1. Is the Python Standard Library included with PyInputPlus?

Ans: No, the Python Standard Library is separate from PyInputPlus. PyInputPlus is a third-party library and is not included in the Python Standard Library by default.

The Python Standard Library is a collection of modules and packages that come bundled with Python and are installed automatically when you install Python itself. These modules provide a wide range of functionalities, such as file I/O, regular expressions, networking, data processing, and more. You can use the modules from the Python Standard Library without any additional installation.

On the other hand, PyInputPlus is an external library developed by the Python community, and it provides additional input validation and error handling functionalities on top of the built-in `input()` function. PyInputPlus is not part of the Python Standard Library, which means you need to install it separately if you want to use its features in your Python programs.

You can install PyInputPlus using pip, the package manager for Python:

pip install pyinputplus

Once installed, you can import and use PyInputPlus in your Python code to interact with users and validate their input according to specific rules.

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

Ans: The practice of importing PyInputPlus as `pypi` (or any other short alias) is not a common convention. Instead, the most common convention is to import it with a shorter alias like `pyip` or even just `pyinputplus`. Using an alias is done to make the code more concise and easier to read.

For example, importing PyInputPlus as `pyinputplus`:

import pyinputplus as pyip

# Now you can use pyip in your code to interact with users and validate input.

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

Using a shorter alias reduces the amount of typing required to use the library and makes the code more readable. It's especially useful when you have to call PyInputPlus functions multiple times in the same code.However, it's worth noting that the choice of alias is entirely up to the developer, and `pypi` could be used if it fits the developer's preference or coding style. As long as the alias is not misleading and doesn't conflict with other module names, it is a valid choice. The most important thing is to be consistent and clear in your code to enhance its readability and maintainability.

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

Ans: In PyInputPlus, `inputInt()` and `inputFloat()` are two different functions used to get integer and floating-point input from the user, respectively. They are part of the PyInputPlus library, which provides additional input validation and error handling features on top of Python's built-in `input()` function.

Here's how you distinguish between `inputInt()` and `inputFloat()`:

a.) `inputInt()`: This function is used to get integer input from the user. It validates the input to ensure that it can be converted into an integer. If the input is not a valid integer (e.g., contains non-numeric characters or decimal points), PyInputPlus will ask the user to enter a valid integer.

Example of using `inputInt()`:

import pyinputplus as pyip

# Get an integer input from the user

number = pyip.inputInt(prompt="Enter an integer: ")

print(f"You entered: {number}")

b.) `inputFloat()`: This function is used to get floating-point input from the user. It validates the input to ensure that it can be converted into a floating-point number. If the input is not a valid floating-point number (e.g., contains non-numeric characters or multiple decimal points), PyInputPlus will ask the user to enter a valid floating-point number.

Example of using `inputFloat()`:

import pyinputplus as pyip

# Get a floating-point input from the user

number = pyip.inputFloat(prompt="Enter a floating-point number: ")

print(f"You entered: {number}")

In both cases, PyInputPlus will keep prompting the user until a valid input is provided. It also provides additional features, such as specifying minimum and maximum values, allowing the user to enter "blank" values, and handling various edge cases, making the input validation process more robust and user-friendly.

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 along with the `min` and `max` arguments. The `min` and `max` arguments allow you to set the range of valid input values for the integer.

Here's an example of how to achieve this:

import pyinputplus as pyip

# Get an integer input from the user with minimum and maximum values specified

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

print(f"You entered: {number}")

With the `min=0` and `max=99` arguments, PyInputPlus will ensure that the input is a whole number (integer) within the range of 0 to 99 (inclusive). If the user enters a value outside this range or a non-integer value, PyInputPlus will keep prompting the user until a valid input is provided.

For example:

Enter a whole number between 0 and 99: 123

Number must be at most 99.

Enter a whole number between 0 and 99: 50.5

'50.5' is not an integer.

Enter a whole number between 0 and 99: 42

You entered: 42

In this example, if the user enters '123', PyInputPlus will reject it because it's greater than 99. If the user enters '50.5', PyInputPlus will reject it because it's not an integer. Only when the user enters a valid whole number between 0 and 99 (e.g., '42') will the program proceed to the next step.

