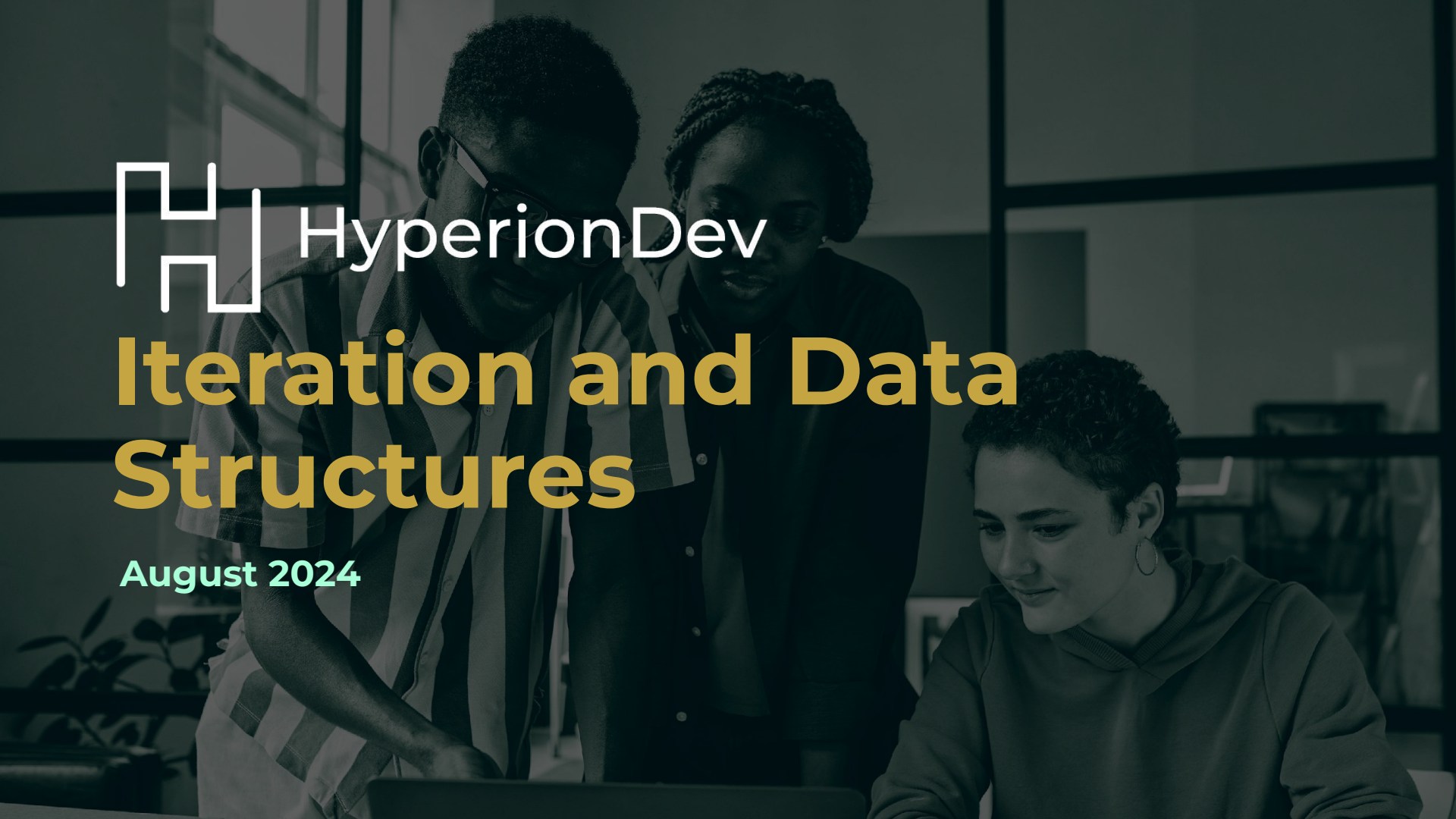

HyperionDev

Iteration and Data Structures

August 2024

Data Science Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- 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**

Data Science 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**

Learning Outcomes

- **Demonstrate** iteration using for and while loops to handle repetitive tasks in Python.
- **Compare** for and while loops and lists and dictionaries, and evaluate their use cases.
- **Create and manipulate** data structures such as lists and dictionaries to store and process collections of related data in Python

HyperionDev

Problem Statement

In programming, we are often interested in how to do the same task repetitively, given different inputs and producing different outputs.

- How can we produce this behaviour in our code, and,
- How can we store the inputs and outputs of the tasks?

