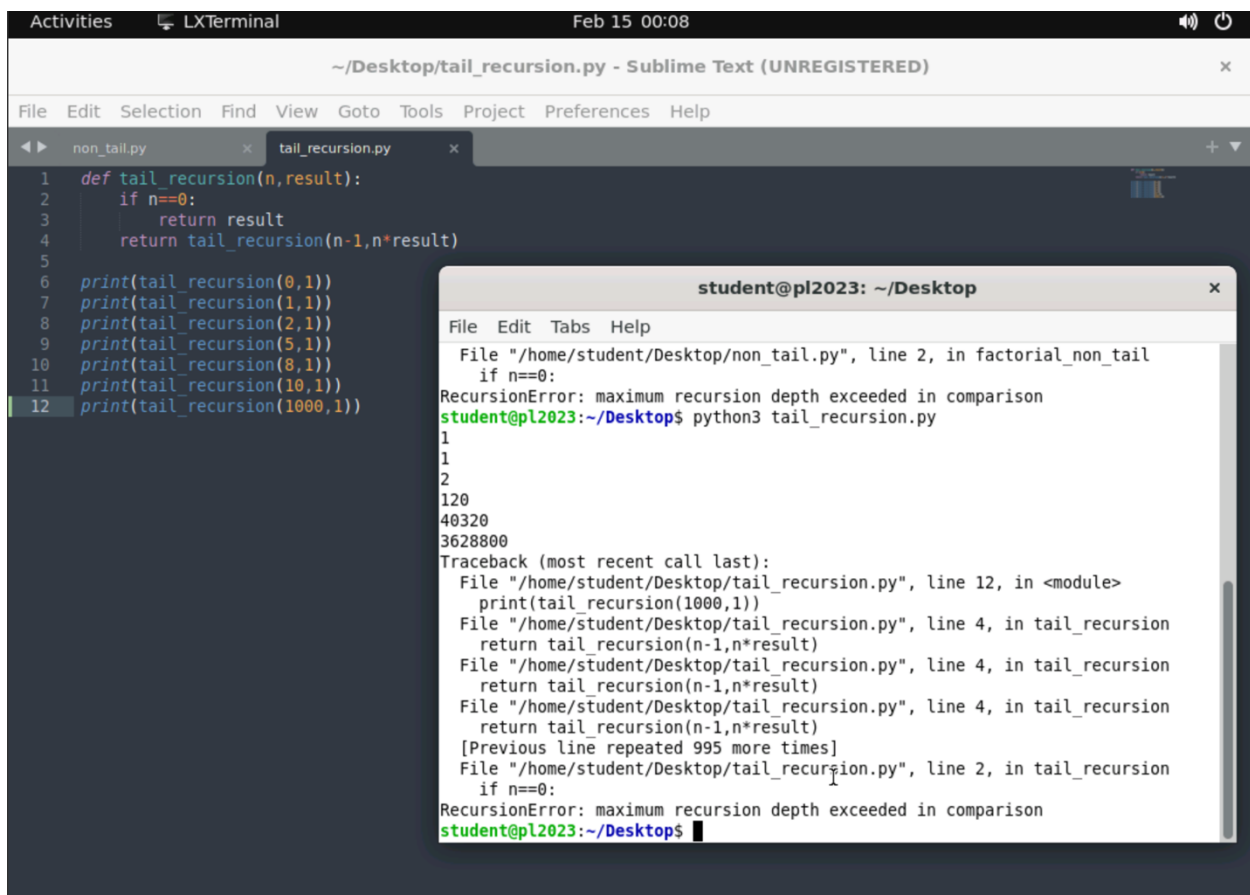


## Tail Recursion

In SBCL tail-recursive and non-tail-recursive version were able to handle larger inputs as compared to Python because in python we eventually hit system limits on the stack size which depicts that SBCL supports tail recursive optimization whereas Python doesn't.

