

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет по лабораторной работе № 5
«Разработка приложений с интерфейсом командной строки»**

по дисциплине «Основы программной инженерии»

Выполнила:
Первых Дарья Александровна,
2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ

```
1  #!/usr/bin/env python3
2  #- coding: utf-8 -*-
3
4  import getopt, sys
5
6  full_cmd_argument = sys.argv
7
8  argument_list = full_cmd_argument[1:]
9  print(argument_list)
```

Рисунок 1 - Пример работы с модулем getopt

```
#!/usr/bin/env python3
# #- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.add_argument(
    "square", type=int, help="display a square of a given number"
)
parser.add_argument(
    "-v", "--verbose", action="store_true", help="increase output verbosity"
)
args = parser.parse_args()
```

Рисунок 2 - Пример работы с модулем argparse, программа, возводящая в квадрат значение позиционного аргумента и формирующая вывод в зависимости от аргумента опционального

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
subparsers = parser.add_subparsers(help='List of commands')

list_parser = subparsers.add_parser('list', help='list contents')
list_parser.add_argument('dirname', action='store', help='Directory to list')

create_parser = subparsers.add_parser('create', help='Create a directory')
create_parser.add_argument('dirname', action='store', help='new directory to create')

```

Рисунок 3 – Пример работы с использованием субпарсеров

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.parse_args()

```

Рисунок 4 – Пример простого разбора аргументов

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.add_argument("echo")

args = parser.parse_args()
print(args.echo)
```

Рисунок 5 – Пример работы с позиционными аргументами

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--verbosity", help="increase output verbosity")

args = parser.parse_args()
if args.verbosity:
    print("verbosity turned on")
```

Рисунок 6 – Пример работы с опциональными аргументами

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.add_argument(
    "--verbose",
    help="increase output verbosity",
    action="store_true"
)

args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

Рисунок 7 – Пример работы с булевой переменной

```
-# !/usr/bin/env python3
-# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.add_argument(
    "square",
    type=int,
    help="display a square of a given number"
)
parser.add_argument(
    "-v"
    "--verbose",
    action="store_true",
    help="increase output verbosity"
```

Рисунок 8 – Пример работы с короткими именами `-v` и `-verbosity`

```

parser = argparse.ArgumentParser()
parser.add_argument(
    "square",
    type=int,
    help="display a square of a given number"
)
parser.add_argument(
    "-v",
    "--verbosity",
    type=int,
    help="increase output verbosity"
)

args = parser.parse_args()
answer = args.square ** 2
if args.verbosity == 2:
    print("the square of {} equals {}".format(args.square, answer))
elif args.verbosity == 1:
    print("{}^2 == {}".format(args.square, answer))

```

Рисунок 9 – Пример совместной работы с позиционными и опциональными аргументами

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse

parser = argparse.ArgumentParser()
parser.add_argument(
    "square",
    type=int,
    help="display a square of a given number"
)
parser.add_argument(
    "-v",
    "--verbosity",
    type=int,
    choices=[0, 1, 2],
    help="increase output verbosity"
)

```

Рисунок 10 – Пример работы с опциональными аргументами с параметром + позиционным аргументов

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument(
    "square",
    type=int,
    help="display a square of a given number"
)
parser.add_argument(
    "-v",
    "--verbosity",
    action="count",
    help="increase output verbosity"
)

args = parser.parse_args()
answer = args.square ** 2
if args.verbosity == 2:
```

Рисунок 11 – Пример выбора значения из заранее определённого списка


```

parser = argparse.ArgumentParser()
parser.add_argument(
    "square",
    type=int,
    help="display a square of a given number"
)
parser.add_argument(
    "-v",
    "--verbosity",
    action="count",
    default=0,
    help="increase output verbosity"
)

args = parser.parse_args()
answer = args.square ** 2
if args.verbosity == 2:
    print("the square of {} equals {}".format(args.square, answer))
elif args.verbosity == 1:

```

Рисунок 12 – Пример подсчёта количества заданных аргументов `-v action='count'`

```

import argparse

parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count", default=0)

args = parser.parse_args()
answer = args.x ** args.y
if args.verbosity >= 2:
    print("{} to the power {} equals {}".format(args.x, args.y, answer))
elif args.verbosity >= 1:
    print("{}^{} == {}".format(args.x, args.y, answer))
else:
    print(answer)

```

Рисунок 13 – Пример добавления опции `default=0`, чтобы когда аргумент не задан, значение переменной было не `None`, а `0`

```

import argparse

parser = argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")

args = parser.parse_args()
answer = args.x ** args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print("{} to the power {} equals{}".format(args.x, args.y, answer))
else:
    print("{}^{} == {}".format(args.x, args.y, answer))

