12. Опишите правила построения выражений в языке Python.

Все данные в языке Python представлены объектами. Объект - некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства (атрибуты) и операции над ними (методы). Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Каждый объект имеет тип данных и значение. Для доступа к объекту предназначены переменные. При инициализации в переменной сохраняется ссылка на объект (адрес объекта в памяти компьютера). Благодаря этой ссылке можно в дальнейшем изменять объект из программы.

Каждая переменная должна иметь уникальное имя, состоящее из латинских букв, цифр и знаков подчеркивания, причем имя переменной не может начинаться с цифры. Кроме того, следует избегать указания символа подчеркивания в начале имени, поскольку идентификаторам с таким символом определено специальное назначение. Например, имена, начинающиеся с символа подчеркивания, не импортируются из модуля с помощью инструкции from module import \*, а имена, включающие по два символа подчеркивания - в начале и в конце, для интерпретатора имеют особый смысл.

В качестве имени переменной нельзя использовать ключевые слова. Помимо ключевых слов, следует избегать совпадений со встроенными идентификаторами. Дело в том, что, в отличие от ключевых слов, встроенные идентификаторы можно переопределять, но это может вызвать проблемы в дальнейшем написании программы.

В Python 3 объекты могут иметь следующие типы данных:

bool - логический тип данных. Может содержать значения true или false, которые ведут себя как числа 1 и 0 соответственно.

Nonetype-объект со значением None (обозначает отсутствие значения).

В логическом контексте значение None интерпретируется как False.

int - целые числа. Размер числа ограничен лишь объемом оперативной памяти.

float-вещественные числа;

complex -комплексные числа;

str-Unicode-cтpoки;

bytes -неизменяемая последовательность байтов;

bytearray -изменяемая последовательность байтов;

list-списки.

Тип данных list аналогичен массивам в других языках программирования.

tuple – кортежи;

dict - словари. Тип данных dict аналогичен ассоциативным массивам в других языках программирования.

tset- множества (коллекции уникальных объектов);

frozenset - неизменяемые множества;

ellipsis - обозначается в виде трех точек или слова Ellipsis. Тип ellipsis используется в расширенном синтаксисе получения среза;

function – функции;

module – модули;

type- классы и типы данных. Все данные в языке Python являются объектами, даже сами типы данных!

Основные типы данных делятся на изменяемые и неизменяемые. К изменяемым типам относятся списки, словари и тип bytearray.

К неизменяемым типам относятся числа, строки, кортежи, диапазоны и тип bytes. Например, чтобы получить строку из двух других строк, необходимо использовать операцию конкатенации, а ссылку на новый объект присвоить переменной.

Кроме того, типы данных делятся на последовательности и отображения. К последовательностям относятся строки, списки, кортежи, диапазоны, типы bytes и bytearray, а к отображениям - словари.

Последовательности и отображения поддерживают механизм итераторов, позволяющий произвести обход всех элементов с помощью метода \_next\_ ( ) или функции next ( )

В языке Python используется динамическая типизация. Это означает, что при присваивании переменной значения интерпретатор автоматически относит переменную к одному из типов данных.

Python в любой момент времени изменяет тип переменной в соответствии с данными, хранящимися в ней.

Для преобразования типов переменных существуют следующие функции

bool ([<Объект>]) - преобразует объект в логический тип данных.

int ( [ <Объект> [, <Система счисления>] ] ) - преобразует объект в число. Во втором параметре можно указать систему счисления (значение по умолчанию - 10)

float ( [ <Число или строка> J ) - преобразует целое число или строку в вещественное число.

str ( [ <Объект> J) - преобразует объект в строку.

bytes (<Строка>, <Кодировка> [, <Обработка Ошибок>] ) - преобразует строку в объект типа bytes. В третьем параметре моrут быть указаны значения "strict" (значение по умолчанию), "replace" или "ignore". bytes (<Последовательность>) - преобразует последовательность целых чисел от 0 до 255 в объект типа bytes. Если число не попадает в диапазон, то возбуждается исключение ValueError.

bytearray (<Строка>, <Кодировка> [, <Обработка Ошибок>)] ) - преобразует строку в объект типа bytearray. В третьем параметре могут быть указаны значения "strict" (значение по умолчанию), "replace" или "ignore"

bytearray(<Последовательность>)- преобразует последовательность целых чисел от 0 до 255 в объект типа bytearray. Если число не попадает в диапазон, то возбуждается исключение ValueError.

list(<Последовательность>) - преобразует элементы последовательности в список.

tuple(<Последовательность>) - преобразует элементы последовательности в кортеж.

