

# Ein Überblick über KDevelop

Sven Brauch (scummos)



# Inhalt

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

# Inhalt

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

## Was ist KDevelop 4.x?

- ▶ im Kern: **C++-IDE** basierend auf KDE- und Qt-Bibliotheken
- ▶ Komplettes Rewrite von Version 3
- ▶ erstes 4.0-Release war 2008

Session Project Run Navigation | File Edit Tools Editor Code | Window Settings Help

Execute Debug New Save Stop All Build > Quick Open... isPrime(int, const std::vector< in

test.cpp

Line: 5 Col: 43

```
1 #include <vector>
2 #include <iostream>
3
4 bool isPrime(int check, const std::vector<int>& foundPrimes) {
5     for ( auto it = foundPrimes.begin(); it != foundPrimes.end(); it++ ) {
6         if ( check % *it == 0 ) {
7             return false;
8         }
9     }
10    return true;
11 }
12
13 std::vector<int> calculatePrimes(int stop = 20) {
14     std::vector<int> results;
15     results.push_back(2);
16     int currentNo = 3;
17     for ( ; currentNo < stop; currentNo += 2 ) {
18         if ( isPrime(currentNo, results) ) {
19             results.push_back(currentNo);
20         }
21     }
22     return results;
23 }
24
25 int main(int argc, char** argv) {
26     std::cout << calculatePrimes(20)[5];
27 }
```

\_\_gnu\_cxx::\_\_normal\_iterator< const\_pointer, vector > it  
Scope: isPrime Kind: Variable definition  
Decl.: test.cpp:5 Show uses

Code Browser Problems Konsole

## Was ist KDevelop 4.x? (2)

- ▶ weitere Sprachen können über **Plugins** integriert werden
- ▶ derzeit:
  - ▶ **PHP**
  - ▶ **Python**
  - ▶ ohne Release aber gut: **ruby**
  - ▶ in den Anfängen: QML / JS, Java
- ▶ Editor: KTextEditor, bekannt aus **kate**, kwrite, kile etc.

## Was heißt eigentlich „KDE“?

- ▶ „KDE“ im Volksmund:
  - ▶ Anwendungen (kate, amarok, konqueror, ...)
  - ▶ Desktop (plasma-desktop, KWin, ...)
- ▶ eigentlich: Community; auch: **Satz von Bibliotheken** (kdelibs) basierend auf Qt plus viele Anwendungen (deshalb seit 4.4 auch „KDE SC“)
- ▶ KDevelop hat technisch gesehen nichts mit dem „KDE Desktop“ zu tun
- ▶ Abhängigkeiten sind nicht der Rede wert

KDevelop? Was ist das eigentlich?

**Was ist das Ziel?**

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

# Inhalt

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

# Philosophie von KDevelop

- ▶ **keine** (signifikanten) **Projektdateien** — Projekt = Verzeichnis
- ▶ keine direkte Interaktion mit Compiler
  - ▶ stattdessen: Unterstützung für **seperate Build-Systeme** wie CMake
- ▶ keine „Bindung“ an IDE
- ▶ Fokus: hervorragende Code-Tools

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

**Architektur von KDevelop**

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop und KDevPlatform

Die Definition-Use-Chain (duchain)

Sprach-Plugins

Editor-Komponente

# Inhalt

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

# KDevelop und KDevPlatform

## KDevPlatform

- ▶ **Framework** für das Schreiben einer IDE
- ▶ Basis für Sprach- und andere Plugins

## KDevelop

- ▶ auf KDevPlatform basierende IDE
- ▶ Eigentliche Anwendung (sehr klein) + **Sammlung von Plugins** für KDevPlatform
  - ▶ C++ Sprachunterstützung
  - ▶ gdb-Debugger-Plugin
  - ▶ CMake-Projektmanager
  - ▶ ...

