Thema 7: Programmieren von Funktionen

1 Einführung

In dieser Unterrichtseinheit lernst du ein weiteres Element von TigerJython kennen, nämlich **Funktionen**.

Zuerst wirst du erfahren, was Funktionen sind und wie man sie in TigerJython programmiert. Im weiteren Verlauf konzentrieren wir uns vor allem auf **mathematische Funktionen** und besonders auf das Bestimmen des **grössten gemeinsamen Teilers (ggT)** zweier Zahlen.

Neben kurzen **theoretischen Abschnitten** besteht die Lektion vor allem aus **vielen Übungen**, in denen du das neu gewonnene Wissen sofort anwenden kannst.

Was **brauchst du**, um diese Lektion zu bearbeiten?

- Wichtig ist dein **Vorwissen** aus den vorangegangenen Lektionen (Programmieren von einfachen **Algorithmen** mit TigerJython, Einsatz von **Schleifen**, Definition von **Unterprogrammen mit und ohne Parameter**, die Verwendung von **Variablen** und der Einsatz der **bedingten Anweisung** (**if… else… elif…**).
- Du benötigst einen **Computer**, auf dem die Programmierumgebung TigerJython zur Verfügung steht.
- Du brauchst auch einen **Bleistift u**nd evtl. einen **Radiergummi** ☺.

Ansonsten kannst du die ganze Lektion **selbstständig** und **in deinem Tempo** durcharbeiten. Gehe der Reihe nach vor – die Themen und Aufgaben bauen aufeinander auf. Frage bei deiner Lehrperson nach, wenn du nicht weiterkommst.

Im Anhang 1 werden dir ausserdem die **Lösungen zu sämtlichen Aufgaben** zur Verfügung gestellt (Kapitel 8, ab S. 16). Aber natürlich empfehlen wir dir, die Aufgaben **zuerst selbst zu lösen**. Du lernst viel mehr dabei – und ausserdem: **Du kannst das!!**

Und los geht's!

INHALT

2 Funktionen als eine spezielle Form von Unterprogrammen

Du hast in den bisherigen Lektionen schon zahlreiche Algorithmen in TigerJython programmiert. Dabei haben wir auch immer wieder mit dem Schlüsselwort **def** Unterprogramme verwendet, die dann im weiteren Verlauf deines Programms verwendet werden konnten.

Aufgabe 1

Nenne drei Vorteile, die das Programmieren mit Unterprogrammen mit sich bringt.

- ……

- ……

- ……

Nun lernst du eine spezielle Form von Unterprogrammen kennen, nämlich die **Funktionen**. Funktionen hast du wahrscheinlich schon **im Fach Mathematik** kennengelernt. Schauen wir uns als Beispiel für eine Funktion den folgenden Ausdruck an: **$f(x) = x^2$**.

- f ist der **Name** für die Funktion.
- Das x links und rechts vom Gleichheitszeichen ist eine **Variable**, die einen beliebigen Zahlenwert annehmen kann.
- Das Gleichheitszeichen sagt nun aus, dass zwischen den Variablenwerten links und rechts vom Gleichheitszeichen **eine statische Beziehung** besteht. Anders ausgedrückt: Wenn man der Variablen x links vom Gleichheitszeichen einen Wert zuweist (zum Beispiel x = 3), dann gibt es **auf der Seite rechts genau einen Wert**, der die Gleichung erfüllt (in diesem Fall $3^2 = 9$), also $f(3) = 3^2 = 9$.

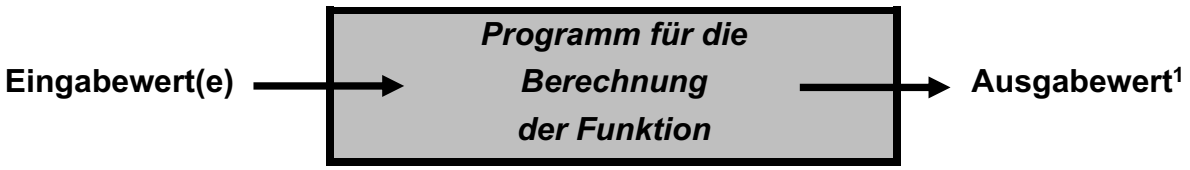
Funktionen in der Informatik sind jenen in der Mathematik sehr ähnlich. Auch sie können – wie alle anderen Unterprogramme - eine oder mehrere Anweisungen ausführen. Aber **am Ende der Ausführung** einer Funktion wird **immer nur ein einziger Wert zurückgegeben**. Mit diesem von der Funktion berechneten Wert führt dann das Programm, das die Funktion **aufgerufen** hat, weitere Anweisungen aus.

Das Spezielle an Funktionen in der Informatik ist also, dass wir uns auf **den Prozess der Berechnung des Funktionswertes konzentrieren** (während es in der Mathematik eher um die genaue Beschreibung der Funktion selbst geht).

Kurz gesagt: Einer Funktion als Programm werden…

- ein **Eingabewert** oder **mehrere Eingabewerte** übergeben,
- die durch die **Anweisungen innerhalb der Funktion**
- zu **einem Ausgabewert** verarbeitet werden.

Oder graphisch dargestellt:



[1] Hinweis für den weiteren Unterrichtsverlauf: Wir werden sehen, dass bei komplexen Datenstrukturen eine Funktion auch mehrere Ausgabewerte berechnen und zurückgeben kann.

3 Das Programmieren von einfachen Funktionen

Unten siehst du zwei Beispiele für Funktionen in TigerJython.

```
1  def f(x):
2      u = 3 * x
3      v = u + 5
4      w = 2 * v
5      return w
```

Beispiel 1

```
 9  def g(x):
10      if x < 5:
11          return 6 * x
12      else:
13          return 7
```

Beispiel 2

Beachte dazu die folgenden Bemerkungen:

- Eine Funktion wird – wie andere Unterprogramme - mit dem **Schlüsselwort def** deklariert.
- In Klammern hinter dem Funktionsnamen f wird - als **formaler Parameter** - die **Variable x** genannt. Wenn die Funktion aufgerufen wird, wird sie durch einen beliebigen Eingabewert – den sogenannten **aktualen Parameter** – ersetzt.
- Im eingerückten Teil können **Anweisungen und Befehle** ausgeführt werden, die für die Bestimmung des Ausgabewerts nötig sind.
- Am Ende wird mit dem Schlüsselwort **return** der **Ausgabewert zurückgegeben** und – dies ist ganz wichtig - die **Funktion wird gestoppt**.

```
16 #Funktion
17 def f(x):
18     u = 3 * x
19     v = u + 5
20     w = 2 * v
21     return w
22
23 #Aufruf der Funktion, Hauptprogramm
24 print(f(1))
```

Beispiel 3

Aufgabe 2

Nehmen wir an, das oben aufgeführte Beispiel 1 würde ergänzt durch den Befehl **print(f(1))**, wie das in Beispiel 3 der Fall ist. Beantworte die folgenden Fragen dazu in Stichworten (ohne den Code in TigerJython zu implementieren!).

