ВМК МГУ

Задание 1. Основы Python

Общее описание задания

Курс по методам машинного обучения, 2022-2023, Мишустина Маргарита

1 Характеристики задания

- **Длительность:** 1 неделя
- **Юнит-тестирование:** 22 балла; можно сдавать после дедлайна со штрафом в 40%; Публичная часть
- **Почта:** ml.cmc@mail.ru
- **Темы для писем на почту:** ВМК.ML[Задание 1][unit-tests]

2 Описание задания

Прежде, чем приступить к заданию

- Убедитесь, пожалуйста, что у вас стоит виртуальное окружение с `Python3` и установленными библиотеками нужных версий отсюда, а также стоит `jupyter-lab` или `jupyter-notebook`.
- Убедитесь, что у вас есть доступ к Google Colab

Внимание!
К данному заданию приложен очень информативный ноутбук (ipynb), содержащий в себе большое количество информации про python.

- Скачать его можно вот отсюда по ссылке
- Или в проверяющей системе, нажав на «`Дополнительные файлы для решения`».

Кроме того, в данном задании 7 задач; задачи 1-5, 6 и 7 находятся в *отдельных вкладках*. Их условия будут продублированы в данном файле.

Как можно локально протестировать ваше решение
Для выполнения задания нужно скачать из *проверяющей системы:*

1. Скрипт для тестирования `run.py` (вкладка «`скрипт для тестирования`»)
2. Архив с тестами (вкладка «`публичные тесты`»)
3. Архив с шаблонами решения (вкладка «`шаблон решения`»)

Внимание!
Для каждого из заданий (1-5, 6, 7) эти файлы **разные**! Кладите файлы из разных заданий в разные папки, чтобы их не перепутать!
Для заданий 1-5
Зайдите в задание `Введение в Python.1-5`. Разархивируйте архив с тестами в папку `python_intro_public_test` и рядом с этой папкой положите тестовый скрипт `run.py`. Также Разархивируйте архив с шаблонами решения и положите их в так же рядом с `run.py`. В файлах с шаблонами решений необходимо написать необходимые функции, а затем, после успешного локального тестирования, сдать их в проверяющую систему. Для запуска тестов вам понадобится библиотека `pytest`.
Таким образом, в произвольной директории (назовем ее Python_task1_5) должны находиться файлы `run.py` и решения задач `task15.py` и тд. Там же должна быть создана директория `python_intro_public_test`,

содержимое которой должно представлять собой распакованный архив с публичными тестами, то есть содержать `01_unittest_task1_input` и тд:

> Python_task1_5

> run.py

> task15.py

> python_intro_public_test

> 01_unittest_task1_input

Как запускать тесты — будет расписано ниже в каждой задаче.

Замечание: Запрещается пользоваться библиотеками, импорт которых не объявлен в файле с шаблонами функций (актуально для заданий 1 - 6)

Замечание: Задания, в которых есть решения, содержащие в каком-либо виде взлом тестов, дополнительные импорты и прочие нечестные приемы, будут автоматически оценены в 0 баллов без права пересдачи задания (актуально для заданий 1 - 6, для задания 7 см. правила в соответствующем файле).

3 Задача 1 (2 балла)

Формулировка: Требуется написать функцию hello(x), которая принимает 1 параметр, равный по умолчанию None.

Если в качестве этого параметра передается пустая строка или происходит вызов без аргументов, то функция возвращает строку «Hello!», иначе, функция выводит «Hello, x!», где x - значение параметра функции.

Пояснение: Файл с решением task15.py должен содержать функцию hello(x).

Пример работы: После знака комментария указан желаемый вывод.

```
from task15 import hello

print(hello()) # Hello!
print(hello('')) # Hello!
print(hello('Masha')) # Hello, Masha!
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task15.py.

$ python run.py unittest task1

4 Задача 2 (3 балла)

Формулировка: Требуется написать функцию int_to_roman(x), которая принимает натуральное число x : $x \in [1, 3999]$, записанное арабскими цифрами, и конвертирует его в такое же число, записанное римскими цифрами.

Немного о переводе арабских чисел в формат римских:

- 2 записывается как II, то есть две единицы складываются вместе.
- 12 записывается как XII, что означает 10 + 2 или X + II.
- 27 записывается как XXVII, то есть 20 + 5 + 2 или XX + V + II.

Римские цифры записывают от наибольшего к наименьшему слева направо. Однако цифра, обозначающая четыре, не записывается как IIII. Вместо этого число 4 записывается как IV. Поскольку единица стоит перед пятеркой, мы вычитаем ее, получая четыре.

Тот же принцип применим и к числу девять, которое записывается как IX. Существует шесть случаев, когда используется вычитание:

- I помещается перед V (5) и X (10), чтобы сделать 4 и 9.
- X помещается перед L (50) и C (100), чтобы получить 40 и 90.
- C помещается перед D (500) и M (1000), чтобы получить 400 и 900.

Пояснение: Файл с решением task15.py должен содержать функцию int_to_roman(x).
Пример работы: После знака комментария указан желаемый вывод.

```
from task15 import int_to_roman

print(int_to_roman(3)) # III
print(int_to_roman(54)) # LIV
print(int_to_roman(1996))  # MCMXCVI
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task15.py.

$ python run.py unittest task2

5 Задача 3 (3 балла)

Формулировка: Требуется написать функцию longest_common_prefix(x), которая принимает список строк и возвращает наибольший общий префикс для строк в списке строк. Пробельные символы в начале строки не учитывать. Изменять входной список нельзя. Если такого префикса нет, требуется вернуть пустую строку. Если на вход поступает пустой список, то следует вернуть пустую строку.

