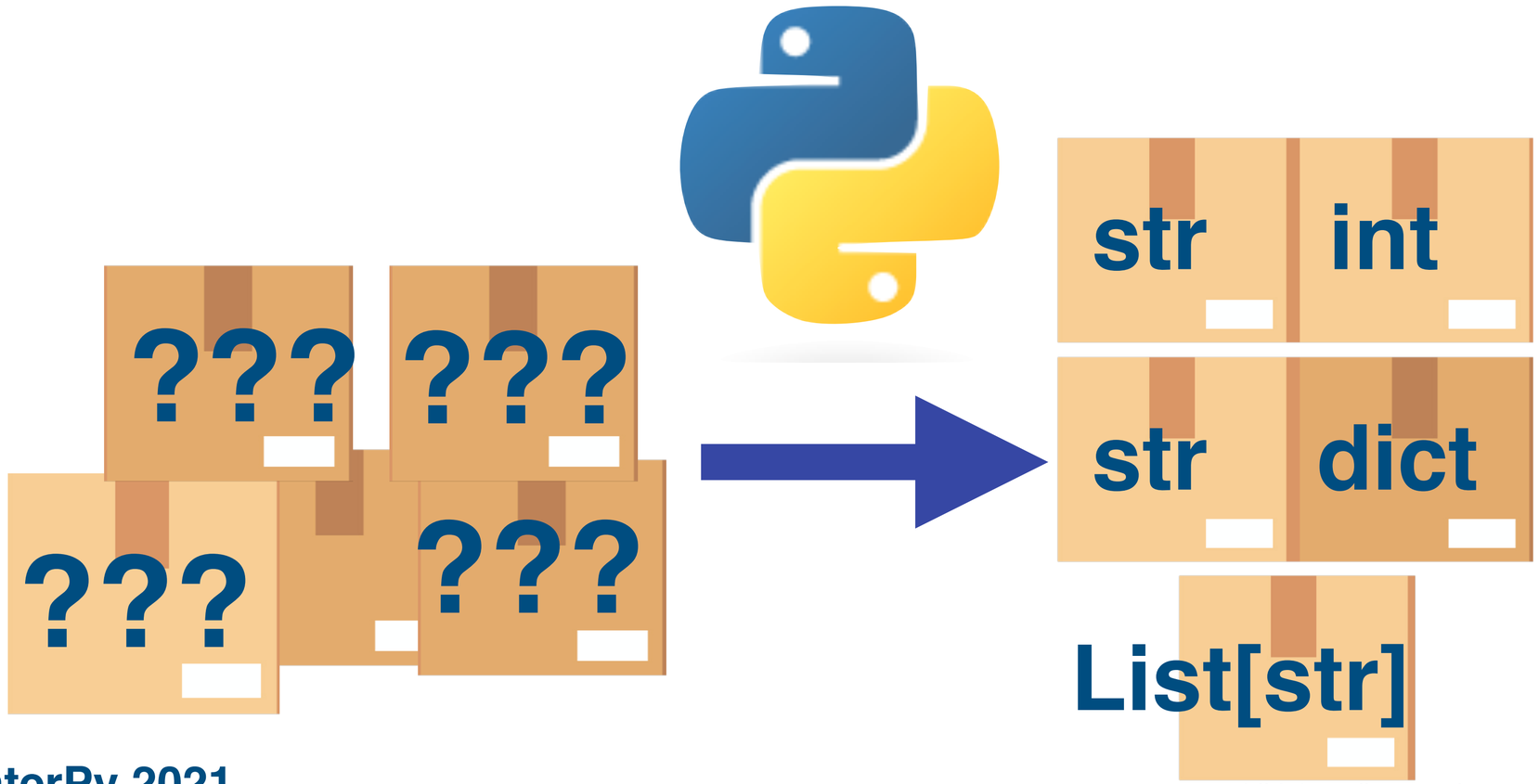
Typehints für besseres Python



Stefan Baerisch, stbaer.com, enterPy 2021

bild: freepiks premium, python.org

Was ist ein Typsystem?

Computer kennen keine Typen. Es gibt keinen Speicher für Strings, Zahlen oder Objekte.

