Solutions – Operations using Data Types and Operators

Exercise 1

Problem 1:

```
message = "He said "There are some who call me...Tim""
```

The above error can be fixed in one of two ways:

Solution 1:

```
message = "He said \"There are some who call me...Tim\""
```

Here we have used the backslashes just in front of each of the double quotes in the string. The backslash escapes the special meaning of those characters, meaning that they are treated as standard quotes and not as string delimiters.

Solution 2:

```
message = 'He said "There are some who call me...Tim"'
```

Here we have just switched the outer quotes (the string delimiters) to single quotes. It is perfectly fine then to have double quotes within the string.

Problem 2:

```
message = 'He's not the messiah-he's a very naughty boy!'
```

Again, this error can be fixed in one of two ways:

Solution 1:

```
message = 'He\'s not the messiah-he\'s a very naughty boy!'
```

Here we have used the backslashes just in front of each of the single quotes/apostrophes in the string. The backslash escapes the special meaning of those characters, meaning that they are treated as apostrophes and not as string delimiters.

Solution 2:

```
message = "He's not the messiah-he's a very naughty boy!"
```

Here we have just switched the outer quotes (the string delimiters) to double quotes. It is perfectly fine then to have single quotes within the string.

Please note: It is recommended that you avoid using the backslash where possible, as it does hurt readability. If you can solve the string delimiter issue by just switching the outer quotes, then that is the best plan. If both apostrophes and speech marks reside inside your string though, then it will not be possible just to switch the outer quotes and you will need to use the backslash.

Exercise 2

You were tasked with determining a value for each of the following 4 data types (int, float, string, boolean) that is considered both truthy (evaluates to True) and falsy (evaluates to False).

The following four statements evaluate to False:

```
bool(False)
```

```
False
```

```
bool(0.00)
```

```
False
```

```
bool(0)
```

```
False
```

```
bool("")
```

```
False
```

The boolean value of False is the only obvious one here, which evaluates to False.

A numeric value equal to zero (so the integer 0 and the floating point value 0.00) evaluate to False.

An empty string is the only string that evaluates to False.

The following four statements evaluate to True:

```
bool(True)
```

```
True
```

```
bool(0.0001)
```

```
True
```

```
bool(-1)
```

```
True
```

```
bool(" ")
```

```
True
```

The boolean value True is the obvious one here, evaluating to True.

Any numeric value that is not zero evaluates to True.

Any non-empty string evaluates to True. This string only has a space in it, but that is a valid character and means that the string is not empty.

Exercise 3

You were tasked with modifying the following code snippet so that it does not lead to a runtime error:

```
savings = 1050

print("You have €" + savings + " in the bank!")
```

Solution 1:

You can use the str() conversion function to convert the integer value of 1050 to a string. Concatenation can then be performed:

```
savings = 1050

print("You have €" + str(savings) + " in the bank!")
```

Solution 2:

You can individually pass an arbitrary number of comma-separated arguments to the print function. These arguments do not need to be of the same type:

```
savings = 1050

print("You have €", savings, " in the bank!")
```

You may have noticed that when you pass multiple arguments to the print function, they are printed separated by a single space by default. You can change the default behaviour by setting the sep parameter (short for separator) to an empty string as follows:

```
savings = 1050

print("You have €", savings, " in the bank!", sep="")
```

Exercise 4

You were tasked with fixing the following code snippet, as it leads to a runtime error being raised:

```
balance = 350.00
deposit = input("Please enter the amount that you wish to deposit")

balance += deposit
```

The information on the error is shown below:

```
TypeError                                 Traceback (most recent call last)
<ipython-input-3-095fb042cd7e> in <module>
      2 deposit = input("Please enter the amount that you wish to deposit")
      3
----> 4 balance += deposit

TypeError: unsupported operand type(s) for +=: 'float' and 'str'
```

To fix the code snippet, the float() conversion function should be used (you should side with the float() function over the int() function as we wish to allow the depositing of Cents as well as Euros).

