**

Syrian Arab Republic

Ministry of Higher Education

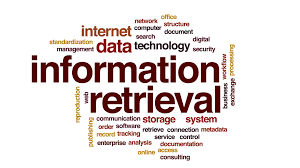
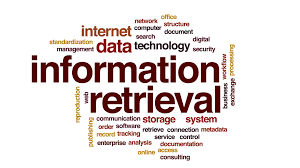
Syrian Virtual University

الجمهورية العربية السورية

الجامعة الافتراضية السورية

وزارة التعليم العالي

وظيفة مقرر استرجاع البيانات

****

**أستاذ المقرر الدكتور: باسل الخطيب.**

**رمز المادةSIR.601**

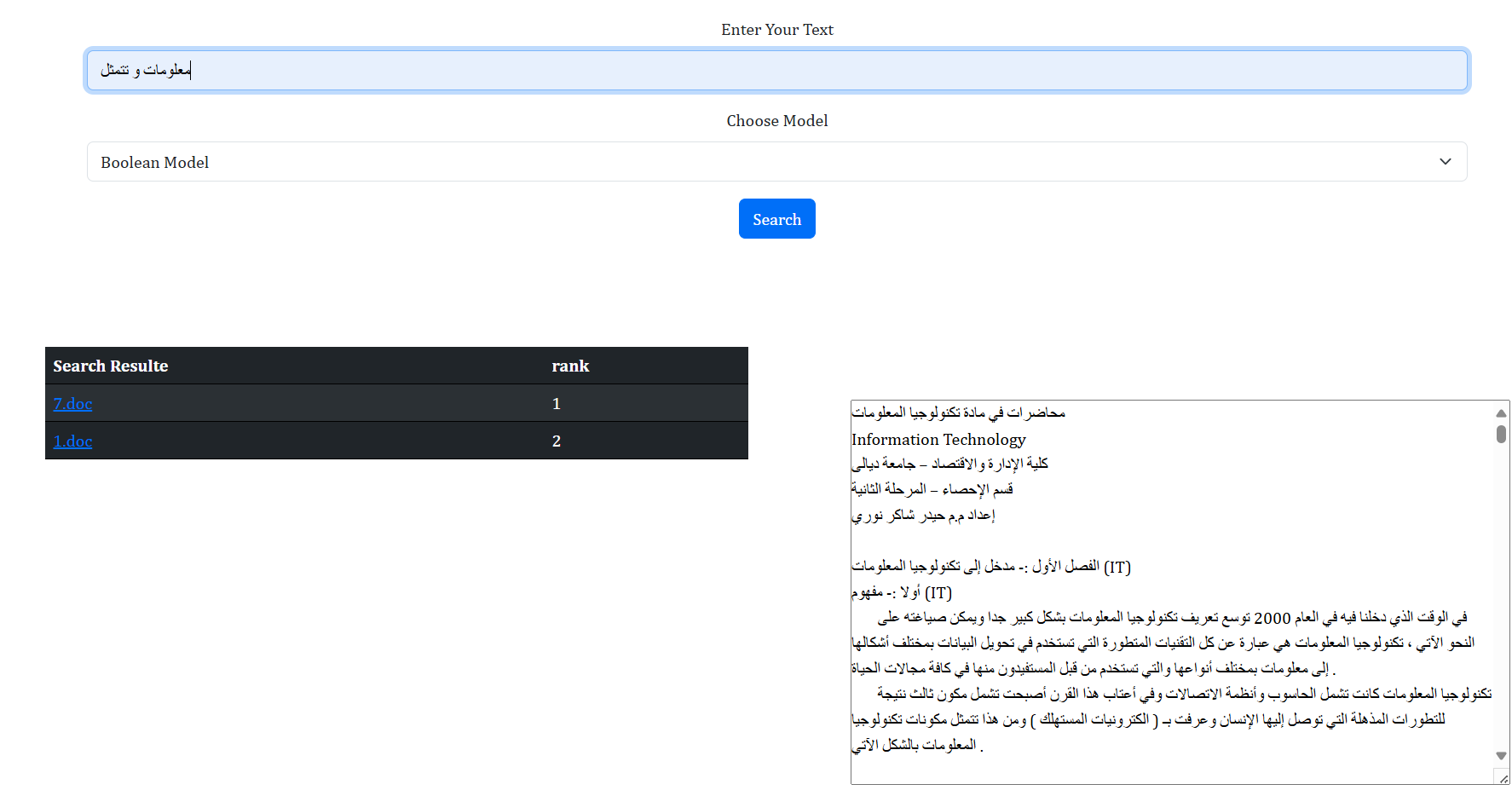
**الفصل والصف: c1S23\_.**

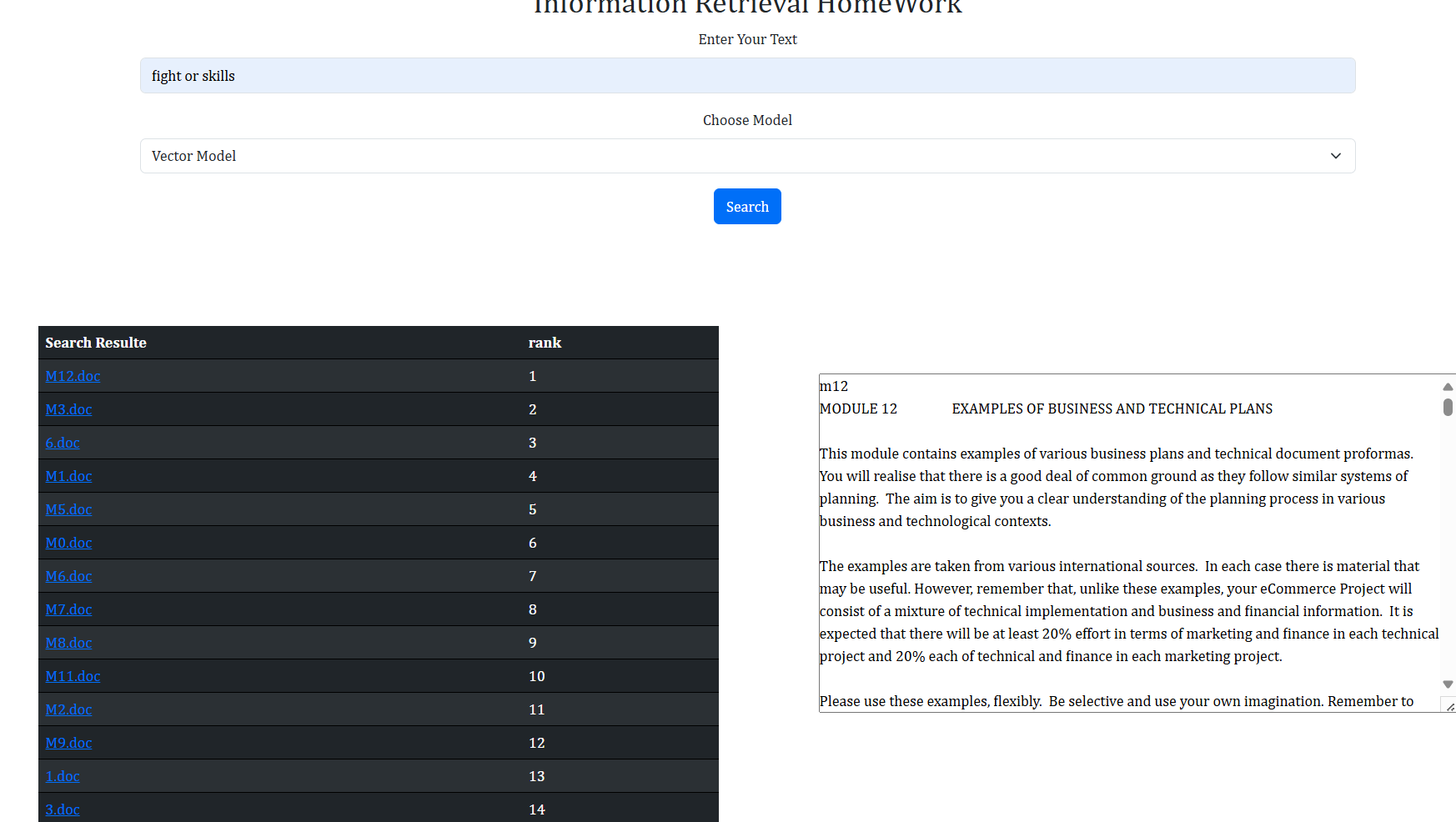
**اسماء الطلاب المشاركة بالوظيفة:**

1. **طارق بكري (Tarek\_131341)**
2. **ريم الجزائري ( Rim\_113780)**
3. **طارق بكري (Tarek\_131341)**
4. **جودت مروان سويدان ( Jawdat\_128503 )**

