

# Тема 4.

# ООП в Python

# Классы и объекты в языке Python

---

# Классы и объекты в языке Python

---

- Создание класса
- Синтаксис:

**class** имя\_класса[(имя\_класса\_предка)]:  
# поля и методы класса

- Пример простейшего класса:  
**class** MyClass:  
    **pass**

*Имена классов в Python  
принято писать в  
формате CamelCase*

# Классы и объекты в языке Python

---

- **Создание экземпляра**

- **Синтаксис:**

**имя\_объекта = имя\_класса( )**

- **Например,**

**my\_object = MyClass()**

# Классы в языке Python

---

- **Атрибуты класса**
- **Атрибута́ми класса** считаются не только имена переменных внутри класса, но и имена функций, определенных в классе и подклассы класса.
- **Атрибуты наследуются** всеми экземплярами класса и могут создаваться также и внутри методов класса.
- **Доступ к атрибутам класса** осуществляется через символ точки.

# Классы в языке Python

---

- **Атрибуты класса**
- Все атрибуты можно разделить на 2 группы:
  - *встроенные* атрибуты – это служебные атрибуты, предоставляемые классом object всем своим потомкам
  - *пользовательские* атрибуты
- Список атрибутов класса / объекта можно получить с помощью команды  
**dir(класс/объект).**

# Классы в языке Python

---

- **Атрибуты класса**
- ***Встроенные атрибуты :***
  - **`__new__(cls[,...])`** – конструктор, создает экземпляр (объект) класса. Сам класс передается в качестве аргумента
  - **`__init__(self[,...])`** – инициализатор. Принимает созданный объект класса из конструктора
  - **`__del__(self)`** – деструктор. Вызывается при удалении объекта сборщиком мусора

# Классы в языке Python

---

- **Атрибуты класса**
- ***Встроенные атрибуты :***
  - **`__str__(self)` – возвращает строковое представление объекта**
  - **`__hash__(self)` – возвращает хэш-сумму объекта**
  - **`__doc__` – документация класса (тип строка)**
  - **`__dict__` – словарь, в котором хранится пространство имен класса**
  - **и др.**



# Классы в языке Python

---

- Встроенные атрибуты

- Инициализатор `__init__()`

– это специальный метод, который при создании экземпляра класса автоматически создает ему атрибуты

```
def __init__(self, [список аргументов]):  
    self.имя_аргумента1=значение  
    ...  
    self.имя_аргументаN=значение
```

# Классы в языке Python

---

- Встроенные атрибуты
- Инициализатор `__init__()`

```
class SomeClass(object):  
    def __init__(self, value=42):  
        self.attr1 = value # создаем и инициализируем атрибут
```

```
obj = SomeClass()  
obj2 = SomeClass(86)  
print(obj.attr1, obj2.attr1) # 42 86
```

# Классы в языке Python

---

- Встроенные атрибуты
- Деструктор `__del__()` – вызывается автоматически перед уничтожением объекта.
  - метод реализуют, если перед уничтожением экземпляра класса нужно выполнить какие-то дополнительные действия.
  - Когда именно сборщиком мусора будет вызван деструктор неизвестно. (аналогия `finalize()` java)

