



Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

FS24 CAS PML - Python

Niklaus Johner

niklausbernhard.johner@bfh.ch

Über mich

Niklaus Johner: niklausbernhard.johner@bfh.ch



- ▶ 1999 - 2004: Master in Physik, EPFL
- ▶ 2004 - 2009: Doktorarbeit in Physik, EPFL
- ▶ 2009 - 2010: Postdoc, Comput. Biophysics, UniBasel
- ▶ 2011 - 2014: Researcher, Comput. Biophysics, WCMC NY
- ▶ 2014 - 2017: Postdoc, Comput. Biophysics, UniBasel
- ▶ 2017-2023: Software developer, 4teamwork, BE
- ▶ Seit 2023: Researcher, Bioinformatik, CHUV

Kurs format

- ▶ Alternieren zwischen:
 - ▶ Kurze Präsentationen
 - ▶ Übungen
- ▶ 1.5 - 2 Tage Python Programmierung
- ▶ Dann Externe Module:
 - ▶ Dateien lesen/schreiben
 - ▶ Datenhandlung und Datenanalyse
- ▶ Kurs ist auch auf GitHub Verfügbar: https://github.com/njohnner/python_course

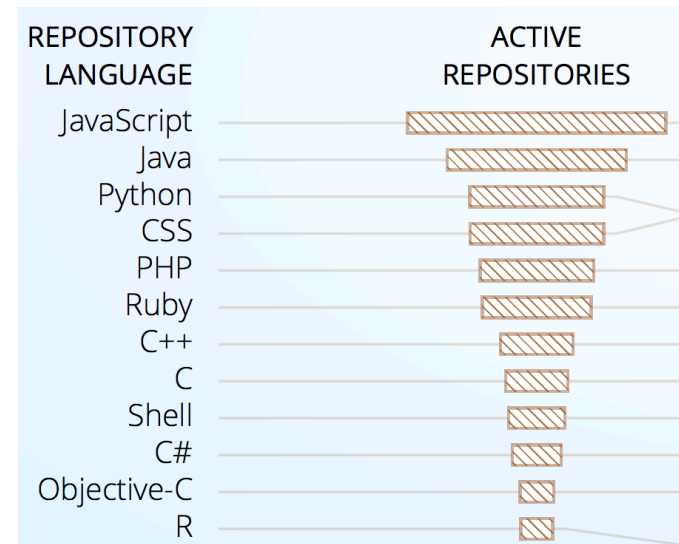
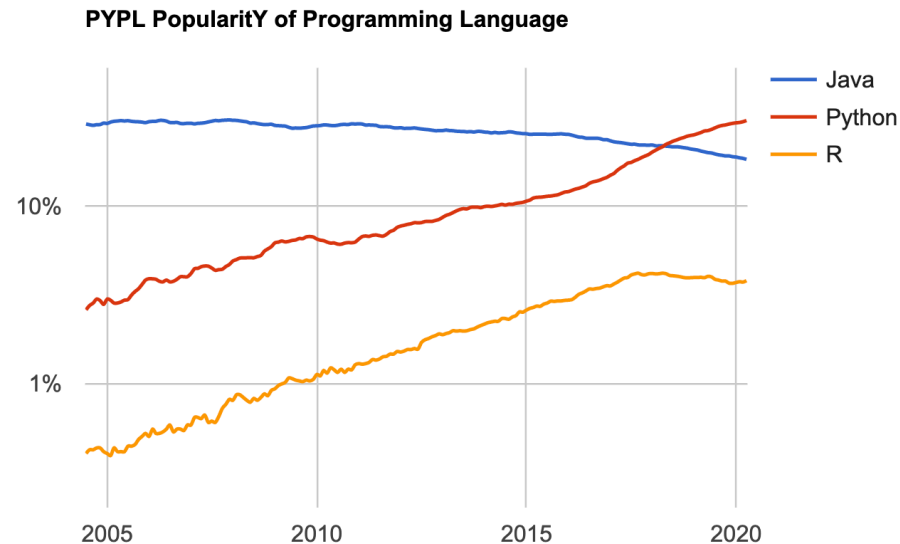
FS24 CAS PML - Python

1. Einführung

Warum Python?

