Typesetting and highlighting Unicode source code with LaTeX: a package comparison

Roberto Giacomelli, Gianluca Pignalberi

Abstract

LaTeX offers its users several packages to typeset and highlight source code, ranging from the very basic one (verbatim) to the almost complete one (listings). How do they perform when their input is a Unicode source code? This paper addresses this question with focused tests and tries to lead readers to a choice, be it either definitive or case-driven.

Sommario

LaTeX offre ai suoi utenti diversi pacchetti per comporre ed evidenziare del codice sorgente; questi pacchetti vanno dal più spartano (verbatim) a quello pressoché completo (listings). Come si comportano quando il loro input è un sorgente Unicode? L'articolo risponde a questa domanda con dei test specifici e cerca di condurre i lettori a una scelta, sia essa definitiva o da prendere di volta in volta.

1 Introduction

Global world needs a global approach to languages. Computer programs are no exception: they have local versions that should run smoothly on every computer. Documenting those programs and the corresponding source code for local communities should be as straightforward as it is running them. So it is extremely important that typesetting programs, and especially TeX, manages different languages and different alphabets.

In the TeX world multilingual support is very well developed: fonts representing non-Latin alphabets are included and supported, babel and polyglossia support a lot of languages with specific hyphenation and typographic rules; inputenc supports a nearly-Unicode input; XeLaTeX and LuaLaTeX are Unicode-compliant; the Junicode font (Baker, 2014), distributed with TeX, is an Open Type font directly usable with XeLaTeX/LuaLaTeX and is an invaluable tool for medievalists. Linguists who write documents containing "exotic" alphabets are indeed very well supported. Can computer scientists and engineers expect such a support when typesetting source code? This paper aims at addressing such a question.

After a short introduction on the Unicode standard presented in section 2, in section 3 we list the packages considered for this comparative test and talk about a couple of external tools (`pygments` and `fold`) that can (or have to) support some of the listed packages. Section 4 reports on the two small source codes (Unicode-encoded) used as typesetting benchmark. The final section 5 reports on the results of our test examining in detail every typeset code and summarizing the same results in a synoptic table.

2 The Unicode World

2.1 Why is Unicode so important?

Human kind developed thousands of languages and tens of thousands of symbols to write them—the letters.[1] Up to some years ago, computers could handle 128 or 256 characters at a time thanks to character codes like ASCII or EBCDIC. Those codes were born when memory, storage devices, processors and, therefore, computers and computation time were extremely expensive and precious.

Then Unicode came—a consortium that aimed at unifying all of the alphabets on earth and at creating *the* standard. It is impossible to represent tens of thousands of letters, ideograms, numbers and symbols with a narrow code like ASCII, so Unicode encodes characters with 8 bits or more. Needless to say, this standard needs more memory and computation resources than the past standards, but modern computers are powerful enough to quickly manage Unicode.

Unicode also offers a unique way to help linguists to write their multilingual documents and computer developers to write their international programs without worrying about local characters codes: every character has a code number not shared by any other character. The first 128 characters are the same of ASCII (i.e., Unicode keeps the ASCII order) though the corresponding binary representation may vary according to the chosen format (i.e., UTF-8 encodes the letter A with 41_{h}, like ASCII, while UTF-32 encodes the same letter with 00000041_{h}; see section 2.2).[2] While linguists can benefit from the straight way to write and store multilingual documents, programmers are not just bound to Unicode text strings: they can even use Unicode characters in variable names. In our opinion this is a great improvement in code readability.

Be careful: you need a Unicode editor to encode/decode a Unicode (source) file. Having such

1. Not to mention ideographic written languages.
2. From now on, stretching the term meaning, we will refer to characters beyond the 127^{th} as "Unicode characters".

an editor does not ensure file intelligibility. Indeed the editor can only show those glyphs forming the font in use and not necessarily a Unicode font has all the glyphs recommended by the standard.

2.2 How Unicode was designed

The Unicode encoding is a freely-available set of specifications for multilingual information interchange. As every other text encoding, Unicode provides a unique number—the *code point*— for every character to be represented. The code point identifies a specific character which is stored in memory as a sequence of bytes. The code point is independent of the hardware[3] and of the software platform and comes in a variety of multibyte formats called UCS *Transformation Format*—UTF. UCS stands for Universal Character Set. The Unicode standard ensures that every represented character will need at most 21 bits.

There are three main UTFs: UTF-8, co-created by Rob Pike and Ken Thompson, UTF-16 and UTF-32. The first one is the most widely used format and it is the *de facto* standard encoding for text, XML and JSON files;[4] the detailed UTF-8 encoding/decoding algorithm follows.

A Unicode code point is transformed from/into a sequence of one or more bytes according to the UTF rules. Table 1 shows such lookup table for UTF-8. It is straightforward to understand that a UTF-8 character is 1 to 4 bytes long according to its code point.

