User input in Python

Let's delve into a practical scenario where users input their name, check-in date, and check-out date. With this information, we'll craft a tailored message to enhance their experience. Imagine a hotel reservation system where guests provide these details.

Now, let's explore how we can generate a warm and personalized greeting based on their input.

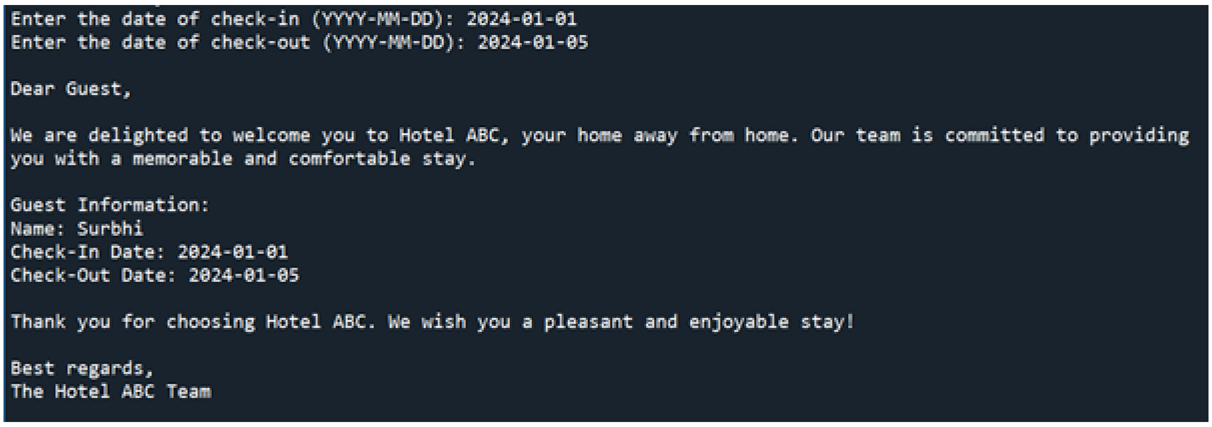
(In Python)

- guest_name = input("Please enter your name: ")

```
check_in_date = input("Enter the date of check-in (YYYY-MM-DD): ")

check_out_date = input("Enter the date of check-out (YYYY-MM-DD): ")

print(f"""

Dear Guest,

We are delighted to welcome you to Hotel ABC, your home away from home. Our team is committed to providing you with a memorable and comfortable stay.

Guest Information:

Name: {guest_name}

Check-In Date: {check_in_date}

Check-Out Date: {check_out_date}

Thank you for choosing Hotel ABC. We wish you a pleasant and enjoyable stay!

Best regards,

The Hotel ABC Team

""")
```

After running the script and entering "Surbhi" as the name, 2024-01-01 as check_in_date, and 2024-01-05 as check_out_date the output would look like this:

Output:



```
Enter the date of check-in (YYYY-MM-DD): 2024-01-01
Enter the date of check-out (YYYY-MM-DD): 2024-01-05

Dear Guest,

We are delighted to welcome you to Hotel ABC, your home away from home. Our team is committed to providing
you with a memorable and comfortable stay.

Guest Information:
Name: Surbhi
Check-In Date: 2024-01-01
Check-Out Date: 2024-01-05

Thank you for choosing Hotel ABC. We wish you a pleasant and enjoyable stay!

Best regards,
The Hotel ABC Team
```

Try inputting your own values like including the hotel name, to see a personalized message.

Building on personalized interactions, let's focus on a numerical scenario. Consider a situation where users provide two numerical values, and we use these inputs to perform a set of mathematical operations.

Imagine a dynamic calculator feature within our hotel reservation system.

But before we delve into the dynamic calculator feature within our hotel reservation system, let's take a moment to understand a fundamental concept in programming: typecasting. Typecasting is the process of converting one data type into another.

In simpler terms, it's a bit like transforming information into a shape that fits a particular job. Imagine you have a toy that only works with square blocks. If you have a round block, you need to change its shape (typecast it) into a square one so the toy can use it. In computer programming, typecasting does something similar with different kinds of information, making them suitable for specific tasks or situations.

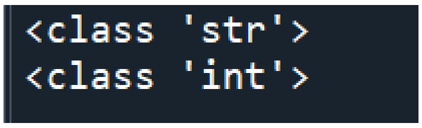
In programming, you may encounter situations where you must convert a number represented as a string into an actual numeric value or vice versa. Let's see how this is done.

(In Python)

- string_value = "123"

```
print(type(string_value))
string_value = int(string_value)
print(type(string_value))
```

output:



```
<class 'str'>
<class 'int'>
```

In this example, string_value is initially a string representing the number "123". The int() function is used to typecast it into an integer. The type() tells us about what type of data is stored in the variable.

In the first print statement, it shows that string_value is initially of type string (<class 'str'>). After the conversion using int(), the second print statement shows that string_value is now of type integer (<class 'int'>).

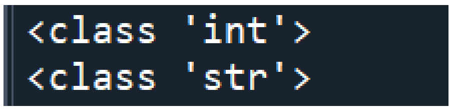
In future videos, more details about the type() function and its applications will be covered.

Now let's see another example where we are converting an Integer to string.

