Technische Hochschule Nürnberg

Übungen

FWPM Skriptsprache Python

Sommersemester 2025

Prof. Dr.-Ing. Jürgen Krumm

HINWEIS: Die nachfolgenden Aufgaben beziehen sich auf Python mindestens ab der Version 3.11.

EMPFEHLUNG: Damit Sie Ihre Lösungen auch noch später anschauen können, speichern Sie am besten Ihre Python-Skripte mit den Aufgabennummern im Dateinamen ab.

Variablen und Ausdrücke

1. Aufgabe: Überprüfen Sie, ob die in der nachfolgenden Tabelle angegebenen Namen von Python als gültige Variablennamen akzeptiert werden. Schreiben Sie für ungültige Namen eine korrekte Version in die zweite Spalte der Tabelle. Versuchen Sie dabei, maximal ein Zeichen hinzuzufügen oder, wo das nicht ausreicht, nur die störenden Zeichen durch ein möglichst neutrales Zeichen zu ersetzen.

Name	Korrektur
`3a`	a3 oder _3a
`Änderung`	
`ÄNDERUNG`	
`Mit,1`	Mit_1
`Label$3`	
`Drei-fach-Wert`	Drei_fach_Wert
`Füße_gemessen`	

---

2. Aufgabe: Was ergeben die folgenden Python-Ausdrücke als Wert und Datentyp? HINWEIS: Wenn Sie ohne ausprobieren nicht auf die Lösung kommen, können Sie in Python mit der Funktion `type(x)` den Typ eines Ausdrucks `x` zurückgeben lassen.

```
5 / 2     2.5
5 // 2    2
5 % 2     1
5.5 % 2   1.5
3-3*2      -3
(1+2)/3    1
3 & 2       2 # da bitweise und
3 | 2       3
not 0       1
2**2000       2^2000 #als int
2.0**200      2^200 #als float
```

Zeichenketten / Formatierung

3. Aufgabe: Erklären Sie, wie Python die folgenden Code-Zeilen interpretiert. Welche Werte haben die Variablen nach den jeweiligen Zuweisungen?

```
wert_1="An" + "merkung"
wert_2="-"*3 + " Lösung " + 3*"-"
wert_3=kein_wert
wert_4=7*"7" + 8
wert_4
```

Anmerkung
--- Lösung ---
Fehler
777777 +8 -> Fehler beim Anhänge
wert_4 nicht definiert

---

4. Aufgabe: Die Variable `erg` wird über einen Formatierungstext aus den Werten den Variablen a und b gebildet nach der folgenden Form:

fmt muss zuvor als String und a, b als Variable definiert sein

```
erg = fmt % (a, b)
```

Wie lautet die Zeichenkette `fmt`, mit der für a=1.2 und b=4e-3 folgender Text in `erg` gespeichert wird?

fmt = " 'a={}, b={}' ".format(a,b)

```
'a=1.2, b=0.004'
```

Wie kann die gleiche Zuweisung `erg=…` mit einer Formatierung über einen F-String erreicht werden?

erg=f" 'a={a}, b={b}' "

---

in VS-Code

5. Aufgabe: Eine formatierte Ausgabe der Variablen a und b ergab für a=12 und b=4 und den Aufruf `txt % (a,b,a)` die Darstellung: `a/dez=12, b/dez=4, a/hex=0xc.`

Schreiben Sie eine passende print-Anweisung über die Funktion `str.format()` zur formatierten Ausgabe der beiden Variablen.

Wie sieht die print-Anweisung über die Formatierung mit einem F-String aus?

---

6. Aufgabe: Erzeugen Sie durch Slicing und Striding aus der Zeichenkette `txt=7*"ABC"` eine neue Zeichenkette, die nur die Buchstaben C enthält. Wie lautet der entsprechende Ausdruck? Wie lautet die sich ergebende Zeichenkette?

txt[2::3] #ohne Endwert -> ganzes Array

---

7. Aufgabe: Schreiben Sie einen einfachen XML-Parser, der XML-Tags „<…>“ erkennen und auf dem Bildschirm ausgeben kann. Tag-Attribute und die gesonderte Behandlung von Zeichenketten können Sie ignorieren. Verwenden Sie als Textvorlage für den XML-Inhalt das Skriptfragment `xml_vorlage.py`.

