1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Прежде всего, хочется отметить, что здесь представлены теоретические основы алгоритмизации и программирования. Чтение материала начинающему позволит лучше изучить предмет. Тем не менее, стоит упомянуть, что в тексте будут приводиться примеры различных функций (термин «функция» уже может быть непонятен читателю), результаты их работы, терминологические названия (списки, кортежи и пр.), однако написанное может не объясняться в деталях.

Таким образом, мы сразу должны оговорить, что многое станет более понятным и доступным, как только вы начнете работать со средой программирования Python, напишете свои первые программы. Тогда вы сможете более осознанно перечитать теоретический материал и сделать соответствующие выводы.

1.1 Алгоритм. Свойства алгоритма. Способы описания алгоритма.

Если мы хотим написать программу на каком-либо языке программирования, то сначала нам следует составить алгоритм решения задачи.

Алгоритм - это точное и простое описание последовательности действий для решения данной задачи. Алгоритм содержит несколько шагов, которые должны выполняться в определенной последовательности. Каждый шаг алгоритма может состоять из одной или нескольких простых операций.

Каждый из нас ежедневно использует различные алгоритмы: инструкции, правила, рецепты и т. д. Обычно мы это делаем не задумываясь. Например, открывая дверь ключом, никто не размышляет над тем, в какой последовательности выполнять действия. Однако чтобы кого-нибудь научить открывать дверь, придется четко указать и сами действия, и порядок их выполнения. Например:

1. Достать ключ.
2. Вставить ключ в замочную скважину.
3. Повернуть ключ два раза против часовой стрелки.
4. Вынуть ключ.

Представим, что мы поменяли местами второе и третье действия. Мы сможем выполнить и этот алгоритм, но дверь не откроется, т. е. алгоритм станет невыполнимым.

Для алгоритма важен не только набор действий, но и то, в каком порядке они выполняются. Понятие алгоритма в информатике является фундаментальным.

Таким же, какими являются понятия точки, прямой и плоскости в геометрии, вещества в химии, пространства и времени в физике и т. д.

Свойства алгоритма:

1. дискретность (прерывность, раздельность) - алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов (этапов);
2. определенность - каждый шаг алгоритма должен быть четким и однозначным. Выполнение алгоритма носит механический характер и не требует никаких дополнительных сведений о решаемой задаче;
3. результативность - алгоритм должен приводить к решению задачи за конечное число шагов;
4. массовость - алгоритм решения разрабатывается в общем виде, т. е. он должен быть применим для решения некоторого класса задач, различающихся лишь исходными данными.

Способы описания алгоритмов

1. словесный;
2. графический;
3. табличный;
4. формульный.

Словесный способ каждый из нас использует ежедневно, пересказывая собеседнику, например, различные инструкции, правила, кулинарные рецепты, т. е. какую-то последовательность, приводящую к конечному результату.

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. Часто для лучшего понимания той или иной ситуации нам проще начертить какую-то схему или план, согласно которым мы будем действовать. В программировании данный способ предпочтителен, поскольку позволяет с помощью последовательности функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий, представить ход решения той или иной задачи. Такое представление алгоритма называется структурной схемой алгоритма, или блок-схемой.

Табличный способ используется, например, в бухгалтерии при составлении ежегодных отчетов, сводок и т. д.

Формульный способ находит свое применение при решении задач из области математики, физики и т. д. Например, при решении квадратного уравнения мы приступаем к нахождению дискриминанта уравнения, а затем, в зависимости от полученного результата, находим корни уравнения по известным всем формулам.

1.2 Назначение функциональных блоков

На условные обозначения в схемах алгоритмов, программ, данных и систем распространяется ГОСТ 19.701—90. «ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения». При отрисовке блок-схем мы будем использовать функциональные блоки, представленные ниже. Заметим, что некоторые их названия или отрисовка не всегда совпадают с указанными в ГОСТ, поэтому при создании схем

алгоритмов для курсовых, дипломных работ, безусловно, следует обратиться к существующему стандарту.

