PP03

Nobody expects the Spanish Inquisition!

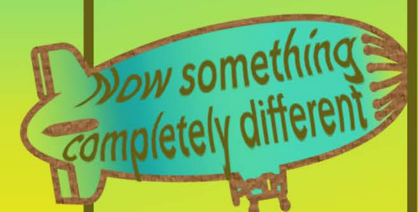
How can (Monty) Python® help with your daily job around deliverables?

Are you tired of checking specifications against standard metadata? Find it utterly boring to check deliveries, with annoying questions such as: Do you have all the required items for your submission? Are they in the correct format? What about those pesky links in the DEFINE.XML? Someone wants to sell you a Dead Parrot and you want immediate evidence of that? We applied for some funds to the Ministry of Silly Walks to build a TKINTER GUI with several functions to avoid manual work around our submissions and deliveries. A few of our programmers ended up in an Argument Clinic but eventually, we did something great. We had fun, hopefully you will too! Join us on our Python journey of how this smart programming language can help with your daily tasks.

Advantages and disadvantages of Python program language

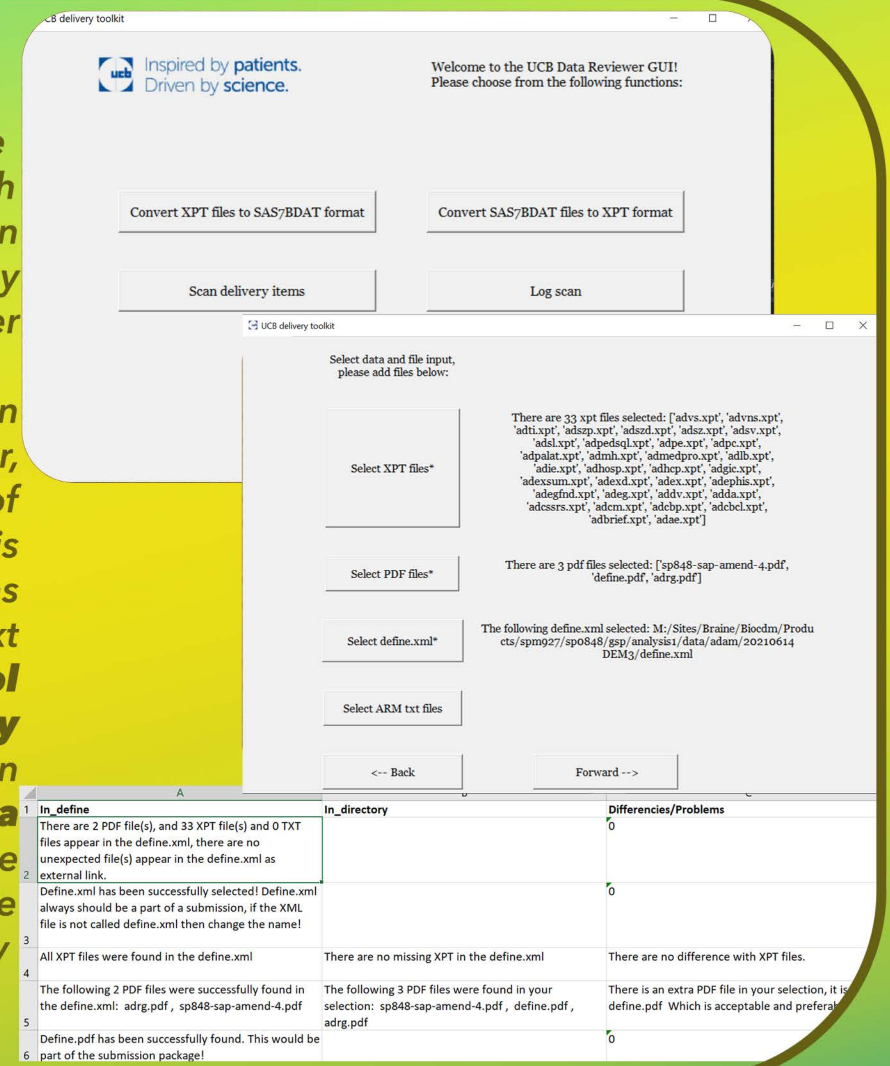
- *Flexibility*
- *Improved productivity*
- *Smart, easy to learn and use*
- *Strong community, massive libraries*
- *Several packages supporting data science, such as NumPy, Pandas, Pyplot and SASPy which can make a direct connection with SAS.*

- *Speed limitations*
- *Memory usage ineffeciency*
- *Less developed database layers*



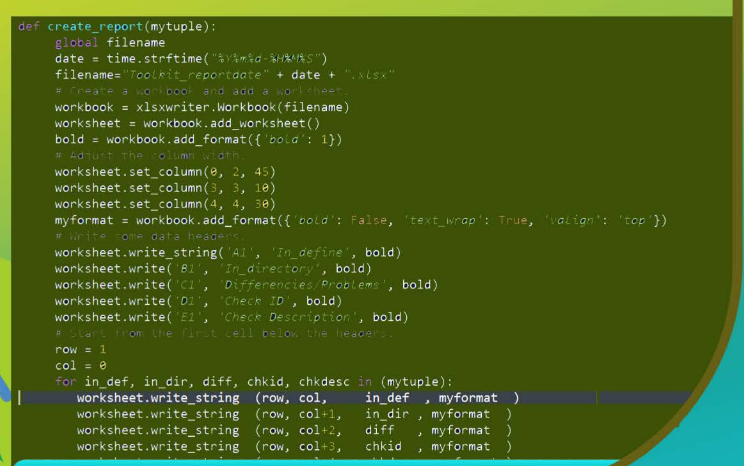
*Nobody expects the Spanish Inquisition! Introducing **Tkinter** - the new and (rather surprisingly!) easy-to-use **GUI** toolkit. Here one can smoothly create frames and widgets, set up menus, or assemble buttons with ease. Tkinter is written in Python meaning other Python packages are easily integrated. If you don't have any supplementary Python modules, that is fine also as Tkinter works as a stand-alone tool.*

*Our idea was to create a tool which can help to scan deliveries and support submission related activities. So far, we have developed four basic functions as a proof of concept. One of them is the **delivery scan**. Using this function, the user should provide the delivery items such as XPT files, define.xml, all submission related pdf files and txt files which belong to the define's ARM section. **The tool reads those items, cross-checking that the delivery** contains all of the required files and all correspondence can be found in the define.xml. Next, the tool **generates a report in excel format** containing any findings of the delivery package. Behind the scenes we were using the **openpyxl** to read xml files and in order to get the delivery items we used Tkinter's own function called 'askopenfilenames'.*



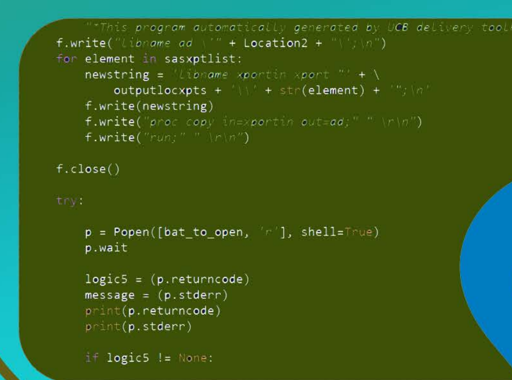
*The report creation is fairly simple, all checks should produce a 5-element list which will be appended to a tuple. Each check will produce a list individually. Similarly to an 'if-else' branch, Python has its own feature called '**try**' where it attempts to run a piece of code and if this code runs into a system error it skips that particular part. With this, **one can handle unexpected issues** and provide instructions on what to do when it runs into problems - essentially skip it and move on. **This ensures all of the checks run smoothly** and produces the required results list.*

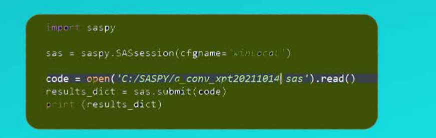
*Our report function utilizes the '**xlswriter**' Python package. It creates a new formatted XLS file with the date-time stamp as its title. Using a 'for loop', it reads the tuple elements and put those into different columns. At the end of the cycle it increments the row index, so each list goes into one line. Finally, it saves a copy of the generated file and launches Excel for it to be displayed on-screen.*

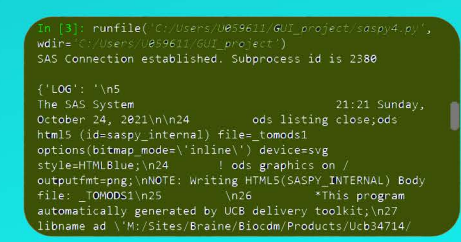


*In order to facilitate the capturing of the required information from the define, we needed to build a python program which would extract what we needed. With many levels of xml tags to contend with, this was no mean feat! Digging down into the XML code, **we isolated the appropriate XML tag** which allowed us to gather the key sequence sort order for comparison with the actual dataset sort order.*

*In our daily life we have many manual tasks such as reviewing delivery packages, running routine tasks, or overseeing the outcome of SAS programs. Python is an ideal instrument to generate new routine codes, allowing the user to save time for more in-depth tasks. Since we all know: **No time to lose!** With the **operating system** interface package **(OS)** one can read or write file contents, which is useful for interfacing with other applications or file formats. The **SASPy package** (courtesy of SAS Institute Inc.) provides **direct access for Python to SAS** allowing the user to directly reach a specific server or local SAS workstation, while running SAS code with no concurrent SAS session needed. From here, the **outcome is extremely productive** since one can **mix both programs' benefits.** One can do the difficult data manipulation with SAS and feed the data back for further analysis or visualization. Since Python's data structures are so flexible, one can easily store a batch of data in a specific object thus allowing a programmer to substitute SAS macros, providing further automation. SASPy is also able to read and print the log file of a SAS session, so Python can serve as an **application program interface (API).** There is an initiative which takes care of this collaboration where one can learn so much more. They like to use the abbreviation **BFFR** which means '**best friend forever**' as a reference to how fruitful the collaboration has been.*







András Kása (andras.kasa@ucb.com) UCB Pharma, Monheim am Rhein, Germany