- Wie heisst in Beispiel 3 der *formale Parameter*? .....

- Welchen Wert hat in Beispiel 3 der *aktuale Parameter*? .....

- Welchen Namen hat die *Variable*, deren Wert von der Funktion zurückgegeben wird? .....

- Welcher *Wert* wird von f(x) berechnet, wenn print(f(1)) aufgerufen wird? .....

Aufgabe 3

Aufgabe 3a

Was wird jeweils auf der Konsole ausgegeben, wenn die folgenden Befehle eingegeben werden?

(a) print(f(4)) …… (c) print(f(g(1))) ……
(b) print(g(4)) …… (d) print(g(f(1))) ……

Aufgabe 3b

Welche Werte liegen nach dem Ausführen der folgenden Ausdrücke in den Variablen w, x, y und z?

(a) w = f(3) …… (c) y = g(w) ……
(b) x = f(4) …… (d) z = g(f(3))……

Nun kennst du die grundlegenden Elemente, um in TigerJython Funktionen zu programmieren. Daher bist jetzt du dran!

Aufgabe 4

Aufgabe 4a

Schreibe eine Funktion betrag(x), welche den Ausgabewert x zurückgibt, wenn x positiv ist und andernfalls -x (z.B. -3). Überprüfe die Korrektheit deiner Funktion mit den Befehlen:

```
print(betrag(3))
print(betrag(-5))
print(betrag(0))
```

Aufgabe 4b

Achte dabei besonders auf die Ausgabe beim dritten Aufruf der Funktion. Kannst du diese Ausgabe erklären? Falls ja, notiere in Stichworten.

……..

Aufgabe 4c

Ergänze die Funktion so, dass die «seltsame» Ausgabe von 4b vermieden wird.

4 Die Modulo-Operation und Funktionen

Bereits in einer früheren Lektion haben wir die sogenannte Modulo-Operation kennen gelernt, die in der Informatik von fundamentaler Bedeutung ist.

Zur Erinnerung: Das Ergebnis einer Modulo-Operation liefert den **Rest einer ganzzahligen Division**. So ergibt beispielsweise 18 mod 4 das Resultat 2, weil 18 : 4 = 4 «Rest 2».

Wenn du die Modulo-Operation bereits sicher beherrschst, kannst du die folgende Aufgabe überspringen.

Aufgabe 5

Hier als kleine Übung ein paar einfache Modulo-Rechnungen.

Aufgabe 5a)

Schreibe das Ergebnis der folgenden Ausdrücke auf.

(a) 7 mod 3 …… (d) 8 mod 3 ……
(b) 24 mod 8 …… (e) 11 mod 3 ……
(c) 13 mod 7 …… (f) 14 mod 0 ……

Nun wenden wir die Modulo-Operation in einer Funktion von TigerJython an.

```
1 def kT(x):
2     i = 2
3     while i <= x:
4         if x % i == 0:
5             return i
6         else:
7             i = i+1
```

Beispiel 4

Aufgabe 6

Beantworte die folgenden Fragen, die sich auf das Beispiel 4 beziehen.

Aufgabe 6a

Erkläre in 2 bis 3 Sätzen, wie der Ausgabewert von kT(x) mit dem Eingabewert x zusammenhängt.

……

……

……

……

Aufgabe 6b

Welche Ausgabe liefert die Funktion für den Befehl print(kT(15))?

……

Aufgabe 6c

Wofür steht, wenn du die Lösung in der Aufgabe 6b betrachtest, das «kT» im Funktionsnamen?

……

5 Wissenswertes über Primzahlen… Dann wird richtig programmiert!

Aus dem Mathematik-Unterricht weisst du schon, dass eine **Primzahl** dadurch gekennzeichnet ist, dass sie **nur durch 1 und durch sich selbst ohne Rest teilbar ist**. Beispielsweise gibt es für die Zahl 13 keinen ganzzahligen Teiler, der grösser als 1 oder kleiner als 13 ist, um die Zahl 13 ohne Rest zu teilen.

Aufgabe 7

Was folgt aus dieser Definition einer Primzahl in Bezug auf die Modulo-Operation, wenn sie auf Primzahlen angewendet wird (siehe oben, Aufgabe 5)? Erkläre deine Antwort am Beispiel der Zahl 13.

……

……

Mit Hilfe der Modulo-Operation und deinem Wissen über Primzahlen kannst du nun bereits nicht-triviale Funktionen mit TigerJython schreiben.

Aufgabe 8

Aufgabe 8a

Schreibe eine Funktion istPrim(x), welche den Ausgabewert 1 liefert, falls x eine Primzahl ist, und andernfalls 0. (Hinweis: Schreibe zuerst die in Aufgabe 6 analysierte Funktion kT(x) ab und verwende sie in deinem Programm.) Teste dein Programm.

Aufgabe 8b

Vorbemerkung: Es gibt unendlich viele Primzahlen: 2 ist die erste Primzahl (die einzige, die gerade ist!), 3 ist die zweite, 5 die dritte usw.

Schreibe nun eine Funktion PZ(n), welche als Ausgabewert die n-te Primzahl in dieser unendlich langen Liste ausgibt. **Verwende dazu** die schon programmierten Funktionen kT(x) und istPrim(x) aus Aufgabe 8a!

Aufgabe 8c

Freiwillige Zusatzaufgabe: Gib die fünfte Primzahl aus (11), die elfte (31), die 100. (541) und die 5000. (70657). Was fällt dir auf, wenn du die 5000. Primzahl ausgibst? Kannst du die Beobachtung erklären? Notiere in Stichworten.

……..

6 Die Bestimmung des grössten gemeinsamen Teilers (ggT) zweier Zahlen

Nun kommen wir zum Kernstück dieser Lektion – es verbindet dein **Wissen über das Programmieren von Funktionen**, dein **Wissen über die Modulo-Operation** und **dein Wissen über Primzahlen**. Denn eine wichtige Problemstellung in der Zahlentheorie und auch ganz besonders in der Informatik (vor allem in der Kryptologie) ist die Bestimmung des **grössten gemeinsamen Teilers (ggT)** von zwei ganzen Zahlen (ohne Rest!).[2]

Die Zahl 4 ist beispielsweise der ggT der beiden Zahlen 8 und 12. Die Zahl 6 ist zwar grösser als 4 und teilt die Zahl 12 ebenfalls ohne Rest, aber die Zahl 8 lässt sich nicht ohne Rest durch 6 teilen – deshalb kann 6 nicht der ggT von 8 und 12 sein.

Schwieriger und damit auch interessanter wird das Finden des ggT für grössere und sehr grosse Zahlen. Was ist z. B. das Ergebnis von ggT(972, 333) oder gar von ggT(278567936478, 856390745278190)? Mehr dazu in den folgenden Abschnitten.