```

Рисунок 14 – Пример того, что опциональный аргумент может содержать параметр

```

import argparse

parser = argparse.ArgumentParser(
    description="calculate x to the power of y"
)
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")

args = parser.parse_args()
answer = args.x ** args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print("{} to the power {} equals{}".format(args.x, args.y, answer))

```

Рисунок 15 – Пример программы, которая возводит не в квадрат, а в указанную степень

```
PS C:\Users\podar\study\PyCharm Community Edition 2021.2.3\5LR> python zd.py add pr.json -n="apple" -s="five" -c="32"
PS C:\Users\podar\study\PyCharm Community Edition 2021.2.3\5LR> python zd.py display pr.json
+-----+-----+-----+-----+
| № | Наименование товара | Название магазина | Стоимость |
+-----+-----+-----+-----+
| 1 | apple | five | 32 |
+-----+-----+-----+-----+
```

Рисунок 16 – Ввод информации для идз

ВОПРОСЫ

1. В чем отличие терминала и консоли?

Терминал (от лат. terminus — граница) — устройство или ПО, выступающее посредником между человеком и вычислительной системой. Обычно данный термин используется, когда точка доступа к системе вынесена в отдельное физическое устройство и предоставляет свой пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых протоколов).

Консоль – компьютер с клавиатурой и монитором.

2. Что такое консольное приложение?

Консольное приложение console application — вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

3. Какие существуют средства языка программирования Python для построения приложений командной строки?

Python 3 поддерживает несколько различных способов обработки аргументов командной строки. Встроенный способ – использовать модуль sys. С точки зрения имен и использования, он имеет прямое отношение к библиотеке C (libc). Второй способ – это модуль getopt, который обрабатывает как короткие, так и длинные параметры, включая оценку значений параметров. Кроме того, существуют два других общих метода. Это модуль argparse, производный от модуля optparse, доступного до Python 2.7. Другой метод – использование модуля docopt, доступного на GitHub.

4. Какие особенности построение CLI с использованием модуля sys ?

Это базовый модуль, который с самого начала поставлялся с Python. Он использует подход, очень похожий на библиотеку C, с использованием argc и argv для доступа к аргументам. Модуль sys реализует аргументы командной строки в простой структуре списка с именем sys.argv. Каждый элемент списка представляет собой единственный аргумент. Первый элемент в списке sys.argv [0] – это имя скрипта Python. Остальные элементы списка, от sys.argv [1] до sys.argv [n], являются аргументами командной строки с 2 по n. В качестве разделителя между аргументами используется пробел. Значения аргументов, содержащие пробел, должны быть заключены в кавычки, чтобы

их правильно проанализировал `sys`. Эквивалент `argc` – это просто количество элементов в списке. Чтобы получить это значение, используйте оператор `len()`.

5. Какие особенности построение CLI с использованием модуля `getopt` ?

Как вы могли заметить ранее, модуль `sys` разбивает строку командной строки только на отдельные фасы. Модуль `getopt` в Python идет немного дальше и расширяет разделение входной строки проверкой параметров. Основанный на функции C `getopt`, он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений. На практике для правильной обработки входных данных требуется модуль `sys`. Для этого необходимо заранее загрузить как модуль `sys`, так и модуль `getopt`. Затем из списка входных параметров мы удаляем первый элемент списка (см. код ниже) и сохраняем оставшийся список аргументов командной строки в переменную с именем `arguments_list`.

```
# Include standard modules

import getopt, sys

# Get full command-line arguments

full_cmd_arguments = sys.argv

# Keep all but the first

argument_list = full_cmd_arguments[1:]

print(argument_list)
```

Аргументы в списке аргументов теперь можно анализировать с помощью метода `getopts()`. Но перед этим нам нужно сообщить `getopts()` о том, какие параметры допустимы. Они определены так:

```
short_options = "ho:v"

long_options = ["help", "output=", "verbose"]
```

Для метода `getopt()` необходимо настроить три параметра – список фактических аргументов из `argv`, а также допустимые короткие и длинные параметры.