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

Ans: In PyInputPlus, the keyword arguments `allowRegexes` and `blockRegexes` are used to transfer regular expression patterns that define which strings are allowed and blocked, respectively, during input validation.

- `allowRegexes`: This argument allows you to pass a list of regular expression patterns. When you use `allowRegexes`, PyInputPlus will only accept input strings that match at least one of the regular expression patterns provided in the list. If the user's input does not match any of the allowed patterns, PyInputPlus will reject the input and prompt the user to enter a valid input.

Example using `allowRegexes`:

import pyinputplus as pyip

# Define a list of regular expression patterns for allowed input

allowed\_patterns = [r'^[A-Za-z]+$', r'^\d{2}$']

# Get an input string that matches one of the allowed patterns

name = pyip.inputStr(prompt="Enter your name: ", allowRegexes=allowed\_patterns)

print(f"Hello, {name}!")

In this example, the user's input will be accepted only if it consists of alphabetical characters or exactly two digits. If the input does not match any of these patterns, PyInputPlus will prompt the user to enter a valid input.

- `blockRegexes`: This argument allows you to pass a list of regular expression patterns. When you use `blockRegexes`, PyInputPlus will reject input strings that match any of the regular expression patterns provided in the list. If the user's input matches any of the blocked patterns, PyInputPlus will prompt the user to enter a different input.

Example using `blockRegexes`:

import pyinputplus as pyip

# Define a list of regular expression patterns to block certain input

blocked\_patterns = [r'^\d+$', r'password']

# Get an input string that does not match any of the blocked patterns

username = pyip.inputStr(prompt="Enter a username: ", blockRegexes=blocked\_patterns)

print(f"Username '{username}' accepted.")

In this example, the user's input will be rejected if it contains only digits or if it contains the word 'password'. If the input matches any of these blocked patterns, PyInputPlus will prompt the user to enter a different input.Both `allowRegexes` and `blockRegexes` provide powerful ways to customize input validation using regular expressions, allowing you to define specific patterns that are either allowed or blocked in the user's input.

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

Ans: If a blank input is entered three times consecutively while using `inputStr(limit=3)` in PyInputPlus, it will raise a `TimeoutException`. The `limit` argument specifies the maximum number of times the user is allowed to enter invalid input (in this case, blank input) before PyInputPlus raises the `TimeoutException`.

Here's an example of how `inputStr(limit=3)` works:

import pyinputplus as pyip

# Get an input string with a limit of 3 retries for invalid input

try:

name = pyip.inputStr(prompt="Enter your name: ", limit=3)

except pyip.TimeoutException:

print("Three consecutive blank inputs. Input process terminated.")

else:

print(f"Hello, {name}!")

In this example, if the user enters a blank input three times in a row, PyInputPlus will raise a `TimeoutException`, and the message "Three consecutive blank inputs. Input process terminated." will be printed. If the user provides valid input before reaching the limit of three retries, the input process will proceed as expected, and the user's input will be processed accordingly. The `limit` argument is useful for limiting the number of retries for invalid input, preventing the input process from being stuck in a loop when the user repeatedly provides incorrect or unwanted input. It allows you to set an upper bound on the number of attempts before gracefully handling the situation and continuing with the program's flow or taking appropriate action based on the specific use case.

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

Ans: If blank input is entered three times consecutively while using `inputStr(limit=3, default='hello')` in PyInputPlus, it will return the default value `'hello'` after the third attempt. The `limit` argument specifies the maximum number of times the user is allowed to enter invalid input (in this case, blank input) before PyInputPlus returns the default value.

Here's an example of how `inputStr(limit=3, default='hello')` works:

import pyinputplus as pyip

# Get an input string with a limit of 3 retries for invalid input and a default value 'hello'

name = pyip.inputStr(prompt="Enter your name: ", limit=3, default='hello')

print(f"Hello, {name}!")

In this example, if the user enters a blank input three times in a row, PyInputPlus will return the default value `'hello'`, and the message "Hello, hello!" will be printed. If the user provides valid input before reaching the limit of three retries, the provided input will be used, and the message will be customized accordingly.

The `default` argument allows you to provide a default value that will be returned when the maximum number of retries is reached. It's a way to handle cases where the user doesn't provide the desired input and to ensure that the program continues with a predefined value even if the user's input is not received as expected.