

# فك الشيفرات السرية بلغة البايثون

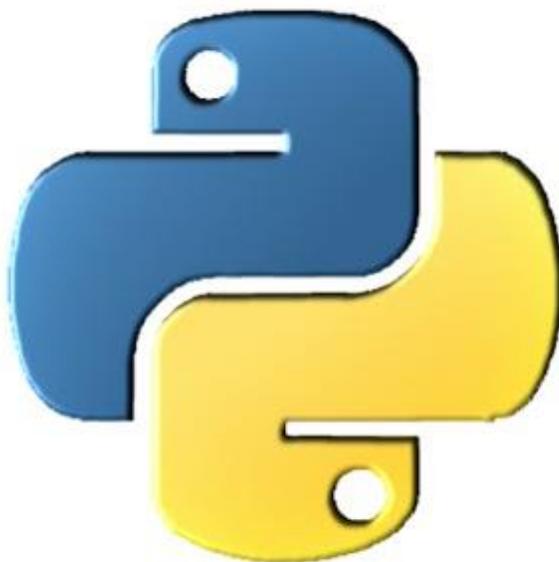
دليل عملي لكتابه برمج التشفير وفك التشفير بلغة البرمجة بايثون



المهندس: جميل حسين طويله

# فأك الشيفرات السرية بلغة البايثون

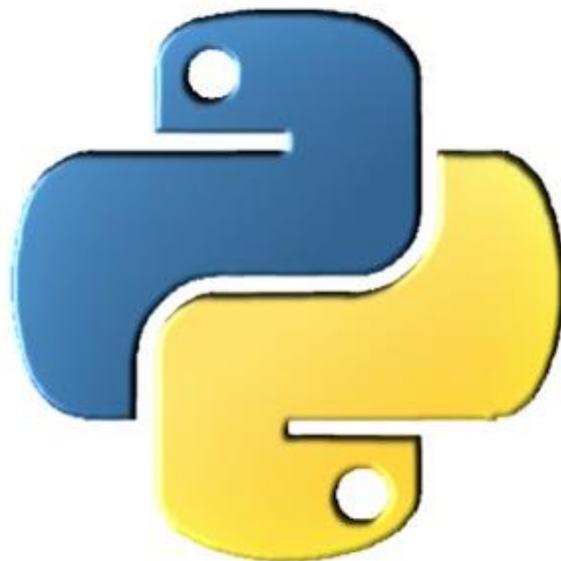
---



جميل حسين طويله

# Hacking Secret Ciphers with Python

---



Jameel Huseen Tawelh



الأفلام والعرض التلفزيونية تجعل عملية الاختراق تبدو مثيرة من خلال عمليات إدخال التعليمات وظهور سيل من الأصفار والواحدات على الشاشة.

هم يجعلون عملية الاختراق تبدو كشيء يجب أن تكون عبقي لتعلمها.

هم يجعلون الاختراق يبدو كالسحر، هو ليس سحر، هو يعتمد على أجهزة الكمبيوتر وكل عمل يقوم به الكمبيوتر هو عمل مبني على قاعدة منطقية يمكن تعلمها وفهمها.

ليس من الصعب التعلم.

في هذا الكتاب أنا أفترض أنك لا تعرف شيء عن التشفير cryptography أو البرمجة programming وهذا الكتاب سوف يساعدك لتعلم خطوة بخطوة كيفية كتابة البرامج التي تقوم بكسر وفك شيفرة الرسائل المشفرة.

استمتع ...

## حول هذا الكتاب:

يوجد العديد من الكتب التي تعلم المبتدئين كيفية كتابة الرسائل السرية باستخدام الشифرات كما يوجد كتب أخرى تعلم المبتدئين كيفية فك الشيفرة أو كسر الشيفرة **hack cipher** ولكن لا يوجد كتب تعلم المبتدئين كيفية كتابة برامج تقوم بفك الشيفرة، هذا الكتاب هو الأول في هذا المجال.

هذا الكتاب هو للمبتدئين الذين لا يعرفون أي شيء عن التشفير **encryption** أو الاختراق **.cryptography** أو كتابة الشيفرة وفكها (**علم التشفير**) **hacking**.

الشيفرات في هذا الكتاب (باستثناء شيفرة RSA cipher في الفصل الأخير) هي كلها شيفرات من القرن الماضي وأجهزة الكمبيوتر الحديثة الآن تملك القدرة الحسابية لكسر أو فك الرسائل المشفرة.

لم تعد المنظمات أو الأشخاص يستخدمون هذه الشيفرات وبالتالي لن تواجه أي مشاكل قانونية بسبب المعلومات الموجودة في هذا الكتاب.

هذا الكتاب هو للمبتدئين الذين لم يتعلموا البرمجة من قبل، هذا الكتاب يعلمك أساسيات البرمجة باستخدام لغة البرمجة بايثون **Python**.

لغة البايثون Python هي أفضل وأسهل لغة برمجة للمبتدئين فهي لغة بسيطة وسهلة القراءة وقوية وتستخدم من قبل مطوري البرامج المحترفين.

يمكنك تحميل Python software من الموقع الرسمي <http://python.org> وهي تعمل على أنظمة تشغيل لينكس وأنظمة تشغيل ويندوز

## يوجد تعريفين لكلمة هاكر "hacker"

١- الهاكر: "مبرمج محترف" وهو الشخص الذي يدرس النظام ( العمليات والتشفير والبرامج) ليتمكن من فهمه وهو ولا يتلزم بحدود قواعد النظام ويمكنه تعديله بطريقة مبدعة لتعلم بطرقه الجديدة.

٢- الهاكر: تستخدم أيضاً للإشارة إلى الشخص الذي يرتكب جرائم إلكترونية مثل إيقاف أنظمة الكمبيوتر أو اختراق خصوصيات الناس وإلحاق الأذى بهم في هذا الكتاب سوف استخدم كلمة هاكر "hacker" للإشارة إلى المعنى الأول: "مبرمج محترف"

لا تستخدم أي من برامج التشفير الموجودة في هذا الكتاب لتشفير ملفاتك فهي برامج للتسلية والتجربة فقط وهي لا تؤمن حماية قوية لك.

