**Assignment – 8**

Q1. Is the Python Standard Library included with PyInputPlus?

Ans1. No, the Python Standard Library is not included with PyInputPlus. PyInputPlus is a third-party library that provides additional functionality for taking input from the user in Python. It simplifies common input operations and provides features like input validation, timeout handling, and default values. However, PyInputPlus does not include or depend on the entire Python Standard Library. It is designed to work alongside the standard library and other third-party libraries to enhance input functionality in Python programs.

Q2. Why is PyInputPlus commonly imported with import pyinputplus as pypi?

Ans2.   
The import statement import pyinputplus as pypi is a common convention used by developers when importing the PyInputPlus library in their Python code. It allows them to create a shorter alias or shorthand for the library's name, making it easier to reference and use throughout the code.

By importing pyinputplus as pypi, developers can use the shortened name pypi to access the functions and classes provided by the PyInputPlus library. This approach helps in reducing the amount of code needed to access PyInputPlus functionality, especially when using its functions repeatedly.

Q3. How do you distinguish between inputInt() and inputFloat()?

Ans3. In PyInputPlus, the inputInt() and inputFloat() functions are used to obtain integer and floating-point input from the user, respectively. These functions have different behaviors and are used to handle specific types of input.

Here's a breakdown of how you can distinguish between inputInt() and inputFloat():

inputInt():

inputInt() is used when you specifically want to receive an integer input from the user.

It will keep prompting the user until they enter a valid integer value.

If the user enters a non-integer value, an error message is displayed, and the user is prompted again.

If the user enters a valid integer, the function returns that integer value.

Example usage of inputInt():

**Code:**

import pyinputplus as pypi

number = pypi.inputInt("Enter an integer: ")

print("You entered:", number)

inputFloat():

inputFloat() is used when you specifically want to receive a floating-point input from the user.

It prompts the user until they enter a valid floating-point value.

If the user enters a non-numeric value or an integer, an error message is displayed, and the user is prompted again.

If the user enters a valid floating-point number, the function returns that float value.

Example usage of inputFloat():

**Code:**import pyinputplus as pypi

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

print("You entered:", number)

Q4. Using PyInputPlus, how do you ensure that the user enters a whole number between 0 and 99?

Ans4.   
To ensure that the user enters a whole number between 0 and 99 using PyInputPlus, you can use the inputInt() function in combination with the min and max parameters. Here's an example:

**Code:**

import pyinputplus as pypi

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

print("You entered:", number)

Q5. What is transferred to the keyword arguments allowRegexes and blockRegexes?

Ans5. In PyInputPlus, the keyword arguments allowRegexes and blockRegexes are used to specify regular expressions that control which inputs are allowed or blocked, respectively. Regular expressions are patterns that define sets of strings.

Here's a breakdown of what is transferred to these keyword arguments:

allowRegexes:

* The allowRegexes keyword argument accepts a list of regular expressions.

These regular expressions define patterns for the inputs that are allowed.

If an input matches any of the regular expressions in the allowRegexes list, it is considered valid.

If an input does not match any of the regular expressions in the allowRegexes list, it is considered invalid.

By using allowRegexes, you can define custom patterns to allow specific input formats.

Example usage:

import pyinputplus as pypi

# Only allow inputs that contain either 'apple' or 'orange' as a substring

fruit = pypi.inputStr("Enter a fruit: ", allowRegexes=['apple', 'orange'])

print("You entered:", fruit)

* blockRegexes:

The blockRegexes keyword argument also accepts a list of regular expressions.

These regular expressions define patterns for the inputs that are blocked.

If an input matches any of the regular expressions in the blockRegexes list, it is considered invalid.

If an input does not match any of the regular expressions in the blockRegexes list, it is considered valid.

By using blockRegexes, you can define patterns to block specific input formats.

Example usage:

import pyinputplus as pypi

# Block inputs that contain digits or special characters

name = pypi.inputStr("Enter your name: ", blockRegexes=[r'\d', r'\W'])

print("You entered:", name)

Q6. If a blank input is entered three times, what does inputStr(limit=3) do?

Ans6. If a blank input is entered three times consecutively and the **inputStr()** function is called with the **limit=3** parameter, the function will raise a TimeoutException.

Here's how inputStr(limit=3) behaves in this scenario:

The **limit** parameter specifies the maximum number of allowed retries for invalid inputs.

When the user enters a blank input (an empty string) for the first time, it is considered invalid.

The function will display an error message and prompt the user again for input.

If the user enters a blank input for the second time, it is again considered invalid, and the function will display an error message and prompt for input again.

If the user enters a blank input for the third time consecutively, the function will raise a TimeoutException.

The TimeoutException indicates that the user has exceeded the maximum number of allowed retries for input validation.

Example usage:

import pyinputplus as pypi

try:

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

print("You entered:", name)

except pypi.TimeoutException:

print("Input limit exceeded.")

Q7. If blank input is entered three times, what does inputStr(limit=3, ‘hello’) do?

Ans7. If a blank input is entered three times consecutively and the **inputStr()** function is called with the parameters **limit=3** and **default='hello'**, the function will return the default value **'hello'** instead of raising a **TimeoutException**.

Here's how **inputStr(limit=3, default='hello')** behaves in this scenario:

The **limit** parameter specifies the maximum number of allowed retries for invalid inputs.

When the user enters a blank input (an empty string) for the first time, it is considered invalid.

The function will display an error message and prompt the user again for input.

If the user enters a blank input for the second time, it is again considered invalid, and the function will display an error message and prompt for input again.

If the user enters a blank input for the third time consecutively, instead of raising a **TimeoutException**, the function will return the default value **'hello'**.

The default value is used as the output when the input limit is reached, and no valid input is provided.

Example usage:

import pyinputplus as pypi

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

print("You entered:", name)