Комплексные типы в Python

1. **Список (list)** — упорядоченный изменяемый массив.
2. **Кортеж (tuple)** – упорядоченный неизменяемый массив.
3. **Словарь (dictionary)** – ассоциативный массив.
4. **Множество (set)** – неупорядоченный набор уникальных значений.

Списки в Python

Список (list) в Python — это гетерогенный индексный динамический массив.

Примеры списков:

[]

[1]

[1, 2, 3, 4, 5, 6, 7, 8, 12]

[1, "two", 3, 4.0, ["a", "b"], (5,6)]

Индексация списков в Python

Python обладает продвинутыми возможностями получения элемента списка по его индексу.
Прямая индексация начинается с первого элемента массива, имеющего индекс 0, обратная – с последнего, имеющего индекс -1.
Возможно задавать срезы – выделять несколько элементов массива, задавая индекс первого элемента среза и элемента, следующего за последним, между которыми ставится двоеточие.

```
>>> x = ["first", "second", "third", "fourth"]
>>> x[0]
‘first'
>>> x[2]
‘third'
>>> x[-1]
'fourth'
>>> x[-2]
'third'
>>> x[1:-1]
['second', 'third']
>>> x[0:3]
['first', 'second', 'third'] 3
>>> x[-2:-1]
['third']
>>> x[:3]
['first', 'second', 'third']
>>> x[-2:]
['third', 'fourth']
```

Методы и функции по работе со списками

Методы:

- **append(item)**: добавляет элемент item в конец списка
- **insert(index, item)**: добавляет элемент item в список по индексу index
- **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
- **clear()**: удаление всех элементов из списка
- **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
- **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
- **count(item)**: возвращает количество вхождений элемента item в список
- **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
- **reverse()**: расставляет все элементы в списке в обратном порядке

Функции:

- **len(list)**: возвращает длину списка
- **sorted(list, [key])**: возвращает отсортированный список
- **min(list)**: возвращает наименьший элемент списка
- **max(list)**: возвращает наибольший элемент списка

Кортежи в Python

Подобны спискам, но в отличие от них иммутабельны, т.е. не могут быть изменены после своего создания. Если списки задаются посредством перечисления элементов внутри квадратных скобок, то в кортежах элементы перечисляются внутри круглых скобок. Примеры:

```
()
(1,)
(1, 2, 3, 4, 5, 6, 7, 8, 12)
(1, "two", 3L, 4.0, ["a", "b"], (5, 6))
```

Кортежи и списки могут преобразовываться друг в друга с помощью функций tuple() и list().

```
>>> x = [1, 2, 3, 4]
>>> tuple(x)
(1, 2, 3, 4)
>>> x = (1, 2, 3, 4)
>>> list(x)
[1, 2, 3, 4]
```

Словари в Python

Словари представляют собой наборы переменных типа “ключ - значение”. Ключи в словаре должны быть уникальными. Примеры:

```
>>> x = {1: "one", 2: "two"}
>>> x["first"] = "one"
>>> x[(“Python", “snake", 2019)] = (1, 2, 3)
>>> list(x.keys())
['first', 2, 1, (' Python ', ' snake’, 2019)]
>>> x[1]
'one'
>>> x.get(1, "not available")
'one'
>>> x.get(4, "not available")
'not available'
```

Множества в Python

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5}
>>> 1 in x
True
>>> 4 in x
False
```

Строки

В большинстве современных языков программирования используется строковый тип. Тип String обладает рядом свойств и методов. Например, свойство Length в C# (метод length() в Java, функция ***len(str)*** в Python) позволяет определить длину строки.

Строки могут объединяться (конкатенировать), включать в себя управляющие последовательности (начинаются со знака \ (escape – символ)), включать в себя значения переменных (т.н. форматированный вывод).

Управляющие последовательности: \n, \t, \\, \', \" и т.д.

Форматированный вывод: print(f'Hello {name}')

Методы работы со строками Python

isalpha(str): возвращает True, если строка состоит только из алфавитных символов

isdigit(str): возвращает True, если все символы строки - цифры

isnumeric(str): возвращает True, если строка представляет собой число

startswith(str): возвращает True, если строка начинается с подстроки str

endswith(str): возвращает True, если строка заканчивается на подстроку str

lower(): переводит строку в нижний регистр

upper(): переводит строку в верхний регистр

title(): начальные символы всех слов в строке переводятся в верхний регистр

strip(): удаляет начальные и конечные пробелы из строки

center(width): если длина строки меньше параметра width, то слева и справа от строки равномерно добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по центру

find(str[, start [, end]): возвращает индекс подстроки в строке. Если подстрока не найдена, возвращается число -1

replace(old, new[, num]): заменяет в строке одну подстроку на другую

split([delimeter[, num]]): разбивает строку на подстроки в зависимости от разделителя

join(strs): объединяет строки в одну строку, вставляя между ними определенный разделитель

Процедурное программирование

Исторически самая первая из парадигм. Предполагает наличие последовательно выполняемых инструкций, размещаемых внутри процедур (функций), позволяющих увеличить повторное использование кода.

Имена процедур обычно представляют собой описания некоторых действий (глаголы). Входные параметры представляют собой данные необходимые для работы процедур, выходные параметры – результаты их выполнения.

Классический процедурный язык – C.
Процедурные конструкции преобладают в скриптовых ЯП (Perl, JavaScript, PHP, Python).

Функции в Python

```
def reverseThreeDigits(number):
        numBefore = int(number)
        s = str(numBefore)
        return s[2] + s[1] + s[0]


number = input("Введите число: ")
while len(number) != 3 or not number.isdigit():
        print("Это - не трехзначное число")
        number = input("Введите число: ")
new_number = reverseThreeDigits(number)
print(new_number)
```

Параметры по значению и ссылке

Параметры по значению:

```
>>> a, b = 4, 5
>>> a
4
>>> b
5
>>> def square(a,b):
        a = a * a
        b = b * b
        print(f'a = {a}')
        print(f'b = {b}')

>>> square(a,b)
a = 16
b = 25
>>> a
4
>>> b
5
```

Параметры по ссылке:

```
>>> list_of_int = [4, 5]
>>> list_of_int
[4, 5]
>>> def square(list):
        for idx, val in enumerate(list):
            list[idx] = val * val
        print(list)

>>> square(list_of_int)
[16, 25]
>>> list_of_int
[16, 25]
```

Рекурсия

Рекурсия – повторное обращение к процедуре изнутри самой процедуры. Для того, чтобы рекурсия работала необходимо определить граничные условия, при которых происходит выход из рекурсии.

Пример рекурсивной функции расчета факториала в Python:

```
def factorial(x):
        return 1 if x==0 else x * factorial(x-1)
```

Обработка исключений

Задается с помощью конструкции **try…except**.

```
try:
    number1 = int(input("Введите первое число: "))
    number2 = int(input("Введите второе число: "))
    print("Результат деления:", number1/number2)
except ValueError:
    print("Преобразование прошло неудачно")
except ZeroDivisionError:
    print("Попытка деления числа на ноль")
except Exception:
    print("Общее исключение")
print("Завершение программы")
```

Можно выбрасывать собственные исключения с помощью оператора **raise**.