1. Знакомство с языком.

Python — самый быстрорастущий язык программирования. По данным рейтинга IEEE Spectrum он является самым широко используемым на планете. Python используется в таких компаниях, как Google, Mail, Яндекс, YouTube и других. В большинстве университетов мира студенты начинают программировать именно на нём. Также на этом языке программирования часто пишут игровую логику.

Базовые понятия программирования:

Алгоритм — это набор последовательных шагов для достижения какой-либо цели.

Программа — это алгоритм, записанный на языке программирования.

Язык программирования — формальный язык, предназначенный для записи компьютерных программ.

В языках программирования существует множество команд, каждая из которых выполняет определенные действия. Такие команды называются функциями.

Функция — алгоритм, названный каким-то именем.

Мы не видим команды, включенные в алгоритм так называемых стандартных функций, так как у нас нет доступа к их просмотру, а используем только имя, которым назван алгоритм. Когда мы пишем в своем коде имя функции, команды, включенные в ее алгоритм, выполняются автоматически. Функция может вести себя по-разному. Ей могут быть нужны какие-то данные, чтобы ее работа стала однозначной. Такие данные называют аргументами функции. Тогда команда запуска (говорят ещё «вызова») функции будет выглядеть так:

имя\_функции(аргумент1, аргумент2, …)

Аргументы пишутся через запятую! Скобки нужны, чтобы в них перечислить все нужные аргументы. Кроме того, скобки после имени дают понять языку программирования, что мы имеем дело с функцией, поэтому в тех случаях, когда никакие аргументы не нужны, выполнить её можно так:

имя\_функции()

В Python для вывода текста используется функция print.

Пример:

print ("Hello, world!")

Переменная ― это ссылка на выделенную область памяти компьютера, в которую мы можем положить нужную нам информацию на время, а потом достать, когда она нам понадобится.

Для того чтобы положить в выделенную область памяти какие-то данные, используется оператор присваивания, который к тому же создаёт переменную.

Пример:

text="Привет!"

print(text)

print(text)

print(text)

Функция input() ― получение данных от пользователя.

Пример:

answer = input(“What is your name?”)

1. Типы данных.

Для того, чтобы можно было понять, с какими данными программа работает в данный момент и какие действия можно с ними совершать, в программировании ввели такое понятие как “тип данных”.

Верно, мы уже работали с типом данных “строка”. В программировании он обозначается как “string”. Строка — это набор символов. Помимо него, есть и другие типы данных, например, целые числа — int, и дробные числа (числа с плавающей точкой) — float.

В значение переменных мы поместили результат работы функции input(). Она считывает те символы, которые написал человек и всегда возвращает строку (string)!

Для изменения типа мы должны дать следующее указание программе: считать символы, введённые пользователем при помощи функции input(), и привести их к новому типу данных — “целое число”.

Приведение, или, по другому, преобразование типов — это автоматическое преобразование значения одного типа данных в значение другого типа в процессе выполнения операций с данными.

Оно выполняется, если данные, участвующие в операции, имеют разные типы. Обратите внимание, что преобразование может осуществляться лишь тогда, когда данные удовлетворяют тому типу, в который преобразуются. Например, если вы попытаетесь привести строку 1.51.51 к типу данных float, программа выдаст ошибку.

«Приведение к типу данных “целое число” осуществляется с помощью функции int(). Её аргументом выступает строка, а результатом выполнения — число»

Пример:

number\_one = input("Введите число №1 ")

number\_two = input("Введите число №2 ")

number\_one = int(number\_one)

number\_two = int(number\_two)

answer = number\_one + number\_two

print(answer)

«При помощи функции input() мы запрашиваем у пользователя число и она возвращает нам строку, поэтому в аргумент функции int() можно сразу записать всю конструкцию input().

Пример:

number\_one = int(input("Введите число №1 "))

number\_two = int(input("Введите число №2 "))

answer = number\_one + number\_two

print(answer)

Функция str() приводит тип данных в строку.

