Web Engineering Praktikum

Vorübung: Ruby

- Ruby als Rails-Grundlage
  - In diesem Kurs benötigen wir keine Feinheiten der Sprache, elementare Kenntnisse der prozeduralen und OO-Programmierung sowie einiger Standard-Klassen werden genügen.

- Wichtige Punkte, bitte bei der Erarbeitung betonen
  - Kontroll-Strukturen: if-elsif-else, case-when
  - Schleifen: while, until; Umgang mit **each** u.a. Iteratoren und damit Blöcken
  - Umgang mit Standard-Klassen und ihren Methoden: Array, Hash, String
  - OO-Programmierung: Klassen anlegen, Vererbung

Vorübung: Ruby

- Ruby-Tutorial
  - Ausgangspunkt für die Beschäftigung mit Ruby ist die Homepage dieser Sprache: **https://www.ruby-lang.org**
  - Im rechten Seitenkasten finden Sie Rubrik „Der Einstieg ist einfach“
  - Aufgabe 0.1: Nachvollziehen der Schritte aus „Ruby in 20 Minuten“
  - Aufgabe 0.2: Lesen von „Ruby für Umsteiger“ + „Von Java zu Ruby“ bzw. “Von Python zu Ruby”

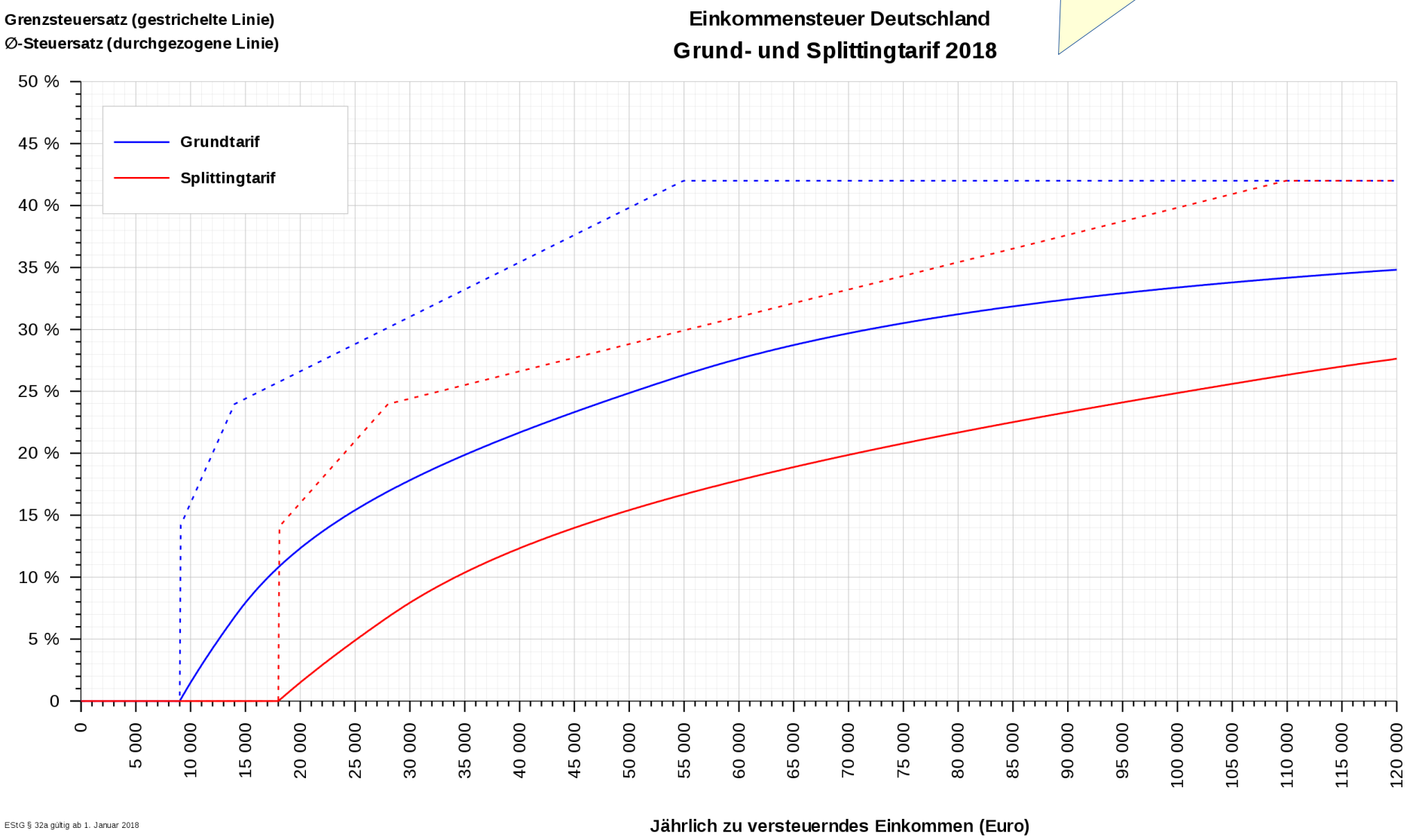
Aufgabe 1

- Ziel
  - **Prozedurales, zahlenorientiertes Arbeiten mit Ruby und der Kommandozeile lernen**
  - Anwendungsfall: Ermittlung von Einkommensteuer-Werten
    - Dabei lernen Sie auch gleich noch etwas über unser Steuer-System…

- Material, Hinweise:
  - Angaben aus der Kommandozeile erhalten Sie im Programm über die Array-artige Konstante **`ARGV`**
  - Berechnungsgrundlagen zum Einkommensteuertarif finden Sie z.B. hier: **`https://de.wikipedia.org/wiki/Einkommensteuer_(Deutschland)`**
  - Formal erhalten Sie die Einkommensteuer-Funktion durch Integration des Grenzsteuersatzes. In der Praxis läuft das auf Addition von ein paar Trapez- und Rechteckflächen hinaus.

Aufgabe 1

Veraltet – implementieren Sie die 5 Zonen gemäß Tarif 2020



Aufgabe

- Implementieren Sie folgende Methoden/Funktionen:

```
