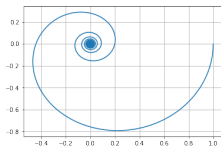




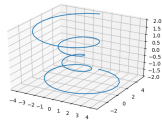
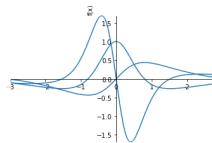
Python fürs Ingenieur-Studium

Erstsemester-Einführung der Fakultät Eul, 11.10.2019

<https://fsfw-dresden.de/python-ws>



```
f1 := -\frac{2x \sin(x)}{(x^2 + 1)^2} + \frac{\cos(x)}{x^2 + 1}
...
f2 := \frac{8x^2 \sin(x)}{(x^2 + 1)^2} - \frac{4 \cos(x)}{x^2 + 1} - \sin(x) - \frac{2 \sin(x)}{x^2 + 1}
...
f1 simplify() := \frac{-2x \sin(x) + (x^2 + 1) \cos(x)}{(x^2 + 1)^2}
...
f2 simplify() := \frac{8x^2 \sin(x) - (x^2 + 1)^2 \sin(x) - 2(x^2 + 1)(2x \cos(x) + \sin(x))}{(x^2 + 1)^3}
```



<Eigenwerbung>

Wer sind wir?



- ▶ Hochschulgruppe an der TU (gegründet 2014, ca. 7 P.)
- ▶ Studierende (TU, HTW) und andere Leute
- ▶ Hochschulen als Zielgruppe (Multiplikationswirkung) und Arbeitsfeld (Räume, Strukturen)

Wer sind wir?



- ▶ Hochschulgruppe an der TU (gegründet 2014, ca. 7 P.)
- ▶ Studierende (TU, HTW) und andere Leute
- ▶ Hochschulen als Zielgruppe (Multiplikationswirkung) und Arbeitsfeld (Räume, Strukturen)

- ▶ Bisherige Projekte
 - ▶ Linux-Install-Party, Linux-Presentation-Day
 - ▶ Verschlüsselungsgewinnspiel
 - ▶ Monatliche Sprechstunde zu \LaTeX u.a.
 - ▶ Formulierung eines Programmpapiers
 - ▶ „Uni-Stick“: 8 GB mit freier Software

Warum machen wir das? Aus Überzeugung!



- ▶ *Überzeugung 1*: freie und quelloffene Software ist (oft) besser (technische + nicht technische Argumente)

Warum machen wir das? Aus Überzeugung!



- ▶ *Überzeugung 1*: freie und quelloffene Software ist (oft) besser (technische + nicht technische Argumente)
- ▶ *Überzeugung 2*: öffentlich finanzierte wissenschaftliche Inhalte (AutorInnen, GutachterInnen) sollten nicht von öffentlich finanzierten Bibliotheken für horrenden Summen von Zeitschriften-Verlagen gekauft werden müssen

Software: Frei versus Proprietär



Vier Freiheiten freier Software

Verwenden



Verstehen



Verändern



Verbreiten



Software: Frei versus Proprietär



Vier Freiheiten freier Software

Verwenden



Verstehen



Verändern



Verbreiten



Vorteile:

- ▶ Kontrolle behalten
- ▶ Erkenntnisgewinn
- ▶ Lizenzkosten: 0 €
- ▶ Anpassbarkeit an eigene Bedürfnisse
- ▶ Hersteller-Unabhängigkeit
(kein Vendor Lock-in)

Nachteile Proprietärer Software

- ▶ Intransparenz (Bsp: Wahlsoftware)
- ▶ Hintertüren? (Win10-Verbot für Dienstgebrauch)
- ▶ Abhängigkeit

Software: Frei versus Proprietär



Vier Freiheiten freier Software

Verwenden



Verstehen



Verändern



Verbreiten



Vorteile:

- ▶ Kontrolle behalten
- ▶ Erkenntnisgewinn
- ▶ Lizenzkosten: 0 €
- ▶ Anpassbarkeit an eigene Bedürfnisse
- ▶ Hersteller-Unabhängigkeit
(kein Vendor Lock-in)

Nachteile Proprietärer Software

- ▶ Intransparenz (Bsp: Wahlsoftware)
- ▶ Hintertüren? (Win10-Verbot für Dienstgebrauch)
- ▶ Abhängigkeit

Warum gibt es MS-Office eigentlich kostenlos für Studierende?

