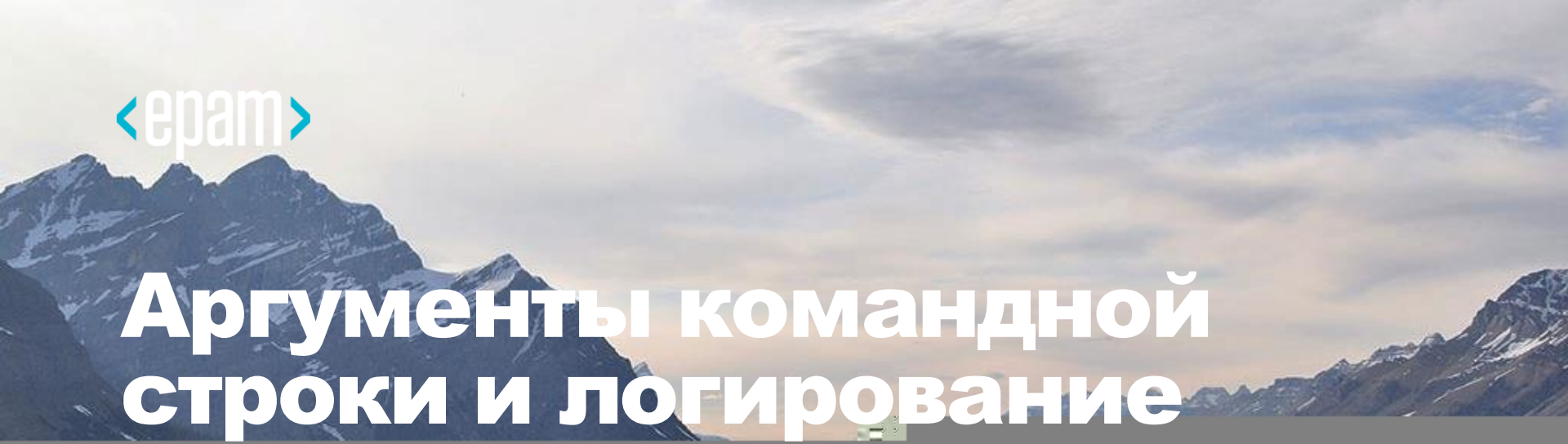


Аргументы командной строки и логирование

Python – Лекция #15

JULY 02, 2020

ТЕМЫ

1. Аргументы командной строки
2. Логирование

Часть 1

АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ

ОПРЕДЕЛЕНИЕ

Программы по способу взаимодействия с пользователем:

- UI
- CLI

Аргументы командной строки (CLI arguments) — параметры, предоставляемые программе в момент её запуска

Используются для выбора поведения программы при запуске

SYS.ARGV

sys.argv – список, содержащий аргументы командной строки, переданные программе

Задача: Напечатать переданные аргументы командной строки

```
import sys

if __name__ == "__main__":
    for param in sys.argv:
        print(param)
```

python params.py param1 param2 param3

КОНТРОЛЬ ЗА КОЛИЧЕСТВОМ ПЕРЕДАННЫХ ПАРАМЕТРОВ

Задача: Передать привет переданному параметру, если не передан - миру

```
import sys

if __name__ == "__main__":
    if len (sys.argv) > 1:
        print("Hello,
{}!".format(sys.argv[1]))
    else:
        print("Hello, world!")
```

python params.py Mr. Anderson

ИМЕНОВАННЫЕ АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ

Задача: Передать привет переданному параметру, если не передан – миру с помощью именованного параметра -n/--name

```
import sys

if __name__ == "__main__":
    if len(sys.argv) == 1:
        print("Hello, world!")
    else:
        if len(sys.argv) != 3:
            print("Error. There has to be 3 args. Received: {}".format(len(sys.argv)))
            sys.exit(1)

        param_name, param_value = sys.argv[1], sys.argv[2]

        if param_name == "--name" or param_name == "-n":
            print("Hello, {}!".format(param_value))
        else:
            print("Error. Unknown parameter '{}'".format(param_name))
            sys.exit(1)
```

БИБЛИОТЕКА ARGPARSE

Принцип работы с argparse:

- Создаем экземпляр класса *ArgumentParser*
- Добавляем в него информацию об ожидаемых параметрах с помощью метода add_argument (по одному вызову на каждый параметр)
- Разбираем командную строку помощью метода parse_args, передавая ему полученные параметры командной строки (кроме нулевого элемента списка sys.argv).
- Начинаем использовать полученные параметры.

