Documentatie en Argumenten

Bas Terwijn

March 13, 2025

1 Introductie

Dit document behandelt Docstrings als documentation en verschillende aspecten van argumenten van Python functies en een heel programma.

1.1 Doelen

- documentatie lezen
- zelf documentatie schrijven
- 'default argument values' gebruiken
- 'named arguments' gebruiken
- '*args and *kwargs' gebruiken
- 'command line arguments' gebruiken

2 Documentatie Lezen

De officieële Python documentatie is te vinden op https://docs.python.org, dus is dat een goed startpunt als u iets wilt opzoeken over Python. Maar soms is de informatie hier erg technisch of beknopt en helpt een websearch. Een andere manier om documentatie te vinden is door een Python interpreter te starten en de `help()` functie te gebruiken. Documentatie over de `print()` functie vindt u bijvoorbeeld met:

```
python # start the Python interpreter
>>> help(print)
```

wat deze uitvoer geeft:

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

Documentatie opvragen is ook mogelijk in Visual Studio Code door op een functie/variabele/type/... te klikken en "Ctrl-K Ctrl-I" ("Cmd-K Cmd-I" voor MacOS) te typen.

3 Zelf Documentatie Schrijven

Om zelf documentatie te schrijven gebruiken we het Docstrings formaat direct na de eerste regel van de functie-definitie. Een Docstring begint en eindigt met `"""` en beschrijft kort en bondig wat een functie doet. Beschrijf daarbij ook elke parameter en de eventuele return-waarde. Hier is een voorbeeld:

src/compute_distance.py

```python
import math

def compute_distance(coordinate: list[float]) -> float:
    """ Returns the Euclidean distance of a 'coordinate' to the
    origin using the Pythagoras theorem.
    """
    total = 0
    for c in coordinate:
        total += square(c)
    return math.sqrt(total)
```

Het is ook mogelijk om een Docstring te schrijven voor een module (Python file), class of methode om daarmee uit te leggen wat hun doel is.

3.1 Type Annotatie

Geef voor elk argument en eventuele return-waarde ook het type voor extra duidelijkheid door gebruik te maken van type annotations. In het voorbeeld schrijven we `: list[float]` na argument `coordinate` om het type van dit argument aan te geven, en schrijven we `-> float` aan het einde van de functie-declaratie om het type van de return-waarde aan te geven.

Met programma mypy kan eventueel daarna ook gebruikt worden om te controleren of uw functie met argumenten van het juiste type wordt aangeroepen, iets wat kan helpen bij het debuggen van uw code.

3.2 Docstring Extractie

Docstrings worden automatisch gelezen voor documentatie-doeleinden en zijn daarna beschikbaar in Visual Studio Code en met de `help()` functie:

```
python -i compute_distance.py # start interpreter and load compute_distance.py
>>>  help(compute_distance)
```

```
Help on function compute_distance in module __main__:

compute_distance(coordinate: list[float]) -> float
    Returns the Euclidean distance of a 'coordinate' to the
    origin using the Pythagoras theorem.
(END)
```

4 Default Argument Values

Python ondersteunt default argument values in functie-definities. Een default argument value is de waarde die een argument krijgt als de functie wordt aangeroepen zonder dat argument. In dit voorbeeld wordt `add()` aangeroepen eerst met 2, dan met 1, en dan met 0 argumenten:

src/add.py

```
def add(a=100, b=200):
    return a + b

print('add(1, 2):', add(1, 2) )  # calls add(  1,   2)
print('add(1):',    add(1)    )  # calls add(  1, 200)
print('add():',     add()     )  # calls add(100, 200)
```

De argumenten waarvoor geen waarde is gegeven krijgen dun hun default waarde zoals is te zien in de uitvoer:

```
add(1, 2): 3
add(1): 201
add(): 300
```

