Introduction to programming, Lesson 3: functions, files, modules

Warm-up

1. What numbers will the program print? Why?

```
i = 10
while i <= 20:
  if i % 2 == 0:
    print '%d is even' % I
    i = i + 1
  if i % 2 == 1:
    print '%d is odd' %i
    i = i + 1
```

2. Which result do you expect? How to fix the script?

```
def sum(a,b)
  return a + b
a = 3
b = 5
print sum
```

New material

3. Let us again consider the problem of ordering three numbers entered by a user from a keyboard. This time, we would like to consider as few cases as possible. We will first implement a special function swap that receives two integer variables a, b and returns a, b if a < b and b, a otherwise. Here is an example of a function that returns two variables:

```
def diff_sum(a,b):
  return (a-b, a+b)
a = 3
b = 2
diff, sum = diff_sum(a,b)
print diff
print sum
```

The object (a-b, a+b) is called a tuple. Now order three variables using the function swap. It suffices to consider three different cases.

4. **Lists**

Similar to string, which is a sequence of characters, sometimes we need a sequence of int or a sequence of something else. In Python, sequences are implemented using lists.

```
animals = ['giraffe', 'lion', 'monkey', 'cat']
taille = [5.0, 1.0, 0.7, 2.0]
mix = ['giraffe', 5.0, 'monkey', 2]
```

Lists can be also created by using a function append.

```
animals = []
animals.append('giraffe')
animals.append('lion')
animals.append('monkey')
animals.append('cat')
```

We can call elements of a list by using indexes. Indexes of a list of length n change from 0 to n-1.

```
print animals[0], animals[1], animals[-1]
```

len gives you the length of a list.

```
print len(animals)
```

We can use for to iterate over elements of a list.

```
l = ['a', 'b', 'c', 'd']
for element in l:
    # Do something with the element
```

5. Write a function that takes a list and returns True if it contains a duplicate.
6. You can consider a string as a list of its characters. Write a script that prints out all characters of a string, one per line.
7. **Files**

```
file_name = "example.txt"
in_file = open(file_name,'r')#Open example.txt for reading
content = in_file.read()
print content
in_file.close()
```

Create a file example.txt. In the same folder, create a script that reads example.txt and prints out the content of example.txt. Try to do the same thing when example.txt is in a different folder.

```
in_file = open(file_name,'r')#Open file_name for reading
lines = in_file.readlines()
for line in lines:
  # Do something with the line
in_file.close()
```

Sometimes it is more convenient to work with files line by line. Create a file example.txt, where each line contains an integer. In the same folder, create a script that reads example.txt and prints out all the integers in the file that belong to the interval [10,20].

```
out_file = open(file_name,'w')#Open file_name for writing
animals = ['elephant', 'cat', 'dog']
for animal in animals:
  out_file.write(animal)
out_file.close()
```

Fix this script so that it writes the name of each animal on a separate line. If the output file already exists and you want to add something to the end, you must use mode 'a' (append) instead of 'w'.

8. **Modules**

Python developers have implemented many useful functions, and you should use them. Functions are grouped in collections called 'modules'. For example, there is a module math that contains different mathematical functions (https://docs.python.org/2/library/math.html) , there is a module re that allows you to work with regular expressions (https://docs.python.org/2/library/re.html), and so on.

```
from math import *
root = sqrt(49)
angle = pi/6
print sin(pi/6)
```

Use module math to compute the logarithm of 27 base 3.

Introduction to programming, Lesson 3: functions, files, modules

Harder exercises

9. **A Hundred Thousand Billion Poems**

   A Hundred Thousand Billion Poems (original French title: *Cent mille milliards de poèmes*) is a book by Raymond Queneau, published in 1961. The book is a set of ten sonnets printed on card with each line on a separate strip. As all ten sonnets have not just the same rhyme scheme but the same rhyme sounds, any lines from a sonnet can be combined with any from the nine others, allowing for 10^{14} (= 100,000,000,000,000) different poems. When Queneau ran into trouble creating the book, he solicited the help of mathematician Francois Le Lionnais. Your task will be to write a script that generates a random poem from a similar book. Download this file: http://bit.ly/2H4OhKw. It contains 14 groups of lines, and each group consists of 10 lines. Groups are separated by an empty line.

   1. Write a script that reads the file and generates a list of 14 lists, where each list corresponds to a group and contains 10 lines.
   2. Modify your script so that it returns a poem of 14 lines, where each line is chosen at random from each group. Use Python module `random`.

10. **Text generator**

    Write a function that generates a list of pairs of consecutive words in a text. Save this function in a module. Write a script that imports the module you have just created, receives a name of a file, reads the file, and prints out the pairs of consecutive words in the file.

11. **Linear interpolation**

    When we cover the numerical libraries, we will see they include many alternatives for interpolation and function approximation. Nevertheless, let's write our own function approximation routine as an exercise. In particular, without using any imports, write a function `linapprox` that takes as arguments

    1. A function `f` mapping some interval [*a*,*b*] into ˠ
    2. Two numbers `a` and `b` providing the limits of this interval
    3. An integer `n` determining the number of grid points
    4. A number `x` satisfying `a <= x <= b`

    and returns the piecewise linear interpolation of `f` at `x`, based on `n` evenly spaced grid points `a = point[0] < point[1] < ... < point[n-1] = b`. Aim for clarity, not efficiency.

12. **(*) Binomial distribution and bond portfolio**

    The **binomial random variable** *Y*› *Bin*(*n*,*p*) represents the number of successes in *n* binary trials, where each trial succeeds with probability *p*. Without any import besides `from numpy.random import uniform`, write a function `binomial_rv` such that `binomial_rv(n, p)` generates one draw of *Y*. Use the function `binomial_rv` to model the following situation. Imagine that we have a bond portfolio that contains three bonds each with a principal balance of $1,000,000 and each with a maturity date one year hence. If at maturity the bond does not default we will receive the principal balance plus 15.00% simple interest. If at maturity the bond does default we will receive the recovery on that bond which is expected to be 40% of the principal balance. Each bond has an annual default probability of 0.10 and defaults are independent. Your script must generate the value of total return (which is a random variable), in one years time.

Hint: If *U* is uniform on (0,1) and *p*a (0,1), then the expression `U < p` evaluates to `True` with probability *p*.