For example (as in KORPELA (2006)), UTF-8 encodes the code point U+212B—associated to the character Å—with 3 bytes because it belongs to the third interval of table 1 as $800 < 212B < FFFF$. What about those 3 bytes? The first step is to convert the hexadecimal code point into a binary number: $21\,2B_{16} \rightarrow 00100001\,00101011_2$. Now we use this number, or better, its 16 bits, to fill in the blanks (we highlight them in italics; the numbers in roman are those set by the standard) in range 3 of table 1: 1110*0010* 10*000100* 10*101011*. Now convert this 3-bytes binary sequence into the hexadecimal equivalent: $11100010\,10000100\,10101011_2 \rightarrow E2\,84\,AB_{16}$. $E2\,84\,AB_{16}$ is the 3-bytes sequence we are looking for: the UTF-8 representation of Å in memory. If we revert the procedure starting from the E2 84 AB sequence we immediately realize that the first byte starts with 1110 and so the sequence, and hence the character, belongs to the third range. Then we go back to get the code point 212B.

Please notice that the encoder always maps characters supposing that the input data are represented according to the requested format but without any guarantee that this is true.

It should be clear that data are still written in a file as a stream of bytes. A second consequence is that a Unicode string of fixed length (i.e., containing a fixed number of glyphs) cannot be represented by a fixed number of bytes. Generally speaking, UTF-8 does not allow a direct indexing of Unicode strings (we are not considering the trivial case of a text of only ASCII symbols that have a direct one byte representation because we cannot rely on this specific case). Hence programming languages that support Unicode strings are not likely to provide this indexing operator.

Other transformations like UTF-16 and UTF-32 are much simpler than UTF-8: the first one uses 16 or 32 bits to encode every character, while the second one uses 32 bits. A text encoded in UTF-32 is of fixed length in terms of memory request and it is straight to decode such a text without any of the passages described for UTF-8; even indexing is straightforward. Differently, UTF-16 needs to be encoded similarly to UTF-8 and hence does not provide file indexing. Despite being slightly more complex than UTF-32, UTF-16 saves memory when used to encode a text with a lot of characters that UTF-8 would encode with 3 bytes (mostly Asian). Nevertheless, UTF-8 holds a number of big advantages[5] over the simpler UTF-16 and UTF-32 formats:

- it is ASCII-compatible: any UTF-8 code point less than 128 is guaranteed to have the same binary representation of ASCII;
- it saves memory: 4-bytes encoded characters are not particularly common in practice; the most part of web pages is plain ASCII (i.e., HTML tags and CSS properties);
- it has no endianness issues: only UTF-16- or UTF-32-encoded texts can be big- or little endian-ordered;
- it is the standard (for the internet, at least): storing files in UTF-8 allows direct reading and writing, without any re-encoding step.

2.3 Unicode, programmers and typesetters

Many modern programming languages take advantage of the Unicode standard and their compilers or interpreters have to read Unicode source files. Go by Google (GRIESEMER *et al.*, 2014) and Rust by Mozilla Foundation (HOARE e RUST PROJECT DEVELOPERS, 2014) are such new mainstream languages which also attempt to be *system programming languages*. They both require UTF-8-encoded source files.

3. The sequence of bytes is often not: it should be noticed that some Unicode formats take into account the endianness.

4. The JSON Javascript Object Notation is a data description format used over the internet in the client-server communication.

5. More information can be retrieved on `http://utf8everywhere.org/website`.

TABLE 1: The multibyte UTF-8 encodes code points to numbers from 1 to 4 bytes long. Ranges, here hexadecimally represented, are grouped by length and have fixed data in specific positions just like a prefix notation. This technique immediately addresses the decoding process and, most important, how many bytes are needed to represent the current character.

Byte 0	Byte 1	Byte 2	Byte 3
Range 1 → 0 ... 7F			
0 a_6 a_5 a_4 a_3 a_2 a_1 a_0			
Range 2 → 80 ... 7FF			
1 1 0 a_{10} a_9 a_8 a_7 a_6	1 0 a_5 a_4 a_3 a_2 a_1 a_0		
Range 3 → 800 ... FFFF			
1 1 1 0 a_{15} a_{14} a_{13} a_{12}	1 0 a_{11} a_{10} a_9 a_8 a_7 a_6	1 0 a_5 a_4 a_3 a_2 a_1 a_0	
Range 4 → 10000 ... 10FFFF			
1 1 1 1 0 a_{20} a_{19} a_{18}	1 0 a_{17} a_{16} a_{15} a_{14} a_{13} a_{12}	1 0 a_{11} a_{10} a_9 a_8 a_7 a_6	1 0 a_5 a_4 a_3 a_2 a_1 a_0

Even LaTeX programmers/typesetters can easily compile UTF-8-encoded documents, provided

1. they use XeTeX or LuaTeX—Unicode compliant— or pdfTeX with the `utf8` input encoding as typesetting engines;

2. they use a Unicode-compliant font that has, at least, all of the needed glyphs.

Typesetting a Unicode source code into a LaTeX document (i.e., for documentation purpose) can be trickier than typesetting Unicode text and additionally requires that

3. the specific package correctly handles Unicode sources.

