УРОК 30. МЕТОД КЛАССА

ЧТО ТАКОЕ МЕТОД КЛАССА И В ЧЕМ ОТЛИЧИЕ ОТ ФУНКЦИИ

Метод класса - это функция, определенная внутри класса, которая выполняет определенные операции с данными объекта класса.

Отличие метода класса от обычной функции состоит в том, что он автоматически принимает первым аргументом ссылку на экземпляр класса (self) и имеет доступ к полям и другим методам объекта.

Первый аргумент self в методах класса представляет ссылку на экземпляр класса, для которого вызывается метод. С помощью этого аргумента метод может получать доступ к полям и другим методам объекта. Self - это соглашение об именовании, и его можно заменить на любое другое допустимое имя переменной, но по соглашению разработчиков Python используется именно self.

МЕТОД __INIT__

Метод __init__ является конструктором класса и вызывается при создании нового объекта класса.

Он используется для инициализации полей объекта. В методе __init__ можно задать значения по умолчанию для полей, которые будут применяться при создании объекта.

```
Unset
class MyClass:
  def __init__(self, name):
    self.name = name

my_object = MyClass("John")
print(my_object.name)
# Результат: "John"
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Найдите ошибку в коде:

```
Unset
class Person:
  def __init__(self):
    self.name = name

bob = Person('Bob')
print(bob.name) # 'Bob'
alice = Person('Alice')
print(alice.name) # 'Alice'
```

ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ В МЕТОДАХ

Методы класса могут иметь значения по умолчанию для аргументов. Это позволяет вызывать методы с определенными значениями, но также дает возможность передать аргументы с другими значениями, если необходимо.

```Unset
class MyClass:
  def greet(self, name=""):
    if name:
      print(f"Hello, {name}!")
    else:
      print("Hello!")

my_object = MyClass()
my_object.greet()
# Результат: "Hello!"

my_object.greet("John")
# Результат: "Hello, John!"
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Объясните, чем является name и greeting, str и "Привет" в данном фрагменте кода?

```
Unset
def greet(name: str, greeting: str = "Привет") -> str:
  return f"{greeting}, {name}!"
```

СТАТИЧЕСКИЕ ПОЛЯ КЛАССА

Статические поля класса - это поля, которые принадлежат классу, а не экземпляру класса.

Они могут быть доступны через имя класса и не требуют создания экземпляра класса.

```
Unset
class MyClass:
  static_field = "Static Field"

print(MyClass.static_field)
# Результат: "Static Field"
```

При обращении к статическому полю класса можно использовать имя класса или имя объекта класса. Однако рекомендуется использовать имя класса, чтобы подчеркнуть, что это статическое поле и не зависит от конкретного экземпляра.

```
Unset
class MyClass:
  static_field = "Static Field"

my_object = MyClass()

print(MyClass.static_field)
# Результат: "Static Field"

print(my_object.static_field)
# Результат: "Static Field"
```

МЕТОДЫ __STR__ И __REPR__

Магические методы __str__ и __repr__ предоставляют возможность определить строковое представление объекта класса.

Метод __str__ используется для создания "красивого" и информативного представления объекта, которое может быть выведено на печать/

Метод __repr__ используется для создания представления объекта, которое позволяет точно воссоздать объект.

Unset

```
class MyClass:
  def __init__(self, name):
    self.name = name

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

  def __repr__(self):
    return f"MyClass(name='{self.name}')"

my_object = MyClass("John")
print(my_object)
# Результат: "MyClass: John"

print(repr(my_object))
# Результат: "MyClass(name='John')"
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Что происходит при выполнении этого фрагмента кода?

```
Unset
class Cat:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f'Cat(name={self.name}, age={self.age})'
```

В чем отличие с этим фрагментом?

```
Unset
class Cat:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f'Cat(name={self.name}, age={self.age})'

  def __repr__(self):
    return f'Cat(name={self.name!r}, age={self.age!r})'
```

ФУНКЦИЯ ISINSTANCE

Функция isinstance позволяет проверить, является ли объект экземпляром определенного класса или его потомком.

Она возвращает значение True, если объект является экземпляром класса, и False в противном случае.

```
Unset
class MyClass:
  pass

class MySubClass(MyClass):
  pass

my_object = MyClass()
my_sub_object = MySubClass()

print(isinstance(my_object, MyClass))
# Результат: True

print(isinstance(my_sub_object, MyClass))
# Результат: True

print(isinstance(my_object, MySubClass))
# Результат: False
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Что вернет следующий фрагмент кода?

```
Unset
x = 10
print(isinstance(x, int))
```

СОЗДАНИЕ КЛАССА

При создании своего класса необходимо определить его имя, поля, методы и другие атрибуты, которые необходимы для его функционирования. Внутри класса можно использовать методы для изменения состояния объекта и выполнения операций с данными. Также можно использовать наследование для создания подклассов, которые наследуют поля и методы родительского класса и могут добавлять свою уникальную функциональность.

Unset

```
class MyClass:
  def __init__(self, name):
    self.name = name

  def greet(self):
    print(f"Hello, {self.name}!")

my_object = MyClass("John")
my_object.greet()
# Результат: "Hello, John!"
```

ПРАКТИЧЕСКАЯ РАБОТА

1. Создать класс Person с полями имя, год рождения, рост, вес и методами для вычисления возраста, индекса массы тела.
   Создать конструкторы от разного числа аргументов.
   Переопределить методы __str__, __repr__.

2. Написать класс Circle, который задает круг по радиусу и может вычислять площадь и длину окружности.
   Продумать интерфейс и методы.

ПОЛЕЗНЫЕ МАТЕРИАЛЫ

1. Классы в Python
2. Python class: как работать с классами, разбор на примерах