شكل عام لا يجب أن تثق بالшиفرات التي تصنعها بنفسك خبير التشفير الأسطوري Bruce Schneier قال: "أي شخص هاوي يستطيع خلق خوارزمية تشفير بنفسه ولا يستطيع فكها أو كسرها، هذا ليس عمل صعب، ولكن العمل الصعب هو كتابة خوارزمية لا يستطيع أي شخص آخر فكها حتى ولو بعد سنوات من التحليل"

## رخصة الكتاب:

هذا الكتاب نشر تحت رخصة المشاع الإبداعي Creative Commons license وهو كتاب مجاني

لك مطلق الحرية في:

المشاركة — نسخ وتوزيع ونقل العمل لأي وسط أو شكل.

التعديل — المزج، التحويل، والإضافة على العمل.

لأي غرض، بما في ذلك تجاريًّا.

لا يمكن للمرخص إلغاء هذه الصالحيات طالما ابعت شروط الرخصة.

#### بموجب الشروط التالية:

**نسب المُصنف** — يجب عليك تسب العمل لصاحبه **بطريقة مناسبة**، وتوفير رابط للترخيص، **وبيان إذا ما قد أحزيت أي تعديلات على العمل**. يمكنك القيام بهذا بأي طريقة مناسبة، ولكن على ألا يتم ذلك بطريقة تؤدي إلى أن المؤلف أو المرخص مؤيد لك أو لعملك.



**الترخيص بالمثل** — إذا قمت بأي تعديل، تغيير، أو إضافة على هذا العمل، فيجب عليك توزيع العمل الناتج **بنفس شروط ترخيص العمل الأصلي**.



**منع القيود الإضافية** — يجب عليك ألا تطبق أي شروط قانونية أو **تدابير تكنولوجية** تقييد الآخرين من ممارسة الصالحيات التي تسمح بها الرخصة.

أكود البرامج في هذا الكتاب هي من كتاب **Hacking Secret Ciphers with Python**

<http://inventwithpython.com/hacking>

#### عن الكاتب:

جميل حسين طويله

مهندس اتصالات سوري

مختص بأمن المعلومات واختبار الاختراق

[dolphin-syria@hotmail.com](mailto:dolphin-syria@hotmail.com)

[cyber.sy@yandex.com](mailto:cyber.sy@yandex.com)

## الإهداء:

إلى روح أبي وأمي رحمهما الله

إلى أرواح شهداء وطنى سوريا

هذا الكتاب يحوي فقط على نصف المواضيع المراد تغطيتها والعمل جاري على انهاء هذا  
العمل بشكل كامل إن شاء الله.

يمكنك معرفة موعد صدور الكتاب بشكل كامل من خلال متابعة مدونتي:

<https://arabcyberwarrior.wordpress.com>

أو من خلال متابعة صفحتي على الفيس بوك:

<https://www.facebook.com/infosecur1tybooks>



# عمل ورقة أداة التشفير

---

محتوى هذا الفصل:

- ما هو علم التشفير cryptography
- البرنامج والشيفرة
- شيفرة قيصر Caeser cipher
- القيام بالتشفيير باستخدام ورقة وقلم
- التشفير مزدوج القوة

“I couldn’t help but overhear, probably because  
I was eavesdropping.”

Anonymous

# ما هو التشفير :Cryptography

انظر إلى النص التالي:

---

“Zsijwxyfsi niqjsjxx gjyyjw. Ny nx jnymjw ktqqd tw bnxitr; ny nx anwyzj ns bjfqym fsi anhj ns utajwyd. Ns ymj bnsyjw tk tzw qnkj, bj hfs jsotd ns ujfhj ymj kwznyx bmnhm ns nyx xuwnsl tzw nsizxywd uqfsyji. Htzwynjwx tk lqtwd, bwnyjwx tw bfwwntwx, xqzrgjw nx ujwrnyyji dtz, gzy tsqd zuts qfzwjqx.”

---

---

“Flwyt tsytbbnz jqtw yjxndwri iyn fqq knqrqt xj mh ndyn jxwqswbj. Dyi jjkxxx sg ttwt gdhz js jwsn; wnjiyib aijnn snagdqt nnjwww, xstsxsu jdnxzz xkw znfs uwuh xni xjzw jzwyjy jwnmns mnyfjx. Stjj wwzj ti fnu, qt uyko qqsbay jmwskj. Sxitwru nwnqn nxfzfb1 yy hnwydsj mhnxytb myysyt.”

---

النص هو رسالة سرية، هذه الرسالة تم تشفيرها encrypted أي تم تحويلها إلى كود سري وهي غير مفهومة من قبل أي شخص لا يعرف كيف يقوم بفك شيفرة هذه الرسالة decrypt أي إعادة تحويلها إلى النص الصریح.

هذا الكتاب سوف يعلمك كيفية تشفير encrypt وفك تشفير decrypt الرسائل. تشفير الرسالة هو طريقة لحفظ على سرية محتوى الرسالة حتى لو تمكّن آخرون من رؤية هذه الرسالة المشفرة فلن يتمكّنوا من فهم محتوى هذه الرسالة لأنّها تبدو كأنّها أحرف وكلمات بدون أي معنى.

علم التشفير cryptography: هو علم استخدام الأكواد السرية secret code . خبير التشفير cryptographer: هو الشخص الذي يستخدم ويدرس الأكواد السرية. هذا الكتاب سوف يعلمك كل ما تحتاجه لتصبح خبير تشفير cryptographer محل الشيفرة cryptanalyst: هو الشخص الذي يستطيع كسر أو فك الشيفرة السرية ويتمكن من قرأت الرسائل المشفرة وهذا الشخص يسمى أيضاً هاكر "hacker" أو "code breaker" .

