Конспект

Структуры данных

Множества точно полезных структур данных.

Предлагаю начать с множества, что это такое.

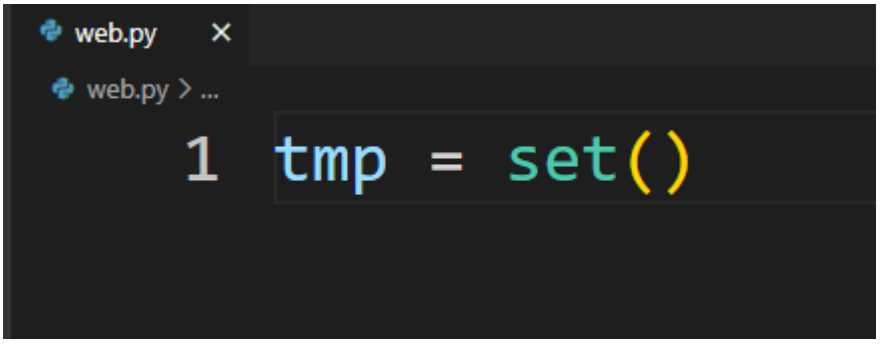
Множества – это неупорядоченные наборы простых объектов. Они необходимы тогда, когда присутствие объекта в наборе важнее порядка или того, сколько раз данный объект там встречается. Используя множества, можно осуществлять проверку принадлежности, определять, является ли данное множество подмножеством другого множества, находить множеств и так далее.

То есть коротко - Множество в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке. если мы попытаемся много раз добавить число 3 оно добавится 1 раз.

При том множеству не особо интересно в каком порядке мы будет добавлять элементы, они просто будут там храниться.

Мы сможем проверять есть ли эти элементы в множестве и еще пара операций которые мы рассмотрим далее.

Чтобы объявить пустое множество достаточно сделать так:



Если же мы хотим добавить в множество элементы то:

```
3 tmp2= {2, 3, 3, 4}
4 print(tmp2)
```

Мы попытались вывести 3 - два раза, что произошло? Правильно 3 у нас записалась один раз.
То есть мы видим, что если объект уже был значит он не добавится второй раз, потому что записываются индивидуальные значения.

Рассмотрим **основные методы работы с множеством.**

Для добавление элементов используется команда **add**

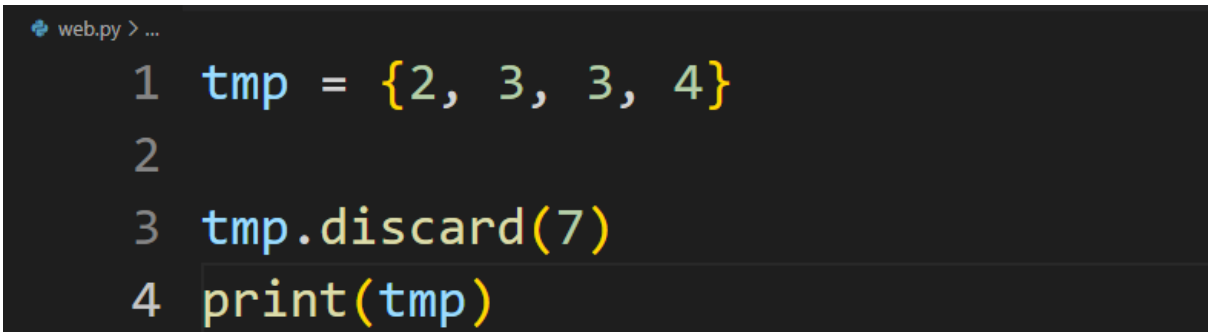
```
tmp = {2, 3, 3, 4}

tmp.add(7)    # добавить элемент в мно
print(tmp)
```

Теперь удаление, если мы хотим удалить какое то значение то можно использовать **remove**, но она будет выдавать ошибку если такого значения не существует.

```
1 tmp = {2, 3, 3, 4}
2
3 tmp.remove(7)
4 print(tmp)
```

Поэтому мы будем использовать команду **discard.**



Поэтому во множествах я рекомендую забыть про remove и использовать discard.

Давайте рассмотрим какую-нибудь практическую задачу на множества.
Представим что у Вас есть компания и вы раздаете промокоды и вам нужно определять какой промокод уже был использован, а какой нет.

