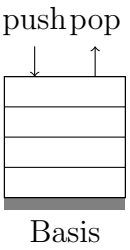
Datenstrukturen: Stack (Stapelspeicher) Lösung

Ein *Stack* ist eine dynamische Datenstruktur, welche im wesentlichen die folgenden drei Operationen unterstützt.

- *`push(item)`*: Ein Objekt (`item`) auf den Stack legen
- *`pop()`*: Das oberste Objekt zurückgeben und es entfernen.
- *`peek()`*: Das oberste Objekt zurückgeben, ohne es zu entfernen.



Kurzform des Stack-Prinzips: *Last In First Out* (LIFO)

Anwendungen

- Browser-History
- Undo-Funktion von Anwendungsprogammen
- Auswertung von Postifix-Ausdrücken
- Übersetzung von Infix-Ausdrücken in Postfix-Form
- Verwaltung des Arbeitsspeichers von Computern
- Backtracking-Algorithmen

Python-Implementation eines Stacks

Die Klasse `Stack` wird als Python-Liste mit diesen Methoden implementiert:

- Der Konstruktor `s = Stack()` erzeugt einen leeren Stack.
- `s.push(item)` legt `item` auf den Stack ab.
- `s.pop()` entfernt das oberste Element vom Stack und liefert es als Wert zurück.
- `s.peek()` gibt das oberste Stackelement als Wert zurück ohne es zu entfernen.
- `s.is_empty()` gibt `True` bzw. `False` zurück, wenn der Stack leer bzw. nichtleer ist.
- `s.size()` gibt die Anzahl der Elemente im Stack zurück.

Der Quellcode

```
class Stack:

    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[-1]

    def is_empty(self):
        return self.items == []

    def size(self):
        return len(self.items)
```

Anwendung

Die Funktion `parencheck(string)` überprüft, ob die runden Klammern (engl. *parenthesis*) im Argument `string` korrekt gesetzt sind.

Lösungsidee: Erzeuge einen leeren Stack `s`. Durchlaufe die Zeichenkette zeichenweise. Ist das Symbol eine öffnende Klammer, kommt es auf den Stack. Ist das Symbol eine schliessenden Klammer, so entferne das oberste Stackelement. Überprüfe, ob diese Operation auf einem leeren Stack ausgeführt wird. Wenn ja, gibt es mehr schliessende als öffende Klammern und der Ausdruck ist falsch. Nachdem der gesamte String verarbeitet wurde, ist noch zu prüfen, ob noch Klammern auf dem Stack liegen. Wenn ja, dann gibt es mehr öffnende als schliessende Klammern und der Ausdruck ist falsch. Andernfalls ist die Klammerung korrekt.

```
from stack import Stack

def parencheck(string):
    s = Stack()
    for symbol in string:
        if symbol == '(':
            s.push(symbol)
        if symbol == ')':
            if s.is_empty():
                return False
            else:
                s.pop()
    return s.is_empty() # True, falls Stack leer
```