

Praktikum Ingenieurinformatik

Teil 1, Programmieren

Termin 4
Funktionen, Debugger

1. Programmieren von Funktionen

2. Ermitteln von Primzahlen

3. Arbeiten mit dem Debugger

4. Übungsaufgaben

Aufgabe 1: Eingabe von ganzen Zahlen

Die Eingabe von ganzen Zahlen über die Tastatur kann mittels `x = int(input("Eingabe: "))` programmiert werden. Allerdings erfolgt in diesem Fall keine Überprüfung, ob der eingegebene Wert gültig ist – oder ob überhaupt eine Zahl eingegeben wird!

- Geben Sie das folgende Skript in Spyder ein und versuchen Sie zu verstehen, wie die Eingabe ganzer Zahlen hier gelöst ist.

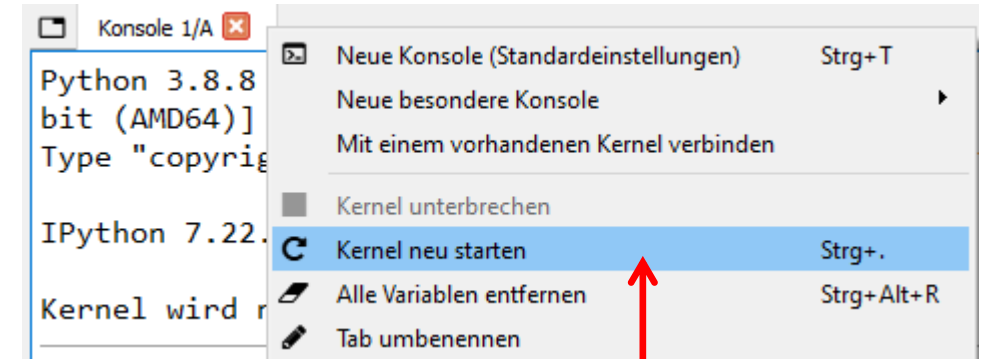
```
def input_int(msg, i_min, i_max):  
    i = int(input(msg))  
    while i < i_min or i > i_max:  
        print("Bitte korrekten Wert eingeben!")  
        i = int(input(msg))  
    return i  
  
my_int = input_int("Zahl 10 ... 20 eingeben: ", 10, 20)  
print(f"Es wurde {my_int} eingegeben")
```

- Was passiert, wenn eine **Zahl außerhalb des erlaubten Bereichs** eingegeben wird?
Was passiert, wenn statt einer Zahl **ein Text eingegeben** wird?

Aufgabe 2: Python-Module

Falls Sie die Funktion `input_int()` häufiger verwenden möchten, dann ist es sinnvoll, diese Funktion in einem separaten Python-Modul abzuspeichern. Erzeugen Sie dazu in Ihrem Praktikums-Verzeichnis **eine neue Datei `input_tools.py`** mit dem folgenden Inhalt:

```
def input_int(msg, i_min, i_max):  
    i = int(input(msg))  
    while i < i_min or i > i_max:  
        print("Bitte korrekten Wert eingeben!")  
        i = int(input(msg))  
    return i
```



Führen Sie einen **Neustart des Python-Kernels** durch und versuchen Sie anschließend, die Funktion `input_int()` aufzurufen.

```
In [1]: x = input_int("Test: ", 1, 10)
```

```
Traceback (most recent call last):
```

```
File "<ipython-input-1-66b6abcdad6e>", line 1, in <module>  
    x = input_int("Test: ", 1, 10)
```

```
NameError: name 'input_int' is not defined
```

Warum funktioniert das nicht?

Aufgabe 3: Aufruf von Funktionen aus anderen Modulen

Die Funktion `input_int()` im Python-Modul `input_tools.py` kann auf verschiedene Arten aufgerufen werden.

In [1]: `import input_tools`

A

In [2]: `x1 = input_tools.input_int("Variante A: ", 1, 10)`

Variante A: 15

Bitte korrekten Wert eingeben!