6.1 ggT zweier Zahlen: Variante 1 – Nice, aber naiv…

Einen naiven Ansatz für eine Funktion, welche den ggT von zwei Zahlen a und b bestimmt, siehst du in Beispiel 5:

```
1 def ggT(a,b):
2     i = 1
3     t = 1
4     while i <= b:
5         if (a % i == 0 and b % i == 0):
6             t = i
7         i = i + 1
8     return t
```

Beispiel 5

Aufgabe 9

Schreibe das Programm aus Beispiel 5 ab. Erkläre direkt im Programmcode für jede Programmzeile, **was genau** das Programm tut. **Wichtig:** Mit dem Zeichen # kannst du den eigentlichen Programmcode und den Kommentar voneinander trennen. Beispiel für die Zeile 1:

#a, b: zwei Zahlen, die der Funktion ggT übergeben werden
1: **def** ggT(a,b*):*

 # Hier positionierst du deinen Text als Kommentar in TigerJython.
2: i = 1

Speichere das Programm zusammen mit dem Kommentar, wenn du fertig bist.

[2] Der ggT spielt etwa beim Kürzen von Brüchen eine wichtige Rolle; vor allem aber ist die moderne Verschlüsselung im Internet ohne die Verfahren, den ggT zu bestimmen, undenkbar! Wir werden später im Informatik-Unterricht beim RSA-Verschlüsselungsverfahren darauf zurückkommen.

Warum haben wir diesen Ansatz «naiv» genannt? Hast du eine Vermutung dazu? Du kannst, falls du etwas ahnst, deine Überlegung gleich in der folgenden Aufgabe überprüfen – und deinen Computer garantiert in die Knie zwingen! Versprochen!

Als Erstes erhältst du den Programmcode, den du benötigst:

```
1  from time import clock
2
3  #Hier kommt dein Code rein
4
5  t1 = clock()
6  print(ggT(10**9, 10**9 + 4))
7  t2 = clock()
8
9  deltaT=t2-t1
10 print(deltaT)
```

Beispiel 6

Aufgabe 10

Aufgabe 10a

Öffne eine neue TigerJython-Datei und kopiere den Programm-Code aus Aufgabe 9 hinein. Anschliessend musst du den Code mit den Zeilen aus Beispiel 5 ergänzen. Achte darauf, dass der Befehl ***from*** *time* ***import*** *clock* in der allerersten Zeile des Programms steht.

Zuletzt ersetzt du die Parameter zahl1 und zahl2 der Reihe nach mit den untenstehenden Zahlen, lässt jeweils das Programm laufen und notierst am Ende, welchen Wert der Befehl print(deltaT) ausgegeben hat.

1. Versuch	zahl1 = 10**6[3]	zahl2 = 10**6 + 4	→ deltaT: ……
2. Versuch	zahl1 = 10**7	zahl2 = =10**7 + 4	→ deltaT: ……
3. Versuch	zahl1 = 10**8	zahl2 = 10**8 + 4	→ deltaT: ……
4. Versuch	zahl1 = 10**9	zahl2 = 10**9 + 4	→ deltaT: ……
5. Versuch	zahl1 = 10**12	zahl2 = 10**12 + 4	→ deltaT: ……

Aufgabe 10b

Wir haben von einem «naiven» Ansatz gesprochen. Was bedeutet in diesem Zusammenhang das Wort «naiv», wenn du das Ergebnis in Aufgabe 10a betrachtest?

……

[3] 10**6 bedeutet hier, dass 6 der Exponent von 10 ist. Dies ist die Darstellung in TigerJython für die Zahl 10^6. Das Entsprechende gilt natürlich für 10**8, 10**9 usw.

6.2 ggT: Variante 2 – Schon viel schneller!

So viel vorweg: Es gibt eine schnellere Variante zur Berechnung des ggT zweier Zahlen, eine sehr viel schnellere Variante. Zu verdanken haben wir sie griechischen Mathematikern, die ein aus China stammendes Verfahren optimiert haben. Die griechische Version wurde im 3. Jh. v. Chr. von Euklid aufgeschrieben, sie wird deshalb auch **Euklidischer Algorithmus** genannt. Um diesen zu verstehen, müssen wir allerdings etwas tiefer in die Zahlentheorie einsteigen. Die folgenden Abschnitte setzen voraus, dass du folgende Begriffe schon verstehst: a) **Teiler einer Zahl**, b) **Teilermenge einer Zahl**, c) **Teilermenge von zwei Zahlen** a und d) **grösster gemeinsamer Teiler zweier Zahlen**. Wenn nötig, findest du im **Anhang 2 (ab Seite 24)** Erklärungen dazu. Ansonsten kannst du sogleich mit der Lektüre fortfahren.

Der springende Punkt ist: Wenn zwei Zahlen a und b einen gemeinsamen Teiler t haben, dann teilt dieser Teiler t auch das Ergebnis der Subtraktion von a und b. Da der ggT in jedem Fall zwei Zahlen teilt, gilt auch für ihn, dass der ggT(a,b) = ggT(a-b, b). Machen wir uns diese zunächst seltsam klingende Aussage an einem Beispiel klar.

1. Seien a = 84 und b = 18. Wenn wir die Teilermengen von a und b aufstellen, dann sehen wir: Der grösste gemeinsame Teiler von a und b ist die Zahl 6. Das heisst, die Zahl 6 teilt sowohl die Zahl 84 und die Zahl 18, sie ist einer von allen Teilern beider Zahlen.
2. Wenn wir nun 84 – 18 rechnen, dann ist das Ergebnis 66. Und auch 66 muss wieder durch 6 teilbar sein (66 : 6 = 11), da wir mit der Zahl 18 in jedem Fall ein Vielfaches von 6 abziehen (3 * 6, sonst wäre 6 ja kein Teiler von 18).
3. Somit können wir von 66 wiederum 18 abziehen, erhalten 48 – und auch dieses Ergebnis ist durch 6 teilbar.
4. Wiederholen wir die Subtraktion ein drittes und viertes Mal, dann erhalten wir als Ergebnis die Zahl 12.
5. Nun stellen wir fest: 18 ist grösser als 12, aber die Zahl 6 muss immer noch ein gemeinsamer Teiler der beiden Zahlen sein, denn wir haben immer nur mit Vielfachen von 6 gerechnet.
6. Somit können wir die Subtraktion einfach weiterführen, indem wir die beiden Zahlen vertauschen! 18 wird das neue a, 12 das neue b; nun berechnen wir 18 – 12 = 6.
7. 12 ist wiederum grösser als 6, aber die Zahl 6 muss immer noch ein gemeinsamer Teiler von beiden Zahlen sein.
8. Wir machen dasselbe wie in Schritt 6: a = 12, b = 6. 12 – 6 = 6.
9. Nun wird a = 6 und b = 6; wenn wir die Rechnung ausführen, ist das Resultat 0. Die Zahl 6 aber war gerade der ggT(84,18), siehe oben Schritt 1 – oder anders gesagt: **Sobald die Subtraktion von a – b das Ergebnis 0 liefert, wissen wir, dass in der Variablen b aus dem Schritt 8 der ggT(a, b) gespeichert ist.**

Gilt dieses Verfahren tatsächlich für alle Zahlen oder haben wir mit unserem Beispiel einfach Glück gehabt? Den mathematischen Beweis dafür, dass dies immer stimmt, findest du ausführlich im **Anhang 2 ab der Seite 26**.

