

Variables

Lists

Functions

While-loop

Methods

Match-case

For-loop

Type-conversion

Implicit Conversion

Implicit conversion is a process where the compiler automatically converts a value of one data type to another data type without any explicit instruction from the programmer. This is often done to ensure compatibility between different data types in an expression or statement.

For example, in the following code snippet, the integer value 5 is implicitly converted to a double value 5.0 when it is added to a double value 2.5:

```
double sum = 5 + 2.5;
```

In this case, the compiler automatically converts the integer 5 to a double 5.0 before performing the addition, resulting in the sum being 7.5.

Implicit conversion is often used to simplify code and make it more readable, but it can also lead to unexpected results if not used carefully. For example, in the following code snippet, the integer value 5 is implicitly converted to a double value 5.0 when it is multiplied by a double value 2.5:

```
double product = 5 * 2.5;
```

In this case, the compiler automatically converts the integer 5 to a double 5.0 before performing the multiplication, resulting in the product being 12.5.

Implicit conversion is a useful feature of many programming languages, but it is important to understand how it works and to use it carefully to avoid unexpected results.

Implicit conversion is a process where the compiler automatically converts a value of one data type to another data type without any explicit instruction from the programmer. This is often done to ensure compatibility between different data types in an expression or statement.

For example, in the following code snippet, the integer value 5 is implicitly converted to a double value 5.0 when it is added to a double value 2.5:

List Indices

List Indices

Tuples

Enumerate

F-Strings

What

Python 3.6 introduced f-strings (formatted string literals) as a new way to format strings. They are a simple and concise way to embed expressions inside string literals.

F-strings are prefixed with `f` and contain curly braces `{}` to denote expressions to be formatted. For example:

```
name = "John"
age = 30
greeting = f"Hello, {name}! You are {age} years old."
```

The above code will output: `Hello, John! You are 30 years old.`

F-strings are useful for formatting strings in a readable and maintainable way. They are also faster than other string formatting methods like `str.format()` and `%` formatting.

Here are some examples of f-strings in use:

```
# Example 1: Simple string formatting
name = "Alice"
age = 25
print(f"Name: {name}, Age: {age}")
```

```
# Example 2: String interpolation
x = 10
y = 20
print(f"The sum of {x} and {y} is {x + y}.
```

```
# Example 3: String formatting with alignment
name = "Bob"
age = 35
print(f"Name: {name:10}, Age: {age:10}")
```

```
# Example 4: String formatting with format specifiers
name = "Charlie"
age = 40
print(f"Name: {name:10}, Age: {age:10}, Height: {1.75:.2f}m")
```

Why

F-strings are a more concise and readable way to format strings compared to other methods like `str.format()` and `%` formatting.

They are also faster than other string formatting methods, especially for large strings or strings that are formatted frequently.

F-strings are also more flexible than other methods, allowing for more complex formatting options like alignment and format specifiers.

Here are some examples of why f-strings are a better choice for string formatting:

```
# Example 1: Concise and readable
name = "Alice"
age = 25
print(f"Name: {name}, Age: {age}")
```

```
# Example 2: Faster formatting
x = 10
y = 20
print(f"The sum of {x} and {y} is {x + y}.
```

```
# Example 3: More flexible formatting
name = "Bob"
age = 35
print(f"Name: {name:10}, Age: {age:10}, Height: {1.75:.2f}m")
```

```
# Example 4: String formatting with format specifiers
name = "Charlie"
age = 40
print(f"Name: {name:10}, Age: {age:10}, Height: {1.75:.2f}m")
```

```
# Example 5: String formatting with format specifiers
name = "Charlie"
age = 40
print(f"Name: {name:10}, Age: {age:10}, Height: {1.75:.2f}m")
```

```
# Example 6: String formatting with format specifiers
name = "Charlie"
age = 40
print(f"Name: {name:10}, Age: {age:10}, Height: {1.75:.2f}m")
```

Write Text Files, Read Text Files

File Paths

List Comprehensions

List Comprehensions

Comments

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

With-Context Manager

If, Elif, and Else Conditionals

Slicing

Indexing

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Iteration

Dictionaries

Try-Except and Exceptions

Try-Except

try: `code that may raise an exception`

except: `code to be executed if an exception is raised`

else: `code to be executed if no exception is raised`

finally: `code to be executed regardless of whether an exception is raised`

try: `code that may raise an exception`

except: `code to be executed if an exception is raised`

else: `code to be executed if no exception is raised`

finally: `code to be executed regardless of whether an exception is raised`

try: `code that may raise an exception`

except: `code to be executed if an exception is raised`

else: `code to be executed if no exception is raised`

finally: `code to be executed regardless of whether an exception is raised`

Continue Statement

Continue Statement

Continue Statement

Continue Statement

Continue Statement

Custom Functions

Custom Functions

Function Arguments

Multiple Arguments

Multiple Arguments

Multiple Arguments

Multiple Arguments

Decoupling Output

Default Arguments

Default Arguments

Doc Strings

Doc Strings

Decoupling Functions

Local Modules and Import

Standard Modules

Git and Github

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Git is a distributed version control system that tracks changes in your code over time and allows you to manage your project's history.

Third-Party Modules

Third-Party Modules

GUI

Web Development with Streamlit

Heroku Deployment

Prerequisites

Getting Started

Deployment Steps

Configuration

Monitoring

FAQ

Conclusion

PDF Generation with PdfKit

PDF Generation with PdfKit

Data Handling with Pandas

Data Handling with Pandas

Sending Emails