

الكافى

فى البايثون



الجزء الاول

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 بالمئة

PyQt5

تصميم وجهات تطبيقات عملاقة لبايثون فى ثوانى وتوليد
الاكواد اتوماتيكيا التطبيق معرب بالكامل



ابو حبيب الحسينى

أبو حبيب الحسيني

هذا الكتاب سيجعلك تصنع تطبيقات عملاقة
لبايثون في ثواني معدودة

أبو حبيب الحسيني

فَدَعَ الصَّبَا فَلَقَدْ عَدَاكَ زَمَانُهُ ** وَلَزَهْدُ فِعْمُرِكَ مَرَّ
مِنْهُ الأَطْيَبُ

ذهب الشبابُ فما له من عودةٍ ** وأتى المشيبُ فأين
منهُ المهربُ

دع عنك ما قد كان في زمن الصبا ** واذكر ذنوبك
وابكها يا مُذنبُ

واذكر مناقشة الحسابِ فإنه ** لا بدَّ يُحصي ما
جنيتَ ويكتبُ

لم ينسَهُ الملكانِ حينَ نسيتهُ ** بل أثبتاهُ
وأنتَ لاهِ تلعبُ

والروحُ فيك وديعةٌ أودعتها ** ستردُّها بالـرغمِ
منك وتُسلَبُ

وغرورُ دنياك التي تسعى لها ** دارٌ حقيقتها متاعٌ
يذهبُ

والليلُ فاعلمْ والنهارُ كلاهما ** أنفاسُنَا فيها تُعدُّ
وتُحسبُ

وجميعُ ما خلفتهُ وجمعتهُ ** حقاً يقيناً بعد موتك
يُنهبُ

تَبَّأ لدارٍ لا يدومُ نعيمُها ** ومشيدها عمَّا قليلٍ
يَخربُ

فاسمعْ هُدَيْتَ نَصِيحَةً أَوْلَاكُهَا** بَرِّ نَصُوْحٌ لِلْأَنْسَامِ
مُجْرَبٌ

صَحِبَ الزَّمَانَ وَأَهْلَهُ مُسْتَبْصِرًا** وَرَأَى الْأُمُورَ بِمَا تَوُوبُ
وَتَعَقَّبُ

لَا تَأْمَنِ الدَّهْرَ يَوْمًا إِنَّهُ** مَا زَالَ قَدَمًا لِلرَّجَالِ
يُؤَدِّبُ

وَعَوَاقِبُ الْأَيَّامِ فِي غَصَاتِهَا** مَضُّ يُذَلُّ لَهُ الْأَعْرُ
الْأَنْجَابُ

فَعَلَيْكَ تَقْوَى الْمَلِكِ فَالزَّمْهَا تَفْرُ** إِنْ التَّقِيَّ هُوَ الْبَهِيُّ
الْأَهْيَبُ

وَاعْمَلْ بِطَاعَتِهِ تَنْلُ مِنْهُ الرِّضَا** إِنْ الْمَطِيْعَ لَهُ لَدِيهِ
مُقَرَّبُ

وَاقْنَعْ فِي بَعْضِ الْقِنَاعَةِ رَاحَةً** وَالْيَأْسُ مِمَّا فَاتَ فَهُوَ
الْمَطْلَبُ

فَلَقَدْ نَصَحْتُكَ إِنْ قَبِلْتَ نَصِيحَتِي** فَالْنُصْحُ أَعْلَى مَا يُبَاعُ
وَيُوهَبُ

ملحوظة

ان وجود الكلمات الانجليزية فى وسط الجمل العربيه قد ينقل بعض الكلمات من مكانها فتظهر الجمل بشكل غير صحيح فقد قمنا بجعل اتجاه الكتاب من اليسار الى اليمين لتفادى هذا الامر ورغم ذلك لم يعالج بنسبة 100%

أبو حبيب الحسينى

فهرس الكتاب

ملحوظة مهمة جدا.....	2
فهرس الكتاب.....	9
نبذة قصيره عن لغة بايثون.....	15
تحميل بايثون.....	17
بناء جملة بايثون مقارنة بلغات البرمجة الأخرى.....	19
تثبيت بايثون.....	20
تشغيل كود بايثون من ملف.....	21
سطر أوامر بايثون.....	22
بناء جملة بايثون.....	28
تنفيذ بناء جملة بايثون.....	28
المسافة البادئة فى بايثون.....	28
متغيرات بايثون.....	30
تعليقات.....	31
تعليقات بايثون.....	32
إنشاء تعليق.....	32
تعليقات تجمع سطور كثيره.....	33
متغيرات بايثون.....	35
المتغيرات.....	35
إنشاء المتغيرات.....	35
تحويل انواع المتغيرات او تحديد نوعها من البداية.....	36
نوع المتغير.....	36
اقتباسات مفردة أم مزدوجة؟.....	37
حساسية الحالة.....	38
كيف الاختيار لاسماء المتغيرات.....	38
الاسماء الوصفية للمتغيرات.....	40

حالة الجمل.....	40
تعين قيم عديدة لمتغيرات كثيرة.....	41
كيف انشاء قيمة واحدة لمتغيرات كثيرة.....	41
تجزئة وتفكيك مصفوفة او مجموعة الى متغيرات.....	42
الإخراج او الطباعة.....	43
المتغيرات العامة.....	45
مفهوم المتغيرات العامة.....	45
تابع المتغير العام.....	47
أنواع البيانات فى بايثون.....	48
شرح أنواع الاجرائات.....	48
هذه الدالة للحصول على نوع البيانات.....	49
تحديد نوع البيانات.....	50
تحويل نوع البيانات اوالمتغيرات المحدده.....	51
الارقام الصحيحة.....	53
الارقام العشرية.....	54
الاعداد المركبة.....	55
نوع التحويل.....	56
رقم عشوائي.....	57
التحويل الرقمى فى بايثون.....	57
تحديد نوع متغير.....	58
التعامل مع النصوص فى بايثون.....	59
تابع نصوص.....	60
تعيين نصوص إلى متغير.....	60
اقتباسات النصوص متعددة الاسطر.....	61
التعامل مع النصوص كا مصفوفات من الحروف.....	62
عمل حلقة على نصوص.....	63
ايجاد طول النص.....	63

تحقق من وجود كلمة فى النص.....	64
كيف وضع شرط.....	64
تحقق إذا لم يكن الشرط محقق.....	65
جلب العناصر من البداية.....	66
جلب العناصر حتى النهاية.....	66
استخدام الارقام السالبة.....	67
كيف تعديل النصوص.....	67
التحويل الى الأحرف الكبيرة.....	68
التحويل إلى احرف صغيرة.....	68
إزالة المسافة.....	69
استبدال النصوص.....	69
تقسيم النصوص.....	70
العلامات المفردة والمزدوجة.....	70
تعيين نصوص إلى متغير.....	71
طرق التقطيع.....	72
تعديل النصوص.....	72
الأحرف الكبيرة.....	73
أحرف صغيرة.....	73
إزالة المسافة.....	74
تابع استبدال النصوص.....	74
التجزئة والتقسيم.....	75
دمج النصوص.....	75
التنسيق - للنصوص.....	76
كيف تنسيق النصوص.....	77
اثثناء احرف من النصوص.....	79
موضع الاثثناء.....	80
القيم المنطقية.....	80

تقييم البيانات.....	82
المعاملات الرياضية.....	82
انواع المعاملات الرياضية.....	82
معاملات بايثون الحسابية.....	83
معاملات تعيين بايثون.....	83
معاملات المقارنة بايثون.....	84
عوامل التشغيل المنطقية في بايثون.....	85
معاملات منطقية.....	85
معاملات البحث.....	86
Bitwise معاملات بايثون.....	86
قوائم بايثون.....	87
قائمة.....	87
التعامل مع القائمة.....	88
ترتيب القوائم.....	88
القائمة قابله للتغيير.....	88
السماح بالتكرارات.....	89
تحديد طول القائمة يعنى عدد العناصر.....	89
عناصر القائمة و أنواع الاجرائات.....	90
تعريف القائمة.....	91
انشاء القوائم.....	91
مجموعات بايثون او (المصفوفات).....	92
الوصول الى عناصر القائمة.....	93
طرق الوصول.....	93
استخدام الارقام السالبة.....	94
نطاق الارقام.....	94
العمل ب الارقام السالبة.....	96
تحقق من وجود العنصر.....	97

تغيير قيمة العنصر.....	98
ادراج او تغيير العناصر.....	98
إدراج عناصر.....	100
تابع إضافة عناصر القائمة.....	101
تابع ادراج العناصر.....	101
مثال اخر تابع إدراج عناصر.....	102
توسيع القائمة.....	103
الاضافات قابلة للنسخ.....	103
إزالة عناصر القائمة.....	104
كيف إزالة العنصر المحدد.....	104
إزالة العنصر المحدد.....	105
امسح القائمة.....	106
الحلقات على القوائم.....	107
حلقة بسيطة على القائمة.....	107
طباعة جميع العناصر الموجودة في القائمة، واحداً تلو الآخر:.....	107
الحلقة عن طريق رقم العنصر.....	108
while استخدام حلقة.....	108
for in النسخ باستخدام.....	109
تابع for in حلقة.....	110
بناء الجملة.....	111
الدالة لإنشاء تكرار: range() تستطيع استخدام.....	113
فرز او ترتيب القوائم.....	115
فرز القائمة أبجديا او عدديا.....	115
ترتيب تنازلي.....	116
تخصيص دالة الفرز.....	117
فرز غير حساس لحالة الأحرف.....	118

ترتيب عكسي.....	119
نسخ القوائم.....	119
انسخ القائمة.....	120
ضم القوائم.....	121
كيف تضم قائمتين.....	121
دوال القائمة.....	123
دوال القائمة.....	123
تعمق فى ال Tuple.....	124
كيف التعامل مع العناصر.....	125
غير قابل للتغيير.....	125
السماح بالتكرارات.....	125
طول المصفوفة.....	126
بعنصر واحد Tuple إنشاء.....	126
التحكم فى الاجرائات - Tuple عناصر.....	127
دالة انشاء الجداول Tuple().....	129
مجموعات بايثون (المصفوفات).....	129
تفريغ Tuple.....	130
باستخدام النجمة*.....	132
الحلقة على المصفوفة.....	133
حلقة الاولى Tuple.....	133
حلقة من خلال رقم العنصر.....	134
while استخدام حلقة.....	135
ضم المصفوفات او الجدوال كما فى القوائم تمام.....	136
ضرب المصفوفة.....	136
دوال Tuple.....	137
اهم دوال المصفوفة.....	137
مجموعات بايثون.....	137

تعيين.....	138
تعيين العناصر.....	139
غير مرتبة.....	139
غير قابل للتغيير.....	139
التكرارات غير المسموح بها.....	139
احصل على طول المجموعة.....	140
مجموعة العناصر - أنواع الاجرائات.....	141
14 لإنشاء مجموعة. set() من الممكن أيضًا استخدام مُنشئ 2	
عناصر الوصول.....	143
144 الدالة. add() لإضافة عنصر واحد إلى مجموعة استخدم	
إضافة عناصر من مجموعة أخرى إلى المجموعة الحالية،	
145 الدالة. update() استخدم	
146 الدالة update	
remove() لإزالة عنصر في مجموعة، استخدم	
147 الدالة. disHosini_Datd() أو	
148 الدالة DisHosini_Datd	
149 الدالة: pop() قم بإزالة العنصر الأخير باستخدام	
150 الحلقات على المجموعات	
150 عناصر الحلقة	
151 الدالة union	
152 جلب التتابع	
154 جلب الاختلاف	
155 تعيين الأساليب والفنكشن	
157 قاموس	
158 عناصر القاموس	
159 الترتيب	

قابل للتغيير.....	159
التكرارات غير مسموح بها.....	160
طول القاموس.....	160
أنواع البيانات فى عناصر القاموس -	161
طباعة نوع بيانات القاموس:	162
dict() منشئ القاموس.....	162
احصل على قيمة المفتاح :	163
احصل على المفاتيح.....	164
القيم.....	165
العناصر.....	167
تحقق من وجود مفتاح.....	169
تغيير عناصر القاموس.....	170
تغيير القيم.....	170
إضافة عناصر القاموس.....	171
إضافة العناصر.....	171
تحديث القاموس.....	172
إزالة العناصر.....	173
الحلقات على القواميس.....	176
حلقة بسيطة على القاموس.....	176
نسخ القواميس.....	178
انسخ قاموسًا.....	178
القواميس المتداخلة.....	179
قواميس متداخلة.....	180
دوال قاموس بايثون.....	182
دوال القاموس.....	182
كيف وضع شروط.....	183
تكرار راعى المسافات البادئة فى بايثون.....	184

elif	185
اضافة شروط جديدة الى القديمة	188
كلمة او للخيار بين شرطين	189
عمل شروط متداخلة	190
كلمة المرور	190
الحلقات	191
حلقة بينما	191
كلمة الاستراحة	192
كلمة الاستمرار	193
اذا تحقق الشرط افعل كذا والا افعل كذا	193
ضوابط للحلقات	194
نستطيع إيقاف الحلقة قبل أن Break باستخدام عبارة يتم تكرارها عبر جميع العناصر:	195
نستطيع إيقاف النسخ الحالي continue باستخدام عبارة للحقة، والاستمرار في العبارة التالية:	196
range () دالة	197
حلقات متداخلة	200
كلمة المرور على حلقة فور	201

نبدا باذن الله تعالى

نبذة قصيره عن لغة بايثون

يجب ان تعلم اخی القارئ انك محظوظ مرتين المرة الاولى لانك صادفت هذه السلسله التي ستأخذك الى عالم بايثون من البداية حتى الاحتراف والمرة الثانية لانك تتعلم هذه اللغة الرائعة و الرائغة فى المجال فان مميزات بايثون لا تعد ولا تحصى سنذكر القليل منها الان باذن الله تعالى ان لغة بايثون تتقدم بشكل غير طبيعى على كل لغات العالم فى استمرار مدهش لم يحدث قبل ذلك والاكثر إستخداما من سنة 2017 حتى الان واكثر لغة تمت الهجرة اليها من لغات اخرى فى العالم وتصنف بايثون بنها ثانى اسهل لغة فى العالم بعد لغة روبى مع العلم ان بايثون تتفوق على روبى فى الاداء وفى الشهرة وفى عدد المكتبات والدعم الفنى وعدد المستخدمين وفى مجال الذكاء الاصطناعى ومجالات اخرى كثيرة ويتوقع خبراء البرمجة فى العالم ان تتصدر بايثون جميع اللغات فى ما بعد فقد جمعت بايثون بين سهوله واختصار ديلفى وبيسكال فى التكويد وقوة وسيطرة السي بلس فى الاداء ، وهذه كل مميزات اللغات جمعتها لك بايثون فى لغة واحدة كل من التصنيفات لكل المصادر اكدت ان المستقبل للغة بايثون ولا شك فى ذلك ، فلغة بايثون قد قدمت الكثير الى عالم البرمجة وخدمة

المطور، اكثر من اى لغة اخرى فلقد درست انا جميع لغات الدت نت من ميكروسوفت حتى لغات البيسك القديمة درستها فيجوال بيسك 6 و 4 واشهد ان بايثون اكثر لغة تخدم المطور بشكل رائع وتسهل عليه الامور فلولا انها سهلة الاستخدام و بسيطة جدا لما هاجر اليها مئات الاف من المطورين حول العالم لتصبح بايثون الاكثر استخداما من 2017 حتى الان ، و البدئ بها لتعلم المبادئ البرمجية إن كنت مبتدئ في المجال من أساسه ، البايثون يتم إستخدامها اليوم في مجموعة كبيرة من المجالات ، فى برمجة الكمبيوتر وانظمة التشغيل بكل انواعها و مجالات أيضا ، خاصة بالعدد والالات يتم إستخدام بايثون في مجال الذكاء الاصطناعي ، و مجال ال شبكات، و كذلك التحكم فى البيانات الضخمة ، والسيرفرات و يكفيك ان وكالة ناسا الفضائية تعتمد على لغة بايثون اعتماد شبه كلى ولذلك ، لغة البرمجة بايثون ، مرنة و كثيره الاستخدامات فقد اقتحمة بايثون كل التخصصات تقريبا ، و إن تتعلم البرمجة اليوم ، فإختيارك للغة بايثون ليس بخيار سيئ ، فهي أولا و أخيرا واحدة من اللغات البرمجية الاكثر شهره واستخداما فى كبرى الشركات العامه .

إذا كنت من مستخدمي نظام لينكس فلن تحتاج الى تنصيب بايثون على جهازك ما عليك سوى فتح الترمينال وكتابة كلمة

python3

تحميل بايثون

بضغط واحدة قم بتنزيل بايثون مجاناً من موقع بايثون الرسمي

<https://Python.org>

مستخدمي لينكس لا تفعل هذا فان بايثون 3 وبايثون 2 مدمجين في اى توزيعه لينكس اى كانت نوعها او اسمها ما عليك سوى فتح التيرمينال وكتابة كلمة **python3**

والان اكواد بايثون تعمل على نظام لينكس تستطيع استخدامها بدلا من التيرمينال او انشا ملف بامتداد بي واى واضغط عليه كليك يمين واختر له **python3** برنامج التشغيل

والان ملفات بايثون تعمل على نظام لينكس باكلمة بضغط زر واحدة على الملف يتم تنفيذ الاكواد التى بداخل الملف

بداية تستطيع استخدام اى محرر جربه بنفسك لن نفرض عليك محرر بعينه لان اى محرر سيعطيك نفس النتيجة ان شاء الله تعالى، تستطيع تعديل كود بنفس الطريقة وعرض الكود على اى نظام تشغيل واى محرر والنتيجة واحدة.

