



# CoGrammar

## Higher Order Functions

**SKILLS  
FOR LIFE**

**SKILLS BOOTCAMPS**



Department  
for Education

# Software Engineering Lecture Housekeeping

---

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.  
**(FBV: Mutual Respect.)**
- 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 Open Classes.  
You can submit these questions here: [Open Class Questions](#)

## Software Engineering Lecture Housekeeping cont.

---

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

# Progression Criteria

## ✓ **Criterion 1: Initial Requirements**

- Complete 15 hours of Guided Learning Hours and the first four tasks within two weeks.

## ✓ **Criterion 2: Mid-Course Progress**

- Software Engineering: Finish 14 tasks by week 8.
- Data Science: Finish 13 tasks by week 8.

## ✓ **Criterion 3: Post-Course Progress**

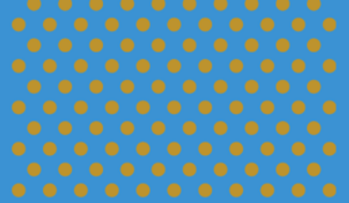
- Complete all mandatory tasks by 24th March 2024.
- Record an Invitation to Interview within 4 weeks of course completion, or by 30th March 2024.
- Achieve 112 GLH by 24th March 2024.

## ✓ **Criterion 4: Employability**

- Record a Final Job Outcome within 12 weeks of graduation, or by 23rd September 2024.

# Lecture Objectives

1. Recall functions and define what a function in programming is.
2. Expand on functions by exploring Higher Order Functions.
3. Apply the above knowledge to improve data management in programs.