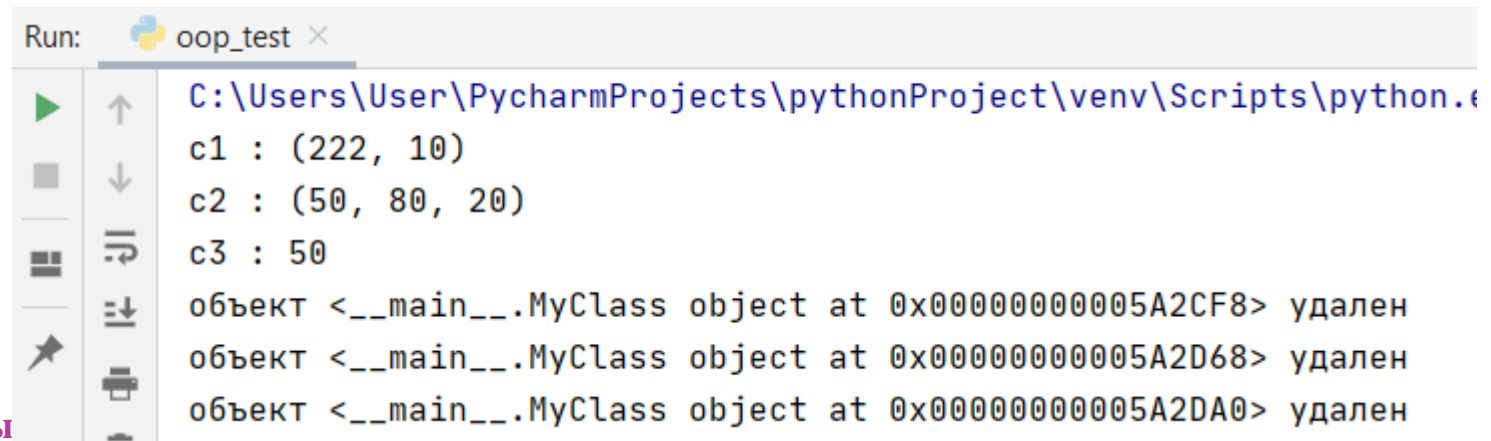
# Классы в языке Python

- Метод `__del__()`

```
class MyClass:  
    def __del__(self): # создаем деструктор  
        print(f"объект {self} удален")
```

```
MyClass.x = 50  
c1, c2, c3 = MyClass(), MyClass(), MyClass()  
c1.y, c2.y, c2.z = 10, 80, 20  
c1.x = 222  
print(F"c1 : {(c1.x, c1.y)}")  
print(F"c2 : {(c2.x, c2.y, c2.z)}")  
print(F"c3 : {c3.x}")
```

*Деструктор  
автоматически  
вызывается  
сборщиком мусора  
для всех объектов  
класса, на которые  
нет ссылок*



```
Run: oop_test x  
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe  
c1 : (222, 10)  
c2 : (50, 80, 20)  
c3 : 50  
объект <__main__.MyClass object at 0x000000000005A2CF8> удален  
объект <__main__.MyClass object at 0x000000000005A2D68> удален  
объект <__main__.MyClass object at 0x000000000005A2DA0> удален
```

# Классы в языке Python

---

- Встроенные атрибуты
- Метод `__str__()`
  - вызывается встроенной функцией `str()`, инструкцией `print` и оператором форматирования с символом формата `'s'` для получения строкового представления, наиболее пригодного для вывода

# Классы в языке Python

- Встроенные атрибуты
- Метод `__str__()`

```
class Ball:  
    def __init__(self, color):  
        self.color = color
```

```
ball = Ball('Grey')  
print(ball)
```

Run: oop\_test ×

```
C:\Users\User\PycharmProjects\pythonProject\>  
<__main__.Ball object at 0x00000000029AEE80>
```

```
class Ball:  
    def __init__(self, color):  
        self.color = color  
  
    def __str__(self) -> str:  
        return f"Ball: color {self.color}"
```

```
ball = Ball('Grey')  
print(ball)
```

Run: oop\_test ×

```
C:\Users\User\PycharmProjects\pythonProject\>  
Ball: color Grey
```

# Классы в языке Python

---

- Встроенные атрибуты

- Метод `__repr__()`

– вызывается встроенной функцией `repr()` и при использовании обратных кавычек (``expr``) (а также функцией `str()` и инструкцией `print`, если метод `__str__()` не определен.

Обычно такое представление должно выглядеть как выражение языка Python, пригодное для создания объекта с тем же значением

# Классы в языке Python

- Встроенные атрибуты: Метод `__repr__()`

```
class Ball:
    def __init__(self, color):
        self.color = color

    def __str__(self) -> str:
        return f"_str_: {self.color}"
```

```
lst = [Ball('Grey'), Ball('Blue'), Ball('Red')]
print(lst)
for b in lst: print(b)
```

Run: oop\_test x



```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/
[<__main__.Ball object at 0x0000000002A21F28>, <__main__.Ball object at
_str_: Grey
_str_: Blue
_str_: Red
```



# Классы в языке Python

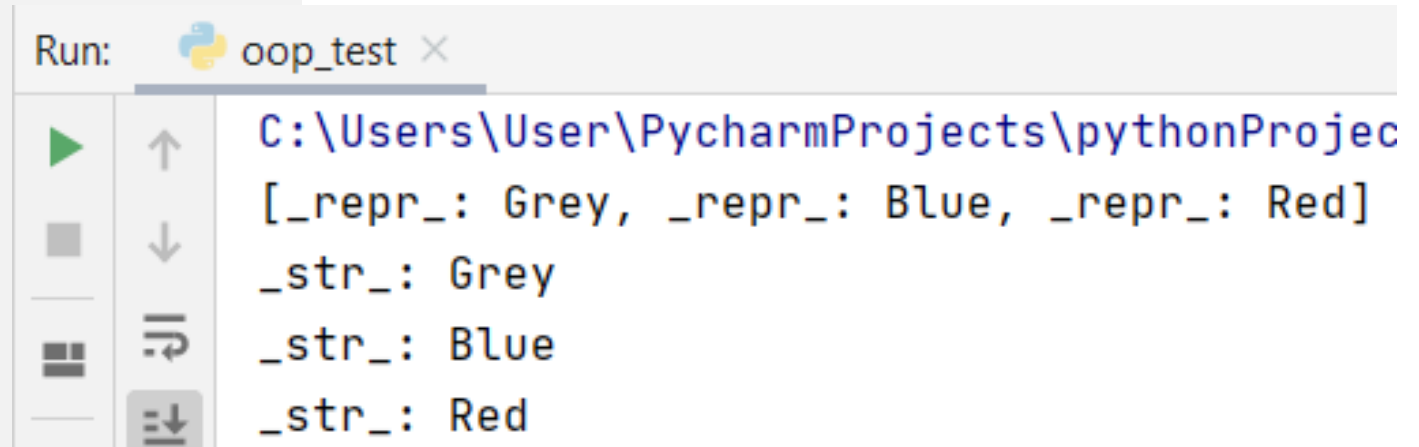
- Встроенные атрибуты: Метод `__repr__()`

```
class Ball:
    def __init__(self, color):
        self.color = color

    def __str__(self) -> str:
        return f"_str_: {self.color}«

    def __repr__(self) -> str:
        return f"_repr_: {self.color}"
```

```
lst = [Ball('Grey'), Ball('Blue'), Ball('Red')]
print(lst)
for b in lst: print(b)
```



