Welcome to this CoGrammar session:

Recursion

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators
answering questions.

CoGrammar

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions]

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: Feedback on Lectures

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✓ Criterion 1: Initial Requirements

- ***Guided Learning Hours (GLH):*** Minimum of 15 hours
- ***Task Completion:*** First 4 tasks

Due Date: 24 March 2024

✓ Criterion 2: Mid-Course Progress

- ***Guided Learning Hours (GLH):*** Minimum of 60 hours
- ***Task Completion:*** *First* 13 tasks

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

- ***Completion:*** All mandatory tasks, including Build Your Brand and resubmissions by study period end
- ***Interview Invitation:*** Within 4 weeks post-course
- ***Guided Learning Hours:*** Minimum of 112 hours by support end date (10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

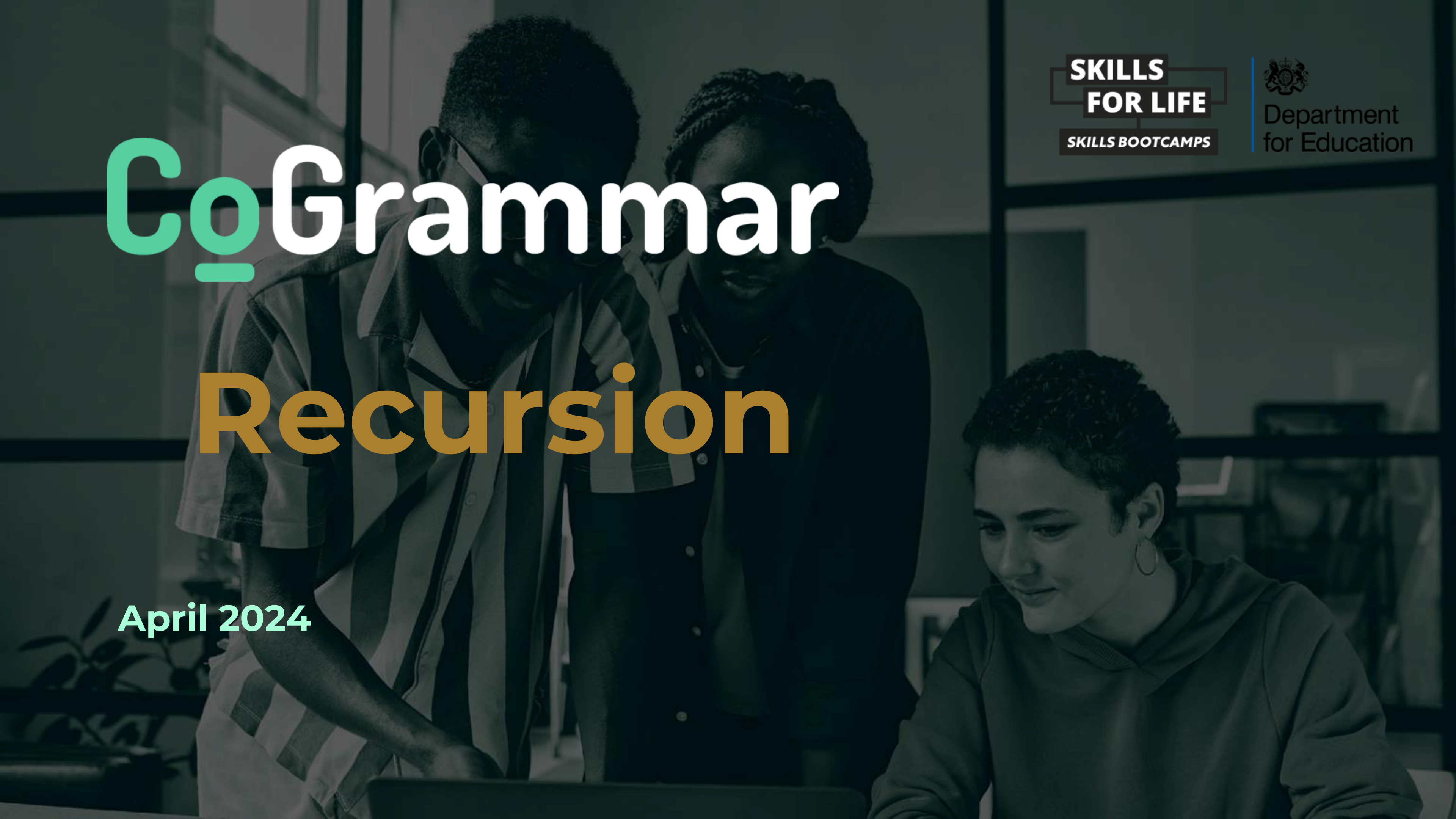
- ***Final Job or Apprenticeship Outcome:*** Document within 12 weeks post-graduation
- ***Relevance:*** Progression to employment or related opportunity

CoGrammar

Learning Objectives & Outcomes

- **Understand** the concept of **recursion** and its role in programming.
- **Understand** the concept of **iteration** and its role in programming.
- **Identify when recursion is** an **appropriate** solution and when it may not be.
- **Implement recursive functions** to solve problems.

