A Gentle Introduction to Python TeX

Andrew Mertz, William Slough

Overview
A Question of Primes
Introduction to Python TeX
Mathematics with Sympy
Plots with matplotlib
Web Services
Conclusions
Extra Examples

A Gentle Introduction to Python TeX

Andrew Mertz    William Slough

Mathematics and Computer Science Department
Eastern Illinois University

October 23, 2013
Overview

A Question of Primes

Introduction to PythonTeX

Mathematics with Sympy

Plots with matplotlib

Web Services

Conclusions

Extra Examples
Python Overview

- General purpose, high-level programming language
- Multi-paradigm: object-oriented, imperative, functional
- Comprehensive standard library
- Origins from late 1989
- Free and open-source
Overview
A Question of Primes
Introduction to PythonTeX
Mathematics with Sympy
Plots with matplotlib
Web Services
Conclusions
Extra Examples

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

- NumPy: Base N-dimensional array package
- SciPy library: Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D plotting
- IPython: Enhanced Interactive Console
- Sympy: Symbolic mathematics
- pandas: Data structures & analysis

More information...
“I would like to write a \LaTeX script that produces all the prime numbers between the numbers $n$ and $m$, where $n < m$. How can I do this? I feel it should not be that hard, but I cannot seem to program it.”

— Kevin\(^\dagger\)

\(^\dagger\)\texttt{tex.stackexchange.com/questions/134305/how-to-produce-a-list-of-prime-numbers-in-latex/134366#134366}
The first 30 prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, and 113. You may not find this fact very startling; but you may be surprised to learn that the previous sentence was typeset by saying

```latex
\text{The first thirty prime numbers are \texttt{primes}\{30\}.}
```

\TeX{} did all the calculation by expanding the \texttt{primes} macro, so the author is pretty sure that the list of prime numbers given above is quite free of typographic errors.
Knuth’s Code, ✡ ✡-worthy

\newif\ifprime \newif\ifunknown % boolean variables
\newcount\n \newcount\p \newcount\d \newcount\a % integer variables
\def\primes#1{2,~3% assume that #1 is at least 3
\n=#1 \advance\n by-2 % n more to go
\p=5 % odd primes starting with p
\loop\ifnum\n>0 \printifprime\advance\p by2 \repeat}
\def\printp{, % we will invoke \printp if p is prime
\ifnum\n=1 and~\fi % and precedes the last value
\number\p \advance\n by -1 }
\def\printifprime{\testprimality \ifprime\printp\fi}
\def\testprimality{{\d=3 \global\primetrue
\loop\trialdivision \ifunknown\advance\d by2 \repeat}}
\def\trialdivision{{\a=\p \divide\a by\d
\ifnum\a>d \unknownntrue\else\unknownfalse\fi
\multiply\a by\d
\ifnum\a=\p \global\primefalse\unknownfalse\fi}
David Carlisle’s Response

\makeatletter
\def\primes#1#2{{% 
  \def\comma{\def\comma{, }}% 
  \count@\@ne\@tempcntb#2\relax\@curtab#1\relax \@primes}}
\def@primes{\loop\advance\count@\@ne \expandafter\ifx\csname p-\the\count@\endcsname\relax \ifnum\@tempcntb<\count@\else \ifnum\count@<\@curtab\else\comma\the\count@\fi\fi\else\repeat \@tempcnta\count@\loop\advance@tempcnta\count@ \expandafter\let\csname p-\the@tempcnta\endcsname@one \ifnum@tempcnta<@tempcntb\repeat \ifnum@tempcntb>@count@\expandafter@primes\fi} 
\makeatother
Karl Koeller’s Response

A solution using the \pgfmathisprime macro provided by Alain Matthes’ tkz-euclide package:

\usepackage{tkz-euclide}
\newif\ifcomma
\newcommand{\primes}[2]{%
  \commafalse%
  \foreach\numb in {#1,...,#2}{%
    \pgfmathisprime{\numb}%
    \ifnum\pgfmathresult=1
      \ifcomma, \numb\else\numb\global\commatrue\fi%
      \ifcomma, \numb\else\numb\global\commatrue\fi%
    \fi%
  }%
Can Python\TeX\ Make This Simpler?

