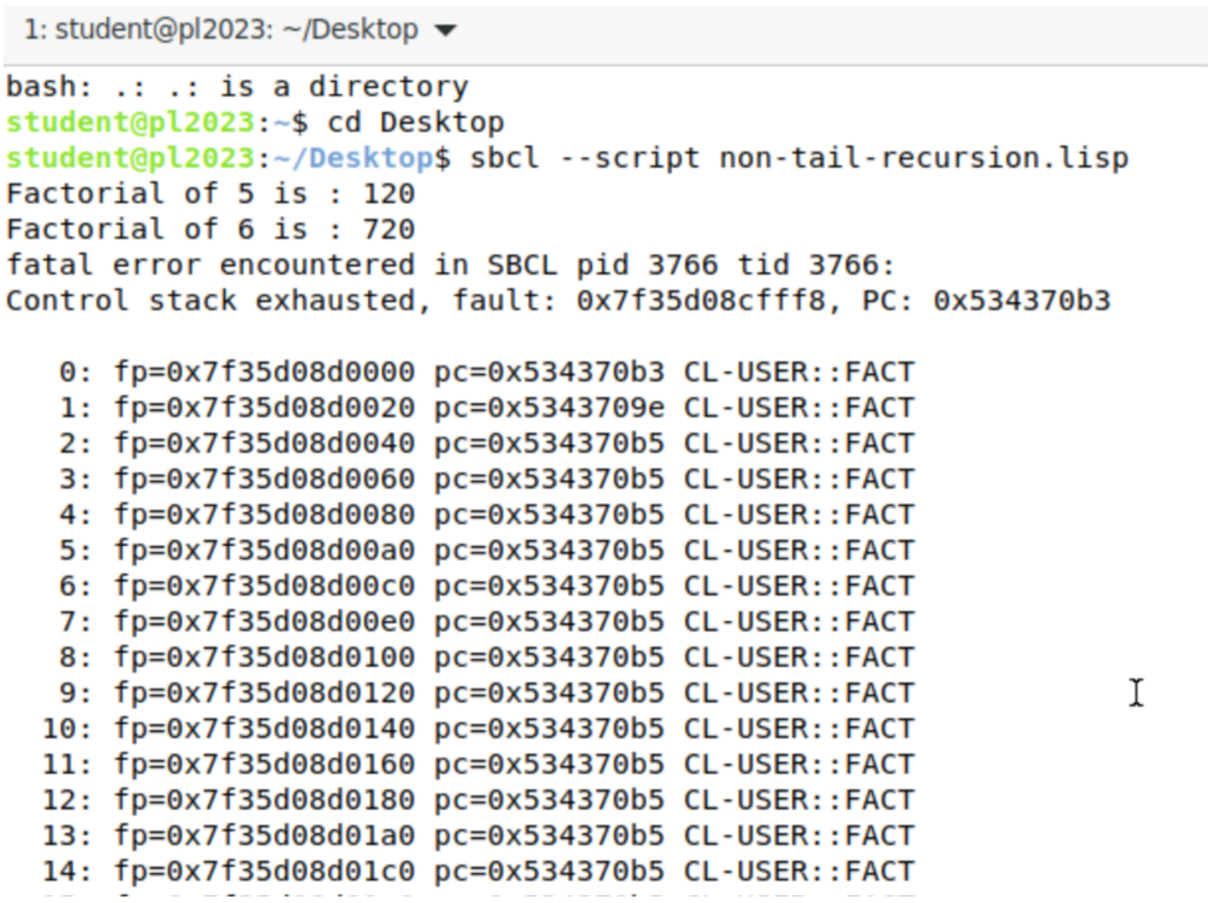
Assignment - 3

Sai Nishanth Mettu - sm11326

SBCL Compiler : I have used both VITAL & https://rextester.com/l/common_lisp_online_compiler to cross verify.