المنصات التى تعمل عليها

- **Mac، Windows** تعمل لغة بايثون على منصات مختلفة
- وما إلى ذلك، **Raspberry Pi، Linux** و
- فى لغة بايثون بناء جملة بسيط مشابه للغة الإنجليزية
- فى لغة بايثون بناء جملة يسمح للمطورين بكتابة برامج ذات أسطر أقل من بعض لغات البرمجة الأخرى
- تعمل بايثون على نظام مترجم فوري، مما يعني أنه يمكن تنفيذ الاكواد بمجرد كتابتها. وهذا يعني أن النماذج الأولية يمكن أن تكون سريعة جدًا
- يمكن التعامل مع بايثون بدالة إجرائية، أو بدالة موجهة للكائنات، أو بدالة وظيفية

- أحدث إصدار رئيسي من بايثون هو بايثون 3، والذي سنستخدمه في هذا الكتاب. ومع ذلك، فإن بايثون 2، على الرغم من عدم تحديثها بأي شيء آخر غير التحديثات الأمنية، لا تزال تحظى بشعبية كبيرة.
- في هذا الكتاب، سيتم كتابة بايثون في محرر النصوص. من الممكن أو **Thonny** كتابة لغة بايثون في بيئة تطوير متكاملة، مثل والتي تكون مفيدة **Eclipse** أو **Netbeans** أو **Pycharm** بشكل خاص عند إدارة مجموعات أكبر من ملفات بايثون.

بناء جملة بايثون مقارنة بلغات البرمجة الأخرى

- تم تصميم بايثون لسهولة القراءة، ولها بعض أوجه التشابه مع اللغة الإنجليزية مع تأثير الرياضيات.
- تستخدم بايثون أسطرًا جديدة لإكمال الأمر، على عكس لغات البرمجة الأخرى التي غالبًا ما تستخدم الفواصل المنقوطة أو الأقواس.
- تعتمد بايثون على المسافة البادئة، باستخدام المسافة، لتحديد النطاق؛ مثل نطاق الحلقات والدوال والكلاسات. غالبًا ما تستخدم لغات البرمجة الأخرى الأقواس المتعرجة لهذا الغرض.

مثال ↘ ↘

```
print("Abo Habib Al-Hosiny !")
```

```
#Abo Habib Al-Hosiny !
```


أبو حبيب الحسيني

تثبيت بايثون

سيتم تثبيت لغة بايثون بالفعل على العديد من أجهزة الكمبيوتر Mac النصوص وأجهزة

للتحقق مما إذا كان موجود بايثون مثبتًا على جهاز كمبيوتر يعمل بنظام
ابحث في شريط البداية عن بايثون أو قم بتشغيل ما يلي، **Windows**
(**cmd.exe**) في سطر الأوامر:

C:\Hosini_Data\Your Name>Python --version

ثم **Mac** أو **Linux** للتحقق مما إذا كان موجود بايثون مثبتاً على نظام
نواكتب **Terminal** افتح **Mac** افتح سطر الأوامر أو على **Linux** على

Python-version

إذا وجدت أنه ليس موجود لغة بايثون مثبتة على جهاز الكمبيوتر،
فتستطيع تنزيلها مجاناً من الموقع
التالي: <https://www.Python.org/>

تشغيل كود بايثون من ملف

بايثون هي لغة برمجة مفسرة، وهذا يعني أنك كمطور تكتب ملفات بايثون
في محرر نصوص ثم تضع تلك الملفات بمتداد مترجم بايثون ليتم (.py)
تنفيذها.

دالة تشغيل ملف بايثون هي كالتالي في سطر الأوامر

C:\Hosini_Data\Your Name>Python Abo_Habib.py

. هو اسم ملف بايثون "Abo_Habib.py" حيث

والذي يمكن إجراؤه **Abo_Habib.py** لنكتب أول ملف بايثون ، يسمى باستخدام أي محرر نصوص.

اسم الملف **Abo_Habib.py**

```
print("Abo Habib Al-Hosiny !")
```

سهل هكذا. **احفظ الملف** . افتح سطر الأوامر، وانتقل إلى الدليل الذي قمت بحفظ ملفك فيه، وقم بتشغيل:

```
C:\Hosini_Data\Your Name>Python\Abo_Habib.py
```

:يجب أن يكون الناتج كما يلي

```
Abo Habib Al-Hosiny !
```

.تهانينا، لقد قمت بكتابة وتنفيذ أول برنامج بايثون لك

سطر أوامر بايثون

لاختبار اسطر قصيرة من الاكواد في بايثون، في بعض الأحيان يكون من الأسرع والأسهل عدم كتابة الاكواد في ملف. أصبح هذا ممكناً لأنه يمكن تشغيل بايثون كسطر أوامر بحد ذاته.

Linux أو **Mac** أو **Windows** اكتب ما يلي في سطر أوامر

C:\Hosini_Data\Your Name>Python

"py": أو، إذا لم يعمل الأمر ، تستطيع تجربة

C:\Hosini_Data\Your Name>py

من هناك تستطيع كتابة أي كود بايثون، بما في ذلك **مثال** \ \ نا الذي
:سبق ذكره في الكتاب

C:\Hosini_Data\Your Name>Python

**Python(v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC
v.1900 32 bit (Intel)] on win32**

**Type "help", "copyright", "credits" or "license" for
more information.**

>>> print("Abo Habib Al-Hosiny !")

Abo Habib Al-Hosiny !

:في سطر الأوامر **"Abo Habib Al-Hosiny !!!!"** والتي سوف تكتب

C:\Hosini_Data\Your Name>Python

**python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017,
06:04:45) [MSC v.1900 32 bit (Intel)] on win32**

**Type "help", "copyright", "credits" or "license" for
more information.**

>>> print("Abo Habib Al-Hosiny !")

Abo Habib Al-Hosiny !

عندما تنتهي من سطر أوامر بايثون، تستطيع ببساطة كتابة ما يلي للخروج
:من واجهة سطر أوامر بايثون

| **exit()**

قبل البدء تعرف على بعض الدوال العامة التي
سيتم شرح معظمها والباقي في الجزء الثانى

abs() تُرجع القيمة المطلقة لرقم ما

إذا كانت جميع العناصر الموجودة في كائن قابل **True** تُرجع **all()**

لنسخ صحيحة

إذا كان أي عنصر في كائن قابل للنسخ صحيحًا **True** تُرجع **Any()**

ascii يُرجع نسخة قابلة للقراءة من كائن ما. يستبدل أحرف (**ascii**) بحرف اثثناء

bin() يُرجع النسخة الثنائية لرقم

bool() تُرجع القيمة المنطقية للكائن المحدد

bytearray() يُرجع مصفوفة من البايتات

callable() إذا كان الكائن المحدد قابلاً **True** تعيد القيمة

False للاستدعاء، وإلا فإنها تعيد القيمة

المحدد **Unicode** تقوم بإرجاع حرف من رمز (**chr()**)

classmethod() يحول الدالة إلى دالة الكلاس

complex() نوع بيانات في بايثون يُرجع رقمًا مركبًا

delattr() يحذف السمة المحددة (الخاصية أو الدالة) من الكائن

المحدد

dict() انشاء و إرجاع قاموس (مصفوفة)

dir() يُرجع قائمة بخصائص وأساليب واكواد الكائن المحدد

divmod() يُرجع القسمة والباقي عند قسمة الوسيطة 1 على الوسيطة

2

enumerate() يأخذ مجموعة (على سبيل ال **مثال** صف) ويعيدها

ككائن تعداد

eval() يقوم بتقييم وتنفيذ التعبير

exec() ينفذ الكود (أو الكائن) المحدد

filter() استخدم دالة التصفية لاستبعاد العناصر الموجودة في كائن

قابل للنسخ

format() يقوم بتنسيق قيمة محددة داخل اقواس لدمج متغيرات الى

النص

Frozenset () نوع بيانات إرجاع كائن **Frozenset**

getattr() إرجاع قيمة السمة المحددة (خاصية أو دالة)
globals() تقوم بإرجاع جدول الرموز العام الحالي كقاموس
hasattr() إذا كان الكائن المحدد يحتوي على السمة **True** تُرجع (المحددة (الخاصية/الدالة)
hash () تُرجع قيمة التجزئة لكائن محدد
help() ينفذ نظام المساعدة المدمج
hex() يحول رقمًا إلى قيمة سداسية عشرية
id() يُرجع معرف الكائن

int() تقوم بتحويل البيانات و بإرجاع رقم صحيح
isinstance() إذا كان الكائن المحدد هو مثيل لكائن **True** يُرجع (محدد
محدد
إذا كانت الكلاس المحددة هي **True** يُرجع **issubclass()** التابع
كلاس فرعية لكائن محدد
iter() إرجاع كائن
len() يُرجع طول الكائن او عدد العناصر
list() تقوم بإرجاع قائمة و هو نوع بيانات شبيهة بالمصفوفة
locals() يُرجع قاموسًا محدثًا لجدول الرموز المحلي الحالي
Map () تُرجع المحدد مع تطبيق الدالة المحددة على كل عنصر
max() يُرجع أكبر عنصر في كائن قابل للنسخ
Memoryview() يُرجع كائن عرض الذاكرة
min() يُرجع أصغر عنصر في كائن قابل للنسخ
next() يُرجع العنصر التالي في كائن قابل للنسخ
object() إرجاع كائن جديد
oct() يحول الرقم إلى رقم ثماني

open() فتح ملف وإرجاع كائن ملف
ord() للحرف المحدد **Unicode** تحويل عدد صحيح يمثل
pow() ايجاد **y** إلى قوة **x** يُرجع قيمة
print() يطبع إلى جهاز الإخراج القياسي
property() يحصل على خاصية ويحددها ويحذفها
range() يُرجع نصوص من الأرقام، بدءًا من 0 ويزيد بمقدار 1 (افتراضيًا)
نسخة قابلة للقراءة من كائن ما **repr()** تعيد الدالة
round() لتقريب الأرقام
set() تقوم بإرجاع كائن مجموعة جديد
setattr () يعين سمة (خاصية/دالة) للكائن
staticmethod() يحول الدالة إلى دالة ثابتة
str() تقوم بإرجاع كائن نص
sum() يجمع عناصر
super() يُرجع كائنًا يمثل الكلاس الأصلية
tuple () إرجاع صف وهو نوع بيانات في بايثون
type() يُرجع نوع الكائن
vars() للكائن **__dict__** تُرجع الخاصية

✓ بناء جملة بايثون

تنفيذ بناء جملة بايثون

كما تعلمنا في الصفحة السابقة، يمكن تنفيذ بناء جملة بايثون عن طريق الكتابة مباشرة في سطر الأوامر:

```
>>> print("Abo Habib Al-Hosiny !")  
Abo Habib Al-Hosiny !
```

أو عن طريق إنشاء ملف بايثون على الخادم، باستخدام ملحق الملف وت تشغيله في سطر الأوامر، `.py`.

```
C:\Hosini_Data\Your Name>Abo_Habib_File.py
```

المسافة البادئة في بايثون

. تشير المسافة البادئة إلى المسافات في بداية سطر الاكواد

بينما في لغات البرمجة الأخرى تكون المسافة البادئة في الاكواد مخصصة
لسهولة القراءة فقط، فإن المسافة البادئة في بايثون مهمة جداً
. تستخدم بايثون المسافة البادئة للإشارة إلى كتلة من الاكواد

مثال ↘ ↘

```
if 5 > 2:
```

```
print("Abo Habib Al-Hosiny !")
```

:سوف تعطيك بايثون خطأ إذا تخطيت المسافة البادئة

مثال ↘ ↘

:خطأ في بناء الجملة

```
if 5 > 2:
```

```
print("Abo Habib Al-Hosiny !")
```

عدد المسافات متروك لك كمبرمج، والاستخدام الأكثر شيوعاً هو أربعة،
ولكن يجب أن تكون واحدة على الأقل

مثال ↘ ↘

```
if 5 > 2:
```

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



```
if 5 > 2:
```

```
    print("Al-Hosiny !")
```

يجب عليك استخدام نفس عدد المسافات في نفس كتلة الاكواد ، وإلا فسوف تعطيك بايثون خطأ

مثال ↘ ↘

خطأ في بناء الجملة

```
if 5 > 2:
```

```
    print("Abo Habib Al-Hosiny !")
```

```
    print("Abo Habib Al-Hosiny !")
```

متغيرات بايثون

في بايثون، يتم إنشاء المتغيرات عندما تقوم بتعيين قيمة لها

مثال ↘ ↘

:المتغيرات في بايثون

| x = 5

| y = "Abo Habib Al-Hosiny !"

.ليس في لغة بايثون أمر للاعلان عن متغير

تعليقات

.تتمتع بايثون بإمكانية التعليق لغرض التوثيق داخل الاكواد

:تبدأ التعليقات بـ #، وستعرض بايثون بقية السطر كتعليق

مثال ↘ ↘

:التعليقات في بايثون

| #This is a comment.

| print("Abo Habib Al-Hosiny !")

Abo Habib Al-Hosiny

✓ تعليقات بايثون

يمكن استخدام التعليقات لشرح كود بايثون.
يمكن استخدام التعليقات لجعل الاكواد أكثر قابلية للقراءة.
يمكن استخدام التعليقات لمنع التنفيذ عند اختبار الاكواد.

إنشاء تعليق

تبدأ التعليقات بـ #، وسوف تتجاهلها بايثون

مثال ↘ ↘

```
#This is a comment
```

```
print("Abo Habib Al-Hosiny !")
```

```
Abo Habib Al-Hosiny
```

يمكن وضع التعليقات في نهاية السطر، وستتجاهل بايثون بقية السطر

مثال ↘ ↘

```
print("Abo Habib Al-Hosiny !") #This is a comment
```

| //Abo Habib Al-Hosiny

ليس من الضروري أن يكون التعليق نصًا يشرح الكود، ويمكن استخدامه أيضًا لمنع بايثون من تنفيذ الكود:

مثال ↘ ↘

| #print("Abo Habib Al-Hosiny !")

| print("Al-Hosiny !*")

| //Al-Hosiny

تعليقات تجمّع سطور كثيره

ليس في لغة بايثون حقًا صيغة للتعليقات كثيره الأسطر.

لإضافة تعليق متعدد الأسطر، تستطيع إدراج حرف # لكل سطر

مثال ↘ ↘

| #This is a comment

| #written in

| #more than just one line

| print("Abo Habib Al-Hosiny !")

Abo Habib Al-Hosiny

أو، ليس تمامًا كما هو مقصود، تستطيع استخدام نصوص كثيرة الأسطر بما أن بايثون ستتجاهل القيم الحرفية للنص التي لم يتم تعيينها لمتغير، فتستطيع إضافة نصوص كثيرة الأسطر (علامات الاقتباس الثلاثية) في الكواد ، ووضع تعليقك بداخلها

مثال ↘ ↘

=====

هذا تعليق متعدد الاسطر

اذا لم تقم بتعيين له اسم متغير فسيعتبره بايثون تعليق

ولكن اذا عينت له اسم متغير فسيصبح متغير نصي

=====

```
print("Abo Habib Al-Hosiny !")
```

Abo Habib Al-Hosiny

نعيد مرة اخرى للتاكيد طالما لم يتم تعيين النصوص لمتغير، ستقرأ بايثون الكود، لكنها تتجاهله بعد ذلك، وتكون قد قمت بعمل تعليق متعدد الأسطر.

متغيرات بايثون ✓

المتغيرات

. المتغيرات عبارة عن حاويات لتخزين قيم الاكواد

إنشاء المتغيرات

.ليس فى لغة بايثون أمر لاعلان عن متغير .
يتم إنشاء المتغير في اللحظة التي تقوم فيها بتعيين قيمة له لأول مرة

مثال ↘ ↘

```
x = 5
```

```
y = "Abo Habib"
```

```
print(x)
```

```
print(y)
```

لا يلزم التصريح عن المتغيرات بأي نوع معين ، وتستطيع حتى تغيير النوع بعد تعيينها

مثال ↘ ↘

```
x = 4 # x is of type int
x = "Habib_" # x is now of type str
print(x)
```

تحويل أنواع المتغيرات أو تحديد نوعها من البداية

إذا كنت تريد تحديد نوع بيانات المتغير، فيمكن القيام بذلك عن طريق الإرسال.

مثال ↘ ↘

```
x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3) # z will be 3.0
```

نوع المتغير

`type()` تستطيع نوع بيانات المتغير باستخدام الدالة.

مثال ↘ ↘

```
x = 5
```

```
y = "Abo Habib"
```

```
print(type(x))
```

```
print(type(y))
```

أقتباسات مفردة أم مزدوجة؟

يمكن تحديد متغيرات النصوص إما باستخدام علامات الاقتباس المفردة أو المزدوجة:

مثال ↘ ↘

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

```
x = 'Habib'
```


حساسية الحالة

أسماء المتغيرات حساسة لحالة الأحرف

مثال ↘ ↘

:سيؤدي هذا إلى إنشاء متغيرين

```
a = 4
```

```
A = "Habib_"
```

```
#A will not overwrite a
```

أسم المتغير

كيف الاختيار لإسماء المتغيرات

أو اسم أكثر وصفاً (y و x مثل) يمكن أن يكون للمتغير اسم قصير (العمر، اسم السيارة، الحجم الإجمالي). قواعد متغيرات بايثون:

- يجب أن يبدأ اسم المتغير بحرف أو بشرطة سفلية
- لا يمكن أن يبدأ اسم المتغير برقم
- يمكن أن يحتوي اسم المتغير فقط على أحرف أبجدية رقمية (_ و 0-9، Az) وشرطات سفلية

- أسماء المتغيرات حساسة لحالة الأحرف (الثلاثة هي ثلاثة هي ثلاثة متغيرة مختلفة)

مثال ↘ ↘

أسماء المتغيرات الصحيحة:

```
myvar = "Abo Habib"  
my_var = "Abo Habib"  
_my_var = "Abo Habib"  
myVar = "Abo Habib"  
MYVAR = "Abo Habib"  
myvar2 = "Abo Habib"
```

مثال ↘ ↘

أسماء المتغيرات غير الصحيحة:

`2myvar = "Abo Habib"`

`my-var = "Abo Habib"`

`my var = "Abo Habib"`

تذكر أن أسماء المتغيرات حساسة لحالة الأحرف

الأسماء الوصفية للمتغيرات

قد يكون من الصعب قراءة الأسماء المتغيرة التي تحتوي على أكثر من كلمة لوصف المتغير بدلا من التعليق.