3 Packages and external tools

LaTeX users who need to typeset source code in a fancy yet typographically perfect way have several packages to pick from. Every one of them has pros and cons: one is light but too limited, one is way too powerful and configurable yet heavy; one lacks some features though excels in some other tasks. Some of them are incapable of correctly typesetting *and/or* highlighting source code containing those Unicode characters longer than one byte.

Our test will take into account the following packages: verbatim, fancyvrb, jvlisting, listings, minted, verbments, and pythontex and will unveil how they manage Unicode source files and how flexible they are. No other feature, i.e., line numbering, will be taken into account though we will remark which packages can mark (in draft mode) or wrap lines too long.

As already stated, some of those packages can or have to use the tools described in sections 3.1 and 3.2.

3.1 Pygments

Pygments (BRANDL *et al.*, 2014) is a Python project dedicated to syntax highlighting. Every programming language obeys to several syntax and semantic rules and needs a process that transforms a source code written in that language into a real executable. The same mechanism that transforms the source code into a program can be used to highlight or beautify (i.e., indent according to specific styles) the source code.

Pygments, which is a very accurate (meta)lexer, easily highlights different languages;[6] it can also expand its capabilities to include new programming languages. It outputs the results in different formats including LaTeX.

Pygments considers a *lexer* as a programming language analyzer to recognize both syntax and semantic items of a grammar while considers a *style* as a typographical set of rules to apply to the language elements such as keywords, operators, literal values, comments and so on.

Three out of the tested packages, i.e., minted, verbments and pythontex, can only highlight source files through Pygments accessing its script `pygmentize`. The script is also capable to show the user the available lexers and styles respectively with the following commands issued in a terminal session:

```
$ pygmentize -L lexer
$ pygmentize -L style
```

6. The list of lexers supported by Pygments is very long, as reported in the official web site `http://pygments.org/docs/lexers/`.

In addition, we can read the web page `http://pygments.org/docs/lexers/` to get the same information on lexers.

It can be useful a short "user's point of view" of the three packages.

3.1.1 The Verbments package

The latest available version of the Dejan Živković's LaTeX package is currently the 1.2 of the year 2011. Verbments is the union of the words 'verbatim' and 'pygments', the two modules verbments is based on.

One run of the chosen typesetting engine—`pdflatex`, `xelatex` or `lualatex`—with the option `-shell-escape` on the source TeX document suffices. Indeed, the verbatim material is passed to Pygments and the result is saved in an external file (a buffer, as we will refer it from now on) for normal inclusion via `\input`. The working cycle shall be the following: write the source code to highlight in a pyglist environment in the TeX source file—verbments has no macro to include verbatim code from an external file—and run the typesetting engine whenever the content changes.

Subsequent compilations can miss the `-shell-escape` option because the verbments macro simply loads the already generated buffer. If the verbatim material changes, the buffer is not automatically updated and no warning is raised by verbments to alert the user that the document keeps having the older verbatim material. This apparently handy behavior may be dangerous. If our document contains two or more pyglist environments, the buffer always contains the code in the last pyglist. When compiling without `-shell-escape`, all of the source codes typeset in the document are the same: that one saved in the buffer. It might be convenient working this way until the moment we need to compile the camera-ready document or to check sizes. In this case it is necessary to use `-shell-escape`.

The package verbments seems to be an agile and simple tool especially useful for those users who are writing documents with short code fragments such as software manuals, programming books or other kind of teaching materials, that can be "hard-wired" in the document.

3.1.2 PythonTeX as highlighting engine

As we can read from its manual, PythonTeX's goal is primarily to execute and typeset Python code from within LaTeX; as a secondary achievement it provides the way to syntax highlight source code written in any of the languages supported by Pygments.

This package is delivered with TeX Live and MikTeX, so it is immediately available to nearly all TeX users. The Python code included in a TeX document is executed—so some troubles might happen—and the execution results are included back in the document.

The compilation process involves three step:

1. the first run of any TeX typesetting engine on the TeX document produces an external file containing the Python code;
2. the subsequent execution of the `pythontex.py` script on the external file outputs the results of the original Python code;
3. the final re-run of the chosen TeX engine pulls the pythontex output back in the document.

During the upcoming user's editing activity PythonTeX is able to run the Python code only if it has been changed and, in any case, when something has been changed in the `tex` source file; otherwise there is no need to run it again and PythonTeX will not do it.

The time has come to show a minimal example with PythonTeX to highlight the LaTeX code chosen for our benchmark. We picked the code in figure 1, stored in an external file named `alcestis.tex`.

