Дата: 2020/11/23
Предмет: Информатика
Тема: Элементы программирования
Тип занятия: Лекция
Группа: ТКБ-20, ТК-20

1. Вступление

Представьте, что перед вами поставлена задача: даны анкеты учащихся некоторого класса и требуется подсчитать количество мальчиков и девочек в этом классе.

Можно предложить следующий способ решения. На школьной доске отведём место для двух чисел – количества мальчиков и количества девочек, и для начала в этих местах запишем по нулю. Далее будем перебирать анкету за анкетой и смотреть, какой пол указан в просматриваемой анкете. Если пол мужской, то будем заменять число на доске, показывающее количество мальчиков, числом, большим на единицу. Если же пол женский, то такую операцию будем выполнять с числом на доске, показывающим количество девочек. Как только мы переберём все анкеты, на доске останутся два числа, равные количеству мальчиков и девочек в классе.

В данном примере отражены следующие особенности программирования.

- Чтобы решить задачу, необходимо придумать метод, который за конечное количество действий приводит к решению (либо к осознанию, что решения нет). Такой метод называется *алгоритмом*.

- При решении задачи приходится работать с данными, которые имеют свой *тип*. К примеру, данными являются переменные *числа*, записанные на доске, а также *упорядоченное множество* анкет, содержащих *текстовую* информацию о том, является ли ученик мальчиком или девочкой.

- В ходе работы алгоритма иногда приходится повторить одно и то же действие несколько раз. Это называется *циклом*. В нашем примере в цикле просматриваются анкеты учеников и меняются числа на доске.

- В некоторых случаях выполняемые действия зависят от тех или иных условий. Такая алгоритмическая конструкция называется *ветвлением*. В примере, приведённом выше, изменение первого или второго числа на доске зависело от записи в просматриваемой анкете.

- Наконец заметим, что алгоритм, приведённый в примере, осуществлял *исполнитель* – человек. Поэтому очень важно, на каком языке записан алгоритм. Компьютеры «понимают» только язык машинных кодов. Но с машинными кодами человеку очень трудно иметь дело. Поэтому были созданы *языки программирования*, которые позволяют по особым правилам с помощью словесных конструкций записать алгоритм и сохранить так называемый *исходный код* программы в обычном текстовом файле. Далее этот файл

обрабатывается или *транслируется* с помощью специальной программы, которая либо создаёт файл машинных кодов, который уже выполняется в операционной системе (в этом случае метод обработки исходного кода называется *компиляцией*), либо сама выполняет записанные в текстовом файле команды друг за дружкой и таким образом возвращает некоторый результат (в данном случае метод обработки исходного кода называется *интерпретацией*).

Надеемся, что после данного рассуждения становятся понятны слова Никлауса Вирта, создателя языка программирования pascal:

$$\textit{алгоритмы} + \textit{структуры данных} = \textit{программы}.$$

Далее мы рассмотрим язык программирования python. Программы, записанные на этом языке, являются *интерпретируемыми*.

Для написания исходных текстов программ нужно скачать и установить бесплатные интегрированные среды программирования:

- либо официальный релиз Python и IDLE,
- либо Thonny,
- либо Python + Jupyter Notebook.

Наиболее простая IDE – Thonny. Её можно скачать по ссылке:
`https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.exe`

2. Структуры данных и ввод-вывод

2.1. Хранение данных в памяти во время работы программы

Данные, используемые в программе, должны находиться в памяти компьютера. Чтобы иметь доступ к области памяти с данными, эта область должна быть поименована (с адресом памяти должно быть связано некоторое имя).

В ходе работы программы содержимое поименованной области памяти может изменяться. В этом случае данная область памяти и имя, с ней связанное, называется *переменной*. Если же содержимое области памяти не меняется, то в этом случае мы имеем дело с *постоянной величиной* или *константой*.