Listen / Tupel / Indizierung

8. Aufgabe: Was gibt das folgende Skript aus? Welche Typen haben die Variablen a, b, c und d nach ihren entsprechenden Zuweisungen?

```
a=[11,22,33]
b="Nachts um zwei Uhr"
c=(a, b, None)
d=[(1,2,3)]
print(a[2:4])
print(a[::-1])
print(len(a), len(b), len(c), len(d))
print(c)
```

---

9. Aufgabe: Streichen Sie im folgenden Skript die Zeilen heraus, die nicht korrekter Python-Code sind. Begründen Sie, warum Python diese Zeilen nicht akzeptiert. Was geben die nicht-gestrichenen print-Aufrufe aus?

   HINWEIS: Sie können das Skript mit dem Interpreter testen und dabei fehlerhafte Anweisungen löschen oder besser auskommentieren.

```
a=(1)
b=([1],[2])
c="Sonnenschein"
print(a[0])
c[1]="O"
b[0]=3
d=b
b[0][0]=4
d[1].append(None)
print(a,b)
print(len(d))
```

   Was ergeben die Zuweisungen an die Variablen `b` und `d`?

---

10. Aufgabe: Formulieren Sie die Zuweisung eines Tupels mit dem einzigen Element `True` an die Variable `t`. Wie kann der Wert dieses Elements nachträglich auf `False` geändert werden?

---

11. Aufgabe: Übersetzen Sie das folgende C-Programm in ein Python-Skript mit gleicher Funktion. Das C-Feld `data[]` können Sie mit einer Liste aus Tupeln realisieren. HINWEIS: Mit dem Python-Befehl `for` können Sie über eine Liste iterieren und mit `continue` in den nächsten Iterationsdurchlauf springen.

```
// File: cprog.c
#include <stdio.h>

struct WinRect {
 int x, y, w, h;
 char *descr;
};

struct WinRect data[] = {
 { 10, 5, 5, 5, "menu" },
 { 0,0, 200, 20, "titlebar" },
 { 50, 30, 15, 5, "end button" },
};

int main() {
 int tx = 60; // test-x
 int ty = 32;  // test-y

 int nelems = sizeof(data) / sizeof(struct WinRect);
 for (int idx = 0; idx < nelems; idx++) {
  int dx = tx - data[idx].x;
  int dy = ty - data[idx].y;

  if ((dx < 0) || (dx >= data[idx].w)) continue;
  if ((dy < 0) || (dy >= data[idx].h)) continue;
  printf("test point (%d,%d) inside rect '%s'\n",
             tx, ty, data[idx].descr);
 }
 return 0;
}
```

Dictionary

12. Aufgabe: Für eine Supermarkt-Software sollen Lebensmittel in einer Datenbank als Python-Dictionary über Zahlen kodiert werden. Testweise existieren bereits folgende Zuordnungen:

    1000=Kartoffeln, 1020=Gurken, 720=Bananen, 702=Kiwis, 5000=Schokolade, 5010=Kartoffelchips

    Schreiben Sie ein Python-Skript, das nach Eingabe einer Zahl den Lebensmittelnamen ausgibt. Eine unbekannte Zahl soll entsprechend zurückgemeldet werden.

Funktionen

13. Aufgabe: Schreiben Sie ein Python-Skript mit zwei Funktionen `my_and(a,b)` und `my_or(a,b)`, die das Verhalten der Operatoren `and` und `or` mit if-Abfragen nachbilden. Berechnen Sie die Ergebnisse der folgenden Logik-Ausdrücke nur durch Anwendung Ihrer Funktionen `my_and()` und `my_or()`. Vergleichen Sie Ihre Lösung mit der Berechnung über die Operatoren. Hier die zu überprüfenden Logik-Ausdrücke:

```
"Adam" or False and "Eva"

("Adam" or False) and "Eva"

"Adam" or (False and "Eva")
```

HINWEIS: Schauen Sie mit „`help("and")`“ oder „`help("or")`“, wie die beiden Logik-Operatoren arbeiten.