Sprachen haben Typen um Operationen und z.B. Speicherverwaltung zu festzulegen

Typsysteme in Programmiersprachen

Static ⟺ **Dynamic**

Weak ⟺ **Strong**

Required ⟺ **Gradual / Optional**

C's Typsystem

```c
#include "stdio.h"

int main() {
    int a = 32;
    char * b = (char*)&a;
    b[1] = 4;
    printf("%i",a);
}
```

```
>cc main.c
>./a.out
1056
>
```

Java's Typsystem

```java
// <snip>
String a = "1";
int b = 10 + a;
int c = 10 + (a++);

// ~ 2000
HashMap<String,ArrayList<String>> vals = new HashMap<String,ArrayList<String>>();

// since Java 10, 2018
var vals2 = new HashMap<String, ArrayList<String>>();
// <snip>
```

```
>javac Example.java
Example.java:10: error: incompatible types: String cannot be converted to int
        int b = 10 + a;
                     ^
Example.java:11: error: bad operand type String for unary operator '++'
        int c = 10 + (a++);
                      ^
2 errors
```

Javascript's Typsystem

```
let a = "1";
let b = 10 + a;
let c = 10 + a++;
console.log(`b:${b}\nc:${c}\n`)
```

```
>node main.js
b:101
c:11
```

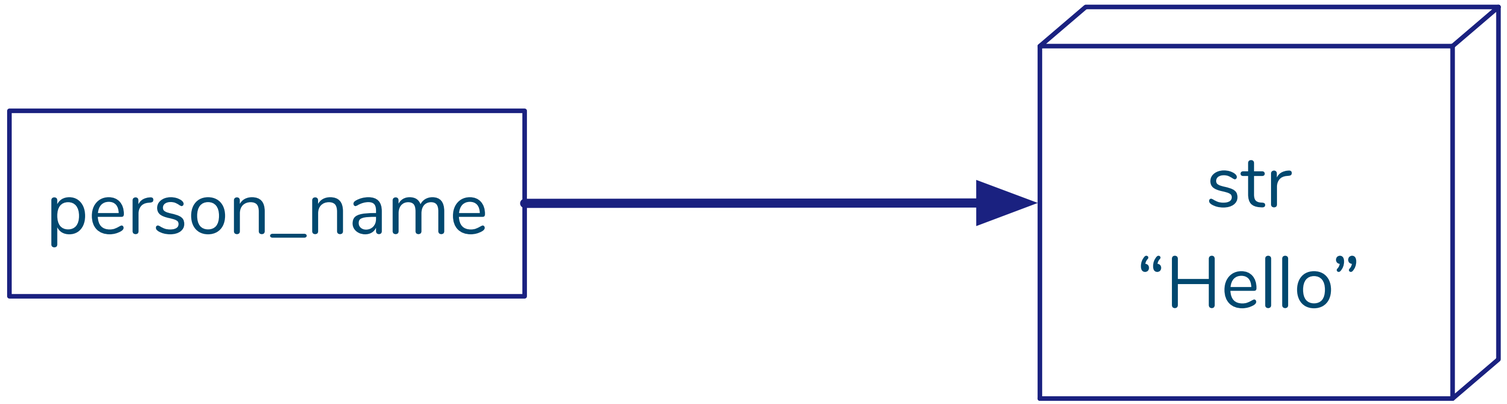
Python's Typsystem

```python
a = "1"
b = 10 + a
a += 1
c = 10 + a
print(f"\nb:{b}\nc:{c}")
```

