Python для анализа данных и решения задач машинного обучения

Часть 1. Основы Python. Настройка среды разработки.

Занятие 2

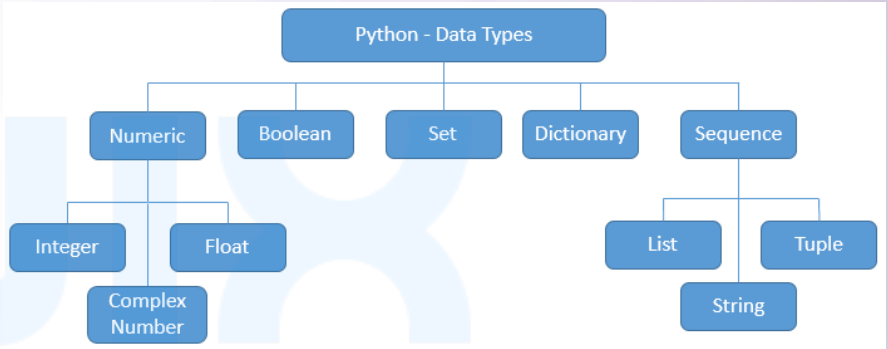
Тема: Базовые возможности, встроенные типы данных и управляющие конструкции в Python.

Цель: Получить базовые знания о синтаксисе, семантике и возможностях языка Python.

Встроенные типы данных

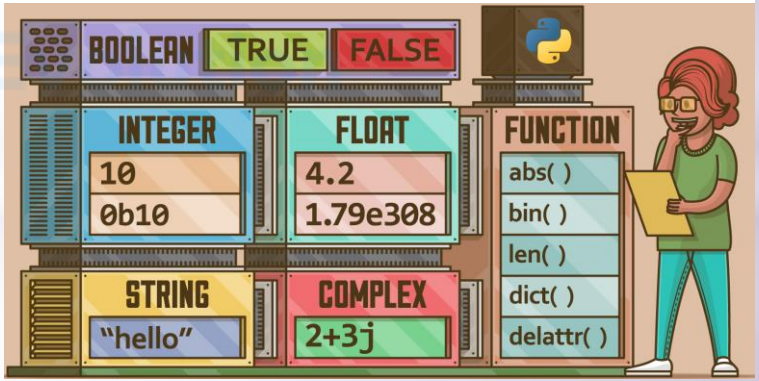
В Python есть несколько стандартных типов данных:

- Numeric (числа);
- Strings (строки);
- Lists (списки);
- Dictionaries (словари);
- Tuples (кортежи);
- Sets (множества);
- Boolean (логический тип данных);



Условная классификация данных:

- изменяемые;
- неизменяемые;
- упорядоченные;
- неупорядоченные.



Числовые и логические типы

Четыре числовых типа Python — целые числа, числа с плавающей запятой (с плавающей точкой), комплексные числа и логические значения:

- целые числа: 10, –5, 143, 355, 888888888888, –7777777777 (размер целых чисел ограничивается только объемом доступной памяти);
- числа с плавающей точкой: 3,0, 31e12, 6e – 4;
- комплексные числа: 3 + 2j, –4- 2j, 4,2 + 6,3j;
- логические значения: True, False.

```
>>> x = complex(1, 2)
>>> print(x)
(1+2j)
>>> y = complex(3, 4)
>>> print(y)
(3+4j)
>>> z = x + y
>>> print(x)
(1+2j)
>>> print(z)
(4+6j)
>>> z = x * y
>>> print(z)
(-5+10j)
>>> z = x / y
>>> print(z)
(0.44+0.08j)
```

Списки

Список в Python это:

- последовательность элементов, разделенных между собой запятой и заключенных в квадратные скобки;
- изменяемый упорядоченный тип данных.

