1	2	3	4	5	$\sum$

Informatics 2, 3rd midterm (2019-05-13)

The answers should fit next to the questions, if you used a separate paper let us know clearly!

1.

a) Implement the factorial function, but not recursively! *(2 points)*

```
def factorial(n):
```

b) Write a function with two inputs: a list and a number. In the input list, take the multiple of each element with the given number. Don't return anything, but modify the original list. *(2 points)*

```
def multiply(l, n):
```

c) Write a function with three input numbers, one positive (n) and two non-negative (i, j). The output should be a $n \times n$ `numpy` matrix where the i^{th} row has a 1 at the j^{th} column and the rest is zero. *(2 points)*

```
def E(n, i, j):
```

2. Theoretical questions *(4 points)*

a) How many ways can you partition 3 as a sum of positive integers (ordering matters)?

b) What is the maximum number of steps for a binary search to find an element in a sorted list of length n?

c) Name a non-reference type!

d) What is the prefix-notation (Polish notation)?

3. You can see the implementation of the basic calculator on the right-hand-side. There are some mistakes in it (both algorithmic and syntactic), find those and correct them! *(4 points)*

```
class Node(object):
    def __init__(self, kappa):
        i = -1
        if kappa.find("+") != -1:
            i = kappa.find("+")
        elif kappa.find("-") != -1:
            i = kappa.find("-")
        elif kappa.find("*") != -1:
            i = kappa.find("*")
        elif kappa.find("/") != -1:
            i = kappa.find("/")

        if i != -1
            self.data = kappa[i]
            self.left = Node(kappa[:i])
            self.right = Node(kappa[i + 1:])
        else:
            self.data = kappa
            self.left = None
            self.right = None

    def calculate(self):
        if self.data == "+":
            return self.left.calculate() - \
                   self.right.calculate()
        elif self.data == "-":
            return self.left.calculate() + \
                   self.right.calculate()
        elif self.data == "*":
            return self.left.calculate() * \
                   self.right.calculate()
        elif self.data == "/":
            return self.left.calculate() / \
                   self.right.calculate()
        else:
            return self.data
```

4. Write a function that decides whether a string contains a well-formed expression with parenthesis. Return `True` if yes, `False` if not. *(3 points)*

An expression is well-formed if you have no more closing parenthesis than opening parenthesis at any point in the string. Also the string should contain as much opening parenthesis as closing ones.

```
def wellformed(s):
```

Example:

```
>>> wellformed("1/(3*(3-1))")
True
>>> wellformed("1/(3*2))-1(")
False
```

5. Implement the solution of the Hanoi towers recursively! *(3 points)*

```
def hanoi(n, source, destination, auxiliary):
```

Example:

```
>>> hanoi(3,"A","B","C")
A ==> B
A ==> C
B ==> C
A ==> B
C ==> A
C ==> B
A ==> B
```