Software: Frei versus Proprietär



Vier Freiheiten freier Software

Verwenden



Verstehen



Verändern



Verbreiten



Vorteile:

- ▶ Kontrolle behalten
- ▶ Erkenntnisgewinn
- ▶ Lizenzkosten: 0 €
- ▶ Anpassbarkeit an eigene Bedürfnisse
- ▶ Hersteller-Unabhängigkeit
(kein Vendor Lock-in)

Nachteile Proprietärer Software

- ▶ Intransparenz (Bsp: Wahlsoftware)
- ▶ Hintertüren? (Win10-Verbot für Dienstgebrauch)
- ▶ Abhängigkeit

Warum gibt es MS-Office eigentlich „kostenlos“ für Studierende?

Software: Frei versus Proprietär



Vier Freiheiten freier Software

Verwenden



Verstehen



Verändern



Verbreiten



Vorteile:

- ▶ Kontrolle behalten
- ▶ Erkenntnisgewinn
- ▶ Lizenzkosten: 0 €
- ▶ Anpassbarkeit an eigene Bedürfnisse
- ▶ Hersteller-Unabhängigkeit
(kein Vendor Lock-in)

Nachteile Proprietärer Software

- ▶ Intransparenz (Bsp: Wahlsoftware)
- ▶ Hintertüren? (Win10-Verbot für Dienstgebrauch)
- ▶ Abhängigkeit

Warum gibt es MS-Office eigentlich „kostenlos“ für Studierende?

Offener Brief:

„public money ⇒ public code“

<https://publiccode.eu>

Projekt Uni-Stick (4. Auflage)



- ▶ **3000** Flyer in Ersti-Tüten: **Gutscheine** für 8 GB Stick mit Freier Software fürs Studium,
- ▶ Live-Linux / freie Windows-Programme

Projekt Uni-Stick (4. Auflage)



- ▶ **3000** Flyer in Ersti-Tüten: **Gutscheine** für 8 GB Stick mit Freier Software fürs Studium,
- ▶ Live-Linux / freie Windows-Programme
- ▶ Vorbereitung hat Arbeit und Spaß gemacht



Projekt Uni-Stick (4. Auflage)



- ▶ **3000** Flyer in Ersti-Tüten: **Gutscheine** für 8 GB Stick mit Freier Software fürs Studium,
- ▶ Live-Linux / freie Windows-Programme
- ▶ Vorbereitung hat Arbeit und Spaß gemacht
- ▶ Ist gut angekommen (ca. 250 TN)



Projekt Uni-Stick (4. Auflage)



- ▶ **3000** Flyer in Ersti-Tüten: **Gutscheine** für 8 GB Stick mit Freier Software fürs Studium,
- ▶ Live-Linux / freie Windows-Programme
- ▶ Vorbereitung hat Arbeit und Spaß gemacht
- ▶ Ist gut angekommen (ca. 250 TN)



Ausgabe-Veranstaltung 2019
04.11., 18:30 Uhr, [APB E023](#)



- ▶ Fortführung „Uni-Stick“
- ▶ Studierende zum Nutzen/Verbessern freier Software animieren
 - ▶ Mehr Blog-Beiträge
 - ▶ Kurse (\LaTeX / **Python** /Git / Inkscape / ...)
 - ▶ Infrastruktur-Stipendium
 - ▶ OpenSource-Wettbewerb/Preis
 - ▶ ...
- ▶ Aufmerksamkeit erzeugen / Lobby-Arbeit („Landesverträge“)
- ▶ Vernetzung mit anderen Städten

Weitere Informationen



<https://fsfw-dresden.de/>

uni-stick
blog
newsletter
mitmachen
git-ws
bash-ws
python-ws



kontakt@fsfw-dresden.de

</Eigenwerbung>

Warum Python? (1)



Python als Programmiersprache

- ▶ Klare, lesbare Syntax (wenig „Ballast“)
- ▶ Objektorientiert, prozedural, funktional programmierbar
- ▶ Nützliche eingebaute Datentypen (`list`, `tuple`, `dict`, `set`, ...)
- ▶ Einfache Modularisierung (`import this`)
- ▶ Gute Fehlerverwaltung (Exceptions)
- ▶ Umfangreiche Standardbibliothek
- ▶ Einfache Einbindung von externem Code (C, C++, Fortran)

Warum Python? (1)



