` and `cat`). This small script also raises numerous programming questions (D2, D3):

- Security (S4): are the temporary files uniquely named? are you deleting the same files that was created (Time-Of-Check-to-Time-Of-Use - TOCTOU issues can be discussed)? what happens if you download an extremely large file? what happens if the file name contains `http://something;rm -rf /`? is it safe to trust the path and call `wget` (and `cat`) or maybe better to call `/usr/bin/wget` (even though this makes this less portable)? There is no input validation at all in this script, so it is not robust, but complete input validation in examples can destroy the simplicity of the examples. Robustness should many times be discussed related to examples instead of included.

- Efficiency (S1): run this just a few times and you will quickly realize that you need some caching to avoid downloading the same files over and over, it quickly gets annoying. Is it really necessary to use two loops, maybe they should be joined to one?

- Usability: this quickly provides interesting information and a number of questions regarding how to present this: should we launch a web browser? use a dialog box? just format nicely in the terminal window?

This example can be used as an introductory example or as an summary example, dependent on whether the tutorial has a top-down or bottom-up approach. When used as a summary example it reinforces (S3) a large set of features of scripting languages.

5 Proposed framework

A framework for introductory scripting tutorials should be practical with plenty of examples, but also easy to use as basic reference material since the purpose is learning syntax and special features of the language at hand (and scripting languages in general) more than teaching students basic programming. Many student are scared of new languages. A tutorial document will be their first encounter with a new language and needs to be as comfortable and motivating as possible.

Figure 4 shows the structure of the proposed framework. The introduction should include practical information about the tutorial: which tools/virtual machines needed, and how the examples can be retrieved from the internet (`wget` or similar) in order to avoid wasting time typing in all the examples or copying/pasting. Copying/pasting from slides also many times cause character mapping problems, e.g. with characters embracing string literals such as quotation marks. Following the practical information there should be a short introduction to the origin and history of the language, before a simple introductory script (e.g. “hello world”) is presented with an practical demo of how it is compiled/interpreted and executed. This simple script should also serve to present the basic syntax of the language. It might be good to also mention relevant debugging/helper

```
while read line; do                          # line by line from STDIN
   url[$i]=$line
   tempfilename[$i]="$RANDOM.tmp"            # create random file name
   echo "retrieving $line to ${tempfilename[$i]}"
   wget -q -O "${tempfilename[$i]}" $line # download webpage
   ((i++))
done
echo "***** Relevant hits *****"
for i in ${!tempfilename[@]}; do             # each downloaded webpage
   while read line; do                       # parse line by line
      if [[ $line =~ $1 ]]; then             # match CLI-arg regexp
         echo -n ${BASH_REMATCH[0]}          # print the match
         if [[ ${url[$i]} =~ http://([^/]+) ]]; then
            echo " from ${BASH_REMATCH[1]}" # print domainname
         fi
      fi
   done <<< "$(cat ${tempfilename[$i]})"     # feed with here document
   rm ${tempfilename[$i]}                    # clean up
done
```

Figure 3: A glue example in Bash.

tools, but since this is a tutorial session, it is important to quickly get started with the actual programming contents. Dependent on the pedagogical preference between top-down and bottom-up approaches, further introductory examples might be given to show new and motivating features of the language at hand and scripting in general.

A natural starting point for any programmer moving into a new language is *variables* (or containers in general). Variables in this context is a broad term under which we also can introduce arrays and compound data types. Even though we might use some *input/output* in the examples used in the variables section (which is not a problem since we are teaching programmers), it should follow as a separate topic after variables *if the scripting language is shell-oriented*. Input/output is of special importance in many scripting languages since scripts are very often used in command line pipes, or used for parsing (structured) text from files or data that is output from system commands. It is also appropriate to introduce the here document in this context since it is a program constructs mostly found in scripting languages.

The classical control structures for branching (*conditionals*) and looping (*loops*) are if/else, switch/case, for, while and do. In addition to covering the scripting language specific syntax for these statements, it is important to emphasize the operators used in conditional statements. A scripting language can offer several operators not commonly found in system programming languages, and operators differ significantly in both syntax and semantics. Weight should also be put on foreach statement since it is commonly used with associative arrays. Piping or feeding data from system commands into the script was introduced in the input/output section, but should be reinforced at this stage by combining this with a loop statement (applies especially for shell-oriented languages).

