path to profiling records:

Namespace `src`

Sub-modules

- src.main
- src.package

Namespace `src.main`

Sub-modules

- src.main.addscriptdir2path
- src.main.process_bam_files

Module `src.main.addscriptdir2path`

append run script directory to system path

Functions

Function `add_package2env_var`

```
def add_package2env_var() -> None
```

re-define system path to include modules, packages, and libraries in environment variable using dotenv

Function `adddirname2syspath`

```
def adddirname2syspath()
```

adds run script directory to system path

Function `check_srcpkgpath_env_var`

```
def check_srcpkgpath_env_var(
    environment_variable_file: str
) -> None
```

check that the declared environment variables in the .env file exist

Function `find_env_var_file`

```
def find_env_var_file(
    path2runscript_dir: str
) -> list
```

finds the environmental variable file

Function `get_dir2put_env_file`

```
def get_dir2put_env_file() -> str
```

get the suggested directory to put the environment variable file.

Function `load_package_paths`

```
def load_package_paths(
    env_var_files: str
) -> None
```

load more than one environment variable files

Function `run_script_base_dir`

```
def run_script_base_dir() -> str
```

get run script base directory

Function `specifically_load_files`

```
def specifically_load_files(
    file2load: str
) -> None
```

specifically load environment variable files

Module `src.main.process_bam_files`

This script: 1. performs read counts for the regions specified in an input BED file and outputs it in JSON format. 2. extract reads in the regions and convert it into a FASTA file

Required ——= - Python >= 3.10 - python-dotenv>=1.0.0 - for additional dependencies, see requirements.txt

Functions

Function `main`

```
def main() -> None
```

main function to run commandline arguments and call other functions to run.

Module `src.package`

Sub-modules

- src.package.bamoperations
- src.package.commandlineoperations
- src.package.datastructureoperations
- src.package.enums
- src.package.fileoperations
- src.package.profiling

Module `src.package.bamoperations`

Sub-modules

- src.package.bamoperations.bamoperations

Module `src.package.bamoperations.bamoperations`

A collection of classes or functions that performs bam processing operations

Classes

Class `BamOperator`

```
class BamOperator(
    bam_files: list,
    bed_file: str,
    output_directory: str
)
```

Performs bam processing operations

Constructor

Parameters bam_files :list Path to bam files bed_file:str Path to bwa meth bam file output_directory:str Path to output directory

Class variables

Variable `bam_files_directory`

Variable `bed_file`

Variable `output_directory`

Methods

Method `process_bam_files`

```
def process_bam_files(
    self
) -> None
```

process bam files

Module `src.package.commandlineoperations`

Sub-modules

- main

Module `__main__`

A collection of classes or functions that evaluates proteins from nodes using keyword, pathway comments, function comments, and catalytic activity comments

Classes

Class `CliInputArgumentGetter`

```
class CliInputArgumentGetter
```

Wrapper for argparse that returns an object of the class for ease of use

Static methods

Method `check_input_arguments`

```
def check_input_arguments(
    cli_input_arguments: argparse.Namespace
) -> None
```

check or verify input arguments

Method `get_cli_input_arguments`

```
def get_cli_input_arguments(
    args=None
) -> argparse.Namespace
```

gets input arguments from the commandline interface

Module `src.package.datastructureoperations`

Sub-modules

- src.package.datastructureoperations.listoperations

Module `src.package.datastructureoperations.listoperations`

Sub-modules

- **main**

Module `__main__`

A collection of functions that performs list handling operations within a script.

Functions

Function `get_first_element`

```
def get_first_element(
    a_list_item: Union[list, tuple, ForwardRef(None)]
) -> str
```

gets the first element of a list or tuple

Function `get_second_element`

```
def get_second_element(
    a_list_item: Union[list, tuple, ForwardRef(None)]
) -> str
```

gets the second element of a list or tuple

Function `get_third_element`

```
def get_third_element(
    a_list_item: Union[list, tuple, ForwardRef(None)]
) -> str
```

gets the third element of a list or tuple

Module `src.package.enums`

Sub-modules

- src.package.enums.delimiter_enums

Module `src.package.enums.delimiter_enums`

A collection of classes or functions that defines delimiter enums

Classes

Class `Delimiters`

```
class Delimiters
```

defines delimiters

Class variables

Variable `FASTA_IDENTIFIER`

Variable `HYPHEN`

Variable `NEW_LINER`

Variable `TAB_SEPERATOR`

Module `src.package.fileoperations`

Sub-modules

- **main**
- src.package.fileoperations.filehandlers

Module `__main__`

A collection of functions that performs file writing operations.

Classes

Class `FileWriter`

```
class FileWriter(
    file_path,
    mode
)
```

The FileWriter class handles the writing of the data to a file.

Creates a new FileWriter object.

:param file_path: Path to the file. :param mode: Mode of the file.

Methods

Method `write_json`

```
def write_json(
    self,
    data: <module 'json' from '/usr/lib/python3.10/json/__init__.py'>
) -> None
```

Writes to json format

:param data: String data to write.

Method `write_str`

```
def write_str(
    self,
    data: str
) -> None
```

Writes data to a file. Data here is a string

:param data: String data to write.

Module `src.package.fileoperations.filehandlers`

list of functions that handle files

Functions

Function `globally_get_all_files`

```
def globally_get_all_files(
    path2directory: str,
    file_extension=None
) -> list
```

gets all files in a directory if supplied with a path to all files e.g. (/path/to/file, 'fasta')

Function `read_csv`

```
def read_csv(
    csv_file: str,
    delimiter: str
) -> list[str]
```

get csv contents as Generators

Module `src.package.profiling`

Sub-modules

- main

Module `__main__`

A collection of functions that performs profiling logging tasks within a script.

Functions

Function `begin_profiling`

```
def begin_profiling(
    path_to_file: str
) -> tuple
```

logs program processes, logs start run time,checks memory using resource in kilobytes divided by 1000 for memory usage in Mb

Function `end_profiling`

```
def end_profiling() -> float
```

logs end run time

Function `print_mem`

```
def print_mem() -> str
```

determine memory usage. mem = divides by 1k to get the measurement in kilobytes (rather than bytes). mem = divides by 1k again to get the measurement in megabytes (rather than kilobytes as per first mem above)

Classes

Class `ProfileLogger`

```
class ProfileLogger(
    profile_began: tuple,
    profile_end: float
)
```

logs profiling runs

Methods

Method `log_profiling`

```
def log_profiling(
    self
) -> None
```

logs memory and time associated with script