هناك العديد من التقنيات التي تستطيع استخدامها لجعلها أكثر قابلية للقراءة:

حالة الجمل

كل كلمة، باستثناء الأولى، تبدأ بحرف كبير

`myVariableName = "Abo Habib"`

يتم فصل كل كلمة باندرسكول

```
my_variable_name = "Abo Habib"
```

تعيين قيم عديدة لمتغيرات كثيرة

تتيح لك لغة بايثون تعيين قيم لمتغيرات كثيرة في سطر واحد

مثال ↘ ↘

```
x, y, z = "Al Badrashen", "Habib", "Al Hosiny"  
print(x)  
print(y)  
print(z)
```

ملاحظة: تأكد من أن عدد المتغيرات يطابق عدد القيم، وإلا فسوف تحصل على خطأ.

كيف انشاء قيمة واحدة لمتغيرات كثيرة

نوتستطيع تعيين نفس القيمة لمتغيرات كثيرة في سطر واحد

مثال ↘ ↘

```
x = y = z = "Al Badrashen"  
print(x)  
print(y)  
print(z)
```

تجزئة وتفكيك مصفوفة او مجموعة الى متغيرات

وما إلى ذلك **Tuple** إذا كان موجود مجموعة من القيم في قائمة، فإن . يسمح لك بايثون باستخراج القيم إلى متغيرات. وهذا ما يسمى التفريغ

مثال ↘ ↘

فك قائمة:

```
fruits = ["Abo", "Habib", "Al Hosiny"]  
x, y, z = fruits  
print(x) # Abo  
print(y) # Habib  
print(z) # Al Hosiny
```

الإخراج او الطباعة

. غالبًا لإخراج المتغيرات او النصوص بوجه عام `print()` تُستخدم دالة

مثال ↘ ↘

```
x = "python awesome"  
print(x)
```

:الدالة، تستطيع إخراج متغيرات كثيرة، مفصولة بفاصلة `print()` في

مثال ↘ ↘

```
x = "_Abo Habib Al-Hosiny _"  
y = "in"  
z = "Egypt"  
print(x, y, z)
```

: تستطيع أيضًا استخدام `+` لإخراج متغيرات كثيرة

مثال ↘ ↘

```
x = "_Abo Habib Al-Hosiny _"  
y = "in "  
z = "Egypt"  
print(x + y + z)
```

بالنسبة للأرقام، + يعمل الحرف كمشغل رياضي

مثال ↘ ↘

```
x = 5  
y = 10  
print(x + y)
```

الدالة، عندما تحاول دمج نصوص ورقم مع + ، ستعطيك `print()` في بايثون خطأ

مثال ↘ ↘

```
x = 5  
y = "Abo Habib"  
print(x + y)
```

الدالة هي الفصل بينها **print()** أفضل دالة لإخراج متغيرات كثيرة في
بفواصل، والتي تدعم أيضًا أنواع الإجراءات المختلفة

مثال ↘ ↘

```
x = 5
```

```
y = "Abo Habib"
```

```
print(x, y)
```

✓ المتغيرات العامة

مفهوم المتغيرات العامة

تُعرف المتغيرات التي يتم إنشاؤها خارج الدالة (كما في جميع الأمثلة
أعلاه) بالمتغيرات العامة.

يمكن للجميع استخدام المتغيرات العامة، سواء داخل الدوال أو خارجها.

مثال ↘ ↘

قم بإنشاء متغير خارج الدالة، واستخدمه داخل الدالة

```
x = "in Egypt"
```



```
def Hosini():
```

```
print( "_Abo Habib Al-Hosiny_" + x)
```

```
Hosini()
```

إذا قمت بإنشاء متغير بنفس الاسم داخل دالة، فسيكون هذا المتغير محليًا، ولا يمكن استخدامه إلا داخل الدالة. سيبقى المتغير الشامل الذي يحمل نفس الاسم كما كان، عامًا وبالقيمة الأصلية.

مثال ↘ ↘

قم بإنشاء متغير داخل دالة بنفس اسم المتغير العام

```
x = "Egypt"
```

```
def Hosini():
```

```
x = "fantastic"
```

```
print("python" + x)
```

```
Hosini()
```

```
print("python" + x)
```

تابع المتغير العام

عادةً، عندما تقوم بإنشاء متغير داخل دالة، يكون هذا المتغير محليًا، ولا يمكن استخدامه إلا داخل تلك الدالة.

الكلمة **global** لإنشاء متغير عام داخل دالة، تستطيع استخدام المحجوزة .

مثال ↘ ↘

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

```
def Hosini():
```

```
    global x
```

```
    x = "fantastic"
```

```
Hosini()
```

```
print("python" + x)
```

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

مثال ↘ ↘

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

```
x = "Egypt"
```

```
def Hosini():
```

```
    global x
```

```
    x = "fantastic"
```

```
Hosini()
```

```
print("python" + x)
```

✓ أنواع البيانات في بايثون

شرح أنواع الإجراءات

في البرمجة، نوع البيانات هو مفهوم مهم

يمكن للمتغيرات تخزين اكواد من أنواع مختلفة، ويمكن للأنواع
المختلفة القيام بأشياء مختلفة.

تحتوي لغة بايثون على أنواع الاجرائات التالية افتراضياً، في هذه الكلاسات:

إدخال النص: **str**
الأنواع الارقام: **int, float, complex**
أنواع المصفوفات: **list, tuple, range**
نوع التعيين: **dict**
أنواع المجموعة: **set, frozenset**
النوع المنطقي: **bool**
الأنواع الثنائية: **bytes, bytearray, _memoryview**
بلا نوع: **NoneType**

هذه الدالة للحصول على نوع البيانات

type() تستطيع نوع البيانات لأي كائن باستخدام الدالة

مثال ↘ ↘

x: اطبع نوع بيانات المتغير

```
x = 5
```

```
print(type(x))
```


تحديد نوع البيانات

في بايثون، يتم تعيين نوع البيانات عندما تقوم بتعيين قيمة لمتغير:

انواع البيانات	مثال ↘ ↙
str	x = "Abo Habib Al-Hosiny !!"
int	x = 20
float	x = 20.5
complex	x = 1j
list	x = ["Abo", "Habib", "Al Hosiny"]
tuple	x = ("Abo", "Habib", "Al Hosiny")
range	x = range(6)
dict	x = {"name": "Abo Habib", "age": 36}
set	x = {"Abo", "Habib", "Al Hosiny"}
frozenset	x = frozenset({"Abo", "Habib", "Al Hosiny"})
bool	x = True
bytes	x = b" Al Habib"
bytearray	x = bytearray(5)
memoryview	x = memoryview(bytes(5))

x = None	NoneType
-----------------	-----------------

تحويل نوع البيانات او المتغيرات المحدده

إذا كنت تريد تحديد نوع البيانات ، تستطيع استخدام دوال المنشئ التالية:

مثال ↘ ↘	Data Type
<code>x = str("Abo Habib Al-Hosiny !!")</code>	<code>str</code>
<code>x = int(20)</code>	<code>int</code>
<code>x = float(20.5)</code>	<code>float</code>
<code>x = complex(1j)</code>	<code>complex</code>
<code>x = list(("Abo", "Habib", "Al Hosiny"))</code>	<code>list</code>
<code>x = tuple(("Abo", "Habib", "Al Hosiny"))</code>	<code>tuple</code>
<code>x = range(6)</code>	<code>range</code>
<code>x = dict(name="Abo Habib", age=36)</code>	<code>dict</code>
<code>x = set(("Abo", "Habib", "Al Hosiny"))</code>	<code>set</code>
<code>x = frozenset(("Abo", "Habib", "Al Hosiny"))</code>	<code>frozenset</code>
<code>x = bool(5)</code>	<code>bool</code>

<code>x = bytes(5)</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>

أرقام في بايثون

هناك ثلاثة أنواع رقمية في بايثون:

- `int`
- `float`
- `complex`

يتم إنشاء متغيرات الأنواع الرقمية عندما تقوم بتعيين قيمة لها

مثال ↙ ↘

```
x = 1 # int
y = 2.8 # float
z = 1j # complex
```

type() للتحقق من نوع أي كائن في بايثون، استخدم الدالة

مثال ↘ ↘

```
print(type(x))  
print(type(y))  
print(type(z))
```

الأرقام الصحيحة

أو عدد صحيح، هو عدد صحيح، موجب أو سالب، بدون كسور، Int، عشرية، بطول غير محدود.

مثال ↘ ↘

الأعداد الصحيحة:

```
x = 1  
y = 35656222554887711  
z = -3255522
```

```
print(type(x))  
print(type(y))  
print(type(z))
```


الأرقام العشرية

رقم بين النقطة " هو رقم، موجب أو سالب، يحتوي على واحد أو أكثر "، من الكسور العشرية.

مثال ↘ ↘

:

```
x = 1.10
```

```
y = 1.0
```

```
z = -35.59
```

```
print(type(x))
```

```
print(type(y))
```

```
print(type(z))
```

للإشارة "e" الأرقام العشرية يمكن أيضاً أن يكون أرقاماً علمية بحرف إلى قوة الرقم 10.

مثال ↘ ↘

:

```
x = 35e3
```

```
y = 12E4
```

```
z = -87.7e100
```

```
print(type(x))
```

```
print(type(y))
```

```
print(type(z))
```

الأعداد المركبة

باعتباره الجزء التخيلي "z" تتم كتابة الأعداد المركبة بحرف

مثال ↙ ↘

```
x = 3+5j
```

```
y = 5j
```

```
z = -5j
```

```
print(type(x))
```

```
print(type(y))
```

```
print(type(z))
```


نوع التحويل

`float()` الدوال `int()` تستطيع التحويل من نوع إلى آخر باستخدام `complex()` و

مثال ↘ ↘

التحويل من نوع إلى آخر:

```
x = 1 # int
```

```
y = 2.8 # float
```

```
z = 1j # complex
```

```
#convert from int to float:
```

```
a = float(x)
```

```
#convert from float to int:
```

```
b = int(y)
```

```
#convert from int to complex:
```

```
c = complex(x)
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

```
print(type(a))
```

```
print(type(b))
```

```
print(type(c))
```

ملاحظة: لا تستطيع تحويل الأرقام المركبة إلى نوع أرقام آخر.

رقم عشوائي

دالة لإنشاء أرقام عشوائية، لكن بايثون `random()` ليس فى لغة بايثون والتي يمكن استخدامها لإنشاء أرقام `random` لديها وحدة مدمجة تسمى عشوائية:

مثال ↘ ↘

قم باستيراد الوحدة العشوائية، واعرض رقمًا عشوائيًا بين 1 و9

```
import random
```

```
print(random.randrange(1, 10))
```

✓ التحويل الرقمي فى بايثون

تحديد نوع متغير

قد تكون هناك أوقات تريد فيها تحديد نوع لمتغير. يمكن القيام بذلك عن طريق الوحدة. بايثون هي لغة موجهة للكائنات، وبالتالي فهي تستخدم الكلاسات لتحديد أنواع الاجراءات ، بما في ذلك أنواعها البدائية.

لذلك يتم إجراء عملية الوحدة في بايثون باستخدام دوال المنشئ:

- **int()** - إنشاء رقم صحيح من عدد صحيح حرفي، أو عدد صحيح حرفي (عن طريق إزالة جميع الكسور العشرية)، أو نصوص حرفية (بشرط أن تمثل النصوص عددًا صحيحًا)
- **float()** - ينشئ رقمًا عشريًا من عدد صحيح حرفي، أو عدد عشري حرفي، أو نصوص حرفية (بشرط أن تمثل النصوص عددًا عشريًا أو عددًا صحيحًا)
- **str()** - إنشاء نصوص من مجموعة واسعة من أنواع الاجراءات ، بما في ذلك النصوص والأعداد الصحيحة والقيم الحرفية العشرية

مثال ↘ ↘

الأعداد الصحيحة:

```
x = int(1) # x will be 1
```

```
y = int(2.8) # y will be 2
```

```
z = int("3") # z will be 3
```

مثال ↘ ↘

```
x = float(1) # x will be 1.0  
y = float(2.8) # y will be 2.8  
z = float("3") # z will be 3.0  
w = float("4.2") # w will be 4.2
```

مثال ↘ ↘

نصوص:

```
x = str("s1") # x will be 's1'  
y = str(2) # y will be '2'  
z = str(3.0) # z will be '3.0'
```

✓ التعامل مع النصوص في بايثون

تابع نصوص

النصوص في لغة بايثون محاطة بعلامات اقتباس مفردة أو علامات اقتباس مزدوجة.

'مرحب' هو نفس "مرحبا".

`print()`: تستطيع عرض نصوص حرفية باستخدام الدالة

مثال ↙ ↘

```
print(" Al Habib")
```

```
print('Hello')
```

تعيين نصوص إلى متغير

يتم تعيين نصوص إلى متغير باستخدام اسم المتغير متبوعاً بعلامة : يساوي والنص :

مثال ↙ ↘

```
a = " Al Habib"
```

```
print(a)
```

اقتباسات النصوص متعددة الأسطر

تستطيع تعيين نصوص كثيرة الأسطر لمتغير باستخدام ثلاث علامات اقتباس:

مثال ↘ ↘

تستطيع استخدام ثلاث علامات اقتباس مزدوجة

```
a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)
```

أو ثلاثة علامات اقتباس واحدة

مثال ↘ ↘

```
a = "Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."  
print(a)
```


ملحوظة:، يتم إدراج فواصل الأسطر في نفس الموضع كما في الكود

التعامل مع النصوص كا مصفوفات من الحروف

مثل العديد من لغات البرمجة الشائعة الأخرى، النصوص او الرقمية في بايثون عبارة عن مصفوفات من البايتات تمثل أحرف يونيكود ومع ذلك، لا تحتوي لغة بايثون على نوع بيانات للأحرف، فالحرف الواحد هو مجرد نصوص بطول 1 . يمكن استخدام الأقواس المربعة للوصول إلى عناصر النصوص

مثال ↘ ↘

احصل على الحرف في الموضع 1 (تذكر أن الحرف الأول له الموضع 0):

```
a = "Abo Habib Al-Hosiny !"  
print(a[1])
```

عمل حلقة على نصوص

نظرًا لأن النصوص عبارة عن مصفوفات، فنستطيع تكرار الأحرف بحلقة **for** الموجودة في نصوص باستخدام

مثال ↘ ↘

:"قم بالتمرير بين الحروف الموجودة في كلمة "موز

```
for x in "Habib":  
    print(x)
```

ايجاد طول النص

len() للحصول على طول النصوص، استخدم الدالة

مثال ↘ ↘

: طول النصوص **len()** ترجع الدالة

```
a = "Abo Habib Al-Hosiny !"  
print(len(a))
```


تحقق من وجود كلمة في النص

للتحقق من وجود عبارة أو حرف معين في نص، نستطيع استخدام الكلمة المحجوزة **in**.

مثال ↘ ↘

موجودة في النصوص التالي "free" تحقق مما إذا كانت كلمة

```
txt = "The best things in life are free!"  
print("free" in txt)
```

كيف وضع شرط

مثال ↘ ↘

"free": اطبع فقط في حالة وجود كلمة

```
txt = "The best things in life are free!"  
if "free" in txt:  
    print("Yes, 'free' is present.")
```

تحقق إذا لم يكن الشرط محقق

للتحقق من عدم وجود عبارة أو حرف معين في نصوص ما، نستطيع استخدام الكلمة المحجوزة **not in**.

مثال ↘ ↘

غير موجودة في "expensive" تحقق مما إذا كانت هذه الكلمة:
النصوص التالي:

```
txt = "The best things in life are free!"  
print("expensive" not in txt)
```

الكلمة **if** استخدمه في:

مثال ↘ ↘

: اطبع فقط في حالة عدم وجود كلمة **expensive**

```
txt = "The best things in life are free!"  
if "expensive" not in txt:  
    print("No, 'expensive' is NOT present.")
```


جلب العناصر من البداية

من خلال ترك رقم البداية، سيبدأ النطاق عند الحرف الأول

مثال ↘ ↘

احصل على المواضع من البداية إلى الموضع 5 (غير متضمنة)

```
b = "Abo Habib Al-Hosiny !"  
print(b[:5])
```

جلب العناصر حتى النهاية

من خلال ترك رقم النهاية، سينتقل النطاق إلى النهاية

مثال ↘ ↘

احصل على المواضع من الموضع 2، وحتى النهاية

```
b = "Abo Habib Al-Hosiny !"  
print(b[2:])
```

استخدام الأرقام السالبة

: استخدم الأرقام السالبة لبدء العنصر من نهاية النصوص

مثال ↘ ↘

ستحصل على الجملة ما عدا الحرف قبل الأخير لانك تبتدى منه

```
b = "Abo Habib Al-Hosiny !"
```

```
print(b[-5:-2])
```

✓ كيف تعديل النصوص

في لغة بايثون مجموعة من الأساليب والفرنكشن التي تستطيع استخدامها على النصوص.

التحويل الى الأحرف الكبيرة

مثال ↘ ↘

بإرجاع النصوص بالأحرف الكبيرة `upper()` تقوم الدالة

