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’>*

**Задание:**

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).

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.

**Задание:**

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.

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 andb2 = ” , b1 andb2 ) *b1 andb2 = 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.

**Задание:**

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().

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.

**Задание:**

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

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.

**Задание:**

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.

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***

**Задание:**

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

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*

**Многомерные массивы**

Это массив, который содержит другой массив.

*list[[2,3],4]*  - Это двумерный массив
*print(list[0][1]) -*Элемент с индексом 0 и 1. 3
*print(list[0],[1]) -*Чем отличаются?
*print(list[1]) 4*

Как перебрать массив?

Перебрать – это значит вывести всех из одного массива.

*list = [[2,3][4,5,6]]
s= 0
while s < len(list): 2
 j = 0 3
 while j len(list[s]): 4
 print(list[s][j]) 5
 j+= 1 6
 s += 1 7*

Такой документ можно представить как таблицу.

Сначала одну строку перебрали, потом другую. Вначале s=0 и j=0; далее j меньше, чем длина первого массива.

**Максимальное и минимальное значение**

Создаем массив, где есть цены на некоторые товары. Найдем *мин* и *макс* значение. Этот способ часто используется.

*print = [40,35,27,53,80]
min = prices[0]
max = prices[0]
s = 1
while s < len(prices):
 if prices [s] <min:* Если 35 меньше 40, присваиваем мин знач 35.
 *min = prices[s]* перебираем след,35 и 27, мин знач 27! 27 и 53🡪27.итд
 *if prices [s] > max:* Аналогично и макс значения.
 *max = prices[s]* *s += 1*

print(“Массив:”, prices) Массив: *[40,35,27,53,80]*
print(“Максимальное значение в массиве:”, max) Макс. значение 80
print(“Минимальное значение в массиве”, min) Миним. значение 27

**Задание:**

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».

1. **Цикл *for* и генераторы списков.**

Запомним перевод часто используемых слов:
**if**- если, **else**- иначе, **while** – пока, **for** - для, **for each** –для каждого

Перевод помогает быстрее запомнить и понять программу. Давайте создадим не сложный массив.
*list = [1, 5, 0, -5, 2.5]*

Можно продолжить через функцию **while**, но есть способ проще, через цикл **for**. Указываем переменную, в которую будет перемещено значение на каждой операции цикла, из массива. И через ключевое значение **in** указывается сам массив **(list)**.

Теперь в переменной n будет подставляться соответствующее значение в n=1,5.

*for n in list:
 print(n)*

str = “Python”
Создадим строку str Пайтон, она может работать также как и массив. Если мы захотим вывести первый символ Р:
*print(str[0])* *P*Также как и для массива мы можем использовать функцию for для переменной
*for s in str: P
 print(s) y
 t
 h
 o
 n*

**Генераторы списков.**

*array = list (range(2,15)* Генератор просто создает массив, в котором содержатся
*print(array)* все элементы от 2 до 15, в свободном порядке.
*for n in array:* *print(n)*

**Функция range** – возвращает нам и записывает переменную *list* не как список, это другой объект который называется *другой тип данных*.

Мы не можем с ним работать как с массивом.

Как быстро переименовать переменную везде и сразу?

Правой кнопкой мыши, щелкаем по переменной. Выбираем **🡪 Refactor 🡪Rename 🡪** пишем новое имя.

Посмотрим как еще можно сгенерировать массив с помощью *range* и цикла *for*

*array = [x for x in range (1,10)]
print(array)* Получается массив: *[1,2,3,4,5,6,7,8,9]*

Как читается: в переменную array помещаем значение х на каждой операции array, от 1 до 10. (не вкл 10)

Если мы в х запишем х\*2, то получим:

*array = [x\*2 for x in range (1,10)]
print(array) [2,4,6,8,10,12,14,16,18]*

Чтобы выбрать только четные числа:

*array= [x for x in range(1,10) if x %2 ==0]
print(array) [2,4,6,8]*

Встроенная функция **range()** используется для построения числовых последовательностей. В примере используем команду[*for*](https://pythonclass.ru/python/cikl-for-i-funkciya-range-v-python/#cikl-for) и функцию *range* для выполнения ровно 10 операций с выводом значения от 0 до 10.