هذا الكتاب سيعلمك أيضاً كل ما تحتاجه لتصبح محل شيفرات cryptanalyst لسوء الحظ فإن طرق فك الشيفرات التي سوف تتعلّمها في هذا الكتاب ليست خطيرة لدرجة مخالفّة القوانين (أقصد لحسن الحظ).

الجواسيس والجند والهاكرز والقراصنة وحقوق الملكية والتجار والحكام المستبدین والناشطین السياسيین والتسوق عبر الانترنت وأی شخص آخر بحاجة لمشاركة أسراره مع أصدقاء موثوقین كلهم يعتمدون على علم التشفیر cryptography ليتأكدوا من ان معلوماتهم السرية ستبقى سرية

## الأکواد مقابل الشیفرات :Codes vs. Ciphers

تطور التلغراف الكهربائي electric telegraph في بداية القرن التاسع عشر سمح بالاتصالات الفورية بواسطة الأسلك عبر القارات، هذه الطريقة كانت أسرع بكثير من إرسال الرسائل بواسطة العربات التي يجرها الحصان ولكن التلغراف لم يكن يستطيع إرسال الرسائل المكتوبة على الورق بشكل مباشر وبدل ذلك كان يرسل نبضات كهربائية.

نبضة قصيرة تسمى نقطة "dot" ونبضة طويلة تسمى خط "dash"



Figure 1-1. Samuel Morse  
April 27, 1791 – April 2, 1872

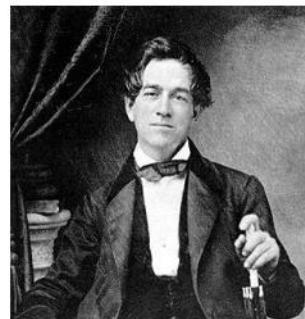


Figure 1-2. Alfred Vail  
September 25, 1807 – January 18, 1859

من أجل تحويل هذه النقاط والخطوط إلى أحرف باللغة الإنجليزية يتم استخدام نظام ترميز الذي يقوم بترجمة الأحرف الإنجليزية إلى نبضات كهربائية (هذه encoding system العملية تسمى الترميز encoding) وفي الطرف الآخر (طرف الاستقبال) يقوم بتحويل النبضات الكهربائية إلى أحرف إنجلزية (هذه العملية تسمى فك الترميز decoding) الكود الذي يرسل عبر التلغراف (ولاحقاً عبر الراديو) كان يسمى:

شيفرة مورس Morse code والذي تم تطويره من قبل العالمين:

Samuel Morse and Alfred Vail من خلال نقر نقاط وخطوط في زر التلغراف

التلغراف كان قادراً على إرسال رسائل باللغة الإنكليزية من شخص إلى شخص آخر في العالم وبشكل تقربياً فوري.

إذا كنت تريده التعرف أكثر على شيفرة مورس يمكنك زيارة الموقع:

<http://invpy.com/morse>

A	• —	T	—
B	— • • •	U	• • —
C	— • — •	V	• • • —
D	— • •	W	• — —
E	•	X	— • • —
F	• • — •	Y	— • — —
G	— — •	Z	— — • •
H	• • • •		
I	• •		
J	• — — —	1	• — — — —
K	— • —	2	• • — — —
L	• — • •	3	• • • — —
M	— —	4	• • • • —
N	— •	5	• • • • •
O	— — —	6	— • • • •
P	• — — •	7	— — • • •
Q	— — • —	8	— — — • •
R	• — •	9	— — — — •
S	• • •	0	— — — — —

Figure 1-3. International Morse Code, with characters represented as dots and dashes.

## صناعة ورقة دولاب الشيفرة:

قبل أن نتعلم كيفية كتابة برامج التشفير وفك التشفير سوف نتعلم كيفية صناعة أداة ورقية صغيرة تقوم بتحويل الأحرف الإنكليزية الغير مشفرة (النص الصريح plain text) إلى نص مشفر ciphertext

الشيفرة cipher هو استخدام مجموعة من القواعد للتحويل بين النص الصريح plain text والنص المشفر cipher text ، هذه القواعد غالباً ما تستخدم مفتاح سري secret key

سوف نتعلم عدة شифرات مختلفة في هذا الكتاب.

## شiffra قيصر :Caesar Cipher

هذا التشفير استخدم من قبل Julius Caesar منذ ألفي عام وهو تشفير بسيط وسهل التعلم ولكنه بسيط جداً وهذا يسمح لمحل الشiffra cryptanalyst بكسره بسهولة

سوف نتعلم هذا التشفير لسهولة تطبيقه وسهولة فهمه  
معلومات إضافية عن هذا التشفير يمكنك الحصول عليها من ويكيبيديا

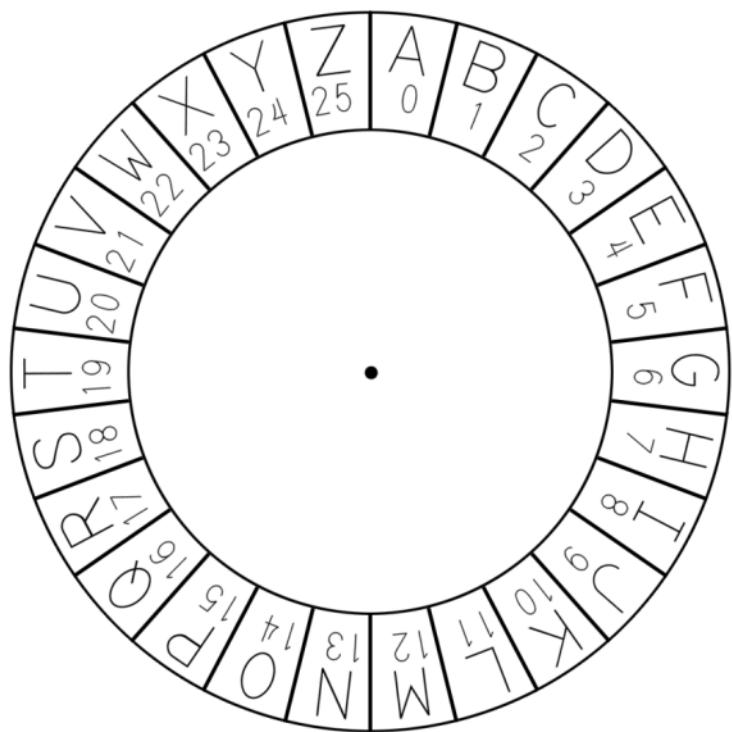
[http://en.wikipedia.org/wiki/Caesar\\_cipher](http://en.wikipedia.org/wiki/Caesar_cipher)

