Week 12 Recursion

This folder includes some interactive content we'll work through together during Week 12!

We'll spend some time trying to code key recursive functions and live code them together.

This deck has the SAME content as the readme file in `other_content/week12_recursion`. I will recommend you open that file during lecture, and I'll project this just for slightly better lecture viewing experience.

Exercise 1: Review tree drawing

First, take a look at `draw_tree.py` in this directory. We're going to talk through how and way this code works. You should copy and paste this into a file on your local machine. Trying running it, then try modifying the *initial function call* so that the tree:

1. Has even more branches
2. Has fewer branches
3. Hits the base case immediately

Exercise 2: Recursive Factorial

Probem: Write a recursive function that returns the factorial of a number. The factorial of n is equal to n times the factorial of n-1.

Start with a single parameter function definition: `def factorial(n):`

Examples:

- 0! = 0 (important: the factorial of 0 is equal to 1.)
- 1! = 1
- 2! = 2 * 1= 2 * 1!
- 3! = 3 * 2 * 1 = 3 * 2!
- so, `factorial(4) # → 24`

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

- Solution

Exercise 3: Sums with recursion

Exercise 3a: Summing all integers from 1 to n

Write a recursive function in Python, recursive_sum, that calculates the sum of all integers from 1 up to a given positive integer n.

Start with a single parameter function definition: `def recursive_sum(n):`

Examples:

- `recursive_sum(4) # Output: 9 (1 + 2 + 3 + 4)`
- `recursive_sum(5) # Output: 15 (1 + 2 + 3 + 4 + 5)`

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

- Solution

Exercise 3b: Summing all integers in a list

Problem: Write a recursive function in Python, recursive_sum_list, that calculates the sum of all integers in a given list.

Start with a single parameter function definition: `def recursive_sum_list(numbers):`

Examples:

- `recursive_sum_list([1, 2, 3, 4]) # Output: 10`
- `recursive_sum_list([5, 7, 3]) # Output: 15`

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

▶ Hint 3: List Indexing

▶ Solution

Exercise 4: Reverse a string

Write a recursive function in Python, recursive_reverse, that reverses a given string. Start with a single parameter function definition: `def recursive_reverse(s):`

Example:

- `recursive_reverse("hello") # Output: "olleh"`
- `recursive_reverse("abc") # Output: "cba"`

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

▶ Hint 3: String Indexing

- Solution

Exercise 5: Palindrome checker

Write a recursive function in Python, is_palindrome, that checks if a given string is a palindrome (reads the same forward and backward). Start with a single parameter function definition: `def is_palindrome(s):`

Examples:

- `is_palindrome("racecar") # Output: True`
- `is_palindrome("hello") # Output: False`

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

▶ Hint 3: String Indexing

- Solution

Exercise 6: Challenge!

"Nested List Sum"

Problem: Write a recursive function in Python, nested_sum, that calculates the sum of all integers in a list. However, this time, some elements of the list might be nested lists themselves. The function should recursively process these nested lists to calculate the total sum.

You can use `isinstance(item, int)` to check an item is an integer and `isinstance(item, list)` to check if an item is a list.

Start with a single parameter function definition: def nested_sum(data):

Examples:

- nested_sum([1, 2, [3, 4], 5]) # Output: 15
- nested_sum([[1, 2, [3]], 4, [5, [6]]]) # Output: 21
- nested_sum([1, [2, [3, [4, [5]]]]]) # Output: 15

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

▶ Solution

Exercise 7: Count Vowels in a String

Problem: Write a recursive function, count_vowels, that counts the number of vowels (a, e, i, o, u) in a given string. The function should be case-insensitive, so both uppercase and lowercase vowels are counted.

Start with a single parameter function definition: `def count_vowels(s):`

Examples:

- `count_vowels("hello") # Output: 2`
- `count_vowels("Recursion is fun!") # Output: 6`
- `count_vowels("xyz") # Output: 0`

▶ Hint 1: Base Case

▶ Hint 2: Recursive Case

- Solution