Run: oop\_test x

```
C:\Users\User\PycharmProjects\pythonProjec
[_repr_: Grey, _repr_: Blue, _repr_: Red]
_str_: Grey
_str_: Blue
_str_: Red
```

# Классы в языке Python

---

- **Атрибуты класса**
- *Пользовательские атрибуты* – создаются программистом во время реализации класса и дальнейшей работы с ним.

```
class Phone:
```

```
    color = 'Grey'
```

```
    def turn_on(self):  
        pass
```

```
    def call(self):  
        pass
```

# Классы в языке Python

---

- Атрибуты класса
- *Пользовательские атрибуты*

**class Phone:**

**color** = 'Grey'

**def turn\_on(self):**  
    **pass**

**def call(self):**  
    **pass**

**dir(Phone)**

```
['_class_', '__delattr__', '__dict__', '__dir__',  
 '__doc__', '__eq__', '__format__', '__ge__',  
 '__getattr__', '__gt__', '__hash__', '__init__',  
 '__init_subclass__', '__le__', '__lt__', '__module__',  
 '__ne__', '__new__', '__reduce__', '__reduce_ex__',  
 '__repr__', '__setattr__', '__sizeof__', '__str__',  
 '__subclasshook__', '__weakref__', 'color', 'call',  
 'turn_on']
```

# Классы в языке Python

---

- **Поля (свойства) класса**
- **Поля (свойства, переменные) можно условно разделить на две группы:**
  - **Статические поля**
  - **Динамические поля**

# Классы в языке Python

---

- **Поля (свойства) класса**
  - **Статические поля (переменные класса)** – это переменные, которые объявляются внутри тела класса и создаются тогда, когда создается сам класс.
  - **Свойства классов устанавливаются простым присваиванием:**  
**имя\_переменной = значение**

# Классы в языке Python

---

- Поля (свойства) класса
  - Статические поля (переменные класса)

```
class Class1:  
    attr1="Hello, World"  
    attr2=4  
    print(attr1, attr2, sep="\n")
```

## Замечание:

Все выражения  
внутри инструкции  
class выполняются  
при создании  
класса, а не его  
экземпляра!

# Классы в языке Python

---

- **Поля (свойства) класса**
  - **Статические поля (переменные класса)**
  - **Доступ к статическим полям может осуществляться как без создания экземпляров (через имя класса), так и через его экземпляры**
  - **Изменение статических полей выполняется только через имя класса**

# Классы в языке Python

```
print(f"Class1 => {(Class1.attr1, Class1.attr2)}")  
# Class1 => ('Hello, World', 4)
```

```
Class1.attr1="Python"
```

```
sm = Class1()
```

```
print(f"sm => {(sm.attr1, sm.attr2)}")  
# sm => ('Python', 4)
```

```
sm.attr1="Java"
```

```
print(f"sm => {(sm.attr1, sm.attr2)}")  
# sm => ('Java', 4)
```

```
print(f"Class1 => {(Class1.attr1, Class1.attr2)}")  
# Class1 => ('Python', 4)
```



# Классы в языке Python

---

- **Поля (свойства) класса**
  - **Статические поля (переменные класса)**
  - **Доступ к статическим полям внутри методов класса также осуществляется через имя класса.**
  - **Статические поля могут быть созданы динамически, если на момент обращения к ним в программе, они еще не были определены.**

# Классы в языке Python

---

- Поля (свойства) класса
  - Статические поля (переменные класса)

```
class Class1:
```

```
    attr1="Hello, World"
```

```
    attr2=4
```

```
Class1.attr3 = 10 # создается новая переменная класса
```

# Классы в языке Python

---

- **Поля (свойства) класса**
  - **Динамические поля (переменные или свойства экземпляра класса)** – это переменные, которые создаются на уровне экземпляра класса.
  - Динамические поля могут быть созданы в методах класса: в этом случае необходимо воспользоваться конструкцией **self.имя\_переменной** внутри одно из методов класса.

