

User input in Python

User Input is an important programming idea: getting information from users. Whether you're making a basic calculator or a fancy data analysis tool, it's important to be able to talk to the people using your program.

But before we jump into the code, let's take a moment to understand the importance of user input: you're writing a program that calculates the area of a rectangle. Instead of hard-coding the length and width, wouldn't it be great if your program could ask the user for those values and calculate the area accordingly? That's where user input becomes a game-changer!

In Python, we have a built-in function called `input()` that allows us to capture user input. This function waits for the user to enter some information and press the Enter key. We can then store that input in a variable for further processing.

Here's a simple example. Imagine we want to greet the user and ask for their name. We can achieve this with just a few lines of code.

- `name = input("Please enter your name: ")
print("Hello, " + name + "! It's a pleasure to meet you.")`

```
name = input("Please enter your name:")  
print("Hello, " + name + "! It's a pleasure to meet you.")
```

When we run this code, the program will display the message "Please enter your name: " and wait for the user to type something in response. After the user enters their name and presses Enter, the program will print a personalized greeting like this:

Output:

```
Please enter your name:Arpita  
Hello,Arpita! It's a pleasure to meet you.
```

The program takes the user's input, stores it in the variable `name`, and then uses that variable to create the greeting message.

Let's explore another example. Imagine we want to calculate the area of a rectangle. We'll ask the user to provide the width and length, and then we'll perform the calculation. So, let's write a simple program that prompts the user for these values.

- width = input("Enter the width: ")
length= input("Enter the length: ")
area = width * length
print("The area of the rectangle is:", area)

```
width = input("Enter the width: ")  
length= input("Enter the length: ")  
area = width * length  
print("The area of the rectangle is:", area)
```

When we run this code, we will encounter an error. The reason for this is that the `input()` function returns the user's input as a string. This means that even if the user enters a number, the `input()` function will interpret it as a sequence of characters i.e a string rather than a numerical value. The variables, `width` and `length`, will hold string values, regardless of whether the user enters numerical values or any other characters. For example, if the user enters 5 for the width and 7 for the length, the variables `width` and `length` will still be of string type, not integers. In the subsequent line, we try to multiply the `width` and `length` variables together, which works fine when they are numbers. However, since the variables are treated as strings, the multiplication operation between strings is not defined, resulting in an error.

```
Enter the width: 5
Enter the length: 7
Traceback (most recent call last):

  File ~\anaconda3\envs\Finlatics\lib\site-packages\spyder_kernels\py3compat.py
in compat_exec
    exec(code, globals, locals)

  File c:\users\maitr\onedrive\desktop\finlatics\input.py:24
    area = width * length

TypeError: can't multiply sequence by non-int of type 'str'
```

To further understand this, we can check the data type of the width and length variables using the type() function. The type() function is a built-in function in Python that allows you to determine the data type of a given value or variable. It helps you understand what kind of data you are working with, whether it's a string, integer, float, list, or any other data type available in Python.

Let's modify our code to include this.

- width = input("Enter the width: ")
length = input("Enter the length : ")
print(type(width))
print(type(breadth))

```
width=input("Enter width:")
length=input("Enter Length:")

#Type Function
#understand the type of data
print(type(width))
print(type(length))
```

Output:

```
Enter width:5
Enter length:7
<class 'str'>
<class 'str'>
```

By running this updated code, we can observe that the data types of width and length are displayed as <class 'str'>. This confirms that the values are indeed treated as strings. To perform mathematical operations on these values, we need to convert them into a numerical data type. This conversion is done using a process called typecasting. Typecasting allows us to change the data type of a variable from one type to another. In our case, we need to typecast the width and length variables from strings to integers to perform multiplication.

Let's modify the code to include the necessary typecasting

- width = int(input("Enter the width: "))

```
length = int(input("Enter the length : "))
```

```
area = width * breadth
```

```
print("The area of the rectangle is:", area)
```

```
width = int(input("Enter the width: "))
length = int(input("Enter the length : "))
area = width * length
print("The area of the rectangle is:", area)
```

In the modified code, we use the int() function to typecast the user input from strings to integers. By wrapping the input() function with int(), we ensure that the values entered by the user are treated as numerical data.

Now, when we run this updated code, we can calculate the area of the rectangle correctly, as the width and length variables are now integers and can be multiplied together.

Alright, let's calculate the area of a rectangle together. Please enter the values for width and length.

[User enters the values: width = 5, length = 7]

We've entered width = 5 and length = 7. Now, let's calculate the area using these values.

```
Enter the width: 5
Enter the length : 7
The area of the rectangle is: 35
```

The area of the rectangle with a width of 5 and a breadth of 7 is 35. Fantastic!

Remember, typecasting is essential when we need to convert data from one type to another, enabling us to perform the desired operations.

In Python, typecasting is done using built-in functions like `int()`, `float()`, `str()`, and so on.

When we use typecasting on valid literals, such as numbers, it successfully converts the value to the desired data type. For example, if we have a string containing the number "5" and we use `int("5")`, it will convert the string to the integer value 5.

However, if we try to typecast a non-numeric value, such as a word like "apple", to an integer using `int("apple ")`, it will result in an error. This is because the value " apple " cannot be converted to an integer since it does not represent a valid number.

Let's demonstrate that typecasting works for valid literals but encounters an error when an invalid value is encountered with an example.

- `a = int(input('Enter first value: '))` # User enters a numerical value (10)

```
b = int(input('Enter second value: ')) # User enters a string value ("hi")
```

```
sum = a + b
```

```
print('The sum is', sum)
```

```
a = int(input('Enter first value: '))
b = int(input('Enter second value: '))
sum = a + b
print('The sum is', sum)
```

Output:

```
Enter first value: 10
Enter second value: hi
Traceback (most recent call last):

  File ~\anaconda3\envs\Finlatics\Lib\site-
  packages\spyder_kernels\py3compat.py:356 in compat_exec
    exec(code, globals, locals)

  File c:\users\maitr\onedrive\desktop\finlatics\input.py:44
    b = int(input('Enter second value: '))

ValueError: invalid literal for int() with base 10: 'hi'
```

When running this code, if the user enters the value 10 for a (a valid numerical value), and then enters the value "hi" for b (an invalid string value), an error will occur. This is because the `int()` function is unable to convert the string "hi" into an integer.

So, it's important to note that typecasting works only when the value being converted is of a compatible data type. Trying to typecast incompatible values will throw an error.

And that concludes our chapter on user input in Python! We have learned the significance of capturing user input and how it enhances our programs' interactivity.