**Начало работы в Python**

* Программа Python пишется в текстовом файле;
* Команды пишутся на отдельных строках друг под другом;
* Отступы — часть синтаксиса и обозначают вложенную часть циклов и команд ветвления.

**Для отступов в Python значительно удобнее использовать пробелы, а не табуляцию**.

Для ввода одной **строки** данных с клавиатуры используется команда input(). Для вывода результата — print();

a = input() *# Ввод строки*

print(a) *# Вывод переменной*

Для запуска программы application.py на «рабочем столе» необходимо в консоли перейти в папку с программой

$ cd ~/Desktop

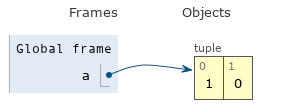
и запустить её:

$ python3 applcation.py

Ссылочная модель данных

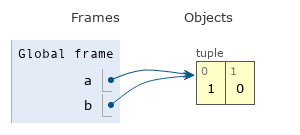
В языке Python, все переменные — ссылки на объекты, хранящиеся в оперативной памяти:

a = (1, 0)



a = (1, 0)

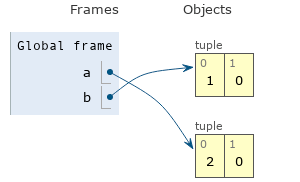
b = a



a = (1, 0)

b = a

a = (2, 0)



Типы данных

Типы данных в Python делятся на *изменяемые* (позволяют изменять внутреннее содержимое переменной) и *неизменяемые*.

Неизменяемые типы данных

* Числовые (int, float, complex);
* Логический (bool)
* Строки (str) **нельзя менять отдельные буквы в строке — только создав новую строку**;
* Кортеж (tuple) **не позволяет изменять набор, но может содержать изменяемые элементы**;
* Замороженное множество (frozenset);
* Функции **имя функции также является переменной, и может быть переопределено**.
* Классы **но не экземпляры классов**

Изменяемые типы данных

* Список (list) — последовательность любых элементов
* Множество (set) — неповторяющийся набор *неизменяемых* элементов
* Словарь (dict) — таблица соответствия *ключ* → *значение*. *Ключ* обязательно *неизменяемый*, *значение* любое.

Для *определения* типа переменной (например x), использовать функцию type. Для *проверти* типа — используется оператор **is**.

*type(x)****is****int.*

Циклы и ветвления

Цикл for

Цикл **for** позволяет "пробегать" по всем элементам списка или генератора.

Синтаксис:

**for** <variable\_name> **in** <data>:

*'''*

*some code*

*'''*

Вывод элементов списка

A = [123, "string", (1,2,3)] *# некоторый список из трёх элементов*

**for** i **in** A:

print(i)

*# или*

**for** i **in** [123, "string", (1,2,3)]:

print(i)

Пример вывода *целочисленной* арифметической прогрессии:

start = 1 *# первый элемент прогрессии*

stop = 100 *# ограничение прогрессии*

step = 2 *# шаг прогрессии*

**for** i **in** range(start, stop, step):

print(i)

Цикл while

Синтаксис:

**while** <условие продолжения цикла>:

*'''*

*some code*

*'''*

Пример вывода *целочисленной* арифметической прогрессии c положительным шагом:

start = 1 *# первый элемент прогрессии*

stop = 100 *# ограничение прогрессии*

step = 2 *# шаг прогрессии*

i = start

**while** i < stop:

print(i)

i += step

Ветвление

Общий синтаксис:

**if** <условие 1>:

*'''*

*выполняется, если <условие 1> истинно*

*'''*

**elif** <условие 2>:

*'''*

*выполняется, если <условие 2> истинно а предыдущие ложны. Может отсутствовать.*

*'''*

**elif** <условие 3>:

*'''*

*выполняется, если <условие 3> истинно а предыдущие ложны. Может отсутствовать.*

*'''*

⋅⋅⋅⋅⋅

**else**:

*'''*

*выполняется, если все условия ложны. Может отсутствовать.*

*'''*

Управление циклом

* **break** — используйте для преждевременной остановки цикла. Выход происходит только из **одного** цикла (в котором написана команда);
* **continue** — используйте для моментального перехода к следующей итерации текущего цикла.

