OPS435 – How does this Code Work (Notes on Assignment 1) / Nischay Behal (123496184)

* High Level Functionality: The code that we are intending to write is primarily a python abstraction of data-conversion meaning given a date in one of the allowed formats (stated later)– the python script when executed will convert it to the format template: 3-letter-name-of-the-month date, Year. The application of the above-mentioned python script is expected in Banking, Healthcare and other industries where adding dates to application is a routine task and for date to be in the correct format is indispensable. Once the following python script is put into action – It will work to validate, process and correctly format the entered date.
* There are 2 main parts of the python script that we are planning to write:

1. Defining Functions
2. Writing code using those functions

The first part of the script will be used to define functions as to what those functions are and define what they are supposed to (Read: What their purpose is ?).

Once defined - The second part of the script will expand upon the functions we’ve defined in part 1, this is part of the code that does the actual work of utilizing the validation function and processing them into correct format.

Here is a step-by-step annotated version of how the code will work(*pseudocode-version*):

1. #!/usr/bin/env python3 *#This is the shebang line , it will be used to define where the python interpreter is located and instructs the IDE to run it as a python version3 script*
2. Import os *#This line imports the os module – which is a module used to interact with the operating system the python script is running on, it is only after we import this module that the functions defined below will work.*
3. Import sys #*This imports the sys module which provides variables and functions used to manipulate the python environment and allows us to read what arguments are passed on onto the script on the command line – for example it is only after importing the sys module that arguments can be passed onto the script from cmd-line such as a1.py 2020 where a1.py can be denoted by sys.argv[0] and 2020 by sys.argv[1] where [n] is the command line index.*
4. **def leap\_year(obj):** #*This defines a function named leap year (denoted as leap\_year with underscore to denote whitespace).This function is used to confirm, in a standalone sense, if a 4 digit number(year) when passed to it is a leap year or not.For a year to be a leap year it must satisfy 2 conditions.One, it must be a multiple of 400 (meaning it must leave a remainder of 0 when divided by 400, this can be denoted in code as year % 400 == 0).Second, it must be a multiple of 4 and not multiple of 100 (year % 4 ==0 OR year % 100 != 0) – these 2 conditions when satisfied simultaneously will determine if a year is leap year or not.If it is a leap year then the function will return ‘True’ (Boolean) and if it is not a leap year then the function will return ‘False’ (Boolean)*
5. return obj % 4 == 0 and (obj % 100 !=0 or obj % 400 == 0) #*This is simply the previous explanation written in code containing 3 comparison operators and 2 boolean operators. We’ve used return to print the final status of the line of code.*
6. Def sanitize(obj1, obj2):

#*Sanitize in literal sense means ‘to clean the unwanted elements’ – this translates to a similar sense here in our script as well.We define and use this function to control what characters are allowed in our argument input and what isn’t – meaning only numbers from 0 to 9 will be allowed in the input argument (we call this allowable range obj2) and the one to be tested : obj1 – this function then using the replace function will replace any character that is present in our input argument but isn’t permissible with blank space – ultimately removing it*

for c in obj1:

if c not in obj2:

obj1 = obj1.replace(c,' ')

return(obj1)

#*size\_check function has been defined to check whether or not our input argument which essentially is a date in one of the 4 formats has 8 characters or not – we’ve used 8 characters since 4 for year, 2 for month and 2 for date add upto 8 characters.This function takes 2 arguments one of which is a string and the other one is an integer against which the length of the string is to be checked.For this purpose, we use the length function on the string object and confirm if it is of 8 characters and if not it will return an error (See later).Since this is a general purpose function the integer can be anything and doesn’t necessarily have to be 8.*

1. Def size\_check(obj,intobj):

Return len(str(obj)) == intobj

#*range\_check function has been defined to check whether the day part of the date of birth (denoted in code as index function dob[6:]) is within the permitted range of years .For the purpose of this code we’ve chose not to include people older than 120 years hence the range becomes (1900,9999) inclusive .This function takes in two arguments viz. an integer (day) and a tuple (range to be checked against) – and returns a Boolean value of whether the integer(obj1) is in the range (fixed, because it’s a tuple) defined.*

1. def range\_check(obj1, obj2):

o\_min = obj2[0]

o\_max = obj2[1]+1

return obj in range(o\_min,o\_max)

#*The usage function is nothing more than a block of functional code containing what needs to be printed at events when the user enters no arguments – and that is what it is – it is a function that doesn’t take in any argument (zero argument function) but prints a line to the user defining / stating how the script must be used from command line and what / how many arguments are to be passed to the function*

1. def usage()

Diagram, schematic

Description automatically generated

Algorithm Flowchart