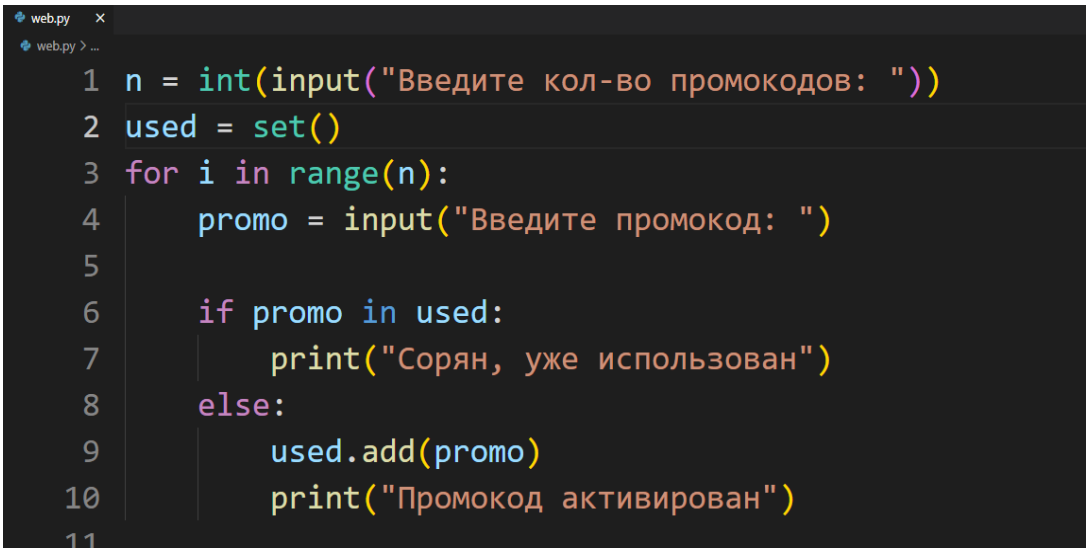
Пускай вводится их количество. Дал рить что “сорян промокод уже не действителен”.

Вводим кол-во промокод.
добавим множество для добавление в него использованных промокодов
С помощью цикла обработаем каждый промокод отдельно.
добавляем ввод промокода
далее проверяем
Чтобы проверить наличие промокода в множестве можем воспользоваться следующей конструкцией.

```
if promo in used:
```

то есть мы проверили, присутствует ли promo в множестве used

если промокод не был использован мы добавляем его в множество.

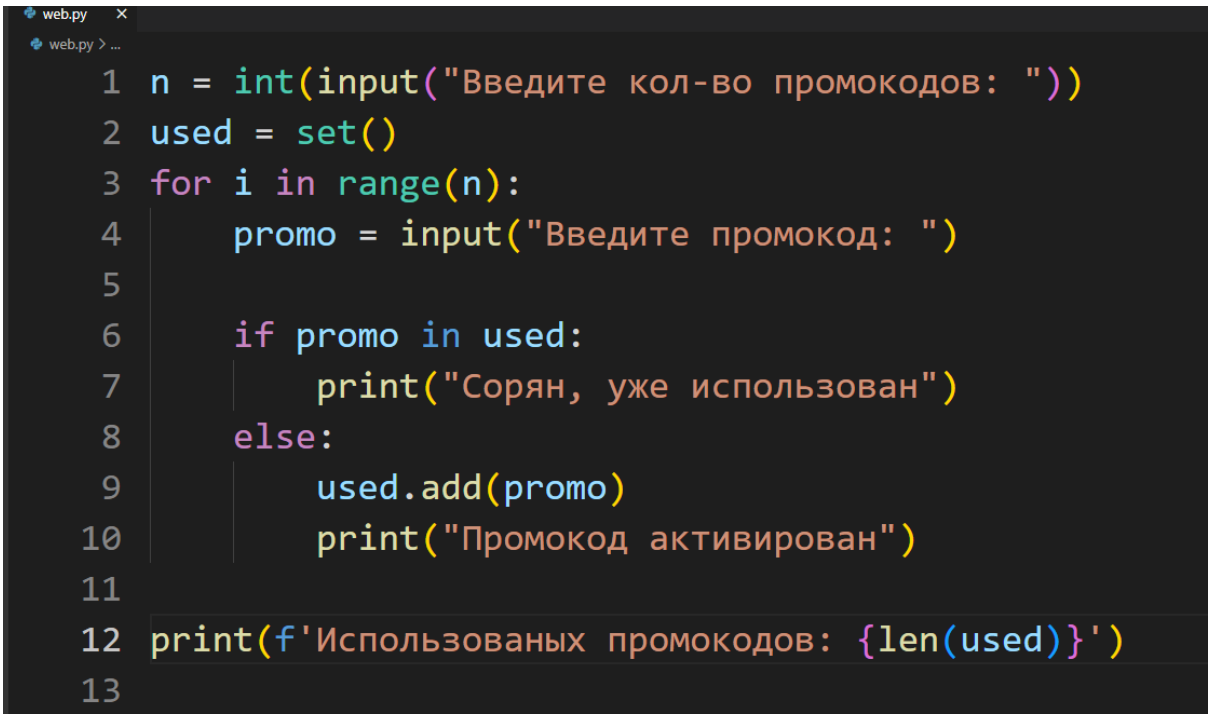


```python
n = int(input("Введите кол-во промокодов: "))
used = set()
for i in range(n):
    promo = input("Введите промокод: ")

    if promo in used:
        print("Сорян, уже использован")
    else:
        used.add(promo)
        print("Промокод активирован")
```

