Практическая работа

Анализ датасета. Функции. Строковые данные

Задание 1. Создание своих функций и их применение

1. Определение функции. Аргументы функции.
2. Вызов функции. (пример 1)
3. Функция как объект**.(пример 2)
4. Функция map()**.(пример 3)
5. Лямбда-функции**.(пример 4)
6. Функция filter()**.(пример 5)

Задание 2. Операции над строками

1. Строка - итерируемый объект. Индексация элементов строки. (пример 6)
2. Поиск подстроки в строке. Срез. (пример 8)
3. Методом find().(пример 9)
4. Метод count().(пример 10)
5. Методы lower() и upper().(пример 11)
6. Метод replace().(пример 12)
7. Регулярные выражения. Модуль re**

Задание 3. Анализ данных текстового файла.

1. Загрузить файл StudentsPerformance.csv
2. Преобразование данных файла в список (пример 12).
3. Преобразование данных файла в словарь (пример 14).
4. Общий алгоритм анализа данных.
5. Написать отчет.
6. Выполнить задание.

Пример 1

Функции

1. Написать функцию возведения в квадрат

```
# Это определение функции, которая возводит число в квадрат
def square(x):
    return x**2

# Это вызов функции. Мы возведём 5 в квадрат и положим 25 в переменную square_result
square_result = square(5)
print('square_result=', square_result)
```

2. Написать функцию, которая определяет стоимость товара в магазине, в котором всё по 60, где только яблоки стоят 30

```
# Определение стоимости в магазине всё по 60, где только яблоки стоят 30
```

```
def count_cost(product):
    if product == "apple":
        cost = 30
    else:
        cost = 60
    return cost

# Попросим функцию посчитать стоимость апельсина
orange_cost = count_cost("orange")
print('orange_cost=', orange_cost)
```

Пример 2

Функции как объект

В *Python* функции являются объектами, поэтому вы можете обращаться с ними, как и с любой другой переменной. Например, вы можете положить функцию **в другую переменную**

```
# Определим функцию, которая печатает "привет"
def say_hello():
    print("Hello")

# Мы можем положить её в другую переменную
greetings = say_hello

# И она сработает так же, как исходная say_hello
greetings()
# => Hello
```

Так же вы можете передавать одну функцию в другую **как аргумент**. Обратите внимание, что когда мы обращаемся с функцией, как с переменной, мы не пишем скобки после имени функции.

```
def apply_the_operation(operation, argument):
    print("I'm using", operation)
    return operation(argument)


def double_string(string):
    return string*2

#передаём функцию double_string в apply_the_operation
apply_the_operation(double_string, 'hello')
# => I'm using <function double_string at 0x7f4c34265
378>
# 'hellohello'
```

Пример 3

Пусть, например, у нас есть список имён, и мы хотим получить для каждого вежливую форму.

```
# => Определим функцию polite_name, которая делает вежливым одно имя
def polite_name(name):
    return 'Mr. ' + name

guests = ["Boris", "Ivan", "Bob"]
guest_iterator = map(polite_name, guests)  # здесь мы применили polite_name к каждому имени
list(guest_iterator)  # вывод
# => ['Mr. Boris', 'Mr. Ivan', 'Mr. Bob']
```

Итак, функция **map** служит для преобразования списка (второй аргумент **map**) в новый, согласно заданному правилу (функции – первый аргумент **map**). Функция преобразования задается как показано выше, кроме того,

можно использовать **map** со встроенными функциями. Вот, например, способ преобразовать строки в числа

```
num_strings = ["10", "1", "4.2", "0.73"]

list(map(float, num_strings))
# => [10.0, 1.0, 4.2, 0.73]
```

И еще одной возможностью для задания функции в **map** является встроенная **lambda**-функция.

Пример 4

создадим простую функцию с помощью **lambda** и положим её в переменную

```
#func
func = lambda x, y: x + y

# после этого мы можем использовать func как обычную функцию
func(1, 2)
# => 3
func('a', 'b')
# => 'ab'

# мы даже можем не давать функции имя, а сразу вызывать
(lambda x: x**2)(8)
```

```
# => 64
```

Немного по-другому работает функция **filter**, хотя имеет похожий синтаксис. Здесь первым параметром тоже значится функция, в которую последовательно подставляются элементы из списка (второго аргумента). Функция (первый аргумент) должна выдавать логический результат. В случае **True** элемент их исходного списка переносится в новый, иначе — нет!

Пример 5
filter

```
nums = [1, 20, 30, 33, 16, 5]

# оставим числа меньшее 30
list(filter(lambda x: x < 30, nums))
# => [1, 20, 16, 5]

# оставим только нечётные числа
list(filter(lambda x: x % 2 == 1, nums))
# => [1, 33, 5]
```

Пример 6
Перебор всех букв строки

```
string = 'Вы - самый крутой студент в МГСУ'
for letter in string:
    print(letter, end = '')
```

Организация символьных строк во многом похожа на списки. Указав в квадратных скобках индекс, можно получить доступ к отдельному символу. Можно также применять срезы (от и до). А вот менять содержание самой строки, например, заменить строчные буквы на заглавные уже нельзя, однако можно получить из старой строки другую (измененную) уже в виде новой переменной.

