# Konzeptstudie: Python als opsi-Skriptsprache

## opsi-script Funktionalität als Python-Paket/Binary

Stand: 10.07.2024

Jan Werner


https://i0.wp.com/blog.knoldus.com/wp-content/uploads/2019/08/installing-linux-apps-package.png?w=640&ssl=1

# Warum opsi-script als Python-Paket?


# Warum opsi-script als Python-Paket?

opsi-script

Eigene, einfache Skriptsprache,
Entwickelt für System-Administratoren,
Zugeschnitten auf die Anforderungen von Softwareinstallationen,
Plattformunabhängig (Windows, Linux, macOS),
Offen

# Warum opsi-script als Python-Paket?

opsi-script

- Aber:
  - 20 Jahre alter gewachsener Quellcode
  - In Pascal
    - Kleine Community
    - Häufig veraltete/unvollständige Bibliotheken
    - Zukunft(?)

# Warum opsi-script als Python-Paket?


**Python ist leistungsstark... und schnell,**
**kann gut mit anderen,**
**läuft überall,**
**ist benutzerfreundlich und leicht zu erlernen,**
**ist offen.**

# Warum opsi-script als Python-Paket?


- Moderne, high-level Programmier/Skriptsprache
- Große Community
- Viele, meist gut gewartete, Bibliotheken
- Zukunftssicher

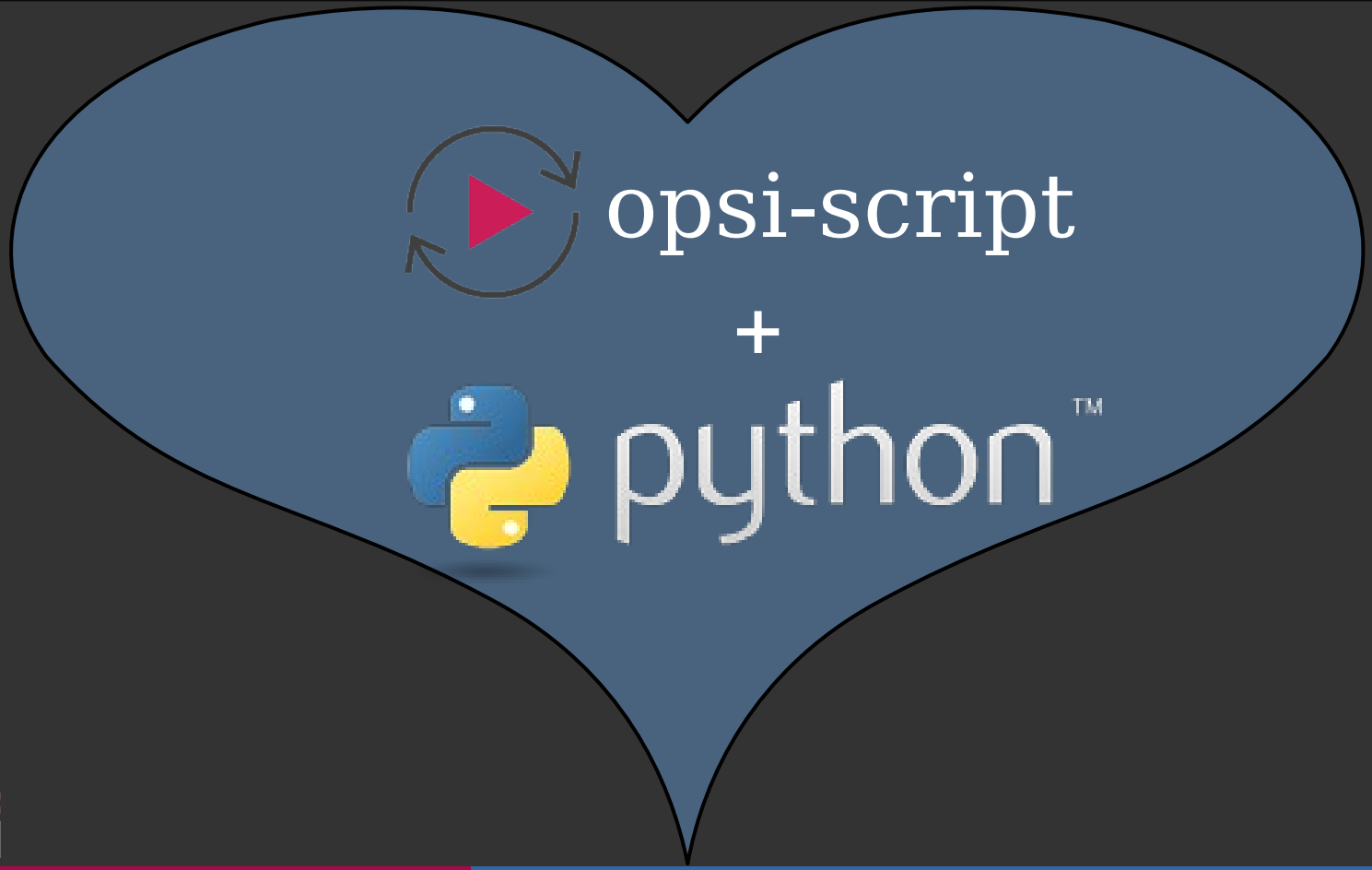
# Warum opsi-script als Python-Paket?

