Devlog - Python Openpyxl

소방서에서 파이썬을 통한 엑셀 업무 자동화를 실전에서 사용해볼 기회가 생겨 코드를 짜봤다.
python의 openpyxl이라는 모듈이 엑셀의 기능을 파이썬으로 접근하게 해주었다. 이를 이용하여 전라남도 강진-장흥 군 주택관리 대장에 기초소방시설이 얼마나 설치되었는가를 입력하는 작업이었다.

하지만 2010년 초부터 2019년까지의 자료도 방대할뿐더러 설치한 센터나 팀마다 서식도 다르고 주소도 정확하지 않아 문제가 생겼다. 코드를 짜기에 앞서 우선은 각각의 엑셀 시트의 주소를 통일하는 작업이 필요했다. 하나의 코드로 작업을 자동화 할 것이기에 모든 주소를 도 서식에 맞추어 서식을 통일하였다.
엑셀의 'TRIM'함수가 큰 도움이 되었다. 자동의 띄어쓰기를 한 칸으로 만들어 주고 앞뒤의 공백을 없애주기 때문이다. 또한 부끄럽게도 F4를 누르면 주소가 절대주소로 자동변환 된다는 것도 이 날 외웠다. 역시 그냥 책으로 보기만 한 것과 직접 사용하며 익히는 것은 천지차이였다.
: ='전라남도 강진군 '&TRIM(D4) or D4 with F4 key

엑셀 주소에는 통이라는 옛날 주소도 상당히 많았는데 이는 엑셀의 와일드카드를 이용한 검색으로 수월하게 데이터들을 걸러낼 수 있었다.
: Excel wildcard search => ??통 delete

처음으로 코드를 짰을 때 이 프로그램이 작동하는지 아니면 무한루프가 돌아가는지 등등 여러 문제를 확인하기 위해 콘솔 로그를 넣어주려 하였으나 관란 메서드를 몰라 그냥 프린트 함수로 출력하였다. 다행이도 파이썬은 한줄 한줄 코드가 차례로 작동하는 인터프리터 언어이기 때문에 프린트 함수로도 코드의 순서나 현황을 파악할 수 있었다. 또한 코드의 작동 시간이나 엑셀 파일을 읽어오는 시간, 저장하는데 걸리는 시간 등을 파악하기 위해 time모듈을 넣어 작동시간 또한 출력했다.
: console.log => print, time.time()

openpyxl이 처음인지라 여러 어려움을 겪었다. 그 중 가장 빈번하게 저질렀던 어처구니없는 실수들을 소개하자면, 시트를 로딩하고 나서 셀 값을 일치하는가를 확인하는데 아무리 육안으로 보았을 때는 일치하는 자료여서 코드에서는 일치하지 않는다고 알려주었다. 사실은 '='수식으로 만들어진 값이 셀에 표시만 되었지 셀 자체 값은 수식이기에 데이터와 수식이 일치하지 않는 데에서 나오는 오류였다. 이를 해결하는 방법은 시트를 로딩할 때 인자로 'data_only=True'를 넣어주어 값만 불러오는 것이었다.
: loading workbooks and sheets with param data_only=True

정말 바보같은 실수의 하나는 바로 sheet.cell와 sheet.cell.value를 헷갈렸던 것이다. 단순히 셀이라 하면 값을 불러오는 줄 알았건만 사실 셀로 불러오는 값은 해당 셀의 시트와 위치 값을 리턴하는 셀 객체의 데이터타입이었고 .value를 붙여주어야만 셀 안의 값을 리턴하였다.