```
a = "Abo Habib Al-Hosiny !"  
print(a.upper())
```

التحويل الى احرف صغيرة

مثال ↘ ↘

بإرجاع النصوص بأحرف صغيرة `lower()` تقوم الدالة

```
a = "Abo Habib Al-Hosiny !"  
print(a.lower())
```

إزالة المسافة

المسافة هي المسافة قبل و/أو بعد النصوص الفعلي، وفي كثير من الأحيان تريد إزالة هذه المسافة.

مثال ↘ ↘

:بإزالة أي مسافة من البداية أو النهاية **strip()** تقوم الدالة

```
a = "  Abo Habib Al-Hosiny !  "
print(a.strip()) # returns "Abo Habib Al-Hosiny !"
```

استبدال النصوص

مثال ↘ ↘

:نص بنص أخرى **replace()** تستبدل الدالة

```
a = "Abo Habib Al-Hosiny !"
print(a.replace("H", "J"))
```


تقسيم النصوص

بإرجاع قائمة حيث يصبح النصوص الموجود بين `split()` تقوم الدالة الفاصل المحدد هو عناصر القائمة.

مثال ↘ ↘

بتقسيم النصوص إلى نصوص فرعية إذا وجدت `split()` تقوم الدالة بمثيلات للفاصل:

```
a = "Abo , Habib , Al-Hosiny !"
print(a.split(",")) # returns ['Abo', ' Habib ', ' Al-
Hosiny !']
```

العلامات المفردة والمزدوجة

النصوص في لغة بايثون محاطة بعلامات اقتباس مفردة أو علامات اقتباس مزدوجة.

"مرحبا" هو نفس 'مرحبا'.

`print()`: تستطيع عرض نصوص حرفية باستخدام الدالة

مثال ↘ ↘

```
print(" Al Habib")
```

```
print('Hello')
```

تعيين نصوص إلى متغير

يتم تعيين نصوص إلى متغير باستخدام اسم المتغير متبوعاً بعلامة : يساوي والنص :

مثال ↘ ↘

```
a = " Al Habib"
```

```
print(a)
```

تقطيع النصوص

طرق التقطيع

تستطيع إرجاع نطاق من الأحرف باستخدام بناء جملة العنصر حدد رقم البداية ورقم النهاية، مفصولين بنقطتين، لإرجاع جزء من النصوص .

مثال ↘ ↘

:احصل على الأحرف من الموضع 2 إلى الموضع 5 (غير متضمن)

```
b = "Abo Habib Al-Hosiny !"
```

```
print(b[2:5])
```

ملاحظة: الحرف الأول يحتوي على رقم 0

✓ تعديل النصوص

فى لغة بايثون مجموعة من الأساليب والڤنكشن اللى تستطيع استخدامها على النصوص.

الأحرف الكبيرة

مثال ↘ ↘

:إرجاع النصوص بالأحرف الكبيرة **upper()** تقوم الدالة

```
a = "Abo Habib Al-Hosiny !"
```

```
print(a.upper())
```

أحرف صغيرة

مثال ↘ ↘

:إرجاع النصوص بأحرف صغيرة **lower()** تقوم الدالة

```
a = "Abo Habib Al-Hosiny !"
```

```
print(a.lower())
```


إزالة المسافة

المسافة هي المسافة قبل و/أو بعد النصوص الفعلي، وفي كثير من الأحيان تريد إزالة هذه المسافة.

مثال ↘ ↘

:إزالة أي مسافة من البداية أو النهاية `strip()` تقوم الدالة

```
a = " Abo Habib Al-Hosiny ! "
```

```
print(a.strip()) # returns "Abo Habib Al-Hosiny !"
```

تابع استبدال النصوص

مثال ↘ ↘

:نص بنص أخرى `replace()` تستبدل الدالة

```
a = "Abo Habib Al-Hosiny !"
```

```
print(a.replace("Al-Hosiny ", "Al-Masry"))
```

التجزئة والتقسيم

بإرجاع قائمة حيث يصبح النصوص الموجود بين `split()` تقوم الدالة الفاصل المحدد هو عناصر القائمة.

مثال ↘ ↘

بتقسيم النصوص إلى نصوص فرعية إذا وجدت `split()` تقوم الدالة بمثيلات للفاصل:

```
a = "Abo , Habib , Al-Hosiny !"
print(a.split(",")) # returns ["Abo " , "Habib " , "Al-
Hosiny !"]
```

ربط النصوص

دمج النصوص

+ لربط نصين أو دمجهما، تستطيع استخدام

مثال ↘ ↘

c: في المتغير b مع المتغير a دمج المتغير

```
a = " Al Habib"  
b = "Al-Hosiny 😊!"  
c = a + b  
print(c)
```

مثال ↘ ↘

": لإضافة مسافة بينهما أضف

```
a = " Al Habib"  
b = "Al-Hosiny 😊!"  
c = a + " " + b  
print(c)
```

التنسيق - للنصوص

كيف تنسيق النصوص

كما تعلمنا في فصل متغيرات بايثون، لا نستطيع الجمع بين النصوص والأرقام مثل هذا

مثال ↘ ↘

```
age = 36
txt = "My name is Abo Habib Al-Hosiny , I am " +
age
print(txt)
```

format() ولكن نستطيع الجمع بين النصوص والأرقام باستخدام هذه الدالة!

الوسائط التي تم تمريرها، وتنسيقها، ووضعها في **format()** تأخذ الدالة النصوص حيث **{}** تكون العناصر النائبة:

مثال ↘ ↘

الدالة لإدراج الأرقام في النصوص **format()** استخدم

```
age = 36
txt = "My name is Abo Habib Al-Hosiny !!!, and I
am {}"
print(txt.format(age))
```


عددًا غير محدود من الوسائط، ويتم وضعه `format()` يأخذ الأسلوب في العناصر النائبة المعنية:

مثال ↘ ↘

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {}
dollars."
print(myorder.format(quantity, itemno, price))
```

تستطيع استخدام رقمنة التنسيق `{0}` للتأكد من وضع الوسيطات في العناصر النائبة الصحيحة:

مثال ↘ ↘

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces
of item {1}."
print(myorder.format(quantity, itemno, price))
```

أحرف اثثناء

اثثناء أحرف من النصوص

لإدراج أحرف غير المقبولة في اللغة في نص، استخدم حرف اثثناء.
حرف اثثناء هو خط مائل عكسي \ متبوعاً بالحرف الذي تريد إدراجه.

مثال ↘ ↘ على الحرف غير المقبول في اللفة هو علامة الاقتباس:
المزدوجة داخل نصوص محاطة بعلامات اقتباس مزدوجة

مثال ↘ ↘

سوف تحصل على خطأ إذا استخدمت علامات الاقتباس المزدوجة
داخل نصوص محاطة بعلامات اقتباس مزدوجة

```
txt = "We are the so-called "Vikings" from the  
north."
```

": لإصلاح هذه المشكلة، استخدم حرف اثثناء

مثال ↘ ↘

يسمح لك حرف اثثناء باستخدام علامات الاقتباس المزدوجة عندما لا يُسمح لك بذلك عادةً:

```
txt = "We are the so-called \"Vikings\" from the north."
```

موضع الاثناء

أحرف الاثناء الأخرى المستخدمة في بايثون:

True أو **False**: تمثل القيم المنطقية إحدى القيمتين

القيم المنطقية

True في البرمجة، غالبًا ما تحتاج إلى معرفة ما إذا كان التعبير هو **False** أو **True**.

True أو **False**، تستطيع تقييم أي تعبير في بايثون، وإحدى الإجابتين: عند مقارنة قيمتين، يتم تقييم التعبير وترجع بايثون الإجابة المنطقية

مثال ↘ ↘

```
print(10 > 9)
```

```
print(10 == 9)
```

```
print(10 < 9)
```

True أو **False**: تقوم بايثون بإرجاع **if** عند تشغيل شرط في عبارة

مثال ↘ ↘

True أم لا **False**: اطبع رسالة بناءً على ما إذا كان الشرط

```
a = 200
```

```
b = 33
```

```
if b > a:
```

```
    print("Abo Habib _ Al Hosini ")
```

```
else:
```

```
    print("Al Badrasheen_")
```


تقييم البيانات

في **False** أو **True** تقييم أي قيمة وتعطيك **bool()** تتيح لك الدالة المقابل،

مثال ↘ ↘

تقييم نصوص ورقم

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

```
print(bool(15))
```

المعاملات الرياضية ✓

انواع المعاملات الرياضية

يتم استخدام العوامل لإجراء العمليات على المتغيرات والقيم
في ال **مثال** ↘ ↘ أدناه، نستخدم **+** المعامل لجمع قيمتين معًا

مثال ↘ ↘

```
print(10 + 5)
```

معاملات بايثون الحسابية

تُستخدم العوامل الحسابية مع القيم الرقمية لإجراء العمليات الحسابية الشائعة:

معامل	اسم	مثال
+	إضافة	$x + y$
-	الطرح	$x - y$
*	عملية الضرب	$x * y$
/	قسم	x / y
%	معامل بايقى القسمة	$x \% y$
**	الأس	$x ** y$
//	تقسيم الكلمة	$x // y$

معاملات تعيين بايثون

يتم استخدام عوامل التعيين لتعيين قيم للمتغيرات:

معامل	مثال	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$

<code>%=</code>	<code>x %= 3</code>	<code>x = x % 3</code>
<code>//=</code>	<code>x //= 3</code>	<code>x = x // 3</code>
<code>**=</code>	<code>x **= 3</code>	<code>x = x ** 3</code>
<code>&=</code>	<code>x &= 3</code>	<code>x = x & 3</code>
<code> =</code>	<code>x = 3</code>	<code>x = x 3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>>>=</code>	<code>x >>= 3</code>	<code>x = x >> 3</code>
<code><<=</code>	<code>x <<= 3</code>	<code>x = x << 3</code>

معاملات المقارنة بايثون

يتم استخدام عوامل المقارنة لمقارنة قيمتين:

معامل	اسم	مثال
<code>==</code>	متساوي	<code>x == y</code>
<code>!=</code>	غير متساوي	<code>x != y</code>
<code>></code>	أكثر من	<code>x > y</code>
<code><</code>	أقل من	<code>x < y</code>
<code>>=</code>	أكبر من أو يساوي	<code>x >= y</code>
<code><=</code>	أقل أو يساوي	<code>x <= y</code>

عوامل التشغيل المنطقية في بايثون

يتم استخدام العوامل المنطقية لدمج العبارات الشرطية

معامل	وصف	مثال ↘ ↙
and	إذا كانت كلا العبارتين True إرجاع	$x < 5$ and $x < 10$
or	إذا كانت إحدى True إرجاع العبارات صحيحة	$x < 5$ or $x < 4$
not	عكس ال، وإرجاع خطأ إذا كانت ال صحيحة	not($x < 5$ and $x < 10$)

معاملات منطقية

يتم استخدام عوامل الهوية لمقارنة الكائنات، ليس إذا كانت متساوية، ولكن إذا كانت في الواقع نفس الكائن، مع نفس موقع الذاكرة

معامل	وصف	مثال ↘ ↙
is	إذا كان كلا المتغيرين True إرجاع هما نفس الكائن	x is y
is not	إذا كان كلا المتغيرين True إرجاع ليسا نفس الكائن	x is not y

معاملات البحث

تُستخدم عوامل تشغيل العضوية لاختبار ما إذا كان هناك تسلسل موجود في كائن:

معامل	وصف	مثال
<code>in</code>	في حالة وجود تسلسل بالقيمة <code>True</code> تُرجع المحددة في الكائن	<code>x in y</code>
<code>not in</code>	في حالة عدم وجود تسلسل <code>True</code> تُرجع بالقيمة المحددة في الكائن	<code>x not in y</code>

Bitwise معاملات بايثون

لمقارنة الأرقام (الثنائية) `Bitwise` يتم استخدام عوامل

معامل	Name	وصف
<code>&</code>	<code>AND</code>	يضبط كل بت على 1 إذا كان كلا البتين 1
<code> </code>	<code>OR</code>	يضبط كل بت على 1 إذا كان أحد البتين هو 1
<code>^</code>	<code>XOR</code>	يضبط كل بت على 1 إذا كان واحد فقط من البتين هو 1
<code>~</code>	<code>NOT</code>	يعكس كل البتات
<code><<</code>	<code>Zero fill left shift</code>	انتقل إلى اليسار عن طريق دفع الأصفار من اليمين واطرئ البتات الموجودة في أقصى اليسار تسقط

>>

**Signed right
shift**

قم بالتحريك لليمين عن طريق دفع
نسخ من أقصى اليسار للداخل من
اليسار، واترك البتات الموجودة في
أقصى اليمين تسقط

✓ قوائم بايثون

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

قائمة

تُستخدم القوائم لتخزين عناصر كثيرة في متغير واحد.

القوائم هي واحدة من أنواع الاجرائات الأربعة في لغة بايثون المستخدمة لتخزين مجموعات من الاكواد ، أما الأنواع الثلاثة الأخرى وكلها ذات خصائص ، **Dictionary** و **Set** و **Tuple** فهي واستخدمات مختلفة

:يتم إنشاء القوائم باستخدام الأقواس المربعة

مثال ↙ ↘

:انشئ قائمة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
| print(HABIB_list)
```


التعامل مع القائمة

عناصر القائمة مرتبة وقابلة للتغيير وتسمح بقيم نسخة

تتم رقمة عناصر القائمة، العنصر الأول به رقم [0]، والعنصر الثاني به رقم [1] وما إلى ذلك

ترتيب القوائم

عندما نقول أن القوائم مرتبة، فهذا يعني أن العناصر لها ترتيب محدد، ولن يتغير هذا الترتيب

إذا قمت بإضافة عناصر جديدة إلى القائمة، فسيتم وضع العناصر الجديدة في نهاية القائمة

ملاحظة: هناك بعض دوال القائمة التي من شأنها تغيير الترتيب، ولكن بشكل عام: ترتيب العناصر لن يتغير

القائمة قابلة للتغيير

القائمة قابلة للتغيير، مما يعني أنه نستطيع تغيير العناصر وإضافتها وإزالتها في القائمة بعد إنشائها

السماح بالتكرارات

بما أن القوائم مرقمة، يمكن أن تحتوي القوائم على عناصر بنفس القيمة:

مثال ↘ ↘

تسمح القوائم بقيم نسخة

```
HABIB_list = ["Abo", "Habib", "Al  
Hosiny", "Abo", "Al Hosiny"]  
print(HABIB_list)
```

تحديد طول القائمة يعني عدد العناصر

len() لتحديد عدد العناصر الموجودة في القائمة، استخدم الدالة

مثال ↘ ↘

طباعة عدد العناصر في القائمة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
print(len(HABIB_list))
```


عناصر القائمة و أنواع الإجراءات

: يمكن أن تكون عناصر القائمة من أي نوع بيانات

مثال ↘ ↘

String و int و boolean: أنواع الاجراءات

```
hosini1 = ["Abo", "Habib", "Al Hosiny"]
```

```
hosini2 = [1, 5, 7, 9, 3]
```

```
hosini3 = [True, False, False]
```

:يمكن أن تحتوي القائمة على أنواع اكواد مختلفة

مثال ↘ ↘

:قائمة تحتوي على نصوص وأعداد صحيحة وقيم منطقية

```
hosini1 = ["Ahmed", 34, True, 40, "Moslem"]
```

تعريف القائمة

من وجهة نظر بايثون، يتم تعريف القوائم على أنها كائنات بنوع الاكواد قائمة

```
<class 'list'>
```

مثال ↘ ↘

ما هو نوع بيانات القائمة؟

```
Hosini_list = ["Abo", "Habib", "Al Hosiny"]  
print(type(Hosini_list))
```

انشاء القوائم

عند إنشاء قائمة جديدة (**list ()**) من الممكن أيضًا استخدام

مثال ↘ ↘

:المنشئ لإنشاء قائمة **list()** استخدام

```
HABIB_list = list(("Abo", "Habib", "Al Hosiny"))
```


note the double round-brackets

print(HABIB_list)

مجموعات بايثون او (المصفوفات)

هناك أربعة أنواع من اكواد المجموعة في لغة برمجة بايثون

- القائمة عبارة عن مجموعة مرتبة وقابلة للتغيير. يسمح بأعضاء نسخة.
- عبارة عن مجموعة مرتبة وغير قابلة للتغيير. يسمح **Tuple** بأعضاء نسخة.
- المجموعة** عبارة عن مجموعة غير مرتبة وغير قابلة للتغيير * وغير مرقمة. لا يوجد أعضاء نسخة.
- القاموس** عبارة عن مجموعة مرتبة ** وقابلة للتغيير. لا يوجد أعضاء نسخة.

عناصر المجموعة غير قابلة للتغيير، ولكن تستطيع إزالة و/أو إضافة * عناصر وقتما تشاء.

اعتبراً من إصدار بايثون 3.7، يتم ترتيب القواميس . في بايثون 3.6 ** . والإصدارات السابقة، تكون القواميس غير مرتبة

عند اختيار نوع المجموعة، من المفيد فهم خصائص هذا النوع. إن اختيار النوع المناسب لمجموعة اكواد معينة قد يعني الاحتفاظ بالمعنى، وقد يعني زيادة في الكفاءة أو الأمان.

✓ الوصول إلى عناصر القائمة

طرق الوصول

يتم رقمنة عناصر القائمة وتستطيع الوصول إليها عن طريق الرجوع إلى رقم العنصر

مثال ↘ ↘

طباعة العنصر الثاني من القائمة:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
print(HABIB_list[1])
```

ملاحظة: يحتوي العنصر الأول على رقم 0

استخدام الأرقام السالبة

الأرقام السالبة تعني البدء من النهاية

يشير إلى العنصر الأخير، **-2** ويشير إلى العنصر الأخير الثاني وما إلى **-1** ذلك.

مثال ↘ ↘

طباعة العنصر الأخير من القائمة:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
print(HABIB_list[-1])
```

نطاق الأرقام

تستطيع تحديد نطاق من الأرقام عن طريق تحديد مكان البدء ومكان نهاية النطاق.

عند تحديد نطاق، ستكون القيمة المرجعة عبارة عن قائمة جديدة تحتوي على العناصر المحددة.

مثال ↘ ↘

إرجاع العنصر الثالث والرابع والخامس:

```
HABIB_list = ["Abo", "Habib", "Al  
Hosiny", "Egypt", "AlBadrashin", "Arabic"]  
print(HABIB_list[2:5])
```

ملحوظة: سيبدأ البحث عند الرقم 2 (متضمن) وينتهي عند الرقم 5 (غير مضمن).

تذكر أن العنصر الأول يحتوي على رقم 0

من خلال ترك قيمة البداية، سيبدأ النطاق عند العنصر الأول

مثال ↘ ↘

يقوم هذا ال مثال ↘ ↘ بإرجاع العناصر من البداية إلى "AlBadrashin" ولكن لا تتضمنها

```
HABIB_list = ["Abo", "Habib", "Al  
Hosiny", "Egypt", "AlBadrashin", "Arabic"]  
print(HABIB_list[:4])
```


من خلال ترك القيمة النهائية، سيستمر النطاق حتى نهاية القائمة:

مثال ↘ ↘

يقوم هذا ال **مثال** ↘ ↘ بإرجاع العنصر الذي إلى النهاية

```
HABIB_list = ["Abo", "Habib", "Al  
Hosiny", "Egypt", "AlBadrashin", "Arabic", "Pytho  
n"]  
print(HABIB_list[2:])
```

العمل بـ الأرقام السالبة

:حدد الأرقام السالبة إذا كنت تريد بدء البحث من نهاية القائمة

مثال ↘ ↘

يقوم هذا ال **مثال** ↘ ↘ بإرجاع العناصر المختارة ويجب ان تعرفها :
بنفسك :

```
HABIB_list = ["Abo", "Habib", "Al  
Hosiny", "Egypt", "AlBadrashin", "Arabic", "Pytho  
n"]  
print(HABIB_list[-4:-1])
```

تحقق من وجود العنصر

لتحديد ما إذا كان هناك عنصر محدد موجود في القائمة، استخدم
المحجوزة **in** الكلمة :

مثال ↘ ↘

موجودة في القائمة "Abo" تحقق مما إذا كانت كلمة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
if "Abo" in HABIB_list:  
print("Yes, 'Abo' is in the HosHos list")
```


تغيير عناصر القائمة

تغيير قيمة العنصر

: لتغيير قيمة عنصر معين

مثال ↘ ↘

تغيير العنصر الثاني:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
HABIB_list[1] = "Ahmed"  
print(HABIB_list)
```

ادراج او تغيير العناصر

لتغيير قيمة العناصر ضمن نطاق معين، حدد قائمة بالقيم الجديدة، وقم بالإشارة إلى نطاق رقمنا التنسيق الذي تريد إدراج القيم الجديدة فيه:

مثال ↘ ↘

شاهد هذا ال مثال ↘ ↘ جلب من العنصر حبيب الى العنصر مصر و لن يتضمن العنصر مصر

```
HABIB_list = ["Abo", "Habib", "Al  
Hosiny", "Egypt", "AlBadrashin", "Python"]  
HABIB_list[1:3] = ["Ahmed", "Al-Masry"]  
print(HABIB_list)
```

إذا قمت بإدراج عناصر أكثر مما قمت باستبداله، فسيتم إدراج العناصر الجديدة في المكان الذي حددته، وسيتم نقل العناصر المتبقية وفقاً لذلك

مثال ↘ ↘

: قم بتغيير القيمة الثانية عن طريق استبدالها بقيمتين جديدتين

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
HABIB_list[1:2] = ["Ahmed", "Al-Masry"]  
print(HABIB_list)
```


ملاحظة: سيتغير طول القائمة عندما لا يتطابق عدد العناصر المدرجة مع عدد العناصر المستبدلة.

إذا قمت بإدراج عناصر أقل مما قمت باستبداله، فسيتم إدراج العناصر الجديدة في المكان الذي حددته، وسيتم نقل العناصر المتبقية وفقاً لذلك:

مثال ↘ ↘

: قم بتغيير القيمة الثانية والثالثة باستبدالها بقيمة واحدة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
HABIB_list[1:3] = ["Al-Masry"]
print(HABIB_list)
```

إدراج عناصر

لإدراج عنصر قائمة جديد، دون استبدال أي من القيم الموجودة، نستطيع الدالة `insert()` استخدام هذه.

بإدراج عنصر في الرقم المحدد `insert()` تقوم الدالة

مثال ↘ ↘

هذا الكود سيستبدال مكان القيمة الثالثة الحسينى بالمصرى وستصبح الحسينى رقم اربعة والمصرى رقم ثلاثة وسيزيد عدد القائمة بواحد

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
HABIB_list.insert(2, "Al-Masry")  
print(HABIB_list)
```

ملاحظة: لل مثال ↘ ↘ أعلاه، ستحتوي القائمة الآن على 4 عناصر

✓ تابع إضافة عناصر القائمة

تابع ادراج العناصر

: لإضافة عنصر إلى نهاية القائمة، استخدم الدالة **Append ()**

مثال ↘ ↘

:الدالة لادراج عنصر **append()** استخدام


```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
HABIB_list.append("in Egypt")
print(HABIB_list)
```

مثال اخر تابع إدراج عناصر

. الدالة لإدراج عنصر قائمة في رقم محدد، استخدم **insert()** الدالة بإدراج عنصر في الرقم المحدد **insert()** تقوم الدالة

مثال ↘ ↘

أدخل عنصرًا باعتباره الموضع الرابع:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
HABIB_list.insert(4, "in Egypt")
print(HABIB_list)
```

ملاحظة: للأمثلة المذكورة أعلاه، ستحتوي القوائم الآن على 4 عناصر.

توسيع القائمة

لادراج عناصر من قائمة أخرى بالقائمة الحالية، ادمج قوائم الدالة `extend()` استخدم

مثال ↘ ↘

`HABIB_list` إلى `Hosini_list` أضف عناصر

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
Hosini_list = ["Python", "JavaScript", "Node.js"]
HABIB_list.extend(Hosini_list)
print(HABIB_list)
```

سيتم إضافة العناصر إلى نهاية القائمة.

الإضافات قابلة للنسخ

لا تحتاج الدالة إلى ادراج القوائم ، تستطيع إضافة أي كائن `extend()` قابل للنسخ (المصفوفة، المجموعات، القواميس وما إلى ذلك)

مثال ↘ ↘

إلى القائمة `Tuple` إضافة عناصر


```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
Hosini_tuple = ("AlBadrashin", "Egypt")
HABIB_list.extend(Hosini_tuple)
Hosini_tuple = ("AlBadrashin", "Egypt",
"Abo", "Habib", "Al Hosiny")
HABIB_list.extend(Hosini_tuple)
print(HABIB_list)
```

✓ إزالة عناصر القائمة

كيف إزالة العنصر المحدد

بإزالة العنصر المحدد `remove()` تقوم الدالة

مثال ↙ ↘

إزالة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
HABIB_list.remove("Habib")
print(HABIB_list)
```

إزالة العنصر المحدد

بإزالة الرقم المحدد **pop()** تقوم الدالة

مثال ↘ ↘

إزالة العنصر الثاني:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
HABIB_list.pop(1)
print(HABIB_list)
```

فستقوم الدالة بإزالة العنصر الأخير **pop()**، إذا لم تقم بتحديد الرقم

مثال ↘ ↘

إزالة العنصر الأخير:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
HABIB_list.pop()
print(HABIB_list)
```

تقوم هذه الدالة أيضًا بإزالة الرقم المحدد **del**

مثال ↘ ↘

إزالة العنصر الأول:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
del HABIB_list[0]  
print(HABIB_list)
```

المحجوزة أيضًا حذف القائمة بالكامل **del** يمكن للكلمة

مثال ↘ ↘

حذف القائمة بأكملها:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
del HABIB_list
```

امسح القائمة

يفرغ القائمة (**clear()**) الأسلوب

القائمة لا تزال موجودة، ولكن ليس لديها محتوى

مثال ↘ ↘

مسح محتوى القائمة:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
HABIB_list.clear()  
print(HABIB_list)
```

✓ الحلقات على القوائم

حلقة بسيطة على القائمة

حلقة **for** تستطيع تكرار عناصر القائمة باستخدام

مثال ↘ ↘

طباعة جميع العناصر الموجودة في
القائمة، واحدًا تلو الآخر:

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
for x in HABIB_list:  
    print(x)
```


الحلقة عن طريق رقم العنصر

تستطيع أيضًا تكرار عناصر القائمة بالإشارة إلى الرقم الخاص بها لإنشاء تكرار مناسب `len()` و `range()` استخدم دوال

مثال ↘ ↘

: طباعة جميع العناصر من خلال الإشارة إلى الرقم

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
for i in range(len(HABIB_list)):  
    print(HABIB_list[i])
```

المفاتيح التي تم إنشاؤه في ال `range()` مثال ↘ ↘ أعلاه هو `[0, 1, 2]`

while استخدام حلقة

حلقة `while` تستطيع تكرار عناصر القائمة باستخدام

الدالة لتحديد طول القائمة، ثم ابدأ من 0 ثم كرر طريقك `len()` استخدم عبر عناصر القائمة من خلال الإشارة إلى أرقامها

تذكر زيادة الرقم بمقدار 1 بعد كل تكرار.

مثال ↘ ↘

حلقة لتصفح جميع رقمنة **while** اطبع جميع العناصر باستخدام التنسيق

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]  
i = 0  
while i < len(HABIB_list):  
    print(HABIB_list[i])  
    i = i + 1
```

for in للنسخ باستخدام

:أقصر بناء جملة للنسخ عبر القوائم **for in** يقدم

مثال ↘ ↘

:ستطبع جميع العناصر الموجودة في القائمة **for** حلقة يد قصيرة


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

```
[print(x) for x in HABIB_list]
```

تابع for in حلقة

بناء جملة أقصر عندما تريد إنشاء قائمة جديدة بناءً على `for in` يوفر قيم القائمة الموجودة.

مثال ↘ ↘:

بناءً على قائمة ، تريد قائمة جديدة تحتوي فقط على التي تحتوي في الاسم "a" على الحرف.

عبارة تحتوي على اختبار `for` سيتعين عليك كتابة `for in` بدون شرطية بداخلها:

مثال ↘ ↘:

```
HosHos = ["Abo", "Habib", "Al  
Hosiny", "AlBadrashin", "Python"]  
Hosini_list = []
```

```
for x in HosHos:  
    if "a" in x:  
        Hosini_list.append(x)
```

```
| print(Hosini_list)
```

تستطيع القيام بكل ذلك باستخدام سطر واحد فقط **for in**، باستخدام من الاكواد :

مثال ↘ ↘

```
| HosHos = ["Abo", "Habib", "Al  
| Hosiny", "AlBadrashin", "Python"]
```

```
| Hosini_list = [x for x in HosHos if "a" in x]
```

```
| print(Hosini_list)
```

بناء الجملة

```
Hosini_list =  
[expression for item in iterable if condition == True]
```

القيمة المرجعة هي قائمة جديدة، مع ترك القائمة القديمة دون تغيير.

حالة

يشبه الشرط عامل التصفية الذي يقبل فقط العناصر التي يتم تقييمها بـ **True**.

مثال ↘ ↘

: قبول فقط العناصر التي ليست

```
Hosini_list = [x for x in HosHos if x != "Abo"]
```

True سيعود **x != "Abo"** if الحالة

لجميع العناصر ، مما يجعل القائمة الجديدة تحتوي على جميع العناصر المختلة

: الشرط اختياري ويمكن حذفه

مثال ↘ ↘

:كلمة **if** بدون

```
Hosini_list = [x for x in HosHos]
```

اشياء متوقعة الحدوث فى الحلقات

يمكن أن يكون الكائن القابل للنسخ أي كائن قابل للنسخ، مثل قائمة أو صف أو مجموعة وما إلى ذلك.

مثال ↘ ↘

الدالة لإنشاء range() تستطيع استخدام تكرار:

```
Hosini_list = [x for x in range(10)]
```

نفس ال مثال ↘ ↘ لكن بشرط

مثال ↘ ↘

قبول الأرقام الأقل من 5 فقط

```
Hosini_list = [x for x in range(10) if x < 5]
```

تعبير

التعبير هو العنصر الحالي في النسخ ، ولكنه أيضًا ال، والتي تستطيع معالجة قبل أن ينتهي بها الأمر كعنصر قائمة في القائمة الجديدة :

مثال ↘ ↘

قم بتعيين القيم في القائمة الجديدة إلى أحرف كبيرة

```
Hosini_list = [x.upper() for x in HosHos]
```

تستطيع ضبط ال على ما تريد

مثال ↘ ↘

"اضبط جميع القيم في القائمة الجديدة على "هالو

```
Hosini_list = ['hello' for x in HosHos]
```

يمكن أن يحتوي التعبير أيضًا على شروط، ليس مثل الاختيار ، ولكن كوسيلة لمعالجة ال

مثال ↘ ↘

استبدل ايجبت بدل حبيب

```
Hosini_list = [x if x !=  
"Egypt" else "Habib" for x in HosHos]
```

✓ فرز او ترتيب القوائم

فرز القائمة أبجدياً او عددياً

دالة تقوم بفرز القائمة أبجدياً (**sort()**) تحتوي كائنات القائمة على
رقميًا، تصاعديًا، افتراضيًا

مثال ↘ ↘

ترتيب القائمة أبجدياً

```
HABIB_list =  
["Egypt", "Python", "AlBadrashin", "JavaScript", "  
Habib"]  
HABIB_list.sort()  
print(HABIB_list)
```

مثال ↘ ↘

فرز القائمة عددياً


```
HABIB_list = [100, 50, 65, 82, 23]
HABIB_list.sort()
print(HABIB_list)
```

ترتيب تنازلي

`reverse = True` للفرز تنازليًا، استخدم وسيطة الكلمة المحجوزة

مثال ↘ ↘

ترتيب القائمة تنازليًا

```
HABIB_list =
["Egypt", "Python", "AlBadrashin", "JavaScript", "
Habib"]
HABIB_list.sort(reverse = True)
print(HABIB_list)
```

مثال ↘ ↘

ترتيب القائمة تنازليًا

```
HABIB_list = [100, 50, 65, 82, 23]
HABIB_list.sort(reverse = True)
print(HABIB_list)
```

تخصيص دالة الفرز

تستطيع أيضًا تخصيص وظيفتك باستخدام وسيطة الكلمة المحجوزة **key = function**.

:ستعيد الدالة رقمًا سيتم استخدامه لفرز القائمة (الرقم الأقل أولاً)

مثال ↘ ↘

:قم بفرز القائمة بناءً على مدى قرب الرقم من 50

```
def Hosini(n):
    return abs(n - 50)
```

```
HABIB_list = [100, 50, 65, 82, 23]
HABIB_list.sort(key = Hosini)
print(HABIB_list)
```


فرز غير حساس لحالة الأحرف

تكون الدالة حساسة لحالة الأحرف، مما يؤدي **sort()**، بشكل افتراضي إلى فرز جميع الأحرف الكبيرة قبل الأحرف الصغيرة

مثال ↘ ↘

يمكن أن يؤدي الفرز الحساس لحالة الأحرف إلى غير متوقعة

```
HABIB_list = ["Habib", "Egypt", "AlBadrashin", "Al  
Hosiny"]  
HABIB_list.sort()  
print(HABIB_list)
```

لحسن الحظ، نستطيع استخدام الدوال كدوال رئيسية عند فرز القائمة
لذا، إذا كنت تريد دالة فرز غير حساسة لحالة الأحرف، فاستخدم
كدالة رئيسية **str.lower**

مثال ↘ ↘

قم بإجراء نوع غير حساس لحالة الأحرف من القائمة

```
HABIB_list = ["Habib", "Egypt", "AlBadrashin", "Al  
Hosiny"]
```

```
HABIB_list.sort(key = str.lower)
print(HABIB_list)
```

ترتيب عكسي

ماذا لو كنت تريد عكس ترتيب القائمة، بغض النظر عن الأبجدية؟
ترتيب الفرز الحالي للعناصر **reverse()** تعكس الدالة

مثال ↘ ↘

عكس ترتيب عناصر القائمة:

```
HABIB_list = ["Habib", "Egypt", "AlBadrashin", "Al  
Hosiny"]
HABIB_list.reverse()
print(HABIB_list)
```

✓ نسخ القوائم

انسخ القائمة

`hosini2 = hosini1`، لا تستطيع نسخ قائمة بمجرد كتابة وسيتم إجراء `hosini1`، `hosini1` سيكون مرجعًا فقط إلى `hosini2`: لأن `hosini2` التغييرات التي تم إجراؤها في

هناك دوال لإنشاء نسخة، إحدى الدوال هي استخدام دالة القائمة `copy()`.

مثال ↘ ↘

`copy()` قم بعمل نسخة من القائمة بالدالة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
Hosini_list = HABIB_list.copy()
print(Hosini_list)
```

`list()` هناك دالة أخرى لعمل نسخة وهي استخدام الدالة

مثال ↘ ↘

`list()` قم بعمل نسخة من القائمة بالدالة

```
HABIB_list = ["Abo", "Habib", "Al Hosiny"]
Hosini_list = list(HABIB_list)
print(Hosini_list)
```

✓ ضم القوائم

كيف تضم قائمتين

هناك عدة دوال لربط أو ربط قائمتين أو أكثر في بايثون.
واحدة من أسهل الدوال هي استخدام **+** المشغل

مثال ↘ ↘

ضم قائمتين:

```
hosini1 = ["a", "b", "c"]
```

```
hosini2 = [1, 2, 3]
```

```
hosini3 = hosini1 + hosini2
```

```
print(hosini3)
```

هناك دالة أخرى لضم قائمتين وهي ادراج كافة العناصر من **extend()** القائمة 2 إلى القائمة 1، واحدًا تلو الآخر

مثال ↘ ↘

ادراج القائمة 2 بالقائمة 1:

```
hosini1 = ["a", "b", "c"]
```

```
hosini2 = [1, 2, 3]
```

```
for x in hosini2:
```

```
    hosini1.append(x)
```

```
print(hosini1)
```

الدالة التي تهدف إلى إضافة عناصر **extend()** أو تستطيع استخدام من قائمة إلى قائمة أخرى:

مثال ↘ ↘

hosini1: في نهاية hosini2 الدالة لإضافة **extend()** استخدم

```
hosini1 = ["a", "b", "c"]
```

```
hosini2 = [1, 2, 3]
```

```
hosini1.extend(hosini2)
```

```
print(hosini1)
```

✓ دوال القائمة

دوال القائمة

في لغة بايثون مجموعة من الأساليب والفرنكشن التي تستطيع استخدامها في القوائم.