opsi-script

+

python™

# Warum opsi-script als Python-Paket?



# Ziele

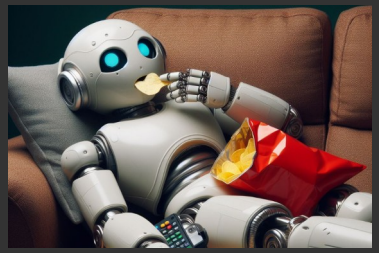
- opsi-script **zusätzlich** als Python-Paket/Binary:
  - Skripte in Python
  - opsi-script Funktionalität durch opsi-eigene Python-Module
  - Python-Objekte die Funktionalität in einem bestimmten Kontext kapseln
    - Leicht zu bedienende Schnittstellen
    - Für Systemadministratoren

# Ziele

- opsi-script **zusätzlich** als Python-Paket/Binary:
  - Skripte in Python
  - opsi-script Funktionalität durch opsi-eigene Python-Module
  - Python-Objekte die Funktionalität in einem bestimmten Kontext kapseln
    - Leicht zu bedienende Schnittstellen
    - Für Systemadministratoren

# Warum opsi-script als Python-Paket?

- gravierende Vorteile für:
  - Entwickler*innen



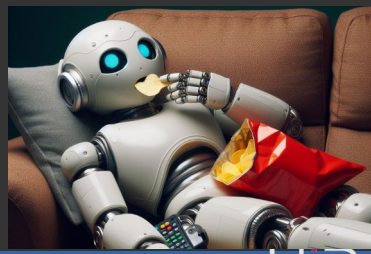
https://techrush.de/wp-content/uploads/2023/12/Blog-3-2.png

  - Anwender*innen

https://www.magazin-schule.de/wp-content/uploads/2023/11/Generation-Z-nicht-faul_Magazin-SCHULE.jpg

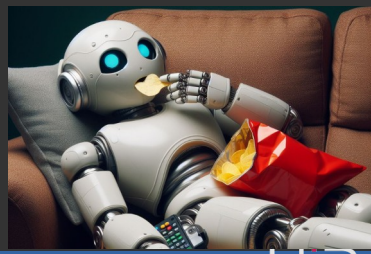
# Warum opsi-script als Python-Paket?

- Vorteile Entwickler*innen:
  - keine Pflege eines eigenen Parsers/Interpreters
  - Python-Entwicklungsumgebung/Know-How vorhanden
  - Synergyeffekte
  - Geschwindigkeitsgewinn bei der Entwicklung



# Warum opsi-script als Python-Paket?

- Vorteile Entwickler*innen:
  - Moderne gut gewartete Bibliotheken
  - Neue Features, die sonst nicht umsetzbar wären
  - Vereinfachte Dokumentation
  - Zukunftssicher



# Warum opsi-script als Python-Paket?

- Vorteile Anwender*innen:
  - Keine extra Skriptsprache lernen
  - Einfach
    - Bei gleicher Funktionalität, Komplexität ähnlich wie ein opsi-script Skript
  - Mächtig
    - Volle Python-Funktionalität, wenn gewünscht/nötig

# Warum opsi-script als Python-Paket?

- Vorteile Anwender*innen:
  - Python-Tools
  - Code-Editoren
    - Syntax-Highlighting
    - Code-Vervollständigung,
    - Syntax-Check
    - Linter (Code-Analyse)

# Schnittstellen für den System-Administrator

# Schnittstellen für den System-Administrator

- Zwei Arten von Schnittstellen
  - Eine für einfache Skript-Erstellung (Easy)
  - Eine für komplexe Skripte (Expert)
- Können kombiniert werden

# Schnittstelle Expert

# Schnittstelle Expert

- Low-Level Schnittstelle
  - Kann mit „Easy“ kombiniert werden
  - Volle Python-Funktionalität wenn benötigt
  - Komplexe Installationen/Aufgaben
  - Fortgeschrittene Paketierer*innen

# Schnittstelle Expert

- Skripten wie in opsi-script
  - Python-Code
  - mit Python-Objekten die Funktionalität in einem bestimmten Kontext kapseln

# Vergleich Python-Code mit opsi-script-Code

# Variabeln

Python:

```
variable = "Value"



variable = "NewValue"
```

opsi-script:

```
DefVar $variable$ = "Value"
;or
DefVar $variable$
Set $variable$ = "Value"

Set $variable$ = "NewValue"
```

