

2. ООП Python

Вопрос 218. Какая последовательность реализации программы в парадигме ООП?

Ответ: разделить программу на фрагменты, описать предметы в виде объектов, организовать связи между объектами

Вопрос 219. Что такое класс?

Ответ: Класс - это составной тип данных, включающий набор атрибутов (свойств) класса, локальных атрибутов экземпляра, методов класса, внутри которого реализуется определенный алгоритм.

LiveCoding: напишите класс

Вопрос 220. Какие бывают атрибуты в классе и где они прописываются?

Ответ: атрибуты КЛАССА доступны всем экземплярам класса – прописываются в объемлющей области внутри класса, атрибуты ЭКЗЕМПЛЯРА доступны только для экземпляра, они хранят уникальное значение для экземпляра. Атрибуты экземпляра также называют локальным атрибутом класса, они прописываются в инициализаторе `__init__`

LiveCoding: напишите класс и все атрибуты в нем.

Вопрос 221. Что такое метод класса?

Ответ: Метод класса - функции класса.

LiveCoding: напишите класс и объявите в нем методы

Вопрос 222. Что такое интерфейс?

Ответ: Понятие интерфейса очень относительно в Python. В более ранних языках программирования (C++) интерфейс – это абстрактный класс, который не имел никакого функционала, а лишь описывал какие атрибуты и методы будут в дочерних полнофункциональных классах. В Python можно реализовать интерфейс, но надобности в этом нет, в отличие от того же C++.

Вопрос 223. Что такое объект (экземпляр) класса?

Ответ: Объект класса в Python представляет собой ИМЯ класса, созданного (определенного/записанного) в коде. Объекты класса поддерживают два вида операций: ссылки на атрибуты и создание экземпляров.

LiveCoding: напишите класс и создайте его экземпляр

Вопрос 224. Что будут, если в экземпляре класса не существует атрибут?

Ответ: Если в экземпляре атрибут не существует, то поиск переходит во внешнее пространство класса, от которого он создан

LiveCoding: напишите класс, создайте экземпляр, вызовите несуществующее свойство экземпляра, но существующее у самого класса

Вопрос 225. Что такое SOLID?

Ответ: SOLID - это пять принципов по первым буквам, которых следует придерживаться при разработке программ в парадигме объектно-ориентированного программирования.

Вопрос 226. Для чего нужно соблюдать SOLID?

Ответ: Они позволяют правильно грамотно построить архитектуру программы, чтобы в будущем мы могли ее легко расширять, модифицировать, она была бы модульной и легко воспринимаемой (читаемой). SOLID применяется для проектов, где много классов. SOLID - это рекомендации.

Вопрос 227. Охарактеризуйте каждый из принципов SOLID?

Ответ:

Single Responsibility Principle (принцип единственной ответственности)

Каждый класс должен выполнять строго обозначенную функцию и быть ограниченным своей задачей. Не надо создавать классы, которые делают все сразу.

Open-Closed Principle (принцип открытости-закрытости)

Классы должны быть закрыты для модификации, но открыты для расширения. Нередко применяется абстрактные классы и полиморфизм.

Liskov Substitution Principle (принцип подстановки Барбары Лисков)

Если у нас есть какой-нибудь дочерний класс, то он должен полностью повторять функционал родительского класса.

Interface Segregation Principle (принцип разделения интерфейса)

Нужно создавать узко специализированные интерфейсы и реализовывать их в специализированных дочерних классах. Общие интерфейсы создавать не надо.

Dependency Inversion Principle (принцип инверсии зависимостей)

Классы должны зависеть от интерфейсов или от абстрактных классов, а не от конкретных классов и функций

Вопрос 228. Что такое инкапсуляция?

Ответ: Инкапсуляция - это сокрытие внутренней структуры объекта от внешнего воздействия, а также объединение в одном классе атрибутов и методов, которые определяют внутренний

алгоритм функционирования данного класса. Благодаря инкапсуляции класс становится единым целым. Работа с ним возможна только через разрешенные (публичные) методы и свойства.

Вопрос 229. Что такое наследование?

Ответ: это свойство строить сложные иерархии, предполагающее наличие родительских и дочерних классов, причем дочерние классы могут использовать функционал (атрибуты и методы) родительского класса и даже расширять их.

Вопрос 230. Что такое полиморфизм?

Ответ: это возможность через единый интерфейс работать с объектами разных классов. В Python используется параметрический полиморфизм - т.е. разными типами объектов мы можем оперировать через их единый базовый (родительский) класс. Т.е. в родительском классе можно вызвать метод, который будет вызывать переопределенные методы соответствующего дочернего класса. Тем самым мы имеем единый интерфейс.

Вопрос 231. После объявления класса ставятся скобки?

Ответ: если класс не является дочерним - скобки не ставятся

Вопрос 232. как можно посмотреть содержимое класса?

Ответ: с помощью метода `__dict__`

LiveCoding: создайте класс, опишите методы, создайте экземпляр, посмотрите содержимое класса и экземпляра

Вопрос 233. Как определить к какому классу относятся объекты, которые мы создали?

Ответ: мы можем проверить с помощью функции `type`

LiveCoding: создайте класс, опишите методы, создайте экземпляр, определите к какому классу он относится.

Вопрос 234. Как проверить к относится ли объект к данному классу или нет?

Ответ: Применением функции `isinstance(obj, class)`

Вопрос 235. Как в классе создать новый атрибут?

Ответ: `ClassName.new_attribute = 2000`

Вопрос 236. Какую функцию можно использовать для добавления нового свойства в класс?

Ответ: Нужно использовать функцию `setattr(название класса, название атрибута в виде строки, значение атрибута)`. Изменить значение этого атрибута можно точно также.

LiveCoding: добавьте новое свойство в класс

Вопрос 237. Как удалить атрибут из класса?

Ответ: Надо воспользоваться функцией `del`,

Например,

```
`del Cat.weight`
```

Или воспользоваться функцией `**delattr**`(класса, свойство класса в виде строки)

```
`delattr(Cat, 'weight')`
```

Вопрос 238. Как проверить, есть ли атрибут в классе?

Ответ: Надо воспользоваться функцией `**hasattr**`(класса, свойство класса в виде строки)

Например,

```
`hasattr(Cat, 'color')`
```

```
`True`
```

Вопрос 239. Как происходит поиск в атрибуте объекта класса?

Ответ: Вначале он ищется в текущем пространстве имен объекта класса. Если его там нет он берется из атрибутов класса

Вопрос 240. Что такое `self`?

Ответ: В ошибке написано, что метод требует 0 позиционных аргументов, а был передан один.

По факту при вызове метода у объекта класса всегда передается один аргумент – `self`. `self` - это ссылка на экземпляр класса

Вопрос 241. Для чего нужно у метода класса указывать параметр `self`?

Ответ: Для того, чтобы вызывать этот метод у объекта класса. Объект при вызове автоматически будет передавать параметр `self`, как принадлежность к этому классу.

Вопрос 242. Что такое `__init__`?

Ответ: `__init__` (`self`) - инициализатор объекта класса, вызывается сразу после создания экземпляра класса

Вопрос 243. `__init__` вызывается самый первый при создании экземпляра класса?

Ответ: нет. Последовательность такая:

1. Перед созданием объекта (экземпляра) класса вызывается магический метод `__new__`

2. Далее происходит создание объекта (экземпляра) в памяти устройства
 3. Далее автоматически вызывается следующий магический метод `__init__`
 4. `__init__` создает локальные свойства, которые в нем прописаны через запятую после `self`
-

Вопрос 244. Что такое `__del__` и прописывается ли он?

Ответ: `__del__` (`self`) - финализатор класса, который вызывается перед удалением объекта. В реальной программе мы не прописываем метод `__del__`, так как он автоматически вызывается с помощью сборщика мусора Python.

Вопрос 245. Когда и в какой момент происходит удаление объектов?

Ответ: Удаление объектов сборщиком мусора (специальный алгоритм Python) происходит в тот момент, когда объекты становятся ненужными - он становится ненужным, когда на объект не ведет ни одна ссылка

Вопрос 246. Что значит удаление объекта?

Ответ: Это значит, что освобождается память, которую этот объект занимал.

Вопрос 247. Когда вызывается метод `__new__`?

Ответ: Метод `__new__` вызывается перед созданием объекта класса

Вопрос 248. Зачем нужен метод `__new__`?

Ответ: Он возвращает ссылку на объект класса.

Вопрос 249. Зачем было создавать 2 разных метода `new` и `init`, которые вызываются при создании объектов класса?

Ответ: Метод `new` можно переопределить и тем самым в программе можно создавать ограниченное число экземпляров класса. Так метод `__new__` вызывается автоматически

Вопрос 250. Чем отличается `cls` от `self`?

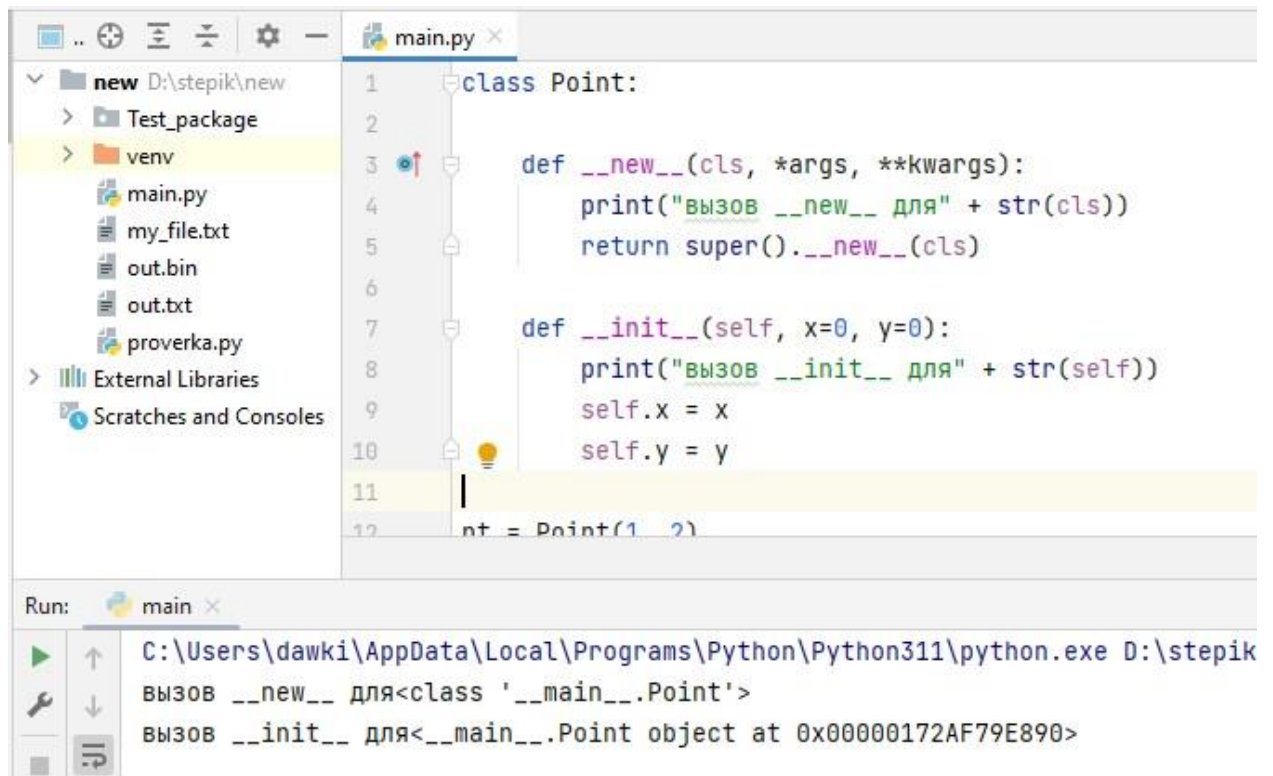
Ответ: `cls` - ссылка на класс, для которого создается объект. `self` - ссылка на создаваемый экземпляр класса.