Yes...
Evaluating Expressions With \texttt{\textbackslash py}

The macro \texttt{\textbackslash py\{expression\}} evaluates a Python expression and typesets its \texttt{value}.

Did you know that $2^{65} = \texttt{\textbackslash py\{2**65\}}$?

Did you know that $2^{65} = 36893488147419103232$?
The macro \texttt{pyc{expression}} evaluates a Python expression and typesets anything that it \texttt{prints}.

Did you know that $2^{65} = \texttt{print}(2**65)$? 

Did you know that $2^{65} = 36893488147419103232$? 

While “printing” adds little in this case, it is important for more complex examples.
A More Complex Example Using `\texttt{pyc}`

\texttt{\textbackslash pyc\{showGoogleMap("Tokyo", 11)\}}
\texttt{\textbackslash pyc\{showGoogleMap("600 Lincoln, Charleston, IL", 14)\}}
Generating Tables With \texttt{pycode}

\begin{pycode}
print(r"\begin{tabular}{c|c}
\hline
$m$ & $2^m$ \\
1 & 2 \\
2 & 4 \\
3 & 8 \\
4 & 16 \\
\end{tabular}"
)
\end{pycode}

\begin{tabular}{c|c}
$m$ & $2^m$ \\
1 & 2 \\
2 & 4 \\
3 & 8 \\
4 & 16 \\
\end{tabular}
\begin{pycode}
print(r"\begin{tabular}{c|c}"
print(r"$m$ & $2^m$ \ \hline"
print(r"%d & %d \" % (1, 2**1))
print(r"%d & %d \" % (2, 2**2))
print(r"%d & %d \" % (3, 2**3))
print(r"%d & %d \" % (4, 2**4))
print(r"\end{tabular}")
\end{pycode}

\begin{tabular}{|c|c|}
\hline
m & $2^m$
\hline
1 & 2 \\
2 & 4 \\
3 & 8 \\
4 & 16 \\
\hline
\end{tabular}
Generating Tables With a Loop

\begin{pycode}
lo, hi = 1, 6
print(r"\begin{tabular}{c|c}\hline"
print(r"$m$ & $2^m$ \ \ \hline")
for m in range(lo, hi + 1):
    print(r"%d & %d \" % (m, 2**m))
print(r"\end{tabular}"
\end{pycode}

<table>
<thead>
<tr>
<th>$m$</th>
<th>$2^m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
</tbody>
</table>
Defining a Function

\begin{pycode}
def fib(n):    # nth Fibonacci value
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
\end{pycode}

Did you know that $F_{10} = \text{fib}(10)$?

Did you know that $F_{10} = 55$?
Sessions

py, pyc, and pycode all have an optional session argument.

This argument determines the name of the Python session in which the code is executed.

Sessions with different names may be executed in parallel providing a speedup.

If a session is not specified, then the default session is used.
Introducing \texttt{pythontexcustomcode}

\begin{pythontexcustomcode}{py}
def makeTable(lo, hi):
    print(r''"egin{tabular}{c|c}
    \hline
    $m$ & $2^m$ \\
    \hline
    
    for m in range(lo, hi + 1):
    %d & %d \\ % (m, 2**m))
    print(r''"\end{tabular}")
    \end{pythontexcustomcode}

The \texttt{pythontexcustomcode} environment evaluates the code block at the start of each “session” – which makes it a great place to define well-tested functions.

\begin{pythontexcustomcode}{py}
python code block
\end{pythontexcustomcode}
## Generating Tables in Multiple Sessions

\texttt{\textbackslash pyc\{table1\}\{makeTable(1, 4)\}}

<table>
<thead>
<tr>
<th>(m)</th>
<th>(2^m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