# Классы в языке Python

---

- Поля (свойства) класса
  - Динамические поля (переменные или свойства экземпляра класса) – это переменные, которые создаются на уровне экземпляра класса.

```
class Class2:
```

```
    attr1 = "Hello, World" #статическое поле
```

```
    def __init__(self, dyn_attr1="FFFFFFF", dyn_attr2=20):
```

```
        self.dyn_attr1 = dyn_attr1 #динамическое поле
```

```
        self.dyn_attr2 = dyn_attr2 #динамическое поле
```

# Классы в языке Python

---

- Поля (свойства) класса
  - Динамические поля
  - Доступ к динамическим полям возможен только через экземпляр класса

```
sm = Class1()  
print(sm.dyn_attr1, sm.dyn_attr2)
```

# Классы в языке Python

---

- **Атрибуты класса**
- Атрибуты в Python допускается создавать динамически **после создания класса**:
  - можно создать как *атрибут объекта класса*,
  - так и *атрибут экземпляра класса*

# Классы в языке Python

---

- **Атрибуты класса**
- *Атрибут объекта класса* доступен всем экземплярам класса, но после изменения атрибута значение изменится во всех экземплярах класса!
- *Атрибут экземпляра класса* может хранить уникальное значение для каждого экземпляра, и изменение его в одном экземпляре класса не затронет значение одноименного атрибута в других экземплярах того же класса!

# Классы в языке Python

---

## Пример

```
class MyClass: # Определяем пустой класс  
    pass
```

```
MyClass.x = 50 # Создаем атрибут объекта класса
```

```
c1, c2 = MyClass(), MyClass() # Создаем два экземпляра класса
```

```
c1.y = 10 # Создаем атрибут экземпляра класса
```

```
c2.y = 80 # Создаем атрибут экземпляра класса
```

```
c2.z = 20 # Создаем атрибут экземпляра класса
```

```
c3 = MyClass()
```

```
print(c1.x, c1.y) # Выведет: 50 10
```

```
print(c2.x , c2.y, c2.z) # Выведет: 50 80 20
```

```
print(c3.x, c3.y) # ОШИБКА у c3 нет атрибута y
```



# Классы в языке Python

---

- **Атрибуты класса**
- **Следует учитывать, что в одном классе могут одновременно существовать атрибут объекта и атрибут экземпляра с одним именем.**
- **Изменение атрибута объекта класса доступно через имя класса, изменение атрибута экземпляра – через имя экземпляра**

# Классы в языке Python

---

- Атрибуты класса

`MyClass.x = 50` # Создаем атрибут объекта класса

`c1, c2 = MyClass() , MyClass()`

`c1.y, c2.y, c2.z = 10, 80, 20`

`c3 = MyClass()`

`MyClass.x = 555` # изменяем атрибут объекта класса

`c1.x = 222` # Создаем атрибут экземпляра c1

`print(c1.x)` # Выведет: 222

`print(c2.x)` # Выведет: 555

`print(c3.x)` # Выведет 555

# Классы в языке Python

---

- **Методы (функции) класса**
- **Методы в Python бывают трех видов:**
  - **методы экземпляра класса (они же обычные методы) (self)**
  - **статические методы (@staticmethod)**
  - **методы класса (cls)**

# Классы в языке Python

---

- **Методы экземпляра класса (обычные методы)**
- Становятся доступными только после создания экземпляра класса.
- Первым аргументом у методов экземпляра обязательно явно указывается параметр **self** (это имя ссылки на объект класса, для которого вызывается метод).

# Классы в языке Python

---

- Методы экземпляра класса (обычные методы)
- Пример

```
class Sotrudnik:
```

```
    def __init__(self, fio="unknown", staj=0):
```

```
        self.Fio = fio
```

```
        self.Staj = staj
```

```
    def __str__(self):
```

```
        return f"Сотрудник -> \n    ФИО:\t{self.Fio}\n    Стаж:\t{self.Staj}"
```

# Классы в языке Python

- Методы экземпляра класса (обычные методы)

- Пример

```
class Sotrudnik:
```

```
...
```

```
    def check_staj(self):
```

