PG
Consulting

User Story 4:
Typprüfung von Funktionsargumenten

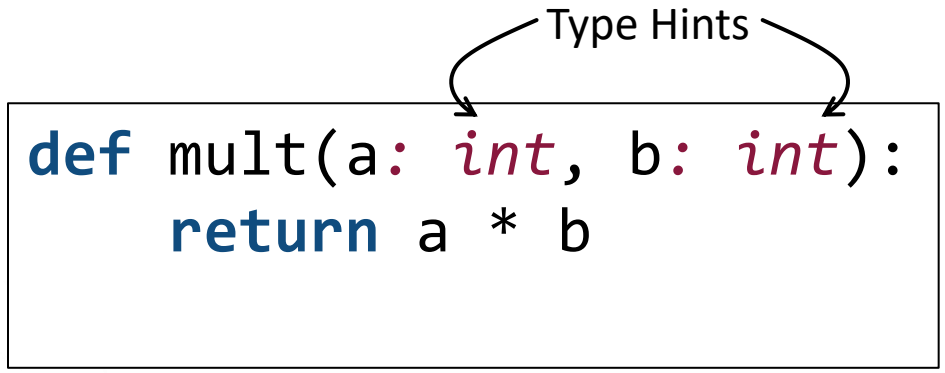
Bernhard Stadler

Klaudia Thellmann

PG Consulting

Worum geht es?

- Funktionen können i. d. R. nicht mit allen Eingaben umgehen
  - Beispiel: Wie multipliziert man „hello“ und „world“?
- Mit *Type Hints* kann man den erwarteten Typ eines Parameters festlegen
- Der Typ jedes Arguments beim Funktionsaufruf muss kompatibel zum Typ des entsprechenden Parameters bei der Funktionsdeklaration sein → Typprüfung



Funktionsdeklaration

```
mult(2, 3)
```
✓

```
mult("hello", "world")
```
✗

Funktionsaufruf

PG Consulting

Problem: Type Hints sind optional

- Funktionsdeklarationen müssen keine Type Hints enthalten
- Typen können auch informell durch Dokumentation festgehalten werden

```
def mult(a, b):
    """Params:
        a (int)
        b: int"""
    return a * b
```

Keine feste Syntax!

- Oder die Angabe fehlt komplett

```
def mult(a, b):
    return a * b
```

PG Consulting

Beispiel in scikit-learn

- Typangaben werden nur in der Dokumentation festgehalten
- Die Syntax ist nicht einheitlich zwischen verschiedenen Klassen und Funktionen

```
class LinearRegression():
    """
    Ordinary least squares Linear Regression...
    Parameters
    ----------
    fit_intercept : bool, default=True
        Whether to calculate...
    normalize : bool, default=False
        This parameter is ignored...
    copy_X : bool, default=True
        If True, X...
    n_jobs : int, default=None
        The number of jobs...
```

PG Consulting

Beispiel in PyTorch

- Typangaben stehen teilweise
  - in der Dokumentation in variierender Syntax

```
class Dropout2d(_DropoutNd):
    """
    During training, randomly…
    Args:
        p (float, optional):
            probability of an element…
        inplace (bool, optional):
            If set to…
```

  - In Python-Interface-Dateien (.pyi) in Form von Type Hints

```
class Linear(Module):
    def __init__(
        self,
        in_features: int,
        out_features: int,
        bias: bool = ...
    ) -> None: ...
```

PG Consulting

User Story

Als Python-Entwickler möchten wir, dass schon während der Programmentwicklung eine Typprüfung durchgeführt wird und wir einen Fehler angezeigt bekommen, wenn ein Argumenttyp bei einem Funktionsaufruf nicht kompatibel ist zum entsprechenden Parametertyp bei der Funktionsdeklaration.

Dadurch können wir Fehler frühzeitig erkennen, ohne das Programm erst ausführen zu müssen.

Hinweis: In künftigen Iterationen werden wir weitere Überprüfungen anfordern. Uns ist wichtig, dass diese Überprüfungen später mit geringem Aufwand in verschiedene IDEs und Code-Editoren eingebaut werden können und überall die gleichen Meldungen liefern.

PG Consulting

Limitierungen

- Funktionsdeklaration:
  - Typinformationen zu Parametern sollen nur aus Type Hints und Dokumentation extrahiert werden
  - Fehlt beides, braucht für den Parameter keine Typprüfung durchgeführt werden
  - Da es keine feste Syntax zur Dokumentation von Parametertypen gibt, sollen vorerst folgende Fälle unterstützt werden:
    - Formen „Variablenname: Typ“ und „Variablenname (Typ)“
    - Primitive Typen int, bool, str
    - Listen, Dicts, Tupel sollen unterstützt werden
    - Klassen- u. Funktionstypen sind optional
- Funktionsaufruf:
  - Es brauchen nur folgende Arten von Argumenten überprüft werden:
    - Literale (3, 4.5, „hello“, True, None, …)
      mit Ausnahme von Listen-, Dictionary- und Tupel-Literalen
    - Variablen mit Type Hint

```
g(f(), [3], 5)
```

✗

```
a: int = f()
b: List[int] = [3]
g(a, b, 5)
```

✓