Python

Tail recursion



The screenshot shows a Sublime Text editor window titled "~ / Desktop / tail\_recursion.py - Sublime Text (UNREGISTERED)". The editor has two tabs: "non\_tail.py" and "tail\_recursion.py". The "tail\_recursion.py" tab is active, showing the following Python code:

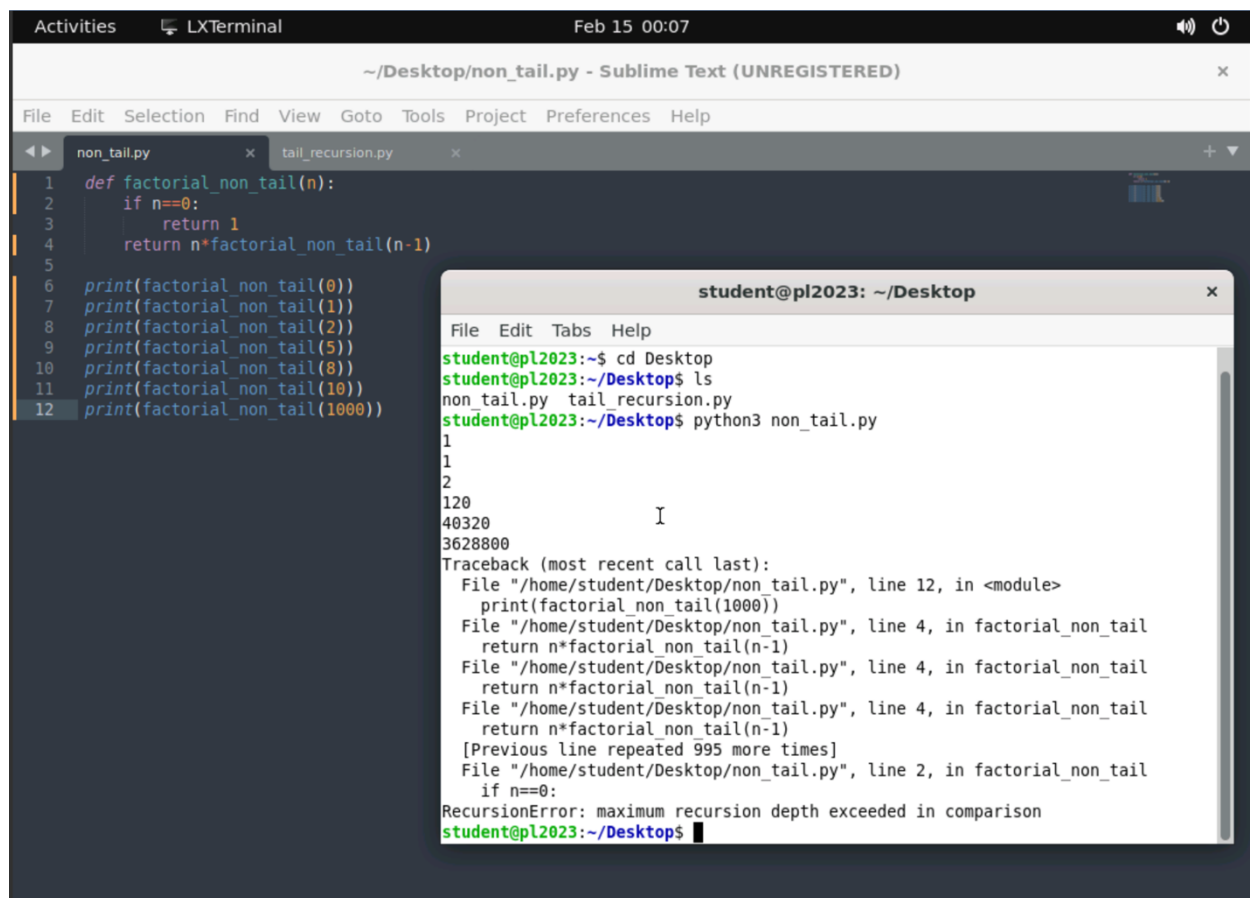
```
1 def tail_recursion(n,result):
2     if n==0:
3         return result
4     return tail_recursion(n-1,n*result)
5
6 print(tail_recursion(0,1))
7 print(tail_recursion(1,1))
8 print(tail_recursion(2,1))
9 print(tail_recursion(5,1))
10 print(tail_recursion(8,1))
11 print(tail_recursion(10,1))
12 print(tail_recursion(1000,1))
```

Below the editor, a terminal window titled "student@pl2023: ~/Desktop" is open. It shows the command "python3 tail\_recursion.py" being executed. The output of the program is:

```
1
1
2
120
40320
3628800
```

Following the output, a traceback is displayed, indicating a "RecursionError: maximum recursion depth exceeded in comparison". The traceback points to line 12 in the module, which is the call to "print(tail\_recursion(1000,1))". The error message is repeated 995 more times, indicating a deep recursion stack.