First of all, we need to know which is the lexer to be used by Pygments to highlight a LaTeX code. (Needless to remember that PythonTeX uses Pygments as a backend to highlight code.) The lexer is, of course, `latex`. The command to perform the task is simply expressed by

`\inputpygments[`⟨*option*⟩`]{`⟨*lexer*⟩`}{`⟨*ext file*⟩`}`

Then we can write a LuaLaTeX source file as the following:

```
% !TEX program = lualatex
\documentclass{article}
\usepackage{fontspec}
\usepackage{pythontex}
\setmonofont[Scale=MatchLowercase]{%
    cmuntt.otf}

\begin{document}
\inputpygments{latex}{alcestis.tex}
\end{document}
```

to finally get the LaTeX document with the pretty-printed (highlighted) source code of `alcestis.tex` in it.[7]

3.1.3 The minted package

The Konrad Rudolph's package seems to be a middle ground between the aforementioned packages:

7. All of the shown source codes written to be compiled with XeLaTeX/LuaLaTeX will use True/OpenType fonts distributed along with TeX. Users do not need to install those fonts on their systems provided that they load them by file name instead of by font name: the typesetting engine will look for the wanted font in standard directories, including those under the distibution main directory. In this way users can try our examples as is, without any other preliminary operation.

it has more features than verbments but is not as complex as pythontex.

minted current version is the 1.7, last changed on 17/9/2011, and it is maintained from 2009 by Geoffrey Poore—the same author of PythonTeX. The author develops both packages using `git` as revision control system. The packages are hosted on `github`, respectively at `https://github.com/gpoore/minted` and `https://github.com/gpoore/pythontex`.

minted is available on CTAN and with TeX Live and MikTeX. The package documentation is also useful to successfully install Pygments, especially on Windows machines.

The workflow is made up of only one step: do *always* compile the source `tex` file with the `-shell-escape` option. Pygments is executed behind the scenes according to the idea to create a complete LaTeX user interface to the Python library.

Remember that the typeset code appearance is a Pygments responsibility, so do not blame it on minted if something goes wrong in syntactic or semantic highlighting of your code. It is not so difficult to write a new lexer in Pygments to manage particular cases.

Equipped with minted, the user is capable of typesetting math within code comments and easily setup a Pygments style. The following example shows such features:

```
% !TEX program = pdflatex

\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage{minted}

% set a different highlighting style
% see the style list running the command
% $ pygmentize -L style
\usemintedstyle{colorful}

\thispagestyle{empty}

\begin{document}
\begin{minted}[mathescape]{rust}
/*
  A Rust (stupid) program to compute
  the sum of the first 100 integer
  as $\sum_{i=1}^{100}i$
*/
fn main() {
    let mut s = 0_u; // infer uint type
    for i in range(1, 101) {
        s += i; // $s \leftarrow s+i$
    }
    println!("{}", s);
}
\end{minted}
\end{document}

% file end
```

The result is the following—please notice that the `colorful` style adds a background light color to the literate string value:

```
/*
  A Rust (stupid) program to compute
  the sum of the first 100 integer
  as ∑_{i=1}^{100} i
*/
fn main() {
    let mut s = 0_u; // infer uint type
    for i in range(1, 101) {
        s += i; // s ← s+i
    }
    println!("{}", s);
}
```

3.1.4 *Pygments stand alone*

Pygments and its script `pygmentize` can be used directly.[8] The procedure is the same implemented by the discussed packages and has two steps:

1. the user executes the script in a terminal to obtain the LaTeX source file containing the syntax-highlighted version of the input code;
2. then `\inputs` or copies and pastes the previous output into the `tex` document.

The `pygmentize` flag `-O` accepts a specific option—`full`— to produce a complete LaTeX source. For example, `pygmentize` processes a Rust source file called `sum.rs` using the `rust` lexer and saves the `latex` output in a file called `output.tex`:

```
$ pygmentize -f latex -l rust \
             -O full -o output.tex \
              sum.rs
```

The file `output.tex` is a complete LaTeX file—though containing some errors—that we can compile as usual. It contains a series of custom macros and colors used in the Verbatim environment to syntax highlight the code.

Without the `full` option, `pygmentize` only produces the Verbatim environment with syntax highlighted code. This is the output file we can directly include in other documents, provided we use fancyvrb.

This can help us typeset and manage code in complex cases, for instance, those cases in which code fragments to be included in the document are in external files and these files must be compiled to verify the correctness of the code itself and even executed. No manual effort is needed when an automatic tool can store the code in one place, extract the interesting portions and syntax highlight them with Pygments. Such a tool could make it very easy creating a complex book containing many code snippets of executable source code.

8. Though the need of doing this operation by hand may seem odd, just think of a document with an occasional source code and no need to burden the compiler including another package or no chance to reliably use the package. This paper has plenty of such latest cases.

3.2 Fold

Fold is a Unix command used for making a text file with long lines more readable on a limited width terminal. Most Unix terminals have a default screen width of 80 characters, and therefore reading files with long lines could get annoying. The fold command puts a line feed every x characters if it does not reach a new line before that point.[9] The user can pass the x value to the command with the `-w` option.

This program—a filter—can be used as a preprocessor when the used package does not break input text lines. The newest versions of fold manage Unicode characters.

