Структура «Словарь» в Python. Использование словарей при решении заданий ЕГЭ

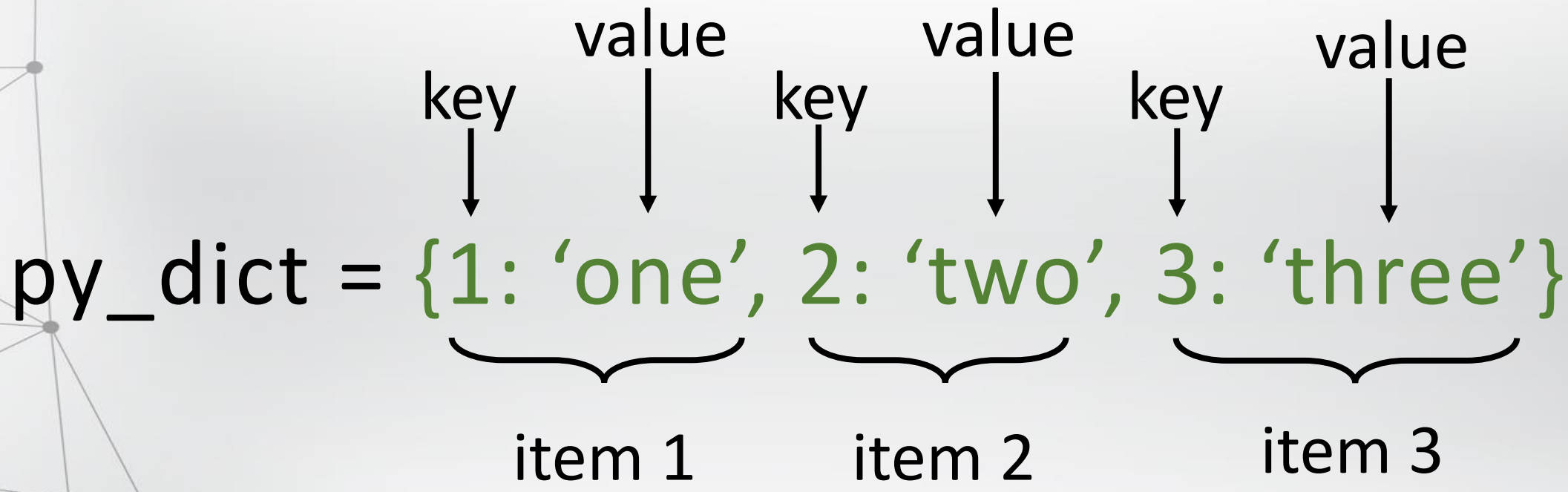
Логинова Светлана Викторовна
учитель информатики МАОУ СОШ № 1
МО Ленинградский район

Словари

- **Коллекции в Python:**
- **списки** – изменяемые коллекции элементов, индексируемые;
- **строки** – неизменяемые коллекции символов, индексируемые;
- **кортежи** – неизменяемые коллекции элементов, индексируемые;
- **множества** – изменяемые коллекции уникальных элементов, неиндексируемые.

Словари – изменяемые коллекции элементов с произвольными индексами – ключами. Ключом может быть произвольный неизменяемый тип данных: целые и действительные числа, строки, кортежи. Ключом в словаре не может быть множество, но может быть элемент типа frozenset: специальный тип данных, являющийся аналогом типа set, который нельзя изменять после создания. Значением элемента словаря может быть любой тип данных, в том числе и изменяемый.

Словари


key
value
key
value
key
value
py_dict = {1: ‘one’, 2: ‘two’, 3: ‘three’}
item 1
item 2
item 3

Словари

```
languages = { 'Python': 'Гвидо ван Россум’,
              'C#': 'Андерс Хейлсберг’,
              'Java': 'Джеймс Гослинг’,
              'C++': 'Бьёрн Страуструп’ }
```

Когда нужно использовать словари

Словари нужно использовать в следующих случаях:

- Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключами являются объекты, а значениями — их количество.
- Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря Num[‘January’] = 1; Num[‘February’] = 2; ….
- Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект.
- Если нужен обычный массив, но при этом максимальное значение индекса элемента очень велико, или будут использоваться не все возможные индексы (так называемый “разреженный массив”), то можно использовать ассоциативный массив для экономии памяти.