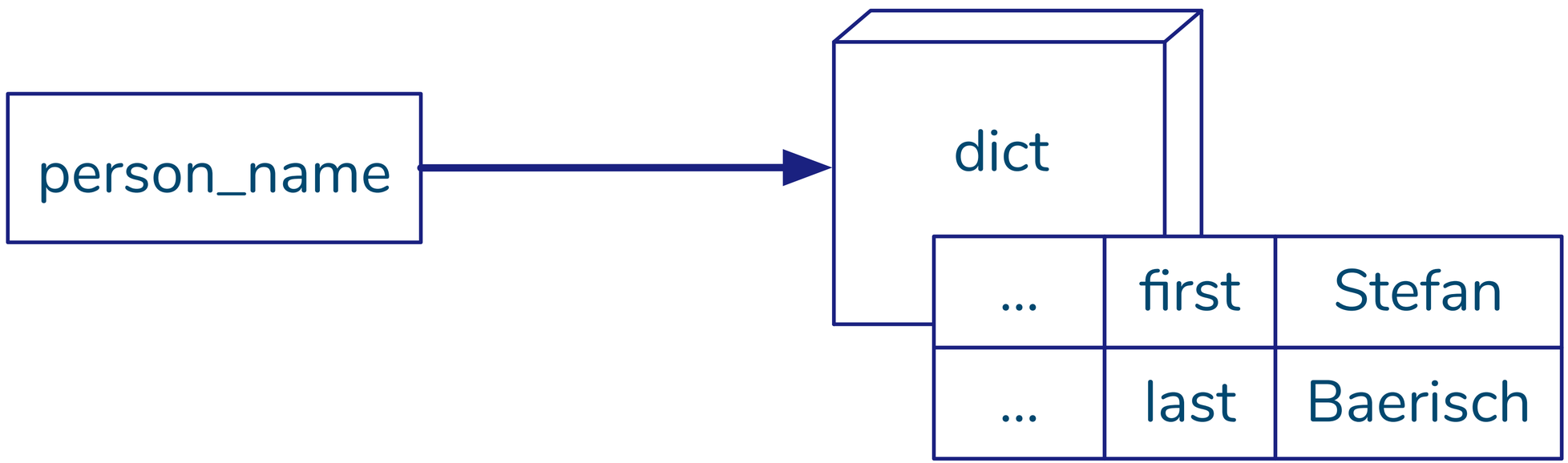
```
>python main.py
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    b = 10 + a
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python's Typsystem - Hintergrund

person_name = “Stefan Baerisch”

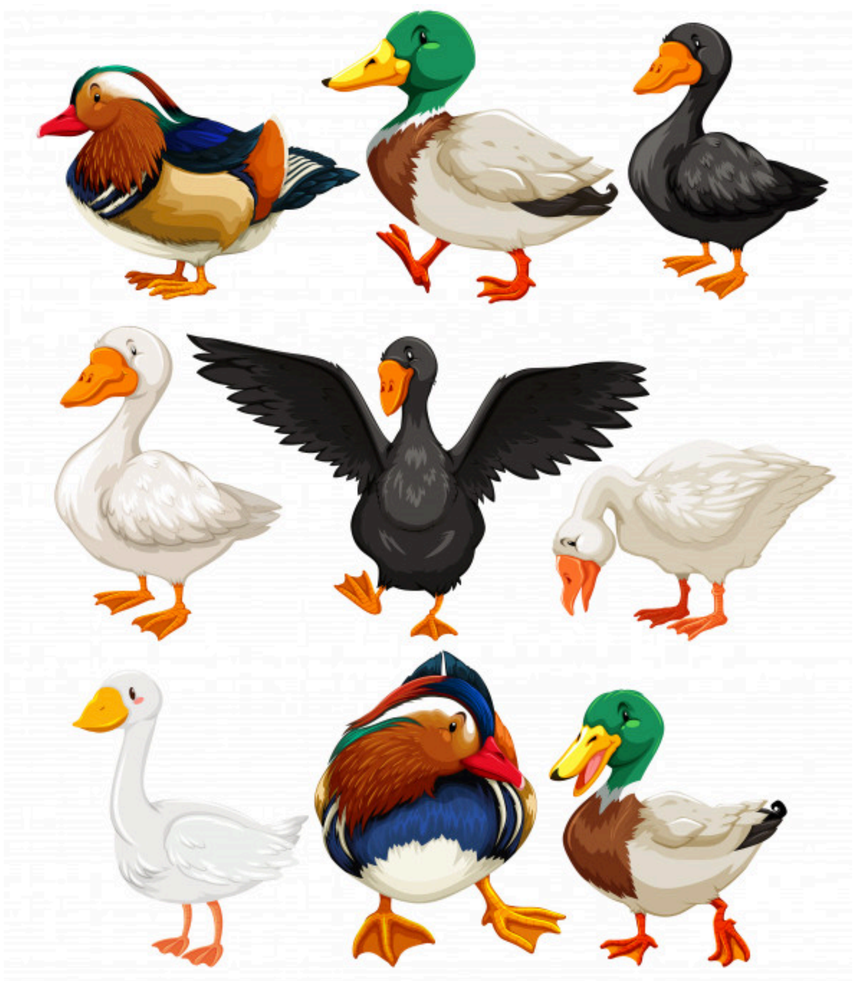


person_name = {“first”:”Stefan”, “last”:”Baerisch”}



Wehe

- Duck Typing: Wenn es wie eine Ente quakt...



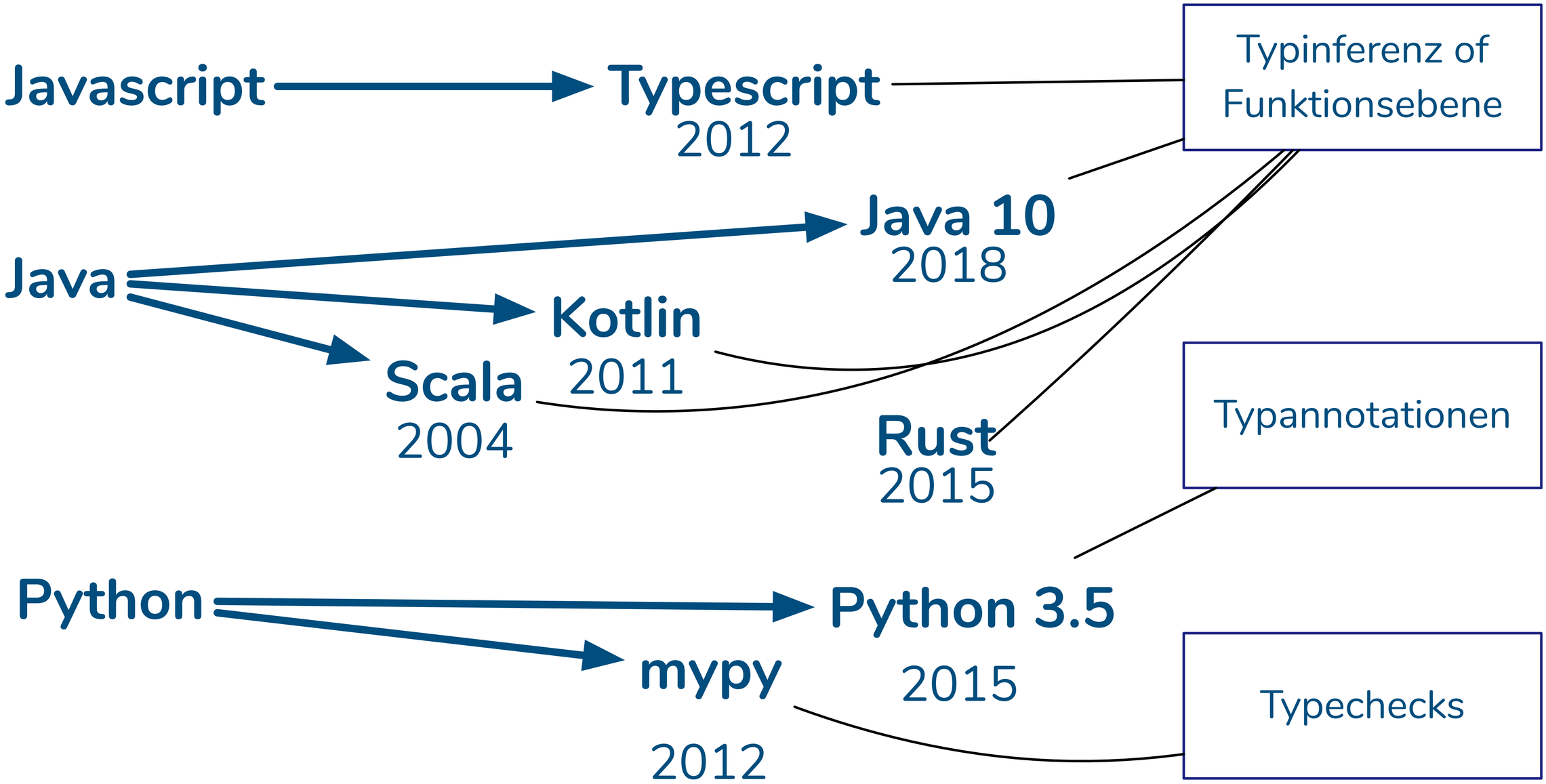
- **WeHe** - Welche Ente wird erwartet???

bild: freepiks premium

Was bedeuten Typsysteme für Anwender?

	Python (ohne Typannotationen)	Python (mit Typannotationen)	Java/Go/Rust
Entwicklung	x	Annotieren Dokumentation	Design & Implementierung
Build	x	optionale Checks	Typchecks
Run	mögliche Fehler	optionale Checks mögliche Fehler	x ggf. Reflection

Trends in Typsystemen



Typsysteme in Python, in 2021

PEP 484 — Type Hints für Funktionen
Python 3.5

PEP 526 — Type Hints für Variablen
Python 3.5

PEP 544 — Structural subtyping
Python 3.8

PEP 585 — Standard Collection Generic Types
Python 3.9

PEP 586 — Literal Types
Python 3.8

PEP 589 — Typed Dicts
Python 3.8

PEP 591 — Final Qualifier
Python 3.8

PEP 593 — Flexible Annotations
Python 3.8

PEP 604 — Union as |
Python 3.10

Ein Beispiel - Einfache Typen

