Функции.

**Метод** — это функция, которая применяется к определенному объекту, используя символ точку:

объект.метод()

Посмотрим как происходит добавление элементов в массив:

# допустим, у нас есть список, содержащий первые 4 буквы латинского алфавита

letters = ['a', 'b', 'c', 'd']

# с помощью метода append() мы добавляем еще один элемент в список

letters.append('e')

**print**(letters)

# ['a', 'b', 'c', 'd', 'e']

Как же получить последний элемент списка, если его точная длина заранее неизвестна? Существует два способа. Первый — использование длины списка. Её можно узнать с помощью встроенной функции len(), которая возвращает длину любого итерируемого объекта. К ним относят строки, списки, кортежи и объекты некоторых других типов данных.

С ними подробнее познакомимся в следующих модулях, а пока что можно получать радость от использования такой функции:

**print**(len(letters))

# 5

Как и ожидалось, длина списка равна 5. Тогда доступ к последнему элементу можно получить, если уменьшить эту длину на 1:

**print**(letters[len(letters)-**1**])

# e

letters.append('f') # добавляем еще одну букву

letters.append('g') # и еще одну

**print**(letters[len(letters)-**1**])

# g

Изменение структуры списка может происходить не только путем ее увеличения (добавления новых объектов), но и удаления уже существующих. Для этого можно использовать метод **pop()**:

**print**(letters)

# ['a', 'b', 'c', 'd', 'e', 'f', 'g']

letters.pop() # вызов метода без аргументов удаляет последний элемент списка

**print**(letters)

# ['a', 'b', 'c', 'd', 'e', 'f']

# был удален последний элемент

letters.pop(**0**) # или можно удалить элемент по его индексу

**print**(letters)

# ['b', 'c', 'd', 'e', 'f']

# был удален нулевой элемент

letters.pop(**3**) # и не обязательно удалять из начала или конца списка

**print**(letters)

# ['b', 'c', 'd', 'f']

# был удален элемент с индексом 3

С помощью **срезов** можно получать сразу несколько элементов списка.

|  |  |  |
| --- | --- | --- |
| **Срез** | **Как работает?** | **Пример** |
| [:] | Возвращает элементы полностью | [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’] |
| [2:] | Возвращает элементы списка, начиная с элемента индекса 2 и до конца списка | [‘c’, ‘d’, ‘e’, ‘f’, ‘g’] |
| [:3] | Возвращает элементы списка от его начала до элемента с индексом 3, **не включая** его | [‘a’, ‘b’, ‘c’] |
| [1:4] | Объединяя предыдущие два способа можно получить элементы из середины. В данном случае начиная с индекса 1 до индекса 4, не включительно. Иными словами, элементы с индексами 1,2 и 3 | [‘b’, ‘c’, ‘d’] |
| [::2] | Задает шаг, через который извлекаются элементы | [‘a’, ‘c’, ‘e’, ‘g’] |
| [::-1] | Используя отрицательный шаг, можно развернуть массив | [‘g’, ‘f’, ‘e’, ‘d’, ‘c’, ‘b’, ‘a’] |

L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# ["б", "в", 1]

Подсказка

  неверно



L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# ["а", 1, 4]

Подсказка

  верно



L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# [1, "в", "б", "а"]

Подсказка

  верно



L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# [4, 3, 2]

Подсказка

  верно



Возможности языка позволяют выполнить определенные действия для каждого элемента списка. Такую операцию можно проделать с помощью функцию **map()**:

map(function, list)

Первый аргумент map() — функция, которую нужно применить к каждому элементу списка, а сам список — второй аргумент. Возвращаемое значение этой функции — объект map, который можно преобразовать, например, обратно в список.