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

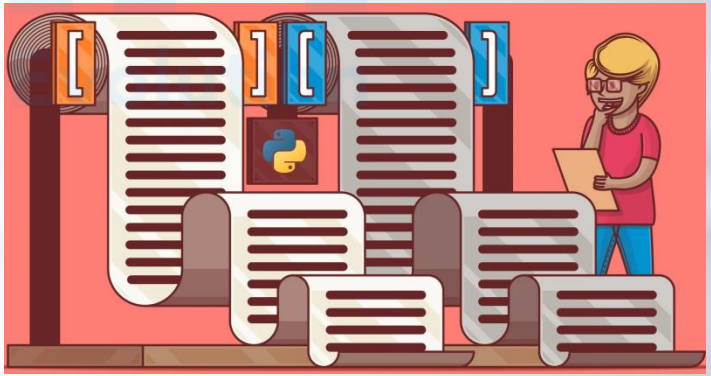
Элементами списка могут быть другие типы в произвольном сочетании: строки, кортежи, списки, словари, функции, объекты файлов и любые числовые типы. Также из списка можно выделить сегмент (срез или слайс).

Индексирование от начала списка использует положительные значения (0 соответствует первому элементу). Индексирование от конца списка использует отрицательные индексы (–1 соответствует последнему элементу).

Списки

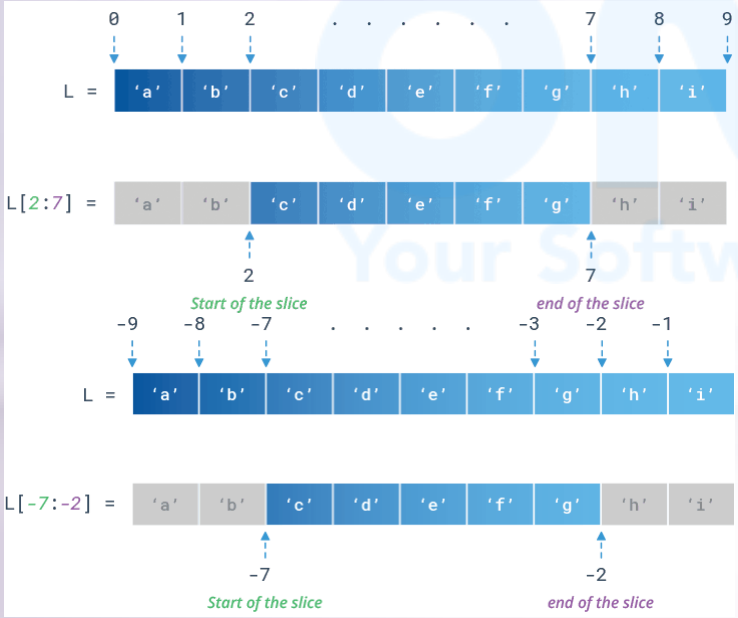
x=	[	"first",	"second",	"third",	"fourth"	]
Положительные индексы		0	1	2	3	
Отрицательные индексы		–4	–3	–2	–1	

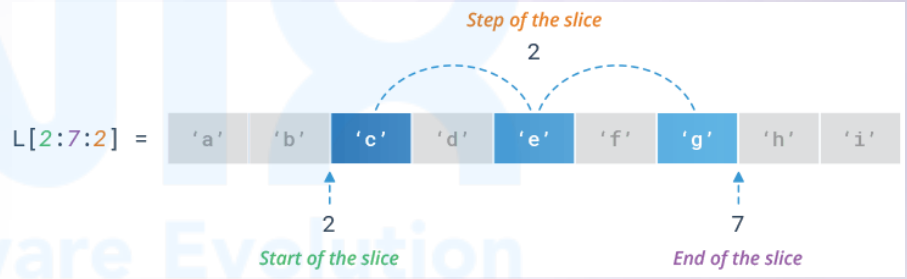
```
>>> x = ["first", "second", "third", "fourth"]
>>> x[0]
'first'
>>> x[2]
'third'
>>> x[-1]
'fourth'
>>> x[-2]
'third'
```



Списки

Сегмент создается записью вида [m:n], где m — индекс начального элемента (включительно), а n — индекс конечного элемента (не включая его). Сегмент [:n] начинается от начала списка, а сегмент [m:] продолжается до конца списка.






```
>>> x[-2:-1]
['third']
>>> x[:3]
['first', 'second', 'third']
```

Списки

Если опущены оба индекса, новый список распространяется от начала до конца исходного списка, то есть список копируется. Этот прием может пригодиться для создания копии, которую можно изменять без модификации исходного списка.

```
>>> y = x[:]
>>> y[0] = '1 st'
>>> y
['1 st', 'second', 'third', 'fourth']
>>> x
['first', 'second', 'third', 'fourth']

>>> original = [[0], 1]
>>> shallow = original[:]
>>> import copy
>>> deep = copy.deepcopy(original)
```

При этом создается «поверхностная» копия списка. Если список содержит другие вложенные списки, то необходимо делать «глубокую» копию. Для этого можно воспользоваться функцией **deepcopy** модуля **copy**. Глубокая копия не зависит от оригинала, и изменения в ней не отражаются на исходном списке.

