Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование. Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается через декораторы, более полноценная поддержка обеспечивается дополнительными фреймворками.

В данном курсе вы узнаете о понятиях в языке Python, таких как: переменные, типы данных, условные операторы, списки и циклы.

Переменные:

Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. В python применяется два типа наименования переменных: camel case и underscore notation.

Camel case подразумевает, что каждое новое подслово в наименовании переменной начинается с большой буквы. Например:

userName = "Tom"

Underscore notation подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания. Например:

user\_name = "Tom"

И также надо учитывать регистрозависимость, поэтому переменные name и Name будут представлять разные объекты.

Определив переменную, мы можем использовать в программе. Например, попытаться вывести ее содержимое на консоль с помощью встроенной функции print:

name = "Tom" # определение переменной name

print(name) # вывод значения переменной name на консоль

Отличительной особенностью переменной является то, что мы можем менять ее значение в течение работы программы:

name = "Tom" # переменной name равна "Tom"

print(name) # выводит: Tom

name = "Bob" # меняем значение на "Bob"

print(name) # выводит: Bob

Переменная хранит данные одного из типов данных. В Python существует множество различных типов данных. В данном случае рассмотрим только самые базовые типы: bool, int, float, complex и str.

Тип bool представляет два логических значения: True (верно, истина) или False (неверно, ложь). Значение True служит для того, чтобы показать, что что-то истинно. Тогда как значение False, наоборот, показывает, что что-то ложно. Пример переменных данного типа:

isMarried = False

print(isMarried) # False

isAlive = True

print(isAlive) # True

Тип int представляет целое число, например, 1, 4, 8, 50. Например:

age = 21

print("Возраст:", age) # Возраст: 21

Тип float представляет число с плавающей точкой, например, 1.2 или 34.76. В качестве разделителя целой и дробной частей используется точка.

height = 1.68

pi = 3.14

weight = 68.

print(height) # 1.68

print(pi) # 3.14

print(weight) # 68.0

Тип complex представляет комплексные числа в формате вещественная\_часть+мнимая\_частьj - после мнимой части указывается суффикс j

complexNumber = 1+2j

print(complexNumber) # (1+2j)

Тип str представляет строки. Строка представляет последовательность символов, заключенную в одинарные или двойные кавычки, например "hello" и 'hello'. В Python 3.x строки представляют набор символов в кодировке Unicode

message = "Hello World!"

print(message) # Hello World!

При этом если строка имеет много символов, ее можем разбить ее на части и разместить их на разных строках кода. В этом случае вся строка заключается в круглые скобки, а ее отдельные части - в кавычки:

text = ("Laudate omnes gentes laudate "

"Magnificat in secula ")

print(text)

Если же мы хотим определить многострочный текст, то такой текст заключается в тройные двойные или одинарные кавычки:

text = ' ' 'Laudate omnes gentes laudate

Magnificat in secula

Et anima mea laudate

Magnificat in secula

' ' '

print(text)

С помощью встроенной функции type() динамически можно узнать текущий тип переменной:

userId = "abc" # тип str

print(type(userId)) # <class 'str'>

Условный оператор ветвления if

Оператор ветвления if позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

Конструкция if

if выражение:

инструкция\_1

инструкция\_2

...

инструкция\_n

После оператора if записывается выражение. Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое True. После выражения нужно поставить двоеточие ":”.

a = 3

if a == 3:

print("hello 2") # hello 2

Конструкция if – else.

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция if – else.

if выражение:

инструкция\_1

инструкция\_2

...

инструкция\_n

else:

инструкция\_a

инструкция\_b

...

инструкция\_x

Пример:

a = 3

if a > 2:

print("H")

else:

print("L")

Конструкция if – elif – else.

if выражение\_1:

инструкции\_(блок\_1)

elif выражение\_2:

инструкции\_(блок\_2)

elif выражение\_3:

инструкции\_(блок\_3)

else:

инструкции\_(блок\_4)

Пример:

a = int(input("введите число:"))

if a < 0:

print("Neg")

elif a == 0:

print("Zero")

else:

print("Pos")

Если пользователь введет число меньше нуля, то будет напечатано "Neg", равное нулю – "Zero", большее нуля – "Pos".

Оператор цикла while

Оператор цикла while выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе if. Синтаксис оператора while выглядит так.

while выражение:

инструкция\_1

инструкция\_2

...

инструкция\_n

a = 0

while a < 7:

print("A")

a += 1

Буква "А” будет выведена семь раз в столбик.

При работе с циклами используются операторы break и continue.

Оператор break предназначен для досрочного прерывания работы цикла while.

Пример.

a = 0

while a >= 0:

if a == 7:

break

a += 1

print("A")

В приведенном выше коде, выход из цикла произойдет при достижении переменной a значения 7. Если бы не было этого условия, то цикл выполнялся бы бесконечно.

Оператор continue запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

Пример.

a = -1

while a < 10:

a += 1

if a >= 7:

continue

print("A")

При запуске данного кода символ "А” будет напечатан 7 раз, несмотря на то, что всего будет выполнено 11 проходов цикла.

Оператор цикла for

Оператор for выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

Пример.

for i in range(5):

print("Hello")

В результате "Hello” будет выведено пять раз.

Внутри тела цикла можно использовать операторы break и continue, принцип работы их точно такой же как и в операторе while.

Если у вас есть заданный список, и вы хотите выполнить над каждым элементом определенную операцию (возвести в квадрат и напечатать получившееся число), то с помощью for такая задача решается так.

lst = [1, 3, 5, 7, 9]

for i in lst:

print(i \*\* 2)

Списки.

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов.

Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией list:

list('список') # ['с', 'п', 'и', 'с', 'о', 'к']

Список можно создать и при помощи литерала:

s = [] # Пустой список

l = ['s', 'p', ['isok'], 2] # Заполненный список

Методы списков:

list.append(x) Добавляет элемент в конец списка

list.extend(L) Расширяет список list, добавляя в конец все элементы списка L

list.insert(i, x) Вставляет на i-ый элемент значение x

list.remove(x) Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует

list.pop([i]) Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент

list.index(x, [start [, end]]) Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)

list.count(x) Возвращает количество элементов со значением x

list.sort([key=функция]) Сортирует список на основе функции

list.reverse() Разворачивает список

list.copy() Поверхностная копия списка

list.clear() Очищает список