# Konstanten

Python:

```
system.program_files_dir
system.system_drive
system.profile_dir

script.path
```

opsi-script:

```
%ProgramFilesDir%
%Systemdrive%
%ProfileDir%

%ScriptPath%
```

# If-Bedingung

Python:

```
if (platform == "macOS"):
        ...
elif (platform == "Linux"):
        ...
elif (platform == "Windows"):
        ...
else:
        ...
```

opsi-script:

```
if ($Platform$ = "macOS")
    ...
elseif ($Platform$ = "Linux")
    ...
elseif ($Platform$ = "Windows_NT")
    ...
else
    ...
endif
```

# Selbst definierte Funktionen

Python:

```
def inc_value(value: int) -> int:
    value = value + 1
    return value
```

opsi-script:

```
DefFunc IncValue($value$ : string) : string
    $value$ = calculate($value$ + "1")
    set $result$ = $value$
endfunc
```

# Zugriff auf Funktionalität über Python-Objekte

# Python-Objekte

- installer
- system
- shell
- product
- registry
- file
- xml
- service
- ...

# Installer

```
installer.execute(path=f"{script.path}/setup.exe")
```

```
installer.output
```

```
installer.exitcode
```

# Shell

```
shell.run("echo 'hello'")

shell.run(f"""
@echo off
Hello {name}
""")

shell.output

shell.exitcode
```

# Registry: Regedit-Format

```
registry.regedit(
    """
    Windows Registry Editor Version 5.00

    [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org]

    [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general]
    "bootmode"="BKSTD"
    "windomain"=""
    "opsiconf"=dword:00000001
    """)
```

# Registry: Regedit-Format

```
registry.regedit(
    f"""
    Windows Registry Editor Version 5.00

    [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org]

    [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general]
    "bootmode"="BKSTD"
    "windomain"="{domain}"
    "opsiconf"=dword:00000001
    """)
```

# Registry-“Sektion“

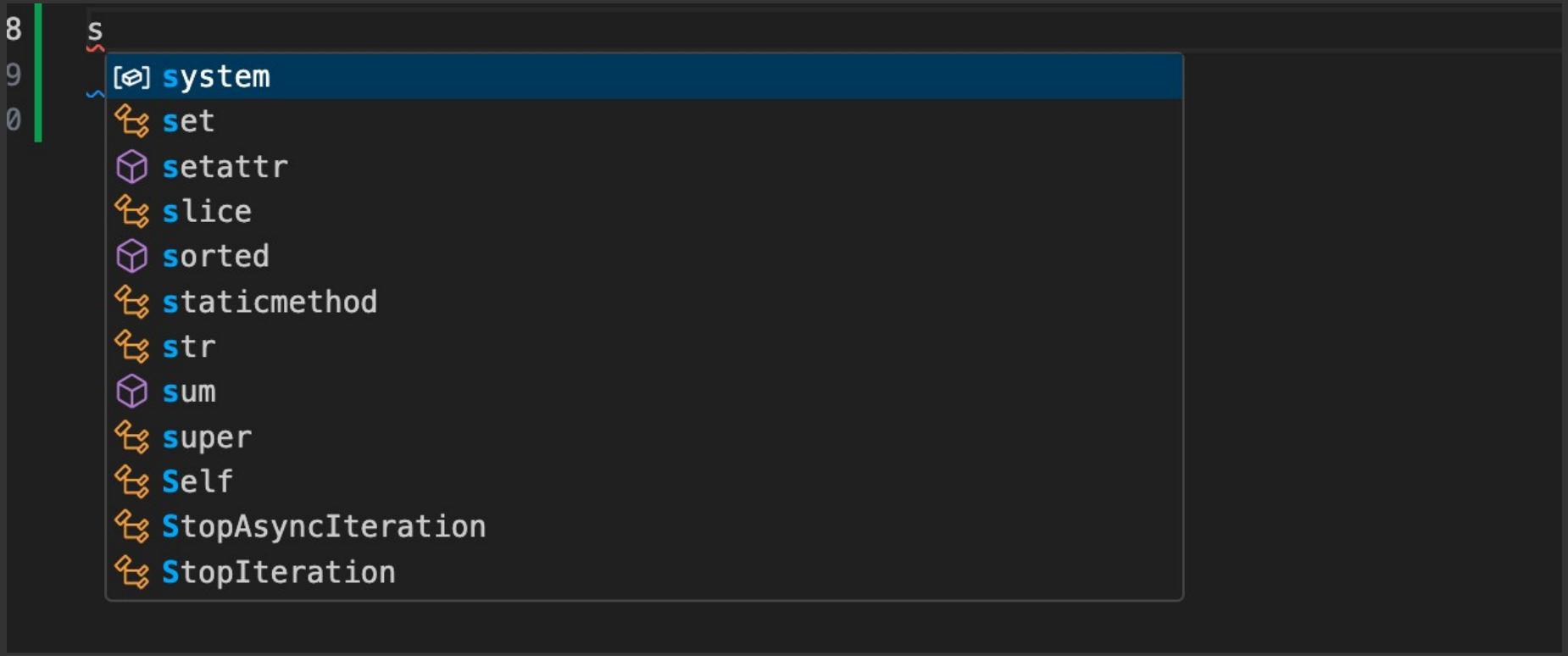
```
with registry.open("HKLM\Sofware\opsi") as reg_opsi:
    reg_opsi.set(key="bootmode", value="BKSTD")
    reg_opsi.set(key="opsiconf", value="dword:1")
    reg_opsi.delete(key="windomain")
```

