

Python

Zusammenfassung

Dario Scheuber /  [Quelldateien](#)

Inhaltsverzeichnis

Variablen	1
Ausgabe	1
Variablen	2
Typen Umwandlung	2
Operatoren	2
Zahlenformate	3
Strings	3
Raw Text	3
Zeichenliterale	3
Indexieren von Strings	4
Schleifen	4
IF-Statement	4
if - elif - else Statement	5
Verknüpfen von Bedingungen	5
Ablaufdiagramm / Flussdiagramm	5
While-Schleife	6
continue	6
break	6
For-Schleife	6
range	6
enumerate	6
zip	7
zip und enumerate	7

To edit the title page, edit `before-body.tex` in the `config` folder.

Variablen

Ausgabe

Eine einfache Variante um Ausgaben zu machen

```
print("Text")           #Text
print("Text1","Text2")  #Text1 Text2
print(10)               #10
print("Text: ",10)      #Text: 10

name = "Hans"
alt = 18
print(f"Der {name} ist {alt} Jahre alt")
#Der Hans ist 18 Jahre alt
```

Variablen

Zuweisungen mit dem = Zeichen

```
x = 10          #in x Wert 10 gespeichert
y = x          #in y wird x gespeichert Wert 10
print(x,y)     #10 10
print("Text: ",10) #Text: 10
```

! Variablennamen

Können nur Buchstaben, Zahlen oder Unterstriche im namen verwendet werden. Zusätzliche Sonderzeichen sind nicht erlaubt.

```
x = 10          #Integer (int) gerade Zahlen <class 'int'>
x = 11.5        #Floating Point (float) gleitkomma Zahlen
               #<class 'float'>
x = "Text"      #Zeichenketten (String) <class 'str'>
x = True        #Boolscher Wert(Boolean) True/False <class 'bool'>
type(x)         #gibt Typ zurück: <class 'bool'>
```

Typen Umwandlung

```
x = 10          #int: 10
int(x)          #int: 10
float(x)        #float : 10.0
str(x)          #String: 10
bool(x)         #bool: True

int(10.6)       #int: 10 --> Kommawert wird abgeschnitten
bool(-4)        #bool: True
bool(0)         #bool: False
bool(0.0)       #bool: False
```

Operatoren

Operation	Befehl
Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Exponent	**
Modulo (Restberechnung)	%
Floor Division	//
Bitwise XOR	^

```
print(2*2)      #4
print(2**3)     #8
print(1.0/2.0)  #0.5
print(1/2)      #0.5
```

```
print(16%5)      #1
print(7//2)      #3
print(1^3)       #0b1 ^ 0b11 = 0b10 = 2
```

Hinweis

Alle Division wurden in Python mit Float realisiert

Zahlenformate

```
print(bin(10))    #Binär: 0b1010
print(oct(10))    #Oktal: 0o12
print(hex(10))    #Hexadezimal: 0xa
```

Strings

In Python sind Strings mit Doppelten- "Text" und Einfachen-Anführungszeichen 'Text' realisiert.

```
print("Text1"+"Text2")  #Text1Text2
print("Text1", "Text2") #Text1 Text2
print("Text"*5)         #TextTextTextTextText
print("Text\tText")     #Text Text
print("Text\nText")     #Text
                        #Text
print("\U0001f600")     #Unicode: :) (Smile)
```

Sonderzeichen	Befehl
Tab	\t
NewLine	\n
Backslash	\\
"-Zeichen	"
'-Zeichen	'

Raw Text

Mit dem r Vorzeichen wird der nachfolgende String genau so Interpretiert.

```
print(r'C:\some\name') #C:\some\name
print('C:\some\name')  #C:\some
                        #ame
```

Zeichenlitterale

Zeichenlitterale sind Strings, welche die Einrückungen etc. übernehmen.

```
print("""
Usage: Text[options]
    -t
    -h Host
""")
```

```
#Usage: Text[options]
#      -t
#      -h Host
```

Indexieren von Strings

! Wichtig

Strings werden mit folgender notations Indexiert: <Stringname>[<start>:<stop>:<step>]. Negative Indexe wird von hinten gezählt, sowie bei einem negativen Step-Index wird die Zeichenkette rückwärts gezählt.

```
word = "Hello this is a Test"
print(word[0])      #H
print(word[1])      #e
print(word[2])      #l
print(word[3])      #l
print(word[4])      #o

print(word[0:5])     #Hello
print(word[5:9])     # thi
print(word[5: ])     # this is a Test
print(word[ :10])    #Hello this

print(word[:2])      #Hloti saTs
print(word[0:10:2])  #Hloti

print(word[-5:])     # Test
print(word[5:-1])    # this is a Tes
print(word[::-1])    #tseT a si siht olleH
```

