

Programmierung für Mathematiker

Übungsaufgaben

Name:

Matriculation number:

Allgemeine Anforderungen und Hinweise:

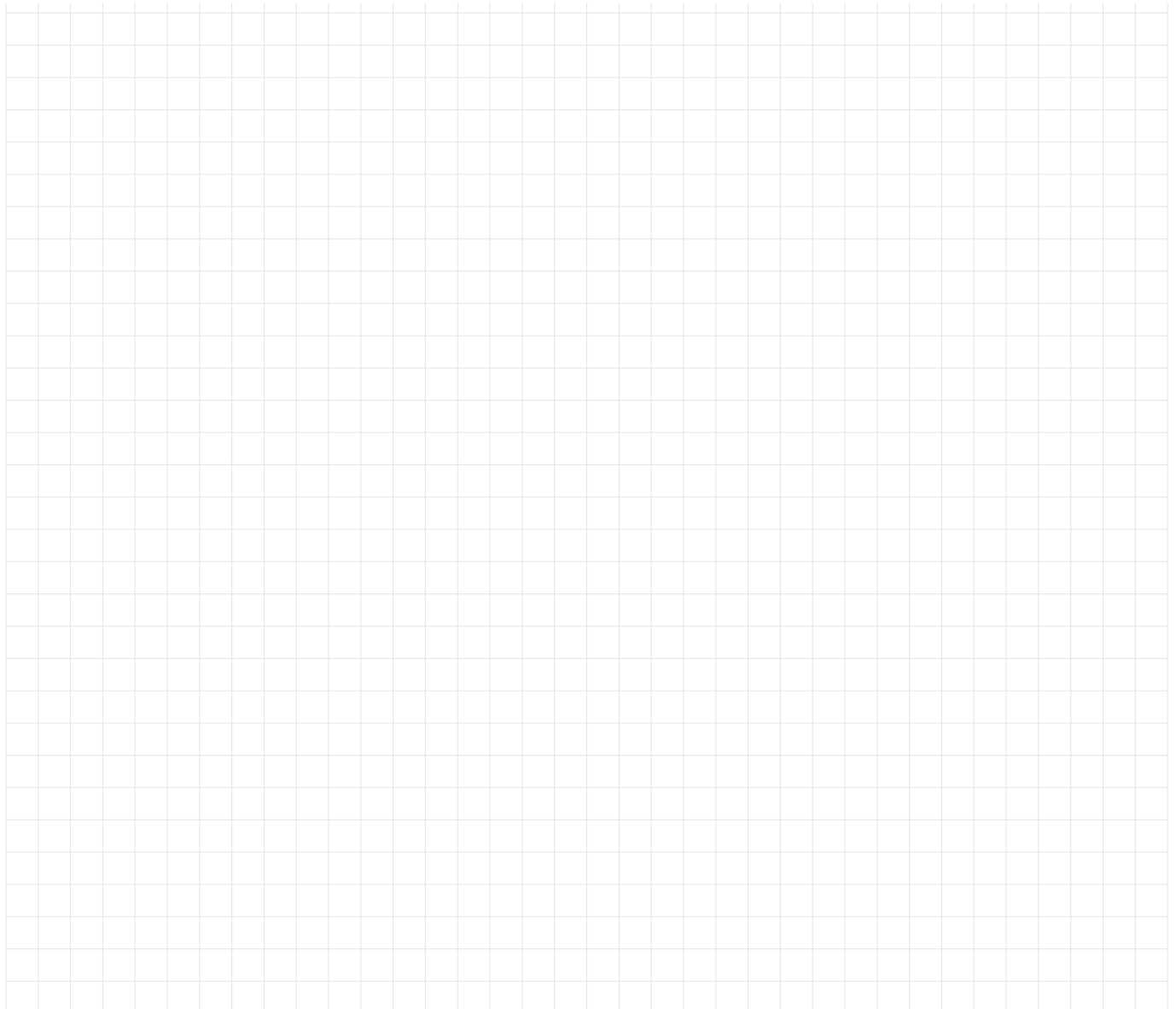
- bla
- bla
- bla

Task:	1	2	3	4	5	6	7	8	9	Total	Grade
Points:											
Total:	5	5	7	5	8	5	2	5	5	47	–

Half Precision: s10e5

Betrachten Sie die binären Gleitkommazahlen gemäß der Parameter s10e5.

