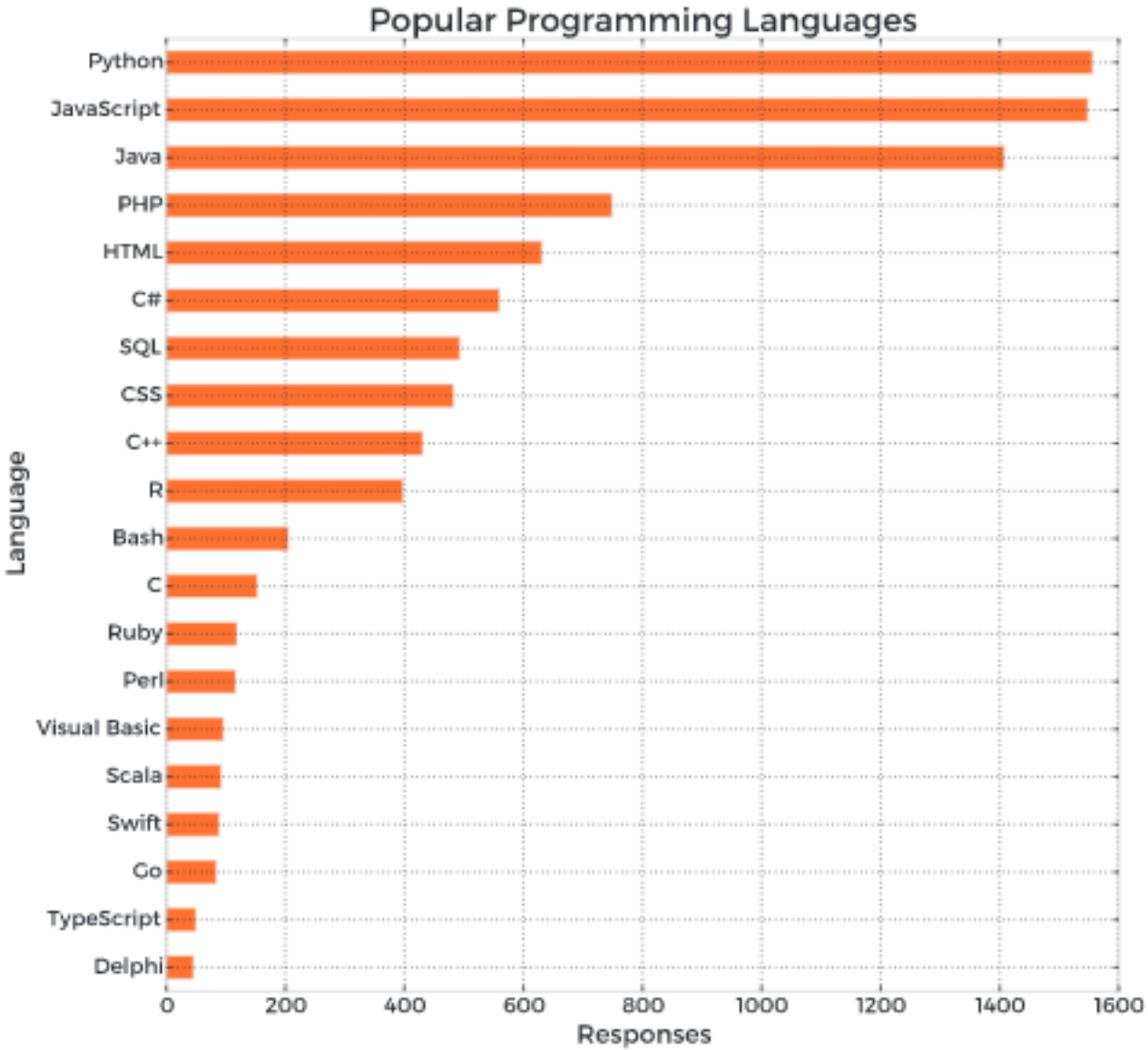
1. KI Übung: Python Basics

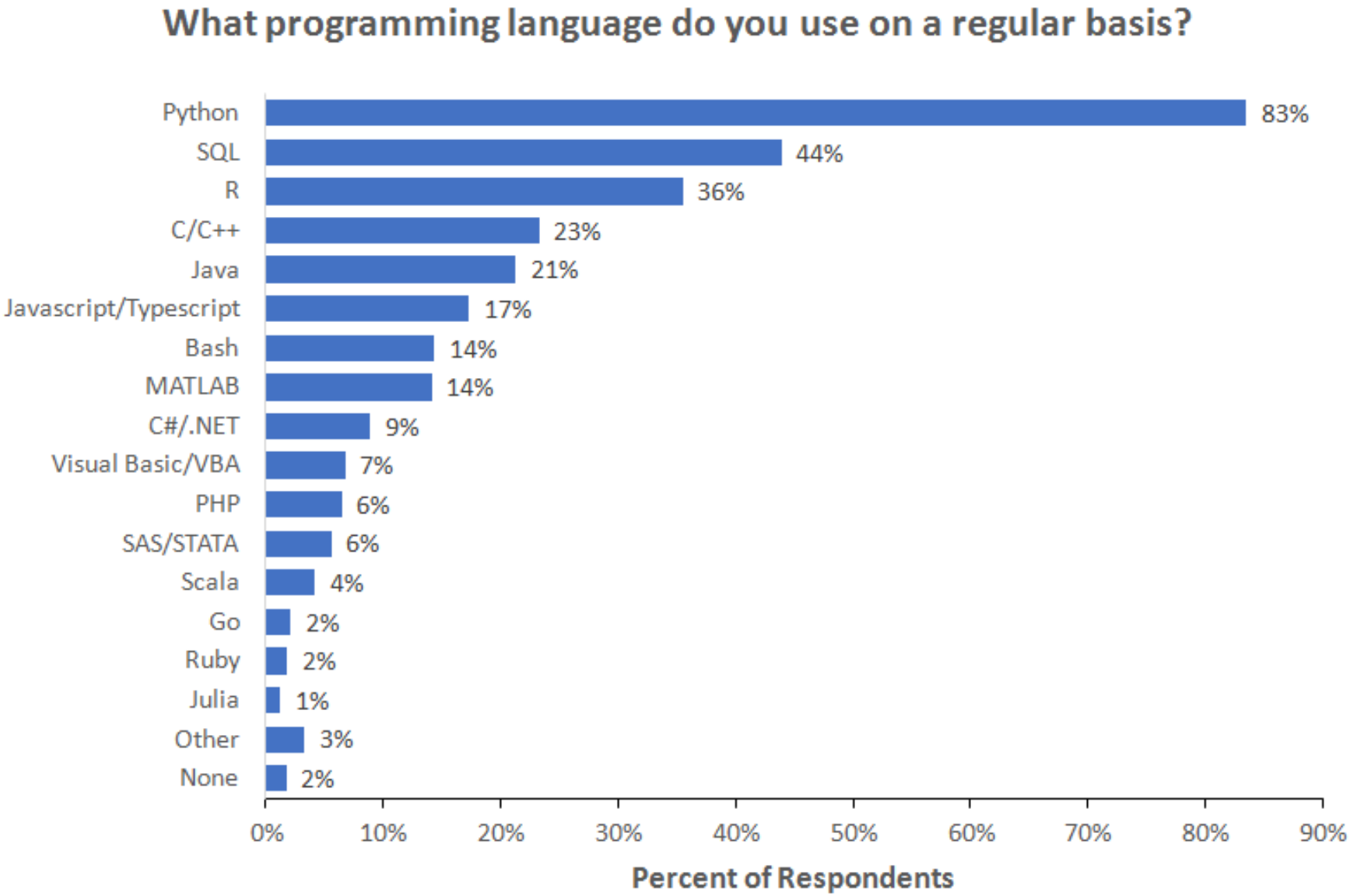
Matthias Tschöpe, Kunal Oberoi

11. November 2019

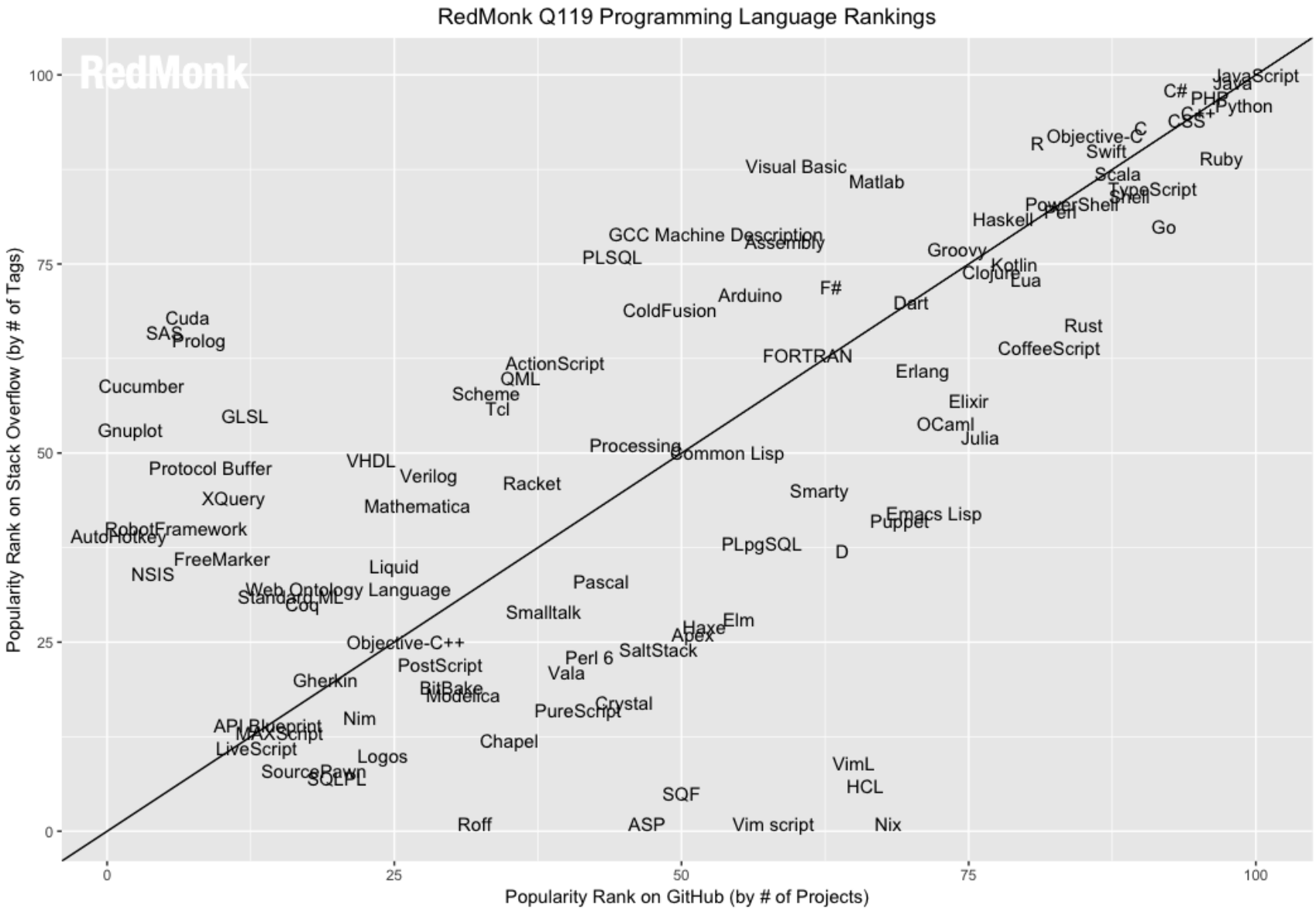
Warum Python? (i)



Warum Python? (ii)



Warum Python? (iii)



Warum Python? (iv)

Vorteile von Python?

- leicht zu lernen (vieles ist selbsterklärend)
- einfach zu lesen (fast Pseudocode)
- vieles kann sehr kompakt implementiert werden (z.B. WebCam Zugriff in einer Zeile)
- plattformunabhängig (Linux, Windows, Mac)
- Objekt orientierte Programmierung (OOP) möglich
- es gibt viele Packages die das Programmieren erleichtern (auch für KI)

Nachteile von Python?

- "langsam" (interpretierte Sprache)
- fast unmöglich große 3D Games zu implementieren
- nicht gut für Multi-Threading

Welche Packages für welche Aufgaben?

- Numpy (Numeric Python) $\rightsquigarrow$ ist das Standardpackage
- Matplotlib $\rightsquigarrow$ zum Plotten von Grafiken (2D und 3D)
- OpenCV, PIL $\rightsquigarrow$ sind zwei Packages zur Bildverarbeitung
- scikit-learn $\rightsquigarrow$ sinnvoll für Data Analysis
- Tensorflow, Pytorch, Keras $\rightsquigarrow$ Deep Learning libraries (GPU-Beschleunigung mit Nvidia-GPU's möglich)
- os $\rightsquigarrow$ z.B. Ordner anlegen, aktuellen Pfad zurückgeben lassen etc.

