1. Is the Python Standard Library included with PyInputPlus?

Ans: No, the Python Standard Library is not included with PyInputPlus. PyInputPlus is a third-party library that provides additional functionality for taking user input in Python, such as input validation and automatic conversion of input to specific data types. However, it does not include the entire Python Standard Library. The Python Standard Library is a collection of modules and packages that come bundled with the Python programming language and provide a wide range of functionality for tasks like file I/O, networking, math operations, and more.

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

Ans: Importing PyInputPlus as `pypi` (or any other chosen alias) is a common practice to make it easier and more convenient to use the library in the code. The alias `pypi` allows developers to refer to the PyInputPlus library using a shorter and more readable name throughout the code.

Using a shorter alias can save typing effort and make the code more concise. It also helps avoid potential naming conflicts with other modules or variables in the codebase. By importing PyInputPlus as `pypi`, developers can use `pypi` as a prefix to access the functions and classes provided by the library, making it clear which parts of the code are related to PyInputPlus.

Here's an example to illustrate how the alias can be used:

```python

import pyinputplus as pypi

name = pypi.inputStr("Enter your name: ")

age = pypi.inputInt("Enter your age: ")

print("Name:", name)

print("Age:", age)

```

In this example, `pypi` is used as a prefix to access the `inputStr` and `inputInt` functions provided by PyInputPlus.

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

Ans: In PyInputPlus, the `inputInt()` and `inputFloat()` functions are used to take user input and ensure that the input is a valid integer or float, respectively.

Here's the difference between `inputInt()` and `inputFloat()`:

1. `inputInt(prompt=None, default=None, limit=None, timeout=None)`: This function prompts the user to enter an integer value. If the user enters a non-integer value, it will re-prompt until a valid integer is provided. It returns the validated integer value.

Example usage:

```python

import pyinputplus as pypi

age = pypi.inputInt("Enter your age: ")

print("Age:", age)

```

2. `inputFloat(prompt=None, default=None, limit=None, timeout=None)`: This function prompts the user to enter a floating-point value. It performs a similar validation process as `inputInt()`, ensuring that the user enters a valid float value. If the input is not a valid float, it re-prompts the user until a valid float is provided. It returns the validated float value.

Example usage:

```python

import pyinputplus as pypi

weight = pypi.inputFloat("Enter your weight: ")

print("Weight:", weight)

```

In summary, `inputInt()` is used for accepting integer input, while `inputFloat()` is used for accepting floating-point input. Both functions validate the input and re-prompt if the user enters an invalid value.

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

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

```python

import pyinputplus as pypi

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

print("Number:", number)

```

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

Note that PyInputPlus will also handle other input validation aspects such as allowing the user to enter a default value, providing a timeout for input, and allowing the user to limit the number of attempts. However, in this example, we have focused on the specific requirement of ensuring a whole number between 0 and 99.

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

Ans: In PyInputPlus, the `allowRegexes` and `blockRegexes` keyword arguments are used to define regular expressions that specify patterns for allowing or blocking certain input values.

Here's a brief explanation of each keyword argument:

1. `allowRegexes`: This keyword argument accepts a list of regular expressions (as strings) or a compiled regular expression pattern. PyInputPlus will compare the user's input against these regular expressions and allow the input only if it matches at least one of the provided patterns.

Example usage:

```python

import pyinputplus as pypi

# Allowing input values that start with 'A' or 'B'

input\_value = pypi.inputStr("Enter a value: ", allowRegexes=[r'^A', r'^B'])

print("Input value:", input\_value)

```

In the above example, the user is allowed to enter values that start with either 'A' or 'B'. If the input doesn't match any of the specified regular expressions, PyInputPlus will re-prompt the user until a matching input is provided.

2. `blockRegexes`: This keyword argument accepts a list of regular expressions (as strings) or a compiled regular expression pattern. PyInputPlus will compare the user's input against these regular expressions and block the input if it matches any of the provided patterns.

Example usage:

```python

import pyinputplus as pypi

# Blocking input values that contain digits

input\_value = pypi.inputStr("Enter a value: ", blockRegexes=[r'\d'])

print("Input value:", input\_value)

```

In the above example, any input that contains digits will be blocked. If the user enters a value that matches any of the specified regular expressions in `blockRegexes`, PyInputPlus will re-prompt until a valid input is provided.

By using `allowRegexes` and `blockRegexes`, you can customize the input validation behavior in PyInputPlus and define specific patterns for allowing or blocking certain input values based on regular expressions.

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

Ans: When using `inputStr(limit=3)` in PyInputPlus and a blank input is entered three times consecutively, the function raises a `ValidationException` after the third attempt. This behavior is a result of the `limit` keyword argument.

Here's an example that demonstrates the usage of `inputStr(limit=3)`:

```python

import pyinputplus as pypi

input\_value = pypi.inputStr("Enter a value: ", limit=3)

print("Input value:", input\_value)

```

In this example, if the user enters a blank input three times in a row, PyInputPlus will raise a `ValidationException` with the error message "Blank values are not allowed." The exception will prevent the code from proceeding beyond the `inputStr()` function call, and the `print()` statement will not be executed.

The `limit` argument allows you to specify the maximum number of attempts the user has to provide a valid input. If the limit is reached without receiving valid input, PyInputPlus raises a `ValidationException` and terminates the input process.

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

Ans: When using `inputStr(limit=3, default='hello')` in PyInputPlus and a blank input is entered three times consecutively, the function returns the default value provided ('hello') after the third attempt. This behavior is a result of combining the `limit` and `default` keyword arguments.

Here's an example that demonstrates the usage of `inputStr(limit=3, default='hello')`:

```python

import pyinputplus as pypi

input\_value = pypi.inputStr("Enter a value: ", limit=3, default='hello')

print("Input value:", input\_value)

```

In this example, if the user enters a blank input three times in a row, PyInputPlus will return the default value 'hello' instead of raising a `ValidationException`. The `inputStr()` function will terminate after the third attempt, and the provided default value will be assigned to the `input\_value` variable.

So, in the case of consecutive blank inputs, `inputStr(limit=3, default='hello')` ensures that the default value is returned after reaching the maximum limit of attempts.