Курс Python base

Образовательный уровень: 1

БАЗОВЫЙ КУРС

СЕРГЕЙ БУРДИН

> Python

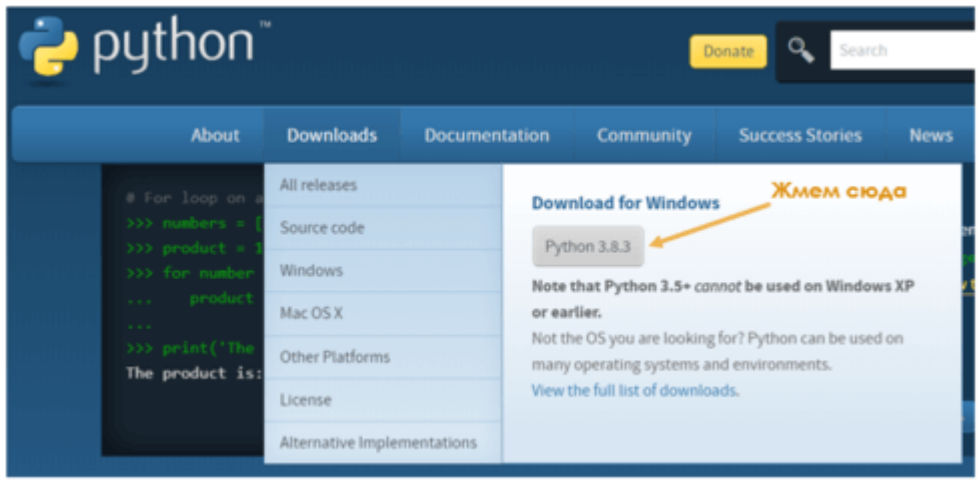
УРОК 1 Установка. программа и оперативная память. ячейки памяти. математические операции. вывод на экран.

Язык программирования Python является текстовым, объектно-ориентированным языком. Что значит - объектно-ориентированный, мы с вами пройдем чуть позже. Также, Python, очень популярен в мире среди множества программистов и с каждым годом набирает популярность, так-как это современный, молодой язык,

программируют искусственный интеллект. Одним словом – это популярный, молодой и «крутой» язык.

Установка среды разработки

Давайте установим среду разработки Python. Для этого зайдем на официальный сайт: https://www.python.org/



Жмем на серую кнопку, как показано на рисунке. После этого будет скачан установочный файл, запускаем его и устанавливаем среду Python.

Как выполняется программа в компьютере

Прежде чем начинать знакомиться с первыми командами языка Python (да и вообще, любого текстового языка программирования), необходимо немного понимать, как устроен компьютер, как он работает и как он выполняет программы.

Сперва вспомним, а кто-то, возможно, узнает впервые, что компьютер (в том числе любое компьютерное устройство: ноутбук, планшет, смартфон) состоит из нескольких основных частей-устройств: центральный процессор, основная память, оперативная память, видеокарта и экран (дисплей).

Любая программа храниться в основной памяти, а когда ее запускает пользователь, она копируется в оперативную память и там уже начинает выполняться компьютером. На самом деле программа выполняется не совсем компьютером, а точнее сказать – операционной системой (например Windows, Android или iOS). И вот этот процесс нам наиболее интересен – как выполняется программа операционной системой. Итак: мы запускаем программу – она копируется в оперативную память и размещается там (рисунок 1):

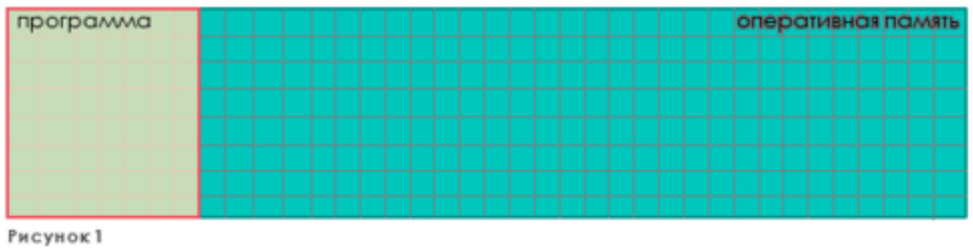


Рисунок 1

На рисунке оперативная память разделена на клетки – это означает, что оперативная память в компьютере разделена на ячейки (на самом деле все немного сложнее – каждая ячейка разделена на более маленькие ячейки – биты, но пока нам достаточно представлять так). Далее программа начинает выполняться, сначала первая команда, потом – вторая и т.д., пока не закончиться программа.

Переменные – ячейки памяти

Давайте познакомимся с самой простой командой в программировании – это создание ячейки памяти (еще их называют переменные, как в математике). Чтобы создать переменную, в Python необходимо придумать имя для нее и с помощью знака равно сразу присвоить ей какое-нибудь значение:

a = 6

посмотрим, что произойдет в оперативной памяти, при выполнении такой команды (рисунок 2):

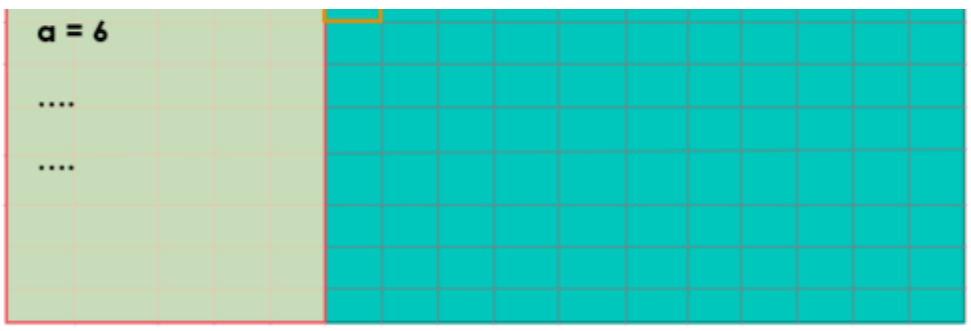


Рисунок 2

Как видно, для нашей программы, в оперативной памяти создалась ячейка с именем – **a** и в нее записалось число – **6**. То есть, операционная система выделила для нашей программы ячейку памяти и теперь в последующих командах программы можно использовать эту ячейку памяти для хранения в ней чисел или строк.

Первая программа

Давайте запустим среду разработки Python и напишем нашу первую программу. Для запуска, необходимо в меню **Пуск** (обычно находится в левом нижнем углу) найти программу – IDLE (Python …). Запускаем ее, откроется окно программы – это окно называют – «**Консоль**». В консоли будут выводиться результаты программы. Чтобы создать новую программу, необходимо выбрать меню – **File**, а в нем выбрать пункт - **New file.** После чего откроется пустое окно, в котором можно писать команды и создавать программу.

Прежде чем проверить команду создания ячейки памяти, с которой мы уже выше познакомились, познакомимся с такой командой как – **print(…)**

Эта команда выводить на экран монитора, а точнее в **консоль** то, что будет указано в скобках, например:

print(2020)

Эта команда выведет на экран число – 2020.

Теперь попробуем написать эту команду в программе и запустить программу (чтобы запустить программу, необходимо выбрать меню – **Run**, а в нем выбрать пункт – **Run module** (либо нажать клавишу – **F5**).

! Проблемная задача: написать эту команду в программе и запустить программу, проверить результат. Попробовать вывести на экран другие числа.

Теперь можно попробовать создать ячейку памяти и записать в нее какое-нибудь число.

! Проблемная задача: создать ячейку памяти, записать в нее свой возраст и вывести на экран эту ячейку памяти.

Решение:

Age = 16

print (Age)

Мы выводили сейчас только числа, а сейчас попробуем вывести строку. В текстовых языках программирования строки всегда пишутся в кавычках:

“Привет, я строка!”

! Проблемная задача: вывести на экран приветствие: “Hello! My name is Mark Zuckerberg”. Только указать свое имя.

Решение:

print (“Hello! My name is Serg.”)

Оказывается, в скобках у команды - **print(…)** можно указывать несколько объектов, например:

print(“My age is”, 16)

Если выполнить эту команду, то на экран выведется:

! Проблемная задача: создать две переменные (ячейки памяти), в одну записать свой возраст, в другую строку – “My age is”. Затем, с помощью команды – **print()** вывести на экран обе эти переменные.

Решение:

Age = 16

Say = “My name is”

print(Say, Age)

В скобках команды – **print(…)** можно указать ни только два объекта, но и больше, разделяя их запятой.

