12. Programmentwurf

12.1 Formatierung

- Eine Anweisung pro Zeile
- Umbrechen wenn Zeilen zu lang
- Implizite Fortsetzung von öffnenden Klammern und Enrückung: Solange Klammer offen ist, kann man weitere Zeilen eifnügen.
- Beispiel:
```
foo = long_function_name(var_one, var_two
                                 vart three, var four)
```
- Progamme müssen kommentiert werden.
  - Programme werden häufiger gelesen als geschrieben.
  - auch der Programmierer vergisst...
  - in Englisch kommentieren, auf Grund des internationalen Datenaustauschs
  - nicht das offensichtliche kommentieren.
- sprechende Vraiblennamen verwenden.

12.2 Kommentare

- Fließtextkommentare
  `Anweisungen # Alles rechts von # wird als Kommentar betrachtet`
- Block-Kommentare: Zeilen, die mit einem Hash beginnen.
- docstring-Kommentare geben dem Programmierer informationen.
```
def fib(n)
    """ Computes the n-th Fibonacci number.
    The argument must be a posotive integer.
    """
```

In [10]:
```
def fib(n):
        """ Computes the n-th Fibonacci number.
        The argument must be a posotive integer.
        """
```

In [11]:
```
help (fib)
```

```
Help on function fib in module __main__:

fib(n)
    Computes the n-th Fibonacci number.
    The argument must be a posotive integer.
```

12.3 Typennotationen

- Typen der Variablen innerhalb von Funktionen werden durch **Typennotationen** angegeben.

- **Syntax:**
  -Variablen: `variable: type`
  -Rückgabewert: `(...)-> type`
- Beispiel

```
def results(
        max_points: int,
        percentage: int,
        points: int
    ) -> str:

    # fill with code
    return
```

- Das obere nennt man auch Funktionsgerüst.

12.3 Beispiele für Funktionswerte überlegen

- sinnvolle Beispiele erarbeiten.
- möglichst alle Auswahlmöglichkeiten sollten abgedeckt sein.
- Randfälle betrachten.
- am besten von Hand nachrechnen.
- Die Funktion assert prüft ob das Ergebnis wahr ist, sonst gibt es eine Fehlermeldung.
- Beispiele können später als Test verwendent werden.
- Beispiele:

```
assert(results(100,50,50 =="pass"))
    assert(results(100,50,30 =="fail"))
    assert(results(100,50,100 =="magna cum laude"))
```

12.4 Test der entwickelten Funktion

In [12]:
```
def results(
            max_points: int,
            percentage: int,
            points: int
            ) -> str:
        passed = (points >= max_points * percentage / 100)
        if passed:
            return 'pass'
        else:
            return 'fail'

assert(results(100,50,50 =="pass"))
assert(results(100,50,30 =="fail"))
assert(results(100,50,100 =="magna cum laude"))
```

12.5 Sonderfälle

- Was passiert wenn unvorhergesehene Werte in einer Funktion eingegeben werden, zum Beispiel negatvie Punkteanzanzahl

- **Lösung 1: Defensives Programmieren.**
  Alle unerwünschten Fälle werden abgefangen und eine Fehlermeldung wird erzeugt.
- **Lösung 2: Design by Contract.**
  Programmiere unter der Annahme, dass nur zulässige Fälle auftreten.

12.6 Main-Funktion

- Wird eine python-Datei importiert, werden sofort alle beinhaltete Funktionen ausgeführt.

Beispiel:

In [13]:
```
#foo.py
def main():
    print("Here ist foo")
main()
```

```
Here ist foo
```

In [14]:
```
#poo.py
import foo
def main():
    print("Hallo poo")

main()
```

```
Hallo poo
```

- um dieses Verhalten zu umgehen gibt es die Zeile `if __name__ = '__main__':`
- `__name__` heißt "Dunder name"
- Dunder bedeutet "double underscores"
- Variable `__name__` beinhaltet den Wert `__main__`, wenn die Datei direkt aufgerufen wurde

Beispiel:

In [15]:
```
#too.py

def main():
    print("Hallo poo")

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

In [16]:
```
#zoo.py
import too
def main():
    print("Hallo poo")

main()
```

Hallo poo