4 The benchmark source codes

We are going to test how LaTeX and the aforementioned packages typeset the source files shown in figures 1 and 2. The first one is a normal LaTeX code; the second one is a Rust code snippet representing a function. Those figures show both source codes as typeset by XeLaTeX and minted using the Deja Vu Mono font which is almost as complete as FreeSerif.

We will compile both the benchmark codes with pdfLaTeX, XeLaTeX and LuaLaTeX and all of the packages involved in the test.

Before letting the results speak for themselves, we want to point out that we expect pdfLaTeX to poorly perform in these tests regardless of the package. The reason is simple: pdfLaTeX cannot show characters other than Latin and a few other symbols unless it uses some specific commands provided by babel. We just cannot use these commands in a verbatim environment or alike because they would appear verbatim. Even in case a method to escape those commands existed so that they would be correctly compiled by LaTeX, it would be impossible to use them in an external code to be included in a document because they would hardly be part of the language grammar and thus accepted by the language compiler. We included pdfLaTeX in this work for completeness' sake.

5 Test and results

This section reports the detailed results. At the end of the section we summarize them in a synoptic table (table 2) that can be useful to have them all at a glance.

Please notice that we compiled every test with the draft option to let LaTeX mark lines too long. We will see which cases will bypass this option though this is one of the features we did not take into account.

9. http://en.wikipedia.org/wiki/Fold_%28Unix%29.

5.1 verbatim

The verbatim package is the simplest package of our test, probably the obvious alternative to the basic LaTeX's verbatim environment. This simple package, unable to syntax highlight or to break lines, can manage Unicode characters with pdfLaTeX and with the other Unicode-compliant typesetters. As we can see in figure 3, Greek letters are not represented. This is not verbatim fault: the first error message we got is

```
! LaTeX Error: Command \textSigma
      unavailable in encoding T1.
```

which means that our forecast was correct (see section 4). We tried to add LGR to the font encoding, but the result is awkward with all of the Latin letters transliterated into Greek. The same figure shows no problems with XeLaTeX and LuaLaTeX.

5.2 fancyvrb

The same problem is common to fancyvrb. It shows exactly the same pros and cons as verbatim and the results are almost identical. The only difference is that overfull hboxes are marked when compiling in draft mode with verbatim and not marked with fancyvrb (figure 4).

5.3 jvlisting

Another tested package is jvlisting. Of course it has the same problems of verbatim and fancyvrb with Greek letters and pdfLaTeX (because of the latter) and cannot highlight keywords but breaks lines too long. This package can mark lines too long that cannot be broken (figure 5).

5.4 listings

The listings package seems to be far better than those listed so far: it highlights a source code according to the rule for a specific language (though the supported languages are quite outdated) and it breaks lines to avoid overfull hboxes. It seems to correctly recognize Unicode characters when used with XeLaTeX and LuaLaTeX but we can see that it writes those characters in quite arbitrary positions. Moreover, when used with pdfLaTeX it has problems with Unicode sequences that it judges malformed, though it blames it on the inputenc package that issues the following message:

```
! Package utf8x Error:
  MalformedUTF-8sequence.
```

You can see the poor results in figure 6.

5.5 minted

Another tested package is minted. It is one of the packages requiring the compile option `-shell-escape` because it invokes the external program `pygmentize`. Though originally written for XeLaTeX, it can be used with pdfLaTeX too. It is

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBΣD Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ', ὁρᾷς γὰρ τἀμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

