Урок 4

Цели занятия:
* Закрепить материал по работе с коллекциями
* Изучить новый материал(Функции)

Основы работы со словарями
Создадим словарь с двумя парами ключ-значение
```
results = {"pass": 0, "fail": 0}
```
Добавим третью пару ключ-значение
```
results["withdrawal"] = 1
```
Обновим значения двух ключей в словаре
```
results["pass"] = 3
results["fail"] = results["fail"] + 1
````
 Выведем значения, ассоциированные с ключами fail, pass и withdrawal соответственно
 ```
print(results["fail"])
print(results["pass"])
print(results["withdrawal"])
```
При запуске данной программы будет создан словарь results с двумя ключами: pass и fail. Значения, ассоциированные с этими ключами, мы сделали нулевыми. Далее словарь пополняется еще одной парой с ключом withdrawal и значением 1. После этого значение, ассоциированное с ключом pass, меняется на 3 при помощи оператора присваивания. В следующей строке происходит обращение к значению ключа fail, к которому прибавляется единица, и новое значение сохраняется в паре с тем же ключом fail, заменяя предыдущее значение. При отображении значений первой будет выведена единица (значение, ассоциированное с ключом fail), затем тройка, соответствующая ключу pass, и снова единица – на этот раз представляющая значение ключа withdrawal

С помощью команды in, мы можем знать, присутствует ли необходимое значение в значениях словаря
```
if x in d.values():
    print("В словаре d есть как минимум одно значение", x)
else:
    print("В словаре d не присутствует значение", x)
 ```

Цикл for может быть использован для осуществления итераций по ключам словаря. На каждой итерации ключ словаря сохраняется во внутренней переменной цикла k.
```
# Создаем словарь
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}
# Выводим на экран все ключи и значения в отформатированном виде
for k in constants:
    print(f"Значение, ассоциированное с ключом {k}: {constants[k]}")
 ```
Сначала создается словарь с именем constants, в котором хранятся некоторые математические константы в виде пар ключ-значение. Цикл for открывает итерации по словарю. На первой итерации ключ pi сохраняется во

временную переменную цикла k, и выполняется тело цикла с выводом на экран первой пары ключ-значение из списка. После этого цикл запускается вновь, и в переменную k уже попадает ключ e. В теле цикла на экран выводится соответствующая строка со значением 2,71. На заключительном проходе по словарю мы узнаем, что квадратный корень из двух равен 1,41.

Циклом for также можно воспользоваться для осуществления итераций по значениям словаря:

```
# Создаем словарь
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}
# Рассчитаем сумму значений в словаре
total = 0
for v in constants.values():
    total = total + v
# Выводим результат
print("Сумма значений составляет", total)
```

Иногда бывает удобнее обращаться к словарям при помощи циклов while, а не for:

```
# Считаем, сколько раз пользователь ввел каждое значение
counts = {}
# Цикл, пока количество уникальных значений в словаре не достигнет пяти
while len(counts) < 5:
    s = input("Введите строку: ")
    # Если в словаре уже есть такой ключ, увеличиваем count на 1.
    # Иначе добавляем пару к словарю со значением count, равным 1.
    if s in counts:
        counts[s] = counts[s] + 1
    else:
        counts[s] = 1
# Выводим все строки и счетчики
for k in counts:
    print(f'{k} появилась в словаре {counts[k]} раз')
```

Программа начинается с создания пустого словаря. После этого запускается цикл while с проверкой на количество пар ключ-значение в словаре. Поскольку словарь пока пуст, условное выражение вернет True, и будет запущено тело цикла. На каждой итерации пользователь будет вводить строку с клавиатуры. После этого при помощи оператора in будет определено, присутствует ли введенный пользователем ключ в списке. Если да, то ассоциированный с этим ключом счетчик (значение) будет увеличен на единицу. В противном случае словарь пополнится новой парой ключ-значение. Цикл будет продолжаться, пока пользователь не введет пять уникальных строк с клавиатуры. После этого на экран будут выведены все пары ключ-значение из списка.