- ▶ Interpretierte Sprache
 - ▶ Kompilation nicht notwendig
 - ▶ Interaktiv Programmieren
- ▶ Aktiv benutzt und weiter entwickelt
- ▶ Free und open-source
- ▶ Aktiv für Forschung und Data science verwendet

Number of tutorial searches on google



<http://github.info>

Python Features

- ▶ High-Level Programmiersprache:
 - ▶ Kein Memory oder Garbage Management
- ▶ Objekt-orientiert
- ▶ Lesbare und einfache Syntax:
 - ▶ Keine formalen Definitionen der Variablen

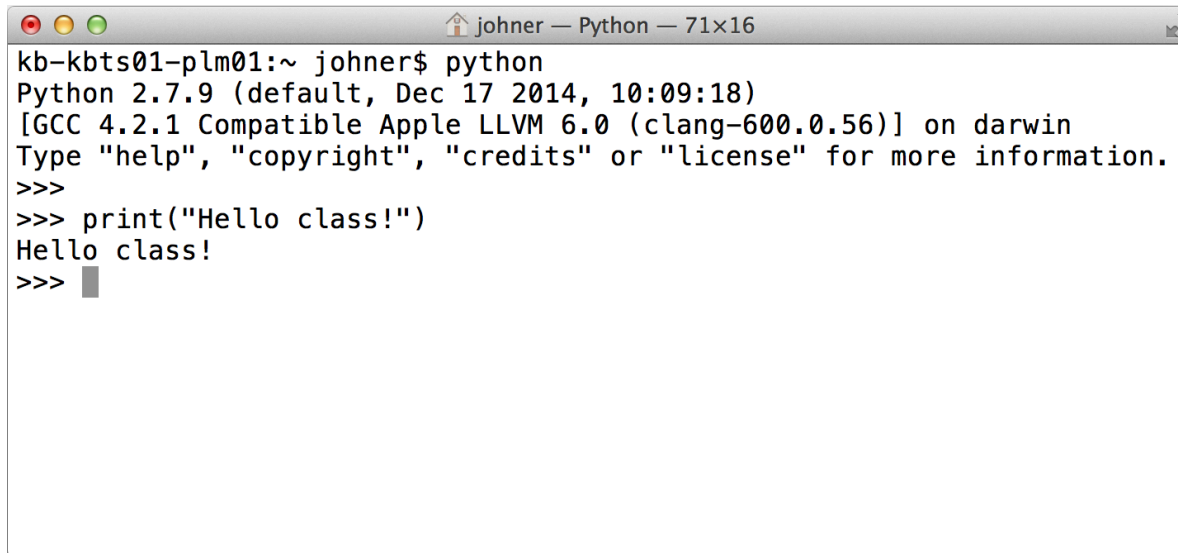
a = 1

b = 3

total = a + b

Python starten

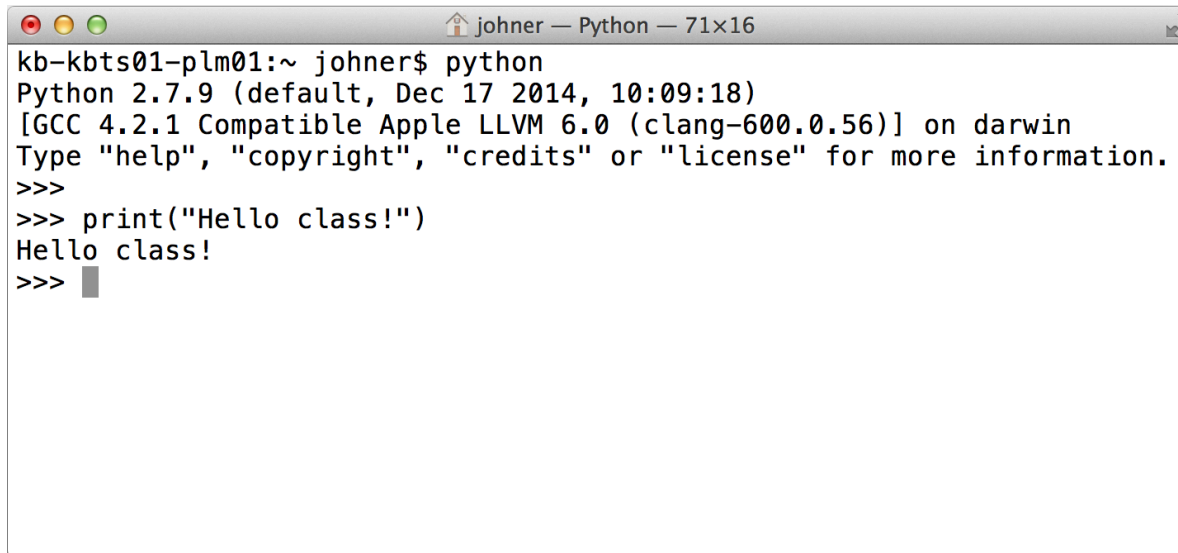
- ▶ Man kann den Python Interpreter einfach von einem Shell-Fenster starten
- ▶ Dann kann man interaktiv programmieren



```
kb-kbts01-plm01:~ johner$ python
Python 2.7.9 (default, Dec 17 2014, 10:09:18)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print("Hello class!")
Hello class!
>>> █
```