ИМЕНОВАННЫЕ АРГУМЕНТЫ: ARGPARSE

Задача: Передать привет переданному параметру, если не передан – миру с помощью именованного параметра -n/--name

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--name', '-n', default='world')
    args = parser.parse_args()
    print("Hello, {}!".format(args.name))
```

python params.py --name Mr.Anderson

ПАРАМЕТРЫ-СПИСКИ

Задача: Передать привет списку имен в переданном параметре, если не передан – миру с помощью именованного параметра -n/--name

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-n', '--name', nargs='+', default=['world'])
    namespace = parser.parse_args()

    for name in namespace.name:
        print("Hello, {}!".format(name))
```

python params.py –n Mr.Anderson Ag.Smith

ВЫБОР ИЗ ВАРИАНТОВ

Задача: Передать привет переданному параметру, если не передан – миру (-n/--name). Список разрешенных для приветствования ограничен: Ann, John, Eva

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-n', '--name', choices=['Ann', 'John', 'Eva'], default='world')
    args = parser.parse_args()
    print("Hello, {}!".format(args.name))
```

python params.py –n John

УКАЗАНИЕ ТИПОВ ПАРАМЕТРОВ

Задача: Передать привет миру -c/--count раз

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-c', '--count', default=1, type=int)
    args = parser.parse_args()

    for _ in range(args.count):
        print("Hello, world!")
```

python params.py –c 3

ПАРАМЕТРЫ – ИМЕНА ФАЙЛОВ

Задача: Прочитать содержимое файла, имя которого передано через обязательный аргумент командной строки

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-n', '--name', type=argparse.FileType(), required=True)
    args = parser.parse_args()

    text = args.name.read()
    print(text)
    args.name.close()
```

python params.py –n just_a_file.txt

ПАРАМЕТРЫ – ФЛАГИ

Задача: Передать привет миру. Попрощаться с миром, если передан флаг --goodbye/-g

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-g', '--goodbye', action='store_true')
    args = parser.parse_args()

    print("Hello, world!")
    if args.goodbye:
        print("Goodbye, world!")
```

python params.py –g

ПРИМЕНЕНИЕ НЕСКОЛЬКИХ ПОЗИЦИОННЫХ ПАРАМЕТРОВ

Задача: Передать привет заданное количество раз заданному имени, используя 2 **позиционных** аргумента

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('name')
    parser.add_argument('count', type=int)
    args = parser.parse_args()

    for _ in range(args.count):
        print("Hello, {}!".format(args.name))
```

python params.py Ann 3

ПРИМЕНЕНИЕ НЕСКОЛЬКИХ ИМЕНОВАННЫХ ПАРАМЕТРОВ

Задача: Передать привет заданное количество раз заданному имени (миру, если не задано), используя 2 **именованных** аргумента

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--name', '-n', default='world')
    parser.add_argument('--count', '-c', type=int, required=True)
    args = parser.parse_args()

    for _ in range(args.count):
        print("Hello, {}!".format(args.name))
```

python params.py –c 3 -n Ann

ОФОРМЛЕНИЕ СПРАВКИ

Задача: Дополнить предыдущую программу справкой

```
import argparse

NAME = 'mr_hello’; VERSION = '0.0.1’; EPILOG = '(c) Andrei Gorlanov 2020'
DESCRIPTION = '''This is a program that are able to greet a person for a number of times'''

if __name__ == '__main__':
    p = argparse.ArgumentParser(prog=NAME, description=DESCRIPTION, epilog=EPILOG, add_help=False)
    p.add_argument('--name', '-n', default='world', help='Name of a person to greet')
    p.add_argument('--count', '-c', type=int, required=True, help='Num of greetings')
    p.add_argument('--help', '-h', action='help', help='Help message')
    p.add_argument('-v', action='version', help='Version', version='%(prog)s {}'.format(VERSION))
    args = p.parse_args()
```

python params.py –c 3 -n Ann

Часть 2

ЛОГИРОВАНИЕ

БИБЛИОТЕКА LOGGING

Принцип работы с logging:

- Устанавливаем уровень логирования методом *basicConfig()*
- Получаем экземпляр класса *Logger*
- Вызываем методы экземпляра, чтобы осуществить запись в лог

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

ЛОГИРОВАНИЕ

Пример

```
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("Main module")

logger.debug("This is a debug message")
logger.info("Informational message")
logger.warning("Warning appeared")
logger.error("An error has happened!")
logger.critical("Critical error. Time to self-termination")
```

ЛОГИРОВАНИЕ В ФАЙЛ

Пример

```
import logging

# add filemode="w" to overwrite
logging.basicConfig(level=logging.INFO, filename='sample.log')
logger = logging.getLogger()

logger.info("Informational message")
```

ЛОГИРОВАНИЕ ИСКЛЮЧЕНИЙ

Пример

```
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("ex")

try:
    raise RuntimeError
except RuntimeError:
    logger.exception("Error!")
```

ОПРЕДЕЛЕНИЕ ФОРМАТА СООБЩЕНИЙ

Пример

```
import logging

LOG_FORMAT = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
logging.basicConfig(level=logging.INFO, format=LOG_FORMAT)
logger = logging.getLogger("Some logger")

logger.info("Informational message")
```