Задачи:

Задача 1: Код Цезаря

Одним из первых в истории примеров шифрования считаются закодированные послания Юлия Цезаря. Римскому полководцу необходимо было посылать письменные приказы своим генералам, но он не желал, чтобы в случае чего их прочитали недруги. В результате он стал шифровать свои послания довольно простым методом, который впоследствии стали называть кодом Цезаря.
Идея шифрования была совершенно тривиальной и заключалась в циклическом сдвиге букв на три позиции. В итоге буква A превращалась в D, B – в E, C – в F и т. д. Последние три буквы алфавита переносились на начало. Таким

образом, буква X становилась A, Y – B, а Z – C. Цифры и другие символы не подвергались шифрованию.
Напишите программу, реализующую код Цезаря. Позвольте пользователю ввести фразу и количество символов для сдвига, после чего выведите результирующее сообщение. Убедитесь в том, что ваша программа шифрует как строчные, так и прописные буквы. Также должна быть возможность указывать отрицательный сдвиг, чтобы можно было использовать вашу программу для расшифровки фраз.

>Функция ord преобразует символ в целочисленную позицию в таблице ASCII. Функция chr возвращает символ в таблице ASCII по позиции, переданной в качестве аргумента.

```
# Запрашиваем у пользователя сообщение и сдвиг
message = input("Введите сообщение: ")
shift = int(input("Введите сдвиг: "))
# Обрабатываем каждый символ для создания зашифрованного сообщения
new_message = ""
for ch in message:
    if ch >= "a" and ch <= "z":
    # Обрабатываем букву в нижнем регистре, определяя ее позицию
    # в алфавите (0-25), вычисляя новую позицию и добавляя букву в
сообщение
        pos = ord(ch) - ord("a")
        pos = (pos + shift) % 26
        new_char = chr(pos + ord("a"))
        new_message = new_message + new_char
    elif ch >= "A" and ch <= "Z":
    # Обрабатываем букву в верхнем регистре, определяя ее позицию
    # в алфавите (0-25), вычисляя новую позицию и добавляя букву в
сообщение
        pos = ord(ch) - ord("A")
        pos = (pos + shift) % 26
        new_char = chr(pos + ord("A"))
        new_message = new_message + new_char
    else:
    # Если это не буква, просто сохраняем ее в сообщении
        new_message = new_message + ch
# Отображаем полученное сообщение
print(f"Новое сообщение {new_message}")
```

Задача 2: Отрицательные, положительные и нули

Напишите программу, запрашивающую у пользователя целые числа, пока он не оставит строку ввода пустой. После окончания ввода на экран должны быть выведены сначала все отрицательные числа, которые были введены, затем нулевые и только после этого положительные. Внутри каждой группы числа должны отображаться в той последовательности, в которой были введены пользователем. Например, если он ввел следующие числа: 3, -4, 1, 0, -1, 0 и -2, вывод должен казаться таким: -4, -1, -2, 0, 0, 3 и 1. Каждое значение должно отображаться на новой строке

```
# Создаем три списка для хранения отрицательных, нулевых и положительных
значений
negatives = []
zeros = []
positives = []
# Запрашиваем числа у пользователя, помещая их в соответствующие списки
```

```
line = input("Введите целое число (Enter для окончания ввода): ")
while line != "":
    num = int(line)
    if num < 0:
        negatives.append(num)
    elif num > 0:
        positives.append(num)
    else:
        zeros.append(num)
    # Запрашиваем следующее число у пользователя
    line = input("Введите целое число (Enter для окончания ввода): ")
# Выводим сначала отрицательные числа, затем нули и после этого
положительные
print("Введенные числа: ")
for n in negatives:
    print(n)
for n in zeros:
    print(n)
for n in positives:
    print(n)
```

Задача 3: Случайные лотерейные номера

Для выигрыша главного приза необходимо, чтобы шесть номеров на лотерейном билете совпали с шестью числами, выпавшими случайным образом в диапазоне от 1 до 49 во время очередного тиража. Напишите программу, которая будет случайным образом подбирать шесть номеров для вашего билета. Убедитесь в том, что среди этих чисел не будет дубликатов. Выведите номера билетов на экран по возрастанию.

```
from random import randrange
MIN_NUM = 1
MAX_NUM = 49
NUM_NUMS = 6
# Используем список для хранения номеров лотерейного билета
ticket_nums = []
# Генерируем NUM_NUMS случайных, но уникальных значений
for i in range(NUM_NUMS):
    # Генерируем номер, которого еще нет в списке
    rand = randrange(MIN_NUM, MAX_NUM + 1)
    while rand in ticket_nums:
        rand = randrange(MIN_NUM, MAX_NUM + 1)
    # Добавляем номер к билету
    ticket_nums.append(rand)
# Сортируем номера по возрастанию и отображаем их
ticket_nums.sort()
print("Номера вашего билета: ", end="")
for n in ticket_nums:
    print(n, end=" ")
print()
```

Задача 4: Эрудит

В известной игре Эрудит каждой букве соответствует определенное количество очков. Общая сумма очков, которую получает игрок, составивший это слово, складывается из очков за каждую букву, входящую в его состав. Чем более употребимой является буква в языке, тем меньше очков

начисляется за ее использование. В таблице ниже приведены все соответствия букв и очков из английской версии игры.

