Grundlagen

Felix Döring, Felix Wittwer, Anton Obersteiner

Python-Kurs

9. November 2021

Gliederung

Dieser Kurs & Links

- Ressourcen
  - Beim Suchen: Vermeidet die python 2.7 Doku
  - `docs.python.org`: offizielle Dokumentation
  - `http://auditorium.inf.tu-dresden.de`
  - `https://github.com/fsr`: iFSR auf Github

ich `mailto:anton.obersteiner1@mailbox.tu-dresden.de`

Der Python Interpreter

- Übliche Python-Versionen: 2.7 und 3.x (bessere Features)
- Python auf `https://www.python.org` heruntergeladen und installieren
- UNIX: `...install... python3 python3-dev`

Der Python Interpreter

- Übliche Python-Versionen: 2.7 und 3.x (bessere Features)
- Python auf `https://www.python.org` heruntergeladen und installieren
- UNIX: `...install... python3 python3-dev`
- Python wird interpretiert, nicht kompiliert

```
1 #SyntaxError: closing parenthesis ')' [...]
2 def func(): print[]
3 #NameError: name 'function' is not defined
4 function()
```

Editor/IDE

Editor einfache Färbung/Vervollständigung

- `https://atom.io` (weil Github)
- `http://www.sublimetext.com/3`
- `https://c9.i`: cloud9 (online, free für open source Projekte)

IDEs mehr Unterstützung → besonders für große Projekte

- PyCharm (free + professional für Studenten)
- IntelliJ, VSCode, ...
- `idle` aus `python3-dev` oder `idle`

```
1 #z.B. in hello_world.py
2
3 print("Hello World")
```

IDLE?

Python ausführen

- Python-Datei: `text_file.py`
- Im Terminal starten: `python3` bzw. `Python.exe`
- Ausführbar machen: Windows: Python wählen
- Linux: Pfad in die erste Zeile!
- dann: graphisch oder `chmod +x Planet.py`

```
1 #!/usr/bin/env python3
2
3 what_i_am = "tired"
4 print("Hi dad, I'm", what_i_am)
5 print("Hi " + what_i_am + ", I'm dad")
```

IDLE! $\rightarrow$ Variablen, Funktionen, Typen

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe
`bool`	Wahrheitswert (`True`, `False`)

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe
`bool`	Wahrheitswert (`True`, `False`)
`function`	Funktionen

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe
`bool`	Wahrheitswert (`True`, `False`)
`function`	Funktionen
`float`	Kommazahl: `d = sqrt(2); d**2 == 2?`

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe
`bool`	Wahrheitswert (`True`, `False`)
`function`	Funktionen
`float`	Kommazahl: `d = sqrt(2); d**2 == 2?`
`None`	Das Nichts
`type`	Grundtyp aller Typen (z.B. `type(x) == int`)

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe
`bool`	Wahrheitswert (`True`, `False`)
`function`	Funktionen
`float`	Kommazahl: `d = sqrt(2); d**2 == 2?`
`None`	Das Nichts
`type`	Grundtyp aller Typen (z.B. `type(x) == int`)
`list`	standard Liste: `L = [1, 2, "drei"]`

Datentypen

builtin Datentypen:

Name	Funktion
`object`	Basistyp, alles erbt von `object`
`str`	Strings, Zeichenketten
`int`	Ganzzahl "beliebiger" Größe
`bool`	Wahrheitswert (`True`, `False`)
`function`	Funktionen
`float`	Kommazahl: `d = sqrt(2); d**2 == 2?`
`None`	Das Nichts
`type`	Grundtyp aller Typen (z.B. `type(x) == int`)
`list`	standard Liste: `L = [1, 2, "drei"]`
`tuple`	unveränderbares n-Tupel
`set`	(mathematische) Menge von Objekten
`frozenset`	unveränderbare (mathematische) Menge von Objekten
`dict`	Hash-Map

Namenskonvention

Klassen *PascalCase*

Variablen, Funktionen, Methoden *snake_case*

protected Variablen, Funktionen, Methoden `_intern` oder `__privat__`

Merke kein Zugriffsmanagement: man könnte, sollte aber nicht

Indent und Kommentare

- Funktionen definieren mit
  `def <funktionsname>([parameter_liste, ...]): block`
- Codeblöcke sind gleichmäßig eingerückt (`Tab != Space`)

Kommentare:

```
#!/usr/bin/env python3
# in python nur einzeilige Kommentare

def my_function(everything):
    """
    my_function takes everything
    and does nothing with it
    a disgrace to its family
    """
    pass
```

Aufgabe

Eine simple Text-Funktion:

```
1 def present(name, alter, ort):
2     return ...
3 """ e.g.: present("Anton", 19, "Dresden") -> "Anton, 19 aus
      Dresden" """
```

