1. **Начало работы**

**Что такое Python**

Python – популярный язык программирования. Создан в 1991 году Гвидо ванРоссумом.

Сейчас используется для:

* веб-разработки (серверной),
* разработки программного обеспечения,
* Datascience,
* парсинга.

**Какие возможности дает Python**

* может использоваться на сервере для создания веб-приложений.
* может использоваться вместе с программным обеспечением для создания рабочих процессов.
* подключается к системам баз данных. С его помощью можно также читать и изменять файлы.
* может обрабатывать большие данные и выполнения сложных математических вычислений.
* используется для быстрого прототипирования или для разработки готового программного обеспечения.

Для начала создадим новую папку, Base:

New🡪Directory🡪Имя папки. Щёлкаем правой кнопкой на папку🡪New🡪Python File🡪 Имя файла. Например: Урок 1.

print(“Hello world!”) *Hello world!*print(‘ “Hello world!” ‘) *”Hello world!”*
print(“Hello ‘\” ‘ world!”) *Hello‘ “ ‘ world!*print(“Hello “\\” world!”) *Hello “\”world!*

**Вычисления в Python:**

print(“2+3”) *2+3*
print(2+3) *5*
print(“Hello, “,”Alex”) *Hello, Alex*

пробел можно заменить sep =”-“
print(“Hello,” “Alex”, “and Misha”,sep=”-“) *Hello,-Alex-and-Misha*

*Чтобы текст перенести на следующую строку добавляем:* ***\n***print(“Hello, \n Alex”) *Hello
 Alex*

1. **Переменные и их типы**

В отличие от других языков программирования, Python не имеет команды для объявления переменной. Переменная создается тогда, когда вы назначили ей значение

**Имя переменной**

Переменная может иметь краткое имя (например,x и y) или более содержательное имя (age, carname, total\_volume).
*Правила для* [*переменных в Python*](https://pythonru.com/uroki/python-dlja-nachinajushhih/2-peremennye)*:*

* Имя переменной должно начинаться с буквы или символа подчеркивания.
* Оно не может начинаться с числа.
* Имя переменной может содержать только буквенно-цифровые символы и символы подчеркивания (Az, 0-9 и \_)
* Имена переменных чувствительны к регистру (age, Age и AGE — три разных переменные)

**Помните, что переменные чувствительны к регистру**

В Питоне переменную объявляют, присваивая ей какое-нибудь значение. Для объявления переменной достаточно просто указать имя и поставить знак “=” (оператор присваивания) и написать значение, которое будет храниться в переменной.

Смысл оператора присваивания отличается от знака равно. В программе выражение “х=1” означает, что теперь в переменной х- хранится значение 1.

В переменных можно хранить не только числа, но и строки.

Когда переменная объявлена и ей присвоено значение, то имя этой переменной можно указать в коде. А в результате будет подставляться ее значение.

name = “Анфиса”
job = “персональный помощник”
print(“Привет, я “+name+ “, твой “ +job+ “! “)*Привет, я Анфиса, твой персональный помощник!*

Типы переменных

2. Дробные (float):
8.3
91.2
17.8

4.Логические: (bool)
True (Истина)
False (Ложь)

3.Строковые (str)
“Hello”
“Python in cool”
“Today is Friday”

1.Целые числа(int):
5
15
17957

Создадим переменную:

age = 25 #int
print(age) *25*print(“Возраст: “, +str(age)) *Возраст: 25*либо, кратко:
print(“Возраст: “, age) *Возраст: 25*

**Функция str принимает параметр и преобразует его в строку.**

temp = 25.9 #float
print(“Температура:”, temp , “градусов. “) *Температура: 25.9 градусов.*

**Преобразование строковых переменных**

username = “Alex:” #str
print(“Имя пользователя: “, username) *Имя пользователя: Alex*

**Преобразование логических переменных**

isexists = True # bool
print(“Существует: “, isexists) *Существует: True*

*Поменяем значение переменной:*

isexists = False
print(“Существует: “, isexists) *Существует: False*

Любая программа выполняется последовательно.
Есть еще одна полезная функция – она позволяет определять тип переменной.