\texttt{\textbackslash pyc\{table2\}\{makeTable(4, 10)\}}

<table>
<thead>
<tr>
<th>(m)</th>
<th>(2^m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
</tr>
</tbody>
</table>
Generating Tables From a Function

\begin{pythontexcustomcode}{py}
    def makeTableFromFunction(lo, hi, funct, label):
        print(r'\begin{tabular}{c|c}
            $m$ & %s \\ \hline' % label)
        for m in range(lo, hi + 1):
            print(r'\%d & \%d \\ ' % (m, funct(m)))
        print(r'\end{tabular}')
    \end{pythontexcustomcode}

\pyc{makeTableFromFunction(7, 11, fib, "$F_{m}$")}

\begin{tabular}{|c|c|}
    \hline
    \textbf{m} & \textbf{$F_m$} \\
    \hline
    7 & 13 \\
    8 & 21 \\
    9 & 34 \\
    10 & 55 \\
    11 & 89 \\
    \hline
\end{tabular}
Generating Tables From a Library Function

Python excels in the quantity and quality of its modules. Modules make additional functions available. To use them, the corresponding module needs to be imported.

\begin{pythontexcustomcode}{py}
import math
\end{pythontexcustomcode}

\begin{pyc}{makeTableFromFunction(30, 33, math.factorial, "$m!$")}

<table>
<thead>
<tr>
<th>$m$</th>
<th>$m!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>265252859812191058636308480000000</td>
</tr>
<tr>
<td>31</td>
<td>8222838654177922817725562880000000</td>
</tr>
<tr>
<td>32</td>
<td>263130836933693530167218012160000000</td>
</tr>
<tr>
<td>33</td>
<td>86833176188118864955181944012800000000</td>
</tr>
</tbody>
</table>


Generating Tables From a Library Function

With the import statement the module name is needed each time a member of the module is used.

To avoid this, a `from import` statement can be used.

This can shadow other functions and should be used with care.

\begin{pythontexcustomcode}{py}
from math import factorial
\end{pythontexcustomcode}

\pyc{makeTableFromFunction(30, 33, factorial, "\$m!\$")}

<table>
<thead>
<tr>
<th>$m$</th>
<th>$m!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>2652528598121910586363084800000000</td>
</tr>
<tr>
<td>31</td>
<td>82228386541779228177255628800000000</td>
</tr>
<tr>
<td>32</td>
<td>2631308369336935301672180121600000000</td>
</tr>
<tr>
<td>33</td>
<td>86833176188118864955181944012800000000</td>
</tr>
</tbody>
</table>
Remember Kevin?

\begin{pythontexcustomcode}{py}
from sympy import prime

def generatePrimes(n):    # Assume n \geq 3
    for i in range(1, n):
        print("%d, " % prime(i))
    print("and \%d\%" % prime(n))
\end{pythontexcustomcode}

The first 30 primes are \texttt{generatePrimes(30)}.

The first 30 primes are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, and 113.
Processing Python\TeX\ Files

1. pdflatex
2. python\TeX
3. pdflatex

my.tex

my.pytxcode

pythontex-files-my

my.pdf
Symbolic Mathematics With Sympy

```python
>>> from sympy import *

>>> var("x, y")  # Define symbolic variables
(x, y)

>>> z = (x + y)**3  # Define an expression

>>> z  # Display z
(x + y)**3

>>> expand(z)  # Display the expansion of z
x**3 + 3*x**2*y + 3*x*y**2 + y**3

>>> latex(expand(z))
'x^{3} + 3 x^{2} y + 3 x y^{2} + y^{3},cv
```
Expanding Binomials

\begin{pycode}
from sympy import *
var("x, y")

binomials = []
for m in range(3, 6):
    binomials.append((x + y)**m)