Рассмотрим пример:

# имеем список с числами с плавающей точкой

L = [**3.3**, **4.4**, **5.5**, **6.6**]

# печатаем сам объект map

**print**(map(round, L)) # к каждому элементу применяем функцию округления

# <map object at 0x7fd7e86eb6a0>

# и результат его преобразования в список

**print**(list(map(round, L)))

# [3, 4, 6, 7]

L = ['3.3', '4.4', '5.5', '6.6']

**print** (list (map ( ??? , L)))

  неверно



Однако, пользуясь функциями split() и map() можно выполнить нужное преобразование:

string = input("Введите числа через пробел:")

list\_of\_strings = string.split() # список строковых представлений чисел

list\_of\_numbers = list(map(int, list\_of\_strings)) # cписок чисел

**print**(sum(list\_of\_numbers[::**3**])) # sum() вычисляет сумму элементов списка

*Напишите программу, которая на вход получает последовательность чисел, а выводит модифицированный список:*

* 1. *Первое и последнее числа последовательности должны поменяться местами.*
  2. *В конец списка нужно добавить сумму всех чисел.*

*Посмотреть ответ для самопроверки*

*# все операции - деление строки по пробелам, преобразование к числам*

*# и приведение объекта map к типу список, можно делать в одной строке*

*L = list(map(float, input().split()))*

*# обмениваем первое и последнее число*

*# с помощью множественного присваивания*

*L[****0****], L[-****1****] = L[-****1****], L[****0****]*

*# находим сумму и добавляем ее в конец списка*

*L.append(sum(L))*

***print****(L)*

*Задание 2.5.7*

*1 point possible (graded)*

*Чему будет равен последний элемент списка, полученного в результате работы алгоритма из последней задачи, если на вход подается последовательность чисел:*

***1******1******2******3******5******8******13******21******34******55***

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

***Словари***

*Использование списков открывает много возможностей, но и имеет свои ограничения. Каждому элементу списка присваивается целочисленный индекс, по которому можно обращаться к нему, модифицировать и даже удалять. С другой стороны, использование целочисленного индекса не является всегда удобным.*

*Например, если мы хотим в одной переменной хранить информацию о человеке (имя, фамилию, электронную почту, номер телефона, почтовый адрес), то нумерация является неудобной и малоинформативной. Более того, в таком случае не принципиален и порядок этих данных — главное, чтобы к ним можно было удобно обратиться.*

*Для этой цели в Python предусмотрены****словари****(dict) — упорядоченные наборы объектов, доступных по ключу. Иными словами, словарь — это совокупность пар ключ-объект.*

*Сам объект, который хранится в словаре, может быть любым. Даже другим словарём. Но на ключи есть важное ограничение: ключ может быть только объектом неизменяемых типов данных, т.е. числом, строкой или кортежем.*

*Как и в случае списков, словарь можно создать пустым, можно сразу наполнить его объектами, а можно расширять постепенно:*

*person = {} # с помощью фигурных скобок можно создать словарь*

*# словарь заполняется по принципу - ключ:объект (через двоеточие)*

*person = {'name' : 'Ivan Petrov'}*

*# в него можно также добавлять новые объекты по ключу*

*person['age'] =* ***25***

*person['email'] = 'ivan\_petrov@example.com'*

*person['phone'] = '8(800)555-35-35'*

***print****(person)*

*# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}*

*Попытка извлечения объекта по несуществующему ключу приведет к ошибке:*

***print****(person['address'])*

*# KeyError: 'address'*

*Можно отдельно получить список ключей:*

***print****(person.keys())*

*# dict\_keys(['name', 'age', 'email', 'phone'])*

*Или список значений:*

***print****(person.values())*

*# dict\_values(['Ivan Petrov', 25, 'ivan\_petrov@example.com', '8(800)555-35-35'])*

*Из словаря аналогично спискам можно удалить объект по его ключу. Словарь является упорядоченным. В функцию pop() всегда нужно передавать ключ удаляемого объекта:*

***print****(person)*

*# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}*

*person.pop('phone')*

***print****(person)*

*# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com'}*

*Задание 2.5.8*

*1 point possible (graded)*

*Какой из перечисленных объектов****не****может быть ключом в словаре?*

*42*



*['id', 42]*



*'id'*



*('id', 42)*



*нет ответа*

*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.9*

*1 point possible (graded)*

*Вместо знаков ??? вставьте название функции, которая удаляет объект из словаря по его ключу.*