FIGURE 1: The LaTeX code of our benchmark.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π _nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π _nums.get(i) {
            let π = 2*i + 3;          // prime value
            let mut k = (n/ π -3)/2; // vec index
            while k+1 > i {
                *π _nums.get_mut(( π *(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return π _nums;
}
```

FIGURE 2: The Rust code of our benchmark.

unable to break lines too long or to mark them, but the cooperation with Pygments gives wonderful results in terms of highlighting, as readers can see in figure 7. The typesetting of the Rust code shows that Pygments fails at recognizing the π as a valid symbol, so Pygments' lexer issues a syntax error and surrounds the symbol with a red square.

5.6 verbments

The following tested package has been verbments which also requires the `-shell-escape` flag in compilation. Differently than minted, we can avoid this flag if the source code to include has not been changed. Indeed verbments, during the first compilation, saves a processed version of the file to include in another file. Unless we change the original source file, we have no need to make verbments update it.

The results obtained with XeLaTeX/LuaLaTeX are as good as usual (but no line break is performed and too long lines are not marked) while results with pdfLaTeX are odd: TeX commands attempting to render missing Unicode characters substitute the real glyphs. We obviously read those commands as they are typeset verbatim (figure 8).

5.7 pythontex

The last tested package is pythontex. We already mentioned its features in section 3.1.2 and will not repeat them here. Its results are exactly the same allowed by minted and the only difference is that pythontex currently runs faster.

5.8 Packages' performances at a glance

Now that we examined in detail all of the tested packages, we want to give the readers the relevant results at a glance, that is why we made the synoptic table 2.

Conclusion

Typesetting source codes in LaTeX can be as straight as loading a package (i.e., verbatim). Typesetting and highlighting source codes is trickier, especially when users need automatic line break. When Unicode characters are involved, the game gets even harder.

This paper tested seven packages written for source code typesetting and summarized their performances with three typesetters: pdfLaTeX, XeLaTeX and LuaLaTeX. Those readers who need to typeset and highlight Unicode source files in their documents have now an overview to address the best choice for themselves.

Having a last look at the results, we only can hope that developers improve the lexers performances and that the TeX community will widen the Unicode support.

Acknowledgment

We wish to acknowledge Claudio Beccari and Tommaso Gordini for writing their guide to code maps and for answering some questions of ours. They told us how they encompassed the problems we found with pdfLaTeX using a trick that avoids using any verbatim-like environment.

We also wish to acknowledge the reviewers and the proof-readers for making the paper far better than it originally was. Despite we thoroughly checked any single piece of information reported, every surviving error is just our fault.

References

Baker, P. S. (2014). *Junicode. The font for medievalists*. URL `http://junicode.sourceforge.net`. Lo leggete dando `texdoc junicode` al prompt del terminale o di DOS.

Beccari, C. e Gordini, T. (2012). «Codifiche in TeX e LaTeX». `http://www.guitex.org/home/images/doc/GuideGuIT/introcodifiche.pdf`.

Brandl, G., Ronacher, A., Pocoo team e Hatch, T. (2014). «Pygments python syntax highlighter». Sito web ufficiale `http://pygments.org/`.

Griesemer, R., Pike, R. e Thompson, K. (2014). «Go programming language». Sito web ufficiale `http://golang.org/`.

Hoare, G. e Rust Project Developers (2014). «Rust programming language». Sito web ufficiale `http://rust-lang.org/`.

Korpela, J. K. (2006). *Unicode Explained*. O'Reilly, Sebastopol, CA.

Unicode (2014). «The unicode consortium». Sito web ufficiale `http://www.unicode.org`.

▷ Roberto Giacomelli
`giaconet dot mailbox at gmail dot com`

▷ Gianluca Pignalberi
`g dot pignalberi at alice dot it`

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBD Con.:
\textit{Tournier, Pierson}.}}
\noindent ‘, ` ` ’
,\ledsidenote{ A.}\\
\pend
\endnumbering
\end{document}
```

(a) pdfLaTeX + verbatim do not show Greek letters but mark lines too long.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut _nums: Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *_nums.get(i) {
            let  = 2*i + 3;       // prime value
            let mut k = (n/-3)/2; // vec index
            while k+1 > i {
                *_nums.get_mut((*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }

    return _nums;
}
```

(b) pdfLaTeX + verbatim do not show Greek letters but mark lines too long.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBED Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ', ὀρᾷς γὰρ τάμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + verbatim show Greek letters *and* lines too long.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π_nums: Vec<bool> = Vec::from_elem(
       if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                             // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π_nums.get(i) {
            let π = 2*i + 3;       // prime value
            let mut k = (n/π-3)/2; // vec index
            while k+1 > i {
               *π_nums.get_mut((π*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }

    return π_nums;
}
```

(d) XeLaTeX/LuaLaTeX + verbatim show Greek letters *and* mark lines too long.

Figure 3: Two Unicode source codes typeset with verbatim. This package lets LaTeX mark lines too long in draft mode.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBD Con.:
\textit{Tournier, Pierson}.}}
\noindent ‘, ` ` ’
,\ledsidenote{ A.}\\
\pend
\endnumbering
\end{document}
```

(a) PDFLaTeX + fancyvrb do not show Greek letters *nor* mark lines too long.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut _nums: Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *_nums.get(i) {
            let  = 2*i + 3;       // prime value
            let mut k = (n/-3)/2; // vec index
            while k+1 > i {
                *_nums.get_mut((*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }

    return _nums;
}
```

(b) PDFLaTeX + fancyvrb do not show Greek letters *nor* mark lines too long.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBΣD Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ`, ὁρᾷς γὰρ τάμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + fancyvrb show Greek letters but do not mark lines too long.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π_nums: Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π_nums.get(i) {
            let π = 2*i + 3;       // prime value
            let mut k = (n/π-3)/2; // vec index
            while k+1 > i {
                *π_nums.get_mut((π*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }

    return π_nums;
}
```

(d) XeLaTeX/LuaLaTeX + fancyvrb show Greek letters but do not mark lines too long.

FIGURE 4: Two Unicode source codes typeset with fancyvrb. The results are almost identical to those of verbatim but the lines too long are not marked.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it
VLPBD Con.:
\textit{Tournier, Pierson}.}}
\noindent ‘,  ` ` ’
,\ledsidenote{ A.}\\
\pend
\endnumbering
\end{document}
```