Пример:

number\_one = int(input("Введите число №1 "))

number\_two = int(input("Введите число №2 "))

answer = number\_one + number\_two

print("Сумма = " + str(answer))

«С типом данных “строка” можно производить операции: конкатенация, или склеивание — “+”, дублирование (мультипликация) — “\*”.

С типом данных “число” можно производить любые арифметические операции: сложение — “+”, вычитание — “-”, умножение — “\*”, деление — “/”, возведение в степень — “\*\*”.

Еще один тип дынных – float. Для того, чтобы привести данные к типу “дробное число” используется функция **float()**».

С типом данных «дробное число» можно производить такие же операции, как и с целыми числами. Разница заключается лишь в том, что программа всегда вернёт в ответ дробное число, то есть целую и дробную часть, разделённые точкой.

1. Условный оператор.

Иногда нам нужно, чтобы компьютер мог использовать написанное нами условие и выяснить, верно оно или нет. Значит, условие должно быть тем, что в логике называют «утверждением».

Утверждение — это выражение, которое может принимать одно из двух значений: истина или ложь.

Утверждение создаться с помощью операторов сравнения.

«В неравенствах сравниваются два выражения, которые располагаются слева и справа от операторов сравнения. Неравенства бывают двух видов — строгие и нестрогие. В строгих неравенствах используются операторы: меньше (<), больше (>), равенство (==), неравенство (!=). В нестрогих: меньше или равно (<=), больше или равно (>=). Разница между двумя видами неравенств заключается в результате проверки утверждения, возвращаемом операторами.

В строгих неравенствах утверждение верно тогда и только тогда, когда однозначно верно условие. Например: (<) меньше — условие верно, если левое выражение меньше правого, (>) больше — наоборот. Или (==) равенство — условие верно, если правое выражение равно левому, (!=) неравенство — наоборот.

В нестрогих же неравенствах утверждение верно тогда, когда верно одно из условий. Например: (<=) меньше или равно — условие верно, правое выражение меньше левого или равно ему, (>=) больше или равно — наоборот.

Операторы сравнения возвращают значение специального логического типа — bool. У этого типа имеется всего два принимаемых значения — True (истина) и False (ложь), получаемых при проверке истинности условия. Значит, утверждение — это выражение, значение которого имеет тип bool.

Для того, чтобы использовать утверждение в качестве условия, нам необходимо поместить его в команду, которая называется «условный оператор».

Эта команда:

* вычислит значение утверждения;
* поймёт, выполнено ли условие;
* в соответствии с этим выберет, какие команды в программе нужно выполнять далее.

В Python это записывается следующим образом:

if <условие>:

<команда 1>

<команда 2>

………………………….

<команда n>

Иногда бывает, что нужно записать ещё один набор команд в программу, который выполнится в том случае, если условие ложно.

В Python это записывается следующим образом: после всех команд, относящихся к if, записывается слово else и сам набор команд.

if <условие>:

<команда 1>

<команда 2>

………………………….

<команда n>

else:

<команда 1>

<команда 2>

………………………….

<команда n>

Обратите внимание, что команды, относящиеся к if и else пишутся на другом уровне вложенности. Другой уровень вложенности — это определённое количество отступов, чтобы программа могла отличить, где в программе команды, а где — условия, а также сами блоки команд между собой. В Python каждый следующий уровень вложенности должен писаться правее предыдущего. Разница между уровнями может быть один пробел или больше. Хорошим стилем считается всегда придерживаться разницы в четыре пробела.

Кроме того, чтобы программа смогла понять, где начинается уровень вложенности, после условия, относящегося к if, и после else ставится двоеточие.

1. Цикл while.

Чтобы запрограммировать повторяющиеся действия, в программировании используют специальную команду — цикл.