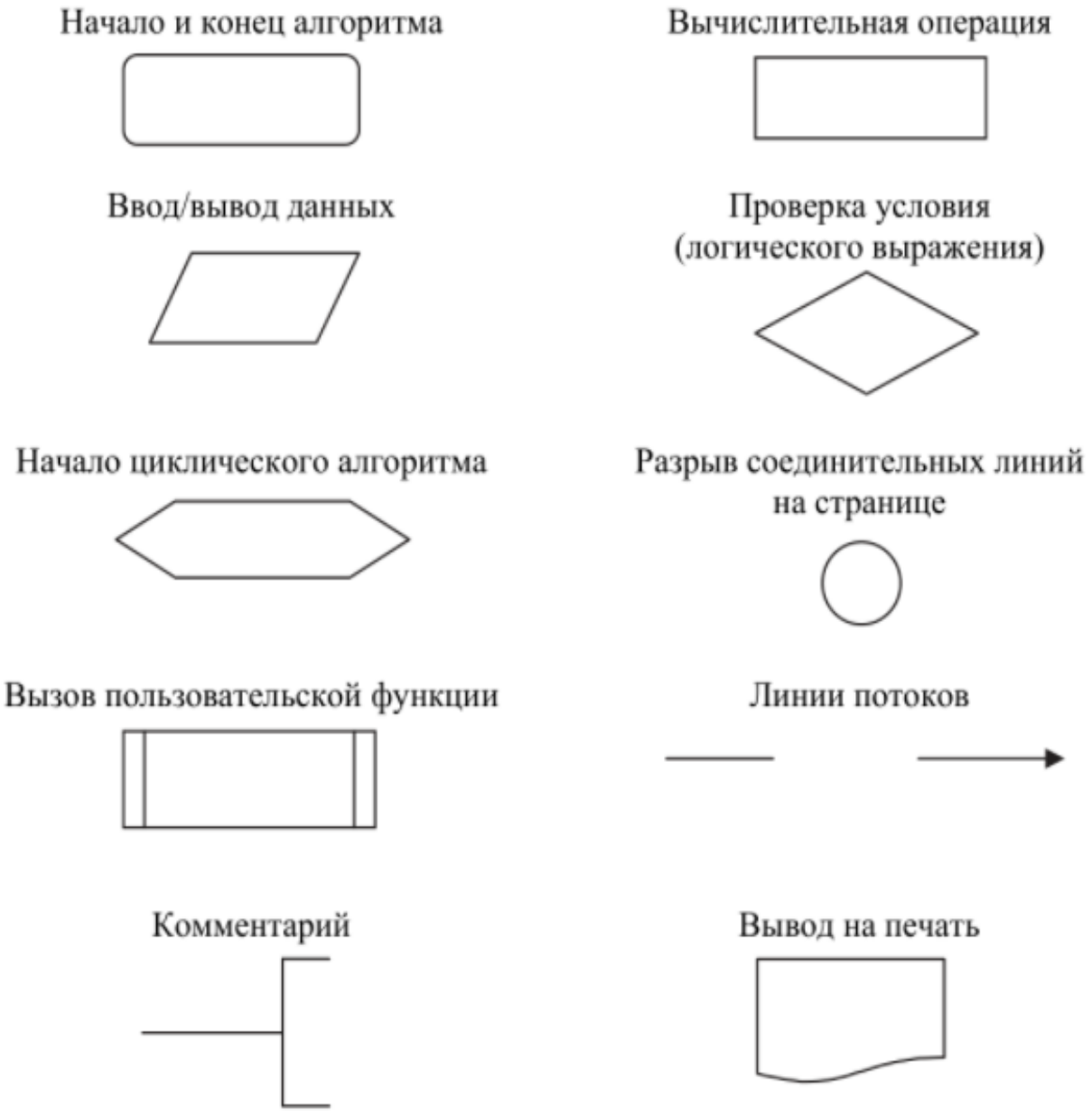

Рисунок 1 – Основные используемые функциональные блоки блок-схем

1.3 Основные этапы решения задач

Процесс решения некоторой задачи в среде программирования, как правило, состоит из нескольких этапов, часть из которых выполняется пользователем, а часть - компьютером.

