**Модуль 4. «Основы ООП. Обработка файлов. Мультимедиа»**

Выполнил Мелега Алексей Валерьевич

**Урок 33-34 Итоговая аттестация по модулю 4. Файлы, библиотеки, мультимедиа. Выполнение задания по обработке файлов разного типа средствами Python**

**1. Космолёт на Луну**

Луна не безжизненный каменистый шар, а источник ценнейшего ресурса и объект потенциального освоения. Обнаруженная на Луне вода может использоваться в системе жизнеобеспечения и оранжереях будущих обитаемых лунных баз, а также для лунного производства, в том числе для выработки компонентов ракетного топлива — кислорода и водорода. Последнее обстоятельство открывает путь для полётов с Луны в дальний космос.

Представьте, что на Луну отправляется группа космических кораблей с переселенцами на борту. Луне суждено будет стать их домом на некоторое продолжительное время, если не произойдёт непредвиденных обстоятельств. На начальном этапе важно благополучно довезти людей и грузы в целостности и сохранности.

Создайте класс **SpaceShip**, экземпляр ss которого инициализируется с аргументами списком содержимого кают и трюмов (id из базы данных в виде целых чисел) и видом (буква латинского алфавита в верхнем регистре, начиная с А). Класс **SpaceShip** должен реализовывать следующую функциональность:

* **str(ss)** выводит свое строковое представление в виде

**«Space Ship** вид корабля: длина списка»;

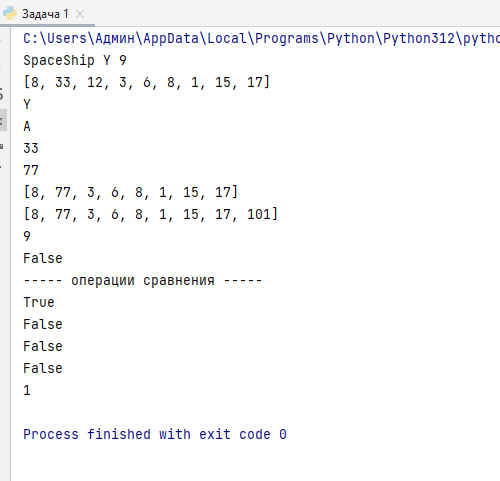
* **ss.values()** возвращает список содержимого;
* **ss.rang()** возвращает вид корабля;
* **ss += value** — повышает вид корабля на value позиций в алфавите, циклически возвращаясь к началу, при превышении длины алфавита;
* **ss[i]** — возвращает значение;
* **ss[i] = value** — заменяет значение;
* **ss.deli(i)** — удаляет элемент из списка по индексу, остальные сдвигаются вперёд;
* **ss.app(value)** — добавляет элемент в конец списка;
* **len(ss)** — возвращает длину списка содержимого;
* **value in ss** — проверяет наличие элемента в списке;
* корабли можно сравнивать:
* **ss1 == ss2** − равны их виды и списки содержимого кают равной длины;
* **ss1 > ss2** − равны их виды и список содержимого кают ss1 длиннее спискf содержимого кают ss2;
* **ss1 < ss2** − равны их виды и список содержимого кают ss2 длиннее списк содержимого кают ss1;
* **ss1 != ss2** − не равны их виды или не равны длины списков;
* **ss(x)** — возвращает минимальное значение с начала списка и до индекса x, включая его.

Код задачи:

**from** space\_ship **import** SpaceShip  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 ss = SpaceShip([8, 33, 12, 3, 6, 8, 1, 15, 17], **'Y'**)  
 print(str(ss))  
 print(ss.values())  
 print(ss.rang())  
 *# Меняем ранг* ss += 2  
 print(ss.rang())  
 print(ss[1])  
 *# Меняем номер в списке* ss[1] = 77  
 print(ss[1])  
 ss.deli(2)  
 print(ss.numb)  
 ss.app(101)  
 print(ss.numb)  
 print(len(ss))  
 print(12 **in** ss)  
 *# Сравнения 2-х кораблей* ss1 = SpaceShip([11, 6, 17, 5, 55, 7, 23, 12, 9], **'Y'**)  
 ss2 = SpaceShip([8, 33, 12, 3, 6, 8, 12, 15, 17], **'Y'**)  
 print(**"----- операции сравнения -----"**)  
 print(ss1 == ss2)  
 print(ss1 > ss2)  
 print(ss1 < ss2)  
 print(ss1 != ss2)  
 print(ss(6))

Класс space\_ship.py

**class** SpaceShip():  
 **def** \_\_init\_\_(self, numb, view):  
 self.numb = numb  
 self.view = view  
  
 **def** \_\_str\_\_(self):  
 **return f'{**self.\_\_class\_\_.\_\_name\_\_**} {**self.view**} {**len(self.numb)**}'  
  
 def** values(self):  
 **return** self.numb  
  
 **def** rang(self):  
 **return** self.view  
  
 **def** \_\_iadd\_\_(self, other):  
 new = ord(self.view) + other  
 **if** new > 90:  
 new = new - 91 + 65  
 self.view = chr(new)  
 **return** self  
  
