Welcome to this CoGrammar session:

Functions Revision

The session will start shortly...

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

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

✓ 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)

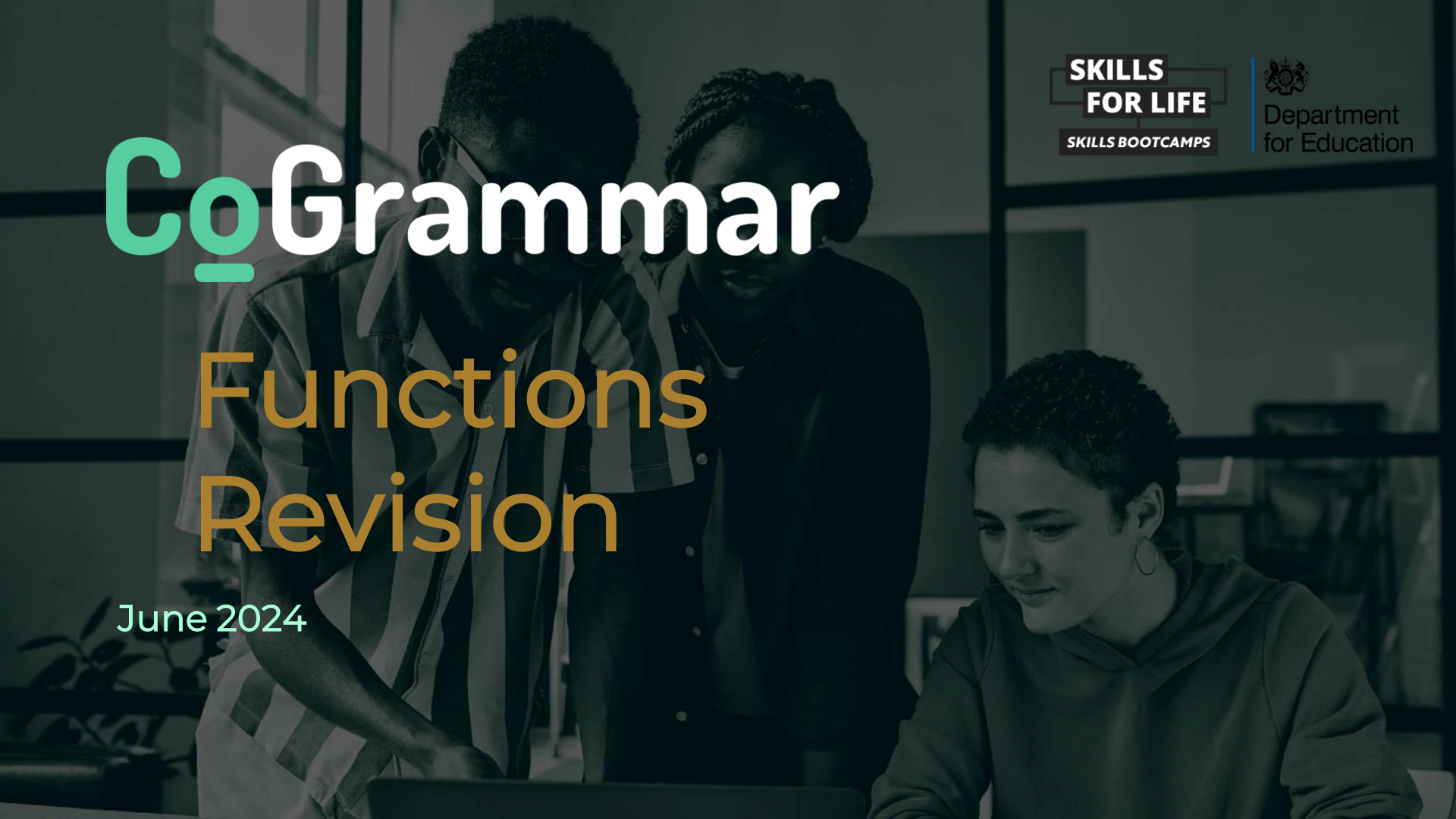
✓ Criterion 4: Demonstrating Employability

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

CoGrammar

Learning Outcomes

- Define what functions are and explain their purpose in coding.
- Utilise built-in functions such as print(), len(), and range() in their own projects.
- Create their own functions, defining custom behaviours and logic to solve specific problems.
- Implement functions that accept input through parameter variables.
- Design functions that return data to the caller.
- Apply higher-order functions, such as map(), filter(), and reduce().



SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar

Functions Revision

June 2024

What are Functions?

- Functions are reusable blocks of code that perform specific tasks.
- Called methods when used in OOP Classes.
- Functions help organise code, make it more readable, and facilitate debugging and maintenance.
- Useful for abstraction.
- Similarity to functions in maths, f(x) takes input x and produces some output.