Пояснение: Файл с решением task15.py должен содержать функцию longest_common_prefix(x).
Пример работы: После знака комментария указан желаемый вывод.

```
from task15 import longest_common_prefix

print(longest_common_prefix(["flo","flow","flight"])) # fl
print(longest_common_prefix(["dog","racecar","car"])) #
print(longest_common_prefix([" flo","flow","fl"])) # fl
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task15.py.

$ python run.py unittest task3

6 Задача 4 (3 балла)

Формулировка: У студента есть кредитная карта банка, на который лежат деньги в долларах. Требуется реализовать класс BankCard со следующим интерфейсом:

```
a = BankCard(total_sum, balance_limit)
"""
total_sum - общая сумма денег на карте у студента в начальный
момент времени, balance_limit - максимальное количество раз,
которое можно посмотреть баланс (если не задано, то нет
ограничений; может быть только натуральным числом)
"""
a(sum_spent)
"""
sum_spent - сумма, которую студент хочет потратить; при таком
обращении sum_spent вычитается из текущей total_sum; если
такой суммы на карте нет, требуется бросить исключение
ValueError и напечатать "Not enough money to spend sum_spent dollars.".
Если попытка снятия денег была успешной, следует написать: You spent sum_spent dollars.
"""
print(a)
"""
при вызове функции print от объекта класса должно выводиться
следующее сообщение "To learn the balance call balance."
a.balance # при таком вызове количество вызовов баланса карты
должно увеличиваться на 1, а возвращаться должна total_sum; если
лимит операций по количеству просмотров баланса превышен,
требуется бросить исключение ValueError и напечатать "Balance check limits exceeded."
"""
a.put(sum_put)
"""
положить sum_put долларов на карту;
```

```
также следует написать: "You put sum_put dollars."
"""
b = BankCard(total_sum_1)
a + b
"""
мердж капиталов :) должен возвращаться объект класса BankCard,
в котором сложены балансы, а из balance_limit выбирается максимум
"""
```

Пояснение: Файл с решением task15.py должен содержать реализацию класса BankCard.
Пример работы: После знака комментария указан желаемый вывод.

```
from task15 import BankCard

a = BankCard(10, 2)
print(a.balance) # 10
print(a.balance_limit) # 1
a(5) # You spent 5 dollars.
print(a.total_sum) # 5
print(a) # To learn the balance call balance.
print(a.balance) # 5
try:
    a(6) # Not enough money to spend 6 dollars.
except ValueError:
    pass
a(5) # You spent 5 dollars.
try:
    a.balance # Balance check limits exceeded.
except ValueError:
    pass
a.put(2) # You put 2 dollars.
print(a.total_sum) # 2
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task15.py.

$ python run.py unittest task4

7 Задача 5 (3 балла)

Формулировка: Целое положительное число называется простым, если оно имеет ровно два различных делителя, то есть делится только на единицу и на само себя. Например, число 2 является простым, так как делится только на 1 и 2. Число 4, например, не является простым, так как имеет три делителя – 1, 2, 4. Также простым не является число 1.

Требуется реализовать функцию-генератор primes(), которая будет генерировать простые числа в порядке возрастания, начиная с числа 2.

Пояснение: Решение должно содержать файл task15.py с функцией-генератором primes().
Пример работы: В комментарии после вызова функции указан желаемый вывод.

```
from task15 import primes

for i in primes():
    print(i)
    if i > 3:
        break
"""
2
3
5
"""
l = itertools.takewhile(lambda x : x <= 17, primes())
print(list(l)) # [2, 3, 5, 7, 11, 13, 17]
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task15.py.

$ python run.py unittest task5

8 Задача 6 (3 балла)

Формулировка: Когда студент прочитал учебник по линейной алгебре и аналитической геометрии, ему стало интересно, сколько слов и в каком количестве встречается в этой книге.

Требуется написать функцию check(s, filename), которая принимает на вход строку – последовательность слов, разделенных пробелом и имя файла; слова состоят из строчных и прописных букв латинского алфавита, а разделяются пробельными символами (ввод считать корректным). Функция должна вывести в файл для каждого уникального слова в этой строке число его повторений (без учёта регистра) в формате "слово количество"(см. пример вывода). Гарантируется, что строка содержит минимум 1 слово.

Слова выводить нужно по алфавиту, каждое уникальное слово должно выводиться только один раз.

Пояснение: Решение должно содержать файл task6.py с функцией check(s, filename).

Пример работы: В многострочном комментарии после вызова функции указано содержимое файла file.txt.

```
from task6 import check

check("a aa abC aa ac abc bcd a", "file.txt")
"""
a 2
aa 2
abc 2
ac 1
bcd 1
"""
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task6.py.

$ python run.py python_intro_public_test

9 Задача 7 (5 баллов)

Формулировка: Это задание – конкурс! Его цель заключается в том, чтобы написать как можно более короткое (по символам) решение задачи.

При подсчете пробельные символы будут автоматически удаляться, поэтому игнорировать символы табуляции и пробелы, которые сделают код более читаемым, не нужно!

Дана строка, состоящая из символов ASCII. Словом называется комбинация из букв латинского алфавита (строчных и заглавных). Требуется написать функцию find_shortest(l), которая позволяет найти длину самого короткого слова в строке.

Пояснение: Решение должно содержать файл task7.py с функцией find_shortest(l).

Пример работы: После знака комментария указан желаемый вывод.

```
from task7 import find_shortest

print(find_shortest('9230847;;;;1;;;++++_______a')) # 1
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task7.py.

$ python run.py test task7