*for x in range(10):
print(x, end=' ') 0 1 2 3 4 5 6 7 8 9*

При передаче только одного аргумента, последовательность будет начинаться с 0 и до значения аргумента (10), не включая его. *Одна из самых распространённых ошибок, это когда разработчик предполагает, что значение аргумента функции range включается в сгенерированную последовательность.*

Приведем пример с двумя аргументами. Выведем последовательность от 20 до 30.

*for x in range(20, 30):
print(x, end=' ')*  *20 21 22 23 24 25 26 27 28 29*

**Создание числового списка с помощью функции range.**

С помощью функции **range** удобно строить числовые списки.

*numbers = list(range(5, 25))
print(numbers)
 [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]*

Функция **range** может создавать последовательности, пропуская числа в заданном диапазоне. Приведем пример построения списка от 0 до 100, с шагом 15.

*numbers = list(range(0, 100, 15))
print(numbers) [0, 15, 30, 45, 60, 75, 90]*

С помощью цикла*for*и функции *range* можно создавать практически любой числовой список. К примеру, создадим список квадратов всех числе от 1 до 10. Операция [возведения в степень](https://www.pythonclass.ru/python/chisla-i-arifmeticheskie-operatory/#vozvedenie-v-stepen) обозначается двумя звездочками (\*\*).

*a = []
for i in range(1, 11):
a.append(i\*\*2)

print(a) [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]*

На каждом цикле переменная возводится в квадрат и добавляется в список.

**Задание:**

1) Создайте список из 5 чисел.

2) Определите сумму чисел в списке и выведите её.

3) Определите среднее арифметическое чисел в списке и выведите его.

**Примечание:** при выполнении заданий используйте цикл for.

**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.

**Задание:**

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.**

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.

**Задание:**

1) Попросите пользователя ввести произвольную строку.

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

3) Выведите кортеж, используя цикл for.

**13. Словари.**

Словари (dict) – это одна из наиболее часто используемых структур данных, позволяющая хранить объекты, для доступа к которым используется ключ.

Похож на массив, но в отличии от него имеет строго числовой вид, а в словарях в качестве индекса может выступать строка, число, и др. тип данных. Чтобы создать словарь используйте функцию dict( ), или с помощью фигурных скобок.

*mydict = dict( )
print(mydict)*

В фигурных скобках указываем набор пар: **ключ : значение** (Указываются через двоеточие).
mydict = {“name”: ‘Jon’, ‘Age’: 35}
*print(mydict) (“name’:’Jon’, ‘Age’:35)*
**name** – это ключ; **Jon** - значение

Также можно создать словарь с помощью функции  *dict( ).(Пробелы не нужны)*

*mydict = dict(Name = “Jon”, Age = 35, isMale =True)
print(mydict) (“Name’:’Jon’,‘Age’:35,’isMale’:True)*

В словаре можно выбрать значение по ключу:

*print(mydict[“Age”]) 35*

Теперь научимся перебирать словарь и находить ключ и значения.
*for key in mydict: Name = Jon
 print(key, ‘=’, mydict[key]) Age = 35
 isMale = True*

Получим тоже самое, только с помощью кортежа. Это почти как обычный массив, но в кортеже нельзя менять элементы.

mytuple = (‘Name’, ‘Age’,’isMale’) *Name = Jon*
*for key in mytuple: Age = 35
 print(key, ‘=’, mydict[key]) isMale = True*

***Используем генератор****.*

Создадим словарь с помощью генератора. В первой части будет ключ, а во второй значение.

*mydict= {str(s\*2): n for n in range(1,10)}
print(mydict) (‘2’:1,’4’:2,’6’:3,’8’:4,’10’:5,’12’:6,’14’:7,’16’:8,’18’:9)*

**Задание:**

1) Создайте словарь с двумя ключами «Name» и «Age» и значениями
«Без имени» и «-1».

2) Попросите пользователя ввести своё имя.

3) Попросите пользователя ввести свой возраст.

4) Примите эти данные и измените соответствующие элементы словаря.

5) Выведите этот словарь (ключи и значения), используя цикл for.

**14 Функции**

Это определенная конструкция, которая имеет своё название и может принимать различные параметры, выполняющие набор инструкций.