Давайте попробуем чуть усложнить задачку и под конец сделать так чтобы мы понимали сколько промокодов было использовано. 67(одинаковые мы не будем использовать).

Так же как и со списками мы можем использовать функцию len.



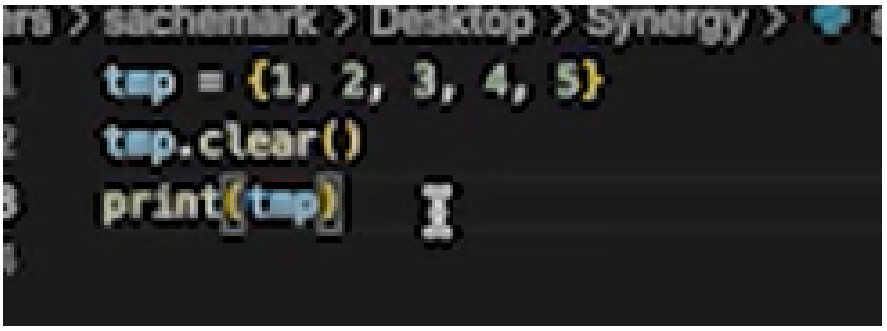
```python
n = int(input("Введите кол-во промокодов: "))
used = set()
for i in range(n):
    promo = input("Введите промокод: ")

    if promo in used:
        print("Сорян, уже использован")
    else:
        used.add(promo)
        print("Промокод активирован")

print(f'Использованых промокодов: {len(used)}')
```

Идем далее.

Чтобы очистить множество можем использовать команду которая используется не только во множествах, но и например в списках. {}



Рассмотрим ряд функций которые позволяют взаимодействовать setам друг с другом.
Сделаем это сразу на примере практической задачи.

У нас было две компании и в этих компаниях работали люди со своими именами.

В 1 компании допустим были 3 человека - андрей катя петя.

В 2 компании были 4 человека - андрей женя маша петя.

И при объединении этих компаний решили уволить сотрудников с одинаковыми именами.

В итоге в объединенной компании останутся работники с разными именами.

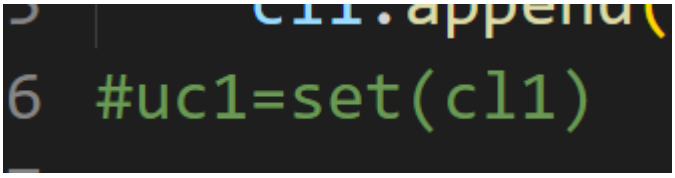
1. количество сотрудников в первой компании
2. список имен людей с первой компании
3. с помощью цикла введем имена сотрудников
4. ввод имен
5. добавляем сотрудника в множество
6. аналогично делаем со второй компанией

```python
c1 = int(input())
cl1=()
for i in range(c1):
    name = input()
    cl1.append(name)
#uc1=set(cl1)

c2 = int(input())
cl2=()
for i in range(c2):
    name = input()
    cl2.append(name)
```

Мы получили список имен обеих компаний, как же нам теперь решить нашу задачу?

Разберемся как создавать множества из списка (закомментить вторую компанию.



То есть мы получим значения нашего списка и преобразуем их в множества, повторяющиеся имена будут удалены. (показать через принт).

Теперь сделаем то же самое со вторым списком.

```python
c1 = int(input())
cl1=[]
for i in range(c1):
    name = input()
    cl1.append(name)
uc1=set(cl1)

c2 = int(input())
cl2=[]
for i in range(c2):
    name = input()
    cl2.append(name)
uc2=set(cl2)
```

Теперь наша задача как-то объединить два множества.

В первой компании у нас имена - uc1 уникальны, так же и во второй.
Так вот, мы сделали два множества, каждое из них отвечает за свою компанию.
Для их объединения используется замечательная функция **union.**

```
c1 = int(input())
cl1 = []
for i in range(c1):
    name = input()
    cl1.append(name)
uc1 = set(cl1)

c2 = int(input())
cl2 = []
for i in range(c2):
    name = input()
    cl2.append(name)
uc2 = set(cl2)

print(uc1.union(uc2))
```

Мы пишем первое множество ставим точку пишем юнион и передаем туда второе множество.
после вывода - Получается каждый сотрудник компании теперь уникален.