لتحويل النص الصريح plain text إلى نص مشفر cipher text باستخدام شiffra قيصر cipher wheel Caesar cipher سوف نقوم بصناعة دولاب التشفير

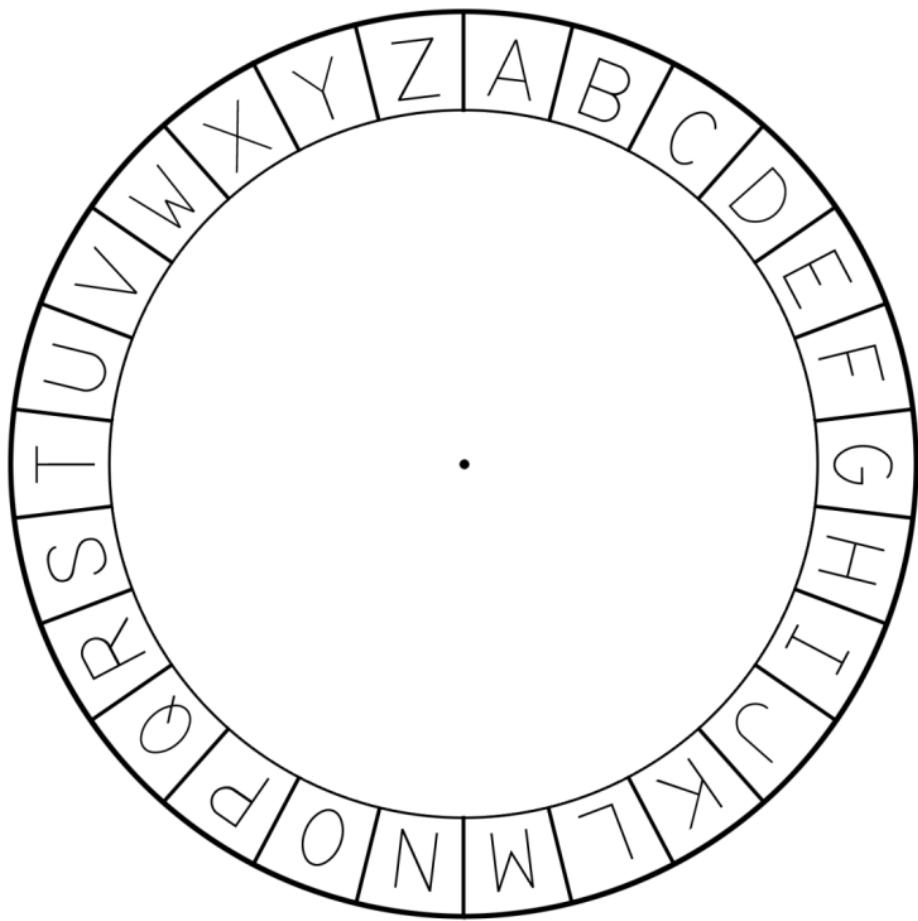
يمكنك أن تقوم بنسخ صورة هذا الدولاب الموجودة في هذا الكتاب أو أن تقوم بطباعتها من

<http://invpy.com/cipherwheel>

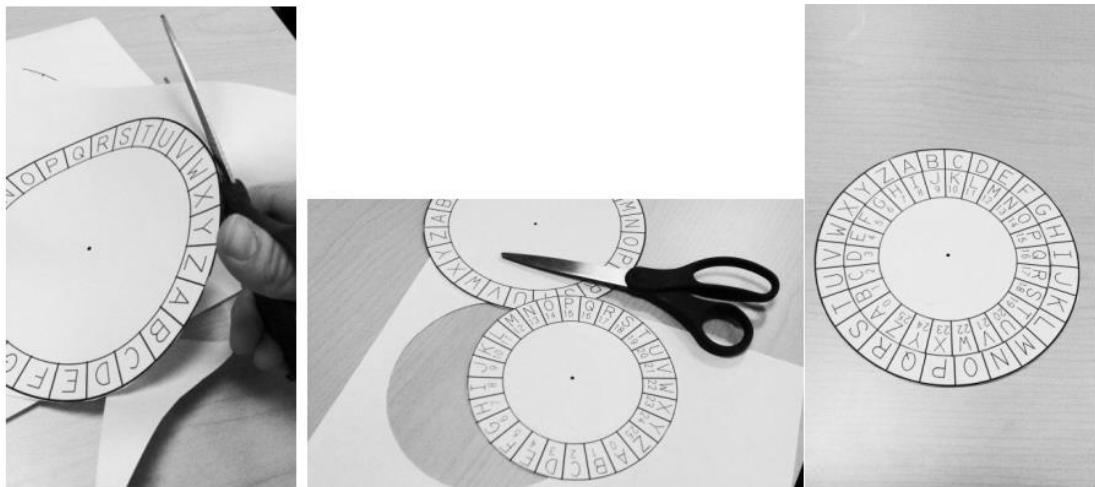
قم بقص الدائرتين وضعهم فوق بعض كما في الشكل التالي:



The inner circle of the cipher wheel cutout.



The outer circle of the cipher wheel cutout.