Вопрос 251. При вызове метода `__new__` используется `args` и `kwargs`. Для чего?

Ответ: В них помещаются аргументы при создании экземпляра класса

Вопрос 252. Напишите метод `__new__` для своего класса?

Ответ:



```
1 class Point:
2
3     def __new__(cls, *args, **kwargs):
4         print("Вызов __new__ для" + str(cls))
5         return super().__new__(cls)
6
7     def __init__(self, x=0, y=0):
8         print("Вызов __init__ для" + str(self))
9         self.x = x
10        self.y = y
11
12    pt = Point(1, 2)
```

Run: main

```
C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepik
Вызов __new__ для<class '__main__.Point'>
Вызов __init__ для<__main__.Point object at 0x00000172AF79E890>
```

Вопрос 253. Что такое паттерн Singleton?

Ответ: В обычной программе мы можем создавать сколько угодно экземпляров класса. В паттерне Singleton возможно создание только одного экземпляра класса. Теперь у нас все экземпляры будут ссылаться на один и тот же объект в памяти, а именно на последний созданный экземпляр - db2. Чтобы ссылаться на первый вызывается метод `__call__`

LiveCoding: реализуйте паттерн Singleton

```
main.py x
1 class DataBase:
2     __instance = None # Ссылка на экземпляр класса
3
4     def __new__(cls, *args, **kwargs):
5         if cls.__instance is None:
6             cls.__instance = super().__new__(cls)
7         return cls.__instance
8
9     def __del__(self):
10        DataBase.__instance = None
11
12    def __init__(self, user, psw, port):
13        self.user = user
14        self.psw = psw
15        self.port = port
16
17    def connection(self):
18        print(f"соединение с БД: {self.user}, {self.psw}, {self.port}")
19
20    def close(self):
21        print("заккрытие соединения с БД")
22
23    def read(self):
24        return "Данные из БД"
25
26    def write(self, data):
27        print(f'запись в БД {data}')
28
29    db = DataBase('root', '1234', 89)
30    db2 = DataBase('superrrot', '652', 40)
31    print(id(db) == id(db2))
```

Вопрос 254. Что такое методы класса и как они обозначаются?

Ответ: Допустим нам надо работать с атрибутами класса, а не с локальными атрибутами экземпляра класса. Для работы с атрибутами всего класса используется декоратор `@classmethod`

LiveCoding: реализуйте метод класса

The screenshot shows a Python IDE with a file named `main.py`. The code defines a class `Cat` with a class attribute `cute = True` and a class method `validator` that checks if the `is_cute` parameter is "yes". The `__init__` method sets `weight` and `age`. The main code creates a `Cat` object and calls the `validator` method with the argument "none". The output shows the object creation and the result `False`.

```
1 class Cat:
2     cute = True
3
4     @classmethod
5     def validator(cls, is_cute="yes"):
6         if is_cute == "yes":
7             return cls.cute
8         return False
9
10    def __init__(self, weight, age):
11        self.weight = weight
12        self.age = age
13
14
15    print(Cat(10, 3))
16    print(Cat.validator("none"))
17
```

Output:

```
C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.
<__main__.Cat object at 0x000001D87120EBD0>
False
```

Вопрос 255. Какая особенность у метода класса `@classmethod`?

Ответ: `@classmethod` относится только к атрибутам класса, но не к локальным атрибутам экземпляра класса.

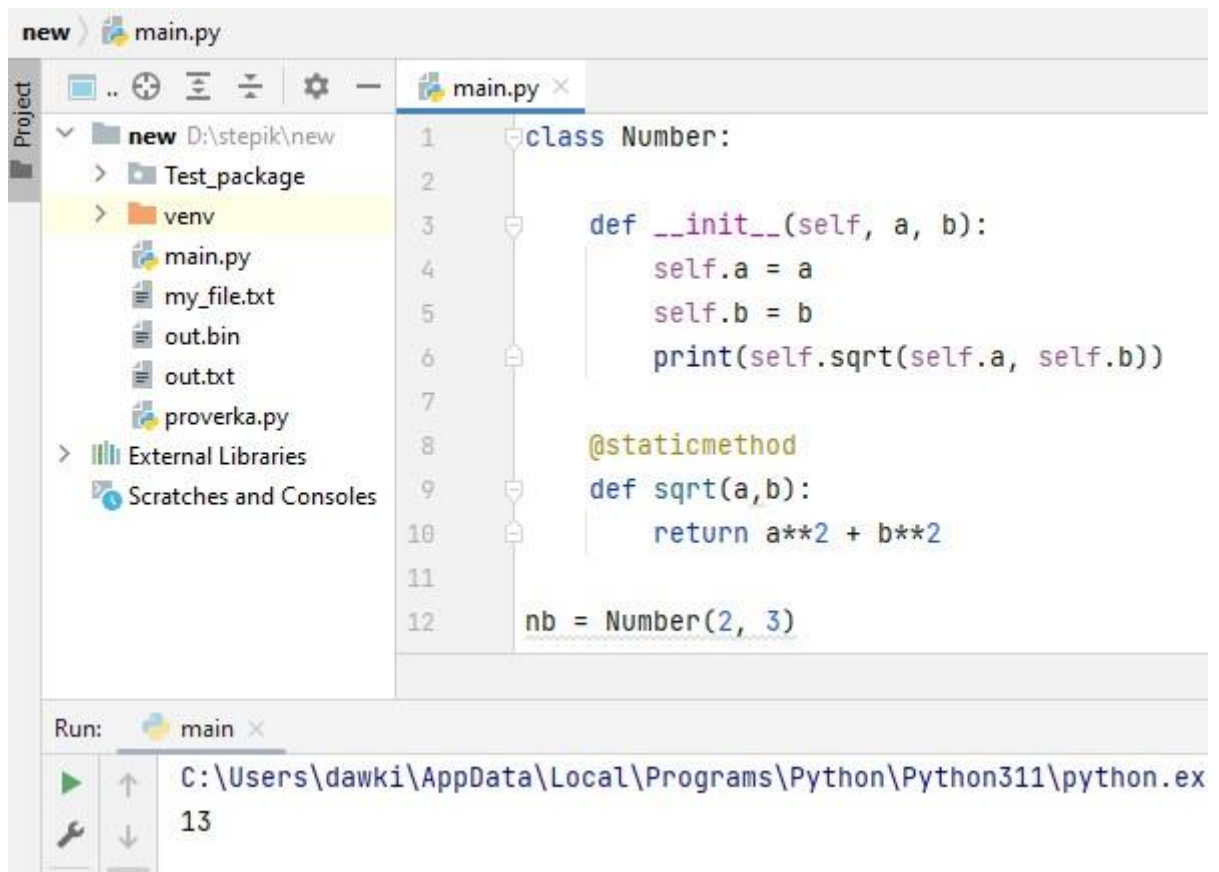
Вопрос 256. Имеет ли метод класса `@classmethod` доступ к экземпляру класса?

Ответ: Сам метод доступа к экземпляру не имеет, но экземпляр-то имеет доступ к методу

Вопрос 257. Что такое статический метод класса `@staticmethod`?

Ответ: Функция `@staticmethod` работает совершенно независимо, он не обращается ни к атрибутам класса, ни к локальным атрибутам экземпляра класса. Декоратор `@staticmethod` определяет независимую сервисную функцию.

LiveCoding: реализуйте статический метод класса



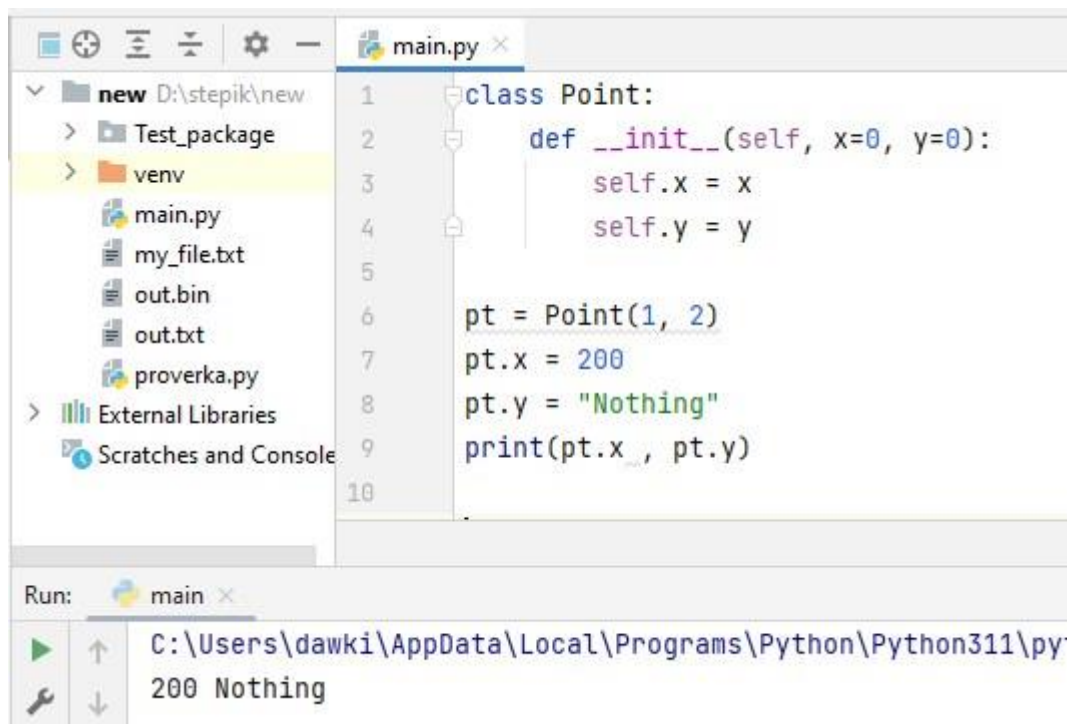
Вопрос 258. Как осуществляется инкапсуляция в Python?

Ответ: Осуществляется с помощью ограничения доступа путем определенного синтаксиса (нижнее подчеркивание у атрибута класса) - Публичный режим доступа public, Защищенный режим доступа protected, Приватный режим доступа private

Вопрос 259. Что делает публичный режим доступа?