Выделяемая область имеет свой размер (в байтах). К примеру, мы можем выделить область величиной один байт и поместить туда двоичное значение `00100001`. Но что оно будет означать? Это может быть, к примеру, десятичное число `33`, либо символ `!`. Программа должна чётко понимать, с данными какого *типа* она работает.

Имеется два способа задания типа данных: *статический*, когда тип данных указывается при задании переменной или константы, и *динамический*, когда тип

данных сопоставляется с константой или переменной при помещении в неё каких-либо конкретных данных. Языки pascal и c++ являются языками со статической типизацией. Язык python – это язык с динамической типизацией.

3. Основы языка Python

3.1. Встроенные типы данных. Простейший ввод-вывод

3.1.1. Встроенные типы данных

В программном коде мы будем использовать следующие встроенные классы (типы) данных:

Тип данных	Пояснение
`int`	целое число (с произвольным количеством знаков)
`float`	число с плавающей точкой
`str`	строка
`bool`	логическая величина, принимающая значения `True` (истина) или `False` (ложь)
`tuple`	кортеж – упорядоченный (перенумерованный) набор, состоящий из разнотипных элементов
`range`	последовательность чисел, обычно используемая в циклах
`list`	список – упорядоченный (перенумерованный) набор разнотипных элементов
`set`	множество – неупорядоченный (не перенумерованный) набор разнотипных элементов
`dict`	словарь – набор разнотипных элементов, перенумерованных не числами, а уникальными значениями – ключами

Первые шесть типов данных являются *неизменяемыми*. Смысл неизменяемости можно пояснить следующим образом. Сравним работу с переменными в Pascal и Python. Когда переменная определяется в языке Pascal (в разделе определения переменных `var`), то ей выделяется область памяти и с этой областью памяти ассоциируется имя переменной. Содержимое этой области памяти может изменяться, но имя, ассоциированное с областью памяти, сохраняется.

Когда переменная определяется в языке Python (в момент первого появления в коде в присваивании или вводе с клавиатуры), то ей выделяется область памяти и с этой областью памяти ассоциируется имя переменной. Если мы хотим поменять значение неизменяемой переменной, то новому значению выделится другая область памяти, имя переменной будет перенесено на новую область памяти, а старое значение будет потеряно.

Списки, кортежи, множества и словари являются изменяемыми в том плане, что мы имеем возможность менять их содержимое, но при этом адрес переменной указанных типов не меняется (но меняются адреса их элементов).

3.1.2. Замечание по поводу оформления программного кода

Программный код, приводимый в издании, будет оформляться моноширинным шрифтом, возможно, меньшего размера, чем основной текст, и располагаться в ограничивающей рамке. Знаки операторов будут обрамляться пробелами. Ниже приведён пример кода программы и выводимого ею результата. Однострочные комментарии в программе пишутся после знака #.

Листинг 1. *Пример программного кода*

```
# суммирование матриц a и b
import numpy
a = numpy.array([
    [2, 5,-4],
    [3, 7, 2]
    ])
b = numpy.array([
    [ 0, 3, 1],
    [-5, 7,-4]
    ])
print(a+b)
```

Результат:

```
[[ 2  8 -3]
 [-2 14 -2]]
```

3.1.3. Способы задания переменных

Переменная может быть задана с помощью оператора присваивания. Ниже приведён пример задания целочисленной переменной `x`, вещественной переменной `y`, строковых переменных `z` и `w`, списка `a`, кортежа `k`, булевой переменной `b`, множества `s`, словаря `d`.

В Python поддерживается множественное присваивание. В примере ниже оно выполнено в последней строке.

