Discrete Mathematics in Python
Ethan Bolker
March 11, 2015

1 Permutations

Here's a function that write out all the permutations of a list.

```
# Python code for permutations and combinations
#
# Not the most efficient — but a good way to learn
# recursion.
#
# http://stackoverflow.com/questions/104420/how-to-generate-all-permutations-of-a-list-in-python

def permute(alist):
    """ generate a list of permutations recursively """
    # there are no permutations of an empty list
    if len(alist) == 0:
        return([])
    # there is just one permutation of a one element list
    if len(alist) == 1:
        return [alist] # note: NOT return( alist )
    answer = []
    # for longer lists, find all permutations of the list
    # without its first element, and put that element in
    # each possible place in each permutation
    for p in permute(alist[1:]):
        for i in range(len(alist)):
            answer.append(p[:i] + [alist[0]] + p[i:])
    return answer

print("permute([1,2,3])")
print(permute([1,2,3]))
print("print(permute([]))")
print(permute([]))
print('permute(["a"]))') # single quotes allow embedded ""
print(permute(["a"]))
print("permute([1,2,2])")
print(permute([1,2,2]))
```

with its outputn

```
python combinatorics1.py
permute([1,2,3])
[[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]
print(permute([]))
[]
permute(["a"]))
[['a']]
permute([1,2,2])
[[1, 2, 2], [2, 1, 2], [2, 2, 1], [1, 2, 2], [2, 1, 2], [2, 2, 1]]
```

Lessons here:

- Turning recursive pseudocode into a recursive program
- Slicing a list returns a copy
- Single and double quotes

- This algorithm isn't smart about repetitions
- Should find the built in function in `itertools`
- Should use `yield` instead of **`return`**

In class a student remembered that $0! = 1$, which suggests that the list of permutations of the empty list should be `[[]]`, not the `[]` in my code. I tried that, which allowed me to treat the empty list as the base case and delete the special treatment for a list of length 1. While I was at it I changed the code so that the *last* element of the list was the new guy.

```
8  def permute(alist):
9      """ generate a list of permutations recursively """
10     # there are no permutations of an empty list
11     if len(alist) == 0:
12         return([[]])
13     answer = []
14     # for longer lists, find all permutations of the list
15     # without its last element, and put that element in
16     # each possible place in each permutation
17     newguy = alist[-1:]
18     for p in permute(alist[:-1]):
19         for i in range(len(alist)):
20             answer.append(p[:i] + newguy + p[i:])
21     return answer
```

This worked just fine. The permutations in the output appear in a different order, of course.

2 Itertools

In class I pointed out that the list of permutations of 10 elements would be very long – length 10! If you wanted that list in order to loop over it you would not want to build the whole list in memory. The `itertools` module can do just that.

Here's the code and the testing code:

```
34 import itertools
35
36 permutelist = itertools.permutations([1,2,3],3)
37
38 print()
39 print("Try to print the object returned by the itertools permutations function")
40 print(permutelist)
41 print("Iterating through that object works")
42 for p in permutelist:
43     print(p)
```