---

14. Aufgabe: Schreiben Sie ein Python-Skript mit folgenden Funktionen:
    1) Die Funktion `get_ones(val)` soll die Anzahl der 1-Bits im übergebenen int-Parameter `val` ermitteln und zurückgeben.

    2) Die Funktion `sort_by_ones(seq)` gibt eine sortierte Version der Liste `seq` zurück. Sortierkriterium ist die Anzahl der 1-Bits in den Listenelementen.
    HINWEIS: Zum Sortieren können Sie die Sortierfunktion `list.sort()` nutzen. Mit `list.sort(key=func)` lässt sich auch eine Funktion angeben, mit der die Wertigkeit der Listenelemente beim Sortieren berechnet werden kann.

       Testen Sie die Funktion z.B. mit der Liste `[1,0x21,7,0x80,0,0b1111]`.

    3) Die Funktion `get_bits_as_list(val)` soll aus einer Ganzzahl `val` eine Liste aus Einsen und Nullen der einzelnen Bitpositionen generieren und zurückgeben. Das erste Listenelement (Index 0) soll das höchstwertige Bit sein.
    Beispiel: `get_bits_as_list(27)` → Rückgabe: `[1, 1, 0, 1, 1]`

---

15. Aufgabe: Schreiben Sie ein Python-Skript, bei dem eine Liste von Zeichenketten, die Ganzzahlen repräsentieren, nach den Beträgen der Zahlenwerte sortiert werden soll. Sortieren Sie mit der Methode `list.sort()`, bei der Sie eine Lambda-Funktion zur Ermittlung der Zahlenwerte angeben.

    Beispiel: Ausgangsliste ["0023", "-11", "7"] → Ergebnis ['7', '-11', '0023'].
    HINWEIS: Zeichenketten in Zahlen umwandeln können Sie zum Beispiel mit `int()` und den Betrag bilden mit `abs()`.

    Welchen Vorteil hat hier die Verwendung der Lambda-Funktion?

---

16. Aufgabe: Schreiben Sie das folgende Programm so um, dass die match-case-Anweisungen (erst ab Python 3.10 verfügbar) durch if-elif-Abfragen ersetzt werden. Können match-case-Anweisungen immer so einfach durch if-elif-Abfragen ersetzt werden? Welche Bedeutung haben die im Funktionskopf angegebenen Typ-Annotationen?

```
def parse_retcode(code : int) -> str:
    match code:
        case 200: return "OK"
        case 301: return "Moved Permanently"
        case 400: return "Bad Request"
        case _: return "other reason"

while True:
    code = int(input("Enter status code: "))
    if code == -1: break
    print(parse_retcode(code))
```

Dateien Lesen und Schreiben

17. Aufgabe: Schreiben Sie ein Skript, das den Inhalt eines Dictionary als XML-Datei ausgibt. Erzeugen Sie die XML-Datei in der Funktion `write_xml(filename, dictionary)`. Beispielsweise solll das Dictionary `d=dict(a=3, b=5, c="hallo")` die folgende XML-Datei ergeben:

```
<?xml version="1.0" encoding="UTF-8"?>
<dict>
 <entry>
  <key>a</key>
  <value>3</value>
 </entry>
 <entry>
  <key>b</key>
  <value>5</value>
 </entry>
 <entry>
  <key>c</key>
  <value>hallo</value>
 </entry>
</dict>
```

HINWEIS: Auf die eigentlich notwendige Kodierung von Sonderzeichen wie z.B. < oder > in Zeichenketten aus dem Dictionary können Sie in dieser Aufgabe verzichten.