![Таблица Эрудит](/res/scrable.png "Таблица очков")

>Для переноса длинных строк используйте знак '\\'

```
# Создаем словарь с соответствием букв и очков за них
points = {"A": 1, "B": 3, "C": 3, "D": 2, "E": 1, "F": 4, "G": 2, "H": 4,
"I": 1, \
         "J": 2, "K": 5, "L": 1, "M": 3, "N": 1, "O": 1, "P": 3, "Q": 10,
"R": 1, \
         "S": 1, "T": 1, "U": 1, "V": 4, "W": 4, "X": 8, "Y": 4, "Z": 10}
# Запрашиваем у пользователя слово
word = input("Введите слово: ")
# Считаем количество очков
uppercase = word.upper()
score = 0
for ch in uppercase:
    score = score + points[ch]
# Выводим результат
print(f"word оценивается в {score} очков.")
```

Функции
>Функция – это предназначенный для выполнения конкретной задачи изолированный блок кода, который можно повторно использовать. Функции делают код программы модульным и более компактным

Для создании функции обычно применяется инструкция **def**. Функция сама по себе не может вызываться, поэтому для вызова функции мы может использовать конструкцию <имя функции>()
```
def Say():
    print('Hello')

Say()
```
```
def area_sq():
    side = 5
    area = side * side
    print(area)

area_sq()
```
Функции могут принимать значения. Для этого в скобочках указывается переменная, в которую будет записано значение, а при вызове функции, указывается само значение:
```
side_sq = int(input())
def area_sq(side):
    area = side * side
    print(area)

area_sq(side_sq)
```
Функция может не только принимать значения, но и возвращать их.Это требуется, когда расчитанное функцией значение требуется вернуть обратно в программу, длядальнейших действий. Переменные созданные изначально в

функции являются локальными.Переменные же в самой программе, являются глобальными.
>Локальные переменные – это переменные, которые объявлены внутри функции и имеют область видимости внутри функции.
Для возврата значения используется оператор **return**
```
def area_sq():
    number = 5
    result = number ** 2
    return result

result = area_sq()
print(result)
```
Функция может принимать и возвращать необходимое количество значений.
```
import random

def lottery():
    tickets = [1, 11, 22, 34, 56, 18, 7]
    ticket1 = random.choice(tickets)
    tickets.remove(ticket1)
    ticket2 = random.choice(tickets)
    return ticket1, ticket2

a, b = lottery()
print(a, b)
```
Возвращающая и принимающая функция, на примере нахождения факториала.
>Факториал - произведение последовательных натуральных чисел от 1 до N включительно.
```
def factorial(num):
    result = 1
    for i in range(1, num + 1):
        result *= i
    return result

print(factorial(5))
```
Для указания переменной, как глобальной используется **global**.
Переменная указанная, как глобальная внутри функции, изменяет все переменные с таких же именем вне функции.
Если у нас вложенная функция, то для изменения значения переменной внешней функции,используется оператор nonlocal:
```
def test():
    global number
    number = 5
    number2 = 1
    def test2():
        nonlocal number2
        number = 3
        number2 = 5
    print(f'{number2=}')
    test2()
    print(f'{number2=}')

print(number)
test()
print(number)
```

```
Функция может возвращать не только значение конкретной переменной, но и любое значение:
```
```
def season(n):
    if 1 <= n <= 2 or n == 12:
        return 'Зима'

print(season(12))
```
```
Пример функции, проверяющей, является ли число простым:
```
```
def isPrime(n):
    for i in range(2, n // 2):
        if n % i == 0:
            return False
    return True

print(isPrime(71))
```
```
Существует еще один тип функции - lambda функции. Это однострочная безымянная функция, которая может, как принимать значения, так и возвращать:
```
```
a = lambda num: num * 2
print(a(5))
```
```
Лямбда функция может принимать и более одного значения:
```
```
a = lambda num1, num2: num1 * num2
print(a(5, 5))
```
```
```