Schleifen

IF-Statement

Das if-Statement wird ausgeführt sobald das die condition true ist

```
if condition == 10:      #Wenn condition gleich
                        #10 ist dann
    print("Text1","Text2") #Text1 Text2
```

Vergleichsoperator	Bedeutung
==	ist gleich?
!=	nicht gleich?
>	größer als
<	kleiner als
>=	größer gleich
<=	kleiner gleich

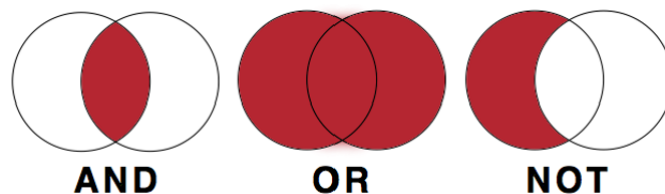
if - elif - else Statement

Bei mehreren if-Statements wird jedes einzelne überprüft auch wenn eines True war. Das elif- oder else-Statement wird nur ausgeführt, wenn das if-Statement false war.

```
if x < 10:
    print("x kleiner als 10")
elif x > 10:
    print("x grösser als 10")
else:
    print("x ist 10")          #Wenn kein Statement True ist
```

Verknüpfen von Bedingungen

```
x = True
y = False
z = x and y          #False
z = x or y           #True
z = not y            #True
```



Ablaufdiagramm / Flussdiagramm

Ein Ablauf- oder Flussdiagramm kann verwendet werden den Prozess /Ablauf eines Algorithmus, Programms, etc. graphisch zu beschreiben.

Baustein	Diagrammzeichen
Terminal/Terminator	
Prozess	
Entscheidung	
Datensatz oder Input/Output	
Flussrichtungspfeil	
Off-Page-Konnektor/Referenz	

While-Schleife

Die `while` Schleife kann verwendet werden, um einen Code Block mehrfach auszuführen, bis eine bestimmte Bedingung eingetreten ist.

```
i = 0
while i < 10:
    print(i)                #0\n 1\n 2\n 3\n 4\n ...
    i += 1

while True:                #Endlosschleife
    print("Endlosschleife")
```

`continue`

'`continue`': Wenn `continue` aufgerufen wird, wird der **Rest** des Code Blockes **übersprungen** und es beginnt ein neuer Durchlauf der Schleife

`break`

'`break`': Wenn `break` aufgerufen wird, wird die **Schleife** sofort **beendet**.

For-Schleife

For in Python wird etwas anders behandelt als in anderen Programmiersprachen. In Python wird `for` verwendet, um **alle Elemente einer Sequenz** zu bearbeiten.

```
for elem in range(10):
    print(elem)            #0\n 1\n 2\n 3\n 4\n ...

for c in "Word":
    print(c)               #W\n o\n r\n d\n

for i in [1,3,5]:
    print(i)               #1\n 3\n 5\n
```

`range`

'`range`': produziert eine Sequenz von (default 0) bis zu dem eingegebenen Wert (10).

```
list(range(10))           #0,1,2,...,9
```

`enumerate`

'`enumerate`': gibt einen Index einer Iterable. `enumerate` liefert dazu ein Tuple (zwei Werte). Diese Werte sind index und Element von einer Iterablen.

```
for i,elem in enumerate("word"):
    print(f"{i}th letter is a {elem}")
#0th letter is a w
#1th letter is a o
#2th letter is a r
#3th letter is a d
```

zip

'zip' um gleichzeitig zwei oder mehr gleichlange Listen zu iterieren. zip aggregiert Element für Element von mehreren Iterables und gibt jeweils ein Tuple mit einem Element von allen Iterables zurück.

1	2	3	4	5	6	Iterable 1
H	A	L	L	O		Iterable 2
(1,H)	(A,H)	(3,L)	(4,L)	(5,O)		Zip Resultat

```
for elem1,elem2 in zip([1,2,3,4,5,6],"hello"):
    print(elem1,elem2)
#1 h
#2 e
#3 l
#4 l
#5 o
```

! zip-Länge

zip gibt nur so lange Tuples aus, wie alle Iterables Elemente haben.

zip und enumerate

```
for idx, tup in enumerate(zip(word_1,word_2)):
    print(f"{idx}th letters of words are: {tup[0]} \t {tup[1]}")
```

! zip und enumerate

Wenn zip und enumerate gleichzeitig angewendet werden, werden leider NICHT 3 Werte zurückgegeben, sondern 1 Index und 1 Tuple (2 Werte) zurückgegeben.