print(“ Тип переменной age:“, type (age)) *Тип переменной age: <class ‘str’>*
print(“ Тип переменной temp:“, type (temp)) *Тип переменной temp: <class ‘float’>*
print(“ Тип переменнойusername:“, type (username))
 *Тип переменной username: <class ‘bool’>*

*Можно вписать целые числа:*print(“ Тип переменной int:“, type (10)) *Тип переменной int: <class ‘int’>*

1. **Комментарии**

**Комментарии** – это способ выражения того, что делает программа. Отмеченные строчки, которые комментируют код. В Питоне они бывают двух типов: *однострочные и многострочные.* Для добавления комментария нужно начинать каждую строку с # и одного пробела.

**Однострочные** – такой тип комментариев нужен для написания простых комментариев во время отладки. Они начинаются с символа #, и заканчиваются автоматически.

*# Хороший код документируется*.

При добавлении комментария нужно убедиться, что у него такой же уровень как и у кода.

Многострочные – позволяют писать комментарии на нескольких страницах. Такой тип подходит для написания чего-то более сложного.

#*Чтобы выучить язык, вы должны следовать этим правилам:
# 1. Знать синтаксис, типы данных и структуру;
# 2. Изучить обработку ошибок и ввод/вывод;
# 3. Читать о продвинутых структурных данных.*

**Docstring в Python**

В Python есть такая особенность, как задокументированные строки (docstring). С их помощью программисты могут быстро добавлять комментарии для каждого модуля, функции, метода или класса.

Задать docstring можно с помощью строковой константы. Она обязана быть первой инструкцией в определении объекта.

У docstring более широкая область применения, чем у комментария. Она должна описывать, что делает функция, а не как. Хорошей практикой считается добавление таких строк в каждую функцию программы.

**Как задать docstring в Python?**

Задать docstring в Python можно с помощью тройных кавычек. Нужно добавить один набор в начале и еще один – в конце. Docstring также могут занимать по несколько строк.

*Примечание:* строки с тремя кавычками также являются docstring в Python, пусть они и могут казаться обычными комментариями.

**В чем отличие между комментарием и docstring?**

Строки, начинающиеся с тройной кавычки, — это все еще обычные строки, которые могут быть написаны в несколько строк. Это значит, что они все еще являются исполняемыми инструкциями. Если же у них нет метки, значит сборщик мусора уничтожит их после исполнения.

Интерпретатор Python не будет игнорировать их так же, как комментарии. Но если такая строка расположена сразу же после объявления функции или класса в верхней части модуля, то она станет docstring. Получить к ним доступ можно следующим образом — myobj.\_\_doc\_\_.:

def the Function():
 '''
 Эта функция демонстрирует использование docstring в Python.
 '''
print("docstring python не являются комментариями.")
print("\nВыведем docstring функции...")
print(the Function.\_\_doc\_\_)

**Комментарии и docstring** добавляют ценности программе. Они делают программы более читаемыми и пригодными для последующей поддержки.
Поэтому всегда добавляйте осмысленные комментарии и docstring, потому что они упрощают процесс взаимодействия.

Можно через переменную вывести многострочные комментарии:

text = ‘“ Первая строка
 Вторая строка
 Третья строка”’

1. **Арифметические операции**

Существует три типа данных:

- Целые числа (int);
- Вещественные (float);
-Комплексные (complex).