## Die Definition-Use-Chain (duchain)

- ▶ für IDE relevante Informationen über Code **in jeder Programmiersprache sehr ähnlich**
- ▶ entsprechend: Abstraktion möglich
- ▶ zwei wohl wichtigste Klassen: „**Definition**“ und „**Use**“
- ▶ deshalb: „**DUChain**“-**Framework** in KDevPlatform für Informationen über Code, unabhängig von der Sprache

```
int x;
```

„DEFINITION“

```
x = 3;
```

„USE“

```
myfunc(x);
```

Abbildung : Ein sehr einfaches Beispiel für eine Definition-Use-Chain

```

void func(int x);      DEF. ①
void func(int* x);    DEF. ②
...
int a = 3;
func(a);              USE OF ①
func(&a);             USE OF ②

```

Abbildung : Eine Verwendung („Use“) eines Objekts (Variable, Funktion etc.) weiß, zu welchem Objekt („Definition“) sie gehört

```
class Foo {  
    int x;  
    char* y;  
};
```

DEFINITION

```
Foo* myfoo;
```

USE

Abbildung : Für Klassen funktioniert das genauso.

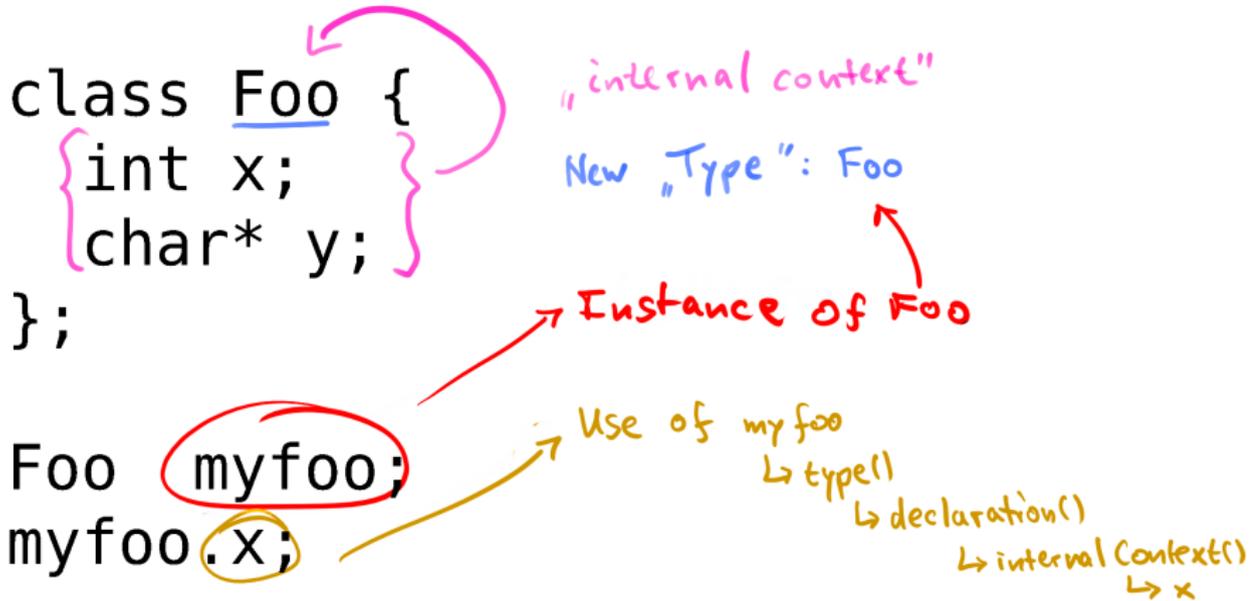


Abbildung : Ein kompliziertes Beispiel. Gelb: So geht das Sprach-Plugin intern vor.