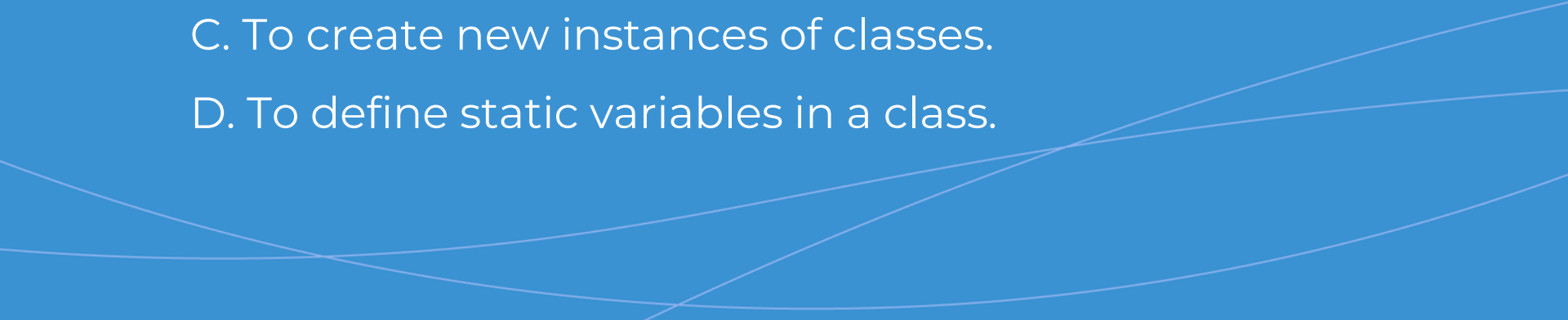


# Poll:

## Assessment


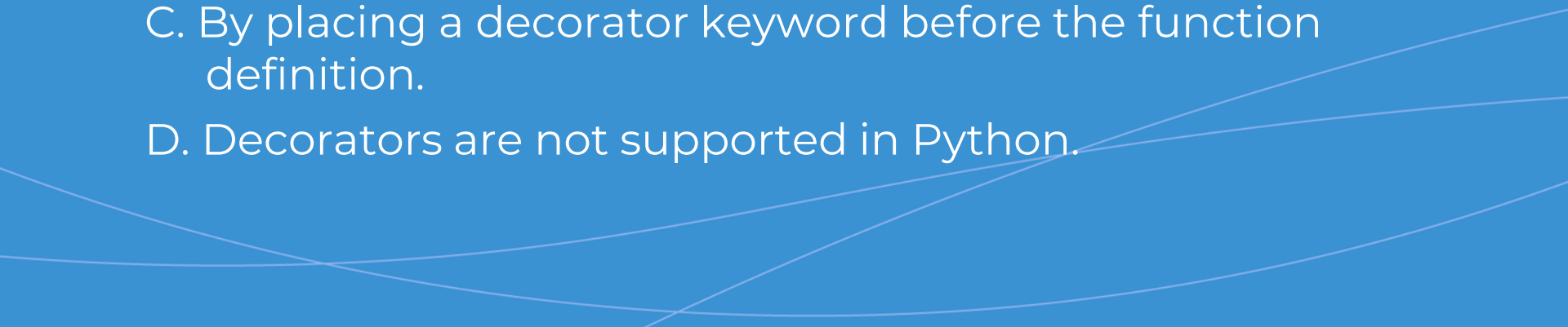


# What is the primary purpose of decorators in Python?

- 
- A. To add metadata to functions and classes.
  - B. To modify the behaviour of functions or methods.
  - C. To create new instances of classes.
  - D. To define static variables in a class.
- 




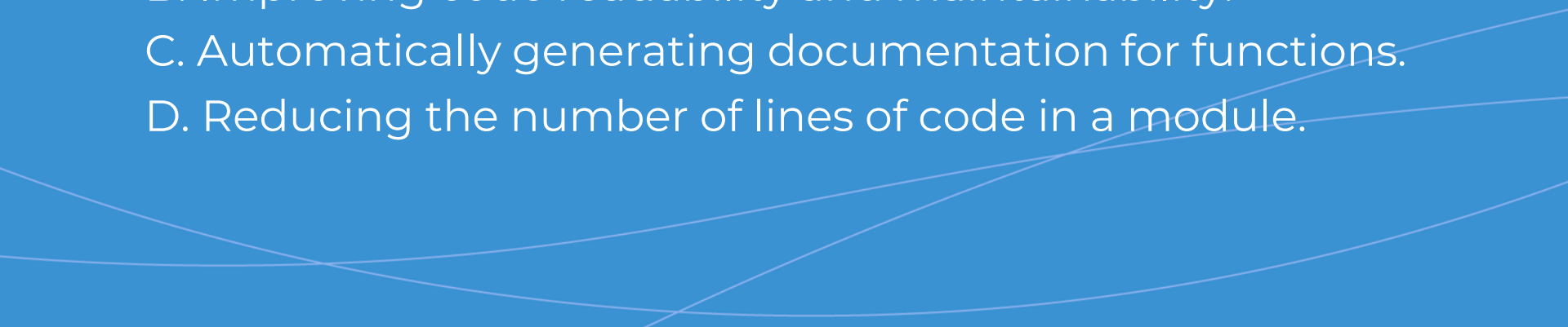
# What is the standard way to define a decorator in Python?

- 
- A. Using the @decorator syntax.
  - B. Using the decorator = decorator\_function(target\_function) syntax.
  - C. By placing a decorator keyword before the function definition.
  - D. Decorators are not supported in Python.
- 





# Which of the following is a common benefit of using decorators?

- 
- A. Making functions private and inaccessible outside the module.
  - B. Improving code readability and maintainability.
  - C. Automatically generating documentation for functions.
  - D. Reducing the number of lines of code in a module.
- 

# What is Functional Programming?

- ★ Functional programming is a programming paradigm in which you operate on functions as you would with any other kinds of values. This makes functions in functional programming first-class, meaning they are just like any other value you work with. Higher order functions and closures are fine examples of how you would treat functions as values.