**أولاً: سنبدأ باستعراض واجهة المستخدم**

|  |
| --- |
|  |
| يُطلب من المستخدم أولاً تزويد النظام بالكلمات التي يرغب في البحث عنها، بعد ذلك، يتعين عليه اختيار الخوارزمية المناسبة لطريقة البحث التي يرغب في تنفيذها، حيث يمكنه اختيارمن بين الثلاث خوارزميات المتاحة.  و هنا سنستعرض مثال لنتائج البحث باللغتين العربية و الانجليزية |





**ثانيا:قرائة الملفات**

import python com

from docx import Document

import win32com.client

import os

def read\_doc\_file(doc\_path):

    pythoncom.CoInitialize()  # Initialize the COM library

    BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

    doc\_path = os.path.join(BASE\_DIR, 'SIR\_app/Data/{0}'.format(doc\_path))

    try:

        if doc\_path.endswith('.docx'):

            # For .docx files, use the python-docx library

            doc = Document(doc\_path)

            paragraphs = [paragraph.text for paragraph in doc.paragraphs]

            return '\n'.join(paragraphs)

        elif doc\_path.endswith('.doc'):

            # For .doc files, use the pywin32 library

            word\_app = win32com.client.Dispatch('Word.Application')

            doc = word\_app.Documents.Open(doc\_path)

            content = doc.Content.Text

            doc.Close()

            return content

        else:

            print(f"Unsupported file format: {doc\_path}")

            return None

    finally:

        pythoncom.CoUninitialize()  # Uninitialize the COM library

في هذا الكود قمنا بإستدعاء المكتبات الضرورية لقراءة الملفات المتاحة لدينا، وقمنا بإنشاء دوال مخصصة تقوم بقراءة المحتوى الموجود في تلك الملفات

**ثالثا:برمجة الخولرزمية البوليانية (boolean modle)**

def preprocess(text):

    text = text.lower()

    text = re.sub(r"[^\w\s]", "", text)

    words = text.split()

    return set(words)

يقوم هذا التايع بتنقيح الملفات النصية من خلال تحويل الاحرف الكبيرة الى صغيرة و ازالة علامات الترقيم مما يؤدي الى تحويل النص الى مجموعة من الكلمات

def create\_index(documents):

    index = {}

    for doc, text in documents.items():

        words = preprocess(text)

        for word in words:

            index.setdefault(word, []).append(doc)

    return index

|  |  |
| --- | --- |
| preprocess | هذه الدالة تقوم بانشاء فهرس للوثائق المعطاة و تقسم النص الى كلمات باستخدم التابع  ثم تعيين مفاتيح الفهرس باستخدام الكلمات |

def parse\_query(query):

    query = preprocess(query)

    stack = []

    precedence = {"not": 3, "and": 2, "or": 1}

    for word in query:

        if word in precedence:

            while stack and stack[-1] in precedence and precedence[word] <= precedence[stack[-1]]:

                yield stack.pop()

            stack.append(word)

        else:

            yield word

    while stack:

        yield stack.pop()

يقوم هذا التابع بتحليل النص المدخل و تحديدة اولوية العمليات البوليانية

def evaluate\_query(query, index):

    query = parse\_query(query)

    stack = []

    for word in query:

        if word in {"not", "and", "or"}:

            if stack:

                a = stack.pop()

            else:

                a = set()

            if stack:

                b = stack.pop()

            else:

                b = set()

            if word == "not":

                stack.append(a - b)

            elif word == "and":

                stack.append(a & b)

            elif word == "or":

                stack.append(a | b)

        else:

            stack.append(set(index.get(word, [])))

    return stack.pop() if stack else set()

تقوم هذه الاجرائية بتقيم النص المدخل و تستخدم الفهارس التي انشأناها مسبقا لاسترجاع الوثائق ذات الصلة بنائا على العمليات المنطقية المدخلة

**رابعا :برمجة الخولرزمية البوليانية الموسعة (boolean modleextended )**

def preprocess(text):

    return re.findall(r'\b\w+\b', text.lower())