How Functions Change Code

- Some Original Code Example:

```
if choice == "1":
    # Just tea
    print("Boil water.")
    print("Add tea bag to cup.")
    print("Add sugar to cup.")
    print("Add milk to cup.")
    print("Add boiling water to cup.")
    print("Stir.")
    print("Your tea is ready!")
if choice == "2":
    # Tea and Scones
    print("Cut open scone.")
    print("Add jam.")
    print("Add cheese.")
    print("Scone reasy!")

    print("Boil water.")
    print("Add tea bag to cup.")
    print("Add sugar to cup.")
    print("Add milk to cup.")
    print("Add boiling water to cup.")
    print("Stir.")
    print("Your tea is ready!")
```

How Functions Change Code

- Abstract code into functions:

```
def make_tea():
    print("Boil water.")
    print("Add tea bag to cup.")
    print("Add sugar to cup.")
    print("Add milk to cup.")
    print("Add boiling water to cup.")
    print("Stir.")
    print("Your tea is ready!")
```

```
def make_scone():
    print("Cut open scone.")
    print("Add jam.")
    print("Add cheese.")
    print("Scone reasy!")
```

How Functions Change Code

- Updated code that is organised, more readable with code reuse implemented:

```
if choice == "1":
    make_tea()
if choice == "2":
    make_scone()
    make_tea()
```

Calling Functions?

- Functions with one required positional input:
  - my_function(input1)
- Functions with two required positional inputs:
  - my_function(input1, input2)
- Functions with one required positional input and one optional keyword input:
  - my_function(input1, keyword_arg=input2)

Why Use Functions?

- Code Reusability: Write once, use multiple times.
- Modularity: Break down complex problems into simpler pieces.
- Maintainability: Easier to update and fix issues in a modular codebase.
- Abstraction: Hide complexity and expose simple interfaces.
- Error checking/validation: Makes this easier, as you can define all rules in one place.

Using Built-In Functions

- Python provides numerous built-in functions for common tasks.
- Examples: print(), len(), and range()

```
# Built-in functions
print("Hello, World!")
print(len("Hello"))
print(list(range(5)))
```

More Python Functions

- The list of functions that you can use in Python doesn't just stop with what is built in.
- Using Pip (python package manager), you can install various packages containing modules.
- To search for packages, visit https://pypi.org/
- Some packages are already installed by default in Python, such as the Math package.
- These modules can be imported into your script using an import statement.

More Python Functions

- Let's take a look at the maths module. Let's say that you want to use round(), which rounds a number off.
- There are multiple ways to access this:
  - import math –or- from math import *

    my_result = math.round(my_num, 2)
  - from math import round

    my_result = round(my_num, 2)

Creating Custom Functions

- Use the def keyword to define a function.
- Define a function to greet a user.

```
# Defining a custom function
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))
```

Functions with Parameters

- Functions can accept inputs through parameters.
- Example: Calculate the area of a rectangle.

```
# Function with parameters
def calculate_area(length, width):
    return length * width

print(calculate_area(5, 3))
print(calculate_area(7, 2))
```

Functions with Return Values

- Use the return statement to return a result from a function.
- Example: Return the square of a number.

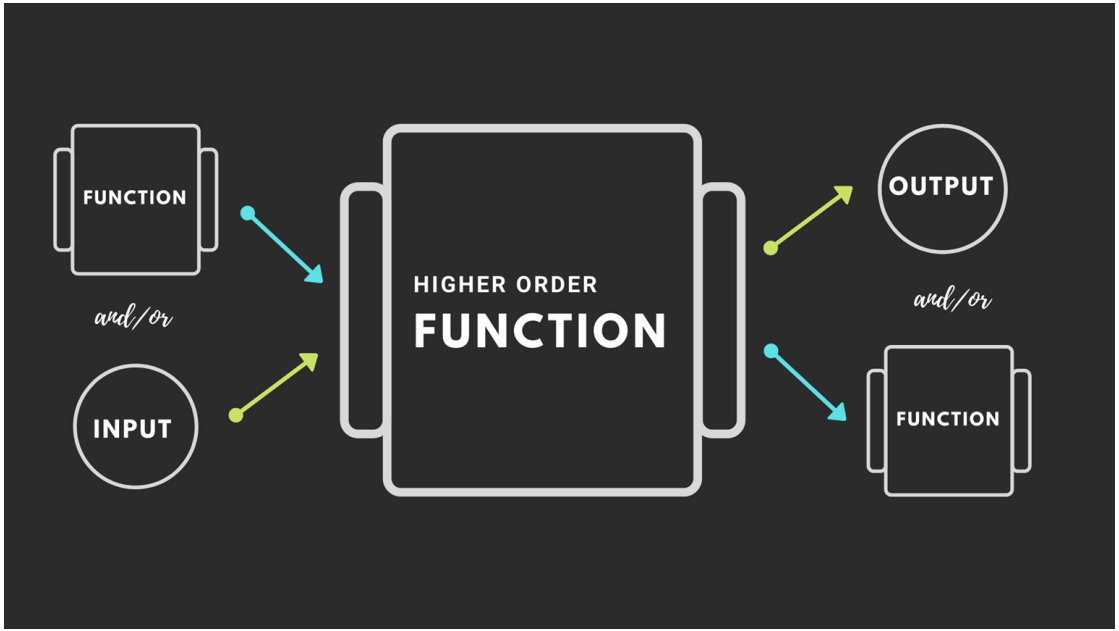
```
# Function with a return value
def square(number):
    return number * number

result = square(4)
print(result)
```

