Calculating Permutations... The Lazy Way!

First of all, what the heck are permutations? Permutations are a mathematical concept. (There are actually several definitions, depending on what kind of math you're doing. Here I'm talking about combinatorics, but if that doesn't mean anything to you, don't worry about it. As always, Wikipedia is your friend.)

The idea is that you take a list of things (could be numbers, could be letters, could be dancing bears) and find all the possible ways to split them up into smaller lists. All the smaller lists have the same size, which can be as small as 1 and as large as the total number of items. Oh, and nothing can be repeated. Mathematicians say things like "let's find the permutations of 3 different items taken 2 at a time," which means you have a sequence of 3 items and you want to find all the possible ordered pairs.

```
import itertools                                #①
perms = itertools.permutations([1, 2, 3], 2) #②
print (next(perms))                             #③
#(1, 2)

print (next(perms))
#(1, 3)

print (next(perms))
#(2, 1)                                        #④

print (next(perms))
#(2, 3)

print (next(perms))
#(3, 1)

print (next(perms))
#(3, 2)

print (next(perms))                                      #⑤
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 21, in <module>
# print (next(perms)) #\u2464
#StopIteration
```

① The `itertools` module has all kinds of fun stuff in it, including a `permutations()` function that does all the hard work of finding permutations.

② The `permutations()` function takes a sequence (here a list of three integers) and a number, which is the number of items you want in each smaller group. The function returns an iterator, which you can use in a for loop or any old place that iterates. Here I'll step through the iterator manually to show all the values.

③ The first permutation of `[1, 2, 3]` taken 2 at a time is `(1, 2)`.

④ Note that permutations are ordered: `(2, 1)` is different than `(1, 2)`.

⑤ That's it! Those are all the permutations of `[1, 2, 3]` taken 2 at a time. Pairs like `(1, 1)` and `(2, 2)` never show up, because they contain repeats so they aren't valid permutations. When there are no more permutations, the iterator raises a `StopIteration` exception.

The *itertools* module has all kinds of fun stuff.

The `permutations()` function doesn't have to take a list. It can take any sequence — even a string.

```
import itertools
perms = itertools.permutations('ABC', 3)  #①
print (next(perms))
#('A', 'B', 'C')                          #②

print (next(perms))
#('A', 'C', 'B')

print (next(perms))
#('B', 'A', 'C')

print (next(perms))
#('B', 'C', 'A')

print (next(perms))
#('C', 'A', 'B')

print (next(perms))
#('C', 'B', 'A')

print (next(perms))
#Traceback (most recent call last):
```

```
#  File "/usercode/__ed_file.py", line 21, in <module>
# print (next(perms))
#StopIteration
```

```
import itertools
perms = itertools.permutations('ABC', 3)

print (list(itertools.permutations('ABC', 3)) )   #③
#[('A', 'B', 'C'), ('A', 'C', 'B'),
# ('B', 'A', 'C'), ('B', 'C', 'A'),
# ('C', 'A', 'B'), ('C', 'B', 'A')]
```

① A string is just a sequence of characters. For the purposes of finding permutations, the string 'ABC' is equivalent to the list `['A', 'B', 'C']`.

② The first permutation of the 3 items `['A', 'B', 'C']`, taken 3 at a time, is `('A', 'B', 'C')`. There are five other permutations — the same three characters in every conceivable order.

③ Since the `permutations()` function always returns an iterator, an easy way to debug permutations is to pass that iterator to the built-in `list()` function to see all the permutations immediately.