Programmstruktur

In der **prozeduralen Programmierung** wird ein Programm in eine Reihe von Funktionen oder Prozeduren unterteilt. Jede Funktion ist für eine bestimmte Aufgabe verantwortlich. Die Funktionen arbeiten auf Daten, die ihnen als Argumente übergeben werden, und geben Ergebnisse zurück. Hier ist ein einfaches Beispiel in Python:

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

def multiply(a, b):
    return a * b

x = add(5, 3)
y = multiply(4, 7)
```

In diesem Beispiel haben wir zwei Funktionen: `add` und `multiply`. Jede Funktion nimmt zwei Argumente entgegen und gibt ein Ergebnis zurück. Die Funktionen sind unabhängig voneinander und können in beliebiger Reihenfolge aufgerufen werden.

Im Gegensatz dazu basiert die **objektorientierte Programmierung (OOP)** auf dem Konzept der "Objekte", die Daten und Methoden zur Manipulation dieser Daten enthalten können. In OOP wird ein Programm als eine Sammlung von Objekten modelliert, die miteinander interagieren. Hier ist ein einfaches Beispiel in Python:

```python
class Calculator:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        return self.a + self.b

    def multiply(self):
        return self.a * self.b

calc = Calculator(5, 3)
x = calc.add()
y = calc.multiply()
```

In diesem Beispiel haben wir eine Klasse `Calculator` mit zwei Methoden: `add` und `multiply`. Jede Methode arbeitet auf den Daten (a und b), die im Objekt gespeichert sind. Die Methoden können nur in dem Kontext eines `Calculator`-Objekts aufgerufen werden.

Zusammenfassend lässt sich sagen, dass die prozedurale Programmierung den Fokus auf die Aktionen legt **(was soll getan werden)**, während die objektorientierte Programmierung den Fokus auf die Objekte legt **(wer tut was)**.

Ansatz

Der **Top-Down-Ansatz** in der prozeduralen Programmierung beginnt mit einer allgemeinen Übersicht des Problems, das in kleinere und kleinere Teile unterteilt wird, bis jedes Teil einfach genug ist, um direkt implementiert zu werden. Dieser Ansatz wird oft mit dem Schreiben von Pseudocode oder dem Erstellen eines Flussdiagramms beginnen, um den allgemeinen Ablauf des Programms zu planen. Hier ist ein einfaches Beispiel:

```
# Top-Level-Funktion
def main():
    daten = daten_eingeben()
    ergebnis = berechnung(daten)
    ergebnis_ausgeben(ergebnis)

# Unterfunktionen
def daten_eingeben():
    # Code zum Einlesen der Daten
    pass

def berechnung(daten):
    # Code zur Durchführung der Berechnung
    pass

def ergebnis_ausgeben(ergebnis):
    # Code zum Ausgeben des Ergebnisses
    pass

# Aufruf der Top-Level-Funktion
main()
```

Im Gegensatz dazu beginnt der **Bottom-Up-Ansatz** in der objektorientierten Programmierung oft mit der Implementierung von kleinen Teilen (den Klassen und Methoden), die dann zu größeren Einheiten (den Objekten) zusammengesetzt werden. Dieser Ansatz konzentriert sich auf die Implementierung der kleinsten Einheiten zuerst und baut dann das Gesamtprogramm auf, indem diese Einheiten zu komplexeren Strukturen zusammengesetzt werden. Hier ist ein einfaches Beispiel:

```
# Klassendefinitionen
class Auto:
    def __init__(self, marke, modell):
        self.marke = marke
        self.modell = modell

    def hupen(self):
        print("Hup, hup!")

class Garage:
    def __init__(self):
        self.autos = []
```

```
    def auto_hinzufuegen(self, auto):
        self.autos.append(auto)

    def alle_autos_hupen(self):
        for auto in self.autos:
            auto.hupen()

# Erstellen von Objekten und Aufrufen von Methoden
auto1 = Auto("Toyota", "Corolla")
auto2 = Auto("Honda", "Civic")

garage = Garage()
garage.auto_hinzufuegen(auto1)
garage.auto_hinzufuegen(auto2)

garage.alle_autos_hupen()
```

In diesem Beispiel haben wir zuerst die kleinsten Einheiten (die Klassen `Auto` und `Garage`) implementiert und dann diese Einheiten zu einem größeren Programm zusammengesetzt, indem wir Objekte erstellt und ihre Methoden aufgerufen haben. Dies ist ein typisches Beispiel für einen Bottom-Up-Ansatz.

Datensicherheit

In der **prozeduralen Programmierung** sind Daten und Funktionen getrennt. Es gibt keine eingebauten Mechanismen, um den Zugriff auf die Daten zu beschränken. Das bedeutet, dass alle Teile des Programms auf alle Daten zugreifen können. Dies kann zu Problemen führen, wenn ein Teil des Programms versehentlich Daten ändert, die es nicht ändern sollte.

In diesem Beispiel kann die Variable `x` von überall im Code geändert werden, was zu unerwarteten Ergebnissen führen kann.