Python Versionen und Installation von Packages?

Python 2.x oder Python 3.x?

Da Python 2.x nur noch bis 2020 unterstützt wird, verwenden wir stets Python 3.x.

Installation von Python inkl. pip?

Homepage: `https://www.python.org/downloads/`

Mögliche Fehler bei der Installation

1. Pfadvariable nicht gesetzt.
2. Für Windows User: Manche Packages lassen sich nur mithilfe inoffizieller Binär-Daten installieren. Siehe hierzu: `https://www.lfd.uci.edu/~gohlke/pythonlibs/`

Virtual Environments

Wir erzeugen eine neue Virtuelle Umgebung mit dem Namen **ki_env** (im aktuellen Ordner), mit:

```
python -m venv ki_env
```

Bevor man mit der Virtuelle Umgebung arbeiten kann, muss sie aktiviert werden. Dies machen wir unter Windows mit dem Befehl:

```
ki_env\Scripts\activate.bat
```

Linux und Mac User aktivieren die Virtuelle Umgebung mit:

```
source ki_env/bin/activate
```

Packages installieren

Wie installieren wir Packages?

Falls nur eine Python Version vorhanden ist, können wir z.B. Numpy wie folgt installieren:

```
pip install numpy
```

Gibt es je eine Python 2.x und eine 3.x Version, dann installieren wir für Python 3.x Numpy wie folgt:

```
pip3 install numpy
```

Syntax

1. keine Typ-Deklaration
2. Initialisierung ohne Semikolon
3. bei if-Anweisungen, Schleifen etc. keine Klammern, sondern Einrücken

1. Beispiel:

```
i = 0
while i < 10:
    print(i)
    i += 1
```

2. Beispiel:

```
for elem in range(10):
    print(elem)
```

Listen und Packages

3. Beispiel (Listen initialisieren):

```
l = []
l = [3] * 5                    # -> l = [3,3,3,3,3]
```

4. Beispiel (Listen verwenden):

```
l = [0,2,3,4,5]
l[0] = 1                       # -> l = [1,2,3,4,5]

start = 1
ende = len(l)
l_1 = l[start:ende]            # noch kuerzer: l[1:]
print(l_1)                     # -> l_1 = [2,3,4,5]

del l_1[2]                     # -> l_1 = [2,3,5]
l_1.append(7)                  # -> l_1 = [2,3,5,7]
l_1.extend([11,13,17])         # -> l_1 = [2,3,5,7,11,13,17]

print(l_1[-1])                 # -> l_1 = 17
```

Dictionaries

5. Beispiel (Dictionaries initialisieren):

```
d = {}
d = dict()
```

6. Beispiel (Dictionaries verwenden):

```
d = {"Vorlesung": "KI",
     "Studierende": 150,
     "Tutoren": ["Kunal", "Matthias"]}

for key, value in d.items():
    if key == "Studierende":
        d[key] += 25

if not "Prof" in d:
    d["Prof"] = "Prof. Dr. Lukowicz"

d.keys()    # -> ["Studierende", "Tutoren", "Prof", ...]
d.values()  # -> [175, ["Kunal", "Matthias"], ...]
```

if-then-else, Funktionen und Packages

7. Beispiel (Funktionen implementieren):

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    elif n > 1:
        return fib(n-1) + fib(n-2)
    else:
        return ["Are you crazy?", "Choose n > 0", -1]
```

8. Beispiel (Packages einbinden):

```
import numpy as np
```

Klassen und List Comprehension

9. Beispiel (Klassen, Methoden und Attribute):

```
class Person:
    def __init__(self, vorname, nachname, alter):
        self.vorname = vorname
        self.nachname = nachname
        self.alter = alter

    def altern_lassen(self):
        print("Juhu,", self.vorname, self.nachname,
              "hat heute Geburtstag.")
        self.alter += 1

ich = Person("Marie", "Mustermann", 21)
print(ich.alter)      # -> 21
ich.altern_lassen()
print(ich.alter)      # -> 22
```

10. Beispiel (List Comprehension):