Python als Programmiersprache

- ▶ Klare, lesbare Syntax (wenig „Ballast“)
- ▶ Objektorientiert, prozedural, funktional programmierbar
- ▶ Nützliche eingebaute Datentypen (`list`, `tuple`, `dict`, `set`, ...)
- ▶ Einfache Modularisierung (`import this`)
- ▶ Gute Fehlerverwaltung (Exceptions)
- ▶ Umfangreiche Standardbibliothek
- ▶ Einfache Einbindung von externem Code (C, C++, Fortran)

⇒

- ▶ Leicht zu lernen
- ▶ Problemorientiert (mächtig und flexibel)
- ▶ Motivationspotenzial ↗, Frustrationspotenzial ↘

Außerdem: Plattformübergreifend / frei und quelloffen / große u. aktive Community

Warum Python? (2)



Python als Werkzeug für Ingenieur*innen:

- ▶ Numerisches Rechnen (lin. Algebra, DGLn, Optimierung, ...)
- ▶ Symbolisches Rechnen (Ableiten, Integrieren, Gl. lösen, ...)
- ▶ Visualisieren (2D, 3D, in Publikationsqualität)
- ▶ Grafische Benutzerschnittstelle (GUI)
- ▶ Kommunikation mit externen Geräten
- ▶ Parallelisierung

Warum Python? (2)



Python als Werkzeug für Ingenieur*innen:

- ▶ Numerisches Rechnen (lin. Algebra, DGLn, Optimierung, ...)
 - ▶ Symbolisches Rechnen (Ableiten, Integrieren, Gl. lösen, ...)
 - ▶ Visualisieren (2D, 3D, in Publikationsqualität)
 - ▶ Grafische Benutzerschnittstelle (GUI)
 - ▶ Kommunikation mit externen Geräten
 - ▶ Parallelisierung
-
- ▶ Python für andere Fächer/Projekte nützlich
- ⇒ Gestärkte „Forschungskompetenz“
(Studien-, Master-, Diplomarbeiten, ...)

Warum Python? (2)



Python als Werkzeug für Ingenieur*innen:

- ▶ Numerisches Rechnen (lin. Algebra, DGLn, Optimierung, ...)
 - ▶ Symbolisches Rechnen (Ableiten, Integrieren, Gl. lösen, ...)
 - ▶ Visualisieren (2D, 3D, in Publikationsqualität)
 - ▶ Grafische Benutzerschnittstelle (GUI)
 - ▶ Kommunikation mit externen Geräten
 - ▶ Parallelisierung
- } heute
- ▶ Python für andere Fächer/Projekte nützlich
- ⇒ Gestärkte „Forschungskompetenz“
(Studien-, Master-, Diplomarbeiten, ...)

Warum Python(3)



Probleme an Fakultät Eul

1. LV Info2 (Java) erscheint wenig relevant
→ Vernachlässigung → später fehlt Programmiererfahrung

Abhilfe: Python statt Java

- ▶ Programmieren lehren mit Python ✓
- ▶ Ingenieur-Probleme Lösen mit Python ✓ (Relevanz → Motivation)
- ▶ Mehr Informationen:

<https://wwwpub.zih.tu-dresden.de/~knoll/pykurs/grundstudium.html>

Warum Python(3)



Probleme an Fakultät Eul

1. LV Info2 (Java) erscheint wenig relevant
→ Vernachlässigung → später fehlt Programmiererfahrung

Abhilfe: Python statt Java

- ▶ Programmieren lehren mit Python ✓
- ▶ Ingenieur-Probleme Lösen mit Python ✓ (Relevanz → Motivation)
- ▶ Mehr Informationen:

<https://wwwpub.zih.tu-dresden.de/~knoll/pykurs/grundstudium.html>

2. Wichtige Lehrveranstaltungen setzen auf MATLAB.
Günstig für Studis, aber sehr teuer danach (Abhängigkeit!)

Abhilfe: Python statt MATLAB

- ▶ Python ist freie Software
- ▶ Python kann nicht alles aber das meiste, was MATLAB kann
- ▶ Python kann noch viel mehr (=richtige Programmiersprache)



Ziele für heute:

- ▶ Erste (erfolgreiche) Schritte in Python
- ▶ Andeuten was möglich ist (und wie)
- ▶ Programmieren lernen



Ziele für heute:

- ▶ Erste (erfolgreiche) Schritte in Python
- ▶ Andeuten was möglich ist (und wie)
- ▶ Programmieren lernen