Python Interaktives Modus

- ▶ Enter-Taste nach einem kompletten Statement:
 - ▶ Das Statement wird interpretiert und ausgeführt
 - ▶ Das Resultat wird ausgedruckt
 - ▶ Das nennt man die Read-Eval-Print-Loop (REPL)

A screenshot of a terminal window titled 'johner — Python — 71x16'. The terminal shows the command 'python' being executed, which starts the Python 2.7.9 interpreter. It displays version information and the path 'kb-kbts01-plm01:~ johner\$'. The prompt '>>>' is shown, followed by the command 'print("Hello class!")' and its output 'Hello class!'. The prompt '>>>' is shown again with a cursor, indicating the interactive loop.

```
kb-kbts01-plm01:~ johner$ python
Python 2.7.9 (default, Dec 17 2014, 10:09:18)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print("Hello class!")
Hello class!
>>> █
```


FS24 CAS PML - Python

2. Grundlagen

Python als Rechner

► Mathematische Operatoren in Python

- Addition: +
- Subtraktion: -
- Multiplikation: *
- Division: /
- Potenz: **
- Ganzzahldivision: //
- Modulo: %

```
In [2]: 1 + 2
```

```
Out[2]: 3
```

```
In [3]: 2 * 3
```

```
Out[3]: 6
```

```
In [4]: 3 / 2
```

```
Out[4]: 1.5
```

```
In [5]: 3 // 2
```

```
Out[5]: 1
```

```
In [6]: 3 % 2
```

```
Out[6]: 1
```

```
In [7]: 3 ** 2
```

```
Out[7]: 9
```

Python als Rechner

- ▶ Anweisungen werden von links nach rechts evaluiert
- ▶ Operatoren Präzedenz
 - ▶ +, -
 - ▶ *, /, //, %
 - ▶ **
- ▶ Klammer zum Gruppieren
 - ▶ Klammern werden zuerst evaluiert
 - ▶ Dann die ganze Anweisung

```
[In [2]: 2 + 1 * 3  
Out[2]: 5
```

```
[In [3]: (2 + 1) * 3  
Out[3]: 9
```

```
[In [4]: 2 * 3 ** 2  
Out[4]: 18
```

```
[In [5]: (2 * 3) ** 2  
Out[5]: 36
```

Assignment Operator

- ▶ Assignment Operator: =
- ▶ *variable_name = value*
 - ▶ Kreiert eine Variable *variable_name*
 - ▶ Weist ihr den Wert *value* zu

```
In [9]: price = 15
...: tax_rate = 0.14
...: total = price*(1 + tax_rate)
```

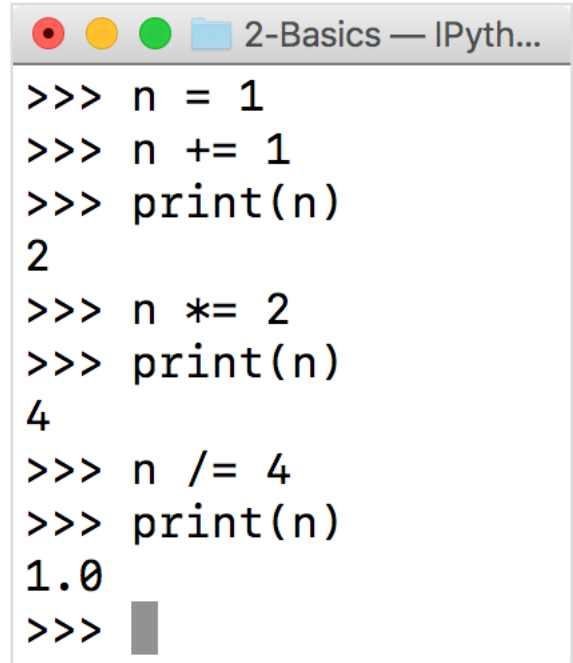
```
[In [10]: print(price)
15
```

```
[In [11]: print(total)
17.1
```