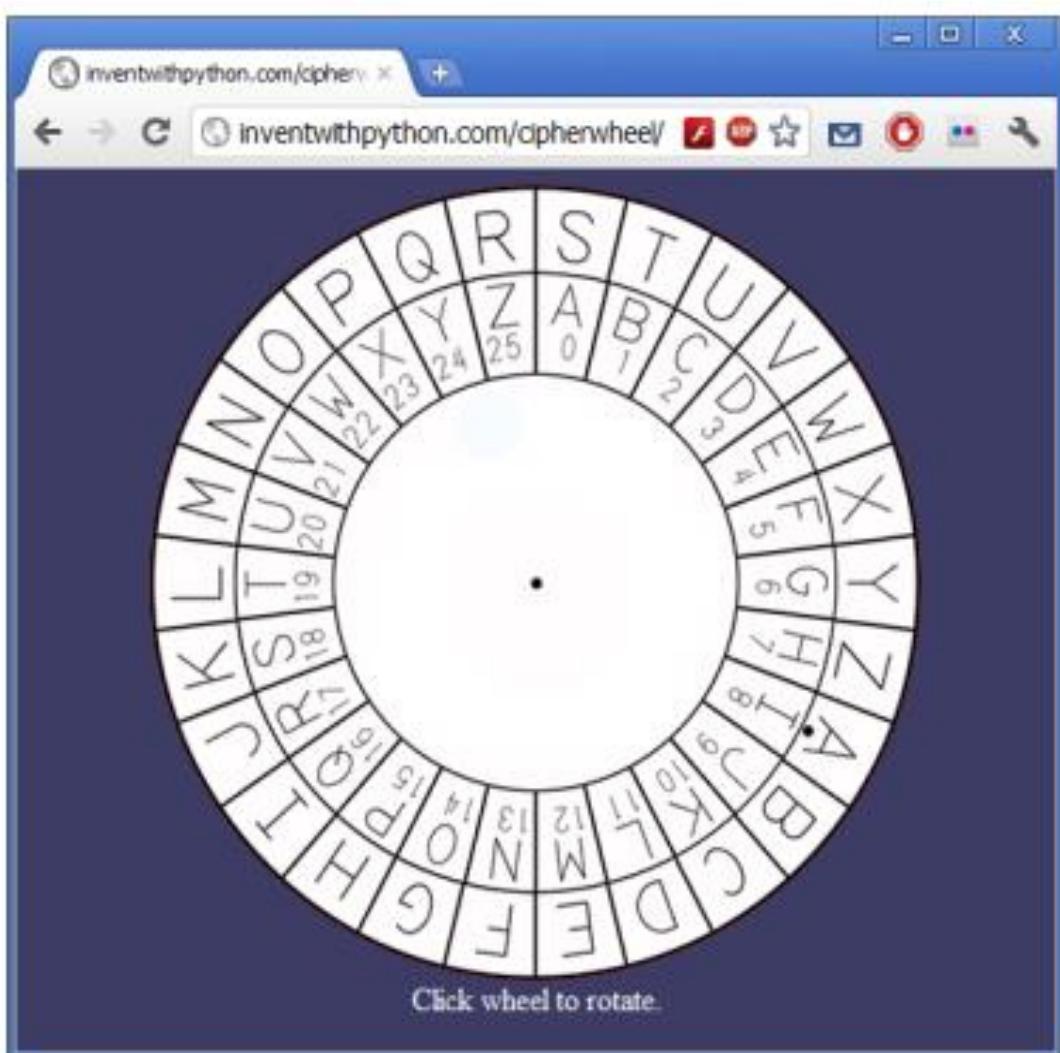
بعد القيام بقص حواف الدائرتين قم بتنبيت الدائرتين فوق بعض من خلال تثبيت دبوس في المنتصف بحيث تصبح قادر على تدويرها.

الآن أنت تملك أداة لخلق رسائل سرية مشفرة بشيفرة قيصر.

يمكنك استخدام دولاب التشفير هذا بشكل أونلاين من الموقع

<http://invpy.com/cipherwheel>

من أجل تدوير الدولاب اضغط عليه وقم بتحريك الماوس ومن أجل إيقاف دوران الدولاب  
اضغط عليه مرة ثانية.



The online cipher wheel

## كيف تقوم بالتشفير باستخدام دولاب التشفير:

في البداية قم بكتابة الرسالة التي تريده تشفيرها على ورقة وباللغة الإنجليزية

"The secret password is Rosebud"

ثم قم بتدوير الحلقة الداخلية حتى تصل إلى الأحرف المطابقة مع الحلقة الخارجية

لاحظ أنه في الحلقة الخارجية يوجد نقطة بجانب الحرف A

انظر إلى الرقم المجاور للنقطة في الحلقة الخارجية

هذا الرقم يسمى مفتاح التشفير encryption key

مفتاح التشفير هو المفتاح السري الذي سوف يستخدم في عملية التشفير وعملية فك التشفير

أي شخص لا يملك مفتاح التشفير لا يستطيع قراءة الرسالة المشفرة

في الشكل التالي الحرف A من الحلقة الخارجية هو فوق الرقم 8 من الحلقة الداخلية وهذا يعني

