# 1 Типы данных

Python – интерпретируемый язык. Интерпретатор Python исполняет программу по одному предложению за раз. В Python для структурирования кода используются пробелы (или знаки табуляции), а не фигурные скобки, как во многих других языках. Двоеточием обозначается начало блока кода с отступом (например после объявления функции, условного оператора if, цикла for и т.д.), весь последующий код до конца блока должен быть набран с точно таким же отступом.

Предложения в Python не обязаны завершаться точкой с запятой. Но ее можно использовать, чтобы отделить друг от друга предложения, находящиеся в одной строчке:

```
a = 5; b = 6; c = 7
```

Впрочем, писать несколько предложений в одной строчке не рекомендуется, потому что код из-за этого становится труднее читать.

Python относится к языкам с неявной динамической типизацией. Неявная типизация означает, что при объявлении переменной вам не нужно указывать её тип, при явной – это делать необходимо.

К основным встроенным типам относятся:

1. None (неопределенное значение переменной)
2. Логические переменные (Boolean Type)

3. Числа: int – целое число, float – число с плавающей точкой, complex – комплексное число

4. Массивы: list – список, tuple – кортеж, range – диапазон

5. str - строки

6. dict – словарь

None – это тип значения null в Python. Если функция явно не возвращает никакого значения, то неявно она возвращает None

Основные числовые типы в Python – int и float. Размер целого числа, которое может быть представлено типом int, зависит от платформы (32- или 64-разрядной), но Python 3 автоматически преобразует слишком большие целые в тип long, способный представить сколь угодно большое целое число. Числа с плавающей точкой представляются типом Python float, который релизован в виде значения двойной точности (64-разрядного). В версии Python 3 деление целых чисел, результатом которого не является целое число, дает число с плавающей точкой. Комплексные числа записывают с j в обозначении мнимой части:

```
c = 1 + 2j
c*=(1 - 2j)
print(c)
```

Для того, чтобы объявить и сразу инициализировать переменную необходимо написать её имя, потом поставить знак ра-

венства и значение, с которым эта переменная будет создана. Например:

```
a = 1
```

объявляет переменную a и присваивает ей значение 1.

Имя переменной не должно совпадать с ключевыми словами интерпретатора Python. Список ключевых слов можно найти здесь. Также его можно получить непосредственно в программе, для этого нужно подключить модуль keyword и воспользоваться командой keyword.kwlist.

```
>>> import keyword
>>> print("Python keywords: ", keyword.kwlist)
```

Проверить является или нет идентификатор ключевым словом можно так:

```
>>> keyword.iskeyword("try")
```

Тип переменной можно определить с помощью функции **type()**.

Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, сегодня является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор.

Если в качестве операндов некоторого арифметического выражения используются только целые числа, то результат тоже будет целое число. Исключением является операция деления,

результатом которой является вещественное число. При совместном использовании целочисленных и вещественных переменных, результат будет вещественным. Список стандартных арифметических операций:

1. Сложение;
2. Вычитание;
3. Умножение;
4. Деление;
5. Получение целой части от деления; 9 // 3
6. Получение остатка от деления; 9 % 5
7. Возведение в степень 5 ** 4;

Комбинации арифметических операторов следуют обычному приоритету

Переменные содержащие текстовую информацию в python относятся к специальному типу string. При объявлении можно использовать как одинарные так и двойные кавычки. Строки – это последовательности символов и потому могут рассматриваться как любые другие последовательности, например списки или кортежи.

```
a = "hello"
b = ' '
c = "world! \n"
d = 3

print(a[0]+a[1]+a[2]+a[3]+a[4])

print ((a+b+c)*d)
```

Полезные функции для работы со строками: len() - длина строки / массива, lower() - переход в нижний регистр, upper() - переход в верхний регистр

```
string = "Some Random String"
print(len(string), string.lower(), string.upper())
```

Два булевых значения записываются в Python как True и False. Булева алгебра представлена в Python стандартными логическими "and"(&) , "or"(|) и "not". У большинства объектов в Python имеется понятие истинности или ложности. Чтобы узнать, какое булево значение соответствует объекту, нужно передать его функции bool()

```
True and True = True
True and False = False
False and False = False

True or True = True
```

```
True or False = True
False or False = False
not True = False
not False = True
```

Пример:

```
bool1=True
bool2=False
print(int(bool1), int(bool2), bool1 and bool2)
```

**Задание 0**: Вычислить результат (True или False) следующих выражений логических выражений, затем проверить с помощью python правильность результата:

```
19 % 4 != 300 / 10 / 10 and False
-(1 ** 2) < 2 ** 0 and 10 % 10 <= 20 - 10 * 2
2 ** 3 == 108 % 100 or 'Cleese' == 'King Arthur'
```

## 1.1 Контейнеры и последовательности

**Кортеж (tuple)** – это одномерная неизменяемая последовательность объектов Python фиксированной длины. Проще всего создать кортеж, записав последовательность значений через запятую:

```
tup1 = 4, 5, 6
tup2 = (1,2,3)
tup2d = ((1,2,3), (4,5,6))
```

К элементам кортежа можно обращаться с помощью квадратных скобок [], как и для большинства других типов последовательностей. Как и в C, C++, Java и многих других языках, нумерация элементов последовательностей в Python начинается с нуля. Хотя объекты, хранящиеся в кортеже, могут быть изменяемыми, сам кортеж после создания изменить (т. е. записать что-то другое в существующую позицию) невозможно:

```
tup = tuple(['foo', [1, 2], True])
tup[2] = False
```

---

```
TypeError T raceback (most recent call last)
<ipython-input-365-c7308343b841> in <module>()
----> 1 tup[2] = False
TypeError: 'tuple' object does not support item assignm
```