Запомнили - union функция которая делает объединение множеств.

Еще есть функция которая выводит

```
for i in range(c2):
    name = input()
    cl2.append(name)
uc2=set(cl2)

print(uc1.intersection(uc2))
```

пересечение, то есть выводит имена которые будут и в первой и во второй компании.

Теперь поговорим о важной теме про изменяемость и неизменяемость.

Так вот обычное множество является изменяемым

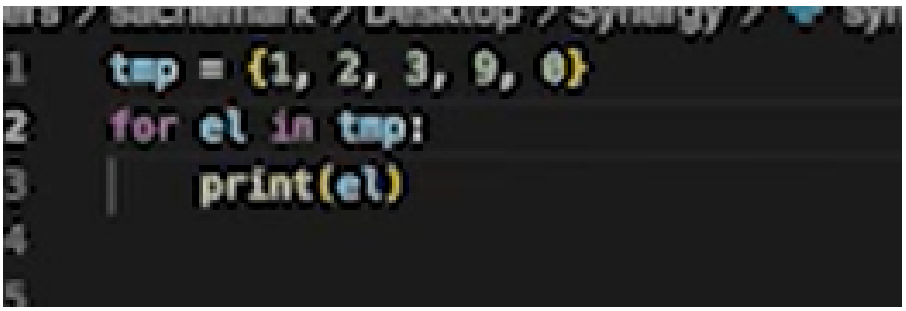
```
s1 = set()
s2 = s1
s2.add(7)
print(s1)
```

Так же есть неизменяемое множество, оно называется **FROZENSET**.

Они достаточно похожи с обычным set, но они неизменяемы.
К ним нельзя применять модифицирующие операции, к примеру add.

чтобы объявить его:

Рассмотрим еще как вообще можно пройтись по нашему множеству.



```
1  tmp = {1, 2, 3, 9, 0}
2  for el in tmp:
3      print(el)
4
```

обратите внимание что выводятся они в другом порядке, потому что set у абсолютно безразлично в каком они порядке хранятся.

Словари

Так же есть еще и словари, зачем они вообще нужны?

Словарь это набор ключей и значений, то есть мы можем к каждому элементу обратиться по ключу.

Представим что у нас есть банк, в банке есть какие-то ячейки.
И на каждой ячейке написан уникальный ключ.
С помощью ключа человек может открыть ячейку.

Допустим приходит семен в банк и говорит - у меня есть такой то ключ, работник банка понимает что у него вот такой ключ, он подходит к ячейке, на которой этот ключ написан, Открывает, ячейку и выдает то что у семена там лежит.

Там нет никаких индексов, то есть мы не можем сказать какая у нас ячейка имеет индекс ноль или какая имеет индекс 3.

Однако мы можем обращаться с ними по ключам.

Как их объявить:

```
1 bank = {'anton': 10, 'dima': 20, 'petya': 4}
```

Через запятую будет прописывать пары - Ключ + Значение ключа. То есть у нас есть ключ Антон и ему соответствует значение 10.

Как же теперь обратиться к значению по ключу?

Это немного похоже на списки, только вместо индекса мы используем наш ключ.

```
1 bank = {'anton': 10, 'dima': 20, 'pe
2
3 print(bank['anton'])
```

Также мы можем обращаться по этому ключу и менять значения.
Потому что он является изменяемым объектом.

```
bank = {'anton': 10, 'dima': 20, 'petya': 4}
bank['anton'] = 159
print(bank)
```

Также важно уточнить что каждый ключ в словаре должен быть уникален.

Если мы попробуем написать еще раз ключ антон и значение к нему 15ть, а затем вывести.

```
bank = {'anton': 10, 'dima': 20, 'petya': 4, 'anton': 15}
print(bank)
```

То мы заметим что Антон у нас всего лишь один, более того будет использован тот Антон который в самом конце.

Давайте попробуем решить практическую задачу.

Пускай у нас есть банк в который приходит икс запросов, каждый запрос бывает двух типов:

Либо запрос на открытие новой ячейки с каким то ключом.
Либо мы хотим положить деньги в ячейку с каким то ключом.
И в конце нам нужно вывести что вообще получилось.
То есть какие есть ячейки сколько в них лежит денег.

Давайте посмотрим как объявить пустой словарь:

```
bank = dict()
```

То есть наш банк это дикт - как раз таки словарь. Если мы в круглых скобках ничего не передаем он будет как раз таки пустой.

