4.7. Balanced Symbols (A General Case)

The balanced parentheses problem shown above is a specific case of a more general situation that arises in many programming languages. The general problem of balancing and nesting different kinds of opening and closing symbols occurs frequently. For example, in Python square brackets, `[` and `]`, are used for lists; curly braces, `{` and `}`, are used for dictionaries; and parentheses, `(` and `)`, are used for tuples and arithmetic expressions. It is possible to mix symbols as long as each maintains its own open and close relationship. Strings of symbols such as

```
{ { ( [ ] [ ] ) } ( ) }

[ [ { { ( ( ) ) } } ] ]

[ ] [ ] [ ] ( ) { }
```

are properly balanced in that not only does each opening symbol have a corresponding closing symbol, but the types of symbols match as well.

Compare those with the following strings that are not balanced:

```
( [ ) ]

( ( ( ) ] ) )

[ { ( ) ]
```

The simple parentheses checker from the previous section can easily be extended to handle these new types of symbols. Recall that each opening symbol is simply pushed on the stack to wait for the matching closing symbol to appear later in the sequence. When a closing symbol does appear, the only difference is that we must check to be sure that it correctly matches the type of the opening symbol on top of the stack. If the two symbols do not match, the string is not balanced. Once again, if the entire string is processed and nothing is left on the stack, the string is correctly balanced.

The Python program to implement this is shown in ActiveCode 1. The only change appears in line 16 where we call a helper function, `matches`, to assist with symbol-matching. Each symbol that is removed from the stack must be checked to see that it matches the current closing symbol. If a mismatch occurs, the boolean variable `balanced` is set to `False`.

Run Load History

```
1 from pythonds.basic import Stack
2
3 def parChecker(symbolString):
4     s = Stack()
5     balanced = True
6     index = 0
7     while index < len(symbolString) and balanced:
8         symbol = symbolString[index]
9         if symbol in "([{":
10            s.push(symbol)
```

(SimpleBalancedParentheses.html) (ConvertingDecim

```
11          else:
12              if s.isEmpty():
13                  balanced = False
14              else:
```

Activity: 4.7.1 Solving the General Balanced Symbol Problem (parcheck2)

These two examples show that stacks are very important data structures for the processing of language constructs in computer science. Almost any notation you can think of has some type of nested symbol that must be matched in a balanced order. There are a number of other important uses for stacks in computer science. We will continue to explore them in the next sections.

You have attempted 1 of 2 activities on this page

user not logged in

(SimpleBalancedParentheses.html) (ConvertingDecim