`main()`

- Esta função é o **ponto de partida** do CLI (*Command Line Interface*) do Mythril.
- É responsável por **capturar os inputs fornecidos pelo usuário** e iniciar o processo de análise com base nessas entradas.

Código - Observações

- Locais marcados com `# f`:
  Indicam que, durante a execução do contrato analisado, a condição **não foi satisfeita** e, portanto, **não entrou** na condicional associada.

- Locais marcados com `# v`:
  Indicam que a condição **foi satisfeita** e, durante a execução, o contrato **entrou** na condicional associada.

Essas marcações ajudam a identificar o comportamento do contrato em diferentes cenários de execução.

```python
def main() -> None:
    """The main CLI interface entry point."""

    rpc_parser = get_rpc_parser()
    utilities_parser = get_utilities_parser()
    runtime_input_parser = get_runtime_input_parser()
    creation_input_parser = get_creation_input_parser()
    output_parser = get_output_parser()

    parser = argparse.ArgumentParser(
        description="Security analysis of Ethereum smart contracts"
    )
    parser.add_argument("--epic", action="store_true",
help=argparse.SUPPRESS)
    parser.add_argument(
        "-v", type=int, help="log level (0-5)", metavar="LOG_LEVEL",
default=2
    )

    subparsers = parser.add_subparsers(dest="command", help="Commands")
    safe_function_parser = subparsers.add_parser(
        SAFE_FUNCTIONS_COMMAND,
```

```
        help="Check functions which are completely safe using symbolic
execution",
        parents=[
            rpc_parser,
            utilities_parser,
            creation_input_parser,
            runtime_input_parser,
            output_parser,
        ],
        formatter_class=RawTextHelpFormatter,
    )
    analyzer_parser = subparsers.add_parser(
        ANALYZE_LIST[0],
        help="Triggers the analysis of the smart contract",
        parents=[
            rpc_parser,
            utilities_parser,
            creation_input_parser,
            runtime_input_parser,
            output_parser,
        ],
        aliases=ANALYZE_LIST[1:],
        formatter_class=RawTextHelpFormatter,
    )
    create_safe_functions_parser(safe_function_parser)
    create_analyzer_parser(analyzer_parser)

    disassemble_parser = subparsers.add_parser(
        DISASSEMBLE_LIST[0],
        help="Disassembles the smart contract",
        aliases=DISASSEMBLE_LIST[1:],
        parents=[
            rpc_parser,
            utilities_parser,
            creation_input_parser,
            runtime_input_parser,
        ],
        formatter_class=RawTextHelpFormatter,
    )
    create_disassemble_parser(disassemble_parser)

    concolic_parser = subparsers.add_parser(
        CONCOLIC_LIST[0],
        help="Runs concolic execution to flip the desired branches",
        aliases=CONCOLIC_LIST[1:],
        parents=[],
        formatter_class=RawTextHelpFormatter,
    )
```

```
    create_concolic_parser(concolic_parser)

    foundry_parser = subparsers.add_parser(
        FOUNDRY_LIST[0],
        help="Triggers the analysis of the smart contract",
        parents=[
            rpc_parser,
            utilities_parser,
            output_parser,
        ],
        aliases=FOUNDRY_LIST[1:],
        formatter_class=RawTextHelpFormatter,
    )

    _ = subparsers.add_parser(
        LIST_DETECTORS_COMMAND,
        parents=[output_parser],
        help="Lists available detection modules",
    )
    read_storage_parser = subparsers.add_parser(
        READ_STORAGE_COMNAND,
        help="Retrieves storage slots from a given address through rpc",
        parents=[rpc_parser],
    )
    contract_func_to_hash = subparsers.add_parser(
        FUNCTION_TO_HASH_COMMAND, help="Returns the hash signature of the
function"
    )
    contract_hash_to_addr = subparsers.add_parser(
        HASH_TO_ADDRESS_COMMAND,
        help="converts the hashes in the blockchain to ethereum address",
    )
    subparsers.add_parser(
        VERSION_COMMAND, parents=[output_parser], help="Outputs the version"
    )

    create_read_storage_parser(read_storage_parser)
    create_hash_to_addr_parser(contract_hash_to_addr)
    create_func_to_hash_parser(contract_func_to_hash)
    create_foundry_parser(foundry_parser)
    subparsers.add_parser(HELP_COMMAND, add_help=False)

    # Get config values

    args = parser.parse_args()
    parse_args_and_execute(parser=parser, args=args)
```