Урок 3

Цели урока:
1. *Закрепить прошедшие темы*
2. *Изучить коллекции*

***
Программа на закрепление

Написать консольную игру пятнашки. Условия:
* Игровое поле состоит из 16 ячеек.
* Поле заполняется числами от 1 до 15(включительно) в случайном порядке
* Одна ячейка поля остается пустая
* Пользователь может передвинуть число в пустое место
* Пользователь выигрываает, когда все ячейки начиная с первой
заполнены в порядке возрастания.Последняя ячейка пустая

```
import random
import os
import time
import pyfiglet
from progress.bar import IncrementalBar

print(pyfiglet.figlet_format("TAG GAME"))
print('Добро пожаловать в игру пятнашки')
print('Правила: соберите все костяшки в порядке возрастания')
input('Нажмите Enter для начала игры...')

mylist = [10, 22, 35, 44, 60, 69, 78, 100]
bar = IncrementalBar('Загрузка: ', max=len(mylist))

for item in mylist:
    bar.next()
    time.sleep(random.uniform(0, 0.3))
bar.finish()

if os.name == 'nt':
    os.system('cls')
else:
    print('Консоль не очищена')
number_list = [i for i in range(1, 16)]
# for i in range(1, 16):
#     number_list.append(i)
number_list.append(' ')
result_list = list(zip(*[iter(number_list)] * 4))
for i in range(len(result_list)):
    result_list[i] = list(result_list[i])
random.shuffle(number_list)
area = list(zip(*[iter(number_list)] * 4))
for i in range(len(area)):
    area[i] = list(area[i])
col_width = max(len(str(num)) for row in area for num in row) + 2

while result_list != area:
    os.system('cls')
    for row in area:
        print(''.join(str(num).ljust(col_width) for num in row))
    row1 = int(input('Введите строку, откуда вы хотите переместить
элемент: ')) - 1
```

```
    column1 = int(input('Введите столбец, откуда вы хотите переместить
элемент: ')) - 1
    row2 = int(input('Введите строку, куда вы хотите переместить элемент:
')) - 1
    column2 = int(input('Введите столбец, куда вы хотите переместить
элемент: ')) - 1
    if area[row2][column2] == ' ':
        area[row1][column1], area[row2][column2] = area[row2][column2],
area[row1][column1]
    else:
        print('Ячейка занята')
        continue

print('Поздравляю! Вы победили')


```
Создание виртуального окружения и активация:
```
python -m venv vevn
cd venv/Scripts
.\activate
```


Для установки библиотек:
```
pip install название
```
Создание текстового файла с указанием всех установленных библиотек:

```
pip freeze > requirements.txt
```
Установка библиотек из списка requirements:
```
pip install -r requirements.txt
```

Коллекции
Коллекции - это типы данных контейнера которые можно использовать для хранения данных. Коллекции могут хранить списки, наборы, кортежи и словари. Каждый из этих типов данных имеет свои собственные характеристики.

Тип данных	Изменяемость	Индексированность	Уникальность	Создание
Список	да	да	нет	list()
Кортеж	нет	да	нет	tuple()
Множество	да	нет	да	set()
Словарь	да	нет	да*	dict()

Множества
Множество в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке.

Создание множества:
```
a = set()  # создание пустого множества
a = set('привет')  # создание множества из строки
a = {'a', 'b', 'c', 'd'}

```

##Операции со множествами:

* *len(s)* - число элементов в множестве (размер множества).
```
print(len(a))
```
* *x in s* - принадлежит ли x множеству s.
```
print(b in a)
```
* *set.isdisjoint(other)* - истина, если set и other не имеют общих элементов.
```
print(a.isdisjoint(b))
```
* *set == other* - все элементы set принадлежат other, все элементы other принадлежат set.
```
print(a == b)
```
* *set.issubset(other) или set <= other* - все элементы set принадлежат other.
```
print(a.issubset(b))
```
* *set.issuperset(other) или set >= other* - аналогично.

* *set.union(other, ...) или set | other | ...* - объединение нескольких множеств.
```
print(a.union(b))
```
* *set.intersection(other, ...) или set & other & ...* - пересечение.
```
print(a.intersection(b))
```
* *set.difference(other, ...) или set - other - ...* - множество из всех элементов set, не принадлежащие ни одному из other.
```
print(a.difference(b))
```
* *set.symmetric_difference(other) или set ^ other* - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
```
print(a.symmetric_difference(b))
```
* *set.copy()* - копия множества.
```
b = a.copy()
print(b is a)
```
* *set.update(other, ...); set |= other | ...* - объединение.

```
a.update(b)
print(a)
```
* *set.intersection_update(other, ...); set &= other & ...* - пересечение.
```
a.intersection_update(b)
print(a)
```
* *set.difference_update(other, ...); set -= other | ...* - вычитание.
```
a.difference_update(b)
print(a)
```
* *set.symmetric_difference_update(other); set ^= other* - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
```
a.symmetric_difference_update(b)
print(a)
```
* *set.add(elem)* - добавляет элемент в множество.
```
a.add('e')
print(a)
```
* *set.remove(elem)* - удаляет элемент из множества. KeyError, если такого элемента не существует.
```
a.remove('a')
print(a)
```
* *set.discard(elem)* - удаляет элемент, если он находится в множестве.
```
a.discard('f')
print(a)
```
* *set.pop()* - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
```
a.pop()
print(a)
```
* *set.clear()* - очистка множества.
```
a.clear()
print(a)
```
Неизменяемые множества(frozenset)
Единственное отличие set от frozenset заключается в том, что set - изменяемый тип данных, а frozenset - нет
Создание неизменяемого множества:
```
a = set('привет')
a = frozenset(a)
# или
a = frozenset('привет')
```

Словарь

Словарь — неупорядоченная структура данных, которая позволяет хранить пары «ключ — значение».

Создание словаря:
```
a = {} # создание пустого словаря
a = dict.fromkeys(['b', 'c'])
a = dict.fromkeys(['b', 'c'], 100)
a = {'b': 1, 'c': 2}
a = dict(b='1', c='2')
a = dict([(1, 3), (5, 2)])

```

Операции со словарями:
* *dict.clear()* - очищает словарь.
```
a.clear()
print(a)
```
* *dict.copy()* - возвращает копию словаря.
```
b = a.copy()
print(b is a)
```
* *dict.fromkeys(seq[, value])* - создает словарь с ключами из seq и значением value (по умолчанию None).
```
a = dict.fromkeys(['b', 'c'])
```
* *dict.get(key[, default])* - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
```
print(a.get('c'))
```
* *dict.items()* - возвращает пары (ключ, значение).
```
print(a.items())
```
* *dict.keys()* - возвращает ключи в словаре.
```
print(a.keys())
```
* *dict.pop(key[, default])* - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
```
print(a.pop('b'))
```
* *dict.popitem()* - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.
```
print(a.popitem())
```
* *dict.setdefault(key[, default])* - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).
```
print(a.setdefault('e'))
```

* *dict.update([other])* - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
```
a.update({'e':5})
```
* *dict.values()* - возвращает значения в словаре.
```
print(a.values())
```