## Die Definition-Use-Chain (duchain) (2)

- ▶ wird beim Öffnen eines Projekts für alle Dokumente berechnet
- ▶ bei jeder Änderung eines Dokuments wird die DUCHain für dieses Dokument neu berechnet
- ▶ KDevPlatform sorgt für persistentes Caching (*\$HOME/.cache/kdevduchain*)

## Sprach-Plugins

- ▶ Hauptaufgabe: bekommt Text aus Editor, soll Definition-Use-Chain berechnen
- ▶ Übliches Vorgehen:  
Text  $\xrightarrow{\text{Parser}}$  AST<sup>1</sup>  $\xrightarrow{\text{Declaration / Context / ... Builder}}$  DUChain
- ▶ Declaration / Context / ... Builder: verbindet KDevPlatforms API mit sprach-spezifischem „Visitor“

---

<sup>1</sup>AST steht für „Abstract Syntax Tree“

## Beispiel: Wie funktioniert das beim Python-Plugin?

```
my_var = 3
another_var = my_var
```

- ▶ Erstes Statement: Zuweisung  $\Rightarrow$  visitAssignment(...)
  - ▶ Berechne Typ der rechten Seite: int
  - ▶ Erstelle eine „Definition“ für my\_var
  - ▶ Setze Typ von my\_var auf „Instanz von int“
- ▶ Zweites Statement: Zuweisung  $\Rightarrow$  visitAssignment(...)
  - ▶ Berechne Typ der rechten Seite:  
Durchsuche aktuellen Kontext nach einer „Definition“ für my\_var
  - ▶ Erstelle eine „Definition“ für another\_var
  - ▶ Setze Typ von another\_var auf „Instanz von int“
  - ▶ Erstelle eine „Use“ für my\_var

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

## Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop und KDevPlatform

Die Definition-Use-Chain (duchain)

Sprach-Plugins

Editor-Komponente

```
def changeLimits(self):
    modes = {
        self.ui.limits_autoFromData : "autoData",
        self.ui.limits_autoFromIntegrated : "autoIntegrated",
        self.ui.limits_manual : "manual"
    }
    for elem, mode in modes.iteritems():
        if elem.isChecked():
            self.limitsMode = mode
    manualControls = [
        self.ui.limits_slider_x,
        self.ui.limits_slider_y,
        self.ui.limits_text_x,
        self.ui.limits_text_y
    ]
    for elem in manualControls:
        if self.limitsMode != "manual":
            elem.setDisabled(True)
        else:
            elem.setDisabled(False)

    if self.invertData:
        sliderRange = (-expectedAdRange[1]-20, expectedAdRange[0]+20)
    else:
        sliderRange = (expectedAdRange[0]-20, expectedAdRange[1]+20)
    self.ui.limits_slider_x.setRange(*sliderRange)
    self.ui.limits_slider_y.setRange(*sliderRange)
```

str mode

Container: [changeLimits](#) Kind: **Variable**

Decl: [main.py :308](#) [Show uses](#)