Erweiterungsmöglichkeit bei Interesse: Lesen Sie mit Ihren XML-Parser aus Aufgabe 7 die XML-Daten wieder in ein Dictionary ein. Es reicht, alle Werte als Zeichenketten im Dictionary abzuspeichern. Für diese Aufgabe sollte Ihr XML-Parser jetzt auch den Text zwischen zwei Tags verarbeiten. Das Einlesen können Sie z. B. in einer Funktion `read_xml(filename)` realisieren, die ein Dicitonary zurückgibt.

Module / Pakete

18. Aufgabe: Für das Modul `temperature` (Datei *temperature.py*) aus der Präsentation implementieren Sie jetzt die fehlenden Funktionen. Führen Sie dazu folgende Schritte aus:

    - Führen Sie die Datei *temperature.py* als eigenes Skript aus. Was gibt Python dabei aus? Was passiert im Vergleich dazu, wenn Sie die Datei in der Python-Shell oder aus einem anderen Python-Skript mit „`import temperature`“ laden? Welche Bedeutung hat die if-Abfrage mit der Variable `__name__` am Schluss von *temperature.py*?
    - Implementieren Sie die beiden Umrechnungsfunktionen zwischen Celsius- und Fahrenheit-Angaben (`celsius2fahrenheit()` und `fahrenheit2celsius()`).

---

19. Aufgabe: Das Modul `temperature` (Datei *temperature.py*) aus der vorherigen Aufgabe soll jetzt Teil eines Pakets mit dem Namen `conv` werden. Führen Sie zur Erstellung des Pakets folgende Schritte aus:

    - Legen Sie z.B. mit einem Dateisystem-Explorer oder der Kommandozeile Ihres Computers einen Ordner `conv` an. Dieser Ordner dient als Paketverzeichnis.
    - Kopieren Sie die Datei *temperature.py* in das Paketverzeichnis.
    - Implementieren Sie ein zweites Modul von Paket `conv` mit dem Namen `distance` Im Paketverzeichnis. Darin sollen die beiden Funktionen `meter2inch()` und `inch2meter()` zwischen Meter- und Zoll-Angaben umrechnen. Dokumentieren Sie die Umrechnungsfunktionen und kurz das Module selbst am Anfang der Quelldatei. Überprüfen Sie, wie die Dokumentationstexte im Hilfesystem angezeigt werden.
    - Schreiben Sie ein Test-Skript, das die Module von Paket `conv` lädt und die Umrechnungsfunktionen überprüft, z.B. durch Hin- und Rückkonvertieren der Angaben.

---

20. Aufgabe: Ermitteln Sie mit der Funktion `glob()` aus dem Modul `glob` alle Python-Quelldateien (Dateiendung .py) im aktuellen Verzeichnis. Durchsuchen Sie alle diese Dateien nach dem Text `print`. Wenn der Text in einer Datei enthalten ist, lassen Sie den Namen dieser Datei ausgegeben.

Fehlerbehandlung

21. Aufgabe: Überlegen Sie sich mindestens 5 Fehler, die im Python-Interpreter zur Laufzeit eines Skripts auftreten können. Versuchen Sie herauszufinden, wie die entsprechende Exception von Python heißt.

---

22. Aufgabe: Welche Laufzeitfehler können Sie beim folgenden Python-Skript erzeugen?

```
idx = 0
val = 2
seq=[1, 2]

neuwert = seq[idx]
schwelle = 42 / val
print("Schwelle =", schwelle)
```

Nennen Sie mindestens zwei verschiedene Fälle. Schreiben Sie eine Ausnahmebehandlung mit dem `try`-Befehl, die diese Fehler abfängt. Jeder dieser Fehler soll über eine eigene print-Ausgabe erklärt werden. Alle anderen Fehler sollen die allgemeine Meldung „unbekannter Fehler“ hervorrufen.

Datenstrukturen / Objekte / Klassen

23. Aufgabe: Schreiben Sie Ihr Skript aus Aufgabe 11 so um, dass statt einem Tupel eine Python-Klasse namens `WinRect` zum Speichern der Datenwerte verwendet wird. Außerdem soll jetzt eine Methode `contains(x,y)` in `WinRect` vorhanden sein, mit der ermittelt wird, ob ein Punkt im Rechteck enthalten ist.