Простейшие математические операции

Во всех текстовых языках программирования, в том числе, конечно, и в Python реализованы математические операции. Это значит, что мы можем в командах писать эти операции и компьютер, во время выполнения этих команд в программе, будет вычислять их в соответствии с правилами математики.

Например:

a = 5

b = 48

c = a + b

print(c)

В результате работы программы на экран выведется:

53

Типы переменных:

Если попытаться выполнить следующую программу:

a = 5

b = ‘2020’

c = a + b

Python выдаст ошибку:

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Если перевести с английского, это значит – «операция плюс не может быть применена с числовому типу и строковому». Дело в том, что в оперативной памяти каждая ячейка (переменная) имеет свой тип. Типов существует несколько, но сейчас нас интересует два типа:

- целочисленный (называется – **int**)

- строковый (называется – **str**)

Тип у переменной определяется, когда в нее что-то записывается (или другими словами, когда ей что-то присваивается). Если в переменную записывается целое число, то ее тип становиться – **int**, если же записывается строка, то тип становиться – **str.**

Логично, что нельзя произвести математическую операции (+, -, *, /) между строкой и числом.

! Проблемная задача:

Выполнить на листочке или в графическом редакторе (например – **Paint**) следующую программу:

c = a + b

b = b + 3

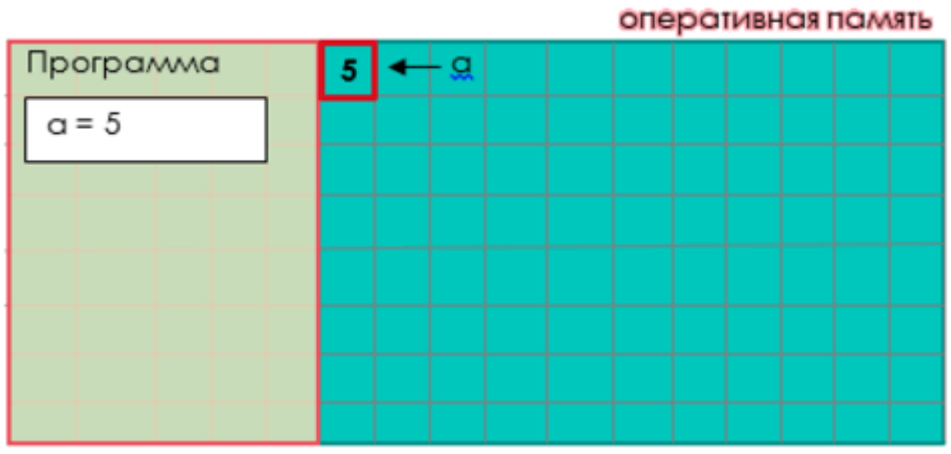
a = c * b

print (2*a - a)

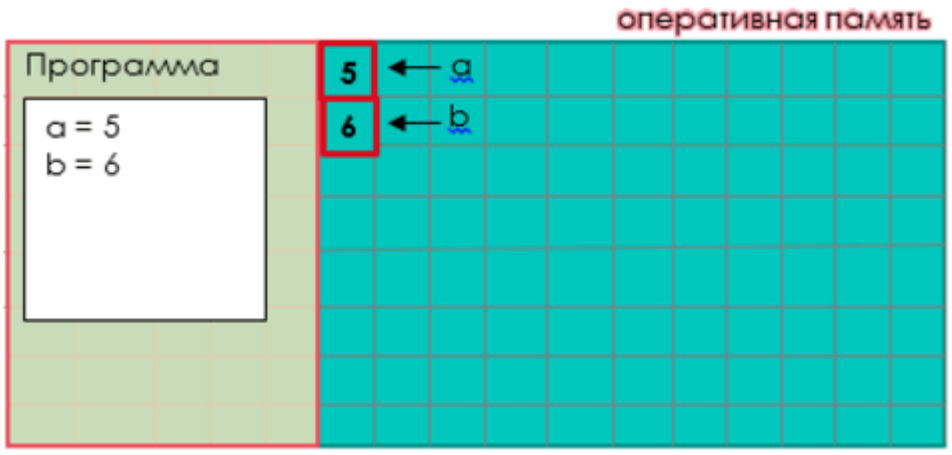
Для каждой команды нарисовать, что будет в оперативной памяти.

Решение:

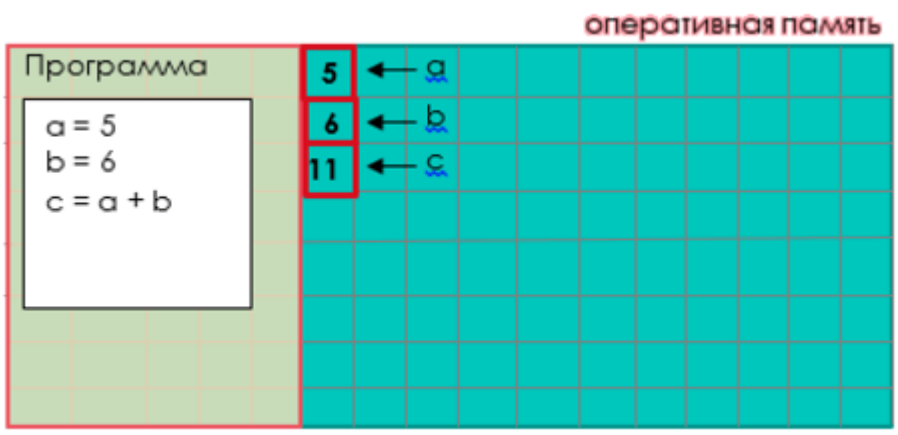
Шаг 1



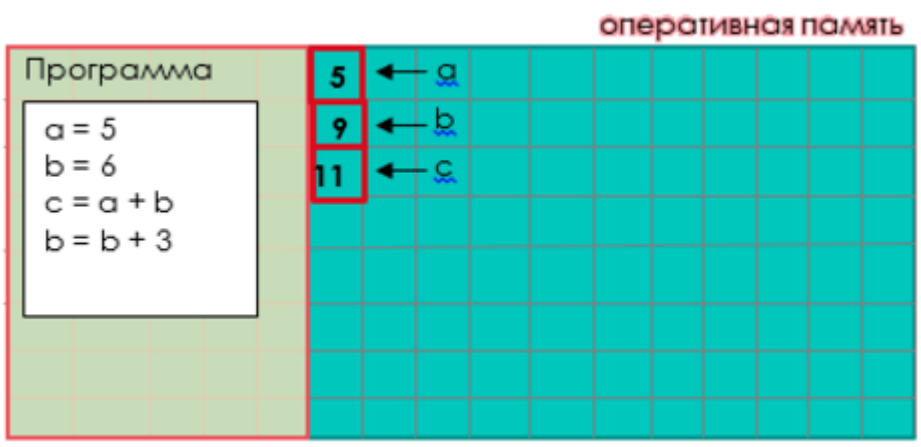
Шаг 2

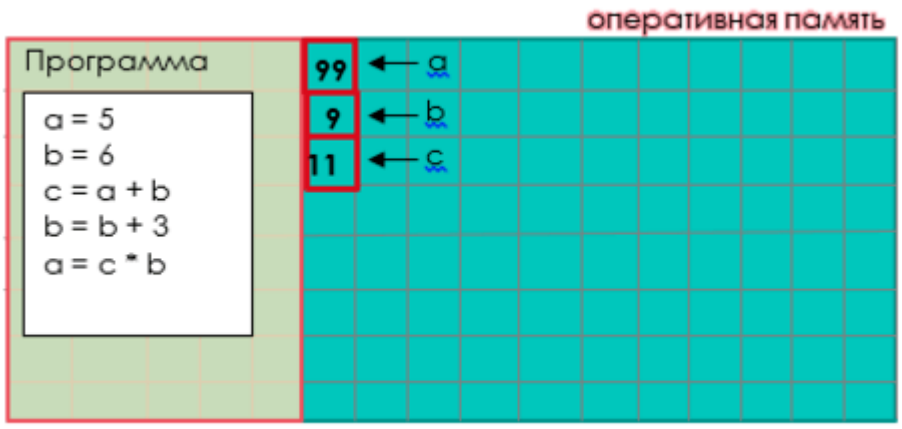


Шаг 3



Шаг 4





Шаг 6

Консоль (экран)

99

Урок 2. условный оператор

На прошлом уроке мы узнали, как:

- создавать переменные (ячейки памяти)

- производить с ними математические операции (относится к числовым переменным) - выводить на экран числовую и текстовую информацию. Команда - **print()**

А также узнали, что каждая переменная имеет определенный тип, например – **int** (целое число) и – **str** (строка).