Листинг 2. *Задание переменных с помощью присваивания*

```
x = 2
y = 2.1545
z = 'строка'
```

```
w = "строка"
a = [1, "d", 2.36]
k = (1, 3.14, "d")
b = True
s = {1, 2, "3"}
d = {"Иван": 28, "Пётр": 45}
alpha, beta = 30, 60
```

Переменные можно ввести с клавиатуры. Это делается с помощью функции `input`. Необязательным аргументом этой функции является строка – приглашение пользователю. По умолчанию данная функция возвращает строковую величину. Следовательно, чтобы ввести с клавиатуры целое или вещественное число, нужно результат функции `input` преобразовать с помощью функций `int` и `float` в целый и вещественный тип, соответственно. Ниже показан ввод строковой переменной `s`, целочисленной переменной `n` и вещественной переменной `f`

Листинг 3. *Ввод переменных с клавиатуры*

```
s = input("Введите строку s: ")
n = int(input("Введите целое число n: "))
f = float(input("Введите вещественное число f: "))
```

Булеву переменную более корректно задавать присваиванием. Заметим, что конструкция `b = bool(input())` будет работать. При этом, если с клавиатуры введена непустая строка символов, то в `b` будет записано `True`. Если же была введена пустая строка, то в `b` будет записано `False`.

3.1.4. Преобразование типов данных

Как уже было показано в предыдущем подпункте, строковые данные, вводимые с клавиатуры, можно преобразовать в целочисленные и вещественные (если введённые строки можно в такие данные преобразовать). При этом применялись функции, совпадающие по имени с именами соответствующих типов данных. Такой принцип действует для всех указанных выше типов. К примеру, можно строку преобразовать в список, а список – во множество. Множество содержит только неповторяющиеся значения, и это можно использовать, например, при подсчёте уникальных элементов списка или строки.

Листинг 4. *Преобразование типов*

```
myString = "информатика"
myList = list(myString)
print(myList)
mySet = set(myList)
print(mySet)
```

Результат:

```
['и', 'н', 'ф', 'о', 'р', 'м', 'а', 'т', 'и', 'к', 'а']
{'к', 'р', 'н', 'ф', 'м', 'а', 'о', 'т', 'и'}
```

3.1.5. Допустимые имена переменных

В Python имена переменных должны подчиняться правилу: имя может состоять из букв латиницы, цифр и знака подчёркивания, причём, цифра не может быть первым символом в имени.

Желательно, чтобы имя переменной соответствовало её смысловой нагрузке в программе. Так, `value` более приемлемо, чем `v`. Вместе с тем имя не должно быть слишком длинным.

Прописные и строчные буквы в именах различаются. Так, имена `namedVertices` и `namedvertices` являются различными.

Следует избегать совпадений с уже имеющимися именами, определёнными в языке. Например, не стоит давать переменной, хранящей максимальное из двух чисел, имя `max`, так как есть функция с таким же именем.

3.1.6. Вывод текста на экран

Для вывода текстовой информации служит функция `print`, аргументами которой могут быть величины различных типов. Количество аргументов может быть произвольным.

При решении вычислительных задач будет необходимо уметь выводить вещественные значения с определённым количеством знаков после запятой. Для этого можно применять метод строки `format`.

В примере ниже показано, как во второй строке число `x` выводится в таком же виде, как оно было задано; в третьей строке число выводится в 10 позициях, из которых 3 позиции отведены на дробную часть; в четвёртой строке количество позиций на экране подбирается автоматически, из них 2 позиции отведены на дробную часть; в пятой строке число выводится в экспоненциальной форме (то есть, в виде `1.235e+02` $= 1.235 \cdot 10^2$). Заметим, что целая и дробные части вещественного числа разделяются точкой.

Листинг 5. *Форматный вывод значений*

```
x = 123.45678
print("Значение: ",x)
print("Значение: {:10.3f}".format(x))
print("Значение: {:0.2f}".format(x))
print("Значение: {:10.3e}".format(x))
```

Результат:

```
Значение:  123.45678
Значение:    123.457
Значение: 123.46
Значение:  1.235e+02
```

Иной способ вывода на экран числовых значений и результатов вычисления формул состоит в использовании f-строк. Рассмотрим пример. Пользователь вводит с клавиатуры некоторое вещественное число, а программа выводит квадрат этого числа, оставляя 2 знака после десятичной точки.