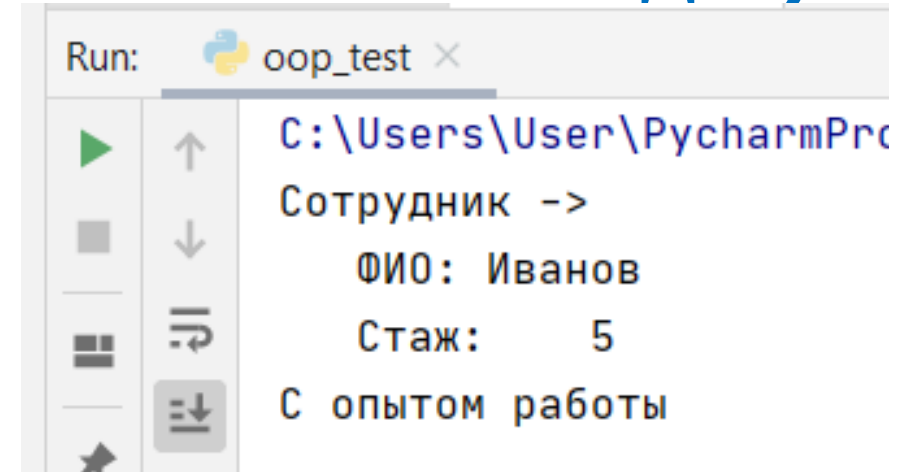
```
        print(self)
```

```
        print("С опытом работы") if self.Staj >= 3 \
```

```
            else print ("Без опыта работы")
```

```
sotr = Sotrudnik("Иванов", 5)
```

```
sotr.check_staj()
```



# Классы в языке Python

---

- **Статические методы**
- **Статические методы – это обычные функции, которые помещены в класс для удобства и тем самым располагаются в области видимости этого класса.**
- **Статические методы можно вызывать как через имя класса (без создания объекта), так и через имя экземпляра**

# Классы в языке Python

---

- **Статические методы**
- При определении статического метода параметр `self` не используется!
- Чтобы создать статический метод в Python, необходимо воспользоваться специальным декоратором - `@staticmethod`.



# Классы в языке Python

---

- **Статические методы**

```
class Sotrudnik:
```

```
    sotr_count = 0
```

```
    def __init__(self, fio="unknown", staj=0):
```

```
        self.Fio = fio
```

```
        self.Staj = staj
```

```
        Sotrudnik.sotr_count += 1
```

```
    @staticmethod
```

```
    def info():
```

```
        print(f"Всего сотрудников: {Sotrudnik.sotr_count}")
```

# Классы в языке Python

---

- **Методы класса**
- **Методы класса связываются с классом, в котором определены**
- **Методы класса могут менять состояние самого класса, что в свою очередь отражается на ВСЕХ экземплярах данного класса.**
- **Не могут менять отдельные объекта класса**

# Классы в языке Python

---

- **Методы класса**
- Для создания методов класса используется декоратор **@classmethod**.
- При этом в качестве первого параметра передается служебное слово **cls** (ссылка на сам класс, а не объект).

# Классы в языке Python

---

- Методы класса

```
class Sotrudnik:
```

```
    def __init__(self, fio="unknown", staj=0, post="служащий"):
        self.Fio, self.Staj, self.Post = fio, staj, post
```

```
    def __str__(self) -> str:
        return f"ФИО:\t{self.Fio}\nСтаж:\t{self.Staj}\nДолжность:\t{self.Post}"
```

```
    @classmethod
```

```
    def head_sotrudnik(cls, fio, staj):
        head_sotrudnik = cls(fio, staj, "Начальник отдела")
        return head_sotrudnik
```

```
sotr3 = Sotrudnik.head_sotrudnik("Сидоров", 15)
print(type(sotr3),sotr3, sep="\n")
```

# Классы в языке Python

- **Методы класса**

**class Sotrudnik:**

**def \_\_init\_\_**

**self.Fio,**

**def \_\_str\_\_**

**return f'**

**@classmethod**

**def head\_s**

**head\_s**

**return h**

Run:

oop\_test x

C:\Users\User\PycharmProjects\pythonPro

<class '\_\_main\_\_.Sotrudnik'>

Сотрудник ->

ФИО: Сидоров

Стаж: 15

Должность: Начальник отдела

.Post}"

**sotr3 = Sotrudnik.head\_sotrudnik("Сидоров", 15)**

**print(type(sotr3),sotr3, sep="\n")**

# Классы в языке Python

---

- **Методы класса**
- **Метод класса принимает ссылку на класс через параметр `cls`.**
- **Методы класса часто используют, когда необходимо создать специфичный объект текущего класса или объекты унаследованных классов прямо внутри метода.**