Variante A: 5

In [3]: `import input_tools as inp`

B

In [4]: `x2 = inp.input_int("Variante B: ", 1, 10)`

Variante B: 1

In [5]: `from input_tools import input_int`

C

In [6]: `x3 = input_int("Variante C: ", 1, 10)`

Variante C: 9999

Bitte korrekten Wert eingeben!

Variante C: 10

In [7]: `from input_tools import *`

D

In [8]: `x4 = input_int("Variante D: ", 1, 10)`

Variante D: 9

- Probieren Sie alle vier Varianten am eigenen Rechner aus.
- Welches sind die Vor- und Nachteile der vier gezeigten Varianten?

Funktionen aus der Python-Standardbibliothek

In der Python-Standardbibliothek gibt es viele Module mit praktischen Funktionen für alle möglichen Anwendungsfälle. **Standardmodule sollten nicht mittels `*` importiert werden** (Variante D auf der vorherigen Folie), da es aufgrund der sehr großen Anzahl von Funktionen schnell zu einem Durcheinander und ggf. zu Namens-Kollisionen kommen würde.

Das Skript zum Plotten einer Sinusfunktion (vergl. dritter Praktikumstermin) würde man daher besser so programmieren, wie es hier gezeigt ist.

Weitere Beispiele zur Arbeit mit Modulen finden sich im offiziellen Python-Tutorial:
<https://docs.python.org/3/tutorial/modules.html>

```
import matplotlib.pyplot as plt  
from math import sin, pi
```

} !

```
xx = [] # Liste mit x-Werten  
yy = [] # Liste mit y-Werten
```

```
x = -2 * pi  
while x <= 2 * pi:  
    y = sin(x)  
    xx.append(x)  
    yy.append(y)  
    x = x + 0.01
```

```
plt.plot(xx, yy, "r--")  
plt.grid(True)  
plt.show()
```

1. Programmieren von Funktionen

2. Ermitteln von Primzahlen

3. Arbeiten mit dem Debugger

4. Übungsaufgaben

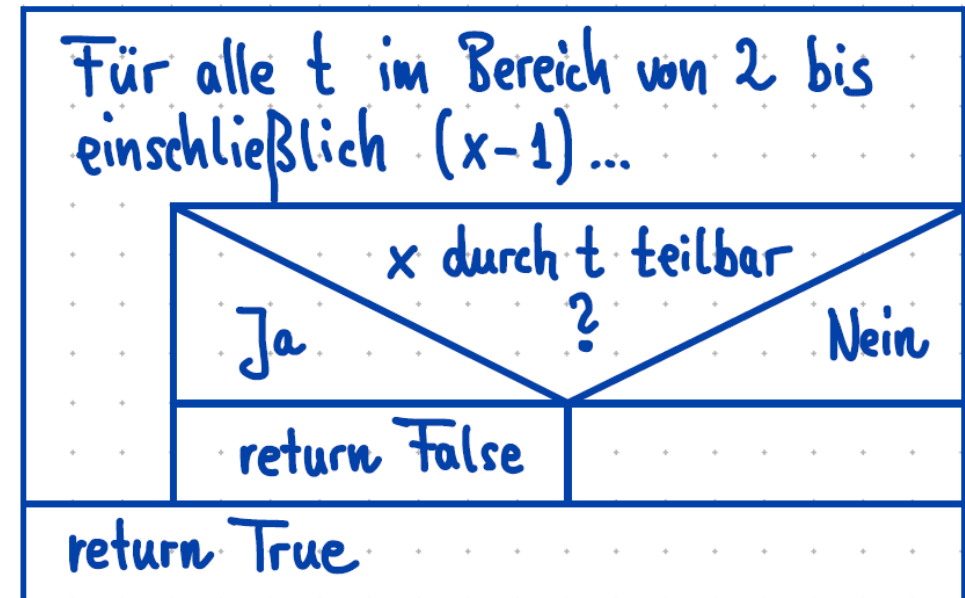
Aufgabe 4a: Primzahlen

Eine Primzahl (von lateinisch numerus primus „erste Zahl“) ist eine natürliche Zahl, die größer als 1 und **ausschließlich durch sich selbst und durch 1 teilbar** ist. *Quelle: [1]*