```python
def add(a:int, b:int) -> int:
    return a + b

if __name__=='__main__':
    print(add(1,2))

    print(add("hello ","world"))
```

```
python ex1.py
3
hello world

mypy ex1.py
ex1.py:8: error: Argument 1 to "add" has incompatible type "str"; expected "int"
ex1.py:8: error: Argument 2 to "add" has incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

Ein Beispiel - Komplexere Typen (1)

```python
from typing import Optional

class Person:
    def __init__(self,name:str,age:int):
        self.name = name
        self.age = age

def oldest_person(persons:list[Person]) -> Optional[Person]:
    if len(persons) == 0:
        return None
    result : Person = persons[0]
    for p in persons[:-1]:
        if p.age > result.age:
            result = p
    return result

if __name__=='__main__':
    persons = [Person("stefan",43), Person('anni', 32), Person('bert', 27)]
    print(oldest_person(persons).name)
```

ex.py:20: **error:** Item **"None"** of **"Optional[Person]"** has no attribute **"name"**
Found 1 error in 1 file (checked 1 source file)

Ein Beispiel - Komplexere Typen (2)

```python
from typing import Optional

class Person:…

def oldest_person(persons:list[Person]) → Optional[Person]:…

if __name__=='__main__':
    persons = [Person("stefan",43), Person('anni', 32), Person('bert', 27)]
    person = oldest_person(persons)
    if person:
        print(person.name)
```

$ mypy ex.py

Success: no issues found in 1 source file

Beispiel : Typed Dicts

```python
from typing import  TypedDict,Optional

class PersonBase(TypedDict):
    name : str
    age: int

class Person(PersonBase,total=False):
    job: str

def greet(person:Person) -> None:
    print(f"{person['name']}:{person.get('age',0)}")
```

```python
person : Person = {'name':'Stefan','age':43}
greet(person)

person2 : Person = {'name':'Stefan','age':43, 'job':'freelancer'}
greet(person2)

person3 : Person = {'name':'jane'}
greet(person3)
```

```
s05_typed_dict.py:23: error: Missing key 'age' for TypedDict "Person"
Found 1 error in 1 file (checked 1 source file)
```