Aufgabe 11

Aufgabe 11a

Im Anhang 2 findest du die mathematischen Grundlagen zu dieser Lektion. In einer Aufgabe (siehe Aufgabe M5, Seite 28) sollst du mit dem gerade beschriebenen Verfahren den ggT(972, 333) berechnen. **Empfehlung**: Löse diese Aufgabe, wenn du mit der Teilaufgabe 11b) Schwierigkeiten hast.

Aufgabe 11b

Bei der Berechnung im obigen Beispiel mit den Zahlen a = 84 und B = 18 wurde sozusagen «zwischen den Zeilen» ein Algorithmus zur Bestimmung des ggT formuliert. Er enthält die Operationen:

- Subtraktion von zwei Zahlen
- Vergleich von zwei Zahlen
- Schleife (wiederholtes Abziehen einer Zahl von einer anderen)
- Austausch von Werten
- eine Abbruchbedingung

Schreibe eine Funktion ggT2(a,b), die den ggT der beiden Zahlen a und b aufgrund des oben in natürlicher Sprache verfassten «Algorithmus» zurückgibt.

6.3 ggT: Variante 3 – Einfach (und) genial!

Nun kommen wir zur Quintessenz dieser Lektion, die dein Wissen über Funktionen, die Modulo-Operation und den ggT zusammenbringt – wir implementieren den **Euklidischen Algorithmus** zur Berechnung der Funktion ggT(a,b). Eleganter geht's nicht! Überlege dir kurz: Was haben wir eigentlich gemacht, als wir im vorhergehenden Abschnitt (6.2) den ggT für die beiden Zahlen 84 und 18 berechnet haben?

Genau: Wir haben 18 so lange von 84 abgezogen, bis das Ergebnis kleiner war als die Zahl 18 selbst (nach dem 5. Schritt erhielten wir die Zahl 12). 12 ist aber gerade der **Rest** von 84 – 4 * 18. **Das aber ist nichts anderes als die Modulo-Operation, denn 84 = 4 * 18 + 12!** Und mit dem Wissen, dass die beiden Zahlen (12 und 18) immer noch denselben Teiler haben, haben wir ihre Reihenfolge umgedreht und wiederum die Modulo-Operation angewendet. Damit reduziert sich der Algorithmus zur Bestimmung des ggT(a, b) auf 3 Schritte!

1. Führe die Operation a mod b durch (falls a > b).
2. Weise a den Wert von b und b das Ergebnis von a mod b zu.
3. Wiederhole die Schritte 1. und 2. so lange, bis a mod b == 0, dann ist b = ggT(a, b).

Mit der Modulo-Operation kann man also auch bei grossen Zahlen den ggT mit ein paar wenigen Rechenschritten bestimmen – und das kannst du jetzt in den folgenden drei Aufgaben (12, 13, 14) selbst überprüfen!

Aufgabe 12

Berechne den ggT(a, b) mit Euklids Algorithmus. Sei a = 312 und b = 91. Als Werkzeug kannst du die nachfolgende Tabelle verwenden.

a	b	a mod b

Anmerkung: Die Implementierung des Euklidischen Algorithmus zur Bestimmung des ggT ist auf zwei Arten möglich, wähle eine davon aus (Variante A oder Variante B).

Aufgabe 13 (Variante A: Iterative Programmierung)

Implementiere eine Funktion zur Bestimmung des ggT von zwei Zahlen ggT(a, b) mittels der Modulo-Operation.

Verwende dabei unter anderem eine **while-Schleife**.

Aufgabe 14 (Variante B: Rekursive Programmierung)

Implementiere eine Funktion zur Bestimmung des ggT von zwei Zahlen ggT(a, b) mittels der Modulo-Operation.

Wende dabei die **Rekursion** an, die wir im Zusammenhang mit den Koch-Kurven kennen gelernt haben.

Aufgabe 15 (Testen des Euklidischen Algorithmus)

Aufgabe 15a

Wiederhole den Test aus Aufgabe 10 – doch dieses Mal verwendest du statt des «naiven» Algorithmus jenen, der auf Euklid bzw. der Modulo-Operation basiert.

1. Versuch zahl1 = 10**6[4] zahl2 = 10**6 + 4 → deltaT: ……
2. Versuch zahl1 = 10**7 zahl2 = =10**7 + 4 → deltaT: ……
3. Versuch zahl1 = 10**8 zahl2 = 10**8 + 4 → deltaT: ……
4. Versuch zahl1 = 10**9 zahl2 = 10**9 + 4 → deltaT: ……
5. Versuch zahl1 = 10**12 zahl2 = 10**12 + 4 → deltaT: ……

Aufgabe 15b

Dein Fazit?

……

……

……

[4] 10**6 bedeutet hier, dass 6 der Exponent von 10 ist. Dies ist die Darstellung in TigerJython für die Zahl 10^6.

7 Schlussbetrachtung

Wenn du alle Aufgaben bis zu diesem Punkt erfolgreich durchgearbeitet hast, dann hast du nun Kenntnis über vier grundlegende Aspekte der Informatik gewinnen können:

- Du weisst, was eine **Funktion** ist, und kannst dieses Wissen beim Programmieren in TigerJython anwenden.
- Aus mathematischer Sicht kennst du nun die genaue Bedeutung der **Modulo-Operation** und kannst sie zum Programmieren von wichtigen Algorithmen verwenden – zum Beispiel zur Bestimmung des ggT.
- Ausserdem hast du gesehen, dass es für viele algorithmische Probleme **verschiedene Lösungsmethoden** gibt, die alle zum richtigen Ergebnis führen. Man kann beispielsweise den Euklidischen Algorithmus zur Bestimmung des ggT sowohl **iterativ** wie auch **rekursiv** implementieren.
- Du hast aber auch gesehen, dass man nicht alle Lösungsmethoden in der Praxis tatsächlich verwenden kann – da sie zum Beispiel bei grossen Eingaben eine viel zu lange **Laufzeit** haben, um das richtige Ergebnis zu liefern. Eine **effiziente Lösungsmethode** konnten wir durch unser Wissen aus der Mathematik herleiten – so auch bei der Formulierung des Algorithmus in den Aufgaben 13 und 14, die auf zahlentheoretische Einsichten zurückgehen, die schon Euklid im 3. Jh. vor Christus bekannt waren.

Funktionen, die **Modulo-Operation**, die **iterative** beziehungsweise **rekursive Programmierung** und **Laufzeit** – auf alle diese Konzepte werden wir im weiteren Verlauf des Informatik-Unterrichtes immer wieder zurückkommen und sie vertiefen.