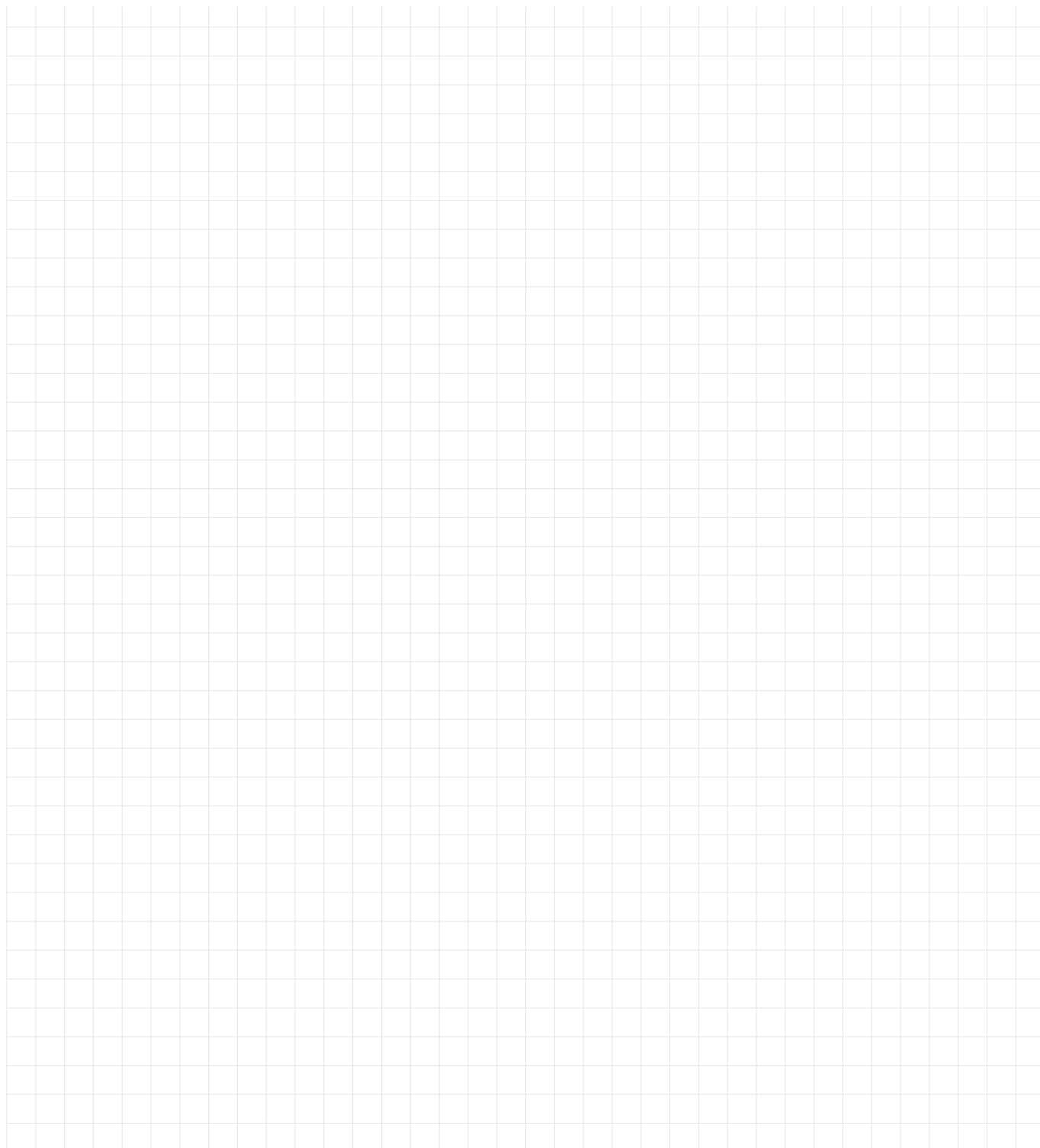
1. Ermitteln Sie den Verschiebewert B (BIAS) für den Exponenten gemäß des IEEE-754 Standards, sowie den Exponentenwertebereich e_{\min} und e_{\max} .
2. Mit vielen Bits wird eine Gleitkommazahl hier gespeichert? Erklären Sie die Bedeutung der einzelnen Bits und wie daraus die Gleitkommazahl gebildet wird.
3. Bestimmen Sie die relative Maschinengenauigkeit macheps .
4. Geben Sie die Bitmuster mit zugehörigem (ungefähren) Zahlenwert an:
 - a) Signed zero und signed infinity: $\pm 0, \pm \infty$
 - b) Kleinste und größte positive normale Zahl
 - c) Kleinste und größte positive denormalisierte Zahl (*subnormals*)
 - d) Kleinste und größte positive Nichtzahl (NaN)
 - e) 00111110000000001
 - f) $\text{fl}(-4)$
 - g) $\text{fl}(\frac{2}{3})$, wobei $\frac{2}{3} = (0.\overline{10})_2 = (0.10101010\dots)_2$



