Der Kochrezept-Vergleich

- Programmierer == Kochrezept-Schreiber

Der Kochrezept-Vergleich

- Programmierer == Kochrezept-Schreiber
- Computer == Koch

Python als Interpreter Sprache

- im Gegensatz zu Programmiersprachen wie C, C++ oder Fortran eine *Interpretersprache*

Python als Interpreter Sprache

- im Gegensatz zu Programmiersprachen wie C, C++ oder Fortran eine *Interpretersprache*
- Python-Code muss nicht kompiliert werden

Python als Interpreter Sprache

- im Gegensatz zu Programmiersprachen wie C, C++ oder Fortran eine *Interpretersprache*
- Python-Code muss nicht kompiliert werden
- Code wird zur Laufzeit des Programmes vom Python Interpreter abgearbeitet

Python als Interpreter Sprache

- im Gegensatz zu Programmiersprachen wie C, C++ oder Fortran eine *Interpretersprache*
- Python-Code muss nicht kompiliert werden
- Code wird zur Laufzeit des Programmes vom Python Interpreter abgearbeitet
- Programm (hier mit Namen *hello_world.py*) starten per

```
python hello_world.py
```

Hello World

Simples Programm zum Start:

- Wir legen eine Textdatei **hello_world.py** mit folgendem Inhalt an:

```
print "Hello World"
```

Hello World

Simples Programm zum Start:

- Wir legen eine Textdatei **hello_world.py** mit folgendem Inhalt an:

```
print "Hello World"
```

- Glückwunsch zum ersten Python-Programm!

Hello World

Simples Programm zum Start:

- Wir legen eine Textdatei **hello_world.py** mit folgendem Inhalt an:

```
print "Hello World"
```

- Glückwunsch zum ersten Python-Programm!
- Zum Vergleich “Hello World” in Java:

```
class Main{
    public static void main(String[] args){
        System.out.println("Hello World");
    }
}
```

Hello World - Details

- Der *print* Befehl lässt Programme in die Konsole schreiben

Hello World - Details

- Der *print* Befehl lässt Programme in die Konsole schreiben
- Was ist “Hello World”?

Hello World - Details

- Der *print* Befehl lässt Programme in die Konsole schreiben
- Was ist “Hello World”?
- `type("Hello World")`

Hello World - Details

- Der *print* Befehl lässt Programme in die Konsole schreiben
- Was ist “Hello World”?
- `type("Hello World")`
- Ausgabe:

```
<type 'str'>
```

Datentypen

Folgende grundlegende Datentypen:

- **Integer**: ganzzahlige Zahlen

Datentypen

Folgende grundlegende Datentypen:

- **Integer**: ganzzahlige Zahlen
- **Float**: Gleitkomma-Zahlen

Datentypen

Folgende grundlegende Datentypen:

- **Integer**: ganzzahlige Zahlen
- **Float**: Gleitkomma-Zahlen
- **String**: Zeichenketten

Datentypen

Folgende grundlegende Datentypen:

- **Integer**: ganzzahlige Zahlen
- **Float**: Gleitkomma-Zahlen
- **String**: Zeichenketten
- **Boolean**: Logische Werte, können True oder False sein

Datentypen

Folgende grundlegende Datentypen:

- **Integer**: ganzzahlige Zahlen
- **Float**: Gleitkomma-Zahlen
- **String**: Zeichenketten
- **Boolean**: Logische Werte, können True oder False sein
- **NoneType**: Spezieller Datentyp, der z.B. als Default-Argument in Funktionen verwendet wird (kommt später)

Variablen

- Jeder dieser Datentypen kann nun in einer Variablen gespeichert werden.