8 Anhang 1: Lösungen zu den Aufgaben 1-15

Aufgabe 1 (Vorteile des Programmierens mit Unterprogrammen)

- Man kann mit Unterprogrammen lange Programme in einzelne Programmteile zerlegen, sie also strukturieren, und so das gesamte Programm **übersichtlicher** gestalten.
- Man kann den Code von Unterprogrammen **beliebig oft verwenden**.
- Mit Hilfe von Unterprogrammen kann man Parametern Werte übergeben, um so das Programm **präzise zu steuern**.
- Es ist einfacher, die **Programme zu überprüfen (zu verifizieren)**, in ihnen Fehler zu suchen und die Fehler zu finden.

Aufgabe 2 (Variablen und Parameter)

- Formaler Parameter: x
- Aktualer Parameter: 1
- Variable: w
- Wert: **16**
  x (formaler Parameter)
  1 (aktualer Parameter)
  u = 3 * 1 = 3
  v = 3 + 5 = 8
  w = 2 * 8
  w wird als Ergebnis der Funktion an den Aufrufer print() zurückgegeben.

Aufgabe 3 (Ausgaben berechnen)

Lösung zu 3a

a) print(f(4)) Ausgabe auf der Konsole: 34
b) print(g(4)) Ausgabe auf der Konsole: 24
c) print(f(g(1))) Ausgabe auf der Konsole: 46
d) print(g(f(1))) Ausgabe auf der Konsole: 7

Begründung für c)

- Wie man an diesem Beispiel sieht, kann statt eines konkreten Parameters (hier anstelle einer Zahl) **auch eine andere Funktion an eine Funktion übergeben werden**. D.h., es wird **der Wert g(1)** der Funktion g auf dem Argument 1 an x (x = g(1) = ...) **als Argument der Funktion f** übergeben. Somit erhält man den Wert f(g(x)) = f(g(1)) = f (...) als Resultat. Es wird zuerst der Wert in der inneren Klammer, dann jener in der äusseren Klammer ausgewertet, ansonsten «wüsste» die Funktion f(x) gar nicht, aufgrund welchen Wertes sie die Berechnung durchführen muss.

Begründung für d)

- Der Aufruf von f(1) liefert, wie wir schon in Aufgabe 2 gesehen haben, den Wert 16. Wenn nun g(16) aufgerufen wird, dann ist die Bedingung in **if x < 5** nicht erfüllt (sie hat den Wert **false**). Somit wird der **else**-Zweig ausgeführt, der unabhängig von der Eingabe den Wert 7 zurückgibt (**return 7).**

Lösung zu 3b

a) w = 28 c) y = 7
b) x = 34 d) y = 7

Anmerkung zu c) und d)

- In c) wird der Wert der Variablen w verwendet, der zuvor, d.h. in a), dort abgespeichert wurde. In d) wird der Wert für f(3) direkt in g(f(3)) berechnet, das Ergebnis für f(3) ist natürlich dasselbe wie in a), daher sind auch die Ausgaben in c) und d) identisch.

Aufgabe 4 (Die Funktion betrag(x) implementieren)

Lösung zu 4a

```
1 def betrag(x):
2     if x > 0:
3         return x
4     else:       #Alternativ: elif x < 0
5         return -x
6
7 print(betrag(8))
```

Die Ausgabe auf der Konsole für print(betrag(8)) ist 8, ebenso die Ausgabe für print(betrag(-8)).

Lösung zu 4b

Auf der Konsole wird der Wert **None** ausgegeben. Der Grund ist, dass das Programm den Fall, dass x == 0 ist, nicht berücksichtigt. In Python wird in diesem Fall standardmässig der Wert **None** zurückgegeben – ein Objekt, das auf nichts zeigt. Was das genau bedeutet, werden wir in einem fortgeschrittenen Stadium des Informatik-Unterrichts anschauen.

Lösung zu 4c

Wir berücksichtigen explizit, dass im Aufruf auch der Wert «0» übergeben werden kann.

```
1  def betrag(x):
2
3      if x > 0:
4          return x
5      elif x < 0:
6          return -x
7      else:       #Alternative: elif x == 0:
8          return 0
9
10
11
12 print(betrag(-3)) #Ausgabe 3
13 print(betrag(4)) #Ausgabe 4
14 print(betrag(0)) #Ausgabe 0
```

Aufgabe 5 (Einfache Modulo-Berechnungen)

Lösung zu 5a

(a) 1 (d) 2
(b) 0 (e) 2
(c) 6 (f) Nicht erlaubt, da die Division durch die Zahl 0 auch hier nicht geht.

Aufgabe 6 (Die Funktion *kT(x)*)

Lösung zu 6a

- Zunächst wird eine (Zähler-)Variable i auf den Wert 2 initialisiert.
- Dann wird für die Zahl, die über den Parameter x an die Funktion übergeben wurde, die Modulo-Operation angewendet.
- Wenn die Modulo-Operation den Wert 0 liefert, dann wird der aktuelle Wert zurückgegeben (return i).
- Der letzte Schritt wird so lange wiederholt, bis die Moduloperation x % i den Wert 0 liefert – und das ist spätestens dann der Fall, wenn i den Wert von x erreicht hat.

Lösung zu 6b

- Sie liefert den **Wert 3**.

x = 15
i = 2
15 mod 2 = 1 (also $\neq$ 0)
i = i + 1 = 3
x mod 3 = 0
Die **while**-Schleife wird verlassen, der Wert 3 wird zurückgegeben.

Lösung zu 6c

- kT bedeutet **k**leinster **T**eiler; die Funktion **ermittelt den kleinsten Teiler von x**, der x ohne Rest teilt.

Aufgabe 7 (Modulo-Operation und Primzahlen)

- Wenn die Modulo-Operation auf eine Primzahl x angewendet wird, dann liefert die Operation nur den Wert 0, wenn x mod 1 oder x mod x gerechnet wird.
- Es gibt keine ganzzahligen Teiler von 13, die grösser als 1 und kleiner als 13 sind – sonst wäre 13 eben keine Primzahl.

