IWGS – Informatische Werkzeuge in den Geistes- und Sozialwissenschaften – WS 2018/19
Assignment 0: Happening
– Given 15.11.2018 –

Problem 0.1 (Echo)

Write a program which asks the user for input (hint: use the input function) and then repeats the user's answer by printing it.

Solution: This is a simple exercise. We use the input function (with a string prompt as an argument), save the result in a variable, and then print that variable. Done.

```
echo = input("Please enter some text: ")
print(echo)
```

Problem 0.2 (Ifs)

Create a variable and assign it an arbitrary whole number. Afterwards, create an if-statement and test the following conditions: If the value stored in your variable is greater than zero, print "The number is positive". If not, print "The number is negative".

Test your program with different values! Is the printed statement always correct? What happens if the value of your variable is exactly zero? How can you amend your program, such that it prints "The number is zero", if it is zero, and only prints one of the other two statements, if it is not?

Solution: The first bit is relatively easy. Look below for an idea on how to write if-then-else-statements in python. It is important that you don't forget the colon after the boolean expression and don't forget to indent the lines after the if and the else. Otherwise, it will not work.

```
nat = 1337

if nat > 0:
    print("The number is positive")
else:
    print("The number is negative")
```

To finish the exercise, we need a third condition, that checks if the number is exactly zero. We can do this by including an **elif**. Another way would have been to have two *nested* if blocks.

```
if nat > 0:
    print("The number is positive")
elif nat == 0:
    print("The number is zero")
else:
    print("The number is negative")
```

Problem 0.3 (Loops)

Write a program, which prints all numbers between 0 and 10. Use a while-loop. Can you amend your program, such that it prints the numbers in reverse order, i.e. the numbers printed are $9, 8, 7, 6, 5, \ldots$?

Solution: A while-loop keeps running, as long as the boolean condition it is given is still true. So we create a variable (called i here) and set it to 1 as a starting value. Then, while i is still less than 10, we print i.

Don't forget to increment (increase by 1) the variable before the loop repeats, or you will be stuck in an endless loop that prints 1 forever.

```
i = 1
while i < 10:
    print(i)
    i = i + 1
```

To print the numbers in reverse, instead of starting at one and incrementing, we can start at 9 and decrement (decrease by one) the variable every time.

```
i = 9
while i > 0:
    print(i)
    i = i - 1
```

As is so often the case, there are many solutions and many ways of doing a certain thing. This is a solution that leaves the loop untouched and still does the correct thing.

```
i = 1
while i < 10:
    print(10-i)
    i = i + 1
```

Problem 0.4 (Lists)
Create a list containing the names of a few friends of yours. Write a for-loop, which for each entry in the list prints a greeting message containing the name. For example, given a list with the names "Paul" and "Maria" your program should print "Hello Paul" and "Hello Maria".

Solution: To create a list in python, you can use square brackets around all elements you wish to include (in this case, strings), separated by commas (look below for an example). Save the result to a variable, if you want to use it later in the program.

If you then use a for-loop like below, the new variable you introduced (called friend here) will take the value of all elements of the list, one after another, starting with the first. You can then (in the so-called *body* of the function, which must be indented) do something that refers to the variable. Like printing a greeting. The loop will execute the same thing for all elements of the list that you are looping over.

```
all_friends = ["Paul", "Amir", "Maria", "Alessia", "Max"]

for friend in all_friends:
    print("Hello", friend)
```

Problem 0.5 (Strings)
Create a variable which contains the following text (feel free to copy from Wikipedia instead of typing it out):

> "Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales."

Count how often the letter 'p' occurs in the text (hint: search for the count function on the internet). Print the number of occurrences. Right now, your program only counts the number of lower-case 'p's. Revise your program, such that both 'p' and 'P' are counted.

Solution: The count function is a built-in function that can be called on any string (by using dot-notation). It takes (at least) one argument, the substring that is to be searched for, and returns the number of occurences.

```
text = "Pythong is an interpreted ..." # and so on
print(text.count("p"))
```

If we want to count both upper- and lowercase "p"s, we have multiple options. We could, for example, save both results in variables and add them up.

Or we could call lower (another in-built function on strings). This will make the entire string lowercase. Now, when we search for the lowercase "p", this will also count those that used to be uppercase.

```
print(text.lower.count("p"))
```

Problem 0.6 (Dictionaries)

Create a dictionary which associates names with email addresses. For example, create an entry with key `"Jonas"` and value `"jonas.betzendahl@fau.de"` and one with key `"Philipp"` and value `"philipp.kurth@fau.de"`. Add as many entries as you like.

Then, ask the user for a name (hint: use the input function). Look up the corresponding email address in your dictionary and print it. What happens if the user enters a name, which is not in the dictionary?

Solution: You can write a dictionary in python with curly braces, like so:

```
teachers = { "Jonas" : "jonas.betzendahl@fau.de",
             "Philipp" : "philipp.kurth@fau.de",
             "Michael" : "michael.kohlhase@fau.de" }
```

It is important to use curly braces (not square ones, that is lists!) and separate all entries (key-value-pairs) with commas. Now that we have the dictionary, we need to access it. We can ask the user for a string with input and then access the dictionary "at that string" (this is another way of saying we want the value associated with that string) by using square brackets.

```
name = input("Whose email would you like? ")
email = teachers[name]
print(email)
```
