Лабораторная работа №2
Модули в Python. Текстовые файлы

Цель: настоящей работы состоит в том, чтобы изучить Пользовательские функции и основы функционального программирования в Python

.

Задачи:

Выполнение практической работы предполагает решение ***следующих задач***:
1. Изучение процесса создания создания модуля.
2. Использование модулей
3. Повторная загрузка модулей.
4. Подготовка отчета, содержащего минимальный объем информации по каждому этапу выполнения работы.

Модули. Инструкции import и from. Повторная загрузка модуле

Последовательность выполнения работы :

Использован материал И.В. Мироновой, Программирование на языке Python. Часть 2: Учебное пособие - М.: Финансовый университет, департамент анализа данных, принятия решений и финансовых технологий, 2019. -51 с.

1. Модули

Модулем в языке Python называется файл с программой.

1. Чтобы воспользоваться модулем, нужно выполнить инструкцию **import**.

```
import math
x = math.sqrt(2)
```

Или так:

```
import math as mm  #указываем псевдоним
x = mm.sqrt(2)
```

Или так:

```
from math import sqrt, exp, log, sin   #только нужные имена
x=sqrt(2)
```

Получить информацию о содержимом модуля:

```
>>> dir(math)
```

Использование модулей

Модулем в языке Python называется файл с программой.

Чтобы воспользоваться модулем, нужно выполнить инструкцию import.

```
import math
x = math.sqrt(2)
```

Или так:

```
import math as mm  #указываем псевдоним
x = mm.sqrt(2)
```

Или так:

```
from math import sqrt, exp, log, sin   #только нужные имена
x=sqrt(2)
```

Получить информацию о содержимом модуля:

```
>>> dir(math)
```

Использование собственных модулей

Содержимое модуля ***mod_test.py***:

```
def f1():
    print("функция f1")

x = 123
```

Содержимое модуля, в котором используется модуль mod_test:

```
import mod_test as mm

mm.f1()
print(mm.x)
```

Оба файла размещаем в одной папке.

После подключения ***mod_test.py*** внутри папки будет создан каталог **__pycache__**с файлом ***mod_test.cpython-35.pyc***. Этот файл содержит скомпилированный байт-код исходного модуля.

Если его переименовать в ***mod_test.pyc***, то полученный файл можно использовать вместо файла ***mod_test.py***, например, при распространении программ.

Повторная загрузка модулей

Модуль загружается один раз при первой операции импорта.

Если в модуль вносились изменения, то его нужно перезагрузить. Иначе вы будете работать со старой версией модуля.

Для повторной загрузки модуля выполните команды:

```
>>> import imp
>>> reload(mm)
```

Или, если псевдоним не использовался,

```
>>> from imp import reload
>>> reload(mod_test)
```

Пути поиска модулей

Поиск подключаемого модуля выполняется в папках, указанных в списке sys.path модуля sys. Чтобы увидеть этот список, выполните

```
>>> import sys
>>> sys.path
```

При поиске список просматривается слева направо. Поиск прекращается после первого найденного модуля.

Из программы список sys.path можно изменить, используя его методы:

```
import sys
sys.path.append(r"C:\Мои модули") #в конец списка
sys.path.insert(0, r"C:\Мои модули") #в начало списка
```

2. Текстовые файлы

Текстовый файл - это файл, в котором хранится последовательность символов в определенной кодировке. Файл может быть разбит на строки с помощью специального символа «конец строки».

2.1. Открытие и закрытие файла

При открытии файла создается объект, с помощью которого выполняется работа с файлом.

```
f1 = open('myfile.txt', 'r')

f1 = open(r'D:\Файлы\myfile.txt')

f1 = open('D:\\Файлы\\myfile.txt')
```

Режимы доступа к текстовым файлам:

'r' - чтение из текстового файла. Если файл не существует, то выдается сообщение об ошибке (режим по умолчанию);

'w' - запись в текстовый файл. Если файл существует, то содержимое заменяется. Если файл не существует, то он создается;

'a' - дозапись в текстовый файл. Если файл существует, то данные дописываются в конец. Если файл не существует, то он создается.

Закрыть файл:

```
f1.close()
```

2.2. Чтение текстового файла

Метод ***read()*** позволяет прочесть из файла указанное количество символов.

Метод возвращает строку, состоящую из прочитанных символов.

Каждый следующий вызов **read()** начинает работу оттуда, где завершил работу предыдущий.

Если все символы прочитаны, то очередной вызов read( )возвратит пустую строку. Например, так можно прочитать весь файл по одному символу:

```
s = ''
s1 = f1.read(1)
while s1 != '':
    s = s + s1
    s1 = f1.read(1)
print(s)
```

Если количество символов не указать, будет возвращен весь текстовый файл одной строкой:

```
s = f1.read()
print(s)
```

Метод ***readline()*** позволяет прочитать указанное количество символов текущей строки. Если в строке символов меньше, чем указано, то будут прочитаны символы до конца строки.

Если количество символов не указать, будут прочитаны все символы с текущей позиции до конца строки.

Метод возвращает строку, состоящую из прочитанных символов, включая символ '\n'.

Каждый следующий вызов readline() начинает работу оттуда, где завершил работу предыдущий.

Если все символы прочитаны, то очередной вызов readline() возвратит пустую строку.

```
s = ''
s1 = f1.readline()
while s1 != '':
    s = s + s1
    s1 = f1.readline()
print(s)
```

Метод ***readlines( )*** читает текстовый файл в список, элементами которого являются строки файла. Символ '\n' включается в строку.

```
L = f1.readlines()
```

Перебор строк файла в цикле for:

```
for line in f1:
    print(line.rstrip('\n'))
```

В этом примере в прочитанных строках убирается символ конца строки.

2.3. Запись в текстовый файл

Файл должен быть открыт с ключом 'w' или 'a'.

Метод ***write()*** записывает строку в файл. Метод не вставляет символ '\n' в конце строки автоматически. Для перехода на новую строку символ нужно явно помещать в файл.

Метод ***writelines( )*** записывает элементы списка строк в файл.

Функция ***print()*** записывает информацию в файл, если ее необязательный параметр file равен ссылке на открытый файл.

```
f2 = open('результат.txt', 'a')
f2.write("строка1\nстрока2\n")
L = ['строка3\n', 'строка4\n']
f2.writelines(L)
print('строка5', end='', file = f2)
f2.close()
```

Требования к результатам выполнения работы

Условия успешной сдачи лабораторной работы:

1. Подготовлено пошаговое описание решений представленных задач.
2. Разработана программная реализация представленных задач.
3. Подготовлено краткое описание разработанного программного кода.
4. Программная реализация представленных примеров выложены в личный репозиторий на GitHub.

Литература

Копырин, А. С. Программирование на Python : учебное пособие / А. С. Копырин, Т. Л. Салова. — Сочи : СГУ, 2018. — 48 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/147665 (дата обращения: 15.07.2021). — Режим доступа: для авториз. пользователей.