You can use the function in a couple of different places in the code. In the first example below, the user input is stored in the variable called deposit, which is currently of type string. The next line of code converts the data type of deposit to be of type float, with the result stored back in the float variable. The addition can then be carried out on the two floats:

```
balance = 350.00
deposit = input("Please enter the amount that you wish to deposit")
deposit = float(deposit)

balance += deposit
```

A shorter and neater way of writing this would be to use the float() conversion function on the line of code where we take the user input. In the following example, the string returned from the input function is immediately converted to a float before being stored in the deposit variable. Again, the addition can now be carried out without any issues:

```
balance = 350.00
deposit = float(input("Please enter the amount that you wish to deposit"))

balance += deposit
```

Exercise 5

You were tasked with amending the following code snippet so that a whole number with no decimal places is stored in the temp variable:

```python
temp = 19.9

print('The current temp is', temp, 'degrees')
```

```
The current temp is 19.9 degrees
```

To amend this code and store just the whole number of 19 in the variable temp, the int() conversion function should be used.

```python
temp = int(19.9)

print('The current temp is', temp, 'degrees')
```

```
The current temp is 19 degrees
```

Please note that if you attempted to convert the temp variable to an int down in the print function, this will only convert it for the purposes of printing it out. So the 19.9 is converted to an int (19) for presentation purposes. No assignment has been made though, so the temp variable will still hold 19.9 after the print statement has executed:

```python
temp = 19.9

print('The current temp is', int(temp), 'degrees')

print(temp)
```

```
The current temp is 19 degrees
19.9
```

Exercise 6

You were tasked with creating a list of watches. The original list can be created as follows:

watches = ['Compass Watch', 'GPS Watch', 'Stun Watch', 'Nerve Gas Watch', 'Inflatable Dingy Watch', 'Android Watch', 'Lunchtime Alarm Watch(1:30pm)']

The additional element can be added to the end of the list by using the append method:

```
watches.append('Jingly Jangly Watch(for the evening ambiance)')
```

Removing the 'Android Watch' can be done as follows:

```
watches.remove('Android Watch')
```

The following line of code inserts the 'Apple Watch' at the start of the list:

```
watches.insert(0, 'Apple Watch')
```

The following line of code uses indexing to grab the 'Inflatable Dingy Watch' from the list and stores it in the variable 'current_watch':

```
current_watch = watches[5]
```

Below is the rations list for the next part of the exercise:

```
rations = ['Mars Bar', 'Powdered Soup', 'Dry Cereal', 'Trifle']
```

We can sort the list items alphabetically using the following line of code:

```
rations.sort()
```

You can pull the last item out of the list in either of the following ways:

1. You can pass in the index number of the last element manually

   ```
   lunch = rations[3]
   ```

2. You can pass in -1 as the index number. This is more dynamic, as it will always grab the last element from the list even if the list changes

   ```
   lunch = rations[-1]
   ```

We can grab the first three elements from the rations list using the following slicing syntax:

```
breakfast_of_champions = rations[:3]
```

No start has been specified, so we start at the beginning of the list. A stop of 3 has been specified, but the stop is exclusive. This means that we will retrieve elements from the start of the list, up to but not including the element at index 3.

Exercise 7

You were tasked with writing two expressions that successfully check whether:

a) A gamer has unlocked a new mission
b) A gamer has unlocked a new weapon

You were given the following sample data to work with:

```
hurdle_time = 9.95
```

```
height_from_road = 10.75
```

The following criteria was given for the silly side games:

a) Hyper Hurdles should be completed in 10 seconds or under
b) The Car in We Have Lift-off should rise off the ground by more than 11 centimetres

If both criteria are met, a new mission is unlocked. The following statement will check if both criteria has been met:

```
unlock_mission = hurdle_time <= 10 and height_from_road > 11
```

If either criteria are met, a new weapon is unlocked. The following statement will check if either of the criteria has been met:

```
unlock_weapon = hurdle_time <= 10 or height_from_road > 11
```

End of Exercises