Aufgabe 8 (Die Funktion *istPrim(x)*)

Lösung zu 8a

```
1  def kT(x): #Kleinster Teiler einer Zahl, siehe Aufgabe 6
2      i = 2
3      while i <= x:
4          if x % i == 0:
5              return i
6          else:
7              i = i+1
8
9  def istPrim(x):
10     if kT(x) == x:
11         return 1
12     else:
13         return 0
14
15 print(istPrim(15)) #Liefert auf der Konsole den Wert 0
16 print(istPrim(13)) #Liefert auf der Konsole den Wert 1
```

Lösung zu 8b

```
1  def kT(x): #Kleinster Teiler einer Zahl
2      i = 2
3      while i <= x:
4          if x % i == 0:
5              return i
6          else:
7              i = i+1
8
9  def istPrim(x):
10     if kT(x) == x:
11         return 1
12     else:
13         return 0
14
15 def PZ(n):
16     i = 0   # Zählervariable
17     z = 1   # Initialisierung von z,
18             # in welcher die jeweils aktuelle Primzahl gespeichert wird
19     while i < n:
20         z = z + 1
21         if istPrim(z) == 1:
22             i = i + 1
23     return z
24
25 print(PZ(12))
```

Lösung zu 8c

- Es dauert rund 15 Sekunden, also für einen so schnellen Computer – sehr lange, bis das Ergebnis ausgegeben wird.
- Da der Computer ja «nicht weiss», welche Zahl eine Primzahl ist, muss er dies **für jede einzelne Zahl** überprüfen. Dies geschieht in Programmzeile 21. Da aber an dieser Stelle indirekt auch die Funktion kT(x) aufgerufen wird und sich dort ebenfalls eine while-Schleife befindet, die viele Rechenschritte erfordert, dauert es lange (viel zu lange), bis das gewünschte Ergebnis ausgegeben wird. Es handelt sich also nicht um einen optimalen Algorithmus (mehr dazu später in dieser Lektion).

Aufgabe 9 (Naiver Algorithmus zum Bestimmen des ggT(a, b))

```python
#a, b: Zahlen, die der Funktion übergeben werden
def ggT(a,b):

#Deklaration und Initialisierung der Zählervariable i
    i = 1

#Deklaration und Initialisierung der Variable, in welcher der ggT gespeichert wird
    t = 1

#while-Schleife, die solange ausgeführt wird,
#bis die Zählervariable den Wert von b erreicht hat
    while i <= b:

#Anwendung der Modulo-Operation a mod i und b mod i
#Nur wenn BEIDE Operationen den Wert 0 liefern, wird der Wert in i der Variablen t zugewiesen
#Denn in diesem Fall teilt i sowohl a und b und kommt als ggT in Frage.
        if (a % i == 0 and b % i == 0):
            t = i

#Wenn die if-Anweisung nicht true liefert, dann wird die Variable i um 1 erhöht
#und die while-Schleife erneut ausgeführt.
        i = i + 1

#Wenn die Bedingung in der while-Schleife nicht mehr erfüllt ist,
#d.h., wenn i grösser als b wird, dann wird
# - die while-Schleife verlassen
# - der Wert in t wird als ggT mit der Anweisung return zurückgeben
# - und - ganz wichtig - das Unterprogramm wird beendet (terminiert).
    return t

#Programmtest: ggt(2368, 1280) ergibt 64
print(ggT(2368, 1280))
```

Aufgabe 10 (Testen des «naiven» Algorithmus die Bestimmung des ggT)

Lösung zu 10a

- 10**6/10**6 + 4 ca. 0,5 Sekunden
- 10**7/10**7 + 4 ca. 1,7 Sekunden
- 10**8/10**8 + 4 ca. 13 Sekunden
- Für den 4. Versuch (Grössenordnung 1 Milliarde, 10**9) dauert es (auf meinem Rechner) bereits rund 2 Minuten 17 Sekunden – also für einen modernen Computer sehr lange, bis das Ergebnis ausgegeben wird.
- Beim 5. Versuch (10**12) erscheint auch nach einer halben Stunde kein Ergebnis.

Lösung zu 10b

Zwar liefert der Algorithmus das richtige Ergebnis, er «funktioniert» also. Aber er wäre in der Praxis nicht zu gebrauchen, da er schon bei Zahlen von der Grössenordnung 10^9 viel zu lange benötigt, um seine Aufgabe zu erfüllen, er ist **zwar korrekt, aber nicht effizient**.

Aufgabe 11 (Euklidischer Algorithmus ohne Modulo)

Lösung zu 11a

Siehe dazu im Anhang 2 die Lösung zur Aufgabe M5, Seite 29

Lösung zu 11b

```python
def ggT2(a, b):


#Schleife, die überprüft, ob a-b 0 ergibt;
#solange das nicht gilt, machen wir weiter
    while a - b != 0:

#Schleife, die die Subtraktion a - b vornimmt
#Abbruchbedingung: a wird grösser als b
        while a > b:
            a = a - b

#Drei Anweisungen, um die Werte von a und b zu vertauschen,
#falls a > b == False
        temp = a
        a = b
        b = temp

#Sobald a - b == 0 true liefert (Abbruchbedingung),
#wird die äussere Schleife verlassen und in b ist der ggT gespeichert
    return b

print(ggT2(36488,984584)) #--> Ausgabe 8
```

Aufgabe 12 (Euklidischer Algorithmus mit Papier und Bleistift)

a	b	a mod b	
312	91	39	312 = 3 * 91 + 39 39 = 312 – 3 * 91
91	39	13	91 = 2 * 39 + 13 13 = 91 – 2 * 39
39	**13**	0	39 = 3 * 13 + 0 0 = 39 – 3 * 13

Aufgabe 13 (Euklidischer Algorithmus mit Modulo, iterativ)

```python
def ggT(a,b):

# Prüfen, ob a grösser als b ist,
#sonst werden die Werte getauscht
    if a < b:
        temp = a
        a = b
        b = temp

#while-Schleife zum Bestimmen des ggT;
#Abbruchbedingung ist a mod b == 0
    while (a % b > 0):
            r = a % b
            a = b
            b = r

#Wir wissen, dass im Falle von a mod b == 0
#der ggT(a,b) in b gespeichert ist
    return b

print(ggT(18, 84)) #--> Ergebnis ist 6
```

Aufgabe 14 (Euklidischer Algorithmus mit Modulo, rekursiv)

```python
def ggt_rek(a,b):

# Prüfen, ob a grösser als b ist,
#sonst werden die Werte getauscht
    if a < b:
        temp = a
        a = b
        b = temp

#Prüfen, ob die Abbruchbedingungen erfüllt ist
    if a % b == 0:
        return b

#Wenn die Abbruchbedingungen nicht erfüllt ist,
#wird die Funktion ggt_rek(a, b) rekursiv aufgerufen
    else:
        return ggt_rek(b, a % b)

print(ggt_rek(84,18)) #--> Ausgabe 6
```

Aufgabe 15 (Testen des Euklidischen Algorithmus für die Bestimmung des ggT)

Lösung zu 15a

	Euklid	**«Naiv» (siehe Aufgabe 10)**
• 10**6/10**6 + 4	0,003 Sekunden	ca. 0,5 Sekunden
• 10**7/10**7 + 4	0,00024 Sekunden	ca. 1,7 Sekunden
• 10**8/10**8 + 4	0,00023 Sekunden	ca. 13 Sekunden
• 10**8/10**8 + 4	0,00023 Sekunden	ca. 13 Sekunden
• 10**9/10**9 + 4	0,0029 Sekunden	mehr als 2 Minuten
• 10**12/10**12+4	0,00027 Sekunden	??
• 10**250/10**250+4	0,00033	????????????????