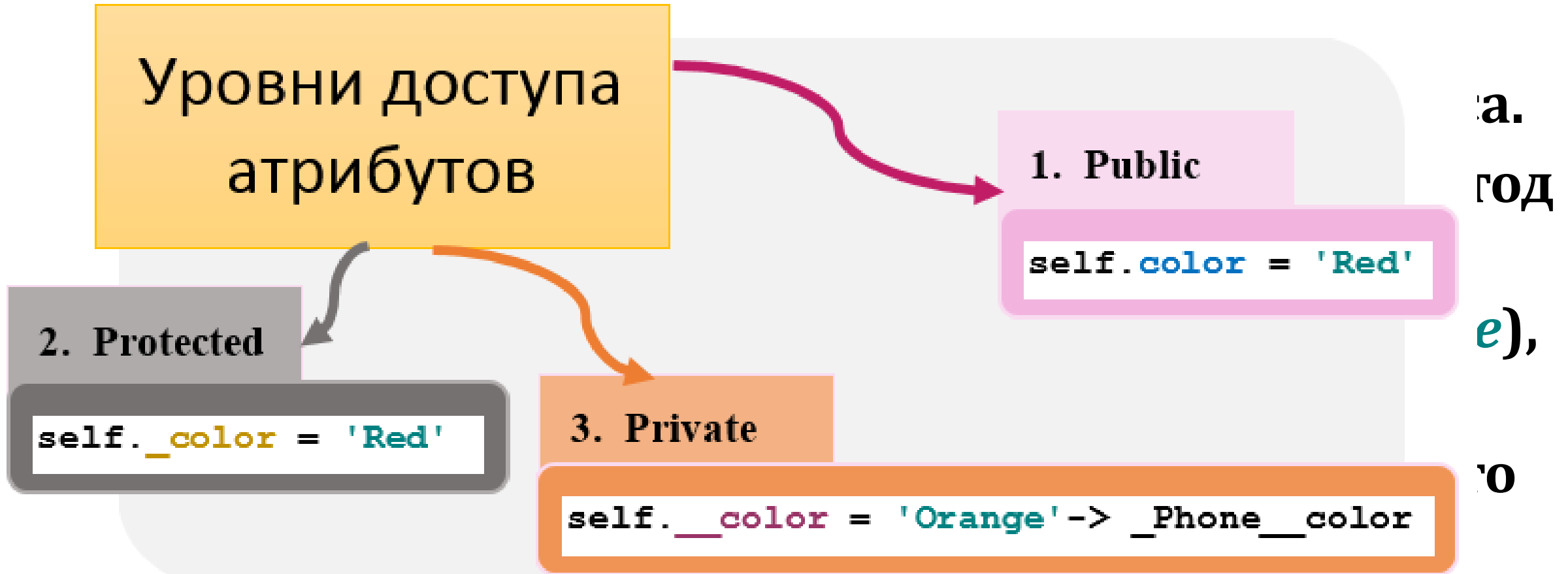
# Классы в языке Python

---

- **Уровни доступа атрибутов в Python**
- В Python отсутствует механизм, который мог бы запретить доступ к переменной/методу внутри класса.
- Соглашение создателей Python: если переменная/метод начинается :
  - с одного нижнего подчеркивания (*`_protected_example`*), то она/он считается защищенным (protected)
  - с двух нижних подчеркиваний (*`_private_example`*) – то private

# Классы в языке Python

- Уровни доступа атрибутов в Python





# Классы в языке Python

---

- **Уровни доступа атрибутов в Python**
- *Публичные атрибуты*
  - **Все члены класса в Python являются публичными по умолчанию.**
  - **Любой член класса может быть доступен за пределами самого класса.**

# Классы в языке Python

---

- Уровни доступа атрибутов в Python
- *Публичные атрибуты*

```
class Ball:
```

```
    def __init__(self, color):
```

```
        # Объявляем публичное поле color
```

```
        self.color = color
```

```
ball = Ball('Grey')
```

```
print(ball.color) # Grey
```

```
ball.color = 'Red'
```

```
print(ball.color) # Red
```

# Классы в языке Python

---

- Уровни доступа атрибутов в Python
- *Защищенные атрибуты*
  - Имя начинается с одного подчеркивания
  - Доступ извне класса все равно возможен!

**class** Ball:

**def** **\_\_init\_\_**(**self**, color):

*# Объявляем защищенное поле \_color*

**self**.\_color = color

```
ball = Ball('Grey')
print(ball._color) # Grey
ball._color = 'Red'
print(ball._color) # Red
```

# Классы в языке Python

---

- **Уровни доступа атрибутов в Python**
- *Частные (приватные) атрибуты*
  - **Имя начинается с двух подчеркиваний**
  - **Доступ напрямую получить нельзя, НО ...**

**Все имена, начинающиеся с двух символов подчеркиваний, преобразуются по правилу:**

**`__ИмяКласса__имя_атрибута`**

# Классы в языке Python

---

- Уровни доступа атрибутов в Python
- Частные (приватные) атрибуты

```
class Ball:
```