Можно сказать, что команда – **print() –** это команда, с помощью которой программа выводит информацию на экран, то есть, таким образом программа передает информацию пользователю.

Ввод информации пользователем, команда input()

? Как вы думаете, какое противоположное действие этой команде?

Команда, с помощью которой, пользователь передает информацию программе.

Эта команда называется – **input()**

Если перевести ее на русский, то получится слово **ВВОД**. Что делает эта команда? Она заставляет компьютер перейти в режим ожидания. Выполнение программы встает на паузу, пока пользователь не напечатает какой либо текст или число и не нажмет клавишу **ENTER**. После того, как пользователь нажимает на клавишу **ENTER**, компьютер отправляет в нашу программу строку из символов, которые напечатал пользователь и команда **input()** становится равна этой строке. Можно представить, что **input() –** становится как переменная, которая равна строке, которую напечатал пользователь. На языке программистов говорят: “функция **input()** возвращает строку, которую ввел пользователь”.

А теперь давайте опробуем эту команду в работе. Откроем среду разработки Python и напишем программу, которая будет сначала запрашивать у пользователя строку, а затем выводить ее на экран (в консоль).

```
information = input()

print(information)
```

Мы создали переменную – **information**, присвоили ей команду **input()**, которая запросит от пользователя ввод строки и нажатие клавиши **ENTER**. То есть, то, что введет пользователь запишется в переменную **information**. А затем вывели значение этой переменной на экран.

Возможно, кого-то смутило имя нашей переменной – **information**, так вот, оказывается, у профессиональных программистов, принято называть переменные целыми словами или даже словосочетаниями, обозначающими смысл переменной. Например переменной, в которой хранится возраст человека, дают имя – **Age**.

Переменной, хранящей значение стоимости автомобиля – **PriceCar**.

Решение:

a = input()

b = input()

print(a + b)

Как мы видим, программа работает некорректно. При вводе, например чисел 45 и 65, программа выводит на экран число 4565. Но это только на первый взгляд, кажется неправильным.

! Проблемная задача: ответить на вопрос - почему программа выводит такой результат?

Решение:

Дело в том, что **input()** имеет тип – **str**, и даже, если пользователь вводит число, **input()** возвращает строку с этим числом.

Например, если пользователь напечатал 7843, **input()** вернет строку “7843”. И операция **+** просто “склеит” друг с другом эти строки.

Как же быть?

В программировании команды чаще называются функциями. То есть print() – это функция, input() – это тоже функция. Функция может просто выполнять какое-либо действие, как например print(), а может быть и равна какому либо значению, как например функция input().

В Python есть такая функция – **int(...)**. Она меняет тип объекта, который записан в скобках или на языке программистов: меняет тип аргумента (аргумент – это то, что записано в скобках у функции) на целочисленный тип (**int**). То есть, если мы применим эту команду к функции **input(),** то результат будет уже иметь тип **int** и переменные, в которые мы сохраняем числа, введенные пользователем, будут иметь целочисленный тип. Соответственно операция **+** будет выполнена с ними как с числами.

! Проблемная задача: скорректировать код программы из предыдущей проблемной задачи, так, чтобы на экран выводилась сумма двух чисел, введенных пользователем.

Решение:

a = int(input())

b = int(input())

print(a + b)

Условный оператор – if

Чтобы программа могла как-то обрабатывать получаемую от пользователей информацию, она должна уметь проверять эту информацию на соответствие каким-либо условиями.

Например, программа авторизации пользователя при входе в систему Windows должна проверить пароль, введенный пользователем. Программа, которая обрабатывает информацию от видеокамер на дорогах, должна проверять не превышает ли скорость проезжающих автомобилей допустимое значение.

Для того, чтобы проверять какие-либо условия и, в зависимости от того, выполняются они или нет, выполнять необходимые команды, в Python есть оператор – **if**.

На самом деле, все как в жизни. **If** – переводится как **ЕСЛИ**. Ведь в жизни и быту, мы тоже задаем условия и мысленно их порой проверяем употребляя слово - **ЕСЛИ**.

Например:

Если на улице идет дождь, то возьму с собой зонт.

Если вода кипит, то заварю чай.

Так и в Python:

if password == '@12345':

обычного **=**. Двойной равно используется лишь для проверки условия – равно или нет.

И еще заметьте, что команда **print('Access is allowed')** написана с отступом относительно команды **if**. Такой отступ ставиться с помощью клавиши **TAB** и используется для того, чтобы показать, какие команды должны выполниться при выполнении условия.

! Проблемная задача: написать калькулятор.

Решение:

```
a = int(input('Enter first number: '))

b = int(input('Enter second number: '))

sign = input('Enter a sign of operation: ')

if sign == '+':

    print(a, '+', b, '= ', a + b)

elif sign == '-':

    print(a, '-', b, '= ', a - b)

elif sign == '*':

    print(a, '*', b, '= ', a * b)

elif sign == '/':

    print(a, '/', b, '= ', a / b)

else:

    print('unknow operation')
```

Урок 3. цикл for

Сегодня мы познакомимся с оператором цикла. Оператор цикла – это такое служебное слово в Python, с помощью которого можно заставить компьютер повторить несколько раз какие-либо команды. Например, если мы хотим вывести на экран 5 раз слово – «Ура!», мы напишем:

```
print('Ура!')

print('Ура!')

print('Ура!')

print('Ура!')

print('Ура!')
```

И это не очень оптимально, тем, более если придется вывести на экран 1000 таких надписей.

Поэтому для программирования придумали оператор цикла. Их существует несколько, и сегодня мы познакомимся с одним из них и называется он:

for

Давайте по порядку внимательно разберем как использовать этот оператор.

Оператор **for** всегда используется с переменной счетчиком. Переменная счетчик – это такая ячейка памяти, в которой будет храниться номер повторения. Пишется это так:

for i

Дальше пишется специальное слово – **in.**

А после него записывается интервал, который задает от какого до какого числа будет меняться переменная счетчик. Этот интервал задается с помощью команды – **range(num1, num2),** где **num1** – левая граница интервала, а **num2** – правая граница интервала.

Итак, теперь все соберем вместе:

```
for i in range(1, 5):
```

И после двоеточия в следующей строке с отступом (отступ играет такую же роль как и у оператора **if**) мы пишем команды, которые будут повторяться. Давайте выведем на экран пять раз слово «Ура!»:

```
for i in range(1, 5):
    print('Ура!')
```

В результате мы увидим, что слово «Ура!» вывелось только 4 раза. Оказывается, в Python функция **range(num1, num2)** так устроена, что задает интервал не до **num2,** а до предыдущего числа, то есть в нашем случае интервал будет от 1 до 4, и если мы хотим, чтобы цикл выполнился 5 раз, то необходимо написать вот так:

```
for i in range(1, 6):
    print('Ура!')
```

! Проблемная задача: написать программу, которая запрашивает у пользователя натуральное число и потом выводит на экран все числа от 0 до введенного числа.

Решение:

```
num = int(input('Введите натуральное число: '))
for i in range(0, num + 1):
    print(i)
```

! Проблемная задача: написать программу, которая запрашивает у пользователя натуральное число и выводит на экран сумму всех чисел от1 до введенного числа.

Решение:

```
num = int(input('Введите натуральное число: '))
summ = 0
for i in range(0, num + 1):
    summ = summ + i
print(summ)
```

Итак, мы познакомились с оператором цикла – **for**. С помощью него можно выполнять повтор необходимых команд несколько раз.

У цикла – **for** есть фишка, а точнее у функции – **range()**, которая задает интервал. Оказывается в скобках функции **range()** можно указать всего лишь один параметр-число, и тогда цикл просто повторит команды, которые идут в следующих строках после оператора **for**, количество раз равное указанному в скобках числу.

Например:

```
for i in range(6):
    print('Hello!')
```

Данный фрагмент программы выведет на экран 6 раз приветствие “Hello!”.

А переменная - **i** в процессе выполнения цикла будет принимать значения от 0 до 5.

компьютеров на складе. В конце каждого дня в программу заводится количество компьютеров, которые привезли на склад и количество компьютеров, которые продали. В конце каждой недели, программа выводит итог – количество компьютеров, оставшихся на складе.

