General Information & Communication Technology I (350101) Fall 2015

Michael Kohlhase
Jacobs University Bremen
http://kwarc.info/kohlhase

September 9. 2015

Abstract

This document accompanies the python tutorial in GenICT. It contains a sequence of simple (but increasingly difficult) problems designed to practice the art of recursive programming.

The problems in this document are intended for self-study, they are supplied with solutions.

As most students have never programmed python (or programmed at all), most students only manage to solve the first five. This is to be expected, and sufficient, since the purpose of the tutorial is to get students started at all and jointly remove the first roadblocks, so that they can continue alone (or in groups) after that.

The problems from the first three assignments should be doable after the first two lectures on python, the later problems can be tackled as the lecture progresses.

Practice Problems 1: Python Basics

Problem 1.1 (Maximum)
Define a function `mymax` that takes two numbers as arguments and returns the larger of them. Use the `if-then-else` construct available in python.

Note: It is true that Python has the `max` function built in, but writing it yourself is nevertheless a good exercise.

Solution: We make an if statement and check which number is bigger.

```
def mymax(a, b):
    if a > b:
        return a
    else:
        return b
```

It is easily possible to write this in one line.

```
def mymax_smart(a, b):
    return a if a > b else b
```

Problem 1.2 (Sum to 10)
Write a function `makes10` that takes two integer arguments and returns `True`, iff their sum is 10.

Solution:
The idea is to define a function and then make an if statement.

```
def makes10(a, b):
    if a+b == 10:
        return True
    else:
        return False
```

We can also condense this into one line:

```
def makes10_smart(a, b):
    return a+b == 10
```

Problem 1.3 (Positive/Negative)
Write a function `posneg` that takes two arguments and returns `True`, iff one is negative and one is positive.

Solution: We can just use and if statement:

```
def posneg(a, b):

    if (a < 0 and b > 0) or (a > 0 and b < 0):
        return True
    else:
        return False
```

Alternatively can also use the mathematical property that the product of two numbers is negative iff exactly one of them is negative:

```
def posneg_smart(a, b):
  return a*b < 0
```

Problem 1.4 (Squares)
Write a python program that prints all the square numbers from 1 to 10.

Solution:

```
for i in range(1, 11):
    print(i*i)
```

We can also use list comprehensions:

```
[print(i**2) for i in range(1, 11)]
```

Problem 1.5 (Printing a Square of Stars)
Write a function `printSquare` that takes an integer n as argument and prints a square with $n \times n$ stars. For instance `printSquare(6)` would yield

```
******
******
******
******
******
******
```

Solution: We can do a straightforward loop:

```
def print_square(n):
    for i in range(n):
        for j in range(n):
            print("*",end="") # The end argument stops print from printing a newline or space
        print() # Print a newline
```

We can also use string multiplication.

```
def print_square_smart(n):
    print((("*"*n+"\n")*n)[:-1])
```

Problem 1.6 (Squares to file)
Write a program that prints all the square numbers from 1 to 10 to a file named `squares.txt`.

Solution:

```
def squares_to_file():
    with open("squares.txt", "w") as squares_file:
        for i in range(1, 11):
            print(i*i, file=squares_file)

def squares_to_file_smart():
    open("squares.txt", "w").write("\n".join([str(i*i) for i in range(1, 11)]))
```

Problem 1.7 (Membership)
Write a function `member` that takes a value (i.e. a number, string, etc.) x and a list l of values and returns `True` if x is a member of l and `False` otherwise.

Note: For example `member(1,[1,2,3])` returns `True`.

Note: Note that this is exactly what the in operator does, but for the sake of the exercise you should pretend python lacks this operator. A `for/in` loop is OK though.

Solution: We can iterate through the list and check each element. If we do not find the element we are searching for we return False.

```
def member(x, the_list):
    for element in the_list:
        if element == x:
            return True
    return False
```

Python also has a built-in operator to do this in a smarter way:

```
def member_smart(x, the_list):
    return x in the_list
```

Problem 1.8 (Guessing Numbers)
Write a program where you guess a number. The program should draw an integer number (use `n = random.randint(1, 100)`) and then you should guess the number by inputting a number on the keyboard. The program should tell you then whether the your guessed number is smaller or bigger than the hidden number, and let you try again until you have sucessfully guessed the number.

Note: Keep in mind that you need to import the `random` module.

Solution:

```
def guess_me():
    # we get a random number
    the_number = random.randint(1, 100)

    print("I picked a random number between 1 and 100. ")

    # we make a first guess
    while True:
        the_guess = input("Guess: ")
        try:
            the_guess = int(the_guess)
            break
        except:
            print("That is not a number. Try again. ")

    # while the guess is not correct, we keep guessing
    while the_guess != the_number:
        if the_guess < the_number:
            # if the guess is smaller than the number
            print("The number is bigger than", the_guess)
        else:
            # if not, it has to be bigger
            print("The number is smaller than", the_guess)

        # so we need to keep guessing
        while True:
            the_guess = input("Guess: ")
            try:
                the_guess = int(the_guess)
                break
            except:
                print("That is not a number. Try again. ")

    print("That's right, the number is", the_number)
```

It is also possible to shorten this code:

```
def guess_me_smart():
    # get a random number between 1 and 100
    the_number = random.randint(1, 100)

    print("I picked a random number between 1 and 100. ")

    # we define a function that reads a guess from the input
    def read_guess():
        while True:
            try:
                return int(input("Guess: "))
            except:
                print("That is not a number. Try again. ")

    # and then we keep reading the number until
    the_guess = read_guess()
    while the_guess != the_number:
        # we also shorten the code for printing a message.
        print("The number is %s than %d\n" % ('smaller' if the_guess > the_number else 'bigger', the_guess))
        the_guess = read_guess()

    # we can say it is the right one
    print("That's right, the number is", the_number)
```

Problem 1.9 (99 Bottles)
"99 Bottles of Beer" is a traditional song in the United States and Canada. It is popular to sing on long trips, as it has a very repetitive format which is easy to memorize, and can take a long time to sing. The song's simple lyrics are as follows:

> 99 bottles of beer on the wall, 99 bottles of beer.
> Take one down, pass it around, 98 bottles of beer on the wall.

The same verse is repeated, each time with one fewer bottle. The song is completed when the singer or singers reach zero.

Your task here is write a python program capable of generating all the verses of the song.

Solution: We can just iterate over the number 0 to 99 in reverse order and then for each number print the appropriate line.

```
def ninety_nine_bottles():
    """Prints the lyrics to "99 Bottles of Beer". """
    for i in range(99, 0, -1):
        print(i, "bottles of beer on the wall,", i, "bottles of beer.")
        print("Take one down, pass it around,", i-1, "bottles of beer on the wall.")
```

Using list comprehensions we can write this in one print statement

```
def ninety_nine_bottles_smart():
    [print("%d bottles of beer on the wall, %d bottles of beer. \n" +
           "Take one down, pass it around, %d bottles of beer on the wall. "
           % (i, i, i-1)) for i in range(0, 99, -1)]
```

Problem 1.10 (Recognizing Palindromes)
Define a function `palindrome` that recognizes palindromes (i.e. words that look the same written backwards). For example, `palindrome("radar")` should return `True`.

Solution:

```
def is_palindrome(word):
  # first we need to reverse the word
  # here we treat the string as a list, reverse that and then put it back together
  reversed_word = "".join(reversed(word))

  # now we check if the reversed word is equal to the normal word
  return word == reversed_word
```