 **def** \_\_getitem\_\_(self, item):  
 **return** self.numb[item]  
  
 **def** \_\_setitem\_\_(self, key, value):  
 self.numb[key] = value  
  
 **def** deli(self, key):  
 **del** self.numb[key]  
  
 **def** app(self, value):  
 self.numb.append(value)  
  
 **def** \_\_len\_\_(self):  
 **return** len(self.numb)  
  
 **def** \_\_contains\_\_(self, item):  
 **return** item **in** self.numb  
  
 **def** \_\_eq\_\_(self, other):  
 **return** self.view == other.view **and** len(self.numb) == len(other.numb)  
  
 **def** \_\_gt\_\_(self, other):  
 **return** self.view == other.view **and** len(self.numb) > len(other.numb)  
  
 **def** \_\_lt\_\_(self, other):  
 **return** self.view == other.view **and** len(self.numb) > len(other.numb)  
  
 **def** \_\_ne\_\_(self, other):  
 **return** self.view != other.view **or** len(self.numb) != len(other.numb)  
  
 **def** \_\_call\_\_(self, \*args, \*\*kwargs):  
 x = args[0]  
 **return** min(self.numb[:x + 1])



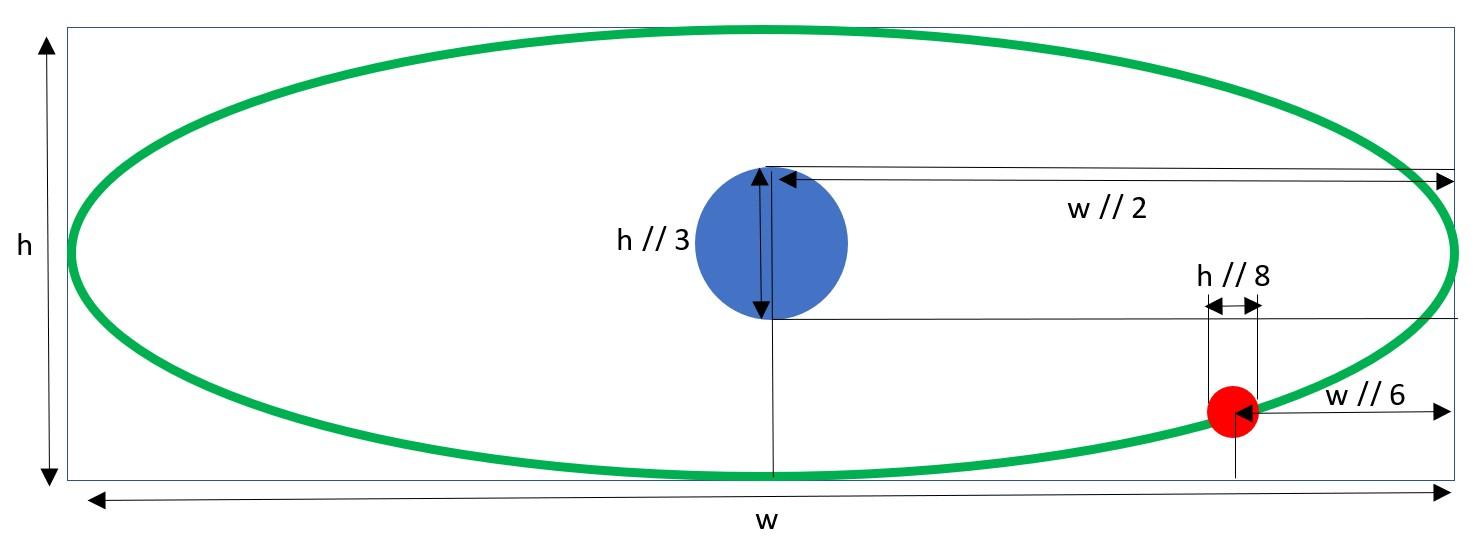
**2. Две планеты**

Две планеты из далёкой галактики взаимодействуют примерно также, как Луна и Земля. Планета Альфа вращается по орбите вокруг планеты Бетта по сильно вытянутой орбите, в результате чего на ней наблюдаются значительные колебания температуры.

Напишите класс **Planet(Image)**, у которого есть метод **orbit(xywh, fill)**, строящий взаимное расположение планеты Альфа и планеты Бетта, а также линию орбиты. Описание аргументов метода **orbitа**:

* **xywh** — кортеж из 4 целых чисел: **x** и **y** — координаты левого верхнего угла рисунка, **w** и **h** — размеры, относительно которых определяются остальные параметры рисунка;
* **fill** — кортеж из 3 цветов в 16-ричном формате, соответственно, цвет планеты Бетта, цвет планеты Альфа и цвет линии орбиты.

