**Introduction to working with files.**

**Step 1. Reading data from text files**

**READING DATA FROM FILES**

In real-world tasks, we receive data from external sources. Imagine that you need to read a certain data.txt file, which contains analytics for a certain advertising campaign.

In this work we will learn how to read the contents of files and write them, for example, to a list. During the lesson, we will work with the data.txt file (and its Excel counterpart data.xlsx).

Let's start small! For convenience in various tasks, we split the source file into small derivative files with a limited amount of data.

First, consider a file that contains only one column of the data.txt file with user IDs (user\_id column).

File – user\_ids.txt

**Step 1.**

To work with this file, download it yourself.

This is what the user ID column looks like:

**1010**

**1036**

**1041**

**1041**

**1042**

**Step 2. Read file**

The following construction is used to read files:

**with** open('user\_ids.txt', 'r') **as** f:

**for** line **in** f:

In it we specify the file name (user\_ids.txt), the read parameter ('r') and designate the file as f.

Then in the “for line in f” loop we go through each line of the file writing its content to the *line* variable. Display the file contents on the screen:

**with** open('user\_ids.txt', 'r') **as** f:

**for** line **in** f:

**print**(line)

What value will be displayed last on the screen?

**Step 2. Reading file content into list**

For some reason, user\_id values are displayed on the screen with extra "spaces" between lines. This is due to the fact that in text files at the end of the line there is always a service line feed character (denoted as \ n and is hidden from the user in text editors). To avoid this, you can use the *strip ()* method. It removes line breaks from the beginning and end of a line.

**with** open('user\_ids.txt', 'r') **as** f:

**for** line **in** f:

**print**(line.strip())

To perform operations with a set of ID users, let’s write each successive ID to the user\_ids list:

user\_ids = []

**with** open( 'user\_ids.txt', 'r' ) **as** f:

**for** line **in** f:

user\_id = line.strip()

user\_ids.append(user\_id)

Exercise

Display the first 5 elements of the user\_ids list. What ID will be the last in this set?



**Step 3. Count the unique numbers of elements**

Exercise

In the last step, we got the list of user IDs from a file.

How many items are there in this list? Reminder: the length of the list can be calculated using the *len()* function.



**Count the unique numbers of elements**

In the user\_ids list some ID of users are duplicated because these users have made several purchases. Let’s say we need to understand total number of unique users.

To count unique IDs of this list just use *set()* function. This function converts a list to the so-called set type, which contains a unique set of list elements. Let’s look at a simple example.

Let a list of numbers (from zeros and ones) be given:

list\_of\_zeros\_and\_ones = [**0**, **1**, **0**, **1**, **0**, **1**]

Then the set() function will return us a set of unique list elements, i.e. 0 and 1:

**print**(set(list\_of\_zeros\_and\_ones))

Let’s apply this functions to out set of user IDs. That is, we get a set of unique user IDs from the user\_ids list:

unique\_ids = set(user\_ids)

Now we can count the number of unique users in the file:

**print**( 'Unique IDs in the file {}'.format(len(unique\_ids)))

Exercise

How many unique IDs are there in the user\_ids.txt file?

**Files with headers and multiple columns**

**Step 1. Reading header files**

Often the file contains a header that we don't need to take into account in the calculations:

with open( 'user\_ids\_headers.txt', 'r' ) as f:  
      for line in f:  
            print( line.strip( ) )

Output:  
user\_id  
1010  
1036  
1041  
1041  
1042  
1042  
1042  
1042  
1042  
1042

How can a header get in the way?

If we count the header along with the rest of the user IDs, then it will introduce an error in many of our calculations: the number of unique users in the file, the average number of orders per user, and others. Let's look at a way that will allow you to ignore the title in the calculation of the required metrics.

For this exercise, we'll use the header file:

File – user\_ids\_header.txt

Previously, we always read data from the first line of the file:

**with** open('user\_ids\_headers.txt', 'r') **as** f:

**for** line **in** f:

**print**(line.strip())

Exercise

Which line will appear first when reading the user\_ids\_headers.txt file in the way described above?



**Step 2. How avoid the first line**

In order, not take into account the first line, you can use a simple trick. It will be useful to you in many work tasks:

* 1. Let’s create Started variable, which is equal to True before starting the file.
  2. In the step of the loop convert it to False value and do nothing else Thus, we skip the first line of the file (i.e. header).
  3. In the second and next steps of the loop the Started value will always be False. And in these steps we already perform the operations we need with the string:

Started = True

**with** open('user\_ids\_headers.txt', 'r') **as** f:

**for** line **in** f:

# "if Started" is equivalent to "if Started == True"

**if** Started:

# this line is executed once only on the first step of the loop

Started = False

**else**:

# this line is executed from the second and in all the next steps of the loop

**print**(line.strip())

**Step 3. Reading data from files with multiple columns**

Now we have worked with only one column, but how to read data from a file with several columns, i.e. whole table?