Plattform: Jupyter Notebook (mit Python Kernel)

- ▶ Backend: (lokaler) Webserver; Frontend: interaktives Dokument im Browser
- ▶ Notebooks kombinieren Quellcode, Programm-Ausgaben und Dokumentation (inkl. \LaTeX -Formeln)

Vorbereitungen



1. Software installieren/aktivieren

(Im PC-Pool, mit Ubuntu gebootet: Terminal-Fenster öffnen mit Rechtsklick auf den Desktop bzw. STRG+ALT+T, Text einfügen: STRG+SHIFT+V);

kopier-freundliche Version: siehe <http://fsfw-dresden.de/fsfw-dresden.de/python-ws>

```
Listing: initscript.sh
```

```
#!/bin/bash
```

```
# activate anaconda python distribution
```

```
. /opt/anaconda/bin/activate
```

```
cd $HOME
```

```
PKDIR=pykurs-wise1920
```

```
mkdir -p $PKDIR
```

```
cd $PKDIR
```

```
pip install --user ipyde
```

```
git clone https://github.com/fsfw-dresden/python-ws-notebooks.git notebooks
```

2. Jupyter-Notebook im aktuellen Verzeichnis starten:

```
jupyter notebook ./
```



Important keyboard shortcuts

Command Mode (press Esc to enable)

- ▶ Shift-Return - execute cell, activate next
- ▶ h - show keyboard shortcuts
- ▶ m - change cell type to markdown
- ▶ y - change cell type to code
- ▶ a - new cell above

Edit Mode (press Return to enable)

- ▶ Shift-Return - execute cell, activate next
- ▶ Tab - code-completion or indent
- ▶ Shift-Tab - tooltip
- ▶ Ctrl-Z - undo



Important keyboard shortcuts

Command Mode (press Esc to enable)

- ▶ Shift-Return - execute cell, activate next
- ▶ h - show keyboard shortcuts
- ▶ m - change cell type to markdown
- ▶ y - change cell type to code
- ▶ a - new cell above

Edit Mode (press Return to enable)

- ▶ Shift-Return - execute cell, activate next
- ▶ Tab - code-completion or indent
- ▶ Shift-Tab - tooltip
- ▶ Ctrl-Z - undo

→ Now play around with `demo1.ipynb`



Es folgt: Überblick über Python-Syntax und Datentypen



Es folgt: Hastiger Überblick über Python-Syntax und Datentypen

Numerical Data Types



- ▶ Integer

```
>>> type(1)
<type 'int'>
```

- ▶ floating point number

```
>>> type(1.0)
<type 'float'>
```

- ▶ complex number

```
>>> type(1 + 2j)
<type 'complex'>
```

- ▶ Operations

Addition	+
Subtraction	-
Division	/
Integer division	//
Multiplication	*
Taking powers	**
Modulo	%

- ▶ Built-in functions

- ▶ `round`, `pow`, etc.
- ▶ see `dir(__builtins__)`

- ▶ Module `math`

- ▶ see `help(math)`

NoneType and boolean values



▶ None

- ▶ universal value for “undefined”

```
>>> type(None)
<type 'NoneType'>
```

▶ Boolean values

- ▶ True and False

```
>>> type(True)
<type 'bool'>
```

Data Type	False-Value
NoneType	None
int	0
float	0.0
complex	0 + 0j
str	""
list	[]
tuple	()
dict	{}
set	set()

Operations



Operation	Shortcut
$x = x + y$	$x += y$
$x = x - y$	$x -= y$
$x = x * y$	$x *= y$
$x = x / y$	$x /= y$
$x = x \% y$	$x \% = y$
$x = x ** y$	$x ** = y$
$x = x // y$	$x //= y$

Comparison operations
$x == y$
$x != y$
$x < y$
$x <= y$
$x > y$
$x >= y$

Strings (objects of type str)



```
str1 = "abc"  
str2 = 'xyzabcefg'hi'  
str3 = """  
    multi  
    line  
    string  
    """
```

Escape Sequence	Meaning
<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\"</code>	escaping "
<code>\'</code>	escaping '
<code>\\</code>	escaping \

```
>>> str2[0] # indexing starts at 0  
'x'  
>>> str2[1:4]  
'yza'  
>>> str2[-3:]  
'ghi'
```

String Formating



- ▶ General Syntax

```
"value of x={} and y={}".format(x, y)
```

- ▶ Examples

```
>>> a = 'H'  
>>> b = 'ello World'  
>>> "{}{}{} {}".format(a, b, 5)  
>>> "{}{}{} {}".format(a, b, 5)  
'Hello World'
```

- ▶ Extension (see also: [reference](#))

```
>>> "a={:06.2f} and b={:05.2f}".format(3.007, 42.1)  
'a=003.01 and b=42.10'
```

- ▶ important methods of class

str:

index, replace, split, join,
format, startswith, endswith, ...

