Discrete Mathematics in Python
Ethan Bolker
March 9, 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`**

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}
\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}
```