Ответ: Позволяет менять значение атрибута экземпляра класса

LiveCoding: реализуйте публичный режим доступа



```
1 class Point:
2     def __init__(self, x=0, y=0):
3         self.x = x
4         self.y = y
5
6     pt = Point(1, 2)
7     pt.x = 200
8     pt.y = "Nothing"
9     print(pt.x, pt.y)
10
```

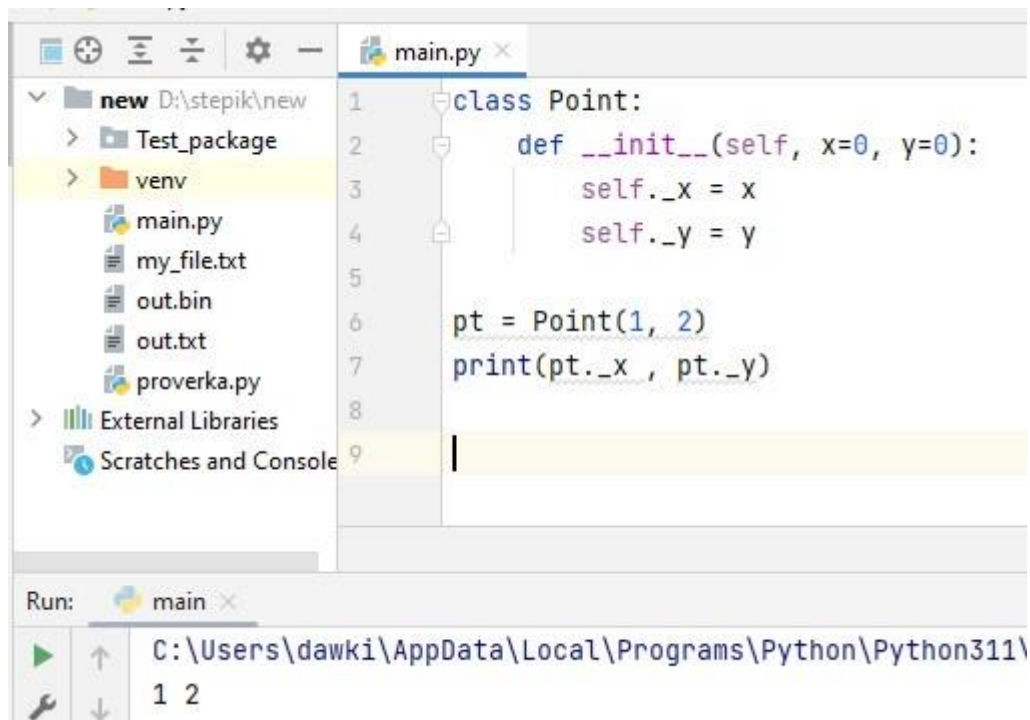
Run: main

C:\Users\dawki\AppData\Local\Programs\Python\Python311\py
200 Nothing

Вопрос 260. Что делает защищенный режим доступа и какая у него особенность?

Ответ: Обозначается с помощью ОДНОГО нижнего подчеркивания в `__init__` у локального атрибута класса. Как видно `protected` работает точно также, как и `public`. ****Нижнее подчеркивание нужно для сигнализации программисту, что данное свойство является защищенным, но никак не ограничивает доступ к нему****. Оно лишь предупреждает программиста и все.

LiveCoding: реализуйте защищенный режим доступа



```
1 class Point:
2     def __init__(self, x=0, y=0):
3         self._x = x
4         self._y = y
5
6     pt = Point(1, 2)
7     print(pt._x, pt._y)
8
9
```

Run: main

C:\Users\dawki\AppData\Local\Programs\Python\Python311\py
1 2

Вопрос 261. Что делает приватный режим доступа и какая у него особенность?

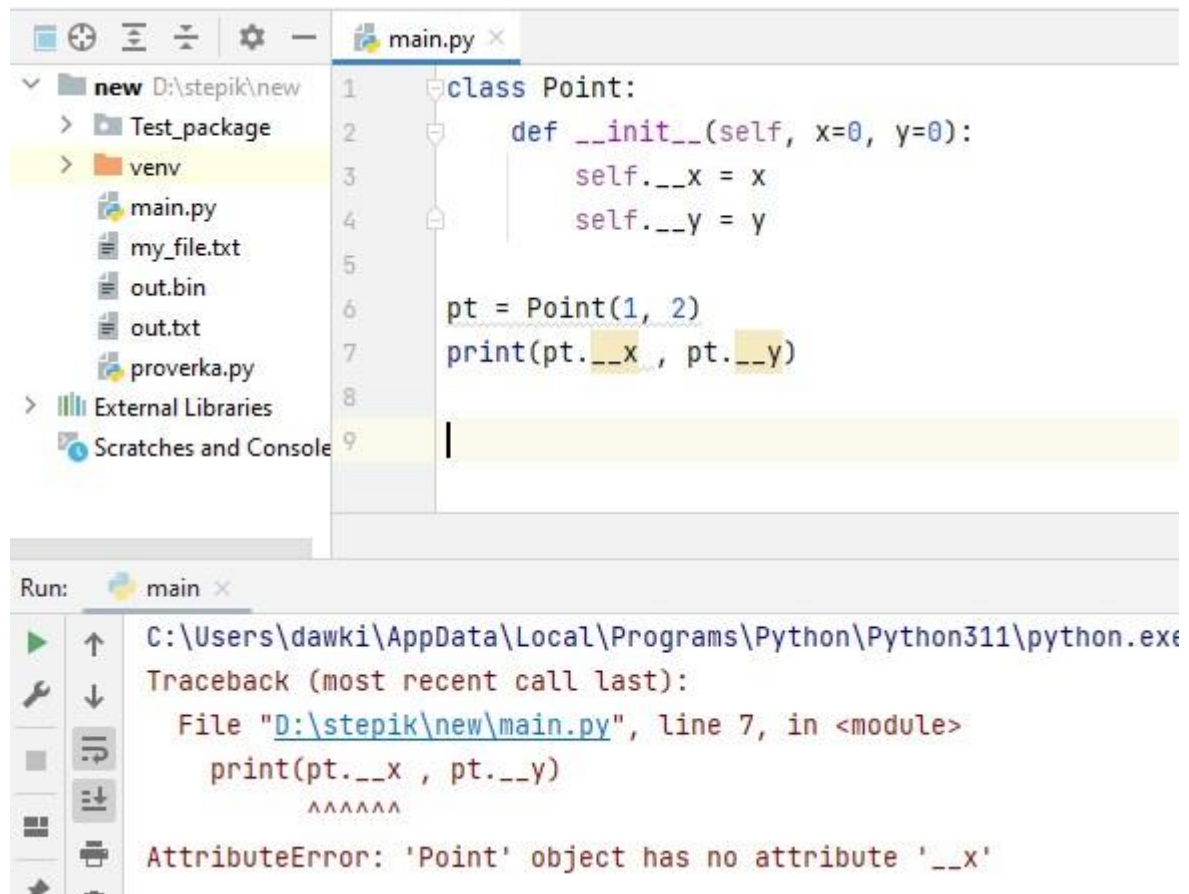
Ответ: Обозначается с помощью ДВУХ нижних подчеркиваний в `__init__` у локального атрибута класса.

У приватного режима две роли:

- скрывает приватные переменные и методы от прямого доступа к ним (по исходным именам) вне класса
- предостерегает программиста (использующего класс) от прямого использования приватных атрибутов вне класса

Из вне мы не можем обращаться к этим атрибутам. Но внутри класса можем

LiveCoding: реализуйте приватный режим



```
1 class Point:
2     def __init__(self, x=0, y=0):
3         self.__x = x
4         self.__y = y
5
6     pt = Point(1, 2)
7     print(pt.__x, pt.__y)
8
9
```

Run: main

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe

Traceback (most recent call last):

File "D:\stepik\new\main.py", line 7, in <module>

print(pt.__x, pt.__y)

^^^^^^

AttributeError: 'Point' object has no attribute '__x'

Вопрос 262. Как можно работать с защищенными локальными атрибутами?

Ответ: В классе Point есть 2 вспомогательных метода `set_coord` и `get_coord`, которые работают с защищенными локальными атрибутами. Такие методы соответственно называются сеттарами и геттерами. Геттеры и сеттеры используются для того, чтобы не нарушать внутренняя логика работы класса. Это публичные методы для считывания и записи значений в приватные переменные класса или его объектов.

Вопрос 263. Зачем в классе создавать приватные атрибуты и еще дополнительно работать с ними из вне?

Ответ: Это реализует инкапсуляцию - класс следует воспринимать как единое целое, и чтобы случайно или нарочно не нарушить правильность алгоритма внутри этого класса, следует взаимодействовать с ним через публичные свойства и методы.

Вопрос 264. Как узнать какие приватные свойства есть у экземпляра класса?

Ответ: С помощью функции `dir`

Вопрос 265. Как надежно запретить обращение к методам класса?

Ответ: Чтобы действительно запретить обращение к методам класса нужно установить модуль `accessify`. `pip install accessify`. Модуль `accessify` следует использовать только тогда, когда нужно надежно защитить какие-нибудь методы.

Вопрос 266. Какой самый простой способ для работы с приватными атрибутами класса?

Ответ: Существует более простой способ работы с приватными атрибутами класса через специальный объект - `property` (переводится "свойство")

Вопрос 267. Для чего служит функция `property`?

Ответ: В классе есть атрибуты, которые являются закрытыми данными, т.е. приватными. К приватным данным можно обратиться только через геттеры и сеттеры.

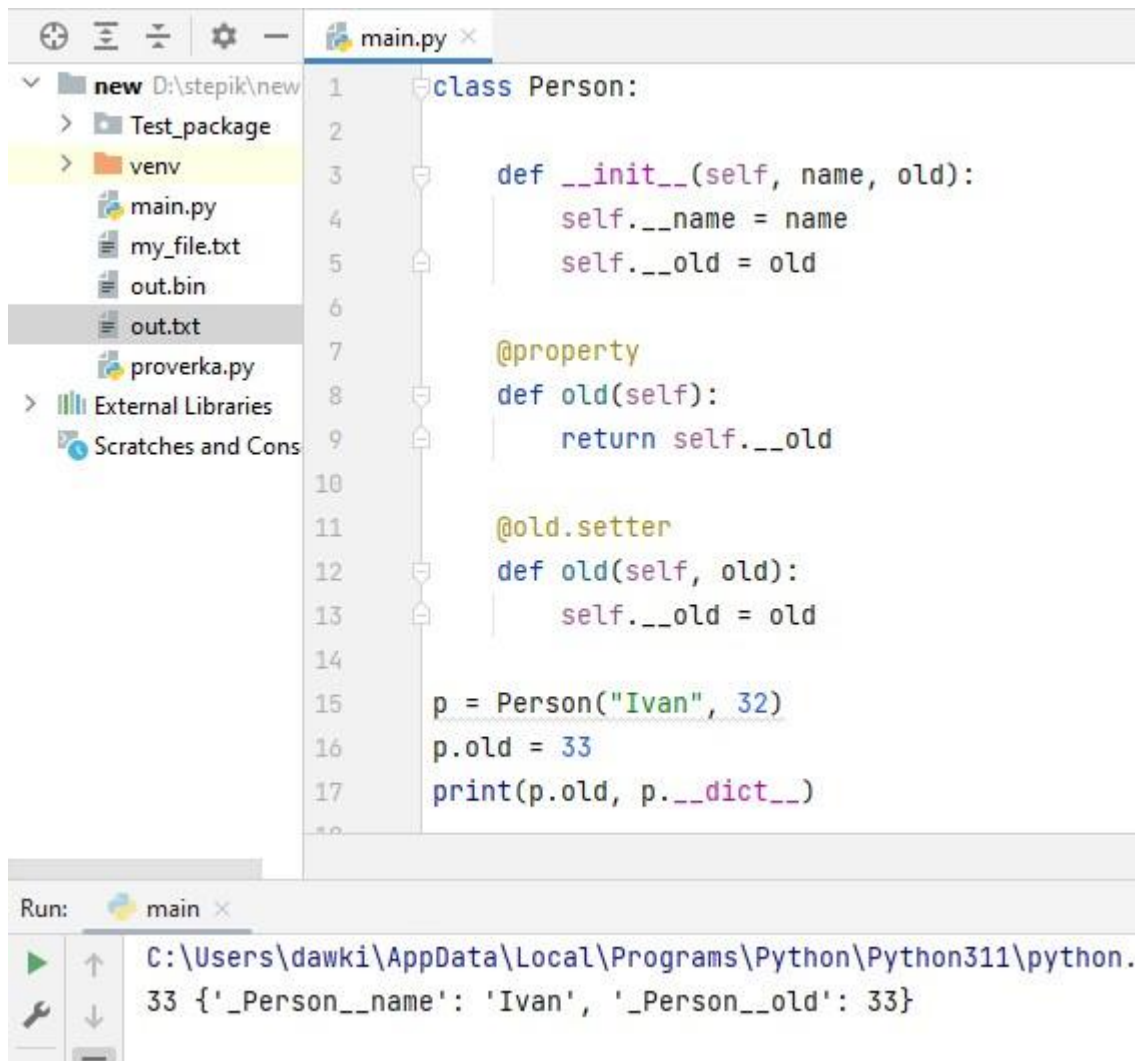
Сеттер - устанавливает и меняет, геттер - возвращает то, что изменили.

