الجمهورية اليمنية
وزارة التعليم العالي والبحث العلمي
جامعة الجــزيرة - إب

كلية العلوم والهندسة
قسم الحاسبات
برنامج تقنية المعلومات
مقرر الذكاء الاصطناعي

الذكاء الاصطناعي عملي

م/ ياسر الشاعري

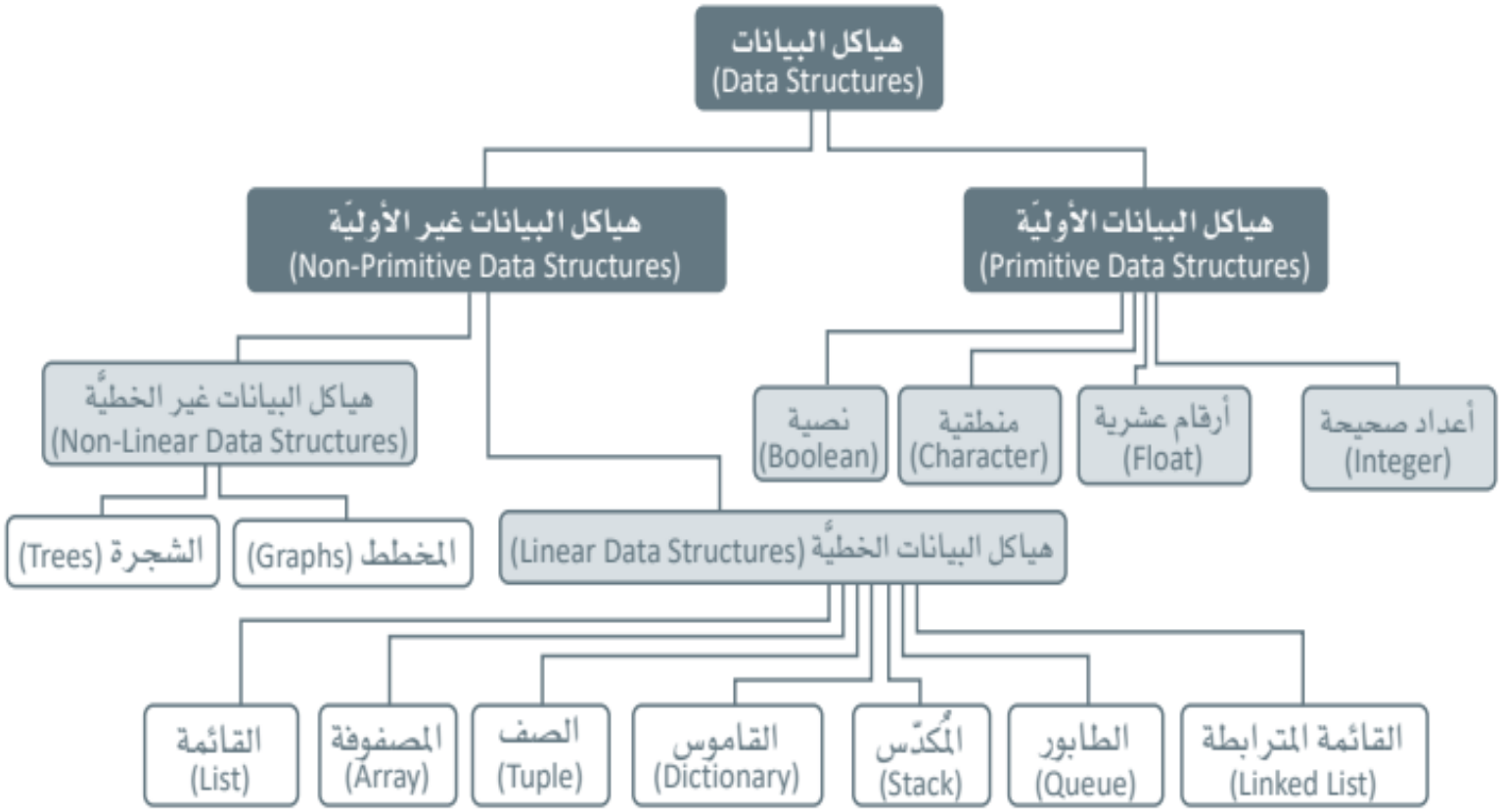
DATA STRUCTURE

Lab-2

هياكل البيانات (Data Structure):