Создание словаря

```
Capitals = {'Russia': 'Moscow', 'Belarus': ‘Belarus', 'USA': 'Washington'}
Capitals = dict(Russia = 'Moscow', Belarus = Belarus', USA = 'Washington')
Capitals = dict([("Russia", "Moscow"), ("Belarus ", "Belarus"), ("USA", "Washington")])
Capitals = dict(zip(["Russia", "Belarus", "USA"], ["Moscow", "Belarus", "Washington"]))
```

Словари

languages = { 'Python': 'Гвидо ван Россум’,
'C#': 'Андерс Хейлсберг’,
'Java': 'Джеймс Гослинг’,
'C++': 'Бьёрн Страуструп’ }

К значению словаря можно обратиться по его ключу.

```
print('Создателем языка C# является', languages['C#’])
```

Вывод:

Создателем языка C# является Андерс Хейлсберг

Вывод словаря

```
languages = { 'Python': 'Гвидо ван Россум’,
              'C#': 'Андерс Хейлсберг’,
              'Java': 'Джеймс Гослинг’,
              'C++': 'Бьёрн Страуструп’ }
```

```
print(languages)
```

Вывод:

```
{'Python': 'Гвидо ван Россум', 'C#': 'Андерс Хейлсберг', 'Java': 'Джеймс Гослинг',
'C++': 'Бьёрн Страуструп'}
```

Методы словарей

Метод **items()** – возвращает словарные пары ключ: значение, как соответствующие им кортежи

languages = { 'Python': 'Гвидо ван Россум’,
'C#': 'Андерс Хейлсберг’,
'Java': 'Джеймс Гослинг’,
'C++': 'Бьёрн Страуструп’ }

```
print(languages.items())
```

Вывод:

dict_items([('Python', 'Гвидо ван Россум'), ('C#', 'Андерс Хейлсберг'), ('Java', 'Джеймс Гослинг'), ('C++', 'Бьёрн Страуструп')])

Методы словарей

Метод **keys()** – возвращает список ключей словаря

languages = { 'Python': 'Гвидо ван Россум’,
'C#': 'Андерс Хейлсберг’,
'Java': 'Джеймс Гослинг’,
'C++': 'Бьёрн Страуструп’ }

```
print(languages.keys())
```

Вывод:

dict_keys(['Python', 'C#', 'Java', 'C++'])

Методы словарей

Метод **values()** – возвращает список значений словаря

languages = { 'Python': 'Гвидо ван Россум’,
'C#': 'Андерс Хейлсберг’,
'Java': 'Джеймс Гослинг’,
'C++': 'Бьёрн Страуструп’ }

```
print(languages.values())
```

Вывод:

dict_values(['Гвидо ван Россум', 'Андерс Хейлсберг', 'Джеймс Гослинг', 'Бьёрн Страуструп'])

Методы словарей

Получение значения элемента по ключу, записывается так же, как и для списков: **A[key]**. Если элемента с заданным ключом не существует в словаре, то возникает исключение **KeyError**.

Другой способ определения значения по ключу — метод **get()**: **A.get(key)**. Если элемента с ключом get нет в словаре, то возвращается значение **None**. В форме записи с двумя аргументами **A.get(key, val)** метод возвращает значение **val**, если элемент с ключом **key** отсутствует в словаре.

```
print(languages.get('C#'))
print(languages.get('VisualBasic'))
print(languages.get('VisualBasic', 'Microsoft'))
```

Вывод:

Андерс Хейлсберг

None

Microsoft

Перебор элементов словаря

- Организация перебора элементов по ключу:

```
for key in A:
    print(key, A[key])
```

- Проверить принадлежность элемента словарю можно операциями in и not in, как и для множеств.
- Для добавления нового элемента в словарь нужно просто присвоить ему какое-то значение: A[key] = value.
- Для удаления элемента из словаря можно использовать операцию del A[key] (операция возбуждает исключение KeyError, если такого ключа в словаре нет.

