

الكافى

فى البايثون



الجزء الثانى

أبو حبيب الحسينى

ملحوظة مهمة جدا

ان وجود الكلمات الانجليزية فى وسط الجمل العربية ينقل بعض الكلمات من مكانها فتظهر الجمل بشكل غير صحيح ويصعب فهما وهذا عيب فى الترميز (يو تى اف)

مثال على هذا الكلام

الحلقة كتلة من الاكواد التي سيتم **for** المحجوزة في **else** تحدد الكلمة
تنفيذها عند انتهاء الحلقة

لاحظ هنا ان الجملة اصبحت غير مفهومة الى حد ما او غير مرتبة بشكل صحيح لان بعض الكلمات نقلت من مكانها بسبب وضع كلمات انجليزية وسط الجمل العربية فافى مثل هذه الحالات حاول ان تستنتج الجمله بنفسك وتفهما

حاولنا تقليل هذا العيب قدر المستطاع باستخدام بكتابة المصطلحات الانجليزية باللغة العربية مثل (سي اس اس) او (نود جى اس) وكذلك نقلنا اتجاة الصفحة من اليسارى الى اليمين لتفادى هذا العيب وللأسف لم تتم المعالجة بنسبة مئة بالمئه

Glade

عملاق تصميم وجهات بايثون فى ثوانى وتوليد
الاكواد اتوماتيكيا
انشا برامج ضخمة ببرمجة الاحداث فقط



ابو حبيب الحسينى

شرح احترافى لهذا العملاق المجانى فارس انشاء البرامج فى ثوانى وهو مدمج
فى معظم توزيعات لينكس

الفهرس

الفهرس.....	6
دوال بايثون.....	13
إنشاء دالة.....	13
استدعاء دالة.....	13
البرامطرات.....	14
الكلمات أو البرامطرات؟.....	15
عدد البرامطرات.....	15
البرامطرات الاثنائية ، *.....	16
وبرامترات الكلمات الرئيسية.....	17
**kwargs ، برامترات الكلمات الرئيسية الاثنائية.....	18
القيمة الاساسيه الافتراضية.....	18
تمرير القائمة كبرامطر.....	19
إرجاع القيم من اكواد الداله.....	20
قوانين الارجاع.....	20
دالة lambda.....	22
بناء الجملة.....	22
لماذا نستخدم دالة لمبادا؟.....	23
مصفوفات بايثون.....	25
المصفوفات بدلا من القوائم والقواميس التي درسناها فى الجزء الاول.....	25
ما هي المصفوفة؟.....	26
تعامل طبيعى مثل القوائم تمام.....	27
ارجاع طول المصفوفة نفس طريقة القوائم.....	27
كلاسات وكائنات بايثون.....	28
كلاسات/كائنات بايثون او الفئات.....	28
كيف (class) إنشاء الفئات.....	28
إنشاء كائن.....	29
الدالة __init__().....	29
الدالة __str__().....	31
دوال الكائن.....	32
الكلمة Self.....	33
تعديل خصائص الكائن.....	34

حذف خصائص الكائن.....	34
حذف الكائنات.....	35
كلمة المرور اذا كان الكلاس فارغ.....	35
الوراثة فى بايثون.....	35
سهولة وراثة بايثون.....	36
إنشاء كلاس اساسى.....	36
إنشاء كلاس فرعى.....	37
جلب الكائن كنسخة.....	38
كيف جلب نسخة من الكلاس.....	38
النسخ من خلال الحلقات.....	40
إنشاء نسخة.....	41
StopIteration.....	42
نطاق الوصول فى بايثون.....	43
النطاق المحلي.....	43
النطاق العام.....	45
الكلمة الرئيسية العامة.....	46
وحدات بايثون او الموديول.....	47
ما هي الوحدة؟.....	48
إنشاء وحدة.....	48
استخدم الوحدة النمطية موديول.....	48
المتغيرات فى الوحدة النمطية موديول.....	49
تسمية الوحدة النمطية موديول.....	50
إعادة تسمية الوحدة النمطية موديول.....	50
وحدات جاهزة مدمجة فى اللغة.....	51
dir() باستخدام الدالة.....	51
استيراد من الوحدة النمطية موديول الى التطبيق.....	52
الوقت والتاريخ.....	53
تاريخ الإخراج.....	54
إنشاء كائنات التاريخ.....	55
دالة strftime().....	56
دوال الرياضيات.....	58
وحدة الرياضيات.....	59
فى بايثون JSON.....	61
تحويل من جسون إلى قاموس بايثون والعكس.....	61
JSON التحويل من بايثون إلى.....	62
اهمية التنسيق فى جسون.....	65

json.dumps().....	66
RegEx وحدة.....	67
RegEx في بايثون.....	67
RegEx دوال.....	68
الأحرف الأولية.....	68
تنسيقات خاصة.....	69
مجموعات.....	70
findall() الدالة.....	70
search دالة البحث.....	72
split دالة التقسيم.....	73
sub() دالة.....	74
طرق البحث.....	75
PIP استخدام.....	78
PIP ما هو.....	78
ما هي الحزمة؟.....	78
PIP تحقق من تثبيت.....	78
تثبيت نسخة.....	79
باستخدام الحزمة.....	79
البحث عن الحزم.....	80
إزالة الحزمة.....	80
قائمة الحزم.....	81
معالجة الاخطا فى بايثون.....	82
معالجة الاستثناء.....	82
العديد من الاستثناءات فى الاخطاء.....	83
نفيذ اجراء اذا فشل الكود.....	84
التنفيذ على اى حال.....	85
رفع استثناء.....	86
إدخال المستخدم.....	87
تنسيق النصوص فى بايثون.....	88
format ().....	88
قيم كثيره.....	90
رقمنة التنسيق.....	90
اسماء مختصرة للارقام.....	91
التعامل مع الملفات فى لغة بايثون.....	92
فتح ملف بايثون.....	92
بناء الجملة.....	93

فتح ملف بايثون.....	93
افتح ملفًا على الخادم.....	93
قراءة أجزاء فقط من الملف.....	95
قراءة السطور.....	95
إغلاق الملفات.....	96
كتابة ملف بايثون.....	97
الكتابة إلى ملف موجود.....	97
إنشاء ملف جديد.....	98
حذف الملف في بايثون.....	99
حذف ملف.....	99
التحقق من وجود الملف:	100
احذف المجلد.....	100
برنامج التشغيل ماي إس كيو إل.....	101
MySQL قم بتثبيت برنامج تشغيل.....	101
MySQL في بايثون.....	102
إنشاء اتصال.....	102
اولا إنشاء برنامج التشغيل.....	103
تحقق من وجود قاعدة البيانات.....	104
إنشاء الجدول MySQL بايثون.....	106
إنشاء جدول.....	106
تحقق من وجود الجدول.....	107
المفتاح الأساسي.....	108
الإدراج في الجدول.....	110
إدراج في الجداول.....	110
إدراج صفوف كثيرة.....	111
معرف الخانة.....	113
حدد العناصر من جدول.....	114
اختيار الأعمدة معينة.....	115
fetchone() باستخدام دالة.....	116
كلمة أين MySQL بايثون.....	117
حدد مع اختيارات معينة.....	117
التحكم في البحث.....	118
SQL منع الهندسة العكسية.....	120
ترتيب حسب MySQL بايثون.....	121
فرز الجدوال.....	121
الطلب حسب الوصف.....	122

MySQL حذف الجدوال.....	123
كيف حذف جدول.....	123
احذف فقط إذا كان موجودا.....	124
تحديث الجداول MySQL.....	125
كيف يتم تحديث الجدول.....	125
تابع منع الهندسة العكسية.....	127
MySQL حدود بايثون.....	128
الحد من ال.....	128
أبدأ البحث من موضع آخر.....	129
ضم إلى جدولين أو أكثر.....	130
الانضمام الأيسر.....	132
الانضمام الصحيح.....	133
اختبار بايمونجو.....	135
كيف إنشاء برنامج التشغيل.....	136
تحقق من وجود قاعدة البيانات.....	137
إنشاء مجموعة MongoDB.....	138
إنشاء مجموعة.....	138
تحقق من وجود المجموعة.....	139
إدراج مستند MongoDB بايثون.....	140
إدراج في المجموعة.....	140
_id قم بإرجاع حقل.....	141
إدراج مستندات كثيرة.....	142
قم بإدراج مستندات كثيرة بمعرفات محددة.....	143
MongoDB البحث في.....	145
ابحث عن واحدة.....	145
اوجد الكل.....	146
إرجاع بعض الحقول فقط.....	147
MongoDB استعلام بيثون.....	149
تصفية الجدول.....	150
الاستعلام المتقدم.....	151
تصفية مع التعبيرات العادية.....	152
MongoDB فرز و ترتيب.....	153
فرز.....	153
ترتيب تنازلي.....	154
MongoDB Drop المجموعات.....	154
حذف المجموعة.....	155

تحديث المجموعة.....	156
تحديث متعدد.....	157
MongoDB حدود جلب البيانات.....	158
تحديد مستندات او سجلات معينة فى الجداول.....	158
مكتبة NumPy.....	162
التعلم عن طريق القراءة.....	163
NumPy مقدمة.....	163
NumPy؟ بختصار.....	163
NumPy الفرق بينها وبين القوائم والقواميس.....	163
لماذا نامبى أسرع من القوائم؟.....	164
ما هي اللغة التي تمت كتابة نامبى بها؟.....	164
NumPy أين توجد برنامج التشغيل.....	165
البدء مع NumPy.....	165
NumPy تثبيت.....	165
NumPy عمليات الاستيراد.....	165
np اسم مختصر لـ NumPy.....	166
NumPy التحقق من إصدار.....	167
إنشاء المصفوفات NumPy.....	167
NumPy ndarray قم بإنشاء كائن.....	167
الأبعاد في المصفوفات.....	169
(-D) مصفوفات (0).....	169
D- مصفوفات 1.....	170
مصفوفات ثنائية الأبعاد.....	170
مصفوفات ثلاثية الأبعاد.....	171
التحقق من عدد الأبعاد؟.....	172
مصفوفات الأبعاد العليا.....	173
NumPy رقمة مصفوفة.....	173
الوصول إلى عناصر المصفوفة.....	174
الوصول إلى المصفوفات ثنائية الأبعاد.....	175
الوصول إلى المصفوفات ثلاثية الأبعاد.....	176
الأرقام السالبة.....	178
تقطيع المصفوفات.....	178
التقطيع بالسالب.....	180
التقطيع.....	181
تقطيع المصفوفات ثنائية الأبعاد.....	182
أنواع الاجرائات.....	183

أنواع الاجرائات الافتراضية في بايثون.....	183
NumPy أنواع الاجرائات في.....	184
التحقق من نوع بيانات المصفوفة.....	185
إنشاء مصفوفات بنوع بيانات محدد.....	186
ماذا لو لم يكمن تحويل القيمة؟.....	187
تحويل نوع البيانات على المصفوفات الموجودة.....	187
النسخ والعرض في NumPy.....	189
الفرق بين النسخ والعرض.....	190
النسخ:.....	190
العرض:.....	191
تمتلك اكوادها Array تحقق مما إذا كانت.....	192
NumPy شكل او سمة المصفوفة.....	193
كيف شكل مصفوفة.....	193
tuple؟ ماذا يمثل شكل.....	194
إعادة تشكيل المصفوفات.....	195
D-2 إلى D-1 إعادة التشكيل من 1.....	196
D-3 إلى D-1 إعادة التشكيل من 1.....	196
هل نستطيع إعادة تشكيل أي شكل؟.....	197
إرجاع نسخة أو عرض؟.....	198
البعد غير معروف.....	199
تسطيح المصفوفات.....	199
NumPy تكرار مصفوفة.....	200
تكرار المصفوفات.....	201
تكرار المصفوفات ثنائية الأبعاد.....	201
تكرار المصفوفات ثلاثية الأبعاد.....	203
nditer() تكرار المصفوفات باستخدام.....	204
تكرار المصفوفة مع أنواع الاجرائات المختلفة.....	205
النسخ مع احجام التقطيع المختلفة.....	206
ndenumerate() النسخ التعدادي باستخدام.....	206
الانضمام والدمج بين المصفوفات NumPy.....	208
NumPy الانضمام بين المصفوفات.....	208
الانضمام إلى المصفوفات باستخدام دوال الفهرس.....	209
التراص او التتابع او التتابع على طول المصفوفة.....	210
التراص او التتابع على طول الأعمدة.....	211
التراص او التتابع على طول الارتفاع (للعق).....	212
NumPy التقسيم.....	212

NumPy تقسيم المصفوفات.....	212
تقسيم إلى مصفوفات صغيرة.....	214
تقسيم المصفوفات ثنائية الأبعاد.....	215
NumPy البحث.....	218
البحث في مصفوفة.....	218
قم بفرز البحث.....	220
فرز المصفوفات.....	222
فرز مصفوفة ثنائية الأبعاد.....	224

دوال بايثون

الدالة عبارة عن كتلة من الاكواد التي تعمل فقط عند استدعائها

تستطيع تمرير الاكواد ، المعروفة باسم الكلمات، إلى دالة

يمكن للدالة إرجاع الاكواد لذلك

إنشاء دالة

def : يتم تعريف الدالة في لغة بايثون باستخدام الكلمة المحجوزة

مثال ↘ ↘

```
def alHosini():  
    print("Abo Habib Al_Hosiny *_* 🍀🍀")
```

استدعاء دالة

:لاستدعاء دالة، استخدم اسم الدالة متبوعًا بقوسين

مثال ↘ ↘

```
def alHosini():  
    print("Abo Habib Al_Hosiny *_* 🍀🍀")
```

```
alHosini()
```

البرامترات

يمكن تمرير المعلومات إلى الدوال كوسائط

يتم تحديد الوسائط بعد اسم الدالة، داخل الأقواس. تستطيع إضافة أي عدد تريده من الوسائط، ما عليك سوى الفصل بينها بفاصلة

عندما يتم (hoshos) يحتوي مثال \ \ التالي على دالة ذات وسيطة واحدة استدعاء الدالة، نقوم بتمرير الاسم الأول، والذي يتم استخدامه داخل الدالة لطباعة الاسم الكامل:

مثال \ \

```
def alHosini(hoshos):
```

```
    print(hoshos + " Habib")
```

```
alHosini("Yaseen")
```

```
alHosini("Ahmed")
```

```
alHosini("Omar")
```

غالبًا ما يتم اختصار الوبرامتورات إلى وبرامتورات في وثائق بايثون.

الكلمات أو البرامتورات؟

يمكن استخدام الكلمة والوسيلة لنفس الشيء: المعلومات التي يتم تمريرها إلى دالة .

مت منظور الدالة:

الكلمة هي المتغير المدرج داخل الأقواس في تعريف الدالة.

الوسيلة هي القيمة التي يتم إرسالها إلى الدالة عند استدعائها.

عدد البرامترات

بشكل افتراضي، يجب استدعاء الدالة بالعدد الصحيح من الوسائط. وهذا يعني أنه إذا كانت وظيفتك تتوقع وسيطتين، فيجب عليك استدعاء الدالة باستخدام وسيطتين، وليس أكثر ولا أقل.

مثال ↘ ↘

تتوقع هذه الدالة وسيطتين، وتحصل على وسيطتين

```
def alHosini(hoshos, bibo):
```

```
    print(hoshos + " " + bibo)
```

```
alHosini("Habib", "Al Hosiny")
```

إذا حاولت استدعاء الدالة باستخدام وسيطة واحدة أو ثلاث وبرامترات، فسوف تحصل على خطأ:

مثال ↘ ↘

تتوقع هذه الدالة وسيطتين، ولكنها تحصل على وسيطة واحدة فقط

```
def alHosini(hoshos, bibo):
```

```
    print(hoshos + " " + bibo)
```

```
alHosini("Emil")
```

* البرامترات الالثنائية

إذا كنت لا تعرف عدد الوسائط التي سيتم تمريرها إلى وظيفتك، فقم بإضافة * قبل اسم الكلمة في تعريف الدالة.

بهذه الدالة ستتلقى الدالة مجموعة من الوسائط، وتستطيع الوصول إلى العناصر وفقاً لذلك:

مثال ↘ ↘

إذا كان عدد الوسائط غير معروف، أضف * قبل اسم الكلمة:

```
def alHosini(*kids):
```

```
    print("The youngest child is " + kids[2])
```

```
alHosini("Habib", "Ahmed", "Omar", "Yaseen")
```

في وثائق بايثون ***args** غالباً ما يتم اختصار البرامترات الالثنائية إلى

وبرامترات الكلمات الرئيسية

. تستطيع أيضاً إرسال الوسائط باستخدام صيغة المفتاح = القيمة

. بهذه الدالة لا يهم ترتيب البرامترات

↓↓↓ مثال ↘ ↘

```
def alHosini(habib3, habib2, habib1):  
    print("The youngest habib is " + habib3)
```

```
alHosini(habib1 = "Habib", habib2 = "Ahmed", habib3  
= "Omar")
```

في **kwargs** إلى **Keyword Arguments** غالبًا ما يتم اختصار عبارة
وثائق بايثون

، برامترات الكلمات الرئيسية الإلتنائية ****kwargs**

إذا كنت لا تعرف عدد وسائط الكلمات الرئيسية التي سيتم تمريرها إلى
وظيفتك، أضف علامتين نجميتين: ****** قبل اسم الكلمة في تعريف الدالة

بهذه الدالة ستتلقى الدالة قاموسًا للوسائط، وتستطيع الوصول إلى العناصر وفقًا
لذلك:

مثال ↘ ↘

إذا كان عدد وبرامترات الكلمات الرئيسية غير معروف، أضف رقمًا
:مزدوجًا ****** قبل اسم الكلمة


```
def alHosini(**kid):  
    print("His last name is " + kid["bibo"])
```

```
alHosini(hoshos = "Ahmed", bibo = "Habib")
```

في وثائق ****kwargs** الاثثنائية إلى **Kword** غالبًا ما يتم اختصار وبرنامجات بايثون.

القيمة الافتراضية

يوضح **مثال** التالي كيفية استخدام قيمة الكلمة الافتراضية إذا قمنا باستدعاء الدالة بدون وسيطة، فإنها تستخدم القيمة الافتراضية

مثال

```
def alHosini(Books = "JavaScript"):  
    print("I am from " + Books)
```

```
alHosini("")  
alHosini("jQuery")  
alHosini()  
alHosini("Vue.js")
```

تمرير القائمة كبرامطر

تستطيع إرسال أي نوع من أنواع الاجراءات إلى دالة (نص أو رقم أو قائمة أو قاموس وما إلى ذلك)، وسيتم التعامل معها على أنها نفس نوع البيانات داخل الدالة.

على سبيل **مثال** ↘ ↘ إذا قمت بإرسال قائمة كوسيط، فستظل قائمة عندما تصل إلى الدالة:

مثال ↘ ↘

```
def alHosini(food):  
    for x in food:  
        print(x)
```

```
abibo= ["Abo", "Habib", "Al Hosiny"]
```

```
alHosini(abibo)
```

إرجاع القيم من اكواد الداله

العبارة **return** للسماح للدالة بإرجاع قيمة، استخدم:

مثال ↘ ↘

```
def alHosini(x):  
    return 5 * x
```

```
print(alHosini(3))
print(alHosini(5))
print(alHosini(9))
```

قوانين الارجاع

تقبل بايثون أيضًا عودية الدالة، مما يعني أن الدالة المحددة تستطيع استدعاء نفسها من داخلها .

الارجاع هو مفهوم رياضي وبرمجي شائع. وهذا يعني أن الدالة ستعيد لك ناتج الاكواد الى تمت بداخلها او يمكن ان تستدعي نفسها. مرة اخرى عند حدوث امر معين وهذا له فائدة أنه تستطيع تكرار الاكواد واختصرها بدلا من كتابتها مرات عديدة للوصول السريع .

يجب أن يكون المطور حذرًا جدًا فيما يتعلق بالنسخ لأنه قد يكون من السهل جدًا الانزلاق إلى كتابة دالة لا تنتهي أبدًا، أو دالة تستخدم كميات زائدة من الذاكرة أو طاقة المعالج. ومع ذلك، عند كتابتها بشكل صحيح، يمكن أن تكون الارجاع دالة فعالة جدًا وأنيقة رياضياً للبرمجة.

هي دالة قمنا بتعريفها لتسمي نفسها **Habib()**، في هذا مثال كالاكواد، والذي يتناقص (-1) في كل **k** نحن نستخدم المتغير ("recurse"). مرة نكرر فيها. تنتهي الارجاع عندما لا يكون الشرط أكبر من 0 (أي عندما يكون 0).

بالنسبة للمطور الجديد، قد يستغرق الأمر بعض الوقت لمعرفة كيفية عمل ذلك بالضبط، وأفضل دالة لمعرفة ذلك هي اختباره وتعديله.

مثال ↘ ↘

```
def Habib(k):  
    if(k > 0):  
        result = k + Habib(k - 1)  
        print(result)  
    else:  
        result = 0  
    return result
```

```
print("\n\nAbo Habib Al-Hosiny *_*")  
Habib(6)
```

دالة lambda

دالة لمبادا هي دالة مجهولة صغيرة

يمكن لدالة لمبادا أن تأخذ أي عدد من الوسائط، ولكن يمكن أن تحتوي على تعبير واحد فقط.

بناء الجملة

lambda arguments : expression

يتم تنفيذ التعبير وإرجاع ال

مثال ↘ ↘

نأرجع ال **a** أضف 10 إلى الوسيطة

```
x = lambda a : a + 10  
print(x(5))
```

أن تأخذ أي عدد من الوسائط Lambda يمكن لدوال

مثال ↘ ↘

نأرجع ال **b** بالوسيطة **a** اضرب الوسيطة

```
x = lambda a, b : a * b  
print(x(5, 6))
```

مثال ↘ ↘

نأرجع ال **c** و **b** و **a** تلخيص الوسيطة

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

لماذا نستخدم دالة لمبادا؟

بشكل أفضل عند استخدامها كدالة مجهولة داخل دالة `lambda` تظهر قوة أخرى.

لنفترض أن موجود تعريف دالة يأخذ وسيطة واحدة، وسيتم ضرب هذه الوسيطة برقم غير معروف:

```
def HoSini(n):
    return lambda a : a * n
```

استخدم تعريف الدالة هذا لإنشاء دالة تضاعف دائماً الرقم الذي ترسله

مثال ↘ ↘

```
def HoSini(n):
    return lambda a : a * n
```

```
HosHos = HoSini(2)
```

```
print(HosHos(11))
```

أو استخدم نفس تعريف الدالة لإنشاء دالة تضاعف دائماً الرقم الذي ترسله ثلاث مرات:

مثال ↘ ↘

```
def HoSini(n):  
    return lambda a : a * n
```

```
HosHos = HoSini(3)
```

```
print(HosHos(11))
```

أو استخدم نفس تعريف الدالة لإجراء كلتا الوظيفتين في نفس البرنامج

مثال ↘ ↘

```
def HoSini(n):  
    return lambda a : a * n
```

```
HosHos = HoSini(2)
```

```
HosHos = HoSini(3)
```

```
print(HosHos(11))
```

```
print(HosHos(11))
```

✓ مصفوفات بايثون

ملاحظة: لا تحتوي لغة بايثون على دعم مدمج للمصفوفات، ولكن يمكن استخدامها قوائم بايثون بدلاً من ذلك.

المصفوفات بدلا من القوائم والقواميس التي درسناها في الجزء الاول

ملاحظة: توضح لك هذه الصفحة كيفية استخدام القوائم كمصفوفات، ومع ذلك، للعمل مع المصفوفات باحرف في بايثون، سيتعين عليك استيراد مكتبة، نامباى التي تتيح لك التعامل مع المصفوفات بدلا من القوائم والقواميس والجداول التي درسناها وذه ما سنعرفة فى الفصول القادمة.

تستخدم المصفوفات لتخزين قيم كثيرة في متغير واحد

مثال ↘ ↘

قم بإنشاء مصفوفة تحتوي على أسماء المستخدمين

```
Hosinii = ["Abu Habib Al Hosiny *_  
*_", "Omar", "Mohamed"]
```

ما هي المصفوفة؟

المصفوفة عبارة عن متغير خاص يمكنه الاحتفاظ بأكثر من قيمة واحدة في المرة الواحدة.

إذا كانت موجود قائمة بالعناصر (قائمة بأ، على سبيل مثال ↘ ↘)، فقد يبدو:
تخزين المستخدمين في متغيرات فردية كما يلي:

HAbiB1 = "Habib"

HAbiB2 = "Omar"

HAbiB3 = "Mohamed"

ومع ذلك، ماذا لو كنت تريد التنقل بين المستخدمين والعثور على مستخدم معينة؟ وماذا لو لم يكن موجود 3 سيارات بل 300؟

!الحل هو مصفوفة

يمكن للمصفوفة أن تحتوي على العديد من القيم تحت اسم واحد، وتستطيع الوصول إلى القيم عن طريق الإشارة إلى الرقم .

تعامل طبيعي مثل القوائم تمام

. تستطيع الرجوع إلى عنصر مصفوفة بالإشارة إلى رقم الرقم

مثال ↘ ↘

:احصل على قيمة عنصر المصفوفة الأول

| x = Hosinii[0]

مثال ↘ ↘

:تعديل قيمة عنصر المصفوفة الأول

Hosinii[0] = "Abo Osama"

إرجاع طول المصفوفة نفس طريقة القوائم

الدالة لإرجاع طول المصفوفة (عدد العناصر في المصفوفة) `len()` استخدم

مثال ↘ ↘

المصفوفة `Hosinii` إرجاع عدد العناصر في

`x = len(Hosinii)`

ملاحظة: يكون طول المصفوفة دائماً أكبر من مؤشر المصفوفة الأعلى بمقدار واحد

✓ كلاسات وكائنات بايثون

كلاسات/كائنات بايثون أو الفئات

بايثون هي لغة برمجة كائنية التوجه

كل شيء تقريبًا في بايثون هو كائن، له خصائصه وأساليبه.
تشبه الكلاس منشئ الكائن، أو "مخطط" لإنشاء الكائنات.

كيف (class) إنشاء الفئات

class: لإنشاء فصل دراسي، استخدم الكلمة المحجوزة

مثال ↘ ↘

x مع فئة تسمى **Hosini_Class** قم بإنشاء كلاس باسم

```
class Hosini_Class:
```

```
x = 5
```

إنشاء كائن

لإنشاء كائنات **Hosini_Class** نستطيع الآن استخدام الكلاس المسماة

مثال ↘ ↘

x: ثم اطبع قيمة **HAbibo** أنشئ كائنًا باسم

```
HAbibo = Hosini_Class()
```

```
print(HAbibo.x)
```

() __init__ الدالة

الأمثلة المذكورة أعلاه هي كلاسات وكائنات في أبسط أشكالها، وليست مفيدة حقًا في تطبيقات الحياة الواقعية.

. () __init__ لفهم معنى الكلاسات علينا أن نفهم دالة

والتي يتم تنفيذها دائمًا، () __init__ تحتوي كافة الكلاسات على دالة تسمى . عند بدء الفصل

لتعيين قيم لخصائص الكائن، أو العمليات الأخرى () __init__ استخدم الدالة :الضرورة التي يجب القيام بها عند إنشاء الكائن

مثال ↘ ↘

لتعيين قيم () __init__ أنشئ كلاس لوصف "المستخدم"، واستخدم الدالة :للإسم والعمر

```
class Hosini_Class2:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

```
HAbibo = Hosini_Class2("Abo Habib", 36)
```

```
print(HAbibo.name)  
print(HAbibo.age)
```

تلقائيًا في كل مرة يتم فيها استخدام `()__init__` ملاحظة: يتم استدعاء الدالة الكلاس لإنشاء كائن جديد.

`()__str__` الدالة

فيما يجب إرجاعه عندما يتم تمثيل كائن الكلاس `()__str__` تتحكم الدالة كنص.

فسيتم إرجاع تمثيل النصوص للكائن، `()__str__` إذا لم يتم تعيين الدالة

مثال ↘ ↘

`()__str__`: تمثيل النصوص لكائن بدون الدالة

```
class Hosini_Class2:
```

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
HAbibo = Hosini_Class2("Abo Habib", 36)
```

```
print(HAbibo)
```

مثال ↘ ↘

`__str__()`: تمثيل النصوص لكائن باستخدام الدالة

```
class Hosini_Class2:
```

```
def __init__(self, name, age):
```

```
    self.name = name
```

```
    self.age = age
```

```
def __str__(self):
```

```
    return f"{self.name}({self.age})"
```

```
HAbibo = Hosini_Class2("Abo Habib", 17)
```

```
print(HAbibo)
```

دوال الكائن

يمكن أن تحتوي الكائنات أيضًا على دوال. الأساليب والـ `__str__` في الكائنات هي دوال تنتمي إلى الكائن.

دعونا ننشئ دالة في كلاس المستخدم

مثال ↘ ↘

`HAbibo`: أدخل دالة تطبع تحية وقم بتنفيذها على الكائن

```
class Hosini_Class2:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def HoSini(self):
        print("Hello my name is " + self.name)

HAbibo = Hosini_Class2("Abo Habib", 364)
HAbibo.HoSini()
```

هي مرجع للمثيل الحالي للكلاس ، ويتم استخدامها **self** ملاحظة: الكلمة . للوصول إلى المتغيرات التي تنتمي إلى الكلاس

الكلمة Self

هي كلمة تشير الى الكلاس من داخله بدلا من كتابة اسمة مرات **self** الكلمة عديدة او مرجع إلى المثيل الحالي للكلاس ، ويتم استخدامها للوصول إلى المتغيرات التي تنتمي إلى الكلاس .

تستطيع تسميتها كما تريد، ولكن يجب ، **self** ليس من الضروري أن يتم تسميتها : أن تكون الكلمة الأولى لأي دالة في الفصل

مثال ↘ ↘

self بدلاً من **Ahmed** و **Hosini_object** استخدم الكلمات

```
class Hosini_Class2:
    def __init__(Hosini_object, name, age):
        Hosini_object.name = name
        Hosini_object.age = age

def HoSini(Ahmed):
    print("Hello my name is " + Ahmed.name)

HAbibo = Hosini_Class2("Abo Habib", 36)
HAbibo.HoSini()
```

تعديل خصائص الكائن

تستطيع تعديل الخصائص على كائنات مثل هذا

مثال ↘ ↘

إلى 40 HAbibo تحديد سن

```
HAbibo.age = 40
```

حذف خصائص الكائن

: الكلمة المحجوزة **del** تستطيع حذف خصائص الكائنات باستخدام

مثال ↘ ↘

HAbibo احذف خاصية العمر من الكائن

del HAbibo.age

حذف الكائنات

: الكلمة المحجوزة **del** تستطيع حذف الكائنات باستخدام

مثال ↘ ↘

HAbibo حذف الكائن

del HAbibo

كلمة المرور اذا كان الكلاس فارغ

مثل الاجراءات في الجزء الاول لا يمكن أن تكون التعريفات لمحتوى **Class** الكلاس فارغة، ولكن إذا كان موجود تعريف لاي خصائص بدون محتوى لسبب العبارة لتجنب حدوث خطأ **pass** ما ، فقم بإدخال

مثال ↘ ↘

```
class Hosini_Class2:  
    pass
```

الوراثة فى بايثون ✓

سهولة وراثة بايثون

يتيح لنا الوراثة تحديد كلاس ترث جميع الأساليب والفنكشن والخصائص من كلاس أخرى.

. الكلاس الأصلية هي الكلاس الموروثة منها، وتسمى أيضاً الكلاس المحجوزة

الكلاس التابعة هي الكلاس التي ترث من كلاس أخرى، وتسمى أيضاً الكلاس المشتقة.

إنشاء كلاس أساسى

يمكن لأي كلاس أن تكون كلاس أصل، وبالتالي فإن بناء الجملة هو نفس إنشاء أي كلاس أخرى:

مثال ↘ ↘

```
lastname وخصائص firstname باسم Hosini_Class2 أنشئ كلاس  
printname: ودالة
```

```
class Hosini_Class2:
    def __init__(self, hoshos, bibo):
        self.firstname = hoshos
        self.lastname = bibo

    def printname(self):
        print(self.firstname, self.lastname)

#Use the Hosini_Class2 class to create an object, and
#then execute the printname method:

x = Hosini_Class2("Abo Habib", "Al Hosiny * _*")
x.printname()
```

إنشاء كلاس فرعي

لإنشاء كلاس ترث الدالة من كلاس أخرى، أرسل الكلاس الأصل كمعلمة عند إنشاء الكلاس الفرعية:

مثال ↘ ↘

والتي سوف ترث الخصائص والأساليب، **Hosini3** قم بإنشاء كلاس باسم الكلاس **Hosini_Class2** والفنكشن من

```
class Hosini3(Hosini_Class2):
    pass
```

الكلمة المحجوزة عندما لا تريد إضافة أي خصائص أو **pass** ملاحظة: استخدم أساليب أخرى إلى الفصل الدراسي.

الآن تتمتع كلاس الطالب بنفس خصائص وأساليب كلاس المستخدم.

مثال ↘ ↘

الدالة **printname** الكلاس لإنشاء كائن، ثم قم بتنفيذ **Hosini3** استخدم

```
x = Hosini3("Mike", "Olsen")
```

```
x.printname()
```

✓ جلب الكائن كنسخة

➤ **__iter__()** و **__next__()**.

هو كائن عام على الكلاسات تستطيع استخدامة يحتوي على عدد لا يحصى من القيم.

هو كائن يمكن النسخ عليه، مما يعني أنه تستطيع اجتياز جميع القيم

من الناحية الفنية، في بايثون، هو كائن ينفذ بروتوكول ، والذي يتكون من **__iter__()** و **__next__()**. الأساليب والفنكشن

كيف جلب نسخة من الكلاس

القوائم والمصفوفة والقواميس والمجموعات كلها كائنات قابلة للنسخ. إنها حاويات قابلة للنسخ تستطيع نسخة منها دالة تُستخدم للحصول على نسخة (**iter()**) كل هذه الكائنات لها

مثال ↘ ↘

قم بإرجاع نسخة من صف، واطبع كل قيمة

```
Hosini_tuple = ("Abo", "Habib", "Al Hosiny")  
Hosini5 = iter(Hosini_tuple)
```

```
print(next(Hosini5))  
print(next(Hosini5))  
print(next(Hosini5))
```

حتى النصوص هي كائنات قابلة للنسخ، وتستطيع إرجاع نسخة

مثال ↘ ↘

النصوص هي أيضًا كائنات قابلة للنسخ، تحتوي على مصفوفة من الأحرف

```
Hosini_str = "Habib"  
Hosini5 = iter(Hosini_str)
```

```
print(next(Hosini5))
```

```
print(next(Hosini5))
print(next(Hosini5))
print(next(Hosini5))
print(next(Hosini5))
print(next(Hosini5))
```

النسخ من خلال الحلقات

حلقة للنسخ عبر كائن قابل للنسخ **for** نستطيع أيضاً استخدام

مثال ↘ ↘

Tuple تكرار قيم:

```
Hosini_tuple = ("Abo", "Habib", "Al Hosiny")
```

```
for x in Hosini_tuple:
    print(x)
```

مثال ↘ ↘

: تكرار أحرف النصوص

```
Hosini_str = "Habib"
```

```
for x in Hosini_str:  
    print(x)
```

بالفعل بإنشاء كائن نسخة وتنفيذ الدالة التالية لكل حلقة **for** تقوم الحلقة

إنشاء نسخة

__iter__() لإنشاء كائن/كلاس كنسخة، يجب عليك تنفيذ الأساليب والفنكشن **__next__()** والكائن **__init__()** والـ

كما تعلمت في فصل **الكلاسات**، فإن جميع الكلاسات لديها دالة والتي تسمح لك بإجراء بعض التهيئة عند إنشاء الكائن، **__init__()** تسمى بشكل مشابه، تستطيع إجراء عمليات (التهيئة وما إلى **__iter__()** تعمل الدالة ذلك)، ولكن يجب دائمًا إرجاع كائن النسخ نفسه

الدالة أيضًا بإجراء العمليات، ويجب عليك إرجاع **__next__()** تسمح لك هذه العنصر التالي في المصفوفات

مثال ↘ ↘

أنشئ نسخة يُرجع أرقامًا، بدءًا من الرقم 1، وسيزيد كل تسلسل بمقدار واحد (يُرجع 1,2,3,4,5 وما إلى ذلك):

```
class Hosini_Numbers:  
    def __iter__(self):  
        self.a = 1
```

```
return self
```

```
def __next__(self):  
    x = self.a  
    self.a += 1  
    return x
```

```
Hosini_Class = Hosini_Numbers()  
Hosini5 = iter(Hosini_Class)
```

```
print(next(Hosini5))  
print(next(Hosini5))  
print(next(Hosini5))  
print(next(Hosini5))  
print(next(Hosini5))
```

StopIteration

سيستمر مثال ↘ ↘ أعلاه إلى الأبد إذا كان موجود ما يكفي من عبارات **for** أو إذا تم استخدامه في حلقة، **next()**.

العبارة **StopIteration** لمنع استمرار النسخ إلى الأبد، نستطيع استخدام

الدالة، نستطيع إضافة شرط إنهاء لإثارة خطأ إذا تم النسخ **__next__()** في هذه:
لعدد محدد من المرات

مثال ↘ ↘

توقف بعد 20 تكرارًا

```
class Hosini_Numbers:
```

```
    def __iter__(self):
```

```
        self.a = 1
```

```
        return self
```

```
    def __next__(self):
```

```
        if self.a <= 20:
```

```
            x = self.a
```

```
            self.a += 1
```

```
            return x
```

```
        else:
```

```
            raise StopIteration
```

```
Hosini_Class = Hosini_Numbers()
```

```
Hosini5 = iter(Hosini_Class)
```

```
for x in Hosini5:
```

```
    print(x)
```

✓ نطاق الوصول فى بايثون

المتغير متاح فقط من داخل المنطقة التي تم إنشاؤه فيها. وهذا ما يسمى النطاق.

النطاق المحلي

ينتمي المتغير الذي تم إنشاؤه داخل دالة إلى النطاق المحلي لتلك الدالة، ولا يمكن استخدامه إلا داخل تلك الدالة.

مثال ↘ ↘

:المتغير الذي تم إنشاؤه داخل دالة متاح داخل تلك الدالة

```
def HoSini():
```

```
    x = 300
```

```
    print(x)
```

```
HoSini()
```

نطاق الدالة داخل دالة

غير متاح خارج الدالة، x كما هو موضح في مثال ↘ ↘ أعلاه، فإن المتغير: ولكنه متاح لأي دالة داخل الدالة

مثال ↘ ↘

:يمكن الوصول إلى المتغير المحلي من دالة داخل الدالة

```
def HoSini():  
    x = 300  
    def myinnerfunc():  
        print(x)  
    myinnerfunc()
```

HoSini()

النطاق العام

المتغير الذي تم إنشاؤه في النصوص الرئيسي لكود بايثون هو متغير عام. وينتمي إلى النطاق العام. المتغيرات العامة متاحة من داخل أي نطاق، عام ومحلي.

مثال ↘ ↘

:المتغير الذي تم إنشاؤه خارج الدالة يكون عامًا ويمكن لأي شخص استخدامه

```
x = 300
```

```
def HoSini():  
    print(x)
```

HoSini()

```
print(x)
```

نعيد احذر تسمية المتغيرات العشوائية

إذا كنت تعمل بنفس اسم المتغير داخل الدالة وخارجها، فستعاملهما بايثون كمتغيرين منفصلين، أحدهما متاح في النطاق العام (خارج الدالة) والآخر متاح في النطاق المحلي (داخل الدالة):

مثال ↘ ↘

x: ثم سيقوم الكود بطباعة العمومي **x**، ستقوم الدالة بطباعة المحلي

```
x = 300
```

```
def HoSini():
```

```
    x = 200
```

```
    print(x)
```

```
HoSini()
```

```
print(x)
```

الكلمة الرئيسية العامة

إذا كنت بحاجة إلى إنشاء متغير عام، ولكنك عالق في النطاق المحلي، المحجوزة **global** فتستطيع استخدام الكلمة

المحجوزة تجعل المتغير عامًا **global** الكلمة.

مثال ↘ ↘

الكلمة المحجوزة ، فإن المتغير ينتمي إلى النطاق **global** إذا كنت تستخدم العام:

```
def HoSini():
```

```
    global x
```

```
    x = 300
```

```
HoSini()
```

```
print(x)
```

الكلمة المحجوزة إذا كنت تريد إجراء تغيير على متغير **global** استخدم أيضًا عام داخل دالة.

مثال ↘ ↘

لتغيير قيمة متغير عام داخل دالة، قم بالإشارة إلى المتغير باستخدام : المحجوزة **global** الكلمة :

```
x = 300
```

```
def HoSini():
```

```
    global x
```

| `x = 200`

| `HoSini()`

| `print(x)`

✓ وحدات بايثون او الموديول

ما هي الوحدة؟

ضع في اعتبارك أن الوحدة النمطية موديول هي نفس مكتبة الاكواد فى لغة بايثون التى تقوم بستدعائها للحصول على الاجراءات بداخلها .
ملف يحتوي على مجموعة من الدوال التي تريد تضمينها في التطبيق.

إنشاء وحدة

لإنشاء وحدة نمطية، ما عليك سوى حفظ الكود الذي تريده في ملف بامتداد ملف `.py`. بايثون

مثال ↘ ↘

`Habib_module.py` احفظ هذا الكود في ملف اسمه

```
def Hosini7(name):  
    print("Hello, " + name)
```

أستخدم الوحدة النمطية موديول

import: نستطيع الآن استخدام الوحدة التي أنشأناها للتو، باستخدام العبارة

مثال ↘ ↘

نستخدم دالة الترحيب، `Habib_module` قم باستيراد الوحدة المسماة

```
| import Habib_module
```

```
| Habib_module.Hosini7("Al Hosiny *_*")
```

ملاحظة: عند استخدام دالة من وحدة نمطية، استخدم الصيغة: `Module_name.function_name`.

المتغيرات في الوحدة النمطية موديول

يمكن أن تحتوي الوحدة على دوال ، كما هو موضح بالفعل، ولكن أيضًا: متغيرات من جميع الأنواع (المصفوفات والقواميس والكائنات وما إلى ذلك)

مثال ↘ ↘

`Habib_module.py` احفظ هذا الكود في الملف

```
Hosini_Class21 = {  
    "name": "Abo Habib",  
    "age": 36,  
    "Books": " C# , C++ , Node.js , python , F# , VBA , VBS"  
}
```

مثال ↘ ↘

وقم بالوصول إلى قاموس `Habib_module` قم باستيراد الوحدة المسماة `Hosini_Class21`:

```
import Habib_module
```

```
a = Habib_module.Hosini_Class21["age"]  
print(a)
```

تسمية الوحدة النمطية موديول

تستطيع تسمية ملف الوحدة النمطية موديول كما تريد، ولكن يجب أن يكون له `.py` امتداد الملف

إعادة تسمية الوحدة النمطية موديول

`as` تستطيع إنشاء اسم مختصر عند استيراد وحدة نمطية، باستخدام الكلمة المحجوزة:

مثال ↘ ↘

قم بإنشاء اسم مختصر لـ `Habib_module` call `mx`:

```
| import Habib_module as mx
```

```
| a = mx.Hosini_Class21["age"]
```

```
| print(a)
```

وحدات جاهزة مدمجة في اللغة

هناك العديد من الوحدات في لغة بايثون، والتي تستطيع استيرادها وقتما تشاء.

مثال ↘ ↘

:الوحدة `platform` استيراد واستخدام

```
| import platform
```

```
| x = platform.system()
```

```
| print(x)
```

dir() باستخدام الدالة

توجد دالة مضمنة لسرد كافة أسماء الدوال (أو أسماء المتغيرات) في الوحدة **dir()** النمطية موديول . الدالة

مثال ↘ ↘

قم بإدراج كافة الأسماء المحددة التي تنتمي إلى وحدة النظام الأساسي

```
| import platform
```

```
| x = dir(platform)
```

```
| print(x)
```

على كافة الوحدات، وكذلك تلك التي تقوم **dir()** ملاحظة: يمكن استخدام الدالة بإنشائها بنفسك.

استيراد من الوحدة النمطية موديول الى التطبيق

تستطيع اختيار استيراد أجزاء معينة فقط من الوحدة النمطية موديول ، الكلمة المحجوزة **from** باستخدام

مثال ↘ ↘

على دالة واحدة وقاموس واحد **Habib_module** تحتوي الوحدة المسماة

```
def Hosini7(name):  
    print("Hello, " + name)
```

```
Hosini9 = {  
    "name": "Abo Habib",  
    "age": 36,  
    "mail": "aabwhbyb31@gmail.com"  
}
```

مثال ↘ ↘

: فقط من الوحدة النمطية موديول **Hosini9** قم باستيراد قاموس

```
from Habib_module import Hosini9  
  
print (Hosini9["age"])
```

المحجوزة ، لا تستخدم اسم **from** ملاحظة: عند الاستيراد باستخدام الكلمة
الوحدة عند الإشارة إلى العناصر الموجودة في الوحدة. مثال

↘ ↘ **Hosini9["age"]**: لا **Habib_module.Hosini9["age"]**

الوقت والتاريخ

التاريخ في بايثون ليس نوع بيانات خاصًا به، لكن نستطيع استيراد وحدة للعمل مع التواريخ ككائنات تاريخ **datetime** اسمها

مثال ↘ ↘

قم باستيراد وحدة التاريخ والوقت وعرض التاريخ الحالي

```
import datetime
```

```
x = datetime.datetime.now()
```

```
print(x)
```

تاريخ الإخراج

عندما ننفذ الكود من مثال ↘ ↘ أعلاه ستكون ال

```
2023-11-19 16:36:05.920549
```

يحتوي التاريخ على السنة والشهر واليوم والساعة والدقيقة والثانية والميكروثانية على العديد من الدوال لإرجاع معلومات حول كائن **datetime** تحتوي الوحدة التاريخ.

فيما يلي بعض الأمثلة، وسوف تتعلم المزيد عنها لاحقًا في هذا الفصل

مثال ↘ ↘

إرجاع السنة واسم أيام الأسبوع

```
| import datetime
```

```
| x = datetime.datetime.now()
```

```
| print(x.year)
```

```
| print(x.strftime("%A"))
```

إنشاء كائنات التاريخ

كلاس (منشئ) **datetime()** لإنشاء تاريخ، نستطيع استخدام **datetime** الوحدة.

ثلاثة كلمات لإنشاء تاريخ: السنة والشهر واليوم **datetime()** يتطلب الفصل.

مثال ↘ ↘

إنشاء كائن تاريخ:

```
| import datetime
```

```
| x = datetime.datetime(2020, 5, 17)
```

| `print(x)`

ساعة، دقيقة،) أيضًا كلمات للوقت والمنطقة الزمنية `datetime()` يأخذ الفصل `None`)، لكنها اختيارية، ولها قيمة افتراضية `0`، `tzzone`، ثانية، ميكروثانية (للمنطقة الزمنية).

`strftime()` دالة

على دالة لتنسيق كائنات التاريخ في نصوص قابلة `datetime` يحتوي الكائن للقراءة.

لتحديد تنسيق `format` وتأخذ معلمة واحدة، `strftime()` يتم استدعاء الدالة: النصوص التي يتم إرجاعها

مثال ↘ ↘

عرض اسم الشهر

| `import datetime`

| `x = datetime.datetime(2018, 6, 1)`

| `print(x.strftime("%B"))`

مرجع لجميع رموز التنسيق المقبول في اللفة

Directive	وصف	مثال
%a	أيام الأسبوع، نسخة قصيرة	Wed
%A	أيام الأسبوع، النسخة الكاملة	Wednesday
%w	أيام الأسبوع كرقم 0-6، 0 هو الأحد	3
%d	يوم من الشهر 01-31	31
%b	اسم الشهر، نسخة قصيرة	Dec
%B	اسم الشهر، النسخة الكاملة	December
%m	الشهر كرقم 01-12	12
%y	سنة، نسخة قصيرة، بدون قرن	18
%Y	السنة، النسخة الكاملة	2018
%H	الساعة 00-23	17
%I	الساعة 00-12	5
%p	صباحا مساءا	PM
%M	الدقيقة 00-59	41
%S	الثانية 00-59	8
%f	ميكروثانية 000000-999999	548513
%z	إزاحة التوقيت العام المنسق (UTC).	100
%Z	وحدة زمنية	CST
%j	رقم اليوم من السنة 001-366	365
%U	رقم الأسبوع من السنة، الأحد هو أول يوم في الأسبوع، 00-53	52
%W	رقم الأسبوع من السنة، يوم الاثنين هو أول يوم في الأسبوع، 00-53	52
%c	النسخة المحلية من التاريخ والوقت	Mon Dec 31 17:41:00 2018
%C	قرن	20
%x	النسخة المحلية من التاريخ	43465
%X	النسخة المحلية من الوقت	0.74
%%	شخصية	%
%G	ايزو 8601 سنة	2018
%u	أيام الأسبوع (1-7) ISO 8601	1

تحتوي لغة بايثون على مجموعة من الدوال الرياضية ، بما في ذلك وحدة رياضية واسعة النطاق، والتي تتيح لك أداء المهام الرياضية على الأرقام.

دوال الرياضيات

للعثور على أدنى أو أعلى قيمة **min()** and **max()** يمكن استخدام الدالات في كائن قابل للنسخ:

مثال ↘ ↘

```
x = min(5, 10, 25)
```

```
y = max(5, 10, 25)
```

```
print(x)
```

```
print(y)
```

:القيمة المطلقة (الإيجابية) للرقم المحدد **abs()** ترجع الدالة

مثال ↘ ↘

```
| x = abs(-7.25)
```

```
| print(x)
```

`pow(x, y)` (x^y) إلى قوة x تقوم الدالة بإرجاع قيمة

مثال ↘ ↘

أعد قيمة 4 للقوة 3 (مثل $4 * 4 * 4$):

```
| x = pow(4, 3)
```

```
| print(x)
```

وحدة الرياضيات

والتي تعمل على `math`، تحتوي بايثون أيضًا على وحدة مدمجة تسمى توسيع قائمة الدوال الرياضية.

`math`: لاستخدامها، يجب عليك استيراد الوحدة

```
| import math
```

تستطيع البدء في استخدام أساليب **math**، عندما تقوم باستيراد الوحدة ووثابت الوحدة

على سبيل **مثال** ↘ ↘ بإرجاع الجذر التربيعي لرقم **math.sqrt()** تقوم الدالة

مثال ↘ ↘

```
| import math
```

```
| x = math.sqrt(64)
```

```
| print(x)
```

بتقريب الرقم لأعلى إلى أقرب عدد صحيح، وتقوم **math.ceil()** تقوم الدالة بتقريب الرقم للأسفل إلى أقرب عدد صحيح له، وإرجاع **math.floor()** الدالة ال:

مثال ↘ ↘

```
| import math
```

```
| x = math.ceil(1.4)
```

```
| y = math.floor(1.4)
```

```
| print(x) # returns 2
```

```
| print(y) # returns 1
```

الثابت **math.pi**، **PI (3.14...)** يُرجع قيمة

مثال ↘ ↘

```
| import math
```

```
| x = math.pi
```

```
| print(x)
```

هو بناء جملة لتخزين الاكواد وتبادلها JSON.

JavaScript هو نصوص مكتوب باستخدام تدوين كائن JSON.

في بايثون JSON

والتي يمكن استخدامها, json تحتوي لغة بايثون على حزمة مدمجة تسمى JSON. للعمل مع اكواد

مثال ↘ ↘

json: استيراد وحدة

```
| import json
```

تحويل من جسون إلى قاموس بايثون والعكس

`json.loads()` فتستطيع تحليلها باستخدام `JSON` إذا كان موجود نصوص الدالة.

• وستكون ال قاموس بايثون

مثال ↘ ↘

إلى بايثون `JSON` التحويل من

```
| import json
```

```
| # some JSON:
```

```
| x = '{"name":"Abo Habib", "age":300, "city":"New  
| York"}'
```

```
| # parse x:
```

```
| y = json.loads(x)
```

```
| # the result is a python dictionary:
```

```
| print(y["age"])
```

JSON التحويل من بايثون إلى

باستخدام JSON إذا كان موجود كائن بايثون، فتستطيع تحويله إلى نصوص الدالة `json.dumps()`.

مثال ↘ ↘

JSON التحويل من بايثون إلى

```
| import json
```

```
| # a object (dict):
```

```
| x = {  
|     "name": "Abo Habib",  
|     "age": 900,  
|     "city": "New York"  
| }
```

```
| # convert into JSON:
```

```
| y = json.dumps(x)
```

```
| # the result is a JSON string:
```

```
| print(y)
```

مثال ↘ ↘

وطباعة القيم، JSON تحويل كائنات بايثون إلى نصوص

```
import json
```

```
print(json.dumps({"name": "Abo Habib", "age": 20}))  
print(json.dumps(["Abo", "Habibs"]))  
print(json.dumps(("Abo", "Habibs")))  
print(json.dumps("Hello Abo Habib "))  
print(json.dumps(42))  
print(json.dumps(31.76))  
print(json.dumps(True))  
print(json.dumps(False))  
print(json.dumps(None))
```

يتم تحويل كائنات بايثون إلى مكافئ JSON، عند التحويل من بايثون إلى JSON (JavaScript):

بايثون	جسون
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true

False	false
None	null

مثال ↘ ↘

تحويل كائن بايثون الذي يحتوي على جميع أنواع الاجراءات الصحيحة:

```
import json
```

```
x = {  
    "name": "Abo Habib",  
    "age": 40,  
    "UI": True,  
    "UN": False,  
    "children":  
    ("Habib", "Ahmed", "Omer", "Mohamed", "Mahmoud", "Yas  
een"),  
    "lang": None,  
    "cars": [  
        {"model": "BMW 230", "mpg": 27.5},  
        {"model": "Ford Edge", "mpg": 24.1}  
    ]  
}
```

```
print(json.dumps(x))
```

أهمية التنسيق في جسون

جسون ليس من السهل قراءتها، بدون JSON، يطبع مثال ↘ ↘ أعلاه نصوص مسافات بادئة وفواصل أسطر.

على كلمات لتسهيل قراءة ال `json.dumps()` تحتوي الدالة

مثال ↘ ↘

الكلمة لتحديد عدد المسافات البادئة `indent` استخدم

| `json.dumps(x, indent=4)`

تستطيع أيضًا تحديد الفواصل، القيمة الافتراضية هي `":", "`، مما يعني استخدام فاصلة ومسافة لفصل كل كائن، ونقطتين ومسافة لفصل المفاتيح عن القيم:

مثال ↘ ↘

تستطيع تغيير الفاصل الافتراضي

الكلمة لتغيير الفاصل الافتراضي `separators` استخدم

| `json.dumps(x, indent=4, separators=(".", " = "))`

json.dumps()

:على كلمات لترتيب المفاتيح في ال `json.dumps()` تحتوي الدالة

مثال ↘ ↘

:الكلمة لتحديد ما إذا كان يجب فرز أم لا `sort_keys` استخدم

| `json.dumps(x, indent=4, sort_keys=True)`

استخدام RegEx

أو التعبير العادي، عبارة عن نصوص من الأحرف التي تشكل نمط `RegEx`.
بحث.

للتحقق مما إذا كانت النصوص تحتوي على نمط `RegEx` يمكن استخدام
البحث المحدد.

RegEx وحدة

والتي يمكن استخدامها `re` تحتوي لغة بايثون على حزمة مدمجة تسمى
للعمل مع التعبيرات العادية.

:الوحدة `re` استيراد

`import re`

في بايثون RegEx

تستطيع البدء في استخدام التعبيرات العادية، `re` عندما تقوم باستيراد الوحدة

مثال ↘ ↘

"Al-Badrasheen" وتنتهي بـ "The" ابحث في النصوص لمعرفة ما إذا كانت تبدأ بـ "Al-Badrasheen":

`import re`

```
txt = "The rain in Al-Badrasheen"
```

```
x = re.search("^The.*Al-Badrasheen$", txt)
```

RegEx دوال

مجموعة من الدوال التي تسمح لنا بالبحث عن نصوص `re` توفر الوحدة مطابقة:

Function	وصف
<code>findall</code>	إرجاع قائمة تحتوي على كافة التطابقات
<code>search</code>	لعرض كائن مطابقة إذا كان هناك تطابق في أي مكان في النصوص
<code>split</code>	إرجاع قائمة حيث تم تقسيم النصوص في كل عملية

الأحرف الأولية

الأحرف الأولية هي أحرف ذات معنى خاص:

Character	وصف	مثال
[]	مجموعة من المستخدميات	"[a-m]"
\	يشير إلى تسلسل خاص (يمكن استخدامه أيضًا للهروب من الأحرف)	"\d"
.	أي حرف (باستثناء حرف السطر الجديد)	"he..o"
^	ابدا ب	"^hello"
\$	ينتهي ب	"planet\$"
*	صفر أو أكثر من الأحداث	"he.*o"
+	حدث واحد أو أكثر	"he.+o"
?	صفر أو حدث واحد	"he.?o"
{ }	بالضبط العدد المحدد من مرات الظهور	"he.{2}o"
	إما او	"falls stays"
()	التقاط والمجموعة	

تنسيقات خاصة

المصفوفات الخاص هو الذي \ يتبعه أحد الأحرف الموجودة في القائمة أدناه، وله معنى خاص:

Character	وصف	مثال
\A	إرجاع تطابق إذا كانت الأحرف المحددة في بداية النصوص	"\AThe"
\b	إرجاع تطابق حيث تكون الأحرف المحددة في البداية (يتأكد "r" في نهايتها من أن النصوص يتم التعامل معها على أنها	"r\bain" "ain\b"

	("سلسلة خام	
\B	إرجاع تطابق حيث توجد الأحرف المحددة، ولكن يتأكد (ليس في بداية (أو في النهاية) للكلمة في البداية من أن النصوص يتم "r" الحرف ("التعامل معها على أنها "سلسلة خام	r"\Bain" r"ain\B"
\d	إرجاع تطابق حيث تحتوي النصوص على أرقام (أرقام من 0 إلى 9)	"\d"
\D	تقوم بإرجاع تطابق حيث لا تحتوي النصوص على أرقام	"\D"
\s	إرجاع تطابق حيث تحتوي النصوص على حرف مسافة بيضاء	"\s"
\S	تقوم بإرجاع تطابق حيث لا تحتوي النصوص على مسافة بيضاء	"\S"
\w	إرجاع تطابق حيث تحتوي النصوص على أي "w" والأرقام من 0، Z، إلى a الأحرف من) أحرف كلمة (إلى 9، وحرف _ الشرطة السفلية	"\w"
\W	تقوم بإرجاع تطابق حيث لا تحتوي النصوص على أي أحرف كلمة	"\W"
\Z	إرجاع تطابق إذا كانت الأحرف المحددة في نهاية النصوص	"Al-Badrashe en\Z"

مجموعات

المجموعة عبارة عن مجموعة من الأحرف داخل زوج من الأقواس المربعة [] ذات معنى خاص:

Set	وصف
[arn]	(n أو r أو a) لعرض تطابق حيث يوجد أحد الأحرف المحددة.
[a-n]	a و n لعرض تطابق لأي حرف صغير، أبعدياً بين

[^arn]	ا و r و n لعرض تطابق لأي حرف باستثناء
[0123]	لعرض تطابق حيث يوجد أي من الأرقام المحددة (0، 1، 2، أو 3)
[0-9]	لعرض تطابق لأي رقم بين 0 و 9
[0-5][0-9]	لعرض تطابق لأي أرقام مكونة من رقمين من 00 و 59
[a-zA-Z]	أو أحرف صغيرة أو z و a لعرض تطابق لأي حرف أبجديًا بين أحرف كبيرة
[+]	في المجموعات، +، *، ..، ، ()، \$، {} ليس لها أي معنى خاص، لذا [+] يعني: عرض تطابق لأي حرف + في النصوص

() findall الدالة

بإرجاع قائمة تحتوي على كافة التطابقات **findall()** تقوم الدالة

مثال ↘ ↘

طباعة قائمة بجميع التطابقات

```
import re

txt = "The rain in Al-Badrasheen"
x = re.findall("ai", txt)
print(x)
```

تحتوي القائمة على التطابقات بالترتيب الذي تم العثور عليه به

إذا لم يتم العثور على أي تطابقات، فسيتم إرجاع قائمة فارغة

مثال ↘ ↘

قم بإرجاع قائمة فارغة إذا لم يتم العثور على أي تطابق

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"  
| x = re.findall("Portugal", txt)  
| print(x)
```

search دالة البحث

عن نصوص مطابقة، وتقوم بإرجاع **كائن** `search()` تبحث الدالة **مطابقة** إذا كان هناك تطابق.

إذا كان هناك أكثر من تطابق، فسيتم إرجاع التواجد الأول فقط للمطابقة

مثال ↘ ↘

: ابحث عن أول حرف مسافة في النصوص

```
| import re
```

```
txt = "The rain in Al-Badrasheen"
```

```
x = re.search("\s", txt)
```

```
print("The first white-space character is located in  
position:", x.start())
```

إذا لم يتم العثور على أي تطابقات، يتم إرجاع القيمة **None**:

مثال ↘ ↘

قم بإجراء بحث لا يُرجع أي تطابق

```
import re
```

```
txt = "The rain in Al-Badrasheen"
```

```
x = re.search("Portugal", txt)
```

```
print(x)
```

split دالة التقسيم

إرجاع قائمة حيث تم تقسيم النصوص في كل عملية **split()** تقوم الدالة

مثال ↘ ↘

:انقسم عند كل حرف مسافة

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"
```

```
| x = re.split("\s", txt)
```

```
| print(x)
```

:الكلمة **maxsplit** تستطيع التحكم في عدد التكرارات عن طريق تحديد

مثال ↘ ↘

:قم بتقسيم النصوص فقط عند التواجد الأول

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"
```

```
| x = re.split("\s", txt, 1)
```

```
| print(x)
```


دالة sub()

المطابقات بالنص الذي تختاره **sub()** تستبدل الدالة:

مثال ↘ ↘

استبدل كل حرف مسافة بالرقم 9

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"  
| x = re.sub("\s", "9", txt)  
| print(x)
```

الكلمة **count** تستطيع التحكم في عدد البدائل عن طريق تحديد

مثال ↘ ↘

استبدل أول حدثين:

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"  
| x = re.sub("\s", "9", txt, 2)  
| print(x)
```

طرق البحث

كائن المطابقة هو كائن يحتوي على معلومات حول البحث وال

إذا لم يكن هناك تطابق، فسيتم إرجاع القيمة بدلاً من كائن **None**: ملاحظة المطابقة.

مثال ↘ ↘

قم بإجراء بحث سيُرجع كائن مطابقة

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"  
| x = re.search("ai", txt)  
| print(x) #this will print an object
```

يحتوي كائن المطابقة على خصائص ودوال تستخدم لاسترجاع معلومات حول البحث، وال:

- تقوم بإرجاع صف يحتوي على موضعي البداية والنهاية للعملية **span()**.
- تقوم بإرجاع النصوص التي تم تمريرها إلى الدالة **string**.
- وإرجاع جزء النصوص الذي كان هناك تطابق فيه **group()**.

مثال ↘ ↘

اطبع الموضع (موضع البداية والنهاية) لتكرار المطابقة الأول.

"S": يبحث التعبير العادي عن أي كلمات تبدأ بالحرف الكبير

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"
```

```
| x = re.search(r"\bS\w+", txt)
```

```
| print(x.span())
```

مثال ↘ ↘

اطبع النصوص التي تم تمريرها إلى الدالة:

```
| import re
```

```
| txt = "The rain in Al-Badrasheen"
```

```
| x = re.search(r"\bS\w+", txt)
```

```
| print(x.string)
```

مثال ↘ ↘

اطبع جزء النصوص الذي يوجد به تطابق

"S": يبحث التعبير العادي عن أي كلمات تبدأ بالحرف الكبير

```
import re
```

```
txt = "The rain in Al-Badrasheen"
```

```
x = re.search(r"\bS\w+", txt)
```

```
print(x.group())
```

إذا لم يكن هناك تطابق، فسيتم إرجاع القيمة بدلاً من كائن **None**: ملاحظة المطابقة.

PIP استخدام

PIP ما هو؟

هو مدير حزم لحزم بايثون، أو الوحدات النمطية إذا أردت PIP

PIP ملاحظة: إذا كان موجود إصدار بايثون 3.4 أو أحدث، فسيتم تضمين افتراضياً.

ما هي الحزمة؟

تحتوي الحزمة على كافة الملفات التي تحتاجها للوحدة النمطية. الوحدات هي مكتبات أكواد بايثون التي تستطيع تضمينها في مشروعك.

PIP تحقق من تثبيت

انتقل عبر سطر الأوامر إلى موقع دليل البرنامج النصي لـ بايثون، واكتب ما يلي:

مثال ↘ ↘

PIP: التحقق من إصدار

```
C:\Users\Your Name\AppData\Local\Programs\python\-32\Scripts>pip --version
```

تثبيت نسخة

مثبتاً، فتستطيع تنزيله وتثبيته من هذه **PIP** إذا لم يكن موجود الصفحة: <https://pypi.org/project/pip/>

تنزيل الحزمة سهل جدا

تنزيل الحزمة التي تريدها PIP افتح واجهة سطر الأوامر واطلب من

انتقل عبر سطر الأوامر إلى موقع دليل البرنامج النصي لـ بايثون، واكتب ما يلي:

مثال ↘ ↘

"camelcase" قم بتنزيل الحزمة المسماة

```
C:\Users\Your Name\AppData\Local\Programs\python\32\Scripts>pip install camelcase
```

! لقد قمت الآن بتنزيل وتثبيت الحزمة الأولى

باستخدام الحزمة

بمجرد تثبيت الحزمة، فهي جاهزة للاستخدام

إلى مشروعك "camelcase" قم باستيراد حزمة

مثال ↘ ↘

"camelcase" استيراد واستخدام

```
| import camelcase
```

```
| c = camelcase.CamelCase()
```

```
txt = "Abo Habib Al-Hosiby *_* "
```

```
print(c.hump(txt))
```

البحث عن الحزم

ابحث عن المزيد من الحزم على <https://pypi.org/>.

إزالة الحزمة

الأمور لإزالة الحزمة **uninstall** استخدم

مثال ↙ ↘

قم بإلغاء تثبيت الحزمة المسماة "camelcase":

```
C:\Users\Your Name\AppData\Local\Programs\python\python36-32\Scripts>pip uninstall camelcase
```

تأكيد رغبتك في إزالة حزمة الجمل PIP سيطلب منك مدير الحزم

```
Uninstalling camelcase-02.1:
```

```
Would remove:
```

```
c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camecase-0.2-py3.6.egg-info
```

```
c:\users\Your Name\appdata\local\programs\python\
python36-32\lib\site-packages\camelcase\*
Proceed (y/n)?
```

وستتم إزالة الحزمة **y** اضغط

قائمة الحزم

:الأمر لسرد جميع الحزم المثبتة على نظامك **list** استخدم

مثال ↘ ↘

:قائمة الحزم المثبتة

```
C:\Users\Your Name\AppData\Local\Programs\python\
python36-32\Scripts>pip list
```

:

Package	Version

camelcase	0.2
mysql-connector	2.1.6
pip	18.1
pymongo	3.6.1
setuptools	39.0.1

✓ معالجة الاخطا فى بايثون

اختبار كتلة من الاكواد بحثاً عن الأخطاء **try** تتيح لك الكتلة معالجة الخطأ **except** تتيح لك الكتلة تنفيذ الاكواد في حالة عدم وجود خطأ **else** تتيح لك الكتلة تنفيذ الاكواد ، بغض النظر عن كتل المحاولة **finally** تتيح لك الكتلة والاستثناء

معالجة الاستثناء

عند حدوث خطأ، أو استثناء كما نسميه، ستتوقف لغة بايثون عادةً وتولد رسالة خطأ.

:العبارة **try** يمكن معالجة هذه الاستثناءات باستخدام

مثال ↘ ↘

:يتم تعريفه **x** سُنشئ الكتلة استثناءً، لأنه لم **try**

try:

print(x)

except:

print("An exception occurred")

نظراً لأن كتلة المحاولة تثير خطأً، فسيتم تنفيذ كتلة الاستثناء بدون كتلة المحاولة، سوف يتعطل البرنامج ويظهر خطأً

مثال ↘ ↘

لم يتم تعريفه **x** سيؤدي هذا الكلمة إلى ظهور خطأ، لأنه

```
| print(x)
```

العديد من الاستثناءات في الأخطاء

تستطيع تحديد أي عدد تريده من كتل الاستثناء، على سبيل **مثال** ↘ ↘ ، إذا كنت تريد تنفيذ كتلة خاصة من الاكواد لنوع خاص من الأخطاء:

مثال ↘ ↘

وأخرى للأخطاء **NameError** اطبع رسالة واحدة إذا ظهرت كتلة المحاولة الأخرى:

```
| try:  
|     print(x)  
| except NameError:  
|     print("Variable x is not defined")
```

except:

```
print("Something else went wrong")
```

نقد إجراء إذا فشل الكود

الكلمة المحجوزة لتحديد كتلة الاكواد التي سيتم تنفيذها **else** تستطيع استخدام في حالة عدم ظهور أي أخطاء

مثال ↘ ↘

لا تولد الكتلة أي خطأ **try**، ↘ ↘ في هذا مثال

try:

```
print("Hello Abo Habib ")
```

except:

```
print("Something went wrong")
```

else:

```
print("Nothing went wrong")
```

التنفيذ على أي حال

سيتم تنفيذ الكتلة، إذا تم تحديدها، بغض النظر عما إذا كانت كتلة **finally** . المحاولة تثير خطأ أم لا

مثال ↘ ↘

```
try:  
    print(x)  
except:  
    print("Something went wrong")  
finally:  
    print("The 'try except' is finished")
```

قد يكون هذا مفيداً لإغلاق الكائنات وتنظيف الموارد

مثال ↘ ↘

حاول فتح ملف غير قابل للكتابة والكتابة فيه

```
try:  
    f = open("Abo_Habib_file.txt")  
    try:  
        f.write("Lorum Ipsum")  
    except:  
        print("Something went wrong when writing to the  
file")  
    finally:  
        f.close()  
except:  
    print("Something went wrong when opening the file")
```

يمكن أن يستمر البرنامج دون ترك كائن الملف مفتوحًا.

رفع استثناء

باعتبارك مطور لغة بايثون، تستطيع اختيار طرح استثناء في حالة حدوث شرط ما.

الكلمة المحجوزة **raise** لطرح (أورفع) استثناء، استخدم

مثال ↘ ↘

أقل من 0 x يظهر خطأ ويوقف البرنامج إذا كانت

x = -1

if x < 0:

raise Exception("Sorry, no numbers below zero")

المحجوزة لرفع استثناء **raise** يتم استخدام الكلمة

تستطيع تحديد نوع الخطأ الذي يجب رفعه، والنص الذي سيتم طباعته للمستخدم.

مثال ↘ ↘

عددًا صحيحًا x إذا لم يكن **TypeError** قم برفع الخطأ

```
x = "Hello Abo Habib "
```

```
if not type(x) is int:
```

```
raise TypeError("Only integers are allowed")
```

إدخال المستخدم

تسمح بايثون بإدخال المستخدم

وهذا يعني أننا قادرون على مطالبة المستخدم بالإدخال

تختلف الدالة قليلاً في بايثون 3.6 عن بايثون 2.7

الدالة `input()` يستخدم بايثون 3.6 هذه

الدالة `raw_input()` يستخدم بايثون 2.7 هذه

ال مثال ↘ ↘ التالي يطلب اسم المستخدم، وعندما تقوم بإدخال اسم المستخدم، تتم طباعته على الشاشة:

بايثون 3.6

```
BiBo = input("Enter BiBo:")
```

```
print("BiBo is: " + BiBo)
```

2.7 بايثون

```
BiBo = raw_input("Enter BiBo:")  
print("BiBo is: " + BiBo)
```

وتستمر عندما **input()** تتوقف بايثون عن التنفيذ عندما يتعلق الأمر بالدالة. يقدم المستخدم بعض المدخلات.

✓ تنسيق النصوص في بايثون

للتأكد من أن النصوص ستظهر كما هو متوقع، نستطيع تنسيق الدالة **format()** باستخدام

format ()

. الدالة تنسيق الأجزاء المحددة من النصوص **format()** تتيح لك هذه

في بعض الأحيان توجد أجزاء من النصوص لا تستطيع التحكم فيها، ربما تأتي من برنامج التشغيل، أو من إدخال المستخدم؟

للتحكم في هذه القيم، أضف عناصر نائبة (أقواس متعرجة **{}**) في النص، وقم
الدالة **format()** بتشغيل القيم من خلال

مثال ↘ ↘

أضف عنصرًا نائبًا حيث تريد عرض السعر

```
price = 49
```

```
txt = "The price is {} dollars"
```

```
print(txt.format(price))
```

تستطيع إضافة كلمات داخل الأقواس المتعرجة لتحديد كيفية تحويل القيمة

مثال ↘ ↘

قم بتنسيق السعر ليتم عرضه كرقم مكون من رقمين عشريين

```
txt = "The price is {:.2f} dollars"
```

تحقق من جميع أنواع التنسيق في مرجع ابو حبيب على الموقع الرسمي
[String format\(\)](#) ١.

قيم كثيره

إذا كنت تريد استخدام المزيد من القيم، فما عليك سوى إضافة المزيد من القيم
إلى `format()`:

```
print(txt.format(price, BiBoo, count))
```


نأضف المزيد من العناصر النائبة

مثال ↘ ↘

EBoo = 3

BiBoo = 567

price = 49

Hosini8 = "I want {} pieces of item number {} for {:.2f} dollars."

print(Hosini8.format(EBoo, BiBoo, price))

رقمنة التنسيق

تستطيع استخدام رقمنة التنسيق (رقم داخل الأقواس المتعرجة {0}) للتأكد من وضع القيم في العناصر النائبة الصحيحة

مثال ↘ ↘

EBoo = 3

BiBoo = 567

price = 49

Hosini8 = "I want {0} pieces of item number {1} for {:.2f} dollars."

print(Hosini8.format(EBoo, BiBoo, price))

نوكذلك إذا أردت الرجوع إلى نفس القيمة أكثر من مرة فاستخدم رقم التنسيق قم

مثال ↘ ↘

```
age = 36
name = "Abo Habib"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

اسماء مختصرة للأرقام

تستطيع أيضًا استخدام أسماء مختصرة للأرقام عن طريق إدخال اسم داخل ولكن بعد ذلك يجب عليك استخدام الأسماء، الأقواس المتعرجة `{carname}` عند تمرير قيم الكلمات `txt.format(carname = "Ford")`:

مثال ↘ ↘

```
Hosini8 = "I have a {carname}, it is a {model}."
print(Hosini8.format(carname = "Ford", model
= "Mustang"))
```

التعامل مع الملفات في لغة بايثون ✓

تعد معالجة الملفات جزءًا مهمًا من أي تطبيق ويب.

في لغة بايثون العديد من الدوال لإنشاء الملفات وقراءتها وتحديثها وحذفها.

فتح ملف بايثون

open() الدالة المحجوزة للعمل مع الملفات في بايثون هي الدالة

. كلمتين؛ اسم الملف والوضع **open()** تأخذ الدالة

:هناك أربع دوال (أوضاع) مختلفة لفتح ملف

القراءة - القيمة الافتراضية. يفتح ملفاً للقراءة، خطأ إذا كان الملف غير **"r"** موجود

ادراج - يفتح ملفاً للادراج، وينشئ الملف إذا لم يكن موجوداً **"a"**

الكتابة - يفتح ملفاً للكتابة، وينشئ الملف إذا لم يكن موجوداً **"w"**

إنشاء - إنشاء الملف المحدد، وإرجاع خطأ في حالة وجود الملف **"x"**

بالإضافة إلى ذلك، تستطيع تحديد ما إذا كان يجب التعامل مع الملف كوضع ثنائي أو نصي

النصوص - القيمة الافتراضية. وضع النص **"t"**

ثنائي - الوضع الثنائي (مثل الصور) **"b"**

بناء الجملة

:لفتح ملف للقراءة يكفي تحديد اسم الملف

```
f = open("Abo_Habib_file.txt")
```

الكود أعلاه هو نفسه

```
f = open("Abo_Habib_file.txt", "rt")
```

هما، فلن تحتاج إلى "t" للقراءة والنص "r" نظراً لأن القيمتين الافتراضيتين تحديدهما.

ملاحظة: تأكد من وجود الملف، وإلا فسوف تحصل على خطأ

✓ فتح ملف بايثون

افتح ملفاً على الخادم

لنفترض أن لدينا الملف التالي، الموجود في نفس المجلد مثل بايثون

Abo_Habib_file.txt

Hello! Welcome to Abo_Habib_file.txt

This file is for testing purposes.

Good Luck!

دالة `open()` لفتح الملف، استخدم الدالة

دالة لقراءة `read()` بإرجاع كائن ملف، والذي يحتوي على `open()` تقوم الدالة
بمحتوى الملف:

مثال ↘ ↘

```
f = open("Abo_Habib_file.txt", "r")  
print(f.read())
```

إذا كان الملف موجودًا في موقع مختلف، فسيتعين عليك تحديد مسار الملف،
مثل هذا:

مثال ↘ ↘

افتح ملفًا في موقع مختلف:

```
f = open("D:\\myfiles\\welcome_Abo_Habib.txt", "r")  
print(f.read())
```

قراءة أجزاء فقط من الملف

تقوم الدالة بإرجاع النصوص كاملة، ولكن تستطيع أيضًا **read()** بشكل افتراضي
تحديد عدد الأحرف التي تريد إرجاعها:

مثال ↘ ↘

قم بإرجاع الأحرف الخمسة الأولى من الملف:

```
f = open("Abo_Habib_file.txt", "r")
print(f.read(5))
```

قراءة السطور

الداالة **readline()** تستطيع إرجاع سطر واحد باستخدام

مثال ↘ ↘

اقرأ سطرًا واحدًا من الملف:

```
f = open("Abo_Habib_file.txt", "r")
print(f.readline())
```

مرتين، تستطيع قراءة السطرين الأولين **readline()** من خلال الاتصال

مثال ↘ ↘

اقرأ سطرين من الملف:

```
f = open("Abo_Habib_file.txt", "r")
print(f.readline())
print(f.readline())
```

من خلال تكرار سطور الملف، تستطيع قراءة الملف بأكمله سطرًا تلو الآخر

مثال ↘ ↘

قم بالنسخ عبر سطر الملف سطرًا

```
f = open("Abo_Habib_file.txt", "r")
for x in f:
    print(x)
```

إغلاق الملفات

من الممارسات الجيدة إغلاق الملف دائمًا عند الانتهاء منه

مثال ↘ ↘

أغلق الملف عند الانتهاء منه

```
f = open("Abo_Habib_file.txt", "r")
print(f.readline())
f.close()
```

ملاحظة: يجب عليك دائمًا إغلاق ملفاتك، في بعض الحالات، بسبب التخزين المؤقت، قد لا تظهر التغييرات التي تم إجراؤها على الملف حتى تقوم بإغلاق الملف.

✓ كتابة ملف بايثون

الكتابة إلى ملف موجود

open(): للكتابة إلى ملف موجود، يجب عليك إضافة معلمة إلى الدالة

ادراج - سيتم ادراجه بنهاية الملف - "a"

الكتابة - سوف تحل محل أي محتوى موجود - "w"

مثال ↘ ↘

نألحق المحتوى بالملف "Abo_Habib_file2.txt" افتح الملف

```
f = open("Abo_Habib_file2.txt", "a")
```

```
f.write("Now the file has more Habib content!")
```

```
f.close()
```

```
#open and read the file after the appending:
```

```
f = open("Abo_Habib_file2.txt", "r")
```

```
print(f.read())
```


مثال ↘ ↘

نواستبدل المحتوى "Abo_Habib_file3.txt" افتح الملف

```
f = open("Abo_Habib_file3.txt", "w")  
f.write("Woops! I have deleted the content!")  
f.close()
```

```
#open and read the file after the appending:  
f = open("Abo_Habib_file3.txt", "r")  
print(f.read())
```

ستحل محل الملف بأكمله "w" ملحوظة: الدالة

إنشاء ملف جديد

الدالة مع أحد الكلمات التالية (**open()**) لإنشاء ملف جديد في بايثون، استخدم

إنشاء - سيتم إنشاء ملف، وإرجاع خطأ في حالة وجود الملف - "x"

أراج - سيتم إنشاء ملف إذا كان الملف المحدد غير موجود - "a"

الكتابة - سيتم إنشاء ملف إذا كان الملف المحدد غير موجود - "w"

مثال ↘ ↘

"Habib_file.txt" قم بإنشاء ملف يسمى

```
f = open("Habib_file.txt", "x")
```

!ال: يتم إنشاء ملف فارغ جديد

مثال ↘ ↘

أنشئ ملفًا جديدًا إذا لم يكن موجودًا

```
f = open("Habib_file.txt", "w")
```

✓ حذف الملف في بايثون

حذف ملف

`os.remove()` لحذف ملف، يجب عليك استيراد وحدة نظام التشغيل وتشغيل وظيفتها:

مثال ↘ ↘

إزالة الملف "Abo_Habib_file.txt":

```
import os  
os.remove("Abo_Habib_file.txt")
```

التحقق من وجود الملف:

لتجنب خطأ، قد ترغب في التحقق من وجود الملف قبل محاولة حذفه:

مثال ↘ ↘

تحقق من وجود الملف ثم احذفه:

```
import os
if os.path.exists("Abo_Habib_file.txt"):
    os.remove("Abo_Habib_file.txt")
else:
    print("The file Abo_Habib not exist")
```

لحذف المجلد

الدالة `os.rmdir()` لحذف مجلد بأكمله، استخدم:

مثال ↘ ↘

إزالة المجلد "Abo_Habib_folder":

```
import os
os.rmdir("Abo_Habib_folder")
```

ملاحظة: تستطيع فقط إزالة المجلدات الفارغة.

استخدام ماي إس كيو إل داخل بايثون

هى نظام قواعد بيانات قوى جدا وسريع وفعال **MySQL**

.يمكن استخدام بايثون في تطبيقات قواعد البيانات

MySQL. واحدة من قواعد البيانات الأكثر شعبية هي فى العالم

برنامج التشغيل ماي إس كيو إل

لتمكن من تجربة أمثلة الاكواد في هذا الكتاب ، يجب أن يكون ماي إس كيو إل مثبتًا على جهاز الكمبيوتر

مجانية **MySQL** تستطيع تنزيل برنامج التشغيل على <https://www.mysql.com/downloads/> .

MySQL قم بتثبيت برنامج تشغيل

للوصول إلى برنامج التشغيل **MySQL** تحتاج بايثون إلى برنامج تشغيل **MySQL**.

"**MySQL Connector**" في هذا الكتاب سوف نستخدم برنامج التشغيل

"**MySQL Connector**" لتثبيت **PIP** نوصي باستخدام

. مثبتًا بالفعل في بيئة بايثون **PIP** من المرجح أن يكون

نواكب ما يلي، **PIP** انتقل في سطر الأوامر إلى موقع

"MySQL Connector" تنزيل و تثبيت

```
C:\Users\Your Name\AppData\Local\Programs\python\python36-32\Scripts>python pip install mysql-connector-python
```

MySQL. لقد قمت الآن بتنزيل و تثبيت برنامج تشغيل

MySQL في بايثون

مثبتًا "MySQL Connector" لاختبار ما إذا كان التثبيت ناجحًا، أو إذا كان بالفعل، قم بإنشاء صفحة بايثون بالمحتوى التالي:

Hosini_mysql.py:

```
import mysql.connector
```

"MySQL" إذا تم تنفيذ الاكواد أعلاه دون أي أخطاء، فسيتم تثبيت وجاهز للاستخدام "MySQL Connector"

إنشاء اتصال

. ابدأ بإنشاء اتصال بقاعدة البيانات

MySQL : استخدم اسم المستخدم وكلمة المرور من برنامج التشغيل

Habib_mysql_connection.py:

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password"  
)
```

```
print(Habib_DB)
```

SQL. تستطيع الآن البدء في الاستعلام عن قاعدة البيانات باستخدام عبارات

إنشاء قاعدة البيانات MySQL بايثون

لولا إنشاء برنامج التشغيل

MySQL، لإنشاء برنامج التشغيل في

مثال

"Abo_Habib_database" إنشاء برنامج التشغيل باسم

```
| import mysql.connector
```

```
| Habib_DB = mysql.connector.connect(  
|     host="localhost",  
|     user="Abo_Habib_User",  
|     password="Abo_Habib_password"  
| )
```

```
| Hosini_cursor = Habib_DB.cursor()
```

```
| Hosini_cursor.execute("CREATE DATABASE  
| Abo_Habib_database")
```

إذا تم تنفيذ الاكواد أعلاه دون أي أخطاء، فقد قمت بإنشاء برنامج التشغيل بنجاح.

تحقق من وجود قاعدة البيانات

تستطيع التحقق من وجود برنامج التشغيل عن طريق سرد جميع قواعد البيانات "في نظامك باستخدام عبارة "إظهار قواعد البيانات

مثال ↘ ↘

قم بإرجاع قائمة بقواعد بيانات نظامك

```
| import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("SHOW DATABASES")
```

```
for x in Hosini_cursor:  
    print(x)
```

أو تستطيع محاولة الوصول إلى قاعدة البيانات عند إجراء الاتصال

مثال ↘ ↘

"Abo_Habib_database": حاول الاتصال بقاعدة البيانات

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```


✓ إنشاء الجدول MySQL بايثون

إنشاء جدول

"CREATE TABLE" استخدم عبارة، MySQL لإنشاء جدول في
تأكد من تحديد اسم قاعدة البيانات عند إنشاء الاتصال

مثال ↘ ↘

:"قم بإنشاء جدول "العملاء"

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("CREATE TABLE customers  
(name VARCHAR(255), address VARCHAR(255))")
```

إذا تم تنفيذ الاكواد أعلاه دون أي أخطاء، فقد قمت الآن بإنشاء جدول بنجاح

تحقق من وجود الجدول

تستطيع التحقق من وجود جدول من خلال سرد جميع الجداول في قاعدة البيانات باستخدام عبارة "إظهار الجداول":

مثال ↘ ↘

قم بإرجاع قائمة بقواعد بيانات نظامك

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("SHOW TABLES")
```

```
for x in Hosini_cursor:  
    print(x)
```

المفتاح الأساسي

عند إنشاء جدول، يجب عليك أيضًا إنشاء عمود يحتوي على مفتاح فريد لكل سجل.

يمكن القيام بذلك عن طريق تحديد المفتاح الأساسي

"**INT AUTO_INCREMENT PRIMARY KEY**" نحن نستخدم العبارة

والتي ستقوم بإدراج رقم فريد لكل سجل. ابتداءً من 1، وزيادة بمقدار واحد لكل سجل.

مثال ↙ ↘

قم بإنشاء مفتاح أساسي عند إنشاء الجدول:

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("CREATE TABLE customers (id  
INT AUTO_INCREMENT PRIMARY KEY, name  
VARCHAR(255), address VARCHAR(255))")
```

ALTER TABLE: إذا كان الجدول موجودًا بالفعل، فاستخدم الكلمة المحجوزة

مثال ↘ ↘

إنشاء مفتاح أساسي في جدول موجود

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
host="localhost",  
user="Abo_Habib_User",  
password="Abo_Habib_password",  
database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("ALTER TABLE customers ADD  
COLUMN id INT AUTO_INCREMENT PRIMARY KEY")
```

إدراج في الجدول ✓

إدراج في الجداول

"INSERT INTO" استخدم عبارة، MySQL لملء جدول في

مثال ↘ ↘

:"أدخل سجلاً في جدول "العملاء"

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "INSERT INTO customers (name, address) VALUES  
(%s, %s)"  
val = ("Abo Habib", "Al Masry 21")  
Hosini_cursor.execute(sql, val)
```

```
Habib_DB.commit()
```

```
print(Hosini_cursor.rowcount, "record inserted.")
```

يجب إجراء التغييرات، وإلا لن **Habib_DB.commit()**. هام!: لاحظ العبارة
يتم إجراء أي تغييرات على الجدول.

إدراج صفوف كثيرة

الدالة **executemany()** لإدراج صفوف كثيرة في جدول، استخدم
هي قائمة من المجموعات، التي **executemany()** الكلمة الثانية للدالة
تحتوي على الاكواد التي تريد إدراجها

مثال ↘ ↘

: املأ جدول "العملاء" بالاكواد

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
host="localhost",  
user="Abo_Habib_User",  
password="Abo_Habib_password",  
database="Abo_Habib_database"
```

)

Hosini_cursor = Habib_DB.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

```
val = [  
    ('Omar', 'AL_Badrashin 4'),  
    ('Amy', 'Abo st 652'),  
    ('Yaseen', 'Mountain 21'),  
    ('MOHamed', 'Valley 345'),  
    ('Sandy', 'Ocean blvd 2'),  
    ('Hend', 'Green Grass 1'),  
    ('Esaa', 'Sky st 331'),  
    ('Susan', 'One way 98'),  
    ('MOHamed', 'Yellow Garden 2'),  
    ('Ben', 'Park Lane 38'),  
    ('waleed', 'Central st 954'),  
    ('Chuck', 'Main Road 989'),  
    ('Hosini7', 'Sideway 1633')  
]
```

Hosini_cursor.executemany(sql, val)

Habib_DB.commit()

print(Hosini_cursor.rowcount, "was inserted.")

معرف الخانة

تستطيع معرف الصف الذي أدرجته للتو عن طريق سؤال كائن المؤشر

ملاحظة: إذا قمت بإدراج أكثر من صف واحد، فسيتم إرجاع معرف الصف الأخير المدرج.

مثال ↘ ↘

أدخل صفًا واحدًا، ثم قم بإرجاع المعرف

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "INSERT INTO customers (name, address) VALUES  
(%s, %s)"  
val = ("Michelle", "Blue Village")  
Hosini_cursor.execute(sql, val)
```



```
Habib_DB.commit()
```

```
print("1 record inserted, ID:", Hosini_cursor.lastrowid)
```

حدد العناصر من جدول

"SELECT" استخدم عبارة، MySQL للاختيار من جدول في

مثال ↘ ↘

: حدد كافة السجلات من جدول "العملاء"، واعرض

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("SELECT * FROM customers")
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

الدالة التي تجلب جميع المصفوفة من آخر **fetchall()** ملاحظة: نستخدم عبارة تم تنفيذها.

اختيار الأعمدة معينة

متبوعة "SELECT" لتحديد بعض الأعمدة فقط في الجدول، استخدم عبارة باسم (أسماء) الأعمدة:

مثال ↘ ↘

:حدد أعمدة الاسم والعنوان فقط

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",
```

```
password="Abo_Habib_password",  
database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("SELECT name, address FROM  
customers")
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

fetchone() باستخدام دالة

الدالة **fetchone()** إذا كنت مهتمًا بصف واحد فقط، فتستطيع استخدام هذه

: ستعيد الدالة الصف الأول من ال **fetchone()**

مثال ↘ ↘

: جلب صف واحد فقط

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(
```

```
host="localhost",
user="Abo_Habib_User",
password="Abo_Habib_password",
database="Abo_Habib_database"
)

Hosini_cursor = Habib_DB.cursor()

Hosini_cursor.execute("SELECT * FROM customers")

Hosini_result = Hosini_cursor.fetchone()

print(Hosini_result)
```

✓ كلمة أين MySQL بايثون

حدد مع اختيارات معينة

عند تحديد السجلات من جدول، تستطيع تصفية التحديد باستخدام عبارة "WHERE":

مثال ↘ ↘

ال: "Park Lane 38" حدد السجل (السجلات) التي يكون العنوان فيها

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "SELECT * FROM customers WHERE address  
='Park Lane 38'"
```

```
Hosini_cursor.execute(sql)
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

التحكم في البحث

تستطيع أيضاً تحديد السجلات التي تبدأ أو تتضمن أو تنتهي بحرف أو عبارة معينة.

:استخدم % لتمثيل أحرف البدل

مثال ↘ ↘

"حدد السجلات التي يحتوي عنوانها على كلمة "دالة

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "SELECT * FROM customers WHERE address LIKE  
'%way%'"
```

```
Hosini_cursor.execute(sql)
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

SQL منع الهندسة العكسية

عندما يتم توفير قيم الاستعلام من قبل المستخدم، يجب عليك اثتثناء من القيم.

وهو أسلوب شائع لاختراق الويب لتدمير SQL، وذلك لمنع الهندسة العكسية. قاعدة البيانات أو إساءة استخدامها.

على دوال للاثتثناء من قيم الاستعلام `mysql.connector` تحتوي الوحدة

مثال ↘ ↘

placeholder: دالة `%s` اثتثناء من قيم الاستعلام باستخدام

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "SELECT * FROM customers WHERE address = %s"  
adr = ("Yellow Garden 2", )
```

```
Hosini_cursor.execute(sql, adr)
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:
```

```
print(x)
```

ترتيب حسب MySQL بايثون ✓

فرز الجداول

لفرز الجداول بترتيب تصاعدي أو تنازلي **ORDER BY** استخدم عبارة

بفرز تصاعدياً بشكل افتراضي. لفرز **ORDER BY** تقوم الكلمة المحجوزة

DESC بترتيب تنازلي، استخدم الكلمة المحجوزة

مثال ↘ ↘

ترتيب أبجدياً حسب الاسم: ال

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(
```

```
host="localhost",
```

```
user="Abo_Habib_User",
```

```
password="Abo_Habib_password",
```

```
database="Abo_Habib_database"
```

```
)
```



```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "SELECT * FROM customers ORDER BY name"
```

```
Hosini_cursor.execute(sql)
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

الطلب حسب الوصف

لفرز بترتيب تنازلي DESC استخدم الكلمة المحجوزة

مثال ↙ ↘

ترتيب عكسيا أبجديا حسب الاسم

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"
```

|)

| Hosini_cursor = Habib_DB.cursor()

| sql = "SELECT * FROM customers ORDER BY name
| DESC"

| Hosini_cursor.execute(sql)

| Hosini_result = Hosini_cursor.fetchall()

| for x in Hosini_result:
| print(x)

✓ حذف الجدول MySQL

كيف حذف جدول

"DROP TABLE" تستطيع حذف جدول موجود باستخدام عبارة

مثال ↘ ↘

:"حذف جدول "العملاء"

| import mysql.connector

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "DROP TABLE customers"
```

```
Hosini_cursor.execute(sql)
```

احذف فقط إذا كان موجوداً

إذا كان الجدول الذي تريد حذفه محذوفاً بالفعل، أو غير موجود لأي سبب آخر، لتجنب حدوث خطأ IF EXISTS فتستطيع استخدام الكلمة المحجوزة

مثال ↘ ↘

:احذف جدول "العملاء" إن وجد

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",
```

```
user="Abo_Habib_User",  
password="Abo_Habib_password",  
database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "DROP TABLE IF EXISTS customers"
```

```
Hosini_cursor.execute(sql)
```

✓ تحديث الجداول MySQL

كيف يتم تحديث الجدول

تستطيع تحديث السجلات الموجودة في الجدول باستخدام عبارة "UPDATE":

مثال ↘ ↘

"Canyon 123" إلى "Valley 345" استبدل عمود العنوان من

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
host="localhost",
```

```
user="Abo_Habib_User",
password="Abo_Habib_password",
database="Abo_Habib_database"
)

Hosini_cursor = Habib_DB.cursor()

sql = "UPDATE customers SET address = 'Canyon 123'
WHERE address = 'Valley 345'"

Hosini_cursor.execute(sql)

Habib_DB.commit()

print(Hosini_cursor.rowcount, "record(s) affected")
```

يجب إجراء التغييرات، وإلا لن **Habib_DB.commit()**. هام!: لاحظ العبارة
يتم إجراء أي تغييرات على الجدول

السجل **WHERE** تحدد جملة **UPDATE**: في بناء جملة **WHERE** لاحظ جملة
فسيتم **WHERE**، أو السجلات التي يجب تحديثها. إذا قمت بحذف جملة
!تحديث جميع السجلات

تابع منع الهندسة العكسية

من الممارسات الجيدة ائتناء من قيم أي استعلام، وذلك أيضًا في اكواد التحديث.

وهو أسلوب شائع لاختراق الويب لتدمير SQL، وذلك لمنع الهندسة العكسية قاعدة البيانات أو إساءة استخدامها.

للائتناء من القيم **%s** العنصر النائب **mysql.connector** تستخدم الوحدة في عبارة الحذف:

مثال ↘ ↘

أسلوب العنصر النائب **%s** تخطي القيم باستخدام

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "UPDATE customers SET address = %s WHERE  
address = %s"  
val = ("Valley 345", "Canyon 123")
```

```
Hosini_cursor.execute(sql, val)
```

```
Habib_DB.commit()
```

```
print(Hosini_cursor.rowcount, "record(s) affected")
```

✓ MySQL حدود بايثون

الحد من ال

تستطيع تحديد عدد السجلات التي يتم إرجاعها من الاستعلام، باستخدام عبارة "LIMIT":

مثال ↘ ↘

"حدد السجلات الخمسة الأولى في جدول "العملاء":

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
Hosini_cursor.execute("SELECT * FROM customers  
LIMIT 5")
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

ابدأ البحث من موضع آخر

إذا كنت تريد إرجاع خمسة سجلات، بدءًا من السجل الثالث، فتستطيع استخدام الكلمة المحجوزة "OFFSET":

مثال ↘ ↘

ابدأ من الموضع 3، وقم بإرجاع 5 سجلات

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database")
```


|)

| Hosini_cursor = Habib_DB.cursor()

| Hosini_cursor.execute("SELECT * FROM customers
LIMIT 5 OFFSET 2")

| Hosini_result = Hosini_cursor.fetchall()

| for x in Hosini_result:
| print(x)

الانضمام **MySQL** بايثون

ضم إلى جدولين أو أكثر

تستطيع دمج صفوف من جدولين أو أكثر، استنادًا إلى عمود مرتبط بينهما،
باستخدام عبارة JOIN.

: "ضع في اعتبارك أن موجود جدول "المستخدمين" وجدول "المنتجات"

المستخدمين

```
{ id: 1, name: 'Habib', fav: 154},  
{ id: 2, name: 'Omar', fav: 154},  
{ id: 3, name: 'Hana', fav: 155},  
{ id: 4, name: 'Yaseen', fav:},  
{ id: 5, name: 'MOHamed', fav:}
```

مديرين

```
{ id: 154, name: 'Abo Ali Al Masry' },  
{ id: 155, name: 'Abo Osama },  
{ id: 156, name: 'Abo Hosam' }
```

حقل المستخدمين وحقل الناتج **fav** يمكن دمج هذين الجدولين باستخدام **id** ات.

مثال ↘ ↘

:انضم إلى المستخدمين والناتج ات لمعرفة اسم الناتج المفضل للمستخدمين

```
import mysql.connector
```

```
Habib_DB = mysql.connector.connect(  
    host="localhost",  
    user="Abo_Habib_User",  
    password="Abo_Habib_password",  
    database="Abo_Habib_database"  
)
```

```
Hosini_cursor = Habib_DB.cursor()
```

```
sql = "SELECT \  
users.name AS user, \  
products.name AS favorite \  
FROM users \  
INNER JOIN products ON users.fav = products.id"
```

```
Hosini_cursor.execute(sql)
```

```
Hosini_result = Hosini_cursor.fetchall()
```

```
for x in Hosini_result:  
    print(x)
```

كلاهما سيعطيك **INNER JOIN** بدلاً من **JOIN** ملاحظة: تستطيع استخدام نفس ال

الانضمام الأيسر

INNER في مثال \ \ أعلاه، تم استبعاد هانا وابو حبيب من ال، وذلك لأن **JOIN** يعرض فقط السجلات التي يوجد بها تطابق

إذا كنت تريد إظهار كافة المستخدمين، حتى لو لم يكن لديهم ناتج مفضل، **LEFT JOIN** فاستخدم عبارة

مثال ↘ ↘

حدد كافة المستخدمين والنتائج المفضل لديهم

```
sql = "SELECT \  
users.name AS user, \  
products.name AS favorite \  
FROM users \  
LEFT JOIN products ON users.fav = products.id"
```

الإنضمام الصحيح

إذا كنت تريد إرجاع جميع النتائج والمستخدمين الذين جعلوها مفضلة لديهم، حتى لو لم يكن هناك أي مستخدم يجعلها المفضلة لديهم، فاستخدم عبارة **RIGHT JOIN**:

مثال ↘ ↘

حدد جميع النتائج والمستخدمين (المستخدمين) الذين اختاروها كمفضلاتهم

```
sql = "SELECT \  
users.name AS user, \  
products.name AS favorite \  
FROM users \  
RIGHT JOIN products ON users.fav = products.id"
```

MongoDB

يمكن استخدام بايثون في تطبيقات قواعد البيانات
MongoDB الأكثر شعبية هي NoSQL واحدة من قواعد بيانات

مما يجعل JSON، بتخزين الاكواد في مستندات تشبه MongoDB يقوم
قاعدة البيانات مرنة جدا وقابلة للتطوير

لنتمكن من تجربة أمثلة الاكواد في هذا الكتاب ، ستحتاج إلى الوصول إلى
MongoDB برنامج التشغيل

المجانية MongoDB تستطيع تنزيل برنامج التشغيل
على <https://www.mongodb.com> .

➤ mongoDB

للوصول إلى برنامج التشغيل MongoDB تحتاج بايثون إلى برنامج تشغيل
MongoDB.

"PyMongo" MongoDB في هذا الكتاب سوف نستخدم برنامج تشغيل

"PyMongo" لتثبيت PIP نوصي باستخدام

. مثبتًا بالفعل في بيئة بايثون PIP من المرجح أن يكون

نواكتب ما يلي، PIP انتقل في سطر الأوامر إلى موقع

"PyMongo": تنزيل وتثبيت

```
C:\Users\Your Name\AppData\Local\Programs\python\python36-32\Scripts>python pip install pymongo
```

mongoDB. لقد قمت الآن بتنزيل وتثبيت برنامج تشغيل

اختبار بايمونجو

مثبتًا بالفعل، قم "pymongo" لاختبار ما إذا كان التثبيت ناجحًا، أو إذا كان بإنشاء صفحة بايثون بالمحتوى التالي:

demo_mongodb_test.py:

```
import pymongo
```

وجاهز "pymongo" إذا تم تنفيذ الاكواد أعلاه دون أي أخطاء، فسيتم تثبيت للاستخدام.

إنشاء برنامج التشغيل MongoDB في بايثون

كيف إنشاء برنامج التشغيل

MongoClient، ابدأ بإنشاء كائن MongoDB لإنشاء برنامج التشغيل في الصحيح واسم قاعدة البيانات التي IP للاتصال بعنوان URL ثم حدد عنوان. تريد إنشاءها.

بإنشاء قاعدة البيانات إذا لم تكن موجودة، وإجراء اتصال MongoDB سيقوم بها.

مثال ↘ ↘

"Abo_Habib_database": إنشاء برنامج التشغيل تسمى

```
| import pymongo
```

```
| Hosini_client =
```

```
| pymongo.MongoClient("mongodb://localhost:27017/")
```

```
| Habib_DB = Hosini_client["Abo_Habib_database"]
```

لا يتم إنشاء برنامج التشغيل حتى تحصل على MongoDB، هام: في المحتوى!

حتى تقوم بإنشاء مجموعة (جدول)، مع مستند واحد على MongoDB ينتظر الأقل (سجل) قبل أن يقوم فعليًا بإنشاء قاعدة البيانات (والمجموعة)

تحقق من وجود قاعدة البيانات

لا يتم إنشاء قاعدة البيانات حتى تحصل على MongoDB، تذكر: في المحتوى، لذلك إذا كانت هذه هي المرة الأولى التي تقوم فيها بإنشاء برنامج التشغيل، فيجب عليك إكمال الفصلين التاليين (إنشاء مجموعة وإنشاء مستند) ! قبل التحقق من وجود قاعدة البيانات

تستطيع التحقق من وجود برنامج التشغيل عن طريق سرد جميع قواعد البيانات الموجودة في نظامك:

مثال ↙ ↘

قم بإرجاع قائمة بقواعد بيانات نظامك

```
| print(Hosini_client.list_database_names())
```

أو تستطيع التحقق من برنامج التشغيل محددة بالاسم

مثال ↘ ↘

"تحقق من وجود " قاعدة بيانات باسم ابو حبيب فى برنامج التشغيل

```
dblist = Hosini_client.list_database_names()
if "Abo_Habib_database" in dblist:
    print("The database exists.")
```

✓ إنشاء مجموعة MongoDB

SQL . هي نفس الجدول في قواعد بيانات MongoDB المجموعة في

إنشاء مجموعة

استخدم كائن قاعدة البيانات وحدد اسم ، MongoDB لإنشاء مجموعة في المجموعة التي تريد إنشاءها.

بإنشاء المجموعة إذا لم تكن موجودة MongoDB سيقوم

مثال ↘ ↘

"قم بإنشاء مجموعة تسمى " العملاء

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")  
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

لا يتم إنشاء المجموعة حتى تحصل على المحتوى، MongoDB هام: في حتى تقوم بإدراج مستند قبل أن يقوم بإنشاء المجموعة MongoDB ينتظر فعليًا.

تحقق من وجود المجموعة

لا يتم إنشاء المجموعة حتى تحصل على المحتوى، MongoDB تذكر: في لذلك إذا كانت هذه هي المرة الأولى لك في إنشاء مجموعة، فيجب عليك إكمال الفصل التالي (إنشاء مستند) قبل التحقق من وجود المجموعة

تستطيع التحقق من وجود مجموعة في برنامج التشغيل عن طريق سرد كافة المجموعات:

مثال ↙ ↘

: قم بإرجاع قائمة بجميع المجموعات في قاعدة البيانات

```
print(Habib_DB.list_collection_names())
```

أو تستطيع التحقق من مجموعة معينة بالاسم

مثال ↘ ↘

"تحقق من وجود مجموعة العملاء"

```
Habib_col_list = Habib_DB.list_collection_names()
if "customers" in Habib_col_list:
    print("The collection exists.")
```

✓ إدراج مستند MongoDB بايثون

SQL هو نفس السجل في قواعد بيانات MongoDB المستند في

إدراج في المجموعة

في مجموعة، MongoDB لإدراج سجل، أو مستند كما يطلق عليه في `insert_one()` نستخدم الدالة

هي قاموس يحتوي على الاسم (الأسماء) `insert_one()` الكلمة الأولى للدالة والقيمة (القيم) لكل حقل في المستند الذي تريد إدراجه

مثال ↘ ↘

"أدخل سجلاً في مجموعة العملاء"

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
mydict = { "name": "Abo Habib", "address": "Al Masry  
37" }
```

```
x = Hosini_col.insert_one(mydict)
```

_id قم بإرجاع حقل

الذي يحتوي `InsertOneResult` بإرجاع كائن `insert_one()` تقوم الدالة والتي تحتوي على معرف المستند المدرج `inserted_id`، على خاصية

مثال ↘ ↘

`_id` أدخل سجلاً آخر في مجموعة "العملاء" وأرجع قيمة الحقل

```
mydict = { "name": "Omar", "address": "AL_Badrashin  
27" }
```

```
x = Hosini_col.insert_one(mydict)
```

print(x.inserted_id)

حقلًا لك ويعين معرفًا فريدًا لكل MongoDB فسيضيف **_id** إذا لم تحدد حقلًا مستند.

MongoDB يتم تحديد أي حقل، لذلك قام **_id** في مثال ↘ ↘ أعلاه لم فريد للسجل (المستند) **_id** بتعيين.

إدراج مستندات كثيرة

نستخدم MongoDB لإدراج مستندات كثيرة في مجموعة في الدالة **insert_many()** هذه.

هي قائمة تحتوي على قواميس تحتوي **insert_many()** الكلمة الأولى للدالة: على الاكواد التي تريد إدراجها

مثال ↘ ↘

import pymongo

Hosini_client =

pymongo.MongoClient("mongodb://localhost:27017/")

Habib_DB = Hosini_client["Abo_Habib_database"]

Hosini_col = Habib_DB["customers"]

Hosini_list = [

{ "name": "Amy", "address": "Abo st 652"},

{ "name": "Yaseen", "address": "Mountain 21"},

```
[{"name": "MOHamed", "address": "Valley 345"},
{"name": "Sandy", "address": "Ocean blvd 2"},
{"name": "Hend", "address": "Green Grass 1"},
{"name": "Esaa", "address": "Sky st 331"},
{"name": "Susan", "address": "One way 98"},
{"name": "MOHamed", "address": "Yellow Garden 2"},
{"name": "Ben", "address": "Park Lane 38"},
{"name": "waleed", "address": "Central st 954"},
{"name": "Chuck", "address": "Main Road 989"},
{"name": "Habib", "address": "Badrasheen 1633"}
]
```

```
x = Hosini_col.insert_many(Hosini_list)
```

```
#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

الذي `InsertManyResult` بإرجاع كائن `insert_many()` تقوم الدالة والتي تحتوي على معرفات المستندات `inserted_ids`، يحتوي على خاصية المدرجة.

قم بإدراج مستندات كثيرة بمعرفات محددة

بتعيين معرفات فريدة لمستندك، `MongoDB` إذا كنت لا تريد أن يقوم عند إدراج المستند (المستندات) `_id` فتستطيع تحديد حقل

تذكر أن القيم يجب أن تكون فريدة من نوعها. لا يمكن أن يكون للمستندين `_id` نفس.

مثال ↘ ↘

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
Hosini_list = [
```

```
{ "_id": 1, "name": "Abo Habib", "address": "Al Masry  
37"},
```

```
{ "_id": 2, "name": "Omar", "address": "AL_Badrashin  
27"},
```

```
{ "_id": 3, "name": "Amy", "address": "Abo st 652"},
```

```
{ "_id": 4, "name": "Yaseen", "address": "Mountain 21"},
```

```
{ "_id": 5, "name": "MOHamed", "address": "Valley  
345"},
```

```
{ "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
```

```
{ "_id": 7, "name": "Hend", "address": "Green Grass 1"},
```

```
{ "_id": 8, "name": "Esaa", "address": "Sky st 331"},
```

```
{ "_id": 9, "name": "Susan", "address": "One way 98"},
```

```
{ "_id": 10, "name": "MOHamed", "address": "Yellow  
Garden 2"},
```

```
{ "_id": 11, "name": "Ben", "address": "Park Lane 38"},
```

```
{ "_id": 12, "name": "waleed", "address": "Central st
```

```
954"},  
  { "_id": 13, "name": "Chuck", "address": "Main Road  
989"},  
  { "_id": 14, "name": "Hosini7", "address": "Sideway  
1633"}  
]
```

```
x = Hosini_col.insert_many(Hosini_list)
```

```
#print list of the _id values of the inserted documents:  
print(x.inserted_ids)
```

✓ MongoDB البحث في

للعثور على الاكواد في **find_one()** الدوال **find()** نستخدم MongoDB في المجموعة.

للعثور على الاكواد في جدول في **SELECT** تمامًا مثلما يتم استخدام عبارة **MySQL** برنامج التشغيل.

ابحث عن واحدة

نستطيع استخدام MongoDB لتحديد الاكواد من مجموعة في الدالة **find_one()** هذه.

بإرجاع التواجد الأول في التحديد **find_one()** تقوم الدالة.

مثال ↘ ↘

:ابحث عن المستند الأول في مجموعة العملاء

```
| import pymongo
```

```
| Hosini_client =
```

```
| pymongo.MongoClient("mongodb://localhost:27017/")
```

```
| Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
| Hosini_col = Habib_DB["customers"]
```

```
| x = Hosini_col.find_one()
```

```
| print(x)
```

أوجد الكل

find() نستطيع أيضاً استخدام هذه، MongoDB لتحديد الاكواد من جدول في الدالة.

بإرجاع كافة التكرارات في التحديد **find()** تقوم الدالة

هي كائن استعلام. في هذا مثال ↘ ↘ ، **find()** الكلمة الأولى للأسلوب نستخدم كائن استعلام فارغاً، والذي يقوم بتحديد كافة المستندات الموجودة في المجموعة.

في **SELECT *** لا توجد كلمات في دالة البحث () تمنحك نفس مثل **MySQL**.

مثال ↘ ↘

قم بإرجاع جميع المستندات الموجودة في مجموعة "العملاء"، وقم بطباعة كل مستند:

```
| import pymongo
```

```
| Hosini_client =  
| pymongo.MongoClient("mongodb://localhost:27017/")  
| Habib_DB = Hosini_client["Abo_Habib_database"]  
| Hosini_col = Habib_DB["customers"]
```

```
| for x in Hosini_col.find():  
|     print(x)
```

إرجاع بعض الحقول فقط

. هي كائن يصف الحقول التي سيتم تضمينها في **find()** الكلمة الثانية للدالة . هذه الكلمة اختيارية، وإذا تم حذفها، سيتم تضمين جميع الحقول

مثال ↘ ↘

`_ids`: قم بإرجاع الأسماء والعناوين فقط، وليس

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
for x in Hosini_col.find({}, {"_id": 0, "name": 1, "address":  
1 }):
```

```
print(x)
```

إلا إذا كان أحد الحقول (غير مسموح لك بتحديد القيمتين 0 و 1 في نفس الكائن إذا قمت بتحديد حقل بالقيمة 0، فإن كافة الحقول الأخرى (`_id` هو حقل تحصل على القيمة 1، والعكس صحيح

مثال ↘ ↘

:سيستبعد هذا مثال ↘ ↘ "العنوان" من ال

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
Habib_DB = Hosini_client["Abo_Habib_database"]
Hosini_col = Habib_DB["customers"]
```

```
for x in Hosini_col.find({}, {"address": 0 }):
    print(x)
```

مثال ↘ ↘

إلا إذا (ستحصل على خطأ إذا قمت بتحديد القيمتين 0 و 1 في نفس الكائن
_id): كان أحد الحقول هو حقل

```
import pymongo
```

```
Hosini_client =
pymongo.MongoClient("mongodb://localhost:27017/")
Habib_DB = Hosini_client["Abo_Habib_database"]
Hosini_col = Habib_DB["customers"]
```

```
for x in Hosini_col.find({}, {"name": 1, "address": 0 }):
    print(x)
```

✓ MongoDB استعمال بيثون

تصفية الجدول

عند البحث عن مستندات في مجموعة، تستطيع تصفية باستخدام كائن استعلام.

هي كائن استعلام، ويتم استخدامها للحد من **find()** الوسيطة الأولى للأسلوب البحث.

مثال ↘ ↘

"Park Lane 38": ابحث عن المستند (المستندات) التي تحمل العنوان

```
| import pymongo
```

```
| Hosini_client =
```

```
| pymongo.MongoClient("mongodb://localhost:27017/")
```

```
| Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
| Hosini_col = Habib_DB["customers"]
```

```
| Hosini_query = { "address": "Park Lane 38" }
```

```
| mydoc = Hosini_col.find(Hosini_query)
```

```
| for x in mydoc:
```

```
|     print(x)
```

الاستعلام المتقدم

لإجراء استعلامات متقدمة، تستطيع استخدام المعدلات كقيم في كائن الاستعلام.

على سبيل **مثال** ↘ ↘ ، للعثور على المستندات التي يبدأ فيها حقل **address**: {"\$gt": "S"}: أو أعلى (أبجدياً)، استخدم المعدل أكبر من "S" العنوان "بالحرف "S":

مثال ↘ ↘

أو أعلى "S" ابحث عن المستندات التي يبدأ عنوانها بالحرف

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
Hosini_query = {"address": {"$gt": "S"}}
```

```
mydoc = Hosini_col.find(Hosini_query)
```

```
for x in mydoc:
```

```
    print(x)
```

تصفية مع التعبيرات العادية

تستطيع أيضًا استخدام التعبيرات العادية كمعدل.

لا يمكن استخدام التعبيرات العادية إلا للاستعلام عن النصوص.

"S" للعثور فقط على المستندات التي يبدأ فيها حقل "العنوان" بالحرف "\$regex": "^S": استخدم التعبير العادي

مثال ↘ ↘

"S": ابحث عن المستندات التي يبدأ عنوانها بالحرف

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
Hosini_query = {"address": {"$regex": "^S"}}
```

```
mydoc = Hosini_col.find(Hosini_query)
```

```
for x in mydoc:
```

```
print(x)
```

✓ MongoDB فرز و ترتيب

فرز

الدالة لفرز و بترتيب تصاعدي أو تنازلي **sort()** استخدم

معلمة واحدة لـ "اسم الحقل" ومعلمة واحدة لـ "الاتجاه" **sort()** تأخذ الدالة (الصاعد هو الاتجاه الافتراضي).

مثال ↘ ↘

ترتيب أبجديا حسب الاسم

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
mydoc = Hosini_col.find().sort("name")
```

```
for x in mydoc:
```

```
print(x)
```


ترتيب تنازلي

استخدم القيمة -1 كمعلمة ثانية للفرز تنازليًا

تصاعدي # (اسم، "1")

تنازلي # (اسم، "-1")

مثال ↘ ↘

ترتيب عكسياً أبجدياً حسب الاسم

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
mydoc = Hosini_col.find().sort("name", -1)
```

```
for x in mydoc:
```

```
print(x)
```

✓ **MongoDB Drop المجموعات**

حذف المجموعة

باستخدام MongoDB، تستطيع حذف جدول، أو مجموعة كما يطلق عليها في الدالة **drop()** هذه.

مثال ↘ ↘

"حذف مجموعة العملاء"

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
Hosini_col.drop()
```

صحيحًا إذا تم حذف المجموعة بنجاح، وخطأ إذا كانت **drop()** تُرجع الدالة المجموعة غير موجودة.

تحديث المجموعة

باستخدام MongoDB تستطيع تحديث سجل أو مستند كما يطلق عليه في الدالة **update_one()**.

هي كائن استعلام يحدد المستند الذي **update_one()** الكلمة الأولى للدالة سيتم تحديثه.

ملاحظة: إذا عثر الاستعلام على أكثر من سجل واحد، فسيتم تحديث التواجد الأول فقط.

الكلمة الثانية هي كائن يحدد القيم الجديدة للمستند.

مثال ↙ ↘

"Canyon 123" إلى "Valley 345" قم بتغيير العنوان من

```
import pymongo
```

```
Hosini_client =
```

```
pymongo.MongoClient("mongodb://localhost:27017/")
```

```
Habib_DB = Hosini_client["Abo_Habib_database"]
```

```
Hosini_col = Habib_DB["customers"]
```

```
Hosini_query = { "address": "Valley 345" }
```

```
Hosini_values = { "$set": { "address": "Canyon 123" } }
```

```
Hosini_col.update_one(Hosini_query, Hosini_values)
```

```
#print "customers" after the update:  
for x in Hosini_col.find():  
    print(x)
```

تحديث متعدد

لتحديث كافة المستندات التي تفي بمعايير الاستعلام،
الداالة **update_many()** استخدم

مثال ↘ ↘

"S": قم بتحديث كافة المستندات التي يبدأ عنوانها بالحرف

```
import pymongo
```

```
Hosini_client =  
pymongo.MongoClient("mongodb://localhost:27017/")  
Habib_DB = Hosini_client["Abo_Habib_database"]  
Hosini_col = Habib_DB["customers"]
```

```
Hosini_query = { "address": { "$regex": "^S" } }  
Hosini_values = { "$set": { "name": "Minnie" } }
```

```
x = Hosini_col.update_many(Hosini_query, Hosini_values)
```

```
print(x.modified_count, "documents updated.")
```

✓ MongoDB حدود جلب البيانات

تحديد مستندات أو سجلات معينة في الجداول

الدالة **limit()** نستخدم MongoDB، للحد من في

معاملًا واحدًا، وهو رقم يحدد عدد المستندات التي سيتم **limit()** تأخذ الدالة إرجاعها.

:"اعتبر أن موجود مجموعة "العملاء

```
{'_id': 1, 'name': 'Habib', 'address': 'Al Masry 37'}
{'_id': 2, 'name': 'Omar', 'address': 'AL_Badrashin 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Abo st 652'}
{'_id': 4, 'name': 'Yaseen', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'MOHamed', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Hend', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Esaa', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'MOHamed', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'waleed', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Hosini7', 'address': 'Sideway 1633'}
```

مثال ↘ ↘

حدد بإرجاع 5 مستندات فقط:

```
| import pymongo
```

```
| Hosini_client =  
| pymongo.MongoClient("mongodb://localhost:27017/")  
| Habib_DB = Hosini_client["Abo_Habib_database"]  
| Hosini_col = Habib_DB["customers"]
```

```
| Hosini_result = Hosini_col.find().limit(5)
```

```
| #print the result:  
| for x in Hosini_result:  
|     print(x)
```

Method	وصف
capitalize()	تحويل الحرف الأول إلى أحرف كبيرة
casefold()	تحويل النصوص إلى حالة صغيرة
center()	إرجاع نصوص في الوسط
count()	إرجاع عدد المرات التي تحدث فيها قيمة محددة في

	نص
encode()	إرجاع نسخة مشفرة من النصوص
endswith()	يُرجع صحيحًا إذا انتهت النصوص بالقيمة المحددة
expandtabs()	يضبط حجم علامة التبويب للنص
find()	يبحث في النصوص عن قيمة محددة ويعيد الموضع الذي تم العثور عليه فيه
format()	تنسيق القيم المحددة في نص
format_map()	تنسيق القيم المحددة في نص
index()	يبحث في النصوص عن قيمة محددة ويعيد الموضع الذي تم العثور عليه فيه
isalnum()	إذا كانت جميع الأحرف في النصوص True إرجاع عبلة عن حروف أبجدية رقمية
isalpha()	إرجاع صحيح إذا كانت جميع الأحرف في النصوص بالأحرف الأبجدية
isascii()	إذا كانت جميع الأحرف في النصوص True إرجاع ascii عبلة عن أحرف
isdecimal()	إذا كانت جميع الأحرف في النصوص True إرجاع عبلة عن أرقام عشرية
isdigit()	إذا كانت جميع الأحرف في النصوص True إرجاع عبلة عن أرقام
isidentifier()	إذا كانت النصوص عبلة عن معرف True إرجاع
islower()	إرجاع صحيح إذا كانت كافة الأحرف في النصوص صغيرة
isnumeric()	إرجاع صحيح إذا كانت كافة الأحرف في النصوص رقمية
isprintable()	إذا كانت كافة الأحرف الموجودة في True إرجاع النصوص قابلة للطباعة

isspace()	إذا كانت جميع الأحرف في النصوص True إرجاع عبارة عن مسافات بيضاء
istitle()	إذا كانت النصوص تتبع قواعد True تُرجع القيمة العنوان
isupper()	إذا كانت جميع الأحرف في True تُرجع القيمة النصوص مكتوبة بأحرف كبيرة
join()	تحويل عناصر النسخ إلى نص
ljust()	إرجاع نسخة مضبوطة على اليسار من النصوص
lower()	تحويل نصوص إلى حالة صغيرة
rstrip()	إرجاع نسخة القطع اليسرى من النصوص
maketrans()	إرجاع جدول الترجمة لاستخدامه في الترجمات
partition()	تقوم بإرجاع صف حيث يتم تقسيم النصوص إلى ثلاثة أجزاء
replace()	إرجاع نصوص حيث يتم استبدال قيمة محددة بقيمة محددة
rfind()	يبحث في النصوص عن قيمة محددة ويعيد آخر موضع تم العثور عليه فيه
rindex()	يبحث في النصوص عن قيمة محددة ويعيد آخر موضع تم العثور عليه فيه
rjust()	إرجاع نسخة صحيحة ومبررة من النصوص
rpartition()	تقوم بإرجاع صف حيث يتم تقسيم النصوص إلى ثلاثة أجزاء
rsplit()	يقسم النصوص عند الفاصل المحدد، ويعيد قائمة
rstrip()	إرجاع نسخة القطع الصحيحة من النصوص
split()	يقسم النصوص عند الفاصل المحدد، ويعيد قائمة
splitlines()	يقسم النصوص عند فواصل الأسطر ويعيد القائمة

startswith()	يُرجع صحيحًا إذا كانت النصوص تبدأ بالقيمة المحددة
strip()	إرجاع نسخة مهينه من النصوص
swapcase()	يتم تبديل الحالات، حيث تصبح الأحرف الصغيرة كبيرة والعكس صحيح
title()	تحويل الحرف الأول من كل كلمة إلى أحرف كبيرة
translate()	إرجاع نصوص مترجمة
upper()	تحويل نصوص إلى حالة احرف كبيرة
zfill()	يملأ النصوص بعدد محدد من القيم 0 في البداية

شرح بعض المكتبات المهمة فى بايثون

✓ مكتبة NumPy

هي مكتبة بايثون. للتحكم فى النصوص المصفوفات بشكل احترافى او اكبر مرونة ورغم انها تتقارب مع استخدام المجموعات والقوائم والقواميس التى درسناها فى الفصول الاولى الى انها تضيف لك ادوات جديدة سنتعرف علي استخداماتها فى الفصل القادم باذن الله تعالى

للعمل مع المصفوفات NumPy يستخدم

"بايثون Numerical" هو اختصار لـ NumPy.

التعلم عن طريق القراءة

لقد أنشأنا 43 صفحة في هذا الكتاب عن هذه المكتبة لتتعلم المزيد فهي مهم جدا لمن يريد احتراف اللغة NumPy.

البدء بمقدمة أساسية وينتهي بإنشاء مجموعات اكواد عشوائية وتخطيطها، NumPy يمكنك التحكم فيها بسهولة كبيرة والعمل باستخدام دوال

✓ NumPy مقدمة

NumPy؟ باختصار

كما ذكرنا هي مكتبة بايثون تسهل العمل مع النصوص والمصفوفات

.بدلا من القوائم المدمجة في اللغة تستخدم للعمل مع المصفوفات NumPy كما أن لديها دوال للعمل في مجال الجبر الخطي، وتحويل فوربييه، والمصفوفات

في عام 2005 على يد ترافيس أوليفانت. إنه مشروع NumPy تم إنشاء مفتوح المصدر وتستطيع استخدامه بحرية

.بايثون Numerical تعني NumPy

الفرق بينها وبين القوائم والقواميس NumPy؟

.في بايثون لدينا قوائم تخدم غرض المصفوفات، لكنها بطيئة في المعالجة

إلى توفير كائن مصفوفة أسرع بما يصل إلى 50 مرة من قوائم NumPy يهدف بايثون التقليدية.

وهو يوفر الكثير من الدوال، NumPy ndarray يسمى كائن المصفوفة في سهلاً جداً ndarray الداعمة التي تجعل العمل معه أمراً

تُستخدم المصفوفات بشكل متكرر في علم الاكواد ، حيث تعد السرعة والموارد أمراً مهماً جداً.

علم الاكواد: هو فرع من علوم الكمبيوتر حيث ندرس كيفية تخزين الاكواد واستخدامها وتحليلها لاستخلاص المعلومات منها

لماذا نأمن أسرع من القوائم؟

يتم تخزين مصفوفات نامبي في مكان واحد متواصل في الذاكرة على عكس القوائم، بحيث يمكن للعمليات الوصول إليها ومعالجتها بكفاءة عالية

يُسمى هذا السلوك بالمحلية المرجعية في علوم الكمبيوتر

هذا هو السبب الرئيسي وراء كون نامبي أسرع من القوائم. كما تم تحسينه للعمل مع أحدث بنى وحدة المعالجة المركزية

ما هي اللغة التي تمت كتابة نامبي بها؟

هي مكتبة بايثون وهي مكتوبة جزئياً بلغة بايثون، ولكن معظم الأجزاء NumPy C++ أو C التي تتطلب حساباً سريعاً مكتوبة بلغة

NumPy أين توجد برنامج التشغيل

موجود في مستودع جيت هاب NumPy الكود المصدري لـ هذا <https://github.com/numpy/numpy>

. يمكن العديد من الأشخاص من العمل على نفس قاعدة الاكواد: github

البدء مع NumPy ✓

NumPy تثبيت

مثبتين بالفعل على النظام، فسيكون تثبيت PIP إذا كان موجود بايثون و NumPy أمراً سهلاً جداً

:قم بتثبيته باستخدام هذا الأمر

```
C:\Users\Your Name>pip install numpy
```

عليها NumPy إذا فشل هذا الأمر، فاستخدم توزيعه بايثون التي تم تثبيت وما إلى ذلك Spyder و Anaconda بالفعل، مثل

NumPy عمليات الاستيراد

بمجرد تثبيت نامبي ، قم باستيراده إلى تطبيقاتك عن طريق إضافة : المحجوزة import الكلمة

| `import numpy`

وأصبح جاهزاً للاستخدام NumPy الآن تم استيراد

مثال ↘ ↘

| `import numpy`

| `arr = numpy.array([1, 2, 3, 4, 5])`

| `print(arr)`

np اسم مختصر لـ NumPy

. الاسم المختصر **np** تحت NumPy عادةً ما يتم استيراد

الاسم المختصر: في بايثون الاسم المختصر هو اسم بديل للإشارة إلى نفس الشيء.

:الكلمة المحجوزة أثناء الاستيراد **as** قم بإنشاء اسم مستعار باستخدام

| `import numpy as np`

numpy بدلاً من **np** NumPy الآن يمكن الإشارة إلى حزمة

مثال ↘ ↘

```
| import numpy as np  
| arr = np.array([1, 2, 3, 4, 5])  
| print(arr)
```

NumPy التحقق من الإصدار

السمة `__version__` يتم تخزين نصوص الإصدار تحت

مثال ↘ ↘

```
| import numpy as np  
| print(np.__version__)
```

إنشاء المصفوفات NumPy

NumPy ndarray قم بإنشاء كائن

للعمل مع المصفوفات. يسمى كائن المصفوفة في NumPy يتم استخدام `NumPy ndarray`.

الدالة **array()** باستخدام **NumPy** كائن **ndarray** نستطيع إنشاء

مثال ↘ ↘

```
| import numpy as np  
| arr = np.array([1, 2, 3, 4, 5])  
| print(arr)  
| print(type(arr))
```

تخبرنا دالة **type ()** بايثون بنوع الكائن الذي تم تمريره إليها. كما هو الحال **numpy.ndarray** النوع **arr** في الكود أعلاه فإنه يوضح هذا

نستطيع تمرير قائمة أو صف أو أي كائن يشبه **ndarray** لإنشاء كائن **ndarray**: الدالة، وسيتم تحويله إلى **array()** المصفوفة إلى

مثال ↘ ↘

NumPy: لإنشاء مصفوفة **Tuple** استخدم

```
| import numpy as np  
| arr = np.array((1, 2, 3, 4, 5))  
| print(arr)
```

الأبعاد في المصفوفات

البعد في المصفوفات هو مستوى واحد من عمق المصفوفة (المصفوفات المتداخلة).

المصفوفة المتداخلة: هي المصفوفات التي تحتوي على مصفوفات كعناصر لها.

(-D) مصفوفات (0)

هي العناصر الموجودة في المصفوفة. كل قيمة Scalars، أو -D المصفوفات 0 هي مصفوفة هي مصفوفة 0.

مثال ↘ ↘

بقيمة 42 -D قم بإنشاء مصفوفة ذات قيمة 0

```
| import numpy as np
```

```
| arr = np.array(42)
```

```
| print(arr)
```


D-مصفوفات 1

كعناصرها مصفوفة أحادية D-تسمى المصفوفة التي تحتوي على مصفوفات 0 D-الأبعاد أو 1.

. هذه هي المصفوفات الأكثر شيوعًا والمحمولة

مثال ↘ ↘

قم بإنشاء مصفوفة أحادية الأبعاد تحتوي على القيم 1,2,3,4,5:

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5])
```

```
| print(arr)
```

مصفوفات ثنائية الأبعاد

المصفوفة التي تحتوي على مصفوفات أحادية الأبعاد كعناصرها تسمى مصفوفة ثنائية الأبعاد.

غالبًا ما تستخدم هذه لتمثيل المصفوفة أو الموترات من الدرجة الثانية.

على وحدة فرعية كاملة مخصصة لعمليات المصفوفة NumPy يحتوي `numpy.mat` تسمى

مثال ↘ ↘

قم بإنشاء مصفوفة ثنائية الأبعاد تحتوي على مصفوفتين بالقيم 1,2,3 و 4,5,6:

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
| print(arr)
```

مصفوفات ثلاثية الأبعاد

المصفوفة التي تحتوي على مصفوفات ثنائية الأبعاد (مصفوفات) كعناصرها تسمى مصفوفة ثلاثية الأبعاد.

غالبًا ما تستخدم هذه لتمثيل موتر من الدرجة الثالثة.

مثال ↘ ↘

قم بإنشاء مصفوفة ثلاثية الأبعاد تحتوي على مصفوفتين ثنائي الأبعاد، يحتوي كلاهما على مصفوفتين بالقيم 1,2,3 و 4,5,6:

```
| import numpy as np
```

```
| arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
```

```
print(arr)
```

التحقق من عدد الأبعاد؟

السمّة التي تُرجع عددًا صحيحًا يخبرنا بعدد **ndim** NumPy Arrays توفر أبعاد المصفوفة.

مثال ↘ ↘

تحقق من عدد أبعاد المصفوفات:

```
import numpy as np
```

```
a = np.array(42)
```

```
b = np.array([1, 2, 3, 4, 5])
```

```
c = np.array([[1, 2, 3], [4, 5, 6]])
```

```
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
```

```
print(a.ndim)
```

```
print(b.ndim)
```

```
print(c.ndim)
```

```
print(d.ndim)
```

مصفوفات الأبعاد العليا

يمكن أن تحتوي المصفوفة على أي عدد من الأبعاد

ndmin. عند إنشاء المصفوفة، تستطيع تحديد عدد الأبعاد باستخدام الوسيطة

مثال ↘ ↘

قم بإنشاء مصفوفة ذات 5 أبعاد وتأكد من أنها تحتوي على 5 أبعاد:

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4], ndmin=5)
```

```
| print(arr)
```

```
| print('number of dimensions :', arr.ndim)
```

في هذا المصفوفة، البعد الأعمق (5 ديم) لديه 4 عناصر، 4 ديم لديه عنصر واحد هو المتجه، 3 ديم لديه عنصر واحد هو المصفوفة مع المتجه، 2 ديم لديه عنصر واحد هو مصفوفة ثلاثي الأبعاد و يحتوي التعيين الأول على عنصر واحد عبارة عن مصفوفة رباعية الأبعاد.



NumPy رقمة مصفوفة

الوصول إلى عناصر المصفوفة

رقمنة المصفوفة في نومي هي نفس الوصول إلى عنصر المصفوفة العادية التي درسناها في الجزء الأول.

تستطيع الوصول إلى عنصر المصفوفة بالرجوع إلى الرقم الخاص به.

بالرقم 0، مما يعني أن العنصر الأول له **NumPy** تبدأ الأرقام في مصفوفات رقم 0، والثاني له رقم 1 وما إلى ذلك

مثال ↘ ↘

احصل على العنصر الأول من المصفوفة التالية:

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4])
```

```
| print(arr[0])
```

مثال ↘ ↘

احصل على العنصر الثاني من المصفوفة التالية.

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4])
```

```
| print(arr[1])
```

مثال ↘ ↘

احصل على العنصرين الثالث والرابع من المصفوفة التالية وأضفهما.

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4])
```

```
| print(arr[2] + arr[3])
```

الوصول إلى المصفوفات ثنائية الأبعاد

للوصول إلى العناصر من المصفوفات ثنائية الأبعاد، نستطيع استخدام أعداد صحيحة مفصولة بفواصل تمثل البعد والرقم الخاص بالعنصر.

فكر في المصفوفات ثنائية الأبعاد مثل جدول يحتوي على صفوف وأعمدة، حيث يمثل البعد الصف ويمثل الرقم العمود.

مثال ↘ ↘

:الوصول إلى العنصر الموجود في الصف الأول والعمود الثاني

```
import numpy as np
```

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
```

```
print('2nd element on 1st row: ', arr[0, 1])
```

مثال ↘ ↘

قم بالوصول إلى العنصر الموجود في الصف الثاني والعمود الخامس:

```
import numpy as np
```

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
```

```
print('5th element on 2nd row: ', arr[1, 4])
```

الوصول إلى المصفوفات ثلاثية الأبعاد

للوصول إلى العناصر من المصفوفات ثلاثية الأبعاد، نستطيع استخدام أعداد صحيحة مفصولة بفواصل تمثل أبعاد ورقم العنصر.

مثال ↘ ↘

الوصول إلى العنصر الثالث من المصفوفة الثانية من المصفوفة الأولى:

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]])
```

```
| print(arr[0, 1, 2])
```

مثال ↘ ↘

يطبع القيمة `arr[0, 1, 2]` 6

ولهذا السبب:

الرقم الأول يمثل البعد الأول، والذي يحتوي على مصفوفتين

`[[1, 2, 3], [4, 5, 6]]`

و:

`[[7, 8, 9], [10, 11, 12]]`

منذ اخترنا `0`، ويتبقى لدينا المصفوفة الأولى

`[[1, 2, 3], [4, 5, 6]]`

الرقم الثاني يمثل البعد الثاني، والذي يحتوي أيضًا على مصفوفتين

`[1, 2, 3]`

و:

`[4, 5, 6]`

وبما أننا اخترنا `1`، يتبقى لدينا المصفوفة الثانية

`[4, 5, 6]`

الرقم الثالث يمثل البعد الثالث والذي يحتوي على ثلاث قيم

4

5

6

نوبما أننا اخترنا 2، انتهى بنا الأمر إلى القيمة الثالثة
6

الأرقام السالبة

استخدم الأرقام السالبة للوصول إلى مصفوفة من النهاية

مثال ↘ ↘

اطبع العنصر الأخير من اللون الثاني:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy تقطيع مصفوفة في

تقطيع المصفوفات

التقطيع في لغة بايثون يعني أخذ العناصر من رقم معين إلى رقم معين آخر.
[start:end]. نقوم بتمرير العنصر بدلاً من الرقم هكذا

[start:end:step]. نستطيع أيضًا تحديد التقطيع كالتالي

إذا لم تنجح ، فسيتم اعتباره 0

إذا لم نتجاوز طول المصفوفة المعتبر في هذا البعد

إذا لم نجتاز الاتقطيع فسيتم اعتبارها 1

مثال ↘ ↘

قم بتقسيم العناصر من الرقم 1 إلى الرقم 5 من المصفوفة التالية

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
| print(arr[1:5])
```

ملاحظة: تتضمن ال رقم البداية، ولكنها تستبعد رقم النهاية

مثال ↘ ↘

العناصر من الرقم 4 إلى نهاية المصفوفة:

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
| print(arr[4:])
```

مثال ↘ ↘

:العناصر من البداية إلى الرقم 4 (غير مدرجة)

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
| print(arr[:4])
```

التقطيع بالسالب

:استخدم ناقص للإشارة إلى رقم من النهاية

مثال ↘ ↘

:عنصر من الرقم 3 من النهاية إلى الرقم 1 من النهاية

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
| print(arr[-3:-1])
```

التقطيع

القيمة لتحديد التقطيع **step** استخدم

مثال ↘ ↘

قم بإرجاع كل العناصر الأخرى من الرقم 1 إلى الرقم 5

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
| print(arr[1:5:2])
```

مثال ↘ ↘

قم بإرجاع كل عنصر آخر من المصفوفة بأكملها

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
| print(arr[::2])
```

تقطيع المصفوفات ثنائية الأبعاد

مثال ↘ ↘

من العنصر الثاني، قم بتقطيع العناصر من الرقم 1 إلى الرقم 4 (غير متضمن)

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
| print(arr[1, 1:4])
```

ملاحظة: تذكر أن العنصر الثاني يحتوي على رقم 1

مثال ↘ ↘

من كلا العنصرين، قم بإرجاع الرقم 2

```
| import numpy as np
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print(arr[0:2, 2])
```

مثال ↘ ↘

من كلا العنصرين، عنصر الرقم 1 إلى الرقم 4 (غير متضمنة)، سيعيد هذا مصفوفة ثنائية الأبعاد:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print(arr[0:2, 1:4])
```

أنواع الاجراءات ✓

أنواع الاجراءات الافتراضية في بايثون

افتراضياً، تحتوي لغة بايثون على أنواع الاجراءات التالية:

- **strings** - يستخدم لتمثيل الاكواد النصية، ويرد النصوص تحت علامات "Ahmed" ↘ ↘ الاقتباس. على سبيل مثال
- **integer** - يستخدم لتمثيل الأعداد الصحيحة. على سبيل مثال ↘ ↘ - 1، -2، -3

- **float**- 1.2 ، مثال $\searrow \searrow$ على سبيل الحقيقة. تستخدم لتمثيل الأعداد الحقيقية. 42.42
- **boolean**- يستخدم لتمثيل صحيح أو خطأ
- **complex**- مثال $\searrow \searrow$ على سبيل المركبة. يستخدم لتمثيل الأعداد المركبة. $2.0 + 1.0j$, $1.5 + 2.5j$

NumPy أنواع الاجرائات في

على بعض أنواع الاجرائات الإضافية، ويشير إلى أنواع NumPy يحتوي والأعداد الصحيحة غير **u** الأعداد الصحيحة **i** الاجرائات بحرف واحد، مثل **الموقعة وما إلى ذلك**.

والأحرف المستخدمة NumPy فيما يلي قائمة بجميع أنواع الاجرائات في لتمثيلها.

- **i**- عدد صحيح
- **b**- منطقية
- **u**- عدد صحيح غير موقعة
- **f**- رقم عشري
- **c**- رقم عشري معقد
- **m**- دلتا زمنية
- **M**- التاريخ والوقت
- **O**- هدف
- **S**- نص
- **U**- نصوص يونيكود
- **V**- جزء ثابت من الذاكرة للنوع الآخر (فولس او خطأ)

التحقق من نوع بيانات المصفوفة

تقوم بإرجاع **dtype** على خاصية تسمى **NumPy** يحتوي كائن المصفوفة
نوع بيانات المصفوفة

مثال ↘ ↘

نوع البيانات لكائن المصفوفة

```
| import numpy as np  
| arr = np.array([1, 2, 3, 4])  
| print(arr.dtype)
```

مثال ↘ ↘

نوع البيانات لمصفوفة تحتوي على نصوص

```
| import numpy as np  
| arr = np.array(['Abo', 'Habib', 'Al Hosiny'])  
| print(arr.dtype)
```


إنشاء مصفوفات بنوع بيانات محدد

الدالة لإنشاء المصفوفات، ويمكن لهذه الدالة أن تأخذ وسيطة `array()` نستخدم: تسمح لنا بتحديد نوع البيانات المتوقع لعناصر المصفوفة `dtype`: اختيارية

مثال ↘ ↘

: إنشاء مصفوفة بنص نوع البيانات

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4], dtype='S')
```

```
| print(arr)
```

```
| print(arr.dtype)
```

.تحديد الحجم أيضاً `U` نستطيع `S` و `f` و `u` و `i` بالنسبة لـ

مثال ↘ ↘

:قم بإنشاء مصفوفة تحتوي على عدد صحيح من نوع البيانات يبلغ 4 بايت

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4], dtype='i4')
```

```
| print(arr)
```

```
| print(arr.dtype)
```

ماذا لو لم يكمن تحويل القيمة؟

برفع قيمة **NumPy** إذا تم تحديد نوع لا يمكن إدراج العناصر فيه، فسيقوم **ValueError**.

عندما يكون نوع الوسيطة **ValueError** في بايثون يتم رفع **ValueError**: التي تم تمريرها إلى دالة غير متوقع/غير صحيح.

مثال ↘ ↘

إلى عدد صحيح (سيؤدي ذلك 'a' لا يمكن تحويل نصوص غير صحيحة مثل: إلى حدوث خطأ)

```
import numpy as np
```

```
arr = np.array(['a', '2', '3'], dtype='i')
```

تحويل نوع البيانات على المصفوفات الموجودة

أفضل دالة لتغيير نوع بيانات مصفوفة موجودة هي عمل نسخة من المصفوفة الدالة **astype()** باستخدام هذه

بإنشاء نسخة من المصفوفة، وتسمح لك بتحديد نوع **astype()** تقوم الدالة بالبيانات كمعلمة.

العدد **'i'** الأرقام العشرية أو **'f'** يمكن تحديد نوع البيانات باستخدام نص، مثل **float** الصحيح وما إلى ذلك. أو تستطيع استخدام نوع البيانات مباشرة مثل الصحيح **int** الأرقام العشرية والعدد.

مثال ↘ ↘

قيمة الكلمة **'i'** تغيير نوع البيانات من عدد عائم إلى عدد صحيح باستخدام

```
| import numpy as np
| arr = np.array([1.1, 2.1, 3.1])
| Habib_Array = arr.astype('i')
| print(Habib_Array)
| print(Habib_Array.dtype)
```

مثال ↘ ↘

قيمة الكلمة **int** تغيير نوع البيانات من عدد عائم إلى عدد صحيح باستخدام

```
| import numpy as np
| arr = np.array([1.1, 2.1, 3.1])
```

```
Habib_Array = arr.astype(int)
```

```
print(Habib_Array)
```

```
print(Habib_Array.dtype)
```

مثال ↘ ↘

تغيير نوع البيانات من عدد صحيح إلى منطقي

```
import numpy as np
```

```
arr = np.array([1, 0, 3])
```

```
Habib_Array = arr.astype(bool)
```

```
print(Habib_Array)
```

```
print(Habib_Array.dtype)
```

✓ النسخ والعرض فى NumPy

الفرق بين النسخ والعرض

الفرق الرئيسي بين نسخة وعرض المصفوفة هو أن النسخة عبارة عن مصفوفة جديدة، والعرض هو مجرد عرض للمصفوفة الأصلية.

تمتلك النسخة الاكواد وأي تغييرات يتم إجراؤها على النسخة لن تؤثر على المصفوفة الأصلية، وأي تغييرات يتم إجراؤها على المصفوفة الأصلية لن تؤثر على النسخة.

لا يمتلك العرض الاكواد وأي تغييرات يتم إجراؤها على العرض ستؤثر على المصفوفة الأصلية، وأي تغييرات يتم إجراؤها على المصفوفة الأصلية ستؤثر على العرض.

النسخ:

مثال ↙ ↘

:أنشئ نسخة، وقم بتغيير المصفوفة الأصلية، واعرض كلا المصفوفتين

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
x = arr.copy()
```

```
arr[0] = 42
```

```
print(arr)
```

```
print(x)
```

يجب ألا تتأثر النسخة بالتغييرات التي تم إجراؤها على المصفوفة الأصلية.

العرض:

مثال ↘ ↘

قم بعمل عرض، وقم بتغيير المصفوفة الأصلية، وعرض كلا المصفوفتين

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
x = arr.view()
```

```
arr[0] = 42
```

```
print(arr)
```

```
print(x)
```

يجب أن يتأثر العرض بالتغييرات التي تم إجراؤها على المصفوفة الأصلية.

إجراء تغييرات في دالة العرض

مثال ↘ ↘

قم بإنشاء عرض وتغيير العرض وعرض كلا المصفوفتين

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
x = arr.view()
```

```
x[0] = 31
```

```
print(arr)
```

```
print(x)
```

يجب أن يتأثر المصفوفة الأصلية بالتغييرات التي تم إجراؤها على دالة العرض.

تمتلك اكوادها Array تحقق مما إذا كانت

كما ذكرنا أعلاه، النسخ تمتلك الاكواد ، والمشاهدات لا تمتلك الاكواد ، لكن كيف نستطيع التحقق من ذلك؟

إذا كانت **None** التي تُرجع **base** على السمة **NumPy** تحتوي كل مصفوفة المصفوفة تمتلك الاكواد.

تشير السمة إلى الكائن الأصلي **base** وبخلاف ذلك

مثال ↘ ↘

:اطبع قيمة السمة المحجوزة للتحقق مما إذا كان المصفوفة تمتلك اكوادها أم لا

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
x = arr.copy()
```

```
y = arr.view()
```

```
print(x.base)
```

```
print(y.base)
```

✓ NumPy شكل او سمة المصفوفة

شكل المصفوفة هو عدد العناصر في كل بعد.

كيف شكل مصفوفة

والتي تُرجع صفًا يحتوي **shape** على سمة تسمى **NumPy** تحتوي مصفوفات كل رقم على عدد العناصر المقابلة.

مثال ↘ ↘

طباعة شكل مصفوفة ثنائية الأبعاد:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```


| `print(arr.shape)`

يُرجع مثال ↘ ↘ أعلاه (4,2)، مما يعني أن المصفوفة لها بعدان، حيث يحتوي البعد الأول على عنصرين والثاني يحتوي على 4

مثال ↘ ↘

متجه ذي القيم 1,2,3,4 وتحقق `ndmin` أنشئ مصفوفة ذات 5 أبعاد باستخدام: من أن البعد الأخير له القيمة 4

| `import numpy as np`

| `arr = np.array([1, 2, 3, 4], ndmin=5)`

| `print(arr)`

| `print('shape of array:', arr.shape)`

tuple؟ ماذا يمثل شكل

تشير الأعداد الصحيحة في كل رقم إلى عدد العناصر الموجودة في البعد المقابل.

في مثال ↘ ↘ أعلاه في الرقم-4 لدينا القيمة 4، لذا نستطيع القول أن البعد الخامس (1 + 4) يحتوي على 4 عناصر

NumPy إعادة تشكيل مصفوفة

إعادة تشكيل المصفوفات

إعادة التشكيل تعني تغيير شكل المصفوفة

شكل المصفوفة هو عدد العناصر في كل بعد

ومن خلال إعادة التشكيل نستطيع إضافة أو إزالة الأبعاد أو تغيير عدد العناصر في كل بعد

أبو حبيب الحسيني

D إلى D 2 -إعادة التشكيل من 1

مثال ↘ ↘

قم بتحويل المصفوفة أحادية الأبعاد التالية التي تحتوي على 12 عنصرًا إلى مصفوفة ثنائية الأبعاد.

:سيحتوي البعد الخارجي على 4 مصفوفات، تحتوي كل منها على 3 عناصر

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

```
| Habib_Array = arr.reshape(4, 3)
```

```
| print(Habib_Array)
```

D إلى D 3 -إعادة التشكيل من 1

مثال ↘ ↘

قم بتحويل المصفوفة أحادية الأبعاد التالية التي تحتوي على 12 عنصرًا إلى مصفوفة ثلاثية الأبعاد.

سيحتوي البعد الخارجي على مصفوفتين تحتوي كل منهما على 3 مصفوفات،
تحتوي كل منها على عنصرين

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

```
| Habib_Array = arr.reshape(2, 3, 2)
```

```
| print(Habib_Array)
```

هل نستطيع إعادة تشكيل أي شكل؟

نعم، بشرط أن تكون العناصر المطلوبة لإعادة التشكيل متساوية في الشكلين
إلى 4 عناصر في D نستطيع إعادة تشكيل مصفوفة مكونة من 8 عناصر 1
صفين مصفوفة ثنائية الأبعاد ولكن لا نستطيع إعادة تشكيلها إلى مصفوفة
 $3 \times 3 = 9$ مكونة من 3 عناصر و3 صفوف ثنائية الأبعاد لأن ذلك يتطلب 3
عناصر.

مثال ↘ ↘

حاول تحويل مصفوفة أحادية الأبعاد تحتوي على 8 عناصر إلى مصفوفة
ثنائية الأبعاد تحتوي على 3 عناصر في كل بُعد (سيظهر خطأ)

```
| import numpy as np
| arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
| Habib_Array = arr.reshape(3, 3)
| print(Habib_Array)
```

إرجاع نسخة أو عرض؟

مثال ↘ ↘

:تحقق مما إذا كانت المصفوفة التي تم إرجاعها عبارة عن نسخة أم عرض

```
| import numpy as np
| arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
| print(arr.reshape(2, 4).base)
```

ال. مثال ↘ ↘ أعلاه يُرجع المصفوفة الأصلية، لذا فهي دالة عرض

البعد غير معروف

"يُسمح لك ب بُعد واحد " غير معروف

وهذا يعني أنه ليس عليك تحديد رقم دقيق لأحد الأبعاد في دالة إعادة التشكيل بحساب هذا الرقم لك **Numpy** قم بتمريرها **-1** كقيمة، وسيقوم

مثال ↘ ↘

تحتوي على 8 عناصر إلى مصفوفة ثلاثية الأبعاد تحتوي **D** تحويل مصفوفة 1
2: على عناصر

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
| Habib_Array = arr.reshape(2, 2, -1)
```

```
| print(Habib_Array)
```

ملاحظة: لا نستطيع المرور **-1** إلى أكثر من بعد واحد

تسطيح المصفوفات

تعني تسوية المصفوفة تحويل مصفوفة كثيره الأبعاد إلى مصفوفة أحادية الأبعاد.

للقيام بذلك **reshape(-1)** نستطيع استخدامها

مثال ↘ ↘

تحويل المصفوفة إلى مصفوفة ثنائية الأبعاد:

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
| Habib_Array = arr.reshape(-1)
```

```
| print(Habib_Array)
```

ملاحظة: هناك الكثير من الدوال لتغيير أشكال المصفوفات في

numpy flatten، **ravel** وكذلك لإعادة ترتيب العناصر **rot90**، **flip**

وما إلى ذلك. وتندرج هذه ضمن القسم المتوسط إلى المتقدم **fliplr**، **flipud** و

numpy.

NumPy تكرار مصفوفة

تكرار المصفوفات

نفس الطريقة للمصفوفات العادية التي درسناها في الجزء الأول النسخ يعني المرور عبر العناصر واحدًا تلو الآخر.

نستطيع القيام بذلك، **numpy** عندما نتعامل مع المصفوفات كثيره الأبعاد في حلقة بايثون المحجوزة **for** باستخدام

إذا قمنا بالنسخ على مصفوفة أحادية الأبعاد، فسوف تمر عبر كل عنصر واحدًا تلو الآخر.

مثال ↘ ↘

:التالية D-كرر على عناصر المصفوفة 1

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3])
```

```
| for x in arr:
```

```
|     print(x)
```

تكرار المصفوفات ثنائية الأبعاد

.في المصفوفة ثنائية الأبعاد، سيتم المرور عبر جميع المصفوفة

مثال ↘ ↘

كرر على عناصر المصفوفة ثنائية الأبعاد التالية:

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
| for x in arr:
```

```
| print(x)
```

واحدًا تلو $n-1$ فسوف تمر عبر البعد $n-D$ إذا قمنا بالنسخ على مصفوفة الآخر.

لإرجاع القيم الفعلية، أي الكميات القياسية، علينا تكرار المصفوفات في كل بُعد.

مثال ↘ ↘

كرر على كل عنصر عددي من المصفوفة ثنائية الأبعاد:

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
| for x in arr:
```

```
for y in x:  
    print(y)
```

تكرار المصفوفات ثلاثية الأبعاد

في المصفوفة ثلاثية الأبعاد، سيتم المرور عبر جميع المصفوفات ثنائية الأبعاد

مثال ↘ ↘

تكرار على عناصر المصفوفة ثلاثية الأبعاد التالية:

```
import numpy as np
```

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

```
for x in arr:  
    print(x)
```

لإرجاع القيم الفعلية، أي الكميات القياسية، علينا تكرار المصفوفات في كل بُعد.

مثال ↘ ↘

النسخ وصولاً إلى العددية:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]])
```

```
for x in arr:  
    for y in x:  
        for z in y:  
            print(z)
```

nditer() تكرار المصفوفات باستخدام

هي دالة مساعدة يمكن استخدامها من التكرارات المحجوزة **nditer()** الدالة جدًا إلى التكرارات المتقدمة جدًا. فهو يحل بعض المشكلات المحجوزة التي نواجهها في النسخ ، فلنتناولها مع الأمثلة.

النسخ على كل عنصر عددي

الحلقات المحجوزة ، عند النسخ خلال كل عدد من المصفوفة، نحتاج **for** في والتي قد يكون من الصعب كتابتها للمصفوفات **for n** إلى استخدام حلقات ذات الأبعاد العالية جدًا.

مثال ↘ ↘

قم بالنسخ من خلال المصفوفة ثلاثية الأبعاد التالية

```
import numpy as np
```

```
arr = np.array([[1, 2], [3, 4]], [[5, 6], [7, 8]])
```

```
for x in np.nditer(arr):  
    print(x)
```

تكرار المصفوفة مع أنواع الإجراءات المختلفة

الوسيلة وتمرير نوع البيانات المتوقع لتغيير **op_dtypes** نستطيع استخدام نوع بيانات العناصر أثناء النسخ.

نوع بيانات العنصر في مكانه (حيث يكون العنصر في **Numpy** لا يغير المصفوفة) لذا فهو يحتاج إلى مساحة أخرى لتنفيذ هذا الإجراء، وتسمى هذه **nditer()** المساحة الإضافية المخزن المؤقت، ومن أجل تمكينها نمرر **flags=['buffered']**.

مثال ↙ ↘

نسخ المصفوفة كنص:

```
import numpy as np
```

```
arr = np.array([1, 2, 3])
```

```
for x in np.nditer(arr, flags=['buffered'],  
op_dtypes=['S']):  
    print(x)
```

النسخ مع أحجام التقطيع المختلفة

• نستطيع استخدام التصفية ويتبعها النسخ

مثال ↘ ↘

قم بالنسخ من خلال كل عنصر عددي في المصفوفة ثنائية الأبعاد مع تخطي عنصر واحد:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for x in np.nditer(arr[:, ::2]):
    print(x)
```

النسخ التعدادي باستخدام ndenumerate()

التعداد يعني ذكر الرقم المصفوفاتي للأشياء واحدًا تلو الآخر

في بعض الأحيان نطلب فهرسًا مطابقًا للعنصر أثناء
ويمكن استخدام الدالة لحالات الاستخدام تلك (**ndenumerate()**) ، النسخ

مثال ↘ ↘

التالية Dتعداد عناصر المصفوفات 1

```
| import numpy as np  
| arr = np.array([1, 2, 3])  
| for idx, x in np.ndenumerate(arr):  
| print(idx, x)
```

مثال ↘ ↘

تعداد عناصر المصفوفة ثنائية الأبعاد التالية

```
| import numpy as np  
| arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
| for idx, x in np.ndenumerate(arr):  
| print(idx, x)
```

✓ الانضمام والدمج بين NumPy المصفوفات

NumPy الانضمام بين المصفوفات

الانضمام يعني وضع محتويات مصفوفتين أو أكثر في مصفوفة واحدة.

نربط NumPy نربط الجداول بناءً على مفتاح، بينما في SQL، في المصفوفات حسب المحاور.

نقوم بتمرير نصوص من المصفوفات التي نريد ضمها إلى بالإضافة إلى المحور. إذا لم يتم تمرير المحور بشكل `concatenate()` الدالة صريح، فسيتم اعتباره 0.

مثال ↘ ↘

الانضمام إلى مصفوفتين

```
| import numpy as np
```

```
| Hosini = np.array([1, 2, 3])
```

```
| HabiB = np.array([4, 5, 6])
```

```
| arr = np.concatenate((Hosini, HabiB))
```

| `print(arr)`

مثال ↘ ↘

ضم مصفوفتين ثنائي الأبعاد على طول المصفوفة (المحور = 1)

| `import numpy as np`

| `Hosini = np.array([[1, 2], [3, 4]])`

| `HabiB = np.array([[5, 6], [7, 8]])`

| `arr = np.concatenate((Hosini, HabiB), axis=1)`

| `print(arr)`

الإنضمام إلى المصفوفات باستخدام دوال الفهرس

القراص او التابع هو نفس المصفوفات، والفرق الوحيد هو أن القراص او التابع يتم على طول محور جديد.

نستطيع تسلسل مصفوفتين أحاديي الأبعاد على طول المحور الثاني مما قد يؤدي إلى وضعهما فوق الآخر، أي. القراص أو التابع

الدالة مع **stack()** نقوم بتمرير نصوص من المصفوفات التي نريد ضمها إلى المحور. إذا لم يتم تمرير المحور بشكل صريح فسيتم اعتباره 0

مثال ↘ ↘

```
| import numpy as np
```

```
| Hosini = np.array([1, 2, 3])
```

```
| Habib = np.array([4, 5, 6])
```

```
| arr = np.stack((Hosini, Habib), axis=1)
```

```
| print(arr)
```

القراص أو التابع أو التابع على طول المصفوفة

الاضافة أو التابع على طول **hstack()** دالة مساعدة NumPy يوفر المصفوفة.

مثال ↘ ↘

```
| import numpy as np  
| Hosini = np.array([1, 2, 3])  
| HabiB = np.array([4, 5, 6])  
| arr = np.hstack((Hosini, HabiB))  
| print(arr)
```

التراص او التتابع على طول الأعمدة

التتابع على طول الأعمدة **vstack()** دالة مساعدة NumPy يوفر

مثال ↘ ↘

```
| import numpy as np  
| Hosini = np.array([1, 2, 3])  
| HabiB = np.array([4, 5, 6])  
| arr = np.vstack((Hosini, HabiB))  
| print(arr)
```

التراص أو التابع على طول الارتفاع (للعمق)

الاضافة على طول الارتفاع، وهو نفس `dstack()` دالة مساعدة NumPy يوفر العمق.

مثال ↘ ↘

```
| import numpy as np
| Hosini = np.array([1, 2, 3])
| Habib = np.array([4, 5, 6])
| arr = np.dstack((Hosini, Habib))
| print(arr)
```

NumPy التقسيم

NumPy تقسيم المصفوفات

التقسيم هو عملية عكسية للانضمام.

يؤدي الانضمام إلى دمج مصفوفات كثيرة في مصفوفة واحدة ويقوم التقسيم بتقسيم مصفوفة واحدة إلى عدة

لتقسيم المصفوفات، ونمرر لها المصفوفة التي نريد `array_split()` نستخدم تقسيمها وعدد الانقسامات

مثال ↘ ↘

تقسيم المصفوفة إلى 3 أجزاء

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 6])
```

```
| Habib_Array = np.array_split(arr, 3)
```

```
| print(Habib_Array)
```

ملاحظة: القيمة المرجعة هي مصفوفة تحتوي على ثلاث مصفوفات

إذا كانت المصفوفة تحتوي على عناصر أقل من المطلوب، فسيتم تعديلها من النهاية وفقاً لذلك

مثال ↘ ↘

تقسيم المصفوفة إلى 4 أجزاء

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5, 6])
```

```
Habib_Array = np.array_split(arr, 4)
```

```
print(Habib_Array)
```

المتاحة ولكنها لن تقوم بضبط العناصر **split()** ملاحظة: لدينا أيضًا الدالة عندما تكون العناصر أقل في مصفوفة المصدر للتقسيم كما في **مثال** ↘ ↘ قد تفشل **split()** عملت بشكل صحيح ولكنها **array_split()** أعلاه، وقد

تقسيم إلى مصفوفات صغيرة

هي مصفوفة تحتوي على كل قسم **array_split()** القيمة المرجعة للدالة كمصفوفة.

إذا قمت بتقسيم مصفوفة إلى ثلاث مصفوفات، فتستطيع الوصول إليها تمامًا: مثل أي عنصر مصفوفة

مثال ↘ ↘

:الوصول إلى المصفوفات المقسمة

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5, 6])
```

```
Habib_Array = np.array_split(arr, 3)
```

```
print(Habib_Array[0])
```

```
print(Habib_Array[1])
```

```
print(Habib_Array[2])
```

تقسيم المصفوفات ثنائية الأبعاد

استخدم نفس بناء الجملة عند تقسيم المصفوفات ثنائية الأبعاد

الدالة، وقم بتمرير المصفوفة التي تريد تقسيمها وعدد `array_split()` استخدم الانقسامات التي تريد القيام بها.

مثال ↘ ↘

قم بتقسيم المصفوفة ثنائية الأبعاد إلى ثلاث مصفوفات ثنائية الأبعاد

```
import numpy as np
```

```
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
```

```
Habib_Array = np.array_split(arr, 3)
```

```
print(Habib_Array)
```

يُرجع مثال \ \ أعلاه ثلاث مصفوفات ثنائية الأبعاد

لننظر إلى مثال \ \ آخر، هذه المرة يحتوي كل عنصر في المصفوفات ثنائية الأبعاد على 3 عناصر

مثال \ \

قم بتقسيم المصفوفة ثنائية الأبعاد إلى ثلاث مصفوفات ثنائية الأبعاد

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12],  
| [13, 14, 15], [16, 17, 18]])
```

```
| Habib_Array = np.array_split(arr, 3)
```

```
| print(Habib_Array)
```

يُرجع مثال \ \ أعلاه ثلاث مصفوفات ثنائية الأبعاد

بالإضافة إلى ذلك، تستطيع تحديد المحور الذي تريد إجراء التقسيم حوله

يعرض مثال \ \ أدناه أيضًا ثلاث مصفوفات ثنائية الأبعاد، ولكنها مقسمة على طول الصف (المحور = 1)

مثال ↘ ↘

قم بتقسيم المصفوفة ثنائية الأبعاد إلى ثلاث مصفوفات ثنائية الأبعاد على طول المصفوفة.

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12],  
| [13, 14, 15], [16, 17, 18]])
```

```
| Habib_Array = np.array_split(arr, 3, axis=1)
```

```
| print(Habib_Array)
```

`hstack()` عكس `hsplit()` الحل البديل هو استخدام

مثال ↘ ↘

الدالة لتقسيم المصفوفة ثنائية الأبعاد إلى ثلاث مصفوفات `hsplit()` استخدم ثنائية الأبعاد على طول المصفوفة.

```
| import numpy as np
```

```
| arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12],  
| [13, 14, 15], [16, 17, 18]])
```

```
| Habib_Array = np.hsplit(arr, 3)
```


| `print(Habib_Array)`

لـ `dsplit()` `vsplit()` `dstack()` و `vstack()` ملاحظة: تتوفر بدائل مماثلة لـ

NumPy البحث ✓

البحث في مصفوفة

تستطيع البحث في مصفوفة عن قيمة معينة، وإرجاع الأرقام التي تحصل على تطابق.

الدالة `where()` للبحث في مصفوفة، استخدم

مثال ↘ ↘

:ابحث عن الأرقام التي تكون قيمتها 4

```
| import numpy as np
```

```
| arr = np.array([1, 2, 3, 4, 5, 4, 4])
```

```
| x = np.where(arr == 4)
```

| `print(x)`

مثال ↘ ↘ أعله سوف يُرجع صفًا
مما يعني أن القيمة 4 موجودة في الرقم 3 و5 و6

مثال ↘ ↘

:ابحث عن الأرقام التي تكون فيها القيم زوجية

| `import numpy as np`

| `arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])`

| `x = np.where(arr%2 == 0)`

| `print(x)`

مثال ↘ ↘

:ابحث عن الأرقام التي تكون فيها القيم فردية

| `import numpy as np`

| `arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])`

```
x = np.where(arr%2 == 1)
```

```
print(x)
```

قم بفرز البحث

والتي تقوم بإجراء بحث ثنائي في `searchsorted()` هناك دالة تسمى المصفوفة، وتقوم بإرجاع الرقم حيث سيتم إدراج القيمة المحددة للحفاظ على ترتيب البحث.

من المفترض أن يتم استخدام هذه الدالة في المصفوفات `searchsorted()` التي تم فرزها.

مثال ↙ ↘

ابحث عن الأرقام التي تم إدراج القيمة 7 فيها:

```
import numpy as np
```

```
arr = np.array([6, 7, 8, 9])
```

```
x = np.searchsorted(arr, 7)
```

```
print(x)
```

توضيح مثال ↘ ↘ : يجب إدراج الرقم 7 في الرقم 1 ليظل ترتيب الفرز تبدأ الدالة البحث من اليسار وترجع الرقم الأول حيث لم يعد الرقم 7 أكبر من القيمة التالية.

البحث من الجانب الأيمن
افتراضياً، يتم إرجاع الرقم الموجود في أقصى اليمين، ولكن نستطيع إرجاع الرقم الموجود في أقصى اليمين بدلاً من ذلك **side='right'**

مثال ↘ ↘

ابحث عن الأرقام التي يجب إدراج القيمة 7 فيها، بدءاً من اليمين

```
| import numpy as np  
| arr = np.array([6, 7, 8, 9])  
| x = np.searchsorted(arr, 7, side='right')  
| print(x)
```

توضيح مثال ↘ ↘ : يجب إدراج الرقم 7 في الرقم 2 ليظل ترتيب الفرز تبدأ الدالة البحث من اليمين وترجع الرقم الأول حيث لم يعد الرقم 7 أقل من القيمة التالية.

قيم كثيره

للبحث عن أكثر من قيمة، استخدم مصفوفة بالقيم المحددة

مثال ↘ ↘

ابحث عن الارقام التي يجب إدراج القيم 2 و4 و6 فيها

```
| import numpy as np
```

```
| arr = np.array([1, 3, 5, 7])
```

```
| x = np.searchsorted(arr, [2, 4, 6])
```

```
| print(x)
```

القيمة المرجعة هي مصفوفة: [3 2 1] تحتوي على الارقام الثلاثة حيث سيتم إدراج 2، 4، 6 في المصفوفة الأصلية للحفاظ على الترتيب

NumPy مصفوفات الفرز

فرز المصفوفات

• الفرز يعني وضع العناصر في تسلسل مرتب

المصفوفات المرتبة هو أي تسلسل له ترتيب يتوافق مع العناصر، مثل الرقمنة التنسيقية أو الأبجدية، تصاعدياً أو تنازلياً.

والتي ستقوم بفرز **sort()** على دالة تسمى **NumPy ndarray** يحتوي كائن مصفوفة محددة.

مثال ↘ ↘

فرز المصفوفة:

```
| import numpy as np
```

```
| arr = np.array([3, 2, 0, 1])
```

```
| print(np.sort(arr))
```

ملاحظة: تقوم هذه الدالة بإرجاع نسخة من المصفوفة، مع ترك المصفوفة الأصلية دون تغيير.

تستطيع أيضاً فرز مصفوفات النصوص، أو أي نوع بيانات آخر

مثال ↘ ↘

فرز المصفوفة أبجدياً:

```
| import numpy as np
```

```
arr = np.array(['Habib', 'Al Hosiny', 'Abo'])
```

```
print(np.sort(arr))
```

مثال ↘ ↘

فرز مجموعة منطقية

```
import numpy as np
```

```
arr = np.array([True, False, True])
```

```
print(np.sort(arr))
```

فرز مصفوفة ثنائية الأبعاد

إذا كنت تستخدم دالة الترتيب () على مصفوفة ثنائية الأبعاد، فسيتم فرز كلا المصفوفتين:

مثال ↘ ↘

فرز مصفوفة ثنائية الأبعاد

```
import numpy as np
```

```
| arr = np.array([[3, 2, 4], [5, 0, 1]])
```

```
| print(np.sort(arr))
```

أبو حبيب الحسيني

أبو حبيب الحسيني

أبو حبيب
الحسيني



التكملة في الجزء الثالث ان شاء الله تعالى