# Berechnet den Grenzsteuersatz aus dem gegebenen
# zu versteuernden Einkommen (zvE).
# Beispiele:
#   grenzsteuersatz(9409.0)         → 0.14    # gerundet
#   grenzsteuersatz(10000.0)        → 0.1515  # gerundet
def grenzsteuersatz(zvE)            → aFloat
  # Ihre Implementierung hier!
end


# Berechnet die zu zahlende Einkommensteuer aus dem
# gegebenen zu versteuernden Einkommen (zvE)
# Beispiele:
#   ek_steuer(9409.0)               → 0.0
#   ek_steuer(10000.0)              → 86.29 # gerundet
def ek_steuer(zvE)                  → aFloat
  # Ihre Implementierung hier!
end
# BEACHTEN: Das zvE wird zur Berechnung abgerundet!
# Beispiel: zvE = 25340,95          → 25340,00 zu verwenden
```

Aufgabe

Testen Sie nun diese Funktionen

- Vorgabe: Ergänzen Sie Code, der ein, zwei oder drei Zahlen von der Kommandozeile entgegen nimmt und deutet als
  - zvE bzw.
  - zvE1, zvE2 bzw.
  - zvE1, zvE2, zvE_increment

Tipp:
Das erste Argument hinter dem Programmnamen erhalten Sie mit **ARGV[0]**, das n-te mit ARGV[n-1] (immer als String, Umwandlung ggf. mit **to_i** bzw. **to_f**)

```
$ ruby 00-eksteuer.rb 10000
```

→ Ausgabe von EK-Steuer und Grenzsteuersatz

```
$ ruby 00-eksteuer.rb 10000 20000
```

→ Ausgabe von zvE, EK-Steuer und Grenzsteuersatz in Tabellenform, hier: zwei Zeilen

```
$ ruby 00-eksteuer.rb 10000 30000 2000
```

→ Ausgabe von zvE, EK-Steuer und Grenzsteuersatz in Tabellenform, hier: 11 Zeilen

Aufgabe

Abschlussteil

- Ausgaben für Fahrtkosten, Fachbücher und Rechner-Hardware zählen steuerlich zu den „Werbungskosten“. Sie senken das zu versteuernde Einkommen (zvE)

- Szenario: Jemand mindert sein zvE mittels Werbungskosten um 2000 €

- Berechnen Sie, wie viel Einkommensteuer er spart bei ursprünglich
  (a) **5000**, (b) **10000**, (c) **20000**, bzw. (d) **50000** € zvE,
  also die Differenz der EK-Steuer bei 20000 bzw. 18000 € zvE im Fall (c) usw.

