**28.8 Практическая работа**

Цели практической работы

Отработать:

* применение множественного наследования;
* создание и применение примесей и абстрактных классов;
* применение декораторов setter и property, а также classmethod для более правильного и чистого кода.

Закрепить понимание порядка разрешения методов в Python.

Что входит в работу

* Задача 1. Работа с файлом 2.
* Задача 2. Математический модуль.
* Задача 3. Дата.

Задача 1. Работа с файлом 2

Что нужно сделать

Реализуйте модернизированную версию контекст-менеджера File:

* теперь при попытке открыть несуществующий файл менеджер должен автоматически создавать и открывать этот файл в режиме записи;
* на выходе из менеджера должны подавляться все исключения, связанные с файлами.

Что оценивается

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений приватных атрибутов используются сеттеры и геттеры с соответствующими декораторами.
  + Для создания нового класса на основе уже существующего используется наследование.
  + Для статических и классовых методов используется декоратор classmethod.
* Сообщения о процессе получения результата осмысленны и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.

Задача 2. Математический модуль

Что нужно сделать

Вася использует в своей программе очень много различных математических вычислений, связанных с фигурами. Например, нахождение их площадей или периметров. Поэтому, чтобы не захламлять код огромным количеством функций, он решил выделить для них отдельный класс, подключить как модуль и использовать по аналогии с модулем math.

Реализуйте класс MyMath, состоящий как минимум из следующих методов (можете бонусом добавить и другие методы):

* вычисление длины окружности,
* вычисление площади окружности,
* вычисление объёма куба,
* вычисление площади поверхности сферы.

**Пример основного кода:**

res\_1 = MyMath.circle\_len(radius=5)  
res\_2 = MyMath.circle\_sq(radius=6)  
print(res\_1)  
print(res\_2)

**Результат:**

31.41592653589793

113.09733552923255

Что оценивается

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений приватных атрибутов используются сеттеры и геттеры с соответствующими декораторами.
  + Для создания нового класса на основе уже существующего используется наследование.
  + Для статических и классовых методов используется декоратор classmethod.
* Формат вывода соответствует примеру.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.

Задача 3. Дата

Что нужно сделать

Реализуйте класс Date, который должен:

* проверять числа даты на корректность;
* конвертировать строку даты в объект класса Date, состоящий из соответствующих числовых значений дня, месяца и года.

Оба метода должны получать на вход строку вида ‘dd-mm-yyyy’.

При тестировании программы объект класса Date должен инициализироваться исключительно через метод конвертации, например:

date = Date.from\_string('10-12-2077')

Неверный вариант: date = Date(10, 12, 2077)

**Пример основного кода:**

date = Date.from\_string('10-12-2077')  
print(date)  
print(Date.is\_date\_valid('10-12-2077'))  
print(Date.is\_date\_valid('40-12-2077'))

**Результат:**  
День: 10    Месяц: 12    Год: 2077  
True  
False

Что оценивается

* Результат вычислений корректен.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
* Для получения и установки значений приватных атрибутов используются сеттеры и геттеры с соответствующими декораторами.
* Для создания нового класса на основе уже существующего используется наследование.
* Для статических и классовых методов используется декоратор classmethod.
* Формат вывода соответствует примеру.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.

Что оценивается в практической работе

* Практическая работа сдана через GitLab.
* Структура папок и файлов репозитория соответствует репозиторию python\_basic.
* Все задачи выполнены в соответствующих папках и файлах main.py.
* Описания коммитов осмысленны и понятны: 111, done, «я сделалъ» — неверно; added m15 homework, 14.3 fix: variables naming — верно.
* Использованы именованные индексы, не просто i (подробнее — в видео 7.2).
* Использованы правильные числа, без дополнительных действий со стороны пользователя, без +1 (подробнее — в видео 7.4).
* Правильно оформлен input, без пустого приветствия для ввода (подробнее — в видео 2.3).
* Переменные и функции имеют значащие имена, не только a, b, c, d (подробнее — в видео 2.3).
* Есть пробелы после запятых и при бинарных операциях.
* Нет пробелов после имён функций и перед скобками: print (), input () — неверно; print() — верно.
* Правильно оформлены блоки if-elif-else, циклы и функции; отступы одинаковы во всех блоках одного уровня.
* Все входные и выходные файлы называются так, как указано в заданиях.
* Работа с файлами осуществляется с помощью контекстного менеджера with.
* Для обработки исключений используются блоки try-except.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений приватных атрибутов используются сеттеры и геттеры с соответствующими декораторами.
  + Для создания нового класса на основе уже существующего используется наследование.
  + Для статических и классовых методов используется декоратор classmethod.
* Если классы вынесены в отдельный модуль, то импортируются определённые классы (запись вида from garden import \* считается плохим тоном).
* Классы и методы/функции имеют прописанную документацию (хотя бы минимальную).
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.
* Во всех декораторах используется functools.wraps.