Если в арифметическом выражении используются только целые числа, то результат будет целое число. Исключение операции деления.
при совместном использовании целых и вещественных чисел. Результат будет вещественным.

|  |  |  |
| --- | --- | --- |
| Арифметические операции: | Ввод | Вывод |
| x =59 | print(“x = “, x) | x = 59 |
| y = 20 | print(“y = “, y) | y = 20 |
| Сложение | print(“x+y =”, x+y) | x +y = 79 |
| Вычитание | print(“x-y = ”, x-y ) | x – y = 39 |
| Умножение | print(“x\*y = ”, x\*y) | x \* y = 1180 |
| Деление | print(“x/y = ”, x/y ) | x / y = 2.96 |
| Остаток от деления | print(“x%y=”, x%y) | x % y =19 |
| Целая часть | print(“x // y=”, x // y ) | X // y = 2 |
| Возведение в степень | print(“x\*\*y =”, x\*\*y) | X \*\* y = 26124033550 |
| Возведение в степень | print(“2\*\*3 =” 2\*\*3) | 2 \*\* 3 = 8 |
| Выражение в скобках | print(“(2+3\*3)\*2=”,(2+3\*3)\*2) | (2+3\*3)\*2 = 22 |

**Работа с комплексными числами**

Для создания комплексного числа можно использовать функцию complex (а,b), в которую к качестве первого аргумента передается действительная часть, а в качестве второго – мнимая. (a + b).

|  |  |
| --- | --- |
| z = 1 + 2j |  |
| x = complex(3, 2) |  |
| print(z) | *(1 + 2j)* |
| print(x) | *(3 + 2j)* |

**Комплексные числа можно складывать, вычитать, делить, умножать, возводить в степень.**

|  |  |  |
| --- | --- | --- |
| print(x + z) | Сложение | *( 4 + 4j)* |
| print(x – z) | Вычитание | *( 2 + 0j)* |
| print(x \* z) | Умножение | *( -1 + 8j)* |
| print(x / z) | Деление | *(1.4 – 0.8j)* |
| print(x \*\* z) | Возведение в степень | *(-1.112272.. – 0.126351..j)* |
| print(x \*\* 3) | Возведение в степень | *(-9 +46j)* |

**Биты –** это числа, которые находятся в двоичной системе счисления.
Принимают всего два значения 0 и 1, т.е.
0-0; 6-110
1-1; 7-111
2-10; 8-1000
3-11; 9-1001 *bin(ob)-Двоичная система счисления*
4-100 10-1010 *hex(ox)-16тичная*5-101 11-1011 и тд *oct(0o)-8тичная*

Битовые операции

|  |  |
| --- | --- |
| print(“x=”, x, “=”, bin(x)) | x = 5= 0b 101 |
| print(“x=”, x, “=”, hex(x)) | x = 5 0x 5 |
| print(“x=”, x, “=”, hex(20)) | x = 5 0x 14 |
|  |  |
| print(“20 = ”, hex(20)) | 20 = 0x 14 |
| print(“10 = ”, oct(10)) | 10 = 0o 12 |
|  |  |
| print(“x=”, x, “=”, bin(x)) | x = 5 = 0b 101 |
| print(“x=”, y, “=”, bin(y)) | y = 7 = 0b 111 |
| print(“x | y =”, bin( x | y ) | x | y = 0b 111 |
|  |  |
| print(“x & y = ”, bin(x & y) | x & y =0b 101 |
| print(“x ^ y = ”, bin(x ^ y) | x ^ y = 0b 10 |
| print(“x ~ y = ”, bin(x ~ y) | ~x = -0b 110 |
| print(“x << y = ”, bin(x << y) | x << 1 = 0b 1010 |
| print(“x >> y = ”, bin(x >> y) | x >> 1 = 0b 10 |

Чтобы перевести систему из 10тичной в двоичную выбираем функцию “bin”.
Приставка 0b – означает двоичное число, и читается как 1-01, а не 101.