تقوم بتنقيح النص النصي عن طريق تحويل النص إلى أحرف صغيرة وإزالة جميع الأحرف غير الحروف الأبجدية.

def calculate\_weight(term, document, documents):

    tf = document.count(term)

    idf = math.log(len(documents) / (sum(term in d for d in documents.values()) + 1e-10))

    return tf \* idf

تقوم بحساب وزن الكلمات بنائا على المعادلة tf\*idf

def create\_index(documents):

    index = defaultdict(dict)

    for filename, content in documents.items():

        for term in set(preprocess(content)):

            index[term][filename] = calculate\_weight(term, content, documents)

    return index

تقوم بتقسم النص الى كلمات باستخدام الدالة preprocessor و فهرسة الكلمات و حساب وزن كل كلمة باستخدام التابع calculate\_weight

def parse\_query(query):

    operators = {'or': 0, 'and': 1, 'not': 2}

    stack = []

    output = []

    for term in query.split():

        if term not in operators:

            output.append(term)

        else:

            while stack and stack[-1] in operators and operators[term] <= operators[stack[-1]]:

                output.append(stack.pop())

            stack.append(term)

    while stack:

        output.append(stack.pop())

    return output

def evaluate\_query(expression, index, all\_docs):

    stack = []

    for term in expression:

        if term in {'and', 'or', 'not'}:

            if not stack:

                print(f"Error: operator '{term}' without sufficient operands.")

                return

            b = stack.pop()

            if term == 'not':

                stack.append(not\_operation(b, all\_docs))

            else:

                if not stack:

                    print(f"Error: operator '{term}' without sufficient operands.")

                    return

                a = stack.pop()

                stack.append(boolean\_operation(a, b, term))

        else:

            stack.append(index.get(term, {}))

    return sorted(stack[0], key=stack[0].get, reverse=True) if stack else []

def not\_operation(docs, all\_docs):

    return {doc: weight for doc, weight in all\_docs.items() if doc not in docs}

def boolean\_operation(docs1, docs2, op):

    if op == 'and':

        return {doc: min(docs1.get(doc, 0), docs2.get(doc, 0)) for doc in set(docs1) & set(docs2)}

    elif op == 'or':

        return {doc: max(docs1.get(doc, 0), docs2.get(doc, 0)) for doc in set(docs1) | set(docs2)}

تحليل النص المدخل و تحديد اولوية العمليات البوليانية ثم تقيم الاستعلام بنائا على الفهرس و الوزن المنشئ سابقا

**خامسا :برمجة خولرزمية النموذج الشعاعي ( modlevector )**

def calculate\_tf(term, document):

    return document.split().count(term)

def calculate\_idf(term, documents):

    if isinstance(documents, set):

        document\_count = 1 if term in documents else 0

    else:

        document\_count = sum(1 for doc in documents.values() if term in doc)

    return math.log(len(documents) / (document\_count + 1e-10))

def create\_vector(document, query\_terms):

    vector = {}

    for term in query\_terms:

        tf = calculate\_tf(term, document)

        idf = calculate\_idf(term, {document})  # Pass a set containing the current document

        vector[term] = tf \* idf

    # Remove terms with zero vector values

    vector = {term: value for term, value in vector.items() if value != 0}

    return vector

نقوم اولا بانشاء شعاع عن طريق حساب عدد مرات تكرار الكلمة في نص و معكوسه ثم استخدام المعادلة التالية tf\*idf

و تزيل المتجهات التي قيمتها تساوي صفر

def cosine\_similarity(vector1, vector2):

    dot\_product = sum(vector1.get(term, 0) \* vector2.get(term, 0) for term in set(vector1) & set(vector2))

    magnitude1 = math.sqrt(sum(value \*\* 2 for value in vector1.values()))

    magnitude2 = math.sqrt(sum(value \*\* 2 for value in vector2.values()))

    return dot\_product / (magnitude1 \* magnitude2 + 1e-10)

def rank\_documents(query\_vector, document\_vectors):

    rankings = [(doc, cosine\_similarity(query\_vector, doc\_vector)) for doc, doc\_vector in document\_vectors.items()]

    rankings.sort(key=lambda x: x[1], reverse=True)

    return rankings

هنا سنقوم بحساب التشابه الكوسيني من خلال التابع cosine\_similarity و بعد ذلك يتم ترتيب المستندات بنائا على التشابه الكوسيني مع الاستعلام المدخل