*d.??? ('key')*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.10*

*1 point possible (graded)*

*Что выведет программа? Впишите получившуюся строку****без использования кавычек****.*

*d = {'day' :* ***22****, 'month' :* ***6****, 'year' :* ***2015****}*

***print****("||".join(d.keys()))*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.11 (Внешний источник)*

*Посмотреть ответ для самопроверки*

***Практический пример***

*Использование списков и словарей по-отдельности обеспечивает удобный и эффективный способ хранения данных, а их совместное использование открывает еще больше возможностей.*

*Рассмотрим, как может храниться база данных абитуриентов, поступающих в университет. Информация о каждом абитуриенте может храниться в виде словаря с ключами ФИО, Количество баллов, Заявление (о согласии на зачисление):*

*abit1 = {"ФИО" : 'Фадеев О.Е.', "Количество баллов" :* ***283****, "Заявление" : True}*

*abit2 = {"ФИО" : 'Дружинин И.Я.', "Количество баллов" :* ***278****, "Заявление" : False}*

*abit3 = {"ФИО" : 'Афанасьев Д.Н.', "Количество баллов" :* ***276****, "Заявление" : True}*

*abits = [abit1, abit2, abit3]*

***print****(abits)*

*# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}]*

*Этот список, по мере поступления документов, можно пополнять:*

*abit4 = {"ФИО" : 'Любимчиков А.Я.', "Количество баллов" :* ***269****, "Заявление" : True}*

*abits.append(abit4)*

***print****(abits)*

*# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}, {'ФИО': 'Любимчиков А.Я.', 'Количество баллов': 269, 'Заявление': True}]*

*Использование списка словарей с одним и тем же набором ключей позволяет обрабатывать все эти данные целиком, например, ранжировать по количеству баллов или фильтровать по наличию заявления. В следующих модулях мы научимся выполнять такие операции.*

***Уникальные элементы списка***

*Представим, что у нас есть список номеров и абонентов мобильного оператора, которым принадлежат эти номера. Всегда найдутся люди, на которых зарегистрировано несколько номеров. В таком случае в списке номеров-абонентов данные клиентов будут повторяться. Однако, например, для анализа клиентской базы может понадобиться выделить из общего списка только уникальные данные (скажем, уникальные фамилии).*

*Можно подойти к этой задаче с помощью циклов, пробегая по каждому элементу исходного списка и добавляя в новый только уникальные элементы. Уже звучит так, что для, казалось бы, такой простой операции нужно написать много кода. Не правда ли? Это правда. Руководствуясь философией Python — простое лучше, чем сложное, — хотелось бы иметь более простой способ это сделать. И он есть!*

*Для решения практических задач такого рода в Python есть ещё один изменяемый тип данных, который мы упоминали, но подробно не разбирали —****множества****(set).*

***Множество****— это неупорядоченный набор уникальных элементов. Иными словами, во множествах не могут повторяться элементы, а хранятся они в памяти компьютера в произвольном порядке.*

*Создать множество можно несколькими способами:*

*a = {'a', 'b', 'c', 'd'} # используя синтаксис { }*

*Или, что нам будет более полезно, множество можно создать из списка с помощью приведения типов:*

*L = [****1****,****1****,****2****,****3****,****2****]*

*b = set(L)*

***print****(b)*

*# {1,2,3}*

*В начале мы имели список из 5 элементов, два из которых встречались дважды. «Обернув» исходный список в множество, мы получили только уникальные элементы! И не потребовалось писать много строк кода, чтобы это сделать. Осталось только множество вернуть обратно в списковое представление, опять же используя явное приведение типов.*

*b\_list = list(b)*

***print****(b\_list)*

*# [1,2,3]*

*А для краткости все эти операции можно записать в одну строку, ведь в Python естественным образом заложена лаконичность кода.*

*c = list(set(L))*

***print****(c)*

*# [1,2,3]*

*Задание 2.5.12*

*Задание на самопроверку.*

*Напишите программу, которая на вход принимает текст и выводит количество уникальных символов.*

*Решение*

*2.5.13*

*1 point possible (graded)*

*Используя алгоритм из прошлой задачи, найдите количество уникальных символов в тексте. Скопируйте его к себе в консоль целиком.*