1-й этап. Общая постановка задачи.

На этом этапе описывается содержание задачи, составляется перечень исходных данных.

2-й этап. Разработка математической модели.

Цель этого этапа состоит в установлении формализованных связей между исходными данными и искомыми результатами. Этап заключается в записи расчетных формул или функциональных зависимостей.

3-й этап. Разработка алгоритма.

Этап состоит в описании последовательности действий, в результате которых может быть получено решение задачи.

4-й этап. Разработка программы.

Программа составляется в полном соответствии с разработанным алгоритмом решения задачи.

5-й этап. Отладка программы.

Процесс поиска ошибок в программе и их устранение.

6-й этап. Анализ результатов.

Позволяет принять решение о необходимости внесения изменений в программу, проведении дополнительных расчетов или их окончании.

1.4 Алфавит языка Python

Изучение любого языка начинается с изучения алфавита, так как из букв складываются слова, а из слов - предложения. То же происходит и при изучении языка программирования. Сначала мы должны уяснить, какие символы можно использовать для записи слов языка, из которых можно формировать определенные конструкции. Итак, в алфавит языка Python входят:

1. Латинские буквы от a до z и от A до Z.

В Python есть различия между прописными и строчными буквами алфавита, например, chislo, CHISLO, Chislo - разные имена переменных.

2. Цифры от 0 до 9.
3. Специальные символы, например +, -, *, /.
4. Зарезервированные (служебные) слова: for, if, class, def и т. д.

1.5 Идентификаторы и общие правила их написания

Для того чтобы программа решения задачи обладала свойством массовости, следует употреблять не конкретные значения величин, а использовать их обозначения для возможности изменения по ходу выполнения программы их значений. Для обозначения в программе переменных и постоянных величин используются имена - **идентификаторы** *(identification* - установление соответствия объекта некоторому набору символов).

Программа на Python представляет собой последовательность инструкций, которые называются **операторами.** Необходимо учитывать, следующее:

1. в идентификатор не могут входить пробелы, специальные символы такие как α, alfa, delta и другие;
2. идентификатор начинается только с буквы или со знака подчеркивания;
3. идентификатор может состоять из букв, цифр и знака подчеркивания;

4. при написании идентификаторов можно использовать как прописные, так и строчные буквы латинского алфавита;

5. идентификатор не должен являться зарезервированным словом.

Например:

summal правильно
2delta ошибка
B1ock_35 правильно
Nomer.doma ошибка
Сумма ошибка

1.6 Оператор присваивания

Действия, выполняемые компьютером в процессе решения задачи, записываются в виде операторов алгоритмического языка. Изменение значения переменной осуществляется оператором присваивания. Присваивание в Python означает связывание значения с некоторым именем переменной.

Действие, выполняемое этим оператором, обозначается знаком «=».

Слева от этого знака записывается имя той переменной, которой нужно присвоить новое значение (например, x=). Справа может быть:

1. Число, например **x=5**.

Прокомментируем предложение, выделенное выше полужирным шрифтом. Дело в том, что при выполнении оператора **x=5** в оперативной памяти по некоторому адресу будет создан объект, получивший значение 5. Затем будет создана переменная **x,** которой будет присвоен адрес этого объекта. Такой механизм занесения значений в ячейки памяти отличает Python от других языков программирования. Однако в дальнейшем при объяснении материала, связанного с операторами присваивания, мы будем использовать упрощенный вариант и выражаться так: **«в ячейку x заносится число 5».**

2. Выражение, например, арифметическое x=(3*a)/2 или логическое x=a>1.

3. Другая переменная, например x=a.

Операторы выполняются в той последовательности, в которой они записаны в программе. Например, фрагмент программы нахождения среднего арифметического включает операторы: a=5 b=10 s=a+b v=s/2.

Возможна запись оператора присваивания в следующем виде:

= оператор - умножение с присваиванием, например, x=5 идентичен оператору x=x*5.

/= оператор - деление с присваиванием, например, x/=5 идентичен оператору x=x/5.