Пример использования:

i = 1

**while** **True**: *# запускаем бесконечный цикл*

i += 1 *# увеличиваем i*

**if** i > 100:

**break** *# останавливаемся, если перешли через 100*

**if** i % 2 == 0:

**continue** *# пропускаем, если число чётное*

print(i) *# печатаем, если дошли до сюда*

Списки, словари и множества

Создание

*# list comprehensions*

*# A = [expression iteration filter]*

A = [i **for** i **in** range(10) **if** i % 2 == 0]

*# A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*

*# dict (set) comprehensions*

*# A = {expression iteration filter}*

A = {i **for** i **in** range(10) **if** i % 2 == 0}

*# A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*

A = {i:i\*i **for** i **in** range(10) **if** i % 2 == 0}

*# A = {0:0, 1:1, 2:4, 3:9, 4:16, 5:25, 6:36, 7:49, 8:64, 9:81}*

Операции со списками

Со списками можно легко делать много разных операций.

| **Операция** | **Действие** |
| --- | --- |
| x **in** A | Проверить, содержится ли элемент в списке. Возвращает **True** или **False**. |
| x **not** **in** A | То же самое, что **not**(x **in** A). |
| A.index(x) | Индекс первого вхождения элемента x в список, при его отсутствии генерирует ошибку. |
| A.count(x) | Количество вхождений элемента x в список. |
| A.append(x) | Добавить в конец списка A элемент x. |
| A.insert(i, x) | Вставить в список A элемент x на позицию с индексом i. Элементы списка A, которые до вставки имели индексы i и больше сдвигаются вправо. |
| A.extend(B) | Добавить в конец списка A содержимое списка B. |
| A.pop() | Удалить из списка последний элемент, возвращается значение удаленного элемента. |
| A.pop(i) | Удалить из списка элемент с индексом i, возвращается значение удаленного элемента. Все элементы, стоящие правее удаленного, сдвигаются влево. |

Работа с множествами

| **Операция** | **Значение** |
| --- | --- |
| x **in** A | принадлежит ли элемент x множеству A (возвращают значение типа bool) |
| x **not** **in** A | то же, что **not** x **in** A |
| A.add(x) | добавить элемент x в множество A |
| A.discard(x) | удалить элемент x из множества A |
| A.remove(x) | удалить элемент x из множества A |
| A.pop() | удаляет из множества один случайный элемент и возвращает его |
| A ⎪ B  A.union(B) | Возвращает множество, являющееся объединением множеств A и B. |
| A ⎪= B  A.update(B) | Записывает в A объединение множеств A и B. |
| A & B  A.intersection(B) | Возвращает множество, являющееся пересечением множеств A и B. |
| A &= B  A.intersection\_update(B) | Записывает в A пересечение множеств A и B. |
| A - B  A.difference(B) | Возвращает разность множеств A и B (элементы, входящие в A, но не входящие в B). |
| A -= B  A.difference\_update(B) | Записывает в A разность множеств A и B. |
| A ^ B  A.symmetric\_difference(B) | Возвращает симметрическую разность множеств A и B (элементы, входящие в A или в B, но не в оба из них одновременно). |
| A ^= B  A.symmetric\_difference\_update(B) | Записывает в A симметрическую разность множеств A и B. |
| A <= B  A.issubset(B) | Возвращает true, если A является подмножеством B. A ⊆ B |
| A >= B  A.issuperset(B) | Возвращает true, если B является подмножеством A. A ⊇ B |
| A < B | A ⊂ B |
| A > B | A ⊃ B |

*Поведение list.discard и list.remove различается тогда, когда удаляемый элемент отсутствует в множестве: list.discard не делает ничего, а метод remove возвращает ошибку, как и list.pop.*

Операции со словарем

| **Операция** | **Значение** |
| --- | --- |
| value = A[key] | Получение элемента по ключу. Если элемента с заданным ключом в словаре нет, то возникает ошибка. |
| value = A.get(key) | Получение элемента по ключу. Если элемента в словаре нет, то get возвращает None. |
| value = A.get(key, default\_value) | То же, но вместо None метод get возвращает default\_value. |
| key **in** A | Проверить принадлежность *ключа* словарю. |
| key **not** **in** A | То же, что not key in A. |
| A[key] = value | Добавление нового элемента в словарь. |
| **del** A[key] | Удаление пары ключ-значение с ключом key. Возбуждает ошибку, если такого ключа нет. |
| value = A.pop(key) | Удаление пары ключ-значение с ключом key и возврат значения удаляемого элемента. Если такого ключа нет, то возбуждается ошибка. |
| value = A.pop(key, default\_value) | То же, но вместо генерации исключения возвращается default\_value. |
| A.pop(key, **None**) | Это позволяет проще всего организовать безопасное удаление элемента из словаря. |
| len(A) | Возвращает количество пар *ключ-значение*, хранящихся в словаре. |

Функции

Простая функция

**def** <имя функции>(<параметры функции>):

*'''*

*тело функции*

*'''*

Для возвращения результата функцией и окончания работы функции используется команда **return**.

