

# PL - HW3 - Tail Recursion

## SBCL:

Non-tail recursive:

```
(defun factorial (n)
  (if (zerop n)
      1
      (* n (factorial (- n 1)))))
```

```
(defun factorial (n)
  (if (zerop n)
      1
      (* n (factorial (- n 1)))))
(format t "~d" (factorial 64000))
```

STDIN

Input for the program ( Optional )

```
65: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8850
66: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8850
67: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8870
68: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8890
69: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd88b0
70: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd88d0
71: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd88f0
72: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8910
73: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8930
74: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8950
75: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8970
76: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8990
77: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd89b0
78: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd89d0
79: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd89f0
80: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8a10
81: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8a30
82: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8a50
83: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8a70
84: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8a90
85: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8ab0
86: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8ad0
87: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8af0
88: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8b10
89: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8b30
90: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8b50
91: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8b70
92: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8b90
93: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8bb0
94: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8bd0
95: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8bf0
96: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8c10
97: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8c30
98: CL-USER::FACTORIAL, pc = 0x52b1c75f, fp = 0x7fb252dd8c50
```

fatal error encountered in SBCL pid 3(tid 0x7fb25355cb80):  
Control stack exhausted, fault: 0x7fb252dd7ff8, PC: 0x52b1c72f

Tail recursive:

```
(defun factorial (n &optional (acc 1))
  (if (zerop n)
      acc
      (factorial (- n 1) (* acc n))))
```

Code	STDIN
<pre>(defun factorial (n &amp;optional (acc 1))   (if (zerop n)       acc       (factorial (- n 1) (* acc n))))  (format t "~d" (factorial 100000))</pre>	<p>Input for the program ( Optional )</p> <hr/> <p>Output:</p> <p>282429407960347874293421578024535518477494926091224850578918086542977</p>

## Python:

Non-tail recursive:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

[illegible]

Tail recursive:

```
def factorial(n, acc=1):
    if n == 0:
        return acc
    return factorial(n-1, n*acc)
```

Input for the program ( Optional )

[illegible]

When executed in SBCL, the tail recursive implementation enables computation of larger factorials without encountering stack overflow issues, whereas the non-tail recursive counterpart overflows the stack at approximately a factorial of 64000.

In Python, both implementations fail due to stack overflow at around 1000, as Python lacks tail call optimization.

Thus, SBCL offers support for proper tail call optimization, unlike Python. Consequently, the tail recursive approach in SBCL can compute factorials indefinitely, while in Python, it offers no advantage over the non-tail recursive version.