1. **Логические операции.**

Такие же операции существуют и для переменных bool, которые называются логические операции. Создадим две переменные:

b1 = True
b2 = False
print(“b1 = ”, b1 ) *b1 = True*print(“b2 = ”, b2 ) *b2 = False*

Приступим к логическим операциям:

* + - * 1. Логическое “or” (или) даёт нам True, если хотя бы один из операторов будет True.Если обе переменные будут False, то результат будет False. Аналогично True.

print("b1 or b2 = ” , b1 or b2 ) *b1 or b2 = True*

2) Логическое “and” (и) возвращает True, тогда и только тогда когда оба оператора имеют значение True. Если значения разные, то результат –False.

print("b1 and b2 = ” , b1 and b2 ) *b1 and b2 = False*

3) Логическое “not” (отрицание), меняет значение на противоположное

print(“not b1 = ”, not b1) *not b1 = False*

4) Исключающее “!=”(или), Trueбудет в том случае, если значения различны.,
т.еb1 125b2.

print(“b1 != b2 =”, b1 != b2) *b1 != b2= True*

5) Проверка на равенство обозначается “= =” , не путаем со знаком “=” - это знак присваивания.
print(”b1 = = b2 =”, b1 = = b2) b1 = = b2 = False

Создадим новые переменные:

x = 5
y = 7
print(“x =”, x) *x = 5*
print(“y = ”,y) *y =7*
*Проверим какое значение больше*:
print(“x>y =”, x>y) *x>y = False*
print(“x<y =”, x<y) *x<y = True*
print(“x and b1 or (x> 10) = ”, x and b1 or (x> 10)) *x and b1 or (x> 10) = True*
*В начале выполняется действие в скобках*.
x> 10, т.е 5 неравно 10, то False, далее х=5 тогда, когда 5 преобразуется в булинскую переменную, в результате True (Любое число кроме ноля превращается в True, а ноль превращается False ) b1 –True.
*True и True или False: =>True или False =>True*
True будет тогда. Когда хоть один из операторов будет True.

print(“x > 10 or y < 7 = ”,x > 10 or y < 7) *x > 10 or y < 7 = False*

False or False = False. т.к оба оператора False.

1. **Строковые операции**

На этом уроке мы поработаем со строковыми операциями. Их немного, но не путаем с функциями. Давайте разберем основные операции создадим 2 строки.

mystr1 = “abc”
mystr2 = “xyz”

Выведем результат, который называется *конкатенация,* что означает сложение, когда к одной строке прибавляют другую.

print(“Конкатенация строк mystr1 и mystr2 = ”, mystr1 + mystr2)

*Конкатенация сток mystr1 и mystr2 = abcxyz*

Теперь давайте разберем пример, как принимать и считывать данные, посмотрим, что с ними можно сделать.

Мы попросим пользователя ввести два числа, сначала одно, потом второе. Далее выведем ему сумму этих чисел и сумму этих строк.

Создадим новую переменную с использованием функции *input.*

**input-** это стандартная функция, её задача это передача данных введенных пользователем в функцию.

*number1 = input(“Введите первое число:”)
print(“Вы ввели: ”, number1)

number2 = input(“Введите второе число:”)
print(“Вы ввели: ”, number2)*

По просьбе вводим первое число 15, далее ENTER, и вводим второе число. Пока мы не введем числа. операция не завершится.

А теперь сложим числа и строки :

*print(“Сумма строк =”, number1 + number2)*

*print(“Сумма чисел = ”, int(number1) + int(number2))*

Вводим числа в программу, нажимаем ENTER, смотрим результат.

1. **Условный оператор.**

**Условный оператор или оператор ветвления**- это оператор, конструкция языка программирования, обеспечивающая выполнение определённой команды (набора команд) только при условии истинности некоторого логического выражения, либо выполнение одной из нескольких команд (наборов команд) в зависимости от значения некоторого выражения.

Оператор ветвления **if** позволяет выполнить определенный набор инструкций. в зависимости от условия.

Конструкция **if**:
 **if выражение:**

 **инструкция\_1**

 **инструкция-2 Блок**

В Питоне принято делать отступ 4 пробела, или кнопкой TAB(табуляция)

До сих пор, когда мы писали какую-нибудь программу, мы знали, что выполняться все инструкции будут по очереди.

Условный оператор помогает определять нужно ли вообще какие-то инструкции выполнять или нет. Они зависят от условий, которые мы укажем.

*print(“Введите 0,1 или 2: ”)
a = input( )
if a = = “0”:
 print(“Вы ввели ноль”)*

Проверяем: вводим ноль в нижнем окне.
если ввели другое число, то строка уже не выполнена.