Формат ввода: сначала программа запрашивает число – сколько компьютеров привезли на склад в День 1, затем запрашивает число – сколько компьютеров продали в День 1. Далее операция повторяется для День 2, День 3 и т.д. до День 7 (в неделе 7 дней). После программа выводит на экран итог – **количество оставшихся на складе компьютеров.**

Решение:

```
print('---Склад магазина---')

print()

result = 0

for i in range(7):

    print('Сколько компьютеров привезли в день', i + 1, ': ')

    nget = int(input())

    print('Сколько компьютеров продали в день ', i + 1, ': ')

    nsale = int(input())

    result = result + nget - nsale

print('На складе осталось', result, 'компьютеров')
```

Урок 4. цикл **while**

На прошлом уроке мы познакомились с циклом **for**, в Python есть еще один замечательный цикл, который называется цикл **while**. Давайте познакомимся с ним и узнаем, как его использовать.

Если цикл **for** повторяет команды конкретное количество раз, использует переменную счетчик, которая меняется в каком-то интервале и повторяется он столько раз, сколько раз меняется переменная счетчик, то цикл **while** повторяется не конкретное количество раз, а до тех пор, пока выполняется заданное **условие цикла**. И задается он вот так:

while **условие цикла**:

    команда 1

    команда 2

    …

Рассмотрим работу этого цикла на простом примере:

```
a = 0

while a < 6:

    print(a)

    a = a + 1
```

На экран выведутся все числа от 0 до 5. Как видно, сначала, в ячейку памяти – **a** сохраняется число – 0 (ноль), затем запускается цикл, который будет повторять операции:

```
    print(a)

    a = a + 1
```

до тех пор пока будет выполняться условие: **a < 6**, но так-как ячейка памяти – **a,** каждый раз увеличивается на 1 за счет операции:

Рассмотрим работу данного цикла на примере программы, которая предоставляет доступ в какую-либо систему по паролю:

```
access = False

while not access:

    password = input('Введите пароль: ')

    access = password == '123'

print('Доступ разрешен')
```

На этом примере, мы с вами познакомимся с новым типом переменных (ячеек памяти). Помимо числовых и строковых типов, в программировании, в том числе и в Python есть такой тип – **Boolean** или по-русски – «логический» тип. Переменный типа **Boolean** могут принимать всего два значения: **True** и **False. True –** означает «Правда», а **False –** «Ложь».

Итак, в примере мы создали ячейку памяти (переменную) логического типа **access** и присвоили ей значение **False,** т. е. «Ложь». Далее мы запускаем цикл **while** и в качестве условия цикла используем выражение – **not access.** Слово **not** обозначает русскую частицу «не», то есть, если **access** равна **False** (Ложь), то выражение **not access** – означает «не Ложь», то есть «Правда». Поэтому условие цикла будет «Правда», значит выполнится и цикл запуститься.

Далее в цикле мы запрашиваем у пользователя ввод строки и переменной **access** присваиваем результат условия password == '123', а результат любого условия – это либо «Правда» (**True**), либо «Ложь» (**False**). В данном примере правильный пароль – «123». И цикл снова проверяет условие, то есть выражение **not access**. Получается, что если пользователь вводит неверный пароль, то переменной **access** присваивается «Ложь», а если пользователь введет правильный пароль – «123», то условие цикла не выполниться, цикл закончится и выполниться уже следующая команда:

```
print('Доступ разрешен')
```

На экран выведется сообщение «Доступ разрешен».

! Проблемная задача: написать программу, используя цикл **while**, которая запрашивает у пользователя число, а затем выводит на экран такое число, квадрат которого ближе всех находится к введенному числу.

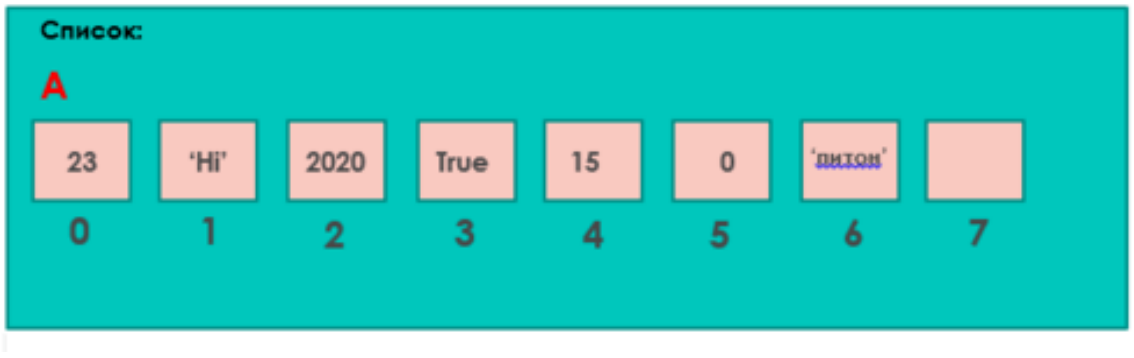
Решение:

```
n = int(input())

x = 1

while x**2 < n:

    x = x +1


if (n - (x - 1)**2) <= (x**2 - n):

    print(x-1)

else:

    print(x)
```

Урок 5. списки

На этом уроке мы познакомимся с такой штукой (объектом), как «списки».

Список – эта такая структура, которую можно сравнить с поездом, в котором каждый вагон – это ячейка памяти. Сам список (поезд) имеет имя, а каждая ячейка памяти (каждый вагон) имеет порядковый номер.



На рисунке изображен пример списка, его имя - **A**, он состоит из 8-ми ячеек памяти, или правильно говоря: из 8-ми элементов. Заметьте, что нумерация элементов всегда начинается с **нуля.** И элементы списка все содержат разные значения. Например элемент списка:

A[0] содержит число – **23**, а элемент списка:

A[6] содержит строку – **“питон”**

Как вы уже, наверное, догадались, чтобы обратиться или совершить какое либо действие с конкретным элементом списка, к нему обращаются используя имя списка и в квадратных скобках указывают его порядковый номер.

Теперь давайте попробуем поработать со списками в Python. Для начала создадим список. Создается он также, как и переменные (ячейки памяти). Мы просто пишем имя списка и присваиваем ему какие-либо значения:

MyList = [23, 'Hello!', True, 'питон']

Список создан. Теперь попробуем вывести на экран какие-нибудь его элементы.

print(Mylist[0])

print(MyList[3])

print(MyList[1], MyList[2])

На экране выведется:

23

питон

Hello! True

Список очень удобен для хранения какой-либо информации об нескольких объектах или для хранения значений разных свойств одного объекта.

Список имен друзей:

ListOfName = ['Василий', 'Яна', 'Петр', 'Леонид', 'Виктория']

Список значений характеристик автомобиля (цвет, максимальная скорость, мощность л/с, марка):

ListOfCarProperties = ['white', 240, 170, 'BMW']

! Проблемная задача: создать список в программе из 10 элементов и вывести все элементы списка на экран используя цикл **for**

Наверное, у многих возник вопрос, как сделать так, чтобы элементы списка ввел пользователь программы? Это можно сделать, например, используя функцию – **append()**, которая добавляет в список новый элемент, причем добавляет в конец списка, то есть справа. Еще фишка этой функции заключается в том, что она является не самостоятельной функцией, а принадлежит списку. Это значит, что используется она в таком формате:

MyList.append(45)

То есть, мы пишем имя списка, а затем через точку вызываем функцию **append()** и в скобках указываем значение добавляемого элемента.

Например, если мы создадим такой список:

MyList = [34, 2, 5000]

и выполним команду:

MyList.append(45)

Поэтому, чтобы список сформировал пользователь, мы можем сначала создать пустой список:

MyList = []

Затем запросить у пользователя количество элементов списка с помощью команды **input()**.

Запустить цикл **for**, задав количество повторений (итераций) цикла равное количество элементов списка, которое ввел пользователь, и в цикле уже запрашивать у пользователя значения элементов и с помощью команды **append()** добавлять их в список.

! Проблемная задача: реализовать описанный выше алгоритм создания списка пользователем на Python.

Решение:

```
N = int(input())

ListN = []

for i in range(N):

    ListN.append(input())
```

Помимо описанного выше алгоритма создания списка пользователем программы, существует и другой алгоритм, без использования команды **append().**

В Python можно сразу создать список заданной длинны (с заданным количеством элементов), с помощью такой команды:

```
ListN = [0]*N
```

После этой команды в оперативной памяти будет создан список с количеством элементов равным – **N**, и все элементы будут равны нулю.