*The Zen of PythonBeautiful is better than ugly.Explicit is better than implicit.Simple is better than complex.Complex is better than complicated.Flat is better than nested.Sparse is better than dense.Readability counts.Special cases aren't special enough to break the rules.Although practicality beats purity.Errors should never pass silently.Unless explicitly silenced.In the face of ambiguity, refuse the temptation to guess.There should be one-- and preferably only one --obvious way to do it.Although that way may not be obvious at first unless you're Dutch.Now is better than never.Although never is often better than \*right\* now.If the implementation is hard to explain, it's a bad idea.If the implementation is easy to explain, it may be a good idea.Namespaces are one honking great idea -- let's do more of those!*

*нет ответа*



*Loading*

*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Множества в Python аналогичны математическим множествам, поэтому для них существует несколько собственных операций.*

|  |  |  |
| --- | --- | --- |
| ***Операция*** | ***Название*** | ***Смысл*** |
| *set.union(other)* | *Объединение* | *Возвращает множество, состоящее из элементов set и other.* |
| *set.intersection(other)* | *Пересечение* | *Возвращает множество, состоящее из элементов, которые встречаются и в set, и в other.* |
| *set.difference(other)* | *Разность* | *Возвращает множество элементов set, которые не встречаются в other.* |
| *set.symmetric\_difference(other)* | *Симметричная разность* | *Возвращает множество, включающее все элементы исходных множеств, которые не принадлежат обоим одновременно.* |

*Чтобы не ходить далеко за примером, вернёмся к базе абонентов мобильного оператора. Пусть у нас есть множество абонентов (для простоты — фамилии) и множество должников, а мы хотим получить множество абонентов, не имеющих долгов.*

*abons = {"Иванов", "Петров", "Васильев", "Антонов"}*

*debtors = {"Петров", "Антонов"}*

*non\_debtors = abons.difference(debtors)*

***print****(non\_debtors)*

*# {'Васильев', 'Иванов'}*

*Задание 2.5.14*

*1 point possible (graded)*

*Найдите ошибку в коде и перепишите строку с ошибкой полностью. Представленная ниже программа должна находить множество символов, которые встречаются в двух строках одновременно.*

*a = input("Введите первую строку: ")*

*b = input("Введите вторую строку: ")*

*a\_set, b\_set = set(a), set(b) # используем множественное присваивание*

*a\_and\_b = a\_set.union(b\_set)*

***print****(a\_and\_b)*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.15*

*1 point possible (graded)*

*Напишите программу, которая на вход получает две последовательности целых чисел, а возвращает список элементов, встречающихся только в первой или во второй последовательности, но не в двух одновременно. Какую операцию над множествами вы использовали?*

*Как и в предыдущем задании, нужно подобрать подходящую функцию множества. Выглядеть это будет так: set.something(other), где something и есть выбранная вами функция. Именно её нужно ввести в поле ответа.****Обратите внимание:****необходимо указать только название функции, без скобок.*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

***Проверьте себя***

*Напишите числа в порядке возрастания через пробел, которые выведет программа из предыдущего задания, если на вход подаются две последовательности чисел:*

***1******2******3******4******5******6******7******8***

***2******4******6******8******10******12***

*Ответ*

*НазадВперёд*

person = {} # с помощью фигурных скобок можно создать словарь

# словарь заполняется по принципу - ключ:объект (через двоеточие)

person = {'name' : 'Ivan Petrov'}

# в него можно также добавлять новые объекты по ключу

person['age'] = **25**

person['email'] = 'ivan\_petrov@example.com'

person['phone'] = '8(800)555-35-35'

**print**(person)

# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}

**print**(person['address'])

# KeyError: 'address'

**print**(person.keys())

# dict\_keys(['name', 'age', 'email', 'phone'])

**print**(person.values())

# dict\_values(['Ivan Petrov', 25, 'ivan\_petrov@example.com', '8(800)555-35-35'])

**print**(person)

# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}

person.pop('phone')

**print**(person)

# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com'}

Напишите программу, которая получает на вход название книги - title, фамилию автора - author и год выпуска - year.

