PRERELEASE CODE

Stephen Yules 0061

Mr.Kieran
Lanna International School Thailand

Contents

TASK 1

Task 1.1 Pseudocode

```
DECLARE students : ARRAY[1:9] OF STRING
DECLARE name, email, dob, spaces, student : STRING
DECLARE i, n, emaillen, namegap, emailgap : INTEGER

FOR i ← 1 TO LENGTH(students)
          OUTPUT "Input the student's name."
          INPUT name
          OUTPUT "Input the student's email."
          INPUT email
          OUTPUT "Input the student's date of birth."
          INPUT dob
          students[i] ← name & "*" & email & "*' & dob
ENDFOR

spaces ← "                                              "

OUTPUT "Here are all the students with headers."
OUTPUT "Name", LEFT(spaces, 12), "Email", LEFT(spaces, 20), "Date of birth"

FOR i ← 1 TO LENGTH(students)
          student ← students[i]
          n ← 1
          WHILE MID(student, n, 1) <> "*"
                    n ← n + 1
          ENDWHILE
          name ← LEFT(student, n - 1)
          emaillen ← LENGTH(students[i]) - LENGTH(name) - 11
          email ← MID(students[i], n + 1, emaillen)
          dob ← RIGHT(student, 9)

          namegap ← 16 - LENGTH(name)
          emailgap ← 25 - emaillen
          OUTPUT name, LEFT(spaces, namegap), email, LEFT(spaces, emailgap), dob
ENDFOR
```

Commented [S1]: An array of 9 elements is used just as a base since it doesn't state the actual number of students in the class. This is done for all arrays.

Commented [S2]: Here i is used as an index to keep track of the element in the array 'students'.

Commented [S3]: Storing the student's details into the array.

Commented [S4]: This string is used to make the table format for later in the code.

Commented [S5]: This is the header for the table using spaces so that everything is in columns.

Commented [S6]: I had this code working fine multiple times, but over time I found ways to make it shorter.

Commented [S7]: Here n is used as a count to keep track of the character that is being compared to obtain the student's name.

Commented [S8]: Since at this point n is at the placement of the *, having n-1 makes it the exact length of the name.

Commented [S9]: Having the length of the entire string subtracting the length of the name, which is known, and 11 because 9 is the length of the date if the correct format is used from the question being ex: 25Sep2005 and 2 for the 2 '*' giving me the length of the email left.

Commented [S10]: n is incremented so that it skips the * and starts at the first letter of the email.

Commented [S11]: The date format used is DDMMMYYYY.

Commented [S12]: These variables allow me to know how many spaces need to be inserted into the OUTPUT so that the variables line up with the headers.

Task 1.1 Python

```
# students : ARRAY[0:8]
# variables : ARRAY[0:2]
# name, email, dateOfBirth, spaces : STRING
# i, namegap, emailgap : INTEGER

students = []

for i in range(9):
	students.append("")

for i in range(len(students)):
	name = input("Input the student's name: ")
	email = input("Input the student's email: ")
	dateOfBirth = input("Input the student's date of the birth: ")
	students[i] = name + "*" + email + "*" + dateOfBirth

spaces = "                                        "
print("Name" + spaces[:12] + "Email" + spaces[:20] + "Date Of Birth")
for student in students:
	variables = student.split("*")
	namegap = 16 - len(variables[0])
	emailgap = 25 - len(variables[1])
	print(variables[0] + spaces[:namegap] + variables[1] + spaces[:emailgap] + variables[2])
```

Commented [S13]: I use 9 elements for my arrays like with the pseudocode.

Commented [S14]: This initializes 9 spots in the array students and leaves “” as a placeholder.

Commented [S15]: For each empty spot in students, this changes the “” to the students details at the index.

Commented [S16]: This is the header to keep as a basis for the columns.

Commented [S17]: Using split, it separates the string whenever an * appears and stores it into the array variables. Ex: If the string was Stephen*gmail*green it would store Stephen, then gmail and lastly green into the array.

Commented [S18]: This is used to calculate how many spaces are needed for the table to be lined up.

Commented [S19]: Since the element for the name, email and date of birth in inside the array variables I can just print them out here.

Task 1.2 Pseudocode

```
PROCEDURE printstudents(students : ARRAY OF STRING)

    DECLARE i : INTEGER

    OUTPUT “Here are the students with their names, email and date of birth separated by *.”

    FOR i ← 1 TO LENGTH(students)
        IF students[i] <> ""
            THEN
                OUTPUT students[i]
        ENDIF
    ENDFOR
ENDPROCEDURE
```

Commented [S20]: The array ‘students’ is brought in to be used and is assumed to be full of names and empty spaces.

Commented [S21]: Checks if the element in the array is empty or not, and skipping the OUTPUT if it is empty. The table output isn’t used as it isn’t asked and would just a copy of the previous task.

Task 1.2 Python

```
# student : STRING

def printstudents(students):

    print("Here are the students with their names, email and date of birth seperated by *.")

    for student in students:
        if student != "":
            print(student)
```

Commented [S22]: Like the pseudocode, students is called from the previous question or I assume it already has been filled.

Commented [S23]: Checks if the element is empty or not, and printing if it isn’t empty.