А далее можно запустить цикл **for** и каждому элементу списка присвоить функцию **input()**, которая запросит значение для текущего элемента у пользователя:

```
N = int(input())

ListN = [0]*N

print(ListN)

for i in range(N):

    ListN[i] = input()
```

Урок 6 min и max елементы списка

Сегодня мы продолжим работать со списками и изучим алгоритм нахождения минимального и максимального элементов списка. Но для начала давайте решим такую задачку:

! Проблемная задача: представьте, что вам по очереди показывают карточки с числами, на каждой карточке написано только одно произвольное число, при этом у вас тоже есть карточка, на которую вы можете записывать, тоже только одно число, но вы можете стирать его ластиком и записывать другое, главное, что на вашей карточке всегда должно быть только одно число (для этого задания можно подготовить карточки или листочки для учеников и раздать ластики).

Задача: после того, как будут показаны все карточки с числами, на вашем листочке должно быть записано самое максимальное число из всех показанных.

Алгоритм нахождения максимального элемента в списке очень похож на алгоритм, который мы применили в предыдущей задаче. Если представить, что карточки с числами – это элементы списка, а карточка, на которой вы записывали максимальное число – это ячейка памяти (переменная), то получается, что данный алгоритм можно представить на Python.

выводит на экран максимальныи элемент списка.

Решение:

```
N = int(input())
ListN = []
for i in range(N):
    ListN.append(int(input()))


Max = ListN[0]
for i in range(1, N):
    if Max<ListN[i]:
        Max = ListN[i]
print(Max)
```

! Проблемная задача: написать программу, которая находит минимальный элемент в списке.

! Проблемная задача: написать программу, которая находит два максимальных элемента в списке и два минимальных

Решение:

```
n = int(input('Enter number of elements in list (more than 2): '))
List = []
for i in range(n):
    List.append(int(input('Enter element: ')))


print('Your list:')
print(List)
```

```
# find two max elements
Max1 = List[0]
Max2 = List[1]
for i in range(2, n):
    if List[i] > Max1:
        if Max1 > Max2:
            Max2 = Max1
        Max1 = List[i]
    else:
        if List[i] > Max2:
            if Max2 > Max1:
                Max1 = Max2
```

print(Max1, Max2)

УРОК 7 сортировка списка

На этом уроке мы разберем следующий очень важный алгоритм – это алгоритм сортировки списка по убыванию и возрастанию.

Алгоритм сортировки по возрастанию основан на алгоритме поиска минимального элемента, а алгоритм сортировки по убыванию, соответственно основан на алгоритме поиска максимального элемента.

! Проблемная задача: придумать идею алгоритма сортировки списка.

Решение:

Сначала мы находим минимальный элемент в списке.

Затем меняем его местами с элементом номер 0 (ноль).

Далее находим минимальный элемент в списке без учета элемента 0 (ноль).

Затем меняем его местами с элементом номер 1

Далее находим минимальный элемент в списке без учета элементов 0 и 1

Затем меняем его местами с элементом номер 2

И т.д.

! Проблемная задача: реализовать алгоритм сортировки на Python

Решение:

```
N = int(input())
ListN = [0]*N
for i in range(N):
    ListN[i] = int(input())


index = 0
for i in range(N - 1):
    Min = ListN[i]


    # ищем минимальный элемент
    for j in range(i + 1, N):
        if ListN[j] < Min:
            Min = ListN[j]
            index = j


    # меняем местами текущий элемент и минимальный
    bufer = ListN[index]
    ListN[index] = ListN[i]
    ListN[i] = bufer
```

Как вы заметили, мы стали использовать новый символ в программе – это хэштэг **#.**

Этот знак используется для создания комментариев в программе. Все что написано после данного значка игнорируется при выполнении программы.

Комментарии очень важны, так как в больших программах помогают не запутаться. А также в профессиональной среде часто одну большую программу пишут сразу несколько программистов и потом передают свои программы друг другу и комментарии помогают быстро разобраться в чужом коде.

! Проблемная задача:

УРОК 8 функции

Основой любого языка программирования являются его команды. Эти команды еще по-другому называют – функции. Мы с вами уже изучили такие функции:

print()

input()

int()

range()

append()

Оказывается, в программировании можно создавать свои функции и это невероятно важно и полезно. В общем-то, это один из базовых инструментов, на котором основано все программирование! Даже простое приложение пришлось бы программировать долго и супер сложно, а сложные программы были бы в тысячи раз сложней для написания.

Сейчас мы с вами разберем как создавать свои функции, применим эти знания на примере и вам станет понятно почему это так важно и полезно.

Предположим, вам нужно написать большую математическую программу, которая может решать множество задач: решать уравнения, строить графики функций, решать различные задачки и т.д. Но как вы знаете из курса математики, многие задачи сводятся к решению линейного уравнения вида:

a***x** + b = 0, где a, b – это числовые коэффициенты.

И во многих местах программы пришлось бы прописывать код (команды) решения этого уравнения.

Это очень неудобно. И для этого можно создать свою функцию. На Python это делается с помощью специального оператора – **def.** После которого пишется имя вашей функции, после имени открываются скобки и в скобках указываются аргументы функции (как в математике). Аргументов может и не быть. А потом после двоеточия в следующих строках с отступом относительно слова **def** пишутся команды, которые должна выполнять функция. Разберем на примере:

```
def eqline(a, b):
    x = -b / a
    return x


# решим уравнение: 5x – 9 = 0
print('x =', eqline(5, -9))
```

Как видно мы создали функцию **eqline(a, b),** название от английского «линейное уравнение». У функции два аргумента – коэффициенты линейного уравнения. Слово **return** – означает, что то, что будет написано после него, вернет функция, то есть это и будет значение функции. Как раз это значение мы и выводим на экран внутри функции

Важный момент: аргументы a и b указанные в скобках – это ячейки памяти, которые принадлежат только самой функции **eqline(a, b)**. Из самой программы к ним невозможно получить доступ, они могут использоваться только внутри функции. Им можно только задать значения путем вызова функции. В нашем примере мы вызываем функцию с аргументами 5 и (-9), значит, что внутри функции ячейка памяти **a = 5,** а ячейка памяти **b = - 9**.

! Проблемная задача: написать функцию, которая возводит в степень число.

Вообще, слово **return,** которое возвращает значение функции, может отсутствовать, в таком случае функция ничего не будет возвращать, она просто будет что-то делать. Например, в Python есть такая функция, и вы с ней уже хорошо знакомы – это функция **print().** И в самом деле, она ничего не возвращает, а просто выводит на экран то, что находится у нее в скобках.

Кстати, то, что находится у любой функции в скобках – называется аргумент (как и в математике). У функции может быть хоть сколько аргументов, а может и вообще не быть аргументов. Например, функцию **input()** мы с вами использовали без аргументов, с пустыми скобками.

Давайте создадим функцию, которая будет делать что-нибудь полезное, но возвращать ничего не будет. К примеру, давайте создадим функцию **переводчик**, но для простоты, она у нас будет знать только 3-5 слов. Будем передавать в функцию в качестве аргумента слово на английском, а функция будет выводить на экран перевод.

! Проблемная задача: попробовать самим написать такую функцию.

Решение:

```
def translater(word):
    if word == 'ball':
        print('мяч')
    elif word == 'table':
        print('стол')
    elif word == 'car':
        print('автомобиль')
    elif word == 'house':
        print('дом')
    elif word == 'mouse':
        print('мышь')
    else:
        print('я не знаю такого слова')


print('Привет, я переводчик с английского на русский!')
w = input('Введите слово: ')
translater(w)
```

УРОК 9 работа с файлами

На этом уроке мы узнаем, как работать с текстовыми файлами с помощью команд языка Python.

которои находится программа), а если где-то в другом месте, то запомните или запишите полныи путь.

В файл запишите какой-нибудь текст. Я записал такой:

Привет!

Я файл.