Списки

Операция	Назначение	Пример
`[]`	Создает пустой список	`x = []`
`len`	Возвращает длину списка	`len(x)`
`append`	Добавляет один элемент в конец списка	`x.append('y')`
`extend`	Добавляет список в конец списка	`x.extend(['a', 'b'])`
`insert`	Вставляет новый элемент в произвольную позицию списка	`x.insert(0, 'y')`
`del`	Удаляет элемент или сегмент из списка	`del(x[0])`
`remove`	Находит и удаляет заданное значение из списка	`x.remove('y')`
`reverse`	Переставляет элементы списка в обратном порядке «на месте»	`x.reverse()`
`sort`	Сортирует список «на месте» (то есть, с изменением сортируемого списка)	`x.sort()`
`+`	Объединяет два списка	`x1 + x2`
`*`	Создает несколько копий из элементов списка	`x = ['y'] * 3`
`min`	Возвращает наименьший элемент списка	`min(x)`
`max`	Возвращает наибольший элемент списка	`max(x)`
`index`	Возвращает позицию значения в списке	`x.index['y']`
`count`	Подсчитывает количество вхождений значения в списке	`x.count('y')`
`sum`	Суммирует элементы списка (если они поддерживают суммирование)	`sum(x)`
`in`	Сообщает, присутствует ли элемент в списке	`'y' in x`

Кортежи

Кортеж в Python это:

- последовательность элементов, которые разделены между собой запятой и заключены в скобки;
- неизменяемый упорядоченный тип данных.

Создание кортежа практически не отличается от создания списка: переменной присваивается последовательность значений. Кортеж представляет собой последовательность, заключенную в круглые скобки (и). По наличию запятых Python «понимает», что круглые скобки обозначают кортеж, а не способ группировки.

```
>>> x = 3
>>> y = 4
>>> (x + y) # сумма
7
>>> (x + y,) # кортеж
(7,)
>>> () # пустой кортеж
()
```

Для устранения неоднозначности в случае одноэлементного кортежа Python требует, чтобы за элементом кортежа следовала запятая.

Кортежи

Для удобства Python позволяет размещать кортежи в левой части оператора присваивания. В этом случае переменным из кортежа присваиваются соответствующие значения из кортежа в правой части оператора. Простой пример:

```
>>> (one, two, three, four) = (1, 2, 3, 4)
```

Python распознает кортежи в контексте присваивания даже без круглых скобок. Значения в правой части упаковываются в кортеж, а затем распаковываются в переменные из левой части:

```
>>> one, two, three, four = 1, 2, 3, 4
```

В Python 3 появился расширенный синтаксис распаковки, который позволяет элементу с пометкой * поглотить любое количество элементов, не имеющих парного элемента:

```
>>> x = (1, 2, 3, 4)
>>> a, b, *c = x
>>> a, b, c
(1, 2, [3, 4])
```

Строки

Строка в Python это:

- последовательность символов, заключенная в кавычки;
- неизменяемый упорядоченный тип данных.

Строки могут ограничиваться одинарными (' '), двойными (" "), утроенными одинарными (''' ''') или утроенными двойными (""" """) кавычками; они могут содержать символы табуляции (\t) и символы новой строки (\n).

Строки также являются неизменяемыми. Операторы и функции, которые работают с ними, возвращают новые строки, полученные на основе оригинала. Операторы (**`in`**, + и **`*`**) и встроенные функции (**`len`**, **`max`** и **`min`**) работают со строками так же, как они работают со списками и кортежами. Синтаксис индексирования и сегментирования работает аналогичным образом для получения элементов и сегментов, но он не может использоваться для добавления, удаления или замены элементов.