هياكل البيانات هي تقنية لتخزين وتنظيم البيانات في الذاكرة لاستخدامها بكفاءة.

يُطلَق على البيانات البسيطة كذلك البيانات الأوليّة، أو الخام، أو الأساسية.



هياكل البيانات الخطية Linear Data Structures

المكدس Stack

قد يكون حجم المُكدّس ثابتًا أو متغيّرًا ديناميكيًا.
تُطبِّق لغة البايثون المُكدّسات باستخدام القوائم.

قاعدة المُضاف آخرًا يَخرُج أولًا

(Last In First Out-LIFO):
آخر عنصر مُضاف يمكن الوصول إليه أولًا.

العمليات في المَكدّس Operations on the stack

هناك عمليتان رئيستان في المُكدّس:

- إضافة عنصر (Push): تُستخدَم العملية لإضافة عنصر في قمة المُكدّس.
- حذف عنصر (Pop): تُستخدَم العملية لحذف عنصر من قمة المُكدّس.

المُكدّس في لغة البايثون Stack in Python

تُمثَّل المُكدّسات في لغة البايثون باستخدام القوائم التي بدورها تُقدِّم بعض العمليات التي يُمكن تطبيقها مباشرةً على المُكدّسات.

جدول 1.2: عمليات المُكدّس

العملية	الوصف
listName.append(x)	إضافة العنصر x إلى نهاية القائمة.
listName.pop()	حذف العنصر الأخير من القائمة.

تُطبَّق عملية إضافة عنصر للمُكدّس في لغة البايثون باستخدام دالة append.

لتشاهد المثال في الشكل 1.15 في مفكرة جوبيتر:

1. أنشئ المُكدّس لتخزين مجموعة من الأرقام (1، 21، 32، 45).
2. استخدِم عملية **حذف عنصر** (Pop) من المُكدّس مرتين لحذف العنصرين الأخيرين منه.
3. استخدِم عملية **إضافة عنصر** (Push) إلى المُكدّس لإضافة عنصر جديد إليه.

```
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)
```

تُستخدَم الدالة print(myStack.pop()) لعرض القيم المُسترجَعة من دالة myStack.Pop().

```
Initial stack: [1, 21, 32, 45]
45
32
The new stack after pop: [1, 21]
The new stack after push: [1, 21, 78]
```

```
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()
```

تُستخدَم الدالة len لعرض طول المُكدّس.

يُستخدَم هذا الأمر لحذف كل العناصر من المُكدّس.

```
Initial stack: [1, 21, 32, 45]
size of stack 4
[]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [3], in <cell line: 9>()
      7     myStack.pop()
      8 print(myStack)
----> 9 myStack.pop()

IndexError: pop from empty list
```

يظهر الخطأ لأن المُكدّس فارغ وأنت كتبت أمر حذف عنصر منه.

خطأ الفهرس IndexError

ستلاحظ ظهور خطأ عندما كتبتَ أمر حذف عنصر من المُكدّس الفارغ وتسبب هذا في غَيْض المُكدّس (Stack Underflow). عليك دومًا التحقق من وجود عناصر في المُكدّس قبل محاولة حذف عنصر منه.

في البرنامج التالي ستنشئ مُكدّسًا جديدًا وتضيف العناصر إليه، أو تحذفها منه، سيظهر بالبرنامج قائمة تطلب منك تحديد الإجراء الذي تود القيام به في كل مرة.