Beispiel Data Classes

```
from dataclasses import dataclass,field

kg = int

@dataclass
class Equipment:
    name: str
    weight: kg

@dataclass
class Hiker:
    name: str
    equipment: list[Equipment] = field(default_factory=list)
```

```
hiker_1 = Hiker("Stefan", [Equipment("backpack", 5), Equipment("flask",2)])
hiker_2 = Hiker("Joe")
hiker_1_clone = Hiker("Stefan", [Equipment("backpack", 5), Equipment("flask",2)])

print(hiker_1)
print(f"h1 == h2? {hiker_1 == hiker_2}")
print(f"h1 == h1_clone? {hiker_1 == hiker_1_clone}")
```

```
Hiker(name='Stefan', equipment=[Equipment(name='backpack', weight=5), Equipment(name='flask', weight=2)])
h1 == h2? False
h1 == h1_clone? True
```

Beispiel pydantic

```
from datetime import date
from datetime import datetime

from pydantic import BaseModel, Field

class Person(BaseModel):
    age: int
    name: str
    created: date = Field(datetime.date(datetime.today()))


if __name__ == '__main__':
    p1 = Person(name='stefan', age=43)
    print(p1)

    p2 = Person(name=10101, age="101", created="2020-01-01")
    print(p2)

    p3  = Person(name='joey')
```

```
age=43 name='stefan' created=datetime.date(2021, 2, 14)
age=101 name='10101' created=datetime.date(2020, 1, 1)
python-BaseException
[...]
age
  field required (type=value_error.missing)
```

Typannotationen erstellen

Dynamisch bleiben

```
from typing import Any

def add(a,b):
    return a +b

def add2(a:Any, b:Any) -> Any:
    return a + b
```

```
print(add(1,2))
print(add2(1,2))
print(add("A","B"))
print(add2("A","B"))
```

Typen für einfache Datentypen

```
def adder_1(a:int,b:int) -> int:
    result:int= a + b
    return result

s: str = "abc"
f: float = 1.23
b: bytes = b'123'

print(adder_1(1,2))   # run
print(adder_1("hello ", "world")) #also runs
```

```
>mypy s01_simple.py
s01_simple.py:11: error: Argument 1 to "adder_1" has incompatible type "str"; expected "int"
s01_simple.py:11: error: Argument 2 to "adder_1" has incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
>
```

Typen für Listen, Tuple, und Dicts

```
from typing import List, Dict

def sum_1(vs: List) -> int:
    return sum(vs)

def sum_2(vs: List[int]) -> int:
    return sum(vs)

def filter_dict(vs: Dict[str, int], min: int = 10) -> Dict[str, int]:
    result: Dict[str, int] = {k: v for k, v in vs.items() if v >= min}
    return result

def keylist_py39(vs: dict[str, int]) -> list[str]:
    return list(vs.keys())

def tuplelist_py39(vs: dict[str, int]) -> list[tuple[str, int]]:
    return list(vs.items())

vs_1 = [0, 1, 2, 3, 5, 8]
vs_2 = {'a': 11, 'b': 4, 'c': 20}
```

Optionale Werte and Alternativen

```
from typing import Optional, Union


def get_index(vs:list[int],val:int) -> Optional[int]:
    try:
        return vs.index(val)
    except ValueError:
        return None
```

```
def str_len(v: Union[int,float,str]) -> int:
    return len(str(v))


vs=[1,3,4]

print(get_index(vs,3))
print(get_index(vs,5))
```

Literale

```python
from typing import Literal

def myopen(name:str, mode:Literal['r', 'w', 'rw']):
    print(f"Opening {name} mode={mode}")

print(myopen("hello", "rw"))
print(myopen("hello", "aaa"))
```

```
>mypy s4_literals.py
s4_literals.py:10: error: Argument 2 to "myopen" has incompatible type
"Literal['aaa']"; expected "Union[Literal['r'], Literal['w'], Literal['rw']]"
Found 1 error in 1 file (checked 1 source file)
```

Typen für eigene Datentypen (Nominal)

```
class Person():
    def __init__(self,name:str,age:int):
        self.name= name
        self.age = age

class Employee(Person):
    def __init__(self,name:str,age:int, job:str):
        super().__init__(name,age)
        self.job = job

def greet(p:Person):
    print(f"Hello {p.name}")

p = Employee("stefan", 43, "freelance")

greet(p)
```

