Dein erstes Python-Programm

Ein seltsames Beispiel

Hier ist ein vollständiges, funktionierendes Python-Programm.

Es macht wahrscheinlich absolut keinen Sinn für dich, mach dir darüber keine Sorgen, wir werden es später Zeile für Zeile zerlegen und dann auch verstehen, was es macht.

Aber lies es dir einfach mal durch und schau, ob du etwas damit anfangen kannst.

odbchelper.py

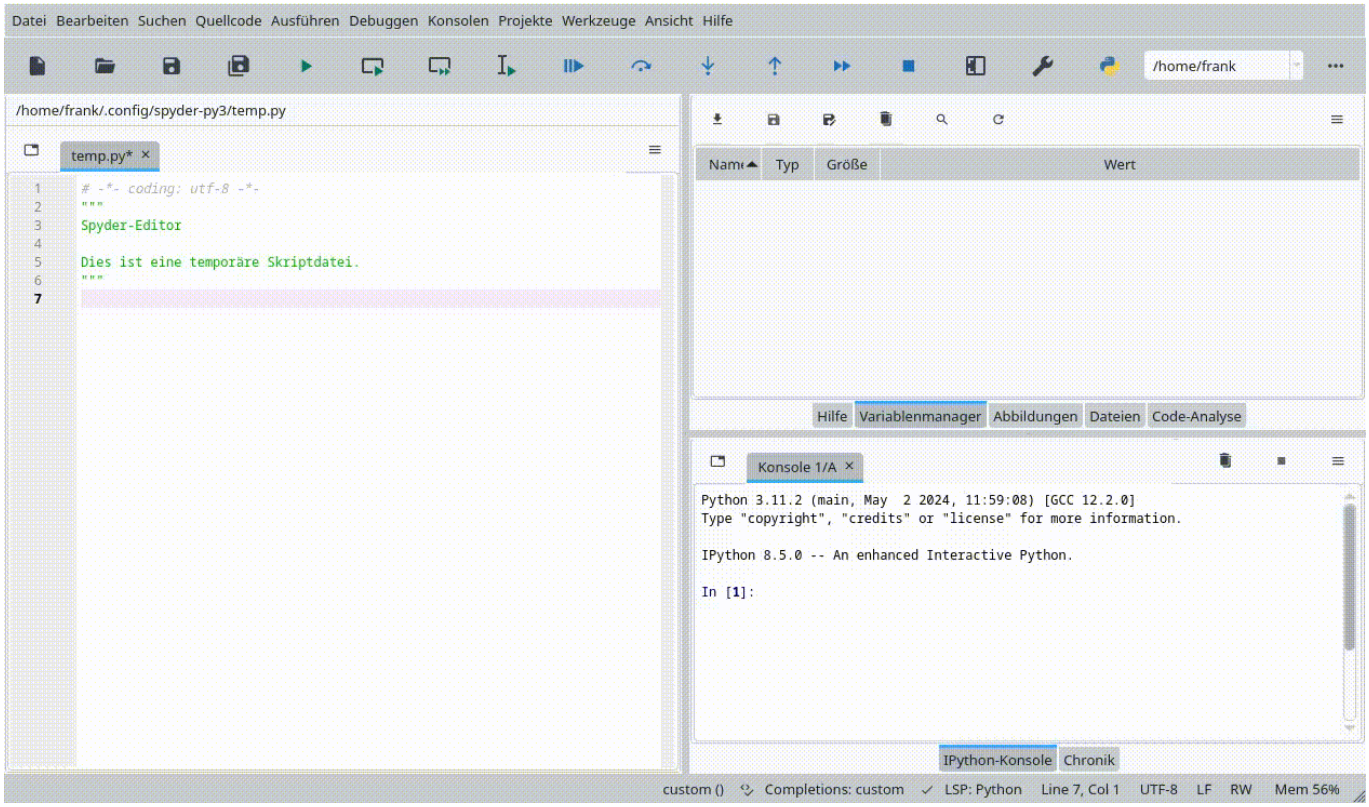
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 22 17:25:44 2024

