Implementation documentation for 1st task for IPP 2020/2021
Name and surname: Matúš Fabo
Login: xfabom01

Script `interpret.py`:

Requirement:
The goal of `parse.php` script is to check source code written in IPPcode21 for lexical and syntactic errors and convert correct IPPcode21 code to XML representation for further use by `interpret.py` interpreter. The specifications for this script are found in `ipp21spec.pdf`.

Implementation:
Overall structure of this script is split into 4 main parts:

- **Constants**
  There are 4 constants that contain necessary information that will not change during runtime, but can be changed if given specification changes.
  - `XML` – Dictionary containing information about XML element structure
  - `INSTRUCTION` – This constant is a dictionary that contains instruction parameters for syntactic analysis of instruction parameters
  - `REGEX` – This constant is a dictionary that contains regular expressions for lexical analysis of instruction parameters
  - `ERRCODE` – This constant is a dictionary that encapsulates error codes with names for better code readability

- **Program argument handling**
  Argument handling is done with python builtin library `argparse` and mutually exclusive arguments are done manually.

- **XML validation**
  - Well-formed checking is done by python `lxml` library when converting to `etree` structure
  - Syntactic & lexical analysis is done manually with `element_integrity` function
  - Instruction order checking and fixing is done before syntactic and lexical analysis with python builtin `sorted` function

- **Code execution**
  To make code more readable, all helper functions for execution are defined inside `execute` function. Code execution itself is split into 3 parts:
  - **Helper functions** – Functions that help with semantic checking and functions to execute the instructions.
  - **Code execution** – The execution itself is wrapped in `instr_dot_exe` function. This function contains functions for every instruction to check for semantics and finally execute the instruction. And to avoid creating long and unreadable state machine, calling the instruction functions is done by python builtin `eval` function.
  - **Program runtime** – Initializing initial state of program runtime and running the program itself – calling `instr_dot_exe` function for every instruction.

Implementation documentation for 1st task for IPP 2020/2021
Name and surname: Matúš Fabo
Login: xfabom01

Script `test.php`:

Requirement:
The goal of `test.php` script is to test `parse.php` and `interpret.py` implementation and generate HTML visualization of the result.

Implementation:
The script is split into 4 parts:

- **Constants & containers**
  - `HELP_MSG` – Constant string containing information about program usage
  - `ERRCODE` – Constant array encapsulating error codes under string keys
  - `ARGS` – Array that holds all program arguments
  - `TEST` – Array that holds test results

- **Parsing program arguments**
  Argument parsing is done by looping through every element in `argv` variable and storing them in `ARGS` array

- **Testing**
  Testing is split into 2 parts:
  - **Passing through directories** – If `recursive` flag was set, script will go through every directory in `directory` program argument and tests every test inside

  - **Testing** – Script goes through every `*.src` file and plugs its contents in either `parse.php` or `interpret.py` script, depends on which flag was set. If neither flag was set, `*.src` file contents are sent to `parse.php` script and its output is sent directly to `interpret.py` script. The scripts return code and output is then compared with corresponding `*.rc` and `*.out` files. The output of `parse.php` script is compared with JExamXML program.

- **HTML generation**
  HTML generation is not implemented at this point in time