%= оператор - остаток от деления с присваиванием, например, x%=5 идентичен оператору x=x%5.

+= оператор - сложение с присваиванием, например, x+=5 идентичен оператору x=x+5.

-= оператор - вычитание с присваиванием, например, x-=5 идентичен оператору x=x-5.

1.7 Типы данных

Типы данных относятся к самым фундаментальным понятиям любого языка программирования. Для **определения (объявления)** переменных, интерпретатору или компилятору нужна следующая информация:

1. **имя переменной** - по имени осуществляется связь переменной в программе с оперативной памятью компьютера;
2. **тип переменной** - позволяет компилятору определить, какого вида информация хранится в переменной;
3. **значение переменной** - определяет содержание информации, которая помещается в переменную.

В программах, написанных на Python, нет такого раздела как описания переменных. Сравните: в языках программирования Delphi, Pascal, среде программирования Lazarus вы обязаны объявить переменные, которые будете использовать в своей программе в разделе описания Var. В противном случае они просто не станут работать.

В языке программирования Microsoft Visual Basic можно обойтись без объявления переменных в разделе описания Dim, но тогда пользователь должен знать об особенностях типа данных Variant и придет к выводу о том, что переменные все таки лучше описывать. В языке программирования Visual C# описать переменную можно непосредственно в тот момент, когда программист начинает ее использовать. Так вот, ничего из перечисленного в Python нет, вернее, типы данных есть, но в объявлении переменной нет необходимости.

В Python используется так называемая **динамическая типизация,** когда типы данных выясняются во время выполнения программы. Переменная сохраняет ссылку на объект определенного типа, а не сам объект. В момент переприсваивания значения другого типа переменная будет ссылаться на другой объект, при этом изменится и тип переменной. Python будет изменять тип переменной в вашей программе в соответствии с теми данными, которые вы собираетесь «отправить» в ту или иную переменную.

Выяснить, на какой тип данных ссылается переменная, поможет функция **type,** имеющая синтаксис:

type(имя переменной) или **type(значение).**

Например, в программе вы можете написать оператор **type(a)** или **type(196228),** и если переменная **a** имела в первом случае, предположим, целочисленный тип, то ответ будет таким: **<class 'int'>.** Во втором случае, очевидно, что **196228** - целое число, поэтому результат будет таким же. Слово **int** как раз и указывает на принадлежность к целочисленному типу.

Перечислим и дадим характеристику основным типам данных в Python.

Целые типы данных.

Используются для представления целых чисел. Размер числа ограничен объемом имеющейся оперативной памяти. Как уже было сказано, при вызове функции type, как type(10) результатом будет

<class 'int'>

Над данными целого типа определены следующие арифметические операции:

+ сложение;
- вычитание;
* умножение;
/ деление;
// деление нацело;
% остаток от целочисленного деления.

Результат операции % можно вычислить по формуле: a%b=a - (a/b)*b.

Ниже приведены результаты выполнения примеров с помощью операций % и //:

36 % 6 = 0, 5 % 2 = 1, 2 % 5 = 2, 20 // 3 = 6, 0 % 5 = 0.

Вещественные типы данных.

В языке Python допускается представление вещественных (дробных) значений в форме с плавающей и фиксированной точкой.

В форме **с фиксированной точкой** число представляется последовательностью десятичных цифр со знаком «плюс» или «минус». Например:

7.32; 456.721; 0.015;192.0;-15.0.

Форма **с плавающей точкой** (экспоненциальный формат) используется для представления очень больших или очень маленьких чисел. В этой форме число записывается в виде:

$\pm\ \boldsymbol{m\ E} \pm \boldsymbol{p}$,

где m мантисса числа; **E** - символ, обозначающий основание десятичной системы счисления чисел; **p** - порядок (степень) числа.

Например:

7.32e+00; 4.56721e+02; 1.5e-02; 1.92e+02; -1.5e+01.

За вещественными типами данных закреплен тип **float. type(1.5e-02)**

Результатом функции **type** будет **<class 'float'>.**

Строковые типы данных.

Значениями строковых переменных являются строковые константы (строки). В языке Python объекты строкового типа обозначены как **str.**

Результатом функции **type** для **type**("Лабораторная работа") будет **<class 'str'>.**

В следующем примере показаны некоторые приемы работы со строками. Один из самых популярных операторов в Python - это оператор **print,** который служит для вывода на экран текста, при этом текст должен быть заключен в двойные кавычки. Поскольку Python чувствителен к регистру, оператор **print** записывается строчными буквами, в противном случае (при записи **Print** или **PRINT)** работать он не будет.

Далее следует отметить, что значение, заключенное в кавычки и переданное оператору **print,** является строкой. Кавычки, в которые заключается строковая константа, могут быть одиночными либо двойными. Внутри строкового выражения, заключенного в двойные кавычки, могут встречаться строковые выражения, заключенные в одиночные кавычки, но не в двойные.

Наконец, строка, заключенная в тройные кавычки, позволит вывести строковое выражение, размещенное в нескольких строках рисунок 2.

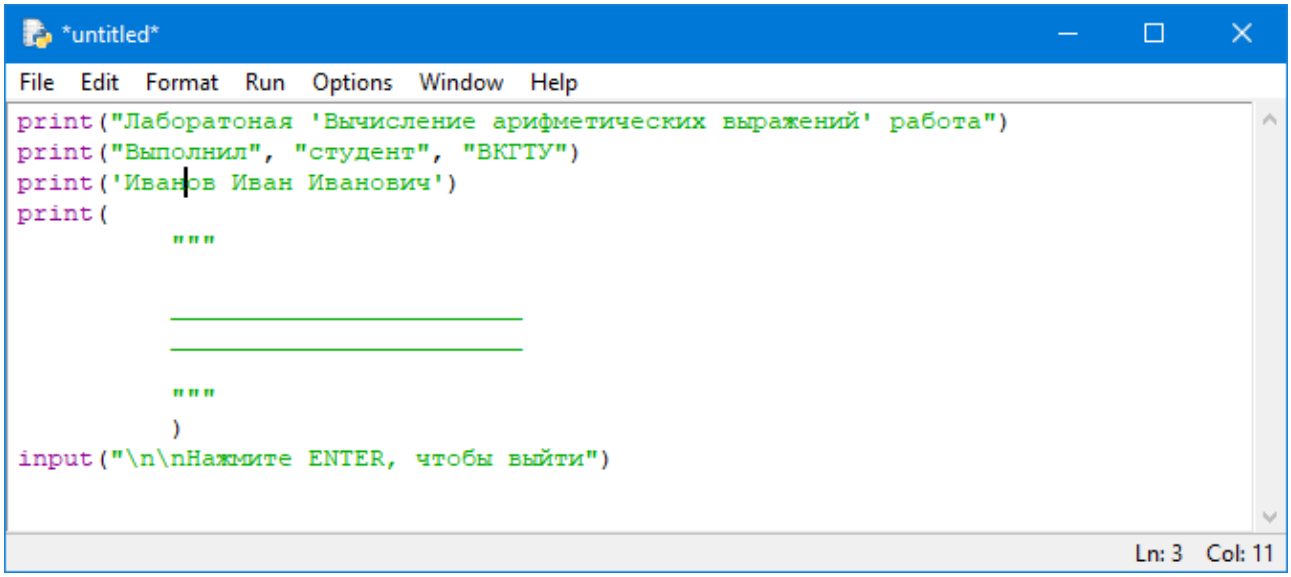

Рисунок 2 - Листинг выполнения операций со строками

Заключительный в этом коде оператор **input** позволяет остановить запущенную на выполнение программу, и выводит сообщение "Нажмите ENTER, чтобы выйти". В данной программной строке используется, так называемая Escape- последовательность, состоящая из обратного слеша и символа **n**, повторяющаяся несколько раз **(\n\n)**. Следует отметить, что во многих языках программирования в консольном режиме используются Escape-последовательности и Python не исключение. С их помощью можно повысить наглядность выводимых на экран строк, например:

\n - переводит курсор на следующую строку;

\t - равносильно использованию клавиши Tab;

\\ - выводит обратный слеш \;

\' - выводит одиночную кавычку;

\" - выводит двойную кавычку.