Lists



- ▶ Syntax

```
[value_1, ..., value_n]
```

- ▶ Can contain values of any type

- ▶ Can be changed

- ▶ Can be sorted

- ▶ Important methods

```
append, count, index, insert,  
pop, remove, reverse, sort
```

⚠ sort and reverse work „in place“ (return-value: None)

- ▶ Examples

```
>>> m = [7, 8, 9]
>>> n = ['a', 'z', 1, False]
>>> m.append('x')
>>> m[0]
7
>>> m[-1]
'x'
>>> m[:] # start to end
[7, 8, 9, 'x']
>>> m.pop(0)
7
>>> m.reverse()
>>> print(m)
['x', 9, 8]
```

Tuple



- ▶ Syntax
(value_1, ..., value_n)
- ▶ Can **not** be changed
- ▶ → Access much faster than to list
- ▶ Can contain elements of any type
- ▶ important methods
index

▶ Examples

```
>>> t = (7,8,9)
>>> t[0]
7
>>> t[-1]
9
>>> t[:] # start to end
(7,8,9)
>>> z = ('a', 'z', 1, False)
>>> t.index(8)
1
>>> z.index('a')
0
```

Sequential data types



str, tuple, list, (numpy.array)

Operation	Meaning
<code>s in x</code>	tests, whether s is element of x
<code>s not in x</code>	tests, whether s is not element of x
<code>x + y</code>	concatenation of x and y
<code>x * n</code>	concatenation, such that n copies of x exist
<code>x[n]</code>	return the n-th element of x
<code>x[n:m]</code>	return the subs-sequence from index n til m (excluding m)
<code>x[n:m:k]</code>	same with step-size k
<code>len(x)</code>	number of elements
<code>min(x)</code>	minimum
<code>max(x)</code>	maximum

Dictionaries (Associative Arrays)



- ▶ Key-value-pairs
 - ▶ Keys must be immutable objects
 - ▶ Each key can occur only once
- ▶ Syntax

```
{ Key_1: Value_1,  
  Key_2: Value_2,  
  ... }
```
- ▶ Access via
 - ▶ `d.get(key, default)`
or
 - ▶ `d[key]`
- ▶ Important methods
 - ▶ keys, values, items

Examples

```
>>> d = {"Germany": "Berlin", "Peru": "Lima"}
```

```
>>> type(d)  
<type 'dict'>
```

```
>>> e = {1: "a", 2: "b", 400: "c", 1.3: d}  
>>> e[1]  
'a'
```

```
>>> d.get("Germany")  
'Berlin'
```

```
# no entry -> None (no output)  
>>> d.get("Bavarya") # -> None
```

```
# with default value  
>>> d.get("Bavarya", "unknown capital")  
'unknown capital'
```

```
>>> d["Bavaria"]  
KeyError: 'Bavaria'
```

Sets



- ▶ Syntax
`set([element_1, ..., element_n])`
- ▶ Every element is contained only once
- ▶ Has no specified order
- ▶ Can be changed (`frozenset` is immutable)
- ▶ Important methods:
add, remove, union, difference, issubset, issuperset

Examples

```
>>> engineers = set(['Jane', 'John',  
... 'Jack', 'Janice'])  
>>> programmers = set(['Jack', 'Sam',  
... 'Susan', 'Janice'])  
>>> managers = set(['Jane', 'Jack',  
... 'Susan', 'Zack'])  
>>> s1 = engineers.union(programmers)  
>>> s2 = engineers.intersection(managers)  
>>> s3 = managers.difference(engineers)  
>>> engineers.add('Marvin')  
>>> print(engineers)  
set(['Jane', 'Marvin',  
'Janice', 'John', 'Jack'])
```

Data Types - Final Remarks



- ▶ In Python **everything is an object** (even functions, classes, modules)
- Everything has a type: `type(object)`

