**Assignment 8**

1. Is the Python Standard Library included with PyInputPlus?

No, the Python Standard Library is not included with PyInputPlus. PyInputPlus is a separate Python library that provides additional functionality for taking user input beyond what is provided by the Python Standard Library. However, PyInputPlus does rely on some modules from the Python Standard Library, such as **re** for regular expression matching, **time** for implementing timeouts, and **tkinter** for building graphical user interfaces. So, while PyInputPlus does not include the Python Standard Library itself, it does depend on it.

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

PyInputPlus is commonly imported with **import pyinputplus as pypi** to shorten the module name and make it easier to use in code.

By assigning the alias **pypi** to **pyinputplus**, you can reference the PyInputPlus module using the shorter name **pypi** instead of the longer name **pyinputplus**. This can help make your code more readable and easier to type, especially if you use PyInputPlus frequently throughout your program.

Additionally, **pypi** is a common abbreviation for PyPI (Python Package Index), which is the repository where PyInputPlus is hosted. So, using **pypi** as the alias for **pyinputplus** can help avoid confusion between the two.

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

**inputInt()** and **inputFloat()** are two functions provided by the PyInputPlus library for taking user input as integers and floating-point numbers, respectively. The key difference between the two is that **inputInt()** only accepts integer values, while **inputFloat()** accepts both integer and floating-point values.

Here's an example that demonstrates the difference between **inputInt()** and **inputFloat()**:

pythonCopy code

import pyinputplus as pyip # This will only accept integer values num1 = pyip.inputInt(prompt="Enter an integer: ") # This will accept both integer and floating-point values num2 = pyip.inputFloat(prompt="Enter a number: ")

In the example above, the **inputInt()** function will only accept integer values, meaning that if the user enters a non-integer value, PyInputPlus will raise a **pyinputplus.RetryLimitException** error. On the other hand, the **inputFloat()** function will accept both integer and floating-point values, so if the user enters an integer value, it will be automatically converted to a float.

So, if you need to accept only integer values, use **inputInt()**, and if you need to accept both integer and floating-point values, use **inputFloat()**.

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

You can use the **inputInt()** function from PyInputPlus to ensure that the user enters a whole number between 0 and 99. Here's an example:

pythonCopy code

import pyinputplus as pyip # This will only accept integer values between 0 and 99 num = pyip.inputInt(prompt="Enter a number between 0 and 99: ", min=0, max=99)

In the example above, the **inputInt()** function will only accept integer values between 0 and 99. The **min** parameter sets the minimum value that the user can enter, which is 0 in this case, and the **max** parameter sets the maximum value that the user can enter, which is 99. If the user enters a value outside this range or a non-integer value, PyInputPlus will raise a **pyinputplus.RetryLimitException** error.

By default, **inputInt()** also ensures that the input is a whole number, so you don't need to specify any additional parameters to enforce this.

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 the user input should be allowed or blocked, respectively. These arguments take a list of regular expression strings as input.

Here's an example of how to use these keyword arguments:

pythonCopy code

import pyinputplus as pyip # This will only accept input that matches the regular expression 'cat|dog' animal = pyip.inputStr(prompt="Enter 'cat' or 'dog': ", allowRegexes=['cat|dog']) # This will block input that matches the regular expression '\d+' text = pyip.inputStr(prompt="Enter text: ", blockRegexes=[r'\d+'])

In the example above, the **allowRegexes** parameter is used to specify that the input should match the regular expression **'cat|dog'**. This means that the user can only enter the strings **'cat'** or **'dog'**.

The **blockRegexes** parameter is used to specify that the input should not match the regular expression **'\d+'**, which matches any sequence of one or more digits. This means that the user cannot enter any input that contains one or more digits.

Note that you can specify multiple regular expressions in a list by separating them with the **|** character. You can also use the **r** prefix before the regular expression string to indicate that it is a raw string, which can be helpful when the regular expression contains backslashes.

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

If a blank input is entered three times in a row, the **inputStr(limit=3)** function from PyInputPlus will raise a **pyinputplus.RetryLimitException** error after the third attempt.

The **limit** parameter specifies the number of retries allowed before the function raises a **RetryLimitException** error. In this case, the **limit** parameter is set to 3, so if the user enters a blank input three times in a row, PyInputPlus will raise the **RetryLimitException** error.

Here's an example:

pythonCopy code

import pyinputplus as pyip # This will only accept non-blank input and allow up to 3 retries text = pyip.inputStr(prompt="Enter some text: ", limit=3)

In this example, the **inputStr()** function will only accept non-blank input. If the user enters a blank input, PyInputPlus will ask them to enter input again. The function will continue to ask for input until the user enters non-blank input or until the number of retries exceeds the **limit** parameter, which is 3 in this case.

If the user enters a blank input three times in a row, PyInputPlus will raise the **RetryLimitException** error.

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

If blank input is entered three times in a row, the **inputStr(limit=3, default='hello')** function from PyInputPlus will return the string **'hello'**.

The **limit** parameter specifies the number of retries allowed before the function raises a **pyinputplus.RetryLimitException** error. In this case, the **limit** parameter is set to 3, so if the user enters a blank input three times in a row, PyInputPlus will return the default value, which is **'hello'** in this case.

Here's an example:

pythonCopy code

import pyinputplus as pyip # This will only accept non-blank input and allow up to 3 retries, using 'hello' as default value text = pyip.inputStr(prompt="Enter some text: ", limit=3, default='hello')

In this example, the **inputStr()** function will only accept non-blank input. If the user enters a blank input, PyInputPlus will ask them to enter input again. The function will continue to ask for input until the user enters non-blank input or until the number of retries exceeds the **limit** parameter, which is 3 in this case.

If the user enters a blank input three times in a row, PyInputPlus will return the default value **'hello'**.

Note that you can specify any string as the default value by passing it as an argument to the **default** parameter. If you don't specify a default value, PyInputPlus will return **None** if the user enters a blank input three times in a row.