The `sorted()` function in Python is used to sort any iterable object, like lists, tuples, or strings, and returns a new sorted list. It doesn't modify the original object but creates a new one with the elements sorted in ascending order by default.

Here's how you can use it:

```python
# Sorting a list
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
sorted_list = sorted(my_list)
print(sorted_list)  # Output: [1, 1, 2, 3, 4, 5, 6, 9]

# Sorting a tuple
my_tuple = (7, 2, 5, 1, 8)
sorted_tuple = sorted(my_tuple)
print(sorted_tuple)  # Output: [1, 2, 5, 7, 8]

# Sorting a string
my_string = "hello"
sorted_string = sorted(my_string)
print(sorted_string)  # Output: ['e', 'h', 'l', 'l', 'o']
```

You can also customize the sorting order and behavior by using optional arguments. For example, you can use the `reverse` parameter to sort in descending order:

```python
# Sorting a list in descending order
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
sorted_list_desc = sorted(my_list, reverse=True)
print(sorted_list_desc)  # Output: [9, 6, 5, 4, 3, 2, 1, 1]
```

Or you can use the `key` parameter to specify a function that will be called on each element to determine its sorting key:

```python
# Sorting a list of tuples based on the second element of each tuple
my_list = [(1, 'z'), (3, 'x'), (2, 'y')]
sorted_list = sorted(my_list, key=lambda x: x[1])
print(sorted_list)  # Output: [(3, 'x'), (2, 'y'), (1, 'z')]
```

These are just a few examples of how versatile and powerful the `sorted()` function can be for sorting data in Python.

In Python, the `input()` function is used to receive input from the user through the keyboard. It allows the program to pause and wait for the user to enter some text or data, which is then read as a string. Here's a breakdown of how it works:

1. **Syntax**: The syntax of the `input()` function is straightforward:

```python
variable_name = input(prompt)
```

Here, `prompt` is an optional string argument that represents the message or prompt displayed to the user before they enter their input. It's displayed on the screen as a prompt to guide the user on what to input. If the `prompt` argument is omitted, the function simply waits for the user to input text.

2. **Input**: Once the `input()` function is executed, the program waits for the user to type something and press Enter. Whatever the user types is treated as a string and is returned by the `input()` function.

3. **Usage**: You can assign the input to a variable to store it for later use in your program. For example:

```python
name = input("Enter your name: ")
print("Hello,", name)
```

In this example, the program prompts the user to enter their name. Whatever the user types is stored in the variable `name`, which is then printed out as part of a greeting message.

4. **Type Conversion**: Keep in mind that the `input()` function always returns a string, even if the user enters a number. If you expect numerical input, you'll need to explicitly convert the input to the desired data type using functions like `int()` or `float()`.

```python
```

```
age = input("Enter your age: ")
age = int(age)  # Convert the input string to an integer
```

5. **Input from Command Line**: When you're running a Python script from the command line, you can also provide input directly in the terminal after executing the script. The `input()` function can be used in this context to read input from the command line.

```bash
python script.py
```

After executing the script, you can type input directly into the terminal, and the `input()` function will read it.