- ▶ Type checking (→ True or False):
 - ▶ Exact matching: `type("abc") == type("xyz")`
 - ▶ Better: respecting inheritance `isinstance(x, str)`
 - ▶ Allow multiple types:
`isinstance(x, (int, float, complex))`

- ▶ Useful construction:
`assert isinstance(x, int) and x > 0`

Distinction of Cases: if, elif, else



► Syntax

```
# note the indention
if <condition1>:
    ...
elif <condition2>:
    ...
else:
    ...
```

► Examples

```
>>> x = 1
>>> if x == 1:
...     print("x is 1")
...
x is 1
>>> x = 4
>>> if x == 1:
...     print("x is 1")
... elif x == 3:
...     print("x is 3")
... else:
...
print("x is neither 1 nor 3")
x is neither 1 nor 3
```

Iterate over a Sequence: for-loop

► Syntax:

```
for <variable> in <sequence>:  
    ...
```

► easily construct sequences:

► `range`-function → iterator

```
range(stop)  
range(start, stop)  
range(start, stop, step)
```

```
>>> list(range(4))  
[0, 1, 2, 3]
```

```
>>> list(range(1, 10, 2))  
[1, 3, 5, 7, 9]
```

conversion to list only for printing

► Examples:



```
>>> seq = ['a', 'b', 42]  
>>> count = 0  
>>> for elt in seq:  
...     print(elt*2)  
aa  
bb  
84  
  
>>> for i in range(3):  
...     print(2**i)  
1  
2  
4
```

Loop while condition is true



▶ Syntax

```
while <condition>:  
    ...
```

▶ break

terminates the loop

```
while <condition1>:  
    if <condition2>:  
        break
```

▶ continue

immediately starts next cycle

```
while <condition1>:  
    if <condition2>:  
        continue
```

▶ Examples

```
>>> x = 4  
>>> while x > 1:  
...     print(x)  
...     x -= 1  
...     print("finished")  
4  
3  
2  
finished
```

Functions

▶ Syntax

```
def func_name(Param_1, ..., Param_n):  
    ...  
    return <result>
```

▶ No explicit return-value → None

▶ Empty function with keyword pass:

```
def empty():  
    pass
```

▶ default values for optional parameters

```
def test(x=23):  
    print(param)
```

▶ Arbitrary number of arguments

```
def func(*args, **kwargs):  
    print(type(args)) # -> tuple  
    print(type(kwargs)) # -> dict
```

▶ Examples



```
>>> def print_sum(a, b):  
...     print(a + b)  
>>> print_sum(1, 2)  
3  
>>> def print_prod(a, b, c=0):  
...     print(a*b + c)  
>>> print_prod(2, 4)  
8
```

```
# better readable  
>>> print_prod(a=2, b=4)  
8  
>>> print_prod(2, 4, 1)  
9  
>>> print_prod(c=2, a=4, b=1)  
6
```

Local Variables (Scopes)



Listing: local-variables.py

```
def square(z):
    x = z**2 # x: local variable
    print(x)
    return x

x, a = 5, 3 # "unpacking" a tuple

square(a) # -> 9
square(x) # -> 25
print(x) # -> 5 (not changed)

def square2(z):
    print(x) # here: x is taken from global scope
    return z**2

def square3(z):
    print(x) # Error (local variable not yet known)
    x = z**2 # x is local variable due to write access
    return x
```


General Syntax



- ▶ Semantic blocks are defined by **indentation level** (instead of, e. g., { ... })
 - ▶ defacto-standard: 4 spaces per level (do not use TABs)
 - ▶ every good text editor can be configured adequately (spyder: TAB indentation, SHIFT+TAB dedention of highlighted lines)

- ▶ Comments and docstrings:

single line comments begin with a hash

```
def my_function(x, y):  
    """This is a docstring.  
    It can span multiple lines  
    """  
  
    """unassigned multi-line strings can  
    be abused as multi-line comment  
    """
```

- ▶ Recommended maximum line length 80 (or 100) characters (readability)
- ▶ If you need more:
 - ▶ Check possibility to split up into two commands (readability)
 - ▶ Within braces newlines are ignored
 - ▶ Backslash (\) allows line continuation in expression

Keywords (Reserved words)



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

They cannot be used as variable name or similar.