Lecture Overview

- Data Structures
  - Lists
  - Dictionaries
- Iteration
  - While Loops
  - For Loops

HyperionDev

Data Structures

HyperionDev

Data Structures

A data organisation and storage format that is usually chosen for efficient access to data.

- They are fundamental to organising and managing data efficiently.
- In Data Science, this is particularly important because we often work with large data sets, that we need to access quickly and frequently.
- In Python, there are 4 built-in data types which hold collections of data:
  - **Lists:** Ordered, mutable collection of items.
  - **Dictionaries:** Collection of key-value pairs.
  - **Sets:** Ordered, immutable collection of items.
  - **Tuples:** Unordered collections of unique items.

HyperionDev

Lists

Ordered, mutable collections of data.

- Items in a list are known as **elements**.
- Elements do not have to be unique nor of the same type.
- Lists are **mutable**, meaning that elements in the list can be changed.
- Use **square brackets** to create lists and separate values with **commas**:

```
my_list = [1, "two", "buckle", True]
```

- We can access elements in a list using indexing, which is based on the element's position in the list:

```
print(my_list[0])  # 1
print(my_list[3])  # True
```

HyperionDev

Lists

- The most commonly used list functions are:
  - Adding an element

```
my_list.append("three")
my_list.insert(0, "zero")
```

  - Deleting an element

```
my_list.remove("zero")
my_list.pop(3)
```

  - Manipulating the list: sorting, reversing etc.

```
my_list.sort()
my_list.reverse()
```

HyperionDev

Strings

- Strings are considered to be **immutable** collections of sequences in Python.
- We can access characters in our strings the same way we can access elements in a list:

```
string = "hello"
print(string[0]) # h
```

- We can also manipulate strings using the same methods that we use on lists:

```
string.find("h")
```

HyperionDev

Dictionaries

Collections of key-value pairs, where each key is unique.

- Unlike lists, dictionaries distinguish each element in the collection using a **key** instead of an **index**.
- When we use dictionaries to study languages, we look up definitions of a given word by looking up the word in the dictionary.
- In the data structure, we can access the **value associated with a key** value by looking up the key in the dictionary.
- Each element in a dictionary is a **key-value pair**.
- To create a dictionary, keys and values are **separated by colons (:)** and pairs are **separated by commas** and **enclosed in curly brackets {}**.

Dictionaries

- We can also use the **dict** function to create dictionaries:

```
my_dict = {"name": "Zahra", "age": 24}
my_dict = dict(name = "Zahra", age = 24)
```

- To access values in a dictionary:

```
my_name = my_dict["name"]
```

- To add elements to a dictionary:

```
my_dict["bday"] = "13 November"
```

- To delete elements in a dictionary:

```
my_dict.pop("bday")
```

Iteration

HyperionDev

Iteration

The process of repeatedly executing a set of instructions or a block of code

- In Data Science, it is often necessary to perform repetitive tasks on large datasets.
- Iteration allows us to **efficiently process data**, **automate repetitive tasks**, and **apply operations across collections of data**.
- **Loops** allow us to repeat the same block of code multiple times, for a set number of times or until a certain condition is met.
- In Python, we use different types of **loops** for iteration.
  - **While Loops:** Repeats until a specified condition is met.
  - **For Loops:** Repeats for each item in a specified sequence.

While Loop

Repeats a block of code until a specified condition is met.

- When creating a while loop, specify a **condition** and a **block of code**.
- The block of code will continue to execute while the **condition is evaluated to True** and stop once the condition is evaluated to False.
- The structure of a while loop is as follows:

```
while (condition):
    # Block of code to repeat
```

HyperionDev

For Loop

Repeats a block of code for each element of a specified collection.

- For loops are generally used when the **number of executions is known**.
- The structure of the for loop is as follows:

```
for loop_variable in sequence:
    # Block of code to repeat
```

- A sequence (like a list, tuple or string) must be specified when creating a for loop, an existing sequence can be used directly or one can be created for the loop using the **range(start, step, stop)** function:

```
range(20)       # [0 .. 19]
range(1, 20)    # [1 .. 19]
range(1, 2, 20) # [1, 3, 5, .. 19]
```

HyperionDev

Loop Control Statements

- These statements when we need to intercept the usual flow of the loops in our program.
  - **break:** exits the loop immediately.
  - **continue:** skips the rest of the loop's current iteration and moves to the next iteration.
  - **pass:** acts as a placeholder in a loop when no action is required.

Questions and Answers

HyperionDev

Thank you for attending

HyperionDev