- Programmieren Sie die Funktion `check_prime(x)` zum Überprüfen, ob der Wert `x` eine Primzahl ist (Rückgabe der Funktion: `True`) oder nicht (Rückgabe: `False`).
- Sie können sich am abgebildeten Struktogramm orientieren.

```
def check_prime(x):  
    # TODO:  
    # Die Funktion check_prime()  
    # muss noch programmiert werden.  
  
for my_x in range(2, 20):  
    p = check_prime(my_x)  
    print(f"{my_x:2d} --> {p}")
```

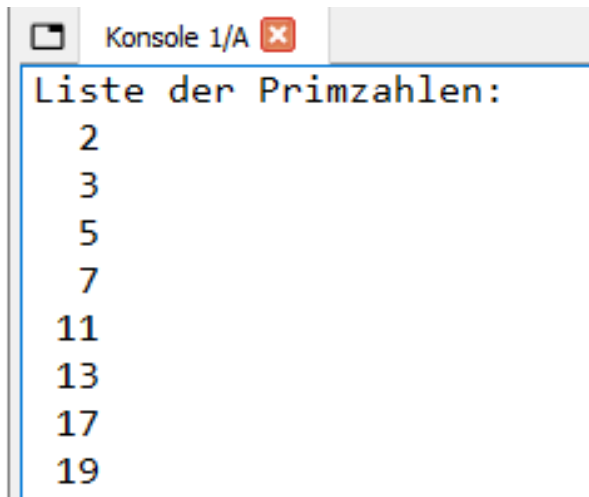
Funktion `check_prime(x)`:



Aufgabe 4b: Primzahlen

Ändern Sie das von Ihnen erstellte Skript so, dass auf dem Bildschirm nur noch eine Liste der Primzahlen ausgegeben wird (siehe Bildschirmfoto).

Zahlen, die keine Primzahlen sind, sollen also gar nicht auf dem Bildschirm erscheinen.



```
Konsole 1/A x
Liste der Primzahlen:
2
3
5
7
11
13
17
19
```

```
def check_prime(x):
    # TODO:
    # Die Funktion check_prime()
    # muss noch programmiert werden.

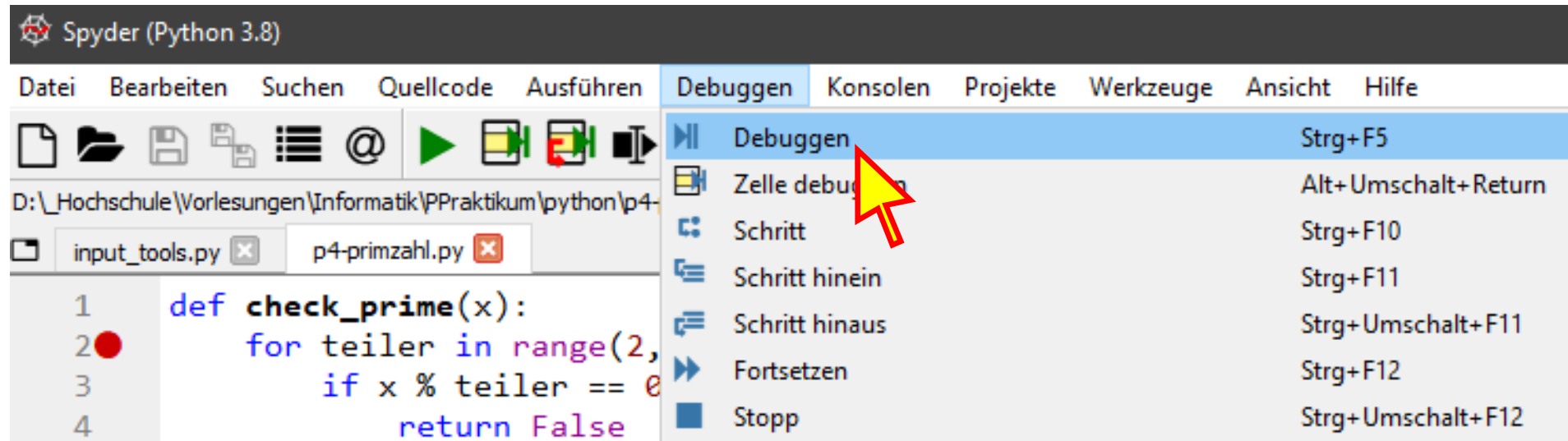
for my_x in range(2, 20):
    p = check_prime(my_x)
    print(f"{my_x:2d} --> {p}")
```