CoGrammar



CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

Recursion

April 2024

What is Recursion?

- Recursion is a **programming technique** where a function calls itself to solve a problem by breaking it down into smaller, similar subproblems.
- This **self-referential approach** allows for elegant and concise solutions to certain types of problems.
- In recursion, **a base case** is typically defined to provide a **stopping condition** for the recursive calls. When the base case is reached, the recursion unwinds, and the function returns results back up the call stack.

Why Recursion?

- Recursion offers **simplicity**, **modularity**, and **flexibility** in solving certain types of problems.
- It allows for concise and elegant code, promotes code reuse, and is particularly **effective for** tackling **problems with repetitive, self-similar structures**.
- While it may not be suitable for every problem, recursion is a valuable tool in a programmer's toolkit, enabling the **solution of complex problems with clarity and efficiency**.

What is Iteration?

- Iteration is a **fundamental programming concept** that involves repeating a set of instructions or a process multiple times until a specific condition is met.
- Iteration provides a way to **execute code repeatedly**, **often with slight variations** or modifications each time.
- In iteration, **a loop structure** is commonly used to achieve repetition.
- Iteration involves **executing a block of code** repeatedly **until a certain condition is satisfied**. This allows for the efficient handling of repetitive tasks and is essential for automating processes in programming.

CoGrammar

Types of Iteration

- **Count-controlled Iterations**

  Where the number of repetitions is predetermined **based on a fixed count or iteration variable**. For example, a loop may be set to execute a certain number of times specified by a loop counter or a predefined limit.

- **Condition-controlled Iterations**

  Where the repetition continues until a specific condition evaluates to false. The condition is typically **based on the evaluation of a boolean expression**, such as checking for the end of a data stream or the satisfaction of a particular condition.

Why Iteration?

- Iterations excel in providing **efficiency**, **readability**, and **direct control** over execution in a broader range of situations.
- Iterations provide **a versatile alternative to recursion**, especially in scenarios where simplicity, modularity, and flexibility are not the primary concerns.
- Iterations typically offer **better performance and predictable resource usage** compared to recursion, making them suitable for handling large datasets or deep levels of nesting.

CoGrammar

Recursion vs Iteration

- Recursion and iteration (loops) can be used to achieve the same results. However, unlike loops, which work by **explicitly specifying a repetition structure**, recursion uses **continuous function calls to achieve repetition**.
- Recursion is a somewhat advanced topic and problems that can be solved with recursion can also most likely be solved by using simpler looping structures.
- **Recursion** is a useful programming technique that, in some cases, **can** enable you to **develop natural, straightforward, simple solutions to otherwise difficult problems**.

CoGrammar

Recursion vs Iteration ...

- The following **guidelines** will help you **to decide** which method to use depending on a given situation:
  - **When to use recursion?**

    When compact, understandable, and intuitive code is required and where you want to avoid the need for explicit variable state management.
  - **When to use iteration?**

    When there is limited memory and faster processing is required and where more direct control over the flow of execution is required.

The Case for Recursion

- Recursion is suitable **for solving problems that exhibit repetitive**, **self-similar structures**, such as:
  - factorial calculation
  - Fibonacci sequence generation
  - tree traversal
- Recursion **requires careful handling of base cases** to avoid infinite recursion and stack overflow errors.

Recursive Functions

- Normally a **recursive function** uses conditional statements to determine whether or not to call the function recursively.
- The **main benefits** of recursion are:
  - compactness of code,
  - ease of understanding the code,
  - and having fewer variables.

Main Components

- **Base Case**

  The function **returns a value** when a certain condition is satisfied, **without any other recursive calls**.

- **Recursive Case**

  The function **calls itself** with an inputs that is a step **closer to the base case**.

CoGrammar

Base Case Component

- Base cases are the terminating conditions that **stop the recursion** and prevent the function from infinitely calling itself.
- These are **the simplest instances** of the problem that can be solved directly without further recursion.
- Without base cases, the recursive function would continue indefinitely, leading to stack overflow errors or infinite loops.

