Grundkenntnisse zu Python, die benötigt werden

Zahlen und strings

Zahlen [integer, float] und Zeichenketten [string] werden in Python als verschiedene Datentypen behandelt. Die Zahl 123 wird daher anders behandelt als die Zeichenkette mit den Zeichen 1, 2 und 3 hintereinander. Das Problem ist, dass man bei der Arbeit zwar für die Zeichenkette "123“ oder '123' schreiben muss und nicht 123, weil das eben die Schreibweise einer Zahl ist, andererseits bei der Ausgabe das Ergebnis von print("123") und print(123) dasselbe ist.
Weil diese Unterscheidung zwischen Zahlen und strings gemacht wird, braucht Python Funktionen zur Umwandlung einer Zeichenkette in eine Zahl und umgekehrt:
int("123") → 123
str(123) → "123"
Mit strings und zahlen gleichzeitig kann man nicht rechnen. "123"+5 geht nicht, allerdings...
"123"+"5" → "1235" und natürlich
123+5 → 128

Definition von Variablen

Wenn unser Programm sich etwas merken soll, brauchen wir eine Variable. Eine Variable ist ein Platz im Speicher, die man mit einem Namen ansprechen kann. Für dieses "Ansprechen" gibt es zwei Varianten, den lesenden Zugriff – dann wollen wir den Wert haben, den sich das Programm gemerkt hat – und den schreibenden Zugriff – dann wollen wir den Wert setzen, den das Programm sich merken soll. Damit das Programm diese beiden Typen von Zugriffen unterscheiden kann, muss es eine allgemein verbindliche Schreibweise dafür geben, das ist die …

Wertzuweisung

Die Anweisung

```
zahlzeichen="123"
```

legt im Speicher – wenn es diese Variable noch nicht gibt – eine neue Variable mit dem Namen zahlzeichen an und weist ihr den Wert "123" zu. Wenn es diese Variable schon gibt, wird ihr bisheriger Wert auf den neu zugewiesenen Wert gesetzt.
In der Anweisung

```
zahl=int(zahlzeichen)
```

finden wir außer der Wertzuweisung in die Variable zahl auch einen lesenden Zugriff auf die Variable zahlzeichen. zahlzeichen wird also gelesen, der Funktion int (…) übergeben, welche die 123 als Wert zurück gibt, die dann der Variablen zahl zugewiesen wird.

wxTextCtrl

Ein wxTextCtrl enthält grundsätzlich in seinem Textfeld einen string. Wenn wir den Inhalt als Zahl interpretieren wollen, müssen wir ihn zunächst in eine Zahl umwandeln. Bei ganzzahligen Werten ist also die Funktion int(...) zu verwenden, bei nicht ganzzahligen Werten die Funktion float(...).
Probleme gibt es, wenn die Zeichenkette im Textfeld keine Zahl ergibt. Dann ergibt die Umwandlung einen Fehler, auf den das Programm reagieren sollte. Für solche Fälle gibt es in Python eine Fehlerbehandlung, zu der wir aber wohl nicht mehr kommen werden. In unseren Programmen müssen wir dann darauf vertrauen, dass der Benutzer keine

fehlerhafte Eingabe macht.

= ist nicht gleich

Das Zeichen = steht also nicht für den Vergleich von zwei Ausdrücken, wie wir das von der Mathematik her kennen. Daher ist eine Programmzeile wie

```
betrag = betrag + 10
```

durchaus sinnvoll. Sie bedeutet: Lies den bisherigen Inhalt der Variablen betrag, addiere dazu 10 und speichere das Ergebnis nun als neuen Wert von Betrag ab.

Übung:

Schreibe die Programmzeilen zu den folgenden Aufgabenstellungen auf:

- Erhöhe den Wert der Variablen x um 1000.
- Zähle den Wert der Variablen stand weiter.
- Verdopple den Wert der Variablen anzahl.
- Speichere in der Variablen name den sinnvoll verbundenen string aus den strings in den Variablen vorname und nachname.
- Lies den Inhalt aus dem wxTextCtrl – Objekt **textCtrl1**, von dem wir annehmen, dass er für eine Zahl steht, zaehle den Inhalt aus einer Variablen betrag hinzu und speichere das Ergebnis wieder in **textCtrl1** ab.

Denk dir weitere Übungen aus und lass sie deine Nachbarin / deinen Nachbarn bearbeiten.

Aber == ist gleich und != ist ungleich

Sehr gewöhnungsbedürftig, nicht wahr! Also ergibt jeweils …

```
print( 3==4 )  → False
print( 4==4 )  → True
print( 3!=4 )  → True
print( 4!=4 )  → False
```

Man braucht das in Bedingungen.

Übung:

1. Schreibe die Ausgaben zu den folgenden Programmzeilen auf:

```
pruefzahl=22
if (pruefzahl==22):
   print( "22" )
else:
   print( "anders" )
pruefzahl=70
if (pruefzahl==22):
   print( "22" )
else:
   print( "anders" )
```

2. Schreibe eine Prüffunktion, die einen string aus vier Zeichen überprüft, ob er mit dem Inhalt einer Variablen passwort übereinstimmt.