Variablen

- Jeder dieser Datentypen kann nun in einer Variablen gespeichert werden.
- Zuweisungsoperator “=”

Variablen

- Jeder dieser Datentypen kann nun in einer Variablen gespeichert werden.
- Zuweisungsoperator “=”
- ```
  ganze_zahl = 10
  komma_zahl = 1.234
  komplexe_zahl = 3.123 + 2j
  string_variable = "Hallo Welt"
  boolsche_variable = True
  none_variable = None
  ```
  ```

Variablen

- Jeder dieser Datentypen kann nun in einer Variablen gespeichert werden.
- Zuweisungsoperator "="
- ```
  ganze_zahl = 10
  komma_zahl = 1.234
  komplexe_zahl = 3.123 + 2j
  string_variable = "Hallo Welt"
  boolsche_variable = True
  none_variable = None
  ```
- ```
  print type(ganze_zahl)
  print type(komma_zahl)
  print type(komplexe_zahl)
  print type(string_variable)
  print type(boolsche_variable)
  print type(none_variable)
  ```

Variablen

- Jeder dieser Datentypen kann nun in einer Variablen gespeichert werden.
- Zuweisungsoperator “=”
- ```
  ganze_zahl = 10
  komma_zahl = 1.234
  komplexe_zahl = 3.123 + 2j
  string_variable = "Hallo Welt"
  boolsche_variable = True
  none_variable = None
  ```
- ```
  print type(ganze_zahl)
  print type(komma_zahl)
  print type(komplexe_zahl)
  print type(string_variable)
  print type(boolsche_variable)
  print type(none_variable)
  ```
- -> Arbeit an Studenten übergeben

Variablen

- `<type 'int'>`
  `<type 'float'>`
  `<type 'complex'>`
  `<type 'str'>`
  `<type 'bool'>`
  `<type 'NoneType'>`

Variablen

- ```
  <type 'int'>
  <type 'float'>
  <type 'complex'>
  <type 'str'>
  <type 'bool'>
  <type 'NoneType'>
  ```
- Gültige Variablennamen: keine Zahlen/Sonderzeichen am Anfang
  ```

Kommentare

- Wichtig für den Programmierer

Kommentare

- Wichtig für den Programmierer
- Hilft dabei, fremden/alten eigenen Code zu verstehen

Kommentare

- Wichtig für den Programmierer
- Hilft dabei, fremden/alten eigenen Code zu verstehen
- ```
  print 1, 2, 3 # wird ausgeführt, denn die Raute
                # beginnt erst nach dem Befehl
  # print 4, 5, 6  wird nicht ausgeführt,
  # da zu Beginn der Zeile eine Raute steht
  ```
  ```

Operatoren

Operatoren können:

- Variablen neue Werte zuweisen

Operatoren

Operatoren können:

- Variablen neue Werte zuweisen
- Werte miteinander vergleichen

Operatoren

Operatoren können:

- Variablen neue Werte zuweisen
- Werte miteinander vergleichen
- mathematische Ausdrücke berechnen

Operatoren

- Zuweisungsoperator $=$

  ```
  meine_variable = 3
  ```

  Dieser Operator weist einer Variablen einen Wert zu.

Operatoren

- Zuweisungsoperator =

```
meine_variable = 3
```

Dieser Operator weist einer Variablen einen Wert zu.

- Arithmetische Operatoren +, -, *, /, %, **

```
print 4 * 3
print 10 / 5
print 11 / 4
print 10 + 3 - 11
print 14 % 3
print 10**2
```

Operatoren

- Zuweisungsoperator =

  ```
  meine_variable = 3
  ```

  Dieser Operator weist einer Variablen einen Wert zu.

- Arithmetische Operatoren +, -, *, /, %, **

  ```
  print 4 * 3
  print 10 / 5
  print 11 / 4
  print 10 + 3 - 11
  print 14 % 3
  print 10**2
  ```

  - / berechnet Integer-Division: 11/4 ist 2 (Rest 3)

Operatoren

- Zuweisungsoperator =

  ```
  meine_variable = 3
  ```

  Dieser Operator weist einer Variablen einen Wert zu.

- Arithmetische Operatoren +, -, *, /, %, **