그리고 멍청함의 백미는 바로 행과 열을 헷갈렸던 것이다. row에 열 값을 넣고, column에 행 값을 넣은 다음 왜 계속 값이 아니라 Nonetype을 리턴하는 거지 하고 고뇌하였다. 당연히 아무것도 없는 셀을 참조해놓고 값이 나오기를 마냥 기다리고 있었던 것이었다. 자신이 실수하였을 거라는 생각은 1도 하지않은 채 말이다. 또한 파이썬에서는 &&나 ||가 아니라 그냥 영어 단어 and, or이 키워드로 사용되었고 I++ 가 아니라 I += 1로 표기되는 것도 이 날 알았다. 파이썬이라는 언어를 처음 다뤄본 것도 아닌데 바보 같이 잊고 살았거나 제대로 기초 공부를 안했거나 했던 것이다. 또한 파이썬에서 해당 셀이 비었는지 안 비었는

지를 확인하는 방법으로 if value is None:이라는 is None 키워드가 있다는 것도 역시 이 날 알았다. 포문을 돌리다보니 continue와 break도 사용해보게 되었다.

가장 소중한 경험은 예외처리였다. 코린이인 나로써는 예외처리는 상당히 고수의 영역이라 마냥 생각하고 있었는데 때마침 로그를 출력하는 프린트 함수에서 Nonetype을 stringd로 바꿀 수 없다는 Typeerror 오류가 나왔다. 중간중간 빈칸이 있는 셀이 존재하기 때문에 생겨난 오류인데 이를 try-catch를 통해 무사히 넘겼으며 pass 키워드와 continue 키워드의 차이점 또한 이 날 배우게 되었다.

자료를 검색해보았을 때 해당 시트의 데이터가 들어있는 최대 열을 자동을 파악해주는 함수로
.get_highest_row가 있다고 책에서 보았지만 작동하지 않았다. 아마 현 시점에서는 .max_row만 된ㄴ 듯하다. 정확한 모듈 관련 자료는 아직 파악하지 못하였다. .max_row로도 안되는 시트는 그냥 무식하게 마지막 행 숫자를 포문의 stop 값으로 넣어주었다.
: for I in range(1, sheet1.max_row, 1):

openpyxl에서 제공하는 편리한 기능 중 하나는 바로 열 문자의 인덱스를 일일이 숫자로 써줄 필요가 없다는 것이었다. 그저 'column_index_form_string()'에 문자 값을 넣어주면 알파벳 순서로 인덱스를 리턴해주었다. 처음에는 일일이 숫자를 세어 열에 접근하였지만 이 기능을 알고 난 후 매우 작업이 편리해졌다. 얼마나 이게 귀찮았냐면 차라리 리스트나 딕셔너리를 만들어 버릴까 하고도 생각을 할 정도였다.
: we should use => 'column_index_form_string('AG')'

또한 하나의 워크북 안에 다양한 시트들이 들어있을 때 편리하게 시트에 접근할 수 있는 기능 또한 지언하였다. 시트의 이름으로 접근하는 get_worksheet_by_name(), 혹은 tlxdml 순서로 접근하는 단순한 worksheets[0] 리스트가 매우 편리하였다. 만약 모든 시트를 순회해야한다면 그냥 인덱스에 I를 집어넣고 포문을 돌리면 끝인 것이다.

주소에는 과거 사용하던 번지 주소와 도로명 주소가 혼합 되어있어서 어쩔 수 없이 이중 포문을 사용하게 되었는데 이 안에는 또한 여러 변수가 혼재해 있어서 알고리즘을 짜는데 상당히 고통스러웠다. 사실 다 끝나고 나서 보면 아무것도 아닌 걸로 한시간을 씨름하고 있던 멍청한 나지만 당시에는 알고리즘을 짜서 만든 포문 고드가 작동하는 사실이 그렇게 기쁠 수가 없었다. 백준 알고리즘을 꾸준히 공부해야 하는 이유 역시 새롭게 피부 깊숙이 깨달았다.

마지막으로 annotation!!! 주석을 달아야 이후 알아보기도 쉽고 타인이 접근하였을 때에도 사용하기가 편하다. 항상 주석을 보기 깔끔하고 적절하게 사용하는 법을 익혀야한다고 생각하였다.