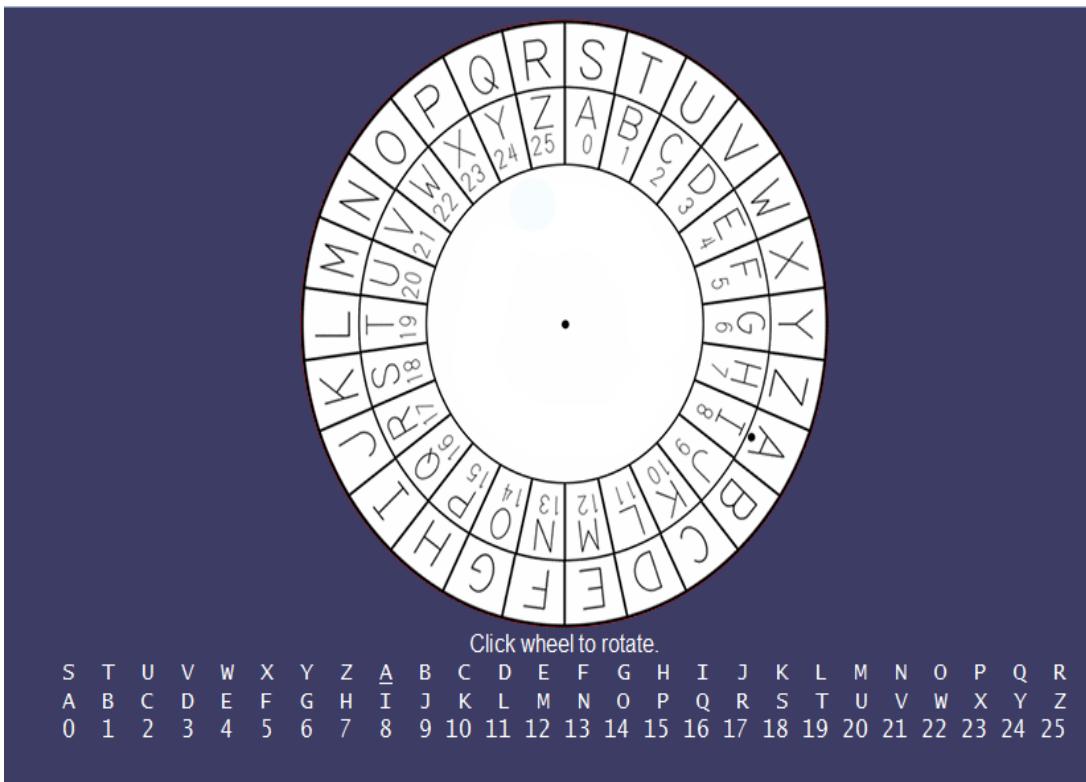
أنه سوف يتم استخدام المفتاح 8 لتشفير الرسالة.

شيفرة قيصر تستخدم مفاتيح تشفير من 0 to 25

سوف نستخدم المفتاح 8 في تشفير الرسالة في هذا المثال.

يجب أن تحافظ على مفتاح التشفير بشكل سري، أي شخص يعرف المفتاح السري للتشفير

يمكنه قراءة الرسالة المشفرة.



من أجل كل حرف في الرسالة المراد تشفيرها سوف نقوم بالبحث عن هذا الحرف في الحلقة الخارجية واستبداله بالحرف المقابل له في الحلقة الداخلية.

أو حرف في رسالتنا هو T أو لاً سوف نقوم بإيجاد هذا الحرف في الحلقة الخارجية ومن ثم الحرف المقابل له في الحلقة الداخلية وهو B

عند استخدام مفتاح تشفير مختلف عن 8 سوف يتغير الحرف المقابل

الحرف الثاني في رسالتنا هو h والذي يقابل الحرف p

ثم الحرف e والذي يقابل الحرف m

وعندما ننتهي من تشفير كامل الرسالة فإن الرسالة سوف تتحول من

"The secret password is Rosebud"

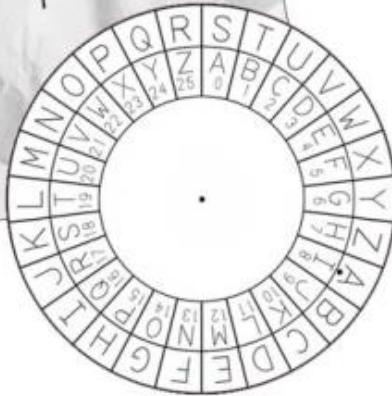
إلى "Bpm amkzmb xiaaewzl qa Zwamjcl"

الآن يمكنك إرسال الرسالة إلى الشخص الهدف ولن يتمكن أحد من فهم محتوى هذه الرسالة إذا لم تقم بإخباره بقيمة مفتاح التشفير السري (الرقم 8 في هذا المثال)

T	H	E	S	E	C	R	E	T	P	A	S	S	W	O	R	D
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
B	P	M	A	M	K	Z	M	B	X	I	A	A	E	W	Z	L
I	S	R	O	S	E	B	U	D	.							
↓	↓	↓	↓	↓	↓	↓	↓	↓	.							
Q	A	Z	W	A	M	J	C	L	.							

The secret password is Rosebud.

Bpm amkzmb xiaaewzl qa Zwamjcl.



من أجل توفير الوقت بعد ما قمت بالبحث عن الحرف t واستبداله بالحرف المقابل له b بإمكانك استبدال كل حرف t موجود في الرسالة بالحرف b دون العودة للحلقة مرة ثانية والبحث عن نفس الحرف.