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

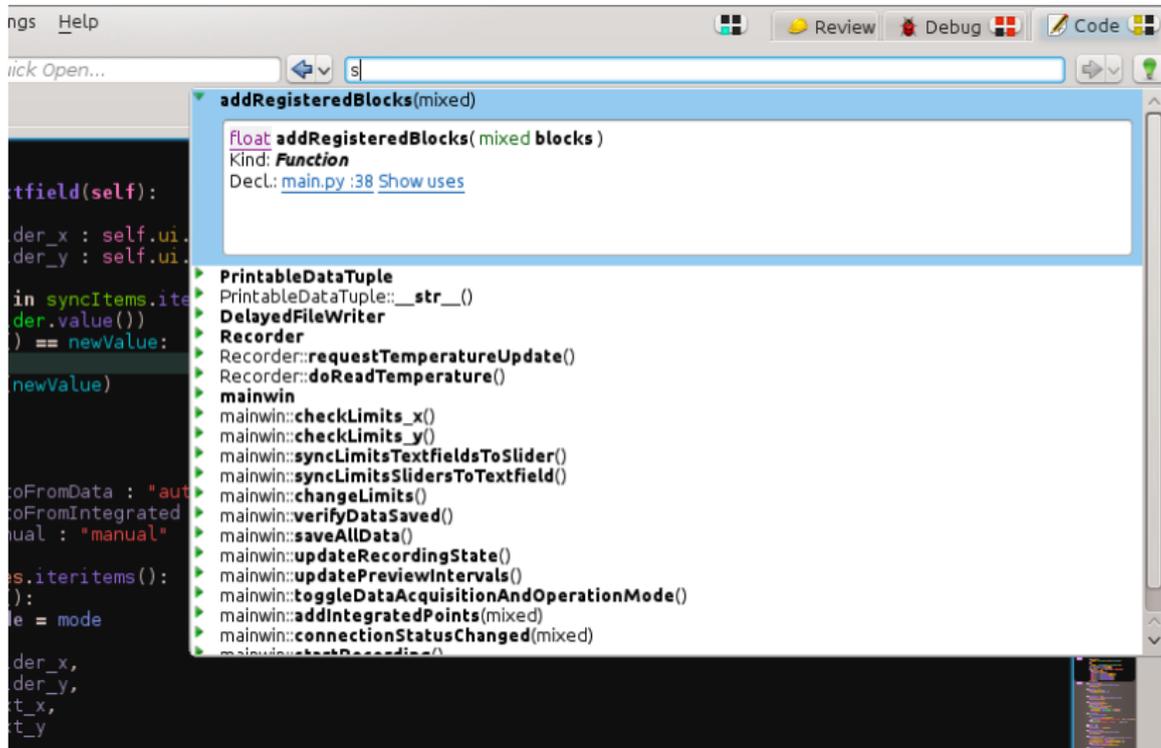
Sonstiges

KDevelop und KDevPlatform

Die Definition-Use-Chain (duchain)

Sprach-Plugins

Editor-Komponente



KDevelop? Was ist eigentlich?

Was ist das Ziel?

## Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop und KDevPlatform

Die Definition-Use-Chain (duchain)

Sprach-Plugins

Editor-Komponente

The screenshot shows the KDevelop IDE interface. On the left, the 'Classes' view displays a tree structure of classes and methods. The 'mainwin' class is expanded, showing various methods. The method 'handleRecordFinished()' is selected and highlighted. A tooltip is displayed over this method, providing details: 'void handleRecordFinished()', 'Container: mainwin Kind: Function', and 'Decl: main.py :615 Show uses'. On the right, the 'main.py' code editor is visible, showing Python code with line numbers 288 to 318. The code includes a 'syncLimits' function and a 'changeLimit' function. The 'handleRecordFinished()' method is partially visible at the bottom of the code editor.

Classes

Search:

- datetime
- ds1629Client
- i2cConnection
- mainwin
  - IOErrorOccured(mixed)
  - \_\_init\_\_()
  - addDataPoint(mixed)
  - addIntegratedPoints(mixed)
  - changeIntegrationMode()
  - changeLimits()
  - checkLimits\_x()
  - checkLimits\_y()
  - clearData()
  - connectionStatusChanged(mixed)
  - handleQueuedData()
  - handleRecordFinished()**
  - indexToTime(mixed)
  - invertDataToggled()
  - length()
  - maybeDointegration()
  - openDataFile()
  - recalculateIntegratedData()
  - redrawPlot()
  - saveAllData()
  - startRecording()
  - syncLimitsSlidersToTextField()
  - syncLimitsTextfieldsToSlider()
  - targetPath()

void handleRecordFinished()  
Container: mainwin Kind: Function  
Decl: main.py :615 Show uses

```
288 self.redraw
289 ....
290 def syncLimitsS
291 syncItems =
292 self.ui
293 self.ui
294 }
295 for slider,
296 newValu
297 if text
298 cor
299 textfie
300 self.redraw
301 .....
302 def changeLimit
303 modes = {
304 self.ui
305 self.ui
306 self.ui
307 or elem, n
308 if elem
309 self
310 manualConti
311 self.ui
312 self.ui
313 self.ui
314 self.ui
315 self.ui
316 ]
317 for elem in
318 if self
```

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

**Architektur von KDevelop**

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop und KDevPlatform

Die Definition-Use-Chain (duchain)

Sprach-Plugins

**Editor-Komponente**

# Editor-Komponente

- ▶ **derselbe Editor** wie in Kate, KWrite, Kile, ...
- ▶ **vi-Eingabemodus!**
- ▶ skriptbar, z. B. Snippets