Примеры функций:

**def** out\_name():

*'''*

*Функция просто выводит строку.*

*Здесь можно не не применять return*

*'''*

print("My name is NONAME")

**def** my\_out(x):

*'''*

*Функция выводит данные и из тип.*

*'''*

**if** type(x) **is** dict: *# вывод словаря*

print('Dict')

**for** i **in** x:

my\_out(i)

print('> ', end='')

my\_out(x[i])

print('End dict')

**elif** type(x) **in** (set, frozenset, list, tuple):

print('Iter')

**for** i **in** x:

my\_out(i)

print('End iter')

**elif** type(x) **in** (int, float, complex):

print("digit", x)

**elif** type(x) **is** str:

print("str", x)

**elif** type(x) **is** bool:

print("bool", x)

**else**:

print("unknown type")

Функции с параметрами по умолчанию

Синтаксис

**def** <имя функции>(<параметр> = <значение по умолчанию>):

*'''*

*тело функции*

*'''*

Примеры функций:

**def** my\_sum(a, b=10, c=100):

*'''*

*Функция возвращает сумму a+b+c*

*'''*

**return** a + b + c

*# Пример вызова функции*

print(my\_sum(1,2,3)) *# 1 + 2 + 3= 6*

print(my\_sum(1)) *# 1 + 10 + 100 = 111*

print(my\_sum(1, c=4)) *# 1 + 10 + 4 = 15*

**ВАЖНО** Инициализация данных по умолчанию проводиться только один раз — во время создания функции. При вызове функции происходит присваивание незаданным параметрам **инициализированные** значения. Т.е. **не стоит использовать в качестве значений по умолчанию изменяемые данные**.

Функции с неизвестным количеством параметров

Синтаксис

**def** <имя функции>(\*argv, \*\*kwargs):

*'''*

*argv - список неименованных данных*

*kwargs - словарь именованных параметров*

*'''*

Примеры функций:

**def** my\_sum(\*argv):

s = 0

**for** i **in** argv:

s += i

**return** s

*# Пример вызова функции*

print(my\_sum(1,2,3)) *# 1 + 2 + 3= 6*

print(my\_sum(1)) *# 1 = 1*

print(my\_sum(1, 4)) *# 1 + 4 = 5*

Работа со строками

Срезы (slices)

Срез (slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Есть три формы срезов. Самая простая форма среза: взятие одного символа строки, а именно, S[i] — это срез, состоящий из одного символа, который имеет номер i, при этом считая, что нумерация начинается с числа 0. То есть если S='Hello', то S[0]=='H', S[1]=='e', S[2]=='l', S[3]=='l', S[4]=='o'.

Номера символов в строке (а также в других структурах данных: списках, кортежах) называются **индексом**.

Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера -1. То есть S[-1]=='o', S[-2]=='l', S[-3]=='l', S[-4]=='e', S[-5]=='H'.

Или в виде таблицы:

| **Строка S** | **H** | **e** | **l** | **l** | **o** |
| --- | --- | --- | --- | --- | --- |
| Индекс | S[0] | S[1] | S[2] | S[3] | S[4] |
| Индекс | S[-5] | S[-4] | S[-3] | S[-2] | S[-1] |

Если же номер символа в срезе строки S больше либо равен len(S), или меньше, чем -len(S), то при обращении к этому символу строки произойдет ошибка.

Срез с двумя параметрами: S[a:b] возвращает подстроку из b - a символов, начиная с символа c индексом a, то есть до символа с индексом b, не включая его. Например, S[1:4]=='ell', то же самое получится если написать S[-4:-1]. Можно использовать как положительные, так и отрицательные индексы в одном срезе, например, S[1:-1] — это строка без первого и последнего символа (срез начинается с символа с индексом 1 и заканчивается индексом -1, не включая его).

Если опустить второй параметр (но поставить двоеточие), то срез берется до конца строки. Например, чтобы удалить из строки первый символ (его индекс равен 0, то есть взять срез, начиная с символа с индексом 1), то можно взять срез S[1:], аналогично если опустить первый параметр, то срез берется от начала строки. То есть удалить из строки последний символ можно при помощи среза S[:-1]. Срез S[:] совпадает с самой строкой S.

Если задать срез с тремя параметрами S[a:b:d], то третий параметр задает шаг, как в случае с функцией range(), то есть будут взяты символы с индексами a, a + d, a + 2\*d и т.д. При задании значения третьего параметра, равному 2, в срез попадет каждый второй символ, а если взять значение среза, равное -1, то символы будут идти в обратном порядке.

Основные методы

| **Метод** | **Описание** |
| --- | --- |
| s.find(sub, start=0) | Ищет sub в строке s[start:]. -1, если не найдена |
| s.index(sub, start=0) | Ищет sub в строке s[start:]. Ошибка, если sub не найдена |
| s.replace(old, rep, cnt=-1) | Возвращает строку s, с заменёнными old на rep, не более чем cnt раз. Если cnt == -1, то заменяет все вхождения |
| s.split(sep=**None**, maxcnt=-1) | Создаёт список подстрок s, используя sep в качестве разделителя. Если sep == None, разделяет по любым пробельным символам |