Funktionen advanced

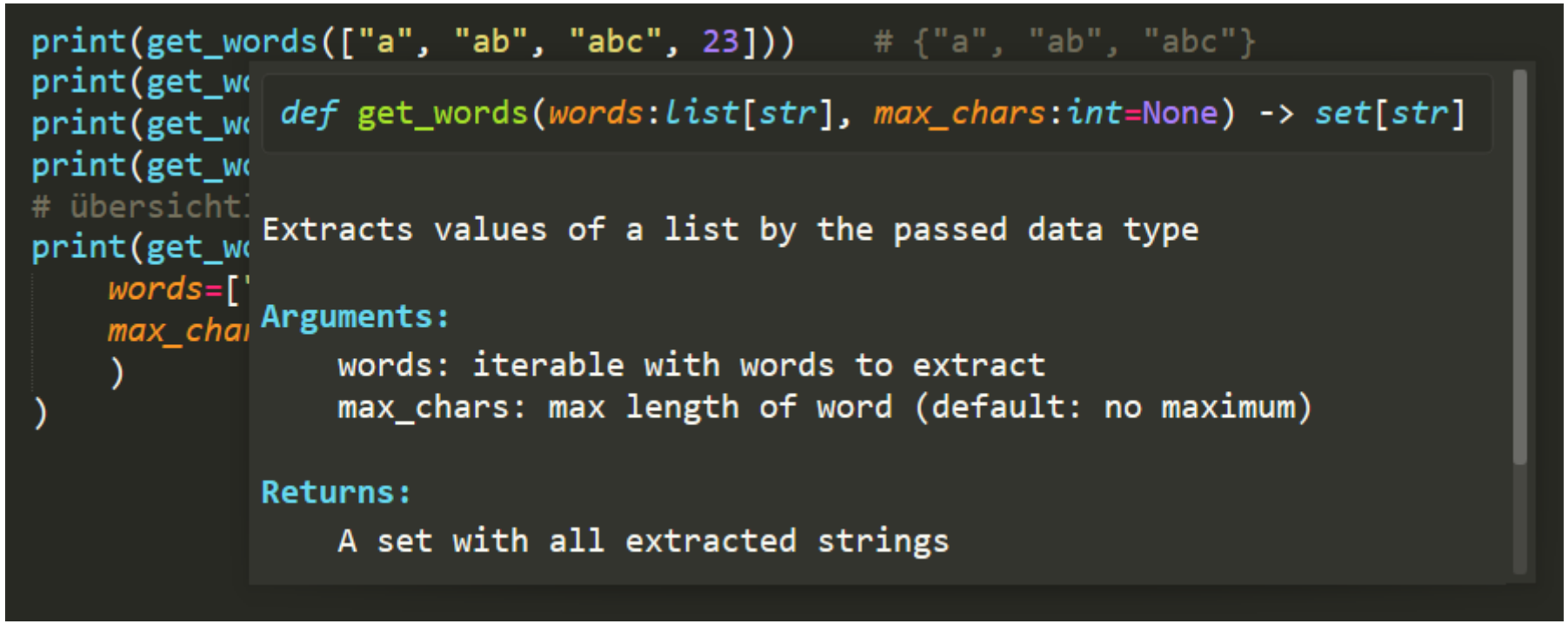
Funktionen können viel mehr

```python
def calculate_payroll(worked_hours:int, hourly_rate:float, currency:str="€") -> str:
    return str(worked_hours * hourly_rate) + currency

print(calculate_payroll(10, 12.98, "€")) # 129.8€
print(calculate_payroll(5, 74.0, ))   # 370.0€
print(calculate_payroll(23.45, 2, ""))   # 46.9
```