  ```
  print 4 * 3
  print 10 / 5
  print 11 / 4
  print 10 + 3 - 11
  print 14 % 3
  print 10**2
  ```

  - / berechnet Integer-Division: 11/4 ist 2 (Rest 3)
  - Kombination von Zuweisungs- und arithmetischen Operatoren:

    ```
    x = 4*3 + 13/5
    print x
    ```

Operatoren

- Vergleichsoperatoren $==$, $<$, $<=$, $>$, $>=$

Operatoren

- Vergleichsoperatoren $==$, $<$, $<=$, $>$, $>=$
  - vergleichen zwei Werte miteinander und geben einen booleschen Wert zurück.

```
print 4 + 3 == 7
print -11 < -13
```

Operatoren

- Vergleichsoperatoren $==$, $<$, $<=$, $>$, $>=$
  - vergleichen zwei Werte miteinander und geben einen booleschen Wert zurück.

    ```
    print 4 + 3 == 7
    print -11 < -13
    ```

  - Ausgabe:

    ```
    True
    False
    ```

Operatoren

- Logische Operatoren not, and, or

Operatoren

- Logische Operatoren not, and, or
  - Eingabe: boolesche Werte, geben booleschen Wert zurück.

Operatoren

- Logische Operatoren not, and, or
  - Eingabe: boolesche Werte, geben booleschen Wert zurück.
  - **not** negiert einen booleschen Wert:

```
print not False # Ausgabe?
print not True  # Ausgabe?
```

Operatoren

- Logische Operatoren not, and, or
  - Eingabe: boolesche Werte, geben booleschen Wert zurück.
  - **not** negiert einen booleschen Wert:

```
print not False # Ausgabe?
print not True  # Ausgabe?
```

  - **and** ergibt *True* wenn beide Eingabewerte *True* sind, ansonsten *False*

```
print True and True # Ausgabe?
print True and False # Ausgabe?
print False and False # Ausgabe?

print 1+2==3 and 4*4==16
print 1+2==3 and False
x = 7
print x == 7 and x+3 == 10 and True
```

Operatoren

- **or** ergibt *True* wenn mindestens einer der beiden Eingabewerte *True* ist.

```
print True or True
print True or False
print False or False
x = 5
print x<5 or x>5
```

Schleifen

Zwei Arten von Schleifen:

For-Schleife

- Wiederhole einen Code-Teil eine bestimmte (feste) Anzahl von Malen

Schleifen

Zwei Arten von Schleifen:

For-Schleife

- Wiederhole einen Code-Teil eine bestimmte (feste) Anzahl von Malen
- Z.B.

```
for i in range(10):
  print i # Einrückung wichtig!
```

Schleifen

Zwei Arten von Schleifen:

For-Schleife

- Wiederhole einen Code-Teil eine bestimmte (feste) Anzahl von Malen
- Z.B.

```
for i in range(10):
  print i # Einrückung wichtig!
```

- Ausgabe:

```
0
1
2
...
7
8
9
```

Schleifen

Zwei Arten von Schleifen:

For-Schleife

- Wiederhole einen Code-Teil eine bestimmte (feste) Anzahl von Malen
- Z.B.

```
for i in range(10):
  print i # Einrückung wichtig!
```

- Ausgabe:

```
0
1
2
...
7
8
9
```

Bedingte Anweisungen (if-statements)

Computer muss auf verschiedene Situationen (unterschiedliche Variablenwerte etc.) reagieren können:

```
if <Bedingung>:
    <Anweisung 1>
elif <Bedingung 2>:
    <Anweisung 2>
else:
    <Anweisung 3>
```

<Bedingung> ist dabei ein Ausdruck, der einen boolschen Wert zurückgibt. Die Anweisungen sind beliebiger Python-Code.

Beispiel

