1. Dealing with exceptions

Fill `____` parts of the implementation below. `sum_of_list` function takes a list as argument and calculates the sum of values in the list. If some element in the list can not be converted to a numeric value, it should be ignored from the sum.

In [24]:
```
def sum_of_list(values):
    result = 0
    for val in values:
        try:
            numeric_val = float(val)
        except (ValueError, TypeError) as e:
            numeric_val = 0
        result += numeric_val

    return result
```

In [25]:
```
list1 = [1, 2, 3]
list2 = ['1', 2.5, '3.0']
list3 = ['', '1']
list4 = []
list5 = ['John', 'Doe', 'was', 'here']
nasty_list = [KeyError(), [], dict()]

assert sum_of_list(list1) == 6
assert sum_of_list(list2) == 6.5
assert sum_of_list(list3) == 1
assert sum_of_list(list4) == 0
assert sum_of_list(list5) == 0
assert sum_of_list(nasty_list) == 0
```

2. Using custom exceptions

Implement `verify_short_string` function which takes a single string as argument. If the length of the input string is more than ten characters, the function should raise `TooLongString` exception (note: you have to create `TooLongString` yourself). The function does not have to return anything.

In [18]:
```
# Your implementation here

def verify_short_string(a):
    if len(a) > 10:
        raise TooLongString(a)


class TooLongString(Exception):
    def __init__(self, string):
        self.string = string
    def __str__(self):
        return f'{self.string} is more than 10 characters.'
```

In [19]:
```
# These should not raise
verify_short_string('short')
verify_short_string('10   chars')

# This should raise
try:
    verify_short_string('this is long')
except TooLongString as e:
    # This is ok
    pass
else:
    # This means that there was no exception
    assert False
```