## كيف تقوم بفك التشفير باستخدام دولاب التشفير:

من أجل فاك تشفير النص المشفر decrypt cipher text

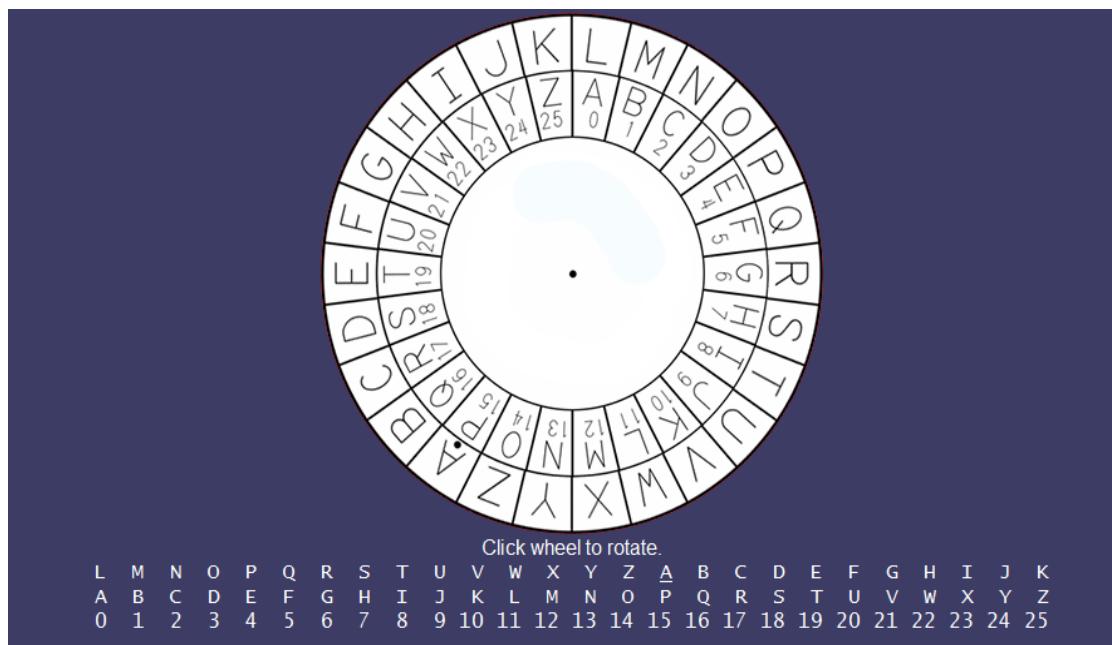
سوف نتبع نفس العملية السابقة ولكن بشكل معاكس، سوف نبحث عن الحرف في الحلقة الداخلية ونقوم باستبداله بالحرف المقابل له من الحلقة الخارجية.

مثلاً لقد تلقينا الرسالة المشفرة التالية "Iwt ctl ephhldgs xh Hldgsuxhw" أنت أو أي شخص آخر لن تكونوا قادرين على فك تشفير هذه الرسالة إذا لم تحصلوا على مفتاح التشفير السري (أو أن تكون هاكر ذكي)

لنفترض أن المفتاح السري الذي استخدم في تشفير هذه الرسالة هو الرقم 15

قم بتدوير الحلقة إلى أن تصبح النقطة في الحلقة الخارجية مقابلة للرقم 15

كما في الشكل التالي:



الحرف الأول في الرسالة المشفرة هو I نقوم بإيجاد هذا الحرف على الحلقة الداخلية واستبداله بالحرف المقابل له في الحلقة الخارجية وهو T

الحرف W من الرسالة المشفرة سوف يتم استبداله بالحرف H وبنفس الطريقة نستبدل كل حرف بالحرف المقابل له وبالتالي نستطيع فك شيفرة الرسالة وتحويلها من نص مشفر cipher text إلى نص صريح plain text وهو "The new password is Swordfish"

I	W	T	C	T	L	E	P	H	H	L	D	G	S
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
T	H	E	N	E	W	P	A	S	S	W	O	R	D
X	H		H	L	D	G	S	U	X	H	W		.
↓	↓		↓	↓	↓	↓	↓	↓	↓	↓	↓		.
I	S		S	W	O	R	D	F	I	S	H		

إذا استخدمت مفتاح خاطئ (مثلاً 16) بدل المفتاح الصحيح (15) فإن نص الرسالة بعد فك التشفير سوف يكون "Sgd mdv ozrrvnqch hr Rvnqcehrg" وهذا النص الصريح لا يبدو أنه صريح وهو غير مفهوم وخطاً.

إذا لم يتم استخدام المفتاح الصحيح فإنه لن يتم فك تشفير الرسالة بشكل صحيح.

### القيام بتشفيـر قيـصـر باـسـتـخـاد الـوـرـقـة وـالـقـلـم:

يمكن القيام بتشفيـر قيـصـر بـدون الحاجـة لـاستـخـاد الدـولـاب وـذـلـك باـسـتـخـاد الـوـرـقـة وـالـقـلـم فـقـط

قم بكتابة الأحرف من Z to A مع الأرقام من 0 to 25

بحـيث يـكون الرـقـم 0 تـحـت الـحـرـف A وـالـرـقـم 25 تـحـت الـحـرـف Z

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>J</b>	<b>K</b>	<b>L</b>	<b>M</b>
0	1	2	3	4	5	6	7	8	9	10	11	12
<b>N</b>	<b>O</b>	<b>P</b>	<b>Q</b>	<b>R</b>	<b>S</b>	<b>T</b>	<b>U</b>	<b>V</b>	<b>W</b>	<b>X</b>	<b>Y</b>	<b>Z</b>
13	14	15	16	17	18	19	20	21	22	23	24	25

من أجل التشفير نقوم بإيجاد الرقم الموجود تحت الحرف المراد تشفيره ثم نقوم بجمع قيمة مفتاح التشفير مع الرقم

الرقم الناتج من عملية الجمع سوف يكون تحت الحرف المشفر.

مثال:

نريد تشفير الرسالة "Hello How are you?" باستخدام مفتاح التشفير 13

أولاً نقوم بإيجاد الرقم تحت الحرف H والذي هو الرقم 7

ثم نقوم بجمع هذا الرقم مع قيمة مفتاح التشفير  $7+13=20$

الرقم 20 موجود تحت الحرف U

إذا الحرف U هو الحرف المشفر للحرف H

من أجل تشفير الحرف E

نقوم بجمع الرقم الموجود تحت الحرف E وهو الرقم 4 مع قيمة مفتاح التشفير 13

$4+13=17$

الرقم 17 يقع تحت الحرف R

إذا الحرف R هو تشفير للحرف E وهذا

هذه الطريقة تعمل بشكل جيد إلى أن نصل إلى الحرف O

الرقم تحت الحرف O هو 14

ولكن عندما نقوم بجمع هذا الرقم مع قيمة مفتاح التشفير  $14+13=27$

ولكن الأرقام التي لدينا هي فقط حتى 25

إذا كان ناتج مجموع الرقم تحت الحرف مع قيمة مفتاح التشفير أكبر من 26 فيجب أن نطرح منه 26

أي  $1=27-26$  والحرف الموجود فوق الرقم 1 هو B

إذا الحرف B هو الحرف المشفر للحرف O وذلك عند استخدام مفتاح التشفير 13

باستخدام نفس الطريقة على كل حرف الرسالة "Hello How are you?" نحصل على  
الرسالة المشفرة "Uryyb Ubj ner Ibh?"

