Document name	:	HR.Skills.A2W5ST
Document description	:	This document contains the supporting topic: “Testing: Input/Output”.
Document version	:	V 1.0
Written by	:	Danny de Snoo
Date(s) written	:	14/03/2024, 15/03/2024

Supporting topic description:

Implementing a correct program is not easy. Programs always tend to have bugs. Input/Output Test is one of the common techniques to test the correctness of a program. The main idea is to design certain inputs, run the program with the inputs and check if the produced output satisfies expected output.

1. Make a brief research about input/output testing with some examples. Make a summary of the techniques with the examples.
2. Choose one of the problems you have solved for this week.
    a. Certainly there are some normal inputs for which your program is supposed to provide a correct output. Think about three different expected inputs and see if the output is correct.
    b. There are some inputs that are not expected. For example, if your program is supposed to receive an integer between 0 and 10, then a negative integer is not expected. Choose three different unexpected inputs and test your program. If your program crashes, then fix your program such that provides a proper message instead of crashing.
    c. Usually, users unexpectedly make mistakes when trying to give an input. For example, if they try to enter a digit between 0 and 10, mistakenly they press 6; . Check if your program is resilient for such errors.

Sources used for making this document:

1.	University of Twente, Reseach information	Document: Tre96-SCT.pdf
2.	ETSI	URL: https://portal.etsi.org/Services/Centre-for-Testing-Interoperability/ETSI-Approach/Conformance
3.	ProfessionalQA.com	URL: https://www.professionalqa.com/input-output-model

1. Make a brief research about input/output testing with some examples. Make a summary of the techniques with the examples.

Testing is an operational way of testing the correctness of a system implementation by means of experimenting with the system.
Test are run in a test environment and observations are made during the execution of test.

When testing certain output will be check to see if a valid response is given by a certain part of a system. Calculations made by the system will be verified to see if the system returns expected outputs given certain inputs.
Verification is an important part of testing Input and Output to see if there are any logic errors.

Testing and verification are complementary techniques used for analyzing and checking the correctness of the system implementation.

For testing input and output you can perform the following tests:
a. Conformance testing
b. Performance testing
c. Robustness testing
d. Reliability testing

Conformance testing:
Testing if a system implementation is following the requirements of a specification. This also includes testing if an implementation gives a correct output following the inputs as required by the specification.

Performance testing:
Testing if the performance of a system implementation is as expected.
If a system has to perform a simple calculation. It shouldn't take long to come with a certain output given a certain input.

Robustness testing:
Testing if a system implementation is robust and can keep working without crashing.

Reliability testing:
Testing if the system implementation gives the same multiple times, to see if the system implementation is reliable enough for production use.

2. Choose one of the problems you have solved for this week.

   Chosen problem – taxifares.py:

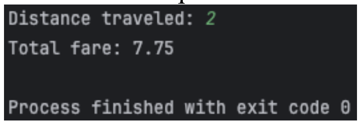
```
def calculate_fare(distance):
    # Turn the given distance (Kilometers) to Meters
    meters = distance * 1000

    # Set the base fare
    base_fare = 4.00
    # Calculate the amount_times based on a certain number of meters
    times_amount = meters / 140

    # Preform a check if the amount_times isn't a number with a decimal. If so, round
it up.
    if times_amount > round(times_amount, 0):
        times_amount = round(times_amount, 0) + 1

    # Preform the times in which we do the number of 140 meter times 0.25
    fare_meterprice = times_amount * 0.25
    # Add the result from above to the base rate
    total_fare = base_fare + fare_meterprice

    # Return the results
    return total_fare


if __name__ == '__main__':
    # Request an input from the user and then print the results from the function
    input_kilometers = input("Distance traveled: ")
    print(f"Total fare: {round(calculate_fare(float(input_kilometers)), 2)}")
```

a. Certainly there are some normal inputs for which your program is supposed to provide a correct output. Think about three different expected inputs and see if the output is correct.

Input test #1

Input: 2
Expected output: 7.75

Returned output:

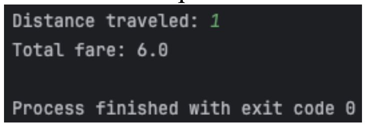


```
Distance traveled: 2
Total fare: 7.75

Process finished with exit code 0
```

Input test #2

Input: 1
Expected output: 6

Returned output:

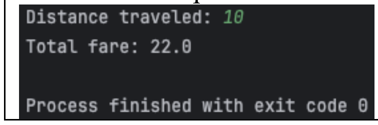


```
Distance traveled: 1
Total fare: 6.0

Process finished with exit code 0
```

Input test #3

Input: 10
Expected output: 22

Returned output:



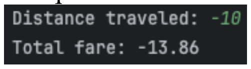
```
Distance traveled: 10
Total fare: 22.0

Process finished with exit code 0
```

b. There are some inputs that are not expected. For example, if your program is supposed to receive an integer between 0 and 10, then a negative integer is not expected. Choose three different unexpected inputs and test your program. If your program crashes, then fix your program such that provides a proper message instead of crashing.