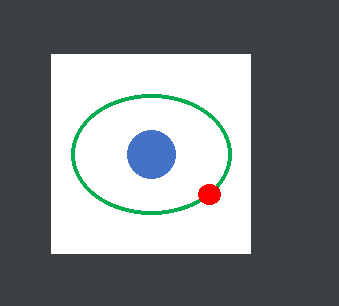
Орбита представляет собой эллипс толщиной 3 пикселя, планеты — круги. Все размеры и расположение указаны на рисунке:



**Пример ввода**

|  |
| --- |
| **from PIL import Image**    **img = Image.new('RGB', (200, 200), '#000000') drw = Planet(img) drw.orbit((20, 40, 160**, **120), \ ('#ffffff', '#dddddd', '#999999'))**  **img.save('result.png')** |

from PIL import Image, ImageDraw  
  
  
class Planet():  
 def \_\_init\_\_(self, img):  
 self.img = img  
  
 def orbit(self, xywh, fill):  
 x, y, w, h = xywh  
 draw = ImageDraw.Draw(self.img)  
 draw.ellipse((x, y, w + x, h + y), outline=fill[2], width=4)  
 draw.ellipse((w // 2 - h // 5 + x,  
 h // 2 - h // 5 + y,  
 w // 2 + h // 5 + x,  
 h // 2 + h // 5 + y), fill=fill[0], width=0)  
 draw.ellipse((w - w // 6 - h // 16 + x,  
 h // 4 \* 2 + h // 2 - h // 11 + y // 2,  
 w - w // 6 + h // 8 + x,  
 h // 4 \* 2 + h // 2 + h // 11 + y // 2), fill=fill[1], width=0)  
  
  
img = Image.new('RGB', (200, 200), '#ffffff')  
drw = Planet(img)  
drw.orbit((20, 40, 160, 120), ('#4472c6', '#fe0000', '#00ac50'))  
img.save('result.png')



**3. Без вредных привычек**

Создайте программу, которая составит таблицу вредных привычек, от которых удалось быстро избавиться. В файле in.json находится список словарей с ключами:

привычка; продолжительность; вредность; проявление

Необходимо выбрать привычки, проявление которых не больше 2, и записать их в файл out.csv в порядке увеличения вредности, в случае одинаковой вредности записать в порядке алфавита. Заголовки данных при выводе сделайте такими:

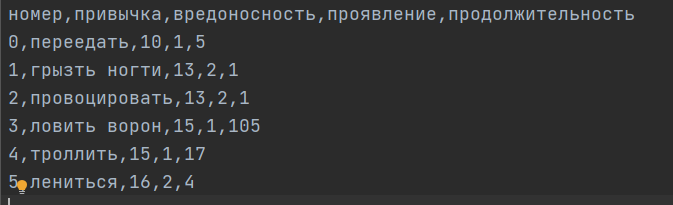
номер; привычка; вредоносность; проявление; продолжительность

В качестве разделителей используйте точку с запятой.

**Файл in.json**

|  |
| --- |
| <https://disk.yandex.ru/d/zJGe1sq8D2_i8Q> |

import json, csv  
  
# Открываем json  
file\_name = "in.json"  
with open(file\_name, encoding='utf-8') as json\_file:  
 person = json.load(json\_file)  
data = list(filter(lambda x: x["проявление"] <= 2, person))  
data.sort(key=lambda x: (x["вредность"], x["привычка"]))  
  
# Записываем в csv  
with open('out.csv', 'w', newline='', encoding='utf-8') as csvfile:  
 fieldnames = ['номер', 'привычка', 'вредоносность', 'проявление', 'продолжительность']  
 writer = csv.DictWriter(csvfile, fieldnames=fieldnames)  
 writer.writeheader()  
 for i, elem in enumerate(data):  
 writer.writerow({'номер': i,  
 'привычка': elem['привычка'],  
 'вредоносность': elem['вредность'],  
 'проявление': elem['проявление'],  
 'продолжительность': elem['длительность']})  
print('Данные записаны.')



**3. Минипроект Self control**

Многим знакома проблема самоконтроля во время игры за компьютером. Игра способна так увлечь, что забываешь про дела, даже если они срочные. Предлагаем вам создать программу таймер., которая сначала запросит временной интервал в формате *часы:минуты:секунды*, затем, спустя указанное время воспроизведёт звук из файла формата .mp3.

Для работы программы вам понадобится класс datetimeиз модуляdatetime [(https://pythonworld.ru/moduli/modul-datetime.html)](https://pythonworld.ru/moduli/modul-datetime.html), и функция playsound из модуля playsound[(https://pypi.org/project/playsound/)](https://pypi.org/project/playsound/).

**Файл sound.mp3**

|  |
| --- |
| <https://disk.yandex.ru/d/NgUI1Nbl0vkrlQ> |