Чтобы создать функцию пишем **def …( ):** с помощью отступов мы определяем какие инструкции нужно выполнять.

*def print python( ):
 print(“Python”)*

*print python( ) Python*

Создадим функцию, которая суммирует элементы.

***def*** *sum(x, y): -****Сложение*** *return x+y это Функция*

*print(“Python”) это Значение
s = sum(5,12)
print(s)*

***def*** *sub(x-y): -****Вычитание*** *reture x-y*

*s = sub(10,15)*

Функции могут содержать в себе не обязательные параметры.

***def*** *summaprint(x, y, r = False):
 s = sum(x +y)
 if r:
 return s
 else:
 print (s)*

*summprint(15, 7)
print(summaprint(15, 7, True)) 22*

Сумма **х+у** в зависимости от параметра **r** будет возвращать(суммировать) или печатать в консоль. Если переменная **r** является истиной, то мы возвращаем ***s****,* иначе просто печатаем результат.

*print(sub(10, 5)) 5
print(sub(y=10, x=5)) – здесь указаны прямые значения т.е (5-10) -5*

Есть возможность создавать большое количество параметров, для этого используется \*. Она представляет собой массив, содержащий все параметры. Найдем сумму всех чисел, которые передал пользователь.

***def*** *bigsum(\*numbers):
 s = 0
 for n in numbers:
 s += n
 return s*

*print(bigsum(1, 5, 7, 0, 1)) 14*

Есть возможность указать \*\*. Когда мы передаем в словарь в виде таких значений (х=10, у=5)

***def*** *printdict(\*\*dict):
 for key in dict:
 print(key, “=”, dict[key]*

*printdict(name= “Jon”, age=25) name= Jon
 age = 25*

**Анонимные функции**

Это функции, которые создаются с помощью *lambda* выражений. Принимает два параметра и печатает их сумму.

*print(“Анонимные функции”)
lambda func = x, y: print(x +y)
lambda func(5, 7) 12*

*result = (lambda x, y: x + y)(1, 3)
print(result) 4*

*def getmax(arr):
 max = arr(0)
 for n in arr:
 if n > max:
 max=n
 return max*

*print(“-------“)
print(getmax([5,7,0,12,1])) 12
print(getmax([-5,-7,1,10,50,99])) 99*

Эта функция одной строчкой выполняет 5 те же строк.

**Задание:**

1) Создайте функцию, которая проверяет чётное число передано в параметре или нет. Она должна возвращать True или False.

2) Создайте функцию, которая принимает список и возвращает максимальное значение из списка.

3) Создайте функцию с переменным числом аргументов, внутри которой должно выводиться среднее арифметическое переданных чисел.

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

**15. Глобальные и локальные перемены.**

Глобальные – это переменные, к которым имеется доступ всегда и везде.
Локальные – это перемеренные, к которым есть доступ на определенном участке кода.

Преобразуем параметры типа float. Далее вернем через команду *return.*

*def sum(x, y):* ***(1)*** *s = float(x) +float(y)
 return s*

*x = input(“Введите первое число: ”)* ***(2)*** *y = input(“Введите второе число: ”)* ***(2)*** *print(“Сумма равна:”, sum( x , y))*

*Теперь пользователь вводит два числа, и выводим сумму:*

*Введите первое число:* ***7*** *Введите второе число:* ***12*** *Сумма равна:* ***19***

**Вопрос:** Где тут локальные перемены, а где глобальные?

х и у (2) – строго глобальные, к ним есть доступ, откуда угодно. А переменные *sum(x, y) (1)* являются локальные. К ним нет доступа. Хоть глобальные *(х и у)* и локальные *sum(x, y)* называются одинаково, но между ними нет не чего общего.
Тоже самое переменная **s** (она находится внутри функции). Доступ к ней есть только внутри этой функции. За пределами возникнет ошибка.

Создадим переменную **isprint = False**, если переменная является истиной (True), то **s** напечается через print, если ложь (False) мы ее вернём.

