# Angel Obregón // Ангел Обрегон

# Bots in Wikidata

This chapter examines the automation of processes in Wikidata. On many occasions we want to correct recurring errors or enter large amounts of data in Wikidata instead of modifying properties one by one. For data entry we have at our disposal several tools that facilitate our work, such as OpenRefine or QuickStatements, but repetitive modifications and insertions over time should be done with a bot.

## Requirements

We can program a bot in several programming languages, but to facilitate our task we can use Pywikibot, a collection of tools programmed in Python to facilitate access to information in Wikimedia Foundation projects. We have three options to run our programs:

1. using a web shell like PAWS,
2. setting up a cloud infrastructure like Toolforge, or
3. running it on our own computer.

Any of the three options can be used thanks to the explanations we have in the [official documentation](https://www.mediawiki.org/wiki/Manual:Pywikibot/Installation). If we use our own computer we must install Python, following the instructions in the documentation. After the installation we can check if it is installed correctly by pressing "py -V" in the Windows command prompt or in the Linux console (from now on everything must be done from here), and it will show us the version we are going to work with. Then we must install Pywikibot and configure it. This configuration has several steps:

* Execute the following command to create the configuration file: “py pwb.py generate\_user\_files”.
* Execute the following command to login: “py pwb.py login”.
* Review the files user-config.py and user-password.py where we can indicate the user names that will make the modifications.

To view information or perform small tests we can use our username, but in case of making many modifications in Wikidata it is recommended to create a bot account and ask for permission to run it. To have this flag it is usually required to have some knowledge of editing in Wikimedia projects, which will certify that we will not make serious mistakes and we have the support of other users.

Normally to obtain this flag we will be asked to perform a specific number of test editions, to certify that such changes make sense, since we should have a low or zero error rate. We could also perform tasks with our user account, but with a low rate of edits.

## Our first script

Once we have correctly configured Pywikibot we can run our first script. We will go to the folder where Pywikibot is located and there we create a new file called test.py.

We can edit the file with any text or code editor such as Visual Studio Code, Notepad++ or a simple Notepad. Once inside the file we will put the following lines of code:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u"Q16583338")

print(page.get())

We save the file and execute it by typing the following instruction:

py pwb.py test.py

If everything has worked correctly we will be seeing in our console all the text written in the Obnica river article in Wikidata (the Q16583338 element). In the first line of code we have imported Pywikibot. Then we have indicated the name of the project we want to work on, as we could do it in any Wikipedia language or Commons. Then we indicate a specific element, and finally we tell it that we want to get all the content.

“Page” is a module that contains many methods that can be useful to us. To learn more about it we can go to the [official documentation](https://doc.wikimedia.org/pywikibot/master/_modules/pywikibot/page.html). In addition to the "get" method, we have at our disposal the possibility of knowing the title of the element using the “title” method:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u"Q16583338")

print(page.title())

We can also find out if this element is a redirect page with the “isRedirectPage” method. For it we are going to change the previous element and we are going to indicate another one that is a redirect page (Q16583333):

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u" Q16583333")

print(page.title())

print(page.isRedirectPage())

Caution: so far, the exercises were very simple, but as we progress we have to be careful with tabs, which let Python know what information goes inside the control structures.

The previous exercise only returns true or false, but the usual use of this method is to ask if it is a redirect page or not, and act accordingly, warning the user that this element is a redirect page, as in the following example:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u" Q16583333")

if (page.isRedirectPage()):

print("Attention, this is a redirect, not an element")

But, what happens if we are analyzing several elements and we find one that does not exist? Possibly our program would return several errors because it could not analyze the interior of the element. For this, we can catch the errors in Python, but we can also ask if the element exists or not:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u" Q107043778")

if not page.exists():

print("Sorry, but this element does not exist!")

Exercise: Find how we can capture errors in Python and improve the previous program so that we can capture possible errors returned by the program.

