1. **Is the Python Standard Library included with PyInputPlus?**

Ans. No, the Python Standard Library is separate from PyInputPlus. The Python Standard Library is a collection of modules and packages that are included with the Python programming language by default. It provides a wide range of functionality for tasks such as file handling, networking, database access, and more.

On the other hand, PyInputPlus is a third-party library that provides additional functionalities for user input validation and handling in Python. It is not part of the Python Standard Library and needs to be installed separately.

While the Python Standard Library offers its own modules for user input, PyInputPlus provides additional features like input validation, retrying on invalid input, and timeout functionality, making it a convenient choice for handling user input in certain scenarios.

1. **Why is PyInputPlus commonly imported with import pyinputplus as pypi?**

Ans. PyInputPlus is commonly imported with the alias `pypi` (short for PyInputPlus) for brevity and convenience. Using a shorter alias reduces the amount of typing required to reference PyInputPlus functions and classes throughout the code.

By importing PyInputPlus as `import pyinputplus as pypi`, you can use the shorter alias `pypi` to access the functions and classes provided by PyInputPlus. For example, instead of writing `pyinputplus.inputStr()`, you can simply write `pypi.inputStr()`.

This practice of using aliases for module imports is common in Python and allows developers to choose shorter and more convenient names for frequently used modules, making the code easier to read and write. However, the choice of the specific alias (`pypi` in this case) is not fixed and can be customized according to personal preference or coding conventions.

1. **How do you distinguish between inputInt() and inputFloat()?**

Ans. The `inputInt()` and `inputFloat()` functions in PyInputPlus are used to obtain user input as integers and floating-point numbers, respectively. Here's how you can distinguish between the two:

1. `inputInt()`: This function is used to accept user input as an integer. It validates the input to ensure that the user enters a valid integer. If the input is not a valid integer, it prompts the user to enter again until a valid integer is provided. It returns the validated integer value.

Example usage of `inputInt()`:

**import pyinputplus as pypi**

**num = pypi.inputInt("Enter an integer: ")**

**print("You entered:", num)**

2. `inputFloat()`: This function is used to accept user input as a floating-point number. It validates the input to ensure that the user enters a valid floating-point number. If the input is not a valid float, it prompts the user to enter again until a valid float is provided. It returns the validated float value.

Example usage of `inputFloat()`:

**import pyinputplus as pypi**

**num = pypi.inputFloat("Enter a floating-point number: ")**

**print("You entered:", num)**

**4. Using PyInputPlus, how do you ensure that the user enters a whole number between 0 and 99?**

In summary, `inputInt()` is used when you specifically want to accept integer input, while `inputFloat()` is used when you want to accept floating-point number input. The functions handle the validation of the input type for you, ensuring that the user provides the expected type of input.

To ensure that the user enters a whole number between 0 and 99 using PyInputPlus, you can utilize the `inputInt()` function with additional arguments to enforce the required conditions. Here's an example:

import pyinputplus as pypi

num = pypi.inputInt("Enter a whole number between 0 and 99: ", min=0, max=99)

print("You entered:", num)

In the above code, the `inputInt()` function is used to prompt the user to enter a whole number between 0 and 99. The `min` argument is set to 0, and the `max` argument is set to 99, defining the valid range for input. If the user enters a value outside this range or a non-integer value, PyInputPlus will prompt the user to enter again until a valid input is provided.

By using the `min` and `max` arguments, you can enforce the specific range of valid values and ensure that the user enters a whole number between 0 and 99.

**5. What is transferred to the keyword arguments allowRegexes and blockRegexes?**

In PyInputPlus, the keyword arguments `allowRegexes` and `blockRegexes` are used to specify regular expressions for pattern matching. Here's what is transferred to these keyword arguments:

1. `allowRegexes`: This keyword argument expects a list of regular expression patterns. These patterns define the allowed input formats that are accepted by PyInputPlus. If an input matches any of the patterns in `allowRegexes`, it is considered valid and allowed. Input that doesn't match any of the patterns will be rejected.

2. `blockRegexes`: This keyword argument expects a list of regular expression patterns. These patterns define the blocked input formats that are not allowed by PyInputPlus. If an input matches any of the patterns in `blockRegexes`, it is considered invalid and will be rejected.

By providing regular expression patterns to these keyword arguments, you can customize the input validation behavior of PyInputPlus. It allows you to define specific patterns to accept or block certain types of input. Regular expressions provide a powerful way to handle complex input requirements based on patterns and matching rules.

For example, you can use regular expressions to allow only numeric inputs, specific formats (such as dates or email addresses), or block inputs that contain certain characters or patterns. These keyword arguments allow you to define flexible and customizable input validation rules using regular expressions.

**6. If a blank input is entered three times, what does inputStr(limit=3) do?**

When `inputStr(limit=3)` is used and a blank input is entered three times consecutively, it will raise a `TimeoutException` indicating that the limit has been reached.

By default, `inputStr()` waits indefinitely for user input. However, by specifying the `limit` argument, you can set a maximum number of times the user can enter input. If the user fails to provide input within the specified limit, PyInputPlus raises a `TimeoutException`.

In the case of `inputStr(limit=3)`, if the user enters a blank input three times consecutively without providing any other input, PyInputPlus will raise a `TimeoutException` after the third attempt. This allows you to handle scenarios where you want to enforce a specific number of input attempts and handle cases where the user doesn't provide the expected input within those attempts.

You can catch the `TimeoutException` using a try-except block to handle it gracefully and perform any desired actions or provide appropriate feedback to the user.

**7. If blank input is entered three times, what does inputStr(limit=3, default=&#39;hello&#39;) do?**

When `inputStr(limit=3, default='hello')` is used and a blank input is entered three times consecutively, it returns the default value 'hello'.

The `default` argument in `inputStr()` allows you to specify a default value that will be returned if the user enters a blank input. In this case, the default value is set to 'hello'.

So, if the user enters a blank input three times consecutively, PyInputPlus will return the default value 'hello' instead of raising a `TimeoutException`. This behavior ensures that you always get a valid input, even if the user doesn't provide any input within the specified limit.

Here's an example to illustrate this behavior:

**import pyinputplus as pypi**

**result = pypi.inputStr("Enter something: ", limit=3, default='hello')**

**print("You entered:", result)**

If the user enters a blank input three times, the output will be:

You entered: hello

By using the `default` argument, you can provide a fallback value to handle cases where the user doesn't provide input within the specified limit, ensuring that you always receive a valid input.