# “Sektions“-Objekte

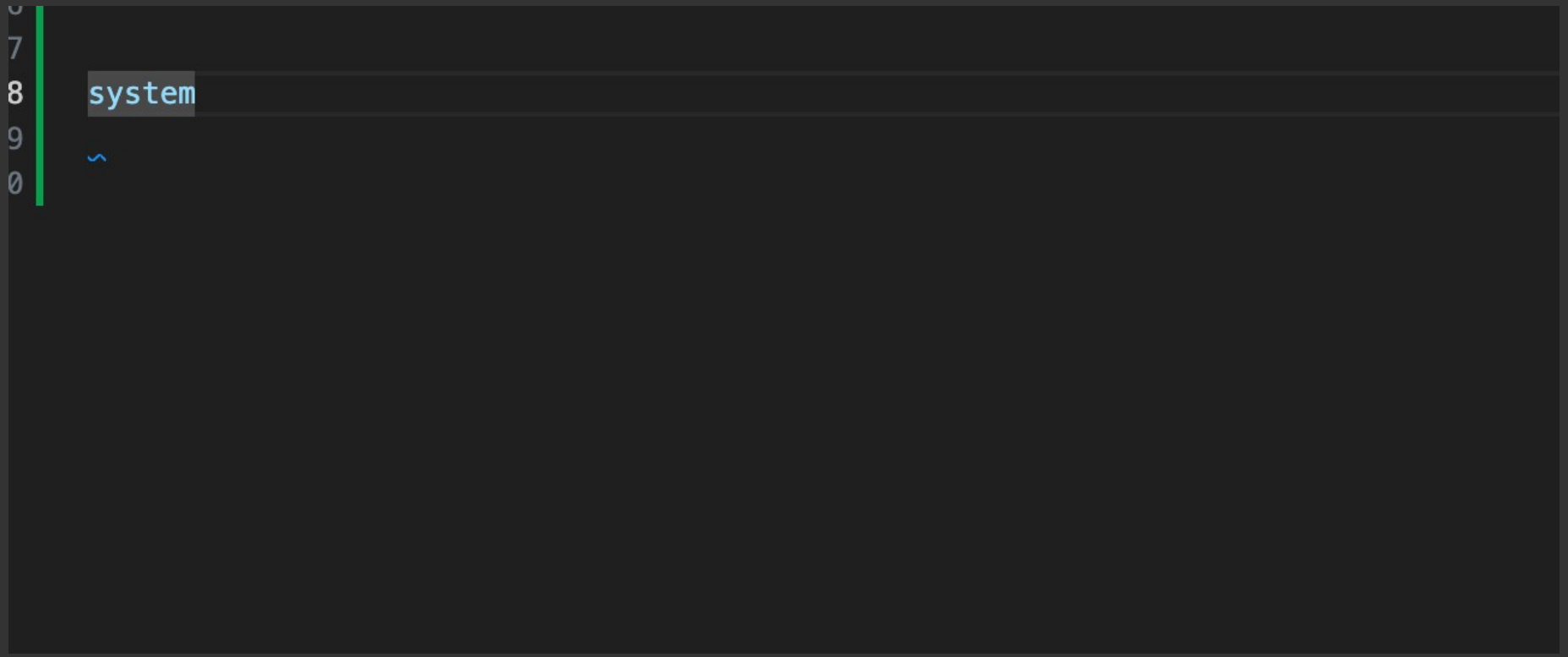
```
with registry.open("HKLM\Sofware\opsi") as reg_opsi:
    reg_opsi.set(key="bootmode", value="BKSTD")
    reg_opsi.set(key="opsiconf", value="dword:1")
    reg_opsi.delete(key="windomain")


with xml.open("config.xml") as config_xml:
    config_xml.set_node(node="greeting", value="Hello")
    config_xml.set_attribute(node="greeting", attribute="color", value="green")
```

