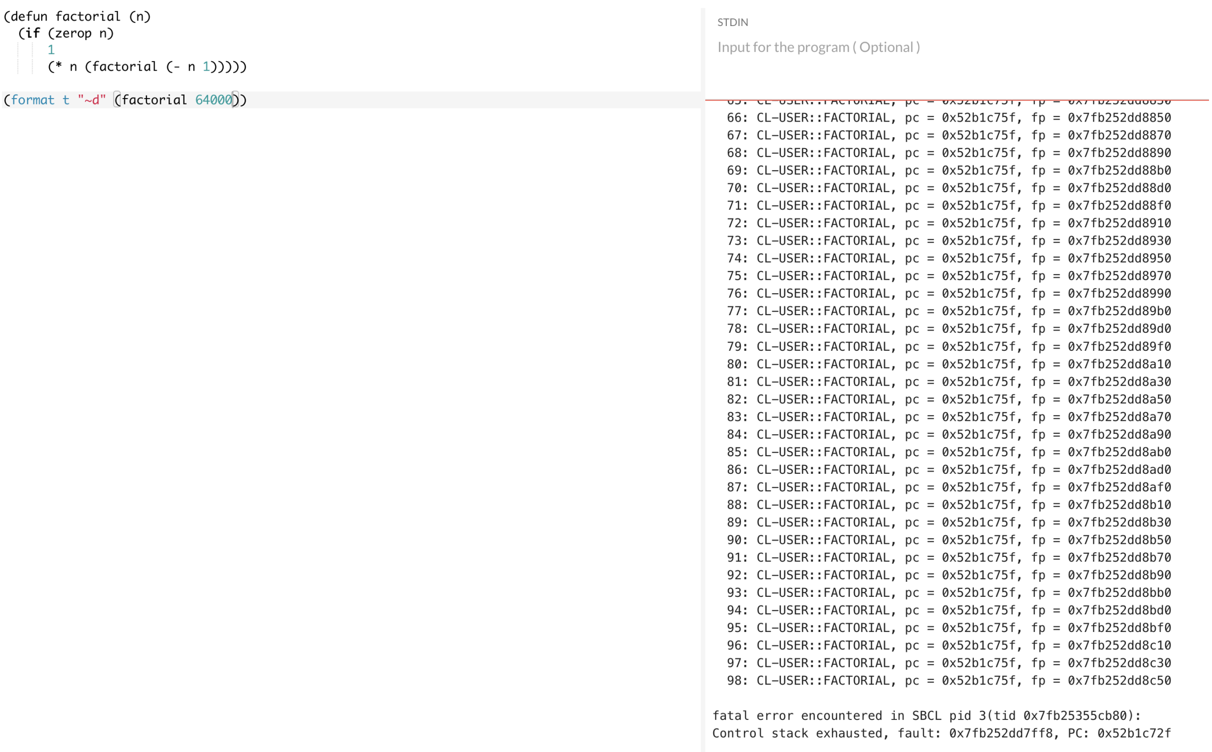
PL - HW3 - Tail Recursion

SBCL:

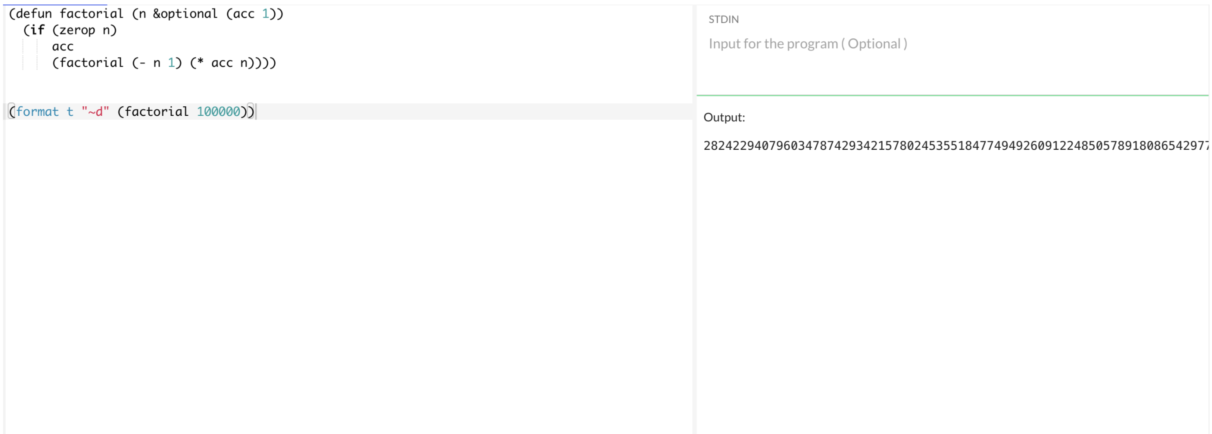
Non-tail recursive:

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



Tail recursive:

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



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


(format t "~d" (factorial 100000))
```

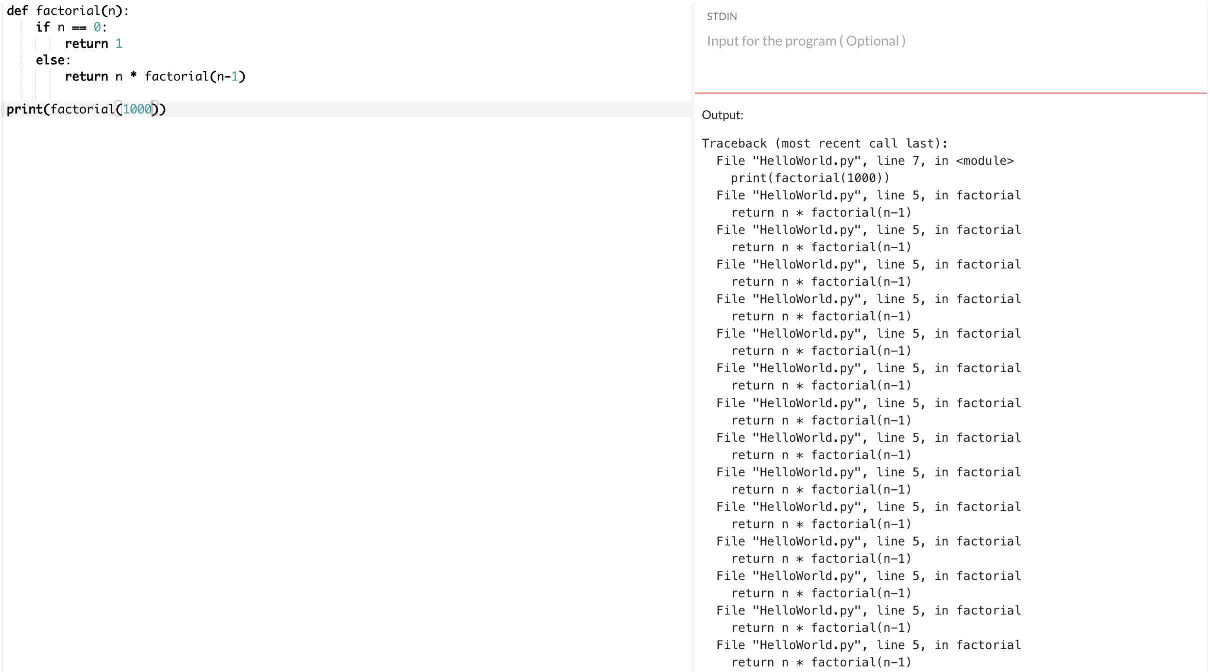
```
STDIN
Input for the program ( Optional )

Output:
2824229407960347874293421578024535518477494926091224850578918086542977
```

Python:

Non-tail recursive:

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



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

print(factorial(1000))
```

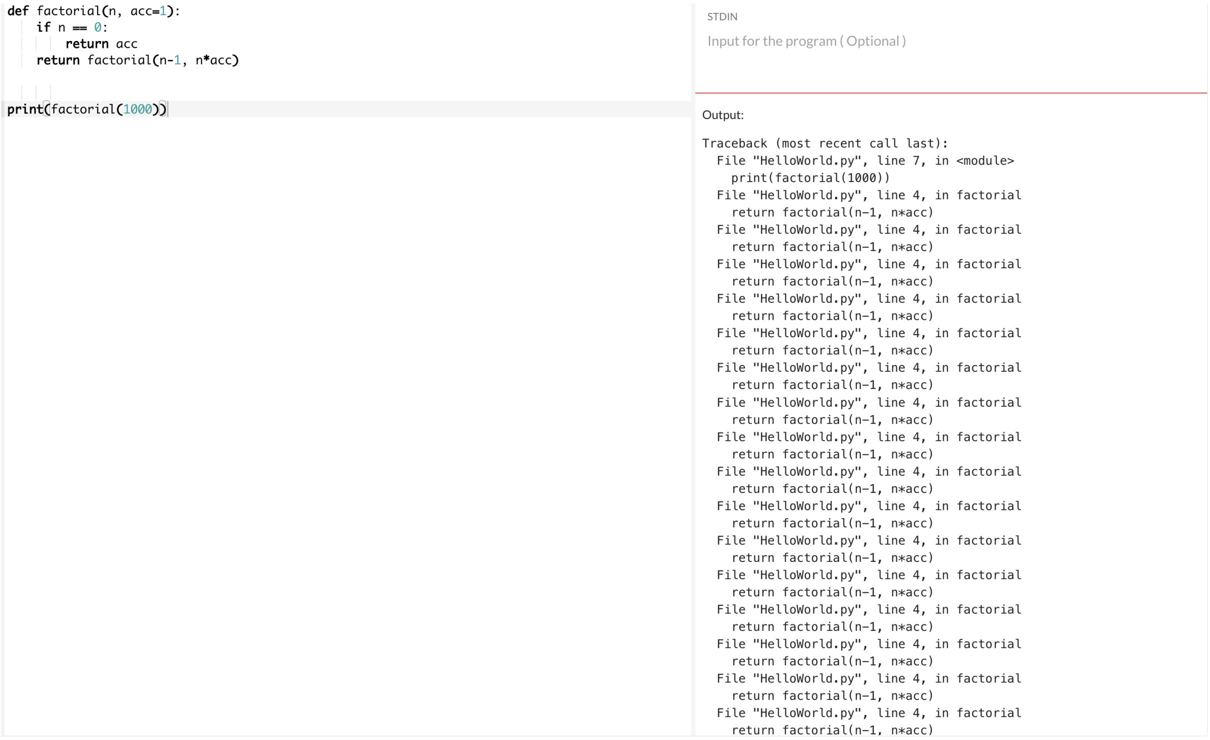
```
STDIN
Input for the program ( Optional )

Output:

Traceback (most recent call last):
  File "HelloWorld.py", line 7, in <module>
    print(factorial(1000))
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
  File "HelloWorld.py", line 5, in factorial
    return n * factorial(n-1)
```

Tail recursive:

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



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.