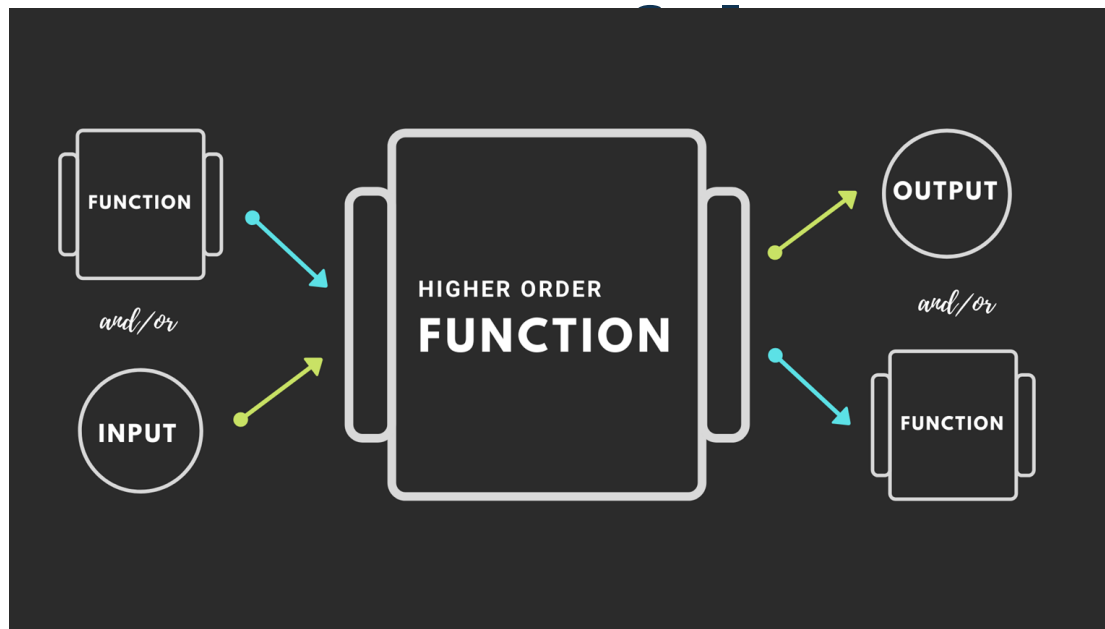
# First Class Functions

When a function is considered first-class, it indicates that the language views it as a value, allowing you to pass it around and assign it to variables. It is seldom used to describe a function, as in "a first-class function." Saying "a language has/hasn't got first-class function support" is far more common. Thus, a language has a "first-class functions" attribute.

# What is a Higher order Function (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



# 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.

- ★ **Functional Composition:** By combining simpler functions to produce more complex ones, HOF enable functional composition. This allows you to build operations step-by-step, resulting in code that is expressive and concise.
- ★ **Declarative Programming\*:** Programming becomes more declarative when HOF is used. Rather than giving explicit instructions on how a task should be completed (imperative), you state what the desired outcome should be, and the HOF handles the rest.

★ **The paradigm of functional programming\*:**

Functional programming paradigms are supported by Python, and HOF are an essential component of this approach. Adopting functional programming concepts can result in code that is easier to read and maintain.

★ **Adaptability when Transforming Data\*:**

Strong tools for handling and altering data are provided by functions like `reduce()`, `filter()`, and `map()`. They let you write operations clearly, often in just one line of code.



★ **Lambda Operations (`lambda <parameters>: <expression>`):**

Working with HOF is made even more concise by Python's support for lambda functions, also known as anonymous functions. You can write quick, disposable functions right within your code.

★ **Expression and Readability:**

HOF, which places more emphasis on the what than the how, can help you write more expressive and readable code. This can result in code that is easier to read and maintain, particularly when working with data manipulation tasks.

★ **Using Iterables in Integration:**

HOF integrates with Python's iterable objects with ease. This makes them particularly useful for tasks involving collections like lists, tuples, and dictionaries..

# Built-in HOF

- ★ A few useful higher-order functions are `map()`, `filter()`, and `reduce()`. `map()` and `filter()` are built-in functions and `reduce()` is contained in the `functools()` module.

```
numbers = [1,2,3,4,5]
square = map(lambda i: i**2 , numbers)
print (square)

print (list(square))
#Output: [1, 4, 9, 16, 25]
```