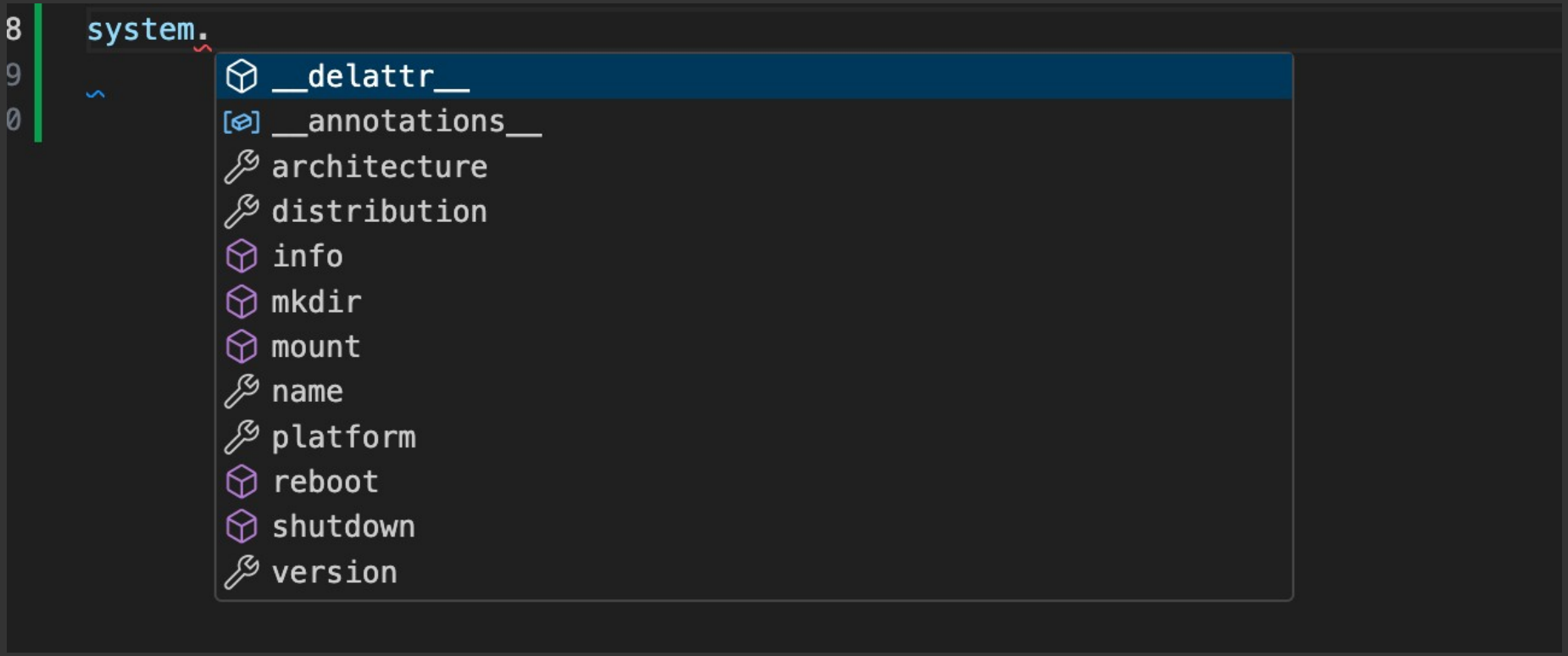
# Code-Autovervollständigung



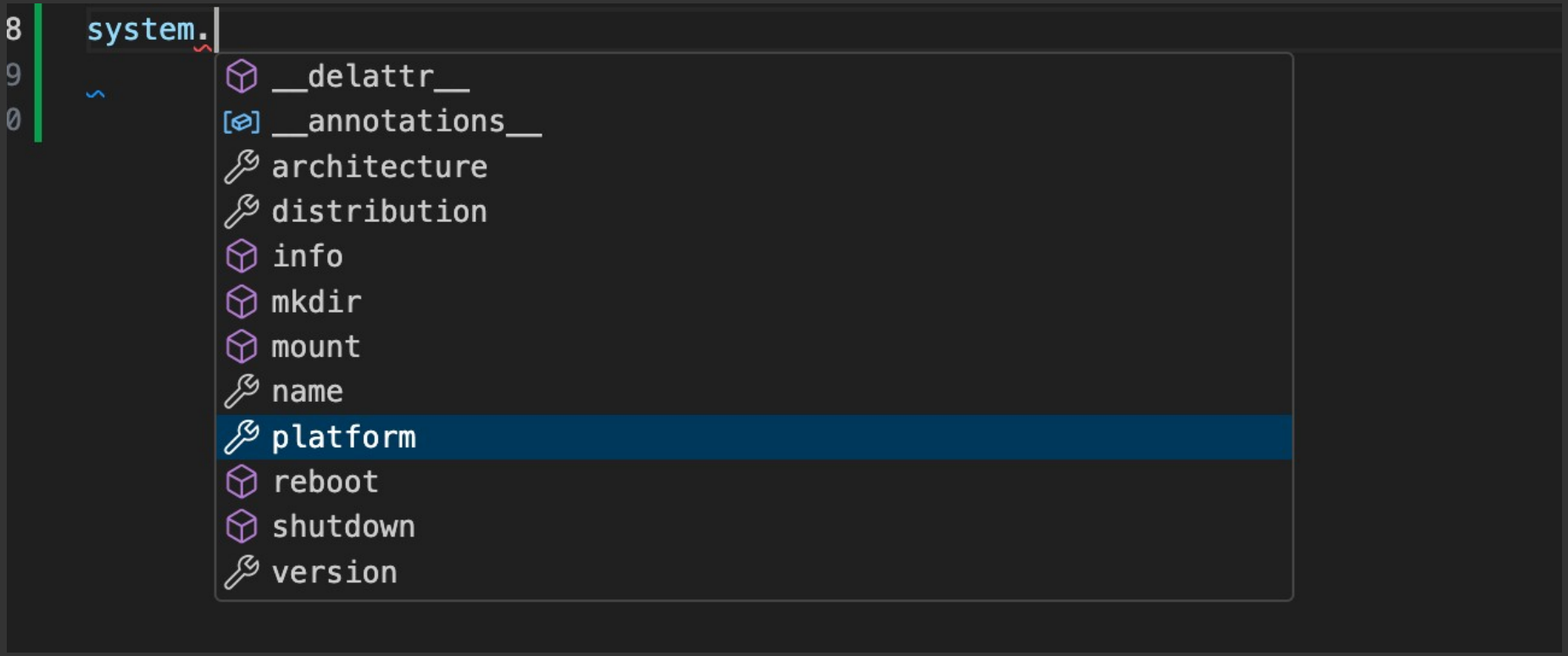
# Code-Autovervollständigung



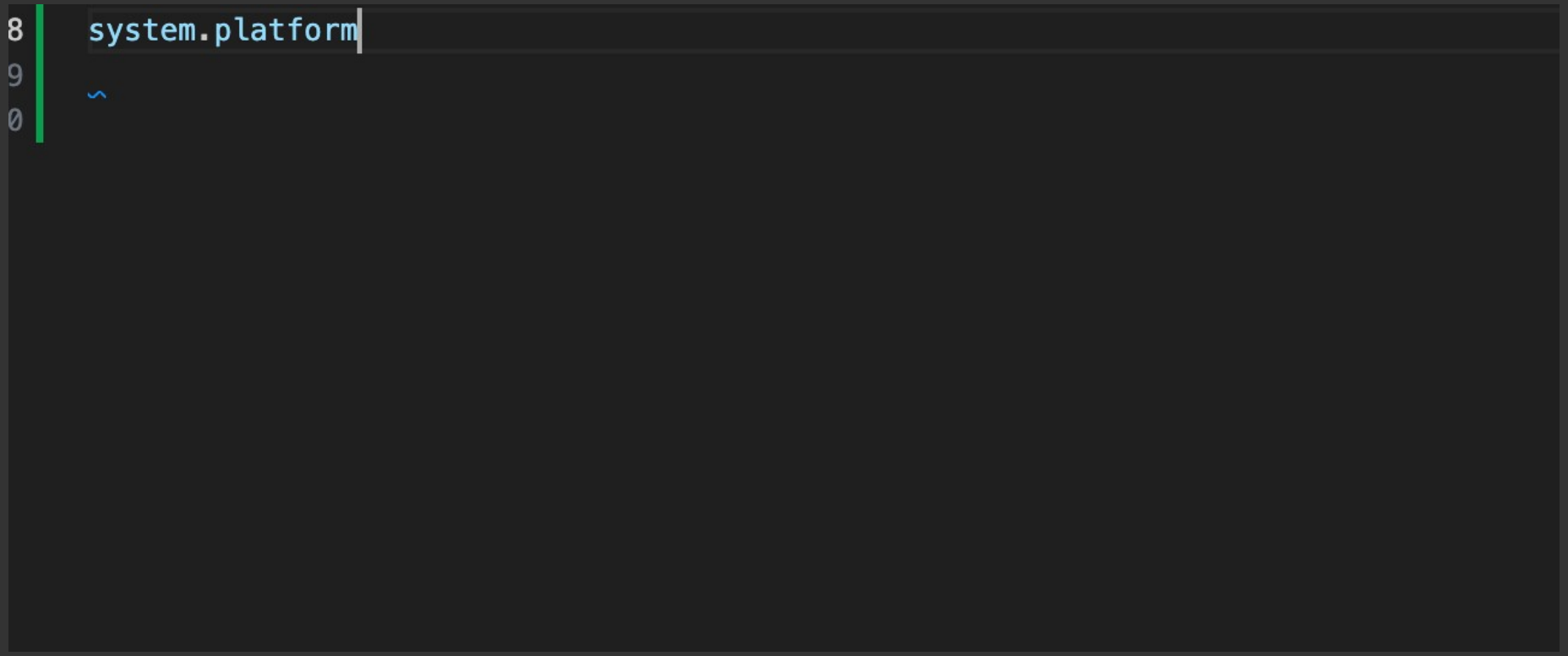
# Code-Autovervollständigung



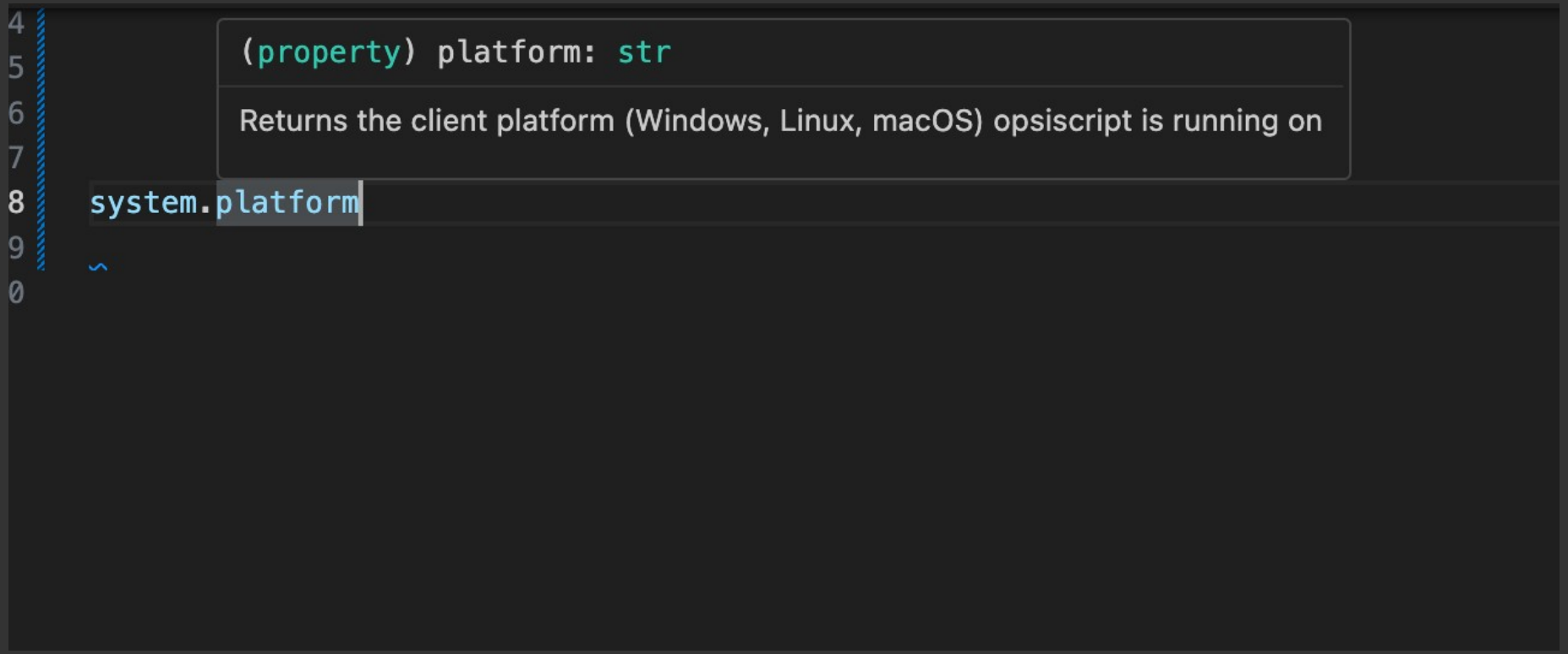
# Code-Autovervollständigung



# Code-Autovervollständigung



# Code-Hints


(property) platform: str
Returns the client platform (Windows, Linux, macOS) opsiscript is running on
system.platform

# Schnittstelle Expert – Beispiel-Skript

## Python

```python
from opsiscript import file, installer, script, system

# „Sektionen“
def install_winnt6():
    file.copy("/files_win10/*.*", "c:/temp/installation")
    installer.execute(path="c:/temp/installation/setup.exe")

# Actions
script.message("Installation von Mozilla")
script.set_loglevel(6)

if system.version >= "6.0":
    install_winnt6()
else:
    script.stop("not a supported OS-Version")
```

## opsi-script

```
[Actions]
Message "Installation von Mozilla"
SetLogLevel=6

if CompareDotSeparatedNumbers(GetMsVersionInfo,">=","6.0")
  sub_install_winnt6
else
  stop "not a supported OS-Version"
endif


; Sektionen
[sub_install_winnt6]
Files_copy_winnt6
WinBatch_Setup

[Files_copy_winnt6]
copy "\files_win10\*.*" "c:\temp\installation"

[WinBatch_Setup]
"c:\temp\installation\setup.exe"
```

