SoSe 2020

Prof. Dr. Margarita Esponda

Objektorientierte Programmierung

2. Übungsblatt

Lernziel: Rekursion vs. Iteration und die Anwendung von Wörterbüchern in Python.

1. Aufgabe (3 Punkte)

Eine Mersenne-Zahl ist eine Zahl der Form $2^n - 1$. Programmieren Sie nur unter Verwendung von Bit-Operatoren eine effiziente Python-Funktion, die bei Eingabe einer natürlichen Zahl n die n-te Mersenne-Zahl berechnet.

2. Aufgabe (6 Punkte)

Definieren Sie eine Python-Funktion **repeats**, die bei Eingabe zweier natürlichen Zahlen **n** und **m**, **m** Zufallszahlen zwischen **0** und **n** mit Hilfe der **randint**-Funktion generiert und die Häufigkeit des Vorkommens der einzelnen Zahlen berechnet.

a) Verwenden Sie dafür eine Python Liste, der als Rückgabewert der Funktion zurückgegeben wird.

Anwendungsbeispiele:

```
>>> reps (10, 100)
[16, 10, 7, 7, 6, 8, 13, 6, 5, 10, 12]
```

Die Interpretation des Ergebnisse ist, dass z.B. die Zahl 0 16 mal vorkam, die Zahl 1 10 mal usw.

b) Verändern Sie die Funktion, indem Sie die Berechnung der Häufigkeiten mit Hilfe eines Dictionary realisieren.

3. Aufgabe (3 Punkte)

Schreibe eine Test-Funktion, die die Ausführungszeiten der 4 Such-Funktionen aus der Vorlesung (iterative und rekursive Linear- bzw. Binärsuche) vergleicht, indem Sie 1000 zufällige Zahlen in einer Liste mit 500 sortierten Zahlen sucht.

Die Ausführungszeiten können mit Hilfe der **time**()-Funktion des **time**-Moduls gemessen werden.

4. Aufgabe (3 Punkte)

Definieren Sie eine Python-Funktion **apply_if**, die als Eingabe eine einstellige Funktion **f**, ein Prädikat **p** (Funktion) und eine Liste **xs** bekommt, und die Funktion **f** nur auf die Elemente der Liste, die das Prädikat **p** erfüllen, anwendet.

Anwendungsbeispiel:

```
>>> apply_if ( factorial, odd,  [2, 5, 7, 4, 9, 6] )
[2, 120, 5040, 4, 362880, 6]
```

Haskell-Implementierung:

```
apply_if f p [] = []
apply_if f p (x:xs) | p x = (f x): apply_if f p xs
                    | otherwise = x : apply_if f p xs
```

Verwenden Sie zum Testen die **factorial** und **odd** Funktionen aus den Vorlesungsfolien.

5. Aufgabe (4 Punkte)

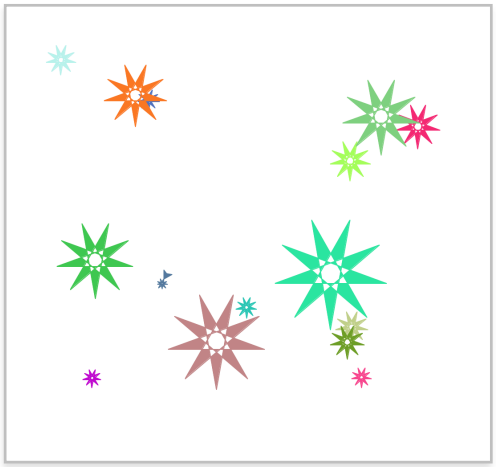
Definieren Sie eine **revDigits** Funktion, die die Ziffernfolge einer beliebigen natürlichen Zahl umkehrt. Verwenden Sie dafür eine endrekursive Hilfsfunktion.

6. Aufgabe (10 Punkte)

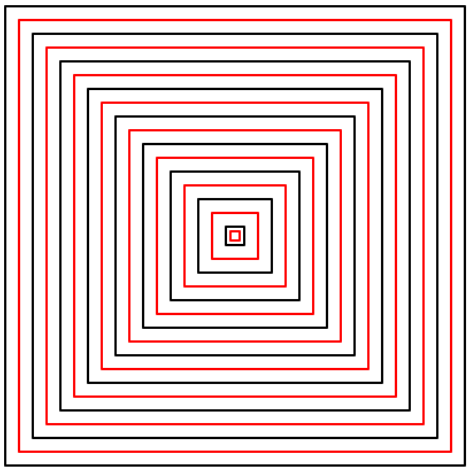
a) Programmieren Sie unter Verwendung des Turtle-Graphic Moduls von Python eine Funktion **paint_star,** die bei Eingabe von **x**, **y** und **size** Argumenten einen Stern an der **x**, **y** Position des Fensters mit Durchmesser **size** zeichnet.

b) Programmieren Sie eine **sky** Funktion, die bei Eingabe einer positive ganze Zahl **n**, **n** Sterne an zufällige **x**, **y** Positionen mit zufälligen Farben und zufälligen Größen zeichnet.

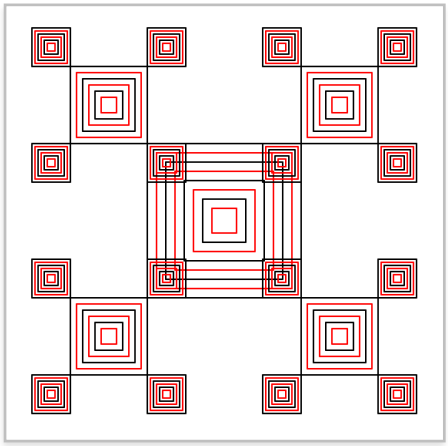
Beispiel einer möglichen Bildausgabe:



c) Definieren Sie eine rekursive **squares** Funktion, die folgendes Bildmuster zeichnet.



d) Mit Hilfe der **squares** Funktion, definieren Sie eine **fractal_squares** Funktion, die folgendes Bild erstellt.



7. Aufgabe (3 Punkte)

Definieren Sie eine **revDigits** Funktion, die die Ziffernfolge einer beliebigen natürlichen Zahl umkehrt.

Anwendungsbeispiel:

>>> **revDigits** (1345001)
1005431
>>> **revDigits** (8390100)
10938

8. Aufgabe (6 Bonuspunkte)

Programmieren Sie die **foldl** und **foldr** Haskell Funktion im Python.

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionen, die den semantischen Inhalt der Variablen oder die Funktionalität der Funktionen darstellen.
2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.
3) Kommentieren Sie Ihre Programme.
4) Verwenden Sie geeignete Hilfsvariablen und Hilfsfunktionen in Ihren Programmen.
5) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.
6) Schreiben Sie getrennte Test-Funktionen für alle 4 Aufgaben für Ihren Tutor.
7) Die Lösungen sollen in Papierform und elektronisch (KVV-Upload) abgegeben werden.