CoGrammar

Recursive Case Component

- Recursive cases define how the function calls itself with modified inputs to **solve smaller instances of the same problem**.
- In recursive cases, the function **applies the same algorithm to a reduced** or modified **version** of the original problem.
- By breaking down the problem into smaller subproblems and solving each subproblem recursively, **the function gradually approaches the base case(s)**.

CoGrammar

Recursive Function Structure

```python
def recursive_function(input):
    # Base case(s)
    if base_condition(input):
        # Return the result directly
        return base_result

    # Recursive case(s)
    else:
        # Modify the input and make a recursive call
        modified_input = modify_input(input)
        recursive_result = recursive_function(modified_input)

        # Further processing of the recursive result
        final_result = process_result(recursive_result)
        return final_result
```

CoGrammar

Recursive Function Structure ...

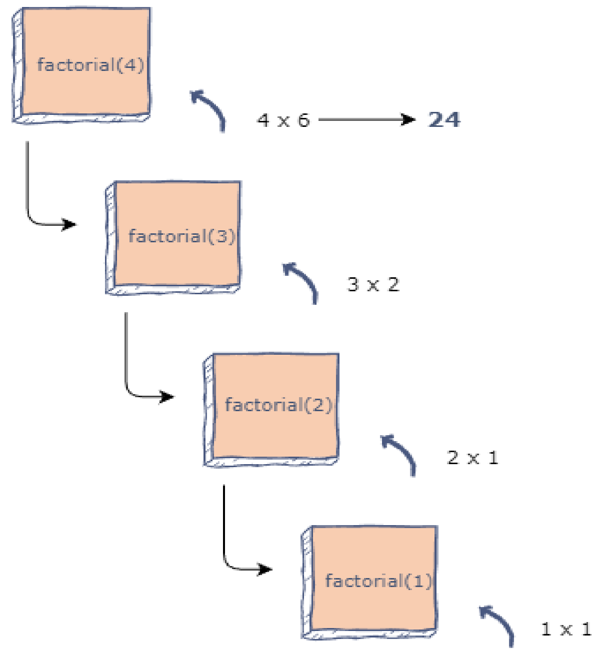
- The function **first checks for base cases** using if statements.
- **If** the base **condition is met**, the function **returns the base result** directly.
- **If** the base **condition is not met**, the function **proceeds to the recursive case(s)**.
- It **modifies** the input **parameters and** makes a recursive **call** to **itself** with the modified input.
- The process **continues** recursively **until the base case(s) are reached**, at which point the recursion **unwinds and returns the final result** back up the call stack.

Recursive Function Example

- **Computing Factorials**
  - Many **mathematical functions** can be defined using recursion. A simple example is a factorial function.
  - The factorial function, n! describes the operation of multiplying a number by all positive integers less than or equal to itself (excluding zero).
  - For example: 4! = 4*3*2*1

CoGrammar

Factorials Diagram



CoGrammar

Factorials Code

```
def factorial(num):
    if num == 1:
        return 1
    else:
        return num * factorial(num-1)
```

CoGrammar

Let’s take a short break

CoGrammar

Let’s get coding!

CoGrammar

Questions and Answers

CoGrammar

Summary: Recursion

- By **combining base cases and recursive cases**, recursive functions effectively break down complex problems into simpler subproblems and solve them iteratively until reaching a termination condition, providing an elegant and efficient approach to problem-solving in programming.

Summary: Structure

```python
def recursive_function(input):
    # Base case(s)
    if base_condition(input):
        # Return the result directly
        return base_result

    # Recursive case(s)
    else:
        # Modify the input and make a recursive call
        modified_input = modify_input(input)
        recursive_result = recursive_function(modified_input)

        # Further processing of the recursive result
        final_result = process_result(recursive_result)
        return final_result
```

CoGrammar

Summary: When to use

- **Use recursion:**

  When compact, understandable, and intuitive code is required and where you want to avoid the need for explicit variable state management.

- **Use iteration:**

  When there is limited memory and faster processing is required and where more direct control over the flow of execution is required.

Homework

- **Binary Search:**
  - Write a recursive function to perform a binary search on a sorted list of integers.
  - The function should return the index of the target element if found, or -1 if not found.
  - Example: binary_search([1, 2, 3, 4, 5, 6, 7, 8, 9], 5) should return index 4 when searching for 5.
- **Practice writing recursive functions** to solve additional problems.

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar