* Overview of Decorators
* Functions as objects
* Syntax for decorators
* Creating decorators
* Annotating decorators
* Decorating functions with arguments
* Using multiple decorators
* Real World Implementation of Decorators
* Exercise and Solution

1. What is a decorator in Python?

A. A design pattern used to enhance the behavior of objects at runtime.

B. A special syntax used to create functions with special capabilities.

C. Function that takes another function as an argument with out modifying its structure.

D. A way to define variables with special properties.

Answer: C

1. What is the syntax for applying a decorator to a function?

A. @decorator

B. #decorator

C. \*decorator

D. $decorator

Answer: A

1. How do you create a decorator function in Python?

A. Define a function and use the @ symbol before its name when applying it to another function.

B. Define a function and use the \* symbol before its name when applying it to another function.

C. Define a function and use the # symbol before its name when applying it to another function.

D. Define a function and use the $ symbol before its name when applying it to another function.

Answer: A

1. What is an annotated decorator in Python?

A. A decorator that has been documented with comments.

B. A decorator that has been marked with a special symbol for easy recognition.

C. A decorator that takes arguments.

D. A decorator that has been tested extensively.

Answer: C

1. How do you annotate a decorator in Python?

A. By defining a decorator function that accepts arguments.

B. By including a comment before the decorator definition.

C. By marking the decorator with a special symbol.

D. By testing the decorator extensively.

Answer: A

1. Can you decorate a function with multiple decorators?

A. No, maximum two decorators can be added to a function.

B. No, a function can only be decorated with one decorator.

C. Yes, multiple decorators can be added to a function

D. Yes, but only if the decorators are defined using different syntax.

Answer: C

1. What is a real-world use case for decorators in Python?

A. Logging function calls to a file.

B. Benchmarking function performance.

C. Validating function arguments.

D. All of the above.

Answer: D

1. Can decorators be used to modify the behavior of built-in Python functions?

A. Yes, allow you to change the behavior of a function without modifying the function itself.

B. No, decorators can only be applied to user-defined functions.

C. Yes, but only if the built-in function is explicitly allowed to be decorated.

D. No, decorators doesn't allow to change the behaviour of a function.

Answer: A

1. What is a common way to handle errors in decorator functions?

A. Raising an exception.

B. Swallowing the error and returning None.

C. Logging the error and continuing execution.

D. Prompting the user for input to handle the error.

Answer: A

1. What is the main benefit of using decorators in Python?

A. They simplify complex functions by breaking them down into smaller pieces.

B. They allow for the dynamic modification of function behavior at runtime.

C. They improve code readability by providing a clear separation of concerns.

D. They make it easier to test and debug functions.

Answer: B