Листинг 6. *Пример использования f-строк*

```
n = float(input("Введите число: "))
print(f"Квадрат числа: {n**2 :0.2f}")
```

Результат:

```
Введите число: 3.44545
Квадрат числа: 11.87
```

Видим, что перед строкой ставится префикс `f`, а внутри строки в фигурных скобках помещено выражение, которое вычисляет квадрат введённого числа и к результату вычисления этого выражения применён форматный вывод.

3.1.7. Три полезные функции

В Python имеются три полезные функции:

Функция	Пояснение
`id(obj)`	возвращает идентификатор объекта `obj` (адрес объекта в памяти)
`type(obj)`	возвращает тип объекта `obj`
`dir(obj)`	возвращает список атрибутов объекта `obj`

Листинг 7. *Пример использования функций* `type`, `dir`, `id`

```
x = [1, 2, 4]
print(id(x))
print(type(x))
print(dir(x))
```

Результат:

```
45460136
<class 'list'>
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__',
'__imul__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Можно заметить, что функция `type` выводит не просто наименование типа данных, но именует его `class`. Суть в том, что в Python самым полнейшим образом реализован объектно-ориентированный подход, и, как говорят, в Python всё является классом. Целое число – это класс, вещественное число – класс, и пр. Даже функция – это тоже класс. Поэтому далее, говоря об элементах программ, мы будем упоминать слово объект или экземпляр класса. Так как у классов определяются атрибуты (свойства и методы), то в дальнейшем мы тоже будем использовать указанные термины.

Функцию `dir` часто используют для изучения свойств и методов объектов, а также для вывода списка имён, определённых в текущей области видимости или в некотором модуле (когда аргументом функции является имя модуля).

Функция `id` полезна для понимания того, как в Python реализуется оператор присваивания. Опять же можно провести сравнение языков Pascal и Python.

Если мы в Pascal присваиваем одной переменной, например, x, значение другой переменной, например, y, то из области памяти переменной y в область памяти переменной x переносится содержимое. Имена переменных остаются закреплёнными за выделенными областями памяти.

Если мы в Python присваиваем одной переменной, например, x, значение другой переменной, например, y, то с областью памяти, в которой находится значение, ассоциированное с именем y, будет и ассоциировано имя x, так что два имени переменных будут указывать на одну и ту же область памяти. Указанный подход к понятию переменной понуждает нас более осторожно относиться к применению операций присваивания.

Листинг 8. *Что происходит при присваивании*

```
x = [1,2,3]
y = ["a","b"]
print(x,y)
print(id(x),id(y))
x = y
print(x,y)
print(id(x),id(y))
```

Результат:

```
[1, 2, 3] ['a', 'b']
45525752 45369184
['a', 'b'] ['a', 'b']
45369184 45369184
```

Так как в Python имена переменных – это «ярлыки» к областям памяти, то можно одновременно «перевешивать» эти «ярлыки« для обмена значениями переменных без использования промежуточной переменной.

Листинг 9. *Обмен значениями переменных*

```
frst = "Петя"
scnd = "Вася"
print(frst, scnd)
frst, scnd = scnd, frst
print(frst, scnd)
```

Результат:

```
Петя Вася
Вася Петя
```

3.2. Арифметические операторы

Перечислим арифметические операторы, используемые в Python.

Оператор	Пояснение
+	сложение
-	вычитание
*	умножение
/	деление
**	возведение в степень
//	неполное частное целых чисел
%	остаток от деления одного целого числа на другое

Листинг 10. *Арифметика в Python*

```
x = 17
y = 7
print('x + y =',x + y)
print('x - y =',x - y)
print('x * y =',x * y)
print('x / y =',x / y)
print('x в степени y =',x ** y)
print('a div b =',x // y)
print('a mod b =',x % y)
```

Результат:

```
x + y = 24
x - y = 10
x * y = 119
x / y = 2.4285714285714284
x в степени y = 410338673
a div b = 2
a mod b = 3
```

3.3. Модули и пакеты

Язык Python позволяет собирать пользовательские функции в отдельные файлы с расширением py (впрочем, такое расширение имеют все файлы с исходным

кодом Python). Файлы, в которых собраны определения функций и/или классов, называются *модулями*.

Несколько модулей составляют *пакет*. Физически пакет – это папка, в которую сложены файлы модулей, а также файл `__init__.py`, в котором может храниться служебная информация (данный файл может быть и пустым, но сам файл с таким именем обязан быть в папке пакета). Именем пакета является имя папки пакета.

Чтобы использовать содержимое, определённое в пакете или модуле, используется инструкция `import`. В следующем пункте мы перечислим функции, определённые в модуле `math` и далее приведём пример создания собственного пакета и его использования.

3.3.1. Модуль math

Чтобы использовать математические функции, необходимо подгрузить модуль `math`. Перечислим функции, определённые в этом модуле.

Функция	Пояснение
`fabs(x)`	абсолютная величина (модуль) числа x
`floor(x)`	целая часть «снизу», наибольшее целое число, меньшее или равное x
`ceil(x)`	целая часть «сверху», наименьшее целое число, большее или равное x
`exp(x)`	экспонента e^x
`log(x)`	натуральный логарифм $\ln x$
`log(x,a)`	логарифм по произвольному основанию $\log_a x$
`pow(x,n)`	степень x^n
`sqrt(x)`	квадратный корень $\sqrt{x}$
`pi`	число $\pi \approx 3.14159$
`e`	число $e \approx 2.71828$
`sin(x)`	синус $\sin x$
`cos(x)`	косинус $\cos x$
`tan(x)`	тангенс $\mathrm{tg}\, x$
`asin(x)`	арксинус $\arcsin x$
`acos(x)`	арккосинус $\arccos x$
`atan(x)`	арктангенс $\mathrm{arctg}\, x$
`radians(x)`	перевод угла x из градусной меры в радианную
`degrees(x)`	перевод угла x из радианной меры в градусную

Как получить доступ к данным функциям? Если мы запишем инструкцию `import math`, то получим доступ ко всем функциям и объектам, определённым в модуле. Вызов функций в тексте программы осуществляется с помощью селектора (имени модуля, после которого идет точка, а за ней следует имя функции). В примере ниже показано вычисление значения выражения $\sin \frac{\pi}{4}$.

Листинг 11. *Использование математического модуля*

```
import math
print(math.sin(math.pi/4))
```

Результат:

```
0.7071067811865475
```

В следующем примере показано вычисление площади треугольника по формуле: $s = 2R^2 \sin\alpha \sin\beta \sin\gamma$ по заданному радиусу описанной окружности R и двум углам α и β, заданным в градусах (третий угол γ вычисляется по первым двум).

Листинг 12.

```
import math
# ввод данных
r = float(input("Введите радиус описанной окружности: "))
alpha = float(input("Введите первый угол треугольника (в градусах): "))
beta = float(input("Введите второй угол треугольника (в градусах): "))
# вычисление третьего угла
gamma = 180 - alpha - beta
# перевод из градусов в радианы
alpha = math.radians(alpha)
beta = math.radians(beta)
gamma = math.radians(gamma)
# вычисление площади и вывод её на экран
s = 2 * r * math.sin(alpha) * math.sin(beta) * math.sin(gamma)
print(f"Площадь треугольника: s")
```

Результат:

```
Введите радиус описанной окружности: 3
Введите первый угол треугольника (в градусах): 36
Введите второй угол треугольника (в градусах): 47
Площадь треугольника: 2.560048006432965
```

Инструкция `import` позволяет задать псевдоним загружаемому модулю (это делается, к примеру, если имя модуля слишком длинно). Тогда в тексте программы вместо исходного имени модуля нужно использовать псевдоним.

Листинг 13. *Использование псевдонима*