1) sbcl - non-tail recursive factorial

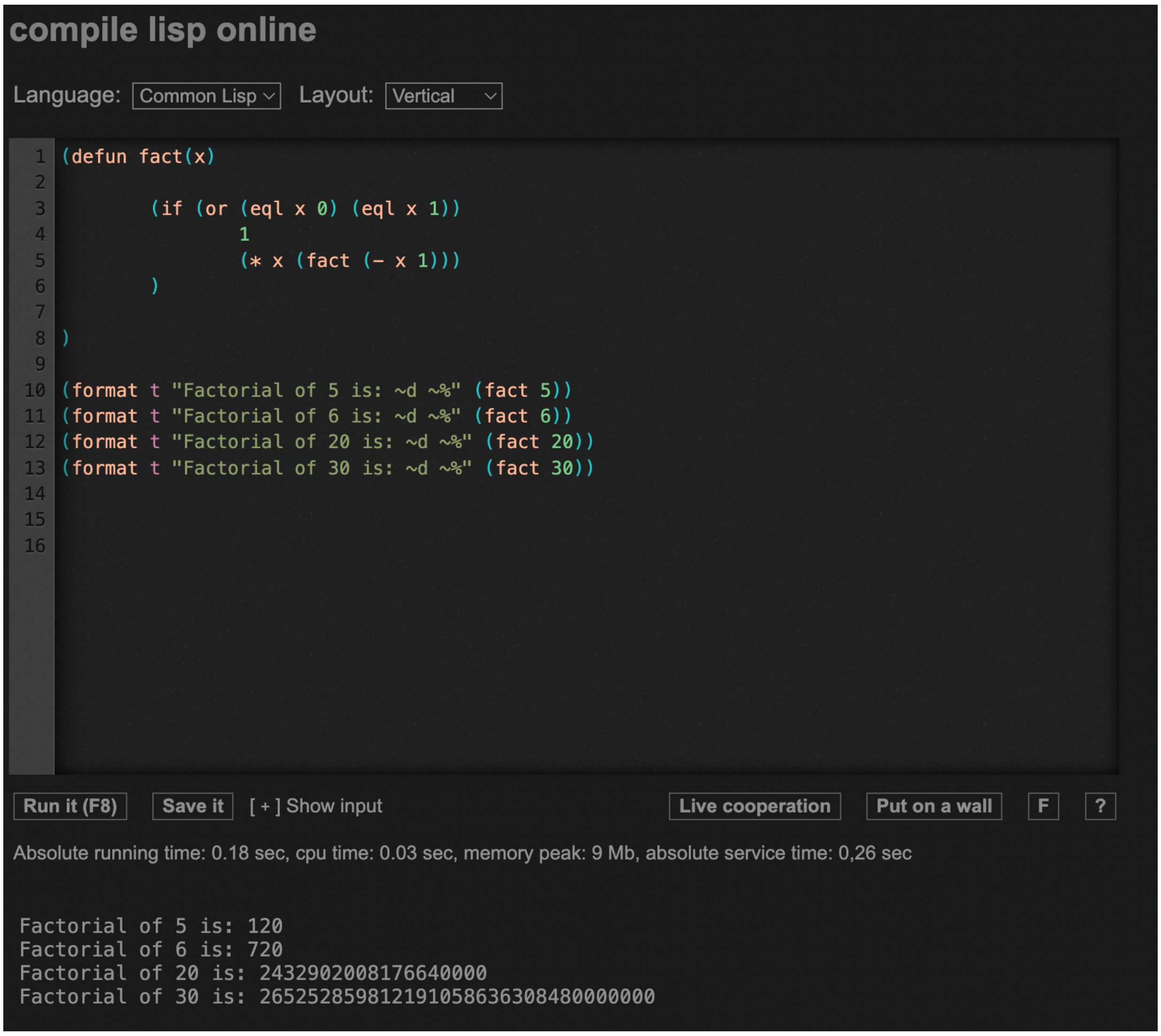


1: student@pl2023: ~/Desktop

```
bash: .: .: is a directory
student@pl2023:~$ cd Desktop
student@pl2023:~/Desktop$ sbcl --script non-tail-recursion.lisp
Factorial of 5 is : 120
Factorial of 6 is : 720
fatal error encountered in SBCL pid 3766 tid 3766:
Control stack exhausted, fault: 0x7f35d08cfff8, PC: 0x534370b3

   0: fp=0x7f35d08d0000 pc=0x534370b3 CL-USER::FACT
   1: fp=0x7f35d08d0020 pc=0x5343709e CL-USER::FACT
   2: fp=0x7f35d08d0040 pc=0x534370b5 CL-USER::FACT
   3: fp=0x7f35d08d0060 pc=0x534370b5 CL-USER::FACT
   4: fp=0x7f35d08d0080 pc=0x534370b5 CL-USER::FACT
   5: fp=0x7f35d08d00a0 pc=0x534370b5 CL-USER::FACT
   6: fp=0x7f35d08d00c0 pc=0x534370b5 CL-USER::FACT
   7: fp=0x7f35d08d00e0 pc=0x534370b5 CL-USER::FACT
   8: fp=0x7f35d08d0100 pc=0x534370b5 CL-USER::FACT
   9: fp=0x7f35d08d0120 pc=0x534370b5 CL-USER::FACT
  10: fp=0x7f35d08d0140 pc=0x534370b5 CL-USER::FACT
  11: fp=0x7f35d08d0160 pc=0x534370b5 CL-USER::FACT
  12: fp=0x7f35d08d0180 pc=0x534370b5 CL-USER::FACT