Lösung zu 15b

Der Euklidische Algorithmus liefert auf sehr elegante Weise ebenso das **richtige Ergebnis** wie der naive Algorithmus, er ist also ebenfalls **korrekt**!

Der Euklidische Algorithmus liefert im Unterschied zu dem naiven Algorithmus die korrekte Lösung jedoch sehr schnell auch für grosse und sehr grosse Zahlen, mit denen ja in der Informatik tatsächlich häufig gearbeitet wird.Er ist also **korrekt** und **effizient**.

9 Anhang 2:
Mathematische Grundlagen des Euklidischen Algorithmus

In diesem Anhang können Interessierte die mathematischen Grundlagen des Euklidischen Algorithmus zur Bestimmung des ggT von zwei Zahlen im Detail nachvollziehen. Da das Erarbeiten dieses Stoffes die Lektion zu Funktionen in TigerJython zeitlich sprengen würde, wird er erst jetzt in diesem Anhang 2 präsentiert. Die Lösungen zu den Übungen finden sich am Ende von Anhang 2.

Frage 1: Was ist ein **Teiler einer Zahl**?

- Ein **Teiler t** einer Zahl a ist eine Zahl, die a **ohne Rest** teilt. Das heisst: a ist ein Teiler von b, wenn b = k * a für ein positives, ganzes k.
- Wenn wir also beispielsweise die Zahl a = 18 haben, dann teilt der Teiler t = 3 a ohne Rest. Das Ergebnis ist folglich wieder eine ganze Zahl, in dem Beispiel die Zahl 6, denn die Umkehrung ist 18 = 3 * 6.
- Die **mathematische Schreibweise** ist b | a. Lies: b teilt a (ohne Rest). Beispiel: b = 3 ist also ein Teiler von a = 18, es gilt b | a.

Übung M1

a) Sei a = 12 und b = 8. Gilt die Aussage b | a? ……

b) Ist die Zahl t = 4 ein Teiler der Zahl a = 64? ……

c) Wie viele Teiler hat die Zahl 12? ……

Frage 2: Was ist die **Teilermenge** einer Zahl?

- Unter der Teilermenge einer Zahl a versteht man **alle Teiler einer Zahl**, also alle Zahlen, die a ohne Rest teilen.
- Nehmen wir wiederum die Zahl a = 18 als Beispiel. Alle Teiler dieser Zahl sind 1, 2, 3, 6, 9 und 18.
- Die **Mengenschreibweise** sieht wie folgt aus: T(18) = {1, 2, 3, 6, 9, 18}

Übung M2

a) Welches ist die Teilermenge von a = 21? ……

b) Wie viele Teiler hat die Zahl 17? ……

c) Gib eine Zahl x an, die die Teilermenge T(x) = {1, 2, 4, 5, 9} hat. ……

Frage 3: Was ist **die Teilermenge von zwei Zahlen**?

- Nehmen wir nun die zwei Zahlen a = 84 und b = 18.
- Die Teilermenge von 84 ist T(84) = {1, 2, 3, 4, 6, 8, 12, 28, 42, 84}; die Teilermenge von b = 18 ist, wie schon dargelegt, T(18) = {1, 2, 3, 6, 9, 18}.
- Die **Teilermenge von beiden Zahlen** sind jene Zahlen, die **sowohl a wie auch b teilen**, also in unserem Fall die Zahlen 1, 2, 4 und 6.
- Die Teilermenge von zwei Zahlen ist somit die **Schnittmenge der beiden Teilermengen**.
- Wir schreiben formal für T(84) und T (12): T(84) $\cap$ (T(12) = {1, 2, 4, 6}.

Übung M3

a) Gib die Teilermenge von a = 24 und b = 10 an. ……

b) Gib zwei unterschiedliche Zahlen an, die die gemeinsame Teilermenge {1, 2, 4, 8} haben. ……

a) Gib die Teilermenge von a = 19 und b = 23 an. ……

Frage 5: Was ist **der grösste gemeinsame Teiler (ggT) von zwei Zahlen**?

- Unter **dem grössten gemeinsamen Teiler zweier Zahlen** a und b versteht man die grösste Zahl, die beide Zahlen ohne Rest teilt, also die **grösste Zahl der Schnittmenge beider Teilermengen**.
- Formal ausgedrückt ergibt sich die Schnittmenge wie folgt:
  - Teilermenge T =T(a) $\cap$ (T(b)
  - g $\in$ T und für jedes Element t aus T gilt t <= g
- Im Falle von a = 84 und b= 18 wäre das die Zahl 6, wie man leicht aus dem letzten Punkt von «Frage 3» sehen kann.

Übung M4

Gib den ggT(a, b) von a = 48 und b = 66 an, indem du…

- die Teilermengen beider Zahlen formal aufschreibst,
- dann die Schnittmenge der Teilermengen formal aufschreibst und
- aus der Schnittmenge den ggT(a, b herausliest.

- ……

- …… → ggT: ……

Fünf Sätze zum ggT(a, b)

(1) Was ist der ggT von ggT(a, a)? a
(2) Was ist der ggT von ggT(a, 1)? 1
(3) Was ist der ggT von ggT(a, 0)? a
(4) Gilt ggT(a, b) = ggT(b, a)? Ja. Die gemeinsame Teilermenge ist dieselbe.
(5) ggT(a, b) = ggT(a – b, b)? ??

Frage 6: Gilt auch **der Satz Teiler(a, b) = Teiler (a – b, b)?**

- Während oben die Sätze 1 – 4 einfach nachzuvollziehen sind, gilt dies für Satz 5 nicht. Um den Satz zu beweisen, schauen wir uns zuerst den Satz **Teiler(a, b) = Teiler (a – b, b)** an.
- Anders gesagt: Wenn die Zahl a und die Zahl b einen gemeinsamen Teiler haben, dann folgt daraus, dass auch die Differenz der beiden Zahlen durch diesen Teiler teilbar sein muss. Warum ist das so?
    - Schritt 1: Jede Zahl a lässt sich mit der Zahl b darstellen als a = q * b + r mit 0 <= r < b. (Beweis: https://youtu.be/rRvsTyfN74c)
    - Beispiel zu Schritt 1: Wenn a = 84 und b = 18, dann ist 84 = 4 * 18 + 12. In diesem Fall wird also der Rest r = 12.
    - Wenn nun sowohl a wie auch b einen gemeinsamen Teiler haben – im Fall von a = 84 und b = 18 zum Beispiel den Teiler 6 –, **dann muss auch der Rest (also a mod b) durch diesen Teiler teilbar sein**.
    - Begründung: Wenn von einem Vielfachen von 6 (a = 14 * 6 = 84) ein anderes Vielfaches von 6 (b = 3 * 6 = 18) abgezogen wird, dann muss auch der Rest dieser Subtraktion ein Vielfaches von 6 (84 – 18 = 66 = 6 * 11) sein. Dies kann man auch **formal herleiten**:
        i. a = q * b + r | gilt, siehe oben, Schritt 1; umformen
        ii. r = a – q * b | durch Umformen von i
        iii. a = x * t | wobei t ein Teiler von a ist
        iv. b = y * t | wobei t ein Teiler von b ist
        v. r = x * t – q * y * t |durch Einsetzen von iii. und iv. in ii.
        vi. r = t(x – q * y) |durch Ausklammern von t
        vii. → Da das Ergebnis des Ausdrucks x – q * y immer **eine ganze Zahl ist**, muss auch die Division r / t eine ganze Zahl sein, also **muss auch r durch t teilbar sein** – falls t | a und t | b.
    - Überprüfen wir das an einem **Beispiel**: a = 55 und b = 25 mit t = 5.
        i. 55 = 2 * 25 + 5
        ii. 5 = 55 – 2 * 25
        iii. a = 11 * 5
        iv. b = 5 * 5
        v. 5 = 11 * 5 – 2 * 5 * 5
        vi. 5 = 5 * (11 – 2 * 5) = 5 * (1)
        vii. 5 / 5 = 1 = 11 – 2 * 5 = 1