- **Vorgabe**:
  - Ihr Programm soll diese Angaben in einer **each**-Schleife ausgeben, die eine Liste (ein Array) der o.g. vier zvE-Werte iteriert.

Aufgabe

- ***Unit Tests***

  - Das Testen einzelner Funktionen oder auch Methoden einer Klasse bezeichnet man als *unit testing*

  - Alle marktüblichen Programmiersprachen bieten Hilfsmittel an, um *unit tests* zu systematisieren und zu automatisieren. Das ist sehr wichtig, um auch größere Projekte sicher und ökonomisch vertretbar zu beherrschen – auch bei späteren Änderungen im Code.
    - Niemand hat Zeit oder Lust, vorhandenen Code wieder und wieder mit allen nur denkbaren Test-Situationen zu konfrontieren
    - Test-Bibliotheken ermöglichen den Aufbau von *test suites*. Das sind Sammlungen von erwarteten Ergebnissen für alle möglichen Code-Situationen, zusammen mit dem zur Berechnung und Vergleich notwendigen Code und einer Reporting-Funktion als Zusammenfassung.

  - Test-Werkzeuge in Ruby (später in Rails integriert):
    - test/unit oder minitest/

Aufgabe

- *Unit Tests*: Ein Beispiel

  - Wir erwarten, dass unsere Funktion **`grenzsteuersatz()`** den Wert 0.1400… bei der Eingaben 9001.0 liefert und darunter 0.0

  - Hier der Code dazu:

```
require "./00-eksteuer"
require "test/unit"

class MyFirstTestsForWebEng < Test::Unit::TestCase
  def test_grenzsteuersatz
    assert_in_delta 0.14, grenzsteuersatz(9001.0), 0.0001
    assert_equal 0.0, grenzsteuersatz(9000.99)
  end
end
```

Implementierungsdatei einbinden

Unit Test-Bibliothek anfordern

Erwartung

Tatsächlich berechnet

Aufgabe

- ***OO-Programmierung***

  - Implementieren Sie die Klasse **`Einkommen`**
    - Exemplare dieser Klasse sollen folgende Methoden unterstützen:

      `ek_steuer()`

      und

      `grenzsteuersatz()`

  - Schreiben Sie ferner *Setter-Methoden* zur Berücksichtigung steuerlicher Abzüge und zur Anwendung des Splitting-Verfahrens. Code-Bespiel:

```
e = Einkommen.new(38490.60)
e.abzüge = 2340.50
e.splitting = false
puts e.ek_steuer  # 7339.20 …
e.splitting = true
puts e.ek_steuer  # 3925,71 …
```

Organisatorisches

Organisatorisches

- Die Aufgabe dient Ihnen als Ziel während Ihrer Einarbeitung in Ruby
- Sie wird nicht bepunktet, aber es wird eine kurze Abnahme geben
  - Sie sollten also nach einer Woche (mehr oder weniger) lauffähigen Code vorzeigen können
  - Haupt-Zweck: Beschäftigen Sie sich mit den Dokumentationen von Ruby, eignen Sie sich die Sprache selbstständig soweit an, dass Sie damit die Aufgabe bearbeiten können

Quellen

Ruby-Dokumentation

- Ruby-Homepage: https://www.ruby-lang.org/
  - Darin erarbeiten: „Ruby in 20 Minuten“ und „Ruby für Umsteiger“

- Dokumentation der Ruby-Klassen:
  - http://ruby-doc.org (insb. „Core 2.x.y“, je nach der von Ihnen installierten Version)