- لإضافة عنصر إلى المُكدّس، اضغط على الرقم 1 من قائمة البرنامج.
- لحذف عنصر من المُكدّس، اضغط على الرقم 2 من قائمة البرنامج.
- للخروج من البرنامج، اضغط على الرقم 3 من قائمة البرنامج.

```
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;
```

الطابور Queue

قاعدة المُضاف أولًا يَخرُج أولًا (First In First Out (FIFO) rule):
العنصر الأول المُضاف إلى القائمة يُعالَج أولًا، والعنصر الأحدث يُعالَج آخرًا.

العمليات في الطابور Operations on the Queue:

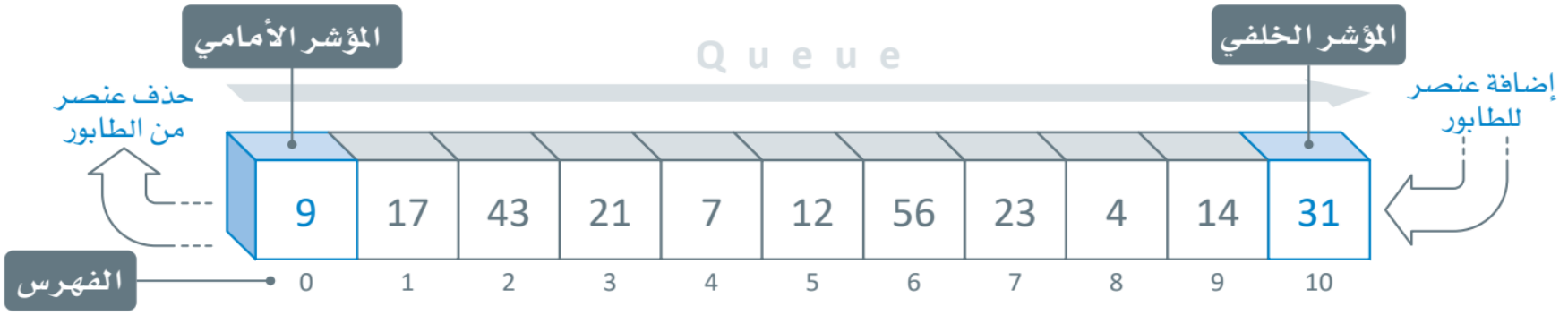
هناك عمليتان رئيستان في الطابور:

- **إضافة عنصر للطابور** (Enqueue): تُستخدَم العملية لإضافة عنصر في آخر الطابور.
- **حذف عنصر من الطابور** (Dequeue): تُستخدَم العملية لحذف عنصر من مقدمة الطابور.

مؤشرات الطابور Queue Pointers

يحتوي الطابور على مؤشرين:

- **المؤشر الأمامي** (Front Pointer): يُشير إلى العنصر الأول في الطابور.
- **المؤشر الأخير** (Rear Pointer): يُشير إلى العنصر الأخير في الطابور.

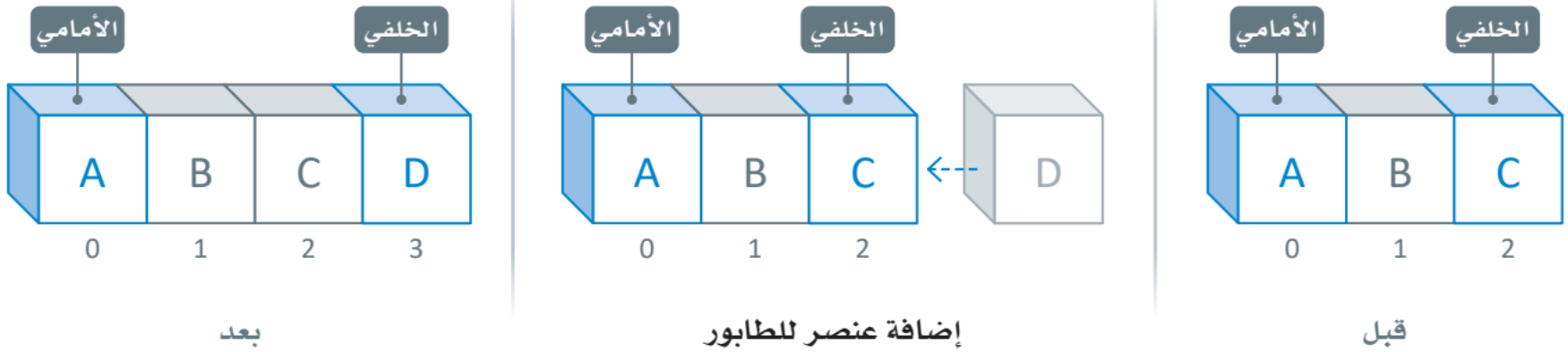


عملية إضافة عنصر للطابور Enqueue Operation

لا يمكنك إضافة عنصر أو حذفه من وسط الطابور.

يُطلق على عملية إضافة عنصر جديد إلى الطابور اسم **إضافة عنصر للطابور** (Enqueue). لإضافة عنصر جديد إلى الطابور:

- تتم زيادة قيمة المؤشر الخلفي بقيمة واحد بحيث يشير إلى موضع العنصر الجديد الذي سيُضاف.
- تتمّ إضافة العنصر.



شكل 1.23: عملية إضافة عنصر للطابور

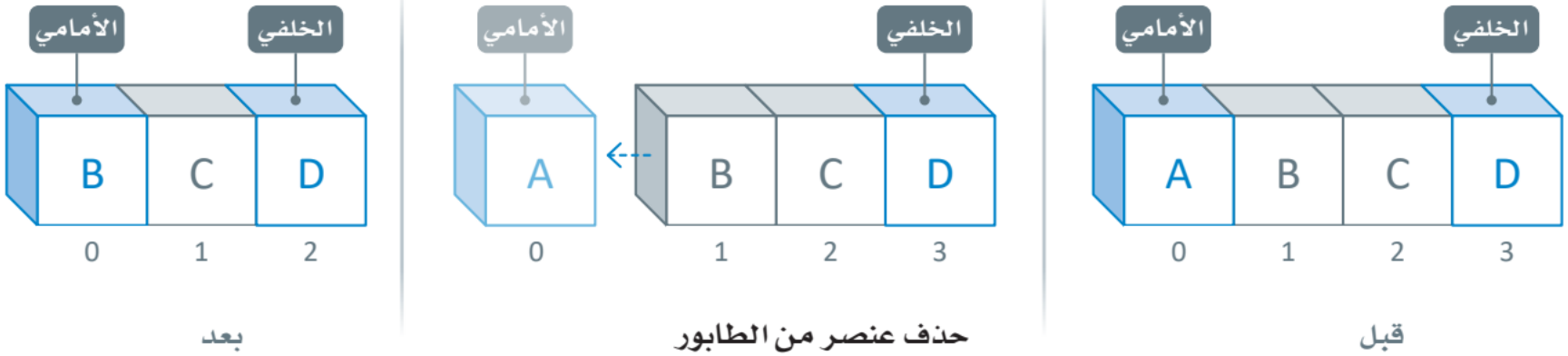
عملية حذف عنصر من الطابور Dequeue Operation

يُطلق على عملية حذف عنصر من الطابور اسم **حذف عنصر من الطابور** (Dequeue).

لحذف عنصر من الطابور:

- يُحذف العنصر المُشار إليه بالمؤشر الأمامي.
- تتم زيادة قيمة المؤشر الأمامي بقيمة واحد بحيث يشير إلى العنصر الجديد التالي في الطابور.

قبل أي إجراء عليك التحقق مما إذا كانت هناك مساحة فارغة في الطابور لإضافة عنصر جديد، وتوافر عنصر واحد على الأقل لتصديره.



شكل 1.24: عملية حذف عنصر من الطابور

الطابور في لغة البايثون Queue in Python

يمكن تمثيل الطابور بعدة طرق متنوعة في لغة البايثون منها **القوائم** (Lists). ويرجع ذلك إلى حقيقة أن القائمة تمثل مجموعة من العناصر الخطيّة، كما يمكن إضافة عنصر في نهاية القائمة وحذف عنصر من بداية القائمة.

ستتعلم فيما يلي الصيغ العامة لبعض العمليات التي يمكن تنفيذها على الطابور:

جدول 1.3: طُرُق الطابور

الطريقة	الوصف
listName.append(x)	تضيف العنصر x إلى القائمة التي تمثل الطابور.
listName.pop(0)	تحذف العنصر الأول من القائمة.

تُستخدَم طريقة listName.pop( ) لكلٍ من هياكل بيانات المُكدّس والطابور. عندما تُستخدَم مع المُكدّس، لا تتطلب الطريقة أي مُعامل. بينما تتطلب الطريقة إضافة صفر إلى المُعامل عندما تُستخدم مع الطابور: listName.pop(0). الفرق بين الدالتين مُوضّح في الجدول 1.4 أدناه.

جدول 1.4: طريقة listName.pop( ) مقابل طريقة listName.pop(0)

الطريقة	الوصف
listName.pop( )	إذا كان مُعامِل الدالة فارغًا، يُحذف العنصر الأخير من نهاية القائمة التي تمثل المُكدّس.
listName.pop(0)	إذا كان مُعامِل الدالة صفرًا، يُحذف العنصر الأول من القائمة التي تمثل الطابور.

سنستعرض لك مثالًا على تطبيق الطابور في لغة البايثون:

- أنشئ طابورًا لتخزين مجموعة من الأرقام (1، 21، 32، 45).
- استخدم عملية حذف عنصر من الطابور مرتين لحذف العنصرين الأوَّلَين منه.
- استخدم عملية إضافة عنصر إلى الطابور لإضافة عنصر جديد إليه.

لبرمجة الخطوات الموضحة بالأعلى بلغة البايثون، ستَستخدِم قائمة البايثون لتنفيذ هيكل الطابور، كما فعلت في المُكدّس.

```
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)
```

```
Initial queue: [1, 21, 32, 45]
The new queue after pop: [32, 45]
The new queue after push: [32, 45, 78]
```

لكي تشاهد ما قد يحدث عندما تحاول حذف عنصر من طابور فارغ، عليكَ أولًا أن تُفرِغ الطابور من العناصر.

```
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)
```

```
Initial queue: [1, 21, 32, 45]
size of queue 4
[]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [6], in <cell line: 9>()
      7     myQueue.pop()
      8 print(myQueue)
----> 9 myQueue.pop()

IndexError: pop from empty list
```

ظهر الخطأ لأنك حاولت حذف عنصر من طابور فارغ.

عليك أن تتحقق دومًا من وجود عناصر في الطابور قبل محاولة حذف عنصر منه.

المُكدّس والطابور باستخدام وحدة الطابور النمطية
Stack and Queue Using Queue Module

يمكن اعتبار القائمة في لغة البايثون بمثابة طابور وكذلك مُكدّس. تُقدِّم لغة البايثون **الوحدة النمطية للطابور** (Queue Module) وهي طريقة أخرى لتنفيذ هيكلَيّ البيانات الموضحين. تتضمن الوحدة النمطية للطابور بعض الدوال الجاهزة للاستخدام التي يمكن تطبيقها على كل من المُكدّس والطابور.

جدول 1.5: وظائف وحدة الطابور النمطية

الوظيفة	الوصف
queueName=queue.Queue( )	تنشئ طابورًا جديدًا اسمه queueName.
queueName.put(x)	تضيف العنصر x إلى الطابور.
queueName.qsize( )	تعود بقيمة حجم الطابور.
queueName.get( )	تعرض وتحذف العنصر الأول من الطابور والعنصر الأخير من المُكدّس.
queueName.full( )	تعود بقيمة True (صحيح) إن كان الطابور ممتلئًا، وقيمة False (خطأ) إن كان الطابور فارغًا، ويمكن تطبيقها على المُكدّس كذلك.
queueName.empty( )	تعود بقيمة True (صحيح) إن كان الطابور فارغًا والقيمة False (خطأ) إن كان الطابور ممتلئًا، يمكن تطبيقها على المُكدّس كذلك.

تُستخدَم وظائف مكتبة الطابور مع كلٍ من المُكدّس والطابور.

ستَستخدِم وحدة الطابور النمطية لإنشاء طابور.

في هذا المثال عليك:

- استيراد مكتبة **الطابور** (Queue) لاستخدام طُرُق الطابور.
- إنشاء طابور فارغ باسم myQueue (طابوري).
- إضافة العناصر a، b، c، d، e إلى الطابور myQueue (طابوري).
- طباعة عناصر الطابور.

```
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)
```

```
a
b
c
d
e
```

عليك استيراد وحدة الطابور في بداية المقطع البرمجي.

أنشئ طابورًا مُكوَّنًا من خمس قيم يقوم المُستخدِم بإدخالها أثناء تنفيذ البرنامج، ثم اطبع هذه القيم، وفي النهاية اطبع حجم الطابور.

```
from queue import *

myQueue = Queue()

# the user enters the elements of the queue for i in range(5):
for i in range(5):
   element=input("enter queue element: ")
   myQueue.put(element)

# print the elements of the queue
for element in list(myQueue.queue):
   print(element)

print ("Queue size is: ",myQueue.qsize())
```

```
enter queue element: 5
enter queue element: f
enter queue element: 12
enter queue element: b
enter queue element: 23
5
f
12
b
23
Queue size is: 5
```

أنشئ برنامجًا للتحقق مما إذا كان الطابور فارغًا أم ممتلئًا.

جدول 1.6: وظائف الوحدة المُستخدمة للمُكدّس

الوظيفة	الوصف
stackName=queue.LifoQueue()	تنشئ مُكدّسًا جديدًا اسمه stackName.
stackName.get()	تحذف العنصر الأخير من المُكدّس.

ستَستخدِم وحدة الطابور لإنشاء مُكدّس فارغ.

```
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)

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

تذكّر أن العمليات في المُكدّس تعمل وفقًا لقاعدة المُضاف آخرًا يَخرُج أولًا (LIFO).

عند استخدام دالة get مع الطابور، ستَستند عمليات الاستدعاء والطباعة إلى قاعدة المُضاف أولًا يَخرُج أولًا (FIFO).

```
e
d
c
b
a
Is the stack empty? True
```

مثال: الطباعة Print

يظهر أمامك في المثال التالي محاكاة لطابور الطباعة في الطابعة. عندما يُرسِل المُستخدِمون أوامر طباعة، تُضاف إلى طابور الطباعة. تَستخدِم الطابعة هذا الطابور لتحديد الملف الذي سيُطبع أولًا.

- افترض أن سعة الطابعة هي فقط 7 ملفات، ولكن في الوقت نفسه، تحتاج إلى طباعة 10 ملفات من الملف A إلى الملف J.
- اكتب برنامجًا يُمثّل طابور الطباعة منذ بدء أمر الطباعة الأول A حتى الانتهاء من كل أوامر الطباعة.
- أضف اللبنة التي تؤكد أن طابور أوامر الطباعة فارغ.

يُمكنك استخدام الخوارزمية الآتية:

1 أنشئ طابور أوامر الطباعة.

2 أدرِج الملفات من A إلى G في طابور أوامر الطباعة.

3 أخرِج الملف A وأدرج الملف H.

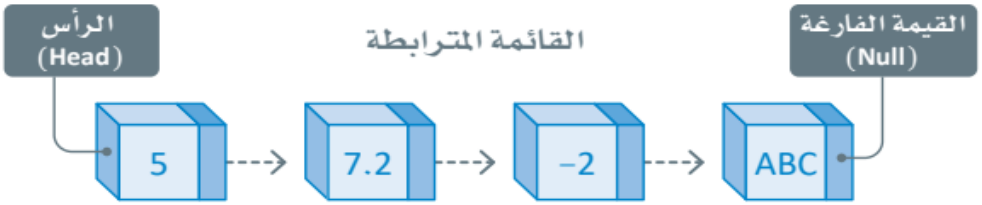
4 أخرِج الملف B وأدرج الملف I.

5 أخرِج الملف C وأدرج الملف J.

6 أخرج الملفات التي تمت طباعتها (D-E-F-G-H-I-J) واحدًا تلو الآخر.

القائمة المترابطة Linked List

القائمة المترابطة هي نوع من هياكل البيانات الخطيَّة، وهي واحدة من هياكل البيانات الأكثر شهرة في البرمجة. القائمة المترابطة تشبه سلسلة من العُقد. تحتوي كل عُقدة على حقلين: حقل البيانات حيث تُخزن البيانات، وحقل يحتوي على المؤشر الذي يُشير إلى العُقدة التالية. يُستثنى من هذا العُقدة الأخيرة التي لا يحمل فيها حقل العنوان أي بيانات. إحدى مزايا القائمة المترابطة هي أن حجمها يزداد أو يقل بإضافة أو حذف العُقد.



شكل 1.27: رسم توضيحي للقائمة المترابطة

القائمة المترابطة (Linked List):

القائمة المترابطة هي نوع من هياكل البيانات الخطيَّة التي تشبه سلسلة من العُقد.

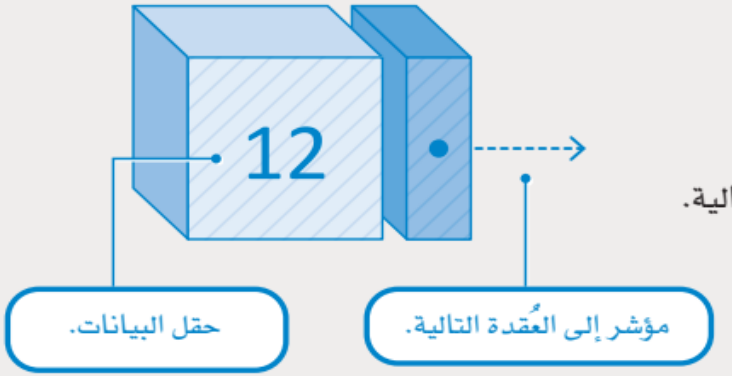
العُقدة (Node):

العُقدة هي اللبنة الفردية المُكوِّنة لهيكل البيانات وتحتوي على البيانات ورابط واحد أو أكثر من الروابط التي تربطها بالعُقد الأخرى.

العُقدة Node

تتكون كل عُقدة في القائمة المترابطة من جزئين:

- الجزء الأول يحتوي على البيانات.
- الجزء الثاني يحتوي على مؤشر يُشير إلى العُقدة التالية.

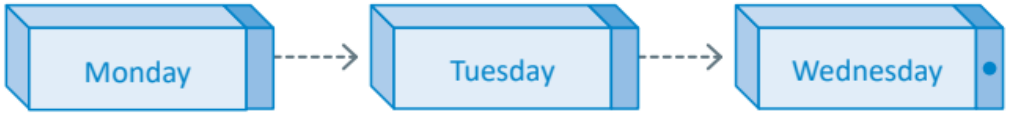


شكل 1.28: رسم توضيحي للعُقد

لقراءة محتوى عُقدة محددة، عليك المرور على كل العُقد السابقة.

القائمة المترابطة في لغة البايثون Linked List in Python

لا تُوفر لغة البايثون نوع بيانات مُحدَّد مُسبقًا للقوائم المترابطة. عليك إنشاء نوع البيانات الخاص بك، أو استخدام مكتبات البايثون التي توفر تمثيلًا لهذا النوع من البيانات. لإنشاء قائمة مترابطة، استخدم فئات البايثون. في المثال الموضح بالشكل 1.32، ستُنشئ قائمة مترابطة مكونة من ثلاث عُقد، كل واحدة تضم يومًا من أيام الأسبوع.



شكل 1.32: مثال على القائمة المترابطة

الفئة (Class):
الفئة هي هيكل بيانات معرّف بواسطة المستخدم، ويحتوي على أعضاء البيانات (**السمات** Properties)، والطرائق (**السلوك** Behavior) الخاصة بها. وتُستخدَم الفئات كقوالب لإنشاء الكائنات.

ستُنشئ أولًا عُقدة باستخدام الفئة.

```
# single node
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)
```

```
Monday
```

الخطوة التالية هي إنشاء قائمة مترابطة تحتوي على عُقدة واحدة، وهذه المرة ستَستخدِم مؤشر الرأس للإشارة إلى العُقدة الأولى.

```
# 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)
```

```
Monday
```

أضِفْ الآن المزيد من العُقد إلى القائمة المترابطة.

```
# single node
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
```

تُستخدَم عبارة while للتنقل من عُقدة إلى أخرى.

```
Monday
Tuesday
Wednesday
```

إضافة العُقدة إلى القائمة المترابطة

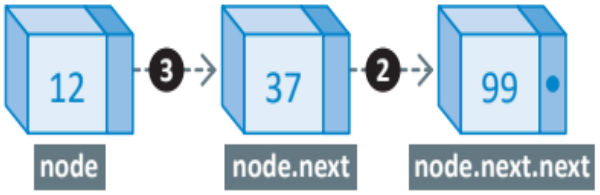
Add a Node to a Linked List

لتتمكن من إضافة عُقدة جديدة، اتبع الخطوات التالية:

- يجب أن يُشير مؤشر العُقدة الأولى إلى عنوان العُقدة الجديدة، حتى تصبح العُقدة الجديدة هي العُقدة الثانية.
- يجب أن يُشير مؤشر العُقدة الجديدة (الثانية) إلى عنوان العُقدة الثالثة.

بهذه الطريقة، لن تحتاج إلى تغيير العناصر عند إضافة عنصر جديد في المنتصف. تقتصر العملية على تغيير قِيَم العناوين في العُقدة التي تُسرِّع من عملية الإضافة في حالة القوائم المترابطة، مقارنةً بحالة القوائم المتسلسلة.

1. أنشئ العُقدة الجديدة.



2. اربط العُقدة 37 بالعُقدة 99.
3. اربط العُقدة 12 بالعُقدة 37 (تمت إضافة العُقدة الجديدة).

حذف العُقدة من القائمة المترابطة Delete a Node from a Linked List

لحذف عُقدة، عليك تغيير مُؤشر العُقدة التي تسبق العُقدة المراد حذفها إلى مؤشر العُقدة التي تلي العُقدة المحذوفة. أصبحت العُقدة المحذوفة (الثانية) عبارة عن **بيانات غير مُفيدة** (Useless Data) وستُخصَّص مساحة الذاكرة التي تشغلها لاستخدامات أخرى.

مثال:

لديك قائمة مترابطة من عنصرين: 12 و99، وتريد إدراج العنصر 37 كعنصر ثانٍ بالقائمة. في النهاية، سيكون لديك قائمة من ثلاثة عناصر: 12 و37 و99.

```
# 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
```