Aufgabe

Eine simple Text-Funktion:

```
1 def present(name, alter, ort):
2     return ...
3 """ e.g.: present("Anton", 19, "Dresden") -> "Anton, 19 aus
      Dresden" """
```

```
1 def present(name, alter, ort):
2     return name + ", " + str(alter) + " aus " + ort
```

Aufgabe

Eine simple Text-Funktion:

```
1 def present(name, alter, ort):
2     return ...
3 """ e.g.: present("Anton", 19, "Dresden") -> "Anton, 19 aus
      Dresden" """
```

```
1 def present(name, alter, ort):
2     return name + ", " + str(alter) + " aus " + ort
```

```
1 print(present("Py-Kurs", "am Di in der 5.", "dem FSR-
      Kurssystem"))
```

Aufgabe

Eine simple Text-Funktion:

```
1 def present(name, alter, ort):
2     return ...
3 """ e.g.: present("Anton", 19, "Dresden") -> "Anton, 19 aus
      Dresden" """
```

```
1 def present(name, alter, ort):
2     return name + ", " + str(alter) + " aus " + ort
```

```
1 print(present("Py-Kurs", "am Di in der 5.", "dem FSR-
      Kurssystem"))
```

Operatoren?

Operatoren

text +, *, %

Operatoren

text	`+`, `*`, `%`
mathematisch	`+`, `-`, `*`, `/`, `//`, `%`, `2 ** 5` nicht: `2 ^ 5`
bitweise	`&`, `|`, `<<`, `>>`, `^` (xor), `~` (invertieren)

Operatoren

text	`+`, `*`, `%`
mathematisch	`+`, `-`, `*`, `/`, `//`, `%`, `2 ** 5` nicht: `2 ^ 5`
bitweise	`&`, `|`, `<<`, `>>`, `^` (xor), `~` (invertieren)
vergleichend	`<`, `>`, `<=`, `>=`, `==` (Wert gleich), `is` (gleiches Objekt/gleiche Referenz)

Operatoren

text	`+`, `*`, `%`
mathematisch	`+`, `-`, `*`, `/`, `//`, `%`, `2 ** 5` nicht: `2 ^ 5`
bitweise	`&`, `|`, `<<`, `>>`, `^` (xor), `~` (invertieren)
vergleichend	`<`, `>`, `<=`, `>=`, `==` (Wert gleich), `is` (gleiches Objekt/gleiche Referenz)
logisch	`and`, `or`, `not`

Operatoren

text	+, *, %
mathematisch	+, -, *, /, //, %, 2 ** 5 nicht: 2 ^ 5
bitweise	&, |, <<, >>, ^ (xor), ~ (invertieren)
vergleichend	<, >, <=, >=, == (Wert gleich), is (gleiches Objekt/gleiche Referenz)
logisch	and, or, not
Aufgabe	if/else, return <wert>, type

```
1 #!/usr/bin/env python3
2 def isnumber(something):
3     """is something an int or a float?"""
4     if frage:
5         tu...
6     else:
7         tu...
8     weiter...
```

If/Else/Boolean

```
def isnumber(something):
    if type(something) == int or type(something) == float:
        return True
    else:
        return False
```

If/Else/Boolean

```
def isnumber(something):
    if type(something) == int or type(something) == float:
        return True
    else:
        return False
```

```
def isnumber(something):
    return (
        type(something) == int or
        type(something) == float
    )
```

besondere Operatoren

- `()` für Aufrufbares (Funktionen):

  ```
  present("A", 19, "DD")
  ```

besondere Operatoren

`()` für Aufrufbares (Funktionen):

```
present("A", 19, "DD")
```

`[]` für Datenstrukturen mit Index:

```
name[0], "name"[1:3]
```

besondere Operatoren

`()` für Aufrufbares (Funktionen):

```
present("A", 19, "DD")
```

`[]` für Datenstrukturen mit Index:

```
name[0], "name"[1:3]
```

`.` für Methoden und Attribute:

```
"name".index("m")
```

Zusatz und Ausblick

Unschön:

```
1 def present(name, alter, ort):
2     return name + ", " + str(alter) + " aus " + ort
```

Zusatz und Ausblick

Unschön:

```
1 def present(name, alter, ort):
2     return name + ", " + str(alter) + " aus " + ort
```

```
1 def present(name, alter, ort):
2     return f"{name}, {alter} aus {ort})"
```

Zusatz und Ausblick

Unschön:

```
1 def present(name, alter, ort):
2     return name + ", " + str(alter) + " aus " + ort
```

```
1 def present(name, alter, ort):
2     return f"{name}, {alter} aus {ort})"
```

```
1 def present_list(name, infos):
2     return f"{name}@({...})"
3
4 positions = ["APB", "E005", "Di5"]
5 debug_list("Py-Kurs", positions)
```