```
import math as m
print(m.sin(m.pi/4))
```

Чтобы получить доступ только к некоторой функции, определённой в модуле, можно использовать инструкцию

`from math import` *`имя_функции`*

В этом случае в тексте программы селектор не используется.

Листинг 14. *Загрузка отдельных функций*

```
from math import sin, pi
print(sin(pi/4))
```

Необходимо заметить, что селектор также не используется в случае, когда загрузка содержимого модуля была осуществлена с использованием инструкции `from math import *`

Листинг 15. *Загрузка всех функций*

```
from math import *
print(sin(pi/4))
```

3.4. Определение пользовательских функций

Пользовательские функции применяются тогда, когда какой-то участок кода приходится вызывать много раз (не обязательно подряд). Приведём синтаксис определения пользовательской функции.

Определение 1. *Пользовательская функция*

```
def имя_функции(последовательность_аргументов):
    выполняемые команды
    return результат
```

Чтобы задать собственную функцию, необходимо дать ей *`имя`*, указать в скобках через запятые аргументы функции (скобки ставятся даже тогда, когда аргументов нет), и после двоеточия с отступом записать код функции. То, что записано далее с отступом, составляет тело функции. После `return` записывается то, что будет возвращать функция в качестве результата. Выполнив инструкцию `return`,

функция заканчивает работу, так что если после указанной инструкции в теле функции есть ещё команды, они выполнены не будут.

Обязательный отступ в Python играет (почти) такую же роль, что и операторные скобки `begin..end` в Pascal. Отсюда следует вывод, что нельзя делать отступы произвольно, ибо отступы имеют значение (к примеру, в Pascal отступы не имеют значения, но использовались для повышения удобочитаемости кода; в Python так делать нельзя). На протяжении всего кода программы нужно соблюдать размеры отступов. По умолчанию на один отступ отводят 4 пробела. Далее мы покажем, что в программе может быть несколько уровней отступов.

Определим, к примеру, функцию $f(x) = \dfrac{3x^3 - 4x^2 + x - 1}{x^3 - 11x + 1}$ и покажем, как её вызвать в программе. Заметим, что в данном примере можно обойтись без промежуточной переменной `y`. Мы ввели её только для того, чтобы показать, что область видимости переменных `x` и `y` ограничена функцией, и попытка вывести содержимое этой переменной на экран приводит к ошибке.

Листинг 16. *Задание и вызов функции* $f(x)$

```
def f(x):
    y = (3 * x**3 + 4 * x**2 + x + 1) / (x**3 - 11 * x + 1)
    return y

x = 10
print(f(x))
print(x)
print(y)
```

Результат:

```
3.8282828282828283
10
Traceback (most recent call last):
  File "D:\example.py", line 8, in <module>
    print(y)
NameError: name 'y' is not defined
```

3.5. Инструкция ветвления if

Инструкция ветвления применяется в том случае, если необходимо выполнить те или иные действия в зависимости от истинности или ложности некоторого условия (или условий).

Определение 2. *Синтаксис ветвления*

```
if условие_1:
    команды, выполняемые, если условие_1 истинно
elif условие_2:
    команды, выполняемые, если условие_2 истинно
..
else:
    команды, выполняемые, если все условия ложны
```

Поясним синтаксис. Сначала проверяется истинность *условия* $_1$. Если оно истинно, то выполняются все действия, записанные после двоеточия с отступом.

Если *условие* $_1$ ложно, то проверяется истинность *условия* $_2$, записанного после `elif`, что является сокращением пары `else if`. Если *условие* $_2$ истинно, то выполняются все дальнейшие действия, записанные с отступом после двоеточия. Блок `elif` может повторяться в условной инструкции произвольное количество раз.

Если ни одно из проверяемых условий не оказалось истинным, то выполняются действия, записанные с отступом после `else`.

3.5.1. Условия

Условия – это выражения, которые возвращают логическое значение `True` или `False` (более научно – логические выражения).

Простейшие условия можно сформировать с помощью операторов сравнения:

Оператор	Пояснение
`==`	равно
`!=`	не равно
`>`	больше
`>=`	больше или равно
`<`	меньше
`<=`	меньше или равно

Если мы желаем узнать, находится ли какое-либо значение в строке, списке, кор-

теже, множестве или словаре, то используем оператор членства `in`. Приведём пример такого рода проверки.

Листинг 17. *Проверка членства*

