# HW5-RyanSponzilli

October 6, 2024

## 1 ASTR 310 Homework 5

#### 1.0.1 1. Parsing a natural language string

Write a Python function that takes a string representing the English name of a nonnegative integer as an argument and returns the corresponding integer. Your function should work for numbers up to 999. If the string provided does not correspond to a valid English number, your function should return `None`. Some examples of arguments and the expected results include:

| Input | Result |
|---|---|
| "nine" | 9 |
| "seventy six" | 76 |
| "one hundred forty" | 140 |
| "four hundred thirteen" | 413 |
| "eight hundred sixty seven" | 867 |
| "four hundred five hundred thirty six" | `None` |
| "two hundred eight thousand four" | `None` |
| "twelve two hundred one" | `None` |
| "five hundred seven twenty" | `None` |
| "eleventy nine" | `None` |

You should demonstrate that your function works on these examples. [20 pts]

[120]:
```python
def parse_number(num: str):

    singles = {
        "one" : "1",
        "two" : "2",
        "three" : "3",
        "four" : "4",
        "five" : "5",
        "six" : "6",
        "seven" : "7",
        "eight" : "8",
        "nine" : "9",
    }
    teens = {
        "ten" : "10",
        "eleven" : "11",
        "twelve" : "12",
        "thirteen" : "13",
        "fourteen" : "14",
        "fifteen": "15",
        "sixteen" : "16",
        "seventeen" : "17",
        "eighteen" : "18",
        "nineteen" : "19"
    }

    tens = {
        "twenty" : "2",
        "thirty" : "3",
        "forty" : "4",
        "fifty" : "5",
        "sixty" : "6",
        "seventy" : "7",
        "eighty" : "8",
        "ninety" : "9",
    }

    if (num.count(" ") == 0):
        if num in singles:
            return singles[num]
        elif num in teens:
            return teens[num]
        elif num in tens:
            return tens[num] + "0"
        else:
            return None
    else:
        elements = num.split(" ")
        parsed_num = ""
        # 100s
        if (elements[0] in singles) & (elements[1] == "hundred"):
            parsed_num += singles[elements[0]]
            if len(elements) == 2:
                parsed_num += "00"
            elements.remove(elements[1])
            elements.remove(elements[0])

        # 10s
        if (len(elements) > 0):
            if (elements[0] in tens):
                if (len(elements) > 1):
                    if (elements[1] in (singles)):
                        parsed_num += tens[elements[0]]
                        parsed_num += singles[elements[1]]
                        elements.remove(elements[1])
                        elements.remove(elements[0])
                else:
                    parsed_num += tens[elements[0]]
                    parsed_num += "0"
                    elements.remove(elements[0])
            elif (elements[0] in singles):
                parsed_num += "0"
                parsed_num += singles[elements[0]]
                elements.remove(elements[0])
            elif (elements[0] in teens):
                parsed_num += teens[elements[0]]
                elements.remove(elements[0])

        if (len(elements) > 0):
            return None

        return parsed_num
```

[121]:
```
print(parse_number("nine"))
print(parse_number("seventy six"))
print(parse_number("one hundred forty"))
print(parse_number("four hundred thirteen"))
print(parse_number("eight hundred sixty seven"))
print(parse_number("four hundred five hundred thirty six"))
print(parse_number("two hundred eight thousand four"))
print(parse_number("twelve two hundred one"))
print(parse_number("five hundred seven twenty"))
print(parse_number("eleventy nine"))
```

```
9
76
140
413
867
None
None
None
None
None
```

### 1.0.2 2. Robust parsing of a text file

**a)** Write functions to read and write files containing tables of variable names and values. You should implement the following functions:

- `file_read(file_name)` — Read the file named `file_name` and return a dictionary containing variable names as keys and their values as the corresponding values. If an I/O error or syntax error occurs (see below), print the offending line, return `None` and ensure that the file (if opened) is closed.
- `file_write(file_name, var_dict, overwrite=False)` — Write the variable dictionary `var_dict` to the file named `file_name`. The optional argument overwrite should control the behavior of the function in the case that the output file already exists. If the variable dictionary does not follow the rules described below, or an I/O error occurs, or overwrite == False and the file already exists, print a helpful error message and return `None`. If the file was successfully written, return `True`.

The files read from/written to should have lines with this format:

`variable = value`

The variable name must start with a letter or underscore (`_`) and can contain letters, digits, or underscores, but no other characters. An arbitrary but nonzero amount of whitespace (spaces or tabs) can come before or after the variable name, the = sign, and the value. The value should be an integer or floating-point number; when reading or writing the value, its type should be preserved (e.g. "6" should be read and written as an int, and "$-$3.2e4" should be read and written as a float). If any of these rules is violated, or a line does not have this format, the file has a syntax error and should be handled as described above.

