1. Is the Python Standard Library included with PyInputPlus?

No, the Python Standard Library is not included with PyInputPlus. PyInputPlus is a third-party library that provides additional functionalities for taking input from users in Python, such as validating input, handling timeouts, and allowing for custom input handling. However, it does not include the entire Python Standard Library. The Python Standard Library is a collection of modules and packages that come with Python itself and provides a wide range of functionality for tasks like file handling, networking, data processing, and more.

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

The practice of importing PyInputPlus as `pypi` is a matter of personal preference and coding style. It allows developers to create a shorter and more convenient alias for the library to use throughout their code.

The name `pypi` is not directly related to the Python Package Index (PyPI), which is a repository for Python packages. It is simply a chosen alias for the PyInputPlus library. The selection of the alias `pypi` is arbitrary and can be changed to any other valid identifier.

By importing PyInputPlus as `pypi`, developers can use shorter and more readable code when calling functions and methods from the library. For example, instead of writing `pyinputplus.inputStr()`, they can simply write `pypi.inputStr()`. This can make the code more concise and improve readability, especially when using multiple functions from the library.

It's important to note that `pypi` is just a convention and not a requirement. Developers can choose any other valid alias when importing PyInputPlus, based on their preference and coding style.

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

In PyInputPlus, the functions `inputInt()` and `inputFloat()` are used to obtain user input as an integer or a float, respectively. Here's how you can distinguish between the two:

1. `inputInt(prompt=None, default=None, limit=None, timeout=None)`:

    - The `inputInt()` function is used to get user input specifically as an integer.
    - It displays an optional prompt to the user and waits for them to enter an integer value.
    - If the user enters a non-integer value, it will repeatedly prompt for a valid integer input until one is provided.

- It also supports additional features like providing a default value, setting input limits, and specifying a timeout for input.

2. `inputFloat(prompt=None, default=None, limit=None, timeout=None)` :

- The `inputFloat()` function is used to get user input specifically as a float.
- Similar to `inputInt()` , it displays an optional prompt and waits for the user to enter a float value.
- If the user enters a non-float value, it will keep prompting until a valid float input is given.
- It also supports features like default values, input limits, and timeouts, just like `inputInt()` .

To summarize, `inputInt()` is used when you want to ensure that the user enters an integer

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 with the `min` and `max` parameters set accordingly. Here's an example:

```
import pyinputplus as pypi

number = pypi.inputInt("Enter a whole number between 0 and 99: ", min
=0, max=99)
print("You entered:", number)
```

In the example above, `inputInt()` is used to prompt the user for input, with the provided prompt message "Enter a whole number between 0 and 99: ". The `min` parameter is set to 0 and the `max` parameter is set to 99, which restricts the accepted input range to be between 0 and 99 (inclusive).

If the user enters a value outside this range or a non-integer value, `inputInt()` will repeatedly prompt for a valid input until the criteria are met. The accepted input will be returned as an integer.

Note that PyInputPlus provides additional functionalities like handling default values, input validation, and timeouts, which can be explored in the library's documentation for more advanced use cases.

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 that define patterns for allowing or blocking certain input values.

Here's an explanation of what is transferred to these keyword arguments:

1. `allowRegexes` :
    - This keyword argument accepts a list of regular expressions.
    - When provided, PyInputPlus checks the user's input against each regular expression in the list.
    - If the user's input matches any of the regular expressions in `allowRegexes` , it is considered valid and accepted.
    - If the input does not match any of the regular expressions in `allowRegexes` , PyInputPlus will prompt for input again until a valid value is provided.
    - The regular expressions in `allowRegexes` define patterns for allowed input values.
2. `blockRegexes` :
    - This keyword argument also accepts a list of regular expressions.
    - When provided, PyInputPlus checks the user's input against each regular expression in the list.
    - If the user's input matches any of the regular expressions in `blockRegexes` , it is considered invalid and blocked.
    - If the input matches any of the regular expressions in `blockRegexes` , PyInputPlus will prompt for input again until a valid value is provided.
    - The regular expressions in `blockRegexes` define patterns for disallowed input values.

By using `allowRegexes` and `blockRegexes` , you can define specific patterns to restrict or allow certain types of input values based on regular expressions. This provides flexibility in validating and filtering user input in PyInputPlus.

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 raises a `pyinputplus.RetryLimitException` exception.

The `limit` parameter in `inputStr()` specifies the maximum number of times PyInputPlus will allow the user to retry entering a valid input. In this case, with `limit=3` , the user is given three chances to provide a non-blank input.

If the user enters a blank input for the first, second, and third attempt, the `inputStr(limit=3)` function will raise the `RetryLimitException` , indicating that the retry limit has been reached. This exception can be caught and handled in your code as needed.

Here's an example that demonstrates the behavior:

```
import pyinputplus as pypi

try:
    user_input = pypi.inputStr("Enter a non-blank input: ", limit=3)
    print("You entered:", user_input)
except pypi.RetryLimitException:
```

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

If a blank input is entered three times consecutively and `inputStr(limit=3, default='hello')` is used, the function will return the default value `'hello'` instead of raising an exception.

The `default` parameter in `inputStr()` specifies the value that PyInputPlus should return if the user enters a blank input and the retry limit is reached. In this case, with `default='hello'` and `limit=3`, if the user provides a blank input three times in a row, the function will return the default value `'hello'` instead of raising an exception.

Here's an example to illustrate this behavior:

```
import pyinputplus as pypi

user_input = pypi.inputStr("Enter a non-blank input: ", limit=3, defa
ult='hello')
print("You entered:", user_input)
```

If the user enters a blank input three times consecutively, the `inputStr()` function will return `'hello'`, and the output will be:

```
You entered: hello
```

The `default` parameter allows you to provide a fallback value when the retry limit is reached, ensuring that you always get a valid input, even if it is a default value.