Hier müssen Sie
etwas ändern ...

1. Programmieren von Funktionen
2. Ermitteln von Primzahlen
3. Arbeiten mit dem Debugger
4. Übungsaufgaben

Der Debugger

Mit dem Debugger (bug = Fehler, Wanze) können Sie Programme an bestimmten Stellen (Breakpoints) anhalten oder ganz langsam, Zeile für Zeile ausführen. Sie können aktuelle Variablenwerte anschauen und sogar während der Laufzeit verändern.



Mit dem Debugger können Sie:

- Fehler im Programm finden,
- sich von der Korrektheit eines Programms überzeugen,
- den Ablauf unbekannter Programme verstehen.

Arbeiten mit dem Debugger in Spyder

- Per Mausklick auf den grauen Rand (direkt neben der Zeilennummer) oder alternativ mit der **Taste F12** wird ein **Breakpoint** entweder an den Beginn des Skripts oder an den Beginn des zu untersuchenden Bereichs gesetzt.
- Nun wird über den Menüpunkt Debuggen → Debuggen oder über die Tastatur mit **Strg+F5** der **Debugger gestartet**.
- Der Programmablauf stoppt am Breakpoint, die aktuelle Position im Programm wird durch einen kleinen Pfeil angezeigt.
- Die aktuellen Variablenwerte sind im „Variablenmanager“ in der Spyder-Benutzeroberfläche sichtbar.
- Mit **Strg+F10** wird die **nächste Programmzeile ausgeführt**.
- Es können auch mehrere Breakpoints gleichzeitig gesetzt werden.
- Über den **Menüpunkt Debuggen → Stopp** wird der Debugger wieder beendet.

Spyder-Benutzeroberfläche mit Debugger

The screenshot shows the Spyder IDE interface with the following components and annotations:

- Breakpoints:** A yellow box with a red arrow pointing to red dots on line 2 and line 7 of the code editor.
- Aktuelle Position:** A yellow box with a red arrow pointing to the blue arrow on line 9 of the code editor.
- Aktuelle Variablenwerte:** A yellow box with a red arrow pointing to the variable manager table.
- Bedienung:** A yellow box containing keyboard shortcuts for debugger actions.

Code Editor Content:

```
def check_prime(x):  
    for teiler in range(2, x):  
        if x % teiler == 0:  
            return False  
    return True  
  
for my_x in range(2, 20):  
    p = check_prime(my_x)  
    print(f"{my_x:2d} --> {p}")
```

Variable Manager Table:

Name	Typ	Größe	Wert
my_x	int	1	2
p	bool	1	True

Bedienung

- F12 – Breakpoint setzen
- Strg+F5 – Debugger starten
- Strg+F10 – Einzelschritt