Higher-Order Functions (HOF)

- A function that takes other functions as arguments (or returns a function) is called a higher order function.
- Higher-order functions operate by accepting a function as an argument, altering it, and then returning the altered function.
- More modular and reusable code can be produced as a result.

What makes HOF so powerful?



CoGrammar

Power of HOF

- **Abstraction and Modularity**: You can abstract intricate patterns and operations into functions by using HOF. This encourages modularity, which improves readability and ease of maintenance for your code.
- **Code Reusability**: You can reuse code more effectively by passing functions as arguments to other functions. This encourages the DRY (Don't Repeat Yourself) principle and lessons redundancy.

Power of HOF

- **Functional Composition:** By combining simpler functions to produce more complex ones, HOF enables functional composition. This allows you to build operations step-by-step.
- **Adaptability when Transforming Data**:
  Strong tools for handling and altering data are provided by functions like map(), filter(), and reduce(). They let you write operations clearly, often in just one line of code.

Applying Higher-Order Functions

- Examples: map(), filter().
- map() and filter() are built-in functions

```
# Higher-order functions
numbers = [1, 2, 3, 4, 5]

# Using map
squared = list(map(lambda x: x * x, numbers))
print(squared)

# Using filter
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)
```

Applying Higher-Order Functions

- Example: reduce().
- reduce() is contained in the functools() module

```
from functools import reduce

# Higher-order functions
numbers = [1, 2, 3, 4, 5]

# Using reduce
total = reduce(lambda x, y: x + y, numbers)
print(total)
```

Best Practices for Functions

- Descriptive Function Names: Instead of foo() or bar(), let's name our functions so that anyone reading our code knows exactly what's going on.

```
def calculate_area(radius):
    # Code for calculating area
```

Best Practices for Functions

- Single Responsibility Principle: One function, one responsibility. Break down your code into smaller, focused functions.

```
# Step 1:
def fetch_data(url):
    print("Code to fetch data from the URL.")

# Step 2:
def process_data(data):
    print("Code to process this data.")
```

Best Practices for Functions

- Avoiding Global Variables: Global variables can be tricky. Stick to local scope and keep your functions pure.

```
count = 0
def increment_count():
    global count
    count += 1
    return count

#   Better:
def increment_count(count):
    return count + 1
```

Best Practices for Functions

- Docstrings: A form of documenting your functions. Think of these as user manuals for each function.

```
def calculate_area(radius):
    """
    Calculate the area of a circle.

    :param radius: The radius of the circle.
    :type radius: float
    :return: The area of the circle.
    :rtype: float
    """
    # Code for calculating area
```

Let’s take a short break

CoGrammar

Let’s get coding!

CoGrammar

Polls

- *Refer to the polls section to vote for you option.*

1. What is the primary purpose of a function in programming?
   a. To store large amounts of data
   b. To execute a specific task or set of tasks
   c. To improve the graphical interface of a program
   d. To connect to a database

Polls

- *Refer to the polls section to vote for you option.*

2. Which of the following is NOT a built-in Python function?
   a. print()
   b. len()
   c. range()
   d. execute()

Polls

- *Refer to the polls section to vote for you option.*

3. **Which of the following best describes a higher-order function in Python?**

   a. A function that prints the highest value in a list.

   b. A function that takes another function as an argument or returns a function as a result.

   c. A function that operates only on numerical values.

   d. A function that can handle an arbitrary number of arguments.

Questions and Answers

CoGrammar

Summary

- Defined what functions are and their purpose.
- Utilised built-in functions like print(), len(), and range().
- Created custom functions to solve problems.
- Implemented functions with parameters to process input data.
- Designed functions with return values.
- Applied higher-order functions for modular code.

Resources: Functions

- Official Python Documentation:
  - https://docs.python.org/3/tutorial/controlflow.html#defining-functions
- Online Tutorials:
  - https://realpython.com/,
  - https://www.w3schools.com/
- Additional Reading:
  - "Automate the Boring Stuff with Python" by Al Sweigart

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar