التدريبات العملية لمقرر الذكاء الاصطناعي ١-١

تدريبات الوحدة الأولى

أساسيات الذكاء الاصطناعي

عدد تدريبات الوحدة ١٩

تدريبات اليوم الأول ١٥

١-١ : حذف و إضافة عناصر للمكدس : ص ٣٠

شرح الأمر البرمجي	الأمر البرمجي
تعريف المكدس و ادخال عناصره	myStack=[1,21,32,45]
طباعة عناصر المكدس	print("Initial stack: ", myStack)
طباعة العنصر المحذوف من المكدس	print(myStack.pop())
طباعة عناصر المكدس الجديد	print("The new stack after pop: ", myStack)
اضافة عنصر جديد للمكدس	myStack.append(78)
طباعة المكدس النهائي	print("The new stack after push: ", myStack)

الكود البرمجي بالكامل

```
myStack=[1,21,32,45]
print("Initial stack: ", myStack)
print(myStack.pop())
print(myStack.pop())
print("The new stack after pop: ", myStack)
myStack.append(78)
print("The new stack after push: ", myStack)
```

١-٢ : حذف جميع عناصر المكدس : ص ٣٠

شرح الأمر البرمجي	الأمر البرمجي
تعريف المكدس و ادخال عناصره	myStack=[1,21,32,45]
طباعة عناصر المكدس	print("Initial stack: ", myStack)
عرض طول المكدس	a=len(myStack)
طباعة طول المكدس	print("size of stack",a)
حلقة تكرارية لحذف جميع عناصر المكدس و طباعته	# empty the stack for i in range(a): myStack.pop() print(myStack)
حذف عنصر من مكدس فارغ !	myStack.pop()

الكود البرمجي بالكامل

```
myStack=[1,21,32,45]
print("Initial stack:", myStack)
a=len(myStack)
print("size of stack",a)
# empty the stack
for i in range(a):
  myStack.pop()
print(myStack)
myStack.pop()
```

١-٣ : إضافة و حذف عناصر من المكدس وفقا لقائمة إجراءات : ص ٣١

الأمر البرمجي	شرح الأمر البرمجي
def push(stack,element): stack.append(element)	دالة إضافة عنصر للمكدس

def pop(stack):
  return stack.pop()
def isEmpty(stack):
  return len(stack)==0
	دالة حذف عنصر من المكدس
	دالة اذا كان المكدس فارغ

def createStack(): return []	دالة انشاء مكدس
newStack=createStack()	انشاء مكدس جديد بدون عناصر (لحين اختيار احد الإجراءات)

while True:
  print("The stack so far is:",newStack)
  print("----------")
  print("Choose 1 for push")
  print("Choose 2 for pop")
  print("Choose 3 for end")
  print("----------")
  choice=int(input("Enter your choice: "))
	طباعة خيارات الإجراءات الثلاث المطلوبة كتوضيح للمستخدم

while choice!=1 and choice!=2 and choice!=3: print ("Error")	طباعة رسالة خطأ لأي رقم مخالف للإجراءات الثلاث
choice=int(input("Enter your choice: "))	السماح للمستخدم بإدخال رقم الاجراء

if choice==1:
  x=int(input("Enter element for push: "))
  push(newStack,x)
elif choice==2:
  if not isEmpty(newStack):
  print("The pop element is:",pop(newStack))
  else:
  print("The stack is empty")
else:
  print("End of program")
  break;
	دالة شرطية لتنفيذ الأمر الصحيح وفقا لرقم الاجراء الذي ادخله المستخدم

الكود البرمجي بالكامل

```
def push(stack,element):
  stack.append(element)
def pop(stack):
  return stack.pop()
def isEmpty(stack):
  return len(stack)==0
def createStack():
  return []
newStack=createStack()
while True:
  print("The stack so far is:",newStack)
  print("----------")
  print("Choose 1 for push")
  print("Choose 2 for pop")
  print("Choose 3 for end")
  print("----------")
  choice=int(input("Enter your choice: "))
  while choice!=1 and choice!=2 and choice!=3:
    print ("Error")
    choice=int(input("Enter your choice: "))
  if choice==1:
    x=int(input("Enter element for push: "))
```

```
    push(newStack,x)
  elif choice==2:
    if not isEmpty(newStack):
      print("The pop element is:",pop(newStack))
    else:
      print("The stack is empty")
  else:
    print("End of program")
    break;
```

١-٤ : حذف و إضافة عناصر للطابور : ص ٣٦

شرح الأمر البرمجي	الأمر البرمجي
تعريف الطابور و ادخال عناصره	myQueue=[1,21,32,45]
طباعة عناصر الطابور	print("Initial queue: ", myQueue)
حذف العنصر من الطابور وفقا للمعامل المكتوب	myQueue.pop(0)
طباعة عناصر الطابور الجديد	print("The new queue after pop: ", myQueue)
اضافة عنصر جديد للطابور	myQueue.append(78)
طباعة الطابور النهائي	print("The new queue after push: ", myQueue)

الكود البرمجي بالكامل

```
myQueue=[1,21,32,45]
print("Initial queue: ", myQueue)
myQueue.pop(0)
myQueue.pop(0)
print("The new queue after pop: ", myQueue)
myQueue.append(78)
print("The new queue after push: ", myQueue)
```

١-٥ : حذف جميع عناصر الطابور : ص ٣٦

شرح الأمر البرمجي	الأمر البرمجي
تعريف الطابور و ادخال عناصره	myQueue=[1,21,32,45]
طباعة عناصر الطابور	print("Initial queue: ", myQueue)
عرض طول الطابور	a=len(myQueue)
طباعة طول الطابور	print("size of queue ",a)
حلقة تكرارية لحذف جميع عناصر الطابور و طباعته	# empty the queue for i in range(a): myQueue.pop(0) print(myQueue)
حذف عنصر من طابور فارغ !	myQueue.pop(0)

الكود البرمجي بالكامل

```
myQueue=[1,21,32,45]
print("Initial queue: ", myQueue)
a=len(myQueue)
print("size of queue ",a)
# empty the queue
for i in range(a):
```

```
myQueue.pop(0)
print(myQueue)
myQueue.pop(0)
```

١-٦ : انشاء طابور بوظيفة وحدة نمطية : ص ٣٧

شرح الأمر البرمجي	الأمر البرمجي
استيراد مكتبة الطابور	from queue import*
انشاء طابور فارغ	myQueue = Queue()

ادخال عناصر الطابور بوظيفة وحده نمطية و طباعتها	# add the elements in the queue
	myQueue.put("a")
	myQueue.put("b")
	myQueue.put("c")
	myQueue.put("d")
	myQueue.put("e")
	# print the elements of the queue
	for element in list(myQueue.queue):
	  print(element)

الكود البرمجي بالكامل

```
from queue import*
myQueue = Queue()
# add the elements in the queue
myQueue.put("a")
myQueue.put("b")
myQueue.put("c")
myQueue.put("d")
myQueue.put("e")
# print the elements of the queue
for element in list(myQueue.queue):
  print(element)
```

١-٧ : انشاء طابور بوظيفة وحدة نمطية لقيم يدخلها المستخدم : ص ٣٨

شرح الأمر البرمجي	الأمر البرمجي
استيراد مكتبة الطابور	from queue import*
انشاء طابور فارغ	myQueue = Queue()

ادخال عناصر الطابور بوظيفة وحده نمطية و طباعتها	for i in range(5):
	  element=input("enter queue element: ")
	  myQueue.put(element)

طباعة عناصر الطابور داخل قائمة	for element in list(myQueue.queue): print(element)
طباعة حجم الطابور	print ("Queue size is: ",myQueue.qsize())

الكود البرمجي بالكامل

```
from queue import *
myQueue = Queue()
for i in range(5):
element=input("enter queue element: ")
myQueue.put(element)
for element in list(myQueue.queue):
print(element)
print ("Queue size is: ",myQueue.qsize())
```

١-٨ : التحقق من أن الطابور ممتلئ أو فارغ: ص ٣٨

شرح الأمر البرمجي	الأمر البرمجي
استيراد مكتبة الطابور	import queue
انشاء طابور فارغ	myQueue = Queue()
ادخال عناصر الطابور بوظيفة وحده نمطية	myQueue.put("a") myQueue.put("b") myQueue.put("c") myQueue.put("d") myQueue.put("e")
طباعة نتيجة التحقق من امتلاء الطابور بوظيفة وحده نمطية	checkFull=myQueue.full() print("Is the queue full? ", checkFull)
طباعة نتيجة التحقق من فراغ الطابور بوظيفة وحده نمطية	checkEmpty= myQueue.empty() print("Is the queue empty? ", checkEmpty)

الكود البرمجي بالكامل

```
import  queue
myQueue = Queue()
myQueue.put("a")
myQueue.put("b")
myQueue.put("c")
myQueue.put("d")
myQueue.put("e")
checkFull=myQueue.full()
print("Is the queue full? ", checkFull)
checkEmpty= myQueue.empty()
print("Is the queue empty? ", checkEmpty)
```

١-٩ : انشاء مكدس و حذف عناصره بوظيفة وحدة نمطية : ص ٣٩

شرح الأمر البرمجي	الأمر البرمجي
استيراد مكتبة الطابور	from queue import*
انشاء مكدس جديد بوظيفة وحدة نمطية	myStack = LifoQueue()
ادخال عناصر المكدس بوظيفة وحده نمطية	myStack.put("a") myStack.put("b") myStack.put("c") myStack.put("d") myStack.put("e")

طباعة عناصر المكدس المحذوفة بوظيفة وحدة نمطية وفقا لقاعدة LIFO	for i in range(5):
	  k=myStack.get()
	  print(k)

طباعة نتيجة التحقق من فراغ المكدس بوظيفة وحده نمطية	checkEmpty= myStack.empty() print("Is the stack empty?", checkEmpty)

الكود البرمجي بالكامل

```
from queue import *
myStack = LifoQueue()
myStack.put("a")
myStack.put("b")
myStack.put("c")
myStack.put("d")
myStack.put("e")
for i in range(5):
  k=myStack.get()
```

```
print(k)
checkEmpty= myStack.empty()
print("Is the stack empty?", checkEmpty)
```

١٠-١ : محاكاة طابور طباعة الملفات : ص ٤٠-٤١

المتغيرات المستخدمة في الأمر البرمجي

printDocument (متغير لطباعة الوثائق) و printQueueSize (حجم طابور الطباعة الحالي) و
printQueueMaxSize (أقصى حجم لطابور الطباعة). كائن(طابور الطباعة) printQueue باستخدام فئة Queue
من مكتبة queue، وتحديد الحجم الأقصى, دالة ١ (فاحصة) addDocument تضيف المستند لطابور الطباعة
دالة ٢ (فاحصة) printDocument تطبع المستند من طابور الطباعة

شرح الأمر البرمجي	الأمر البرمجي
استيراد مكتبة "queue" المستخدمة لإنشاء هيكل البيانات "طابور". استيراد مكتبة "time" لاستخدام وظيفة "sleep" لإضافة تأخير في البرنامج.	from queue import * import time
تهيئة المتغيرات	printDocument = " " printQueueSize = 0 printQueueMaxSize = 7 printQueue = Queue(printQueueMaxSize)
يتم تعريف دالة ١ "addDocument" لإضافة مستند إلى طابور الطباعة. تقوم الدالة بفحص إذا ما كان طابور الطباعة ممتلئًا أم لا باستخدام الدالة "qsize" للحصول على حجم الطابور ومقارنته بالحجم الأقصى	def addDocument(document): printQueueSize = printQueue.qsize()

شرح الأمر البرمجي	الأمر البرمجي
إذا كان الطابور ممتلئًا، يتم طباعة رسالة تشير إلى أن المستند لن يتم إضافته لأن طابور الطباعة ممتلئ	if printQueueSize == printQueueMaxSize: print("!! ", document, " was not sent to print queue.") print("The print queue is full.") print() return

وإلا يتم إضافة المستند إلى طابور الطباعة باستخدام الدالة "put" ويتم استدعاء الدالة "sleep" لإضافة تأخير لمدة ٠,٥ ثانية لمحاكاة الوقت المستغرق لإضافة المستند و طباعة رسالة توضح ارسال المستند للطابور	printQueue.put(document)
	time.sleep(0.5) #Wait 5.0 seconds
	print(document, " sent to print queue.")
	printQueueSizeMessage()

تعريف دالة٢ "printDocument" لطباعة المستند التالي في طابور الطباعة تقوم الدالة٢ بفحص إذا ما كان طابور الطباعة فارغًا أم لا باستخدام الدالة "qsize" للحصول على حجم الطابور ومقارنته بالقيمة صفر.	def printDocument(): printQueueSize = printQueue.qsize()
إذا كان الطابور فارغًا، يتم طباعة رسالة تشير إلى أن الطابور فارغ	if printQueueSize == 0: print("!! The print queue is empty.") print() return
وإلا يتم استدعاء الدالة "get" لاسترداد الوثيقة التالية	printDocument = printQueue.get() time.sleep(1) # wait one second

في الطابور ويتم استدعاء الدالة "sleep" لإضافة تأخير لمدة ثانية واحدة لمحاكاة الوقت المستغرق لطباعة المستندات	print ("OK - ", printDocument, " is printed.") printQueueSizeMessage()

طباعة المستندات بالترتيب الذي تم اضافتها في الكود	def printQueueSizeMessage():
	printQueueSize = printQueue.qsize()
	if printQueueSize == 0:
	print ("There are no documents waiting for printing.")
	elif printQueueSize == 1:
	print ("There is 1 document waiting for printing.")
	else:
	print ("There are ", printQueueSize, " documents waiting for printing.")
	
	print()
	addDocument("Document A")
	addDocument("Document B")
	addDocument("Document C")
	addDocument("Document D")
	addDocument("Document E")
	addDocument("Document F")
	addDocument("Document G")
	printDocument()
	addDocument("Document H")
	printDocument()
	addDocument("Document I")
	printDocument()
	addDocument("Document J")
	addDocument("Document K")
	printDocument()
	printDocument()
	printDocument()
	printDocument()
	printDocument()
	printDocument()
	printDocument()
	printDocument()

الكود البرمجي بالكامل

```
from queue import *
import time
printDocument = " "
printQueueSize = 0
printQueueMaxSize = 7
printQueue = Queue(printQueueMaxSize)


# add a document to print the queue
def addDocument(document):
printQueueSize = printQueue.qsize()
if printQueueSize == printQueueMaxSize:
```

```
print("!! ", document, " was not sent to print queue.")
print("The print queue is full.")
print()
return
printQueue.put(document)
time.sleep(0.5) #Wait 5.0 seconds
print(document, " sent to print queue.")
printQueueSizeMessage()


# print a document from the print queue
def printDocument():
printQueueSize = printQueue.qsize()
if printQueueSize == 0:
print("!! The print queue is empty.")
print()
return
printDocument = printQueue.get()
time.sleep(1) # wait one second
print ("OK - ", printDocument, " is printed.")
printQueueSizeMessage()


# print a message with the size of the queue
def printQueueSizeMessage():
printQueueSize = printQueue.qsize()
if printQueueSize == 0:
print ("There are no documents waiting for printing.")
elif printQueueSize == 1:
print ("There is 1 document waiting for printing.")
else:
print ("There are ", printQueueSize, " documents waiting for printing.")


print()
addDocument("Document A")
addDocument("Document B")
addDocument("Document C")
addDocument("Document D")
addDocument("Document E")
addDocument("Document F")
addDocument("Document G")
printDocument()
addDocument("Document H")
printDocument()
addDocument("Document I")
printDocument()
addDocument("Document J")
addDocument("Document K")
printDocument()
printDocument()
printDocument()
printDocument()
printDocument()
```

```
printDocument()
printDocument()
printDocument()
```

١-١١ : انشاء عقدة باستخدام الفئة : ص ٤٦

شرح الأمر البرمجي	الأمر البرمجي
تعريف فئة العقدة و خصائصها	class Node: def __init__(self, data, next=None):
بيان العقدة	self.data = data # node data
مؤشر العقدة الأخرى	self.next = next # Pointer to the next node
تخزين بيان العقدة و طباعتها	first = Node("Monday") print(first.data)

الكود البرمجي بالكامل

```
class Node:
def __init__(self, data, next=None):
self.data = data # node data
self.next = next # Pointer to the next node
# Create a single node
first = Node("Monday")
print(first.data)
```

١-١٢ : انشاء قائمة مترابطة بعقدة واحدة : ص ٤٦

شرح الأمر البرمجي	الأمر البرمجي
تعريف فئة العقدة و خصائصها	class Node: def __init__(self, data = None, next=None):
بيان العقدة	self.data = data
مؤشر العقدة الأخرى	self.next = next
تعريف فئة القائمة المترابطة و خصائصها	class LinkedList: def __init__(self):
رأس عقدة البداية	self.head = None
تخزين بيان العقدة داخل القائمة المترابطة و طباعتها	Linkedlist1 = LinkedList() Linkedlist1.head = Node("Monday") print(Linkedlist1.head.data)

الكود البرمجي بالكامل

```
# single node
class Node:
def __init__(self, data = None, next=None):
self.data = data
self.next = next
# linked list with one head node
class LinkedList:
def __init__(self):
self.head = None
# list linked with a single node
Linkedlist1 = LinkedList()
Linkedlist1.head = Node("Monday")
print(Linkedlist1.head.data)
```

١-١٣ : انشاء قائمة مترابطة تحتوي عدة عقد : ص٤٧

الأمر البرمجي	شرح الأمر البرمجي
class Node: def __init__(self, data = None, next=None):	تعريف فئة لعقد و خصائصها
self.data = data	بيان العقد
self.next = next	المؤشر للعقدة الأخرى
class LinkedList: def __init__(self):	تعريف فئة القائمة المترابطة و خصائصها
self.head = None	رأس العقدة الأولى

linked_list = LinkedList()
# the first node 	
linked_list.head = Node("Monday")
# the second node 	
linked_list.head.next = Node("Tuesday")
# the third node 	
linked_list.head.next.next = Node("Wednesday")
# print the linked list 	
node = linked_list.head
	تخزين بيانات العقد تسلسليا داخل القائمة المترابطة

while node:
  print (node.data)
  node = node.next
	التنقل من عقدة لعقدة أخرى و طباعة العقد

الكود البرمجي بالكامل

```
class Node:
def __init__(self, data = None, next=None):
self.data = data
self.next = next

# an empty linked list with a head node.
class LinkedList:
def __init__(self):
self.head = None
# the main program
linked_list = LinkedList()
# the first node
linked_list.head = Node("Monday")
# the second node
linked_list.head.next = Node("Tuesday")
# the third node
linked_list.head.next.next = Node("Wednesday")
# print the linked list
node = linked_list.head
while node:
print (node.data)
node = node.next
```

١-١٤ : إضافة عقدة إلى القائمة المترابطة : ص ٤٨

شرح الأمر البرمجي	الأمر البرمجي
تعريف فئة لعقد و خصائصها	class Node: def __init__(self, data = None, next=None):
بيان العقد	self.data = data
مؤشر العقدة الأخرى	self.next = next
تعريف فئة القائمة المترابطة و خصائصها	class LinkedList: def __init__(self):
رأس العقدة الأولى	self.head = None
دالة انشاء العقد	def insertAfter(new, prev): new_node = Node(new) new_node.next = prev.next prev.next = new_node

ادراج العقدتين للقائمة	L_list = LinkedList()
	# add the first two nodes
	L_list.head = Node(12)
	second = Node(99)
	L_list.head.next = second

إضافة العقدة الجديدة بين العقدتين و طباعة القائمة المترابطة	insertAfter(37, L_list.head)
	# print the linked list
	node = L_list.head
	while node:
	print (node.data)
	node = node.next

الكود البرمجي بالكامل

```
# single node
class Node:
def __init__(self, data = None, next=None):
self.data = data
self.next = next
# linked list with one head node
class LinkedList:
def _init_(self):
self.head = None
def insertAfter(new, prev):
# create the new node
new_node = Node(new)
# make the next of the new node the same as the next of the previous node
new_node.next = prev.next
# make the next of the previous node the new node
prev.next = new_node
# create the linked list
L_list = LinkedList()
# add the first two nodes
L_list.head = Node(12)
second = Node(99)
L_list.head.next = second
# insert the new node after node 12 (the head of the list)
insertAfter(37, L_list.head)
# print the linked list
node = L_list.head
while node:
print (node.data)
node = node.next
```

١-١٥ : حذف عقدة من القائمة المترابطة : ص ٤٩

شرح الأمر البرمجي	الأمر البرمجي

تعريف فئة لعقد و خصائصها	class Node:
	  def __init__(self, data = None, next=None):
	  self.data = data
	  self.next = next
	class LinkedList:
	  def __init__(self):
	  self.head = None

تعريف دالة حذف العقدة	def deleteNode(key, follow):
	  # store the head node
	  temp = follow.head

حلقة تكرارية لتحديد العقد و مؤشرات العقد	while(temp is not None):
	  if temp.data == key:
	  break
	  prev = temp
	  temp = temp.next
	  prev.next = temp.next
	  temp = None

ادراج العقد للقائمة وربط المؤشرات	L_list = LinkedList()
	# add the first three nodes
	L_list.head = Node(12)
	second = Node(37)
	third = Node(99)
	L_list.head.next = second
	second.next = third

حذف العقدة المطلوبة و طباعة بقية العقد	deleteNode(37,L_list)
	# print the linked list
	node = L_list.head
	while node:
	  print (node.data)
	  node = node.next

الكود البرمجي بالكامل

```
# single node
class Node:
def __init__(self, data = None, next=None):
self.data = data
self.next = next
# linked list with one head node
class LinkedList:
def __init__(self):
self.head = None
def deleteNode(key, follow):
# store the head node
temp = follow.head
# find the key to be deleted,
# the trace of the previous node to be changed
while(temp is not None):
if temp.data == key:
break
prev = temp
```

```
temp = temp.next
# unlink the node from the linked list
prev.next = temp.next
temp = None
# create the linked list

L_list = LinkedList()
# add the first three nodes
L_list.head = Node(12)
second = Node(37)
third = Node(99)
L_list.head.next = second
second.next = third
# delete node 37
deleteNode(37,L_list)
# print the linked list
node = L_list.head
while node:
print (node.data)
node = node.next
```

التدريبات العملية لمقرر الذكاء الاصطناعي ١-١

تدريبات الوحدة الأولى

أساسيات الذكاء الاصطناعي

عدد تدريبات الوحدة ١٩

تدريبات اليوم الثاني ٤

١-١٦ : إنشاء شجرة باستخدام قاموس البايثون : ص ٥٦

شرح الأمر البرمجي	الأمر البرمجي
إنشاء الشجرة	myTree = {
انشاء الجذر	"a": ["b", "c"], # node
انشاء العقد وفقا للمخطط	"b": ["d", "e"], "c": [None, "f"], "d": [None, None], "e": [None, None], "f": [None, None],
طباعة عناصر الشجرة	print(myTree)

الكود البرمجي بالكامل

```
myTree = {
"a": ["b", "c"], # node
"b": ["d", "e"],
"c": [None, "f"],
"d": [None, None],
"e": [None, None],
"f": [None, None],
}
print(myTree)
```

١-١٧ : إنشاء شجرة بطباعة عدد العقد المنبثقة و عناصرها : ص ٥٦

شرح الأمر البرمجي	الأمر البرمجي
إنشاء الشجرة بتحديد الأصل و الفرع لكل مستوى	myTree = {"Data Structures":["Linear","Non-linear"], "Linear":["Stack","Queue","Linked List"], "Non-linear":["Tree", "Graph"]}
حلقة تكرارية لأصول الشجرة مع طباعة الأصل و عدد عقد كل أصل	for parent in myTree: print(parent, "has",len(myTree[parent]),"nodes")
حلقة تكرارية لفروع الشجرة مع طباعة الفرع	for children in myTree[parent]: print(" ",children)

الكود البرمجي بالكامل

```
myTree = {"Data Structures":["Linear","Non-linear"],
"Linear":["Stack","Queue","Linked List"],
"Non-linear":["Tree", "Graph"]}
for parent in myTree:
print(parent, "has",len(myTree[parent]),"nodes" )
for children in myTree[parent]:
print(" ",children)
```

١-١٨ : إنشاء مخطط موجه : ص ٦١

الأمر البرمجي	شرح الأمر البرمجي
myGraph = { "a" : ["b","c"], "b" : ["c", "d"], "c" : ["d", "e"], "d" : [], "e" : [], }	إنشاء عقد المخطط الموجه
print(myGraph)	طباعة المخطط الموجه
الكود البرمجي بالكامل	
myGraph = { "a" : ["b","c"], "b" : ["c", "d"], "c" : ["d", "e"], "d" : [], "e" : [], } print(myGraph)	

١-١٩ : إنشاء مخطط غير موجه و إضافة حواف جديدة : ص ٦١-٦٢

الأمر البرمجي	شرح الأمر البرمجي

# function for adding an edge to a graph 	
def addEdge(graph,u,v):
graph[u].append(v)
def generate_edges(graph):
edges = []
	انشاء دالة اضافة الحواف للعقد في المخطط الغير موجه لاستدعائها لاحقا

for node in graph:
for neighbour in graph[node]:
# if edge exists then append to the list 	
edges.append((node, neighbour))
return edges
	حلقات تكرارية لاضافة الحواف للعقد و العقد المجاورة لها

myGraph = {"a" : ["b","c"], "b" : ["c", "d"], "c" : ["d", "e"], "d" : [], "e" : [], } print("The graph contents") print(generate_edges(myGraph))	انشاء المخطط الغير موجه و تحديد العقد و مساراتها و طباعة المخطط
addEdge(myGraph,'a','e') addEdge(myGraph,'c','f') print("The new graph after adding new edges") print(generate_edges(myGraph))	استدعاء دالة إضافة الحواف للمخطط وفق العقد المطلوبة وطباعة المخطط الغير متجه الجديد
الكود البرمجي بالكامل	

# function for adding an edge to a graph 	
def addEdge(graph,u,v):
graph[u].append(v)
# function for generating the edges of a graph 	
def generate_edges(graph):
edges = []	

```
# for each node in graph
for node in graph:
# for each neighbouring node of a single node
for neighbour in graph[node]:
# if edge exists then append to the list
    edges.append((node, neighbour))
return edges
# main program
# initialisation of graph as dictionary
myGraph = {"a" : ["b","c"],
"b" : ["c", "d"],
"c" : ["d", "e"],
"d" : [],
"e" : [],
}
# print the graph contents
print("The graph contents")
print(generate_edges(myGraph))
# add more edges to the graph
addEdge(myGraph,'a','e')
addEdge(myGraph,'c','f')
# print the graph after adding new edges
print("The new graph after adding new edges")
print(generate_edges(myGraph))
```

التدريبات العملية لمقرر الذكاء الاصطناعي ١-١

تدريبات الوحدة الثانية

خوارزميات الذكاء الاصطناعي

عدد تدريبات الوحدة ١٩

تدريبات اليوم الثاني ٤

٢-١ : حساب المتوسط : ص ٧٢

الأمر البرمجي	شرح الأمر البرمجي

def mySumGrade (gradesList):
  sumGrade=0
  l=len(gradesList)
  for i in range(l):
  sumGrade=sumGrade+gradesList[i]
  return sumGrade
	أنشاء دالة للجمع لاستدعائها لاحقا
	تحديد طول القائمة (عدد الاعداد)
	
	و حلقة تكرارية لجمع الرقم الجديد مع الأرقام السابقة في القائمة

def avgFunc (gradesList):
  s=mySumGrade(gradesList)
  l=len(gradesList)
  avg=s/l
  return avg
	انشاء دالة للمتوسط لاستدعائها لاحقا

grades=[89,88,98,95]	ادخال العناصر
averageGrade=avgFunc(grades) print ("The average grade is: ",averageGrade)	استدعاء دالة المتوسط ثم طباعة قيمة المتوسط

الكود البرمجي بالكامل

```
def mySumGrade (gradesList):
sumGrade=0
l=len(gradesList)
for i in range(l):
sumGrade=sumGrade+gradesList[i]
return sumGrade
def avgFunc (gradesList):
s=mySumGrade(gradesList)
l=len(gradesList)
avg=s/l
return avg
# program section
grades=[89,88,98,95]
averageGrade=avgFunc(grades)
print ("The average grade is: ",averageGrade)
```

٢-٢ : حساب مضروب رقم باستخدام حلقة for : ص ٧٣

الأمر البرمجي	شرح الأمر البرمجي
def factorialLoop(n):	أنشاء دالة المضروب لاستدعائها لاحقا
result = 1 for i in range(2,n+1): result = result * i return result	حالتي دالة الاستدعاء التكرارية
num = int(input("Type a number: ")) f=factorial(num)	ادخال العدد و استدعاء دالة المضروب
print("The factorial of ", num, " is: ", f)	طباعة نتيجة مضروب العدد

الكود البرمجي بالكامل

```
def factorialLoop(n):
    result = 1
    for i in range(2,n+1):
        result = result * i
    return result
# main program
num = int(input("Type a number: "))
f=factorialLoop(num)
print("The factorial of ", num, " is:",f)
```

٢-٣ : حساب مضروب رقم بدالة المضروب : ص ٧٤

الأمر البرمجي	شرح الأمر البرمجي
def factorial(x):	أنشاء دالة المضروب لاستدعائها لاحقا

if x == 0:
return 1
else:
return (x * factorial(x-1))
	حالتي دالة الاستدعاء التكرارية
	تطبيق معادلة المضروب

num = int(input("Type a number: ")) f=factorial(num)	ادخال العدد و استدعاء دالة المضروب
print("The factorial of ", num, " is: ", f)	طباعة نتيجة مضروب العدد

الكود البرمجي بالكامل

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return (x * factorial(x-1))
# main program
num = int(input("Type a number: "))
f=factorial(num)
print("The factorial of ", num, " is: ", f)
```

٤-٢ بالطريقة الأولى : استخراج أكبر عنصر في قائمة : ص ٧٥

شرح الأمر البرمجي	الأمر البرمجي
أنشاء دالة للعدد الأكبر ليتم استدعائها لاحقا	def findMaxRecursion(A,n):

حالتي دالة الاستدعاء التكرارية
 	
اذا كانت القائمة مكونة من عنصر واحد فهو الأكبر أو يتم مقارنة عناصر القائمة وفقا لفهرس العناصر
	if n==1:
	m = A[n-1]
	else:
	m = max(A[n-1],findMaxRecursion(A,n-1))
	return m

تخزين الاعداد في القائمة و تخزين طول القائمة	myList = [4,73,-5,42] l = len(myList)
استدعاء الدالة و طباعة أكبر عنصر فيها	myMaxRecursion = findMaxRecursion(myList,l) print("Max with recursion is: ", myMaxRecursion)

الكود البرمجي بالكامل

```
def findMaxRecursion(A,n):
    if n==1:
        m = A[n-1]
    else:
        m = max(A[n-1],findMaxRecursion(A,n-1))
    return m
myList = [4,73,-5,42]
l = len(myList)
myMaxRecursion = findMaxRecursion(myList,l)
print("Max with recursion is: ", myMaxRecursion)
```

٤-٢ بالطريقة الثانية : استخراج أكبر عنصر في قائمة :

شرح الأمر البرمجي	الأمر البرمجي
أنشاء دالة للعدد الأكبر ليتم استدعائها لاحقا	def findMaxIteration(A,n):
حالتي دالة الاستدعاء التكرارية حلقة تكرارية لادخال عناصر القائمة و مقارنة كل عنصرين , تحديد الأكبر بينهما	m = A[0] for i in range(1,n): m = max(m,A[i]) return m
تخزين الاعداد في القائمة و تخزين طول القائمة	myList = [4,73,-5,42] l = len(myList)
استدعاء الدالة و طباعة أكبر عنصر فيها	myMaxIteration = findMaxIteration(myList,l) print("Max with iteration is: ", myMaxIteration)

الكود البرمجي بالكامل

```
def findMaxIteration(A,n):
    m = A[0]
    for i in range(1,n):
        m = max(m,A[i])
    return m
# main program
myList = [4,73,-5,42]
l = len(myList)

myMaxIteration = findMaxIteration(myList,l)
print("Max with iteration is: ", myMaxIteration)
```

٢-٥ بالطريقة الأولى : حساب مضاعف رقم : ص ٧٦

الأمر البرمجي	شرح الأمر البرمجي
def powerFunRecursive(baseNum,expNum):	أنشاء دالة لحساب مضاعف العدد لاستدعائها لاحقا

if(expNum==1):
return(baseNum)
else:
return(baseNum*powerFunRecursive(baseNum,expNum-1))
	حالتي دالة الاستدعاء التكرارية
	أذا كانت قوة العدد = ١ سيطبع العدد كما هو و إلا سيحسب الدالة بضرب العدد مرات عدد القوة

base = int(input("Enter number: ")) exp = int(input("Enter exponent: "))	ادخال العدد و القوة المرفوع لها
numPowerRecursion = powerFunRecursive(base,exp) print("Recursion: ", base, " raised to ", exp, " = ",numPowerRecursion)	استدعاء الدالة و طباعة نتيجة مضاعف العدد

الكود البرمجي بالكامل

```
def powerFunRecursive(baseNum,expNum):
    if(expNum==1):
        return(baseNum)
    else:
        return(baseNum*powerFunRecursive(baseNum,expNum-1))
# main program
base = int(input("Enter number: "))
exp = int(input("Enter exponent: "))
numPowerRecursion = powerFunRecursive(base,exp)
print( "Recursion: ", base, " raised to ", exp, " = ",numPowerRecursion)
```

٢-٥ بالطريقة الثانية : حساب مضاعف رقم : ص ٧٦

الأمر البرمجي	شرح الأمر البرمجي
def powerFunIteration(baseNum,expNum):	أنشاء دالة لحساب مضاعف العدد لاستدعائها لاحقا
numPower = 1 for i in range(exp): numPower = numPower*base return numPower	حالتي دالة الاستدعاء التكرارية وضع القوة = ١ ثم حلقة تكرارية لضرب العدد في القوة
base = int(input("Enter number: ")) exp = int(input("Enter exponent: "))	ادخال العدد و القوة المرفوع لها
numPowerIteration = powerFunIteration(base,exp) print("Iteration: ", base, " raised to ", exp, " = ",numPowerIteration)	استدعاء الدالة و طباعة نتيجة مضاعف العدد

الكود البرمجي بالكامل

```
def powerFunIteration(baseNum,expNum):
    numPower = 1
    for i in range(exp):
        numPower = numPower*base
    return numPower
base = int(input("Enter number: "))
exp = int(input("Enter exponent: "))
numPowerIteration = powerFunIteration(base,exp)
print( "Iteration: ", base, " raised to ", exp, " = ",numPowerIteration)
```

٦-٢ : تطبيق خوارزمية الاتساع BFS على مخطط : ص ٨١-٨٢

الأمر البرمجي	شرح الأمر البرمجي

graph = {
"A" : ["B","C"],
"B" : ["D","E"],
"C" : ["F"],
"D" : [],
"E" : [],
"F" : []
}
	ادخال عقد المخطط

visitedBFS = [] def bfs(visited, graph, node): visited.append(node) queue.append(node)	أنشاء دالة خوارزمية الاتساع لاستدعائها لاحقا

while queue:
n = queue.pop(0)
print (n, end = " ")
for neighbor in graph[n]:
if neighbor not in visited:
visited.append(neighbor)
queue.append(neighbor)
	حلقة تكرارية لحذف العقدة من الطابور و طباعتها
	فحص العقدة التالية حسب المستوى و اضافتها للطابور

bfs(visitedBFS, graph, "A")	طباعة جميع عقد المخطط بدأ من الجذر A بخوارزمية BFS

الكود البرمجي بالكامل

```
graph = {
"A" : ["B","C"],
"B" : ["D","E"],
"C" : ["F"],
"D" : [],
"E" : [],
"F" : []
}
visitedBFS = [] # List to keep track of visited nodes
queue = [] # Initialize a queue
# bfs function
def bfs(visited, graph, node):
visited.append(node)
queue.append(node)
while queue:
n = queue.pop(0)
print (n, end = " ")
for neighbor in graph[n]:
if neighbor not in visited:
visited.append(neighbor)
queue.append(neighbor)
# main program
bfs(visitedBFS, graph, "A")
```

٧-٢ : تطبيق خوارزمية العمق DFS على مخطط : ص ٨٤

الأمر البرمجي	شرح الأمر البرمجي

graph = {
"A" : ["B","C"],
"B" : ["D","E"],
"C" : ["F"],
"D" : [],
"E" : [],
"F" : []
}
	ادخال عقد المخطط

visitedDFS = []
def dfs(visited, graph, node):
if node not in visited:
print(node, end=" ")
visited.append(node)
	أنشاء دالة خوارزمية العمق لاستدعائها لاحقا يحتوي تعريف الدالة على (العقد التي تم زيارتها و المخطط و العقدة التي سنبدأ بها)
	
	فحص العقد الى الحواف و اضافتها للمكدس (يستخدم المكدس بصورة غير مباشرة لتتبع الاستدعاءات التكرارية)

for neighbor in graph[node]: dfs(visited, graph, neighbor) dfs(visitedDFS, graph, "A")	اظهار نتيجة بحث عقد المخطط بخوارزمية DFS

الكود البرمجي بالكامل

```
graph = {
"A": ["B", "C"],
"B": ["D", "E"],
"C": ["F"],
"D": [],
"E": [],
"F": []
}
visitedDFS = []  # list to keep track of visited nodes
# dfs function
def dfs(visited, graph, node):
if node not in visited:
print(node, end=" ")
visited.append(node)
for neighbor in graph[node]:
dfs(visited, graph, neighbor)
dfs(visitedDFS, graph, "A")
```

التدريبات العملية لمقرر الذكاء الاصطناعي ١-١

تدريبات الوحدة الثانية

خوارزميات الذكاء الاصطناعي

٦ تدريبات

٢-٨ : التشخيص الطبي الاصدار الأول : ص ٩٠ الى٩٢

الكود البرمجي بالكامل	شرح الأمر البرمجي

import json # a library used to save and load JSON files
# the file with the symptom mapping 	
 	
symptom_mapping_file='symptom_mapping_v1.json'
 	
# open the mapping JSON file and load it into a dictionary 	
with open(symptom_mapping_file) as f:
  mapping=json.load(f)
 	
# print the JSON file 	
print(json.dumps(mapping, indent=2))
	استدعاء مكتبة JSON
	
	متغير لتخزين بيانات ملف الإصدار الأول
	'symptom_mapping_v1
	
	فتح الملف السابق في مكتبة JOSN على شكل قاموس و اسناده لمتغير
	
	طباعة البيانات بتنسيق و اختيار المسافة البادئة المرغوبة

def diagnose_v1(patient_symptoms:list):
  diagnosis=[] # the list of possible diseases
 	
  if "vomiting" in patient_symptoms:
  if "abdominal pain" in patient_symptoms:
  if "diarrhea" in patient_symptoms:
  # 1:vomiting, 2:abdominal pain, 3:diarrhea
  diagnosis.append('food poisoning')
  elif 'fever' in patient_symptoms:
  # 1:vomiting, 2:abdominal pain, 3:fever
  diagnosis.append('food poisoning')
  diagnosis.append('appendicitis')
  elif "lower back pain" in patient_symptoms and 'fever' in patient_symptoms:
  # 1:vomiting, 2:lower back pain, 3:fever
  diagnosis.append('kidney stones')
  elif "abdominal pain" in patient_symptoms and\
  "diarrhea" in patient_symptoms and\
  "fever" in patient_symptoms:\
  # 1:abdominal pain, 2:diarrhea, 3:fever
  diagnosis.append('food poisoning')
  return diagnosis
	تصنيف المرض وفقا للعرض المصاحب :
	
	تعريف دالة على هيئة قائمة يكون معاملها هو العرض المرضي و جعلها قائمة فارغة
	
	دالة الشرط if لربط الأعراض بالتصنيف الملائم للمرض الطبي
	
	استخدام قاعدة المعرفة التالية
	(اذا كان لدى المريض على الأقل ثلاثا من جميع الأعراض المحتملة للمرض يضاف المرض كتشخيص محتمل)

# Patient 1 	
my_symptoms=['abdominal pain', 'fever', 'vomiting']
diagnosis=diagnose_v1(my_symptoms)
print('Most likely diagnosis:',diagnosis)
# Patient 2 	
my_symptoms=['vomiting', 'lower back pain', 'fever']
diagnosis=diagnose_v1(my_symptoms)
print('Most likely diagnosis:',diagnosis)
# Patient 3 	
my_symptoms=['fever', 'cough', 'vomiting']
diagnosis=diagnose_v1(my_symptoms)
print('Most likely diagnosis:',diagnosis)
	اختبار النظام على ثلاث مرض
	
	ادخال اعراض ثلاث مرضى و تشخيص المرض وفقا للأعراض المدونة

٢-٨ : التشخيص الطبي الاصدار الثاني : ص ٩٣ الى٩٥

الكود البرمجي بالكامل	شرح الأمر البرمجي
symptom_mapping_file='symptom_mapping_v2.json' with open(symptom_mapping_file) as f: mapping=json.load(f) print(json.dumps(mapping, indent=2))	متغير لتخزين بيانات ملف الإصدار الثاني 'symptom_mapping_v2' فتح الملف السابق في مكتبة JOSN على شكل قاموس و اسناده لمتغير طباعة البيانات بتنسيق و اختيار المسافة البادئة المرغوبة

def diagnose_v2(patient_symptoms:list,
  symptom_mapping_file:str,
  matching_symptoms_lower_bound:int):
  diagnosis=[]
  with open(symptom_mapping_file) as f:
  mapping=json.load(f)
  disease_info=mapping['diseases']
 	
  for disease in disease_info:
  counter=0
  disease_symptoms=disease_info[disease]
  for symptom in patient_symptoms:
  if symptom in disease_symptoms:
  counter+=1
  if counter>=matching_symptoms_lower_bound:
  diagnosis.append(disease)
  return diagnosis
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v2(my_symptoms,'symptom_mapping_v2.json', 3)
print('Most likely diagnosis:',diagnosis)
 	
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v2(my_symptoms, 'symptom_mapping_v2.json', 4)
print('Most likely diagnosis:',diagnosis)
 	
my_symptoms=['fever', 'cough', 'vomiting']
diagnosis=diagnose_v2(my_symptoms, 'symptom_mapping_v2.json', 3)
print('Most likely diagnosis:',diagnosis)
	تصنيف المرض وفقا للعرض المصاحب :
	
	تعريف دالة على هيئة قائمة بثلاث معاملات (قائمة الأعراض – ملف تعيين الأمراض – الحد الأدنى من الاعراض المتطابقة) و جعلها قائمة فارغة
	
	فتح ملف تعيين الأمراض كقاموس
	
	استخدام حلقة FOR بفحص كل مرض بشكل فردي
	
	اذا كانت قيمة العداد تفوق الحد الأدنى من الأعراض يتم إضافة المرض
	
	استخدام قاعدة المعرفة التالية
	(حساب عدد الأعراض المطابقة لكل مرض و السماح للمستخدم بتحديد عدد الأعراض المطابقة التي يجب توافرها في المرض لتضمينه في التشخيص)

# Patient 1 	
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v2(my_symptoms,'symptom_mapping_v2.json', 3)
print('Most likely diagnosis:',diagnosis)
# Patient 2 	
	ادخال اعراض ثلاث مرضى و تشخيص المرض وفقا بمقارنة كل عرض على حدة مع الاعراض المعروفة للمرض و زيادة العداد في كل مرة يجد تطابق (وفقا لعدد الاعراض)

```
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore
throat"]
diagnosis=diagnose_v2(my_symptoms, 'symptom_mapping_v2.json'
, 4)
print('Most likely diagnosis:',diagnosis)
# Patient 3
my_symptoms=['fever', 'cough', 'vomiting']
diagnosis=diagnose_v2(my_symptoms, 'symptom_mapping_v2.json'
, 3)
print('Most likely diagnosis:',diagnosis)
```

٢-٨ : التشخيص الطبي الاصدار الثالث: ص ٩٦ الى٩٨

الكود البرمجي بالكامل	شرح الأمر البرمجي
`symptom_mapping_file='symptom_mapping_v3.json'`	متغير لتخزين بيانات ملف الإصدار الثالث 'symptom_mapping_ v3
`# open the mapping JSON file and load it into a dictionary`	
`with open(symptom_mapping_file) as f:` `  mapping=json.load(f)`	فتح الملف السابق في مكتبة JOSN على شكل قاموس و اسناده لمتغير
`# print the JSON file` `print(json.dumps(mapping, indent=2))`	طباعة البيانات بتنسيق و اختيار المسافة البادئة المرغوبة

`from collections import defaultdict`
`def diagnose_v3(patient_symptoms:list,`
`  symptom_mapping_file:str,`
`  very_common_weight:float=1,`
`  less_common_weight:float=0.5`
`  ):`
	تعريف دالة تحتوي متغيرات (أعراض المريض – جميع الأعراض للأمراض المختلفة – وزن للأعراض الأكثر شيوعا – ووزن للأعراض الأقل شيوعا)

`  with open(symptom_mapping_file) as f:` `  mapping=json.load(f)`	فتح ملف المرض كقاموس
`  disease_info=mapping['diseases']`	متغير يحتوي معلومات الأمراض

`  # holds a symptom-based score for each potential disease`
`  disease_scores=defaultdict(int)`
`  for disease in disease_info:`	

`  # get the very common symptoms of the disease` `  very_common_symptoms=disease_info[disease]['very common']`	مقارنة أعراض المريض بأعراض كل مرض (هل هي اعراض شائعة أم أقل شيوعا)

`  # get the less common symptoms for this disease`
`  less_common_symptoms=disease_info[disease]['less common']`
`  for symptom in patient_symptoms:`
`  if symptom in very_common_symptoms:`
`  disease_scores[disease]+=very_common_weight`
`  elif symptom in less_common_symptoms:`
`  disease_scores[disease]+=less_common_weight`
	استخدام قاعدة المعرفة التالية
	(بعض الأعراض أكثر شيوعا من أخرى للمرض نفسه , فيتم إعطاء أوزان مخصصة للأعراض الأكثر و الأقل شيوعا و المستخدم يحدد الأوزان التي يراها مناسبة فيشخص المرض أو الأمراض ذات المجموع الموزون الأعلى في التشخيص)

`  # find the max score all candidate diseases`
`  max_score=max(disease_scores.values())`
`  if max_score==0:`
`  return []`
`  else:`	

`  # get all diseases that have the max score`
`  diagnosis=[disease for disease in disease_scores if disease_scores`
`  [disease]==max_score]`
`  return diagnosis, max_score`
	مقارنة اعلى درجة اشتباه
	اذا كانت مساويه للصفر تعود الدالة بقيمة فارغة أو يطبع جميع الأمراض التي لها الدرجة القصوى

ادخال اعراض ثلاث مرضى و تشخيص المرض وفقا لوزن الأعراض الأكثر شيوعا للمرض	# Patient 1
	my_symptoms=["headache", "tiredness", "cough"]
	diagnosis=diagnose_v3(my_symptoms,
	'symptom_mapping_v3.json')
	print('Most likely diagnosis:',diagnosis)
	# Patient 2
	my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
	diagnosis=diagnose_v3(my_symptoms,
	'symptom_mapping_v3.json')
	print('Most likely diagnosis:',diagnosis)
	# Patient 3
	my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
	diagnosis=diagnose_v3(my_symptoms,
	'symptom_mapping_v3.json', 1, 1)
	print('Most likely diagnosis:',diagnosis)

٢-٨ : التشخيص الطبي الاصدار الرابع : ص ٩٩ الى١٠٤

شرح الأمر البرمجي	الكود البرمجي بالكامل

استدعاء مكتبة pandas لتحليل البيانات و استيراد ملفات من نوع css و تخزينها في متغير
 	
(البيانات تشمل ملف ٢٠٠٠ مريض و العمود الأخير يمثل تشخيص الخبير البشري)
	import pandas as pd # import pandas to load and process spreadsheet-type data
	medical_dataset=pd.read_csv('medical_data.csv') # load a medical dataset.
	medical_dataset

امر برمجي اخياري اذا رغبت في التأكد من الأمراض التي شخصها الخبير البشري	set(medical_dataset['diagnosis'])

استخدام خوارزميات معينة عبر مكتبة skleearn الخاصة بتعليم الآلة و نمذجتها
 	
تهيئة الآلة لتدريب على النموذج
 	
استخدام نموذج شجرة القرار (مصنف شجرة القرار) من مكتبة البايثون سكليرن skearn
 	
مرحلة التدريب
اسقاط عمود التشخيص للحصول على الاعراض فقط
 	
الحصول على عمود التشخيص لاستخدامه كهدف التصنيف
 	
بناء شجرة القرار
	from sklearn.tree import DecisionTreeClassifier
	
	def diagnose_v4(train_dataset:pd.DataFrame):
	
	# create a DecisionTreeClassifier
	model=DecisionTreeClassifier(random_state=1)
	
	# drop the diagnosis column to get only the symptoms
	
	train_patient_symptoms=train_dataset.drop(columns=['diagnosis'])
	
	# get the diagnosis column, to be used as the classification target
	train_diagnoses=train_dataset['diagnosis']
	
	# build a decision tree
	model.fit(train_patient_symptoms, train_diagnoses)
	# return the trained model
	return model

عملية التدريب لا تكون على جميع البيانات سيتم تقسيمها لبيانات تدريب ٧٠ % و اختبار ٣٠ % بطريقة عشوائية (١)	from sklearn.model_selection import train_test_split # use the function to split the data, get 30% for testing and 70% for training. train_data, test_data = train_test_split(medical_dataset, test_size=0.3, random_state=1)

#print the shapes (rows x columns) of the two datasets
print(train_data.shape)
print(test_data.shape)
	طباعة النتائج حيث ان العمود الأول عدد المرضى و الثاني يمثل عدد أعمدة البيانات)

from sklearn.tree import plot_tree
 	
import matplotlib.pyplot as plt
 	
my_tree=diagnose_v4(train_data) # train a model
print(my_tree.classes_) # print the possible target labels (diagnoses)
 	
plt.figure(figsize=(12,6)) # size of the visualization, in inches
# plot the tree 	
plot_tree(my_tree,
max_depth=2,
fontsize=10,
feature_names=medical_dataset.columns[:-1]
)
	التمثيل البياني لشجرة القرار

# functions used to evaluate a classifier 	
from sklearn.metrics import accuracy_score,confusion_matrix
# drop the diagnosis column to get only the symptoms 	
test_patient_symptoms=test_data.drop(columns=['diagnosis'])
# get the diagnosis column, to be used as the classification target 	
test_diagnoses=test_data['diagnosis']
# guess the most likely diagnoses 	
pred=my_tree.predict(test_patient_symptoms)
# print the achieved accuracy score 	
accuracy_score(test_diagnoses,pred)
	نسبة دقة النموذج بعد عمليات الاختبار
	
	تحققت دقة النموذج بنسبة ٨١,٦ و هذا يعني انه من بين ٦٠٠ حالة اختبار شخصت شجرة القرار ٤٨٦ حالة بشكل صحيح

confusion_matrix(test_diagnoses,pred)	طباعة مصفوفة الدقة للنموذج لاستعراض الأمثلة التي صنفت البيانات على الخط القطري للمصفوفة تمثل التشخيص الصحيح المتوقع و التي على خارج الخط القطري للمصفوفة تمثل أخطاء النموذج

٢-٩ : أنشاء متاهة ٣*٣ : ص ١٠٩

الأمر البرمجي	شرح الأمر البرمجي
import numpy as np	استدعاء مكتبة البيانات الرقمية
small_maze=np.zeros((3,3))	انشاء متاهة ٣*٣
blocks=[(1, 1), (2, 1), (2, 2)]	تحديد احداثيات العوائق
for block in blocks: small_maze[block]=1	تقدير رقم العائق ب ١ (و يمكن تقدير أي رقم اخر)
small_maze	عرض احداثيات المتاهة حيث ان القيمة صفر تعني فارغ و القيمة ١ تعني ان هناك عائق

الكود البرمجي بالكامل

```
import numpy as np
# create a numeric 3 x 3 matrix full of zeros.
small_maze=np.zeros((3,3))
# coordinates of the cells occupied by blocks
blocks=[(1, 1), (2, 1), (2, 2)]
for block in blocks:
# set the value of block-occupied cells to be equal to 1
small_maze[block]=1
small_maze
```

٢-١٠ : أنشاء متاهة كبيرة و معقدة : ص ١١٠

الأمر البرمجي	شرح الأمر البرمجي

import random
random_maze=np.zeros((10,10))
 	
# coordinates of 30 random cells occupied by blocks 	
blocks=[(random.randint(0,9),random.randint(0,9)) for i in range(30)]
for block in blocks:
  random_maze[block]=1
	انشاء متاهة بحجم و عوائق
	
	انشاء متاهة بأي عدد من الصفوف و الأعمدة و تخزينها في متغير
	
	توليد أي عدد من الخلايا عشوائية التي تشكل العوائق

import matplotlib.pyplot as plt # library used for visualization
def plot_maze(maze):
  ax = plt.gca() # create a new figure
  ax.invert_yaxis() # invert the y-axis to match the matrix
  ax.axis('off') # hide the axis labels
  ax.set_aspect('equal') # make sure the cells are rectangular
  plt.pcolormesh(maze, edgecolors='black', linewidth=2,cmap='Accent')
  plt.show()
plot_maze(random_maze)
	تمثيل المتاهة بالوان و تنسيقات اختيارية

الكود البرمجي بالكامل

```
import random
random_maze=np.zeros((10,10))
# coordinates of 30 random cells occupied by blocks
blocks=[(random.randint(0,9),random.randint(0,9)) for i in range(30)]
for block in blocks:
```

```
random_maze[block]=1
random_maze[block]=1
import matplotlib.pyplot as plt # library used for visualization
def plot_maze(maze):
ax = plt.gca() # create a new figure
ax.invert_yaxis() # invert the y-axis to match the matrix
ax.axis('off') # hide the axis labels
ax.set_aspect('equal') # make sure the cells are rectangular
plt.pcolormesh(maze, edgecolors='black', linewidth=2,cmap='Accent')
plt.show()
plot_maze(random_maze)
```

٢-١١ : تحديد الخلايا الفارغة المجاورة لخلية محددة : ص ١١١

شرح الأمر البرمجي	الأمر البرمجي

انشاء دالة لاستدعاء قائمة تحتوي علي كل الخلايا الفارغة و المجاورة لخلية محددة في أي متاهة	def get_accessible_neighbors(maze:np.ndarray, cell:tuple):
	# list of accessible neighbors, initialized to empty
	neighbors=[]
	x,y=cell
	# for each adjacent cell position
	for i,j in [(x-1,y-1),(x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y),(x+1,y+1)]:
	# if the adjacent cell is within the bounds of the grid and is not occupied by a block
	if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and maze[(i,j)]==0:
	neighbors.append(((i,j),1))
	return neighbors

اختبار اظهار الخلايا المجاورة الفارغة لخلية ما	get_accessible_neighbors(small_maze, (0,0))

الكود البرمجي بالكامل

```
def get_accessible_neighbors(maze:np.ndarray, cell:tuple):
# list of accessible neighbors, initialized to empty
neighbors=[]
x,y=cell
# for each adjacent cell position
for i,j in [(x-1,y-1),(x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y), (x+1,y+1)]:


# if the adjacent cell is within the bounds of the grid and is not occupied by a block
if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and maze[(i,j)]==0:

neighbors.append(((i,j),1))
return neighbors

get_accessible_neighbors(small_maze, (0,0))
```

٢-١٢ : استخدام خوارزمية البحث بالاتساع في حل المتاهة : ص ١١٢ الى ١١٤ص

شرح الأمر البرمجي	الكود البرمجي بالكامل

استخدام دالة **reconstruct_shortest_path** لاعادة و بناء المسار الأقصر و استدعائه لتتبع المسار بصورة عكسية من خلية الهدف إلى خلية البداية

```
def reconstruct_shortest_path(parent:dict, start_cell:tuple, target_cell:tuple):
    shortest_path = []
    my_parent=target_cell # start with the target_cell
# keep going from parent to parent until the search cell has been reached
    while my_parent!=start_cell:
        shortest_path.append(my_parent) # append the parent
        my_parent=parent[my_parent] # get the parent of the current parent
    shortest_path.append(start_cell) # append the start cell to complete the path
    shortest_path.reverse() # reverse the shortest path
    return shortest_path
```

تنفيذ دالة **bfs_maze_solver** بمساعدة الدالتين **reconstruct_shortest_path** و**get_accessible_neighbors( )**

```
from typing import Callable # used to call a function as an argument of another function
def bfs_maze_solver(start_cell:tuple,
                    target_cell:tuple,
                    maze:np.ndarray,
                    get_neighbors: Callable,
                    verbose:bool=False): # by default, suppresses descriptive output text
    cell_visits=0 # keeps track of the number of cells that were visited during the search
    visited = set() # keeps track of the cells that have already been visited
    to_expand = [] # keeps track of the cells that have to be expanded
# add the start cell to the two lists
    visited.add(start_cell)
    to_expand.append(start_cell)
# remembers the shortest distance from the start cell to each other cell
    shortest_distance = {}
# the shortest distance from the start cell to itself, zero
    shortest_distance[start_cell] = 0
# remembers the direct parent of each cell on the shortest path from the start_cell to the cell
    parent = {}
#the parent of the start cell is itself
    parent[start_cell] = start_cell
    while len(to_expand)>0:
        next_cell = to_expand.pop(0) # get the next cell and remove it from the expansion list
        if verbose:
            print('\nExpanding cell', next_cell)
# for each neighbor of this cell
        for neighbor,cost in get_neighbors(maze, next_cell):
            if verbose:
                print('\tVisiting neighbor cell',neighbor)
            cell_visits+=1
            if neighbor not in visited: # if this is the first time this neighbor is visited
                visited.add(neighbor)
                to_expand.append(neighbor)
                parent[neighbor]= next_cell
                shortest_distance[neighbor]=shortest_distance[next_cell]+cost
# target reached
            if neighbor==target_cell:
```

```
# get the shortest path to the target cell, reconstructed in reverse.
            shortest_path = reconstruct_shortest_path(parent, start_cell, target_cell)

            return shortest_path, shortest_distance[target_cell],cell_visits
      else: # this neighbor has been visited before
            # if the current shortest distance to the neighbor is longer than the shortest
            # distance to next_cell plus the cost of transitioning from next_cell to this neighbor
           if shortest_distance[neighbor]>shortest_distance[next_cell]+cost:

             parent[neighbor]=next_cell
             shortest_distance[neighbor]=shortest_distance[next_cell]+cost
# search complete but the target was never reached, no path exists
   return None,None,None
```

استخدام دالة bfs_maze_solver لحل متاهة ٣*٣

```
start_cell=(2,0) # start cell, marked by a star in the 3x3 maze
target_cell=(1,2) # target cell, marked by an "X" in the 3x3 maze
solution, distance, cell_visits=bfs_maze_solver(start_cell,
                                target_cell,
                                small_maze,
                                get_accessible_neighbors,
                                verbose=True)
print('\nShortest Path:', solution)
print('Cells on the Shortest Path:', len(solution))
print('Shortest Path Distance:', distance)
print('Number of cell visits:', cell_visits)
```

٢-١٣ : استخدام خوارزمية البحث بأولوية الأفضل : ص ١١٧ الى ص ١٢٠

شرح الأمر البرمجي	الكود البرمجي بالكامل

تحديد أفضل خلية مرشحة من بين مجموعة من الخلايا المرشحة استنادًا إلى مسافة معينة وتقدير (heuristic) ثابت

تعريف دالتين

constant_heuristi تأخذ مدخلين على هيئة أزواج وترجع قيمة ثابته تساوي ١(كل قيمة تقدير لاي خلية مرشحة هي نفسها بعض النظر عن الخلية المرشحة أو الخلية الهدف

get_best_candidate تـأخـذ ثـلاث مـعـامـلات

- expansion_candidates: مجموعة تحتوي على الخلايا المرشحة.
- shortest_distance: قاموس يربط الخلايا بأقصر المسافات إليها.
- heuristic: دالة تُستخدم لحساب التقدير (heuristic) بين خليتين، ولكنها لا تستخدم في هذا الكود. بدلاً من ذلك، يُستخدم دائمًا قيمة تقدير ثابتة تساوي ١

و نستخدم حلقة for لفحص الخلايا المرشحة في المجموعة و تحديد الأفضل باستخدام طابور الأولوية

```
def constant_heuristic(candidate_cell:tuple, target_cell:tuple):
    return 1
#
def get_best_candidate(expansion_candidates:set,
                       shortest_distance:dict,
                       heuristic:Callable): # قيمة ناتجة عن تقدير المسافة بين الخلية الحالية والمرشحة

    winner = None # متغير اخزن بداخله القيم الفائزة

    best_estimate= sys.maxsize
    for candidate in expansion_candidates:
        # تطبيق الدالة الاستدلالية واحتسابها في الخلايا المرشحة
        candidate_estimate=shortest_distance[candidate]+heuristic(candidate,target_cell)

        if candidate_estimate < best_estimate:
            winner = candidate
            best_estimate=candidate_estimate

    return winner
```

(للبحث بين خلية البداية وخلية الهدف في لوحة المتاه A* الكود السابق يقوم بحل مشكلة المتاهة باستخدام خوارزمية

```
import sys

def astar_maze_solver(start_cell:tuple,
                      target_cell:tuple,
                      maze:np.ndarray,
                      get_neighbors: Callable,
                      heuristic:Callable,
                      verbose:bool=False):

    cell_visits=0

    shortest_distance = {}
    shortest_distance[start_cell] = 0
```

```
parent = {}
parent[start_cell] = start_cell

expansion_candidates = set([start_cell])

fully_expanded = set()

# while there are still cells to be expanded
while len(expansion_candidates) > 0:

   best_cell = get_best_candidate(expansion_candidates,shortest_distance,heuristic)
   if best_cell == None: break

   if verbose: print('\nExpanding cell', best_cell)

   # if the target cell has been reached, reconstruct the shortest path and exit
   if best_cell == target_cell:

      shortest_path=reconstruct_shortest_path(parent,start_cell,target_cell)

      return shortest_path, shortest_distance[target_cell],cell_visits

   for neighbor,cost in get_neighbors(maze, best_cell):

      if verbose: print('\nVisiting neighbor cell', neighbor)

      cell_visits+=1

      # first time this neighbor is reached
      if neighbor not in expansion_candidates and neighbor not in fully_expanded:

         expansion_candidates.add(neighbor)

         parent[neighbor] = best_cell # mark the best_cell as this neighbor's parent
         # update the shortest distance for this neighbor
         shortest_distance[neighbor] = shortest_distance[best_cell] + cost
          # this neighbor has been visited before, but a better (shorter) path to it has just been found
      elif shortest_distance[neighbor] > shortest_distance[best_cell] + cost:

         shortest_distance[neighbor] = shortest_distance[best_cell] + cost

         parent[neighbor] = best_cell

         if neighbor in fully_expanded:
            fully_expanded.remove(neighbor)

            expansion_candidates.add(neighbor)

   # all neighbors of best_cell have been inspected at this point
   expansion_candidates.remove(best_cell)

   fully_expanded.add(best_cell)

return None, None, None # no solution was found
```

استخدام خوارزمية A* لحل مشكلة المتاهة بين الخلية البداية (2,0)والخلية المستهدفة (1,2)في لوحة المتاهة small_maze.بعد تنفيذ الكود

سيتم طباعة المخرجات التالية:

Shortest Path:المسار الأقصر بين الخلية البداية والخلية المستهدفة. إذا تم العثور على حلاً، ستظهر هنا قائمة الخلايا التي تشكل المسار الأقصر.
Cells on the Shortest Path:عدد الخلايا على المسار الأقصر بين الخلية البداية والخلية المستهدفة.
Shortest Path Distance:المسافة على المسار الأقصر بين الخلية البداية والخلية المستهدفة.
Number of cell visits:عدد الزيارات التي تمت للخلايا خلال عملية البحث.

```
start_cell=(2,0) # start cell, marked by a star in the 3x3 maze
target_cell=(1,2) # target cell, marked by an "X" in the 3x3 maze
solution, distance, cell_visits=bfs_maze_solver(start_cell,
                                target_cell,
                                small_maze,
                                get_accessible_neighbors,
                                verbose=True)
print('\nShortest Path:', solution)
print('Cells on the Shortest Path:', len(solution))
print('Shortest Path Distance:', distance)
print('Number of cell visits:', cell_visits)
```

التدريبات العملية لمقرر الذكاء الاصطناعي ١-١

تدريبات الوحدة الثالثة

معالجة اللغات الطبيعية

الدرس الأول

التعلم الموجه

٤ تدريبات (متصلة)

٣-١ : التنبؤ بالانطباع العام عن الفيلم : ص ١٣٥ الى ص ١٤٢

شرح الأمر البرمجي	الكود البرمجي بالكامل
استيراد مكتبة بانداس الخاصة بالتعامل مع جداول البيانات	
	# install the pandas library, if it is missing. !pip install pandas import pandas as pd
تحميل مجموعة بيانات التدريب و الاختبار	
	imdb_train_reviews=pd.read_csv('imdb_train.csv') imdb_test_reviews=pd.read_csv('imdb_test.csv') imdb_train_reviews
اسناد أعمدة النص و القيم إلى متغيرات مستقلة في أمثلة التدريب و الاختبار (استخراج النص من عمود النص للتدريب و الاختبار و استخراج التسميات من عمود التسمية لكل من عمود التدريب و الاختبار هل هو تعليق سلبي أم ايجابي) حيث أن البيانات المدخلة للتنبؤ x و القيم المستهدفة y	

	# extract the text from the 'text' column for both training and testing.
	X_train_text=imdb_train_reviews['text']
	X_test_text=imdb_test_reviews['text']
	# extract the labels from the 'label' column for both training and testing.
	Y_train=imdb_train_reviews['label']
	Y_test=imdb_test_reviews['label']
	X_train_text # training data in text forma

تجهيز البيانات (بيانات التدريب) استخدام أداة CountVectorizer المتوفرة في مكتبة سكليرن لتحويل البيانات إلى متجهات تحويل البيانات النصية إلى أرقام حقيقة لفهمها من قبل خوارزمية التعلم	

	from sklearn.feature_extraction.text import CountVectorizer
	# the min_df parameter is used to ignore terms that appear in less than 10 reviews.
	vectorizer_v1 = CountVectorizer(min_df=10)
	vectorizer_v1.fit(X_train_text) # fit the vectorizer on the training data.
	# use the fitted vectorizer to vectorize the data.
	X_train_v1 = vectorizer_v1.transform(X_train_text)
	X_train_v1

تحويل القيم السابقة الى مصفوفة استخدام التنسيق الكثيف Dense لوضع كل كلمة في عمود و يوضح عدد مرات ظهورها	
	X_train_v1_dense=pd.DataFrame(X_train_v1.toarray(), columns=vectorizer_v1.get_feature_names_out()) X_train_v1_dense
استخدام الدالة getsizeof ( ) التي تحدد حجم الكائنات في البايثون بالبايت لتوفير مساحة الذاكرة و ذلك بالاحتفاظ بالمدخلات الغير صفرية في كل عمود (المصفوفة المتباعدة)	
	from sys import getsizeof print('\nMegaBytes of RAM memory used by the raw text format:', getsizeof(X_train_text)/1000000) print('\nMegaBytes of RAM memory used by the dense matrix format:', getsizeof(X_train_v1_dense)/1000000) print('\nMegaBytes of RAM memory used by the sparse format:', getsizeof(X_train_v1)/1000000)
حذف المصفوفة الكثيفة لتوفير حجم الذاكرة	
	# delete the dense matrix. del X_train_v1_dense

شرح الأمر البرمجي	الكود البرمجي بالكامل

مرحلة التنبؤ

بناء خط أنابيب التنبؤ الأول و استخدام مصنف بايز الساذج المدرب مسبقا على التعلم بالمتجهات احتمالات الكلمات أو العبارات المحددة الواردة في النص للتنبؤ باحتمال انتمائه إلى تصنيف محدد

```
from sklearn.naive_bayes import MultinomialNB
model_v1=MultinomialNB() # a Naive Bayes Classifier
model_v1.fit(X_train_v1, Y_train) # fit the classifier on the vectorized training data.
from sklearn.pipeline import make_pipeline
# create a prediction pipeline: first vectorize using vectorizer_v1, then use model_v1 to predict.
prediction_pipeline_v1 = make_pipeline(vectorizer_v1, model_v1)
```

اختبار مقطع برمجي

```
prediction_pipeline_v1.predict(['One of the best movies of the year. Excellent cast and very interesting plot.',
                                'I was very disappointed with his film. I lost all interest after 30 minutes' ])
```

تنبؤ خط الأنابيب بالقيمة الإيجابية و السلبية للتقييمين بشكل صحيح

```
prediction_pipeline_v1.predict_proba(['One of the best movies of the year. Excellent cast and very interesting plot.',
                                'I was very disappointed with his film. I lost all interest after 30 minutes' ])
```

اختبار دقة خط الأنابيب الجديد في تصنيف التقييمات في مجموعة بيانات اختبار IMDb

```
predictions_v1 = prediction_pipeline_v1.predict(X_test_text) # vectorize the text data, then predict.
predictions_v1
```

تحليل و تصوير نتائج خطوط أنابيب التصنيف و تمثيل مصفوفة الدقة (تقريب دقيق لدرجة التنبؤ)

```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test, predictions_v1) # get the achieved accuracy
```

استدعاء مكتبة ( ) scikit-Piot

```
!pip install scikit-plot;
```

تمثيل مصفوفة الدقة

```
import scikitplot; # import the library
class_names=['neg','pos'] # pick intuitive names for the 0 and 1 labels.
# plot the confusion matrix.
scikitplot.metrics.plot_confusion_matrix(
 [class_names[i] for i in Y_test],
 [class_names[i] for i in predictions_v1],
 title="Confusion Matrix", # title to use
 cmap="Purples", # color palette to use
 figsize=(5,5) # figure size
 );
```

٣-٢ : النموذج المحايد المحلي القابل للتفسير و الشرح : ص ١٤٢ الى ص ١٤٥

شرح الأمر البرمجي	الكود البرمجي بالكامل

استيراد مكتبة النموذج المحايد lime الخاصة بتفسير التنبؤات التي قامت بها نماذج الصندوق الأسود

```
!pip install lime
```

تطبيق النموذج على عبارة سلبية و التأكد من صحة و دقة التنبؤ

```
from lime.lime_text import LimeTextExplainer
# create a local explainer for explaining individual predictions
explainer_v1 = LimeTextExplainer(class_names=class_names)
# an example of an obviously negative review
easy_example='This movie was horrible. The actors were terrible and the plot was very boring.'
# use the prediction pipeline to get the prediction probabilities for this example
print(prediction_pipeline_v1.predict_proba([easy_example]))
```

اظهار درجة لكل كلمة تمثل معامل في نموذج الانحدار الخطي البسيط لتقديم التفسير (يمكن زيادة او انقاص الخصائص المعروضة)

```
# explain the prediction for this example.
exp = explainer_v1.explain_instance(easy_example.lower(),prediction_pipeline_v1.predict_proba,
num_features=10)
# print the words with the strongest influence on the prediction.
exp.as_list()
```

التصور المرئي للدرجات السابقة الخاصة بالكلمات

```
# visualize the impact of the most influential words.
fig = exp.as_pyplot_figure()
```

استخدام النموذج المحايد لتحليل عبارة سلبية

```
exp.show_in_notebook()
```

استخدام النموذج المحايد لعبارة معقدة مأخوذة من مجموعة اختبار IMDB

```
# an example of a positive review that is mis-classified as negative by prediction_pipeline_v1
mistake_example= X_test_text[4600]
mistake_example
```

تقييم النموذج المحايد للعبارة السابقة

```
# get the correct labels of this example.
print('Correct Label:', class_names[Y_test[4600]])
# get the prediction probabilities for this example
print('Prediction Probabilities for neg, pos:',prediction_pipeline_v1.predict_proba([mistake_example]))
```

استخدام المفسر لتوضيح السبب وراء اتخاذ نموذج التنبؤ مثل هذا القرار الخاطئ

```
# explain the prediction for this example.
exp = explainer_v1.explain_instance(mistake_example, prediction_pipeline_v1.predict_proba,
num_features=10)

# visualize the explanation.
fig = exp.as_pyplot_figure()
```

٣-٣ : تحسين البرمجة الاتجاهية للنصوص: ص ١٤٦ الى ص ١٤٩

شرح الأمر البرمجي	الكود البرمجي بالكامل
استيراد مكتبة nltk الخاصة مهام معالجة اللغات الطبيعية المتنوعة	
!pip install nltk	
استيراد مكتبة Gensim الخاصة مهام معالجة اللغات الطبيعية المتنوعة	
!pip install gensim	
استدعاء مكتبةre الخاصة بالبحث في النصوص و معالجتها باستخدام التعبيرات النمطية	
import nltk # import nltk nltk.download('punkt') # install nltk's tokenization tool, used to split a text into sentences. import re # import re from gensim.models.phrases import Phrases, ENGLISH_CONNECTOR_WORDS # import tools from the gensim library	
استخدام دالة sent_tokenize () التابعة لمكتبة Nitk لتقسيم المستند على قائمة من الجمل المقسمة	
# convert a given doc to a list of tokenized sentences. def tokenize_doc(doc:str): return [re.findall(r'\b\w+\b', sent.lower()) for sent in nltk.sent_tokenize(doc)]	
مثال لتقسيم جملة لكلمات	
raw_text='The movie was too long. I fell asleep after the first 2 hours.' tokenized_sentences=tokenize_doc(raw_text) tokenized_sentences	
استقبال الدالة phrases () أربع متغيرات	

sentences=[] # list of all the tokenized sentences across all the docs in this dataset
for doc in X_train_text: # for each doc in this dataset
  sentences+=tokenize_doc(doc) # get the list of tokenized sentences in this doc
# build a phrase model on the given data 	
imdb_phrase_model = Phrases(sentences,
  connector_words=ENGLISH_CONNECTOR_WORDS,
  scoring='npmi',
  threshold=0.25).freeze()	

تطبيق الدالة phrases () على العبارة	
imdb_phrase_model[tokenized_sentences[0]]	
تطبيق الدالة phrases ()على العبارة	
imdb_phrase_model[tokenized_sentences[1]]	
تفسير العبارات في وثيقة معطاه	

def annotate_phrases(doc:str, phrase_model):
  sentences=tokenize_doc(doc)# split the document into tokenized sentences.
  tokens=[] # list of all the words and phrases found in the doc
  for sentence in sentences: # for each sentence
# use the phrase model to get tokens and append them to the list. 	
  tokens+=phrase_model[sentence]
  return ' '.join(tokens) # join all the tokens together to create a new annotated document	

استخدام الدالة annotate_phrases () لتفسير كل من تقنيات التدريب و الاختبار من مجموعة بيانات	

# annotate all the test and train reviews. 	
X_train_text_annotated=[annotate_phrases(doc,imdb_phrase_model) for doc in X_train_text]	

```
X_test_text_annotated=[annotate_phrases(text,imdb_phrase_model) for text in X_test_text]
# visualize the explanation.
```

مثال لمستند من بيانات التدريب من موقع IMDB

```
# an example of an annotated document from the imdb training data
X_train_text_annotated[0]
```

٤-٣ : تحسين البرمجة الاتجاهية للنصوص: ص ١٤٩ الى ص ١٥١

شرح الأمر البرمجي	الكود البرمجي بالكامل

تدريب نموذج TF-IDF (Term Frequency-Inverse Document Frequency) باستخدام مجموعة البيانات التدريبية IMDbواستخدامه لتحويل النصوص إلى تمثيل.TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Train a TF-IDF model with the IMDb training dataset
vectorizer_tf = TfidfVectorizer(min_df=10)
vectorizer_tf.fit(X_train_text_annotated)
X_train_tf = vectorizer_tf.transform(X_train_text_annotated)
```

تدريب نموذج جديد باستخدام التمثيل TF-IDF للبيانات النصية وتقنية Naive Bayes من نوع .MultinomialNB وبعد ذلك، يتم تنفيذ التنبؤات على مجموعة البيانات الاختبارية وحساب الدقة

```
# train a new Naive Bayes Classifier on the newly vectorized data.
model_tf =MultinomialNB()
model_tf.fit(X_train_tf, Y_train)
# create a new prediction pipeline.
prediction_pipeline_tf = make_pipeline(vectorizer_tf, model_tf)
# get predictions using the new pipeline.
predictions_tf = prediction_pipeline_tf.predict(X_test_text_annotated)
# print the achieved accuracy.
accuracy_score(Y_test, predictions_tf)
```

استخدام مثال من مجموعة البيانات الاختبارية الذي قد يكون قد أحدث ارتباكًا للنموذج السابق. الهدف هو فحص التصنيف الصحيح لهذا المثال والتنبؤ بالتصنيف الذي قام به النموذج الجديد

```
# get the review example that confused the previous algorithm
mistake_example_annotated=X_test_text_annotated[4600]
print('\nReview:',mistake_example_annotated)
# get the correct labels of this example.
print('\nCorrect Label:', class_names[Y_test[4600]])
# get the prediction probabilities for this example.
print('\nPrediction Probabilities for neg,
pos:',prediction_pipeline_tf.predict_proba([mistake_example_annotated]))
```

إنشاء مفسر (explainer) باستخدام مكتبة LimeTextExplainer ومن ثم تم استخدامه لشرح التنبؤ الذي تم بواسطة النموذج الجديد للمثال الذي تم استخدامه في الخطوات السابقة. الهدف هو توضيح سبب تصنيف النموذج لهذا المثال مع التركيز على السمات الرئيسية المسهمة في القرار

```
# create an explainer.
explainer_tf = LimeTextExplainer(class_names=class_names)
# explain the prediction of the second pipeline for this example.
exp = explainer_tf.explain_instance(mistake_example_annotated, prediction_pipeline_tf.predict_proba,
num_features=10)
# visualize the results.
fig = exp.as_pyplot_figure()
```