Функция zip()

Функция **zip()** в Python создает итератор, который объединяет элементы из нескольких источников данных. Эта функция работает со списками, кортежами, множествами и словарями для создания списков или кортежей, включающих все эти данные.

```
id = [1, 2, 3, 4]
people = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
record = zip(id, people)
print(record)
print(list(record))
```

Вывод:

<zip object at 0x7f266a707d80>

[(1, 'Elon Mask'), (2, 'Tim Cook'), (3, 'Bill Gates'), (4, 'Yang Zhou')]

Функция zip()

```
# создаём словарь, используя функцию «dict»
id = [1, 2, 3, 4]
people = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
people_dict = dict(zip(id, people))
print(people_dict)
```

Вывод:

{1: 'Elon Mask', 2: 'Tim Cook', 3: 'Bill Gates', 4: 'Yang Zhou'}

Функция zip()

```
# транспонирование матрицы
matrix = [[1, 2, 3], [1, 2, 3]]
matrix_T = [list(i) for i in zip(*matrix)]
print(matrix_T)
```

Вывод:

[[1, 1], [2, 2], [3, 3]]

Операторы упаковки, распаковки словарей

Оператор ** используется для упаковки и распаковки словарей. При использовании перед словарем во время вызова функции оператор ** распаковывает пары ключ-значение словаря в аргументы ключевых слов, которые могут быть переданы в функцию:

```
def print_details(name, age):
    print(f"Name: {name}")
    print(f"Age: {age}")


details = {"name": "John", "age": 30}
print_details(**details)
```

Вывод:

Name: John

Age: 30

Задание 2

№ 8547 (Уровень: Базовый)

(В. Рыбальченко) Логическая функция F задаётся следующим выражением $(\neg a \vee \neg b) \wedge (c \to \neg a) \wedge (t \wedge \neg a \vee c \wedge \neg b)$. На рисунке приведён частично заполненный фрагмент таблицы истинности функции F, содержащий неповторяющиеся строки.

?	?	?	?	F
1	0	0	1	0
1	1	0	1	1
0	0	0	1	0
1	0	0	0	1

```
from itertools import *

def f(a,b,c,t):
    return ((not a) or (not b) ) and (c <= (not a)) and
(t and (not a) or c and (not b))
t = [(1,0,0,1),(1,1,0,1),(0,0,0,1),(1,0,0,0)]
for p in permutations('abct'):
    if [f(**dict(zip(p,r))) for r in t] == [0,1,0,1]:
        print(*p, sep = '')
```

№ 7604 Досрочная волна 2023 (Уровень: Базовый)

Миша заполнял таблицу истинности логической функции $F = \neg(y \wedge \neg x) \wedge \neg(x \equiv z) \wedge w$, но успел заполнить лишь фрагмент из трёх различных её строк, даже не указав, какому столбцу таблицы соответствует каждая из переменных w, x, y, z.

				F
0	0		1	1
0	1	0	1	1
		0		1

```
from itertools import *

def f(x,y,w,z):
    return (not(y and (not x))) and (x != z) and w

for a1,a2,a3,a4 in product([0,1], repeat = 4):
    t = [(0,0,a1,1),(0,1,0,1),(a2,a3,0,a4)]
    if len(t)==len(set(t)):
        for p in permutations('xywz'):
            if [f(**dict(zip(p,r))) for r in t] == [1,1,1]:
                print(*p,sep='')
```

№ 6283 (Уровень: Сложный)