  13: fp=0x7f35d08d01a0 pc=0x534370b5 CL-USER::FACT
  14: fp=0x7f35d08d01c0 pc=0x534370b5 CL-USER::FACT
```

**Here we can see that non-tail recursion works in lisp for 5 and 6...
it successfully prints 120 for 5 and 720 for 6 but in general, it fails for very large number like 999999 or 123456!**

<u>Trying the same thing in the "online" LISP Compiler :</u>



Language: Common Lisp Layout: Vertical

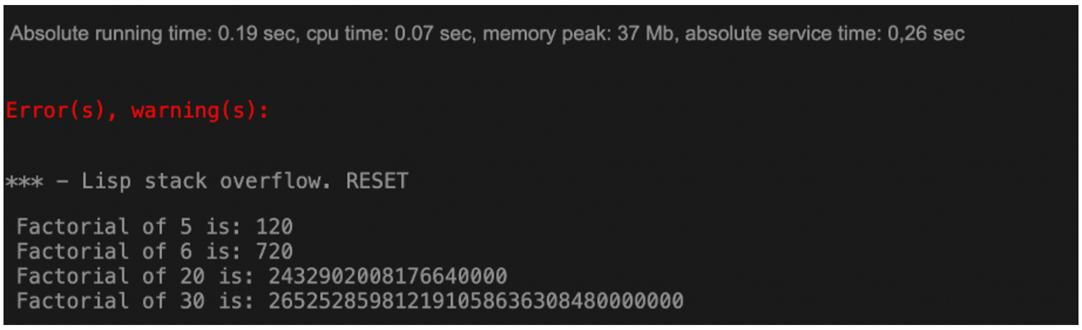
```
(defun fact(x)

        (if (or (eql x 0) (eql x 1))
                1
                (* x (fact (- x 1)))
        )

)

(format t "Factorial of 5 is: ~d ~%" (fact 5))
(format t "Factorial of 6 is: ~d ~%" (fact 6))
(format t "Factorial of 20 is: ~d ~%" (fact 20))
(format t "Factorial of 30 is: ~d ~%" (fact 30))
```

Absolute running time: 0.18 sec, cpu time: 0.03 sec, memory peak: 9 Mb, absolute service time: 0,26 sec

```
Factorial of 5 is: 120
Factorial of 6 is: 720
Factorial of 20 is: 2432902008176640000
Factorial of 30 is: 265252859812191058636308480000000
```

It works for 5,6,20,30...

But for a factorial of a very large number, i.e 999999

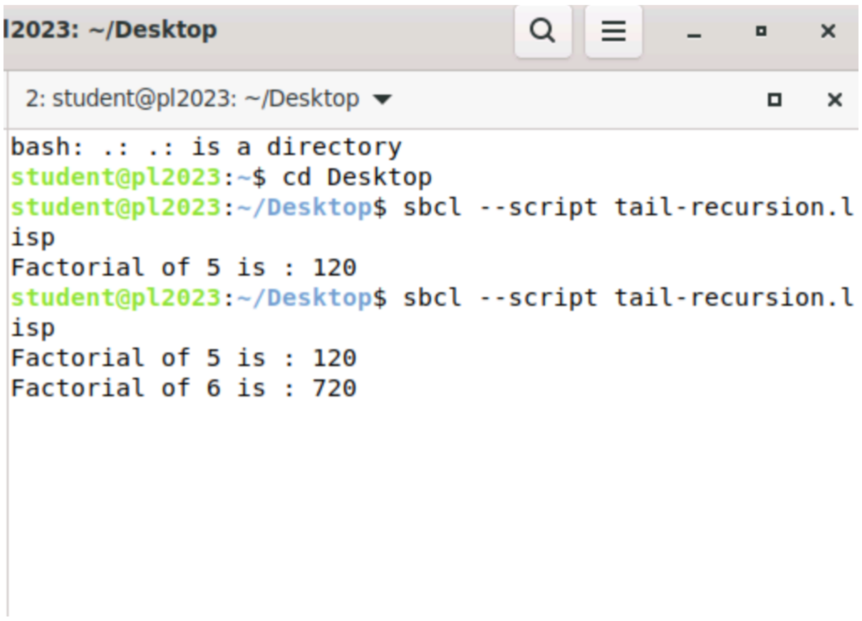


Absolute running time: 0.19 sec, cpu time: 0.07 sec, memory peak: 37 Mb, absolute service time: 0,26 sec

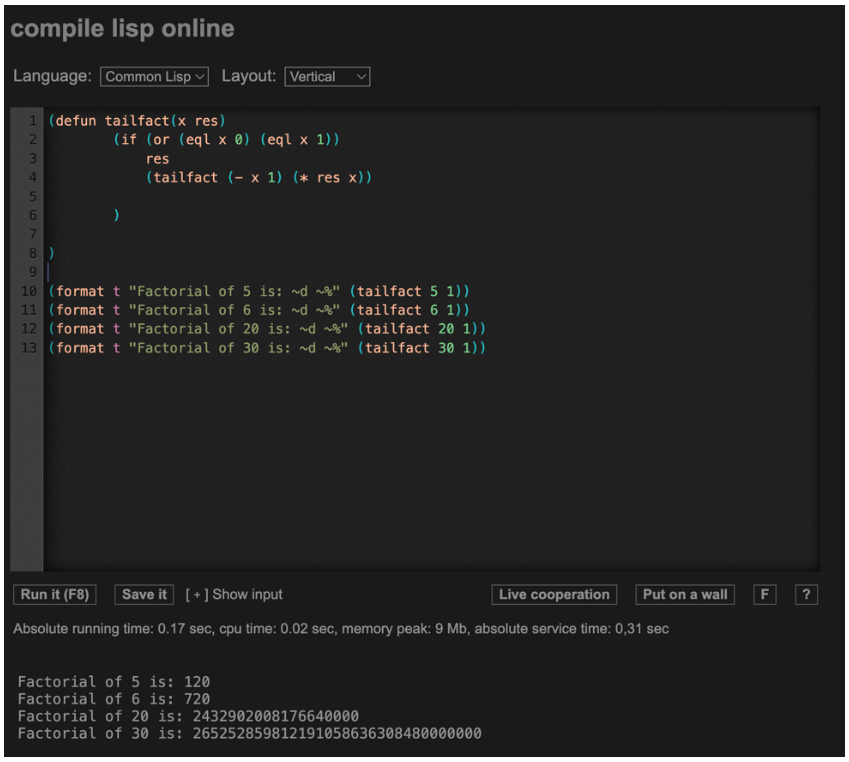
```
Error(s), warning(s):

*** - Lisp stack overflow. RESET

 Factorial of 5 is: 120
 Factorial of 6 is: 720
 Factorial of 20 is: 2432902008176640000
 Factorial of 30 is: 265252859812191058636308480000000
```

Inference : LISP supports Non-Tail Recursion but if we keep increasing the number, we can see that there is a stack overflow! For the factorial of very big numbers!

2) sbcl - tail recursive factorial



```
bash: .: .: is a directory
student@pl2023:~$ cd Desktop
student@pl2023:~/Desktop$ sbcl --script tail-recursion.l
isp
Factorial of 5 is : 120
student@pl2023:~/Desktop$ sbcl --script tail-recursion.l
isp
Factorial of 5 is : 120
Factorial of 6 is : 720
```



Language: Common Lisp Layout: Vertical

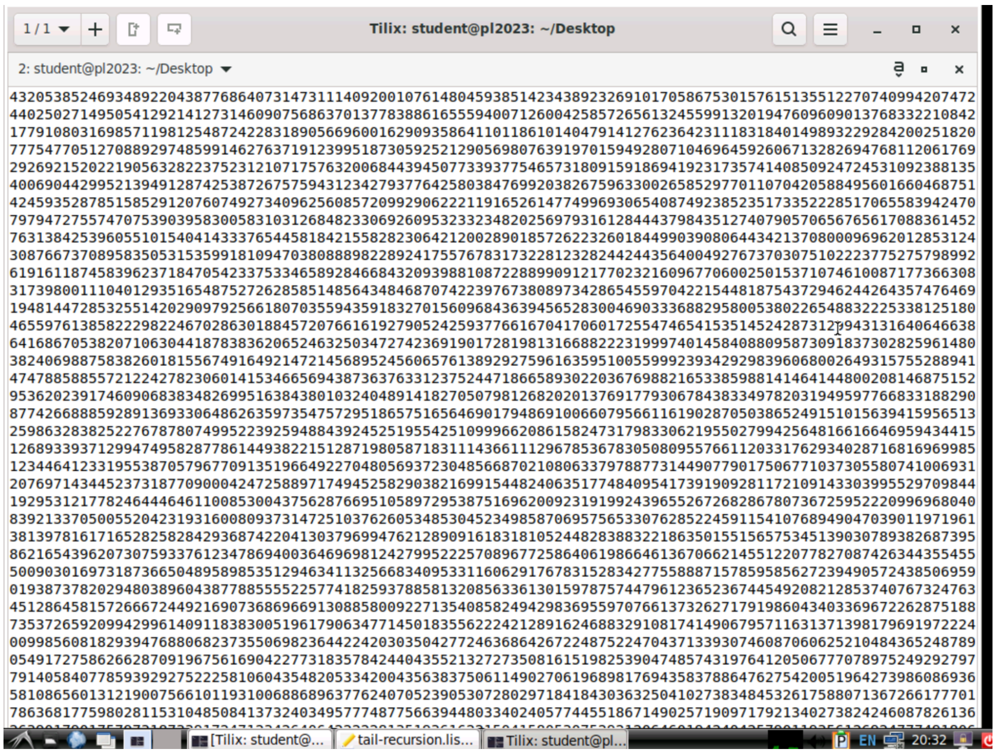
```
(defun tailfact(x res)
        (if (or (eql x 0) (eql x 1))
            res
            (tailfact (- x 1) (* res x))

        )

)

(format t "Factorial of 5 is: ~d ~%" (tailfact 5 1))
(format t "Factorial of 6 is: ~d ~%" (tailfact 6 1))
(format t "Factorial of 20 is: ~d ~%" (tailfact 20 1))
(format t "Factorial of 30 is: ~d ~%" (tailfact 30 1))
```

Absolute running time: 0.17 sec, cpu time: 0.02 sec, memory peak: 9 Mb, absolute service time: 0,31 sec

```
Factorial of 5 is: 120
Factorial of 6 is: 720
Factorial of 20 is: 2432902008176640000
Factorial of 30 is: 265252859812191058636308480000000
```

Inference : **LISP supports Tail Recursion!**

Lets print factorial for a very large number i.e 123456



```
43205385246934892204387768640731473111409200107614804593851423438923269101705867530157615135512270740994207472
44025027149505412921412731460907568637013778388616555940071260042585726561324559913201947609609013768332210842
17791080316985711981254872422831890566960016290935864110118610140479141276236423111831840149893229284200251820
77754770512708892974859914627637191239951873059252129056980763919701594928071046964592606713282694768112061769
29269215202219056328223752312107175763200684439450773393775465731809159186941923173574140850924724531092388135
40069044299521394912874253872675759431234279377642580384769920382675963300265852977011070420588495601660468751
42459352878515852912076074927340962560857209929062221191652614774996930654087492385235173352228517065583942470
79794727557470753903958300583103126848233069260953233234820256979316128444379843512740790570656765617088361452
76313842539605510154041433376544581842155828230642120028901857262232601844990390806443421370800096962012853124
30876673708958350531535991810947038088898228924175576783173228123282442443564004927673703075102223775275798992
61916118745839623718470542337533465892846684320939881087228899091217702321609677060025015371074610087177366308
31739800111040129351654875272628585148564348468707422397673808973428654559704221544818754372946244264357476469
19481447285325514202909792566180703559435918327015609684363945652830046903336882958005380226548832225338125180
46559761385822298224670286301884572076616192790524259377661670417060172554746541535145242873129943131640646638
64168670538207106304418783836206524632503472742369190172819813166882223199974014584088095873091837302825961480
38240698875838260181556749164921472145689524560657613892927596163595100559992393429298396068002649315755288941
47478858855721224278230601415346656943873637633123752447186658930220367698821653385988141464144800208146875152
95362023917460906838348269951638438010324048914182705079812682020137691779306784383349782031949597766833188290
87742668885928913693306486263597354757295186575165646901794869100660795661161902870503865249151015639415956513
25986328382522767878074995223925948843924525195542510999662086158247317983306219550279942564816616646959434415
12689339371299474958287786144938221512871980587183111436611129678536783050809557661120331762934028716816969985
12344641233195538705796770913519664922704805693723048566870210806337978877314490779017506771037305580741006931
20769714344523731877090004247258897174945258290382169915448240635177484095417391909281172109143303995529709844
19295312177824644464611008530043756287669510589729538751696200923191992439655267268286780736725952220996968040
83921337050055204231931600809373147251037626053485304523498587069575653307628522459115410768949047039011971961
38139781617165282582842936874220413037969947621289091618318105244828388322186350155156575345139030789382687395
86216543962073075933761234786940036469698124279952225708967725864061986646136706621455122077827087426344355455
50090301697318736650489589853512946341132566834095331160629176783152834277558887157859585627239490572438506959
01938737820294803896043877885555225774182593788581320856336130159787574479612365236744549208212853740767324763
45128645815726667244921690736869669130885800922713540858249429836955970766137326271791986043403369672262875188
73537265920994299614091183830051961790634771450183556222421289162468832910817414906795711631371398179691972224
00998560818293947688068237355069823644224203035042772463686426722487522470437133930746087060625210484365248789
05491727586266287091967561690422773183578424404355213272735081615198253904748574319764120506777078975249292797
79140584077859392927522258106043548205334200435638375061149027061968981769435837886476275420051964273986086936
58108656013121900756610119310068868963776240705239053072802971841843036325041027383484532617588071367266177701
78636817759802811531048508413732403495777487756639448033402405774455186714902571909717921340273824246087826136
```

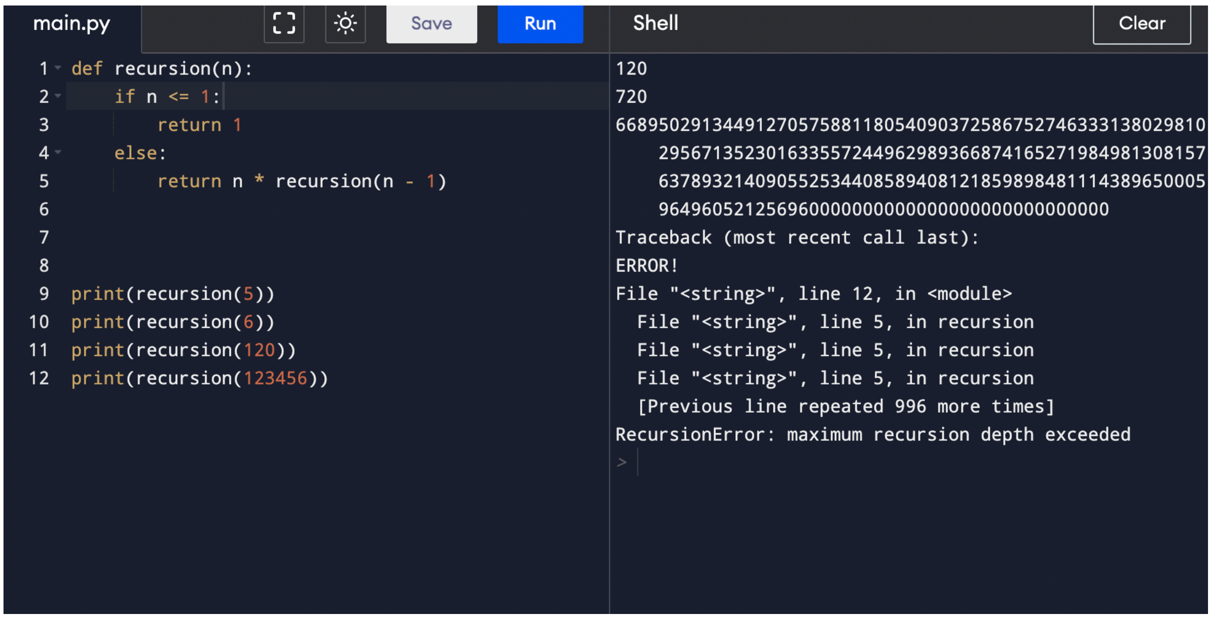
Here with Tail Recursion, we can calculate the factorial of a very huge number too like 999999 or 123456! (OPTIMIZATION!)
Thus it shows that LISP supports “Tail Recursion”.

FINAL INFERENCE **: LISP supports both tail recursion and non-tail recursion and it “optimizes” tail-recursion!**

PYTHON CHECK

Compiler used : https://www.programiz.com/python-programming/online-compiler/

3) python non-tail recursion (normal)

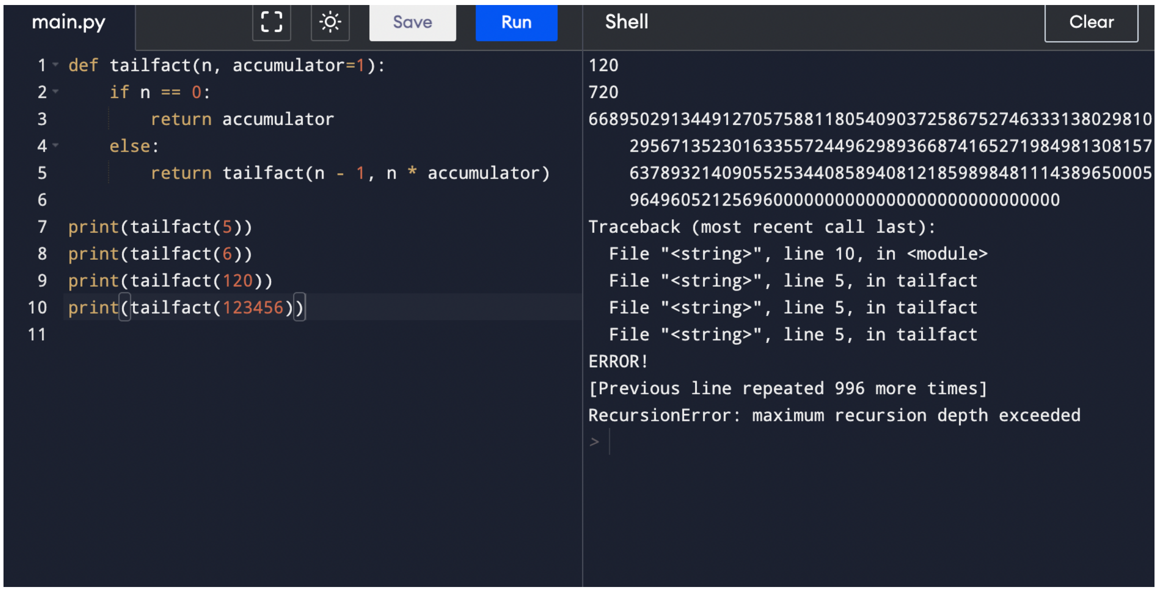


```python
def recursion(n):
    if n <= 1:
        return 1
    else:
        return n * recursion(n - 1)



print(recursion(5))
print(recursion(6))
print(recursion(120))
print(recursion(123456))
```

```
120
720
6689502913449127057588118054090372586752746333138029810
    2956713523016335572449629893668741652719849813081576
    3789321409055253440858940812185989848111438965000596
    4960521256960000000000000000000000000000
Traceback (most recent call last):
ERROR!
File "<string>", line 12, in <module>
  File "<string>", line 5, in recursion
  File "<string>", line 5, in recursion
  File "<string>", line 5, in recursion
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
>
```

Inference : Here you can note that for smaller numbers recursion works fine but for larger numbers it results in a "Recursion Error : maximum recursion depth exceeded".

4) python tail-recursion



```python
def tailfact(n, accumulator=1):
    if n == 0:
        return accumulator
    else:
        return tailfact(n - 1, n * accumulator)

print(tailfact(5))
print(tailfact(6))
print(tailfact(120))
print(tailfact(123456))
```

```
120
720
6689502913449127057588118054090372586752746333138029810
    2956713523016335572449629893668741652719849813081576
    3789321409055253440858940812185989848111438965000596
    4960521256960000000000000000000000000000
Traceback (most recent call last):
  File "<string>", line 10, in <module>
  File "<string>", line 5, in tailfact
  File "<string>", line 5, in tailfact
  File "<string>", line 5, in tailfact
ERROR!
[Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
>
```

Inference : As you can see, even in tail-factorial using python, we get the same "RecursionError". Thus, it shows that Python doesn't support/optimize using tail-recursion.

OVERALL INFERENCE :

Lisp : Supports Tail Recursion and it optimizes the memory stack.
Python : Does not support Tail Recursion and it gives the same Recursion Error, which can be found in non-tail Recursion.

Sai Nishanth Mettu

sm11326