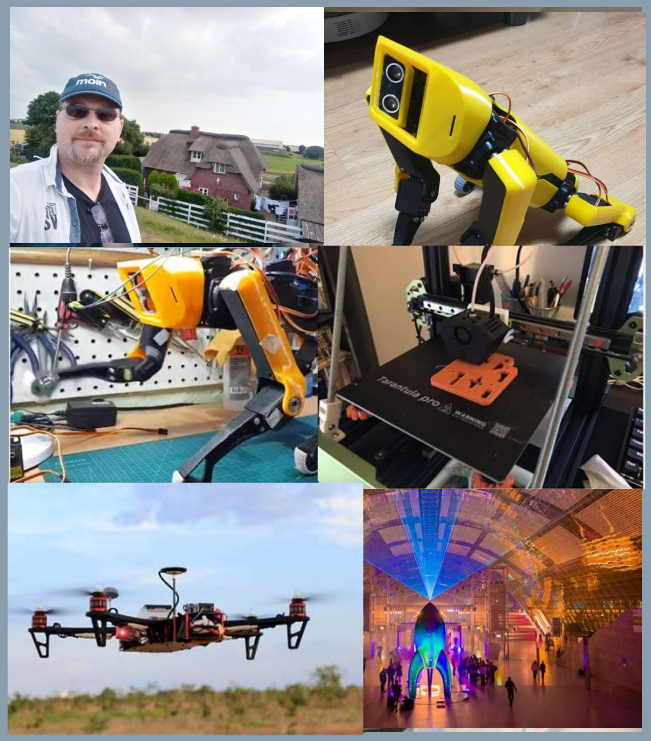
Cloud Command

Python Programmieren

Cloud Command

Zum Referenten:



Andreas Schmidt

- Jahrgang 1975, wohnhaft in Hamburg
- Fachinformatiker für Systemintegration
- Android-App Entwickler
- Java Entwickler
- Administrator für Heterogene Netzwerke
- Kommunikationselektroniker
- Über 20 Jahre im IT-Support und Consulting tätig
- ITIL
- Hardware-Entwicklung

Certified by EXIN
EXIN Ethical Hacking Foundation

Cloud Command

Variablen und Strings

Mengen managen über: set und frozenset

Über das Objekt set ist in Python das Konzept der Mathematik mit Mengen und Mengenlehre nachgebaut.

- In einem Set steckt eine ungeordnete Sammlung von Objekten, die nur 1-Mal vorkommen!
- Aufbau eines Sets über geschweifte Klammern:

- In einem Set steckt eine ungeordnete Sammlung von Objekten, die nur 1-Mal vorkommen!

```
set_a = { 1, 2, 3, 'A', 'B', 'C' }
```

Variablen und Strings

Mengen managen über: set und frozenset

Dies sieht man sehr schön, wenn man ein Tupel mit Dopplungen in ein set umwandelt:

```
werte_als_tupel = (1,1,1,3,5,3,4,5,3)
werte_als_set   = set(werte_als_tupel)
print(werte_als_set)
```

- Als Ergebnis erhalten wir dann ein Set mit jeweils nur einmalig vorkommenden Werten:

```
{1, 3, 4, 5}
```

Cloud Command

Variablen und Strings

Mengen managen über: set und frozenset

- Mengenlehre mit Set
- Das Besondere ist nun, dass über 2 Sets Mengenlehre mit „Schnittmenge“, „Vereinigungsmenge“, „A ohne B“ usw. durchgeführt werden kann.

```
set_a = { 1, 2, 3, 'A', 'B', 'C' }
set_b = { 2, 3, 'B', 'D' }
```

- Wollen wir nun die Schnittmenge (also was in beiden Mengen vorkommt) herausfiltern, läuft dies über das kaufmännische Und &:

```
set_a = { 1, 2, 3, 'A', 'B', 'C' }
set_b = { 2, 3, 'B', 'D' }
print( set_a & set_b )
```

- Als Ergebnis erhalten wir:

```
{2, 3, 'B'}
```

.

Cloud Command

Variablen und Strings

Mengen managen über: set und frozenset

- **Im folgenden Beispiel die üblichen Verdächtigen bei der Mengenlehre:**

```
set_a = { 1, 2, 3, 'A', 'B', 'C' }
set_b = { 2, 3, 'B', 'D' }
print("Set A:")
print(set_a)

print("Set B:")
print(set_b)

print()
print("Schnittmenge über &")
print( set_a & set_b )

print()
print("Vereinigungsmenge über |")
print( set_a | set_b )

print()
print("Differenzmenge über - ")
print( set_a - set_b )

print()
print("Symmmetrische Differnz (entweder-oder) über ^")
print( set_a ^ set_b )

print()
print("Obermenge von  > ")
print( set_a > set_b )
```

Variablen und Strings

Mengen managen über: set und frozenset

Und unsere Ergebnisse:

```
Set A:
{1, 2, 3, 'A', 'B', 'C'}

Set B:
{2, 3, 'D', 'B'}

Schnittmenge über &
{2, 3, 'B'}

Vereinigungsmenge über |
{1, 2, 3, 'A', 'B', 'C', 'D'}

Differenzmenge über -
{'C', 1, 'A'}

Symmetrische Differenz (entweder-oder) über ^
{1, 'A', 'C', 'D'}

Obermenge von >
False
```

Cloud Command

Variablen und Strings

Mengen managen über: set und frozenset

frozenset und Unterschied zu set

- set-Objekte sind veränderbar – diese werden eingefroren und somit unveränderbar über frozenset.
- Die Umwandlung kann man vorwärts wie rückwärts machen, sprich aus einem Set ein Frozenset und rückwärts.

