Note: Based on a exercise sheet from Secure Software Engineering 2016.

2.1 Printing/Escaping (Warm Up)

Write a function escape which contains 5 print statements, each of which prints one of the following strings:

1. Hello World
2. He said: "Hello World"
3. He said: "Hello World. I'm a python programmer."
4. He said: "Printing a Backslash '\' is annoying."
5. She said: "Printing two Backslashes '\\' is even more annoying."

Solution:

```
def escape():
   print("Hello World")
   print("He said: \"Hello World.\"")
   print("He said: \"Hello World. I'm a python programmer.\"")
   print("He said: \"Printing a Backslash '\\' is annoying.\"")
   print("She said: \"Printing two Backslashes '\\\\' is even more annoying.\"")
```

2.2 Iteration (More Warm Up)

Write a function hundred which prints the numbers from 0, . . . , 100. Each number should be on its own line.

Solution:

```
def hundred():
   for x in range(101):
       print(x)
```

2.3 Fizz Buzz

Write a function fizzbuzz that prints the numbers from 0 to 100. But for multiples of three print "Fizz" instead of the number and for multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Solution:

```
def fizzbuzz():
   for i in range (101):
       if i % 3 == 0 and i % 5 == 0:
           s = " FizzBuzz "
```

```
        elif i % 3 == 0:
            s = " Fizz "
        elif i % 5 == 0:
            s = " Buzz "
        else :
            s = i
        print(s)
```

2.4 ASCII

Write a function largest which takes as input a string and returns the largest integer value which belongs to a character in the string. For the string `"AbcdefgZz"` for example, largest should return 122 since `ord("z") == 122` is the largest integer representation of a character in that string. You may assume that the function only gets non-empty strings as input, which is why you do not have to deal with that case.

Solution:

```
def largest(s):
    ret = -1
    for c in s:
        if ord(c) > ret:
            ret = ord(c)
    return ret

# Better (Pythonic) Solution
def largest(s):
    return max(map(ord, s))
```

2.5 Character Frequency

Write a function freq which takes as input a string and returns a dictionary which maps each character of that string to the number of occurrences. For the string "hello world" for example, `freq` should return the dictionary `{'l': 3, 'o': 2, ' ': 1, 'e': 1, 'd': 1, 'h': 1, 'r': 1, 'w': 1}`.

Solution:

```
def freq(s):
    ret = {}
    for c in s :
        ret[c] = ret.get(c, 0) + 1
    return ret
```

2.6 List Parsing

Write a function parselist which takes as input a string representation of a list of integers and returns the list object which corresponds to that representation. For the string `"[0, 1, 1, 2, 3, 5, 8]"` for example, the function should return the list object `[0, 1, 1, 2, 3, 5, 8]` . You may assume a valid

input, i.e. there are only numbers on the list and no other characters than digits, spaces, square brackets and spaces and so on. However, there might be multiple spaces at the beginning, the end or in the middle of the string as for example in `" [ 0, 1 , 2, 3]  "` . Your function should be able to deal with those cases as well.

Solution:

```
def parselist (s):
   return [int(c) for c in s.replace("[","")
      .replace("]","").split(",") if c != " "]
```

2.7 String Reversions

Write a function `reverse`, which reverses a string **without** using the built in reverse function.

Solution:

```
def reverse(s):
   return " ".join(s[i] for i in range(len(s)-1,-1,-1))

# Better : using extended slice syntax
def reverse(s):
   return s[::-1]
```

2.8 String Obfuscation

In this exercise, we want to create two functions which obfuscate and deobfuscate text. The obfuscation works as follows: for each character in the text, we lookup the integer value of the character. Then we transform this integer to binary and create a bitstring out of that binary. So, for examples the character `"H"` becomes the string `"1001000"`, since the integer value of `"H"` is 72, which is 1001000 in binary. We do this for each character of the string and join all the bitstring with the space character as a delimiter. So for the string `"5Hello"` for example, the obfuscated version will be `"0110101 1001000 1100101 1101100 1101100 1101111"`. Note that you may assume that the string only consists of characters whose integer value is in the range [0, 127] 2 . Therefore, each character can and should be represented with a bitstring of length 7. You should therefore pad the string with “0” s in case the integer value is so small that it could be represented with less than 7 bits as for example in the case above with the `"5"` character. So `"110101"` would be wrong, while “0110101” would be correct.

Your tasks are:

1. A function `obfuscate`, which implements this obfuscation and
2. a function `deobfuscate`, which does the reverse.

Solution:

```
def obfuscate(s):
   return " ".join(bin(ord(c))[2:].rjust(7,"0") for c in s)
def deobfuscate (s):
   return " ".join(chr(int(w,2)) for w in s.split())
```

2.9 Bit Magic

Write a function popcount which takes as input a positive integer and returns the number of 1 bits which are contained in the number in binary. For examples `popcount(5)` should be 2 as 5 is 101 in binary.

Solution:

```
def popcount(n):
   ret = 0
   while n > 0:
       ret += n & 1
       n > >= 1
   return ret

# This also works.
def popcount(n):
   return bin(n).count("1")
```

2.10 Serialization

Sometimes it is desirable to store objects to files; this process is called serialization. Therefore, we implement a custom serialization function in this exercise. The task is to serialize a list of persons. A person is represented with a triple which contains, the first name, the last name and the age in years. The following is an example of such a list: `[("Martin", "Hellman", 70), ("Whitfield", "Diffie", 71), ("Leslie", "Lamport", 75)]`. To serialize this list, we need to create a binary representation of it and save it to disk. Your tasks are as follows:

1. Write a function serialize which takes as input a list of persons and a file path and serializes the list and writes it to the file specified by the path.
2. Write a function deserialize , which reverts this process, i.e. it takes as input a file path of a serialized list and returns the list object of people which the file corresponds to.

You are free to design the file format as you wish (try to make it space-efficient, i.e. minimize the file size). Here are some notes:

- You should be able to handle ages from 0 to 255.
- The maximum length for the first and the last name is 255.
- Using the `pickle` module is considered cheating.
- Use the `struct` module as shown in the tutorial.

Solution:

```
import struct

def serialize(persons, filename):
   with open(filename, "w") as f:
       for first, last, age in persons:
           lf = len(first)
           ll = len(last)
```

```
            fmt = "BB{}s{}sB".format(lf, ll)
            out = struct.pack(fmt, lf, ll, first.encode("utf-8"),
                              last.encode("utf-8"), age)
            f.write(out.decode("utf-8"))

def deserialize(filename):
    persons = []
    with open(filename) as f:
        while True:
            data = f.read(2)
            if not data:
                break
            lf, ll = struct.unpack("BB", data.encode("utf-8"))
            fmt = "{}s{}sB".format(lf, ll)
            p = struct.unpack(fmt, f.read(lf + ll + 1).encode("utf-8"))
            person = (p[0].decode("utf-8"), p[1].decode("utf-8"), p[2])
            persons.append(person)
    return persons
```