Курс по методам машинного обучения

Практическое задание № 1

Основы Python

Мишустина Маргарита

Прежде, чем приступить к заданию

Убедитесь, пожалуйста, что у вас стоит виртуальное окружение с `Python3`, а также стоит `jupyter-lab` или `jupyter-notebook`. Зайдите в одно из первых заданий в проверяющей системе (Введение в Python [1-5]) и скачайте оттуда файл во вкладке «`Скрипт для настройки тестового окружения`». Поставьте, пожалуйста, в свое окружение все библиотеки, которые перечислены после комманды `pip3 install`, а также библиотеку `pytest`, которая пригодится в этом и последующих заданиях.

Внимание!

К данному заданию приложен очень информативный ноутбук (ipynb), содержащий в себе большое количество информации про python. *Скачать* его можно из проверяющей системы, нажав на «`Дополнительные файлы для решения`». Кроме того, в данном задании 7 задач; задачи 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»)

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 должны находиться файлы run.py и решения задач task1.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:

> run.py

> task1.py

> python_intro_public_test

> 01_unittest_task1_input

Как запускать тесты — будет расписано ниже в каждой задаче.

1 Задача 1 (4 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: Решение должно содержать файл task1.py с функцией hello(x).

Пример работы: После знака комментария указан желаемый вывод.

```
from task1 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 task1.py.

$ python run.py unittest task1

2 Задача 2 (4 балла)

Формулировка: Требуется написать функцию is_palindrome(x), принимающую целое положительное число и проверить, является ли оно палиндромом, т. е. совпадает ли первая цифра с последней, вторая — с предпоследней и т. д. Представлять число в виде последовательности (строки, списка и т. п.) нельзя. Вывести YES или NO соответственно. Ведущие нули не учитывать (числа, заканчивающиеся на 0, кроме числа 0, — автоматически не палиндромы).

Пояснение: Решение должно содержать файл task2.py с функцией is_palindrome(x).

Пример работы: После знака комментария указан желаемый вывод.

```
from task2 import is_palindrome

print(is_palindrome(121)) # YES
print(is_palindrome(120)) # NO
print(is_palindrome(10))  # NO
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task2.py.

$ python run.py unittest task2

3 Задача 3 (4 балла)

Формулировка: Требуется написать функцию longestCommonPrefix(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.

Пояснение: Решение должно содержать файл task3.py с функцией longestCommonPrefix(x).

Пример работы: После знака комментария указан желаемый вывод.

```
from task3 import longestCommonPrefix

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

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task3.py.

$ python run.py unittest task3

4 Задача 4 (4 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)
# total_sum - общая сумма денег на карте у студента в начальный момент времени
a(sum_spent)
"""
sum_spent - сумма, которую  студент хочет потратить;
при вызове данной функции sum_spent вычитается из текущей total_sum;
если такой суммы на карте нет, требуется сгенерировать исключение ValueError
и напечатать Not enough money to spent sum_spent dollars.
Если попытка снятия денег была успешной,  следует написать:
You spent sum_spent dollars. total_sum dollars are left.
"""
print(a)
"""
при вызове функции print от объекта класса должно выводиться
следующее сообщение
To learn the balance you should put the money on the card,
spent some money or get the bank data.
The last procedure is not free and costs 1 dollar.
"""
a.balance
"""
при таком вызове текущий баланс карты должен уменьшаться на 1,
а возвращаться должна total_sum уже без учета доллара;
если баланс проверить невозможно (недостаточно средств), требуется сгенерировать
исключение ValueError и напечатать Not enough money to learn the balance.
"""
a.put(sum_put)
"""
положить sum_put долларов на карту; также следует написать:
You put sum_put dollars. total_sum dollars are left.
"""
```

Пояснение: Решение должно содержать файл task6.py с реализацией класса BankCard.

Пример работы: После знака комментария указан желаемый вывод.

```
from task4 import BankCard


a = BankCard(10)
print(a.total_sum) # 10
a(5) # You spent 5 dollars. 5 dollars are left.
print(a.total_sum) # 5
print(a)
"""
To learn the balance you should put the money on the card,
spent some money or get the bank data.
The last procedure is not free and costs 1 dollar."""
print(a.balance) # 4
try:
    a(5) # Not enough money to spent 5 dollars.
except ValueError:
    pass
a(4) # You spent 4 dollars. 0 dollars are left.
try:
    a.balance # Not enough money to learn the balance.
except ValueError:
    pass
a.put(2) # You put 2 dollars. 2 dollars are left.
print(a.total_sum) # 2
print(a.balance) # 1
```

Запуск тестов: Следующую команду нужно ввести в командной строке из директории, содержащей файл run.py и task4.py.

$ python run.py unittest task4

5 Задача 5 (4 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 task5.py с функцией-генератором primes().

Пример работы: В комментарии после вызова функции указан желаемый вывод.

```
from task5 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 task5.py.

$ python run.py unittest task5

6 Задача 6 (5 баллов)

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

Требуется написать функцию check(s, filename), которая принимает на вход строку – последовательность слов, разделенных пробелом и имя файла; 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).

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

7 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!

Дан список вложенности ровно 2. Требуется написать функцию process(l), которая возвращает отсортированные по убыванию уникальные квадраты чисел, содержащихся в этом списке.

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

Пример работы: После знака комментария указан желаемый вывод.

```
from task7 import process

print(process([[1, 2], [3], [4], [3, 1]])) # [16, 9, 4, 1]
```

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

$ python run.py test task7