```
# Daten
x = 10

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

# Zugriff auf x von außerhalb der Funktion
y = add(x, 5)
x = 20  # x kann von überall im Code geändert werden
```

Im Gegensatz dazu bietet die **objektorientierte Programmierung (OOP)** Mechanismen zur Kontrolle des Zugriffs auf Daten. In Python gibt es zwar keine strikten Zugriffsspezifizierer wie `private`, `public` und `protected` wie in einigen anderen OOP-Sprachen, aber es gibt Konventionen, die verwendet werden können, um Daten zu schützen.

In Python wird ein Unterstrich (_) vor dem Namen eines Attributs oder einer Methode verwendet, um anzugeben, dass es sich um ein internes Detail der Klasse handelt und nicht von außerhalb der Klasse zugegriffen werden sollte. Zwei Unterstriche (__) vor dem Namen führen zu einer Namensänderung, die es schwieriger macht, auf das Attribut zuzugreifen (obwohl es immer noch möglich ist).

Hier ist ein Beispiel:

```
class MyClass:
    def __init__(self):
        self.__private_attribute = 10  # Dies sollte nicht von außerhalb der Klasse zugegriffen werden

    def get_private_attribute(self):
        return self.__private_attribute

obj = MyClass()
print(obj.get_private_attribute())  # Zugriff auf das private Attribut über eine Methode
```

In diesem Beispiel ist `__private_attribute` ein privates Attribut. Es sollte nicht direkt von außerhalb der Klasse zugegriffen werden. Stattdessen bieten wir eine Methode `get_private_attribute` an, um auf sichere Weise auf das Attribut zuzugreifen.

Diese Art von Datenkapselung und Informationssicherheit ist ein Schlüsselmerkmal der OOP und trägt dazu bei, den Code sicherer und besser organisiert zu machen. Es verhindert, dass Daten versehentlich in unsachgemäßer Weise geändert werden, und erleichtert die Wartung und Erweiterung des Codes.

Daten und Funktionen:

In der **prozeduralen Programmierung** sind Daten und Funktionen getrennt. Daten werden in Variablen gespeichert und Funktionen arbeiten auf diesen Daten. Die Funktionen können Daten als Argumente akzeptieren und Ergebnisse zurückgeben. Hier ist ein einfaches Beispiel in Python:

```
# Daten
x = 10
y = 20

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

# Verwendung der Funktion
z = add(x, y)
```

In diesem Beispiel sind x und y Daten und add ist eine Funktion, die auf diesen Daten arbeitet. Die Daten und die Funktion sind getrennt: Die Funktion add weiß nichts über x und y, es sei denn, sie werden als Argumente übergeben.

Im Gegensatz dazu kapselt die **objektorientierte Programmierung (OOP)** Daten und Methoden in Klassen. Eine Klasse definiert eine Struktur für Objekte, die sowohl Daten (in Form von Attributen) als auch Methoden enthalten können, die auf diesen Daten arbeiten. Hier ist ein einfaches Beispiel in Python:

```python
class Punkt:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def verschieben(self, dx, dy):
        self.x += dx
        self.y += dy

# Erstellen eines Punktes
p = Punkt(10, 20)

# Verwendung der Methode
p.verschieben(5, 5)
```

In diesem Beispiel ist `Punkt` eine Klasse, die Daten (`x` und `y`) und eine Methode (`verschieben`) enthält. Die Methode `verschieben` arbeitet direkt auf den Daten `x` und `y`, die im Objekt gespeichert sind. Die Daten und die Methode sind in der Klasse gekapselt und bilden eine Einheit.

Diese Art der Datenkapselung ist ein Schlüsselmerkmal der OOP und trägt dazu bei, den Code besser organisiert und leichter zu verstehen zu machen. Es ermöglicht auch eine höhere Stufe der Datenabstraktion und kann dazu beitragen, Fehler zu vermeiden, indem es sicherstellt, dass nur die Methoden der Klasse auf ihre Daten zugreifen können.

Überladung und Vererbung.

In der prozeduralen Programmierung sind Überladung und Vererbung nicht möglich, während sie in der OOP möglich sind. (Dazu Später mehr)

Programmgröße

Die prozedurale Programmierung wird für mittelgroße Programme verwendet, während die OOP für große und komplexe Programme verwendet wird.

Weltanschauung

Die **prozedurale Programmierung** basiert auf einer "unrealen" Welt in dem Sinne, dass sie sich auf die Durchführung von Aufgaben und die Manipulation von Daten konzentriert. Sie denkt in Bezug auf Funktionen oder Prozeduren, die auf Daten operieren. In der realen Welt interagieren wir jedoch nicht direkt mit isolierten Funktionen oder Prozeduren, sondern mit Objekten, die sowohl Zustände (Daten) als auch Verhaltensweisen (Methoden) haben.

Zum Beispiel, wenn Sie in der realen Welt ein Auto fahren, denken Sie nicht an eine Reihe von Funktionen wie `starteMotor()`, `beschleunige()`, `lenkeLinks()`, etc. Stattdessen interagieren Sie mit einem Auto-Objekt, das diese Funktionen (jetzt Methoden genannt) kapselt und auch Zustände wie Geschwindigkeit, Kraftstoffmenge, etc. hat.

Die **objektorientierte Programmierung (OOP)** versucht, diese reale Welt genauer abzubilden, indem sie Konzepte wie Klassen, Objekte, Vererbung, Polymorphie usw. einführt. In OOP erstellen wir Klassen, die als Blaupausen für Objekte dienen, und diese Objekte haben sowohl Zustände (Attribute) als auch Verhaltensweisen (Methoden). Dies ermöglicht es uns, reale Welt-Entitäten und ihre Interaktionen genauer und intuitiver zu modellieren.

Zusammenfassend lässt sich sagen, dass die prozedurale Programmierung eine eher funktionsorientierte Sichtweise hat, während die objektorientierte Programmierung eher eine Sichtweise hat, die näher an der Art und Weise liegt, wie wir die reale Welt wahrnehmen und mit ihr interagieren. Beide Ansätze haben ihre Stärken und Schwächen und sind für verschiedene Arten von Problemen geeignet. Es hängt von den spezifischen Anforderungen des Projekts ab, welcher Ansatz am besten geeignet ist.