Цикл — это многократное повторение определённых действий. В нашей жизни есть множество примеров: смена дня и ночи, когда солнце сначала встаёт, затем садится и так по кругу. Сбор в школу или на работу с утра, когда мы каждый день выполняем одинаковые действия раз за разом: проснуться, встать с кровати, почистить зубы, умыться, позавтракать, одеться и т. д. Всё это — циклы.

В Python для подобных целей используется цикл while, или по другому — цикл “пока”. Пока какое-либо условие истинно, выполняем действия, иначе, когда условие становится ложным — прекращаем.

Слово while является ключевым. Оно показывает, что далее следует цикл. После while пишется условие повторения, истинность которого и отвечает за его выполнение. После него обязательно ставится двоеточие. Оно показывает, в каком месте начинается блок исполняемых в цикле команд. Блок команд пишется на другом уровне вложенности».

Синтаксис:

while <условие повторения>:

Команда 1

Команда 2

………………….

Команда n

В циклы мы так же можем складывать условия,

Пример:

while <условие повторения>:

if <условие>:

Команда 2

Команда 3

elif <условие>:

Команда 4

Команда 5

else:

Команда 4

1. Списки.

Всё, что можно запомнить с помощью переменной — это один элемент данных. А список — это последовательность элементов данных. Нам могут быть полезны списки строк (например, покупки) или списки чисел (например, список номеров). Как вы увидите, в одном списке могут быть одновременно и числа, и строки.

checklist = []

Создаём переменную checklist. Чтобы объяснить программе, что мы хотим присвоить переменной в качестве значения список, используются квадратные скобки. Обратите внимание, сейчас внутри них ничего не записано. Это означает, что мы присвоили переменной cheklist пустой список.

И это очень полезная вещь. Довольно часто приходится сначала создавать пустые списки, которые будут заполнены позднее, а не сразу.

checklist = ["one", "two", "three"]

Теперь в списке записана последовательность из элементов, представленных строками. Обратите внимание, что каждая строка записана в кавычках и все элементы перечисляются через запятую.

Точно так же можно заполнить список последовательностью чисел.

Вывод элементов происходит через индексы по отдельности осуществляется: имя переменной, которой присвоено значение списка [индекс элемента]

Обратите внимание на то, что целиком список выводится на экран так же, как он записан внутри программы — в квадратных скобках, элементы разделены запятой и, если это строки, то они выводятся в кавычках.

Метод append(), который позволяет дописать элементы в конец списка.

Существует ещё много различных методов работы со списками. С ними вам предстоит познакомиться самостоятельно.

1. Цикл for.

Цикл for — это цикл с параметром, for, в котором мы можем указать, сколько раз должны повториться команды в его теле.

Цикл for — синтаксис

for i in a:

print(i)

Записывается он следующим образом: сначала пишется ключевое слово — for, затем, прямо в цикле, а не перед ним, как в while, сразу после for указывается имя переменной, которую мы будем использовать для итерации. Затем пишется ключевое слово in. Седом за этим ключевым словом написана переменная, которой присвоено значения какой-то последовательности.

Отличия цикла for от while

1. Не нужно создавать переменную и обнулять её значение до цикла.
2. Не нужно писать условие повторения перед телом цикла.
3. Не нужно следить за итерацией переменной внутри тела цикла.

Циклы for часто используют для повторения каких-либо действий заданное количество раз, либо для изменения переменной от некоторого начального значения до выбранного конечного. В этом случае for используется совместно с функцией range()

Функция range() — генерирует диапазон целых чисел в заданном интервале.

Функция range()

диапазон от a до b

range(a, b)

Первый параметр функции range() (a) задаёт начальное значение переменной, а второй параметр (b) задаёт значение, которое переменная принимать не будет. Поэтому переменная i в цикле for примет значения от a до b - 1.

Пример:

number = int(input("Введите число: "))

final = 0

for i in range(1, number + 1):

final = final + i

print(final)

1. Словари и множества.