Строки

Последовательность	Представляемый символ
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратный слеш
\a	Звуковой сигнал
\b	Backspace
\f	Прогон страницы
\n	Новая строка
\r	Возврат курсора (не то же, что \n)
\t	Табуляция
\v	Вертикальная табуляция

Строки

Операция	Описание	Пример
+	Объединяет две строки	x = "hello " + "world"
*	Дублирует строку	x = " " * 20
upper	Преобразует строку к верхнему регистру	x.upper()
lower	Преобразует строку к нижнему регистру	x.lower()
title	Преобразует первую букву каждого слова в строке к верхнему регистру	x.title()
find, index	Ищет заданную подстроку в строке	x.find(y) x.index(y)
rfind, rindex	Ищет заданную подстроку в строке, начиная с конца строки	x.rfind(y) x.rindex(y)
startswith, endswith	Проверяет начало или конец строки на со впадение с заданной подстрокой	x.startswith(y) x.endswith(y)
replace	Заменяет подстроку новой строкой	x.replace(y, z)
strip, rstrip, lstrip	Удаляет пропуски или другие символы в концах строки	x.strip()
encode	Преобразует строку Юникода в объект bytes	x.encode("utf_8")

Методы не изменяют саму строку. Они возвращают либо позицию в строке, либо новую строку.

Строки

Строковый оператор **%** работает с двумя частями: в левой части размещается строка, а в правой кортеж. Оператор **%** ищет в левой строке специальные форматные последовательности и строит новую строку, заменяя эти форматные последовательности значениями из правой части. В примере форматными последовательностями в левой части являются три экземпляра **%s**, которые означают, что «здесь вставляется строка».

```
>>> "%s is the %s of %s" % ("Ambrosia", "food", "the gods")
'Ambrosia is the food of the gods'
```

Следующая последовательность задает ширину поля выводимого числа равной 6, задает количество символов в дробной части равным 2 и выравнивает число по левому краю в пределах поля:

```
>>> "Pi is <%-6.2f>" % 3.14159 # Форматная последовательность %-6.2f
'Pi is <3.14  >
```

Множества

Множество – это изменяемый неупорядоченный тип данных. В множестве всегда содержатся только уникальные элементы. Множество в Python – это последовательность элементов, которые разделены между собой запятой и заключены в фигурные скобки.

Множество можно создать вызовом **set** для последовательности — например, для списка. При преобразовании последовательности в множество дубликаты исключаются. Методы **add** и **remove** используются для изменения элементов множества. Ключевое слово **in** используется для проверки принадлежности объекта к множеству.

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5}
>>> x.add(6)
>>> x
{1, 2, 3, 5, 6}
```

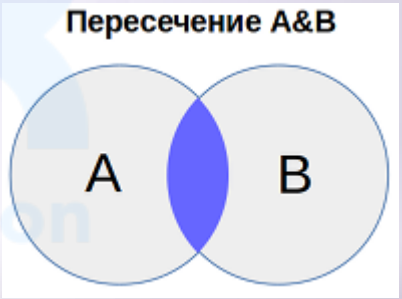
```
>>> x.remove(5)
>>> x
{1, 2, 3, 6}
>>> 1 in x
True
```

Множества

Основные операторы:

- `|` — объединение двух множеств;
- `&` — пересечение множеств;
- `^` — симметричная разность (то есть элементы, входящие только в одно из двух множеств).

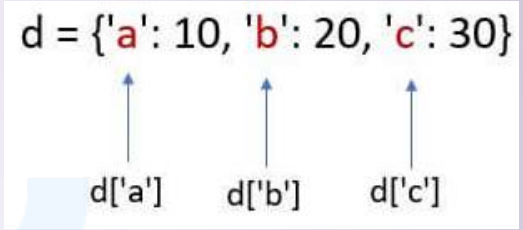
Поскольку множества не являются неизменяемыми, то они не могут быть элементами других множеств. Для решения этой проблемы в Python существует еще один тип множества **`frozenset`**, который ведет себя как множество, но не может изменяться после создания.

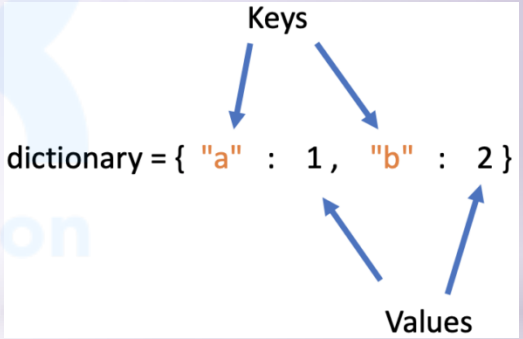


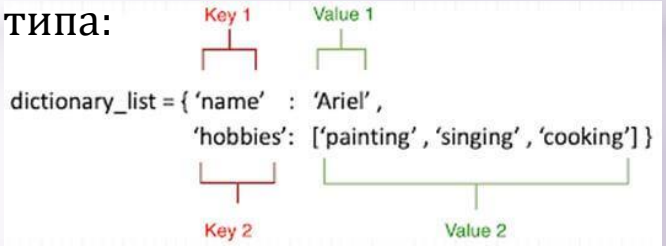
Словари

Словари — это изменяемый упорядоченный тип данных:

- данные в словаре — это пары **`ключ: значение`**;
- доступ к значениям осуществляется по ключу (индексу), а не по номеру, как в списках;
- для значений, хранящихся в словаре, неявный порядок относительно друг друга не определен (в отличие от списка), поскольку ключи не ограничиваются числами;
- словари изменяемы, то есть элементы словаря можно менять, добавлять, удалять;
- ключ должен быть объектом неизменяемого типа: число, строка, кортеж;
- значение может быть данными любого типа.







Словари

Словарь позволяет отобразить одно множество произвольных объектов на другое — логически связанное, но столь же произвольное множество объектов. Логическим аналогом словарей Python служат реальные словари или алфавитные справочники.