```
x = 7

if 0 <= x <= 5:
    print "x lässt sich an einer Hand abzählen"
elif 6 <= x <= 10:
    print "x lässt sich an zwei Händen abzählen"
elif x > 10:
    print "Dieses x ist mir zu groß..."
else:
    print "Dieses x ist mir zu klein..."
```

- Was passiert in diesem Beispiel?

Beispiel

```
x = 7

if 0 <= x <= 5:
    print "x lässt sich an einer Hand abzählen"
elif 6 <= x <= 10:
    print "x lässt sich an zwei Händen abzählen"
elif x > 10:
    print "Dieses x ist mir zu groß..."
else:
    print "Dieses x ist mir zu klein..."
```

- Was passiert in diesem Beispiel?
- Für verschiedene x-Werte ausprobieren!

Ein- und Ausgabe

Ein- und Ausgabe:

- Einlesen und Schreiben von Dateien

Ein- und Ausgabe

Ein- und Ausgabe:

- Einlesen und Schreiben von Dateien
- Einlesen von Tastatureingaben

Schreiben

```
datei = open("meine_datei", "w")
print >> datei, "Dieser Satz steht in der ersten Zeile"
print >> datei, "Und der hier in der zweiten"
datei.close()
```

- “**w**” wie *writeable*

Schreiben

```
datei = open("meine_datei", "w")
print >> datei, "Dieser Satz steht in der ersten Zeile"
print >> datei, "Und der hier in der zweiten"
datei.close()
```

- “**w**” wie *writeable*
- “**r**” wie *readable*

Schreiben

```
datei = open("meine_datei", "w")
print >> datei, "Dieser Satz steht in der ersten Zeile"
print >> datei, "Und der hier in der zweiten"
datei.close()
```

- “**w**” wie *writeable*
- “**r**” wie *readable*
- “**a**” wie *append* (hängt Text an bestehende Datei an)

Problem:

- ```
  datei = open("meine_datei", "w")
  print 1/0 # Das wird mit einer Fehlermeldung abbrechen.
  datei.close() # diese Zeile wird nie ausgeführt werden
  ```
  ```

Problem:

- ```
  datei = open("meine_datei", "w")
  print 1/0 # Das wird mit einer Fehlermeldung abbrechen.
  datei.close() # diese Zeile wird nie ausgeführt werden
  ```
- Besser:

  ```
  with open("meine_datei", "w") as datei:
      print >> datei, "Dieser Satz steht in der ersten Ze
      print >> datei, "Und der hier in der zweiten"
      print 1/0 # Trotz Fehlermeldung wird Python
                # die Datei schließen
  ```
  ```

Problem:

- ```
  datei = open("meine_datei", "w")
  print 1/0 # Das wird mit einer Fehlermeldung abbrechen.
  datei.close() # diese Zeile wird nie ausgeführt werden
  ```
- Besser:

  ```
  with open("meine_datei", "w") as datei:
      print >> datei, "Dieser Satz steht in der ersten Ze
      print >> datei, "Und der hier in der zweiten"
      print 1/0 # Trotz Fehlermeldung wird Python
                # die Datei schließen
  ```
- Ausprobieren!
  ```

Einlesen:

```
with open("meine_datei", "r") as datei:
    for line in datei:
        print line
```

- Mittels for-Schleife über Zeilen der eingelesenen Datei iterieren!

Einlesen:

```
with open("meine_datei", "r") as datei:
    for line in datei:
        print line
```

- Mittels for-Schleife über Zeilen der eingelesenen Datei iterieren!
- Ausprobieren!

Tastatureingabe:

- User gibt etwas in Terminal ein, Python speichert es in Variable

Tastatureingabe:

- User gibt etwas in Terminal ein, Python speichert es in Variable
- ```
  user_input = raw_input("Bitte geben Sie etwas ein: ")
  print "Sie gaben ein:", user_input
  ```
  ```

Übungsaufgaben!

Viel Spaß!