Method	
<u>append()</u>	للاضافة
<u>clear()</u>	لتفريغ القائمة
<u>copy()</u>	لاخذ نسخة من القائمة
<u>count()</u>	إرجاع عدد المرات التي تحدث فيها قيمة محددة في صف
<u>extend()</u>	لدمج
<u>index()</u>	يبحث في الصف عن قيمة محددة ويعيد الموضع الذي تم العثور عليه فيه
<u>insert()</u>	للاضافة
<u>pop()</u>	لحذف برقم العنصر او حذف العنصر الاخير
<u>remove()</u>	لحذف بقيمة العنصر
<u>reverse()</u>	عكس الترتيب
<u>sort()</u>	اعادة الترتيب

✓ Tuple في الـ

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

يتم استخدام المصفوفة لتخزين عناصر كثيرة في متغير واحد.

هو أحد أنواع الاجزئات الأربعة في لغة بايثون والمستخدمه **tuple** لتخزين مجموعات من الاكواد ، أما الأنواع الثلاثة الأخرى وكلها ذات خصائص واستخدامات ، **Dictionary** و **Set** و **List** فهي مختلفة.

. عبارة عن مجموعة مرتبة وغير قابلة للتغيير **tuple**.

. تتم كتابة المصفوفة بأقواس مستديرة.

مثال ↙ ↘

إنشاء صف:

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

```
print(Hosini_tuple)
```

كيف التعامل مع العناصر

مرتبة وغير قابلة للتغيير وتسمح بقيم نسخة Tuple عناصر

العنصر الأول به رقم [0]، والعنصر الثاني به Tuple، يتم رقمة عناصر
رقم [1] وما إلى ذلك

عندما نقول أن المصفوفة مرتبة، فهذا يعني أن العناصر لها ترتيب
محدد، ولن يتغير هذا الترتيب

غير قابل للتغيير

الصف غير قابل للتغيير، مما يعني أنه لا نستطيع تغيير العناصر أو
إضافتها أو إزالتها بعد إنشاء الصف

السماح بالتكرارات

نظرًا لأن المصفوفة مرقمة، فمن الممكن أن تحتوي على عناصر لها
نفس القيمة:

مثال

بقيم نسخة Tuples تسمح


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

طول المصفوفة

الدالة `len()` لتحديد عدد العناصر الموجودة في الصف، استخدم

مثال ↘ ↘

اطبع عدد العناصر في الصف:

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

بعنصر واحد Tuple إنشاء

لإنشاء صف يحتوي على عنصر واحد فقط، عليك إضافة فاصلة بعد العنصر، وإلا فلن تتعرف عليه بايثون على أنه صف.

مثال ↘ ↘

صف عنصر واحد، تذكر الفاصلة

```
Hosini_tuple = ("Abo",)  
print(type(Hosini_tuple))
```

```
#NOT a tuple  
Hosini_tuple = ("Abo")  
print(type(Hosini_tuple))
```

التحكم في الإجراءات – Tuple عناصر

: من أي نوع بيانات Tuple يمكن أن تكون عناصر

مثال ↘ ↘

boolean و int و String أنواع الاجراءات

```
Hosini1 = ("Abo", "Habib", "Al Hosiny")  
Hosini2 = (1, 5, 7, 9, 3)  
Hosini3 = (True, False, False)
```


يمكن أن يحتوي الصف على أنواع اكواد مختلفة

مثال ↘ ↘

صف يحتوي على نصوص وأعداد صحيحة وقيم منطقية

```
Hosini1 = ("Ahmed", 34, True, 40, "Moslem")
```

من وجهة نظر بايثون، يتم تعريف المصفوفة على أنها كائنات من نوع "tuple":

```
<class 'tuple'>
```

مثال ↘ ↘

Tuple؟ ما هو نوع البيانات من

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

دالة انشاء الجداول Tuple()

لإنشاء صف `tuple()` من الممكن أيضًا استخدام مُنشئ

مثال ↘ ↘

لإنشاء صف `tuple()` باستخدام دالة

```
Hosini_tuple = tuple(("Abo", "Habib", "Al  
Hosiny")) # note the double round-brackets  
print(Hosini_tuple)
```

مجموعات بايثون (المصفوفات)

هناك أربعة أنواع من اكواد المجموعة في لغة برمجة بايثون

- **القائمة** عبارة عن مجموعة مرتبة وقابلة للتغيير. يسمح بأعضاء نسخة
- **Tuple** عبارة عن مجموعة مرتبة وغير قابلة للتغيير. يسمح بأعضاء نسخة
- **المجموعة** عبارة عن مجموعة غير مرتبة وغير قابلة للتغيير* وغير مرقمة. لا يوجد أعضاء نسخة
- **القاموس** عبارة عن مجموعة مرتبة** وقابلة للتغيير. لا يوجد أعضاء و نسخ

عناصر المجموعة غير قابلة للتغيير، ولكن تستطيع إزالة و/أو إضافة * عناصر وقتما تشاء.

اعتبراً من إصدار بايثون 3.7، يتم ترتيب القواميس . في بايثون 3.6 **
. والإصدارات السابقة، تكون القواميس غير مرتبة

عند اختيار نوع المجموعة، من المفيد فهم خصائص هذا النوع. إن اختيار النوع المناسب لمجموعة اكواد معينة قد يعني الاحتفاظ بالمعنى، وقد يعني زيادة في الكفاءة أو الأمان.

تفريغ Tuple

عندما نقوم بإنشاء صف، فإننا عادةً ما نقوم بتعيين قيم له. وهذا ما يسمى "التعبئة" صفًا

مثال ↘ ↘

Tuple: التعبئة

HosHos = ("Abo", "Habib", "Al Hosiny")

لكن في بايثون، يُسمح لنا أيضًا باستخراج القيم مرة أخرى إلى "متغيرات". وهذا ما يسمى "التفريغ":

مثال ↘ ↘

تفريغ المصفوفة:

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

```
(Abou, HabiB, Al_Hosiny) = HosHos
```

```
print(Abou)
```

```
print(HabiB)
```

```
print(Al_Hosiny)
```

ملاحظة: يجب أن يتطابق عدد المتغيرات مع عدد القيم في الصف، وإذا لم يكن الأمر كذلك، فيجب عليك استخدام علامة النجمة لجمع القيم المتبقية كقائمة.

* باستخدام النجمة

إلى * an إذا كان عدد المتغيرات أقل من عدد القيم، تستطيع إضافة اسم المتغير وسيتم تخصيص القيم للمتغير كقائمة

مثال ↘ ↘

:"قم بتعيين بقية القيم كقائمة تسمى "حراء

```
HosHos = ("Abo", "Habib", "Al  
Hosiny", "Al_Badrashen", "MahmouD")
```

```
(Abou, HabIB, *Al_Hosiny) = HosHos
```

```
print(Abou)
```

```
print(HabiB)
```

```
print(Al_Hosiny)
```

إذا تمت إضافة العلامة النجمية إلى اسم متغير آخر غير الاسم الأخير، فسوف تقوم بايثون بتعيين قيم للمتغير حتى يتطابق عدد القيم المتبقية مع عدد المتغيرات المتبقية.

مثال ↘ ↘

:"أضف قائمة قيم المتغير "المدار

```
HosHos =
```

```
("C#", "Python", "Node.js", "JavaScript", "PHP")
```

```
(Abou, *python, Al_Hosiny) = HosHos
```

```
print(Abou)
```

```
print(python)
```

```
print(Al_Hosiny)
```

✓ الحلقة على المصفوفة

حلقة الأولى Tuple

حلقة **for** تستطيع تكرار عناصر المجموعة باستخدام

مثال ↘ ↘

قم بالنسخ عبر العناصر وطباعة القيم


```
Hosini_tuple = ("Abo", "Habib", "Al Hosiny")
for x in Hosini_tuple:
    print(x)
```

حلقة من خلال رقم العنصر

تستطيع أيضًا تكرار عناصر المجموعة من خلال الإشارة إلى الرقم الخاص بها.

لإنشاء تكرار مناسب `len()` و `range()` استخدم دوال

مثال ↘ ↘

: طباعة جميع العناصر من خلال الإشارة إلى الرقم

```
Hosini_tuple = ("Abo", "Habib", "Al Hosiny")
for i in range(len(Hosini_tuple)):
    print(Hosini_tuple[i])
```

while استخدام حلقة

حلقة **while** تستطيع تكرار عناصر القائمة باستخدام

الدالة لتحديد طول الصف، ثم ابدأ من 0 ثم كرر طريقك **len()** استخدم
عبر عناصر الصف من خلال الرجوع إلى ارقامها

تذكر زيادة الرقم بمقدار 1 بعد كل تكرار

مثال ↘ ↘

حلقة لتصفح جميع رقمنة **while** اطبع جميع العناصر باستخدام
التنسيق :

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

```
i = 0
```

```
while i < len(Hosini_tuple):
```

```
    print(Hosini_tuple[i])
```

```
    i = i + 1
```


ضم المصفوفات او الجداول كما في القوائم تمام

لضم صفين أو أكثر، تستطيع استخدام + العامل

مثال ↘ ↘

الانضمام إلى مجموعتين

```
Hosini1 = ("a", "b", "c")
```

```
Hosini2 = (1, 2, 3)
```

```
Hosini3 = Hosini1 + Hosini2
```

```
print(Hosini3)
```

ضرب المصفوفة

إذا كنت تريد مضاعفة محتوى صف معين لعدد معين من المرات،
: * فتستطيع استخدام

مثال ↘ ↘

اضرب في 2

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

```
Hosini_tuple = HosHos * 2
```

```
print(Hosini_tuple)
```

✓ دوال Tuple

أهم دوال المصفوفة

في لغة بايثون طريقتان مدمجتان تستطيع استخدامهما في المصفوفة.

Method

وصف

count()

إرجاع عدد المرات التي تحدث فيها قيمة محددة في صف

index()

يبحث في الصف عن قيمة محددة ويعيد الموضع الذي تم العثور عليه فيه

✓ مجموعات بايثون

```
Habib_set = {"Abo", "Habib", "Al Hosiny"}
```


تعيين

تُستخدم المجموعات لتخزين عناصر كثيرة في متغير واحد.

هو أحد أنواع الاجرائات الأربعة في لغة بايثون المستخدمة لتخزين Set

مجموعات من الاكواد ، والأنواع الثلاثة الأخرى

وكلها ذات خصائص ، **Dictionary** و **Tuple** و **List** هي

واستخدامات مختلفة

المجموعة عبارة عن مجموعة غير مرتبة وغير قابلة

للتغيير* وغير مرقمة

ملاحظة: عناصر المجموعة غير قابلة للتغيير، ولكن تستطيع إزالة *

العناصر وإضافة عناصر جديدة

تتم كتابة المجموعات بأقواس مجعدة

مثال ↙ ↘

إنشاء مجموعة

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
print(Hosini_set)
```

ملاحظة: المجموعات غير مرتبة، لذا لا تستطيع التأكد من الترتيب

الذي ستظهر به العناصر

تعيين العناصر

عناصر المجموعة غير مرتبة وغير قابلة للتغيير ولا تسمح بقيم نسخة

غير مرتبة

غير مرتبة يعني أن العناصر الموجودة في المجموعة ليس لها ترتيب محدد.

يمكن أن تظهر عناصر المجموعة بترتيب مختلف في كل مرة. تستخدمها فيها، ولا يمكن الرجوع إليها بواسطة الرقم أو المفتاح.

غير قابل للتغيير

عناصر المجموعة غير قابلة للتغيير، مما يعني أنه لا نستطيع تغيير العناصر بعد إنشاء المجموعة.

بمجرد إنشاء المجموعة، لا نستطيع تغيير عناصرها، ولكن نستطيع إزالة العناصر وإضافة عناصر جديدة.

التكرارات غير المسموح بها

لا يمكن أن تحتوي المجموعات على عنصرين بنفس القيمة

مثال ↘ ↘

سيتم تجاهل القيمة

```
Hosini_set = {"Abo", "Habib", "Al Hosiny", "Abo"}
```

```
print(Hosini_set)
```

احصل على طول المجموعة

الدالة `len()` لتحديد عدد العناصر الموجودة في المجموعة، استخدم

مثال ↘ ↘

احصل على عدد العناصر في المجموعة

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
print(len(Hosini_set))
```

مجموعة العناصر - أنواع الإجراءات

: يمكن أن تكون عناصر المجموعة من أي نوع بيانات

مثال ↘ ↘

أنواع الاجراءات **String** و **int** و **boolean**:

Hosini1 = {"Abo", "Habib", "Al Hosiny"}

Hosini2 = {1, 5, 7, 9, 3, True, False, "Habib", "Al Hosiny"}

Hosini3 = {True, False, False}

:يمكن أن تحتوي المجموعة على أنواع اكواد مختلفة

مثال ↘ ↘

:مجموعة تحتوي على نصوص وأعداد صحيحة وقيم منطقية

Hosini1 = {"Ahmed", 34, True, 40, "Moslem"}

من وجهة نظر بايثون، يتم تعريف المجموعات ككائنات بنوع الاكواد
:""مجموعة

<class 'set'>

مثال ↘ ↘

ما هو نوع البيانات للمجموعة؟

```
Habib_set = {"Abo", "Habib", "Al Hosiny"}  
print(type(Habib_set))
```

من الممكن أيضًا استخدام
إنشاء مجموعة set() مُنشئ

مثال ↘ ↘

:إنشاء مجموعة set() باستخدام مُنشئ

```
Hosini_set = set(("Abo", "Habib", "Al Hosiny")) #  
note the double round-brackets  
print(Hosini_set)
```

عناصر الوصول

لا تستطيع الوصول إلى العناصر الموجودة في مجموعة بالإشارة إلى رقم أو مفتاح

حلقة، أو السؤال **for** ولكن تستطيع تكرار عناصر المجموعة باستخدام عما إذا كانت هناك قيمة محددة موجودة في مجموعة، باستخدام الكلمة **in** الكلمة.

مثال ↘ ↘

قم بالمرور عبر المجموعة وطباعة القيم

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

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


مثال ↘ ↘

تحقق من وجود "حبيب" في المجموعة

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
print("Habib" in Hosini_set)
```

بمجرد إنشاء المجموعة، لا تستطيع تغيير عناصرها، ولكن تستطيع إضافة عناصر جديدة.

إضافة عنصر واحد إلى مجموعة
الدالة `add()` استخدم.

مثال ↘ ↘

الدالة `add()` إضافة عنصر إلى مجموعة باستخدام

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
Hosini_set.add("Egypt")
```

```
print(Hosini_set)
```

إضافة عناصر من مجموعة أخرى إلى
المجموعة الحالية،
الدالة update() تستخدم.

مثال ↘ ↘

Hosini_set: إلى Hosini_list إضافة عناصر من

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
Hosini_list = {"JavaScript", "Python", "Node.js"}
```

```
Hosini_set.update(Hosini_list)
```

```
print(Hosini_set)
```


الدالة update

ليس من الضروري أن يكون الكائن الموجود في الدالة **update()** مجموعة، بل يمكن أن يكون أي كائن قابل للنسخ (صفوف، قوائم، قواميس وما إلى ذلك).

مثال ↘ ↘

إضافة عناصر القائمة إلى المجموعة

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
Hosini_list = ["AlBadrashin", "Egypt"]
```

```
Hosini_set.update(Hosini_list)
```

```
print(Hosini_set)
```



إزالة عنصر في مجموعة، disHosini_Data() أو remove() استخدم الدالة td().

مثال ↘ ↘

الدالة **remove()** إزالة "حبيب" باستخدام

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
Hosini_set.remove("Habib")
```

```
print(Hosini_set)
```

فسيظهر **remove()**، ملاحظة: إذا كان العنصر المراد إزالته غير موجود خطأ.

الدالة DisHosini_Datd

مثال ↘ ↘

الدالة **disHosini_Datd()** لإزالة "حبيب" باستخدام

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
Hosini_set.disHosini_Datd("Habib")
```

```
print(Hosini_set)
```

ملاحظة: إذا كان العنصر المراد إزالته غير موجود، فلن يظهر خطأ **disHosini_Datd()**، موجود

الدالة لإزالة عنصر ما، ولكن هذه الدالة **pop()** تستطيع أيضاً استخدام ستزيل العنصر الأخير. تذكر أن المجموعات غير مرتبة، لذلك لن تعرف العنصر الذي سيتم إزالته.