Отлично, далее мы знаем что нам придет какое-то количество запросов которые придут к нашему банку:

```
bank = dict()
n = int(input())
```

Теперь с помощью цикла поочередно будем обрабатывать все запросы.

```
for i in range(n):
```

Для начала пускай запрос кодируется типом, если это:
create - значит мы хотим открыть ячейку,
если это add значит мы хотим добавить какое то количество денег на нашу ячейку.

Изначально сделаем ввод типа нашего запроса:

```
bank = dict()
n = int(input())

for i in range(n):
    req = input()
```

Теперь у нас есть два варианта:

если **req** = create значит что нам ведется еще ключ ячейки которую мы хотим открыть, пускай это будет K

```
bank = dict()
n = int(input())

for i in range(n):
    req = input(" Введите запрос (create или add): ")

    if req == "create":
        k = input("Введите ключ с которым хотите открыть
```

Чтобы открыть ячейку мы можем просто взять и сказать что

```
if req == "create":
    k = input("Введите ключ с которым хотит
    bank[k] = 0
```

Банк от K = 0, потому что при открытии нашей ячейки денег там нет.

Иначе если запрос add, то мы должны будет добавить какое-то количество денег.
Но чтобы понимать в какую ячейку и сколько денег добавить, мы должны это значение ввести.
Вводим ключ по которому будет обращаться к ячейке.
И вводим amount - количество денег которое хотим добавить:

```
elif req == "add":
    k = input("Введите ключ ячейки для пополнения: ")
    amount = int(input("Сумма для пополнения: "))
```

Теперь добавим проверку на то был ли такой ключ в нашем словаре.
То есть задача проверить присутствует ли какой то ключ в ключах нашего словаря.

```
    if k in bank.keys():
        bank[k] += amount
    else:
        print(" Ключ не найден ")
```

Функция keys возвращает набор ключей которые находятся в словаре, то есть мы проверяем присутствует ли К в наборе ключей keys банка.

Если это так то мы добавляем туда amount денег
Иначе выводим “Ключ не найден”

И выводим что у нас в итоге получилось.

```
bank = dict()
n = int(input("Сколько запросов пришло банку?: "))

for i in range(n):
    req = input("Введите запрос (create или add): ")

    if req == "create":
        k = input("Введите ключ с которым хотите открыть новую ячейку: ")
        bank[k] = 0
        print("Успешно!")

    elif req == "add":
        k = input("Введите ключ ячейки для пополнения: ")
        amount = int(input("Сумма для пополнения: "))

        if k in bank.keys():
            bank[k] += amount
            print("Успешно!")
        else:
            print("Ключ не найден")

    else:
        print("Извините, запрос неверный")
print(f'Итог на конец дня: {bank}')
```

Кортежи

Кортежи в Python представляют собой неизменяемые упорядоченные коллекции объектов. Они очень похожи на списки, но имеют несколько ключевых отличий:

Кортежи создаются с использованием круглых скобок, в то время как списки создаются с помощью квадратных скобок:

```
my_tuple = (1, 2, 3)
```

Кортежи являются неизменяемыми, то есть их элементы не могут быть изменены после создания:

```
my_tuple[0] = 4 # Ошибка!
```

Кортежи могут содержать элементы разных типов данных:

```
my_tuple = (1, "Hello", True)
```

Доступ к элементам кортежа осуществляется по индексу, подобно доступу к элементам списка:

```
print(my_tuple[0]) # Output: 1
```

Методы, применимые к спискам (например, append() или remove()), не могут быть применены к кортежам, так как они неизменяемые:

```
my_tuple.append(4) # Ошибка!
my_tuple.remove(1) # Ошибка!
```

Кортежи могут быть использованы в качестве ключей в словарях, так как они являются неизменяемыми и поэтому хешируемыми:

```
my_dict = {(1, 2): "value"}
```

Методы кортежей включают функции count() и index(), а также операторы сцепления (+) и повторения (*):

```
my_tuple = (1, 2, 3, 2)
print(my_tuple.count(2)) # Output: 2
print(my_tuple.index(3)) # Output: 2
print(my_tuple + (4, 5)) # Output: (1, 2, 3, 2, 4, 5)
print(my_tuple * 2) # Output: (1, 2, 3, 2, 1, 2, 3, 2)
```

Кортежи особенно полезны в случаях, когда необходимо сохранить неизменность коллекции данных. Они также могут быть использованы для возврата нескольких значений из функции, так как кортежи могут быть разбиты на отдельные переменные с помощью оператора распаковки (tuple unpacking).