Сеттеры и геттеры можно упаковать в функцию `property`

Вопрос 268. Что такое объекты-свойства и как они работают?

Ответ: объекты-свойства - это приватные атрибуты экземпляра класса, к которым можно обращаться из вне. Если в классе задан объект-свойство, то в первую очередь выбирается оно, даже если в экземпляре класса есть свойство с таким же именем. Т.е. есть приоритеты

LiveCoding: реализуйте объекты-свойства и обратитесь и изменяйте приватный атрибут экземпляра класса



```
1 class Person:
2
3     def __init__(self, name, old):
4         self.__name = name
5         self.__old = old
6
7     @property
8     def old(self):
9         return self.__old
10
11     @old.setter
12     def old(self, old):
13         self.__old = old
14
15 p = Person("Ivan", 32)
16 p.old = 33
17 print(p.old, p.__dict__)
```

Run: main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.
33 {'_Person__name': 'Ivan', '_Person__old': 33}

Декоратор `@property` ВСЕГДА ставится на геттер, а декоратор с именем метода геттера ставится на сеттер. Теперь можно менять у экземпляра класса свойство экземпляра

У декоратора `@property` есть методы `setter`, `getter` и `deleter`

Вопрос 269. В каких случаях целесообразно использовать объекты-свойства `@property`?

Ответ: - для удобства доступа к приватным атрибутам класса или объектов

- для реализации дополнительной логики (программы) в момент присваивания и считывания информации из атрибутов класса или объектов

- для инициализации локальных свойств в момент создания объектов

Вопрос 270. Что применяют, если требуется много объектов-свойств в классе?

Ответ: Чтобы программу уменьшить делают дескрипторы. Дескриптор - это класс, который делится на 2 вида - `non-data descriptor` (дескриптор не данных), и на `data descriptor` (дескриптор данных)

LiveCoding: напишите дескриптор

```
main.py x
1 class Integer:
2     def __set_name__(self, owner, name):
3         self.name = "_" + name
4
5     def __get__(self, instance, owner):
6         return instance.__dict__[self.name]
7
8     def __set__(self, instance, value):
9         print(f"__set__: {self.name} = {value}")
10        instance.__dict__[self.name] = value
11
12
13 class Point3D:
14     x = Integer()
15     y = Integer()
16     z = Integer()
17
18     def __init__(self, x, y, z):
19         self.x = x
20         self.y = y
21         self.z = z
22
23
24 pt = Point3D(1, 2, 3)
25 print(pt.__dict__)
26
```

Вопрос 271. Какой метод вызывается, когда происходит считывание атрибута через экземпляр класса?

Ответ: Метод записывается так: `def __getattribute__(self, item):`. `item` - это атрибут, к которому идет обращение. Этот метод АВТОМАТИЧЕСКИ вызывается, когда идет считывание атрибута через экземпляр класса.

LiveCoding: покажите практическое использование метода `getattribute`

Для того, чтобы запретить обращаться к определенному атрибуту экземпляра класса


```
8
9
10 def __getattr__(self, item):
11     if item == "x":
12         raise ValueError(f"К {item} обратиться нельзя")
13     else:
14         return object.__getattr__(self, item)
15
16 pt = Point(15, 10)
17 print(pt.y) # К y мы можем обращаться
18 print()
19 print(pt.x) # К x уже запрещено
```

main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepik\new\main.py

10

Traceback (most recent call last):

File "D:\stepik\new\main.py", line 18, in <module>

print(pt.x)

^^^^

File "D:\stepik\new\main.py", line 11, in __getattr__

raise ValueError(f"К {item} обратиться нельзя")

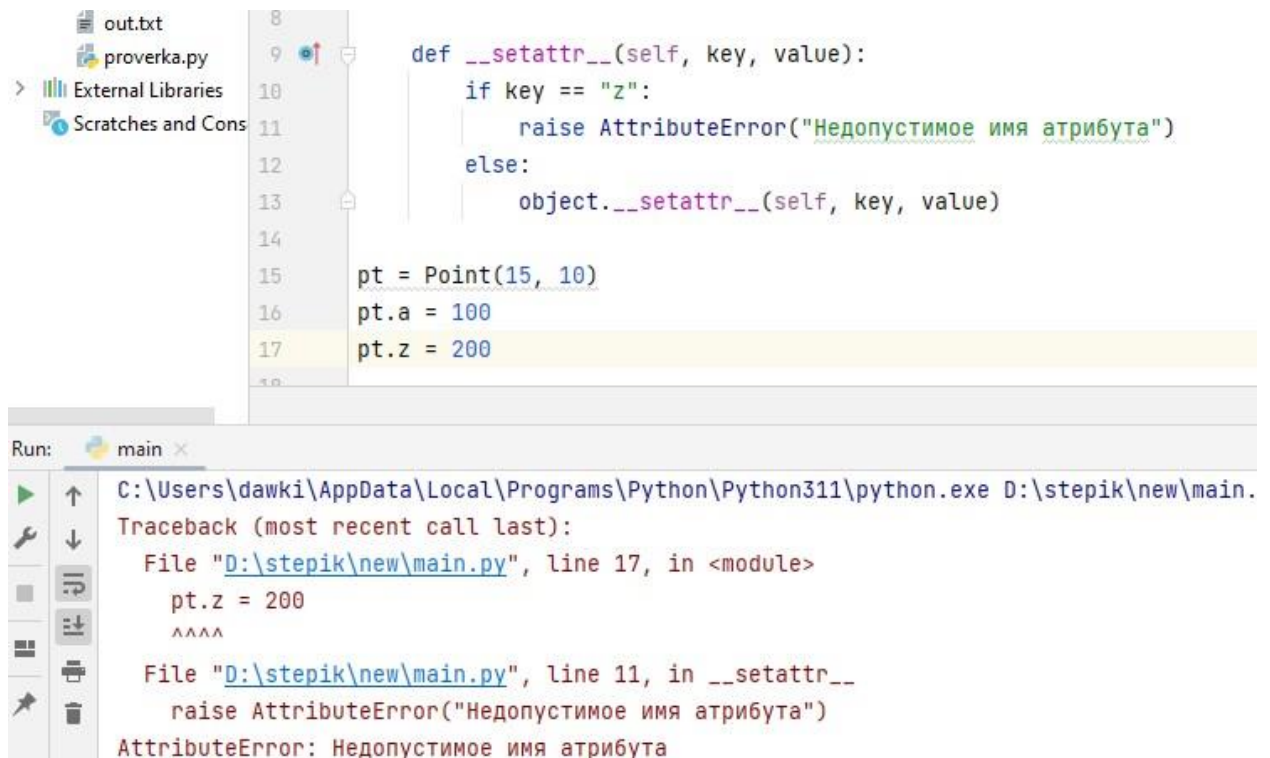
ValueError: К x обратиться нельзя

Вопрос 272. Какой метод вызывается, когда идет присвоение какому-нибудь атрибуту какого-либо значения?

Ответ: Он АВТОМАТИЧЕСКИ вызывается всякий раз, когда идет присвоение какому-либо атрибуту какого-либо значения. `__setattr__(self, key, value)`: - `key` - это имя атрибута, `value` - значение, которое присваивается атрибуту.

LiveCoding: покажите практическое применение `setattr`

С помощью этого магического метода мы можем запретить создавать какой-либо локальный атрибут в экземпляре класса



```
8
9
10
11 def __setattr__(self, key, value):
12     if key == "z":
13         raise AttributeError("Недопустимое имя атрибута")
14     else:
15         object.__setattr__(self, key, value)
16
17 pt = Point(15, 10)
18 pt.a = 100
19 pt.z = 200
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run: main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepik\new\main.

Traceback (most recent call last):

File "D:\stepik\new\main.py", line 17, in <module>

pt.z = 200

^^^^

File "D:\stepik\new\main.py", line 11, in __setattr__

raise AttributeError("Недопустимое имя атрибута")


AttributeError: Недопустимое имя атрибута

Вопрос 273. Какой метод вызывается, когда идет обращение к несуществующему экземпляру класса?

Ответ: Вызывается АВТОМАТИЧЕСКИ каждый раз, когда идет обращение к НЕСУЩЕСТВУЮЩЕМУ экземпляру класса. `__getattr__`

LiveCoding: покажите практическое применение `getattr`

Если идет обращение к несуществующему атрибуту экземпляра класса, то пусть в этом случае возвращается значение `False`. БЕЗ ЭТОГО БУДЕТ ОШИБКА `AttributeError`



```
8
9
10
11 def __getattr__(self, item):
12     return False
13
14 pt = Point(15, 10)
15 print(pt.yy)
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run: main x

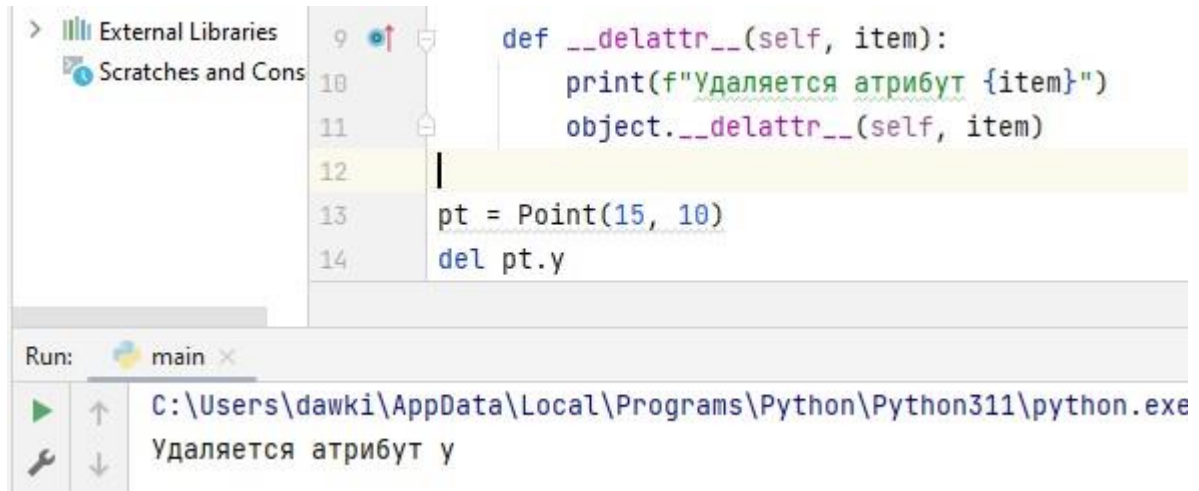
C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepik\new\main.