Compound Operatoren

- ▶ Compound Operatoren sind Abkürzungen um Variablen zu modifizieren:

```
var += val # var = var + val
var -= val # var = var - val
var *= val # var = var * val
var /= val # var = var / val
var **= val # var = var ** val
```



```
>>> n = 1
>>> n += 1
>>> print(n)
2
>>> n *= 2
>>> print(n)
4
>>> n /= 4
>>> print(n)
1.0
>>> █
```

Syntax: Kommentaren

- ▶ Alles was auf einer Zeile nach `#` kommt ist ein Kommentar
- ▶ `"""` Delimitiert den Anfang und Schluss eines Kommentars der mehrere Zeilen lang sein kann

```
31  # This is a comment
32  n = 1 #this is also a comment
33  """
34  This is a
35  multiline comment
36  """
```

Definitionen

- ▶ Eine Funktion: ausführbares Objekt
- ▶ Eine Klasse: ein Objekttyp (Objektvorlage)
- ▶ Eine Instanz: Ein konkretes Objekt einer gewissen Klasse
- ▶ Eine Methode: Funktion, die zu einem Objekt gehört
- ▶ Ein Attribut: sonstiges Objekt, dass zu einem Objekt gehört
- ▶ Argument: Mitgegebene Werte beim aufrufen einer Funktion

```
[In [1]: print("Hello") #function  
Hello
```

```
[In [2]: complex # Class  
Out[2]: complex
```

```
[In [3]: n = complex(1, 2) #instance
```

```
[In [4]: n.real #Attribute  
Out[4]: 1.0
```

```
[In [5]: n.conjugate() #Method  
Out[5]: (1-2j)
```

Syntax: Funktionen, Methoden und Attributen

- ▶ Funktionen werden mit Klammern aufgerufen:
 - ▶ In den Klammern gibt man die Argumente an
- ▶ Ein Attribut *attr* eines Objekts *obj* wird mit der Syntax *obj.attr* geholt
- ▶ Eine Methode *meth* eines Objekts *obj* wird mit der Syntax *obj.meth()* aufgerufen

```
# Object examples:  
print("Hello")    # function  
n = 1 + j          # Complex  
n.real            # Attribute  
n.conjugate()     # Method
```


Syntax: Literal und Klass Instanziierung

- ▶ Eine Klasse wird instanziiert indem man sie wie eine Funktion aufruft.
 - ▶ Instanziierung gibt ein Instanz Objekt von dieser Klasse zurück
- ▶ Basis Klassen können auch mit einem Literal instanziiert werden.

```
str          # class  
str(1)       # instantiation  
"1"         # literal
```

```
complex      # class  
complex(1, 2) # instantiation  
1 + 2j       # literal
```

Syntax: Literal und Klass Instanziierung

- ▶ Eine Klasse wird instanziiert indem man sie wie eine Funktion aufruft.
 - ▶ Instanziierung gibt ein Instanz Objekt von dieser Klasse zurück
- ▶ Basis Klassen können auch mit einem Literal instanziiert werden.

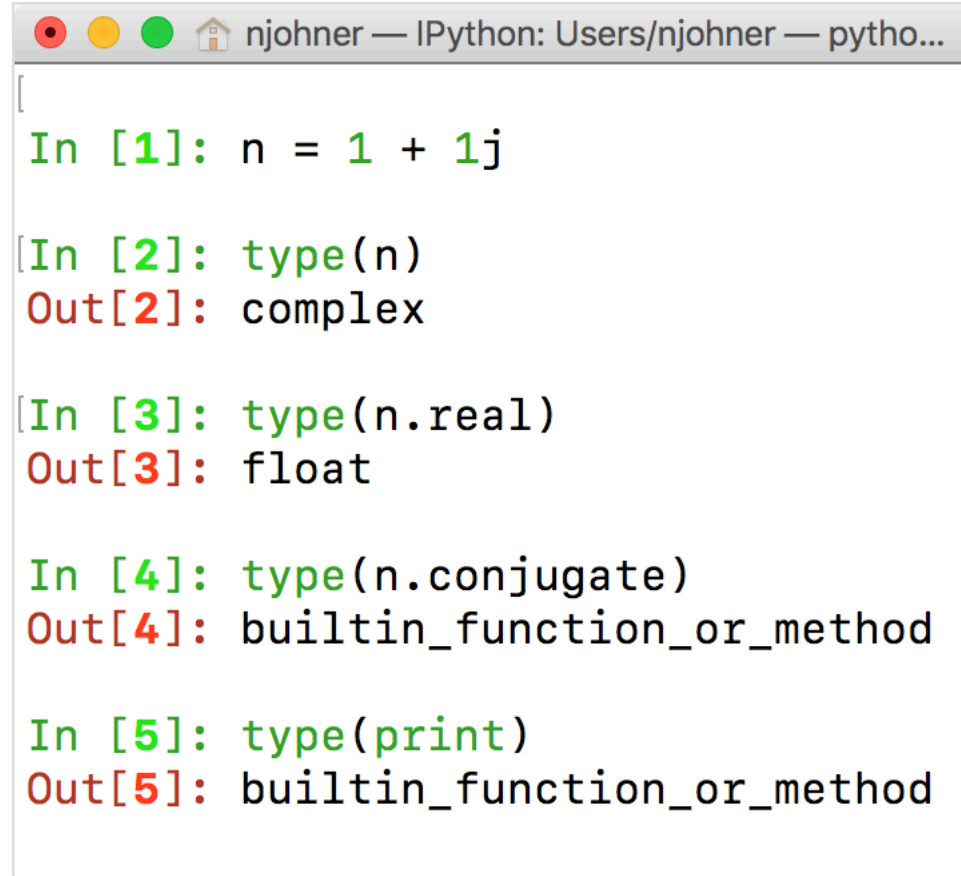
Das kreierte Objekt wird der Variable *message* zugewiesen