Конкатенация (соединение) строк в языке Python возможна с помощью символа +. Например:

print("Казахстан " + "2020"

\+ "ВКГТУ").

Для того чтобы слова, соединяемые операцией конкатенации, не сливались, необходимо добавлять пробел (пробелы) перед закрытием кавычки либо сразу после кавычки так, как это показано в примере. Обратный слеш можно использовать при соединении нескольких строк.

Для того чтобы много раз повторить одну и ту же строку, в языке Python допустимо выполнить операцию умножения над строкой, что показано в следующем примере: **print("ВКГТУ"*5).**

Итоговый результат рассмотренных выше примеров, представлен на рисунке 3.

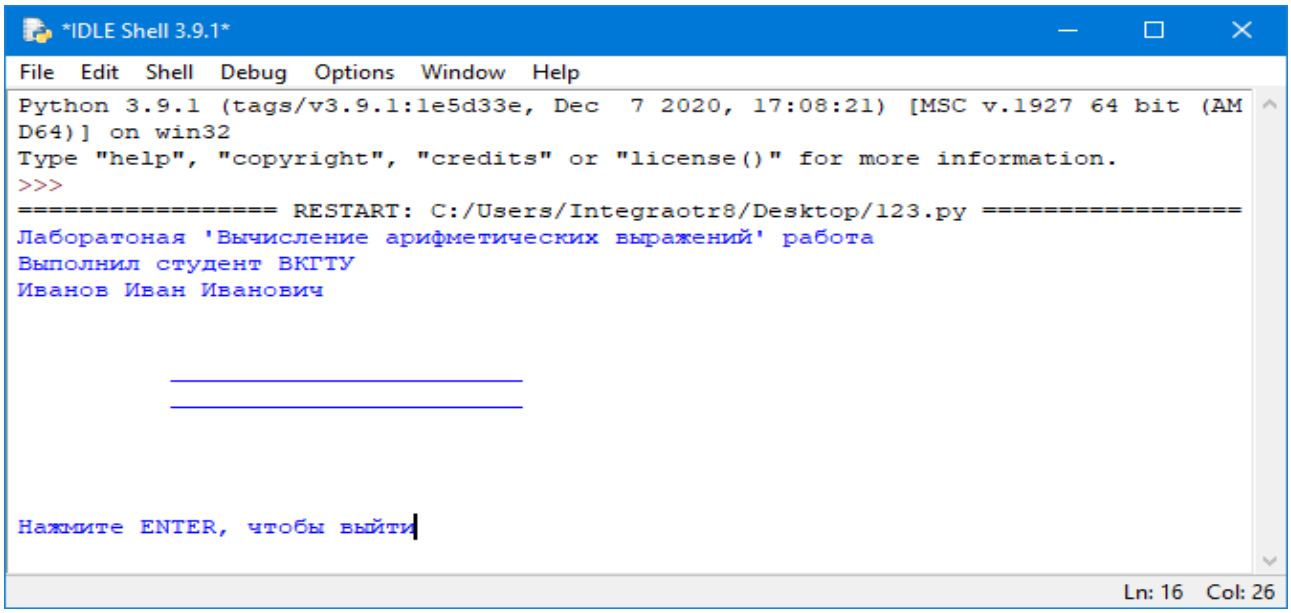

Рисунок 3 - Результат выполнения операций со строками

Логические типы данных (Булевский тип).

Переменные этого типа могут принимать одно из двух значений: **True** (Истина) или **False** (Ложь). Например, значение логического выражения 5>3 есть Истина, а значение 5=3 - Ложь. Значение условия a>5 зависит от значения переменной a и не определено до тех пор, пока неизвестно значение величины a. Объект логического типа в Python обозначается как **bool**.

Применение функции **type** к объектам логического типа **type**(**True**) или **type**(**False**) вызовет ответ <**class** '**bool**'>. Результатом **int**(**True**) будет 1, а **int**(**False**) - 0.

