# **User Guide for crawl.py**

## **Introduction**

The crawl.py script is a web crawling tool designed to extract information from <https://games-stats.com/> and generate code to update New Game Entries at the 'games\_data.php' file in the 'includes' folder from GamePulse project(https://github.com/DoonOnthon/GamePulse). This guide will help you set up, run, and troubleshoot the script.

## **Getting Started**

### **Prerequisites**

Before running the crawl.py script, ensure you have the following installed on your system:

* Python 3.9 or higher
* Selenium library
* WebDriver for your preferred browser (e.g., ChromeDriver for Google Chrome)

### **Installation**

1. **Install Python:** Download and install Python from the [official website](https://www.python.org/).
2. **Install Selenium:** Open a terminal or command prompt and run the following

pip install selenium

1. **Download WebDriver:**
   * For Chrome, download ChromeDriver from [here](https://developer.chrome.com/docs/chromedriver/downloads).
   * Ensure the WebDriver is in your system PATH or specify its location in the script.

### **Script Setup**

1. **Download the crawl.py script:** Save the crawl.py file to a directory on your computer.
2. **Setting the starting and ending pages to crawl:** Open crawl.py and modify line 12 and 13 to set the starting and ending pages of <https://games-stats.com/steam/> to crawl.

## **Basic Usage**

1. **Open a terminal or command prompt:** Navigate to the directory where crawl.py is saved.
2. **Run the script:** Execute the script using the following command:  
     
   python crawl.py

### **Example**

If crawl.py is set up to crawl a specific website, running the script as shown above will start the crawling process and output the results to the console or a specified output file.

## **Advanced Features**

The script may include various advanced options such as:

* **Customizable XPaths:** Modify the XPaths within the script to target different elements on the web page.
* **Output Options:** Change the output method (e.g., print to console, save to file).
* **Error Handling:** Adjust error handling mechanisms to better suit your use case.

### **Customizing XPaths**

To change the target elements, locate the XPath expressions in the script and modify them. For example:

title = driver.find\_element(By.XPATH, 'your-new-xpath').get\_attribute("textContent").split()

## **Troubleshooting**

### **Common Issues**

1. **NoSuchElementException:**
   * Ensure the XPath is correct and matches an existing element on the web page.
   * Add a wait time to allow elements to load properly.
2. **WebDriverException:**
   * Verify that the WebDriver path is correctly set.
   * Ensure the browser version matches the WebDriver version.
3. **Timeouts:**
   * Increase wait times to account for slow page loading.

### 

### **Getting Help**

If you encounter issues not covered in this guide, consider the following resources:

* Selenium documentation: [Selenium Documentation](https://www.selenium.dev/documentation/)
* Online forums and communities such as Stack Overflow

## **Support and Resources**

For further assistance or to report bugs, you can contact the script maintainer or refer to the following resources:

* Selenium GitHub Repository: [SeleniumHQ](https://github.com/SeleniumHQ/selenium)
* Python Documentation: [Python Docs](https://docs.python.org/3/)