После создания, перейдем в Python. Чтобы открыть файл, в Python есть стандартная команда -**open(file)**, где **file** – это строка с адресом файла (путь к файлу). Команда **open(file)** возвращает сам файл, точно также как команда **input()** возвращает строку, которую вводит (печатает) пользователь. Поэтому необходимо создать, как бы, переменную и присвоить ей команду **open()**. Рассмотрим на примере, чтобы стало понятнее:

file = open('MyFiles/myfile.txt')

Как видно, мы создали как бы переменную **file** и присвоили ей функцию **open(...)**. Почему «как бы переменную»? Потому что, на самом деле **file** – это, конечно, не переменная, а это уже **объект**, то есть объект – файл. Получается, что мы открыли файл и теперь в нашей программе он называется **file.** Мы могли бы назвать его и любым другим именем из английских букв. Например, вот так:

f = open('MyFiles/myfile.txt')

или вот так:

myfile1 = open('MyFiles/myfile.txt')

Как вы заметили, путь к файлу написан не полностью, потому что, папка **MyFiles** находиться в той же папке, где и наша программа. Если бы, папка **MyFiles** находилась бы в другом месте компьютера, например по адресу:

C:\Users\Asus\Documents\Scratch Projects

То в команде **open()** пришлось бы написать полный путь:

myfile1 = open(' C:/Users/Asus/Documents/Scratch Projects/MyFiles/myfile.txt')

Итак, файл мы открыли, теперь прочитаем из него информацию и выведем ее на экран. Это можно сделать несколькими способами.

Способ 1

С помощью функции **read().** Эта функция считывает из файла все содержимое, то есть весь текст.

file = open('MyFiles/myfile.txt')

date = file.read()

print(date)

Если в файле был русский текст, то всего скорее на экране мы увидим абракадабру:

РµСЂРёРІРµС‚!

Р Я С„Р°Р№Р».

Это произошло из-за того, что Python умолчанию настроен на английский текст, а для любых других языков компьютер использует кодировки, и для того чтобы, наша программа могла прочитать русский текст, необходимо в команде **open()** указать, что при открытии нашего файла необходимо использовать кодировку.

Разные программы используют разные кодировки, я создал файл в блокноте на Windows 10. Там используется кодировка **UTF-8.** Чтобы указать кодировку, у функции **open()** есть специальный параметр **encoding**, которому и нужно присвоить название кодировки. Вот так:

file = open('MyFiles/myfile.txt', encoding = 'UTF-8')

date = file.read()

print(date)

После этого на экран выведется нормальный текст:

Привет!

Я файл.

классной особенностью цикла **for.** Эта особенность заключается в том, что цикл **for** можно применять не только с функцией **range()** , но и с любым объектом, который состоит из отдельных элементов. Такими объектами являются:

- списки (они состоят из отдельных элементов)

- файлы (они состоят из отдельных строк)

Чтобы стало понятнее рассмотрим на примере, только для начала добавим строк в наш файл. Теперь он выглядит так:

myfile.txt

Привет!

Я файл.

Это моя 3-яя строка

А это 4-ая строка

Напишем вот такой цикл

```
file = open('MyFiles/myfile.txt', encoding = 'UTF-8')
for string in file:
    print(string)
```

На экран выведется:

Привет!

Я файл.

Это моя 3-яя строка

А это 4-ая строка

Обратим внимание на то, как мы использовали цикл **for**. Если перевести на человеческий язык, мы как-бы дали такую команду: пускай переменная **string** переберет все строки в объекте **file** (а это наш файл) и для каждой новой строки вывести ее на экран. То есть, сначала в 1 итерации (итерация – это повторение) цикла в переменную **string** попадет первая строка файла и выполниться команда **print**(**string**), потом во 2 итерации в переменную **string** попадет 2-ая строка из файла и снова выполниться операция **print**(**string**) и т.д. Таким образом выведутся на экран все строки файла.

Также, наверное, вы заметили, что после каждой строки вывелась пустая строка. Это произошло потому, что в текстовых файлах, на самом деле содержатся скрытые символы и один из таких символов – “**/n**”**.** Он обозначает перенос строки, или еще можно сказать, что этот символ как-бы обозначает нажатие клавиши **Enter** (ведь когда мы нажимаем клавишу **Enter** у нас добавляется новая строка). И для Python этот скрытый символ обозначает дополнительную пустую строку.

! проблемная задача: изменить цикл **for** так, чтобы на экран не выводились пустые строки.

Итак, мы научились считывать информацию из файла. Рассмотрим теперь, как можно записывать информацию в файл.

Для этого используется функция-метод файла – **write**(**string**), где **string** – это строка, которую мы хотим записать в файл.

Но тут есть один важный момент: в Python предусмотрены режимы работы с файлом. Если мы попробуем применить операцию **write()**, то выйдет ошибка. Для того чтобы можно было записывать информацию в файл, необходимо открыть его в режиме записи. Сделать это очень легко, нужно при открытии файла указать специальный параметр –

Чтобы изменения сохранились, необходимо обязательно после работы с файлом его закрыть с помощью команды **close().**

> Закрывать файл вообще рекомендуется после любого режима использования!
>
> Нельзя использовать одновременно два режима: чтение и запись.
>
> Открытие файла в режиме записи автоматически удаляет всю информацию из файла.

Если теперь мы откроем файл: **'MyFiles/myoutfile.txt'** через проводник Windows, то увидим в нем строку: «Здесь был Я».

! Проблемная задача: Мальчик Витя путешествовал в прошлом году по России. В файле input.txt записаны в отдельных строках названия городов, через которые проходил его маршрут. Известно, что Витя побывал в некоторых городах несколько раз. Необходимо найти повторяющиеся города и записать в файл output.txt список городов, в которых Витя был больше одного раза, а напротив каждого города указать число – сколько раз он был в этом городе.

Пример файлов:

Input.txt

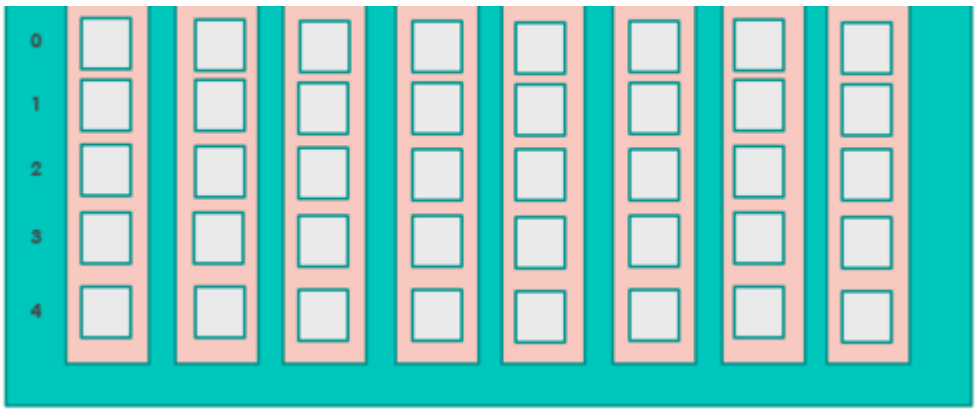
```
Рязань

Москва

Смоленск

Москва

Воронеж

Пермь

Челябинск

Тюмень

Челябинск

Югорск

Челябинск

Пермь

Пермь

Москва
```

output.txt

```
Москва 3

Челябинск 2

Пермь 2
```

Урок 10 двумерные списки

На одном из предыдущих уроков мы проходили списки. Как мы помним, список – это упорядоченные пронумерованные ячейки памяти, которые называются элементами списка. Оказывается, что элементами списка могут быть не только ячейки памяти (переменные), но и любые объекты, например, другие списки. Да, да, элементом списка может быть другой список. Схематично это можно изобразить вот так:



Такие списки называют двухмерными, так как, если смотреть визуально, у них получается, как на плоскости – два измерения, по горизонтали и по вертикали, и у каждого элемента теперь номер задается двумя числами. Проще понимать его как таблицу, тем более что двумерный список и выглядит как таблица.

Давайте создадим такой список, и все посмотрим на примере:

A = [['Москва', 16000, 2561, 1147],

['Воронеж', 1060, 596, 1586],

['Новосибирск', 1600, 502, 1893]]

Как видно мы создали список, в котором три элемента, каждый их этих трех элементов, тоже список из 4-х элементов, которые обозначают:

- название города

- население в тыс.чел.

- площадь в кв.км.

- дата основания

Как вы заметили сам список мы записали не в одну строку, как, казалось бы, положено, а в три. В Python некоторые записи в командах можно переносить на следующую строку, например, сами списки или особо длинные условия (например составное условие у оператора **if**).