print(r"\begin{align*}
\text{for expr in binomials:}
    print(r"\%s &= \%s\\ % (latex(expr), latex(expand(expr))))")
print(r"\end{align*}"
\end{pycode}

\[(x + y)^3 = x^3 + 3x^2y + 3xy^2 + y^3\]
\[(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4\]
\[(x + y)^5 = x^5 + 5x^4y + 10x^3y^2 + 10x^2y^3 + 5xy^4 + y^5\]
A Little Bit of Calculus

\begin{pycode}
functions = [\sin(x), \cos(x), \tan(x)]
print(r''\begin{align*}'')
for f in functions:
  d = Derivative(f, x)
  print(latex(d) + "&=" + latex(d.doit()) + r'"
print(r''\end{align*}'')
print(r''\end{pycode}'')

\[
\frac{d}{dx} \sin(x) = \cos(x)
\]

\[
\frac{d}{dx} \cos(x) = - \sin(x)
\]

\[
\frac{d}{dx} \tan(x) = \tan^2(x) + 1
\]
A Little Bit More

\begin{pycode}
functions = [\sin(x), \cos(x), \tan(x)]
print(r"\begin{align*}"
for f in functions:
    i = Integral(f, x)
    print(latex(i) + "&=\) + latex(i.doit()) + r"\\")
print(r"\end{align*}"
\end{pycode}

\[
\int \sin(x) \, dx = -\cos(x)
\]

\[
\int \cos(x) \, dx = \sin(x)
\]

\[
\int \tan(x) \, dx = -\frac{1}{2} \log(\sin^2(x) - 1)
\]
### Stirling’s Triangle for Subsets

<table>
<thead>
<tr>
<th>$n$</th>
<th>${0}$</th>
<th>${1}$</th>
<th>${2}$</th>
<th>${3}$</th>
<th>${4}$</th>
<th>${5}$</th>
<th>${6}$</th>
<th>${7}$</th>
<th>${8}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>15</td>
<td>25</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>31</td>
<td>90</td>
<td>65</td>
<td>15</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>63</td>
<td>301</td>
<td>350</td>
<td>140</td>
<td>21</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>127</td>
<td>966</td>
<td>1701</td>
<td>1050</td>
<td>266</td>
<td>28</td>
<td>1</td>
</tr>
</tbody>
</table>
from sympy.functions.combinatorial.numbers import *

for n in range(numberOfRightHandColumns):
    print("%d" % n)
    for k in range(n + 1):
        print("& %d" % stirling(n, k))
print(r"\")
Inspired by a plot from matplotlib.org/1.3.1/gallery.html
Plot Details, Part 1

\begin{pycode}
from pylab import *

# Define f(t), the desired function to plot
def f(t):
    return cos(2 * pi * t) * exp(-t)

# Generate the points (t_i, y_i) to plot
    t = linspace(0, 5, 500)
    y = f(t)

# Begin with an empty plot, 5 x 3 inches
clf()
figure(figsize=(5, 3))

# Use TeX fonts
rc("text", usetex=True)
\end{pycode}
# Plot Details, Part 2

```python
# Generate the plot with annotations
plot(t, y)
title("Damped exponential decay")
text(3, 0.15, r"$y = \cos(2 \pi t) e^{-t}$")
xlabel("time (s)"
ylabel("voltage (mV)"

# Save the plot as a PDF file
savefig("myplot.pdf", bbox_inches="tight"

# Include the plot in the current LaTeX document
print(r"\begin{center})
print(r"\includegraphics[width=0.85\textwidth]{myplot.pdf}"
print(r"\end{center}")
\end{pycode}
```
Many powerful and freely available web services can be accessed through the libraries of Python.

Python has excellent JSON, XML and networking libraries.

The first web service we will use is Google’s Geocoding API.

Geocoding is the process of converting an address into geographic coordinates such as latitude and longitude.