Ein *str* Objekt wird vom Literal instanziiert

```
message = "1"  
message = str(1)
```

Nützliche Funktionen

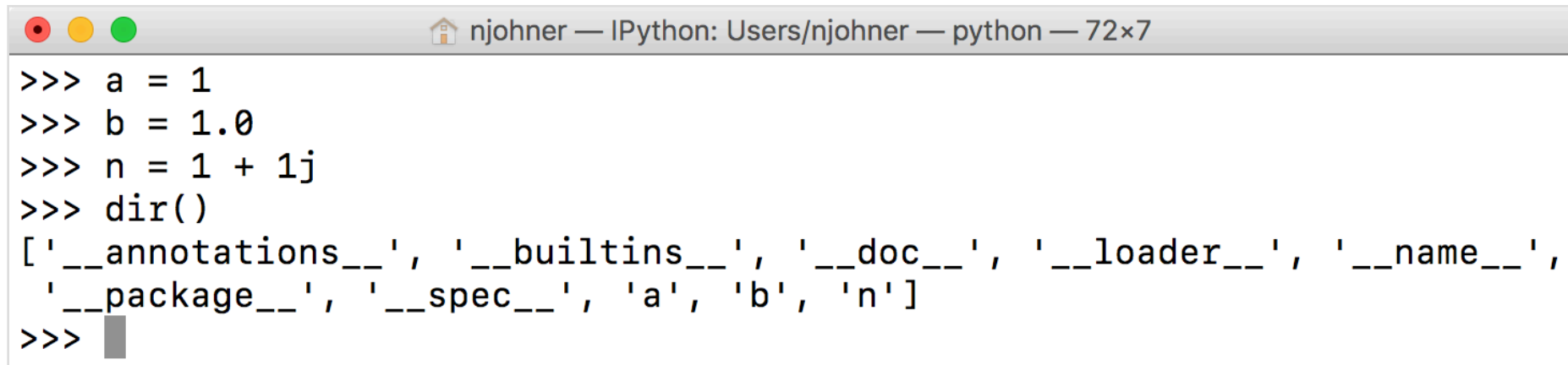
- ▶ ***print(obj)***: Printet das Objekt *obj* im default output (normalerweise die Konsole)
- ▶ ***type(obj)***: Gibt die Klasse von *obj* zurück
- ▶ ***help(obj)***: Zeigt die Hilfe für *obj* an



```
njohnner — IPython: Users/njohnner — pytho...  
[  
In [1]: n = 1 + 1j  
[In [2]: type(n)  
Out[2]: complex  
[In [3]: type(n.real)  
Out[3]: float  
  
In [4]: type(n.conjugate)  
Out[4]: builtin_function_or_method  
  
In [5]: type(print)  
Out[5]: builtin_function_or_method  
]
```

Nützliche Funktionen

- ▶ ***dir()***: Listet alle Namen im namespace
- ▶ ***dir(obj)***: Listet alle Attribute und Methoden von *obj*.

