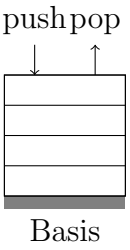
Datenstrukturen: Stack (Stapelspeicher)

- Last In First Out (LIFO)
- Hinzufügen (push) und Entfernen (pop) erfolgen von derselben Seite
- Anwendung: Browser-History; Undo-Funktion von Anwendungsprogammen; Auswertung von Postifix-Ausdrücken und Übersetzung von Infix-Ausdrücken in Postfix-Form; Backtracking-Algorithmen; Verwaltung des Arbeitsspeichers von Computern



Python-Implementation eines Stacks

Die Klasse `Stack` wird als Python-Liste mit den folgenden 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()` liefert das oberste Element des Stacks als Wert zurück ohne es zu entfernen.
- *s*`.is_empty()` liefert `True` bzw. `False` als Rückgabewert, wenn der Stack leer bzw. nicht-leer ist.

Ferner überschreiben wir die Spezialfunktion `__len()__`, damit uns `len(`*s*`)` die Grösse des Stacks *s* zurückgibt.

Der Quellcode

```
1  class Stack:
2
3      def __init__(self):
4          self.items = []
5
6      def push(self, item):
7          self.items.append(item)
8
9      def pop(self):
10         return self.items.pop()
11
```

```
12      def peek(self):
13          return self.items[-1]
14
15      def is_empty(self):
16          return self.items == []
17
18      def __len__(self):
19          return len(self.items)
20
21      def __str__(self):
22          return ' '.join(str(x) for x in self.items) + ' <'
```

Anwendung

Schreibe eine Funktion `check(string)`, das einen String als Argument entgegen nimmt und überprüft, ob die (runden) Klammern in diesem 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.

```
1  from stack import Stack
2
3  def parencheck(string):
4
5      s = Stack()
6
7      for symbol in string:
8          if symbol == '(':
9              s.push(symbol)
10         if symbol == ')':
11             if s.is_empty():
12                 return False
13             else:
14                 s.pop()
15
16     return s.is_empty() # True, falls Stack leer
17
18 # Test
19 expr1 = '(()()(()()))'
20 expr2 = '(()()())()))'
21 print(parencheck(expr1))
22 print(parencheck(expr2))
```