We recommend that in such a "getting started" tutorial not too much weight should be placed on *functions*/subroutines as scripts tend to be short by nature, but this also depends of the intended audience. If the tutorial is intended for a general scripting language and for upcoming programming projects on a larger scale, spending more time on functions might be a good idea. This is also a good place in the structure to introduce the available operators for arithmetic (*math*) because the level of arithmetic support varies a bit and

```
Introduction and Syntax
Variables/Containers
    single/scalar
    array/sequence/list
    associative array
    structures
    special variables
Input/Output
    user
    pipe
    file
    system commands
    here document
Conditionals
    if/else
    operators and precedence
    numerical vs string comparison
    boolean precedence
    switch/case
Loops
    for
    while
    do
    foreach
Math
Subroutines/Functions/Procedures/Methods
    parameter passing
Classes/Objects
Libraries/Modules
Regular expressions
    command line
GUI
Reference material
Case
    analyze for efficiency
    analyze for security
    analyze for usability
Credits
```

Figure 4: The proposed framework structure.

those languages with limited support can implement extended support in a function as the subsequent example.

If the language has support for objects, it can be introduced at this stage or earlier, again dependent on the pedagogical preference of top-down vs bottom-up.

When starting to use a scripting language, students typically encounter regular expressions (regexps/regexes) in practice for the first time. This topic, if previously unknown to the audience, need to be addressed with sufficient space since it is both new and of high importance in the common application domain of many scripting languages. However regular expressions is many times separated from the language tutorial as a tutorial by itself.

After concluding the lecture with a brief introduction to how GUIs can be used, and a section with pointers to more comprehensive reference material, a good trick to make sure everyone gets going is to jointly solve a small case in iterations. This might also be the point in time where students start applying their programming skills outside class. At this stage it is very important to make sure they stick with their taught habits of good programming practice, and keep thinking efficiency, usability and security [1] in every program/program statement they write.

The proposed framework is at the time of this writing implemented for the languages Bash, Perl and PowerShell. The implementations consists of slides, slides with lecture notes, and downloadable examples. They are available from http://blog.hig.no/erikh/category/scripting/.

6 Discussion

Many times scripting languages are introduced in specific domains such as web applications, scientific computing or platform-specific system administration. The usefulness of a common framework depends somewhat on how the intended learning outcome balances between domain specific learning and scripting language learning. Students who will encounter multiple scripting languages might benefit significantly from a common framework where the domain-specificity is adjusted based almost solely on the chosen examples. A common framework might take away the "new language fear" since students know what to expect and can much more easily compare and contrast scripting languages. It can also be argued that an introductory tutorial should always be quite general since students always need to know something about the language at hand before applying it to their specific domain.

The authors have not performed any formal assessment on the learning outcome of using a common framework for several scripting languages, but some indications on the usefulness have been registered. During the spring semester 2011 at Gjøvik University College, 36 students were introduced to Bash and PowerShell in the course Operating Systems, and 7 out of these 36 were also introduced to Perl in a parallel course (Databases and Application Administration). All tutorials based on a draft of the framework in figure 4. During the qualitative assessment of these courses several students commented that they found a common framework for learning these scripting languages useful.

Authors of scripting tutorials have the benefit that students already know programming. They also have the benefit that scripting languages commonly avoid many of the topics students find the most difficult in programming, such as pointers, memory management and virtual functions [10]. This gives us some flexibility when choosing tutorial structure (top-down and bottom-up approaches). Introductory examples can be chosen to demonstrate exiting and motivating scripting language features knowing that students al-

ready are comfortable with basic programming constructs. However, the pitfall of over coverage of material must always be avoided. Tutorials are not similar to extensive textbooks, the focus is to get going in a practical way, staying clear of too much detail and material in general.