Permutation

1. Gegeben sei eine Liste von n Zahlen $\text{perm} = [a_0, a_1, \dots, a_{n-1}]$ mit der Eigenschaft, dass sie genau die Zahlen $0, 1, 2, \dots, n-1$ in beliebiger Reihenfolge enthält. Die Python-Funktion `invert(perm)` gibt eine Liste zurück, in der für alle $i = 0, \dots, n-1$ an der Stelle i die Zahl i an der Stelle a_i liegt. Implementieren Sie einen Test für die Funktion `invert(perm)`.
2. Implementieren Sie die Funktion `test_invert()`

Hintergrund: Diese Liste könnte eine [Permutation](#) darstellen. Permutationen sind fundamentale Operationen, die zum Beispiel eine wichtige Rolle in der Lösung linearer Gleichungssystem spielen.



Maximum einer Liste Die Funktion `maximum(values)` gibt das Maximale Element der Liste `values` zurück.

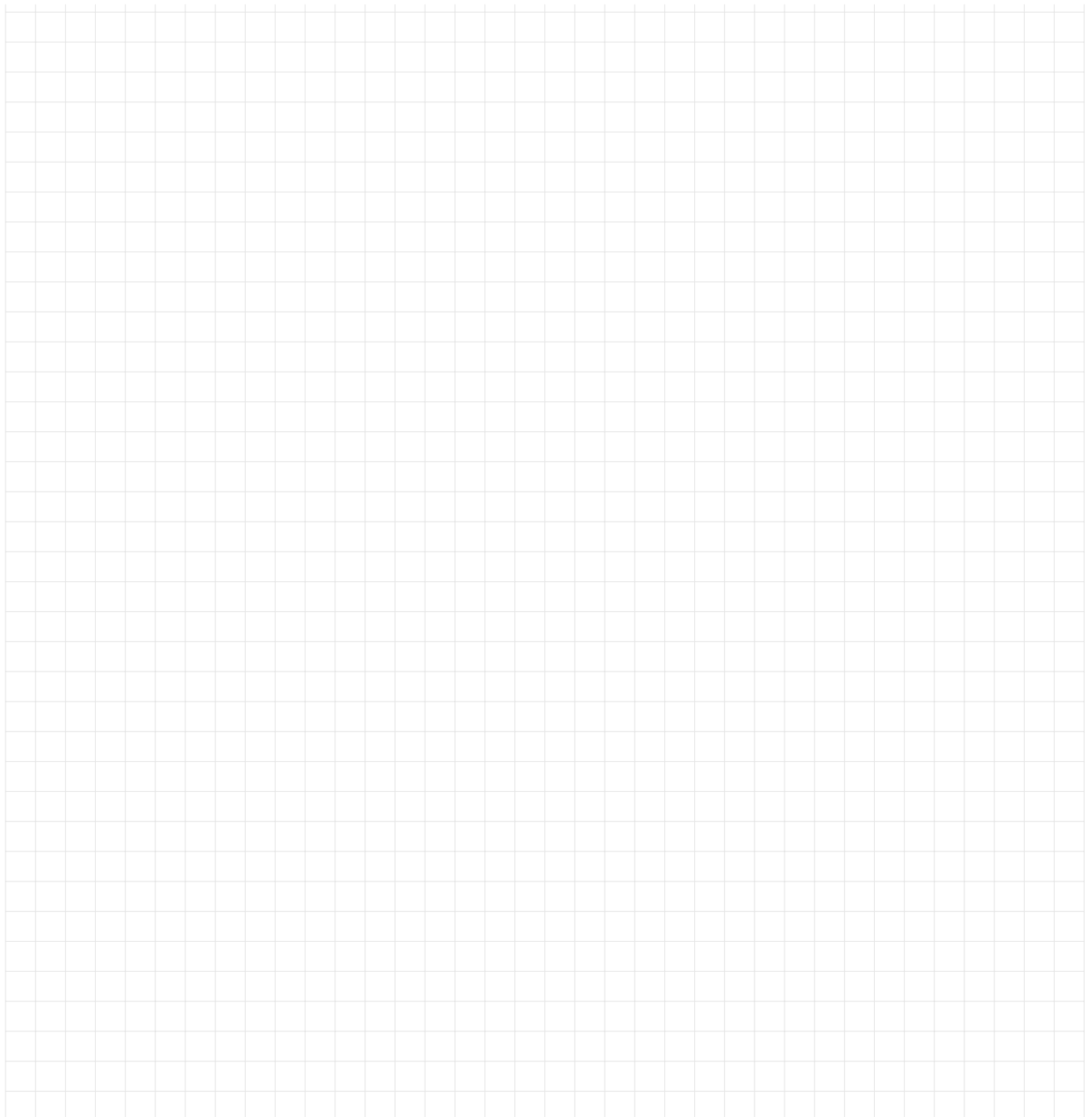
1. Implementieren Sie `test()` und `maximum(values)`.
2. Die Funktion `maximum` soll auch prüfen, ob die eingegebene Liste mindestens ein Element enthält und anderenfalls einen Fehler produzieren. Probieren sie aus, was der Python Befehl

```
1 assert Wert "Hier gibt es ein Problem!"
```

