```
"""
Common module for Python scripts.

This module gives access to general functions.
"""
################################################################################
# Common module for Python scripts.
#
# Written by Karel Zimmer <info@karelzimmer.nl>, CC0 1.0 Universal
# <https://creativecommons.org/publicdomain/zero/1.0>, 2021-2023.
################################################################################

################################################################################
# Import
################################################################################

import argparse
import gettext
import os
import subprocess
import sys
import time

gettext.bindtextdomain('kz', '/usr/share/locale')
gettext.textdomain('kz')
_ = gettext.gettext


################################################################################
# Variables
################################################################################

module_name = 'kz_common.py'
module_desc = (_('Common module for Python scripts'))
module_path = f"{os.path.realpath(os.path.dirname(__file__))}"

ok = 0
error = 1

bold = '\033[1m'
normal = '\033[0m'


################################################################################
# Functions
################################################################################

def check_dpkgd_snapd():
    """
    This function checks for a Debian package manager already running.
    """
    dpkg_wait = 10

    try:
        subprocess.run('ls /snap/core/*/var/cache/debconf/config.dat',
                       shell=True, check=True, stdout=subprocess.DEVNULL,
                       stderr=subprocess.DEVNULL)
    except Exception:
        snaps = False
    else:
        snaps = True

    while True:
        if snaps:
```

```
            try:
                subprocess.run('sudo fuser /var/cache/apt/archives/lock '
                               '/var/lib/apt/lists/lock  /var/lib/dpkg/lock '
                               '/var/cache/debconf/config.dat '
                               '/snap/core/*/var/cache/debconf/config.dat',
                               shell=True, check=True,
                               stdout=subprocess.DEVNULL,
                               stderr=subprocess.DEVNULL)
            except Exception:
                break
            else:
                print(_('Wait {}s for another package manager to finish...').
                      format(dpkg_wait))
                time.sleep(dpkg_wait)
        else:
            try:
                subprocess.run('sudo fuser /var/cache/apt/archives/lock '
                               '/var/lib/apt/lists/lock  /var/lib/dpkg/lock '
                               '/var/cache/debconf/config.dat',
                               shell=True, check=True,
                               stdout=subprocess.DEVNULL,
                               stderr=subprocess.DEVNULL)
            except Exception:
                break
            else:
                print(_('Wait {}s for another package manager to finish...').
                      format(dpkg_wait))
                time.sleep(dpkg_wait)


def check_on_ac_power(program_name):
    """
    This function monitors the power supply.
    """
    if subprocess.run('on_ac_power', shell=True,
                      stderr=subprocess.DEVNULL).returncode == 1:
        print('\n' + _('The computer now uses only the battery for power.\n\n'
              'It is recommended to connect the computer to the wall socket.'))
        try:
            input('\n' + _('Press the Enter key to continue [Enter]: '))
        except KeyboardInterrupt:
            print('\n' + _('Program {} has been interrupted.').
                  format(program_name))
            sys.exit(error)


def check_user_root(program_name, display_name):
    """
    This function checks if the user is root.
    """
    if check_user_sudo() != ok:
        print(_('Already performed by the administrator.'))
        sys.exit(ok)
    else:
        try:
            subprocess.run('sudo -n true', shell=True, check=True,
                           stdout=subprocess.DEVNULL,
                           stderr=subprocess.DEVNULL)
        except Exception:
            try:
                subprocess.run('sudo true', shell=True, check=True)
            except KeyboardInterrupt:
                print('\n' + _('Program {} has been interrupted.').
                      format(program_name))
```

```
                sys.exit(error)
            except Exception as ex:
                print(ex)
                sys.exit(error)


def check_user_sudo():
    """
    This function checks if the user is allowed to use sudo.
    """
    if os.getuid() == 0:
        return(ok)
    try:
        subprocess.run('groups $USER | grep --quiet --regexp=sudo',
                       shell=True, check=True,
                       stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    except Exception:
        return(error)
    else:
        return(ok)


def process_option(program_name, program_desc, display_name):
    """
    This function handles the general options.
    """
    usage_line = _("type '{} --usage' for more information").\
        format(display_name)

    parser = argparse.ArgumentParser(prog=display_name, add_help=False,
                                     usage=usage_line)
    parser.add_argument('-h', '--help', action='store_true')
    parser.add_argument('-u', '--usage', action='store_true')
    parser.add_argument('-v', '--version', action='store_true')
    args = parser.parse_args()

    if args.help:
        process_option_help(display_name, program_desc, program_name)
        sys.exit(ok)
    elif args.usage:
        process_option_usage(display_name)
        sys.exit(ok)
    elif args.version:
        process_option_version(program_name)
        sys.exit(ok)


def process_option_help(display_name, program_desc, program_name):
    """
    This function shows the available help.
    """
    print(_('Usage: {} [OPTION...]\n'
            '\n'
            '{}.\n'
            '\n'
            'Options:\n'
            '  -h, --help     give this help list\n'
            '  -u, --usage    give a short usage message\n'
            '  -v, --version  print program version\n\n'
            "Type 'man {}' or see the \
\x1b]8;;man:{}(1)\x1b\\{} man page\x1b]8;;\x1b\\ for more information.").
          format(display_name, program_desc, display_name, program_name,
                 display_name))
```

```
def process_option_usage(display_name):
    """
    This function shows the available options.
    """
    print(_('Usage: {} [-h|--help] [-u|--usage] [-v|--version]\n'
            '\n'
            "Type '{} --help' for more information.").
          format(display_name, display_name))


def process_option_version(program_name):
    """
    This function displays version, author, and license information.
    """
    build_id = '????-??-?? ??:??'
    cmd = ''
    grep_expr = '# <https://creativecommons.org'
    program_year = '????'

    try:
        with open('/usr/local/etc/kz-build-id') as fh:
            build_id = fh.read()
    except FileNotFoundError:
        build_id = '????-??-?? ??:??'
    except Exception as ex:
        print(ex)
        sys.exit(error)
    finally:
        cmd = f"grep '--regexp={grep_expr}' {module_path}/{program_name}"
        cmd = f"{cmd} | cut --delimiter=' ' --fields=3"
        program_year = subprocess.check_output(cmd, shell=True,
                                               stderr=subprocess.DEVNULL)
        program_year = program_year.decode('utf-8').strip()
        if program_year == '':
            program_year = '????'

        print(_('{} (kz) 365 ({})\n'
                '\n'
                'Written by Karel Zimmer <info@karelzimmer.nl>, CC0 1.0 \
Universal\n'
                '<https://creativecommons.org/publicdomain/zero/1.0>, {}')
              .format(program_name, build_id, program_year))


################################################################################
# Main
################################################################################

if __name__ == '__main__':
    print(_('{}: i am a module').format(module_name))
```