



Serbest Hammade / Resh

Do, 04. Nov 2010



Überblick

Entstehung

Konzept

Einsatzmöglichkeiten

Grundlagen

Arbeitsabläufe

Aufbau eines Python-Scripts

Programmierung

Basisdatentypen

Kontrollstrukturen

Ein- & Ausgaben

Funktionen

Modularisierung

Objektorientierung

Weitere Spracheigenschaften



Überblick

Entstehung

Konzept

Einsatzmöglichkeiten

Grundlagen

Arbeitsabläufe

Aufbau eines Python-Scripts

Programmierung

Basisdatentypen

Kontrollstrukturen

Ein- & Ausgaben

Funktionen

Modularisierung

Objektorientierung

Weitere Spracheigenschaften



Überblick

Entstehung

Konzept

Einsatzmöglichkeiten

Grundlagen

Arbeitsabläufe

Aufbau eines Python-Scripts

Programmierung

Basisdatentypen

Kontrollstrukturen

Ein- & Ausgaben

Funktionen

Modularisierung

Objektorientierung

Weitere Spracheigenschaften



Entstehung

CPython

- **Entwickler: Python Software Foundation (2001)**
- Aktuelle Version: 3.1.2 / 2.7
- Lizenz: PSF Lizenz, GPL kompatibel seit 2001
- weitere Implementierungen: Jython, IronPython, PyPy



Entstehung

CPython

- Entwickler: Python Software Foundation (2001)
- Aktuelle Version: 3.1.2 / 2.7
- Lizenz: PSF Lizenz, GPL kompatibel seit 2001
- weitere Implementierungen: Jython, IronPython, PyPy



Entstehung

CPython

- Entwickler: Python Software Foundation (2001)
- Aktuelle Version: 3.1.2 / 2.7
- Lizenz: PSF Lizenz, GPL kompatibel seit 2001
- weitere Implementierungen: Jython, IronPython, PyPy

Entstehung

CPython

- Entwickler: Python Software Foundation (2001)
- Aktuelle Version: 3.1.2 / 2.7
- Lizenz: PSF Lizenz, GPL kompatibel seit 2001
- weitere Implementierungen: Jython, IronPython, PyPy

Entstehung

CPython

- Entwickler: Python Software Foundation (2001)
- Aktuelle Version: 3.1.2 / 2.7
- Lizenz: PSF Lizenz, GPL kompatibel seit 2001
- weitere Implementierungen: Jython, IronPython, PyPy

Konzept

- **objektorientierte Programmiersprache**

- Programmierparadigmen

- objektorientierte
- aspektorientierte
- funktionale

- Typisierung

- stark
- dynamisch
- Duck-Typing

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

- James Whitcomb Riley

Konzept

- objektorientierte Programmiersprache
- Programmierparadigmen
 - objektorientierte
 - aspektorientierte
 - funktionale

- Typisierung

- stark
- dynamisch
- Duck-Typing

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

- James Whitcomb Riley

Konzept

- objektorientierte Programmiersprache
- Programmierparadigmen
 - objektorientierte
 - aspektorientierte
 - funktionale
- Typisierung
 - stark
 - dynamisch
 - Duck-Typing

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

- James Whitcomb Riley



Konzept

Vor- und Nachteile

- Pro:
 - Open Source
 - klare, schnell erlernbare Sprachsyntax
 - sehr umfangreiche Standard-Bibliothek
- Kontra:
 - bedingt geeignet für performancelastige Probleme
 - Programmierung von Treibern
 - geringerer Bekanntheitsgrad als Java, C(++) oder PHP

Konzept

Vor- und Nachteile

- Pro:
 - Open Source
 - klare, schnell erlernbare Sprachsyntax
 - sehr umfangreiche Standard-Bibliothek
- Kontra:
 - bedingt geeignet für performancelastige Probleme
 - Programmierung von Treibern
 - geringerer Bekanntheitsgrad als Java, C(++) oder PHP



Einsatzmöglichkeiten

- kleine bis große Applikationen
- serverseitige Programmiersprache
- Scriptsprache für C++/Java/...
- Drittanbieterbibliotheken ermöglichen GUIs, Datenbank Anwendungen, etc.