Verzweigungen

In dem auf der vorigen Seite angegebenen Programm wird eine (Programm-)Verzweigung benutzt. Von einer Verzweigung spricht man, wenn ein Programm zwei (oder mehr) verschiedene Wege weiter verfolgen kann. Dabei muss eindeutig klar sein, welche der Varianten zu wählen ist, der weitere Verlauf muss also von einer Bedingung abhängig gemacht werden.

Nach if folgt eine Bedingung

Daher folgt auf das Wort if, das eine Verzweigung einleitet immer zunächst die zu prüfende Bedingung. Der Ausdruck, der dort steht wird nur darauf untersucht, ob er den Wahrheitswerten True oder False entspricht[1].

... dann …

Auf ein if mit einer Bedingung folgt in jedem Fall mindestens der Programmblock, der ausgeführt werden soll, wenn die Bedingung erfüllt ist.
Beachtet bitte, dass ein Programmblock mit einem Doppelpunkt eingeleitet wird und in einer gemeinsamen Einrückungstiefe stehen muss.
Ist die Bedingung nicht erfüllt, dann wird der Programmblock nicht abgearbeitet. In einigen Fällen von Verzweigungen arbeitet das Programm danach also einfach die nachfolgenden Anweisungen weiter ab.

… sonst …

In vielen anderen Anwendungsfällen von Verzweigungen muss aber ein alternativer Programmblock ausgeführt werden, der sonst – Fall. Er wird daher mit dem englischen Wort else eingeleitet und wird genau dann ausgeführt, wenn die Bedingung falsch war.

Mehrfachverzweigungen

Es gibt auch noch eine besondere Möglichkeit Mehrfachverzweigungen durchzuführen, indem zwischen den if – Block und den else – Block ein Block [oder mehrere Blöcke] eingefügt wird, die mit einem elif beginnen, einer Zusammenfassung der Begriffsfolge else if, auf die dann natürlich zunächst noch eine neue Bedingung folgen muss.

Beispiel:

```
def testeZahl(zahl):
    if (zahl<0):
        print( str(zahl)+" ist negativ" )
    elif (zahl>0):
        print( str(zahl)+" ist positiv" )
    else:
        print( str(zahl)+" ist 0" )

testeZahl(-22)                  → -22 ist negativ
testeZahl(2.2)                  → 2.2 ist positiv
testeZahl(22-22)                → 0 ist 0
```

1 Was diese Entsprechung genau bedeutet bitte ggf. in der Hilfe nachlesen.

Schleifen sind Wiederholungen

Eine der grundlegenden Aufgaben bei der Programmierung ist die gewollte mehrfache Wiederholung von Aufgaben. Man bezeichnet das in der Informatik als Schleife, weil das Programm nach der jeder Bearbeitung des Programmblocks (=Schleifenkörper) an den Anfang der Schleife (=Schleifenkopf) zurückkehrt.
Da Endlosschleifen zum Hängenbleiben des Programms führen, muss sich bei der Bearbeitung des Schleifenkörpers etwas verändern (zumindest verändern können).

Kopfgesteuertes while

Daher finden wir in einer Variante zum Programmieren von Schleifen in Python, der while – Schleife, auch eine Bedingung. Die Bedingung steht am Kopf, sie wird also geprüft, bevor die Abarbeitung des Programmblocks beginnt. Wenn sie schon beim erstenmal nicht erfüllt ist, wird der Programmblock gar nicht bearbeitet.
Sonst wird der Programmblock bearbeitet, dann erneut geprüft usw. Irgendwann sollte sich dann der Fall ergeben, dass die Bedingung nicht mehr wahr ist, der Programmblock wird dann übersprungen und die nachfolgenden Anweisungen werden ausgeführt.

Beispiel

Im Beispiel sollen alle Quadratzahlen von Zahlen zwischen 11 und 20 (ohne die 20) ausgegeben werden:

```
def quadrate(von, bis):
    i=von
    while i<bis:
        print( i*i )
        i=i+1

quadrate(11,20)
```

Listenorientiertes for

Während man while – Schleifen im Prinzip immer einsetzen kann, verwendet man for – Schleifen nur dann, wenn schon vor dem Einstieg in die Schleife klar ist, wie oft sie ausgeführt werden soll. Das o.a. Beispiel ist ein solcher Fall.
Bei Python gibt es dabei eine Besonderheit: Es wird keine veränderbare Schleifenvariable verwendet, sondern die möglichen Werte der Schleifenvariablen müssen schon vor der Berabeitung bestimmt werden [In der neuen Python-Version geändert]. Dieselbe Aufgabe von oben wird also mit for so realisiert:

Beispiel

```
def quadrate2(von, bis):
    for i in range(von, bis):
        print( i*i )

quadrate2(11,20)
```

Hier wird also mit range(von, bis) erst eine Liste der Zahlen von 11 bis 19 erstellt, auf die dann die for – Schleife nacheinander zugreift.