## Beispiel für skriptbare Snippets

```
\documentclass{article}

\begin{document}

\begin{table}
\center
\begin{tabular}{cclr}
\toprule
title0 & title1 & title2 & title3 \\
\midrule
content0 & content1 & content2 & content3 \\
\bottomrule
\end{tabular}
\caption{CAPTION}
\end{table}

\end{document}
```

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

**Architektur von KDevelop**

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

KDevelop und KDevPlatform

Die Definition-Use-Chain (duchain)

Sprach-Plugins

**Editor-Komponente**

# Rekapitulation

- ▶ Verwendung **derselben Grund-Datenstrukturen für jede Sprache**
- ▶ Aufgabe der Sprach-Plugins: diese Datenstrukturen „bevölkern“
- ▶ Wer will, kann dank KDevPlatform / KDevelop-Trennung eine neue IDE mit anderer UI, aber demselben Backend schreiben.

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

Überblick

Typen in Python

C-Bibliotheken

Code-Vervollständigung

# Inhalt

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

# kdev-python: Python-Sprachplugin für KDevelop

- ▶ Hauptaufgabe: Möglichst gute **Informationen über Code sammeln**
- ▶ hat außerdem auch z. B. ein Debugger-Plugin
- ▶ relativ **klein** (Gesamt 13k SLOC vs. 170k SLOC bei PyDev (Eclipse), Debugger-Plugin 1200 SLOC vs 15k bei PyDev) dank Abstraktion

## Probleme mit Typen in Python (1)

- ▶ Python benutzt „Duck Typing“
- ▶ Eine Variable kann potentiell **mehrere mögliche Typen** haben
- ▶ Lösung: „unsure“ types

```
1  #!/usr/bin/env python2.7
2
3  import random
4  def myfunc():
5      if random.randint(0, 20) > 10:
6          myvar = 1337
7      else:
8          myvar = "Hello World"
9      return myvar
10
11 print myfunc()
```

unsure (int, str) myfunc()  
Kind: **Function**  
Decl.: [test.py :4](#) [Show uses](#)

Abbildung : Ein einfaches Beispiel für „unsure types“

## Probleme mit Typen in Python (2)

- ▶ in manchen Situationen aber:  
keine Chance (z. B. `exec()` aka `eval`)
- ▶ auch schwierig: Listen mit dynamischer Länge und unterschiedlichen Typen

```
3 class AwesomeNumber():
4     def __init__(self, number):
5         self.number = number
6
7     def myfunc():
8         mylist = []
9         for counter in range(20):
10            if counter % 2 == 0:
11                mylist.append(AwesomeNumber(counter))
12            else:
13                mylist.append("<some stupid odd number>")
14        return mylist[3]
15
16 result = myfunc()
17 result2 = result.number
```

unsure (AwesomeNumber, str) result  
Kind: **Variable**  
Decl: [test.py:16](#) [Show uses](#)

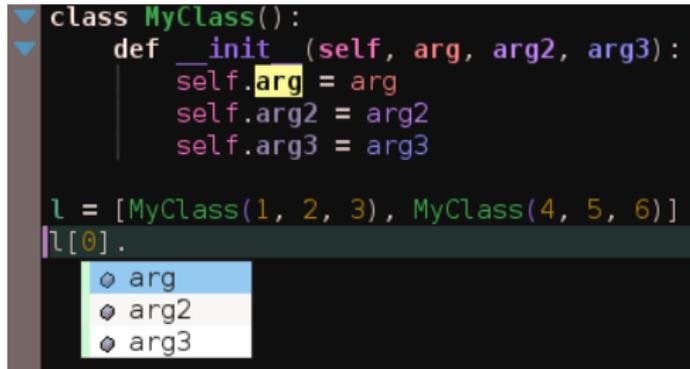
Abbildung : Ein weiteres Beispiel für „unsure types“

## Probleme mit Bibliotheken

- ▶ Python C-Libraries **nicht maschinenlesbar dokumentiert**
- ▶ nur in wenigen Spezialfällen (PyQt, PyKDE) konsistent formatierte Informationen, z. B über Rückgabewerte von Funktionen
- ▶ Lösung: unklar, Versuch: Introspection + manuelle Korrekturen