Next, we are going to analyze an element in more depth, to know all those properties that it has included. To do this we can use the "properties" method, as we describe here:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u"Q16583338")

print(page.properties())

When executing this script, we can see that this method returns a Python dictionary, and that is because the "properties" method returns a dictionary with the properties contained in that element. For example, we can see that in the property "page\_image\_free" we get the picture that appears in the element, that "wb\_claims" contains the amount of properties that are entered in the element, or that "wb\_sitelinks" tells us the amount of links that exist to the rest of the Wikimedia Foundation projects.

The "contributors" method also returns a dictionary. In this case with the number of users who have contributed to the element. It will show us the user name of all the contributors, as well as the number of editions of each one. We also have the method "revision\_count", which returns the number of edits that have been made in total on the element, as shown below:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u"Q16583338")

print(page.contributors())

print(page.revision\_count())

There are other methods, which return more complex information than a data dictionary, such as an object. This is the case of the "linkedPages" method. If we use it as up to now it will return "<pywikibot.data.api.PageGenerator object at 0x000002C6C7DB2FD0>", indicating that it contains an object of type "PageGenerator". To be able to read it we can do it with a for loop:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u"Q16583338")

for linked in page.linkedPages():

print(linked)

As we can see this method provides us with the links to which we can click from the element. First, the elements will be shown and then the properties (including the information found in the references). Obviously if the selected element is not very large, the number of links will not be very large, but if we try this example with a very large element, the number of links will be very large.

To know the number of elements that are linking to a specific one, we also have a method called "backlinks", which contains an object and that we have to go through as we did in the previous exercise. Depending on the element we may find an empty list, so we will select an element that is linked to other elements:

import pywikibot

site=pywikibot.Site('wikidata', "wikidata")

page=pywikibot.Page(site, u"Q6980876")

for links in page.backlinks():

print(links)

In this case we have chosen another river, which is linked twice: by another river and by a redirect page. In this way we can know the number of times a certain element is being linked.

Exercise: Put into practice everything we have seen so far but using an element with a low number, which probably has much more information entered than the examples we have used so far.

In addition to the "Page" module, Pywikibot provides us with many other possibilities. For example, introducing a single element does not make much sense, normally we will want to work with several elements to analyze them and if necessary modify them. For this we can make use of the queries that we have been doing throughout the course.

## Running queries

First, we go to Wikidata Query Service to create our query. We look for possible errors in the data entered, such as people who have a certain country in the property P17. This is a common error as many users enter P17 instead of the country of nationality (P27).

This query could look like this:

SELECT ?item ?itemLabel WHERE {

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

?item wdt:P31 wd:Q5;

wdt:P17 wd:Q36.

}

We must check if the query returns any value, since it could be that there are no errors to be solved. In that case we can change the country, since there will surely be other countries with the same error.

When we get a query that returns any person, we can add the query to our program, like this:

import pywikibot

from pywikibot import pagegenerators

site=pywikibot.Site('wikidata', "wikidata")

query = """

SELECT DISTINCT ?item WHERE {

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

?item wdt:P31 wd:Q5.

?item wdt:P17 wd:Q36.

} """

pages = pagegenerators.WikidataSPARQLPageGenerator(query, site=site)

for item in pages:

print(item.title())

With respect to the previous exercises we have only imported the "pagegenerators", a module that allows us to generate a list of items from certain filters as a SPARQL query as above. We have saved the query in a variable called "query", and then we have passed it to the pagegenerators method called "WikidataSPARQLPageGenerator". The data returned by this method has been traversed with a for loop and we have displayed the title of the item.

In short, this program shows us all the people who have Poland as a country.

We can already obtain several elements at the same time. Now it is time to delve into the content of these. Instead of displaying the title of the article, we will display some parts of its content. For example, sitelinks, aliases, labels or claims.

import pywikibot

from pywikibot import pagegenerators

site=pywikibot.Site('wikidata', "wikidata")

query = """

SELECT DISTINCT ?item WHERE {

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

?item wdt:P31 wd:Q5.

?item wdt:P17 wd:Q36.