Arbeitsabläufe

- Quellcode schreiben & abspeichern
- Python Interpreter auf Quellcode ausführen



Aufbau eines Python-Scripts

```
1 #!/usr/bin/env python  
2  
3 print("Hello World")
```



Basisdatentypen

```
1 #!/usr/bin/env python
2 # Ganze Zahlen
3 a = 21 * 2
4 print(a)
5
6 # Gleitkommazahlen
7 b = 42.42 - 0.42
8 print(b)
9
10 # Zeichenketten
11 c = "Hallo" + " " + "42"
12 print(c)
13
14 # Logische Ausdruecke
15 print( a == b and not(a != b) and a >= b and a <= b and not(a < b) and not(b > a) and
    not(False == True) )
16
17 """ Dieses ist ein mehrzeiliger Kommentar
18 Ausgabe:
19 42
20 42.0
21 Hallo 42
22 True
23 """
```



Basisdatentypen

Sequenzielle Datentypen

```
1 #!/usr/bin/env python
2
3 a = "bird"
4 text = "When I see a bird, "
5 duck = "I call that bird a duck."
6 if a in text:
7     print("'" + a + "' ist im Text '" + text + "' enthalten")
8 if duck not in text:
9     print("'" + duck + "' ist im Text '" + text + "' NICHT enthalten")
10 text += duck
11
12 if duck in text:
13     print("'" + duck + "' ist im Text '" + text + "' enthalten")
14 print(text)
15 print(text[0])
16 print(text[0:4])
17 print(text[0:4:2])
18 print("Laenge:", len(text))
19 print("Min: '" + str(min(text)) + "'")
20 print("Max: '" + str(max(text)) + "'")
```



Basisdatentypen

Sequenzielle Datentypen

```
1 #!/usr/bin/env python
2
3 liste = [1, 4, [2, 'a']]
4 print(liste)
5
6 liste[0] = 42
7 liste[0:1] = [21,42]
8 del liste[0]
9
10 print(liste)
11 liste.append('c')
12 print(liste)
13 liste.extend(liste)
14 print(liste)
```

weitere Sequenzielle Datentypen:

- tuple (immutable list)



Basisdatentypen

weitere Datentypen

Mappings (dict):

```
1 #!/usr/bin/env python
2 woerterbuch = {"a" : "1", "c" : "3"}
3 print(woerterbuch["a"])
4 print(woerterbuch["c"])
```

Menge (set, frozenset)

```
1 #!/usr/bin/env python
2 s = set([0,1,2])
3 t = set([2,3,4])
4 print("s", s)
5 print("t", t)
6
7 print(s | t) # Vereinigungsmenge
8 print(s & t) # Schnittmenge
9 print(s - t) # Differenz
10 print(s ^ t) # symmetrische Differenz
11
12 s |= t # s ist nun die Vereinigungsmenge von s und t
```

Basisdatentypen

weitere Datentypen

Mappings (dict):

```
1 #!/usr/bin/env python
2 woerterbuch = {"a" : "1", "c" : "3"}
3 print(woerterbuch["a"])
4 print(woerterbuch["c"])
```

Menge (set, frozenset)

```
1 #!/usr/bin/env python
2 s = set([0,1,2])
3 t = set([2,3,4])
4 print("s", s)
5 print("t", t)
6
7 print(s | t) # Vereinigungsmenge
8 print(s & t) # Schnittmenge
9 print(s - t) # Differenz
10 print(s ^ t) # symmetrische Differenz
11
12 s |= t # s ist nun die Vereinigungsmenge von s und t
```