- value1 = 456

```
print(type(value1)
value1 = str(value1)
print(type(value1)
```

The output will be:



```
<class 'int'>
<class 'str'>
```

Here, value1 is initially an integer. The str() function is used to typecast it into a string and so the value1 is now a string.

However, not all conversions are possible. You cannot convert words or non-numeric characters directly into integers. For instance:

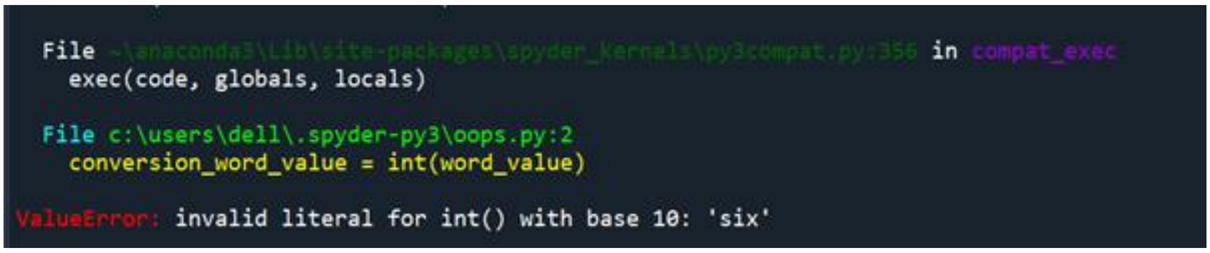
- word_value = "six"

conversion_word_value = int(word_value)

In this example, word_value is a string containing the word "six." If you try to use the int() function to convert it into an integer, it will lead to an error. Python cannot make sense of non-numeric characters when converting to an integer, and it will raise an Error.

Typecasting works well when the conversion makes logical sense, like converting a string representation of a number to an actual numeric value or vice versa.

Output:



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

File c:\users\dell\.spyder-py3\oops.py:2
  conversion_word_value = int(word_value)

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

Now, let's tie this back to our dynamic calculator feature. Users will provide two numerical values, and to perform mathematical operations, we may need to typecast these values based on the specific operation chosen. For instance:

Let's take a closer look at the script:

- num1_str = input("Enter the first numerical value: ")

num2_str = input("Enter the second numerical value: ")

# Typecasting to float for numerical operations

num1 = float(num1_str)

num2 = float(num2_str)

print(f"""

User Inputs:

First Numerical Value: {num1}

Second Numerical Value: {num2}

Results:

1) Sum: {num1 + num2}

2) Subtraction: {num1 - num2}

3) Multiplication: {num1 * num2}

4) Division: {num1 / num2}

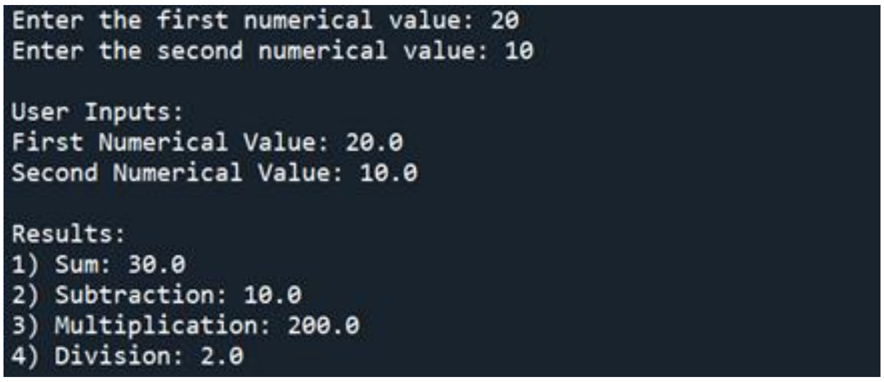
""")

The script prompts the user to enter two numerical values using the input function. The entered values are stored as strings in the variables num1_str and num2_str.

To perform numerical operations, it's necessary to convert the input strings to numerical types.

This code takes user input, typecasts it into integers, and stores these values in the variables num1 and num2. It further performs basic mathematical operations and displays the results as follows.

Output:



```
Enter the first numerical value: 20
Enter the second numerical value: 10

User Inputs:
First Numerical Value: 20.0
Second Numerical Value: 10.0

Results:
1) Sum: 30.0
2) Subtraction: 10.0
3) Multiplication: 200.0
4) Division: 2.0
```

Let’s see another example.

- my_set = {10, 20, 30, 55}

# Convert the set to a list

set_as_list = list(my_set)

# Access and print the first element of the list

first_element = set_as_list[0]

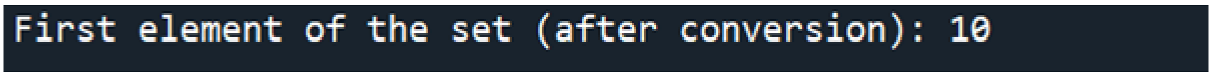
print("First element of the set (after conversion):", first_element)

A set named my_set is created with the elements 10, 20, 30, and 55. Since sets don't directly support indexing, we convert the set to a list using list(my_set) and store it in set_as_list. This conversion allows us to access elements using indices.

After the conversion, we access the first element of the list (set_as_list) using the index [0]. This is the element at position 0 in the list, corresponding to the original set's first element.

Finally, we print the first element of the set (after conversion to a list) using the print() function.

The output would be as follows, which can also be seen on your screen now:



```
First element of the set (after conversion): 10
```