Name :- Deepawali . B. Mhaisagar
Assignment no 8

1. Is the Python Standard Library included with PyInputPlus?
ANS
No, the Python Standard Library is not included with PyInputPlus. PyInputPlus is a separate third-party library that provides additional input functions and validation capabilities on top of the functionalities provided by the Python Standard Library's input() function.

PyInputPlus aims to simplify and enhance user input handling by offering features such as input validation, input retry, timeout functionality, and various input types (e.g., numbers, dates, emails). It is designed to be an easy-to-use and convenient alternative to the built-in input() function.

While PyInputPlus is not a part of the Python Standard Library, it can be installed separately using pip. You can install PyInputPlus by running the following command in your command prompt or terminal:

```
pip install PyInputPlus
```

After installing PyInputPlus, you can import and use it in your Python programs to leverage its extended input handling capabilities.

```
import pyinputplus as pyip

name = pyip.inputStr("Enter your name: ")

age = pyip.inputInt("Enter your age: ")

print("Name:", name)

print("Age:", age)
```

In the example above, pyinputplus is used to prompt the user for their name and age, performing input validation and ensuring the age is an integer.

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

ANS

PyInputPlus is commonly imported with the alias pypi (or any other preferred alias) to make the code shorter and more readable. The choice of the alias is a matter of personal preference, but pypi is a common choice due to its brevity and similarity to the library's name.

Using import pyinputplus as pypi allows you to refer to the PyInputPlus module by the shorter alias pypi throughout your code. This can make the code more concise and easier to read, especially when you frequently use functions from the PyInputPlus library.

Here's an example that demonstrates the usage of import pyinputplus as pypi:

python
Copy code

```
import pyinputplus as pypi

name = pypi.inputStr("Enter your name: ")

age = pypi.inputInt("Enter your age: ")

print("Name:", name)

print("Age:", age)
```

In the example above, pyinputplus is imported with the alias pypi. This allows you to use pypi as a shorthand when calling functions from the PyInputPlus library, such as pypi.inputStr() and pypi.inputInt().

The choice of the alias (pypi in this case) is flexible and can be adjusted to your preference. It's a common practice to choose an alias that is short, meaningful, and easy to remember.

Using an alias can improve code readability by making it more concise and reducing the need to type the full module name repeatedly. However, it's worth noting that the choice of alias is subjective, and you can use any valid identifier as the alias when importing modules.

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

ANS

In PyInputPlus, the inputInt() and inputFloat() functions are used to prompt the user for input and validate whether the input is an integer or a floating-point number, respectively. Here's how you can distinguish between them:

- inputInt(): This function is used when you specifically expect the user to enter an integer value. If the user enters a non-integer value or an invalid input, an error message is displayed, and the user is prompted to enter the value again.

```
import pyinputplus as pyip

num = pyip.inputInt("Enter an integer: ")
```

In the example above, the inputInt() function is used to prompt the user for an integer value. If the user enters a non-integer value, such as "abc" or "3.14", an error message is displayed, and the user is prompted again until they enter a valid integer.

- inputFloat(): This function is used when you expect the user to enter a floating-point number. Similar to inputInt(), if the user enters a non-numeric or invalid input, an error message is displayed, and the user is prompted to enter the value again.

```
import pyinputplus as pyip

num = pyip.inputFloat("Enter a floating-point number: ")
```

In the example above, the inputFloat() function is used to prompt the user for a floating-point number. If the user enters a non-numeric value or an invalid input, such as "abc" or "1.2.3", an error message is displayed, and the user is prompted again until they enter a valid floating-point number.

By using inputInt() and inputFloat() appropriately, you can differentiate between expecting an integer input and a floating-point input, ensuring that the user provides the expected type of value.

Remember to import the pyinputplus module as pyip (or any other preferred alias) using the appropriate import statement before using the inputInt() and inputFloat() functions.

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

ANS

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

```
import pyinputplus as pyip

num = pyip.inputInt("Enter a number between 0 and 99: ", min=0, max=99)
```

In the example above, the inputInt() function is used to prompt the user for a whole number between 0 and 99. The min parameter is set to 0, and the max parameter is set to 99. If the user enters a value outside of this range or a non-integer value, an error message will be displayed, and the user will be prompted again until a valid whole number within the specified range is entered.

