extract_sizes.py

File completing step 2: given a pdf document return a dictionaryof headers and paragraphs

```
import re
import fitz
```

```
def extract_words(file: str) -> dict:
```

```
    Given a filename,
    opens the PDF and extracts words & metadata from each slide

    :param file: String representing file path
    :type: string
    :rtype: dict
    :return: dict, Contains extracted metadata & words from all slides
```

```
    document = fitz.open(file, filetype="pdf")
    doc_data = {}
    doc_data["meta_data"] = document.metadata
    doc_data["data"] = []
    for index, page in enumerate(document):
        page_data = {}
        page_data["slide"] = index+1
        page_data["blocks"] = []
        blocks = page.get_text("dict")["blocks"]
```

iterate through the text blocks

```
        for block in blocks:
```

block contains text

```
            if block['type'] == 0:
```

iterate through the text lines

```
                for line in block["lines"]:
```

iterate through the text spans

```
                    for span in line["spans"]:
                        page_data["blocks"].append({
                            "text": re.sub(r"\W{3,}", " ", span["text"]),
                            "size": span["size"]
                        })
        doc_data["data"].append(page_data)

    return doc_data
```

```
def get_sizes(doc: dict) -> list:
```

```
    Helper function to get unique sizes within a PDF

    :param doc: The list of blocks within a PDF
    :type: list
    :rtype: list
    :return: a list of unique font sizes
```

ensuring object is not None/empty

```
    if not doc:
        return []

    unique_fonts = set()
```

for each page in our document

```
    for page in doc['data']:
```

get the individual text blocks

```
            for block in page['blocks']:
```

can also get font and color

```
                unique_fonts.add(round(block['size']))
```

sort the fonts for later filtering

```
        sorted_fonts = sorted(list(unique_fonts))
        return sorted_fonts
```

```
    def tag_text(unique_fonts: list, doc: dict) -> list:
        """
        Categorizes each text into L, M, or S.

        :param unique_fonts: a list of unique fonts in the powerpoint
        :type unique_fonts: list
        :param doc: a list of blocks per each document page
        :type doc: dict
        :rtype: list
        :return: a list of dicts categorizing texts into respective categories
        """
```

check that both are not None, or empty

```
        if not doc or not unique_fonts:
            return []
```

The Header will be the top 2 font sizes top font size is Title, second would be header

```
        header_lim = unique_fonts[-2]
        all_pages = []

        for page in doc['data']:
            text_dict = {'Header': "", 'Paragraph': "", 'slide': page['slide']}
```

get the individual text blocks

```
            for block in page['blocks']:
```

if the text size is smaller than header or title

```
                if block['size'] < header_lim:
                    text_dict['Paragraph'] += block['text'] + " "
                else:
                    text_dict['Header'] += block['text'] + " "
```

trim any extra whitespace

```
            text_dict['Paragraph'] = text_dict['Paragraph'].strip()
            text_dict['Header'] = text_dict['Header'].strip()
            all_pages.append(text_dict)
        return all_pages
```

```
    def text_to_groupings(doc: dict) -> list:
        """
        Given a pdf document,
        Returns a dictionary of Headers, Paragraphs, and page number

        :param doc: a PDF document containing only words
        :type: dict
        :rtype: list
        :return: dict categorizing each text into its respective category
        """
```

```
        font_count = get_sizes(doc)
        lst_fonts = tag_text(font_count, doc)
        return lst_fonts
```