**Обязательно после логического выражения ставится двоеточие, следующая строка автоматически получается с отступом.**

Давайте выполним еще две команды: (с тем же отступом)

*print(“Это число меньше 10”)*

Вводим число: если 0, то выполняются оба условия.
 если 11 и больше, то не выполнено не одно.

Если убрать отступ, то первое строка не выполнится, а вторая выводиться. Т.к все находится внутри одного блока **if**.

Усложним данный блок. теперь а не равна 0, но равна 1.
Для этого есть команда **elif (**Ставится не отступов)

*elif = = “1”:
 print(“Вы ввели один”)*

После того, как мы ввели 1, то первое условие (**if а = = “0”:)** является ложным False, интерпретатор переходит ко второй команде (**elif**) и ее проверяет. Если условие истинное True, то попадает на print(“Вы ввели один”).
Если первое условие истина, то второй блок пропускается.

**продолжим:**

*elif а = = “2”:
 print(“Вы ввели два: ”)*

Следующий блок выполняется в том случае, если предыдущие были ложными.

*else:
 print(“Некорректный ввод”)*

вводим числа, и проверяем.
также в этот блок можно добавить переменные.

*cont= a == “0” or a == “1” or a == “2”*

Переменная *cont* примет значение True, если мы введем значение 0,1 или 2. И будет False, если что- то другое.

*if cont:
 x = 0
else:
 x = 3*

*print(“x =”, x )*

Переменная х = 0. если это значение ложь, то интерпретатор переходит к следующей инструкции х = 3.
Если введем другое значение, то х = 3.

Эту запись можно упростить и записать в одну строчку:

*x = 0 if cont else 3
print(“x=”, x)*

Проверяем: вводим 0, 1, 2, abc.

1. **Циклы while.**

Циклы- важная вещь в программировании. они позволяют выполнять одну и ту же инструкцию множество раз.

Рассмотрим первый цикл while. Этот цикл требует некоторое условие, которое может быть и True, и False. Если будет ложным, то выйдет из цикла.

*i = 0
while i< 10:
 i += i + 1
 print(“Hello world”)
print(“Цикл завершён”)*

Пока *i< 10,* мы выполняем следующие инструкции: (*i += i + 1)* –команда бесконечности. начиная с нуля, каждому значению прибавляем единицу. (0+1=1; 1+1=2 и т.д. Пока условие выполняется, цикл продолжается. Когда *i* будет *=* 10, условие ложное. Программа выходит из цикла и попадает в инструкцию : *print(“Цикл завершён”)*Запускаем. Смотрим результат.

 *1*
**Пример 2:** *2*
*print(“----“) 3
i = 0 4
while i< 10: 5
 i += 1 6
 print(i)*  *7*
 *8*
 *9*
 *10*

**Пример 3:**

*i = 0 1
while i< 10: 2
 i += 1 3
 if i == 5: 4
 continue 6
 if i == 8: 7
 break Цикл завершен, i = 8
 print(i)
print(“Цикл завершён, i = ”, i )*

**4.Сумма чисел:**

*a = 0
х = 1
to = 10
while x <= 10:
 a += x
 x += 1
print(“Сумма чисел от 1 до ”, to , “равна ”, a)*

 *Сумма чисел от 1 до 10 равна 55
while True:
 code = input(“Введите 0 для выхода:”) Введите 0 для выхода:* ***545*** *if code == “0”: Введите 0 для выхода:* ***вапсм*** *break Введите 0 для выхода:* ***0***

1. **Массивы.**

Массивы – это переменная. которая одновременно может содержать несколько значений. Массивы записываются в квадратных скобках.

*list = [ ]
print = (list)*
*list = [“a”, “b”, “c”, “d” ] [“a”, “b”, “c”, “d” ]
print = (list)*

Можно выбрать один элемент:

*print = (list[1]) b*

Почему именно ‘b’ а не ‘а’?
В программе принято начинать счет с нуля, а не с единицы. Чтобы найти последний элемент, запишем значение [-1];если указать [-2], то будет предпоследний элемент. И тд.

