**Классы и объекты**

**Переменные (повторение)**

Переменные не нужно объявлять заранее. **У переменных нет типа. Тип есть только у данных, на которые указывают переменные**. Для опредения типа используют функцию **type()**

x = 5

print(x) # 5

print(type(x)) # <class 'int'>

x = 3.14

print(x) # 3.14

print(type(x)) # <class 'float'>

x = 'Hello'

print(x) # Hello

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

int, float, str, bool, complex - встроенные типы данных языка. Они **unmutable** (неизменяемые). (Только неизменяемые данные могут быть ключами в словаре).

**Ссылки на объекты**

Все переменные содержат только ссылки на объекты.

Оператор = связывает переменную с объектом в памяти через ссылку.

Если переменная уже существует, то справа от = напишем ссылку на объект, которая будет храниться в переменной.

Если переменной еще нет, то оператор = создает переменную и записывает в нее ссылку (которая указана справа от =).

Для упрощения рисунков дальше вместо ссылок на неизменяемые объекты будем рисовать переменные со значением.

>>> a = 3

1. Создается объект, представляющий число 3 (так как объект неизменяемый, то он создается только если его еще нет, но с логической точки зрения можете считать, что каждый раз создается новый объект).
2. Создается переменная a, если ее еще нет.
3. В переменную а записывается ссылка на объект, представляющий число 3.

Термины:

* **Переменная** - запись в системной таблице, где предусмотрено место для хранения ссылок на объекты.
* **Объект** - это область памяти с объемом, достаточным для представления значения этого объекта.
* **Ссылка** – это автоматически разыменовываемый указатель на объект (связь переменная - объект).

**id - идентификатор объекта**

Чтобы быстро отличать один объект от другого, у каждого объекта есть свой номер. У разных объектов номера разные (уникальные).

Чтобы узнать идентификационный номер объекта (идентификатор), используют встроенную функцию **id(объект)**

В примере создаем 2 разных списка с одинаковым содержимым.

Переменная a ссылается на объект с идентификатором 4292354496.

Переменная b - на другой список с идентификатором 4292357856.

>>> a = [1, 2, 3]

>>> id(a)

4292354496

>>> b = [1, 2, 3]

>>> id(b)

4292357856

**is - ссылаются на один объект**

Ключевое слово **is** проверяет, ведут ли ссылки на один и тот же объект. То есть a is b это id(a) == id(b)

>>> a = [1, 2, 3]

>>> id(a)

4292354496

>>> b = [1, 2, 3]

>>> id(b)

4292357856

>>> a is b

False

>>> a == b

True

>>> a = b # теперь a и b ссылаются на один и тот же объект (одинаковые id)

>>> id(a)

4292357856

>>> id(b)

4292357856

>>> a is b

True

**Классы**

Мы создавали и работали с объектами разных типов. Научимся создавать свои новые типы объектов.

Каждый объект - это данные и что можно делать с этими данными (функции).

В строке 'Hello' данные - это буквы H, e, l, l, o.

Что можно делать со строкой? Найти на каком месте в строке стоит буква 'e' вызвав 'Hello'.index('e'). Сделать все буквы большими 'Hello'.toupper()

Данные называют **атрибутами (переменными) объекта**, а функции - **методами объекта**.

**Описание данных (переменных) и что с ними можно делать (функций) для разных, но похожих объектов, образует класс**.

Строки 'Hello' и 'Good bye' - два разных объекта класса str.

**Класс Segment1 - отрезок на оси Х**

Опишем класс - отрезок на оси Х.

У каждого отрезка есть начало (start) и конец (finish) - это его атрибуты (переменные).

Что можно сделать с отрезком? Его можно напечатать, вычислить длину, сдвинуть по оси Х вправо или влево.

Создадим новый тип и назовем его Segment1.

class Segment1(object):

"""Класс Segment1 описывает отрезки на оси Х"""

def \_\_init\_\_(self, start=0, finish=0):

# Эта функция вызывается, когда мы создаем новый объект класса.

# self - это название переменной, которая указвает на сам объект.

self.start = start # переменная объекта

self.finish = finish

def \_\_str\_\_(self):

return '{} {}'.format(self.start, self.finish)

def \_\_repr\_\_(self):

return '[{}, {}] = {}'.format(self.start, self.finish, self.length())

def length(self):

return abs(self.start - self.finish)

def move(self, dx):

self.start += dx

self.finish += dx

# Закончились отступы - закончилось описание класса.

# Классом можно пользоваться:

s1 = Segment1(-3.5, 7) # вызывается \_\_init\_\_(-3.5, 7)

print(s1.start, s1.finish) # объект.переменная - к полям можно обратиться по имени через точку

x = s1.length() # объект.метод

print('Длина равна', x)

print(s1) # 1 7 вызывается str(s1), которая вызывает s1.\_\_str\_\_()

print(repr(s1)) # [1, 7] = 6 - вызывается repr(s1), которая вызывает s1.repr()

s1.start = 1 # объект.переменная - изменили значение поля start объекта s1

print(s1) # 1 7

s1.move(2) # передвинули отрезок на +2 по оси Х

print(s1) # 3 9

s2 = Segment1(0, 2.5) # другой объект класса Segment1, на него ссылается s2

print(s2) # 0 2.5

s2 = s1

print(s2) # 3 9 теперь s2 тоже ссылается на отрезок [3, 9], на отрезок [0, 2.5] нет ссылок.

Разберем код примера.

Создаем класс так:

class ИмяКласса(object):

описание класса

*ИмяКласса* придумаем сами. Мы придумали Segment1.