<u>Input test #1</u>

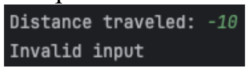
Input: -10

Output:



```
Distance traveled: -10
Total fare: -13.86
```

Output after fix:

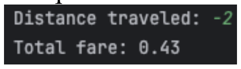


```
Distance traveled: -10
Invalid input
```

<u>Input test #2</u>

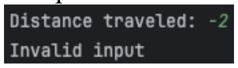
Input: -2

Output:



```
Distance traveled: -2
Total fare: 0.43
```

Output after fix:

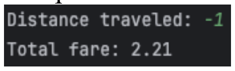


```
Distance traveled: -2
Invalid input
```

<u>Input test #3</u>

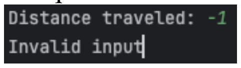
Input: -1

Output:



```
Distance traveled: -1
Total fare: 2.21
```

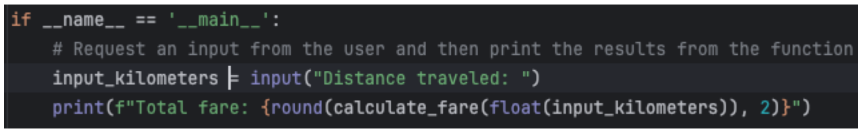
Output after fix:



```
Distance traveled: -1
Invalid input
```

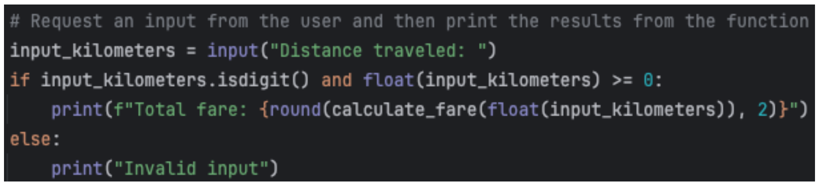
<u>Program fix:</u>

Before:



```python
if __name__ == '__main__':
    # Request an input from the user and then print the results from the function
    input_kilometers = input("Distance traveled: ")
    print(f"Total fare: {round(calculate_fare(float(input_kilometers)), 2)}")
```

After:

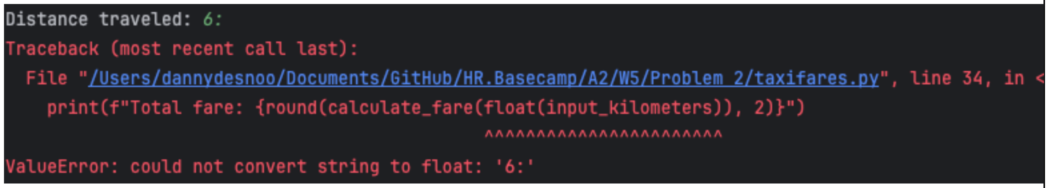


```python
# Request an input from the user and then print the results from the function
input_kilometers = input("Distance traveled: ")
if input_kilometers.isdigit() and float(input_kilometers) >= 0:
    print(f"Total fare: {round(calculate_fare(float(input_kilometers)), 2)}")
else:
    print("Invalid input")
```

c. Usually, users unexpectedly make mistakes when trying to give an input. For example, if they try to enter a digit between 0 and 10, mistakenly they press 6; . Check if your program is resilient for such errors.

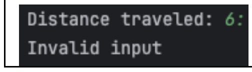
Thanks to the fix that has been performed above, the program can now handle such errors

Before fix:



```
Distance traveled: 6:
Traceback (most recent call last):
  File "/Users/dannydesnoo/Documents/GitHub/HR.Basecamp/A2/W5/Problem 2/taxifares.py", line 34, in <
    print(f"Total fare: {round(calculate_fare(float(input_kilometers)), 2)}")
                                              ^^^^^^^^^^^^^^^^^^^^^^^
ValueError: could not convert string to float: '6:'
```

After fix:



```
Distance traveled: 6:
Invalid input
```