*isprint =* ***False*** *def sum(x , y):
 s = float(x) + float(y)
 if isprint:
 print (s)
 else:
 return s*

Пока в результате мы видим тоже самое, но если мы поменяем значение на True

Добавим ещё одну функцию- вычитание:

 ***def*** *sub(x , y ):
 global result Введите первое число:* ***4*** *result = float(x) - float(y) Введите второе число:* ***3***

*sub(x, y) Сумма равна:* ***7*** *print(“Разность равна: ”, result) Разность равна:* ***1***

**Задание:**

1) Создайте переменную со значением числа пи: «3.141592».

2) Напишите функцию, которая будет возвращать площадь окружности по переданному в параметре радиусу.

3) Проверьте работу функции.

**Примечание:** площадь окружности = пи \* радиус \* радиус. Значение числа пи надо взять из глобальной переменной, созданной в первом пункте.

**16 Модули**

Система модулей позволяет логически организовывать код на Питоне. Группирование кода в модули значительно облегчает процесс написание программы.

**Модуль** – это просто файл содержащий код, который может содержать переменные, класс, функции. Подключить модуль можно через ключевое слово *import* (Импортируем из библиотеки).

*import random* (Теперь чтобы его использовать нужно ниже обратиться к этому модулю через точку и вызвать функцию)

*print(random.randint(0,10)) 6*
Вышло случайное число.

Добавим другую библиотеку:
*import math (<)
print(math.sin(1)) 0,084147098..*

Чтобы не писать постоянно *math,* мы дополним библиотеку: *import math as m*Далее записываем уже: (т.е это псевдоним *m* для модуля *math*) *print(m sin(1))*

Также для удобства часто используется решетка # на *import random* u *print(radom.randint(0,10))*

*# import random
# import math as m*

*from random import*\*

# *print(radom.randint(0,10))
# print(m sin(1))*

*print(randint(0,10)) - означает, что мы импортируем из библиотеки все функции.*

*from math import sin.cos
print(sin(1)) 0,8414709...
print(cos(1)) 0,5403020...*

Импортируем отдельные функции:

import math, cmath as cm

print(math.ceil(10.3)) 4
print(cm.log10(1000) (2.9999)

Создадим свой модуль:

Base => New => Python File: calс(калькулятор)
В новом файле создадим минимальный набор функций.

*def sum(a, b):
 return a + b*

*def sub(a, b):
 return a – b*

*def mult(a, b):
 return a \* b*

*def div(a, b):
 return a / b*

Подключаем этот модуль к библиотеке calс. И напишем не сложную программу, которая будет работать в бесконечном цикле, пока пользователь не попросит выйти.

*import calc

while True:
 print(“1 - Сложение; 2 – вычитание; 3 - Умножение; 4 - Деление; 0 – Выход”)
 code = input(“Введите команду:”)
 if code = = “0”:
 exit(0)
 a = float(input(“Введите первое число:”))
 b = float(input(“Введите второе число:”))
 if code = = “1”:
 r = calc.sum(a, b)
 elif code = = “2”:
 r = calc.sub(a, b)
 elif code = = “3”:
 r = calc.mult(a, b)
 elif code = “4”:
 r = calc.div(a, b)
 print(“ll:”, r)*

*Вводим команду 3
1 число: 5
2 число -5
Результат = -25 /(5\*(-5)=-25)*

***Для общего понятия прилагаю скриншот темы и калькулятора.***





Задание:

1) Создайте свой модуль и подключите его в основном файле.

2) Напишите в модули 3 функции, каждая из которых принимает список. Первая функция – получение максимального значения, вторая – получение минимального значения, третья – получение суммы всех элементов.

3) Проверьте работу этих функций в основном файле.

**17. Исключения**

В языке Python исключений огромное количество, выбрасываются они постоянно. Важно уметь их обрабатывать.

Исключение 1: Давайте приведем к числу строку, которая числом не является.

*а = float(“abc”)
print(a)*
Когда запустим, сразу видим, что у нас вышла ошибка (исключение), которое называется **Value Error.**

Теперь, когда мы знаем название, мы можем без проблем его перехватить.

*try:
 a = float(“abc”)
except Value Error:
 print(“Невозможно привести к числу”)*

**try** – переводится как “попытаться”. Если выходит исключение **Value Error,** то выполняется след инструкция:

Пример: Сделаем бесконечный цикл. Пользователь введет положительное число. Далее преобразуем его из строки в числовой тип данных.