Производить операции над числами позволяют следующие операторы:

+ - сложение:

- - вычитание

\* - умножение:

/ - деление. Результатом деления всегда является вещественное число, даже если производится деление целых чисел. Обратите внимание на эту особенность, если вы раньше программировали на Python 2. В Python 2 при делении целых чисел остаток отбрасывался и возвращалось целое число, в Python 3 поведение оператора изменилось.

/ / - деление с округлением вниз. Вне зависимости от типа чисел остаток отбрасывается.

% - остаток от деления

\*\* - возведение в степень

унарный минус (-) и унарный плюс (+)

Операции над числами разных типов возвращают число, имеющее более сложный тип из типов, участвующих в операции. Целые числа имеют самый простой тип, далее идут вещественные числа и самый сложный тип - комплексные числа. Таким образом, если в операции участвуют целое число и вещественное, то целое число будет автоматически преобразовано в вещественное число, а затем произведена операция над вещественными числами. Результатом этой операции станет вещественное число.

52. Опишите методы генерирования списков.

Самым простым способом создания списка является обычное присваивание ему необходимых значений или объектов. В том случае, когда элементов последовательности немного, их можно просто перечислить один за другим. Но если их количество переваливает за десяток, можно использовать генератор списка. **Данная конструкция обеспечивает его автоматическое заполнение, исходя из определенных инструкций.**

Один из часто используемых способов генерации списков в Python — с условием. Данные конструкции языка Python **позволяют задавать определенные условия для выполнения такой операции**.

Предположим, например, что нам требуется получить список високосных годов в определенном диапазоне. Для начала мы могли бы использовать такой цикл:

leaps =[]

for year in range(1900, 1940):

if (year % 4 == 0 and year % 100 != 0) or (year % 400 ==0):

leaps.append(year)

Но можно использовать и такой способ: заключить выражение и цикл с дополнительным условием в квадратные скобки, в котором цикл используется для создания элементов списка, а условие используется для исключения нежелательных элементов. В результате мы получаем генератор списка в простейшем его виде который записывается, как показано ниже:

[item for item in iterable]

Это выражение вернет список всех элементов объекта iterable и семантически ничем не отличается от выражения list(iterable).

Интересными генераторы списков делают две особенности - они могут использоваться как выражения, и они допускают включение условной инструкции, вследствие чего мы получаем две типичные синтаксические конструкции использования генераторов списков:

[expression for item in iterable]

[expression for item in iterable if condition]

Вторая форма записи эквивалентна циклу:

temp =[]

for item in iterable:

if condition:

temp.append(expressiofl)

Обычно выражение expression является либо самим элементом item, либо некоторым выражением с его участием. Конечно, генератору списков не требуется временная переменная temp[ ], которая необходима в версии с циклом for ... in.

65. Напишите программу, которая определяет число отрицательных чисел во введенной с клавиатуры последовательности (длина последовательности неограниченна).

# coding: utf-8

print('Task 65 Cipher 29 Second year of Mogilev State Polytechnic College')

max = +1

max = int(input("Enter the maximum positive number of the sequence: ")) # ввод максимума

t = 0.0

i = 0

while True: #цикл ввода последовательности и проверка на отрицательность–увеличить счетчик при совпадении- если число больше максимума выход из цикла

t = float(input("Input the sequence : "));

if t <= 0:

i += 1

if t >= max:

break

print("The number of Negative numbers - ", i)



76. Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

- количество строк, не содержащих ни одного нулевого элемента;

- максимальное из чисел, в заданной строке массива.

rows = int(input('Rows: '))

cols = int(input('Columns: '))

arr = [[int(input()) for i in range(cols)] for j in range(rows)]

for row in arr:

print(row)

print('Count rows without zeros: ')

print(len([i for i in arr if 0 not in i]))

n = int(input('Enter row index where you wish to locate the maximum: ')) # индекс массива

print(f'Maximum in row: {max(arr[n])}')