```
print("Иван" in "Иван Иванович Иванов")
print("Иван" in {"Иван": 28,"Пётр": 45})
print("Иван" in ["Степан", "Пётр"])
print("Иван" in {"Иван", "Пётр", "Степан"})
print("Маня" in ("Иван", "Пётр", "Степан"))
```

Результат:

```
True
True
False
True
False
```

Если мы желаем узнать, указывают ли два имени переменных на одну и ту же область памяти, нужно использовать оператор тождественности `is`. Приведём пример такого рода проверки.

Листинг 18. *Проверка тождественности*

```
x = 2
y = x
z = 2
print(x is y)
print(x is z)
print(id(x), id(y), id(z))
```

Результат:

```
True
True
1878877344 1878877344 1878877344
```

Условия можно соединять с помощью логических операторов `or` (*или*, дизъюнкция), `and` (*и*, конъюнкция), `not` (*не*, отрицание). Кроме того, в Python поддерживаются множественные (двойные, тройные и т. д.) неравенства. К примеру, условие $x \in [-3; 5)$ можно записать в виде `x >= -3 and x < 5`, либо в виде `-3 <= x <`

5.

Напомним, что дизъюнкция нескольких условий применяется в том случае, когда нам нужно, чтобы хотя бы одно из условий выполнилось (дизъюнкция ложна тогда и только тогда, когда все условия ложны, а если хотя бы одно условие истинно, то и дизъюнкция истинна). Конъюнкция нескольких условий применяется в том случае, когда нам нужно, чтобы все условия выполнились (конъюнкция истинна тогда и только тогда, когда хотя бы одно из условий истинно, а если хотя бы одно условие ложное, то и конъюнкция ложна). Отрицание истинного условия есть ложное условие, и наоборот, отрицание ложного условия есть истинное условие.

Указанные свойства логических операций можно систематизировать в так называемой таблице истинности.

A	B	not A	A or B	A and B
False	False	True	False	False
False	True	True	True	False
True	False	False	True	False
True	True	False	True	True

Заметим, что операторы `or` и `and` могут быть применены не только к двум операндам, но и к большему их количеству. К примеру, могут быть конструкции `A or B or C or D` или `A and B and C and D`.

Если в логическом выражении нет скобок, то наивысший приоритет имеет оператор `not`, затем – оператор `and`, и самый низкий приоритет у `or`.

Приведём пример решения задачи проверки, находится ли число y между числами x и z (то есть, выполняется ли одно из условий – $x \leqslant y \leqslant z$ или $z \leqslant y \leqslant x$). Условия, прописанные в примере, не содержат скобок, так как учитывается приоритет выполнения логических операторов (операторы сравнения имеют более высокий приоритет, чем логические операторы).

Листинг 19. *Приоритет логических операторов*

```
x = float(input("Введите x: "))
y = float(input("Введите y: "))
z = float(input("Введите z: "))
# способ 1
if x <= y and y <= z or z <= y and y <= x:
    print("y находится между x и z")
else:
    print("y не находится между x и z")
# способ 2
```

```
if x <= y <= z or z <= y <= x:
    print("y находится между x и z")
else:
    print("y не находится между x и z")
```

Результат:

```
Введите x: 0.2
Введите y: 1.8
Введите z: 0.7
y не находится между x и z
y не находится между x и z
```