- Type hinting: *Hinweise* auf Parametertypen & Rückgabetyp geben
  - Keine Pflicht! Duck Typing entscheidet letztlich
- Default-Werte für Parameter (keyword-arguments)
  - Type hint und Defaultwert müssen nicht gleichen typ haben:
  - Häufig: x:int=None

```python
def get_words(words:list[str], max_chars:int=None) -> set[str]:
    """
    Extracts values of a list by the passed data type

    Arguments:
        words: iterable with words to extract
        max_chars: max length of word (default: no maximum)

    Returns:
        A set with all extracted strings

    Raises:
        TypeError: if `words` is not an iterable
    """
    if max_chars:
        return {w for w in words if type(w) == str and len(w) <= max_chars}
    else:
        return {w for w in words if type(w) == str}

print(get_words(["a", "ab", "abc", 23]))    # {"a", "ab", "abc"}
print(get_words(["a", "ab", "abc", 23], 2)) # {"a", "ab"}
print(get_words(23, 2))  # TypeError
print(get_words(words=["a", "ab", "abc"], max_chars=1)) # {"a"}
# übersichtlicher:
print(get_words(
    words=["a", "ab", "abc"],
    max_chars=1
    )
)
```

https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html


print(get_words(["a", "ab", "abc", 23]))   # {"a", "ab", "abc"}
print(get_wo
print(get_wo
print(get_wo
übersicht
print(get_wo
words=['
max_cha
)
)
def get_words(words:list[str], max_chars:int=None) -> set[str]
Extracts values of a list by the passed data type
Arguments:
words: iterable with words to extract
max_chars: max length of word (default: no maximum)
Returns:
A set with all extracted strings

*args

- Problemstellung: *Definiere eine Funktion zur Berechnung des Durchschnittswerts*
- Lösung:

```python
def avg(ints:list[int]) -> float:
    x = 0
    for elem in ints:
        x += elem
    return x / len(ints)
    # geht auch kürzer: return sum(ints) / len(ints)

print(avg([1, 2, 3])) # 2.0
print(avg([1, 2]))    # 1.5
print(avg([1]))       # 1.0
print(avg(1, 2, 3))
# TypeError: avg() takes 1 positional argument but 3 were given
```

- Anmerkung: Funktioniert (meistens) super, aber viele Klammern

*args

- Problemstellung: *Definiere eine Funktion zur Berechnung des Durchschnittswerts*
- Lösung:

```python
def avg(*args) -> float:
    x = 0
    for elem in args:
        x += elem
    return x / len(args)

print(avg(1, 2, 3))     # 2.0
print(avg(1, 2))        # 1.5
print(avg(1))           # 1.0
print(avg(*[1]))        # 1.0
print(avg(*[1, 2]))     # 1.5
print(avg(*[1, 2, 3]))  # 2.0
```

- Weniger Klammern -> weniger Fehlerpotential (meistens)
- Kommt mit Listen (mit *) **und** mit beliebiger Anzahl Parameter zurecht

*args

- args ist nur ein Name, kann auch anders genannt werden:
  - *ints, *strs, *floats …
- Entscheidend ist der **unpacking operator ***

```
x = [1, 2, 3]

print(*x) # 1 2 3
# entspricht:
print(x[0], x[1], x[2])
```

```
x = *x       #SyntaxError
*x[0] = 3    #SyntaxError
```

- args ist *nur* ein Tupel

```
def f(*args):
    return args

print(f(1, 2, 3)) # (1, 2, 3)
```

**kwargs

- Fast identisch zu *args aber:
  - Parameter müssen Name & Wert haben!
  - Z.b. f(x=3, y=10, z=20) statt f(3, 10, 20)
- kwargs ist somit ein **dict**:

```
def f(**kwargs):
    return kwargs

print(f(x=3, y=10))
# {'x': 3, 'y': 10}
```

Parameterreihenfolge

- Best practice: Fixe Parameter, dann *args, dann **kwargs

```python
def f(a, b, *args, **kwargs):
    pass

def f(a, b, **kwargs):
    pass

def f(*args, **kwargs):
    pass

def f(**kwargs):
    pass
```

```python
def f(**kwargs, *args):
    pass

def f(a, *args, b):
    pass
```

- Grundsätzlich kann man args und kwargs immer als letzte Parameter setzen

Use Cases

- Die meisten Funktionen nutzen kwargs für *Konfigurationen*:
  - https://docs.python.org/3/library/functions.html#print
- Übung 1: Verbindung herstellen mit optionalen Konfigurationen
- Übung 2: Funktionseigener Debugger

Wie machts range?

- Die range-“Funktion“ ist eigentlich keine Funktion:
  - https://docs.python.org/3/library/stdtypes.html#range
- Stop ist Pflicht; start & step nicht
- Wenn man start angibt, muss es der erste Wert sein
- Aufgabe: Definiere eine range-Funktion!