Чтобы обратиться к конкретному элементу двумерного списка, необходимо указывать два индекса:

print(A[1][3])

На экран выведется:

1586

! Проблемная задача: вывести на экран список с информацией о городах в понятном для пользователя формате, в виде такой таблицы:

Город	Население (тыс. чел.)	Площадь (кв. м.)	Год основания
Москва	16000	2561	1147
Воронеж	1060	596	1586
Новосибирск	1600	502	1893

Думаю, при выводе такой таблицы возник вопрос, как сделать так, чтобы все значения в столбиках были ровно друг под другом, то есть не были сдвинуты в разные стороны. Особенно это нужно, когда вы не знаете, какой длинны будут названия городов и числа, если, например, список будет заполнять пользователь.

В Python для этого есть классная функция – **ljust (num, char)**. Эта функция является методом, то есть вызывается как функция строки через точку, возвращает эта функция новую строку, которая состоит из старой строки и символов, задаваемых параметром **char**, **num –** это количество символов в новой строке. Рассмотрим на примере:

str1 = 'Hello,'

str2 = 'Good day,'

str3 = 'Bye,'

print(str3.ljust(10, ' '), 'Python')

Результат:

Hello, Python

Good day, Python

Bye, Python

Думаю, тут без комментариев все понятно.

! Проблемная задача: используя функцию **ljust** написать программу, которая запрашивает у пользователя информацию о каких либо объектах, по аналогии с городами (например: страны, известные личности, игровые персонажи, смартфоны и т.д.) заносит их в двумерный список и выводит его на экран в виде таблицы с заголовками.

Решение:

```
n = int(input())
DB = []
Str = []
for i in range(n):
    name = input('Введите название планеты: ')
    Str.append(name)
    print('Введите информацию о планете', name)
    s = input('Введите массу планеты (тысяч тонн): ')
    Str.append(s)
    s = input('Введите количество спутников у планеты: ')
    Str.append(s)
    s = input('Введите название звезды, вокруг которой обращается планета: ')
    Str.append(s)
    DB.append(Str)
    Str = []

for Str in DB:
    print(Str[0].ljust(10,' '), Str[1].ljust(10,' '),
        Str[2].ljust(5,' '), Str[3].ljust(5,' '))
```

Урок 11 множества

На этом уроке мы познакомимся с множествами. Множество – это структура неупорядоченных и неповторяющихся данных. Можно представить, что это тоже набор ячеек памяти или элементов, как и список, только элементы множества не имеют номеров и в множестве все элементы разные или, говоря по-другому, нет одинаковых элементов. Создать множество можно, например, так:

a = {2, 5, 1, 23, 3}

Похоже на то, как создается список, разница лишь в скобках, у списка скобки квадратные, а у множества фигурные.

a = {2, 6, 6, 2}

print(a)

То на экран выведется:

{2, 6}

Это говорит о том, что множество **a** состоит только из двух элементов: **a = {2, 6}.**

Такое свойство множества очень полезно для обработки большого количества информации. Например, представьте, что на въезде в город установлена видеокамера, которая фиксирует марки машин и заносит их в список. В конце дня вам нужно определить какие марки машин заехали в город. Ясно, что в списке будет много повторяющихся элементов и перебирать весь список, проверять каждую марку: была ли она уже раньше – это очень неудобно. С помощью множества это делается одной командой – **set ()**.

Если применить эту команду к списку, то она вернет множество из элементов списка и, конечно, в нем не будет повторяющихся элементов. Рассмотрим на примере:

Список зафиксированных видеокамерой марок автомобилей:

ListOfCar = ['Audi', 'Lada', 'Audi', 'BMW',

'Lada', 'Lada', 'Honda', 'Lada',

'BMW', 'Lada', 'Kia']

Создаем множество и выводим его на экран:

SetOfCar = set(ListOfCar)

print(SetOfCar)

Результат:

{'Lada', 'Kia', 'Honda', 'BMW', 'Audi'}

Как видно, одинаковые элементы отсутствуют.

С помощью команды **set()** можно создать пустое множество (в котором нет ни одного элемента):

a = set()

Теперь рассмотрим основные операции с множествами.

Чтобы добавить элемент в множество в Python есть функция – **add (elem)** и она является методом (самостоятельно не существует, а вызывается у множества через точку). Где **elem** – элемент, который вы хотите добавить. Пример:

a = {1, 5}

a.add(9)

print(a)

На экран выведется:

{1, 5, 9}

Чтобы удалить элемент из множества используется команда – **remove(elem).** Пример:

a = {3, 5, 1, 6, 12}

a.remove(5)

print(a)

Результат:

Еще есть команда – **discard (elem)**, она делает тоже самое, что и **remove (elem)**, только разница в том, что если **remove (elem)** применить к пустому множеству, то будет ошибка, а команда **discard (elem)** не выдаст ошибку.

Также в Python можно проверять есть ли в каком-либо множестве определенный элемент с помощью оператора – **in**. Пример:

```
a = {3, 5, 1, 6, 12}
if 5 in a:
    print('Yes')
else:
    print('No')
if 23 in a:
    print('Yes')
else:
    print('No')
```

Результат:

Yes

No

! Проблемная задача: написать программу, которая отбирает кандидатов на участие в конкурсе по игре в шахматы. Конкурс проводится среди младшего возраста: 5 и 6 лет, а также среди старшего – кому 13 или 14 лет. То есть в конкурс допускаются только те кому 5 лет, 6 лет, 13 или 14 лет. Необходимо отобрать 10 человек. Программа запрашивает у пользователя возраст, если возраст подходит, то программа запрашивает имя и добавляет кандидата в список участников, пока участников не наберется 10.

Примечание, необходимо создать множество {5, 6, 13, 14} и проверять с помощью оператора **in** входит ли в это множество возраст очередного кандидата.

Решение:

```
Set = {5, 6, 13, 14}


ListCont = []
L = []
cont = 0
while cont != 10:
    name = input('Enter your name: ')
    age = int(input('Enter your age: '))
    if age in Set:
        L.append(name)
        L.append(age)
        ListCont.append(L)
```

```
print('List of contestant: ')

for i in range(10):

    print(ListCont[i][0].ljust(10, ' '), ListCont[i][1])
```

УРОК 12 защита информации. шифрование

Сегодня мы познакомимся с таким понятием, как шифрование. Когда вы общаетесь в интернете, все сообщения передаются в зашифрованном виде. Это делается для того, чтобы никто посторонний не мог прочитать ваши сообщения. Наверное, вам было бы неприятно, если любой человек мог прочитать то, что вы пишете. Вообще шифрование возникло сразу, как только люди стали обмениваться между собой сообщениями через электронные каналы связи. Особенно это актуально было во время войн, когда военные подразделения передавали важную информацию друг другу. Если бы противник перехватил сообщение: «Завтра планируем наступление на врага в селе Кукшкино», то он бы подготовился, и атака бы провалилась. В наше время любое сообщение шифруется.

Давайте рассмотрим, как это происходит. Для простоты возьмем 10 букв:

А, О, Е, Р, С, П, Т, Н, И, Ь

Такой набор называется **алфавит**.

Из них мы будем составлять слова, а из слов сообщения.

Например, нам нужно передать такое сообщение:

ПОРА ПЕТЬ ПЕСНИ

Чтобы это сообщение не могли понять, те кто захочет его перехватить, необходимо его закодировать.

ort! Проблемная задача: предложить способ кодировки сообщения.

Предложенные способы можно свести к тому, что мы каждую букву заменяем на определенный код, например вот так:

А – 1 О – 2 Е – 3 Р – 4, С – 5, П – 6, Т – 7, Н – 8, И – 9, Ь – 10

Такой набор называется – **ключ шифра**. А коды, которые соответствуют буквам алфавита называются **кодовыми словами**. То есть, **1, 2, 3, … 10** – это **кодовые слова**. Он является секретным, и, если кто-то узнает его, он сможет расшифровать перехваченное сообщение.

Итак, зашифруем наше сообщение **ПОРА ПЕТЬ ПЕСНИ:**

6241 63710 63589

Как видно, в таком виде вообще непонятно, что это за сообщение. Но тут есть один слабый момент, если попробовать его расшифровать (такой процесс называется дешифровка), то мы столкнемся с тем, что символы **10** можно расшифровать двояко:

Либо – **Ь,** либо – **АО,** то есть у нас может получиться два варианта расшифрованного сообщения:

- **ПОРА ПЕТЬ ПЕСНИ**
- **ПОРА ПЕТАО ПЕСНИ**

Поэтому, в науке шифрования существует такое правило, которое называется «**правило Фано**». Оно говорит о том, что никакое кодовое слово не должно являться началом другого кодового слова.