Полученные данные должны быть преобразованы в словарь book с ключами: title, author, year. Причем year нужно преобраовать в тип int.

title = input("Введите название книги:")  
author = input("Введите фамилию автора:")  
year = int(input("Введите год издания:"))  
  
book = {'title': title,  
 'author': author,  
 'year': year}  
  
print(book)

print("Привет работаем")  
abit1 = {"ФИО" : 'Фадеев О.Е.', "Количество баллов" : 283, "Заявление" : True}  
abit2 = {"ФИО" : 'Дружинин И.Я.', "Количество баллов" : 278, "Заявление" : False}  
abit3 = {"ФИО" : 'Афанасьев Д.Н.', "Количество баллов" : 276, "Заявление" : True}  
  
  
abits = [abit1, abit2, abit3]  
  
print(abits)  
# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}]

Этот список, по мере поступления документов, можно пополнять:

abit4 = {"ФИО" : 'Любимчиков А.Я.', "Количество баллов" : **269**, "Заявление" : True}

abits.append(abit4)

**print**(abits)

# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}, {'ФИО': 'Любимчиков А.Я.', 'Количество баллов': 269, 'Заявление': True}]

**множества** (set).

**Множество** — это неупорядоченный набор уникальных элементов. Иными словами, во множествах не могут повторяться элементы, а хранятся они в памяти компьютера в произвольном порядке.

a = {'a', 'b', 'c', 'd'} # используя синтаксис { }

множество можно создать из списка с помощью приведения типов:

L = [**1**,**1**,**2**,**3**,**2**]

b = set(L)

**print**(b)

# {1,2,3}

Осталось только множество вернуть обратно в списковое представление, опять же используя явное приведение типов.

b\_list = list(b)

**print**(b\_list)

# [1,2,3]

А для краткости все эти операции можно записать в одну строку, ведь в Python естественным образом заложена лаконичность кода.

c = list(set(L))

**print**(c)

# [1,2,3]

Напишите программу, которая на вход принимает текст и выводит количество уникальных символов.

text = input("Введите текст:")

unique = list(set(text))

**print**("Количество уникальных символов: ", len(unique))

text = "The Zen of PythonBeautiful is better than ugly.Explicit is better than implicit.Simple is better than complex.Complex is better than complicated.Flat is better than nested.Sparse is better than dense.Readability counts.Special cases aren't special enough to break the rules.Although practicality beats purity.Errors should never pass silently.Unless explicitly silenced.In the face of ambiguity, refuse the temptation to guess.There should be one-- and preferably only one --obvious way to do it.Although that way may not be obvious at first unless you're Dutch.Now is better than never.Although never is often better than \*right\* now.If the implementation is hard to explain, it's a bad idea.If the implementation is easy to explain, it may be a good idea.Namespaces are one honking great idea -- let's do more of those!"  
#input("Введите текст:")  
print(text)  
unique = list(set(text))  
  
print("Количество уникальных символов: ", len(unique))

Множества в *Python* аналогичны математическим множествам, поэтому для них существует несколько собственных операций.

|  |  |  |
| --- | --- | --- |
| **Операция** | **Название** | **Смысл** |
| set.union(other) | Объединение | Возвращает множество, состоящее из элементов set и other. |
| set.intersection(other) | Пересечение | Возвращает множество, состоящее из элементов, которые встречаются и в set, и в other. |
| set.difference(other) | Разность | Возвращает множество элементов set, которые не встречаются в other. |
| set.symmetric\_difference(other) | Симметричная разность | Возвращает множество, включающее все элементы исходных множеств, которые не принадлежат обоим одновременно. |

Чтобы не ходить далеко за примером, вернёмся к базе абонентов мобильного оператора. Пусть у нас есть множество абонентов (для простоты — фамилии) и множество должников, а мы хотим получить множество абонентов, не имеющих долгов.

abons = {"Иванов", "Петров", "Васильев", "Антонов"}

debtors = {"Петров", "Антонов"}

non\_debtors = abons.difference(debtors)