*print = (list[4])* *-
print = (“Последний элемент:” , list[-1]) d
print = (“Длина массива(Кол-во эл):” len(list)) 4
print = (“Последний элемент:”,list[len(list)-1]) d*

Теперь с помощью цикла введем все элементы массива

*i = 0 a
while i<len(list): b
 print(list[i]) c
i += 1 d*

**Изменяем значение:**

*list[i] = “m” # эл-т ‘b’ заменили ‘m’
print(list[i])*

Заменяем последний элемент ‘d’ на False. *a*

*list[len(list)-1])= False b
print(list[len(list)-1]) c
 d
 m
 False*

**11 Множества.**

Множества в языке Питон - это структура данных, эквивалентная множествам в математике. Множество может состоять из различных элементов, порядок элементов в множестве не определён. Не могут содержать одинаковые элементы. Находятся хаотичным образом.

Создаётся просто. Первый вариант с переменной:
1. *myset = set( )
 print(myset)*

Второй вариант с помощью фигурных скобок: (похожи на массивы)
2. *myset = { }
 print(myset)*

*myset = set(“Pythonn”)
print(myset) (‘y’,’n’,’P’,’t’,’o’,’h’)*(Порядок хаотичный, второй элемент пропускает)

*myset = {‘1’,2, 3,1,’1’)
print(myset) (‘1’,1,2,3)*
“1” и 1 – это разные вещи, строка и число. Поэтому они вывелись оба.

Функция *set( )*, может принять даже массив. Воспользуемся генератором:
*list = [s for s in range(0,10)]
print(list) [0,1,2,3,4,5,6,7,8,9]
myset = set(list)
print(myset) (0,1,2,3,4,5,6,7,8,9)*

**Генерация случайных значений.**

*impor random* – это отдельный модуль, который называется random. Он уже встроен в Питон. Выглядит так:
*print(random.random( )*

Эта функция генерирует случайное число, которое всегда будет строго от нуля до одного. Причем, один не входит. Но для того, чтобы числа были целые(не дробные) умножаем \*10, и преобразуем к типу **int**.

*print(int(random.random( )\*10)*Получается случайное число от 0 до 9.
Используем в генераторе:

*list[int(random.random( )\*10 for s in range(0,10)]
print(list) [8,0,9,5,6,3,3,0,2,0]
myset= set(list)
print(myset) {0,2,3,5,6,8,9}*

преобразуем массив во множество. Видно, что изменился порядок и удалились повторяющиеся элементы.

Теперь преобразуем обратно в массив, с помощью переменной:

*arr = [int(random.random( )\*10) for s in range(0,10)]
print(list) [2,0,9,2,7,0,9,0,1,2]
arr = list(set(arr))
print(arr) [0,1,2,7,9]*

Преобразовали в массив, удалили все элементы без всяких циклов, переборов и тд.

**12 Кортежи.**

Кортежи – это те же списки, которые невозможно отредактировать. Т.е элементы не сможем удалить.

Создать кортеж можно двумя способами:

1. Через функции tuple
2. Через круглые скобки.

*mytuple = tuple( )
print(mytuple)*

*mytuple = (1,”2”, “5”)
print(mytuple)*

*mytuple= (1,)
mytuple = (“s”,)
print(mytuple)*

Кортежи из одного элемента. Нужна запятая, иначе компилятор подумает, что есть лишняя деталь и просто удалит

Рассмотрим еще один пример.

*mytuple = tuple(“Python”)
print(mytuple) (‘P’,’y’,’t’,’h’,’o’,’n’)*

Порядок сохранился в отличии от множества.

Если у вас большой массив данных, который вам надо изучить, но не нужно там ничего менять, то логично использовать кортежи, а не списки. Это сэкономит память и ускорит работу, так как кортежи работают быстрее списков.

Еще одна важная функция кортежей: их можно использовать как ключи словаря. Ведь ключи словаря должны быть неизменяемыми.