# Schnittstelle Expert – Beispiel-Skript

## Python

```
from opsiscript import file, installer, script, system

# „Sektionen“
def install_winnt6():
    file.copy("/files_win10/*.*", "c:/temp/installation")
    installer.execute(path="c:/temp/installation/setup.exe")

# Actions
script.message("Installation von Mozilla")
script.set_loglevel(6)

if system.version >= "6.0":
    install_winnt6()
else:
    script.stop("not a supported OS-Version")
```

## opsi-script

```
[Actions]
Message "Installation von Mozilla"
SetLogLevel=6

if CompareDotSeparatedNumbers(GetMsVersionInfo,">=","6.0")
  sub_install_winnt6
else
  stop "not a supported OS-Version"
endif


; Sektionen
[sub_install_winnt6]
Files_copy_winnt6
WinBatch_Setup

[Files_copy_winnt6]
copy "\files_win10\*.*" "c:\temp\installation"

[WinBatch_Setup]
"c:\temp\installation\setup.exe"
```

# Schnittstelle Easy

# Schnittstelle Easy

- High-Level Schnittstelle
  - Parametrisierung deklarativ über Metadaten
  - einfache opsi-Produkterstellung
  - Standard Installationen
  - Tägliche Arbeit

# Metadaten-Datei

- Metadaten-Datei (TOML- Format)
  - Metadaten-Version
  - Skript-Version
  - Requirements
  - Installer
  - Shells
  - …

# Easy-Schnittstelle

## Metadaten-Datei (TOML):

```
[specification]
version = ‘0.1-alpha’


[[Installers]]
path=‘%scriptpath%/setup.exe’
params=[‘/S’] #optional
```

# Easy-Schnittstelle

## Metadaten-Datei (TOML):

```
[[Installers]]
path='%scriptpath%/setup.exe'
params=['/S']
[[Installers.requirements]]
os_arch=['x86']

[[Installers]]
path='%scriptpath%/setup_64.exe'
params=['/S']
[[Installers.requirements]]
os_arch=['x64']
```

# Vergleich Easy - opsi-script

Easy (TOML):

```
[Specification]
version="0.1-alpha"

[[Installers]]
path = 'c:\temp\installation\setup.exe'
[[Installers.requiremenst]]
os = 'windows'
os_min_version = '6.0'
```

opsi-script:

```
[Actions]

DefVar $MSVersion$
Set $MSVersion$ = GetMsVersionInfo

if CompareDotSeparatedNumbers($MSVersion$,">=","6.0")
  WinBatch_Setup
endif

[WinBatch_Setup]
"c:\temp\installation\setup.exe"
```

# Kombination Easy und Expert-Schnittstelle

## Metadaten-Datei (TOML):

```
[[Installers]]
path='%scriptpath%/setup.exe'
params=['/S']
[[Installers.requirements]]
os_arch=['x86']

[[Installers]]
path='%scriptpath%/setup_64.exe'
params=['/S']
[[Installers.requirements]]
os_arch=['x64']
```

## Skript-Datei (Python):

```
from opsiscript import installer, file, script

installer.execute()

if file.exist("Program.exe"):
    install_manual()
else:
    script.stop("Error: Installation error")
```

Was ist mit „alten“ opsi-script Skripten die ich gerne ins neue Format bringen würde?

# Was ist mit „alten“ opsi-script Skripten?

- opsi-script to Python Konverter
  - Konvertiert opsi-script Skripte nach Python
  - Statisch (evtl. auch zur Laufzeit)
- Plugin/Extension für (VS) Code-Editor
- QuickGuide für Umsteiger*Innen

# QuickGuide: If-Bedingung

Python:

```
if (platform == "macOS"):
        ...
elif (platform == "Linux"):
        ...
elif (platform == "Windows"):
        ...
else:
        ...
```

opsiscript:

```
if ($Platform$ = "macOS")
    ...
elseif ($Platform$ = "Linux")
    ...
elseif ($Platform$ = "Windows_NT")
    ...
else
    ...
endif
```

# Was ist mit „alten“ opsi-Produkten?

# Was ist mit „alten“ opsi-Produkten?

- Laufen so wie bisher
- Werden mit dem „alten“ opsi-script ausgeführt
- „Alter“ opsi-script wird weiterhin gewartet und gepflegt

# Planung

- Endgültiges Konzept festlegen: nach opsiconf
- Alpha-Version: Q1 2025
- Alpha-Version Konvertierungstool: Ende 2025

# Fragen an die Community

- Was haltet ihr generell von dem Ansatz?
- Was für Ideen habt ihr dazu?
- Wo seht ihr Probleme?

Anregungen/Wünsche an:

**opsiscript_to_python@uib.de**

# Vielen Dank für die Aufmerksamkeit!