Python input() function, type conversion, and handling errors

The `input()` function in Python is used to take input from the user. It reads a line from the input and converts it into a string. The `input()` function always returns a string, so if you want to take input as an integer or a float, you need to convert it explicitly using the `int()` or `float()` functions. (We will discuss type conversion in more detail later in this course.)

Here is an example of using the `input()` function to take input from the user:

```
name = input("Enter your name: ")
print(name)
```

In this example, the `input()` function is used to take the user's name as input. The user is prompted to enter their name, and the input is stored in the variable `name`. The `print()` function is then used to display the name.

The `input()` function can also be used without any arguments to take input without a prompt:

```
age = input()
print(age)
```

While this code will work, it is generally a good practice to provide a prompt to the user when using the `input()` function to make the program more user-friendly.

The `input()` function is commonly used in Python programs to take user input for various purposes, such as collecting user information, performing calculations based on user input, and more.

Type Conversion

As mentioned earlier, the `input()` function always returns a string, even if the user enters a number. If you want to perform numerical operations on the input, you need to convert the input to an integer or a float using the `int()` or `float()` functions.

Here is an example of converting user input to an integer:

```
age = input("Enter your age: ")
age = int(age)
print(age)
```

In this example, the user is prompted to enter their age, and the input is converted to an integer using the `int()` function. The age is then stored in the variable `age` and displayed using the `print()` function.

Similarly, you can convert user input to a float using the `float()` function:

```
height = input("Enter your height (in meters): ")
height = float(height)
print(height)
```

In this example, the user is prompted to enter their height in meters, and the input is converted to a float using the `float()` function. The height is then stored in the variable `height` and displayed using the `print()` function.

Note: If the user enters a non-numeric value when using the `int()` or `float()` functions, a `ValueError` will be raised. It is important to handle this error to prevent the program from crashing.

The `input()` function and type conversion are essential concepts in Python programming, especially when working with user input and performing numerical operations. Understanding how to use these functions effectively will help you create more interactive and dynamic programs.

exercise

Write a Python program that takes the user's name, age, and height as input and displays the information back to the user. (15 Minutes)

Example Output

```
Enter your name:
Enter your age:
Enter your height (in feet):
```

Expected Output

```
John
30
5.8
```

Error Handling

Error handling is an essential concept in programming that allows you to handle errors gracefully and prevent your program from crashing. In Python, you can use `try` and `except` blocks to handle errors.

Types of Errors

There are three main types of errors in Python:

1. **Syntax Errors**: Syntax errors occur when the code is not written according to the rules of the programming language. These errors are detected by the Python interpreter when the code is executed.
2. **Runtime Errors**: Runtime errors occur during the execution of the program. These errors are also known as exceptions and can be handled using `try` and `except` blocks.
3. **Logical Errors**: Logical errors occur when the program runs without any errors but produces incorrect results. These errors are more challenging to detect and fix as they are related to the logic of the program.

Handling Errors with `try` and `except`

You can use `try` and `except` blocks to handle runtime errors in Python. The `try` block contains the code that may raise an exception, and the `except` block handles the exception if it occurs.

Here is an example of handling a `ValueError` when converting user input to an integer:

```
try:
    age = input("Enter your age: ")
    age = int(age)
    print(age)
except ValueError:
    print("Invalid input. Please enter a valid age.")
```

In this example, the `try` block contains the code that converts user input to an integer. If the user enters a non-numeric value, a `ValueError` will be raised, and the `except` block will handle the error by displaying a message to the user.

Note: You can use multiple `except` blocks to handle different types of exceptions in Python.

The example above demonstrates how to handle a `ValueError` when converting user input to an integer. You can apply the same concept to handle other types of exceptions in your programs.

Error handling is an essential skill in programming that allows you to create robust and reliable programs. By handling errors gracefully, you can prevent your program from crashing and provide a better user experience.

Exercise

Write a Python program that takes a number as input and calculates the square of the number. Handle the `ValueError` exception if the user enters a negative number or a non-numeric value. (15 Minutes) (Hint: Use the `**` operator to calculate the square of a number.) Remember the square of a number is the number multiplied by itself. and the `input()` function always returns a string.

Example Output

```
Enter a number:
```

Expected Output

```
Enter a number: 5
Square: 25
```

Example Output

```
Enter a number: -5
Error: Please enter a positive number
```

Example Output

```
Enter a number: hello
Error: Please enter a valid number
```

Summary

In this lesson, we covered the `input()` function in Python, type conversion, and error handling. The `input()` function is used to take input from the user, and the `int()` and `float()` functions are used to convert the input to integers and floats, respectively. Error handling with `try` and `except` blocks allows you to handle exceptions gracefully and prevent your program from crashing. Understanding these concepts will help you create more interactive and robust programs in Python.