Aufgabenblatt 3

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihre Python-Datei bis spätestens **Freitag, 24.04.2020 20:00 Uhr** in TUWEL hoch.
- Beachten Sie bitte folgende Punkte
  - Ihre Programme müssen ausführbar sein, d.h. einzelne Programme sollten z.B. keine Sytaxfehler oder Typfehler enthalten.
  - Bei kleinen logischen Fehlern (falsche Ergebnisse) werden wir keine Punkte abziehen. Daher sollten Sie immer versuchen, alle Aufgaben vollständig abzugeben.
  - Ihre Programme sollten nur Konstrukte verwenden, die bisher in der Vorlesung vorgekommen sind. Sie sollten daher keine speziellen Aufrufe verwenden, die möglicherweise einzelne Aufgaben abkürzen.
  - Die Angabedatei muss nicht unbenannt werden.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Funktionen
- Rekursion

Aufgabe 1 (1 Punkt)

In dieser Aufgabe müssen Sie einzelne Funktionen definieren. Bei jeder Funktion finden Sie eine Beschreibung von Annahmen. Sie können immer davon ausgehen, dass diese Annahmen zutreffen und müssen keine spezielle Behandlung für die übergebenen Argumentwerte implementieren (z.B. überprüfen ob ein Wert > 0 ist). Jede Funktion sollte auch zumindest mit den gegebenen Beispielen getestet werden.
Folgende Funktionen sind zu implementieren:

1. Eine Funktion `print_pattern(n)`, die folgendes Muster ausgibt: Es wird eine Linie von Sternen ausgegeben, danach folgt ein Text-Smiley (durch Leerzeichen von den Linien getrennt) und danach wieder eine Linie von Sternen. Die Summe der Sterne entspricht n und wird bei geradem n gleichmäßig auf die linke und rechte Seite aufgeteilt. Bei ungeradem n wird links ein Stern weniger ausgegeben.
   Annahme(n): n ist eine ganze Zahl, $n > 0$.
   Beispiele:
   `print_pattern(5)` liefert `** :-) ***` als Ausgabe
   `print_pattern(6)` liefert `*** :-) ***` als Ausgabe
   `print_pattern(7)` liefert `*** :-) ****` als Ausgabe

2. Eine Funktion `mysum(low, high)`, die die Summe der ganzen Zahlen im Intervall [low, high] zurückliefert.
   Annahme(n): *low* und *high* sind ganze Zahlen, $low < high$.
   Beispiele:
   `mysum(10, 20)` liefert 165 zurück
   `mysum(0, 10)` liefert 55 zurück
   `mysum(-10, 10)` liefert 0 zurück

3. Implementieren sie eine Funktion `powers_of_two(low, high, i)`, die jede i-te Zweierpotenz mit Exponenten im Intervall [low, high] ausgibt. Wird für i kein Argument angegeben, dann wird 1 als Default-Wert verwendet.
   Annahme(n): *low*, *high* und i sind ganze Zahlen, $0 =< low < high$, $i > 0$.
   Beispiele:
   `powers_of_two(2, 5)` liefert `4 8 16 32` als Ausgabe
   `powers_of_two(4, 10, 2)` liefert `16 64 256 1024` als Ausgabe
   `powers_of_two(10, 30, 6)` liefert `1024 65536 4194304 268435456` als Ausgabe

Aufgabe 2 (1 Punkt)

In dieser Aufgabe müssen Sie einzelne Funktionen definieren. Bei jeder Funktion finden Sie eine Beschreibung von Annahmen. Sie können immer davon ausgehen, dass diese Annahmen zutreffen und müssen keine spezielle Behandlung für die übergebenen Argumentwerte implementieren (z.B. überprüfen ob ein Wert > 0 ist). Jede Funktion sollte auch zumindest mit den gegebenen Beispielen getestet werden.
Folgende Funktionen sind zu implementieren:

1. Eine Funktion `check_length(s, t)`, die die Längen der beiden übergebenen Strings überprüft und folgende Fälle unterscheidet:

   - Länge von `s` > Länge von `t`: Rückgabe ist `s`
   - Länge von `s` < Länge von `t`: Rückgabe ist `t`
   - Länge von `s` = Länge von `t`: Rückgabe ist `s + t`

   Annahme(n): s und t sind Strings
   Beispiele:
   `check_length("Hello", "Hell")` liefert `Hello`
   `check_length("Hello", "Helloa")` liefert `Helloa`
   `check_length("Hello", "Hello")` liefert `HelloHello`

2. Implementieren Sie eine Funktion `cat_strings(*texts)`, die alle Argumentenstrings (falls vorhanden) mit Bindestrichen getrennt zurückgibt.
   Annahme(n): Alle Argumente sind Strings.
   Beispiele:
   `cat_strings()` liefert einen Leerstring
   `cat_strings("Hello")` liefert `Hello`
   `cat_strings("Hello", "World")` liefert `Hello-World`
   `cat_strings("Hello", "and", "good", "bye")` liefert `Hello-and-good-bye`

Aufgabe 3 (1 Punkt)

In dieser Aufgabe müssen Sie ein Programm implementieren, das alle Primzahlen bis zu einer bestimmten Obergrenze bestimmt und untereinander ausgibt. Dazu müssen Sie folgende Funktionen definieren:

- Eine Funktion `is_prime(number)`, die bestimmt, ob eine Zahl `number` eine Primzahl ist und das Ergebnis zurückliefert. Sie dürfen dabei den Programmcode für die Primzahlberechnung aus den Folien verwenden.
  Annahme(n): $number >= 0$.

- Eine Funktion `user_input()`, die zunächst eine Begrüßung ausgibt und dann eine Obergrenze einliest und diese zurückliefert.

- Eine Funktion `generate(upper_bound)`, die als Argument die eingelesene Obergrenze übergeben bekommt und dann alle Primzahlen im Intervall [2, Obergrenze] ermittelt. Diese Zahlen werden dabei zu einem String konkateniert, der dann von dieser Funktion zurückgeliefert wird.
  Annahme(n): $upper\_bound >= 0$.

Im Hauptprogramm rufen Sie die Methoden so auf, dass zunächst die Obergrenze eingelesen wird und dann die Zahlen generiert werden. Die generierten Zahlen werden danach im Hauptprogramm ausgegeben. Ein Beispiel für den Ablauf:

```
Welcome to the GPA prime number generator!
Upper bound: 37
2
3
5
7
11
13
17
19
23
29
31
37
```

Aufgabe 4 (1 Punkt)

Sie haben unterschiedliche rekursive Funktionen gegeben. Vor jeder Funktion gibt es einen Kommentarabschnitt, in dem Sie die jeweils nachfolgende Implementierung beschreiben müssen. Behalten Sie bitte die vorgegebene Form bei:

- Aufgabe der Funktion in einem Satz beschreiben.
- Basisfall kurz beschreiben. Dabei kann es auch vorkommen, dass mehrere Basisfälle existieren.
- Rekursionsschritt kurz beschreiben.

Sie können immer davon ausgehen, dass diese Funktionen mit korrekten Argumentwerten (bei int mit Werten $>= 0$, bei Strings mit korrekten Strings) aufgerufen werden.