- Ergebnis:

```
set_a = { 1, 2, 3, 'A', 'B', 'C' }
set_c = frozenset(set_a)
print(set_c)
print(type(set_c))
```

.

```
frozenset({'C', 1, 2, 3, 'B', 'A'})
<class 'frozenset'>
```

Cloud Command

Variablen und Strings

Mengen managen über: set und frozenset

Und locker aus der Hüfte wieder rückwärts:

```
set_a = { 1, 2, 3, 'A', 'B', 'C' }
set_c = frozenset(set_a)

print(set_c)
print(type(set_c))

set_c = set(set_c)
print(set_c)
print(type(set_c))
```

Variablen und Strings

Mengen managen über: set und frozenset

Und das Ergebnis:

```
frozenset({1, 2, 3, 'C', 'B', 'A'})
<class 'frozenset'>

{1, 2, 3, 'C', 'B', 'A'}
<class 'set'>
```

Cloud Command

Variablen und Strings

Mengen managen über: set und frozenset

Beispiel Anzahl Buchstaben in Text zählen mit Hilfe von set

- Anhand der Anweisung **set()** werten wir einen Text aus und zählen die Anzahl der Buchstaben.
- Dabei wird im ersten Code es Schritt für Schritt gemacht.
- Dasselbe kommt dann nochmals komprimiert.

Variablen und Strings

Mengen managen über: set und frozenset

```
inhalt = "anzahl"
# doppelte Buchstaben entfernen
buchstaben = set(inhalt)
print(buchstaben)
# zum Sortieren aus dem SET eine Liste machen
buchstabenliste = list(buchstaben)
print(buchstabenliste)
# sortieren
buchstabensortiert = sorted(buchstabenliste)
print(buchstabensortiert)
# der Reihen nach durchlaufen und Anzahl zählen
# die for-Schleife kommt in einem späteren Kapitel
for einzelbuchstabe in buchstabensortiert:
    print(einzelbuchstabe ,": Anzahl ", inhalt.count(einzelbuchstabe))
```

Variablen und Strings

Mengen managen über: set und frozenset

Als Ergebnis erhalten wir:

```
{'n', 'a', 'l', 'z', 'h'}
['n', 'a', 'l', 'z', 'h']
['a', 'h', 'l', 'n', 'z']
a : Anzahl 2
h : Anzahl 1
l : Anzahl 1
n : Anzahl 1
z : Anzahl 1
```

Variablen und Strings

Mengen managen über: set und frozenset

- **Und nun dasselbe in 3 Zeilen:**

```
inhalt = "Buchstaben zählen"
for einzelbuchstabe in sorted(list(set(inhalt))):
    print(einzelbuchstabe ,": Anzahl ",
inhalt.count(einzelbuchstabe))
```

Als Ergebnis erhalten wir:

```
: Anzahl 1
B : Anzahl 1
a : Anzahl 1
b : Anzahl 1
c : Anzahl 1
e : Anzahl 2
h : Anzahl 2
l : Anzahl 1
n : Anzahl 2
s : Anzahl 1
t : Anzahl 1
u : Anzahl 1
z : Anzahl 1
ä : Anzahl 1
```

Cloud Command

Variablen und Strings

Übersicht aller Methoden zu set

- Über die Anweisung dir(set) erhalten wir alle Methoden zu set

```
>>> dir(set)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__',
'__init_subclass__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__',
'__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__',
'__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference',
'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset',
'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update',
'union', 'update']
```

Cloud Command

Variablen und Strings
Übersicht aller Methoden zu set

- Im Folgenden die Übersicht aller Methoden von set:

Methode	Kurzbeschreibung
.add(neues Element)	fügt ein Element zu einem Set hinzu
.clear()	löschen aller Elemente von dem Set
.copy()	gibt eine Kopie des Sets zurück
.difference()	liefert ein neues Set mit der Rückgabe der Unterschiede der übergebenen Sets
.intersection()	liefert ein neues Set mit der Rückgabe der Unterschiede der übergebenen Sets
.intersection_update()	aktualisiert ein Set mit den Überschneidungen zwischen sich selber und dem übergebenen Set
.isdisjoint()	Überprüft, ob 2 Sets keinerlei Überschneidungen haben. Als Rückgabe wird „True“ oder „False“ geliefert
.issubset()	Überprüft, ob das Set im anderen Set enthalten ist. Rückgabe „True“ oder „False“
.issuperset()	Überprüft, ob jedes Element des Sets im anderen Set enthalten ist. Rückgabe „True“ oder „False“
.pop()	Liefert vom Set ein zufälliges Element zurück und löscht dieses aus dem Set
.remove(wegmitdiesem)	Löscht ein Element (das übergeben wird) aus einem Set
.symmetric_difference()	liefert ein neues Set zurück mit allen Werten beider Sets, die sich nicht überschneiden der übergebenen Sets
.symmetric_difference_update()	aktualisiert ein Set mit den Werten beider Sets, bei denen keine Überschneidungen zwischen sich selber und dem übergebenen Set vorliegt
.union()	liefert alle Werte der übergebenen Sets zurück
.update()	aktualisiert ein Set mit allen Werten von sich selber und den übergebenen Sets

