parameter_consistency

February 13, 2025

[25]:
```
import os
import re
```

[26]:
```
# Directory to search
#directory_to_search = "/home/elias/IdeaProjects/systemds/scripts/builtin"
directory_to_search = "/Users/eliasstrauss/Desktop/TU/systemds-fork/scripts/
 ↪builtin"

matches = []

# Regex pattern for matching function and return statements
pattern = re.compile(r"(?P<func>\b\w+\b\s*=\s*function\s*\([^)]*\)\s*)"  #␣
 ↪Match 'fname = function (...)'
                     r"(?:(?P<return>return\s*\([^)]*\)))?",             #␣
 ↪Match 'return (...)', optional
                     re.DOTALL)                                          #␣
 ↪Enable dotall to match across lines

# Walk through the directory
for root, _, files in os.walk(directory_to_search):
    for file in files:
        file_path = os.path.join(root, file)
        try:
            with open(file_path, 'r', encoding='utf-8') as f:
                content = f.read()

                # replace matrix(nrow(X), ncol(X)) by matrix to avoid the␣
 ↪pattern function (...) to pick up the the first closing bracket from nrow(X)
                # this quick hack is easier than counting opening and closing␣
 ↪brackets...
                pattern_nrow = r"nrow\([^)]*\)"
                content = re.sub(pattern_nrow, "nrow", content)
                pattern_ncol = r"ncol\([^)]*\)"
                content = re.sub(pattern_ncol, "ncol", content)
                pattern_matrix = r"matrix\([^)]*\)"
                content = re.sub(pattern_matrix, "matrix", content)
                pattern_frame = r"as.frame\([^)]*\)"
```

```
                content = re.sub(pattern_frame, "frame", content)
                pattern_list = r"list\(\)"
                content = re.sub(pattern_list, "list", content)

                # Find all matches in the file
                no_match = True
                for match in pattern.finditer(content):
                    function_header = match.group()
                    matches.append({
                        "file_name": file_path,
                        "function_header": function_header
                    })
                    no_match = False
                if no_match:
                    print(file_path)
        except (UnicodeDecodeError, IOError):
            print(f"Could not read file: {file_path}")

results = matches
print(len(results), len([1 for _ in files for _, _, files in os.
 ↪walk(directory_to_search)]))
```

```
434 192
```

[27]:
```
results_inner = [r for r in results if r["function_header"][1] != "_"]
results  = [r for r in results if r["function_header"][1] == "_"]
len(results)
```

[27]:
```
203
```

[28]:
```
# check that there were no errors while parsing
for result in results:
    header = result["function_header"]
    if header.count("(") != header.count(")"):
        print(header)
```

[ ]:
```
# Print results
for result in results:
    print(f"File: {result['file_name']}")
    print(f"Function Header: {result['function_header']}")
    print("-" * 40)
```

[30]:
```
unique_parameters = dict()
intput_parameter_filter = re.compile(r"function\s*\(([^)]*)\)")
param_with_default_pattern = re.compile(r"(\w+)\s+(\w+)\s*=\s*(.+)")
for result in results:
    function_header = result["function_header"]
```

```
    # Search for the parameters inside "function(...)"
    param_match = intput_parameter_filter.search(function_header)
    if param_match:
        param_str = param_match.group(1)  # The content inside the parentheses

        # Split the parameters by commas and process each one
        params = [p.strip() for p in param_str.split(",") if p.strip()]
        for param in params:
            if not '=' in param:
                parts = param.split()
                if not parts[-1] in unique_parameters:
                    unique_parameters[parts[-1]] = 1
                else:
                    unique_parameters[parts[-1]] += 1
            else:
               # Extract the parameter name from possible formats:
                # NAME, DATATYPE NAME, DATATYPE NAME=DEFAULT, DATATYPE NAME =␣
 ↪DEFAULT
                param = param.replace('[', '').replace(']','')
                match = param_with_default_pattern.match(param)
                if match:
                    param_name = match.group(2)
                    if not param_name in unique_parameters:
                        #print(param_name, "[{}]".format(param))
                        unique_parameters[param_name] = 1
                    else:
                        unique_parameters[param_name] += 1
                else:
                    print("Warning no match for: " + param)
print(len(unique_parameters))
```

```
340
```

[31]:
```
names = list(unique_parameters.keys())
names.sort()
```

[ ]:
```
for name in names:
    print("{name: <28} : {c}".format(name=name, c=unique_parameters[name]))
```

[37]:
```
para_by_count = sorted([pair for pair in unique_parameters.items()], key=lambda␣
 ↪pair: (pair[1],pair[0]), reverse=False)
```

[ ]:
```
for n, c in para_by_count:
    print("{name: <28} : {c}".format(name=n, c=c))
```