File Access



Listing: file-access.py

```
# write in text mode
content_lines = ['some\n', 'more', 'content']
with open('text.txt', 'w') as myfile:
    myfile.write('Hello World.')
    myfile.writelines(content_lines)
    # myfile.close() is called automatically
    # when leaving this block

# read in text mode
with open('text.txt', 'r') as myfile:
    header = myfile.read(10) # first 10 byte
    lines = myfile.readlines() # list of lines
    # (starting from file cursor)
```

File Access



Listing: file-access.py

```
# write in text mode
content_lines = ['some\n', 'more', 'content']
with open('text.txt', 'w') as myfile:
    myfile.write('Hello World.')
    myfile.writelines(content_lines)
    # myfile.close() is called automatically
    # when leaving this block

# read in text mode
with open('text.txt', 'r') as myfile:
    header = myfile.read(10) # first 10 byte
    lines = myfile.readlines() # list of lines
    # (starting from file cursor)
```

Read/write binary data: use 'rb' and 'wb'

Appending text or binary data: use 'a' or 'ab'

Some “specialities” of Python



- ▶ Indexing starts with 0
- ▶ Unpacking of sequential data types:

```
>>> x, y, z = range(3)
```

```
>>> y
```

```
1
```

```
>>> mapping = [('green', 560), ('red', 700)]
```

```
>>> for color, wavelength in mapping:
```

```
...     pass
```

```
...     # do stuff
```

- ▶ ∃ extensive standard library („batteries included“)
 - ▶ <http://docs.python.org/3/library/>
 - “Don’t reinvent the wheel!”
- ▶ Important modules:
pickle, sys, os, itertools, unittest, ...

Quellen und Links (Auswahl)



- ▶ Offizielles Tutorial: <http://docs.python.org/3/tutorial/>
- ▶ Interaktives Tutorial: <http://www.learnpython.org/>
- ▶ Ausführlicher gut strukturierter Kurs:
<http://www.diveintopython3.net/>

Offizielle Doku zu wissenschaftlichen Paketen:

- ▶ <http://docs.sympy.org/latest/modules/>
- ▶ <https://docs.scipy.org/doc/numpy-1.13.0/reference/>
- ▶ <https://docs.scipy.org/doc/scipy/reference/>
- ▶ <https://matplotlib.org/contents.html>

Auch hilfreich: google, stackoverflow, ...



- ▶ Fragen?
- ▶ **Unterstützung:** (im Rahmen unserer Möglichkeiten)
 - ▶ <https://fsfw-dresden.de/sprechstunde>
 - ▶ <https://fsfw-dresden.de/python-workshop>
 - ▶ kontakt@fsfw-dresden.de

Projekt Uni-Stick (4. Auflage)



- ▶ **3000** Flyer in Ersti-Tüten: **Gutscheine** für 8 GB Stick mit Freier Software fürs Studium,
- ▶ Live-Linux / freie Windows-Programme
- ▶ Vorbereitung hat Arbeit und Spaß gemacht
- ▶ Ist gut angekommen (ca. 250 TN)



Ausgabe-Veranstaltung 2019
04.11., 18:30 Uhr, **APB E023**



Bits und Bäume



- ▶ Digitalisierung und Nachhaltigkeit verbinden
- ▶ Ressourcen, Freie Software, Privatsphäre, Demokratie, ...
- ▶ Größeres Treffen: 23.10., 19:00 Uhr, [Rosenwerk](#)
- ▶ <https://dresden.bits-und-baeume.org/>

Schlussbemerkungen (3): Blick über den Tellerrand



- ▶ TU-Umweltinitiative
 - ▶ Umweltringvorlesungen (Credit Points)
 - ▶ Offenes Grillen/Essen: 24.10.
 - ▶ Wandern: 02.11.
 - ▶ <https://tuuwi.de/>



Schlussbemerkungen (3): Blick über den Tellerrand



- ▶ TU-Umweltinitiative
 - ▶ Umweltringvorlesungen (Credit Points)
 - ▶ Offenes Grillen/Essen: 24.10.
 - ▶ Wandern: 02.11.
 - ▶ <https://tuuwi.de/>
- ▶ Students4Future:

Klimakrise, was tun?

Sei dabei und kämpfe für (d)eine nachhaltige Zukunft mit uns gemeinsam!

SEMESTERAUFTAKT

FREITAG, 25.10.

16:30 UHR IM BIO-BAU

(ZELLESCHER WEG 20B)



→ <https://ffdd.de>