Informatikabitur – Softwareentwicklung: Python

Ergänzung zum Tafelwerk

Datentypen

Python kennt zwei Arten von Datentypen: **primitive Typen** und **Objekttypen.**

- Der Wert eines primitiven Typs wird direkt gespeichert. Bei einer Zuweisung an eine andere Variable wird der Wert kopiert.
- In Objekttypen wird nicht der Wert, sondern eine Referenz auf das Objekt gespeichert. Bei einer Zuweisung an eine andere Variable wird nur die Referenz kopiert, nicht das Objekt. Objekttypen werden mit der Referenz **None** initialisiert.

Übersicht über Datentypen

Datentyp	Art	Bezeichner	Größe	Wertebereich
Ganze Zahl	primitiv	`int`	32 Bit wird automatisch angepasst	-2^{31} ... $2^{31}-1$
Gleitkommazahl	primitiv	`float`	64 Bit mit – Vorzeichen: 1 Bit – Mantisse: 52 Bit – Exponent: 11 Bit wird automatisch angepasst	
Wahrheitswert	primitiv	`bool`	1 Bit	`True`, `False`
Zeichenkette	Referenz	`str`	wird automatisch angepasst	
Liste	Referenz	`list`	wird automatisch angepasst	

Ganze Zahlen lassen sich in Stellenwertsystemen angeben. Standard ist das Dezimalsystem.

Stellenwertsystem	Basis	Präfix	Beispiel
dezimal	10		`a = 26`
binär	2	`0b`	`a = 0b00011010`
hexadezimal	16	`0x`	`a = 0x1A`

Datentypbestimmung und -umwandlungen

Aufgabe	Python-Funktion
Bestimmung des Typs der Variablen `var`	`type(var)`
Umwandlung der Variablen `var` in eine ganze Zahl	`int(var)`
Umwandlung der Variablen `var` in eine Gleitkommazahl	`float(var)`
Umwandlung der Variablen `var` in einen Wahrheitswert	`bool(var)`
Umwandlung der Variablen `var` in eine Zeichenkette	`str(var)`
Umwandlung der Variablen `var` in eine Liste	`list(var)`

Objekttyp Zeichenkette

Eine Zeichenkette (String) ist ein Objekttyp, der eine Folge von Zeichen repräsentiert. Zeichenketten sind in Anführungszeichen oder Hochkommas zu setzen.

Beispiel: `str1 = "Hallo Welt"`

Operation/Funktionen	Beschreibung
`len(str)`	Gibt die Länge von `str` zurück
`str[index]`	Gibt das Zeichen an der Stelle `index` zurück
`str[m:n]`	Gibt eine Teilzeichenkette von `str` mit Index `m` bis `n-1` zurück
`str1 + str2`	Gibt die durch Verkettung entstandene Zeichenkette zurück
`str1 == str2`	Gibt `True` zurück, falls `str2` identisch zu `str1`, sonst `False`
`str1 in str`	Gibt `True` zurück, falls `str1` in `str` enthalten ist, sonst `False`

Methoden	Beschreibung
`find(str)`	Gibt die Stelle zurück, an dem `str` erstmals auftritt, sonst `-1`
`startswith(str)`	Gibt `True` zurück, falls Zeichenkette mit `str` beginnt, sonst `False`
`endswith(str)`	Gibt `True` zurück, falls Zeichenkette mit `str` endet, sonst `False`
`replace(str1,str2)`	Gibt eine Zeichenkette zurück, in der jedes Vorkommen von `str1` durch `str2` ersetzt wurde
`split(str)`	Gibt eine Liste zurück, die die Zeichenkette am Zeichen `str` teilt
`lower()`	Gibt die Zeichenkette in Kleinbuchstaben zurück
`upper()`	Gibt die Zeichenkette in Großbuchstaben zurück
`islower()`	Gibt `True` zurück, falls Zeichenkette nur aus Kleinbuchstaben besteht, sonst `False`
`isupper()`	Gibt `True` zurück, falls Zeichenkette nur aus Großbuchstaben besteht, sonst `False`

Objekttyp Liste

Die Liste ist eine dynamische Datenstruktur zur Speicherung einer beliebigen Menge von Objekten. Listen werden durch eckige Klammern definiert. Die Elemente sind durch Komma voneinander zu trennen.

Beispiele:

```
list1 = [1, 4, 3, 4, 2, 6]
list2 = ["Mutter", mutter, 165.3]
```

Operationen, Funktionen und Methoden (Auszug)

Operation/Funktion	Beschreibung
`len(list)`	Gibt die Länge von `list` zurück
`list[index] = wert`	Ersetzt das Element an `index` von `list` durch `wert`
`list[index]`	Gibt das Element an `index` von `list` zurück
`list[m:n]`	Gibt eine Teilliste von `list` mit Index `m` bis `n-1` zurück
`list1 + list2`	Gibt die durch Verkettung entstandene Liste zurück
`list1 == list2`	Gibt `True` zurück, falls `list2` identisch zu `list1`, sonst `False`
`var in list`	Gibt `True` zurück, falls `var` in `list` enthalten ist, sonst `False`