Сам вызов метода хранится в инструкции `try - catch`, чтобы скрыть ошибки во время оценки. Исключение возникает, если обнаруживается аргумент, который не является частью списка, как определено ранее. Скрипт в Python выведет сообщение об ошибке на экран и выйдет с кодом ошибки 2.

```
try:

arguments, values = getopt.getopt(argument_list, short_options,

long_options)
```

```
except getopt.error as err:

# Output error, and return with an error code

print(str(err))

sys.exit(2)
```

Наконец, аргументы с соответствующими значениями сохраняются в двух переменных с именами `arguments` и `values`. Теперь вы можете легко оценить эти переменные в своем коде. Мы можем использовать цикл `for` для перебора списка распознанных аргументов, одна запись за другой.

```
# Evaluate given options

for current_argument, current_value in arguments:

if current_argument in ("-v", "--verbose"):

print("Enabling verbose mode")

elif current_argument in ("-h", "--help"):

print("Displaying help")

elif current_argument in ("-o", "--output"):

print(f"Enabling special output mode ({current_value})")
```

Ниже вы можете увидеть результат выполнения этого кода. Далее показано, как программа реагирует как на допустимые, так и на недопустимые программные аргументы:

```
$ python arguments-getopt.py -h

Displaying help

$ python arguments-getopt.py --help

Displaying help

$ python arguments-getopt.py --output=green --help -v

Enabling special output mode (green)

Displaying help

Enabling verbose mode

$ python arguments-getopt.py -verbose

option -e not recognized
```

Последний вызов нашей программы поначалу может показаться немного запутанным. Чтобы понять это, вам нужно знать, что сокращенные параметры (иногда также называемые флагами) могут использоваться вместе с одним тире. Это позволяет вашему инструменту легче воспринимать

множество вариантов.

6. Какие особенности построение CLI с использованием модуля argparse ?

Для начала рассмотрим, что интересного предлагает argparse :

- анализ аргументов `sys.argv` ;
- конвертирование строковых аргументов в объекты Вашей программы и работа с ними;
- форматирование и вывод информативных подсказок.

Одним из аргументов противников включения argparse в Python был довод о том, что в стандартных модулях и без этого содержится две библиотеки для семантической обработки (парсинга) параметров командной строки. Однако, как заявляют разработчики argparse , библиотеки getopt и optparse уступают argparse по нескольким причинам:

- обладая всей полнотой действий с обычными параметрами командной строки, они не умеют обрабатывать позиционные аргументы (positional arguments). Позиционные аргументы — это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример — программа `cp`, имеющая минимум 2 таких аргумента («`cp source destination`»).
- argparse дает на выходе более качественные сообщения о подсказке при минимуме затрат (в этом плане при работе с optparse часто можно наблюдать некоторую избыточность кода);
- argparse дает возможность программисту устанавливать для себя, какие символы являются параметрами, а какие нет. В отличие от него, optparse считает опции с синтаксисом наподобие "`-pf, -file, +rgb, /f` и т.п. «внутренне противоречивыми» и «не поддерживается optpars 'ом и никогда не будет»;
- argparse даст Вам возможность использовать несколько значений переменных у одного аргумента командной строки (nargs);
- argparse поддерживает субкоманды (subcommands). Это когда основной парсер отправляет к другому (субпарсеру), в зависимости от аргументов на входе.

Для начала работы с argparse необходимо задать парсер. Если действие (action) для данного аргумента не задано, то по умолчанию он будет сохраняться (store) в namespace, причем мы также можем указать тип этого аргумента (int, boolean и тд). Если имя возвращаемого аргумента (dest) задано, его значение будет сохранено в соответствующем атрибуте namespace.

В нашем случае:

```
>>> print(parser.parse_args(['-n', '3']))
```

```
Namespace(n='3')
```

```
>>> print(parser.parse_args([]))
Namespace(n=None)
>>> print(parser.parse_args(['-a', '3']))
error: unrecognized arguments: -a 3
```

Остановимся на действиях (actions). Они могут быть следующими:

`store`: возвращает в пространство имен значение (после необязательного приведения типа). Как уже говорилось, `store` — действие по умолчанию;

`store_const`: в основном используется для флагов. Либо вернет Вам значение, указанное в `const`, либо (если ничего не указано), `None`.

`store_true` / `store_false`: аналог `store_const` , но для булевых `True` и `False` ; `append`:

возвращает список путем добавления в него значений аргументов.

`append_const`: возвращение значения, определенного в спецификации аргумента, в список.

`count`: как следует из названия, считает, сколько раз встречается значение данного аргумента.

В зависимости от переданного в конструктор парсера аргумента `add_help` (булевого типа), будет определяться, включать или не включать в стандартный вывод по ключам `['-h', '--help']` сообщения о помощи. То же самое будет иметь место с аргументом `version` (строкового типа), ключи по умолчанию: `['-v', '--version']`. При запросе помощи или номера версии, дальнейшее выполнение прерывается.

```
parser = argparse.ArgumentParser(add_help=True, version='4.0')
```