Над данными логического типа могут выполняться логические операции. Некоторые из них приведенные в таблице 1.

Таблица 1 – Логические операции

Операция	Действие	Выражение
and (конъюнкция – логическое умножение)	Логическое И	A **and** B
or (дизъюнкция – логическое сложение)	Логическое ИЛИ	A **or** B
not (отрицание)	Логическое НЕ	**not** A

В Python, в отличие от некоторых других языков программирования, допустима проверка условия, записанного в таком виде: 1<a<5. Также допустима будет проверка условия (1<a) and (a<5), причем значение этого условия истинно только в том случае, если истинны оба входящих в него простых условия.

Мы рассмотрели основные типы данных, которые в дальнейшем будем использовать при изучении языка программирования Python. Еще раз отметим, что в соответствии с динамической типизацией, вы можете изменить тип переменной, просто присвоив ей иное значение. Например, в ходе выполнения операторов

a = 2017; a = "Новый год"; a = 3.14,

переменная **a** последовательно меняет тип данных с целого на строковый, а потом и на вещественный.

1.8 Функции приведения типов

Как известно, компилятор должен иметь возможность определить тип каждой переменной. Кроме того, если одной переменной присваивается значение другой и при этом необходимо преобразование типов (например, от float к int), такое преобразование должно быть явным.

Приведем список функций языка Python, позволяющих осуществлять явные преобразования типов:

1. y=**bool(объект)** - приводит объект к логическому типу. В переменной y будет храниться ссылка на объект логического типа, а не сам объект, хотя до выполнения оператора присваивания переменная y могла содержать ссылку на объект другого типа. Такая же ситуация будет и с другими функциями преобразования типов данных;
2. y = int(**объект**) - приводит объект к целому типу;
3. y = float(**объект**) - приводит объект к вещественному типу;
4. y = str(**объект**) - приводит объект к строковому типу;
5. y = **list**(последовательность) - преобразует элементы последовательности в список;
6. y = **tuple**(последовательность) - преобразует элементы последовательности в кортеж.

Комментарии.

Для лучшего понимания программы в ней часто записывается пояснительный текст - комментарий. Комментарии выполняют несколько важных функций:

1. делают программу легко читаемой, поясняя смысл отдельных программных строк;
2. временно отключают фрагменты программы при ее отладке.

В языке Python комментарий к программной строке возможен с использованием символа #. Например,

```
# Вычисляется сумма двух чисел
sum=a+b
```

1.9 Запись математических функций

В связи с невозможностью записи некоторых стандартных математических функций с клавиатуры персонального компьютера в языке

Python существуют так называемые встроенные функции, с помощью которых пользователь записывает арифметические выражения.

Некоторые математические функции языка Python представлены в таблице 2. Прежде чем использовать математические функции, необходимо в начале программы написать инструкцию **import math**, однако тогда перед упоминанием каждой функции необходимо будет добавлять имя модуля - **math**, например, y=math.sin(x). Другой способ, который позволит избежать многократного вызова модуля math, - сделать следующую запись в начале программы: **from math import ***.

