Commenting in python

Commenting your code properly will help you and other understand and use it correctly. You should make this a habit when programming anything. Python allows for two types of comments: long multi-line comments and short comments. Your codes should contain a long comment at the beginning, which contains information for somebody using your code. A user does not need to know how the code works, but needs to know what it does without needing to read the code. A fellow programmer (or yourself in the future) needs to be able to follow the code line by line. Here is a simple (silly) example.

```
""" The following function c=prodplusone(a,b) takes as input two
real numbers (float or integer type variables) a and b, and
outputs a real number c, which is equal to ab+1. """

def prodplusone(a,b):
    p=a*b #multiply a and b and store the product as p
    c=p+1 #add 1 to the product to get c
    return c
```

More on lists

Here are some more examples of how to manipulate lists.

```
In:  L=[1,2,3,4]
In:  L[2]
Out: 3
In:  L[2:]
Out: [3,4]
In:  L[:2]
Out: [1,2]
In:  L[8:]
Out: []
In:  L+[5,6]
Out: [1,2,3,4,5,6]
In:  L[:]
Out: [1,2,3,4]
```

Lists may also have lists as their elements:

```
In:  K=[[1,2],[3,4]]
In:  K[0][1]
Out: 2
In:  K[0][:]
Out: [1,2]
```

In python, if say `if L:` then the list L is interpreted as true if it is nonempty, and false otherwise.

List comprehension

List comprehension is an extremely intuitive and efficient way to create lists in python. Take a moment to understand the difference between the last two examples below.

```
In:  [i**2 for i in range(6)]
Out: [0, 1, 4, 9, 16, 25]
In:  [i**2 for i in range(5) if i%2==0]
Out: [0,4,16]
In:  [0 for _ in range(5)] #we can use _ if we don't need a variable
Out: [0,0,0,0,0]
In:  [i+j for i in range(2) for j in range(4)]
Out: [0, 1, 2, 3, 1, 2, 3, 4]
In:  [[i+j for i in range(2)] for j in range(4)]
Out: [[0, 1], [1, 2], [2, 3], [3, 4]]
```

The notation $i**2$ is the way to write i^2 in python, and the notation $i\%2$ is the modulus operator: $i \pmod 2$.

Modules

Modules are packages with extra variable classes and functions. You need to import a module in order to use it. (You will not be able to import any module on the PIC lab computers: some are not installed.) As an example, we import the module `random`. If we execute

```
import random
```

We can then use the functions from random, such as

```
In:  random.random() #uniform real in [0,1]
Out: 0.5381215622636437
In:  random.randint(1,3) #uniform integer in {1,2,3}
Out: 2
In:  L=[3,4,5]
In:  random.shuffle(L) #uniform shuffle of a list
Out: [5,3,4]
In:  random.sample(L,2) #uniform sampling
Out: [3,4]
```

(Clearly your own output is rather unlikely to be equal to the above...) The random module has many other built-in distributions, such as normal, geometric, Poisson,... If we plan to use a module often, we can shorten the name. For example:

```
In: import random as rand
In:  rand.random()
Out: 0.6558883825289238
```

It is also possible to import everything from a module, and use the functions directly without having to specify the module. I think this is very bad practice, because we completely lose track of where the functions come from, and, more importantly, we run the risk of function definitions clashing between modules or with standard python functions.

```
In:  from random import * #this is bad
In:  random()
Out: 0.45064002256903524
```

Exercises

- Write a function that takes as input a natural number n, and outputs a list that looks like $[1^2, 2^2, 3^2, \ldots, n^2]$. You can now do this in a more efficient way than previously. Try `range(1,n)`.
- Use list comprehension to create the following object: $[1, 2, 5, 10, 17, 26, 37, 50, 65, 82]$.
- Use list comprehension to create the following object: $[0, 0, 0, 0, 1, 2, 0, 2, 4, 0, 3, 6, 0, 4, 8, 0, 5, 10]$.
- Use list comprehension to create the following object: $[[1, 2, 3], [2, 4, 6], [3, 6, 9], [4, 8, 16]]$.
- Write a function that takes as input a list, and outputs the list without its last element if the list is nonempty, and an empty list otherwise.
- You play a game where you flip a fair 2-sided coin and roll a fair 6-sided die. If the coin shows H, you receive a number of points equal to the outcome of the die, otherwise, you receive 7 minus the outcome of the die. Write a python code that simulates this game.
- Use only `random.random()` to simulate a 6-sided die roll.
- Write a function that takes as input a parameter $0 \leq p \leq 1$, and outputs a 1 with probability p, and 0 with probability $1 - p$. (A biased coin.)
- Use `random.sample` two write a function that takes as input a list L and a natural number k, and samples k elements from L **with replacement**.
- Write a function that takes a list of integers as input and outputs one uniformly randomly chosen even integer and odd one.