test_extract_sizes.py

Unit tests for extract sizes

```
import json
import unittest
from extract_sizes import (get_sizes, tag_text, text_to_groupings)
```

```
class TestExtractSizes(unittest.TestCase):
```

```
    Test cases for the extract sizes
    Includes testing None cases and valid PDFs
```

```
    def test_font_doc_none(self):
```

```
        Asserts that when None is passed into get_sizes,
        no error happens and None is returned
```

assert when no document is passed, it returns an empty lst

```
        unique_fonts = get_sizes([])
        self.assertEqual(unique_fonts, [])
```

```
    def test_dict_fonts_none(self):
```

```
        Asserts that when None is passed into tag_text,
        no error happens and None is returned
```

assert when no document is passed, it returns an empty dict

```
        dict_none = tag_text([], [])
        self.assertEqual(dict_none, [])
```

```
    def test_font_size_1(self):
```

```
        Tests the unique font size is as expected for Test1
```

```
        filename = "code/data/Test_1.json"
        with open(filename, encoding='utf-8') as file_pointer:
            doc = json.load(file_pointer)
            actual_fonts = get_sizes(doc)

            expected_fonts = [10, 11, 12, 14]
            self.assertEqual(expected_fonts, actual_fonts)
```

```
    def test_text_to_groupings_1(self):
```

```
        Tests the groupings given a file
```

```
        filename = "code/data/Test_1.json"

        with open(filename, encoding='utf-8') as file_pointer:
            doc = json.load(file_pointer)
            actual_dict = text_to_groupings(doc)
```

check the text

```
            page0 = {'Header': 'Possible title',
                     'Paragraph': '',
                     'slide': 0}
            page1 = {'Header': 'Possible heading',
                     'Paragraph': 'Possible subheading Possible paragraph',
                     'slide': 1}
            page2 = {'Header': 'Heading Another Heading',
                     'Paragraph': 'Paragraph',
                     'slide': 2}
            self.assertEqual(page0, actual_dict[0], f'Expected Page 0 to be {page0}')
            self.assertEqual(page1, actual_dict[1], f'Expected Page 1 to be {page1}')
            self.assertEqual(page2, actual_dict[2], f'Expected Page 2 to be {page2}')
            self.assertEqual([page0, page1, page2], actual_dict)
```

Tests the unique font size is as expected for Test2

```
    def test_font_size_2(self):


        filename = "code/data/Test_2.json"
        with open(filename, encoding='utf-8') as file_pointer:
            doc = json.load(file_pointer)
            actual_fonts = get_sizes(doc)

            expected_fonts = [12, 28, 44, 60]
            self.assertEqual(expected_fonts, actual_fonts)
```

Tests the groupings given a file

```
    def test_text_to_groupings_2(self):


        filename = "code/data/Test_2.json"
        with open(filename, encoding='utf-8') as file_pointer:
            doc = json.load(file_pointer)

            actual_dict = text_to_groupings(doc)
```

check the text

```
            page0 = {'Header': 'Test 1 : Computer Science',
                     'Paragraph': '',
                     'slide': 0}
            page1 = {'Header': 'Large Heading',
                     'Paragraph': 'Medium text Small text',
                     'slide': 1}
            page2 = {'Header': 'Large Heading 2',
                     'Paragraph': 'Medium text 2 Small text 2',
                     'slide': 2}
            self.assertEqual(page0, actual_dict[0], f'Expected Page 0 to be {page0}')
            self.assertEqual(page1, actual_dict[1], f'Expected Page 1 to be {page1}')
            self.assertEqual(page2, actual_dict[2], f'Expected Page 2 to be {page2}')
            self.assertEqual([page0, page1, page2], actual_dict)


if __name__ == '__main__':
    unittest.main()
```