Пример 7
Напишите программу, которая "зашифровывает" текст, хранящийся в переменной *proverb*, меняя местами символы (в т.ч. и знаки препинания), стоящие на нечётных и чётных позициях. Результат шифрования нужно вывести на экран с помощью функции print
Пример: после шифровки начало фразы должно выглядеть так: рПгоарммсиыт

```
proverb = 'Программисты - это устройства, преобразующие кофеин в код.'
```

```
new_proverb = ''
length = len(proverb)
for i in range(0,length,2):
    li = proverb[i:i+2]
    st = li[0]
    st = li[1] + st
    new_proverb = new_proverb+st
print(new_proverb)
```

Обратите внимание, преобразование строки сохранено в новую переменную. А операция + применительно к строкам означает сцепление.

Пример 8

Посчитать, названия скольких городов из этого списка состоят из двух и более частей. Обычно, если название города состоит из нескольких частей, то эти части разделены либо пробелами, либо — дефисами.

```
city_list = ['Москва', 'Санкт-Петербург',
'Новосибирск',
'Екатеринбург', 'Нижний Новгород',
'Казань',
'Челябинск', 'Омск', 'Самара', 'Ростов-
на-Дону']
counter = 0
for city in city_list:
    if ' ' in city or '-' in city:
        counter += 1
print('Число городов со сложными названиями -
{}'.format(counter))
```

Оператор `if ' ' in city or '-' in city:`

Проверяет вхождение пробела или дефиса в строку

Пример 9

Известно, что адрес домена в адресе электронной почты следует сразу после символа "@", нам достаточно узнать номер позиции, которую занимает знак "@" и извлечь из строки все символы, следующие за "@".

```
email = 'VeryBigBoss@mgsu.ru'
pos = email.find('@')
domain = email[pos+1:]
print('domain- ',domain)
```

Пример 10

Найдем количество упоминаний имени "Грека" (в разных падежах) в известной скороговорке:

```
tongue_twister = 'Ехал Грека через реку, видит Грека - в реке рак. Сунул Грека руку в реку, рак за руку Греку - цап!'
```

Поскольку при изменении имени Грека по падежам меняется окончание, нам нужно посчитать количество вхождений основы слова (Грек_) в текст скороговорки:

```
counter = tongue_twister.count('Грек')
print(counter)
```

Пример 11

Преобразование строк (**lower** – в нижний регистр, **upper** – в верхний)

```
answer = input('Введите ДА или НЕТ: ')
if answer.lower() == 'да':
    print('Вы ответили "ДА"! Я счастлив!')
```

Пример 12

Замена запятой на точку:

```
new_numbers = []
for number in '3,14 2,71 6,02 11,22 123,987'.split():
    new_numbers.append(float(number.replace(',', '.')))
print(new_numbers)
```

Функция split разделяет единую строку на элементы, формируя из них список. По умолчанию разделителем является пробел. Но в качестве аргумента-разделителя в **split** можно задать и любой другой символ.

Следующие примеры используют файл 'StudentsPerformance.csv'

Вот несколько его первых строчек:

"gender","race/ethnicity","parental level of education","lunch","test preparation course","math score","reading score","writing score"

"female","group B","bachelor's degree","standard","none","72","72","74"

"female","group C","some college","standard","completed","69","90","88"

"female","group B","master's degree","standard","none","90","95","93"

"male","group A","associate's degree","free/reduced","none","47","57","44"

"male","group C","some college","standard","none","76","78","75"

…

CSV – это формат представления табличных данных (comma-separated-value). Строка в файле соответствует строке в таблице, а колонки разделяются запятыми. Сам файл необходимо разместить в той же папке, где и хранится

программный код, а можно, например, просто снабдить его полным путем при открытии, например, так:

```
f = open('C:\\Student\\BigData\\StudentsPerformance.csv')
```

Обратите внимание, знак \ в Python является управляющим и для его применения в строке надо удваивать!

Пример 13

```
f = open('StudentsPerformance.csv')
for line in f:
        print(line)
f.close()
```

!!! Открытый файл после использования ОБЯЗАТЕЛЬНО надо закрыть, чтобы система не заблокировала к нему доступ!

```
f.close()
```

Считаем девочек и мальчиков

```
f = open('StudentsPerformance.csv')

females = 0
males = 0
for line in f:
    info = line.split(',')
    gender = info[0]
    if gender == '"female"':
        females += 1
    elif gender == '"male"':
        males +=1
print('Мальчиков: {}, девочек: {}'.format(males, females))
f.close()
```

Пример 14

Необходимо получать информацию о количестве студентов, относящихся к той или иной этнической группе

```
students = {}
f = open('StudentsPerformance.csv')
for line in f:
    info = line.split(',')
    if info[0] == '"gender"':
```

```
            continue
    # первая строчка (шапка таблицы) не считается !
        else:
            ethnicity = info[1][1:-1]
    # срез [1:-1] исключает " в которые заключен
показатель
            if ethnicity in students:
                students[ethnicity] += 1
            else:
                students[ethnicity] = 1
    print(students)
```

{'group B': 190, 'group C': 319, 'group A': 89, 'group D': 262, 'group E': 140}