5 Named Arguments

Python ondersteunt het gebruik van named arguments in functie-aanroepen. Met named arguments kunt u de argumenten van een functie in willkeurige volgorde specificeren door de naam van het argument te geven, gevolgd door een gelijkteken en de waarde van het argument. Bijvoorbeeld bij het aanroepen van de `print()` functie:

```
print(*args, sep=' ', end='\n', file=None, flush=False)
```

kunt u schrijven:

src/hello_blue_planet.py

```
print("hello", "blue", "planet", end="!!!", sep="--")
```

wat dit als uitvoer geeft:

```
hello--blue--planet!!!
```

In dit voorbeeld krijgt argument `sep` de waarde `--` waardoor er twee min-tekens worden geprint tussen "hello", "blue" en "planet" en niet de default spatie. Argument `end` krijgt de waarde `!!!` waardoor er aan het einde drie uitroeptekens worden geprint en niet de default newline (de volgende print start nu dus op dezelfde regel). In programma:

src/print_to_file.py

```
print("hello", "blue", "planet", file=open("hello.txt", "w"), flush=True)
```

printen we met `file=open("hello.txt", "w")` niet naar de terminal maar naar bestand `hello.txt` en `flush=True` zorgt dat de tekst meteen in het bestand terecht komt en niet eerst tijdelijk in een snellere buffer wordt opgeslagen.

5.1 Default Value `None`

Met `file=None` wordt aangegeven dat argument `file` bij default geen waarde heeft. In de `print()` functie kan dan vervolgens worden besloten wat dat betekent met bijvoorbeeld:

```
if file is None:
    file = sys.stdout  # print to terminal when file is not given
```

6 Opdracht expenses.py

In bestand `assignments/expenses.py` vindt u werkende code waarin uitgaven worden bijgehouden. Als u het bestand uitvoert ziet u een lijst van uitgaven met daaronder het totaal:

```
===== list all expenses:
  10.5 Lunch                Food                2022-03-21
  17.5 Dinner               Food                2022-03-21
   8.0 Breakfast            Food                2022-03-22
  25.0 Groceries            Miscellaneous       2022-03-22
  12.5 Lunch                Food                2022-03-23
  15.0 Movie                Entertainment       2022-03-23
  31.0 Groceries            Miscellaneous       2022-03-24
  19.0 Dinner               Food                2022-03-24
total: 138.5
```

Lees het programmma en run het stap voor stap in de debugger zodat u begrijpt hoe het werkt.

6.1 Functie `add_expense()`

Pas nu de `add_expense()` functie aan zodat, als er geen waarde voor het `category` argument wordt opgegeven, een nieuwe uitgave als `category` de default waarde `"Miscellaneous"` krijgt. En als er geen waarde voor het `date` argument wordt opgegeven, een nieuwe uitgave als `date` de huidige datum krijgt (gebruik voor de huidige datum de `get_current_date()` functie). Gebruik hiervoor deze functie-declaratie:

```
def add_expense(expenses: list[tuple], amount: float, description: str,
                category: str="Miscellaneous", date: str=None):
```

6.2 Functies `list_expenses()` en `total_expenses()`

Voeg aan functies `list_expenses()` en `total_expenses()` het argument `category: str=None` toe. Als `category` een string waarde heeft dan moeten alleen de uitgaven van die categorie geprint en opgetelt worden. Als `category` de waarde `None` heeft, dan moeten alle uitgaven geprint en opgetelt worden.

Voeg ook het argument `date: str=None` toe aan beide functies. Als `date` een string waarde heeft dan moeten alleen de uitgaven op en na die datum geprint en opgetelt worden. Als `date` de waarde `None` heeft, dan moeten alle uitgaven geprint en opgetelt worden.

Tip: Het is een goed idee om het filteren van uitgaven in de `filter_expenses()` functie te implementeren zodat de code hergebruikt kan worden.
Tip: Gebruik de debugger om te zien wat er gebeurt als uw resultaten anders zijn dan verwacht.

6.3 Test

Test uw `expenses.py` bestand in de `assignment` directory met:

```
pytest -v test_expenses.py
```

7 Command Line Arguments

Niet alleen functies kunnen argumenten hebben, ook een heel programma kan argumenten hebben. Deze argumenten worden command line arguments genoemd en zijn beschikbaar in de `sys.argv` lijst na een `import sys`. Als we bijvoorbeeld deze code:

src/command_line_arguments.py

```
import sys

def main():
    print("arguments list:", sys.argv)

if __name__ == '__main__':
    main()
```

uitvoeren met:

```
python command_line_arguments.py  these are   test arguments
```

dan zien we output:

```
arguments list: ['command_line_arguments.py', 'these', 'are', 'test', 'arguments']
```

De eerste waarde in deze lijst is de naam van het programma, de volgende waarden zijn de argumenten die zijn meegegeven bij het starten van het programma gescheiden door spaties.

8 Opdracht

8.1 Command Line Arguments in Debugger

9 args kwargs

write `my_write(*args, *, /, sep=' ', end=\n', flush)`