```
l_2 = [elem for elem in range(100) if elem % 2 == 0]

foo = [i if i % 3 == 0 else False for i in range(100)]
```

Aufgaben (i)

1. Aufgabe

Implementiere eine Funktion `schaltjahr(`j`)` mit $j \in \mathbb{N}$, die berechnet ob ein Jahr j ein Schaltjahr ist. Dabei gelten folgende Regeln:

$$\texttt{schaltjahr(j)} := \begin{cases} \text{“Schaltjahr”}, & \text{wenn } j \text{ durch 400 teilbar ist} \\ \text{“kein Schaltjahr”}, & \text{wenn } j \text{ durch 100 teilbar ist} \\ \text{“Schaltjahr”}, & \text{wenn } j \text{ durch 4 teilbar ist} \\ \text{“kein Schaltjahr”}, & \text{sonst} \end{cases}$$

2. Aufgabe

Schreiben Sie eine Funktion `mse(l1,l2)` die zwei Listen `l1` und `l2` der Längen n nimmt und den *Mean Squared Error* (*MSE*) von `l1` und `l2` berechnet. Der *MSE* ist definiert als:

$$MSE(\texttt{l1},\texttt{l2}) := \frac{1}{n}\sum_{i=1}^{n}(\texttt{l1}_i - \texttt{l2}_i)^2$$

3. Aufgabe

Schreiben Sie eine Funktion `sieb(n)`, die alle Primzahlen $\leq n$ zurück gibt.

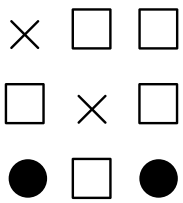
Aufgaben (ii)

4. Aufgabe

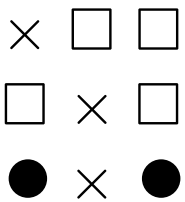
Implementieren Sie das Spiel *Tic Tac Toe*. Verwenden Sie zum Verwalten des Spielfeldes `feld` eine Klasse `Spielfeld`, in der Sie alle notwendigen Methoden implementieren.

Hinweis: Die Funktion `input()` ermöglichst es Ihnen Benutzereingaben einzulesen.

Beispiel:



Spieler 1 ist an der Reihe.
Geben Sie eine Position ein: *2,1*



Aufgaben (iii)

5. Aufgabe

Es sei $M := \{m_1, \ldots, m_n \,|\, \forall i \in \{1, \ldots, n-1\, m_i < m_{i+1}\}$ der Euromünzsatz und $b \in \mathbb{N}$ ein Geldbetrag in Euro. Das Münzwechsel-Problem (*MWP*) ist ein Optimierungsproblem, bei dem die Anzahl an Münzen zum Bezahlen des Betrags b minimiert werden soll. Das heißt, $S = \{s_1, \ldots, s_n\}$ ist eine optimale Lösung für M und b, genau dann wenn:

(i) $b = \sum_{i=1}^{n} s_i m_i$

(ii) $|S| = \min \left\{ |S'| \,\middle|\, S' \text{ ist eine Lösung für } M \text{ und } b \right\}$

Lösen Sie folgende Aufgaben:

(a) Schreiben Sie eine Funktion `greedy_wechsler(b,m)` die das Münzwechsel-Problem optimal für den Euromünzsatz löst und eine Laufzeit von $\mathcal{O}(n)$ hat.

(b) Was fällt ihnen auf, wenn wir den Münzsatz nicht als Integer sondern als Float definieren. Also statt 1 für die 1-Cent-Münze, verwenden wir 0.01.

(c) Angenommen wir verwenden nun einen anderen Münzsatz $M := \{1, 11, 23\}$. Löst Ihr obiger Algorithmus auch für diesen Münzsatz das *MWP* optimal? Begründen Sie ihre Aussage und schreiben Sie ggf eine neue Funktion `opt_wechsler(b,m)` die für jeden Münzsatz das *MWP* optimal löst.

Quellen

1 https://webscripts.softpedia.com/blog/today-s-top-3-programming-languages-javascript-python-jav shtml#sgal_0

2 https://businessoverbroadway.com/2019/01/13/programming-languages-most-used-and-recommended-by-data-s

3 https://redmonk.com/sogrady/2019/03/20/language-rankings-1-19/

4 https://docs.python.org/3/tutorial/venv.html

5 https://www.supinfo.com/articles/single/3425-the-pros-and-cons-of-python/

6 https://www.anaconda.com/end-of-life-eol-for-python-2-7-is-coming-are-you-ready/