```
>>> english_to_french = {} ←Создает пустой словарь
>>> english_to_french['red'] = 'rouge' ←Сохраняет в нем три слова
>>> english_to_french['blue'] = 'bleu'
>>> english_to_french['green'] = 'vert'
>>> print("red is", english_to_french['red']) ←Значение для ключа 'red'
red is rouge
```

Словарь также можно явно определить как серию пар «ключ–значение», разделенных запятыми:

```
>>> english_to_french = {'red': 'rouge', 'blue': 'bleu', 'green': 'vert'}
```

Словари

Операция	Описание	Пример
`{}`	Создает пустой словарь	`x = {}`
`len`	Возвращает количество элементов в словаре	`len(x)`
`keys`	Возвращает представление, содержащее все ключи в словаре	`x.keys()`
`values`	Возвращает представление, содержащее все значения в словаре	`x.values()`
`items`	Возвращает представление, содержащее все элементы в словаре	`x.items()`
`del`	Удаляет элемент из словаря	`del(x[key])`
`in`	Проверяет присутствие ключа в словаре	`'y' in x`
`get`	Возвращает значение ключа или выбранное значение по умолчанию	`x.get('y', None)`
`setdefault`	Возвращает значение, если ключ присутствует в словаре; в противном случае ассоциирует с ключом заданное значение по умолчанию и возвращает его	`x.setdefault('y', None)`
`copy`	Создает поверхностную копию словаря	`y = x.copy()`
`update`	Проводит слияние элементов двух словарей	`x.update(z)`

Управляющие конструкции

Цикл while

```
while условие:
    тело
else:
    завершение
```

Пока условие равно **True**, то тело цикла повторяется раз за разом. Если же условие принимает значение **False**, то цикл **while** выполняет секцию завершение, а затем прекращает выполнение. Часть **else** в циклах **while** не является обязательной и применяется нечасто.

Команда if-elif-else

```
if условие1:
    тело1
elif условие2:
    тело2
elif условие3:
    тело3
...
elif условие(n-1):
    тело(n-1)
else:
    тело(n)
```

Если условие1 равно **True**, выполняется тело1; в противном случае, если условие2 равно **True**, выполняется тело2; в противном случае... и так далее, пока не будет найдено условие, равное **True**, или не будет достигнута секция **else** (тогда выполняется тело(n)). Часть тело после команды if является обязательной. Впрочем, в нее можно включить команду **pass**. Команда **pass** размещается там, где должна находиться команда, но никаких действий она не выполняет:

```
if x < 5:
    pass
else:
    x = 5
```

Управляющие конструкции

Цикл for

```
for элемент in последовательность:
    тело
else:
    завершение
```

В Python цикл **for** перебирает значения, возвращаемые любым итерируемым объектом, то есть любым объектом, который может сгенерировать последовательность значений.

Так, цикл **for** может перебрать элементы списка, кортежа или строки. Итерируемый объект также может быть специальной функцией с именем **range** или специальной разновидностью функций — так называемым **генератором**, или генераторным выражением.

Часть **else** не является обязательной. Как и часть **else** цикла **while**, она используется достаточно редко.

Две специальные команды — **break** и **continue** — также могут использоваться в теле цикла **for**. Команда **break** немедленно завершает цикл **for** даже без выполнения завершающей части (при наличии секции **else**). Команда **continue** пропускает оставшуюся часть тела цикла, и цикл продолжается со следующего элемента, как обычно.

Команда **range** с вызовом **len** для списка часто используется, чтобы сгенерировать последовательность индексов для цикла **for**.

```
x = [1, 3, -7, -1, 2]
for i in range(len(x)):
    if x[i] < 0:
        print(i)
```

Исключения

Исключения (ошибки) перехватываются и обрабатываются сложной командой **try-except-else-finally**. Эта команда также может перехватывать и обрабатывать все исключения или определенные заранее исключения. Любое неперехваченное исключение приводит к выходу из программы.

Конструкция **try** работает таким образом:

- сначала выполняются выражения, которые записаны в блоке **try**;
- если при выполнения блока **try** не возникло никаких исключений, блок **except** пропускается, и выполняется дальнейший код;
- если во время выполнения блока **try** в каком-то месте возникло исключение, оставшаяся часть блока **try** пропускается;
- если в блоке **except** указано исключение, которое возникло, выполняется код в блоке **except**;
- если исключение, которое возникло, не указано в блоке **except**, выполнение программы прерывается и выдается ошибка;
- блок **else** - выполняется в том случае, если не было исключения;
- **finally** - это опциональный блок в конструкции **try**. Он выполняется всегда, независимо от того, было ли исключение или нет (действия, которые надо выполнить в любом случае).

```
try:
    a = input("Введите число 1: ")
    b = input("Введите число 2: ")
    result = int(a)/int(b)
except (ValueError, ZeroDivisionError):
    print("Допущена ошибка в данных!")
else:
    print("Квадрат: ", result**2)
finally:
    print("Расчет окончен!")
```

Команды, блоки и отступы

Python использует отступы для определения границ **блоков** (или тел) управляющих конструкций. **Блок** состоит из одной или нескольких команд, обычно разделенных символами новой строки. Примерами команд Python могут служить команда присваивания, вызовы функций, функция **`print`**, пустая команда **`pass`** и команда **`del`**. Управляющие конструкции (**`if-elif-else`**, циклы **`while`** и **`for`**) являются составными командами.

```
секция составной команды:
    блок
секция составной команды:
    блок
```

Составная команда состоит из одной или нескольких секций, за каждой из которых следует блок с отступом. Составные команды также могут находиться в блоках, как и любые другие команды. В этом случае они создают вложенные блоки.

Также существует пара особых случаев. В одной строке можно разместить сразу несколько команд, разделив их символом «точка с запятой» (;). Блок, содержащий одну строку, может быть размещен в той же строке после двоеточия (:), завершающего секцию составной команды.

Для явного разбиения строки программы используется символ \.

```
>>> x = 1; y = 0; z = 0
>>> if x > 0: y = 1; z = 10
... else: y = -1
...
>>> print(x, y, z)
1 1 10
```

Не рекомендуется часто применять такой синтаксис.

Переменные. Присваивание. Комментарии

Термин «переменная» в Python несколько неточен; точнее было бы назвать переменную «именем» или «ярлыком». Переменные в Python – это «ярлыки», ссылающиеся на объекты из пространства имен интерпретатора Python. На один объект может ссылаться любое количество ярлыков (переменных), и при изменении этого объекта значение, на которое ссылаются все эти переменные, тоже изменяется.