Typen für eigene Datentypen (Structural)

```python
from typing import Protocol

class Person():
    def __init__(self,name:str,age:int):
        self.name= name
        self.age = age

    def greet(self,g:'IGreeter'):
        g.greet(self)

class IGreeter(Protocol):
    def greet(self,p:Person):...

class Greeter():
    def greet(self,p:Person):
        print(f"Hi {p.name}")

p = Person("Stefan", 43)
g = Greeter()
p.greet(g)
```

Typen für generische Funktionen

```
from typing import TypeVar, Mapping

K = TypeVar("K")
V = TypeVar("V")

def lookup(m: Mapping[K, V], k: K) -> V:
    return m[k]

vs = {'name': 'Stefan', 'age': 43}

print(lookup(vs, 'name'))
```

Typen für Funktionen und Callables

```
from collections.abc import Sequence
from typing import Callable, TypeVar

T = TypeVar("T")
C = Callable[[T], None]
S = Sequence[T]

def call_all(fn: C, vs: S):
    for v in vs:
        fn(v)

def print_plus_one(v:int):
    print(v+1)

vs = [1,2,3]

call_all(print_plus_one,vs)
```

Annotationen woher? Kopieren!

```
# Vereinfachter Auszug https://github.com/python
# /typeshed/blob/master/stubs/pathlib2/pathlib2.pyi
from typing import Tuple, Union, TypeVar, Type,List
_P = TypeVar("_P")
class PurePath():
    parts: Tuple[str, ...]
    drive: str
    root: str
    anchor: str
    name: str
    suffix: str
    suffixes: List[str]
    stem: str
    def __new__(cls: Type[_P], *args: Union[str, _PathLike]) -> _P: ...
    def __hash__(self) -> int: ...
    def __lt__(self, other: PurePath) -> bool: ...
    def __le__(self, other: PurePath) -> bool: ...
    def __gt__(self, other: PurePath) -> bool: ...
    def __ge__(self, other: PurePath) -> bool: ...
    def __truediv__(self: _P, key: Union[str, _PathLike]) -> _P: ...
    def __rtruediv__(self: _P, key: Union[str, _PathLike]) -> _P: ...
    def __div__(self: _P, key: Union[str, PurePath]) -> _P: ...
```

Reine Interfacebeschreibung und Typen können in .pyi Dateien abgelegt werden.

Diese stellen für die entsprechende .py Datei Typen bereit

Annotationen woher? Generieren

```python
class Person:
    def __init__(self,name,age):
        self.name = name
        self.age = age
        self._stuff = []

    def add_stuff(self,stuff,weight):
        self._stuff.append((stuff,weight))

    def get_weight(self):
        return sum(w for (_,w) in self._stuff)
```

```python
from person import Person

p = Person("Stefan",43)
p.add_stuff("tent",5)
p.add_stuff("water",2)
print(p.get_weight())
```

>monkeytype run s11_monkeytype.py
>monkeytype stub person > person.pyi

```python
class Person:
    def __init__(self, name: str, age: int) -> None: ...
    def add_stuff(self, stuff: str, weight: int) -> None: ...
    def get_weight(self) -> int: ...
```

Typen prüfen mit mypy

Typechecker für Sourcecode - mypy

- mypy ist (ein) statischer Typechecker
- kein Einfluss auf das Laufzeitverhalten, wird in den Buildprozess eingebunden
- prüft, ob vorhandene Typen korrekt verwendet werden
- Umgang mit nicht typisiertem Code kann konfiguriert werden

Verwendung von mypy

ex6_mypy.py

```
def add(a:int, b:int) -> int:
    return a + b



add(1,2)
add('a','b')
```

```
> mypy ex6_mypy.py
ex6_mypy.py:6: error: Argument 1 to "add" has incompatible type "str"; expected "int"
ex6_mypy.py:6: error: Argument 2 to "add" has incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
>
```

Verwendung von mypy - ignore

ex6_mypy_1.py

```
# type: ignore
def add(a:int, b:int) -> int:
    return a + b

add(1,2)
add('a','b')
```

```
> mypy ex6_mypy_1.py
Success: no issues found in 1 source file
>
```