**Задания тренажеры Python.**

**2. Переменные.**

1) Создайте переменную со значением 10.

2) Выведите данную переменную с помощью функции print().

3) Измените значение переменной на 15.

4) Выведите значение переменной в таком виде: «Значение переменной ИМЯ\_ПЕРЕМЕННОЙ равно ЗНАЧЕНИЕ\_ПЕРЕМЕННОЙ». То есть должно получиться примерно так: «Значение переменной abc равно 15».

5) Создайте 2 строковые переменные со значениями: «Значение переменной» и «равно».

6) Выведите аналогичную строку из пункта 4, но с использованием переменных из пункта 5.

7) Создайте 2 булевских переменных со значениями: True и False.

8) Выведите их так, как написано в 6 пункте (то есть с использованием строковых переменных).

 **4. Арифметические операции**

1) Создайте 2 числовые переменные со значениями на Ваш выбор.

2) Выведите через print() результат: суммы, разности, произведения, остатка от деления, получение целой части при делении, возведении в степень этих чисел.

3) С помощью print() посчитайте, чему равно значение следующего выражения: ((15 \* 10 – 20) / 2) + 14 \* 10 + (-45)

4) Создайте переменную, в которую запишите значение одного из чисел в двоичной системе счисления.

5) Выведите эту переменную.

**5. Логические операции**

1) Самостоятельно подумайте, чему будет равно следующее логическое выражение: True and (True or (False and True or False) and True or True != False)

2) Проверьте себя, выведя результат этого выражения с помощью функции print().

3) Самостоятельно подумайте, чему будет равно следующее логическое выражение: 15 > 20 or (5 < 7 and 8 > 12 or нч12 >= 12 and 15 < 18)

4) Проверьте себя, выведя результат данного выражения с помощью функции print().

**6. Строковые операции**

1) Попросите пользователя ввести 3 числа.

2) Выведите пользователю среднее арифметическое этих чисел.

**Примечание:** среднее арифметическое чисел равно сумме этих чисел поделённое на их количество.

**7. Условный оператор**

1) Попросите пользователя ввести 2 числа.

2) Сохраните в переменную результат деления первого числа на второе при условии, что делитель не равен 0. Если делитель равен 0, то сохранить в переменную строку: «бесконечность».

3) Вывести пользователю ответ в таком виде: «ЧИСЛО\_1 / ЧИСЛО\_2 = ОТВЕТ».

**Примечание:** разумеется, вместо «ЧИСЛО\_1», «ЧИСЛО\_2» и «ОТВЕТ» должны быть подставлены соответствующие переменные.

**8.Цикл while**

1) Напишите программу, которая будет принимать числа от пользователя и суммировать их, пока он не напишет слово «sum».

2) Когда пользователь напишет слово «sum», должна быть выведена сумма всех чисел и начат процесс заново.

3) Если пользователь напишет «exit» или «quit», программа должна быть завершена.

**9.Массивы**

1) Создайте список, состоящий из строк.

2) Выведите все элементы списка в таком виде: «ИНДЕКС\_ЭЛЕМЕНТА – ЭЛЕМЕНТ;»

3) Попросите пользователя ввести индекс того элемента, значение которого он хочет посмотреть.

4) Выведите значение элемента по индексу, полученному от пользователя.

**Примечание:** если пользователь ввёл индекс, которого нет, то написать ему об этом так: «Элемента с таким индексом не существует».

**11 Множества.**

1) Попросите пользователя ввести количество элементов для списка.

2) Создайте список, состоящий из целых случайных чисел от 0 до 100, заданного пользователем количества.

3) Выведите этот список с помощью цикла while.

4) С помощью множеств удалите из списка все повторяющиеся значения.

5) Выведите получившийся список с помощью цикла for.

**12 Кортежи**

1) Попросите пользователя ввести произвольную строку.

2) Создайте кортеж, состоящий из символов, введённой пользователем строки.

3) Выведите кортеж, используя цикл for.