А в нашем примере, кодовое слово для буквы **А** – это **1,** и оно является началом кодового слова для буквы **Ь** – **10**

ort Проблемная задача: исправить ключ шифра для нашего примера, чтобы он соответствовал условию Фано.

Решение:

А – 11 О – 2 Е – 3 Р – 4, С – 5, П – 6, Т – 7, Н – 8, И – 9, Ь – 10

ort

Рещение:

```
def decode(code, w):
    res = ''
    for elem in code:
        if elem[1] == w:
            res = elem[0]
    return res


code = [['a', '0000'],
        ['п', '0001'],
        ['к', '0010'],
        ['т', '0011'],
        ['е', '0100'],
        ['о', '0101'],
        ['у', '0110'],
        ['в', '0111'],
        ['м', '1000'],
        ['р', '1001'],
        [' ', '1010']]


message = input()


result = ''
word = ''
i = 0
for sign in message:
    word = word + sign
    i = i + 1
    if i == 4:
        result = result + decode(code, word)
        i = 0
        word = ''
print(result)
```

УРОК 13 конструкция **try except**

Формула для расчёта индекса массы:

Индекс массы = Масса / (Рост * Рост), где Масса в кг, а Рост в метрах.

Программка будет выглядеть вот так:

height = int (input('Введите ваш рост: '))

weight = int (input('Введите ваш вес: '))

index = height / (weight * weight)

print (index)

А теперь попробуйте ввести вместо чисел какую-нибудь букву или сочетание букв, например, пользователь может подумать, что ему нужно еще добавить единицы измерения и введет:

180 кв.м.

Программа выдаст ошибку:

ValueError: invalid literal for int() with base 10: '180 кв.м'

Функция **int()** можно применять только к строке, состоящей исключительно из цифр.

Это, конечно, недопустимо, если вы пишете программу для широкого круга лиц. Вообще, говорят, что программа должна обладать дружелюбным интерфейсом. Я думаю, каждый из вас ничего хорошего не подумал бы про приложение, которое вы установили бы на смартфон, и оно вылетело с какой-нибудь непонятной ошибкой и, наверное, сразу бы его удалили.

Чтобы исключить такие ситуации в Python есть специальная конструкция из операторов **try except.**

Рассмотрим на нашем примере, как она используется:

```
try:
    height = int (input('Введите ваш рост: '))
    weight = int (input('Введите ваш вес: '))
except:
    print ('Рост и вес необходимо вводить числами без букв')
    height = 0
    weight = 1

index = height / (weight * weight)
print (index)
```

Как видно в блоке **try** мы пишем те операции, которые потенциально могут выдать ошибку. А в блоке except пишем те операции, которые должны выполниться в случае возникновения ошибки.

В нашем, случае, чтобы не исправлять строки:

```
index = height / (weight * weight)
print (index)
```

Команды, которые будут находится в блоке **else**, выполнятся только в случае удачного выполнения блока **try**. Рассмотрим на нашем примере:

```
try:
    height = int (input('Введите ваш рост: '))
    weight = int (input('Введите ваш вес: '))
except:
    print ('Рост и вес необходимо вводить числами без букв')
else:
    index = height / (weight * weight)
    print (index)
```

В таком случае уже необязательно в блоке **except** присваивать значения переменным **height** и **weight**, так как операции:

```
index = height / (weight * weight)
print (index)
```

находятся в блоке **else** и выполнятся только в случае удачного выполнения блока **try**, то есть пользователь введет правильные значения роста и веса.

! Проблемная задача: исправить программу из нашего примера, чтобы, в случае введения пользователем неверных данных, программа выводила сообщение:

«Рост и вес необходимо вводить числами без букв»

И заново запрашивала данные роста и веса. И так до тех пор, пока пользователь не введет правильные значения.

УРОК 14 бесконечный цикл

Мы все с вами каждый день пользуемся разными программами, на компьютере, смартфоне, планшете или ноутбуке и у всех у них есть кое-что общее – это то, что пока мы их не закроем сами – они не закрываются. На компьютере, для этого нужно нажать крестик в правом верхнем углу окна программы, на смартфоне необходимо смахнуть в вверх или в сторону окно программы, где-то просто нажать кнопку **ВЫХОД.**

Наши программы пока выполняются с использованием консольного интерфейса, такой интерфейс имеет программа «**Командная строка**» в **Windows** или «**Терминал**» на **iOS**. Например, чтобы выйти из программы «Командная строка» необходимо ввести команду – **exit**.

Давайте сделаем также и мы, чтобы наши программы тоже имели профессиональный интерфейс и выходили не сами по себе, а тогда, когда пользователь напишет команду – **exit** или **quit**.

! Проблемная задача: на основе программы, которая рассчитывает индекс массы тела с прошлого урока, написать программу, которая ждет команду от пользователя, если пользователь вводит **exit**, то программа закрывается, если пользователь вводит команду **index**, то программа запрашивает рост и вес, затем рассчитывает индекс массы. Если пользователь вводит что-то другое, программа сообщает, что не знает такую команду и снова ждет ввода команды.

Решение:

```
def indexmass():
    try:
        height = int (input('Введите ваш рост: '))
```

```
    except ValueError:
        print ('Рост и вес необходимо вводить числами без букв')
        return indexmass()
    except ZeroDivisionError:
        print ('Значения роста и веса не могуть быть нулевыми')
        return indexmass()
    else:
        return index


work = True
while work:
    command = input('')
    if command == 'exit':
        work = False
    elif command == 'index':
        print(indexmass())
    else:
        print('Такой команды нет')
```

! Проблемная задача: добавить еще несколько команд в программу, например:

- команда convert (конвертирует разные единицы измерения в другие, метры – в футы, градусы в фаренгейты и т.д., на свой выбор)

- команда zodiac (определяет знак зодиака по дате рождения)

УРОК 15 дополнительные команды и фишки в **python**

На этом уроке мы рассмотрим еще несколько разных интересных команд и фишек в Python, а также выберем каждый себе проект – программу. Проект программу можно выбрать по своему интересу из предложенных либо придумать самому.

Возвести число a в степень n:

```
a**n

print(5**3)

125
```

Команду:

```
a = a + 1
```

можно записать кратко:

```
a++
```

Команду:

a = a + b

можно записать кратко:

a += b

Фнукция **len**(**List**) определяет количество элементов в списке **List**

List = [2, 4, 2, 12, 56]

print(len(List))

Выведется:

5

Чтобы перебрать все элементы списка, можно использовать цикл **for** таким образом:

for I in range (len(List)):

    print(List[i])

а можно и так:

for elem in List:

    print(elem)

Функция **list**()

Если вы хотите из строки сделать список, элементами которого будут символы этой строки, причем в том же порядке, как и в строке, то можно применить к строке функцию **list**(**string**)

Пример:

Str = 'This is iphone 11'

List = list(Str)

print(List)

Выведется:

['T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'i', 'p', 'h', 'o', 'n', 'e', ' ', '1', '1']

Метод **split**()

Если вызвать у строки метод – **split**(), то он вернет список, элементами которого будут слова, между которыми в строке был пробел. То есть, этот метод разделяет строку на отдельные слова, а символом разделения является пробел.

Пример:

Str = 'This is iphone 11'

List = Str.split()

print(List)

Выведется:

['This', 'is', 'iphone', '11']

Варианты проектов:

- база данных

- игра крестики-нолики

- игра викторина

- переводчик

- программа, определяющая характер человека

- помощник по математике

- игра морской бой

После выбора тему, ученики могут приступить к выполнению проекта, а преподавателю предлагается индивидуально с каждым учеником обсудить нюансы проекта, составить план, обсудить с чего начать, возможно что-то подсказать.

УРОК 16 презентация проектов

На этом уроке ученики презентуют свои проекты. Каждый выходит и либо с помощью проектора, либо показывая на экране компьютера (в этом случае все собираются у экрана) рассказывает о своей программе: для чего она нужна, какие действия выполняет, в чем ее польза и где может применяться.

Общий план презентации проекта можно представить в виде плана:

- Рассказ о программе
- Демонстрация ее работы
- Тестирование программы остальными учениками (по желанию)
- Демонстрация кода программы
- Рассказ о том, какие используются в программе изученные команды и алгоритмы

Рекомендуется мотивировать учеников задавать вопросы выступающему.

Также, для развития навыка выхода из сложных ситуаций, можно попросить ученика рассказать детально о какой-нибудь части программы, функции.

Можно спросить о том, какие пришлось преодолеть сложности, какой участок кода вызвал наибольшие проблемы.