



# CoGrammar

## Workshop 2 - Sequences



**SKILLS  
FOR LIFE**

**SKILLS BOOTCAMPS**



Department  
for Education

## 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](#)

## Session Housekeeping cont.

---

- For all **non-academic questions**, please submit a query:  
[www.hyperiondev.com/support](http://www.hyperiondev.com/support)
- Report a **safeguarding** incident:  
[www.hyperiondev.com/safeguardreporting](http://www.hyperiondev.com/safeguardreporting)
- We would love your **feedback** on lectures: [Feedback on Workshops](#)



# Skills Bootcamp Certification Overview

✓ **Criteria 4: Original Deadline  
for Demonstrating  
Employability without  
Co-certification**

**Record a job outcome by  
23 September 2024**


✓ **Criterion 4: Updated  
Deadline for Imperial College  
London & University of  
Manchester Co-certification**

**Record a job offer by  
15 July 2024**




# Lecture Objectives

- Understand the difference between data types and data structures
- Effectively compare the performance benefits and drawbacks between different compound data types
- Understand how to formulate an approach to solving a technical problem
- Identify the suitable compound data types to apply to their solution
- Combine understanding of compound data types and problem-solving to justify the approach taken to solve a problem





**Can you work through a coding challenge  
without any external assistance  
(excluding basic syntax checks)**

- 
- A. Yes
  - B. No
  - C. Somewhat
- 

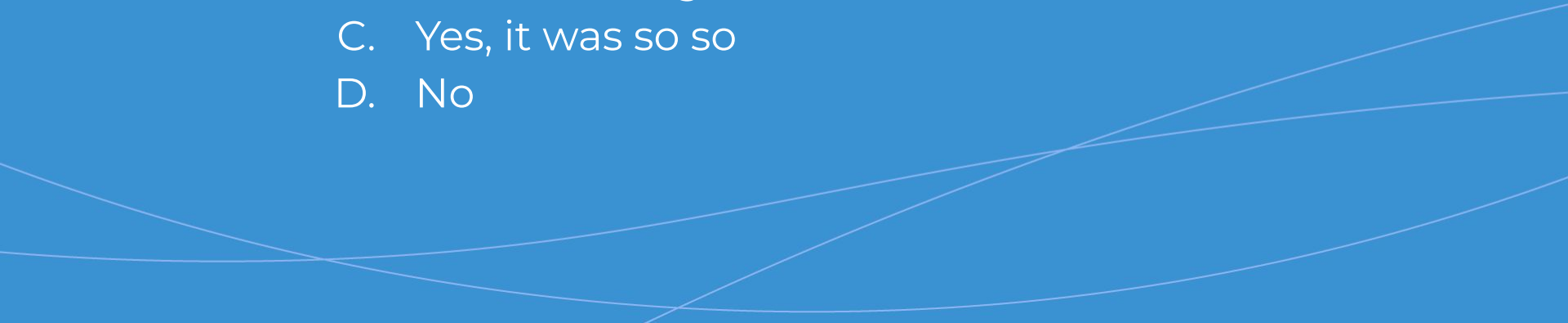


**How often are you attempting coding challenges, (Choose the option that is closest to your frequency)**

- A. Daily
  - B. Weekly
  - C. Monthly
  - D. Never
- 



**Have you done any coding challenges for  
a job application and/or coding interview  
(elaborate on your experience in the  
question section)**

- A. Yes, it went well
  - B. Yes, it didn't go well
  - C. Yes, it was so so
  - D. No
- 



# Lesson Structure

## What are we doing here

1. Analyse a problem statement
2. Learn how to apply a problem solving methodology
3. Analyse different data types and data structures that will help us solve most coding problems
4. Apply these techniques to solve our problem optimally.

# Problem Solving

## What is this?

The process of identifying a problem, understanding the problem and creating a solution to this problem.

