

Informatik I: Einführung in die Programmierung

4. Funktionen: Aufrufe und Definitionen

Albert-Ludwigs-Universität Freiburg



Prof. Dr. Peter Thiemann

26. Oktober 2022

1 Funktionsaufrufe



- Syntax
- Standardfunktionen
- Exkurs: Zeichenkodierung und Unicode

Funktions- Aufrufe

Syntax

Standardfunktionen

Exkurs:
Zeichenkodierung
und Unicode

Mathematische Funktionen

Funktions- Definition

Namens- raum

Rückgabe- werte

- Funktionen sind **Abbildungen** von einem Definitionsbereich in einen Bildbereich.
- Eine Funktion erwartet **Argumente** aus dem Definitionsbereich und gibt einen **Funktionswert** (oder *Rückgabewert*) aus dem Bildbereich zurück.
- Eine Funktion kann **Effekte** haben, z.B.:
 - eine Ausgabe erzeugen,
 - eine Eingabe lesen,
 - uvam
- Viele Standardfunktionen sind in Python vordefiniert.

Funktions-
Aufrufe

Syntax

Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

Standardfunktionen: Typ-Konversion



Die Funktionen `int`, `float`, `complex` und `str` können „passende“ Werte in den jeweiligen Typ konvertieren.

Python-Interpreter

```
>>> int(-2.6) # Konversion nach int durch Abschneiden.
-2
>>> int('vier')
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() ...
>>> complex('42')
(42+0j)
>>> float(4)
4.0
>>> str(42)
'42'
```

Funktions-
Aufrufe

Syntax

Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Bekannt: `print` gibt Werte aus.
- Die Funktion `input` gibt einen Wert aus (wie `print`) und liest dann einen **String** ein.

Python-Interpreter

```
>>> input("Gib mir einen Keks: ")
Gib mir einen Keks: Keks
'Keks'
>>> name = input("Wie heißt du? ")
Wie heißt du? Oskar
>>> print("Hallo,", name + "!")
Hallo, Oskar!
```

Funktions-
Aufrufe

Syntax

Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

Standardfunktionen kombiniert: Eingabe von Zahlen



Da `input` nur Strings einliest, muss die Eingabe konvertiert werden!

Python-Interpreter

```
>>> CM_PER_INCH = 2.54
>>> länge = input("Länge in cm: ")
Länge in cm: 195
>>> länge # ein String
'195'
>>> länge_cm = float(länge)
>>> länge_inches = länge_cm / CM_PER_INCH
>>> print(länge + "cm", "=", str(länge_inches) + "in")
195cm = 76.77165354330708in
```

Funktions-
Aufrufe

Syntax

Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- `abs` liefert den Absolutwert (auch bei `complex`)
- `round` rundet.

```
>>> abs(-2)
2
>>> abs(1+1j)
1.4142135623730951
>>> round(2.500001)
3
```

Funktions-
Aufrufe

Syntax

Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

Standardfunktionen: Zeichenkonversion



- Die Funktion `chr` konvertiert ein `int` in ein **Unicode-Zeichen**.
- Die Funktion `ord` konvertiert in die umgekehrte Richtung.
- In Python werden Zeichen durch einbuchstabile Strings dargestellt.

```
>>> chr(42)
'*'
>>> chr(255)
'ÿ'
>>> ord('*')
42
>>> ord('**')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ord() expected a character, but string of length 2 found
```

Funktions-
Aufrufe

Syntax
Standardfunktion-
en
Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Computer können **Berechnungen** durchführen.
- Seit langem werden mit dem Computer auch **Texte verarbeitet**.
- Wie werden Texte im Computer dargestellt?
- Jedes Zeichen wird durch eine Zahl kodiert. Texte sind Sequenzen von solchen Codezahlen.
- Damit wird auch die **Textverarbeitung** zu einer Berechnung.

Funktions-
Aufrufe

Syntax
Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Einer der ersten Zeichenkodes war **ASCII** (American Standard Code for Information Interchange) – entwickelt für Fernschreiber und Lochstreifen.
- Benötigt 7 Bits und enthält alle druckbaren Zeichen der englischen Sprache sowie nicht-druckbare Steuerzeichen (z.B. Zeilenwechsel).

USASCII code chart

		Column							
		0	1	2	3	4	5	6	7
Row	0	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1
1	0	NUL	DLE	SP	0	@	P	\	p
2	1	SOH	DC1	!	1	A	Q	o	q
3	2	STX	DC2	"	2	B	R	b	r
4	3	ETX	DC3	#	3	C	S	c	s
5	4	EOT	DC4	\$	4	D	T	d	t
6	5	ENQ	NAK	%	5	E	U	e	u
7	6	ACK	SYN	&	6	F	V	f	v
8	7	BEL	ETB	'	7	G	W	g	w
9	8	BS	CAN	(8	H	X	h	x
10	9	HT	EM)	9	I	Y	i	y
11	10	LF	SUB	*	:	J	Z	j	z
12	11	VT	ESC	+	;	K	[k	{
13	12	FF	FS	,	<	L	\	l	
14	13	CR	GS	-	=	M]	m	}
15	14	SO	RS	.	>	N	^	n	~
16	15	SI	US	/	?	O	_	o	DEL

Funktions-
Aufrufe

Syntax
Standardfunktio-
nen
Exkurs:
Zeichenkodierung
und Unicode

Mathematische
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- In anderen Sprachen wurden **zusätzliche Zeichen** benötigt.
- Da praktisch alle Rechner **8-Bit-Bytes** als kleinste Speichereinheit nutzten, standen die Codes mit gesetztem höchstwertigen Bit (128–255) für Erweiterungen zur Verfügung.
- Diverse Erweiterungen, z.B. **ISO-Latin-1** (mit Umlauten, seit 1987).
- Auf dem IBM-PC gab es andere Erweiterungen, **Windows-1252**.
- Sprachen, die nicht auf dem lateinischen Alphabet basieren, haben große Probleme, **ISO-2022-JP**: `日本語を分かりますか`

Funktions-
Aufrufe

Syntax

Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Um für alle Sprachräume eine einheitliche Kodierung für Zeichen zu haben, wurde **Unicode** entwickelt (Version 1.0 im Jahr 1991).
- Seit September 2022 (Version 15.0) unterstützt Unicode **161 Schriften** mit **149,186 Codepoints**, darunter **3,664 Emojis**.
- Organisiert in 17 Ebenen mit jeweils 2^{16} Codepoints (manche allerdings ungenutzt)
- Die ersten 128 Codepoints stimmen mit ASCII überein, die ersten 256 mit ISO-Latin-1.
- Zum Thema Emojis gibt es ein eigenes **Subkomitee** ...
- Im Mai 2019 gab es außerplanmäßig Version 12.1, bei der ein einziges Zeichen hinzugefügt wurde?!

Funktions-
Aufrufe

Syntax
Standardfunktio-
nen

Exkurs:
Zeichenkodierung
und Unicode

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Ein Unicode-Zeichen kann durch eine 32-Bit-Zahl dargestellt werden (UTF-32 oder UCS-4).
- Meist wird nur die Ebene 0 benötigt. Daher ist es effizienter, die Kodierung UTF-16 einzusetzen, bei der Zeichen der Ebene 0 direkt als 16-Bit-Zahl kodiert werden. Zeichen aus anderen Ebenen benötigen 32 Bit.
- Im WWW wird meist UTF-8 eingesetzt, eine Kodierung mit variabler Länge:

Unicode	UTF-8 binär
0–127	0xxxxxxx
128–2047	110xxxxx 10xxxxxx
2048–65535	1110xxxx 10xxxxxx 10xxxxxx
65536–1114111	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

- Wie kommen die **komischen Zeichen** auf Webseiten / in Emails zustande?
- Oft sind ISO-Latin-1/UTF-8 Verwechslungen der Grund!

2 Mathematische Funktionen



- `math`-Modul
- `Direktimport`

Funktions-
Aufrufe

**Mathemati-
sche
Funktionen**

`math`-Modul
`Direktimport`

Funktions-
Definition

Namens-
raum

Rückgabe-
werte



- Funktionen wie `sin` stehen nicht direkt zur Verfügung. Sie müssen durch **Importieren** des **Mathematik-Moduls** `math` bekannt gemacht werden.
- Werte aus dem Modul können durch Voranstellen von `math.` genutzt werden (**Punkt Schreibweise**):

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sin(1/4*math.pi)
0.7071067811865475
>>> math.sin(math.pi)
1.2246467991473532e-16
>>> math.exp(math.log(2))
2.0
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

`math`-Modul
Direktimport

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Die Punktschreibweise verhindert **Namenskollisionen**, ist aber umständlich.
- Direkter Import eines Bezeichners:
`from module import name`
- Direkter Import aller Bezeichner eines Moduls:
`from module import *`

```
>>> pi = 3.14
>>> pi
3.14
>>> from math import *
>>> cos(pi)
-1.0
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

math-Modul
Direktimport

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

3 Funktionsdefinitionen



- Definition
- Einrückungen
- Aufruf
- Argumente, Parameter, Rückgabewerte

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

**Funktions-
Definition**

Definition
Einrückungen
Aufruf
Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

- Ein Python-Programm kann selbst neue Funktionen definieren.
- Geht interaktiv, aber am besten in einer Datei (Skript)
- Eine Funktionsdefinition beginnt mit dem **Schlüsselwort def**, danach kommt der **Funktionsname** gefolgt von der **Parameterliste** und dann ein Doppelpunkt.
- Dann folgt der **Funktionsrumpf** als Block von gleich weit eingerückten Anweisungen, z.B. Zuweisungen oder Funktionsaufrufe.

lumberjack.py

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay")  
    print("I sleep all night and I work all day")
```