Словарь — это неупорядоченный набор произвольных элементов, каждый из которых доступен по уникальному ключу. Другими словами, в нем мы можем создавать объекты и присваивать им желаемые номера-идентификаторы, по которым в дальнейшем сможем получить к ним доступ. Такие номера называются ключами для данных

Чтобы создать словарь, используются фигурные скобки — {}. Внутри них элементы записываются через запятую в следующем формате: ключ для идентификации: элемент (значение).

Пример:

dictionary = {"00451": "F15", "20000": "A2", "00010": "T19", "01840": "A2", "00012": "A3"}

print(dictionary["00451"])

Чтобы записать новое значение в словарь, необходимо применить следующую конструкцию: имя переменной, которой присвоен словарь [ключ] = добавляемое значение. Теперь, если мы выведем получившийся словарь, то увидим, что добавился новый элемент. Обратите внимание, что он добавился не в начало и не в конец словаря. Стандарт языка на самом деле не гарантирует место, куда попадёт новый элемент.

Для поиска вхождения элемента в словарь есть специальный оператор in. Используется он следующим образом

dictionary = {"00451": "F15", "20000": "A2", "00010": "T19", "01840": "A2", "00012": "A3", "01836": "G3"}

if "20000" in dictionary:

print(dictionary["20000"])

В Python структура множеств эквивалента множествам в математике. Их элементами могут стать любые неизменяемые объекты, например строки или числа и, в отличии от списков, в множествах, так же, как и в словарях, порядок хранения элементов не важен.

Элементы множества в программе записываются, как и в словарях, внутри фигурных скобок

Все элементы множества уникальны и входят в него единожды.

ля того, чтобы добавить элементы в множество, используется метод add, записываемый в следующем формате: имя переменной, которой присвоено в качестве значения множество.add(добавляемый элемент

Предположим, что номер нашей заявки 14. И нам необходимо проверить, входит ли он во множество. Для этого есть специальный метод in. Применяется он следующим образом: пишется искомый элемент, затем ключевое слово in, а после него переменная, которой присвоено множество. Если элемент содержится в множестве — программа выведет логическое True (истина), иначе — False (ложь).

Пример:

many = set()

many.add(11)

many.add(6)

many.add(8)

many.add(12)

print(14 in many)

1. Функции.

Функция — это алгоритм, названный каким-либо именем. Поэтому, когда мы пишем, например, ключевое слово print(), в действительности вызывается целый алгоритм, который выводит указанный в параметрах текст. Мы не видим этот алгоритм, так как он вшит в язык Python, но видим результат его работы.

Для того, чтобы создать собственную функцию, используется данная на схеме конструкция. Сначала пишется ключевое слово def, которое даёт программе понять, что с этого места в коде начинается создание функции. Затем пишется имя, с помощью которого мы потом сможем её использовать. Как мы уже знаем, у функции могут быть параметры, необходимые для её работы, поэтому, если в создаваемой нами функции они тоже будут нужны, мы сразу должны указать их в скобках после имени. Далее, на втором уровне, пишутся необходимые команды.

Создание функции:

def имя\_функции (параметр1, параметр2, …):

Команда 1

Команда 2

……

Команда n

Что-бы применить функцию достаточно написать в программе имена функций и указать параметры, необходимые для их работы.

Программа выполняет один алгоритм, потом встречает вызов функции и начинает выполнять другой (в функции). Когда функция заканчивает работу, исполнение возвращается в тот алгоритм, тот участок кода, откуда функция была вызвана. При этом то значение, которое функция вернула, подставляется в этот участок кода

Оно может быть использовано дальше в программе. Например, рассмотрим функцию input().Она выводит на экран текст, записанный в скобках, а затем ждёт ответа от пользователя. Этот ответ как раз и будет возвращаемым значением. Другими словами, функция input() вернёт этот ответ в программу переменной, в качестве значения которой она указана

Для того, чтобы функция, которую мы создали, могла возвращать значение в тот участок кода, откуда она была вызвана, используется специальная команда — return.