

Variables

Lists

1. **1. Lists**

2. **2. Lists**

3. **3. Lists**

4. **4. Lists**

5. **5. Lists**

6. **6. Lists**

7. **7. Lists**

8. **8. Lists**

9. **9. Lists**

10. **10. Lists**

11. **11. Lists**

12. **12. Lists**

13. **13. Lists**

14. **14. Lists**

Functions

While-loop

Methods

Match-case

For-loop

Type-conversion

List Indices

List Indices

Tuples

Enumerate

F-Strings

What

• A string format that allows you to embed expressions inside string literals, using curly braces `{}`.

• The expressions will be replaced with their values.

• The expressions can be arbitrary Python expressions, and are evaluated before the string is created.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

• The expressions can be replaced by the string `{}` if you don't want to embed any expressions.

Write Text Files, Read Text Files

Write Text Files, Read Text Files

File Paths

File Paths

List Comprehensions

List

Python lists are ordered, mutable collections of elements.

They can contain elements of different types (integers, strings, lists, etc.).

Lists are created using square brackets `[]`.

Example: `my_list = [1, 2, 3, 'apple', [4, 5]]`

Lists are mutable, meaning you can change their contents.

Example: `my_list[0] = 10` (changes the first element to 10)

Lists support indexing and slicing.

Example: `my_list[2]` (accesses the third element, 3)

Example: `my_list[1:4]` (slices from index 1 to 4, excluding 4)

Lists are iterable, meaning you can loop over them.

Example: `for element in my_list: print(element)`

Lists are a fundamental data structure in Python.

They are used extensively in various applications.

Understanding lists is essential for Python programming.

Lists are a versatile and powerful tool in Python.

Comments

With-Context Manager

With-Context Manager

If, Elif, and Else Conditionals

Slicing

Slicing

Slicing

Dictionaries

Dictionaries

Try-Except and Exceptions

Try-Except

Try-Except is a way to handle errors in Python. It allows you to catch and handle exceptions that occur during the execution of a program. The basic syntax is as follows:

```
try:  
    # Code that might raise an exception  
except:  
    # Code to handle the exception
```

The `try` block contains the code that might raise an exception. The `except` block contains the code that will be executed if an exception occurs. This allows you to gracefully handle errors and prevent your program from crashing.

For example, if you are trying to open a file that does not exist, Python will raise a `FileNotFoundError`. You can catch this exception and handle it gracefully, such as by printing an error message or creating the file.

Try-Except is a powerful tool for writing robust Python code. It allows you to anticipate and handle potential errors, making your programs more reliable and easier to debug.

Here is a simple example of using Try-Except to handle a file not found error:

```
try:  
    with open('data.txt') as f:  
        data = f.read()  
except FileNotFoundError:  
    print('File not found. Creating a new file.')  
    with open('data.txt', 'w') as f:  
        f.write('')  
    data = f.read()
```

In this example, the `try` block attempts to open the file `data.txt` for reading. If the file does not exist, a `FileNotFoundError` is raised. The `except` block catches this error and prints a message, then creates the file and reads its contents.

Try-Except can be used to handle a wide variety of exceptions in Python. By understanding how to use Try-Except, you can write more resilient and error-tolerant code.

Continue Statement

Continue Statement

Continue Statement

Custom Functions

Custom Functions

Function Arguments

Multiple Arguments

Multiple arguments are used to pass data to a function.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Example: A function that takes multiple arguments.

Decoupling Output

Decoupling Output

Default Arguments

Default Arguments

Doc Strings

Doc strings are used to provide documentation for a function, class, or module. They are written as a triple-quoted string at the beginning of the code block.

Example: A function to calculate the area of a circle.

```
def calculate_area(radius):  
    """  
    Calculate the area of a circle given its radius.  
    """  
    area = 3.14159 * radius ** 2  
    return area
```

The doc string is enclosed in triple quotes and provides a brief description of the function's purpose and usage.

Doc strings are also used for classes and modules to provide a high-level overview of their functionality.

Example: A class representing a circle.

```
class Circle:  
    """  
    A class representing a circle.  
    """  
    def __init__(self, radius):  
        self.radius = radius
```

The doc string for the class provides a brief description of the class's purpose and usage.

Doc strings are an essential part of writing clean, maintainable code and are used by tools like Sphinx to generate documentation.

Example: A module for mathematical constants.

```
"""  
    A module containing mathematical constants.  
    """  
pi = 3.14159  
e = 2.71828
```

The doc string for the module provides a high-level overview of the module's purpose and usage.

Doc strings are a simple yet powerful way to document your code and make it easier for others to understand and use.

Decoupling Functions

Decoupling Functions

Local Modules and Import

Standard Modules

Standard Modules

Git and Github

Git and Github

Third-Party Modules

Third-Party Modules

Web Development with Streamlit

Web Development with Streamlit

Heroku Deployment

Heroku Deployment

PDF Generation with PdfKit

PDF Generation with PdfKit

Data Handling with Pandas

Sending Emails