False

Вопрос 274. Какой метод вызывается, когда удаляется определенный атрибут из экземпляра класса?

Ответ: Магический метод `__delattr__` Он АВТОМАТИЧЕСКИ вызывается всякий раз, когда удаляется определенный атрибут из экземпляра класса.

LiveCoding: покажите практическое применение delattr



```
def __delattr__(self, item):  
    print(f"Удаляется атрибут {item}")  
    object.__delattr__(self, item)  
  
pt = Point(15, 10)  
del pt.y
```

Run: main

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe
Удаляется атрибут y

Вопрос 275. Допустим экземпляр класса создается следующим образом `p = Counter()`. Что означают круглые скобки?

Ответ: эти двойные скобки - это оператор вызова функции или класса. Т.е. когда мы прописываем круглые скобки после названия класса - автоматически вызывается этот метод `call`, причем ему могут быть переданы какие-нибудь параметры. Класс с двойными скобками мы можем вызывать, а экземпляр класса уже не можем. `p()` - нельзя.

Вопрос 276. Что такое функторы?

Ответ: это классы, где можно вызывать экземпляр класса.

LiveCoding: Напишите программу, в которой можно было использовать экземпляры класса следующим образом `c()`, `c(20)`

```
new D:\stepik\new
> Test_package
> venv
  main.py
  my_file.txt
  out.bin
  out.txt
  proverka.py
> External Libraries
> Scratches and Cons

1 class Counter:
2
3     def __init__(self):
4         self.__counter = 0
5
6     def __call__(self, *args, **kwargs):
7         print("Вызывается метод __call__")
8         self.__counter += 1
9         return self.__counter
10
11 c = Counter()
12 c()
```

```
Run: main x
C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe
Вызывается метод __call__
```

```
10
11 c = Counter()
12 c()
13 c()
14 c()
15 res = c()
16 print(res)
17
```

```
Run: main x
C:\Users\dawki\AppData\Local\Programs\Py
Вызывается метод __call__
Вызывается метод __call__
Вызывается метод __call__
Вызывается метод __call__
4
```

```
proverka.py
> External Libraries
Scratches and Cons
10
11 c1 = Counter()
12 c2 = Counter()
13 c1()
14 c2()
15 c2()
16 res = c1()
17 res2 = c2()
18 print(res, res2)
19
```

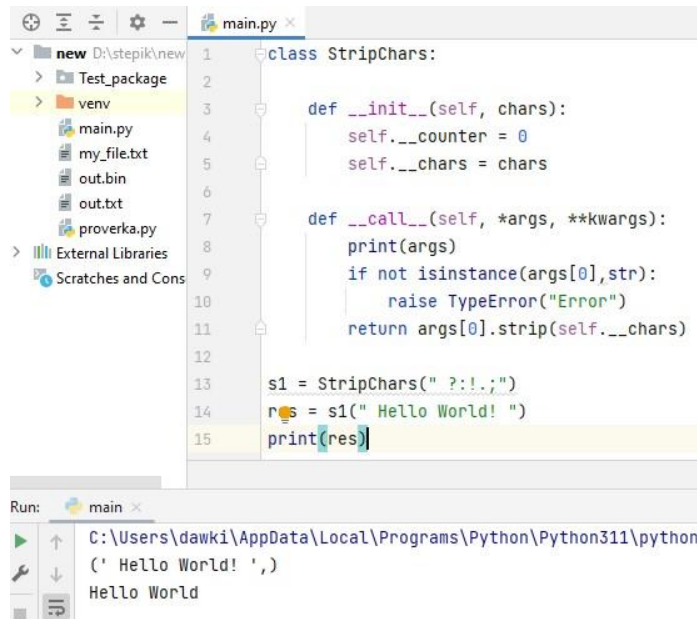
```
Run: main x
C:\Users\dawki\AppData\Local\Programs\Python
Вызывается метод __call__
Вызывается метод __call__
Вызывается метод __call__
Вызывается метод __call__
Вызывается метод __call__
2 3
```

```
main.py x
new D:\stepik\new
> Test_package
> venv
main.py
my_file.txt
out.bin
out.txt
proverka.py
External Libraries
Scratches and Cons
1 class Counter:
2
3     def __init__(self):
4         self.__counter = 0
5
6     def __call__(self, step=1, *args, **kwargs):
7         print("Вызывается метод __call__")
8         self.__counter += step
9         return self.__counter
10
11 c = Counter()
12 c(20)
13 res = c()
14 print(res)
15
```

```
Run: main x
C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe
Вызывается метод __call__
Вызывается метод __call__
21
```

Вопрос 277. Как можно с помощью класса реализовать замену замыканий функций?

Ответ:



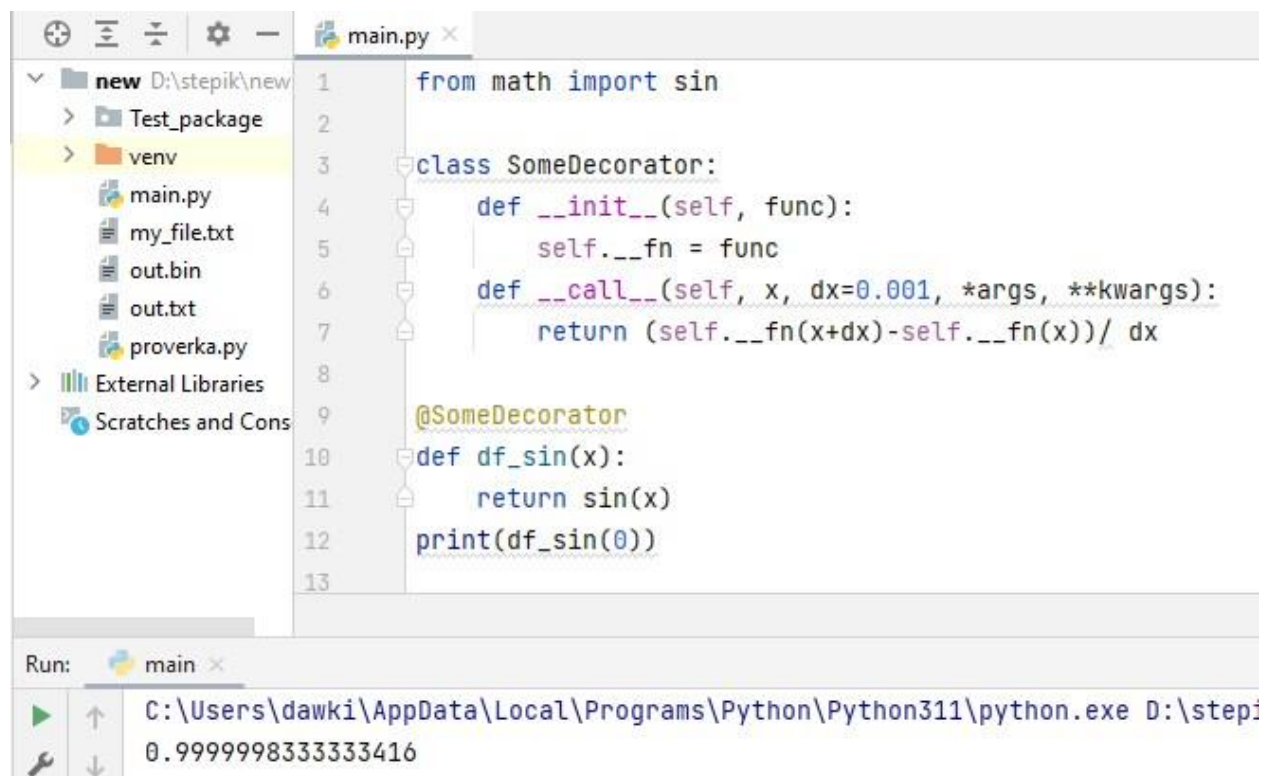
```
1 class StripChars:
2
3     def __init__(self, chars):
4         self.__counter = 0
5         self.__chars = chars
6
7     def __call__(self, *args, **kwargs):
8         print(args)
9         if not isinstance(args[0], str):
10            raise TypeError("Error")
11            return args[0].strip(self.__chars)
12
13 s1 = StripChars(" ?!.,;")
14 res = s1(" Hello World! ")
15 print(res)
```

Run: main ×

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.
(' Hello World! ',)
Hello World

В 2.60 Как создать декоратор с помощью класса?

Ответ:



```
1 from math import sin
2
3 class SomeDecorator:
4     def __init__(self, func):
5         self.__fn = func
6     def __call__(self, x, dx=0.001, *args, **kwargs):
7         return (self.__fn(x+dx)-self.__fn(x))/ dx
8
9 @SomeDecorator
10 def df_sin(x):
11     return sin(x)
12     print(df_sin(0))
13
```

Run: main ×

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepi
0.9999998333333416

Вопрос 278. Как в консоль вывести экземпляр класса с помощью строки? Чтобы не было отображение ячейки памяти?

Ответ: Магический метод `__repr__` Он прописывается программистом для отображения информации об объекте класса в режиме отладки* Это значит, что мы для себя можем вывести служебную информацию.

Магический метод `__str__` Он прописывается программистом для отображения информации об объекте класса для пользователей с вызовом функции `print()` или `str()`

LiveCoding: реализуйте метод для отображения информации в консоль



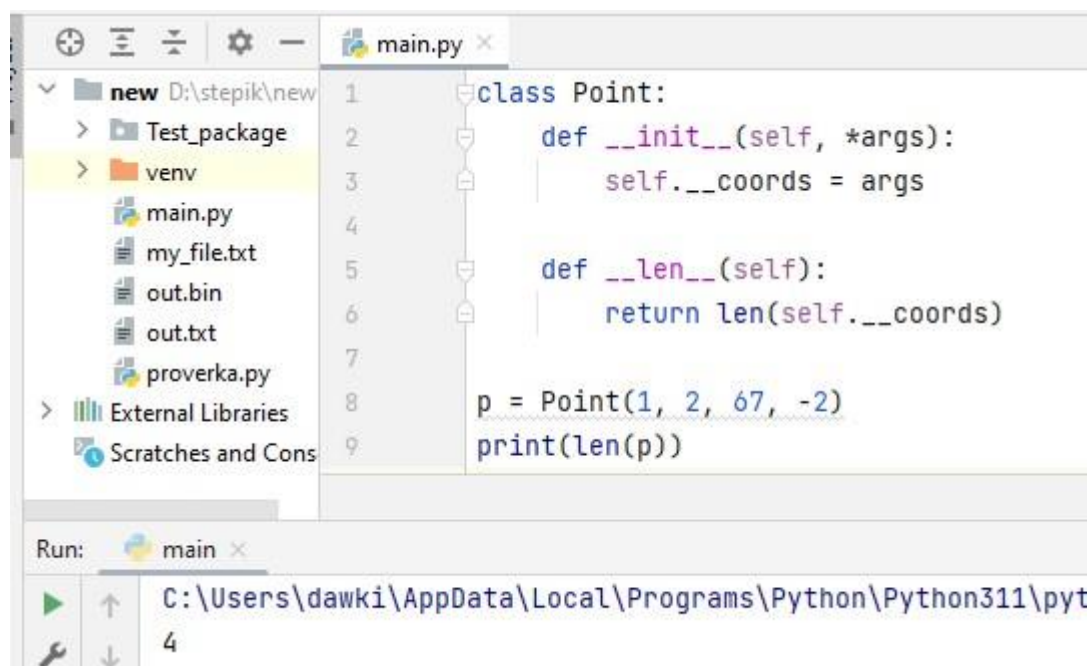
```
1 class Cat:
2
3     def __init__(self, name, age):
4         self.name = name
5         self.age = age
6
7     def __repr__(self):
8         return f"{self.__class__}: {self.name}"
9
10    def __str__(self):
11        return f"My cat named {self.name} is {self.age} years old"
12
13    c = Cat("Tom", 2)
14    print(c)
15
```

Run: main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepik\new\main.py
My cat named Tom is 2 years old

Вопрос 279. Как можно к экземпляру класса применить функцию `len`?