(a) pdfLaTeX + jvlisting show no Greek and break lines too long, or mark those unbreakable.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut _nums: Vec<bool> =
Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2},
// length
        true                                //
value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *_nums.get(i) {
            let  = 2*i + 3;       // prime
value
            let mut k = (n/-3)/2; // vec
index
            while k+1 > i {
                *_nums.get_mut((*(2*k+3) -
3)/2) = false;
                k -= 1;
            }
        }
    }

    return _nums;
}
```

(b) pdfLaTeX + jvlisting show no Greek and break lines too long, or mark those unbreakable.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBΣD
Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ᾽, ὁρᾷς γὰρ τἀμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + jvlisting show Greek.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π_nums: Vec<bool> = Vec::from_elem(
       if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π_nums.get(i) {
            let π = 2*i + 3;       // prime value
            let mut k = (n/π-3)/2; // vec index
            while k+1 > i {
               *π_nums.get_mut((π*(2*k+3) - 3)/2)
= false;
                k -= 1;
            }
        }
    }

    return π_nums;
}
```

(d) XeLaTeX/LuaLaTeX + jvlisting show Greek.

Figure 5: Two Unicode source codes typeset with jvlisting. The package performs like verbatim and fancyvrb but can break lines too long and can mark those non-breakable.

```
\documentclass[a4paper,12pt]{article
   }
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{
   FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.:
   \it ΣVLPBD Con.:
\textit{Tournier, Pierson}.}}
[U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]
[U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]
J+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]
   [U+FFFD][U+FFFD][U+FFFD][U+FFFD]enote{[U+FFFD]\\
\pend
\endnumbering
\end{document}
```

(a) PDFLaTeX + listings show broken Greek letters but highlight *and* break lines too long.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd
   numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool>
   {
    let lim = (n as f64).sqrt() as
       uint;
    let mut[U+FFFD]_numsVec<bool> = Vec
       ::from_elem(
        if n%2 == 0 {n/2 - 1} else {
           n/2}, // length
        true

           // value
    );

    for i in range(0, (lim-3)/2 + 1)
        {
        if[U+FFFD]_numget(i) {
            let[U+FFFD] 2*i + 3;
               // prime value
            let mut k = ([U+FFFD])/2;
               // vec index
            while k+1 > i {
               [U+FFFD]_numget_mut[U+FFFD]
                    *(2*k+3) - 3)/2)
                    = false;
                k -= 1;
            }
        }
    }

    return[U+FFFD]_nums;
}
```

(b) PDFLaTeX + listings show broken Greek letters but highlight *and* break lines too long.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it ΣVLPBD
    Con.:
\textit{Tournier, Pierson}.}}
\noindent ̓Αδμηθ`, ὁρᾷςγὰρτάμὰπράγμαθ    ' ὡςἔχει
,\ledsidenote{ ΛA.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + listings put Greek letters in arbitrary and wrong positions.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π_nums: Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, //
            length
        true                              //
            value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if π*_nums.get(i) {
            let π = 2*i + 3;       // prime
                value
            let mut k = (nπ/-3)/2; // vec index
            while k+1 > i {π
                *_nums.get_mutπ((*(2*k+3) - 3)
                    /2) = false;
                k -= 1;
            }
        }
    }

    return π_nums;
}
```

(d) XeLaTeX/LuaLaTeX + listings put Greek letters in arbitrary and wrong positions.

FIGURE 6: Two Unicode source codes typeset with listings. While the package works bad with XeLaTeX and LuaLaTeX (arbitrarily positioned Unicode characters), it poorly performs with PDFLaTeX: it does not recognize any Unicode character and scrambles the result.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBD Con.:
\textit{Tournier, Pierson}.}}
\noindent ‘, ` ` ’
,\ledsidenote{ A.}\\
\pend
\endnumbering
\end{document}
```

(a) PDFLaTeX + minted do not show Greek letters and cannot break lines but highlighting is very good.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut | _nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *| _nums.get(i) {
            let | = 2*i + 3;          // prime value
            let mut k = (n/ | -3)/2; // vec index
            while k+1 > i {
                *| _nums.get_mut(( | *(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return | _nums;
}
```

(b) PDFLaTeX + minted do not show Greek letters and cannot break lines but highlighting is very good. Pygments' lexer fails at recognizing π as a valid symbol according to the syntax and so surrounds it with a red square.

