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

Answer: No, PyInputPlus is not part of the Python Standard Library. It's a separate third-party library that you need to install separately if you want to use it in your Python projects. PyInputPlus provides functions to prompt the user for input with additional validation features, such as limiting input to certain types or ranges, allowing retry on invalid input, and more.

To use PyInputPlus in your Python code, you first need to install it using pip:
pip install PyInputPlus

Once installed, you can import PyInputPlus into your Python scripts and use its functions for interactive input handling.

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

Answer: Importing PyInputPlus with the alias **pypi** (**import pyinputplus as pypi**) is a common convention, but it's not mandatory. People often use this alias for brevity and convenience. Here are a few reasons why:

1. **Conciseness**: The name **pyinputplus** is quite long to type repeatedly in your code. Using a shorter alias like **pypi** reduces the amount of typing needed while still keeping the code readable.
2. **Avoiding Name Clashes**: If you're working with multiple libraries or modules, using a short, distinct alias helps avoid naming conflicts. **pypi** is unlikely to clash with other commonly used names.
3. **Consistency**: In many cases, developers use common aliases across projects to maintain consistency. If you're collaborating with others or working on multiple projects, using the same alias can make your code more consistent and easier to understand.

   However, you can choose any alias you prefer. Some people might use **p**, **pi**, or even **input** as aliases. The key is to choose something that makes sense to you and your team and doesn't conflict with other names in your codebase.

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

Answer: In PyInputPlus, **inputInt()** and **inputFloat()** are functions used to prompt the user for input, but they differ in the type of input they accept and return:

1. **inputInt(prompt=None, default=None, limit=None, timeout=None)**:
   - **inputInt()** is used to prompt the user for integer input. It keeps prompting the user until they enter a valid integer. If the input is not an integer, it asks the user to enter the value again.
   - Parameters:
     - **prompt**: (optional) A string prompt shown to the user. If not provided, a generic prompt will be used.
     - **default**: (optional) A default value to return if the user enters an empty input.

- **limit**: (optional) An integer, float, or string that determines the maximum value that the input can have. If the user inputs a value greater than the limit, they are asked to enter the value again.
- **timeout**: (optional) A float or integer representing the number of seconds to wait for user input before timing out and using the default value. If not provided, there is no timeout.

- Returns: An integer value entered by the user.

2. **inputFloat(prompt=None, default=None, limit=None, timeout=None)**:
   - **inputFloat()** is used to prompt the user for floating-point input. Similar to **inputInt()**, it keeps prompting the user until they enter a valid floating-point number. If the input is not a float, it asks the user to enter the value again.
   - Parameters and return value are the same as **inputInt()** except that it deals with floating-point numbers.

   Example usages:

```
import pyinputplus as pyip

# Prompting for an integer input
integer_input = pyip.inputInt(prompt="Enter an integer: ")

# Prompting for a float input
float_input = pyip.inputFloat(prompt="Enter a floating-point number: ")
```

   In summary, **inputInt()** is for integer input, while **inputFloat()** is for floating-point input. They both provide similar functionality for validating user input, but they handle different types of numbers.

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

   Answer: To ensure that the user enters a whole number between 0 and 99 using PyInputPlus, you can use the **inputInt()** function with the **min**, **max**, and **greaterThan** parameters. Here's how you can do it:

```
import pyinputplus as pyip

# Prompt the user for input and enforce the conditions
user_input = pyip.inputInt(prompt="Enter a whole number between 0 and 99: ",
                           min=0, max=99)
```

In this code:

- **prompt**: The message displayed to the user when asking for input.
- **min**: Specifies the minimum value allowed. In this case, it's set to 0.
- **max**: Specifies the maximum value allowed. Here, it's set to 99.

With these parameters set, PyInputPlus will keep prompting the user until they enter a whole number between 0 and 99 inclusive. If the user enters a value outside this range or

a non-integer value, PyInputPlus will ask them to enter the value again until it meets the specified conditions.

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

Answer: In PyInputPlus, the keyword arguments **allowRegexes** and **blockRegexes** are used to specify regular expressions that control which input values are allowed or blocked, respectively.

- **allowRegexes**: This argument allows you to specify a list of regular expressions. If the user input matches any of the regular expressions in this list, it is considered valid. Only inputs that match one of the regular expressions in **allowRegexes** are allowed. If **allowRegexes** is specified, any input that does not match any of the regular expressions will be rejected.
- **blockRegexes**: Conversely, this argument allows you to specify a list of regular expressions. If the user input matches any of the regular expressions in this list, it is considered invalid. Inputs that match any of the regular expressions in **blockRegexes** are blocked, and the user is asked to enter the value again.

Both **allowRegexes** and **blockRegexes** accept a list of regular expressions (as strings), allowing you to define complex patterns for accepting or rejecting input values.

For example, to only allow integers between 0 and 99, you could use **allowRegexes**:

```
import pyinputplus as pyip

user_input = pyip.inputInt(prompt="Enter a number between 0 and 99: ", allowRegexes=[r'^[0-9]{1,2}$'])
```

Here, the regular expression **^[0-9]{1,2}$** matches any string consisting of one or two digits, ensuring that the input is an integer between 0 and 99.

Conversely, if you wanted to block certain patterns, you could use **blockRegexes**. For example, to block any input containing the letter 'a', you could use:

```
import pyinputplus as pyip

user_input = pyip.inputStr(prompt="Enter a string without 'a': ", blockRegexes=[r'a'])
```

This would block any input containing the letter 'a', forcing the user to enter a string without that character.

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

Answer: If a blank input is entered three times while using **inputStr(limit=3)**, PyInputPlus will raise a **pyinputplus.RetryLimitException** after the third attempt.

Here's what happens:

1. The user is prompted to enter a string.
2. If the user enters a blank input (i.e., an empty string or just whitespace), PyInputPlus considers it invalid and asks the user to enter the value again.
3. This process continues until the user either enters a non-blank input or until the maximum number of retries is reached.
4. If the user provides a non-blank input within the specified limit, PyInputPlus returns that input.
5. If the user fails to provide a non-blank input within the specified limit (in this case, three times), PyInputPlus raises a **pyinputplus.RetryLimitException**.

   Here's an example of how it's used:

```
import pyinputplus as pyip

try:
    user_input = pyip.inputStr(prompt="Enter a non-blank string: ", limit=3)
    print("You entered:", user_input)
except pyip.RetryLimitException:
    print("You reached the maximum number of retries without providing a non-blank input.")
```

   In this example, if the user fails to provide a non-blank input after three attempts, a **RetryLimitException** will be raised, and the message "You reached the maximum number of retries without providing a non-blank input." will be printed.

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

Answer: If blank input is entered three times while using **inputStr(limit=3, default='hello')**, PyInputPlus will return the default value '**hello**' after the third attempt.

Here's how it works:

1. The user is prompted to enter a string.
2. If the user enters a blank input (i.e., an empty string or just whitespace), PyInputPlus considers it invalid and asks the user to enter the value again.
3. This process continues until the user either enters a non-blank input or until the maximum number of retries is reached.
4. If the user provides a non-blank input within the specified limit, PyInputPlus returns that input.
5. If the user fails to provide a non-blank input within the specified limit (in this case, three times), PyInputPlus returns the default value '**hello**'.

   Here's an example of how it's used:

```
import pyinputplus as pyip
```

```
user_input = pyip.inputStr(prompt="Enter a non-blank string: ", limit=3, default='hello')
print("You entered:", user_input)
```

In this example, if the user fails to provide a non-blank input after three attempts, **'hello'** will be returned as the default value.