Contents

1 Basic Test Results

```
Starting tests...
Mon Nov 23 13:02:08 IST 2015
52b1f19cefd37123ffd6b96e64959da7596d2d23  -

Missing required file: AUTHORS
result_code    missing_file    AUTHORS    1
Extra file submitted: AUTHORS.txt
result_code    extra_file    AUTHORS.txt    1

Archive:  /tmp/bodek.rfUboC/intro2cs/ex5/elinorperl/presubmission/submission
 extracting: src/AUTHORS.txt
  inflating: src/ex5.py
  inflating: src/README

Testing README...
Done testing README...

Listing AUTHORS...
Missing (or empty) AUTHORS file

Running presubmit tests...
13 passed tests out of 13
result_code    ex5    13    1
Done running presubmit tests

Tests completed

Additional notes:

There will be additional tests which will not be published in advance.
```

2 README

```
elinorperl
329577464
Elinor Perl

Talya.adams
312255243
Talya Adams

We discussed the exercise with David Barban

=========================================
=  README for ex5: Comparing Supermarkets  =
=========================================


==================
=  Description:  =
==================
The combined use of dictionaries and XML, we built a program that would compare the items and
prices of the supermarket, taking into account if the item is in stock and giving a penalty
depending on the items lacking in the store, therefore directing the user to the most economic
supermarket containing the maximum amount of the items they desire.

======================
=  Special Comments  =
======================

We used stackoverflow.com
```