## Importance

- Software Engineering
  - Building custom algorithms
  - Designing applications
  - Configuring cloud infrastructure
  - Debugging
- Data Science
  - Framing research questions
  - Efficiently cleaning datasets
  - Designing proper visualization
  - Debugging

## Problem Statement

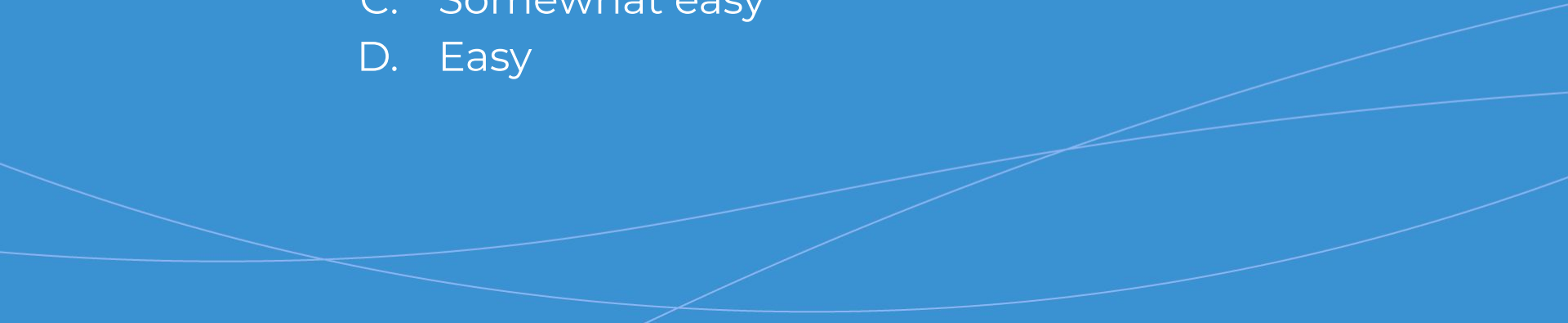
There are  $x$  children seated in a straight line, each child has a number on their t-shirt to uniquely identify them.

The children are playing a game where the teacher will say a number  $y$  and each child will need to move  $y$  seats to the right.

Our goal is to simulate a round of this game and produce the final order that the children will be seated.



**Rate the difficulty of the problem and  
suggest an approach to solving it  
(suggestions in the questions tab)**

- A. Hard
  - B. Somewhat hard
  - C. Somewhat easy
  - D. Easy
- 

# Effective Problem Solving

## Steps

1. Understand the problem (Conceptually)
2. Understand the problem (Deeper)
3. Identify data structures for implementing your solution
4. Formalise your solution
5. Create the solution
6. Optimise the solution!

# Understand the Problem (Conceptually)

## Methods

- Coding Challenge
  - Read through the problem until you can restate it in different words without it losing its meaning
  - Look for constraints that have been explicitly stated
    - Input range, values to expect, etc.
  - Look at example outputs
    - Match example outputs to aspects of the requirements
  - Think of possible edge cases
    - eg. What if the input is null
- Interview
  - Ask questions
    - Possible inputs
    - Expected outputs (if not explicitly stated)
    - What would happen when certain values are passed
      - Null or empty values typically
    - How should the code behave when a certain point is reached

# Understand the Problem (Conceptually)

There are  $x$  children seated in a straight line, each child has a number on their t-shirt to uniquely identify them. The children are playing a game where the teacher will say a number  $y$  and each child will need to move  $y$  seats to the right. Our goal is to simulate a round of this game and produce the final order that the children will be seated.

## What we can gather

- There are 0 to many children (We can safely assume that we can't have negative children)
- There is an order to how the children are presented
- Each child has a unique identifier
- The teacher triggers an action
- Each child has to move a certain amount of times
- Each child can only move to the right
- The final order needs to be produced based on each child moving a certain number of times to the right.

## Questions

- What happens if there are no seats to the right of a given child
  - Are there more open seats?
  - Do the children get eliminated
- What is the largest number that the teacher can call?

## Understand the Problem (Conceptually)

### Answer:

The number of seats will always be equal to the number of children, if there are no seats to the right of a child, the child can move to the first open seat at the furthest point to the left.

### Final Understanding

- We have 0 to many children
- There is an order to how the children are presented
- Each child has a unique identifier
- The teacher triggers an action
- Each child has to move a certain amount of times
- Each child can only move to the right, **Unless they are at the end, then they can move to the first open seat to the left**
- The final order needs to be produced based on each child moving a certain number of times to the right.



