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

No, the Python Standard Library is not included with PyInputPlus. PyInputPlus is a separate third-party library that you can install and import in your Python code using pip. However, PyInputPlus uses some modules from the Python Standard Library, such as `re` for regular expressions and `time` for timeouts.

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

PyInputPlus is commonly imported with the shorter alias `pypi` using the `as` keyword to save keystrokes and make the code more concise. The PyInputPlus library name is quite long, and it can be tedious to type it out each time it is used in the code. Therefore, using a shorter alias like `pypi` makes it easier and quicker to type.

For example, instead of writing `pyinputplus.inputNum()`, we can import PyInputPlus as `import pyinputplus as pypi` and then write `pypi.inputNum()` to use its functions. This is especially useful when we have to use PyInputPlus multiple times in a program.

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

The `inputInt()` and `inputFloat()` functions are both part of the PyInputPlus library, and they are used to prompt the user for integer and floating-point numbers, respectively. The main difference between the two functions is the type of number they accept.

The `inputInt()` function only accepts integer values, while the `inputFloat()` function accepts both integer and floating-point numbers. If the user enters a floating-point number, `inputFloat()` returns it as a `float`, while `inputInt()` would raise a `ValueError` exception.

Here is an example of how to use both functions:

```

import pyinputplus as pypi

# Prompt the user for an integer number

num1 = pypi.inputInt(prompt='Enter an integer: ')

# Prompt the user for a floating-point number

num2 = pypi.inputFloat(prompt='Enter a float: ')

print('You entered an integer:', num1)

print('You entered a float:', num2)

```

In this example, if the user enters an integer such as `42` when prompted for `num1`, `inputInt()` will accept the input and return the integer `42`. If the user enters a floating-point number such as `3.14` when prompted for `num2`, `inputFloat()` will accept the input and return the floating-point number `3.14`. If the user enters a non-numeric value for either prompt, both functions will raise a `ValueError` exception.

**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, we can use the `inputInt()` function with additional arguments to enforce the constraints. Here is an example code snippet that does this:

```python

import pyinputplus as pypi

# Prompt the user for an integer between 0 and 99

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

print('You entered:', num)

```

In this example, we use the `inputInt()` function to prompt the user for an integer number, and we specify the minimum and maximum values that the number can take using the `min` and `max` arguments. In this case, we set `min=0` and `max=99` to ensure that the user enters a number between 0 and 99 (inclusive). If the user enters a number outside this range or a non-integer value, `inputInt()` will reprompt them until a valid input is entered. The input value is then stored in the variable `num` for further use.

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

The `allowRegexes` and `blockRegexes` keyword arguments in PyInputPlus are used to specify regular expressions that should be either allowed or blocked by the input functions.

The `allowRegexes` argument is a list of regular expressions that the input function should allow. If any of the regular expressions in this list match the input, the input will be considered valid.

The `blockRegexes` argument is a list of regular expressions that the input function should block. If any of the regular expressions in this list match the input, the input will be considered invalid and the user will be reprompted for input.

For example, suppose we want to allow the user to enter a string that starts with 'A' and ends with 'B', but does not contain the string 'C'. We can use the `inputStr()` function with `allowRegexes` and `blockRegexes` arguments as follows:

```python

import pyinputplus as pypi

# Prompt the user to enter a string that starts with 'A', ends with 'B', and does not contain 'C'

string = pypi.inputStr(prompt='Enter a string: ', allowRegexes=['^A.\*B$'], blockRegexes=['C'])

print('You entered:', string)

```

In this example, we use the `allowRegexes` argument to specify a regular expression that matches any string that starts with 'A' and ends with 'B', and we use the `blockRegexes` argument to specify a regular expression that matches the string 'C'. This will ensure that the user input is valid only if it matches the allowed regular expression and does not match the blocked regular expression.

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

If a blank input is entered three times, and we use the `inputStr(limit=3)` function from PyInputPlus, it will raise a `TimeoutException`.

The `limit` parameter in the `inputStr()` function sets the maximum number of tries the user has to input valid data. If the user exceeds this limit, PyInputPlus will raise a `TimeoutException`. By default, the `limit` parameter is set to 3.

In this case, a blank input is not considered valid input because it does not match any regular expression or criteria, so PyInputPlus will keep reprompting the user for input until a valid input is entered or the limit is exceeded. If the user enters a blank input three times, the function will raise a `TimeoutException`.

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

```python

import pyinputplus as pypi

# Prompt the user to enter a non-blank string with a limit of 3 tries

string = pypi.inputStr(prompt='Enter a non-blank string: ', limit=3)

print('You entered:', string)

```

If the user enters a non-blank string within three tries, the program will continue and print the entered string. If the user enters a blank input three times, the program will raise a `TimeoutException`.

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

If a blank input is entered three times, and we use the `inputStr(limit=3, default='hello')` function from PyInputPlus, it will return the default value `'hello'`.

The `limit` parameter in the `inputStr()` function sets the maximum number of tries the user has to input valid data. If the user exceeds this limit, PyInputPlus will raise a `TimeoutException`. By default, the `limit` parameter is set to 3.

In this case, a blank input is not considered valid input because it does not match any regular expression or criteria, so PyInputPlus will keep reprompting the user for input until a valid input is entered or the limit is exceeded. If the user enters a blank input three times, the function will return the default value `'hello'`.

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

```python

import pyinputplus as pypi

# Prompt the user to enter a non-blank string with a limit of 3 tries

# If the user enters a blank input 3 times, return the default value 'hello'

string = pypi.inputStr(prompt='Enter a non-blank string: ', limit=3, default='hello')

print('You entered:', string)

```

If the user enters a non-blank string within three tries, the program will continue and print the entered string. If the user enters a blank input three times, the program will return the default value `'hello'` and print it.