هي العنصر الذي تمت إزالته **pop()** القيمة المرجعة للدالة

مثال ↘ ↘

قم بإزالة العنصر الأخير الدالة pop() باستخدام

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
x = Hosini_set.pop()
```

```
print(x)
```

```
print(Hosini_set)
```

الدالة، **pop()** ملاحظة: المجموعات غير مرتبة ، لذا عند استخدام **pop()** فإنك لا تعرف العنصر الذي سيتم إزالته.

مثال ↘ ↘

تفرغ المجموعة **clear()** الدالة

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

```
Hosini_set.clear()
```


| `print(Hosini_set)`

مثال ↘ ↘

: ستحذف الكلمة المحجوزة المجموعة بالكامل `del`

| `Hosini_set = {"Abo", "Habib", "Al Hosiny"}`

| `del Hosini_set`

| `print(Hosini_set)`

✓ الحلقات على المجموعات

عناصر الحلقة

: حلقة `for` تستطيع تكرار العناصر المحددة باستخدام

مثال ↘ ↘

: قم بالمرور عبر المجموعة وطباعة القيم

```
Hosini_set = {"Abo", "Habib", "Al Hosiny"}
```

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

الدالة union

هناك عدة دوال لضم مجموعتين أو أكثر في بايثون.

الدالة التي تُرجع مجموعة جديدة تحتوي **union()** تستطيع استخدام التي **update()** على جميع العناصر من كلتا المجموعتين، أو الدالة تُدرج جميع العناصر من مجموعة إلى أخرى:

مثال ↘ ↘

بإرجاع مجموعة جديدة تحتوي على جميع **union()** تقوم الدالة العناصر من كلتا المجموعتين:

```
Hosini1 = {"a", "b", "c"}
```

```
Hosini2 = {1, 2, 3}
```

```
Hosini3 = Hosini1.union(Hosini2)
```

```
print(Hosini3)
```


مثال ↘ ↘

في Hosini2 بإدراج العناصر الموجودة في **update()** تقوم الدالة Hosini1:

```
Hosini1 = {"a", "b", "c"}
```

```
Hosini2 = {1, 2, 3}
```

```
Hosini1.update(Hosini2)
```

```
print(Hosini1)
```

استبعاد أي عناصر نسخة **update()** سيتم **union()**: ملاحظة

جلب التطابق

فقط بالعناصر الموجودة **intersection_update()** ستحتفظ الدالة في كلتا المجموعتين.

مثال ↘ ↘

set **x** و set **y** احتفظ بالعناصر الموجودة في كل من

```
x = {"Abo", "Habib", "Al Hosiny"}
```

```
y =
```

```
{"PHP", "C#", "Java", "python", "XML", "HTML5", "  
Visual Basic 6"}
```

```
x.intersection_update(y)
```

```
print(x)
```

ستُرجع الدالة مجموعة جديدة تحتوي فقط على العناصر الموجودة في كلتا المجموعتين.

مثال ↘ ↘

set x قم بإرجاع مجموعة تحتوي على العناصر الموجودة في كل من set y و

```
x = {"Abo", "Habib", "Al Hosiny"}
```

```
y = {"PHP", "C#", "Abo"}
```

```
z = x.intersection(y)
```

```
print(z)
```


جلب الاختلاف

فقط `symmetric_difference_update()` ستحتفظ الدالة بالعناصر غير الموجودة في كلتا المجموعتين.

مثال ↘ ↘

احتفظ بالعناصر غير الموجودة في المجموعتين:

```
x = {"Abo", "Habib", "Al Hosiny"}
```

```
y = {"PHP", "C#", "Abo"}
```

```
x.symmetric_difference_update(y)
```

```
print(x)
```

مجموعة جديدة تحتوي `symmetric_difference()` ستُرجع الدالة فقط على العناصر غير الموجودة في كلتا المجموعتين.

مثال ↘ ↘

إرجاع مجموعة تحتوي على كافة العناصر من كلتا المجموعتين، باستثناء العناصر الموجودة في كليهما:

```
x = {"Abo", "Habib", "Al Hosiny"}
```

```
y = {"PHP", "C#", "Abo"}
```

```
z = x.symmetric_difference(y)
```

```
print(z)
```

تعيين الأساليب والفتكشن

تعيين الأساليب والفتكشن

فى لغة بايثون مجموعة من الأساليب والفتكشن المضمنه فى اللغة التي تستطيع استخدامها فى المجموعات.

Method	وصف
add()	يضيف عنصرًا إلى المجموعة
clear()	يزيل كافة العناصر من المجموعة
copy()	إرجاع نسخة من المجموعة
difference()	إرجاع مجموعة تحتوي على الفرق بين مجموعتين أو أكثر
difference_update()	إزالة العناصر الموجودة في هذه

	المجموعة والمضمنة أيضًا في مجموعة أخرى محددة
<u>disHosini_Datd()</u>	قم بإزالة العنصر المحدد
<u>intersection()</u>	تُرجع مجموعة، وهي تقاطع مجموعتين أخريين
<u>intersection_update()</u>	إزالة العناصر الموجودة في هذه المجموعة والتي لا توجد في مجموعة (مجموعات) أخرى محددة
<u>isdisjoint()</u>	إرجاع ما إذا كانت مجموعتان لهما تقاطع أم لا
<u>issubset()</u>	إرجاع ما إذا كانت مجموعة أخرى تحتوي على هذه المجموعة أم لا
<u>issuperset()</u>	إرجاع ما إذا كانت هذه المجموعة تحتوي على مجموعة أخرى أم لا
<u>pop()</u>	إزالة عنصر من المجموعة
<u>remove()</u>	يزيل العنصر المحدد
<u>symmetric_difference()</u>	إرجاع مجموعة ذات فروق متماثلة بين مجموعتين
<u>symmetric_difference_update()</u>	يُدْرَج الاختلافات المتماثلة من هذه المجموعة وأخرى
<u>union()</u>	إرجاع مجموعة تحتوي على اتحاد المجموعات

update()

قم بتحديث المجموعة باتحاد هذه المجموعة وغيرها

الدوال التي لم يتم شرحها ستجدها في الجزء الثاني من الكتاب باذن
الله تعالى

قواميس بايثون

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 🇸🇩 🇸🇩",  
  "Books": " JavaScript, C++ , Node.js , PYthon",  
  "year": 9090  
}
```

قاموس

تُستخدم القواميس لتخزين قيم الاكواد في أزواج المفتاح: القيمة
القاموس عبارة عن مجموعة مرتبة* وقابلة للتغيير ولا تسمح
بالتكرارات .

اعتبراً من الإصدار 3.7 من بايثون، تم ترتيب القواميس . في بايثون
3.6 والإصدارات السابقة، تكون القواميس غير مرتبة .

:تتم كتابة القواميس بأقواس معقوفة، ولها مفاتيح وقيم

مثال ↘ ↘

إنشاء وطباعة القاموس:

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 😊 😊",  
  "Books": " JavaScript, C++ , Node.js , PYthon",  
  "year": 9090  
}  
print(Hosini_Data)
```

عناصر القاموس

عناصر القاموس مرتبة وقابلة للتغيير ولا تسمح بالنسخ.

يتم عرض عناصر القاموس في أزواج المفتاح: القيمة، ويمكن الإشارة إليها باستخدام اسم المفتاح.

مثال ↘ ↘

اطبع قيمة "الكتب" في القاموس:

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 😊 😊",
```

```
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}  
print(Hosini_Data["name"])
```

الترتيب

اعتبرًا من الإصدار 3.7 من بايثون، تم ترتيب القواميس . في بايثون 3.6 والإصدارات السابقة، تكون القواميس غير مرتبة .

عندما نقول أن القواميس مرتبة، فهذا يعني أن العناصر لها ترتيب محدد، وهذا الترتيب لن يتغير .

غير مرتبة يعني أن العناصر ليس لها ترتيب محدد، ولا تستطيع الرجوع إلى عنصر باستخدام الرقم .

قابل للتغيير

القواميس قابلة للتغيير، مما يعني أنه نستطيع تغيير العناصر أو إضافتها أو إزالتها بعد إنشاء القاموس .

التكرارات غير مسموح بها

لا يمكن أن تحتوي القواميس على عنصرين بنفس المفتاح

مثال ↘ ↘

ستحل القيمة محل القيم الموجودة

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 😊 😊",  
  "Books": " JavaScript, C++ , Node.js , PYthon",  
  "year": 4080,  
  "year": 9009  
}  
print(Hosini_Data)
```

طول القاموس

`len()` لتحديد عدد العناصر الموجودة في القاموس، استخدم الدالة

مثال ↘ ↘

اطبع عدد العناصر في القاموس

| print(len(Hosini_Data))

أنواع البيانات في عناصر القاموس -

يمكن أن تكون القيم الموجودة في عناصر القاموس من أي نوع بيانات :

مثال ↘ ↘

نوالمنطقية، والقائمة، intأنواع اكواد النصوص ، و

```
Hosini_Data= {  
    "name": "Abu Habib Al Hosiny 😊 😊",  
    "Commands": False,  
    "year": 1954,  
    "Lang": ["AR", "EN"]  
}
```

من وجهة نظر بايثون، يتم تعريف القواميس ككائنات بنوع الاكواد "dict":

<class 'dict'>

مثال ↘ ↘

طباعة نوع بيانات القاموس:

```
Hosini_Data= {  
    "name": "Abu Habib Al Hosiny 😊 😊",  
    "Books": " JavaScript, C++ , Node.js , PYTHON XML  
HTML5 JavaScript ",  
    "year": 3080  
}  
print(type(Hosini_Data))
```

dict() منشئ القاموس

لإنشاء قاموس **dict()** من الممكن أيضاً استخدام المُنشئ

مثال ↘ ↘

لإنشاء قاموس **dict()** باستخدام دالة

```
Hosini_Data= dict(name = "Abo Habib", age = 35,  
Books = "python VBA")  
print(Hosini_Data)
```

مثال ↘ ↘

: احصل على قيمة المفتاح

```
Hosini_Data= {  
"name": "Abu Habib Al Hosiny 😊 😊",  
"Books": " JavaScript, C++ , PYthon Node.js VBA  
VBS VB.net",  
"year": 3030  
}  
x = Hosini_Data["Books"]
```

نوالتي ستعطيك نفس ال **get()** هناك أيضًا دالة تسمى

مثال ↘ ↘

:"احصل على قيمة مفتاح الكتب


```
x = Hosini_Data.get("Books")
```

احصل على المفاتيح

قائمة بجميع المفاتيح الموجودة في القاموس `keys()` ستعيد الدالة

مثال ↘ ↘

قائمة المفاتيح ومعنى المفتاح اي معرف العنصر ربما يكون رقم او نص

```
x = Hosini_Data.keys()
```

قائمة المفاتيح هي عرض للقاموس، مما يعني أن أي تغييرات يتم إجراؤها على القاموس سوف تنعكس في قائمة المفاتيح

مثال ↘ ↘

أضف عنصرًا جديدًا إلى القاموس الأصلي، وتأكد من تحديث قائمة المفاتيح أيضًا:

```
Hosini_Dat = {
```

```
"name": "Abu Habib Al Hosiny 🇸🇩 🇸🇩",
```

```
"Books": " JavaScript, C++ , Node.js , PYthon",
```

```
| "year": 9090
| }

| x = Hosini_Dat.keys()

| print(x) #before the change

| Hosini_Dat["name"] = "white"

| print(x) #after the change
```

القيم

قائمة بجميع القيم الموجودة في القاموس **values()** ستُرجع الدالة

مثال ↙ ↘

قائمة القيم:

```
| x = Hosini_Data.values()
```

قائمة القيم هي عرض للقاموس، مما يعني أن أي تغييرات يتم إجراؤها على القاموس سوف تنعكس في قائمة القيم.

مثال ↘ ↘

قم بإجراء تغيير في القاموس الأصلي، وتأكد من تحديث قائمة القيم أيضًا:

```
Hosini_Dat = {  
"name": "Abu Habib Al Hosiny 😊 😊",  
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}  
  
x = Hosini_Dat.values()  
  
print(x) #before the change  
  
Hosini_Dat["year"] = 9009  
  
print(x) #after the change
```

مثال ↘ ↘

أضف عنصرًا جديدًا إلى القاموس الأصلي، وتأكد من تحديث قائمة القيم أيضًا:

```
Hosini_Dat = {  
"name": "Abu Habib Al Hosiny 😊 😊",  
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}
```

```
x = Hosini_Dat.values()
```

```
print(x) #before the change
```

```
Hosini_Dat["name"] = "Abu Habib Al Hosiny 😊 😊"
```

```
print(x) #after the change
```

العناصر

كل عنصر في القاموس، كصفوف في القائمة **items()** ستعيد الدالة

مثال ↙ ↘

احصل على قائمة بالمفتاح: أزواج القيمة

```
x = Hosini_Data.items()
```


القائمة التي تم إرجاعها هي عرض لعناصر القاموس، مما يعني أن أي تغييرات يتم إجراؤها على القاموس سوف تنعكس في قائمة العناصر.

مثال ↘ ↘

قم بإجراء تغيير في القاموس الأصلي، وتأكد من تحديث قائمة العناصر أيضًا:

```
Hosini_Dat = {  
"name": "Abu Habib Al Hosiny 😊 😊",  
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}
```

```
x = Hosini_Dat.items()
```

```
print(x) #before the change
```

```
Hosini_Dat["year"] = 9009
```

```
print(x) #after the change
```

مثال ↘ ↘

أضف عنصرًا جديدًا إلى القاموس الأصلي، وتأكد من تحديث قائمة العناصر أيضًا:

```
Hosini_Dat = {  
"name": "Abu Habib Al Hosiny 😊 😊",  
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}  
  
x = Hosini_Dat.items()  
  
print(x) #before the change  
  
Hosini_Dat["name"] = "Al_Masry"  
  
print(x) #after the change
```

تحقق من وجود مفتاح

لتحديد ما إذا كان هناك مفتاح محدد موجود في القاموس، استخدم
المحجوزة **in** الكلمة:

مثال ↘ ↘

تحقق مما إذا كان "النموذج" موجودًا في القاموس

```
Hosini_Data= {  
    "name": "Abu Habib Al Hosiny 😊 😊",  
    "Books": " JavaScript, C++ , Node.js , PYthon",  
    "year": 9090  
}  
if "Books" in Hosini_Data:  
    print("Yes, 'Books' is one of the keys in the  
    Hosini_Datadictionary")
```

✓ تغيير عناصر القاموس

تغيير القيم

تستطيع تغيير قيمة عنصر معين من خلال الإشارة إلى اسمه الرئيسي

مثال ↘ ↘

تغيير "السنة" إلى 2018

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 😊 😊",  
  "Books": " JavaScript, C++ , Node.js , PYthon",  
  "year": 9090  
}  
Hosini_Data["year"] = 2018
```

✓ إضافة عناصر القاموس

إضافة العناصر

تتم إضافة عنصر إلى القاموس باستخدام المفتاح وتعيين قيمة له

مثال ↙ ↘

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 😊 😊",  
  "Books": " JavaScript, C++ , Node.js , PYthon",  
  "year": 9090
```



```
}  
Hosini_Data["name"] = "Al_Hosiny"  
print(Hosini_Data)
```

تحديث القاموس

بتحديث القاموس بالعناصر من وسيطة **update()** ستقوم الدالة معينة. إذا كان العنصر غير موجود، سيتم إضافة العنصر يجب أن تكون الوسيطة عبارة عن قاموس، أو كائن قابل للنسخ مع أزواج المفتاح: القيمة.

مثال ↘ ↘

update() أضف عنصر لون إلى القاموس باستخدام الدالة :

```
Hosini_Data= {  
    "name": "Abu Habib Al Hosiny 😊 😊",  
    "Books": " JavaScript, C++ , Node.js , PYthon",  
    "year": 9090  
}  
Hosini_Data.update({"name": "Al_Hosiny"})
```

إزالة عناصر القاموس

إزالة العناصر

هناك عدة دوال لإزالة العناصر من القاموس:

مثال ↘ ↘

بإزالة العنصر باسم المفتاح المحدد **pop()** تقوم الدالة

```
Hosini_Data= {  
  "name": "Abu Habib Al Hosiny 😊 😊",  
  "Books": " JavaScript, C++ , Node.js , PYthon",  
  "year": 9090  
}  
Hosini_Data.pop("Books")  
print(Hosini_Data)
```


مثال ↘ ↘

بإزالة آخر عنصر تم إدراجه (في الإصدارات **popitem()** تقوم الدالة قبل 3.7، تتم إزالة عنصر عشوائي بدلاً من ذلك)

```
Hosini_Data= {  
    "name": "Abu Habib Al Hosiny 😊 😊",  
    "Books": " JavaScript, C++ , Node.js , PYthon",  
    "year": 9090  
}  
Hosini_Data.popitem()  
print(Hosini_Data)
```

مثال ↘ ↘

المحجوزة بإزالة العنصر الذي يحمل اسم المفتاح **del** تقوم الكلمة المحدد:

```
Hosini_Data= {  
    "name": "Abu Habib Al Hosiny 😊 😊",  
    "Books": " JavaScript, C++ , Node.js , PYthon",  
    "year": 9090  
}
```

```
del Hosini_Data["Books"]
print(Hosini_Data)
```

مثال ↘ ↘

:المحجوزة أيضًا حذف القاموس بالكامل **del** يمكن للكلمة

```
Hosini_Data= {
  "name": "Abu Habib Al Hosiny 😊 😊",
  "Books": " JavaScript, C++ , Node.js , PYthon",
  "year": 9090
}
del Hosini_Data
print(Hosini_Data) #this will cause an error
because "Hosini_Data" no longer exists.
```

مثال ↘ ↘

:تفريغ القاموس **clear()** الدالة