```
>>> a = [1, 2, 3]
>>> b = a
>>> c = b
>>> b[1] = 5
>>> print(a, b, c)
[1, 5, 3] [1, 5, 3] [1, 5, 3]
```

Переменным Python могут присваиваться любые объекты, тогда как в C и многих других языках переменная может хранить значения только того типа, с которым она была объявлена:

```
>>> x = "Hello"
>>> print(x)
Hello
>>> x = 5
>>> print(x)
5
```

Как правило, все символы, следующие за знаком **#** в файле Python, образуют **комментарий** и игнорируются языком.

```
x = 5
x = 3  # Теперь переменная x содержит 3.
x = "# This is not a comment"
```

Преобразование типов данных

Смешанная арифметика

Python поддерживает смешанную арифметику в выражениях, состоящих из чисел разных типов. При этом целочисленный тип (**int**) при необходимости расширяется до дробного (**float**), а дробный — до комплексного (**complex**). То же самое происходит при сравнении чисел разного типа.

```
a = 3
b = complex(a)
print(b)
(3+0j)
```

Системы счисления

Для преобразования чисел в двоичную, восьмиричную и шестнадцатиричную систему служат функции **bin()**, **oct()** и **hex()**. Эти функции возвращают строковые представления чисел, что необходимо учитывать при работе с ними.

```
a = hex(38)
print(a)
0x26
```

Преобразование в строку

Для преобразования в строку используется функция **str()**. Аргументом функции **str()** может выступать число, строка, кортеж, список, множество, словарь, логическое значение, **None**.

```
a = 5.3
x = str(a)
print(x, type(a), type(x))
5.3 <class 'float'> <class 'str'>
```

Преобразование типов данных

Преобразование в список

Для преобразования в список используется функция **list()**. Аргументом функции **list()** может выступать любой итерируемый тип данных (строка, кортеж, список, множество, словарь). При преобразовании строки в список, мы получаем список, состоящий из символов строки. Стоит обратить внимание на то, что при преобразовании словаря в список, в списке оказываются только ключи.

```
b = 'Python'
x = list(b)
print(x)
c = (3, 4)
x = list(c)
print(x)
['P', 'y', 't', 'h', 'o', 'n']
[3, 4]
```

Преобразование в кортеж

Для преобразования в кортеж используется функция **tuple()**. Аргументом функции **tuple()** может выступать любой итерируемый тип данных (строка, кортеж, список, множество, словарь). Преобразование в кортеж происходит по тому же принципу, по которому происходит преобразование в список.

Преобразование в множество

Для преобразования в множество используется функция **set()**. Аргументом функции **set()** может выступать любой итерируемый тип данных (строка, кортеж, список, множество, словарь). Повторяющиеся элементы в множестве будут представлены только **один раз**.

```
a = [5, 6, 5, 6, 7]
x = set(a)
print(x)
{5, 6, 7}
```

Преобразование типов данных

Преобразование в словарь

Для преобразования в словарь используется функция **dict()**. Для преобразования в словарь каждый элемент преобразуемой последовательности должен быть парой. Первым элементом в паре может быть любой неизменяемый тип данных (число, строка, кортеж), а вторым — любой тип данных.

```
f = [[1, 'яблоко'], [2, 'тыква']]
x = dict(f)
print(x)
{1: 'яблоко', 2: 'тыква'}
```

Преобразование в логический тип

Для преобразования в логический тип используется функция **bool()**. Функция **bool()** вернет **False**, если в качестве аргумента выступает пустая строка, нулевое число, None, пустой список, пустой кортеж или пустое множество. Непустая строка, ненулевое число, даже если оно отрицательное, вернут **True**. Непустое множество, непустой список или непустой кортеж, даже если они содержат один пустой элемент, вернут **True**.

```
a = -7
x = bool(a)
print(x)
b = ''
x = bool(b)
print(x)
c = {()}
x = bool(c)
print(x)
True
False
True
```

Полезные ссылки:

https://myrobot.ru/python/index.php

https://pyneng.readthedocs.io/ru/latest/contents.html

https://devpractice.ru/python-lesson-3-data-model/

https://pythonworld.ru/tipy-dannyx-v-python/chisla-int-float-complex.html

https://metanit.com/python/tutorial/3.1.php

https://pythonru.com/uroki/python-dlja-nachinajushhih/konvertacija-tipov-dannyh