für `Wert=False` oder `True` tut und versuchen Sie ihn zu nutzen.

3. **Bonus:** Was bewirkt die (optimierte) Ausführung des Programms via `python -O` hier?

Hintergrund: Im Bereich des wissenschaftlichen Rechnens sind Fehlertoleranz und Performanz manchmal Gegensätze. Die eleganteste Synthese dieser Gegenspieler ist dann ein Code, der sich umschalten lässt.



Callback Funktion und **kwargs

1. Implementieren Sie den unten angegebenen Code. Tätigen Sie die Aufrufe `algorithm(1)` und `algorithm(1, callback=callback)`. Was passiert? Fügen Sie Docstrings hinzu, die kurz das Verhalten der Funktionen `algorithm` und `callback` erklären.
2. Nun soll die Funktion `callback()` nur dann etwas ausgeben, wenn `xstart` größer als ein gewisser `threshold` ist. Fügen Sie diese Eigenschaft und das Schlüsselwortargument `threshold` der Funktion `callback()` hinzu. Verändern Sie dabei die äußere Funktion `algorithm()` nicht! Welche Parameter übergeben Sie an `algorithm()`, wenn Sie nur Iterierte größer als `threshold=1.6` gedruckt bekommen möchten?
Hinweis: Verwenden Sie einen optionalen Parameter: `callback(xstart, threshold=0.0)`.
3. Erläutern Sie, welchen praktischen Vorteil `**kwargs` in verschachtelten Funktionsauswertungen haben kann.

```
1 from random import random
2
3 def algorithm(xstart, callback=None, **kwargs):
4     for _ in range(10):
5         if callback:
6             callback(xstart, **kwargs)
7             xstart += random() - 0.5 # der Aufruf random() generiert eine Zufallszahl in [0,1]
8     return xstart
9
10 def callback(xstart):
11     print(xstart)
```