Consider working with the data\_3\_columns.txt file, which has three columns! In it we have some statistics on the sources and cost of the order.

**with** open('data\_3\_columns.txt', 'r') **as** f:

**for** line **in** f:

**print**(line.strip())

Result:

seo google **20**,**20**

sem yandex **15**,**60**

email promo **13**,**20**

sem yandex **9**,**80**

sem google **14**,**80**

The split method is used to split the line line into columns. The column separator is substituted as an argument. For example, in our file, the separator is a tab (the change in the code is highlighted in red):

**with** open( 'data\_3\_columns.txt', 'r' ) **as** f:

**for** line **in** f:

**line = line.strip().split('\t')**

**print**( line )

['seo', 'google', '20,20']

['sem', 'yandex', '15,60']

['email', 'promo', '13,20']

['sem', 'yandex', '9,80']

['sem', 'google', '14,80']

Now in the *line* variable at each step of the loop the list with columns elements will be written.

**Step 4. Splitting columns**

In the previous example there is a problem: the third column must be number (cost of order), but it is displayed in quotes, also separated by commas (a consequence of copying data from Excel).

**Remember: in all major programming languages, the number is always separated by a dot.**

It's good that Python allows us to coalesce large amounts of data. To correct commas with dots, we can use the *replace()* method. We put a comma as the first argument (the element that we are looking for and replace in the string). The second is a dot (the element we are replacing with).

We already understood from the data that the first and second columns show us the intermediate and main source from which the user came, and the third indicates the cost of the order.

Let's also replace the elements of the column with clear variable names (medium, source, amount\_paid), which reflect what data we have in which column.

**with** open('data\_3\_columns.txt', 'r') **as** f:

**for** line **in** f:

line = line.strip().split('**\t**')

medium = line[**0**]

source = line[**1**]

amount\_paid = line[**2**].replace(',', '.')

**print**(line)

**print**(source, medium, amount\_paid)

Result:

['seo', 'google', '20,20']

google seo **20.2**

['sem', 'yandex', '15,60']

yandex sem **15.6**

['email', 'promo', '13,20']

promo email **13.2**

['sem', 'yandex', '9,80']

yandex sem **9.8**

['sem', 'google', '14,80']

google sem **14.8**

**Step 5. Get a number from the string line**

We replaced commas with dots, but variable amount\_paid is still of string type:

type(amount\_paid)

Result:

str

This will prevent us from doing calculations with it, as with a number. For example, we cannot calculate a amount of orders or find the most expensive one. Let’s convert the order cost from string to the float type using *float()* function:

**with** open('data\_3\_columns.txt', 'r') **as** f:

**for** line **in** f:

line = line.strip().split('**\t**')

medium = line[**0**]

source = line[**1**]

**amount\_paid = float( line[2].replace(',', '.') )**

**print**( source, medium, amount\_paid )

Exercise

What is the amount\_paid value in the last line of the file? Do not forget that the separator must be a period.



**Step 6. Counting the amount in files**

In the last part of this work, we managed to bring the third column to a numerical form. Let's calculate the sum of this column by adding to the total\_sum variable in the loop the cost of each next order. Let's get started?

When working, we use the same data\_3\_columns.txt file.

To understand the process, at each step, we will display the total\_sum as a cumulative total. We take the code from the previous step and add the necessary lines:

total\_sum = **0**

**with** open('data\_3\_columns.txt', 'r') **as** f:

**for** line **in** f:

line = line.strip().split('**\t**')

amount\_paid = float(line[**2**].replace(',', '.'))

**total\_sum += amount\_paid**

**print('** **Current expenses: {:.2f}'.format( total\_sum ))**

Result:

Current expenses: **20.20**

Current expenses: **35.80**

Current expenses: **49.00**

Current expenses: **58.80**

Current expenses: **73.60**

...

Exercise

What is the value of the sum we get in the last example?



**Step 7. Counting amount with filter**

Let's calculate the sum not for all lines, but under the following condition: we need to take only those lines for which the source is equal to “google”. To do this, add just one line to the loop:

total\_sum = **0**

**with** open( 'data\_3\_columns.txt', 'r' ) **as** f:

**for** line **in** f:

line = line.strip().split('**\t**')

medium = line[**0**]

source = line[**1**]

amount\_paid = float( line[**2**].replace(',', '.') )

**if source == 'google':**

total\_sum += amount\_paid

**print**( 'Current expenses google: {:.2f}'.format( total\_sum ) )

Result:

Current expenses google: **20.20**

Current expenses google: **35.00**

Current expenses google: **49.40**

Current expenses google: **63.40**

Current expenses google: **86.00**

...

Current expenses google: **1318.80**

Exercise

What is the amount\_paid for rows that have source == 'yandex' and medium == 'seo'? Reminder: concurrent conditions in an if statement are enumerated using *and*