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.
Using Google’s Geocoding Service

```python
from urllib2 import urlopen
from urllib import urlencode
import json

def findLatlong(address):
    # Build the data needed to call the Goggle API
    query = {"address": address, "sensor": "false"}
    data = urlencode(query)
    url = "http://maps.googleapis.com/maps/api/geocode/json?"
    url += data

    # Fetch and parse
    result = json.load(urlopen(url))
    latlong = result["results"][0]["geometry"]['location']
    return (latlong['lat'], latlong['lng'])

The latitude and longitude of Tokyo is \py{findLatlong("Tokyo")}
The latitude and longitude of Tokyo is (35.6894875, 139.6917064)
```
Executing a Subprocess

Python can run other programs and use their output.

Here we use `webkit2png` to render a web page as an image that is included in the document.

```python
import subprocess

def showWebpage(url, filename):
    subprocess.call(['webkit2png', '-o', filename, '-F', 'javascript', '-w', '5', url])
    print(r'\begin{center}')
    print(r'\includegraphics{%s}' % filename)
    print(r'\end{center}')
```
def showGoogleMap(address, zoomlevel):
    # Find the latitude and longitude of the address

    # Build a web page with the JavaScript needed to load
    # a Google Map at the given location and zoom level

    # Save the web page to a temporary file

    # Use showWebpage to display the map
Issues

- Python\TeX adds significant processing time
  \textit{Appropriate use of sessions can reduce this time, but there is still a large overhead.}

- Debugging Python code within \TeX is difficult
  \textit{Test complex Python code outside of \TeX first}

- \TeX macros that have arguments generated by Python fail on first processing step
  \textit{Add such \TeX macros from within Python}

- Use parentheses for print statements: \texttt{print(x)}.
- Be clear of the differences between \texttt{\textbackslash py} and \texttt{\textbackslash pyc}.
- When using Beamer use the frame option \texttt{fragile=singleslide} if able.
- Be skeptical of SymPy results.
- If all else fails delete the \texttt{pythontex-files} folder.
Questions?
References

Python: python.org

SciPy: scipy.org

PythonTeX (Geoffrey Poore): www.ctan.org/pkg/pythontex

Anaconda (Python distribution): store.continuum.io/cshop/anaconda

webkit2png: github.com/adamn/python-webkit2png
from urllib2 import Request

def shortenURL(longURL):
    # Build the data needed to call the Google API
    url = "https://www.googleapis.com/urlshortener/v1/url"
    query = {"longUrl": longURL, "key": googleAPIKey}
    data = json.dumps(query)
    request = Request(url, data,
                      {"Content-Type": "application/json"})
    result = json.load(urlopen(request))
    shortURL = result["id"]
    print(r"\url{%s}"
          % shortURL)

Here is a short url \pyc{shortenURL(
"http://mirror.jmu.edu/pub/CTAN/macros/latex/contrib/pythontex/pythontex.pdf")}

Here is a short url http://goo.gl/sfT8S5.
Mail Merge

Address: to field
Hello name field, I just wanted to say hello.

to,name
js@example.com,John Smith
mw@example.com,Mike White
tb@example.com,Tom Blue
Mail Merge

\begin{pythontexcustomcode}{py}
import csv

def mailMerge(filename, texcommand):
    csvFile = open(filename, "r")
    csvReader = csv.DictReader(csvFile)
    for row in csvReader:
        setCommand = r"\def\mail%s{%s}"

        for keyValuePair in row.items():
            print(setCommand % keyValuePair)

        print(r"%s\vfill" % texcommand)

\end{pythontexcustomcode}

\newcommand{\mailBody}{
Address: \mailto\nHello \mailname, I just wanted to say hello.
}
\text{Mail Merge}

\texttt{\textbackslash pyc\{mailMerge("../data.csv", r"\mailBody")\}}

Address: js@example.com
Hello John Smith, I just wanted to say hello.

Address: mw@example.com
Hello Mike White, I just wanted to say hello.

Address: tb@example.com
Hello Tom Blue, I just wanted to say hello.