Discrete Mathematics Exercises 1 (February 3, 2016)

Exercise 1: *Permutations*
A *permutation* on a set S is a bijection from S to S itself. We will consider permutations on the set $S = \{0, \ldots, n-1\}$. A permutation p on S is called *trivial* if for all $i \in S$, it holds that $p(i) = i$. Permutations have the following two useful properties:

- If p is a permutation on S, then the *inverse* p^{-1} exists and is a permutation on S as well. (Recall that $f = p^{-1}$ is the function such that for all $i \in S$, $f(p(i)) = i$.)
- If p and q are permutations on S, then their *composition* $p \circ q$ is a permutation as well. (Recall that $f = p \circ q$ is the function such that for all $i \in S$, $f(i) = q(p(i))$ - although it is sometimes defined the other way around.)

A permutation is represented by a list of size n in Python. Every integer from 0 to $n-1$ must appear exatly once in the list for the permutation to be valid. For the permutation p with $p(i) = j$, then item i in the list has value j:

```
# p is a permutation
p[i] == j
```

Consider the module `perm.py` (which can be found on blackboard), which defines a number of useful functions to work with permutations. It is not necessary to understand exactly *how* the various functions are implemented, but you should understand *what* they do and how they can be called; see the examples below.

→ Import the permutation functions by writing

```
from perm import *
```

at the top of your program. This will import all variables and functions defined in the file `perm.py`. Make sure the file `perm.py` is in the same directory as your project/assignment files, otherwise it will not work.

Now we can try the permutation functions:

```
# Create a test permutation of length 20
q = testPermutations(20)
print(q)

# Define a permutation of length 10
p = [1, 2, 3, 0, 5, 4, 6, 7, 8, 9]
print(p)
printPermutation(p)
print(isTrivial(p))
print(p[0])
print(cycles(p))

# Create a trivial permutation of length 10
r = trivialPermutation(10)
print(r)
printPermutation(r)
print(r[0])
print(isTrivial(r))
```

→ Note that when printing a permutation p using the function `printPermuation`, it is shown using the *cycle notation*. Try to understand this notation by considering some small examples, and convince yourself that every permutation can be written this way. Note that permutations can also be created using this cycle notation, using the function `permutationFromCycles`:

```
p = permutationFromCycles(10, [[0, 1, 2, 3], [4, 5]])
print(p)
printPermutation(p)
```

Note that you have to give the number of items in the resulting permutation (10 in the example) as the first argument.

→ Because a permutation is a simple list, you can compare two permutations if they are equal using ==. Try comparing a few permuations.

→ Implement a function `composition` with two arguments p and q for computing the *composition* of the two permutations p and q (represented by $p \circ q$). Test your implementation by typing e.g.

```
p = [1, 2, 3, 0, 5, 6, 4, 8, 7]
printPermutation(p)
q = composition(p, p)
printPermutation(q)
```

which should yield `(0,2)(1,3)(4,6,5)`.

Also try it out with two different permutations `p` and `q`, since composition is not *commutative*: the compositions $p \circ q$ and $q \circ p$ are not necessarily the same!

→ Implement a function `inverse` with an argument p for computing the *inverse* of a permutation p.

→ Implement a method `power` with two arguments p and i, which can be used to compute the ith *power* p^i of a permutation p, where i is an integer: typing `power(p, 5)` should give $p^5 = p \circ p \circ p \circ p \circ p$.

Ensure that the method also works for $i = 0$ and $i < 0$.

→ Write a function `period`, with one argument p, that computes the smallest integer $i \geq 1$ such that p^i is the trivial permutation. Verify the correctness of the answer.

→ Unless you already implemented it smartly, you may notice that your program is rather slow when typing e.g.:

```
p = testPermutation(100)
period(p)
```

Improve your function `period` such that it yields the correct answer in a split second (for arbitrary permutations, including `testPermutation(200)`).

(Hints: Consider cycle lengths. Study some small examples such as `testPermutation(20)`. You have to use some theory that was presented earlier this week.)

→ *(Optional)* Similarly, unless you already implemented it smartly, you may notice that your program is still rather slow when verifying the above answer by typing

```
P = testPermutation(100)
printPermutation(power(p, period(p)))
```

Improve your `power` function such that it yields the correct answer in a split second (for arbitrary permutations).

Hint: use the fact that e.g. p^{107} can be written as

$$p^{107} = p \circ p^2 \circ p^8 \circ p^{32} \circ p^{64}.$$