Python Technical Test - Zuru

Problem Overview:

Write a Python program, that takes a `json` file (which contains the information of a directory in nested structure) and prints out its content in the console in the style of `ls` (linux utility).

Example:

Consider the following file (`structure.json`):

```
{
  "name": "interpreter",
  "size": 4096,
  "time_modified": 1699957865,
  "permissions": "-rw-r--r--",
  "contents": [
    {
      "name": ".gitignore",
      "size": 8911,
      "time_modified": 1699941437,
      "permissions": "drwxr-xr-x"
    },
    {
      "name": "LICENSE",
      "size": 1071,
      "time_modified": 1699941437,
      "permissions": "drwxr-xr-x"
    },
    {
      "name": "README.md",
      "size": 83,
      "time_modified": 1699941437,
      "permissions": "drwxr-xr-x"
    },
    {
      "name": "ast",
      "size": 4096,
      "time_modified": 1699957739,
      "permissions": "-rw-r--r--",
      "contents": [
        {
          "name": "go.mod",
          "size": 225,
          "time_modified": 1699957780,
          "permissions": "-rw-r--r--"
    },
    {
      "name": "ast.go",
      "size": 837,
      "time_modified": 1699957719,
      "permissions": "drwxr-xr-x"
    }
  ]
},
{
  "name": "go.mod",
  "size": 60,
  "time_modified": 1699950073,
  "permissions": "drwxr-xr-x"
},
{
  "name": "lexer",
  "size": 4096,
  "time_modified": 1699955487,
  "permissions": "drwxr-xr-x",
  "contents": [
    {
      "name": "lexer_test.go",
      "size": 1729,
      "time_modified": 1699955126,
      "permissions": "drwxr-xr-x"
    },
    {
      "name": "go.mod",
      "size": 227,
      "time_modified": 1699944819,
      "permissions": "-rw-r--r--"
    },
    {
      "name": "lexer.go",
      "size": 2886,
      "time_modified": 1699955487,
      "permissions": "drwxr-xr-x"
    }
  ]
},
{
  "name": "main.go",
  "size": 74,
  "time_modified": 1699950453,
  "permissions": "-rw-r--r--"
```

```
},
{
  "name": "parser",
  "size": 4096,
  "time_modified": 1700205662,
  "permissions": "drwxr-xr-x",
  "contents": [
    {
      "name": "parser_test.go",
      "size": 1342,
      "time_modified": 1700205662,
      "permissions": "drwxr-xr-x"
    },
    {
      "name": "parser.go",
      "size": 1622,
      "time_modified": 1700202950,
      "permissions": "-rw-r--r--"
    },
    {
      "name": "go.mod",
      "size": 533,
      "time_modified": 1699958000,
      "permissions": "drwxr-xr-x"
    }
  ]
},
{
  "name": "token",
  "size": 4096,
  "time_modified": 1699954070,
  "permissions": "-rw-r--r--",
  "contents": [
    {
      "name": "token.go",
      "size": 910,
      "time_modified": 1699954070,
      "permissions": "-rw-r--r--"
    },
    {
      "name": "go.mod",
      "size": 66,
      "time_modified": 1699944730,
      "permissions": "drwxr-xr-x"
    }
  ]
```

```
    }
  ]
}
```

This is meant to be equivalent to the following structure:

```
interpreter
|-- .gitignore
|-- LICENSE
|-- README.md
|-- ast
|   |-- ast.go
|   |-- go.mod
|-- go.mod
|-- lexer
|   |-- go.mod
|   |-- lexer.go
|   |-- lexer_test.go
|-- main.go
|-- parser
|   |-- go.mod
|   |-- parser.go
|   |-- parser_test.go
|-- token
    |-- go.mod
    |-- token.go
```

As might be evident from the structure, the field `name` refers to the name of the file or directory, `size` refers to the size on disk in bytes, `time_modified` refers to the time the file or directory was last modified, in seconds (epoch), `permissions` refers to the permissions for the file or directory in unix terms. The **field `contents` are only present for items (file/directory), which are directories**, and can contain a list of other items that are present within the directory.

Subtask 1: `ls` (10 points)

First command to implement is `ls`. Name of your program should be `pyls`. For the above structure, your script should parse the json file, and when run like the following:

```
$ python -m pyls
```

Should produce the following output:

```
LICENSE  README.md  ast  go.mod  lexer  main.go  parser  token
```

This lists out the top level (in the directory `interpreter`) directories and files. Notice it does not list `.gitignore`, because that is the default behaviour of `ls`, i.e. files and directories whose names start with `.` are omitted by default.

NOTE: following are some command line arguments that you will need to implement, it should be implemented in such a way, that the command line arguments are composable with each other.

Subtask 2: `ls -A` (2 points)

Implement the argument `-A`, which prints all the files and directories (including files starting with '.'), example:

```
$ python -m pyls -A
```

Should produce the following output:

```
.gitignore  LICENSE  README.md  ast  go.mod  lexer  main.go  parser  token
```

Subtask 3: `ls -l` (10 points)

Implement the argument `-l`, that prints the results vertically with additional information:

```
$ python -m pyls -l

-rw-r--r-- 1071 Nov 14 11:27 LICENSE
-rw-r--r--   83 Nov 14 11:27 README.md
drwxr-xr-x 4096 Nov 14 15:58 ast
-rw-r--r--   60 Nov 14 13:51 go.mod
drwxr-xr-x 4096 Nov 14 15:21 lexer
-rw-r--r--   74 Nov 14 13:57 main.go
drwxr-xr-x 4096 Nov 17 12:51 parser
drwxr-xr-x 4096 Nov 14 14:57 token
```

NOTE: First column corresponds to permissions, 2nd column corresponds to size, 3rd to 5th is date and time, and the last is file or directory name.