#type: ignore am Anfang der Datei nimmt die **Datei** von der Prüfung aus

code() #type: ignore nimmt die Zeile aus

Verwendung von mypy - typing.cast

ex6_mypy_2.py

```
from typing import cast

def add(a:int, b:int) -> int:
    return a + b
add(1,2)

a = cast(int, 'a')
b = cast(int, 'b')
add(a,b)
```

```
> mypy ex6_mypy_1.py
Success: no issues found in 1 source file
>
```

mypy konfigurieren

- Die Konfiguration legt fest
  - Welche **Prüfungen** durchgeführt und **Kriterien** angewendet werden
  - Welche **Datei/Module** untersucht werden
- Die Konfiguration kann im **Python Quellcode** oder einer **Konfigurationsdatei** liegen.

mypy konfigurieren - Beispiele

person_use.py

↓

person.py

↓

person_helper.py

```python
class Person:
    name: str
    age:int
    _stuff: List[Tuple[str,int]]

    def __init__(self,name:str,age:int) -> None:
        self.name = name
        self.age = age
        self._stuff = []

    def add_stuff(self,stuff:str,weight:int) -> None:
        self._stuff.append((stuff,weight))

    def get_weight(self) -> int:
        return sum(w for (_,w) in self._stuff)

    def help(self) -> None:
        ph.help(self)

    def get_any(self) -> Any:
        return None
```

```python
def help(person):
    return None
```

mypy Konfiguration

Ohne Konfiguration

```
> mypy person_use.py
Success: no issues found in 1 source file
```

Strikte Konfiguration

```
[mypy]
warn_redundant_casts = True
warn_unused_ignores = True
disallow_untyped_calls = True
disallow_untyped_defs = True
```

```
mypy --config-file mypy_strict.ini person_use.py
```

```
person_helper.py:3: error: Function is missing a type annotation
person.py:23: error: Call to untyped function "help" in typed context
Found 2 errors in 2 files (checked 1 source file)
```

mypy Konfiguration

Konfiguration mit Ausnahmen

```
[mypy]
disallow_untyped_calls = True
disallow_untyped_defs = True

[mypy-person]
disallow_untyped_calls = False

[mypy-person_helper]
ignore_errors = True
```

```
mypy --config-file mypy_ignore.ini person_use.py
Success: no issues found in 1 source file
```

Typen einführen

Wie Typen einführen (1)

- **Ziel definieren: Welche Vorteile versprechen wir uns?**
  - Fehler vermeiden, **Dokumentation**...
- **Wo nützen Typen am ehesten?**
  - Neuer Code?
  - Komplexer Code?
  - Code mit häufigen Änderungen?
- **Bei welcher Gelegenheit Typen einführen?**
  - Als eigene Tätigkeit
  - Bei Reviews, Onboarding
  - Bei Änderungen
  - Nur für Änderungen

Wie Typen einführen (2) - Kleiner Start

```
[mypy]
ignore_missing_imports = True
ignore_errors = True

[mypy-mycode.*]
ignore_errors = False
```

- Fangen Sie klein an
  - Ein geeignetes Teilprojekt
  - Minimale Checks

Wie Typen einführen (3) - Kleiner Start

```
[mypy]
ignore_missing_imports = True
ignore_errors = True

[mypy-mycode.*]
ignore_errors = False
no_implicit_optional = True
disallow_untyped_defs = True
disallow_untyped_calls = True

[mypy-othercode.*]
ignore_errors = False

[mypy-mycode.tricky.*]
ignore_errors = True
```

- Erweiteren Sie in zwei Richtungen
  - Neuen Code hinzunehmen
  - Checks erweitern
- Im Zweifelsfall: Bevorzugen Sie Breite über Tiefe
  - Nehmen Sie Packages aus der Überprüfung
- Verwenden Sie cast und #type: ignore

Wie Typen einführen (4) - Tipps

- Verwenden Sie Protocol / strukturelle Typen anstelle komplexer Klassen
  - Abstrakte Typen (Sequence anstelle List) sind nützlich als Parameter
- Verwenden Sie Typen als **verifizierbare Dokumentation** für Ihren Code
- Gehen Sie bei der Einführung opportunistisch vor.
- Experimentieren Sie mit Tooling. Es gibt auch Alternativen zu mypy

Vielen Dank