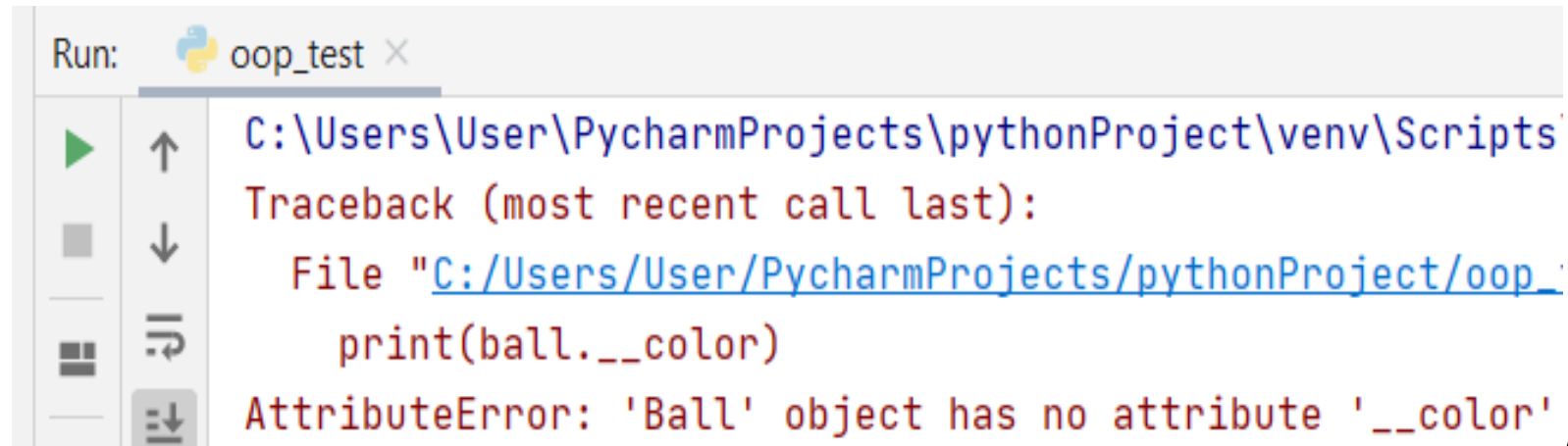
```
    def __init__(self, color):
```

```
        # Объявляем приватное поле color
```

```
        self.__color = color
```

```
ball = Ball('Grey')
```

```
print(ball.__color)
```

A screenshot of a Python IDE's 'Run' window. The window title is 'Run: oop\_test x'. It shows a traceback for an 'AttributeError: 'Ball' object has no attribute '\_\_color''. The traceback indicates the error occurred in a file named 'C:/Users/User/PycharmProjects/pythonProject/venv/Scripts' at the line 'print(ball.\_\_color)'. The IDE interface includes a green play button, a square stop button, and a magnifying glass search icon on the left side of the console.

```
Run: oop_test x
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts
Traceback (most recent call last):
  File "C:/Users/User/PycharmProjects/pythonProject/venv/Scripts
    print(ball.__color)
AttributeError: 'Ball' object has no attribute '__color'
```

# Классы в языке Python

---

- Уровни доступа атрибутов в Python
- Частные (приватные) атрибуты

```
class Ball:
```

```
    def __init__(self, color):
```

```
        # Объявляем приватное поле color
```

```
        self.__color = color
```

```
ball = Ball('Grey')
```

```
print(ball.__Ball__color)    # Grey
```

```
ball.__Ball__color = 'Red'
```

```
print(ball.__Ball__color)    # Red
```

# Классы в языке Python

---

- **Некоторые встроенные атрибуты класса**
- **Для доступа к атрибутам и методам класса можно использовать следующие функции:**

**getattr(<Объект>, <Атрибут>[, <значение по умолчанию>])**

**– возвращает значение атрибута по его названию, заданному в виде строки. С помощью данной функции можно сформировать имя атрибута динамически во время выполнения программы! Если атрибут не найдет – AttributeError!**

# Классы в языке Python

---

`getattr(<Объект>, <Атрибут>[, <значение по умолчанию>])`

```
class MyClass:
```

```
    def __init__(self):
```

```
        self.x = 10
```

```
    def get_x (self) :
```

```
        return self.x
```

```
c = MyClass () #Создаем экземпляр
```

```
print(getattr(c, "x")) #Выведет: 10
```

```
print(getattr(c, "get_x") ()) #Выведет: 10
```

```
print(getattr(c, "y", 0)) #Выведет: 0, т. к. атрибут не найден
```



# Классы в языке Python

---

- **Некоторые встроенные атрибуты класса**
- **Для доступа к атрибутам и методам класса можно использовать следующие функции:**

**`setattr(<Объект>, <Атрибут>, <Значение>)`**

**– задает значение атрибута. Название атрибута указывается в виде строки. Если атрибут не существует, он будет создан.**

# Классы в языке Python

---

- **Некоторые встроенные атрибуты класса**
- **Для доступа к атрибутам и методам класса можно использовать следующие функции:**