Ответ: Прописывается программистом, позволяет применять функцию `len()` к экземпляру класса



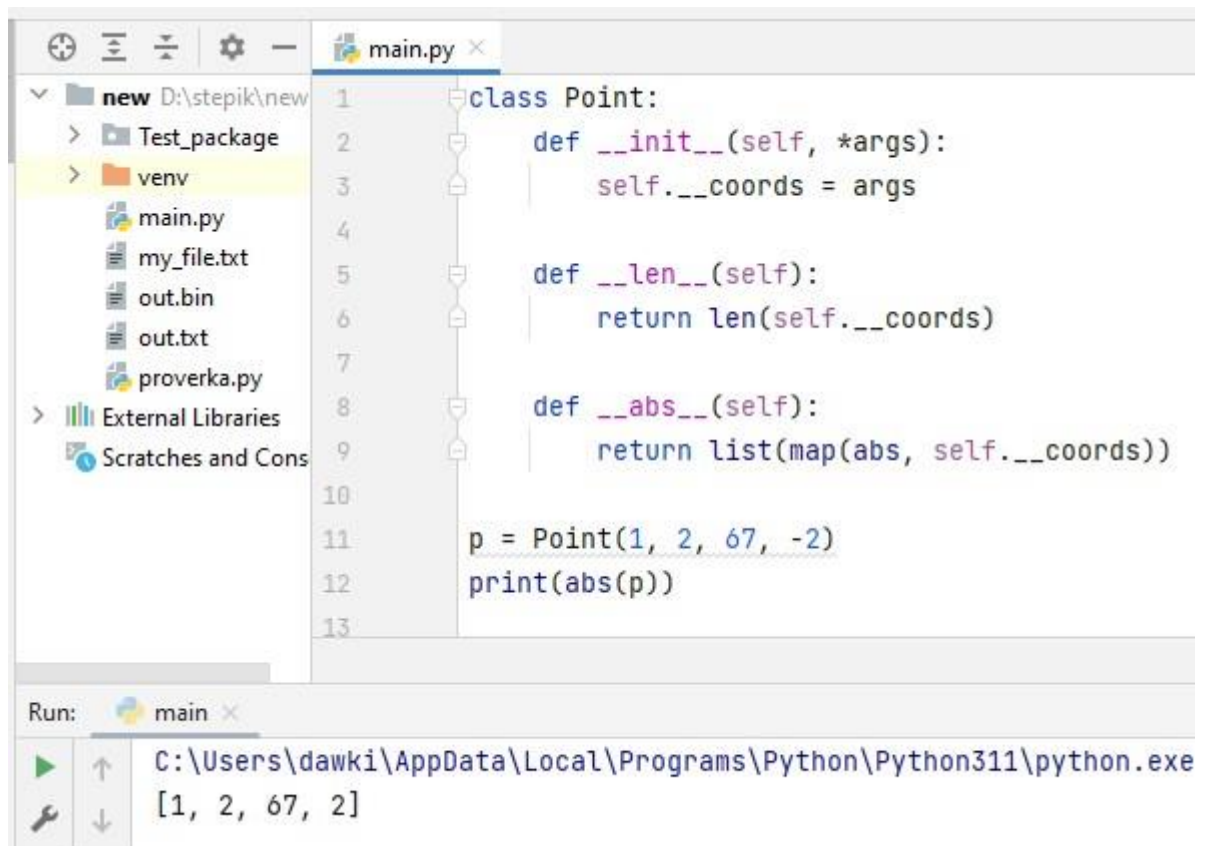
```
1 class Point:
2     def __init__(self, *args):
3         self.__coords = args
4
5     def __len__(self):
6         return len(self.__coords)
7
8     p = Point(1, 2, 67, -2)
9     print(len(p))
```

Run: main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\pyt
4

Вопрос 280. Как можно к экземпляру класса применить функцию abs?

Ответ: Прописывается программистом, позволяет применять функцию abs() к экземпляру класса



```
1 class Point:
2     def __init__(self, *args):
3         self.__coords = args
4
5     def __len__(self):
6         return len(self.__coords)
7
8     def __abs__(self):
9         return list(map(abs, self.__coords))
10
11 p = Point(1, 2, 67, -2)
12 print(abs(p))
13
```

Run: main ×

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe
[1, 2, 67, 2]

Вопрос 281. Как можно сложить экземпляры классов?

Ответ: Если нам надо складывать, вычитать, умножать, делить, делать целочисленные деления и делать деление с остатком между экземплярами класса, или экземпляров и числом - используют специальные арифметические магические методы.

LiveCoding: напишите программу, которая складывает, вычитает, умножает и инкрементирует экземпляры класса, а также сложение с числом.

```

class Clock:
    __DAY = 86400 # Число секунд в одном дне

    def __init__(self, seconds: int): # Аннотация типов
        if not isinstance(seconds, int):
            raise TypeError("Секунды должны быть целым числом")
        self.seconds = seconds % self.__DAY

    def get_time(self):
        s = self.seconds % 60
        m = (self.seconds // 60) % 60
        h = (self.seconds // 3600) % 24
        return f"{self.__get_formatted(h)}:{self.__get_formatted(m)}:{self.__get_formatted(s)}"

    @classmethod
    def __get_formatted(cls, x):
        return str(x).rjust(2, "0")

    def __add__(self, other): # Теперь нам доступно сложение
        if not isinstance(other, (int, Clock)): # Проверяем входные данные
            raise ArithmeticError("Операнд должен быть или объектом Clock или int")
        sc = other
        if isinstance(other, Clock):
            sc = other.seconds
        return Clock(self.seconds + sc)

    def __radd__(self, other): # В том случае если справа у нас число при сложении с объектом класса
        return self + other

    def __iadd__(self, other): # В том случае, если мы хотим применить "+="
        if not isinstance(other, (int, Clock)): # Проверяем входные данные
            raise ArithmeticError("Операнд должен быть или объектом Clock или int")
        sc = other
        if isinstance(other, Clock):
            sc = other.seconds
        self.seconds += sc
        return self

    def __str__(self):
        return f"{self.get_time()}"

```

Пример работы класса для вычисления времени:

```

41
42     c1 = Clock(3600)
43     c2 = Clock(4000)
44     print(c1 + c2)
45     print(c1 + 600)
46     print(700 + c2)
47     c2 += 1675
48     print(c2)

```

main x

↑

↓

0

1

2

C:\Users\dawki\AppData\Local\Programs\Python\P
02:06:40
01:10:00
01:18:20
01:34:35

Оператор	Метод оператора	Оператор	Метод оператора
<code>x + y</code>	<code>__add__(self, other)</code>	<code>x += y</code>	<code>__iadd__(self, other)</code>
<code>x - y</code>	<code>__sub__(self, other)</code>	<code>x -= y</code>	<code>__isub__(self, other)</code>
<code>x * y</code>	<code>__mul__(self, other)</code>	<code>x *= y</code>	<code>__imul__(self, other)</code>
<code>x / y</code>	<code>__truediv__(self, other)</code>	<code>x /= y</code>	<code>__itruediv__(self, other)</code>
<code>x // y</code>	<code>__floordiv__(self, other)</code>	<code>x //= y</code>	<code>__ifloordiv__(self, other)</code>
<code>x % y</code>	<code>__mod__(self, other)</code>	<code>x %= y</code>	<code>__imod__(self, other)</code>

Вопрос 282. Можно ли сравнивать экземпляры классов?

Ответ: Эти магические методы объявляются для того чтобы сравнивать между собой значения экземпляров класса или экземпляр класса с числом.

- `__eq__()` – для равенства `==`
- `__ne__()` – для неравенства `!=`
- `__lt__()` – для оператора меньше `<`
- `__le__()` – для оператора меньше или равно `<=`
- `__gt__()` – для оператора больше `>`
- `__ge__()` – для оператора больше или равно `>=`

LiveCoding: реализуйте сравнение экземпляров класса


```

1 class Clock:
2
3     __DAY = 86400
4
5     def __init__(self, seconds: int):
6         if not isinstance(seconds, int):
7             raise TypeError("Неверное значение секунд")
8         self.seconds = seconds
9
10    @classmethod
11    def checker(cls, other):
12        if not isinstance(other, (int, Clock)):
13            raise TypeError("Ошибка типа данных")
14        return other if isinstance(other, int) else other.seconds
15
16    def __eq__(self, other):
17        sc = self.checker(other)
18        return self.seconds == sc
19
20    def __lt__(self, other):
21        sc = self.checker(other)
22        return self.seconds < sc
23
24    def __gt__(self, other):
25        sc = self.checker(other)
26        return self.seconds > sc
27

```

Далее необходимо сравнивать экземпляры классов

```

27
28     c1 = Clock(500)
29     c2 = Clock(1000)
30     print(c1 == c2, c1 > c2, c1 != c2, c1 < c2)

```

in: main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe

False False True True

Вопрос 283. Что такое hash в Python?

Ответ: Это функция, которая возвращает хеш-значение объекта, если оно есть. Хэш-значения являются целыми числами.

Вопрос 284. Что такое хэш-значение?

Ответ: это определенный набор символов, который присущ только этому массиву входящей информации

Вопрос 285. Что в Python является хешируемыми объектами?