Subtask 4: `ls -l -r` (3 points)

Implement the argument `-r`, that prints the results in reverse:

```
$ python -m pyls -l -r

drwxr-xr-x 4096 Nov 14 14:57 token
drwxr-xr-x 4096 Nov 17 12:51 parser
-rw-r--r--   74 Nov 14 13:57 main.go
drwxr-xr-x 4096 Nov 14 15:21 lexer
-rw-r--r--   60 Nov 14 13:51 go.mod
drwxr-xr-x 4096 Nov 14 15:58 ast
-rw-r--r--   83 Nov 14 11:27 README.md
-rw-r--r-- 1071 Nov 14 11:27 LICENSE
```

Subtask 5: `ls -l -r -t` (5 points)

Implement the argument `-t` that prints the results sorted by `time_modified` (oldest first):

```
$ python -m pyls -l -r -t

drwxr-xr-x 4096 Nov 17 12:51 parser
drwxr-xr-x 4096 Nov 14 15:58 ast
drwxr-xr-x 4096 Nov 14 15:21 lexer
drwxr-xr-x 4096 Nov 14 14:57 token
-rw-r--r--   74 Nov 14 13:57 main.go
-rw-r--r--   60 Nov 14 13:51 go.mod
-rw-r--r-- 1071 Nov 14 11:27 LICENSE
-rw-r--r--   83 Nov 14 11:27 README.md
```

NOTE: Notice, in the above example, the flag `-r` is also present, so the files are printed in reverse order of `time_modified` (i.e. newest first)

Subtask 6: `ls -l -r -t --filter=<option>` (5 points)

Implement the argument `--filter=<option>`, where available options are: `file` and `dir`. This is a custom command, and does not exist in `ls` utility under this name. This command will filter the output based on given `option`.

NOTE: The only valid options are `file` and `dir`. Giving any other options should print out helpful error message.

Example:

1. `dir`

```
$ python -m pyls -l -r -t --filter=dir

drwxr-xr-x 4096 Nov 17 12:51 parser
drwxr-xr-x 4096 Nov 14 15:58 ast
drwxr-xr-x 4096 Nov 14 15:21 lexer
drwxr-xr-x 4096 Nov 14 14:57 token
```

2. `file`

```
$ python -m pyls -l -r -t --filter=file

-rw-r--r--   74 Nov 14 13:57 main.go
-rw-r--r--   60 Nov 14 13:51 go.mod
-rw-r--r-- 1071 Nov 14 11:27 LICENSE
-rw-r--r--   83 Nov 14 11:27 README.md
```

3. `invalid`

```
$ python -m pyls -l -r -t --filter=folder

error: 'folder' is not a valid filter criteria. Available filters are 'dir' and 'file'
```

Subtask 7: Handle Paths (5 points)

Your program should be able to navigate the structure within the `json`. So if the command is:

```
$ python -m pyls -l parser
```

The output will be the contents of the `parser` subdirectory under `interpreter` directory

```
-rw-r--r--  533 Nov 14 16:03 go.mod
-rw-r--r-- 1622 Nov 17 12:05 parser.go
-rw-r--r-- 1342 Nov 17 12:51 parser_test.go
```

If the path is a file, it should list the file itself:

```
$ python -m pyls -l parser/parser.go

-rw-r--r-- 1622 Nov 17 12:05 ./parser/parser.go
```

It should handle relative paths within the directory: `./parser` should be equivalent to `parser` `.` should be equivalent to no argument (i.e. the current directory) If a path does not exist, you should print the following:

```
$ python pyls non_existent_path

error: cannot access 'non_existent_path': No such file or directory
```

Assume there will be no `..` (goes to the parent directory)

Subtask 8: `ls -h` (5 points)

Show human readable size:

```
$ python -m pyls -l parser

-rw-r--r--  533 Nov 14 16:03 go.mod
-rw-r--r-- 1.6K Nov 17 12:05 parser.go
-rw-r--r-- 1.4K Nov 17 12:51 parser_test.go
```

i.e. for large (greater than 1023 bytes) sizes in bytes are converted to kilobyte, megabyte, gigabyte accordingly.

Subtask 9: `ls --help` (5 points)

```
$ python -m pyls --help
```

```
# should print a helpful message.
# should include description and usage
# should list all available commands with choices where applicable
```

Bonus: (10 points)

Include a `pyproject.toml` and configure it so that installing the project using `pip` it adds a `pyls` system command to the system (you may need to add the path to the binary to your system path). So that the script works directly using:

```
$ pyls

LICENSE  README.md  ast  go.mod  lexer  main.go  parser  token
```

General Instructions:

- You are allowed to use only standard library modules and packages, no external modules/packages should be used (except `pytest` if needed).
- Your code should be version controlled, and shared using `GitHub`. Commit messages should be reasonably descriptive.
- Your repository should contain a proper README.md, which should describe the install procedure, and usage.
- Code should be readable, maintainable, and easily composable.
- Code should be Pythonic, follow PEP8 guidelines where possible.
- Your code should include tests (preferably using `pytest`).
- Your code should be properly typed (see `typing`).

General Advice:

- In addition to the correctness, the assignment will be evaluated on the coding style and design of your program.
- In case of time constraint, you are advised to pay attention to the correctness, and design of the solution, rather than trying to complete all the subtasks.