Таблица 2 – Общие математические функции модуля **Math**

Функция	Запись функции в модуле math на языке Python	Выполняемое действие
cos X	**math.cos(X)**	Вычисляет значение косинуса X (X указывается в радианах)
sin X	**math.sin(X)**	Вычисляет значение синуса X (X указывается в радианах)
tg X	**math.tan(X)**	Вычисляет значение тангенса X (X указывается в радианах)
ctg X	**math.cos(X)/ math.sin(X)**	Вычисляет значение котангенса X (X указывается в радианах)
$\|X\|$	**math.fabs(X)**	Вычисляет абсолютное значение числа X
e^X	**math.exp(X)**	Возвращает результат возведения числа **e** в степень X
$\log_{base} X$	**math.log(X, [base])**	Возвращает логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.
ln X	**math.log1p(X)**	Возвращает натуральный логарифм от числа (1 + X). При X → 0 точнее, чем math.log(1+X)
lg X	**math.log10(X)**	Возвращает логарифм X по основанию 10.
mod(x,y)	**math.fmod(X, Y)**	Возвращает остаток от деления X на Y.
$\sqrt{X}$	**math.sqrt(X)**	Возвращает квадратный корень из числа X
acos X	**math.acos(X)**	Возвращает значение арккосинуса от числа X в радианах.
asin X	**math.asin(X)**	Возвращает значение арксинуса от числа X в радианах.
atg X	**math.atan(X)**	Возвращает значение арктангенса от числа X в радианах.
π	**math.pi**	Возвращает значение числа π.
e	**math.e**	Возвращает значение числа e.
degrees(X)	**math.degrees(X)**	Конвертирует радианы в градусы.
radians(X)	**math.radians(X)**	Конвертирует градусы в радианы.
floor(X)	**math.floor(X)**	Возвращает число округленное до ближайшего меньшего целого числа к X.
ceil(X)	**math.ceil(X)**	Возвращает число округленное до ближайшего большего целого числа к X.
X!	**math.factorial(X)**	Возвращает факториал числа X (например 3!=3*2*1).

В таблице 3 представлены некоторые встроенные функции для работы с числами, не требующие подключения модуля **math**.

Таблица 3 – Функции для работы с числами, не требующие подключения модуля **Math**

Функция	Запись функции на языке Python	Выполняемое действие
X ≈	**round(X)**	Возвращает результат округления числа X до ближайшего меньшего целого значения для чисел с дробной частью меньше 0,5 или результат округления числа X до ближайшего большего целого значения для чисел с дробной частью больше 0,5. В случае если дробная часть числа X ровно 0,5, то результат округления числа будет к ближайшему целому четному числу к X.
$\mathbf{X^y}$	**pow(X,Y)** или **X**Y**	Возвращает результат возведения числа X в степеньY.
max(X,…, Z)	**max(**список чисел через запятую**)**	Возвращает большее из списка чисел.
min(X,…, Z)	**min(**список чисел через запятую**)**	Возвращает меньшее из списка чисел.
X+…+Z	**sum(**список чисел через запятую**)**	Возвращает сумму всех чисел из списка.
Int(объект**)**	**int(**объект**)**	Преобразует объект (например, строковое или допустим дробное значение) в целое число
Float(объект**)**	**float(**объект**)**	Преобразует объект (например, строковое или допустим целое значение) в вещественное число

Python предоставляет возможность совершить операции над числами, если программист укажет исходные данные (операнды) и математические действия в операторе **print,** например, инструкция **print(2016+40-20)** выведет на экран число 2036.

1.10 Операции отношения

Операции отношения представлены в таблице 4.

Таблица 4 – Операции отношения

Операция	Описание
Операнд1 < Операнд2	Меньше
Операнд1 > Операнд2	Больше
Операнд1 <= Операнд2	Меньше или равно
Операнд1 >= Операнд2	Больше или равно
Операнд1 != Операнд2	Не равно
Операнд1== Операнд2	Равно

1.11 Контрольные вопросы

1. Что называется алгоритмом? Какими свойствами он обладает?

2. Назовите и поясните способы описания алгоритмов.

3. Нарисуйте функциональные блоки, используемые в блок-схемах. Поясните их назначение.

4. Перечислите этапы решения задачи, выполняемые в процессе ее программирования.

5. Что входит в алфавит языка Python? Поясните понятие «идентификатор» и расскажите об общих правилах написания идентификаторов.

6. В чем заключается действие оператора присваивания? Каковы две формы записи дробных чисел?

7. Какова особенность динамической типизации, используемой в языке Python?

8. Дайте характеристику каждого типа данных языка Python.

9. Какие операции определены над данными целого типа?

10. Каково назначение операторов print и input? Приведите примеры использования таких операторов.

11. Какие логические операции могут выполняться над данными логического типа?

12. Назовите функции приведения типов. Приведите примеры.

13. Для каких целей используются комментарии в программах? Как можно закомментировать участок программного кода в Python?

14. Какие инструкции необходимо прописывать в программах, написанных на языке Python, для использования в них математических функций?