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Eingaben vom Nutzer abfragen in Python

- Wir haben bei den vorherigen Kapiteln immer wieder Daten auf dem Bildschirm ausgegeben.
  - Allerdings werden in der Realität die Daten entweder aus einer Datenbank kommen oder durch einen Nutzer festgelegt, also eingegeben werden.
- Um eine Angabe vom Nutzer anzufordern, gibt es in Python den Befehl input()
  - Diesem Befehl können wir einen Text mitgeben, damit dem Nutzer klar ist, was er denn eingeben soll.
  - Die Eingabe selber wird in einer Variablen gespeichert, mit der wir dann weiterarbeiten können.

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Unser Befehl sieht dann so aus:

```
benutzereingabe = input("Bitte Zahl eingeben")
```

- Das Python-Programm pausiert nun, wenn es diese Programmzeile erreicht hat und wartet ab, bis der Nutzer eine Eingabe gemacht hat.
- Erst nach der Eingabe läuft das Programm dann weiter.
- Erinnere Dich an **Regel #1 : Traue keiner Nutzereingabe**
  - **Erweiter deinen Snitizer1.py!!!!!**

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Nun können wir die Variable, die im Beispiel „benutzereingabe“ genannt wurde, im weiteren Programmablauf nutzen.

```
benutzereingabe = input("Bitte Zahl eingeben")
print(benutzereingabe)
```

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Input liefert Strings

- Als Rückgabewert der input-Funktion erhalten wir eine String. Den Typ einer Variablen können wir uns über die Funktion type ausgeben lassen. Im folgenden Beispiel zum Testen:

```
benutzereingabe = input("Bitte Zahl eingeben")
print(type(benutzereingabe))
```

- Als Ergebnis erhalten wir:

```
class 'str'
```

- Wollen wir aber mit der vom Benutzer eingegebenen Zahl weiterrechnen, müssen wir den String erst in eine Zahl konvertieren.

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

- Wollen wir aber mit der vom Benutzer eingegebenen Zahl weiterrechnen, müssen wir den String erst in eine Zahl konvertieren.
- Hier kommt das Prinzip des „castens“ zum Einsatz.
  - Der Typ einer Variablen umgewandelt.
- Wenn man den Typ einer Variablen umwandelt, spricht man „casting“.
  - Das Wort erinnert nicht zu Unrecht an das Besetzen von Filmrollen (also die Rollenverteilung).
  - Und genau das machen wir mit dem Typ.
  - Wir sagen, du bist nun eine Ganzzahl (integer).

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Wir sagen zu der Eingabe, du bist nun eine Ganzzahl (integer).

```
benutzereingabe = int(benutzereingabe)
```

- Jetzt können wir mit der Benutzereingabe als Zahl arbeiten.
  - Diese Zahl können wir über print ausgeben lassen.

```
print(benutzereingabe)
```

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Wollen wir nun die Zahl mit einem Text am Anfang ausgeben lassen, kommen kleine Probleme zum Vorschein:

- Wir versuchen nun die STRING-Ausgabe "Eingegeben wurde" mit einer Integer zu verknüpfen
  - und das wird mit einer Fehlermeldung quittiert.

```
print("Eingegeben wurde: " + benutzereingabe)
```

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Hier müssen wir den Typ wieder umwandeln und alles ist gut:

- Gibt der Benutzer allerdings die Zahl in andere Form ein
  - (sprich keine Zahl),
  - dann bekommt man Fehlermeldungen.

```
print("Eingegeben wurde: " + str(benutzereingabe))
```

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Tipp für Fortgeschrittene: unsichtbare input-Eingaben (z.B. für Passwort)

- Eigentlich greifen wir nicht besonders tief in die Trickkiste, aber es passt so perfekt an diesem Platz.
  - Mit der bisherigen input-Eingabe sind immer alle Nutzereingaben sichtbar.
  - Allerdings sollten Passworteingaben beim eintippen unsichtbar sein sonst kann jemand Unberechtigtes das Passwort einfach vom Bildschirm ablesen.
  - Dazu nutzen wir das Modul getpass().

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern

Dazu nutzen wir das Modul getpass.

- Folgender Code macht die Magie:

```
from getpass import getpass

nutzername = input("Nutzername: ")
kennwort   = getpass("Passwort: ")

print("Eingegebener Nutzername", nutzername)
print("Eingegebenes Kennwort", kennwort)
```

Cloud Command

Variablen und Strings
input – Nutzereingaben anfordern