## Non-tail recursion



The screenshot shows a Sublime Text editor window titled `~/Desktop/non_tail.py - Sublime Text (UNREGISTERED)` with two tabs: `non_tail.py` and `tail_recursion.py`. The `non_tail.py` tab is active, displaying the following Python code:

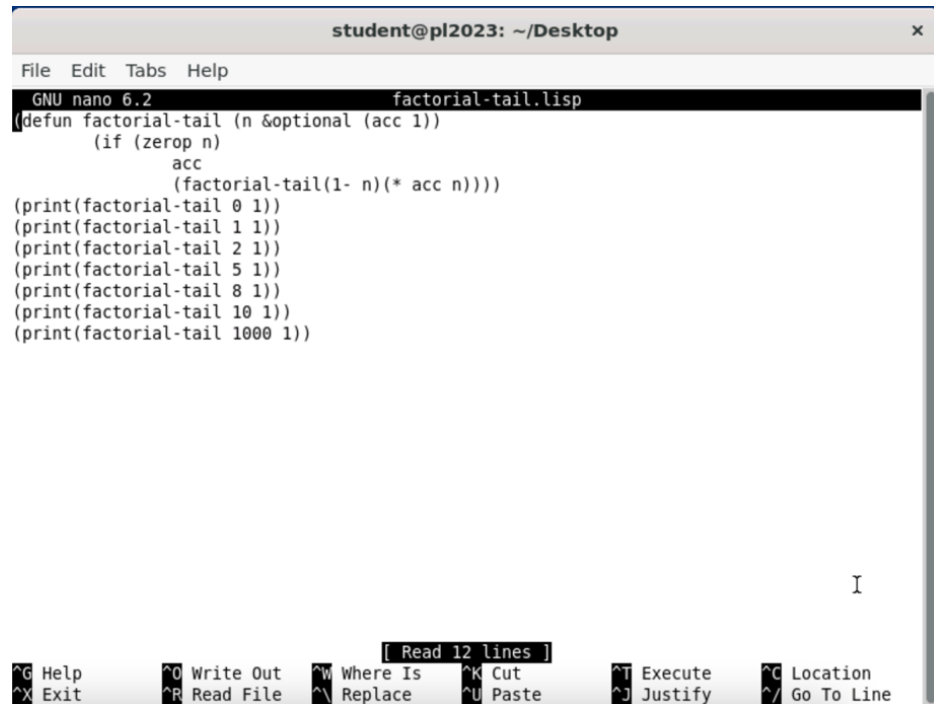
```
1 def factorial_non_tail(n):
2     if n==0:
3         return 1
4     return n*factorial_non_tail(n-1)
5
6 print(factorial_non_tail(0))
7 print(factorial_non_tail(1))
8 print(factorial_non_tail(2))
9 print(factorial_non_tail(5))
10 print(factorial_non_tail(8))
11 print(factorial_non_tail(10))
12 print(factorial_non_tail(1000))
```

Below the editor, a terminal window titled `student@pl2023: ~/Desktop` shows the execution of the script. The terminal output is as follows:

```
student@pl2023:~$ cd Desktop
student@pl2023:~/Desktop$ ls
non_tail.py  tail_recursion.py
student@pl2023:~/Desktop$ python3 non_tail.py
1
1
2
120
40320
3628800
Traceback (most recent call last):
  File "/home/student/Desktop/non_tail.py", line 12, in <module>
    print(factorial_non_tail(1000))
  File "/home/student/Desktop/non_tail.py", line 4, in factorial_non_tail
    return n*factorial_non_tail(n-1)
  File "/home/student/Desktop/non_tail.py", line 4, in factorial_non_tail
    return n*factorial_non_tail(n-1)
  File "/home/student/Desktop/non_tail.py", line 4, in factorial_non_tail
    return n*factorial_non_tail(n-1)
  File "/home/student/Desktop/non_tail.py", line 4, in factorial_non_tail
    return n*factorial_non_tail(n-1)
  [Previous line repeated 995 more times]
  File "/home/student/Desktop/non_tail.py", line 2, in factorial_non_tail
    if n==0:
RecursionError: maximum recursion depth exceeded in comparison
student@pl2023:~/Desktop$
```