Kontrollstrukturen

Fallunterscheidungen

```
1 #! /usr/bin/env python
2 # klassische if-Anweisung
3 x = 3
4 a = ""
5 if x == 1:
6     a = "x hat den Wert 1"
7 elif x == 2:
8     a = "x hat den Wert 2"
9 else:
10    a = "Fehler: Der Wert von x ist weder 1 noch 2"
11 print(a)
12
13 # conditional expression
14
15 b = ("x hat den Wert 1" if x == 1 else "x hat den Wert 2" if x == 2 else "Fehler: Der
16     Wert von x ist weder 1 noch 2")
17 print(b)
```



Kontrollstrukturen

Schleifen

```
1  #!/usr/bin/env python
2  print("while-Schleife")
3  x = 0
4  while True:
5      x += 1
6      if x == 1:
7          continue
8      print(x)
9      if x == 3:
10         break
11
12 print("for-Schleife")
13 # range(stop)
14 # range(start, stop)
15 # range(start, stop, step)
16 for i in range(0, 45, 6):
17     if i == 42:
18         print(str(i) + ": you win!\n")
19     elif i == 0:
20         print(i,": you fail!")
21     else:
22         print(str(i) + ": you fail again!")
23
24 for c in "Hello UnFUG":
25     print(c)
```



Ein- & Ausgaben

```
1 #!/usr/bin/env python
2 # Benutzereingaben
3 pwlist = []
4 password = ""
5 while password != "blub":
6     password = input("Passwort: ")
7     pwlist.append(password)
8 print("ok")
9
10 # Dateien
11
12 fileobj = open("helloworld.py", "r")
13 for line in fileobj:
14     line = line.strip()
15     print(line)
16 fileobj.close()
17
18 fileobj = open("ausgabe.txt", "w")
19 for pw in pwlist:
20     fileobj.write(pw + "\n")
21 fileobj.close()
```



Funktionen

```
1 #!/usr/bin/env python
2
3 """
4
5 def <funktionsname> ([argumente[=standardwert]]):
6 <tab>befehl
7
8 """
9
10 def betrag(a, b = 42):
11     return a + b
12
13 print(betrag(42, 21))
14 print(betrag(42))
```



Modularisierung

- Bibliothek

```
1 #!/usr/bin/env python
2 import math
3 import math as mathematik
4 print(math.sin(math.pi))
5 print(mathematik.sin(mathematik.pi))
6
7 from math import *
8 print(sin(pi))
```

- Kapselung von Funktionen etc. in eine andere Datei

```
1 #!/usr/bin/env python
2 import helloworld
```

Objektorientierung

```
1 #!/usr/bin/env python
2 class Konto(object):
3
4     def __init__(self, inhaber, kontonummer, kontostand, max_tagesumsatz=1500):
5         self.Inhaber = inhaber
6         self.Kontonummer = kontonummer
7         self.Kontostand = kontostand
8         self.MaxTagesumsatz = max_tagesumsatz
9         self.UmsatzHeute = 0
10
11     def geldtransfer(self, ziel, betrag):
12         self.Kontostand -= betrag
13         self.UmsatzHeute += betrag
14         ziel.Kontostand += betrag
15         ziel.UmsatzHeute += betrag
16
17 k1 = Konto("Resh", 12345, 10.0)
18 k2 = Konto("Mutter", 120987, 1000.0)
19 print(k1.Inhaber, k1.Kontostand)
20 print(k2.Inhaber, k2.Kontostand)
21 k2.geldtransfer(k1, 100)
22 print(k1.Inhaber, k1.Kontostand)
23 print(k2.Inhaber, k2.Kontostand)
```



Weitere Spracheigenschaften

Exception

```
1 #!/usr/bin/env python
2 def getA(a):
3     l = [1,2,3]
4     try:
5         print(l[a])
6     except (IndexError, TypeError), e:
7         print("Fehlermeldung:", e.message)
8 print(getA(5))
```



Quellen

- python.org
- python.de