**Isaac Computer Science Student Activity Booklet** 

Recursive Programming Solutions

**Activity 1:** Visualise **Python Tutor:** <http://tiny.cc/37czyz>

* **Run** this code in the Python Tutor visualiser.

def sum\_to\_n(n):

if n == 1:

return 1

else:

return n + sum\_to\_n(n-1)

sum\_to\_n(3)

* Step through the code by pressing the *Next* button
* Click *Edit this code* to change the code
* Use the *Generate permanent link* button to create a link for your own code.
* What does this program do?
* How can visualising code execution help in debugging?

**Activity 2:** Stack Overflow **Trinket:** <https://trinket.io/python3/f5edfef789>

* Create a recursive function that takes an integer argument, prints it out, increments it by 1 and then calls itself, passing in the new value.
* How many calls to itself does it manage before we get a ‘stack overflow’?
* To overcome this issue, we need to introduce what is known as a ‘base case’. Use the recursive structure to write a base case for your function.
* **Hint:** The known condition should be one less than the recursive depth allowed by your machine.

**if** (condition for which answer is known):

statement # base case

**else:** recursive function call # general case

**Activity 3:** Factorial **Trinket:** <https://trinket.io/python3/229debde2f>

* Given the following recursive function to calculate the factorial of a number:

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

print(factorial(5))

* Trace the execution of the factorial function call factorial(5) step-by-step and fill in the table below to show the values of x, whether the base case condition is met (if n = 0) and the value returned at each level of recursion.

|  |  |  |  |
| --- | --- | --- | --- |
| Level | n | if n = 0: | Return |
| 0 | 5 | F |  |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |

* Start at Level 0 with n = 5 and trace the recursive calls until the base case is reached.
* For each level, note whether the condition if X = 0 is true (T) or false (F).
* Calculate the value returned at each level after the base case is reached and the recursion starts to unwind.
* Fill in the final returned value at each level in the “Return” column.

**Activity 4:** Tail Recursion **Trinket:** <https://trinket.io/python/dec0c23d73>

* Given the following recursive function to calculate the factorial of a number using tail recursion:

def factorial(n, a=1):

if n == 0:

return a

else:

return factorial(n - 1, n \* a)

print(factorial(5))

* Trace the execution of the tail recursive function call factorial(5) step-by-step and fill in the table below to show the values of n, whether the base case condition is met (if n == 0) and the value returned at each level of recursion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Level | n | a | if n = 0: | Return |
| 0 | 5 | 1 | F |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

* Start at Level 0 with n = 5 and a = 1 and trace the recursive calls until the base case is reached.
* For each level, note whether the condition if n == 0 is true (T) or false (F).
* Calculate the value returned at each level after the base case is reached and the recursion starts to unwind.
* Fill in the final returned value at each level in the “Return” column.

**Activity 5:** Recursion vs Iteration

**Trinket:** <https://trinket.io/python3/c4e26afe09>

* The algorithm represented using the recursive function below calculates the factorial of a number. Your task is to develop an equivalent iterative function (naive\_factorial) to achieve the same result.

def recursive\_factorial(n):

if n == 0: # base case

return 1

else: # general case

return n \* recursive\_factorial(n - 1)

What you need to do:

* **Task 1:** Write the Python program for the naive\_factorial function described above using the pseudocode below:

FUNCTION naive\_factorial(n):

result ← 1

WHILE n > 0 DO:

result ← result \* n

n ← n - 1

END WHILE

RETURN result

END FUNCTION

* **Task 2**: Test the program by showing the result of entering 5 and 7. Use the following code to compare both functions:

# Testing recursive factorial

print("Recursive Factorial of 5:", recursive\_factorial(5))

print("Recursive Factorial of 7:", recursive\_factorial(7))

# Testing naive factorial

print("Naive Factorial of 5:", naive\_factorial(5))

print("Naive Factorial of 7:", naive\_factorial(7))

* **Task 3:** Compare the result of the naive\_factorial function with the recursive\_factorial function for the inputs 5 and 7. Ensure both functions produce the same output.

|  |
| --- |
| Copy and paste your python solution or add a screenshot here. |

**Activity 6:** Which is best? **Trinket:** <https://trinket.io/python3/40d72ed135>

* Compare the time efficiency of the naive/iterative and the elegant/recursive factorial algorithm.
* **Hint:** Use the following test to measure the efficiency of each algorithm.

import time

start = time.time()

for count in range(10000):

x = 1 + 1 # The code you want to test

end = time.time()

print(end - start)

* Which algorithm was most time efficient?