При попытке присвоить значение похожему на кортеж выражению, состоящему из нескольких переменных, интерпретатор пытается распаковать значение в правой части оператора присваивания:

```
tup = (4, 5, 6)
a, b, c = tup
print(b)
Out[372]: 5
```

Распаковать можно даже вложенные кортежи:

```
tup = 4, 5, (6, 7)
```

```
a, b, (c, d) = tup
print(d)
Out[375]: 7
```

Эта функциональность позволяет без труда решить задачу обмена значений переменных, которая на python решается так:

```
b, a = a, b
```

**Список (list)** В отличие от кортежей, списки имеют переменную длину, а их содержимое можно модифицировать. Список определяется с помощью квадратных скобок [] или конструктора типа list:

```
a_list = [2, 3, 7, None]
```

Для добавления элемента в конец списка служит метод append:

```
a_list.append(15)
```

Метод insert позволяет вставить элемент в указанную позицию списка:

```
a_list.insert(0,125)
```

Операцией, обратной к insert, является pop, она удаляет из списка элемент, находившийся в указанной позиции, и возвращает его:

```
a_list.pop(3)
```

Элементы можно удалять также по значению методом remove, который находит и удаляет из списка первый элемент с указанным значением:

```
a_list.remove(125)
```

Список можно отсортировать на месте (без создания нового объекта), вызвав его метод sort:

```
a = [7, 2, 5, 1, 3]
a.sort()
```

У метода sort есть несколько удобных возможностей. Одна из них – возможность передать ключ сортировки, т. е. функцию, порождающую значение, по которому должны сортироваться объекты. Например, вот как можно отсортировать коллекцию строк по длине:

```
b = ['saw', 'small', 'He', 'foxes', 'six']
b.sort(key=len)
print(b)
Out[403]: ['He', 'saw', 'six', 'small', 'foxes']
```

Из спископодобных коллекций (массивов NumPy, кортежей) можно вырезать участки с помощью нотации, которая в простейшей форме сводится к передаче пары start:stop:step оператору доступа по индексу [].Элемент с индексом start включается в срез, элемент с индексом stop не включается, поэтому количество элементов в результате равно stop – start.

```
seq = [7, 2, 3, 7, 5, 6, 0, 1]
print(seq[1:5])
Out[411]: [2, 3, 7, 5]
```

Любой член пары – как start, так и stop – можно пропустить, тогда по умолчанию подразумевается начало и конец последовательности соответственно. Если индекс в срезе отрицателен, то он отсчитывается от конца последовательности. Если задать шаг -1([::-1]), то список или кортеж будет инвертирован.

**Задание 1**: Представлен список всех методов объекта list. Применить каждый из них, оценить результат работы. Написать программу, которая переставляет первый и последний символы в строке, введенной пользователем.

```
append():Adds an element at the end of the list
clear(): Removes all the elements from the list
copy():Returns a copy of the list
count():Returns the number of elements with the speci-
fied value
index():Returns the index of the first element with
the specified value
insert():Adds an element at the specified position
pop():Removes the element at the specified position
remove():Removes the first item with the specified
svalue
reverse():Reverses the order of the list
sort():  Sorts the list
```

## 1.2 Управляющие конструкции

Как и в других языках программирования, действия можно выполнять по условию, применяя оператор if. Предложение if – одно из самых хорошо известных предложений управления потоком выполнения. Оно вычисляет условие и, если получилось True, исполняет код в следующем далее блоке. После предложения if может находиться один или несколько блоков elif и блок else, который выполняется, если все остальные условия оказались равны False

```
a=5
b=7
if(a>b):
        print('a>b')
elif(a==b):
        print('a=b')
else:
        print(a<b)
```

## 1.3 Циклы

Циклы for предназначены для обхода коллекции (например, списка или кортежа) или итератора. Стандартный синтаксис выглядит так:

```
for item in list:
```

```
    # do something with item
```

Ключевое слово continue позволяет сразу перейти к следующей итерации цикла, не доходя до конца блока.Рассмотрим следующий код, который суммирует целые числа из списка, пропуская значения None:

```
sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
    if value is None:
        continue
    total += value
```

Из цикла for можно выйти досрочно с помощью ключевого слова break. Следующий код суммирует элементы списка, пока не встретится число 5

```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total = 0
for value in sequence:

    if value == 5:
        break
    total+= value
```

Цикл while состоит из условия и блока кода, который выполняется до тех пор, пока условие не окажется равным False или не произойдет выход из цикла в результате предложения break:

```
x = 256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x = x // 2
```

## 1.4 Функции

Функции принимают ноль или несколько входящих аргументов и возвращают соответствующий результат. Определяются при помощи оператора **def**

```
def double(x):
        return x*2
def substract(a,b):
        return a-b
```

Свойства функций:

1. Функция может не заканчиваться инструкцией return, при этом функция вернет значение None.

2. Функция может принимать произвольное количество аргументов или не принимать их вовсе.

3. Функции в Python можно присваивать переменным и передавать в другие функции так же, как любые другие аргументы.

```
def double(a):
    return a * 2
def summation(a,b):
    return a+b
print(summation(double(5),10))
```

Так как функцию можно передать в другую функцию или в саму себя в качестве аргумента /использовать в теле функции, возникает возможность рекурсии: Пример:

```
def F(n):

    if n > 2:

        return F(n-1)+ F(n-2)

    else: return 1
```

**Задание 2**: Написать рекурсивную функцию, вычисляющую факториал числа.

**Задание 3**: Написать функцию, имитирующую игру камень - ножницы - бумага с пользователем. В качестве выражений использовать: rock; paper; scissors.