} """

pages = pagegenerators.WikidataSPARQLPageGenerator(query, site=site)

for item in pages:

item.get()

print(item.sitelinks)

print(item.aliases)

print(item.labels)

print(item.claims)

Exercise: test the queries you have performed earlier in the course and display the results in your console with messages to help you know what is running.

We have shown the information in the Wikidata elements, but now let's see how we can modify them.

## Modifying the values entered in Wikidata

The search for errors that are repeated over time is one of the tasks of a bot programmer, because depending on the task performed by the bot, it will have to be executed regularly, and improve the search for errors over time. For example, the following query returns all professions sorted by the number of times they are being used in Wikidata.:

SELECT ?instanceLabel ?value WHERE {

{

SELECT ?instance (COUNT(DISTINCT ?item) AS ?value) WHERE { ?item wdt:P106 ?instance. }

GROUP BY ?instance

ORDER BY DESC (?value)

}

SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }

}

ORDER BY DESC (?value)

This query can be used to check which professions are being used incorrectly, or have been introduced by vandals. Reviewing the list, we can see that there are some professions that are not correct and have meaningless information such as "Q7404672", "Q7319129", "Q7249866", "Q7244433" and "Q7208069". We can create a list of errors and show the biographies that have these professions entered.

import pywikibot

from pywikibot import pagegenerators

site=pywikibot.Site('wikidata', "wikidata")

delete\_occupation={"Q7404672", "Q7319129", "Q7249866", "Q7244433", "Q7208069"}

for occupation in delete\_occupation:

query = """

SELECT DISTINCT ?item WHERE {

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

?item wdt:P31 wd:Q5.

?item wdt:P106 wd:""" + occupation + """.

} """

pages=pagegenerators.WikidataSPARQLPageGenerator(query, site=site)

for item in pages:

print("These are the people who have {occupation} as their occupation: {title}" . format(occupation=occupation, title=item.title()))

The list has been created in the variable "delete\_occupation". We run through it with a for loop to call the query for each of the professions that we have been adding. This time the query is not the same as before, it has a variable that is modified depending on the value of "delete\_occupation". The query will ask for the people who have a profession that matches the value of "delete\_occupation", and will show it on the screen.

Attention: so far we had used "print" to show the information contained in the variables, but in the previous exercise we have used "format", as it is recommended to give clarity to strings.

Now, if we are sure that this information is wrong and we want to delete it, we must indicate the specific property we want to delete. The problem is that a biography could have three professions, and only one of them is the one we want to delete. We cannot delete all the values of the property, we have to know the specific data to delete.

In the following we will see how this can be done:

import pywikibot

from pywikibot import pagegenerators

site=pywikibot.Site('wikidata', "wikidata")

delete\_occupation={"Q7404672", "Q7319129", "Q7249866", "Q7244433", "Q7208069"}

for occupation in delete\_occupation:

query = """

SELECT DISTINCT ?item WHERE {

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

?item wdt:P31 wd:Q5.

?item wdt:P106 wd:""" + occupation + """.

} """

pages = pagegenerators.WikidataSPARQLPageGenerator(query, site=site)

for item in pages:

print("These are the people who have {occupation} as their occupation: {title}" . format(occupation=occupation, title=item.title()))

item.get()

for valor in item.claims['P106']:

if (str(valor.getTarget())=='[[wikidata:' + occupation + ']]'):

print("<<<<< Deleting {occupation} of {title} >>>>>" . format(occupation=occupation, title=item.title()))

item.removeClaims(valor, summary=u'Deleting erroneous values in P106')

In the previous code what we have done is to maintain the previous code, but adding the last lines, in which we go through the values contained in the property "P106" with the method "claims". If this one coincides with the occupation that we are looking for, we delete it with the method "removeClaims". This method needs the value we want to delete and the edit summary to show the rest of the users what is being done.

Exercise: add more professions that are wrong in the variable "delete\_occupation" and run the bot to perform the deletions of those properties.

Caution: this kind of edits can cause many problems on Wikidata and administrators may block your user account. Make few changes at the beginning, and always check the edit history to see what information has been modified.