## Code-Vervollständigung (1)

```
class MyClass():  
    def __init__(self, arg, arg2, arg3):  
        self.arg = arg  
        self.arg2 = arg2  
        self.arg3 = arg3  
  
l = [MyClass(1, 2, 3), MyClass(4, 5, 6)]  
l[0].
```



- ▶ neben QuickOpen das wohl wichtigste Feature überhaupt
- ▶ gute Typ-Informationen notwendig

## Code-Vervollständigung (2)

`str(sqrt(13 + myobject.mylist.))`

*accessed object*

*Junk ⇒ ignored*

*Member Access*

- ▶ benutzt eigenen kleinen Tokenizer und viele Fallunterscheidungen
- ▶ Zeile wird in Expressions zerlegt, die dann einzeln in den „großen“ Analyzer geworfen werden

## Code-Vervollständigung (3)

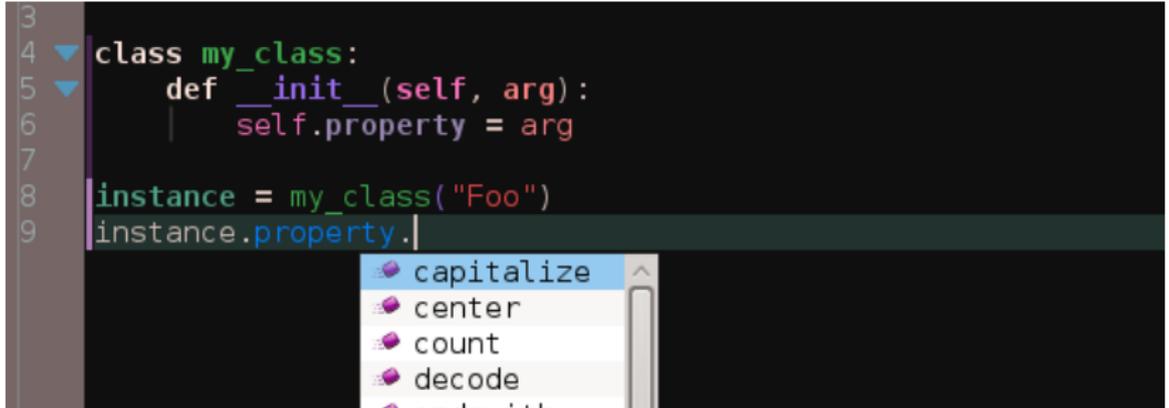
```
mylist[mylist[0].arg3].property[round(foo, 2) + 17.3*math.pi].some_property["Foo"].c
```

- capitalize
- center
- count

Abbildung : Die Vervollständigung funktioniert auch in komplexen Szenarien korrekt

## Beispiele für Code-Vervollständigung (1)

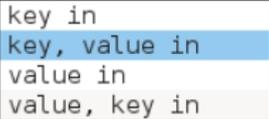
```
3
4 ▼ class my_class:
5 ▼     def __init__(self, arg):
6         self.property = arg
7
8 instance = my_class("Foo")
9 instance.property.
```



- capitalize
- center
- count
- decode
- encode

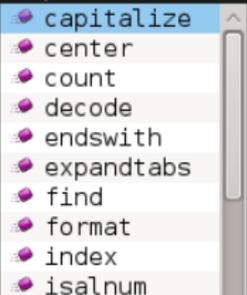
## Beispiele für Code-Vervollständigung (2)

```
1  #!/usr/bin/env python2.7
2  import math
3
4  def myfunc(param):
5      foo = 42
6      my_list = [1, 2, 3]
7      result = [str(key + value + foo) for
```