Task 1.3 Pseudocode

```
DECLARE students : ARRAY[1:9] OF STRING
DECLARE stuname, email, dob, searchingname, tempname : STRING
DECLARE i, emaillen : INTEGER
DECLARE found : BOOLEAN

FOR i ← 1 TO LENGTH(students)
    OUTPUT "Input the student's name."
    INPUT stuname
    OUTPUT "Input the student's email."
    INPUT email
    OUTPUT "Input the student's date of birth."
    INPUT dob
    students[i] ← stuname & "*" & email & "*' & dob
ENDFOR

OUTPUT "Whose email would you like?”
INPUT searchingname

found ← FALSE
i ← 1

WHILE found = FALSE AND i <= LENGTH(students)
    tempname ← LEFT(students[i], LENGTH(searchingname))
    IF tempname = searchingname
        THEN
            emaillen ← LENGTH(students[i]) - LENGTH(tempname) - 11
            email ← MID(students[i], n + 1, emaillen)
            OUTPUT email
            found ← TRUE
        ELSE
            i ← i  + 1
    ENDIF
ENDWHILE

IF found = FALSE
    THEN
        OUTPUT “No student found.”
ENDIF
```

Commented [S24]: Filling the array with students just like Task 1.1.

Commented [S25]: This is the variable used to compare for the name.

Commented [S26]: Here i is representing the rowindex and needs to be initialized as it is incremented with no FOR loop.

Commented [S27]: 2 conditions were used as it breaks the loop if either of them turns false. If the email is found, it breaks the loop as I assume no one has the same first and last name or when the index exceeds the length of the array, meaning it wasn’t found.

Commented [S28]: This is used to obtain the name from the variable in the array, it is easier to take the same length as the searchingname than finding the actual name used in the variable.

Commented [S29]: I used a different method before to find the email but this was shorter and simpler.

Commented [S30]: Same as Task 1.1 to obtain the email and make found TRUE to break out of the loop. Otherwise, i is incremented so that the next element is tested in students.

Commented [S31]: This is used as an error message in case that the email is typed incorrectly or the student doesn’t exist instead of being blank and leaving the user confused.

Task 1.3 Python

```
# students : ARRAY[0:8]
# variables : ARRAY[0:2]
# studentName, email, dateOfBirth, searchingname : STRING
# i : INTEGER
# found : BOOLEAN

students = []

for i in range(9):
        students.append("")

for i in range(len(students)):
        studentName = input("Input the student's name: ")
        email = input("Input the student's email: ")
        dateOfBirth = input("Input the student's date of the birth: ")
        students[i] = studentName + "*" + email + "*" + dateOfBirth

searchingname = input("Input the name of the person whose email you want: ")

found = False
i = 0

while found == False and i < len(students):
        variables = students[i].split("*")
        if variables[0] == searchingname:
                print(variables[1])
                found = True
        else:
                i = i + 1

if found == False:
        print("No students found.")
```

Commented [S32]: Same as Task 1.1 Python.

Commented [S33]: This allows the user to input the student they want to search for.

Commented [S34]: Like pseudocode, i needed to be initialized as it is not incremented by a for loop.

Commented [S35]: This is comparing the current student name after splitting the element in the array with the searchingname to see if this is the correct student or not.

Commented [S36]: Printing the email as it is already found using split.

Task 1.4 Pseudocode

```
PROCEDURE searchbirthmonth(students : ARRAY OF STRING)

    DECLARE matchinglist : ARRAY[1:9] OF STRING
    DECLARE searchmonth, studate, printstring : STRING
    DECLARE matchindex, index, n : INTEGER
    DECLARE found : BOOLEAN

    OUTPUT "Please state the first 3 letters of the month you want to search for, first letter
    capitalized."
    INPUT searchmonth

    matchindex ← 1
    found ← FALSE

    FOR index ← 1 TO LENGTH(students)
        studate ← RIGHT(students[index], 7)
        IF searchmonth = LEFT(studate, 3)
            THEN
                n ← 1
                WHILE MID(students[index], n, 1) <> “*”
                    n ← n + 1
                ENDWHILE
                matchinglist[matchindex] ← LEFT(students[index], n - 1)
                matchindex ← matchindex + 1
                found ← TRUE
        ENDIF
    ENDFOR

    printstring ← “”

    IF found = FALSE
        THEN
            OUTPUT “No students found with this birth month.”
        ELSE
            FOR index ← 1 TO LENGTH(matchinglist)
                IF matchinglist[index] <> “”
                    THEN
                        printstring ← printstring & matchinglist[index] & “, ”
                ENDIF
            ENDFOR

            OUTPUT LEFT(printstring, LENGTH(printstring) - 2)
    ENDIF
ENDPROCEDURE
```

Commented [S37]: This allows me to use the ‘students’ array to find students in it.

Commented [S38]: This array is used to store the students that match with the birth month.

Commented [S39]: Matchindex is initialized as I use this to keep track of the place in the matchinglist array.

Commented [S40]: Index is used instead of i as there is also matchindex so I didn’t want to mix them up.

Commented [S41]: Taking the month out of the date as it is at the end of every line and using the same format as before which is DDMMMYYYY.

Commented [S42]: Selecting the name and inputting it into the array.

Commented [S43]: This allows me to have an error message if no students were found.

Commented [S44]: The FOR loop allows each variable in printstring to be added into a string in a nice format with commas instead of being printed 1 by 1.