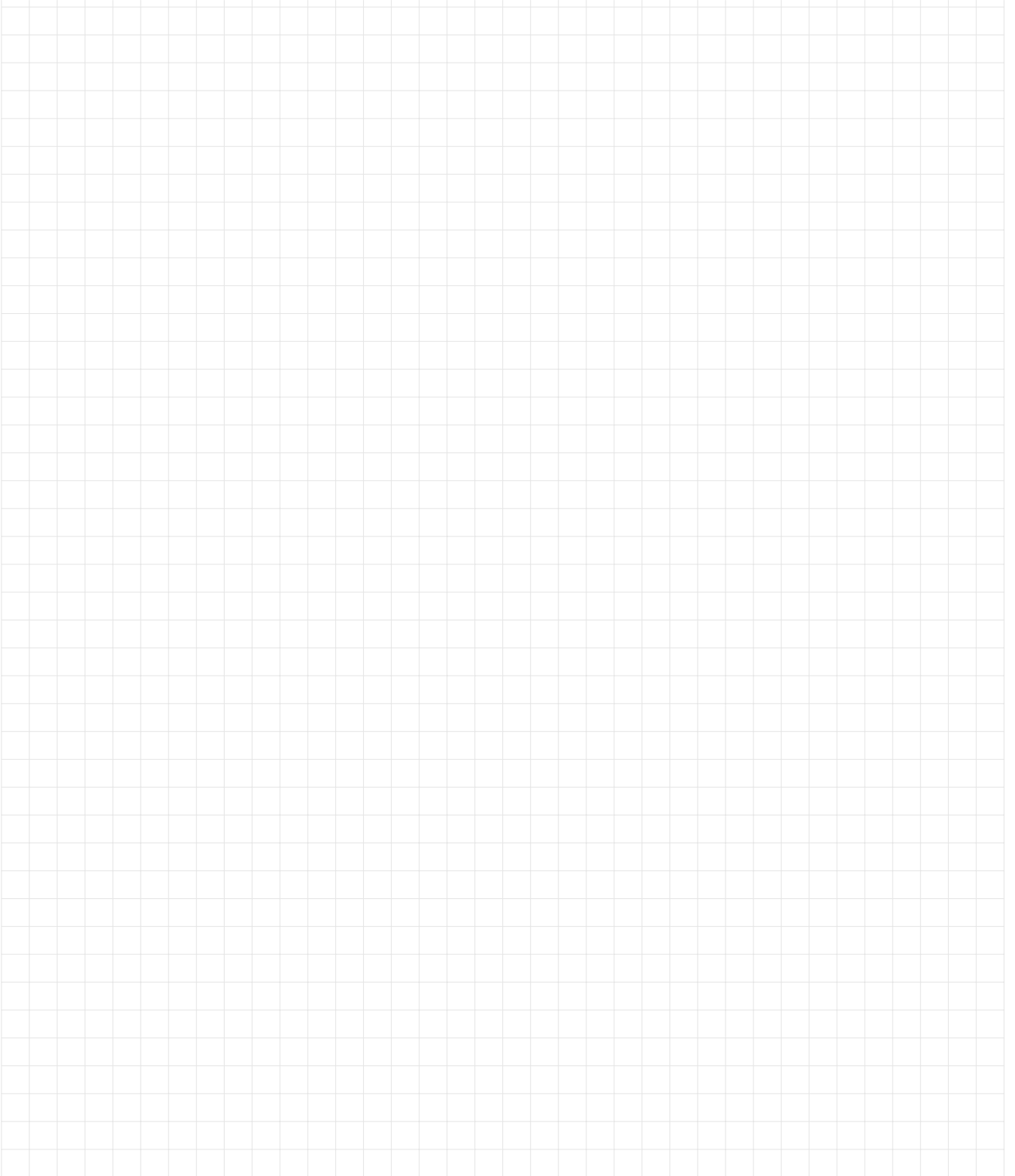

Umwandlung: Dezimal nach Beliebig

Schreiben Sie eine Funktion `dezToAny(d, basis)`, die eine Zahl `d` und eine Basis `basis` als Integer entgegennimmt und die Koeffizienten `a` der Darstellung der Zahl in der Basis als String zurückgibt. Es genügt hierbei, wenn Zahlensysteme bis einschließlich der Basis 16 erlaubt sind.