Funktions-
aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Definition

Einrückungen

Aufruf

Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

- **Einrückungen** am Zeilenanfang sind bedeutungstragend. Vgl FORTRAN:



- Gleiche Einrückung = zusammengehöriger **Block** von Anweisungen
 - In den meisten anderen Programmiersprachen durch Klammerung { } oder klammernde Schlüsselwörter.
 - Wie viele Leerzeichen sollen verwendet werden?
- **PEP8**: 4 Leerzeichen pro Ebene der Einrückung (*keine Tabs* verwenden!)

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Definition

Einrückungen

Aufruf

Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

Intermezzo: play the lumberjack



Python-Interpreter

```
>>> import webbrowser
>>> webbrowser.open('https://www.youtube.com/watch?v=89LfQU1cNFk')
True
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Definition

Eintrückungen

Aufruf

Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

- Funktionsnamen sind Variablennamen
- Funktionen haben einen speziellen Typ
- Selbstdefinierte Funktionen werden wie Standardfunktionen aufgerufen

Python-Interpreter

```
>>> import lumberjack
>>> lumberjack.print_lyrics()
I'm a lumberjack, and I'm okay
I sleep all night and I work all day
>>> print_lyrics = 42
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Definition
Einrückungen

Aufruf
Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

Eine kleine Erweiterung...

lumberjack.py

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay")  
    print("I sleep all night and I work all day")  
  
def repeat_lyrics():  
    print_lyrics()  
    print_lyrics()
```

Was passiert beim Ausführen von `repeat_lyrics()`?

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Definition
Einrückungen

Aufruf
Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

- Selbst definierte Funktionen benötigen oft *Argumente*.
- Die Definition verwendet *formale Parameter* (Variablennamen), an die beim Aufruf die *Argumentwerte* zugewiesen werden.
- `return` beendet die Ausführung der Funktion.
- Der Wert des Ausdrucks nach `return` wird zum **Wert des Funktionsaufrufs**.

convert.py

```
CM_PER_INCH = 2.54
def cm_to_inches (cm):
    return cm / CM_PER_INCH
```

Python-Interpreter

```
>>> import convert
>>> convert.cm_to_inches(195)
76.77165354330708
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Definition
Einrückungen
Aufruf

Argumente,
Parameter,
Rückgabewerte

Namens-
raum

Rückgabe-
werte

4 Namensraum



- Lokale Variablen und Parameter
- Kellertabelle
- Traceback
- Globale Variable

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

**Namens-
raum**

Lokale Variablen
und Parameter
Kellertabelle
Traceback
Globale Variable

Rückgabe-
werte

- Parameter (part1, part2) sind nur innerhalb der Funktion **sichtbar**.
- Lokal (durch Zuweisung) eingeführte Variablen (cat) ebenfalls.
- Globale Variable sind weiterhin lesbar.

cat.py

```
def cat_twice (part1, part2):  
    cat = part1 + part2  
    print (cat)  
    print (cat)
```