---

24. Aufgabe: Die `WinRect`-Instanzen aus Aufgabe 23 werden jetzt zur Laufzeit des Programms aus der Textdatei *rects.txt* gelesen. In der Textdatei steht in jeder Zeile ein Rechteck. Die Werte des Rechtecks (x, y, …) stehen durch Doppelpunkt („:“) getrennt auf der Zeile.

    Speichern Sie die `WinRect`-Instanzen in einem Dictionary und in einer Liste. Sortierschlüssel für das Dictionary soll die Textbeschreibung jedes Rechtecks sein.

    Ihr Programm soll auch nach einer Textbeschreibung fragen und für das zugehörige `WinRect` im Dictionary dann x, y, w und h ausgeben. Fehlt die Textbeschreibung im Dictionary, soll das mit einem erklärenden Text zurückgemeldet werden.

---

25. Aufgabe: Für ein Grafikprogramm sollen Formen wie z.B. Linien und Rechtecke mit Hilfe von Klassen verwaltet werden. Basisklasse ist die Klasse *Shape.* Von dieser Basisklasse leiten Sie die anderen Klassen für die tatsächlichen Formen ab.

    Jede Instanz der Klasse *Shape* und aller davon abgeleiteten Klassen hat eine eindeutige Zahlen-ID als privates Attribut. HINWEIS: Erzeugen Sie die ID durch Hochzählen eines Klassenattributs von *Shape* in __init__().

    Schreiben Sie für die Klasse *Shape* die folgenden Methoden:

`__init__()`	Initialisiere die Form.
`getID()`	Gib ID der Form zurück
`draw(painter)`	Stelle die Form dar. Hier genügt die Ausgabe der Daten auf der Konsole, siehe z.B. die Testausgabe unten. Der Parameter `painter` wird erst bei der Programmierung mit Qt benötigt, soll aber schon hier eingeführt werden.
`moveBy(dx, dy)`	Verschiebe die Form um die Deltawerte `dx` und `dy`. Bei der Basisklasse *Shape* ist hier keine Aktion nötig.

    Schreiben Sie die Klassen *Rect* und *Line* als Ableitungen der Basisklasse *Shape*. Überschreiben Sie in *Rect* und *Line* soweit als notwendig die Methoden von *Shape*. HINWEIS: Sie können `__init__()` von *Shape* in den Initialisieren von *Rect* und *Line* folgendermaßen aufrufen:

```
super().__init__()
```

    Testen Sie Ihr Programm mit dem folgenden Code:

```
s1 = Rect(0, 0, 10, 50)
s2 = Line(0, 0, 50, 10)

painter = None
s1.draw(painter)
s2.draw(painter)

s2.moveBy(5, 10)
s2.draw(painter)
```

    Der Code ergibt z.B. folgende Ausgabe:

```
rect: id #0 x=0, y=0, w=10, h=50
line: id #1 x1=0, y1=0, x2=50, y2=10
line: id #1 x1=5, y1=10, x2=55, y2=20
```

26. Aufgabe: Definieren Sie jetzt zusätzlich zu den Klassen aus der Aufgabe 25 die von *Shape* abgeleitete Klasse *Group*. Die Klasse *Group* fasst mehrere Formen zu einer gemeinsamen Gruppe zusammen.

    Programmieren Sie für die Klasse *Group* die folgenden zusätzlichen Methoden:

`addShape(obj)`	Füge die Form obj an die Formengruppe an
`__iadd__(obj)`	Führe für Form `obj` die Methode `addShape(obj)` aus und gib den self-Zeiger des Klassenobjekts zurück

    Überschreiben Sie in *Group* soweit als notwendig die Methoden von *Shape*. Welche besondere Bedeutung hat die Methode `__iadd__()` für Python?