Commented [S45]: This was a bit of a struggle to figure out a way to print it out nicely.

Commented [S46]: I subtract 2 from the length as the last 2 letters are a comma and space so they need to be removed when outputted.

Task 1.4 Python

```
# matchinglist : ARRAY[0:8]
# variables : ARRAY[0:2]
# searchmonth, student, studate : STRING
# found : BOOLEAN

def searchbirthmonth(students):

    searchmonth = input("Input the first 3 letters of the month you want to search for, first letter
    capitalized: ")

    found = False

    matchinglist = []

    for student in students:
        studate = student[-7:]
        if searchmonth == studate[:3]:
            variables = student.split("*")
            matchinglist.append(variables[0])
            found = True

    if found == False:
        print("No students found.")
    else:
        print(matchinglist)
```

Commented [S47]: Found is initialized as False so that only if it is found it will stay False.

Commented [S48]: This array stores the names of the people who have the correct birth month.

Commented [S49]: Storing the last 7 characters of student allows the month be at the start of it, so that the first 3 letters can be compared in the next if statement.

Commented [S50]: Appending the student’s name if it is the correct month.

Commented [S51]: This allows for a message to tell the user that there were no students if none are found.

Task 1.5.1 Pseudocode

```
DECLARE students : ARRAY[1:9, 1:5] OF STRING
DECLARE spaces : STRING
DECLARE i, namegap, emailgap, dobgap, studentIDgap : INTEGER

FOR i ← 1 TO LENGTH(students)
    OUTPUT “Input the student’s name.”
    INPUT students[i,1]
    OUTPUT “Input the student’s email.”
    INPUT students[i,2]
    OUTPUT “Input the student’s date of birth.”
    INPUT students[i,3]
    OUTPUT “Input the student’s id.”
    INPUT students[i,4]
    OUTPUT “Input the student’s tutor id.”
    INPUT students[i,5]
    OUTPUT “Student completed.”
ENDFOR

OUTPUT “Here are all the students in a table with headers.”

spaces ← “                                                      “

OUTPUT "Name", LEFT(spaces, 12), "Email", LEFT(spaces, 20), "Date of birth", LEFT(spaces, 5),
“StudentID”, LEFT(spaces, 5), “TutorID”

FOR i ← 1 TO LENGTH(students)
    namegap ← 16 - LENGTH(students[i,1])
    emailgap ← 25 - LENGTH(students[i,2])
    dobgap ← 18 - LENGTH(students[i,3])
    studentIDgap ← 14 - LENGTH(students[i,4])
    OUTPUT students[i,1], LEFT(spaces, namegap), students[i,2], LEFT(spaces, emailgap),
    students[i,3], LEFT(spaces, dobgap), students[i,4], LEFT(spaces, studentIDgap), students[i,5]
ENDFOR
```

Commented [S52]: I use the example format from the question with my own number of rows and 5 columns.

Commented [S53]: Here i represents the row index, moving onto the next row after each loop.

Commented [S54]: Since it is a 2D array, I can replace the elements in the array to be stored instead with the input.

Commented [S55]: I added this so it becomes less confusing when the user finishes a student and is able to keep track of their placement easier.

Commented [S56]: Headers for the table using the ‘spaces’ string to sort out spacing.

Commented [S57]: Subtracting the length of the variable from the length of the header so that the columns will line up properly.

Task 1.5.1 Python

```
# students : ARRAY[0:8,0:4]
# spaces : STRING
# i, namegap, emailgap, dobgap, studentIDgap : INTEGER

students = []

for i in range(9):
        students.append([])

for i in range(9):
        students[i].append(input("Input the student's name: "))
        students[i].append(input("Input the student's email: "))
        students[i].append(input("Input the student's date of birth: "))
        students[i].append(input("Input the student's student ID: "))
        students[i].append(input("Input the student's tutor ID: "))

print("")
print("Here are all the students in a table with headers.")

spaces = "                                                              "

print("Name" + spaces[:12] + "Email" + spaces[:20] + "Date of birth" + spaces[:5] + "StudentID" +
spaces[:5] + "TutorID")

for i in range(len(students)):
        namegap = 16 - len(students[i][0])
        emailgap = 25 - len(students[i][1])
        dobgap = 18 - len(students[i][2])
        studentIDgap = 14 - len(students[i][3])
        print(students[i][0] + spaces[:namegap] + students[i][1] + spaces[:emailgap] + students[i][2]
                + spaces[:dobgap] + students[i][3] + spaces[:studentIDgap] + students[i][4])
```

Commented [S58]: Adding 9 rows to the array so 9 students can be added.

Commented [S59]: Since each time it loops it moves to a new row, the user inputs the exact data and that is appended straight away in the correct order as they do not need to be concatenated. There is no need to keep track of the column index.

Commented [S60]: Same logic as Task 1.1 just with more variables.

Commented [S61]: Printing it out in a table format by taking the elements from the students array.

Task 1.5.2 Pseudocode

```
PROCEDURE printstudents(students : ARRAY OF STRING)

    DECLARE rowindex : INTEGER

    OUTPUT “Here are the students details.”

    FOR rowindex ← 1 TO LENGTH(students)
        IF students[rowindex, 1] <> “”
            THEN
                OUTPUT students[rowindex]
        ENDIF
    ENDFOR
ENDPROCEDURE
```

Commented [S62]: This condition checks if the first element is empty or not, and if it is empty doesn’t OUTPUT it as there is nothing.