Ответ: Большинство неизменяемых встроенных объектов Python являются хешируемыми и имеют хеш-значение. Изменяемые контейнеры, такие как списки или словари, не имеют хеш-значений

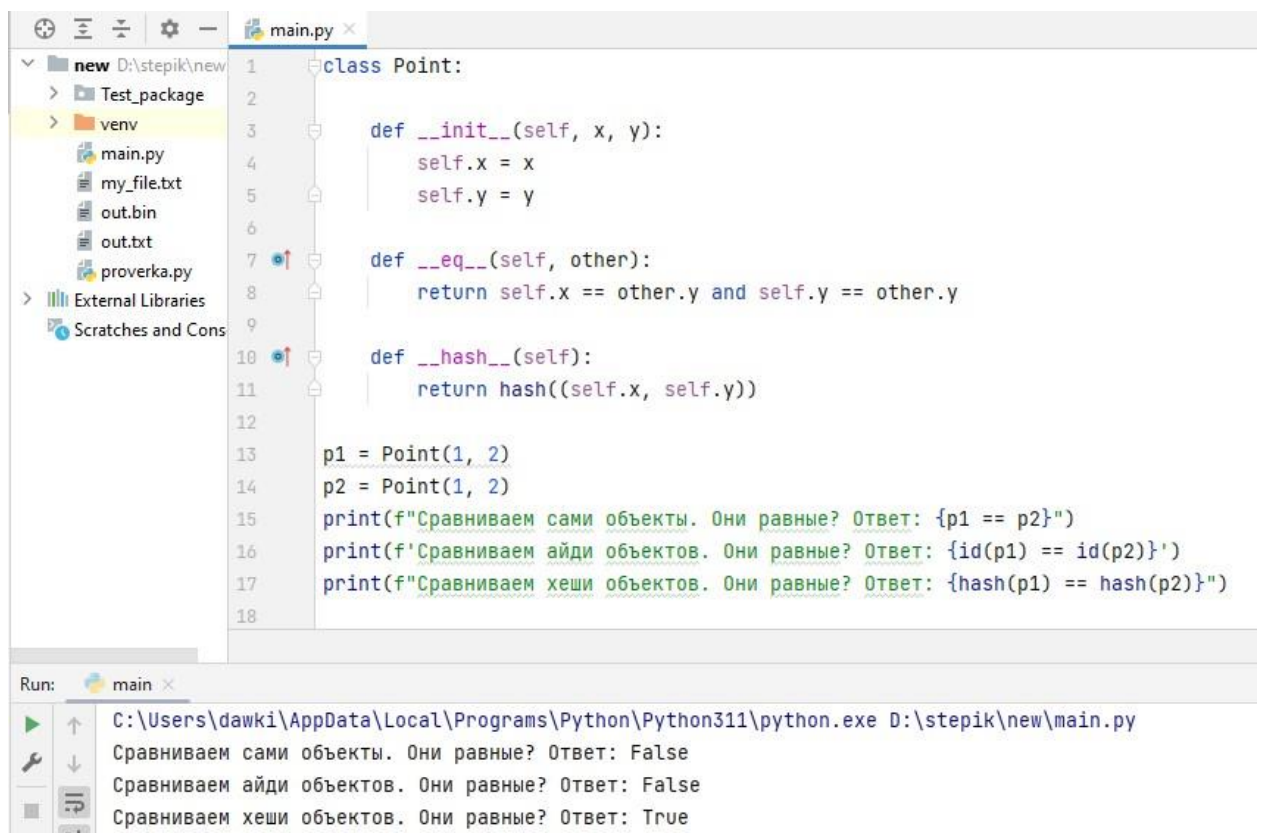
Вопрос 286. Для чего используется hash?

Ответ: Они используются для быстрого сравнения ключей словаря во время поиска в словаре. Дело в том, что ключи в словаре - это хеши, с помощью которых реализуют быстрый алгоритм поиска ключа и значения по ключу в словаре.

Вопрос 287. Экземпляры классы являются хешируемыми объектами?

Ответ: Если ОБА метода `__eq__` и `__hash__` не переопределены, то не являются. При попытке хешировать экземпляры класса будет ошибка. Пользовательские типы могут переопределять метод `__hash__()`, результат которого будет использован при вызове функции `hash()`. Однако, следует помнить, что функция `hash()` обрезает значение в соответствии с битностью хоста.

LiveCoding: реализуйте выполнение hash для экземпляра класса



```
1 class Point:
2
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6
7     def __eq__(self, other):
8         return self.x == other.x and self.y == other.y
9
10    def __hash__(self):
11        return hash((self.x, self.y))
12
13    p1 = Point(1, 2)
14    p2 = Point(1, 2)
15    print(f"Сравниваем сами объекты. Они равные? Ответ: {p1 == p2}")
16    print(f'Сравниваем айди объектов. Они равные? Ответ: {id(p1) == id(p2)}')
17    print(f"Сравниваем хеши объектов. Они равные? Ответ: {hash(p1) == hash(p2)}")
18
```

Run: main

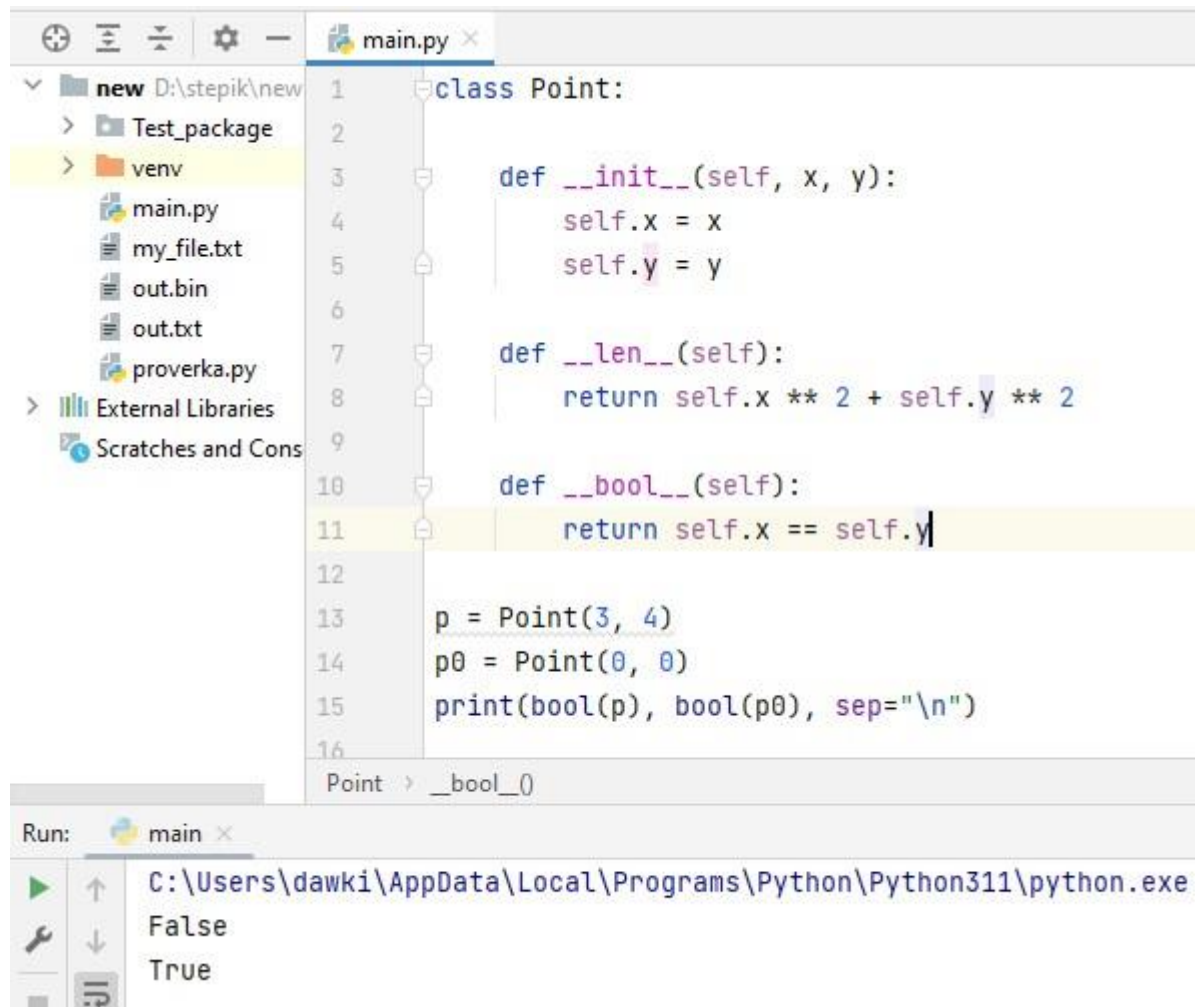
```
C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe D:\stepik\new\main.py
Сравниваем сами объекты. Они равные? Ответ: False
Сравниваем айди объектов. Они равные? Ответ: False
Сравниваем хеши объектов. Они равные? Ответ: True
```

Переопределение метода `__hash__()` стоит делать, когда ключами словаря будут объекты класса при равных координатах. Это тонкая настройка

Вопрос 288. Как правильно можно использовать функцию bool и для чего её надо использовать?

Ответ: Магический метод `__bool__` служит для настройки проверки правдивости (True или False) экземпляра класса. Если методы `__len__` и `__bool__` не определены, то при вызове `bool()` от любого экземпляра класса мы получим True. Метод `__bool__` обязательно должен возвращать True или False и ничего другого. Метод `__bool__` используется с условным оператором `if` и циклом `while`.

LiveCoding: реализуйте функцию bool для экземпляра класса



```
1 class Point:
2
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6
7     def __len__(self):
8         return self.x ** 2 + self.y ** 2
9
10    def __bool__(self):
11        return self.x == self.y
12
13    p = Point(3, 4)
14    p0 = Point(0, 0)
15    print(bool(p), bool(p0), sep="\n")
16
```

Run: main x

C:\Users\dawki\AppData\Local\Programs\Python\Python311\python.exe

False

True

Вопрос 289. Какие методы существуют для работы с последовательностями?

Ответ: Эти методы прописываются в программе, когда экземпляр класса на вход принимает какую-нибудь последовательность.

Магические методы

- `__getitem__(self, item)` – получение значения по ключу `item`;
- `__setitem__(self, key, value)` – запись значения `value` по ключу `key`;
- `__delitem__(self, key)` – удаление элемента по ключу `key`.

LiveCoding: реализуйте методы для работы с последовательностями?

```
main.py x
1 class Student:
2
3     def __init__(self, name, marks):
4         self.name = name
5         self.marks = marks
6
7     def __getitem__(self, item):
8         if 0 <= item <= len(self.marks):
9             return self.marks[item]
10        else:
11            raise IndexError("Неверный индекс")
12
13    def __setitem__(self, key, value):
14        if not isinstance(key, int) or key < 0:
15            raise TypeError("Индекс должен быть целым и неотрицательным")
16
17        if key >= len(self.marks):
18            off = key + 1 - len(self.marks)
19            self.marks.extend([None]*off)
20
21        self.marks[key] = value
22
23    def __delitem__(self, key):
24        if not isinstance(key, int):
25            raise TypeError("Индекс должен быть целым и неотрицательным")
26
27        del self.marks[key]
28
29    st = Student("Сергей", [5, 5, 3, 2, 4, 5])
30    print(st[3]) # Сработает метод __getitem__
31    st[10] = 5 # Сработает метод __setitem__
32    print(st.marks)
33    del st[10] # Сработает метод __delitem__
34    print(st.marks)
```

Вопрос 290. Как сделать экземпляр класса итерируемым?

Ответ: объявить в классе методы `__iter__` и `__next__`

Вопрос 291. Как обозначается наследование в Python?

Ответ: Наследование обозначается в круглых скобках у дочернего класса.

Вопрос 292. Какой порядок поиска атрибутов и методов в дочернем классе?

Ответ: При вызове атрибутов и свойств у экземпляра класса вначале эти атрибуты и свойства ищутся в дочернем классе, а если их там нет, то в родительском классе.

Вопрос 293. Как наследование соотносится с принципом DRY?

Ответ: Наследование помогает избежать дублирования кода - метод можно прописать в родительском классе 1 раз, а вызывать его из экземпляров дочерних классов

Вопрос 294. Какие особенности у self в родительском классе?

Ответ: Параметр self в родительских (базовых) классах может ссылаться не только на экземпляр (объекты) этого же класса, но и на объекты дочерних классов. Если мы в базовом классе хотим вызвать метод дочернего класса - лучше этого не делать из-за двойственности self. Но в дочернем классе можно вызывать методы базового класса. Наследование - это когда один класс (дочерний) расширяет функциональность другого (базового) класса

Вопрос 295. Откуда берутся все магические методы у базового класса в программе?