Frage 7: Der entscheidende Satz zum ggT(a, b): *Wenn wir den ggT von ggT(a, b) kennen, gilt dann auch ggT(a, b) = ggT(a - b, b) für alle a, b mit a > b?*

Die Antwort auf diese Frage lautet «ja» - und das ist ganz wichtig, denn diese Einsicht eröffnet die Möglichkeit, den ggT(a, b) sehr schnell zu berechnen. Wie aber kann man den Satz *ggT(a, b) = ggT(a - b, b)* zeigen?

- Wir legen zunächst fest:
  - t = ggT(a, b) und
  - t' = ggT(a – b, b)
  - Und wir sollen zeigen, dass t = t'.
- Somit ist **t der grösste Teiler** in der Teilermenge von a und b.
  - Wenn t = ggT(a, b) gilt, dann folgt daraus, dass t | a und t | b.
  - Wenn t | a und t | b, dann muss t auch a – b teilen (dies haben wir oben im Abschnitt zu **Frage 6** gezeigt), denn der ggT ist ja auch einfach einer der Teiler.
  - t muss somit auch in der Teilermenge von a – b liegen, also t | a – b.
- Weiter sei **t' der grösste Teiler** in der Teilermenge von ggT(a – b, b).
  - Wenn t' = ggT(a - b, b) gilt, dann folgt daraus, dass t' | a - b und t' | b.
  - Wenn aber gilt, dass t' | a - b und t' | b, dann muss auch t' | a - b + b gelten (dies kann wie oben in **Frage 6** auch für die Addition gezeigt werden), und somit gilt t' | a.
  - t' muss also auch in der Teilermenge von a und b liegen, also gilt t' | a und t' | b.
- Aus diesen Überlegungen folgt:
  - t liegt, wie gezeigt, auch in der Teilermenge von a – b und b, also in derselben Teilermenge wie t' selbst.
  - Da nun t' per definitionem der **grösste** gemeinsame Teiler von a – b, b ist, muss gelten: **t <= t'**.
  - Umgekehrt liegt t' auch in der Teilermenge a und b, also in derselben Teilermenge wie t selbst.
  - Da in dieser Teilermenge per definitionem t der **grösste** gemeinsame Teiler ist, muss auch hier gelten: **t >= t'**.
  - Da t und t' **jeweils Teiler in denselben Teilermengen sind**, können wir die beiden Ungleichungen aneinanderfügen, **es ergibt sich der Ausdruck t <= t' <= t.**
  - Daraus lässt sich ablesen, **dass t = t' sein muss und somit erhalten wir durch Einsetzen ggT(a, b) = ggT(a – b, b).** □

Fazit: Was bringt uns das alles für die Bestimmung von ggT(a, b)?

- Wir wissen jetzt, dass, egal welches der ggT von ggT(a, b) ist, es derselbe Teiler ist wie von ggT(a-b, b)!
- Verfahren:
  1. Damit können wir nun b von a so lange abziehen, bis a kleiner als b wird.
  2. Doch dann gilt wieder dasselbe: Der ggT(a, b) ist auch der ggT von b und dem Rest der wiederholten Subtraktion von a – b.
  3. Das heisst, wir nehmen b als das neue a und den Rest als neues b – und machen weiter bei Schritt 2.
  4. Das machen wir so lange, bis das Ergebnis von a – b den Rest 0 ergibt – dann steht in der Variablen b der ggT von ggT(a, b).

Übung M5

Wende dieses Verfahren an auf den Ausdruck ggT(972, 333).

Lösungen zu den Übungen M1 – M5

Lösung zu M1

a) Nein. 8 teilt 12 nicht ohne Rest (12 = 1 * 8 + 4), daher ist b | a **falsch**.
b) Ja, denn 64 = 16 * 4 + 0, also gilt 4 | 64.
c) Die Zahl 12 hat 6 Teiler, nämlich 1, 2, 3, 4, 6 und 12.

Lösung zu M2

a) T(21) = {1, 3, 7, 21}
b) 2 Teiler, nämlich 1 und 17. Die Zahl 1 teilt jede Zahl. Und jede Zahl hat sich selbst als Teiler, das Ergebnis ist dann immer 1. Zahlen, die nur die Zahl 1 und sich selbst als Teiler haben, sind **Primzahlen**.
c) 360 oder 180. (Die einfachste Variante, um dies zu berechnen, ist: x = 1 * 2 * 4 * 5 * 9)

Lösung zu M3

a) T(24) = {**1**, **2**, 3, 4, 6, 8, 12, 24}; T(10) = {**1**, **2**, 5, 10}; T(24, 10) = {1, 2}
b) 32, 64
c) T(19, 23) = {1}. Es sind zwei Primzahlen, die nur 1 als gemeinsame Teilermenge haben können.

Lösung zu M4

a) T(48) = {1, 2, 3, 4, 6, 8, 12, 16, 24, 48}; T(66) = {1, 2, 3, 6, 11, 22, 33, 66}
b) T(48, 66) = {1, 2, 3, 6}
c) ggT(48, 66) = 6

Lösung zu M5

- Seien a = 972 und b = 333.
- ggT(972, 333) = |Schritt 1
- ggT(639, 333) = |Schritt 1
- ggT((306, 333) |Schritte 2 und 3
- ggT(333, 306) = |Schritt 1
- ggT(27, 306) = |Schritte 2 und 3
- ggT(306, 27) = |Schritt 1, mehrmals wiederholen
- …
- ggT(9, 27) = |Schritte 2 und 3
- ggT(27, 9) = |Schritt 1
- ggT(18, 9) = |Schritt 1
- ggT(9, 9) |Schritt 1 → Schritt i4 → ggT(a, b) = 9.

Dieses Verfahren lässt sich noch wesentlich abkürzen, wenn man statt des wiederholten (und mühsamen) Subtrahierens (Schritt 1) die **Modulo-Operation** verwendet – denn nichts anderes macht man in den Schritten 2 und 3 des obigen Verfahrens.