It is also important to try to make a tutorial into a *programming text* more than a *program text*, examples should not just be presented assuming students will write programs similar to the examples. Examples should be presented together with an appropriate programming process [4]. Again, this must be carefully considered in such a way that the tutorial is not too extensive, but at the same time the tutorial should reinforce good programming practice more than just introduce syntax.

Good scripting tutorials are important to make programming even more accessible, so that students do program a lot because its easy and fun [6].

7 Conclusions and future work

This article is meant as an aid for anyone who would like to teach a scripting language as a second language for students. A common framework with a structure that is easily recognized by students, together with carefully chosen examples, should provide the grounds for an enjoyable learning experience for students.

Future work includes formal assessment on usefulness of a common framework, and studying the proposed framework in the context of additional scripting languages ((Visual)Basic, JavaScript, Ruby, Lua). More experience with teaching based on this framework should also provide more insight into how much material that should be covered in a half-a-day tutorial session.

8 Acknowledgments

The authors would like to thank Alexander Berntsen for much useful critique of the Bash tutorial implementation and good scripting practice in general and Frode Haug for providing comments on the initial drafts of the framework.

References

[1] BISHOP, M. A Clinic for "Secure" Programming. *IEEE Security and Privacy 8* (March 2010), 54–56. doi:http://dx.doi.org/10.1109/MSP.2010.62. 9

[2] BÖRSTLER, J., NORDSTRÖM, M., AND PATERSON, J. H. On the Quality of Examples in Introductory Java Textbooks. *ACM Transactions on Computing Education 11* (February 2011), 3:1–3:21. doi:http://doi.acm.org/1921607.1921610. 4

[3] CODINGUNIT.COM. PHP Tutorials, 2011. [Online; accessed 07-July-2011]. 2

[4] GRIES, D. What Have We Not Learned about Teaching Programming? *IEEE Computer 39* (October 2006), 81–82. doi:10.1109/MC.2006.364. 10

[5] KUHN, B. M. Picking Up Perl, Edition 0.12, A Freely Redistributable Perl Tutorial Book, 2002. [Online; accessed 07-July-2011]. 2

[6] KUMAR, D. REFLECTIONS: Language wars and false dichotomies. *ACM Inroads 1* (August 2010), 10–11. doi:http://doi.acm.org/10.1145/1835428.1835431. 10

[7] LINUXCONFIG.ORG. Bash scripting Tutorial, 2011. [Online; accessed 07-July-2011]. 2

[8] LOUI, R. P. In Praise of Scripting: Real Programming Pragmatism. *IEEE Computer 41* (July 2008), 22–26. doi:10.1109/MC.2008.228. 1, 3

[9] MATLOFF, N. A Quick, Painless Tutorial on the Python Language, 2010. [Online; accessed 07-July-2011]. 2

[10] MILNE, I., AND ROWE, G. Difficulties in Learning and Teaching Programming - Views of Students and Tutors. *Education and Information Technologies 7* (March 2002), 55–66. doi:10.1023/A:1015362608943. 9

[11] OUSTERHOUT, J. K. Scripting: Higher-Level Programming for the 21st Century. *IEEE Computer 31* (March 1998), 23–30. doi:10.1109/2.660187. 1, 3

[12] PEARS, A., SEIDMAN, S., MALMI, L., MANNILA, L., ADAMS, E., BENNEDSEN, J., DEVLIN, M., AND PATERSON, J. A Survey of Literature on the Teaching of Introductory Programming. *SIGCSE Bull. 39* (December 2007), 204–223. doi:http://doi.acm.org/10.1145/1345375.1345441. 1

[13] POWERSHELLPRO.COM. Windows PowerShell Scripting Primer, 2007. [Online; accessed 07-July-2011]. 2, 3

[14] WARREN, P. Teaching programming using scripting languages. *Journal of Computing Sciences in Small Colleges 17* (December 2001), 205–216. 1

[15] WIKIPEDIA. Here document — wikipedia, the free encyclopedia, 2011. [Online; accessed 10-July-2011]. 4

[16] WIKIPEDIA. Tutorial — wikipedia, the free encyclopedia, 2011. [Online; accessed 10-March-2011]. 1