Ответ: Если при записи класса мы ничего в скобках не указываем - это значит, что класс является дочерним самого базового класса object. Именно поэтому у такого класса все атрибуты и методы берутся из базового класса object

Вопрос 296. Как проверить наследование?

Ответ: С помощью функции `issubclass(class, class)`.

Вопрос 297. Что такое расширение дочернего класса?

Ответ: Расширение дочернего класса - это когда в дочернем классе прописывается дополнительный метод или свойство, которого нет в базовом. Расширение - `extended`

Вопрос 298. Что такое переопределение дочернего класса?

Ответ: Переопределение - это когда в дочернем классе прописывается метод или свойство, который есть в базовом. Переопределение - `override`

Вопрос 299. Как обратиться к базовому классу из дочернего?

Ответ: Если существует повторение атрибутов или методов в разных дочерних классах, то можно эти повторяющиеся элементы вывести в базовый класс с помощью функции `super()`.

`super()` - это ссылка на базовый класс, именно это слово используется потому, что название базового класса может измениться. В этом примере `super()` используется для того чтобы попасть в инициализатор базового класса. Инициализатор базового класса всегда вызывается в самую первую очередь, чтобы была правильная инициализация. Делегирование - это использование

функции `super()` для того чтобы переложить выполнение части обязанностей на инициализатор базового класса

LiveCoding: обратитесь к инициализатору базового класса из дочернего класса

Вопрос 300. Что такое `Callable()`?

Ответ: Т.е. callable - это фильтр для вызываемых методов класса.

Вопрос 301. Можно ли из дочернего класса вызвать приватные свойства базового класса?

Ответ: нельзя, надо в базовом классе, вместо приватных свойств записать защищенные (одно подчеркивание)

Вопрос 302. Чем если метод начинается и заканчивается двойными подчеркиваниями - он приватный или публичный?

Ответ: публичный

Вопрос 303. Магические методы являются приватными или публичными?

Ответ: публичными

Вопрос 304. Что такое полиморфизм?

Ответ: Полиморфизм - это возможность работы с объектами разных классов единым образом, то есть через единый интерфейс.

Вопрос 305. Что нужно для реализации полиморфизма?

Ответ: Для реализации полиморфизма необходимо во всех классах назвать одинаковым именем схожие методы. Также во избежание ошибок применяют абстрактные методы.

LiveCoding: реализуйте полиморфизм на любом примере

Вопрос 306. Что такое абстрактные методы?

Ответ: Абстрактные методы - это методы, которые обязательно нужно переопределять в дочерних классах и которые не имеют своей собственной реализации. В этом методы обозначается исключение `NotImplementedError`.

LiveCoding: реализуйте полиморфизм с абстрактными методами на любом примере

Вопрос 307. Нужно ли реализовывать полиморфизм в Python через механизм наследования?

Ответ: не всегда нужно, т.к. полиморфизм встроен в Python через механизм ссылок (каждая переменная - это ссылка на данные любого существующего в программе типа).

Вопрос 308. Какая существует библиотека в Python для реализации абстрактных методов?

Ответ: В языке Python есть еще один распространенный способ объявления абстрактных методов класса через декоратор `abstractmethod` модуля `abc`.

Вопрос 309. Что такое множественное наследование?

Ответ: Множественное наследование - это когда один дочерний класс наследуется от нескольких базовых.

Вопрос 310. Где используется множественное наследование?

Ответ: Множественное наследование используется в Django REST в миксинах. Также может быть использовано при логировании.

Вопрос 311. Важна ли последовательность базовых классов при множественном наследовании?

Ответ: последовательность ВАЖНА, т.е. есть определенный поиск в базовых классах. Это реализуется с помощью функции `super()`. В Python встроен специальный алгоритм обхода базовых классов при множественном наследовании.

Вопрос 312. Что такое MRO?

Ответ: MRO - Method Resolution Order - порядок наследования, Можно посмотреть как будет происходить наследование, с помощью магического метода `__mro__` Когда мы собираемся использовать множественное наследование структуру классов надо продумывать так, чтобы инициализаторы вспомогательных классов (2ые, 3и, 4ые) в инициализаторе имели только `self`.

Вопрос 313. Что такое коллекция `__slots__` и как она работает?

Ответ: коллекция `__slots__` - это кортеж, которая прописывается в классе и служит для ограничения числа создаваемых локальных свойств экземпляра класса, уменьшение занимаемой памяти экземпляром класса, ускорения работы с локальными свойствами и методами.

Мы можем создавать, менять значения и удалять только `x` и `y` экземпляра класса `Point`. Новые локальные свойства мы создавать не можем.

Класс с коллекцией `__slots__` не содержит коллекции `__dict__`

Вопрос 314. Какая особенность коллекции `__slots__` при наследовании?

Ответ: Дочерний класс НЕ НАСЛЕДУЕТ коллекцию `__slots__` родительского класса. Поэтому в дочернем классе создается своя коллекция `__slots__`

Вопрос 315. Что такое вложенные классы и для чего они нужны?

Ответ: Вложенные классы - это такая реализация класса, когда в него входит еще один класс, наряду с атрибутами и методами. Вложенный класс не должен использовать никакие атрибуты

внешнего класса. Вложенные классы служат для удобства программирования. Все, что можно делать через вложения также можно реализовать и без них.

Вопрос 316. Приведите устно примеры вложенных классов?

Ответ: Вложенные классы чаще всего используются во фрейворке Django под названием Meta-классы. В Django они нужны для сортировки данных. В Django метаклассы используются для связи объектов с записями в базе данных через Django ORM. Метаклассы используются, чтобы упростить некоторый функционал.

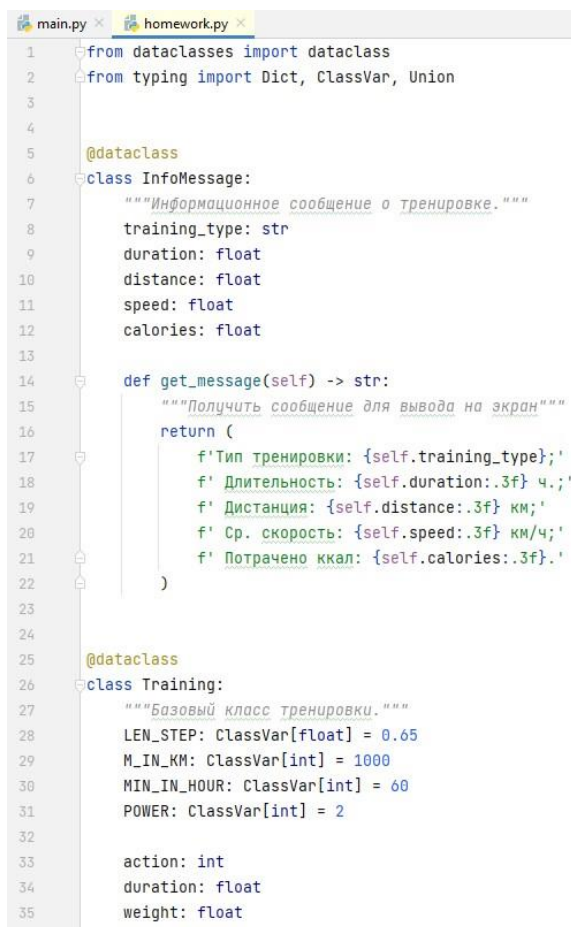
На практике (кроме Django) следует избегать применение метаклассов, т.к. они запутывают чтение кода.

Вопрос 317. Как можно упростить написание инициализаторов класса?

Ответ: воспользоваться библиотекой dataclasses

Вопрос 318. Приведите пример аннотации класса?

Ответ:



```
1 from dataclasses import dataclass
2 from typing import Dict, ClassVar, Union
3
4
5 @dataclass
6 class InfoMessage:
7     """Информационное сообщение о тренировке."""
8     training_type: str
9     duration: float
10    distance: float
11    speed: float
12    calories: float
13
14    def get_message(self) -> str:
15        """Получить сообщение для вывода на экран"""
16        return (
17            f'Тип тренировки: {self.training_type};'
18            f' Длительность: {self.duration:.3f} ч.;'
19            f' Дистанция: {self.distance:.3f} км;'
20            f' Ср. скорость: {self.speed:.3f} км/ч;'
21            f' Потрачено ккал: {self.calories:.3f}.'
22        )
23
24
25 @dataclass
26 class Training:
27     """Базовый класс тренировки."""
28     LEN_STEP: ClassVar[float] = 0.65
29     M_IN_KM: ClassVar[int] = 1000
30     MIN_IN_HOUR: ClassVar[int] = 60
31     POWER: ClassVar[int] = 2
32
33     action: int
34     duration: float
35     weight: float
```

Вопрос 319. Какой инструмент используется для аннотации типов внутри класса:

Ответ: ВСЕГДА ПОЛЬЗОВАТЬСЯ ЛИНТЕРОМ mypy. IDLE не всегда верно работает.

Вопрос 320. В обработке исключений что самое главное при написании `except`?

Ответ: После `except`, как правило, пишется то исключение, которые мы будем отлавливать. Если после `except` ничего не написать - то по умолчанию будут отлавливаться все ошибки (соответствует исключению `Exception`). САМОЕ ГЛАВНОЕ ПРИ НАПИСАНИИ ИСКЛЮЧЕНИЯ У `EXCEPT` НАДО УЧИТЫВАТЬ ИЕРАРХИЮ НАСЛЕДОВАНИЯ У КЛАССОВ ИСКЛЮЧЕНИЙ:

Вопрос 321. Какие существуют порядки записи исключений?

Ответ:

`try/except/else/finally`

`try/except/else`

`try/except/finally`

Вопрос 322. Как работает `else` при обработке исключений?

Ответ: `else` выполняется при штатном выполнении кода в блоке `try`. Т.е. если не произошло никаких `except` (ошибок). Если ошибка произошла, то `else` не сработает.

Вопрос 323. Как работает `finally` при обработке исключений?

Ответ: Выполняется всегда, вне зависимости от того, произошли ли исключения или нет

Вопрос 324. В каких ситуациях чаще всего используется блок `finally`?

Ответ: Чаще всего это работа с файлами. Открыт файл на чтение, мы в него что-то попытаемся записать. ВСЕГДА НУЖНО ОБЯЗАТЕЛЬНО ЗАКРЫВАТЬ ФАЙЛ - поэтому т.к. блок `finally` всегда выполняется в нем нужно всегда закрывать файл. Правда, для работами с файлами используется менеджер контекста.

Вопрос 325. Какая особенность `finally` в функциях?

Ответ: В функции `finally` всегда выполняется до `return`

Вопрос 326. Что такое распространение исключений?

Ответ: Распространение исключений (`propagation exceptions`) - механизм, когда мы можем обрабатывать исключения на разных уровнях стека вызова функции. В критических функциях достаточно генерировать исключение, а их обработку выполнять на глобальном уровне.

Вопрос 327. Что такое оператор `raise`?

Ответ: Оператор `raise` позволяет генерировать различные исключения, после выполнения оператора `raise` программа останавливает свою работу, если исключение не обрабатывается. После оператора `raise` следует указывать объект класса, унаследованного от `BaseException`.

Вопрос 328. Что делает оператор with?

Ответ: with as автоматически делает закрытие файла - файловый поток автоматически закрывается. with вызывает автоматически метод `__enter__`. Как только отработали все строчки или произошло какое-нибудь исключение вызывается метод `__exit__`. as здесь в роли оператора присваивания. Менеджер контекста не всегда только с файлами используется - он может использоваться с экземплярами пользовательского класса.