1. Programmieren von Funktionen
2. Ermitteln von Primzahlen
3. Arbeiten mit dem Debugger
4. Übungsaufgaben

Übungsaufgabe 1

- Schreiben Sie eine Funktion `fill_rand(n)`. Diese Funktion füllt eine Python-Liste **mit n Zufallszahlen im Bereich von -10 bis 10** und gibt die gefüllte Liste als Rückgabewert zurück.
Tipp: `from random import random`
- Schreiben Sie eine Funktion `max_abs(lst)`. Diese Funktion sucht in der Liste `lst` das **Element mit dem größten Betrag** und gibt es als Rückgabewert zurück. Wenn die Liste `lst` leer ist, wird 0 (null) zurückgegeben.
Versuchen Sie, diese Aufgabe **ohne die Funktionen `min()` und `max()`** zu lösen.
- Programmieren Sie ein Skript, das nach der Eingabe der Listenlänge `n` die beiden Funktionen `fill_rand(n)` und `max_abs(lst)` aufruft (siehe Bildschirmfoto).

Name	Typ	Größe	
lst	list	10	[-4.718321433198449, 3.5639390191894833, 0.46319836876...
m	float	1	-8.922127767974281
n	int	1	10

Konsole 1/A

```
In [1]: runfile('D:/_Hochschule/Vorle  
PPraktikum/python')
```

Anzahl n: 10
Element mit max. Betrag = -8.92212776

```
In [2]:
```

Index	Typ	Größe	
0	float	1	-4.718321
1	float	1	3.5639390
2	float	1	0.4631983
3	float	1	-5.246216
4	float	1	5.1062467
5	float	1	-2.929722
6	float	1	-4.088758
7	float	1	8.9186103
8	float	1	2.5671678
9	float	1	-8.922127

Übungsaufgabe 2

- Schreiben Sie eine Funktion `min_max_abs(lst)`, welche die beiden Elemente mit dem größten und mit dem kleinsten Betrag in der Liste `lst` sucht. **Beide Werte werden als Rückgabewerte zurückgegeben.** Bei einer leeren Liste wird jeweils der Wert 0 (null) zurückgegeben.
- Ändern Sie das Programm aus der Übungsaufgabe 1 so, dass statt `max_abs(lst)` jetzt die neue Funktion `min_max_abs(lst)` aufgerufen wird.
- Das folgende Beispiel zeigt eine **Funktion mit mehreren Rückgabewerten**:

```
def demo123() :  
    a = 1  
    b = 2  
    c = 3  
    return a, b, c  
  
# Rückgabewerte in separaten Variablen speichern  
r1, r2, r3 = demo123()  
print(f"Rückgabewerte: {r1}, {r2} und {r3}.")
```


Übungsaufgabe 3

- Bei der Eingabe von int-Werten mittels `n = int(input("n eingeben"))` führt die Eingabe von „abc“ zu einem sog. **ValueError-Ausnahmefehler (engl. Exception)**, weil der Text „abc“ nicht in eine int-Zahl konvertiert werden kann.
- Ausnahmefehler **müssen mittels try ... except abgefangen werden**, wenn sie nicht zu einem Programmabbruch führen sollen.

```
p4-ue3-try-except.py x
1  n = 0
2  try:
3      n = int(input("n eingeben: "))
4      print(f"Eingabe = {n}")
5  except ValueError:
6      print("Bitte gültige Zahl eingeben!")
```

```
Konsole 1/A x
In [1]: runfile('D:/_Hochschule/Vorle
PPraktikum/python')
n eingeben: 123
Eingabe = 123

In [2]: runfile('D:/_Hochschule/Vorle
PPraktikum/python')
n eingeben: abcdefg
Bitte gültige Zahl eingeben!
```

- Erweitern Sie das Programm aus der Übungsaufgabe 2 so, dass Fehleingaben nicht mehr zu einem Programmabbruch führen. Bei einer Fehleingabe soll stattdessen ein „freundlicher Hinweis“ auf dem Bildschirm ausgegeben werden.

Quellenverzeichnis

- [1] Seite „Primzahl“. In: Wikipedia – Die freie Enzyklopädie.
Bearbeitungsstand: 23. Juli 2022, 19:09 UTC. URL:
<https://de.wikipedia.org/w/index.php?title=Primzahl&oldid=224757327>
(Abgerufen: 9. August 2022, 20:56 UTC)