# CoGrammar

## Q & A SECTION

**Please use this time to ask  
any questions relating to the  
topic, should you have any.**

# Understand the Problem (Deeper)

## Visualisation Process

- Draw a diagram that best represents the process to **YOU**
- Go through the steps required to solve the problem, and update the diagram to match the current state

## Goals

- Discover patterns
  - Identify where we might need to use loops
  - Identify aspects that are isolated enough to be their own functions
- See how the process changes as the input changes.
- Identify choices that need to be made
  - eg. 'If I'm at this state, I can move to these states', 'if x is = to y, I can only do z'
- Visualise data structures
  - See where values are being stored as operations are being performed
    - Is there a need for temporary variables
    - Which data structure is best for storing the input and presenting the output

## Tools

[Draw.io](https://draw.io)

[Excalidraw](https://excalidraw.com)

# CoGrammar

## Q & A SECTION

**Please use this time to ask  
any questions relating to the  
topic, should you have any.**

# Data Storage

While gaining an understanding of the problem, you will eventually start to see the core data structures that will be required to solve the problem. This includes:

- Input parameters
- Return value
- Building the output
- Storing temporary values
- Etc

Not all data types are created equally, we need to make wise choices based on the constraints of our problem.

# Primitive Data Types

## What are they

- Built into the programming language
- Not derived from objects

## Benefits

- Very efficient
  - Value types - Stored in the stack
- Cannot be null, so there is always a value

## Primitive Data Types

Type	Description	Default value	Size	Range
boolean	True or false can also be 0 and 1	false	1 bit	True, false
char	Unicode character	\u0000	16 bits	ASCII values from 0 to 255
byte	8 bit number	0	8 bits	-128 to 127
Short	Short integer	0	16 bits	-32768 to 32767
int	Normal integer	0	32 bits	-2147483648 to 2147483647
long	Long integer	0	64 bits	-9223372039854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	Up to 7 decimal digits
double	Double precision float	0.0	64 bits	Up to 16 decimal digits

# Compound Data Types

## What are they

- Consist of primitive data types
- Create objects that store the primitive data types (or other compound data types in some situations)

## Benefits

- Makes storing related data easier
- Come with built-in methods for performing certain operations
- Key to most programming operations

## Cons

- Reference types (stored in the heap)
  - Note, some languages have certain compound data types that are value types

## (main) Compound Data Types

Type	Description	Insertion	Deletion	Value Search
String	A collection of characters	$O(n^2)$	Not supported (but can be done)	$O(n)$
Array	A fixed size collection of value of the same type	$O(1)$	Not supported	$O(n)$
List	Dynamic list of values (typically of the same type)	$O(1)$ - End $O(n)$ - Front	$O(1)$ - End $O(n)$ - Front	$O(n)$
Hash Table (Dictionary)	Collection of unique key-value pairs	$O(1)$	$O(1)$	$O(1)$
Hash Set (Set)	Collection of unique values	$O(1)$	$O(1)$	$O(1)$



# Strings

## What is it?

- A collection of characters (in most languages)
- Stores unicode values
- Most used compound data type due to it's versatility

```
hello_world = "Hello, world"

# slicing
hello_world[1:5] # output: "ello"

# Case
hello_world.upper() # Output: "HELLO, WORLD"
hello_world.lower() # Output: "hello, world"
```

## Strings

### Tips

- Avoid string concatenation
  - In Python or JavaScript, use the a list and the string.join() method
  - In Java or C#, use the string builder.
- If values are being changed continuously, convert the string to a list/array
- Go through the built in string methods [here](#), very helpful
  - Keep in mind that each method will have it's own own time complexity
- Slicing in Python has a constant time complexity, so getting substrings in Python is light work

## Array

### What is it?

- A collection of same data type values
- Has a fixed size
- When working with primitive data types, the values are stored contiguously
- No built in implementation in Python