Beispiel

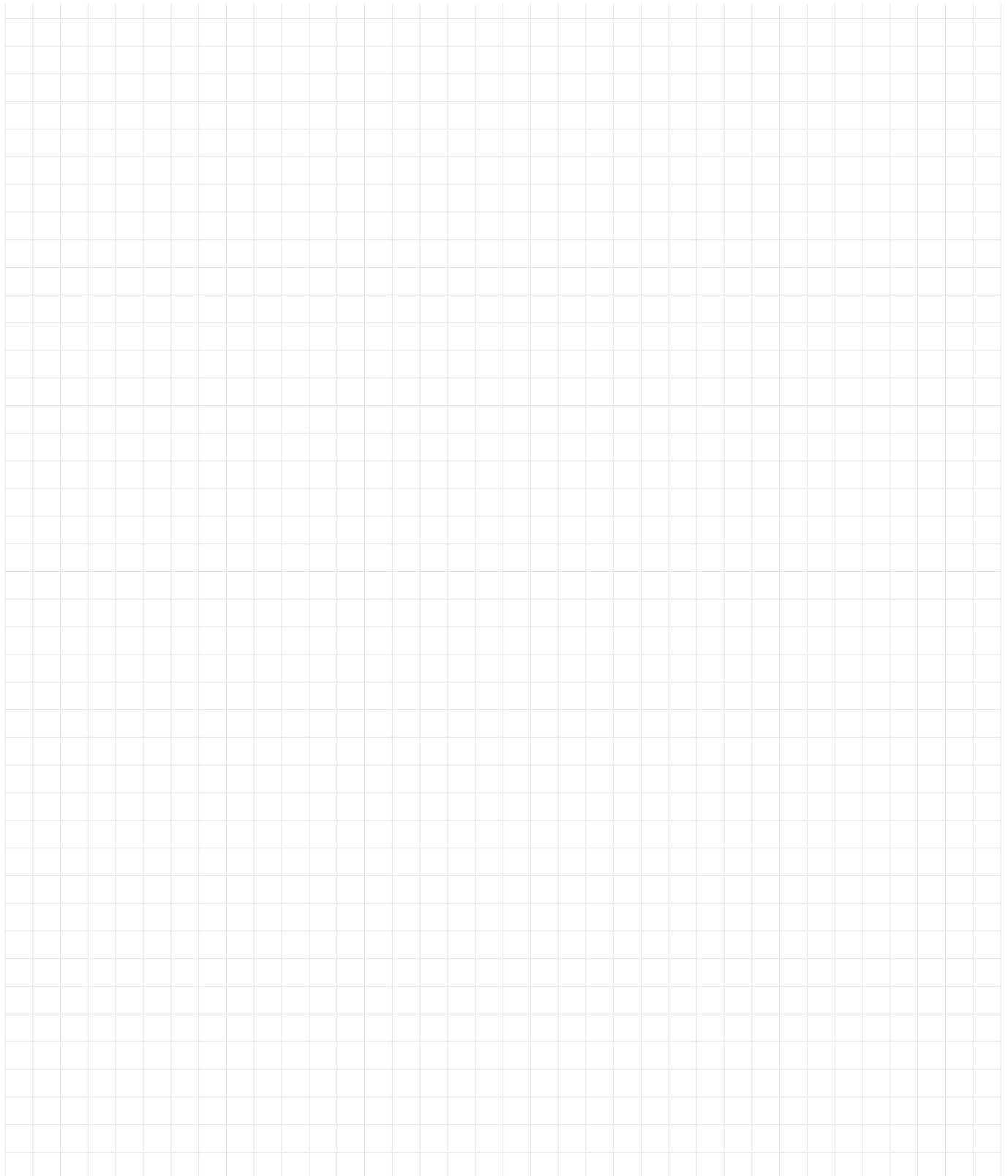
```
a = 15, basis = 16
```

```
>>> dezToAny(a, basis) = 'f'
```



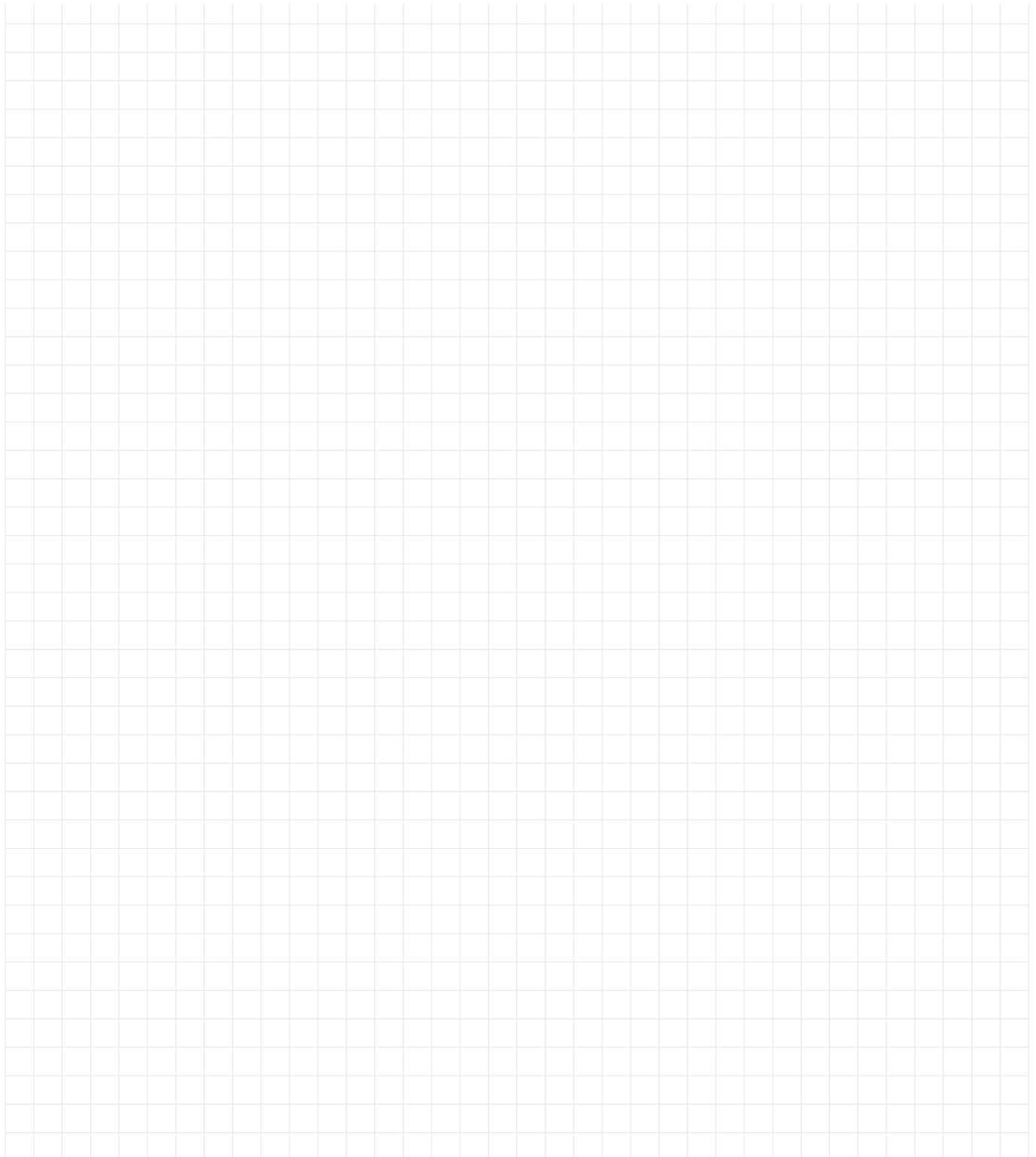
Der “moderne” Euklidische Algorithmus

1. Schreiben Sie eine Python-Funktion, die den größten gemeinsamen Teiler zweier Zahlen mit Hilfe des “modernen” Euklidischen Algorithmus (in der iterativen Variante) bestimmt.
2. Implementieren Sie die rekursive Variante des “modernen” Euklidischen Algorithmus.
3. Testen Sie Ihren Code.