## Lisp

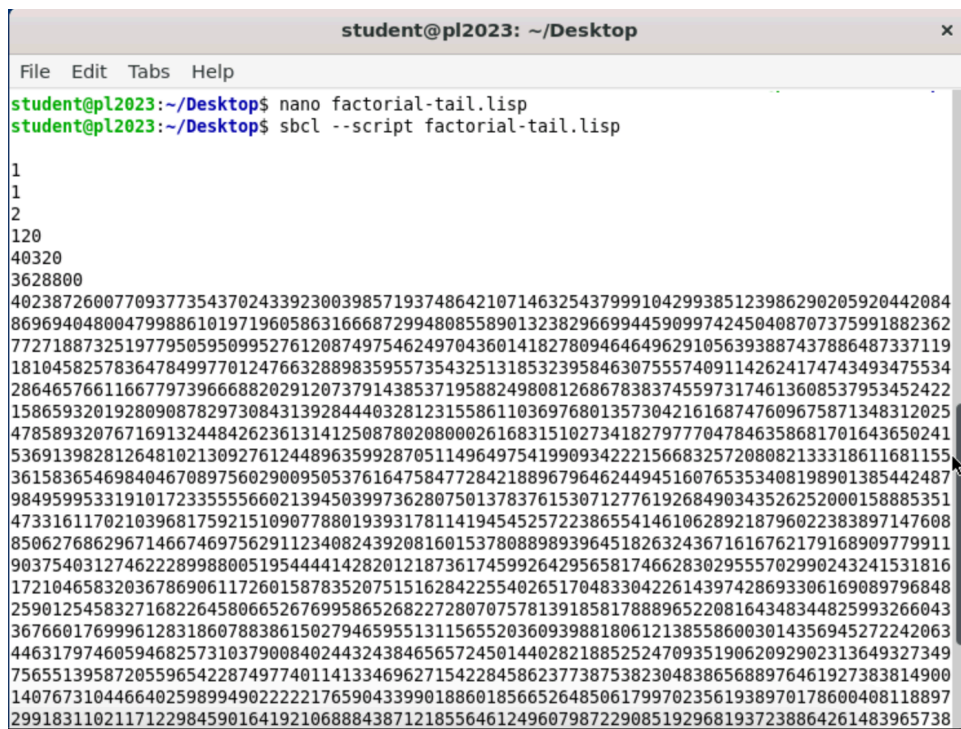
### Tail recursion



The screenshot shows a terminal window titled "student@pl2023: ~/Desktop" with a nano 6.2 editor open to a file named "factorial-tail.lisp". The code defines a tail-recursive function "factorial-tail" that takes two arguments: "n" and an optional "acc" (accumulator) defaulting to 1. The function uses "zerop" to check if "n" is zero. If so, it returns "acc". Otherwise, it recursively calls "factorial-tail" with "1- n" and "(\* acc n)". Below the function definition, several print statements are shown, including "(print(factorial-tail 0 1))", "(print(factorial-tail 1 1))", "(print(factorial-tail 2 1))", "(print(factorial-tail 5 1))", "(print(factorial-tail 8 1))", "(print(factorial-tail 10 1))", and "(print(factorial-tail 1000 1))". The nano editor's status bar at the bottom shows "Read 12 lines" and various keyboard shortcuts like ^G Help, ^O Write Out, ^W Where Is, ^K Cut, ^T Execute, ^C Location, ^X Exit, ^R Read File, ^\_ Replace, ^U Paste, ^J Justify, and ^\_ Go To Line.

```
student@pl2023: ~/Desktop
GNU nano 6.2 factorial-tail.lisp
(defun factorial-tail (n &optional (acc 1))
  (if (zerop n)
      acc
      (factorial-tail(1- n)(* acc n))))
(print(factorial-tail 0 1))
(print(factorial-tail 1 1))
(print(factorial-tail 2 1))
(print(factorial-tail 5 1))
(print(factorial-tail 8 1))
(print(factorial-tail 10 1))
(print(factorial-tail 1000 1))

[ Read 12 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^_ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```



The screenshot shows a terminal window titled "student@pl2023: ~/Desktop". It displays the command "nano factorial-tail.lisp" and then "sbcl --script factorial-tail.lisp". The output shows the results of the factorial function for various inputs: 1, 1, 2, 120, 40320, 3628800, and a long sequence of numbers for the factorial of 1000, starting with 4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084 and ending with 2991831102117122984590164192106888438712185564612496079872290851929681937238864261483965738.