Here's a sample interaction with the program:

```
Enter a number between 0 and 99: 105

Number must be at most 99. Try again.

Enter a number between 0 and 99: -10

Number must be at least 0. Try again.

Enter a number between 0 and 99: 42
```

In this example, entering a value outside the range triggers an error message and prompts the user to try again until a valid input is provided.

By using the min and max parameters with inputInt(), you can ensure that the user enters a whole number within the specified range.

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

ANS

In PyInputPlus, the keyword arguments allowRegexes and blockRegexes are used to specify regular expressions that allow or block certain patterns or input values. Here's how they are used:

allowRegexes: This keyword argument accepts a list of regular expression patterns. When provided, PyInputPlus checks if the user's input matches any of the specified regular expression patterns. If a match is found, the input is considered valid, and the function returns the input value.

```
import pyinputplus as pyip

response = pyip.inputStr("Enter 'yes' or 'no': ", allowRegexes=[r'yes', r'no'])
```

In the example above, the inputStr() function prompts the user to enter either "yes" or "no". The allowRegexes parameter is set to a list containing the regular expression patterns 'yes' and 'no'. If the user's input matches any of these patterns, it is considered valid, and the function returns the input value.

blockRegexes: This keyword argument accepts a list of regular expression patterns. When provided, PyInputPlus checks if the user's input matches any of the specified regular expression patterns. If a match is found, the input is considered invalid, and the function prompts the user to enter the input again.

```
import pyinputplus as pyip

response = pyip.inputNum("Enter a number greater than 10: ", blockRegexes=[r'\d+', r'^[1-9]$'])
```

In the example above, the inputNum() function prompts the user to enter a number greater than 10. The blockRegexes parameter is set to a list containing the regular expression patterns r'\d+' and r'^[1-9]$'. If the user's input matches any of these patterns (i.e., if it contains one or more digits or is a single digit from 1 to 9), it is considered invalid, and the function prompts the user to enter the input again.

By using the allowRegexes and blockRegexes keyword arguments, you can define regular expression patterns to allow or block specific input values or patterns, adding custom validation rules to the PyInputPlus functions.

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

ANS
If a blank input is entered three times in a row when using inputStr(limit=3), the function raises a pyinputplus.RetryLimitException.

The limit parameter in inputStr() specifies the maximum number of retries or attempts allowed before an exception is raised. In this case, with limit=3, the function allows the user to input a value three times before it raises the exception.

Here's an example:

```
import pyinputplus as pyip

response = pyip.inputStr("Enter a value: ", limit=3)
```

If the user enters a blank input three times consecutively, the function will raise a pyinputplus.RetryLimitException with the error message "Blank values are not allowed." This exception indicates that the user has reached the maximum retry limit and did not provide a valid input within those attempts.

To handle this exception, you can use a try-except block:

```
import pyinputplus as pyip

try:
    response = pyip.inputStr("Enter a value: ", limit=3)
except pyip.RetryLimitException:
    print("You reached the maximum number of retries. Please try again later.")
```

By catching the pyinputplus.RetryLimitException, you can handle the case when the user exceeds the retry limit.

Note: By default, if the user enters a non-blank input within the specified limit, the function will return that input value without raising an exception.

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

ANS

If a blank input is entered three times in a row when using inputStr(limit=3, default='hello'), the function returns the default value 'hello' without raising an exception.

The limit parameter in inputStr() specifies the maximum number of retries or attempts allowed before a default value is returned. In this case, with limit=3 and default='hello', the function allows the user to input a value three times. If a blank input is provided in each attempt, the function will return the default value 'hello' instead of raising an exception.

Here's an example:

python

Copy code

```
import pyinputplus as pyip



response = pyip.inputStr("Enter a value: ", limit=3, default='hello')
```

If the user enters a blank input three times consecutively, the function will return the default value 'hello' without raising a pyinputplus.RetryLimitException. This behavior allows you to provide a fallback value in case the user does not provide a valid input within the specified attempts.

By using the default parameter, you can customize the behavior when the retry limit is reached without receiving a valid input.

Note: If the user enters a non-blank input within the specified limit, that input value will be returned instead of the default value.