Python3 -- Syntax

- *Syntax* aus dem Griechischen, bedeutet »Satzbau«
- Hier: erlaubte und verbotene Konstruktionen

- Festgelegt durch Grammatik

- *Syntax-Error*

Python3 -- Syntax

- Programm = Folge von Anweisungen
- z.B.:

```
1 print( "Ausgabe" )
2 print( "Ausgabe2" )
```

Python3 -- Syntax

- Programm = Folge von Anweisungen
- z.B.:

```
1 print( "Ausgabe" )
2 print( "Ausgabe2" )
```

- Abarbeitung von oben nach unten

Python3 -- Syntax

- Programm = Folge von Anweisungen
- z.B.:

```
1 print("Ausgabe")
```

- Komplexere Anweisungen:

```
Anweisungskopf:
    Anweisung
    ...
    Anweisung
```

Python3 -- Syntax

- Konkretes Beispiel für komplexe Anweisung:

```
1  if x > 12:
2      print("x ist größer als 12")
3      print("Wirklich!")
4  print("Immer gedruckt")
```

Einrückung: (4) Leerzeichen ODER Tabulator

NICHT mischen!

Python3 -- Syntax

- Einrückungen im interaktiven Modus

```
>>> x = 123
>>> if x > 10:
...     print("Der Interpreter wartet")
...     print("Zweite Zeile!")
...
Der Interpreter wartet
Zweite Zeile!
>>>
```

Python3 -- Syntax

- Umbruch bei langen Zeilen
- Innerhalb Klammern beliebig

```
>>> x = (
...  10
...-2
...+
...11
...)
>>> x
 19
```

Python3 -- Syntax

- Umbruch bei langen Zeilen
- Mit Backslash fast beliebig

```
>>> x \
... =\
... 23
>>> x
23
```

- An allen Positionen, wo Leerzeichen erlaubt

Python3 -- Syntax

- Zusammenfügen mehrerer Zeilen
- Semikolon

```
>>> print("Eins"); print("Zwei")
Eins
Zwei
```

Python3 -- Syntax

- Zusammenfügen von
- Anweisungskopf und -körper

```
>>> x = True
>>> if x: print("Ja!")
...
Ja!
```

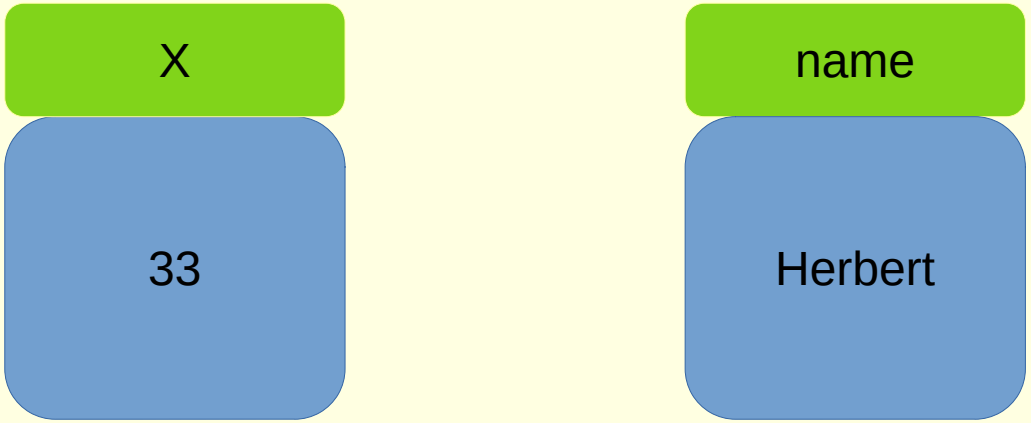
Python3 -- Syntax

- Zusammenfügen von
- Anweisungskopf und -körper
- Auch mit mehreren Anweisungen im Körper

```
>>> x = True
>>> if x: print("Ja!"); print("Und ja!")
...
Ja!
Und Ja!
```

Python3 -- Syntax

- Variablen sind Behälter für Daten
- Haben Namen und Inhalt



Python -- Syntax

Variablennamen:

- Beginnen mit Buchstabe oder "_"
- Gefolgt von Buchstaben, Zahlen oder "_"
- Groß-/Kleinschreibung relevant
- Keine reservierten Wörter

Python -- Syntax

Reservierte / Schlüsselwörter:

`False`	`await`	`else`	`import`	`pass`
`None`	`break`	`except`	`in`	`raise`
`True`	`class`	`finally`	`is`	`return`
`and`	`continue`	`for`	`lambda`	`try`
`as`	`def`	`from`	`nonlocal`	`while`
`assert`	`del`	`global`	`not`	`with`
`async`	`elif`	`if`	`or`	`yield`

Python -- Syntax

Erlaubt sind Schlüsselwörter als Teil des Bezeichners

```
Falsehood

neverbreak

finally_is

lambda4

fromm

Global
```

Python -- Syntax

Kommentare

- Programm darf Erklärungen etc. enthalten
- kein Einfluss auf Ausführung
- für Dokumentation

Python -- Syntax

Kommentare

- Zeilen beginnen mit #

```
1 Vorname = "Fritzi"
2 Nachname = "Meier"
3
4 # Ausgabe des ganzen Namens
5 print(Vorname, Nachname)
```

- Werden bei Ausführung ignoriert

Python -- Syntax

Kommentare

- auch nach Anweisungen

```
1 Vorname = "Fritzi"
2 Nachname = "Meier"
3
4
5  print(Vorname, Nachname) # Ausgabe des ganzen Namens
```

Python -- Syntax

Zeichenketten (Strings)

- Umschlossen von einfachen oder doppelten Anführungszeichen

```
1 Vorname = "Fritzi"
2 Nachname = 'Meier'
3
4 # Ausgabe des ganzen Namens
5 print(Vorname, Nachname)
```

- Kein Unterschied

Python -- Syntax

Zeichenketten (Strings)

- Anführungszeichen in String möglich, wenn verschieden von umschließenden

```
1 A = "Ich sage: 'Ja!' "
2 B = 'Du sagst: "Nein!"'
3 print(A, B)

Ausgabe:
Ich sage: 'Ja!' Du sagst: "Nein!"
```

Python -- Syntax

Zeichenketten (Strings)

- mehrzeilig: drei Anführungszeichen

```
1  strg = """ Zeile 1
2    Zeile 2
3     Zeile 3 """
4
5  print( strg )

Ausgabe:
 Zeile 1
  Zeile 2
   Zeile 3
```

Python -- Syntax

Zeichenketten (Strings)

- mehrzeilig: drei Anführungszeichen
- beide Varianten

```
1  strg = ''' Zeile 1
2    Zeile 2
3     Zeile 3 '''
4
5  print( strg )

Ausgabe:
 Zeile 1
  Zeile 2
   Zeile 3
```

Python -- Syntax

mehrzeilige Strings auch als Kommentare benutzt

```
""" Wir drucken
jetzt ein
Wort aus """

print( "Wort" )
```

Python -- Syntax

- Viele weitere Details
- Einige werden wir kennenlernen
- Neben den Regeln der Syntax gibt sogenannte

  "Best Practice"-Regeln
- Nicht verpflichtend
- Helfen Lesbarkeit, Verständlichkeit

Python – Best Practices

Variablennamen:

- Kleinbuchstaben
- Wörter mit Unterstrich getrennt
- Erklärende Namen

```
1 benutzer_name = "Mmüller"
2 kontostand = 2345
3 ist_registriert = True
```

Python – Best Practices

Leerzeichen um Operatoren und Klammern herum

```
1 benutzer_name="Mmüller"
2
3 print("Herr ",benutzer_name)
4
5 ergebnis=(4+5)*2
```

vs.

```
1 benutzer_name = "Mmüller"
2
3 print( "Herr ", benutzer_name )
4
5 ergebnis = ( 4 + 5 ) * 2
```

Python – Best Practices

Leerzeichen um Operatoren und Klammern herum

```
1 benutzer_name="Mmüller"
2
3 print("Herr ",benutzer_name)
4
5 ergebnis=(4+5)*2
```

vs.

```
1 benutzer_name = "Mmüller"
2
3 print( "Herr ", benutzer_name )
4
5 ergebnis = ( 4 + 5 ) * 2
```

Leerzeilen

Python – Best Practices

Lange Zeilen umbrechen

```
1 transitions = {
2           "q0": {"a": {"q1"}},
3           "q1": {"a": {"q1"}, "": {"q2"}},
4           "q2": {"b": {"q0"}},
5       }
6
7 transitions = { "q0": {"a": {"q1"}}, "q1": {"a": {"q1"}, "" {"q2"
```

Python – Best Practices

Auf Folien:

- wenig Platz
- kein Kontext
- daher oft nicht gemäß best practices