**class** - ключевое слово. Менять нельзя.

**object** - на основе какого класса делаем свой новый класс. Пока будем брать только object. Подробнее расскажем в наследовании классов.

"""Класс Segment1 описывает отрезки на оси Х"""

Это строка документации. Она описывает что делает класс. Можно автоматически из строк документации сделать help по классу, его полям и методам.

def \_\_init\_\_(self, start=0, finish=0):

# Эта функция вызывается, когда мы создаем новый объект класса.

# self - это название переменной, которая указвает на сам объект.

self.start = start # переменная объекта

self.finish = finish

Функция с именем **\_\_init\_\_** вызывается, когда мы делаем объект класса.

Чтобы сделать объект класса, пишут имя класса и в скобках указывают параметры функции \_\_init\_\_.

s1 = Segment1(-3.5, 7) # вызывается \_\_init\_\_(-3.5, 7)

**Первый аргумент любой функции объекта класса - это ссылка на сам объект**. Принято называть ее **self**.

Для обращение к переменной объекта (поля start и finish) из любого его метода нужно писать **self.start** и self.finish.

Для обращения к полям по имени объекта (ссылки s1 и s2) нужно писать **s1.start**, s1.finish, s2.start, s2.finish.

Для обращения к методу объекта изнутри объекта, нужно использовать self. Внутри метода \_\_repr\_\_ мы вызвали метод length того же объекта и обращаемся к нему **self.length()**

* Обратите внимание, первым аргументом функции length идет self.
  + def length(**self**):
* При вызове мы его НЕ указываем:
  + **s1**.length() снаружи объекта, по ссылке s1
  + **self**.length() изнутри объекта, по ссылке self
  + **s1**.move(2) снаружи объекта, по ссылке s1

**Что есть в классе и объекте?**

Все в питоне сделано на основе словарей. Их можно смотреть через **\_\_dict\_\_**

Объект - видим поля объекта.

>>> print(s1.\_\_dict\_\_) # напечатать поля (переменные) объекта

{'start': 3, 'finish': 9}

Класс - видим методы класса.

>>> print(Segment1.\_\_dict\_\_) # напечатать все методы класса

{'\_\_module\_\_': '\_\_main\_\_', '\_\_doc\_\_': 'Класс Segment1 описывает отрезки на оси Х', '\_\_init\_\_': <function Segment1.\_\_init\_\_ at 0xffd75d68>, '\_\_str\_\_': <function Segment1.\_\_str\_\_ at 0xffd75d20>, 'length': <function Segment1.length at 0xffd75cd8>, 'move': <function Segment1.move at 0xffd75c90>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'Segment1' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'Segment1' objects>}

**dir()** - какие переменные и функции доступны для объекта

>>> a = [1, 2, 3]

>>> dir(a)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

**Заключение**

* Термины:
  + Класс - описание похожих объектов. Тип в питоне.
  + Объект - 1 экземпляр класса.
  + атрибуты объекта - переменные объекта.
  + методы объекта - функции объекта.
* Синтаксис
  + class Имя(object): - создаем класс class Segment1(object):
  + s = Segment1(1, 7) - создаем объект класса, вызывается функция \_\_init\_\_ этого класса.
  + self - ссылка на себя, нужно ставить первым аргументом любой функции объекта.
    - def length(self)
    - def move(self, dx)
  + \_\_init\_\_ - конструктор класса, в этой функции определяем переменные объекта.
    - Он вызывается
  + \_\_str\_\_ - этот метод вызывается при печати объекта (преобразовании объекта к строке)
  + \_\_repr\_\_ - этот метод вызывается при вызове функции repr(объект)
* Обращение к полям и методам объекта
  + self.start - изнутри этого же объекта к полю start
  + s1.start - по ссылке s1 на объект к полю start
  + self.length() - изнутри этого же объекта к функции объекта length
  + s1.length() - по ссылке s1 на объект к функции объекта length
* Встроенные переменные и функции
  + type(s1) - тип объекта, на который ссылается переменная s1
  + id(s1) - идентификатор этого объекта
  + a is b - переменные a и b ссылаются на один и тот же объект? id(a) == id(b)
  + dir(s1) - какие переменные и функции доступны для объекта
  + s1.\_\_dict\_\_ - какие переменные содержит объект
  + Segment1.\_\_dict\_\_ - какие методы содержит класс

**Задачи**

**Segment1 normalize**

Допишите в класс Segment1 функцию normalize, которая проверяет, что start НЕ больше finish. Если start и finish перепутаны, то поменяйте их местами. Проверьте функцию.

**Segment1 flip**

Допишите в класс Segment1 функцию flip, которая отображает сам отрезок относительно точки 0. Проверьте функцию.

**Segment1 is\_crossed**

Допишите в класс Segment1 функцию is\_crossed(self, other), которая проверяет, пересекается сам отрезок (self) с другим отрезком (other). Возвращает True (пересекаются хотя бы в 1 точке) или False (не имеют общих точек). Проверьте функцию.

**Segment1 - выровнять отрезки по левому краю**

Если нужно, допишите функции в класс Segment1. Даны отрезки - по 1 на строку, целые числа через пробел. Выровнять отрезки по левому краю (по самому левому) и напечатать в том же порядке.

Input:

0 9

-1 3

-10 -7

5 11

Output:

-10 -1

-10 -6

-10 -7

-10 6

**Segment1 - найти длину самого большого отрезка**

Даны отрезки по 1 отрезку на строку. Напечатать длину самого большого (длинного) отрезка.

**Segment1 - найти самый длинный отрезок**

Даны отрезки по 1 отрезку на строку. Напечатать отрезок самой большой длины. Если таких отрезков несколько - напечатать их все.