



Recursive Programming

PDL Name _____

Date _____

Intended learning outcomes

By the end of this session, you will:

- Be able to explain the principles of recursion, including the concepts of base case and general case.
- Be able to implement recursive functions to solve various computational problems.
- Be able to analyse and compare recursive and iterative approaches to problem-solving.
- Be able to identify and address potential issues such as stack overflow in recursive functions.



To understand recursion,
one must first understand ...
recursion

Stephen Hawking



Functions that call themselves

- A recursive function contains a function call to itself within itself
- A recursive function has two parts
 - i) base case
 - ii) general case
- The base case returns an answer that is known
e.g. `return 1`
- The general case contains the recursive call
- Each successive call should take you closer to a situation that is known i.e. the base case



Recursive Structure

```
if (condition for which answer is known):
    statement # base case
else:
    recursive function call # general case
```



Which line expresses the base case?

```
def sum_to_n(n):
    if n == 1:
        return 1
    else:
        return n + sum_to_n(n-1)
```



Visualisation



- Python Tutor is a resource designed to visualise what happens when code is executed.



- It can be used to understand and debug complex code examples



Unwinding

- Code before the recursive call is executed in "forward order"
- Calculations after the recursive call are executed in "reverse order"

n	n + sum_to_n(n-1)	Return value
5	5 + 10	15
4	4 + 6	10
3	3 + 3	6
2	2 + 1	3
1		1



Stack Overflow



- What happens if we forget the base case?
- If we recursively call too many functions without returning, we cause a stack overflow error



Factorial



```
def recursive_factorial(n):
    if n == 0: # base case
        return 1
    else: # general case
        return n * recursive_factorial(n - 1)

print(recursive_factorial(4))
```

Can you trace this algorithm? Remember the unwinding we learned on slide 8



Base Case



- The function call Factorial(4) should return the value 24, because that is $4 * 3 * 2 * 1$.
- For a situation in which the answer is known, the value of $0!$ is 1.
- So, the base case is

```
if n == 0:
    return 1
```



General Case



- The value of Factorial(n) can be written as n * the product of the numbers from $(n - 1)$ to 1

$$\begin{aligned} & n * (n - 1) * \dots * 1 \\ & \text{or} \\ & n * \text{Factorial}(n - 1) \end{aligned}$$

- Notice, that the recursive call Factorial($n - 1$) gets us "closer" to the base case of Factorial(0)



Unwinding



Activity 3

- Factorial (0) doesn't make a recursive call, but returns a 1, which starts the 'unwind' process
- Until Factorial (0) is called, a stack of uncompleted calculations are stored
- Pay particular attention to the returned value and how that is used in the previous call

```

n * Factorial(n - 1)
= 4 * factorial 3
= 4 * (3 * factorial 2)
= 4 * (3 * (2 * factorial 1))
= 4 * (3 * (2 * (1 * factorial 0)))
= 4 * (3 * (2 * (1 * 1)))
= 4 * (3 * (2 * 1))
= 4 * (3 * 2)
= 4 * 6
= 24

```



Tail Recursion



Activity 4

- Tail recursion is when the call to itself is the last action of the function.

```

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

print(factorial(4))        print(factorial(4))

```



Recursive vs Iteration



Activity 5

- Any task accomplished with iteration can also be achieved with recursion.
- The resulting code will probably be more elegant, compact and simpler to implement.
- The iterative approach could be described as more 'naive'.
- Recursive algorithms are often slower, use more memory and less efficient.



Which is best?



```
def naive_factorial(n):
    result = 1
    while n > 0:
        result *= n
        n -= 1
    return result

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



Recursive vs Iteration

Benefits	Drawbacks
More natural to read	Can run out of memory (i.e. stack overflow)
Quicker to write	More difficult to trace and follow
Suited to certain problems, e.g. Divide and conquer merge sort & quick sort	Requires more memory than an equivalent iterative program
Can reduce the size of a problem with each call e.g. binary search	Usually slower than iterative methods



Conclusions

- Recursion provides an elegant method for solving problems that can be divided into smaller subproblems.
- While often more readable, recursive solutions are not always the most efficient in terms of memory and speed.
- Excessive recursion depth can exceed the call stack limit and crash the program.



Intended learning outcomes

By the end of this session, you will:

- Be able to explain the principles of recursion, including the concepts of base case and general case.
- Be able to implement recursive functions to solve various computational problems.
- Be able to analyse and compare recursive and iterative approaches to problem-solving.
- Be able to identify and address potential issues such as stack overflow in recursive functions.



Game Board



Isaac link



Gameboard

Recursive Programming Booster Session

- | | | |
|---|--|-----------------|
| 1 | Needle in a haystack
Programming Fundamentals / Recursion | A Level
100% |
| 2 | One after another
Programming Fundamentals / Recursion | A Level
100% |
| 3 | Up and down
Programming Fundamentals / Recursion | A Level
100% |
| 4 | Keep counting
Programming Fundamentals / Recursion | A Level
100% |
| 5 | Back and forth
Programming Fundamentals / Recursion | A Level
100% |
| 6 | Round and round
Programming Fundamentals / Recursion | A Level
100% |
| 7 | What's the point?
Programming Fundamentals / Recursion | A Level
100% |



Extra Credit



Trinket link



Extra

1. Write a recursive function that returns the sum of the digits of an integer.
2. Write a recursive function (use no while loops or for loops) that prints all the elements of a list of integers, one per line. The parameters to the function should be a list of integers and the size of the list.
3. Modify the previous function to print out the elements in reverse order.



ISAAC boosters

Keep an eye out for more student booster events

Thank you