*while True:
 a=input(“Введите положительное число:”)
 try:
 a = float(a)
 except Value Error:
 print(“Невозможно привести к числу!”)* else:
 print(“Спасибо за корректный ввод!”)

Вывод:

*Введите положительное число:* ***10*** *Введите положительное число:* ***vjvnjd*** *Невозможно привести к числу!*

У блока Python есть такая возможность указать блок **else:** – вызывается в том случае, если у нас не было исключения.

Существует еще один блок у “try” finally: он будет выставляться в том случае: было ли исключение или нет.

 *finally:
 print(“В любом случае завершаем программу!”)
 exit(0)*

В нижнем окне вводим:

*Введите положительное число:* ***fbfkbm*** *Невозможно привести к числу!
В любом случае завершаем программу!*

**Сгенерируем собственное исключение**

Проверяем:

*if a <= 0:
 raise Exception (“Число не положительное.”)
except Exception as exp:
 print(exp)*
Просим пользователя ввести отрицательное число.

Вывод получится:

*Число не положительное.
В любом случае завершаем программу!*

**Задание:**

1) Узнайте, какое исключение появляется при делении числа на 0.

2) Попросите пользователя ввести 2 числа.

3) Выведите результат деления.

4) Перехватите исключение при делении на 0 и выведите пользователю в качестве результата слово «бесконечность».

**18 Консольные программы**

Интерактивная консоль – это быстрый способ выполнить команды и протестировать код, не создавая файл.
Представляет доступ к истории команд, всем встроенным функциям и модулям. Для начала работы требуется две библиотеки.

*import subrocess
import io*

Создадим переменную “sp” и начнем открывать поток для сабпроцессов.

В качестве массива [ ] укажем команду “dir”. [“dir”] – это содержимое директории.

 *sp = subprocess.Popen([“dir”], stdout =subprocess.PIPE, shell = True)
print(sp)
out = io. Text IOWrapper(sp.stdout, encoding = “****cp866****”)
s = ‘ ‘;
while s:
 s = out. readline( )
 print(s)*

**Задание:**

1) Откройте командную строку в своей системе.

2) Узнайте, с помощью какой команды можно получить текущую дату, используя команду help, либо документацию к своей ОС.

3) Напишите программу, которая выполняет эту команду и выводит результат с помощью функции print().

**19. Отладка программы.**

Отладчик Python позволяет изменять поток программы во время выполнения с помощью встроенных команд. Это позволяет пропустить фрагменты программы, предотвратить запуск какого-либо кода или вернуться назад, чтобы снова запустить код.

*a = 0
while True:
 print(a)
 a + 1 = 0.1
 if (a >= 1):
 exit(0)
 print(“Hello”)*

Как мы видим, обычная программа, и ошибки быть не должно, но в итоге программа работает не 10 циклов, а бесконечно.

Попробуем исправить. Через *print(a)* или через встроенные возможности питона.

Итак 1:

*print(a)-* в принципе хорошо работает, но цикл начинается с крупного числа ~9074, фактически он стер всё что было до, вряд ли получится остановить цикл, когда значение а=0,1.

Для этого есть другой способ, который называется Debug или отладка.
Как это работает? Ставим *breakpoint* (красная точка). Ставим ее на той строчке, которой нужно прерывание программы. В нашем случае №2. Далее **Rum => Debug => Активное окно**.

Автоматически программа прервалась, и открылось окно => **Debugger**. Где видно значение переменной “a”. (a = {int}0) .

Для того чтобы посмотреть, что будет на следующем шаге нажимаем “step over” .

**Step Over  - кнопка с изогнутой стрелкой и точкой  выполняет текущую строку и останавливается на следующей строке. Т.е. вы как бы медленно проходите по строчкам кода.**

После того как **a>1**, программа завершается. в консоли слово “hello” написано 10 раз. как было нужно. Благодаря этой кнопке (красной точке), мы определили место ошибки.

Ставим ее там, где хотим чтобы программа прервалась.

**Rum => Debug**. Смотрим и анализируем.

**Задание:**

1) Создайте список из 5 чисел.

2) Напишите функцию, которая находит все отрицательные числа и выводит их.

3) С помощью отладки пройдите все шаги выполнения программы, анализируя значения всех переменных.