Commented [S63]: I OUTPUT the entire row instead of each variable individually as each students details can be identified easier.

Task 1.5.2 Python

```
# i : INTEGER

def printstudents(students):

    print("Here are all the students.")

    for i in range(len(students)):
        if students[i][1] != "":
            print(students[i])
```

Commented [S64]: Check if the first variable is empty or not and print it if it isn’t empty.

Task 1.5.3 Pseudocode

```
DECLARE students : ARRAY[1:9, 1:5] OF STRING
DECLARE searchingname : STRING
DECLARE i : INTEGER
DECLARE found : BOOLEAN

FOR i ← 1 TO LENGTH(students)
    OUTPUT “Input the student’s name.”
    INPUT students[i,1]
    OUTPUT “Input the student’s email.”
    INPUT students[i,2]
    OUTPUT “Input the student’s date of birth.”
    INPUT students[i,3]
    OUTPUT “Input the student’s id.”
    INPUT students[i,4]
    OUTPUT “Input the student’s tutor id.”
    INPUT students[i,5]
    OUTPUT “Student completed.”
ENDFOR

OUTPUT “Who are you searching for?”
INPUT searchingname

found ← FALSE
i ← 1

WHILE found = FALSE AND i <= LENGTH(students)
    IF searchingname = students[i,1]
        THEN
            OUTPUT students[i,2]
            found ← TRUE
        ELSE
            i ← i + 1
    ENDIF
ENDWHILE

IF found = FALSE
    THEN
        OUTPUT “No student found.”
ENDIF
```

Commented [S65]: This whole inputting part is the same as Task 1.5.1

Commented [S66]: Like before, if either condition becomes false the loop breaks.

Commented [S67]: Using the 2D array is easier for these parts as there is no need to find each individual variable as they are stored individually already.

Commented [S68]: An error message in case.

Task 1.5.3 Python

```
# students : ARRAY[0:8,0:4]
# searchingname : STRING
# i : INTEGER
# found : BOOLEAN

students = []

for i in range(9):
        students.append([])

for i in range(9):
        students[i].append(input("Input the student's name: "))
        students[i].append(input("Input the student's email: "))
        students[i].append(input("Input the student's date of birth: "))
        students[i].append(input("Input the student's student ID: "))
        students[i].append(input("Input the student's tutor ID: "))
        print("Student completed.")

searchingname = input("Who are you searching for: ")

found = False
i = 0

while found == False and i < len(students):
        if students[i][0] == searchingname:
                print(students[i][1])
                found = True
        else:
                i = i + 1

if found == False:
        print("No students found.")
```

Commented [S69]: Only difference between this code and Task 1.3 is how the students are appended and instead of splitting each element, it is able to just take it from students.

Task 1.5.4 Pseudocode

```
PROCEDURE searchbirthmonth(students : ARRAY OF STRING)

    DECLARE matchinglist : ARRAY[1:9] OF STRING
    DECLARE searchmonth, studate, printstring : STRING
    DECLARE matchindex, i : INTEGER
    DECLARE found : BOOLEAN

    OUTPUT "Please state the first 3 letters of the month you want to search for, first letter
    capitalized."
    INPUT searchmonth

    found ← FALSE
    matchindex ← 1

    FOR i ← 1 TO LENGTH(students)
        studate ← students[i,3]
        IF searchmonth = MID(studate, 4, 3)
            THEN
                matchinglist[matchindex] ← students[i,1]
                matchindex ← matchindex + 1
                found ← TRUE
        ENDIF
    ENDFOR

    printstring ← ""

    IF found = FALSE
        THEN
            OUTPUT "No students found with this birth month."
        ELSE
            FOR i ← 1 TO LENGTH(matchinglist)
                IF matchinglist[i] <> ""
                    THEN
                        printstring ← printstring & matchinglist[i] & ", "
                ENDIF
            ENDFOR
            OUTPUT LEFT(printstring, LENGTH(printstring) - 2)
    ENDIF
ENDPROCEDURE
```

Commented [S70]: The only difference between this code and Task 1.4 is that I am able to take the variable directly from the array.

Commented [S71]: For the date, an example of the format is "25 Sep 2003" which is shown as an example in the prerelease, so the month would start at the 4th character here instead.

Commented [S72]: Uses the same code as Task 1.4 to print the array out nicely.

Task 1.5.4 Python

```
# matchinglist : ARRAY[0:8]
# searchmonth, studate : STRING
# i : INTEGER
# found : BOOLEAN

def searchbirthmonth(students):

    searchmonth = input("Input the first 3 letters of the month you want to search for, first letter
    capitalized: ")

    found = False

    matchinglist = []

    for i in range(len(students)):
        studate = students[i][2]
        studate = studate[-8:]
        if searchmonth == studate[:3]:
            matchinglist.append(students[i][0])
            found = True

    if found == False:
        print("No students found with this birth month.")
    else:
        print(matchinglist)
```

Commented [S73]: Instead of -7 which is used in Task 1.4, -8 is used because of the format used for date in the 2D array is "25 Sep 2008" so I need to take the 8 characters on the right.

Commented [S74]: Like the other Task 1.5 Python codes, there is no need to split.

Commented [S75]: If the matchinglist is empty where nothing was appended, it will print the message and otherwise print the array.

Task 1.6 Pseudocode

```
PROCEDURE search(students : ARRAY OF STRING)

    DECLARE choice, searchstring : STRING
    DECLARE searchtype, i : INTEGER
    DECLARE valid, found : BOOLEAN

    OUTPUT "What type of data would you like to search for? Choose from name, email, date of
    birth, studentID or tutorID."

    valid ← FALSE
    searchtype ← 0
    WHILE valid = FALSE
        INPUT choice
        CASE OF choice
            "name" : searchtype ← 1
            "email" : searchtype ← 2
            "date of birth" : searchtype ← 3
            "studentID" : searchtype ← 4
            "tutorID" : searchtype ← 5
            OTHERWISE : OUTPUT "Not a choice. Retry. "
        ENDCASE
        IF searchtype <> 0
            THEN
                valid ← TRUE
        ENDIF
    ENDWHILE

    OUTPUT "Input the data you want to search for."
    INPUT searchstring

    found ← FALSE

    FOR i ← 1 TO LENGTH(students)
        IF searchstring = students[i, searchtype]
            THEN
                OUTPUT students[i]
                found ← TRUE
        ENDIF
    ENDFOR

    IF found = FALSE
        THEN
            OUTPUT "No students found."
    ENDIF
ENDPROCEDURE
```

Commented [S76]: Searchtype represents the element the code would be comparing later with the inputted string so it knows which column to search through.

Commented [S77]: At first the code searched through every single element to see if it matched, but then I thought of this method to make it more user friendly and efficient.

Commented [S78]: Using CASE, it compares to see which variable the user wants to search for and assigns the column to searchtype.

Commented [S79]: This checks if a new variable was assigned or not, and repeating the process if it wasn't changed.

Commented [S80]: Comparing the searchstring to the variable in the array depending on the searchtype.

Commented [S81]: As the question doesn't state what is needed to be found, I just OUTPUT the entire row that equals the searchstring.

Task 1.6 Python

```
# choice, searchstring : STRING
# searchtype, i : INTEGER
# valid, found : BOOLEAN

def search(students):
	print("What type of data would you like to search for? Choose from name, email, date of birth, 
	studentID or tutorID.")

	valid = False
	searchtype = 9

	while valid == False:
		choice = input()
		if choice == "name":
			searchtype = 0
		elif choice == "email":
			searchtype = 1
		elif choice == "date of birth":
			searchtype = 2
		elif choice == "studentID":
			searchtype = 3
		elif choice == "tutorID":
			searchtype = 4
		else:
			print("Not a choice. Retry inputting.")
		if searchtype != 9:
			valid = True

	searchstring = input("Input the data you want to search for: ")

	found = False

	for i in range(len(students)):
		if searchstring == students[i][searchtype]:
			print(students[i])
			found = True

	if found == False:
		print("No students found.")
```

Commented [S82]: For python I use 9 as the base for searchtype as 0 is a placement of an element in an array unlike pseudocode where it starts at 1.

Commented [S83]: Depending on what the input of choice is, searchtype is assigned a different variable. It works like the case in pseudocode.

Commented [S84]: If the input was valid then searchtype would no longer be 9.

Commented [S85]: Since the user specified what element they want to compare data with, it only compares the specific element in each row with the searchstring, and printing the whole row if matching.

Task 2

Task 2.1 Pseudocode

```
DECLARE studentID, email, dob : STRING
DECLARE continue : CHAR

continue ← "y"

OPENFILE "students.txt" FOR WRITE

WHILE continue = "y"
        OUTPUT "Input the student ID with 2 letters followed by 4 digits."
        INPUT studentID
        OUTPUT "Input the student's email."
        INPUT email
        OUTPUT "Input the student's date of birth in DDMMYY format."
        INPUT dob
        WRITEFILE "students.txt", studentID & email & dob & "\n"
        OUTPUT "Would you like to continue? y/n"
        INPUT continue
ENDWHILE

CLOSEFULE "students.txt"
```

Commented [S86]: Since there is no set number of students, this allows the user to input as many students as they want to until they are done, since the file can be written to as many times as the user wants.

Commented [S87]: When writing the file, each variable is concatenated together with \n so that the next time anything is written to the file, it is written on the next line.

Commented [S88]: I use the input of a character instead of using TRUE and FALSE since it shortens the code and still works.

Task 2.1 Python

```
# stuID, email, dob : STRING
# keepgoing : CHAR

with open("students.txt","w") as f:
        keepgoing = "y"
        while keepgoing == "y":
                stuID = input("Input the Student ID with 2 letters followed by 4 characters: ")
                email = input("Input the email of the student: ")
                dob = input("Input the date of birth with a DDMMYY format: ")
                f.write(stuID + email + dob + "\n")
                keepgoing = input("Would you like to continue? y/n: ")
```

Commented [S89]: This opens the file as write without the need to close it later.

Commented [S90]: I also don't use continue as that is a command in Python.

Commented [S91]: Using keepgoing, the user can keep inputting students into the file until they are finished and input 'n' instead of 'y'.

Commented [S92]: Here the code writes the line into the file students.txt and concatenates it with \n since that makes a new line as well so that the next student written is on a new line.

Commented [S93]: This is the input to ask if the user wants to type in another student.

Task 2.2 Pseudocode

```
DECLARE searchID, fileline : STRING
DECLARE found : BOOLEAN

OUTPUT "Input the student ID you want to search for."
INPUT searchID

found ← FALSE

OPENFILE "students.txt" FOR READ
READLINE "students.txt", fileline

WHILE NOT EOF("students.txt") AND found =  FALSE
    IF searchID = LEFT(fileline, 6)
        THEN
            OUTPUT MID(fileline, 7, LENGTH(fileline) - 12)
            found ← TRUE
        ELSE
            READLINE "students.txt", fileline
    ENDIF
ENDWHILE

IF found = FALSE
    THEN
        OUTPUT "No student found."
ENDIF

CLOSEFILE "students.txt"
```

Commented [S94]: It reads the first line so that it can be used in the WHILE loop during the first loop, and has another READLINE at the end so when it becomes the end of the file, it doesn't need to loop through with an empty string.

Commented [S95]: The ID in the example is 6 characters long so I keep that format. Ex: AB1234

Commented [S96]: Since the ID and date are both 6 characters long, the code subtracts 12 from the total length of the fileline to find the length of the email. This prints the email.

Commented [S97]: If the ID doesn't match, it moves onto the next line in the file.

Commented [S98]: This produces a message if a student isn't found.

Task 2.2 Python

```
# searchID, fileline, email : STRING
# found : BOOLEAN

searchID = input("Input the student ID: ")
found = False

with open("students.txt","r") as f:
    fileline = f.readline()
    while found == False and fileline != "":
        if fileline[:6] == searchID:
            email = fileline[6:(len(fileline)-7)]
            print(email)
            found = True
        else:
            fileline = f.readline()

if found == False:
    print("No student found.")
```

Commented [S99]: This opens the file as read instead of write.

Commented [S100]: Readline reads the next line in the text file and assigns it to fileline.

Commented [S101]: The two conditions that can break the loop is when the studentID is found as each one should be unique or the file is empty.

Commented [S102]: The studentID will be the first 6 characters of each line so it only compares the searchID to the first 6 characters.

Commented [S103]: The last 7 characters are the 6 characters used for the date and 1 character is \n since it is read when there is a line after it. This selected the first character after the studentID to the character before the date as they are always the same format.

Commented [S104]: This was an issue at first since it took a while to get rid of the \n, since my testing variables weren't typed correctly.

Commented [S105]: If the ID isn't the same, it goes to the next line and repeats the process.

Task 2.3 Pseudocode

```
DECLARE searchID, fileline, stuID : STRING
DECLARE i : INTEGER
DECLARE found : BOOLEAN

OUTPUT “Input the substring you want to search for.”
INPUT searchID

found ← FALSE

OPENFILE “students.txt” FOR READ
READLINE “students.txt”, fileline

WHILE NOT EOF “students.txt”
    stuID ← LEFT(fileline, 6)
    FOR i ← 1 to (7 - LENGTH(searchID))
        IF searchID = MID(stuID, i, LENGTH(searchID))
            THEN
                OUTPUT fileline
                found ← TRUE
        ENDIF
    ENDFOR
    READLINE “students.txt”, fileline
ENDWHILE

IF found = FALSE
    THEN
        OUTPUT “No one was found.”
ENDIF

CLOSEFILE “students.txt”
```

Commented [S106]: There is only one condition as the substring may match with multiple student’s ID’s.

Commented [S107]: This code allows the searchID to be searched from anywhere inside the substring. It is only done as many times as possible. For example, if searchID is 6 characters long, it only needs to be compared with stuID once.

Commented [S108]: Using MID here pulls the right amount of characters out of the string, moving onto the next amount of characters after each loop.

Commented [S109]: Here the whole fileline is outputted instead in case the user wants to know the full ID of the student.

Task 2.3 Python

```
# searchID, fileline, stuID : STRING
# i : INTEGER
# found : BOOLEAN

found = False

searchID = input("Input substring: ")

with open("students.txt","r") as f:
    fileline = f.readline()
    while fileline != "":
        stuID = fileline[:6]
        for i in range(7-len(searchID)):
            if searchID == stuID[i:i+len(searchID)]:
                print(fileline)
                found = True
        fileline = f.readline()

if found == False:
    print("No one found.")
```

Commented [S110]: This code searches for the substring anywhere inside the code, at any position as long as its in the correct order.

Commented [S111]: Reads every line in the file until it is empty.

Commented [S112]: The reason I do 7 – len(searchID) is so that it loops as many times that is needed. If the searchID is 1 character long, it will loop through 6 times in stuID since there are 6 possible positions that ID can be at. On the other side, if searchID is 6 characters long, it only needs to loop once since there is only one comparable ID.

Commented [S113]: Each time it loops, it moves onto the next character and compares the same length to the searchID. However, for example, if searchID is 4 characters long it will only compare up to i = 3 since if it compares at i = 4, there is no 4th character.

Commented [S114]: Error message in case there is no student with the ID.

Task 2.4 Pseudocode

```
PROCEDURE addstudent()
    DECLARE studentID, email, dob : STRING
    DECLARE keepgoing : CHAR

    keepgoing ← "y"

    OPENFILE "students.txt" FOR APPEND

    WHILE keepgoing = "y"
        OUTPUT "Input the student ID with 2 letters followed by 4 digits."
        INPUT studentID
        OUTPUT "Input the student's email."
        INPUT email
        OUTPUT "Input the student's date of birth in DDMMYY format."
        INPUT dob
        WRITEFILE "students.txt", studentID & email & dob & "\n"
        OUTPUT "Would you like to continue? y/n"
        INPUT keepgoing
    ENDWHILE

    CLOSEFILE "students.txt"

PROCEDURE searchstudent()
    DECLARE searchID, fileline : STRING
    DECLARE found : BOOLEAN

    OUTPUT "Input the substring you want to search for."
    INPUT searchID

    found ← FALSE

    OPENFILE "students.txt" FOR READ
    READLINE "students.txt", fileline

    WHILE NOT EOF "students.txt"
        IF searchID = LEFT(fileline, LENGTH(searchID))
            THEN
            OUTPUT fileline
                found ← TRUE
        ENDIF
        READLINE "students.txt", fileline
    ENDWHILE
```

Commented [S115]: Instead of continue, keepgoing is used as continue as also used below outside of the procedure, so that it doesn't mess anything up with the code.

Commented [S116]: For the procedure addstudent(), it is the same code as Task 2.1 but opening the file for APPEND rather than WRITE as it question requests this. This means that new data is written after the current data rather than overwriting the previous data.

Commented [S117]: This procedure is the same as Task 2.3.

```
    IF found = FALSE
        THEN
            OUTPUT "No student found."
    ENDIF

    CLOSEFILE "students.txt"

DECLARE command : STRING
DECLARE continue : CHAR

continue ← "y"

WHILE continue = "y"
    OTUPUT "Enter add or search to either add a student or search a studentID."
    INPUT command
    CASE OF command
        "add" : CALL addstudent()
        "search" : CALL searchstudent()
        OTHERWISE : OUTPUT "Not a valid command."
    ENDCASE
    OUTPUT "Would you to perform another action? y/n"
    INPUT continue
ENDWHILE
```

Commented [S118]: Having the WHILE loop allows the users to do the commands as many times as they want or switch between them.

Commented [S119]: This calls the procedure that is wanted to either search or add students or give an error message if neither is chosen.

Task 2.4 Python

```
def addstudent():
    # stuID, email, dob : STRING
    # nextstudent : CHAR

    with open("students.txt","a") as f:
        nextstudent = "y"
        while nextstudent == "y":
            stuID = input("Input the Student ID with 2 letters followed by 4 characters: ")
            email = input("Input the email of the student: ")
            dob = input("Input the date of birth with a DDMMYY format: ")
            f.write(stuID + email + dob + "\n")
            nextstudent = input("Would you like to continue? y/n: ")

def searchstudent():
    # searchID, fileline, stuID : STRING
    # i : INTEGER
    # found : BOOLEAN

    found = False

    searchID = input("Input substring: ")

    with open("students.txt","r") as f:
        fileline = f.readline()
        while fileline != "":
            stuID = fileline[:6]
            for i in range(7-len(searchID)):
                if searchID == stuID[i:i+len(searchID)]:
                    print(fileline)
                    found = True
            fileline = f.readline()

    if found == False:
        print("No one found.")

# command : STRING
# keepgoing : CHAR

keepgoing = "y"

while keepgoing == "y":
    command = input("Enter add or search to either add a student or search for a studentID: ")
    if command == "add":
```

Commented [S120]: This opens the file with append rather than write as the students added are written after the old students instead of overwriting them.

Commented [S121]: Instead of keepgoing, I use nextstudent here so it doesn't get confused with the keepgoing variable I use outside of this function.

Commented [S122]: Inputting and writing to the file are the same as Task 2.1.

Commented [S123]: This function is the exact same code as Task 2.3.

Commented [S124]: This variable is used in the same way as Task 2.1 where the user can do the commands as many times as they want.

```
        addstudent()
elif command == "search":
        searchstudent()
else:
        print("Command not available.")
keepgoing = input("Would you like to input another command? y/n: ")
```

Commented [S125]: After the command is inputted it can call the function corresponding with their command or receive an error message where it wasn't found.

Commented [S126]: Asking the user if they want to do another command.

Task 2.5 Pseudocode

```
DECLARE stuID, email, dob : STRING
DECLARE validcount, i : INTEGER
DECLARE valid : BOOLEAN

valid ← FALSE

WHILE valid = FALSE
    OUTPUT “Input the student ID, 2 letters followed by 4 digits.”
    INPUT stuID
    IF LENGTH(stuID) <> 6
        THEN
            OUTPUT “Not right length. Retry.”
        ELSE
            validcount ← 0
            FOR i ← 1 TO 2
                IF MID(stuID, i, 1) >= “A” AND MID(stuID, i, 1) <= “Z”
                    THEN
                        validcount ← validcount + 1
                ENDIF
            ENDFOR
            FOR i ← 1 TO 4
                IF MID(stuID, 2 + i, 1) >= “0” AND MID(stuID, 2 + i, 1) <= “9”
                    THEN
                        validcount ← validcount + 1
                ENDIF
            ENDFOR
            IF validcount = 6
                THEN
                    valid ← TRUE
                ELSE
                    OUTPUT “Invalid student ID. Retry.”
            ENDIF
    ENDIF
ENDWHILE

Output “Input the student’s email.”
INPUT email

valid ← FALSE

WHILE valid = FALSE
    OUTPUT “Input the date of birth in DDMMYY format.”
    INPUT dob
    IF LENGTH(dob) <> 6
        THEN
```

Commented [S127]: The format being used is 2 letters followed by 4 digits.