## Lists

### What is it?

- A dynamic collection of data
- Values can be dynamically inserted and removed

```
values = ['h', 'e', 'l', 'l', 'o']  
  
# Insertion  
values.append('world') # output: ['h', 'e', 'l', 'l', 'o', 'world']  
  
# Joining and slicing  
hello = "".join(values[:5])  
  
# Deletion  
values.pop(2) # output: ['h', 'e', 'l', 'o', 'world']
```

## Lists

### Tips

- Good if order is important and you need to insert and remove values frequently
- Good if you need to store duplicate values
- Can take the place of strings for complex operations
  - In Python main
  - In another language, you can use Arrays if you are not removing or adding values
- Look at the different methods [here](#)
  - Keep in mind that each method runs some underlying code, you can not always be sure that this code is more efficient than creating your own implementation
- Slicing is very efficient

# Dictionary

## What is it?

- Known as a Hash Table or Hash Map
  - Implemented as a dictionary in Python
- Stores key-value pairs
- Has a constant time for accessing values.
- Does not store values in order like a list, array or string.

```
albums = {  
    "Avenge Sevenfold": "Life is But a Dream...",  
    "Casey": "Love is Not Enough"  
}  
  
# Get  
albums["Casey"] # Output: "Love is Not Enough"  
  
# Insert  
albums["Shadow of Intent"] = "Reclamer"  
  
# Update  
albums["Avenge Sevenfold"] = "Hail to the King"
```

## Dictionary

### Tips

- Perfect for showing relationships between unique keys and associated values
  - eg. artist is a unique key, the artist can have many albums (which can potentially have the title as another artist)
- Good choice if order doesn't matter
- Good if you know that you won't have duplicate keys
- Take a look at the dictionary methods [here](#)

## Sets

### What is it?

- Known as a Hash Set
  - Implemented as a Set in Python
- Stores unique values
- Constant lookup time
- Constant insertion time

```
colours = {"red", "green", "blue"}  
  
# Contains Value  
"red" in colours # output: True  
"yellow" in colours # output: False  
  
# Insert Duplicate  
colours.add("red") # output: {"red", "green", "blue"}
```



## Sets

### Tips

- Use them when working with unique values
- If you constantly need to check if a value exists, use a set
  - Lookups have a constant time complexity
- Good if order doesn't matter

## Honourable Mentions

Type	Description	Insertion	Deletion	Value Search
Classes	Used to create custom compound data types	Depends on the data structure	Depends on the data structure	Depends on the data structure
Enumeration (Enum)	User defined data structure for storing related constant values	Not supported	Not supported	O(1)

```
class LinkedList:  
    value: int  
    next: LinkedList
```

```
from enum import Enum  
  
class Day(Enum):  
    Monday = "Monday"  
    Tuesday = "Tuesday"  
    Wednesday = "Wednesday"  
    Thursday = "Thursday"  
    Friday = "Friday"
```

# CoGrammar

## Q & A SECTION

**Please use this time to ask  
any questions relating to the  
topic, should you have any.**

## Problem Statement

There are  $x$  children seated in a straight line, each child has a number on their t-shirt to uniquely identify them.

The children are playing a game where the teacher will say a number  $y$  and each child will need to move  $y$  seats to the right.

Our goal is to simulate a round of this game and produce the final order that the children will be seated.

## Understand the Problem (Conceptually)

### Answer:



The number of seats will always be equal to the number of children, if there are no seats to the right of a child, the child can move to the first open seat at the furthest point to the left.

### Final Understanding

- We have 0 to many children
- There is an order to how the children are presented
- Each child has a unique identifier
- The teacher triggers an action
- Each child has to move a certain amount of times
- Each child can only move to the right, **Unless they are at the end, then they can move to the first open seat to the left**
- The final order needs to be produced based on each child moving a certain number of times to the right.



# Which data types do we need to solve this problem?

- 
- A. List
  - B. String
  - C. Set
  - D. Dictionary
- 

# Recap

## Problem Solving Process

- Understanding the problem
  - Conceptually
  - Visually
- Determine the data types required
  - Weigh the strengths and weaknesses

## Primitive Data Types

- The most basic data types we have
- Value types

## Compound Data Types

- Store a collection of primitive data types
- Key to managing data in our algorithms
- Have different strengths and weaknesses



# CoGrammar

**Thank you for joining!**