**Recursion Issues**

*See the lesson notes at*: [dmaccarthy.github.io/sci/#cs\_new/rec/issue](https://dmaccarthy.github.io/sci/#cs_new/rec/issue)

. Explain in detail what a *call stack* is, how it works, and why it is important in Python and other programming languages.

Answer

2. What is a *stack overflow* and why is it a bad thing?

Answer

3. a) Explain how Python’s recursion depth limit prevents stack overflows and how it also limits your ability to use recursion to solve computing problems.

Answer

b) Copy your **sumWhole** function from **rec1\_math.py** into **rec4\_limit.py**. Run the program to see how large **n** can be before a “maximum recursion depth exceeded” error occurs.

Answer

4. What is *tail recursion*? Compare it to *head recursion*.  How can tail recursion eliminate problems with recursive algorithms? Does Python support tail recursion optimization? Why or why not?

Answer

5. One of the advantages of modern high-level languages such as Python is automatic management of your program’s memory requirements. Some programming languages, such as C, require programmers to explicitly request memory when needed and release the memory when it is no longer required.

a) Explain in detail how the stack and the heap are used to store program data.

Answer

b) Explain what a pointer (also called a reference) is and how it relates to memory management.

Answer

c) Why is it important for data structures like strings and arrays that the data is stored in a contiguous portion of the heap?

Answer

d) In programming environments without automatic memory management, why is it important to deallocate memory that is no longer needed?

Answer

e) Explain the difference between static and dynamic data structures.

Answer

f) Explain the difference between linear and nonlinear data structures.

Answer