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

No, PyInputPlus is not part of the Python Standard Library. PyInputPlus is a separate third-party library that provides additional features and enhancements for user input handling in Python.

The Python Standard Library includes a built-in `input()` function for getting user input, but PyInputPlus extends this functionality with additional features such as input validation, automatic retries, timeout handling, and support for various input types.

To use PyInputPlus, you need to install it separately using a package manager like pip. You can install PyInputPlus using the following command:

```bash

pip install pyinputplus

```

After installing PyInputPlus, you can use it in your Python scripts to handle user input with added features beyond the basic `input()` function.

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

The choice to import PyInputPlus commonly with the alias `pypi` (`import pyinputplus as pypi`) or any other alias is a matter of personal preference or coding style. There is no strict rule specifying the alias that should be used, and developers can choose aliases that make sense and are easy to remember.

In the case of `import pyinputplus as pypi`, the alias `pypi` is used to create a shorter and more convenient reference to the PyInputPlus module. This can make the code more concise and readable, especially if you plan to use PyInputPlus frequently in your script. Using a shorter alias can save typing effort and make the code more visually appealing.

Here's an example of how you might use the `pypi` alias in code:

```python

import pyinputplus as pypi

response = pypi.inputInt(prompt="Enter an integer: ")

print("You entered:", response)

```

In this example, `pypi` is used as a reference to PyInputPlus, making the code shorter and potentially more readable.

Ultimately, the choice of alias is a matter of personal or team preference. Some developers might choose `pypi` for PyInputPlus, while others might use a different alias that they find more intuitive. The key is to select an alias that makes the code clear and readable for you and your team.

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

In PyInputPlus, `inputInt()` and `inputFloat()` are functions for getting user input specifically as integers and floats, respectively. These functions provide additional functionality for validation and handling different input scenarios. Here's how you can distinguish between `inputInt()` and `inputFloat()`:

1. \*\*inputInt():\*\*

- `inputInt()` is used to get an integer input from the user.

- It ensures that the user's input is a valid integer, and it keeps prompting until a valid integer is provided.

- It can take additional parameters such as `min`, `max`, `greaterThan`, `lessThan`, `allowRegexes`, and `blockRegexes` to define constraints on the allowed input values.

```python

import pyinputplus as pyip

# Get an integer input

user\_input = pyip.inputInt(prompt="Enter an integer: ")

print("You entered:", user\_input)

```

2. \*\*inputFloat():\*\*

- `inputFloat()` is used to get a floating-point number input from the user.

- It ensures that the user's input is a valid float, and it keeps prompting until a valid float is provided.

- Like `inputInt()`, it can take additional parameters such as `min`, `max`, `greaterThan`, `lessThan`, `allowRegexes`, and `blockRegexes` to define constraints on the allowed input values.

```python

import pyinputplus as pyip

# Get a float input

user\_input = pyip.inputFloat(prompt="Enter a float: ")

print("You entered:", user\_input)

```

In summary:

- Use `inputInt()` when you specifically need an integer input.

- Use `inputFloat()` when you specifically need a floating-point number input.

Both functions provide a convenient way to handle user input with additional validation, making it more robust and user-friendly. You can customize the behavior by using the optional parameters according to your requirements.

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

To ensure that the user enters a whole number between 0 and 99 using PyInputPlus, you can use the `inputInt()` function along with the `min` and `max` parameters to set the acceptable range. Here's an example:

```python

import pyinputplus as pyip

# Get a whole number between 0 and 99

user\_input = pyip.inputInt(prompt="Enter a whole number between 0 and 99: ", min=0, max=99)

print("You entered:", user\_input)

```

In this example:

- `prompt` is the message displayed to the user.

- `min=0` specifies that the input must be greater than or equal to 0.

- `max=99` specifies that the input must be less than or equal to 99.

If the user enters a value outside the specified range, PyInputPlus will keep prompting until a valid input within the range is provided. The `inputInt()` function will handle the validation and provide a user-friendly interface for obtaining the desired input.

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

In PyInputPlus, the `allowRegexes` and `blockRegexes` keyword arguments are used to specify regular expressions that define patterns for allowed and blocked input values, respectively. These keyword arguments allow you to customize the validation of user input based on regular expressions.

Here's a brief explanation of these keyword arguments:

1. \*\*allowRegexes:\*\*

- The `allowRegexes` keyword argument allows you to specify a list of regular expressions. If the user's input matches any of the regular expressions in this list, it is considered a valid input.

- The user is allowed to enter values that match any of the specified regular expressions.

Example:

```python

import pyinputplus as pyip

import re

# Allow only even numbers

user\_input = pyip.inputInt(prompt="Enter an even number: ", allowRegexes=[r'^[24680]$'])

print("You entered:", user\_input)

```

In this example, the regular expression `r'^[24680]$'` allows only even numbers.

2. \*\*blockRegexes:\*\*

- The `blockRegexes` keyword argument allows you to specify a list of regular expressions. If the user's input matches any of the regular expressions in this list, it is considered invalid, and the user will be prompted to enter a different value.

- The user is blocked from entering values that match any of the specified regular expressions.

Example:

```python

import pyinputplus as pyip

import re

# Block negative numbers

user\_input = pyip.inputInt(prompt="Enter a non-negative number: ", blockRegexes=[r'^-'])

print("You entered:", user\_input)

```

In this example, the regular expression `r'^-'` blocks negative numbers.

These keyword arguments provide a way to enhance the validation of user input by allowing or blocking values based on predefined patterns expressed through regular expressions. Keep in mind that regular expressions are a powerful tool for pattern matching and can be used to enforce specific constraints on user input.

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

If a blank input is entered three times and you are using `inputStr(limit=3)` in PyInputPlus, the function will raise a `pyinputplus.RetryLimitException`. This exception is raised when the user fails to provide valid input within the specified limit.

Here's an example:

```python

import pyinputplus as pyip

try:

# Allow a maximum of 3 retries for non-blank input

user\_input = pyip.inputStr(prompt="Enter a non-blank string: ", limit=3)

print("You entered:", user\_input)

except pyip.RetryLimitException:

print("Exceeded maximum number of retries. Please provide a non-blank input.")

```

In this example:

- `prompt` is the message displayed to the user.

- `limit=3` specifies that the user has a maximum of 3 retries to provide valid input.

If the user enters a blank string three times, the `RetryLimitException` will be raised, and the code within the `except` block will be executed.

You can customize the behavior further by handling the exception based on your application's requirements. This allows you to provide specific messages or take appropriate actions when the user exceeds the retry limit.

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

If blank input is entered three times and you are using `inputStr(limit=3, default='hello')` in PyInputPlus, the function will return the default value 'hello' after the third unsuccessful attempt. The `default` parameter specifies the value to be returned when the user reaches the limit of retries without providing valid input.

Here's an example:

```python

import pyinputplus as pyip

# Allow a maximum of 3 retries for non-blank input, return 'hello' as the default

user\_input = pyip.inputStr(prompt="Enter a non-blank string: ", limit=3, default='hello')

print("You entered:", user\_input)

```

In this example:

- `prompt` is the message displayed to the user.

- `limit=3` specifies that the user has a maximum of 3 retries to provide valid input.

- `default='hello'` specifies the default value to be returned if the user exceeds the retry limit.

If the user enters a blank string three times, the function will return the default value 'hello', and the code will proceed with the default value. This allows you to handle situations where the user doesn't provide the expected input within the specified limit.