```
Hosini_Data= {
  "name": "Abu Habib Al Hosiny 😊 😊",
  "Books": " JavaScript, C++ , Node.js , PYthon",
```



```
"year": 9090
```

```
}
```

```
Hosini_Data.clear()
```

```
print(Hosini_Data)
```

✓ الحلقات على القواميس

حلقة بسيطة على القاموس

حلقة **for** تستطيع تكرار القاموس باستخدام

عند النسخ عبر القاموس، تكون القيمة المرجعة هي مفاتيح القاموس، ولكن هناك دوال لإرجاع القيم أيضًا.

مثال ↘ ↘

:اطبع جميع أسماء المفاتيح في القاموس، واحدًا تلو الآخر

```
for x in Hosini_Data:
```

```
print(x)
```

مثال ↘ ↘

:اطبع جميع القيم الموجودة في القاموس واحدة تلو الأخرى

```
for x in Hosini_Data:  
    print(Hosini_Data[x])
```

مثال ↘ ↘

:الدالة لإرجاع قيم القاموس **values()** تستطيع أيضًا استخدام

```
for x in Hosini_Data.values():  
    print(x)
```

مثال ↘ ↘

:الدالة لإرجاع مفاتيح القاموس **keys()** تستطيع استخدام

```
for x in Hosini_Data.keys():  
    print(x)
```


مثال ↘ ↘

items(): قم بالنسخ عبر المفاتيح والقيم باستخدام الدالة

```
for x, y in Hosini_Data.items():  
    print(x, y)
```

✓ نسخ القواميس

انسخ قاموساً

Hosini2 = Hosini1, لا تستطيع نسخ قاموس بمجرد كتابة والتغييرات التي يتم **Hosini1** سيكون مرجعاً فقط إلى **Hosini2**: لأن **Hosini2** سيتم إجراؤها تلقائياً في **Hosini1** إجراؤها.

هناك دوال لإنشاء نسخة، إحدى الدوال هي استخدام دالة القاموس **copy()**.

مثال ↘ ↘

copy(): قم بعمل نسخة من القاموس بالدالة

```
Hosini_Data = {  
    "name": "Abu Habib Al Hosiny 😊 😊",
```

```
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}  
Hosini_dict = Hosini_Data.copy()  
print(Hosini_dict)
```

dict(). هناك دالة أخرى لعمل نسخة وهي استخدام الدالة

مثال ↘ ↘

dict() : قم بعمل نسخة من القاموس باستخدام الدالة

```
Hosini_Data= {  
"name": "Abu Habib Al Hosiny 😊 😊",  
"Books": " JavaScript, C++ , Node.js , PYthon",  
"year": 9090  
}  
Hosini_dict = dict(Hosini_Data)  
print(Hosini_dict)
```

✓ القواميس المتداخلة

قواميس متداخلة

يمكن أن يحتوي القاموس على قواميس، وهذا ما يسمى القواميس المتداخلة.

مثال ↘ ↘

إنشاء قاموس يحتوي على ثلاثة قواميس:

```
Hosini_family = {  
  "child1" : {  
    "name" : "Habib",  
    "year" : 3004  
  },  
  "child2" : {  
    "name" : "Ahmed",  
    "year" : 3007  
  },  
  "child3" : {  
    "name" : "Mohamed",  
    "year" : 3011  
  }  
}
```

```
"child4" : {  
  "name" : "Omar",  
  "year" : 3011  
}
```

```
}
```

```
}
```

أو، إذا كنت تريد إضافة ثلاثة قواميس إلى قاموس جديد

مثال ↘ ↘

قم بإنشاء ثلاثة قواميس، ثم قم بإنشاء قاموس واحد يحتوي على القواميس الثلاثة الأخرى:

```
child1 = {  
  "name": "Habib",  
  "year": 2004  
}  
child2 = {  
  "name": "Ahmed",  
  "year": 2007  
}  
child3 = {  
  "name": "Omar",  
  "year": 3011  
}
```

```
Hosini_family = {  
  "child1": child1,
```



```
"child2" : child2,
```

```
"child3" : child3
```

```
}
```

✓ دوال قاموس بايثون

دوال القاموس

في لغة بايثون مجموعة من الأساليب والฟังก์شن التي تستطيع استخدامها في القواميس.

Method	وصف
clear()	يزيل كافة العناصر من القاموس
copy()	إرجاع نسخة من القاموس
fromkeys()	إرجاع قاموس بالمفاتيح المحددة
get()	إرجاع قيمة المفتاح المحدد
items()	تقوم بإرجاع قائمة تحتوي على صف لكل زوج من قيم المفاتيح
keys()	إرجاع قائمة تحتوي على مفاتيح القاموس
pop()	إزالة العنصر بالمفتاح المحدد
popitem()	يزيل آخر زوج من قيمة المفتاح الذي تم إدراجه
setdefault()	إرجاع قيمة المفتاح المحدد. إذا كان المفتاح غير موجود: أدخل المفتاح بالقيمة المحددة

<u>update()</u>	يقوم بتحديث القاموس بأزواج القيمة الرئيسية المحددة
<u>values()</u>	إرجاع قائمة بجميع القيم الموجودة في القاموس

الدوال التي لم يتم شرحها سيتم شرحها في الجزء الثانى ان شاء الله تعالى

كيف وضع شروط

تدعم بايثون الشروط المنطقية المعتادة من الرياضيات: مثل اى لغة برمجة

•
. باستخدام الكلمة "if"

مثال

إذا كلمة

a = 33

b = 200


```
if b > a:
```

```
print("Abo Habib _ Al Hosini ")
```

يتم استخدامهما ، **b** و **a** في هذا ال **مثال** ↘ ↘ ، نستخدم متغيرين هو **33** **a** نظرًا لأن **a** أكبر من **b** لاختبار ما إذا كان **if** كجزء من عبلة هو **200** ، فإننا نعلم أن **200** أكبر من **33**، ولذلك نطبع على **b** و **a** أكبر من **b** الشاشة أن.

نكرر راعي المسافات البادئة في بايثون

تعتمد بايثون على المسافة البادئة (مسافة في بداية السطر) لتحديد النطاق في الكود. غالبًا ما تستخدم لغات البرمجة الأخرى الأقواس المتعرجة لهذا الغرض.

↘ ↘ **مثال**

إذا كانت العبلة، بدون مسافة بادئة (ستؤدي إلى ظهور خطأ)

```
a = 33
```

```
b = 200
```

```
if b > a:
```

```
print("Abo Habib _ Al Hosini ") # you will get an  
error
```

elif

هي دالة بايثون لقول "إذا كانت الشروط السابقة **elif** الكلمة المحجوزة "غير صحيحة، فجرب هذا الشرط".

مثال ↘ ↘

```
a = 33
```

```
b = 33
```

```
if b > a:
```

```
    print("Abo Habib _ Al Hosini ")
```

```
elif a == b:
```

```
    print("Al Badrasheen")
```

لذا فإن الشرط الأول ليس صحيحًا، **b** يساوي **a**، في هذا المثال "متساويان **a** و **b**" صحيح، لذلك نطبع على الشاشة أن **elif** ولكن شرط

أي شيء لم يتم اكتشافه بالشروط **else** تلتقط الكلمة المحجوزة السابقة .

مثال ↘ ↘

```
a = 200
b = 33
if b > a:
    print("Abo Habib _ Al Hosini ")
elif a == b:
    print("Al Badrasheen")
else:
    print("JavaScript")
```

وبالتالي فإن الشرط الأول غير ، **b** أكبر من **a** ↘ ↘ في هذا ال مثال غير صحيح، لذلك ننتقل إلى **elif** صحيح، وكذلك شرط "**b** أكبر من **a**" ونطبع على الشاشة أن **else** الشرط **elif** بدون **else** تستطيع أيضًا

مثال ↘ ↘

```
a = 200
b = 33
if b > a:
    print("Abo Habib _ Al Hosini ")
else:
    print("Al Badrasheen_")
```

إذا كان موجود عبارة واحدة فقط تريد تنفيذها، فتستطيع وضعها في نفس if سطر عبارة

مثال ↘ ↘

سطر واحد إذا كان الكلمة

```
| if a > b: print("Abo Habib Al_Hosiny")
```

وواحدة لـ if، إذا كان موجود عبارة واحدة فقط تريد تنفيذها، وواحدة لـ else، فتستطيع وضعها كلها في نفس السطر

مثال ↘ ↘

سطر واحد إذا عبارة أخرى

```
| a = 2  
| b = 330  
| print("A") if a > b else print("B")
```


تُعرف هذه التقنية باسم عوامل التشغيل الثلاثية أو التعبيرات الشرطية .

تستطيع أيضًا عدة عبارات أخرى في نفس السطر

مثال ↘ ↘

مع 3 شروط، **else** سطر واحد إذا عبارة

```
a = 330
```

```
b = 330
```

```
print("A") if a > b else print("=") if a ==
```

```
b else print("B")
```

إضافة شروط جديدة الى القديمة

عاملاً منطقيًا، وتُستخدم للجمع بين **and** تعد الكلمة المحجوزة العبارات الشرطية:

مثال ↘ ↘

a: كان أكبر من **c** وإذا **b** كان أكبر من **a** اختبار إذا

```
a = 200
```

```
b = 33
```

```
c = 500
```

```
if a > b and c > a:
```

```
print("Abo Habib _ Al Hosini 😊😊😊😊")
```

كلمة او للخيار بين شرطين

المحجوزة هي عامل منطقي، وتستخدم للجمع بين العبارات **or** الكلمة الشرطية:

مثال ↘ ↘

c: كان أكبر من **a** أو إذا **b** كان أكبر من **a** اختبار إذا

```
a = 200
```

```
b = 33
```

```
c = 500
```

```
if a > b or a > c:
```

```
print("Abo Habib _ Al Hosini 😊😊")
```


عمل شروط متداخلة

عبارات، وهذا ما يسمى **if** عبارات داخل **if** يمكن أن يكون موجود **if** . العبارات المتداخلة

مثال ↘ ↘

```
x = 41
```

```
if x > 10:
```

```
    print("Above ten,")
```

```
    if x > 20:
```

```
        print("and also above 20!")
```

```
    else:
```

```
        print("but not above 20.")
```

كلمة المرور

لا يمكن أن تكون الاكواد فارغة، ولكن إذا كان موجود كلمة بدون **ifif** . الكلمة لتجنب حدوث خطأ **pass** محتوى لسبب ما ، فقم بإدخال

مثال ↘ ↘

```
a = 33
```

```
b = 200
```

```
if b > a:
```

```
pass
```

✓ الحلقات

حلقة بينما

نستطيع تنفيذ مجموعة من العبارات طالما كان **while** باستخدام حلقة الشرط صحيحًا.

مثال ↘ ↘

طالما أن أقل من 6 i اطبع:

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    i += 1
```

وإلا ستستمر الحلقة إلى الأبد، i ملحوظة: تذكر أن تزيد

المتغيرات ذات الصلة لتكون جاهزة، في هذا ال **while** تتطلب حلقة الذي قمنا بتعيينه ، **i**، مثال \ \ نحتاج إلى تعريف متغير الرقمة على 1

كلمة الإستراحة

نستطيع إيقاف الحلقة حتى لو كان شرط **Break** باستخدام عبارة **while** صحيحًا:

مثال \ \

الخروج من الحلقة عندما يكون رقم 3

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

كلمة الاستمرار

نستطيع تخطي العنصر الحالي من النسخ **continue** باستخدام عبارة:
الحالي والاستمرار في التالي:

مثال ↘ ↘

تابع إلى النسخ التالي إذا كان الرقم 3:

```
i =  
while i < 6:  
    i += 1  
    if i == 3:  
        continue  
    print(i)
```

إذا تحقق الشرط افعل كذا وإلا افعل كذا

نستطيع تشغيل كتلة من الاكواد مرة واحدة **else** باستخدام عبارة:
عندما يصبح الشرط غير صحيح:

مثال ↘ ↘

اطبع رسالة عندما يكون الشرط خاطئاً:


```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

ضوابط للحلقات

للسخ عبر تسلسل (إما قائمة، أو صف، أو **for** يتم استخدام حلقة قاموس، أو مجموعة، أو نص).

في لغات البرمجة الأخرى، **for** هذا أقل تشابهًا مع الكلمة المحجوزة ويعمل بشكل أشبه بدالة السخ كما هو موجود في لغات البرمجة الموجهة للكائنات الأخرى.

نستطيع تنفيذ مجموعة من العبارات، مرة واحدة، **for** باستخدام حلقة لكل عنصر في القائمة، أو صف، أو مجموعة، وما إلى ذلك.

مثال ↙ ↘

:اطبع كل قائمة في قائمة القائمة

```
HosHos = ["Abo", "Habib", "Al Hosiny 🤪🤪"]  
for x in HosHos:  
    print(x)
```

. تعيين متغير رقمة مسبقاً for لا تتطلب حلقة

نستطيع إيقاف Break باستخدام عبارة
الحلقة قبل أن يتم تكرارها عبر جميع
العناصر:

مثال ↘ ↘

:" يكون "الـ x الخروج من الحلقة عندما

```
HosHos = ["Abo", "Habib", "Al Hosiny 🤪🤪🤪"]  
for x in HosHos:  
    print(x)  
    if x == "Habib":  
        break
```


مثال ↘ ↘

يكون، ولكن هذه المرة يأتي الفاصل قبل **x** اخرج من الحلقة عندما الطباعة:

```
HosHos = ["Abo", "Habib", "Al Hosiny 😊😊"]
for x in HosHos:
    if x == "Habib":
        break
    print(x)
```

نستطيع إيقاف `continue` باستخدام عبارة النسخ الحالي للحلقة، والاستمرار في العبارة التالية:

مثال ↘ ↘

لا تطبع ال

```
HosHos = ["Abo", "Habib", "Al Hosiny 🙏🙏"]
for x in HosHos:
```

```
if x == "Habib":  
    continue  
print(x)
```

range () دالة

لنسخ عبر مجموعة من الاكواد لعدد محدد من المرات، نستطيع
استخدام الدالة **range()** ،

نص من الأرقام، تبدأ من 0 افتراضياً، وتزيد **range ()** ترجع الدالة
بمقدار 1 (افتراضياً)، وتنتهي عند رقم محدد.

مثال ↘ ↘

(): باستخدام دالة النطاق

```
for x in range(6):  
    print(x)
```

لاحظ أن النطاق (6) ليس القيم من 0 إلى 6، بل القيم من 0 إلى 5

هي 0 كقيمة البداية، ومع ذلك فمن **range()** القيمة الافتراضية للدالة **range(2, 6)** ، الممكن تحديد قيمة البداية عن طريق إضافة معلمة نواتي تعني القيم من 2 إلى 6 (ولكن لا تشمل 6)

مثال ↘ ↘

باستخدام معلمة البداية:

```
for x in range(2, 6):  
    print(x)
```

افتراضياً بزيادة المصفوفات بمقدار 1، ومع ذلك **range()** تقوم الدالة فمن الممكن تحديد قيمة الزيادة عن طريق إضافة معلمة **range(2, 30, 3)** :
ثالثة:

مثال ↘ ↘

قم بزيادة المصفوفات بـ 3 (الافتراضي هو 1)

```
for x in range(2, 30, 3):  
    print(x)
```

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

مثال ↘ ↘

:اطبع جميع الأرقام من 0 إلى 5، واطبع رسالة عند انتهاء الحلقة

```
for x in range(6):  
    print(x)  
else:  
    print("Abo Habib Al_Hosiny 😊")
```

break إذا تم إيقاف الحلقة بواسطة **else** ملحوظة: لن يتم تنفيذ الكتلة عبارة.

مثال ↘ ↘

else: تكون 3، وانظر ماذا سيحدث للكتلة **x** اكسر الحلقة عندما

```
for x in range(6):  
    if x == 3: break  
    print(x)
```


else:

```
print("Abo Habib Al_Hosiny 😊")
```

حلقات متداخلة

الحلقة المتداخلة هي حلقة داخل حلقة.

:سيتم تنفيذ "الحلقة الداخلية" مرة واحدة لكل تكرار للحلقة الخارجية

مثال ↘ ↘

اطبع كل قيمة لكل قائمة عند مرور الحلقة عليها

```
Hosini_listn = ["Al_Hosiny", "python", "Al  
Badrasheen"]
```

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

```
for x in Hosini_listn:
```

```
    for y in HosHos:
```

```
        print(x, y)
```

كلمة المرور على حلقة فور

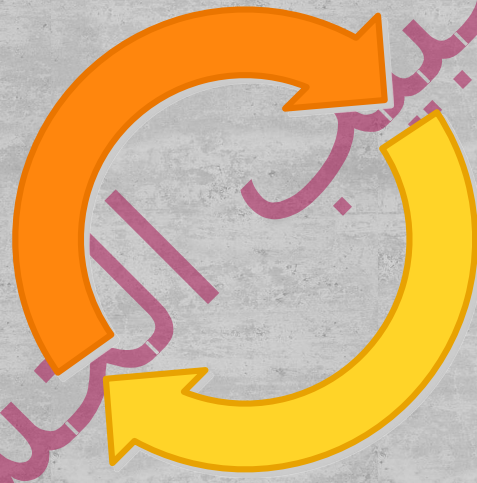
لا يمكن أن تكون الحلقات فارغة، ولكن إذا كان موجود لسبب ما **for** العبارة لتجنب حدوث خطأ **pass** بدون محتوى، فقم بإدخال **for** حلقة

مثال ↘ ↘

```
for x in [0, 1, 2]:  
    pass
```

أبو حبيب الحسيني

أبو حبيب
الهرشيبيني



التكملة في الجزء الثاني باذن الله تعالى