Python-Interpreter

```
>>> import cat  
>>> line1 = 'Bing tiddle '  
>>> line2 = 'tiddle bang.'  
>>> cat.cat_twice(line1, line2)  
Bing tiddle tiddle bang.  
Bing tiddle tiddle bang.  
>>> cat  
NameError: name 'cat' is not  
defined
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Lokale Variablen
und Parameter

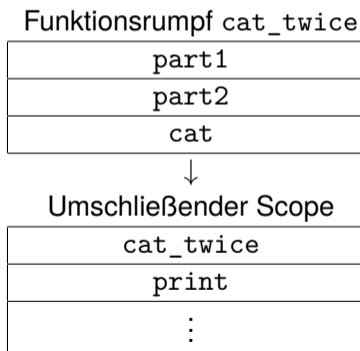
Kellertabelle

Traceback

Globale Variable

Rückgabe-
werte

- Der Rumpf einer Funktion bildet eine **Umgebung** (Scope).
- Die Umgebung **bindet** die lokal definierten Variablen der Funktion (inkl. Parameter).
- Alle weiteren Variablen sind **frei** im Funktionsrumpf und beziehen sich auf einen umschließenden Scope.
- Beim Aufruf der Funktion wird für den Scope ein **Kellerrahmen** (stack frame) für die Werte der Variablen angelegt.



Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Lokale Variablen
und Parameter

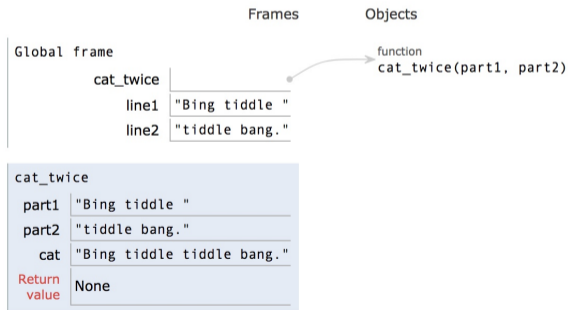
Kellertabelle

Traceback

Globale Variable

Rückgabe-
werte

- Die Variablenbelegungen innerhalb von Funktionsaufrufen (also die Kellerrahmen) können durch eine **Kellertabelle** visualisiert werden (hier hilft <http://pythontutor.com/>). Ende von cat_twice



Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Lokale Variablen
und Parameter

Kellertabelle

Traceback

Globale Variable

Rückgabe-
werte

- Tritt bei der Ausführung einer Funktion ein Fehler auf, z.B. Zugriff auf die nicht vorhandene Funktion `print_twice` in `cat_twice`, dann gibt es ein **Traceback** (entsprechend einer Kellertabelle):

Python-Interpreter

```
>>> def cat_twice(part1, part2):  
...     cat = part1 + part2  
...     print_twice(cat)  
...  
>>> cat_twice('foo ', 'bar!')  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<stdin>", line 3, in cat_twice  
NameError: name 'print_twice' is not defined
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Lokale Variablen
und Parameter

Kellertabelle

Traceback
Globale Variable

Rückgabe-
werte

- Funktionen sollen vorrangig **lokale Variable** und **Parameter** nutzen.
- Funktionen können globale Variablen **lesen**, falls diese nicht durch lokale Variable gleichen Namens **verdeckt** werden (shadowing).

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Lokale Variablen
und Parameter

Kellertabelle

Traceback

Globale Variable

Rückgabe-
werte

Globale Variable (Beispiel)



globalvar.py

```
var = 666
def depp():
    return var
print("depp returns", depp())
def independent(var):
    return var
print("independent returns", independent(42))
def ignorant():
    var = 333
    return var
print("ignorant returns", ignorant(), " var=", var)
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Lokale Variablen
und Parameter

Kellertabelle

Traceback

Globale Variable

Rückgabe-
werte

5 Rückgabewerte



Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

**Rückgabe-
werte**

- *Alle* Funktionsaufrufe geben einen Wert zurück.
- Funktionen wie `print` geben einen speziellen Wert `None` zurück, der **nicht angezeigt** wird.

```
>>> result = print('Bruce')
Bruce
>>> result
>>> print(result)
None
```

- `None` ist der einzige Wert des Typs `NoneType`.

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- Das Schlüsselwort `return` erlaubt die Definition des Rückgabewerts.

```
>>> def sum3(a, b, c):  
...     return a + b + c  
...  
>>> sum3(1, 2, 3)  
6
```

- Funktionen ohne `return` (wie `cat_twice`) geben `None` zurück.

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

return \neq print

- print(.) definiert keinen Rückgabewert!

```
>>> def printsum3(a, b, c):  
...     print(a + b + c)  
...
```

```
>>> sum3(1, 2, 3)
```

```
6
```

```
>>> sum3(1, 2, 3) + 4
```

```
10
```

```
>>> printsum3(1, 2, 3) + 4
```

```
6
```

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

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte

- **Funktionen** sind benannte Programmstücke.
- Standardfunktionen oder selbst definiert
- Beim Aufruf einer Funktion müssen **Argumente** angegeben werden, die die formalen **Parameter** in der Definition mit Werten belegen.
- Funktionen geben einen **Funktionswert** zurück, der mit `return` festgelegt wird (sonst `None`).
- Jede Funktionsdefinition hat eine eigene **Umgebung** für die Parameter und **lokalen** Variablen (durch Zuweisung eingeführt).
- **Globale** Variablen können gelesen werden, falls sie nicht durch einen Parameter oder eine lokale Variable verdeckt werden.
- pythontutor.com visualisiert die Programmausführung mit Hilfe von Kellertabellen.

Funktions-
Aufrufe

Mathemati-
sche
Funktionen

Funktions-
Definition

Namens-
raum

Rückgabe-
werte