```
student@pl2023:~/Desktop$ nano factorial-tail.lisp
student@pl2023:~/Desktop$ sbcl --script factorial-tail.lisp

1
1
2
120
40320
3628800
4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084
8696940480047998861019719605863166687299480855890132382966994459099742450408707375991882362
7727188732519779505950995276120874975462497043601418278094646496291056393887437886487337119
1810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534
2864657661166779739666882029120737914385371958824980812686783837455973174613608537953452422
1586593201928090878297308431392844403281231558611036976801357304216168747609675871348312025
4785893207671691324484262361314125087802080002616831510273418279777047846358681701643650241
5369139828126481021309276124489635992870511496497541990934222156683257208082133318611681155
3615836546984046708975602900950537616475847728421889679646244945160765353408198901385442487
9849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351
4733161170210396817592151090778801939317811419454525722386554146106289218796022383897147608
8506276862967146674697562911234082439208160153780889893964518263243671616762179168909779911
9037540312746222899880051954444142820121873617459926429565817466283029555702990243241531816
1721046583203678690611726015878352075151628422554026517048330422614397428693306169089796848
2590125458327168226458066526769958652682272807075781391858178889652208164348344825993266043
3676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063
4463179746059468257310379008402443243846565724501440282188525247093519062092902313649327349
7565513958720559654228749774011413346962715422845862377387538230483865688976461927383814900
1407673104466402598994902222217659043399018860185665264850617997023561938970178600408118897
2991831102117122984590164192106888438712185564612496079872290851929681937238864261483965738
```

## Non-tail recursion

```
student@pl2023: ~/Desktop
File Edit Tabs Help
GNU nano 6.2 factorial-non-tail.lisp
(defun factorial (n)
  (if (zerop n)
      1
      (* n (factorial (1- n)))))
(print (factorial 0))
(print (factorial 1))
(print (factorial 2))
(print (factorial 5))
(print (factorial 8))
(print (factorial 10))
(print (factorial 1000))

[ Read 12 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

```
student@pl2023: ~/Desktop
File Edit Tabs Help
student@pl2023:~/Desktop$ ls
factorial-non-tail.lisp factorial-tail.lisp non_tail.py tail_recursion.py
student@pl2023:~/Desktop$ nano factorial-non-tail.lisp
student@pl2023:~/Desktop$ sbcl --script factorial-non-tail.lisp

1
1
2
120
40320
3628800
4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084
8696940480047998861019719605863166687299480855890132382966994459099742450408707375991882362
7727188732519779505950995276120874975462497043601418278094646496291056393887437886487337119
1810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534
2864657661166779739666882029120737914385371958824980812686783837455973174613608537953452422
1586593201928090878297308431392844403281231558611036976801357304216168747609675871348312025
4785893207671691324484262361314125087802080002616831510273418279777047846358681701643650241
5369139828126481021309276124489635992870511496497541990934222156683257208082133318611681155
3615836546984046708975602900950537616475847728421889679646244945160765353408198901385442487
9849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351
4733161170210396817592151090778801939317811419454525722386554146106289218796022383897147608
8506276862967146674697562911234082439208160153780889893964518263243671616762179168909779911
9037540312746222899880051954444142820121873617459926429565817466283029555702990243241531816
1721046583203678690611726015878352075151628422554026517048330422614397428693306169089796848
2590125458327168226458066526769958652682272807075781391858178889652208164348344825993266043
3676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063
4463179746059468257310379008402443243846565724501440282188525247093519062092902313649327349
7565513958720559654228749774011413346962715422845862377387538230483865688976461927383814900
```