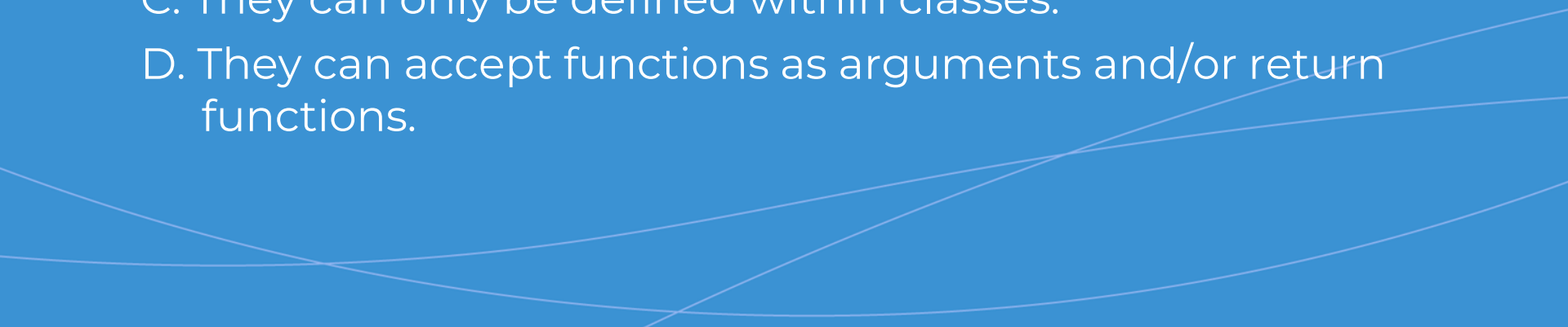
# Poll:

## Assessment




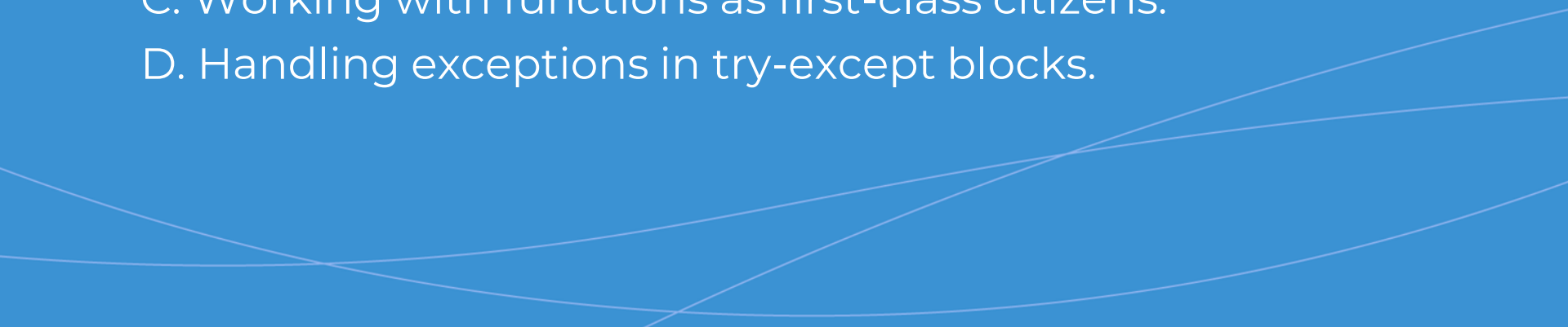


# What is a key characteristic of Higher-Order Functions (HOFs)?

- 
- A. They can only take and return functions as arguments.
  - B. They operate exclusively on numeric data types.
  - C. They can only be defined within classes.
  - D. They can accept functions as arguments and/or return functions.
- 



# What is a common use case for Higher-Order Functions?

- 
- A. Defining class attributes and methods.
  - B. Implementing conditional statements in functions.
  - C. Working with functions as first-class citizens.
  - D. Handling exceptions in try-except blocks.
- 

# Wrapping Up

---

## First Class Functions

Recognizing the power of first-class functions in Python allows us to treat functions as versatile entities leading to elegant, functional and dynamic programming

## Higher Order Functions

In conclusion, practicing with Higher Order Functions not only empowers you to create modular and reusable code, but demonstrates Python's support for functional programming paradigms, making your codebase more expressive and flexible.

# CoGrammar

Questions around Functions



# CoGrammar

**Thank you for joining**