3 AUTHORS.txt

```
Talya.adams, elinorperl
```

4 ex5.py

```python
import xml.etree.ElementTree as ET
PENALTY = 1.25


def get_attribute(store_db, ItemCode, tag):
    """"
    Returns the attribute (tag)
    of an Item with code: ItemCode in the given store
    """
    item = store_db[ItemCode]
    return item[tag]


def string_item(item):
    """
    Textual representation of an item in a store.
    Returns a string in the format of '[ItemCode] (ItemName)'
    In this function, we took the item code and name and made it a string,
    making it compatible to the Hebrew text.
    """
    code = item["ItemCode"]
    name = item["ItemName"]
    return ("["+code+"]\t{"+name+"}")


def string_store_items(store_db):
    """
    Textual representation of a store.
    Returns a string in the format of:
    string representation of item1
    string representation of item2
    """
    rep = ""
    for key in store_db:
        item = store_db[key]
        rep += string_item(item) + "\n"
    return rep


def read_prices_file(filename):
    """
    Read a file of item prices into a dictionary.  The file is assumed to
    be in the standard XML format of "misrad haclcala".
    Returns a tuple: store_id and a store_db,
    where the first variable is the store name
    and the second is a dictionary describing the store.
    The keys in this db will be ItemCodes of the different items and the
    values smaller  dictionaries mapping attribute names to their values.
    Important attributes include 'ItemCode', 'ItemName', and 'ItemPrice'
    """"
    tree = ET.parse(filename)
    root = tree.getroot()
    store_db = {}
    items = root.find("Items")
    for item in items.findall("Item"): #First dictionary of items
        item_dic = {}
        for feature in item: #Second dictionary within of attributes in first.
            value = feature.text
            item_dic[feature.tag] = value
            code = item.find("ItemCode").text
            store_db[code] = item_dic
    store_id=root.find("StoreId").text
    return (store_id,store_db)


def filter_store(store_db, filter_txt):
    """
    Create a new dictionary that includes only the items
    that were filtered by user.
    I.e. items that text given by the user is part of their ItemName.
    Args:
    store_db: a dictionary of dictionaries as created in read_prices_file.
    filter_txt: the filter text as given by the user.
    """
    filtered_store_db = {}
    item_dic = {}
    for item in store_db: #Checks each item in store.
        if filter_txt in store_db[item]["ItemName"]:
            for tag in store_db[item]: #Checks each tag of items in each store
                value = store_db[item][tag]
                item_dic[tag] = value
            filtered_store_db[item] = item_dic
    if filtered_store_db == {}:
        return None
    return filtered_store_db


def create_basket_from_txt(basket_txt):
    """
    Receives text representation of few items (and maybe some garbage
    at the edges)
    Returns a basket- list of ItemCodes that were included in basket_txt
    """
    new_basket = []
    for char in basket_txt.split(): #Loop checking each char in basket_txt
        if char.startswith('[') and char.endswith(']'):
            #conditioning a filter to the string we want inside brackets
            last_digit = char.find(']')
            char = char[1:last_digit]
            new_basket.append(char)
    return new_basket


def get_basket_prices(store_db,basket):
    """
    Arguments: a store - dictionary of dictionaries and a basket -
    a list of ItemCodes
    Go over all the items in the basket and create a new list
    that describes the prices of store items
    In case one of the items is not part of the store,
    its price will be None.
    """
    price = []
    for code in basket: #Loop checking each code in the basket
        if code in store_db: #Condtioning if code is found in the store
            price.append(float(store_db[code]["ItemPrice"]))
        else:
            price.append(None)
    return price


def sum_basket(price_list):
    """
    Receives a list of prices
    Returns a tuple - the sum of the list (when ignoring Nones)
    and the number of missing items (Number of Nones)
    """
```

```python
    sum_price_list = 0
    missing_items = 0
    for price in price_list: #Loop taking each price in the basket
        if price == None: #If item doesn't exist, add it to the missing items.
            missing_items += 1
        else:
            sum_price_list += price
    return (sum_price_list, missing_items)


def basket_item_name(store_db_list, ItemCode):
    """
    stores_db_list is a list of stores (list of dictionaries of
    dictionaries)
    Find the first store in the list that contains the item and return its
    string representation (as in string_item())
    If the item is not avaiable in any of the stores return only [ItemCode]
    """
    for store in store_db_list:
        if ItemCode in store:
            item = store[ItemCode]
            return string_item(item)
    return "["+ItemCode+"]"


def save_basket(basket, filename):
    """
    Save the basket into a file
    The basket reresentation in the file will be in the following format:
    [ItemCode1]
    [ItemCode2]
    ...
    [ItemCodeN]
    """
    basket_file = open(filename,"w")
    codes = ""
    for item in basket: #Checking each item in basket, add it to code + line.
        code = "["+item+"]"
        codes += code+"\n"
    basket_file.write(codes)
    basket_file.close()


def load_basket(filename):
    """
    Create basket (list of ItemCodes) from the given file.
    The file is assumed to be in the format of:
    [ItemCode1]
    [ItemCode2]
    ...
    [ItemCodeN]
    """
    file = open(filename,"r")
    basket = []
    for line in file: #Checking each line in the file
        code = ""
        for num in line: #Checking each number in the line
            if (num != "[") and (num != "]") and (num != "\n"):
                #A condition only adding to code if the character is a number
                code += num
        basket.append(code)
    file.close()
    return basket


def best_basket(list_of_price_list):
    """
    Arg: list of lists, where each inner list is list of prices as created
    by get_basket_prices.
    Returns the cheapest store (index of the cheapest list) given that a
    missing item has a price of its maximal price in the other stores *1.25
    """
    sum_price = []
    for j in range (len(list_of_price_list)):
        price_list = list_of_price_list[j]
        sum = 0
        for i in range (len(price_list)):
            #Adding to the price list, accoring to its value
            if price_list[i] != None:
                sum += price_list[i]
            else:
                sum += penalty(list_of_price_list,i)
        sum_price.append(sum)
    return min_index(sum_price)


def min_index(sum_price_list):
    """
    Receives a list of sums of prices and returns the index of the
    place with the minimal value
    """
    min_index = 0
    min = sum_price_list[0]
    for i in range (len(sum_price_list)):
        if sum_price_list[i] < min: #Checking each i to define smallest value.
            min = sum_price_list[i]
            min_index = i
    return min_index


def penalty(list_price_list,index):
    """
    Calculates the penalty value of a missing item according to the
    maximal price in the other stores *1.25
    """
    max = 0
    for price_list in list_price_list:
        if (price_list[index] == None):
            break
        if (price_list[index] > max): #Defining the maximum value
            max = price_list[index]
    return max*PENALTY
```