**print**(non\_debtors)

# {'Васильев', 'Иванов'}

--

Найдите ошибку в коде и перепишите строку с ошибкой полностью. Представленная ниже программа должна находить множество символов, которые встречаются в двух строках одновременно.

a = input("Введите первую строку: ")

b = input("Введите вторую строку: ")

a\_set, b\_set = set(a), set(b) # используем множественное присваивание

a\_and\_b = a\_set.union(b\_set)

**print**(a\_and\_b)

#a = str(123456789)  
#a = list(str(123456789))  
#print(type(a))  
#for e in str(123456789):  
# print(e)  
#r = map(int.list(a))  
print('3' in str(123456789) and '0' in str(123456789))  
print('5' in str(123456789))  
# True

N = **123**

N\_str = str(N) # преобразуем число в строку

first\_digit = int(N\_str[**0**]) # берём самый левый элемент строки и преобразуем его в число

print(first\_digit % **2** == **0**) # выводим True в случае, если цифра делится на 2, иначе False

# False



passwords = {"Victor" : "7314870", "user" : "12345", "root" : "10 101", "admin": "aaa"}

passwords.keys()

passwords.values()

passwords.get("user") #перебор элементов словаря for name,

password in passwords.items():

print(name +password)

#Задача 2. Как проверить пароль при входе в систему?

nick = input("Введите ваше имя: ")

pwd = input("Введите ваш пароль: ")

if nick in passwords:

if passwords[nick] == pwd:

print("Вы вошли в систему!")

else: print("Неверный пароль!")

else: print("Такого пользователя нет!")

**Вебинар «Python-практикум» (1)**

 Добавить страницу в мои закладки

Добро пожаловать на вебинар «*Python*-практикум»!

**Спикер:**Егор Закутей

**Задачи, разобранные на вебинаре:**1. С клавиатуры введено натуральное число от 1 до 99. Выведите сумму его цифр.

2. С клавиатуры пользователь вводит два числа a и b. Выведите гипотенузу треугольника с заданными катетами.

3. Пользователь вводит с клавиатуры свой e-mail. Выведите домен этой электронной почты.

4. В коде задан список А. Выведите сумму элементов списка с индексами, кратными 3.

5. В коде задан список А. Циклически сдвиньте элементы списка вправо.

6. В коде задан спиоск А. Найдите второй и тертий минимумы этого списка.

A = [1, 2, 3, 4, 5, 6, 7]  
# first\_min = A.pop(A.index(min(A)))  
# second\_min = A.pop(A.index(min(A)))  
# third\_min = A.pop(A.index(min(A)))  
# first\_min = min(A)  
# A.remove(min(A))  
#  
# second\_min = min(A)  
# A.remove(second\_min)  
#  
# third\_min = min(A)  
A.sort()  
print(A[1], A[2])  
  
# print(second\_min, third\_min)

7. Сначала пользователь вводит число n с клавиатуры. После с новой строки он вводит число в n-ичной системе счисления. Нужно вывести это число в десятичной системе.

# s = "12412541234"  
# n = 7 #семеричная система  
# ans = 0  
# for i in range(len(s)):  
# ans += int(s[-i]) \* n \*\* i  
# print(ans)  
n = int(input('система исчисления= ')) # система исчисления  
s = input('число= ') # число  
ans = int(s, n)  
print('десятичное число= ', ans)

8. В коде задан список А, который состоит из списков. Нужно в переменной В получить список, который получается в результате склеивания списков из А.

A = [[1, 2], [3, 4, 5], [6], [7, 8, 9, 10], [11], [12, 13, 14]]  
B = []  
for elem in A:  
 # for i in elem:  
 # B.append(i)  
 # B.extend(elem)  
 # B = B + elem  
 B += elem  
  
print(B)

9. В коде задана переменная text с текстом. Нужно подсчитать для этого текста статистику: сколько раз какой символ в этом тексте встретился.

10. Пользователь вводит число n. Нужно вывести, является ли число простым.

11. Пользователь вводит в консоль строку. Удалите в этой строке каждое второе слово.

12. Пользователь вводит в консоль целое число. Нужно вывести запись этого числа в двоичной системе счисления.