Commented [S128]: This checks if the length of the ID is correct and if not produces an error message.

Commented [S129]: Validcount is an integer that is incremented every time

Commented [S130]: These 2 conditions checks if the character is actually a letter or not, since it is between A and Z.

Commented [S131]: These 2 conditions checks if the character is between 0 and 9, making it a digit and increment validcount if it is.

Commented [S132]: If every character was valid then validcount should equal 6, if it isn’t the user will re-input.

Commented [S133]: No validation is needed for email as emails can be widely different with characters, digits and symbols.

Commented [S134]: The date of birth format is used from the question being DDMMYY.

Commented [S135]: Like with the studentID, the dob should be 6 characters long.

```
                        OUTPUT "Incorrect length. Retry."
                ELSE
                        validcount ← 0
                        IF MID(dob, 1, 1) >= "0" AND MID(dob, 1, 1) <= "3"
                                THEN
                                        validcount ← validcount + 1
                        ENDIF
                        IF MID(dob, 2, 1) >= "0" AND MID(dob, 2, 1) <= "9"
                                THEN
                                        validcount ← validcount + 1
                        ENDIF
                        IF MID(dob, 3, 1) >= "0" AND MID(dob, 3, 1) <= "1"
                                THEN
                                        validcount ← validcount + 1
                        ENDIF
                        FOR i ← 1 TO 3
                                IF MID(dob, i + 3, 1) >= "0" AND MID(dob, i + 3, 1) <= "9"
                                        THEN
                                                validcount ← validcount + 1
                                ENDIF
                        ENDFOR
                        IF validcount = 6
                                THEN
                                        valid ← TRUE
                                ELSE
                                        OUTPUT "Invalid date. Retry."
                        ENDIF
        ENDIF
ENDWHILE

OPENFILE "students.txt" FOR APPEND

WRITEFILE "students.txt", stuID & email & dob & "\n"

CLOSEFILE "students.txt"
```

Commented [S136]: These IF statements check the first 2 letters being the day. The largest day possible is 31, so the first character can be between 0 and 3 and the second character being any digit.

Commented [S137]: For the first letter of the month, it can only be 0 or 1.

Commented [S138]: Since these last 3 characters can be digits between 0 and 9, I put them into a for loop to shorten the code.

Commented [S139]: Validcount is used the same way as before, making sure all 6 characters are valid.

Commented [S140]: This code just appends the student into the file

Task 2.5 Python

```
# stuID, email, dob : STRING
# validcount, i : INTEGER
# valid : BOOLEAN

valid = False

while valid == False:
    stuid = input("Input the Student ID with 2 letters followed by 4 characters: ")
    if len(stuID) != 6:
        print("Not right length.")
    else:
        validcount = 0
        for i in range(2):
            if stuID[i] >= "A" and stuID[i] <= "Z":
                validcount = validcount + 1
        for i in range(2,6):
            if stuID[i] >= "0" and stuID[i] <="9":
                validcount = validcount + 1
        if validcount == 6:
            valid = True
        else:
            print("Invalid ID. Retry.")

email = input("Input the email of the student: ")

valid = False

while valid == False:
    dob = input("Input the date of birth with a DDMMYY format: ")
    if len(dob) != 6:
        print("Not right length. Retry.")
    else:
        if dob[0] < "0" or dob[0] > "3":
            print("Invalid date of birth. Retry.")
        elif dob[1] < "0" or dob[1] > "9":
            print("Invalid date of birth. Retry.")
        elif dob[2] < "0" or dob[2] > "1":
            print("Invalid date of birth. Retry.")
        elif dob[3] < "0" or dob[3] > "9":
            print("Invalid date of birth. Retry.")
        elif dob[4] < "0" or dob[4] > "9":
            print("Invalid date of birth. Retry.")
        elif dob[5] < "0" or dob[5] > "9":
```

Commented [S141]: Making sure the user has to input a valid studentID to move on.

Commented [S142]: Checks if the studentID is the correct length. Like the pseudocode, format is 2 letters followed by 4 digits.

Commented [S143]: Used in the same way as the pseudocode, a count to make sure each character in the studentID is valid.

Commented [S144]: This checks if the first 2 characters are letters and if so, incrementing validcount if it is.

Commented [S145]: Checking if the 4 characters are digit or not.

Commented [S146]: Only if validcount is 6, the loop will be broken.

Commented [S147]: Email doesn't need any validation as they can be widely different.

Commented [S148]: Like the studentID, the date of birth must also be 6 characters long.

Commented [S149]: If any condition is true in this if, the loop will break and the user would need to input another dob. If all conditions are met it becomes True and it is good.

Commented [S150]: The first 2 characters should be at most 31 since that is the most days possible, so they are compared if the characters are in the correct ranges.

Commented [S151]: These 2 digits represent the months, so it should be at most 12.

```
                print("Invalid date of birth. Retry.")
            else:
                valid = True

with open("students.txt", "a") as f:
    f.write(stuID + email + dob + "\n")
```

Commented [S152]: These last 2 characters just need to be digits.

Commented [S153]: Writing the valid string into the file.