**`delattr(<Объект>, <Атрибут>)`**

**– удаляет указанный атрибут. Название атрибута указывается в виде строки. Если атрибут не существует – исключение `AttributeError`.**

# Классы в языке Python

---

- **Некоторые встроенные атрибуты класса**
- **Для доступа к атрибутам и методам класса можно использовать следующие функции:**

**hasattr(<Объект>, <Атрибут>)**

**– проверяет наличие указанного атрибута**

# Классы в языке Python

---

## Пример

```
c = MyClass () #Создаем экземпляр
setattr(c, "x", 200) #Записываем 200 в x
print (getattr (c, "x")) #Выведет: 10
setattr(c, "y", 20) #Создаем атрибут y
print(getattr(c, "y", 0)) #Выведет: 20
delattr ( c, "y") #Удаляем атрибут y
print(getattr(c, "y", 0)) #Выведет: 0, т. к. атрибут не найден
print(hasattr(c, "x")) #Выведет: True
print(hasattr(c, "y")) #Выведет: False
```

```
class MyClass:
    def __init__(self):
        self.x = 10
    def get_x (self) :
        return self.x
```

# Перегрузка операторов

---

# Перегрузка операторов

---

- **Перегрузка операторов в Python – это возможность с помощью специальных методов в классах переопределять различные операторы языка.**
- **Имена таких методов включают два знака подчеркивания в начале и в конце.**
- **К таким методам относятся: `__init__()`, `__del__()`, и т.п., а также метода, отвечающие за арифметические операторы, операторы сравнения и др.**

# Перегрузка операторов

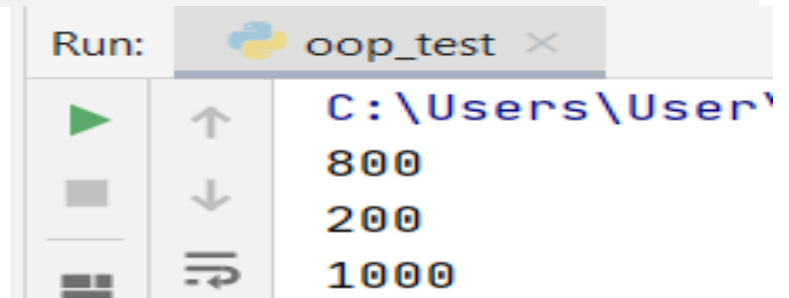
- **Специальные методы**
- Метод **`__add__(self, other)`** – метод вызывается при выполнении оператора «+» над экземплярами класса.

```
class Rectangle:
    def __init__(self, width=0, height=0):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def __add__(self, other):
        return self.area()+other.area()
```

```
rect = Rectangle(20, 40)
print(rect.area())
rect2 = Rectangle(20, 10)
print(rect2.area())
print( rect + rect2)
```

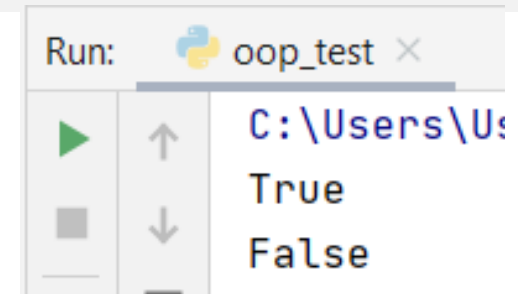


# Перегрузка операторов

- Специальные методы
- Метод `__eq__(self, other)` – метод вызывается при проверке равенства двух экземпляров класса.

```
class Rectangle:  
    def __init__(self, width=0, height=0):  
        self.width = width  
        self.height = height  
  
    def area(self):  
        return self.width * self.height  
  
    def __eq__(self, other):  
        return self.area() == other.area()
```

```
r=Rect(30, 40)  
r2=Rect(60, 20)  
r3=Rect(25, 18)  
print(r == r2)  
print(r == r3)
```





# Перегрузка операторов

---

- Специальные методы

<code>__eq__(self, other)</code>	<code>self == other</code>
<code>__ne__(self, other)</code>	<code>self != other</code>
<code>__lt__(self, other)</code>	<code>self &lt; other</code>
<code>__gt__(self, other)</code>	<code>self &gt; other</code>
<code>__le__(self, other)</code>	<code>self &lt;= other</code>
<code>__ge__(self, other)</code>	<code>self &gt;= other</code>

# Перегрузка операторов

---

- Специальные методы

<code>__add__(self, other)</code>	<code>self + other</code>
<code>__sub__(self, other)</code>	<code>self — other</code>
<code>__mul__(self, other)</code>	<code>self * other</code>
<code>__floordiv__(self, other)</code>	<code>self // other</code>
<code>__truediv__(self, other)</code>	<code>self / other</code>
<code>__mod__(self, other)</code>	<code>self % other</code>
<code>__pow__(self, other)</code>	<code>self ** other</code>