Listenmethode	Beschreibung
`append(wert)`	Fügt `wert` zur Liste hinzu
`clear()`	Leert die Liste
`index(wert)`	Gibt den Index des ersten Auftretens von `wert` zurück
`pop(index)`	Entfernt das Element an der Stelle `index`
`remove(wert)`	Entfernt das erste Auftreten von `wert` in der Liste
`insert(index, wert)`	Fügt `wert` an Stelle `index` ein
`sort()`	Sortiert die Liste
`reverse()`	Kehrt die Reihenfolge der Liste um
`copy()`	Gibt eine Kopie der Liste als neues Objekt zurück

Operatoren und Boolesche Ausdrücke

Arithmetische Operationen

Operator	Beschreibung
`=`	Zuweisung
`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division
`**`	Potenzieren
`//`	ganzzahliger Teil bei Division zweier ganzer Zahlen
`%`	ganzzahliger Rest bei Division zweier ganzer Zahlen (Modulo)
`+=`	Erhöhung um
`-=`	Verminderung um

Boolesche Ausdrücke

Boolesche Ausdrücke liefern entweder den Wert **`True`** oder den Wert **`False`**.

Es gibt Vergleichsoperatoren und logische Operatoren.

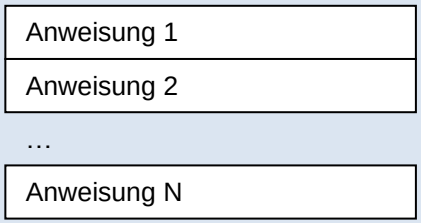
Operator	Beschreibung
`==`	identisch
`!=`	ungleich
`<`	kleiner als
`>`	größer als
`<=`	kleiner gleich
`>=`	größer gleich
`and`	logisch UND
`or`	logisch ODER
`not`	logisch NICHT

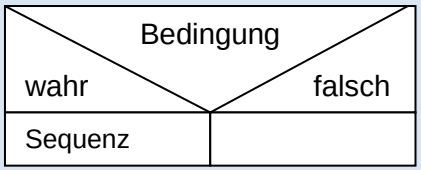
Informatikabitur – Softwareentwicklung

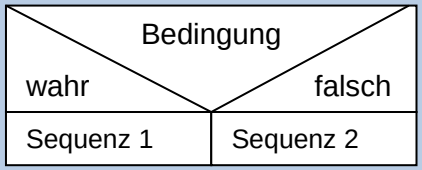
Ergänzung zum Tafelwerk

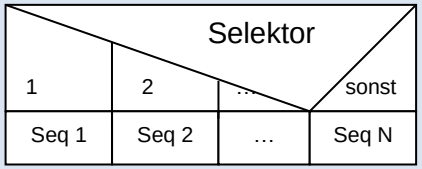
Algorithmische Grundstrukturen

Name	Verbale Formulierung	Struktogramm	Python-Implementation
Sequenz/Verbundanweisung			

Folge	Anweisung 1
	Anweisung 2
	…
	Anweisung N
			`Anweisung 1`
			`Anweisung 2`
			`...`
			`Anweisung N`

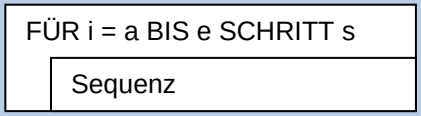
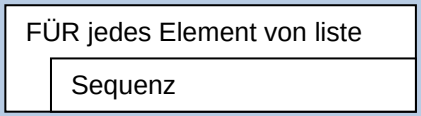
Selektion/Alternative/Auswahl			
Einseitige Auswahl	WENN Bedingung DANN Sequenz		`if Bedingung:` `  Sequenz`

Zweiseitige Auswahl	WENN Bedingung
	DANN Sequenz 1
	SONST Sequenz 2
			`if Bedingung:`
			`  Sequenz 1`
			`else:`
			`  Sequenz 2`

Mehrseitige Auswahl	WENN Selektor
	FALL 1: Sequenz 1
	FALL 2: Sequenz 2
	…
	SONST: Sequenz N
			`if Bedingung:`
			`  Sequenz 1`
			`elif Bedingung:`
			`  Sequenz 2`
			`  …`
			`else:`
			`  Sequenz N`

Informatikabitur – Softwareentwicklung

Ergänzung zum Tafelwerk

Name	Verbale Formulierung	Struktogramm	Python-Implementation
Iteration/Repetition/Wiederholung/Schleife			
Kopfgesteuerte Schleife	SOLANGE Bedingung FÜHRE AUS Sequenz		`while Bedingung:` `  Sequenz`
Zählschleife	FÜR i = Start a BIS Ende e mit SCHRITT s FÜHRE AUS Sequenz		`for i in range(a, e+1, s):` `  Sequenz`
Zählschleife	WIEDERHOLE n-mal FÜHRE AUS Sequenz		`for i in range(n):` `  Sequenz`
Durchlauf durch Liste	FÜR JEDES element DER liste FÜHRE AUS Sequenz		`for element in liste:` `  Sequenz`