(Г. Симаков) Логическая функция F задаётся выражением $\neg(\neg(x \to \neg w) \wedge z) \wedge \neg(w \to z) \wedge (x \to \neg z)$. На рисунке приведён заполненный фрагмент таблицы истинности функции F, содержащий неповторяющиеся наборы аргументов

?	?	?	?	F
1	0		0	1
1	0			0
	1		1	0

Определите, сколько существует различных способов расстановки переменных w, x, y, z, подходящих для данной таблицы истинности?

```
from itertools import *
def f(x,y,z,w):
     return (not ((not (x <= (not w))) and z) ) and (not (w <= z)) and (x <= (not
z))
ans = set()
for a1,a2,a3,a4,a5 in product([0,1], repeat = 5):
    t = [(1,0,a1,0),(1,0,a2,a3),(a4,1,a5,1)]
    if len(t) == len(set(t)):
        for p in permutations('xyzw'):
            if [f(**dict(zip(p,r))) for r in t]==[1,0,0]:
                ans.add(p)
print(len(ans))
```

Задание 24

№ 855 (Уровень: Средний)

Текстовый файл состоит не более чем из 10^6 заглавных латинских букв. Определите символ, который чаще всего встречается в файле между буквами X и Z, так что X стоит слева от него, а Z – справа. В ответе запишите сначала этот символ, а потом сразу (без разделителя) сколько раз он встретился между буквами X и Z. Например, в тексте XBZCXXZXBZXDZDD между буквами X и Z два раза стоит B, по одному разу – X и D. Для этого текста ответом будет B2.

```
s = open('24_855.txt').readline()
d ={}
for i in range(len(s)-2):
    if s[i] == 'X' and s[i+2]=='Z':
        d[s[i+1]] = d.get(s[i+1],0) + 1
sm = max(d,key = d.get)
print(sm,d[sm],sep = '')
```

№ 856 (Уровень: Средний)

Текстовый файл состоит не более чем из 10^6 заглавных латинских букв. Определите символ, который чаще всего встречается в файле сразу после буквы A. В ответе запишите сначала этот символ, а потом сразу (без разделителя) сколько раз он встретился после буквы A. Например, в тексте ABCAABADDD после буквы A два раза стоит B, по одному разу – A и D. Для этого текста ответом будет B2.

```
s = open('24_856.txt').readline()
d ={}
for i in range(len(s)-1):
    if s[i] == 'A':
        d[s[i+1]] = d.get(s[i+1],0) + 1
sm = max(d,key = d.get)
print(sm,d[sm],sep = '')
```

№ 1148 (Уровень: Сложный)

Текстовый файл состоит не более чем из 10^6 символов и содержит только заглавные буквы латинского алфавита (ABC…Z). Текст разбит на строки различной длины. Необходимо найти строку, содержащую наибольшее количество букв Q (если таких строк несколько, надо взять ту, которая в файле встретилась позже). Определите, какая буква встречается в этой строке реже всего (но присутствует!). Если таких букв несколько, надо взять ту, которая стоит раньше в алфавите. Запишите в ответе эту букву, а затем – сколько раз она встречается во всем файле.

Пример. Исходный файл:

ZZQAQB

QAVQAB

BAQTUB

В этом примере в первой и второй строках по две букву Q, в третьей – одна. Берём вторую строку, т.к. она стоит в файле позже. В этой строке реже других встречаются буквы V и B (по одному разу), выбираем букву B, т. к. она раньше стоит в алфавите. В ответе для этого примера надо записать B4, так как во всех строках файла буква B встречается 4 раза.

```
f = open('24_1148.txt')
d = {}
m = ms = 0
s = [x.strip() for x in f]
for i in range(len(s)):
    ds = {}
    for j in s[i]: ds[j] = ds.get(j,0)+1
    m = max(m,ds['Q'])
    if ds['Q']==m: ms = i
    d[i] = d.get(i,ds)
simbol=min(d[ms],key=d[ms].get)
sm = 0
for k in d:
    sm += d[k][simbol]

print(simbol, sm,sep='')
```