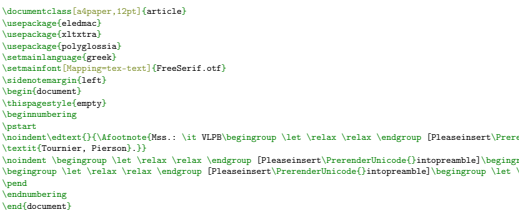
```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBΣD Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ᾽, ὁρᾷς γὰρ τἀμὰ πράγματθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + minted perform almost perfectly. It only lacks line break.

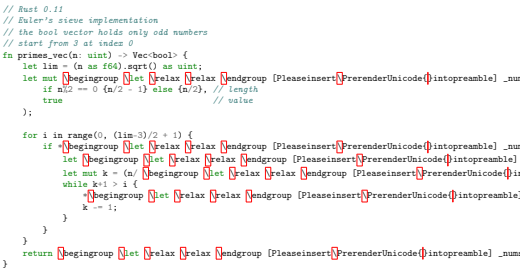
```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π _nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π _nums.get(i) {
            let π = 2*i + 3;          // prime value
            let mut k = (n/ π -3)/2; // vec index
            while k+1 > i {
                *π _nums.get_mut(( π *(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return π _nums;
}
```

(d) XeLaTeX/LuaLaTeX + minted perform almost perfectly. It only lacks line break. Pygments' lexer fails at recognizing π as a valid symbol according to the syntax and so surrounds it with a red square.

FIGURE 7: Two Unicode source codes typeset with minted. The only drawback is that it does not perform line break (users should preprocess source code with `fold`). Pygments' Rust lexer fails at recognizing π as a valid symbol according to the syntax (and so do Go and C lexers).



(a) pdfLaTeX + verbments substitute Greek letters with TeX commands and cannot break lines but highlighting is very good.



(b) pdfLaTeX + verbments substitute Greek letters with TeX commands and cannot break lines but highlighting is very good. Pygments' lexer fails at recognizing π as a valid symbol according to the syntax and so surrounds it with a red square.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBΣD Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ', ὁρᾷς γὰρ τάμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + verbments perform almost perfectly. It only lacks line break.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π _nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π _nums.get(i) {
            let π = 2*i + 3;          // prime value
            let mut k = (n/ π -3)/2; // vec index
            while k+1 > i {
                *π _nums.get_mut(( π *(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return π _nums;
}
```

(d) XeLaTeX/LuaLaTeX + verbments perform almost perfectly. It only lacks line break. Pygments' lexer fails at recognizing π as a valid symbol according to the syntax and so surrounds it with a red square.

FIGURE 8: Our benchmark source codes typeset with verbments. It performs like minted with XeLaTeX/LuaLaTeX but results with pdfLaTeX are odd.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBD Con.:
\textit{Tournier, Pierson}.}}
\noindent μ`,  ` μ` μ'
,\ledsidenote{ A.}\\
\pend
\endnumbering
\end{document}
```

(a) PDFLaTeX + pythontex do not show Greek letters and cannot break lines but highlighting is very good.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut _nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *_nums.get(i) {
            let p = 2*i + 3;          // prime value
            let mut k = (n/p -3)/2; // vec index
            while k+1 > i {
                *_nums.get_mut((p*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return p _nums;
}
```

(b) PDFLaTeX + pythontex do not show Greek letters and cannot break lines but highlighting is very good. Pygments' lexer fails at recognizing π as a valid symbol according to the syntax and so surrounds it with a red square.

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPΒD Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ`, ὁρᾷς γὰρ τἀμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

(c) XeLaTeX/LuaLaTeX + pythontex perform almost perfectly. It only lacks line break.

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π_nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π_nums.get(i) {
            let π = 2*i + 3;          // prime value
            let mut k = (n/π -3)/2; // vec index
            while k+1 > i {
                *π_nums.get_mut((π*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return π _nums;
}
```

(d) XeLaTeX/LuaLaTeX + pythontex perform almost perfectly. It only lacks line break. Pygments' lexer fails at recognizing π as a valid symbol according to the syntax and so surrounds it with a red square.

FIGURE 9: The same Unicode source codes typeset with pythontex which results are exactly like those obtained with minted.

Table 2: Summary of the typesetting and highlighting Unicode source code packages parallel test.

Package	`-shell-escape`	Highlights	Breaks lines	Program	Program + package typeset Unicode
verbatim	✗	✗	✗	P X L	✗[1] ✓ ✓
fancyvrb	✗	✗	✗	P X L	✗[1] ✓ ✓
jvlisting	✗	✗	✓[2]	P X L	✗[1] ✓ ✓
listings	✗	✓	✓	P X L	✗[3] ✗[4] ✗[4]
minted	✓	✓[5]	✗	P X L	✗[1] ✓ ✓
verbments	✓[6]	✓[5]	✗	P X L	✗[7] ✓ ✓
pythontex	✓	✓[5]	✗	P X L	✗[1] ✓ ✓
Legenda	✗ No	✓ Yes	P: pdfLaTeX	X: XeLaTeX	L: LuaLaTeX

[1] The package does manage Unicode characters but the compiler does not and, of course, does not show them along with Latin letters.

[2] The package will mark those long but non-breakable lines.

[3] The package conflicts with `utf8` of inputenc and every Unicode character is judged malformed. The typeset source code shows wrong code points instead of nothing or wrong characters.

[4] The compilation successfully ends but the Unicode characters are arbitrarily, and incorrectly, positioned.

[5] Through Pygments. Its lexer fails at recognizing the π symbol as valid and so marks it with a red square.

[6] The option can be avoided after the first compilation of the source code if the latter does not change.

[7] Unicode characters are replaced by a sequence of TeX commands that are shown as is in the verbatim-like environment.