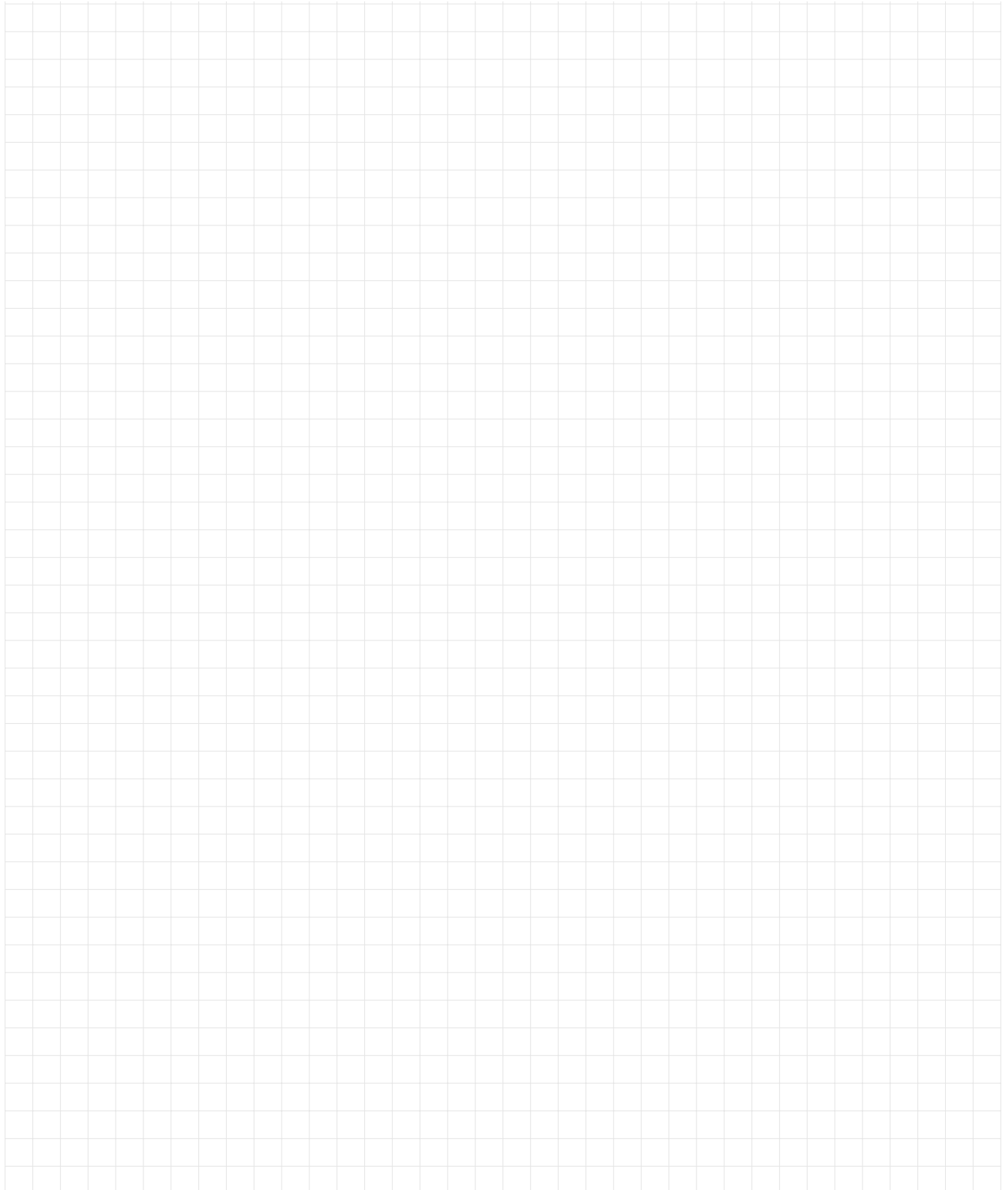
Fakultät

1. Schreiben Sie eine Python-Funktion zur Berechnung der Fakultät $n!$ einer positiven ganzen Zahl. Gehen Sie nach dem Prinzip der iterativen Programmierung vor.
2. Implementieren Sie Ihre Funktion rekursiv.
3. Schreiben Sie ein Programm, das solange eine ganze Zahl n einliest bis n positiv ist und dann den Wert der Fakultät $n!$ berechnet und ausgibt. Bei ungültiger Eingabe weisen Sie den Benutzer darauf hin, dass nur positive ganze Zahlen erlaubt sind. Fügen Sie die Option hinzu das Spiel bei Eingabe von 0 zu beenden.



SelectionSort (MaxSort)

1. Schreiben sie eine Python-Funktion `isSorted(inputList)`, die testet, ob eine gegebene Liste von Zahlen $[a_0, \dots, a_{n-1}]$ absteigend (also beginnend mit dem größten Wert) sortiert ist.
2. Schreibe Sie eine Python-Funktion `sort(inputList)`, die eine Liste mit Hilfe der Funktion `maximum(inputList)` sortiert. Wie oft müssen Sie die Liste durchlaufen?



Heron's Algorithmus/ Babylonisches Wurzelziehen

Berechnen Sie für eine beliebige nichtnegative Zahl $a \geq 0$ die Wurzel \sqrt{a} bis auf einen (relativen) Fehler von 10^{-10} nach der folgenden iterativen Methode:

$$x_{n+1} = \frac{1}{2} \left(\frac{x_n^2 + a}{x_n} \right).$$

1. Schreiben Sie dafür eine Python Funktion `heron(a, x0, tol)`.
2. Wählen Sie verschiedene Anfangswerte x_0 und beobachten Sie das Konvergenzverhalten, indem Sie zum Beispiel die Fehler $|x_n - \sqrt{a}|$ in einer Liste abspeichern.
3. Erweitern Sie die Funktionsschnittstelle um einen Parameter `maxiter` (`heron(a, x0, tol, maxiter)`). Brechen Sie (zusätzlich zum Fehlertoleranzkriterium) die Iteration ab, sobald die Anzahl der Schleifendurchläufe den Wert `maxiter` erreicht hat.

Hinweis: Benutzen Sie die built-in Funktion `abs()` und den Python-Wert `a ** 0.5` zur Fehlermessung.