[8 pts]

[221]:
```python
from os.path import exists

def file_read(file_name):
    try:
        f = open(file_name, 'r')
        lines = f.readlines()
        f.close()
        var_dict = {}
        for line in lines:
            try:
                key, value = line.split("=")
                if follows_formatting_rules_key(key.strip()):
                    if ("." in value) | ("e" in value):
                        var_dict[key.strip()] = float(value.strip())
                    else:
                        var_dict[key.strip()] = int(value.strip())
                else:
                    print(line)
                    return None
            except:
                print(line)
                return None
        return var_dict
    except:
        print("An error occurred")
        return None

def file_write(file_name, var_dict, overwrite = False):
    try:
        if (overwrite == False) & (exists(file_name)):
            print("File already exists")
            return None
        f = open(file_name, 'w')
        for key, value in var_dict.items():
            if follows_formatting_rules_value(value) &␣
 ↪follows_formatting_rules_key(key):
                f.write(f"{key} = {value}\n")
            else:
                print(f"{key} = {value} does not follow formatting rules!")
                return None
        f.close()
        return True
    except:
        print("An error occurred")
        return None

def follows_formatting_rules_value(value):
    if isinstance(value, (float, int)):
        return True
    return False

def follows_formatting_rules_key(key):
    for char in str(key):
        if not (char.isalnum() | (char == '_') | (char == ' ')):
            return False
    if not (str(key)[0].isalpha()) | (str(key)[0] == '_'):
        return False
    return True
```

**b)** Use your functions to read the file "vardict.txt" from the class web page. Print the contents of the variable dictionary you have read. [2 pts]

```
[192]: vardict = file_read("vardict.txt")
       vardict
```

```
[192]: {'electoralVotes': 538,
        'days_in_year': 365.25,
        '__wombats7': 32,
        'RegulationFidgetSpinner': -72.8,
        'marigolds_around_alumni_fountain': 423129,
        'BaselineKaijuDiet2020': 8341000000.0,
        'digits_in_eleven_': 2}
```

**c)** Add the settings - `bears_with_6arms = 13` - `chessRookHeight = 45.327` - `__neighborHondaYear = 2009` - `fortysix__ = 46.0`

to the dictionary and write them to a new file named "vardict2.txt". [5 pts]

[193]:
```
vardict["bears_with_6arms"] = 13
vardict["chessRookHeight"] = 45.327
vardict["__neightborHondaYear"] = 2009
vardict["fortysix__"] = 46.0

file_write("vardict2.txt", vardict)
```

[193]:
```
True
```

**d)** Confirm that you wrote a valid file by reading "vardict2.txt" and printing the contents of the variable dictionary that results. [5 pts]

[194]:
```
vardict2 = file_read("vardict2.txt")
vardict2
```

[194]:
```
{'electoralVotes': 538,
 'days_in_year': 365.25,
 '__wombats7': 32,
 'RegulationFidgetSpinner': -72.8,
 'marigolds_around_alumni_fountain': 423129,
 'BaselineKaijuDiet2020': 8341000000.0,
 'digits_in_eleven_': 2,
 'bears_with_6arms': 13,
 'chessRookHeight': 45.327,
 '__neightborHondaYear': 2009,
 'fortysix__': 46.0}
```

**e)** Check your error handling by making a corrupted version of "vardict.txt" and rerunning part (b). For example, you could create a line in the file that doesn't follow the "variable = value" syntax, or you could break the syntax rules for variable names, or you could change a value so that it is not a valid int or float. Make sure you print the offending line so that the reader can see what caused the problem. [5 pts]

[195]:
```
vardict = file_read("vardict.txt")
vardict
```

```
digits_in_eleven_ = two
```

```
[222]: vardict = file_read("vardict.txt")
       vardict
```

```
9digits_in_eleven_ = 2
```

```
[223]: vardict = file_read("vardict.txt")
       vardict
```

```
BaselineKaijuDiet2020^ = 8.341e9
```

**f)** Check your error handling also by making a corrupted addition to the dictionary and rerunning part (c). Try similar strategies to those in the previous part. Your code should trap errors and return the results specified above, not abort with an unhandled exception. Do at least three different tests for this part and the previous one, and document your functions' behaviors. [5 pts]

```
[215]: vardict = file_read("vardict.txt")
       vardict

       vardict["bears_with_6arms"] = "thirteen"

       file_write("vardict2.txt", vardict, overwrite=True)

       del vardict["bears_with_6arms"]
```

```
bears_with_6arms = thirteen does not follow formatting rules!
```

```
[216]: vardict["789chessRookHeight"] = 45.327

       file_write("vardict2.txt", vardict, overwrite=True)

       del vardict["789chessRookHeight"]
```

```
789chessRookHeight = 45.327 does not follow formatting rules!
```

```
[217]: vardict["__neightborHondaYear$%"] = 2009

       file_write("vardict2.txt", vardict, overwrite=True)
```

```
__neightborHondaYear$% = 2009 does not follow formatting rules!
```