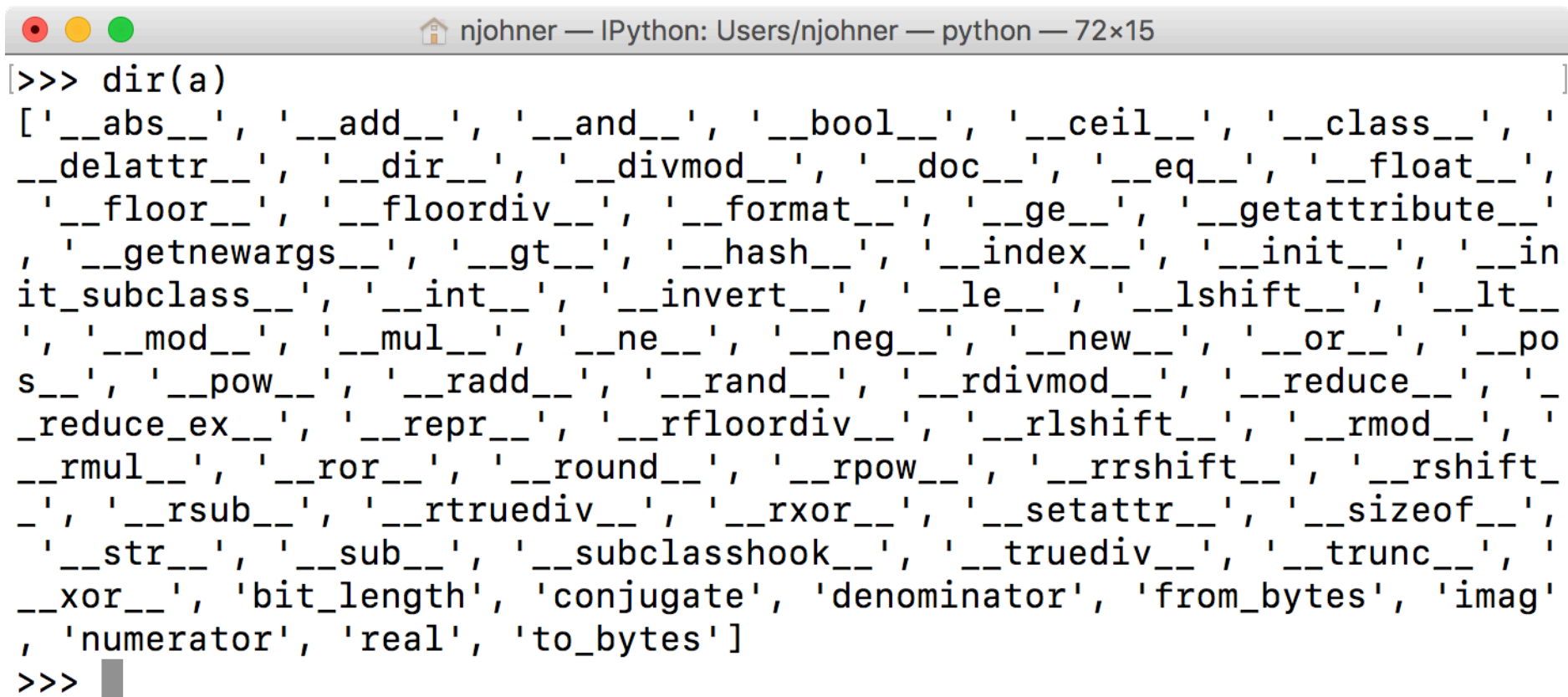


```
njohnner — IPython: Users/njohnner — python — 72x7
>>> a = 1
>>> b = 1.0
>>> n = 1 + 1j
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'a', 'b', 'n']
>>>
```

- ▶ Note: Funktionen Namen die mit `__` anfangen und enden sind Magische Funktionen oder *dunder* Funktionen

Nützliche Funktionen

- ▶ ***dir()***: Listet alle Namen im Namespace auf
- ▶ ***dir(obj)***: Listet alle Attributen und Methoden von *obj* auf.



```
njohnner — IPython: Users/njohnner — python — 72x15
[>>> dir(a)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
 '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
 '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
 '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__in
it_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__
', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__po
s__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '_
reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '_
rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__
', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__',
 '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '_
__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag'
, 'numerator', 'real', 'to_bytes']
>>>
```

Reminders

- ▶ Assignment operator: `=`
- ▶ to get help on an object: `help(obj)`
- ▶ to list all attributes and methods of an object: `dir(obj)`
- ▶ to print something: `print(obj)`