## Beispiele für Code-Vervollständigung (3)

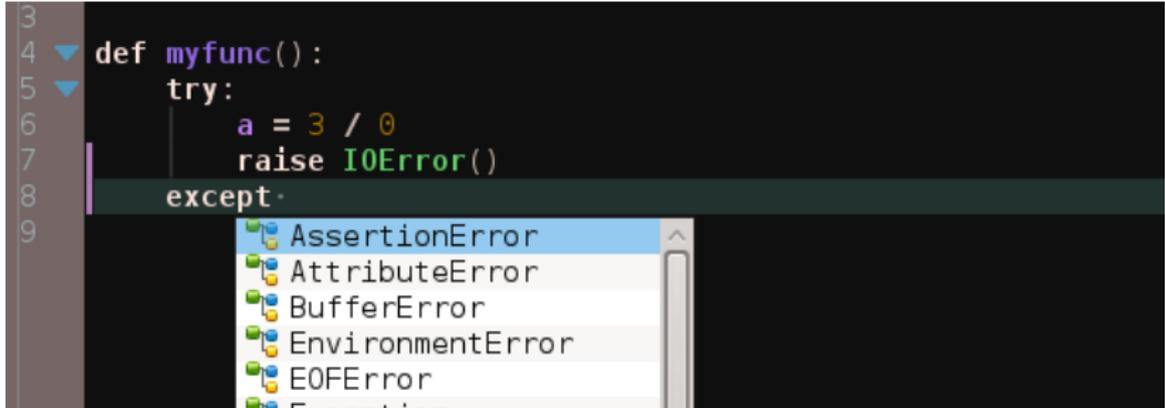
```
3  
4 mytuple = ("Foo", "Bar", [3, 5, 7, 9])  
5 mytuple[0].|
```



- capitalize
- center
- count
- decode
- endswith
- expandtabs
- find
- format
- index
- isalnum

## Beispiele für Code-Vervollständigung (4)

```
3
4 ▼ def myfunc():
5 ▼   try:
6     a = 3 / 0
7     raise IOError()
8   except
9
```



The screenshot shows a code editor with a dark background. The code is as follows:

```
3
4 ▼ def myfunc():
5 ▼   try:
6     a = 3 / 0
7     raise IOError()
8   except
9
```

A dropdown menu is open under the 'except' keyword, listing various Python exception classes. The list includes:

- AssertionError
- AttributeError
- BufferError
- EnvironmentError
- EOFError
- Exception

The 'AssertionError' item is currently selected and highlighted in blue.

## Beispiele für Code-Vervollständigung (5)

```
3
4 class MyClass():
5     def __init__(self, param1, param2):
6         self
    self.param1 = param1
    self.param2 = param2
    self
```

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

Andere Sprachen in KDevelop

Fragen!

Einige Worte zu KDE als Community

Kontakt

# Inhalt

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

## Andere Sprachen in KDevelop?

- ▶ **Ruby**: wird aktiv entwickelt von Miquel, **in sehr gutem Zustand**
- ▶ kdev-qmljs (QML und JavaScript): nur Syntaxprüfung und ein paar diverse Features
- ▶ Java: total kaputt, aber seit kurzem arbeiten zwei Leute daran

KDevelop? Was ist das eigentlich?

Was ist das Ziel?

Architektur von KDevelop

kdev-python: Python-Sprachplugin für KDevelop

Sonstiges

Andere Sprachen in KDevelop

**Fragen!**

Einige Worte zu KDE als Community

Kontakt

# Fragen?

Danke für Eure Aufmerksamkeit!

## Einige Worte zu KDE als Community

- ▶ aktive **IRC-Channel und Mailinglisten** für jedes Projekt
- ▶ einigermaßen **einheitliche Technologien** und Patterns: kennt man sich in einem Projekt aus, so findet man sich in anderen meist schnell zurecht
- ▶ jeder Entwickler hat Schreibzugriff auf alle KDE-Repositories
- ▶ gute git- Bugtracker- und ReviewBoard-**Infrastruktur** ist schon da
- ▶ nette Leute kümmern sich um **Übersetzung** jedes Projekts in dutzende Sprachen
- ▶ Sprints und Konferenzen

# Kontakt

- ▶ `irc.freenode.net #kdevelop`
- ▶ KDevelop-Webseite: `http://kdevelop.org`
- ▶ `kdev-python`: `http://projects.kde.org/kdev-python`
- ▶ `git`: `git clone git://anongit.kde.org/kdevelop` (oder `kdevplatform`, `kdev-python` etc.)