@author: frank
"""

def build_connection_string(params):
    """
    Diese Funktion erzeugt einen eine Zeichenkette, die aus
    einer Liste mit Parametern erzeugt wird, um eine Verbindung
    zu einem Datenbankserver herzustellen
    """
    return ";".join(["%s=%s" % (k, v) for k, v in params.items()])

if __name__ == "__main__":
    # Parameter werden als sogenanntes "Dictionary" definiert
    myParams = {"server":"datenbank.qgm.com", \
                "database":"schueler", \
                "uid":"dbuser", \
                "pwd":"supergeheim" \
                }
    # Die Funktion wird aufgerufen, die gibt eine Zeichenkette
    # zurück, die mit dem print() Befehl direkt ausgegeben wird.
    print(build_connection_string(myParams))
```

Führe das Programm aus und beobachte, was passiert. Kopiere dazu den Code in ein neues Dokument in Spyder und klicke den grünen "Play"-Pfeil an:



Wenn alles klappt, sollte die Ausgabe folgende sein:

`server=datenbank.qgm.com;database=schueler;uid=dbuser;pwd=supergeheim`

Speichere die Datei nun unter dem Namen `odbchelper.py` ab. Der Name ist wichtig, weil wir ihn später nochmal benötigen!

Funktionen in Python

Um in Python Code in funktionale Einheiten zu gliedern, kann man Funktionen verwenden. Diese kann man einfach folgendermaßen deklarieren und dann benutzen:

```
def mach_etwas(parameter):
```

Das Schlüsselwort `def` leitet die Funktionsdeklaration ein, gefolgt vom Funktionsnamen - hier `mach_etwas`, gefolgt von den Argumenten – `parameter` – in Klammern. Mehrere Argumente (hier nicht gezeigt) werden durch Kommas getrennt.

Python-Funktionen geben keinen Datentyp für ihren Rückgabewert an, sie geben nicht einmal an, ob sie einen Wert zurückgeben oder nicht. Eigentlich gibt jede Python-Funktion einen Wert zurück: Wenn die Funktion eine return-Anweisung ausführt, gibt sie diesen Wert zurück, andernfalls gibt sie `None`, den Python Null-Wert zurück.

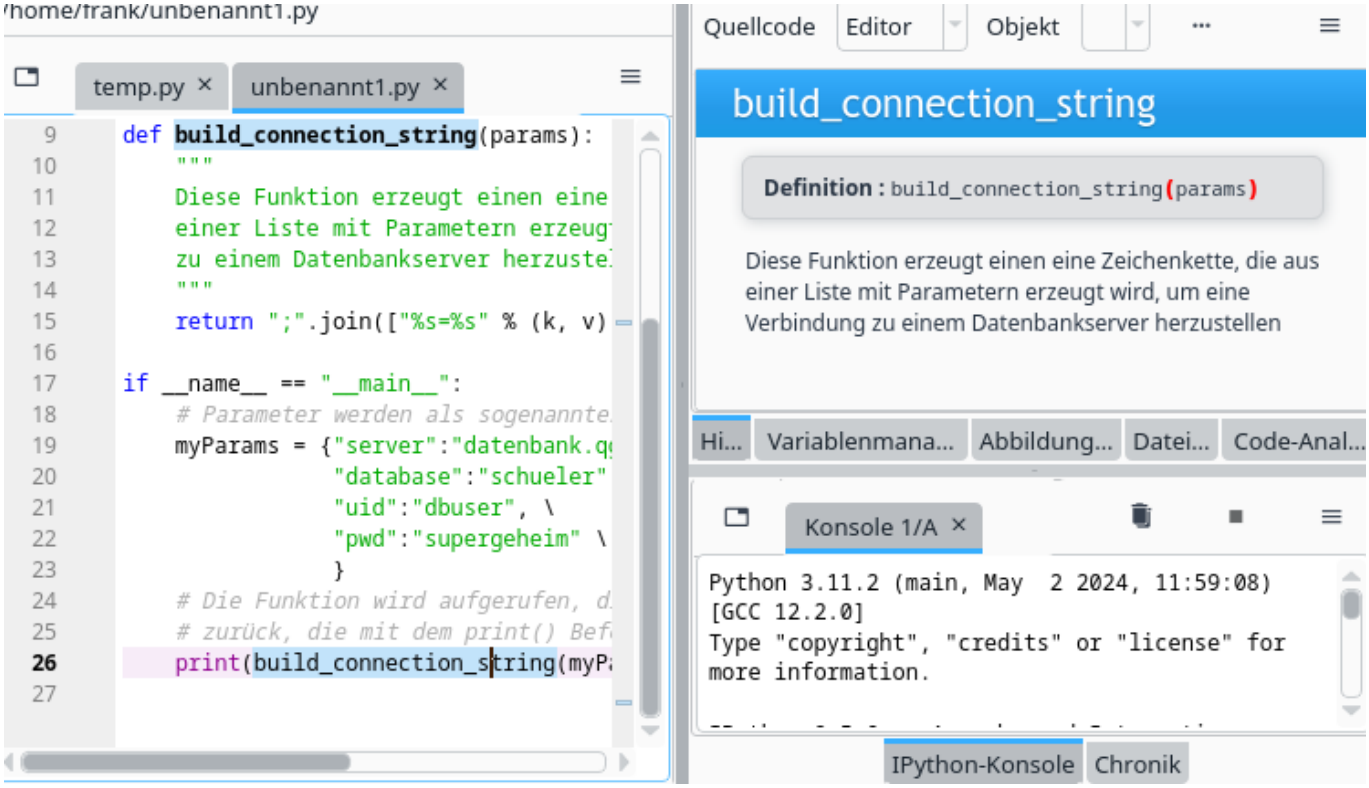
Kleiner Exkurs zu Typisierung in Python

Dokumentation von Funktionen

Du kannst eine Python-Funktion dokumentieren, indem du ihr einen **Docstring** gibst. Der Docstring von `build_connection_string` ist:

```
    """
    Diese Funktion erzeugt einen eine Zeichenkette, die aus
    einer Liste mit Parametern erzeugt wird, um eine Verbindung
    zu einem Datenbankserver herzustellen
    """
```

"Triple Quotes" schließen einen mehrzeiligen String ein. Alles zwischen den Anfangs- und Endzeichen gehört zu einem einzigen String, einschließlich Zeilenumbrüchen und anderen Anführungszeichen. Triple Quotes sind auch eine einfache Möglichkeit, einen String mit sowohl einfachen als auch doppelten Anführungszeichen zu definieren. Aber am häufigsten wirst du sie sehen, wenn ein Docstring definiert wird.



Alles zwischen den Triple Quotes ist der Docstring der Funktion, der dokumentiert, was die Funktion macht. Ein Docstring, falls vorhanden, muss das erste Element in einer Funktion sein (also das erste nach dem Doppelpunkt). Technisch gesehen musst du deiner Funktion keinen Docstring geben, aber du *solltest* es immer tun. Ich weiß, dass du das in jedem Programmierkurs gehört hast, den du jemals besucht hast, aber Python bietet einen zusätzlichen Anreiz: Der Docstring ist zur Laufzeit als Attribut der Funktion verfügbar.

Viele Python-IDEs verwenden den Docstring, um kontextbezogene Dokumentation bereitzustellen, sodass der Docstring als Tooltip angezeigt wird, wenn du einen Funktionsnamen eintippst. Das kann unglaublich hilfreich sein, aber es ist nur so gut wie die Docstrings, die du schreibst.

Alles ist ein Objekt

Die Aussage eben bedeutet, dass die Python-Funktion `Attribute` hat und dass diese Attribute zur Laufzeit verfügbar sind. Attribute sind Eigenschaften von Objekten - das bedeutet eine Funktion ist, wie alles andere in Python, ein Objekt.

Wie kann man aber nun auf den Docstring von `build_connection_string` zugreifen? Der Docstring einer Funktion ist stets im Attribut `__doc__` eines Funktions-Objekts gespeichert, man kann folgendermaßen darauf zugreifen:

```
>>> import odbchelper
>>> params = {"server":"mpilgrim", "database":"master", "uid":"sa",
"pwd":"secret"}
>>> print odbchelper.buildConnectionString(params)
server=mpilgrim;uid=sa;database=master;pwd=secret
>>> print odbchelper.buildConnectionString.__doc__
```

Du kannst das selbst in Spyder testen, indem du die Einfaben auf der IPython-Konsole an den `In-`Prompts eingibst. Die Promptzeichen >>> lässt du dabei weg:

Was ist ein Objekt?

Alles in Python ist ein Objekt, und fast alles hat Attribute und Methoden. Alle Funktionen haben ein eingebautes Attribut `__doc__`, das den im Quellcode der Funktion definierten Docstring zurückgibt. Das `sys`-Modul ist ein Objekt, das (unter anderem) ein Attribut `path` hat.

Das wirft dennoch die Frage auf: Was ist ein Objekt? Verschiedene Programmiersprachen definieren "Objekt" auf unterschiedliche Weise. In einigen bedeutet es, dass alle Objekte Attribute und Methoden haben **müssen**; in anderen bedeutet es, dass alle Objekte Vererbung beherrschen müssen. In Python ist die Definition lockerer; einige Objekte haben weder Attribute noch Methoden, und nicht alle Objekte beherrschen Verrerbung. **Aber alles ist ein Objekt im Sinne davon, dass es einer Variablen zugewiesen oder als Argument an eine Funktion übergeben werden kann**).

Das ist wirklich wichtig: **Alles in Python ist ein Objekt**. Strings sind Objekte. Listen sind Objekte. Funktionen sind Objekte. Sogar Module sind Objekte.