Here's the output:

```
myshell> python combinatorics2.py

...

Try to print the object returned by the itertools permutations function
<itertools.permutations object at 0x027244B0>
Iterating through that object works
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

Notes for next hw:

- Do an assignment for another course using TEXor Python (preferably both).
- Do something with itertools (question not made up yet)
- Improve the integration functions so that they correctly handle functions that may be negative (area under the x axis counts negative) and correctly handle t less than b. This may happen automatically for the trapezoid rule but will need work for monte carlo.
- TBD

Here is the LaTeX source for this document. You can cut it from the pdf and use it to start your answers. I used the `\jobname` *macro for the source file name, so you can call your file by any name you like.*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Combinatorics
% Math 480 Spring 2015
%

\documentclass[10pt]{article}
\usepackage[textheight=10in]{geometry}

\usepackage{verbatim}
\usepackage{amsmath}
\usepackage{amsfonts} % to get \mathbb letters

\usepackage[utf8]{inputenc}
\DeclareFixedFont{\ttb}{T1}{txtt}{bx}{n}{9} % for bold
\DeclareFixedFont{\ttm}{T1}{txtt}{m}{n}{9}  % for normal
% Defining colors
\usepackage{color}
\definecolor{deepblue}{rgb}{0,0,0.5}
\definecolor{deepred}{rgb}{0.6,0,0}
\definecolor{deepgreen}{rgb}{0,0.5,0}

\usepackage{listings}

%Python style from
%http://tex.stackexchange.com/questions/199375/problem-with-listings-package-for-python-syntax-colo
\newcommand\pythonstyle{\lstset{
  language=Python,
  backgroundcolor=\color{white}, %%%%%%%
  basicstyle=\ttm,
  keywordstyle=\ttb\color{deepblue},
  emph={MyClass,__init__},
  emphstyle=\ttb\color{deepred},
  stringstyle=\color{deepgreen},
  commentstyle=\color{red},  %%%%%%%%
  frame=tb,
  showstringspaces=false,
  numbers=left,numberstyle=\tiny,numbersep =5pt
}}

\usepackage{hyperref}

\begin{document}

\pythonstyle{}

%%%%%%%%%%%%%%%% start here %%%%%%%%%%%%%%%%
\begin{center}
\Large{
Discrete Mathematics in Python \\
Ethan Bolker \\
\today
}
\end{center}

\section{Permutations}
```

```
Here's a function that write out all the permutations of a list.

\lstinputlisting{combinatorics1.py}

with its outputn

\begin{verbatim}
python combinatorics1.py
permute([1,2,3])
[[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]
print(permute([]))
[]
permute(["a"]))
[['a']]
permute([1,2,2])
[[1, 2, 2], [2, 1, 2], [2, 2, 1], [1, 2, 2], [2, 1, 2], [2, 2, 1]]
\end{verbatim}

Lessons here:
\begin{itemize}
\item Turning recursive pseudocode into a recursive program
\item Slicing a list returns a copy
\item Single and double quotes
\item This algorithm isn't smart about repetitions
\item Should find the built in function in \lstinline!itertools!
\item Should use \lstinline!yield! instead of \lstinline!return!
\end{itemize}

In class a student remembered that $0! = 1$, which suggests that the
list of permutations of the empty list should be \lstinline![[]]!, not
the \lstinline![]! in my code. I tried that, which allowed me to treat
the empty list as the base case and delete the special treatment for a
list of length 1. While I was at it I changed the code so that the
\emph{last} element of the list was the new guy.

\lstinputlisting[firstnumber=8,firstline=8,lastline=21]{combinatorics2.py}

This worked just fine. The permutations in the output appear in a
different order, of course.

\section{Itertools}

In class I pointed out that the list of permutations of 10 elements
would be very long -- length 10! If you wanted that list in order to
loop over it you would not want to build the whole list in memory. The
\lstinline!itertools! module can do just that.

Here's the code and the testing code:

\lstinputlisting[firstnumber=34,firstline=34,lastline=44]{combinatorics2.py}

Here's the output:

\begin{verbatim}
myshell> python combinatorics2.py

...

Try to print the object returned by the itertools permutations function
<itertools.permutations object at 0x027244B0>
```

```
Iterating through that object works
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
\end{verbatim}

Notes for next hw:

\begin{itemize}
\item Do an assignment for another course using \TeX or Python
  (preferably both).

\item Do something with itertools (question not made up yet)

\item Improve the integration functions so that they correctly handle
  functions that may be negative (area under the x axis counts
  negative) and correctly handle $t$ less than $b$. This may happen
  automatically for the trapezoid rule but will need work for monte
  carlo.

\item TBD
\end{itemize}

\newpage
\emph{
Here is the \LaTeX{} source for this document. You can cut it from the
pdf and use it to start your answers. I used the} \verb!\jobname!
\emph{macro for the source file name, so you can call your file by any
  name you like.}
\verbatiminput{\jobname}

\end{document}
```