FILEHANDLER и FORMATTER

Пример

```
import logging

LOG_FORMAT = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'

logger = logging.getLogger("Some logger")
file_handler = logging.FileHandler("some_logger.log")
formatter = logging.Formatter(LOG_FORMAT)
file_handler.setFormatter(formatter)
logger.addHandler(file_handler)

logger.warning("Warning message")
```

LOGGING CONFIGURATION. FILE

Пример конфигурационного файла

```
[loggers]
keys=root,someLogger

[handlers]
keys=fileHandler, consoleHandler

[formatters]
keys=myFormatter

[logger_root]
level=CRITICAL
handlers=consoleHandler

[logger_someLogger]
level=INFO
handlers=fileHandler
qualname=someLogger

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=myFormatter
args=(sys.stdout,)

[handler_fileHandler]
class=FileHandler
formatter=myFormatter
args=("some_logger.log",)

[formatter_myFormatter]
format=%(asctime)s - %(name)s -
%(levelname)s - %(message)s
datefmt="%Y.%m.%d %H:%M:%S"
```

LOGGING CONFIGURATION. FILE

Пример использования конфигурационного файла

```
import logging.config

logging.config.fileConfig('logging.conf')

logger = logging.getLogger("someLogger")
logger.info("Informational message")

root_logger = logging.getLogger()
root_logger.critical("Critical")
```

LOGGING CONFIGURATION. DICT

Пример использования конфигурационного файла

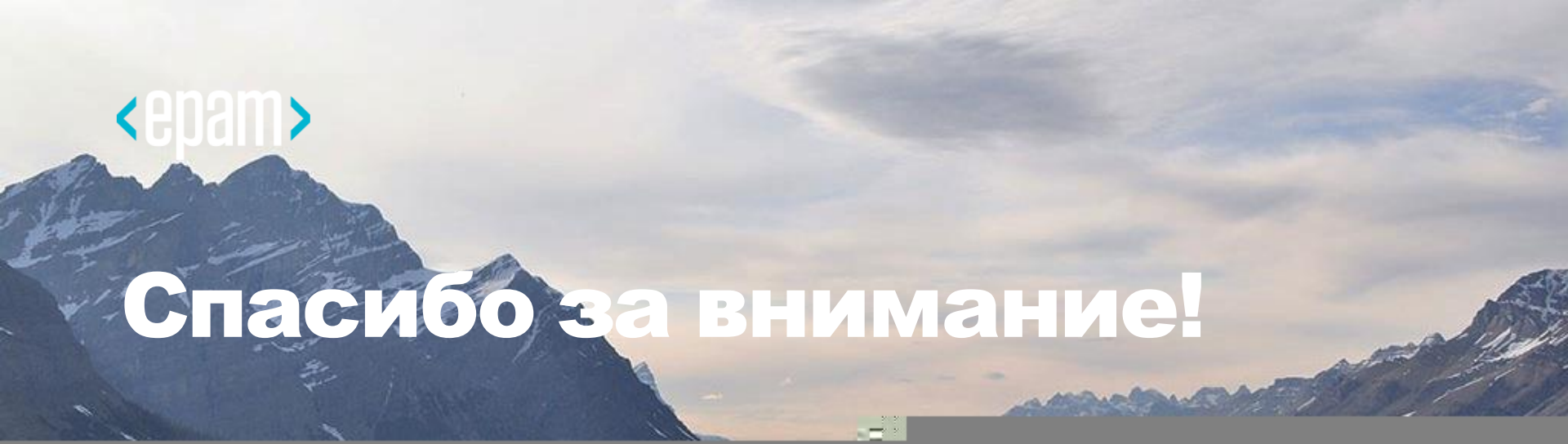
```
import logging.config

LOG_FORMAT = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
LOG_CONFIG = {
    "version": 1,
    "handlers": {
        "fileHandler": {
            "class": "logging.FileHandler", "formatter": "myFormatter",
            "filename": "some_logger2.log"
        }
    },
    "loggers": {"someLogger": {"handlers": ["fileHandler"], "level": "INFO"}},
    "formatters": {"myFormatter": {"format": LOG_FORMAT, "datefmt": "%Y.%m.%d %H:%M:%S"}}
}

logging.config.dictConfig(LOG_CONFIG)
logger = logging.getLogger("someLogger")
logger.info("Program started")
logger.info("Done!")
```

Домашнее задание

Пример использования конфигурационного файла

Задача.

Написать аналог unix утилиты **ls** на Python.
Ls - утилита Unix, которая печатает в стандартный вывод содержимое каталогов.
https://ru.wikipedia.org/wiki/Ls
Чем больше опций/ключей отражено, тем лучше.
Запрещается использовать вызов исходной утилиты в коде.



Спасибо за внимание!