    Testen Sie Ihr Programm mit dem folgenden Code:

```
s1 = Rect(0, 0, 10, 50)
s2 = Line(0, 0, 50, 10)
s3 = Group()
s3.addShape(s1)
s3 += s2

painter = None
s2.draw(painter)
s3.moveBy(5, 10)
s3.draw(painter)
```

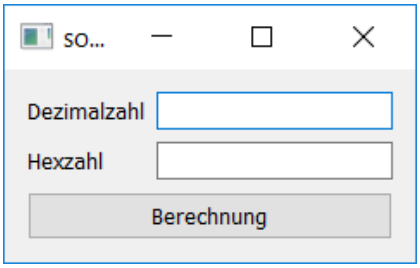
    Der Code ergibt z.B. folgende Ausgabe:

```
line: id #1 x1=0, y1=0, x2=50, y2=10
group: id #2 num_shapes=2
rect: id #0 x=5, y=10, w=10, h=50
line: id #1 x1=5, y1=10, x2=55, y2=20
```

GUI-Entwicklung mit Qt

27. Aufgabe: Schreiben Sie ein Qt-Programm, mit dem Sie Dezimalzahlen hexadezimal darstellen können. Die Dezimalzahlen werden in ein Textfeld eingegeben. Nach Drücken eines Berechnungsbuttons erscheint in einem zweiten Textfeld die Darstellung mit Hexadezimalziffern.

    Die Eingabemaske kann z.B. wie nebenstehend gezeigt aussehen. Im Beispiel wird das Widget `QLabel` für die Darstellung von Beschreibungstext und das Widget `QLineEdit` für die Ein-und Ausgabe der Zahlen verwendet. Recherchieren Sie z.B. im Internet, wie Sie diese Widgets benutzen können.



---

28. Aufgabe: Schreiben Sie ein Qt-Programm zur Darstellung einer Vektorgrafik auf dem Bildschirm. Für die Elemente der Vektorgrafik können Sie die Klassen aus Aufgabe 26 anpassen oder erweitern.

    HINWEIS: Sie können außerdem das Beispielprogramm *tb_qt_paint.py* als Vorlage verwenden. Die draw()-Methoden der einzelnen Shapes sollen jetzt in den Painter zeichnen anstatt print() zu verwenden.

    Lassen Sie sich z.B. die folgende Formengruppe `g` darstellen:

```
g=Group()
g.addShape(Line(20,  50, 20, 100))
g.addShape(Line(20, 100, 40, 110))
g.addShape(Line(40, 110, 60, 100))
g.addShape(Line(60, 100, 60,  50))
g.addShape(Line(60,  50, 40,  40))
g.addShape(Line(40,  40, 20,  50))
g.addShape(Line(25,  85, 40,  95))
g.addShape(Line(40,  95, 55,  85))
g.addShape(Rect(45, 60, 5 ,5))
g.addShape(Rect(30, 60, 5 ,5))
```

    Erweiterungsmöglichkeit: Ändern Sie das Programm so ab, dass Sie mit den Cursortasten Links / Rechts / Hoch / Runter die Position der Formengruppe um jeweils 5 Pixel in die entsprechende Richtung verschieben können.

    Anmerkungen:
    - Tastatureingaben für ein Qt-Element können Sie z.B. durch Deklarieren der Methode `keyPressEvent()` abfragen.
    - Mit der Methode `update()` können Sie ein Qt-Element neu zeichnen lassen.

Arbeiten mit NumPy und Matplotlib

29. Eine elektrische Schaltung hat die folgende Übertragungsfunktion:

$$H(f) = \frac{1}{1 + \mathrm{j}2\pi fCR}$$

Schreiben Sie ein Programm, das mit matplotlib den Frequenzgang als Bodeplot (dB-Betrag und Phase) im Frequenzbereich 10 Hz bis 10 MHz ausgibt.
Rechnen Sie mit C=1 nF und R=1590 Ω.

HINWEISE:

- Beim Bode-Plot ist die Frequenz auf der X-Achse logarithmisch aufgetragen
- Beim Betragsgang ist die Y-Achse als dB-Auftragung dargestellt.
- Den Betrag einer komplexen Zahl `x` können Sie mit `numpy.abs(x)` ausrechnen, den Winkel mit `numpy.angle(x)`. und den 10er-Logarithmus mit `numpy.log10(x)`.