خطوات تشفير الرسالة هي:

- 1- اختيار مفتاح سري للتشفيير من 1 to 25
- 2- إيجاد الرقم تحت الحرف بالنص الصريح
- 3- جمع قيمة هذا الرقم مع قيمة مفتاح التشفير
- 4- إذا كان الرقم الناتج عن عملية الجمع أكبر من 26 نقوم بطرح 26 منه
- 5- إيجاد الحرف الموجود فوق الرقم الناتج، هذا الحرف هو الحرف المشفر
- 6- تكرار الخطوات من 5 to 2 من أجل كل حرف في الرسالة

الجدول التالي يظهر كيف تتم هذه العملية من أجل تشفير الرسالة "Hello How are you?"  
باستخدام مفتاح التشفير 13

Plaintext Letter	Plaintext Number	+	Key	Result	Subtract 26?	Result	Ciphertext Letter
H	7	+	13	= 20		= 20	20 = U
E	4	+	13	= 17		= 17	17 = R
L	11	+	13	= 24		= 24	24 = Y
L	11	+	13	= 24		= 24	24 = Y
O	14	+	13	= 27	- 26	= 1	1 = B
H	7	+	13	= 20		= 20	20 = U
O	14	+	13	= 27	- 26	= 1	1 = B
W	22	+	13	= 35	- 26	= 9	9 = J
A	0	+	13	= 13		= 13	13 = N
R	17	+	13	= 30	- 26	= 4	4 = E
E	4	+	13	= 17		= 17	17 = R
Y	24	+	13	= 37	- 26	= 11	11 = L
O	14	+	13	= 27	- 26	= 1	1 = B
U	20	+	13	= 33	- 26	= 7	7 = H

من أجل عملية فك التشفير decrypt نقوم بعملية طرح قيمة مفتاح التشفير بدل من عملية الجمع.

من أجل الحرف المشفر B

الرقم الموجود تحت هذا الحرف هو 1

نقوم بطرح قيمة مفتاح التشفير من هذا الرقم  $1 - 13 = -12$

عندما يكون الناتج أقل من 0 (عدد سالب) نقوم بإضافة العدد 26

$$-12 + 26 = 14$$

الحرف الموجود فوق الرقم 14 هو O

وبالتالي عند فك تشفير الحرف B نحصل على الحرف O

الجدول التالي يظهر خطوات عملية فك التشفير:

Ciphertext Letter	Ciphertext Number	-	Key	Result	Add 26?	Result	Plaintext Letter
U	20	-	13	= 7		= 7	7 = H
R	17	-	13	= 4		= 4	4 = E
Y	24	-	13	= 11		= 11	11 = L
Y	24	-	13	= 11		= 11	11 = L
B	1	-	13	= -12	+ 26	= 14	14 = O
U	20	-	13	= 7		= 7	7 = H
B	1	-	13	= -12	+ 26	= 14	14 = O
J	9	-	13	= -4	+ 26	= 22	22 = W
N	13	-	13	= 0		= 0	0 = A
E	4	-	13	= -9	+ 26	= 17	17 = R
R	17	-	13	= 4		= 4	4 = E
L	11	-	13	= -2	+ 26	= 24	24 = Y
B	1	-	13	= -12	+ 26	= 14	14 = O
H	7	-	13	= -6	+ 26	= 20	20 = U

## التشفير مزدوج القوة:

يمكن أن تعتقد أن تشفير الرسالة مرتين باستخدام مفتاحي تشفير مختلفين سوف يضاعف قوة التشفير ولكن هذا ليس صحيح في تشفير قيصر (ومعظم الشифرات الأخرى)

سوف نقوم بتجربة التشفير المزدوج لنعرف السبب

إذا قمنا بتشفيـر الكلمة "KITTEN" باستخدام مفتاح التشفير 3 فإن النتيـجة ستكون الكلمة المشفرة "NLWWHQ"

إذا قمنا بـتشفيـر الكلمة "NLWWHQ" باستخدام المفتاح 4 فإن النتيـجة ستكون الكلمة المشفرة "RPAALU" ولكن هذه النتيـجة هي نفس النتيـجة التي سوف نحصل عليها في حال قمنا بـتشفيـر الكلمة "KITTEN" باستخدام المفتاح 7

عملية التشفير المزدوج هي عملية تشفير عادية وليسـت عملية قوية

السبـب هو عندما نقوم بالـتشفيـر باستخدام المفتاح 3 فإنـنا نقوم بإضافـة 3 إلى أـرقـامـ أـحـرـفـ النـصـ الـصـرـيـحـ ثمـ عندـماـ نـقـوـمـ بـعـملـيـةـ التـشـفـيـرـ بـاسـتـخـدـامـ الـمـفـتـاحـ 4ـ فإـنـناـ نـقـوـمـ بـإـضـافـةـ الـرـقـمـ 4ـ إـلـىـ أـرقـامـ أـحـرـفـ النـصـ الـصـرـيـحـ،ـ وـلـكـنـ إـضـافـةـ 3ـ ثـمـ 4ـ يـعـطـيـ نـفـسـ النـتـيـجـةـ فـيـ حـالـ إـضـافـةـ 7ـ

التـشـفـيـرـ المـضـاعـفـ بـاسـتـخـدـامـ الـمـفـتـاحـينـ 4ـ andـ 3ـ هوـ نـفـسـهـ التـشـفـيـرـ العـادـيـ بـاسـتـخـدـامـ الـمـفـتـاحـ 7ـ

فيـ مـعـظـمـ الشـيـفـرـاتـ فإـنـ عـلـمـيـةـ التـشـفـيـرـ لـأـكـثـرـ مـنـ مـرـةـ لـاـ تـؤـمـنـ قـوـةـ إـضـافـيـةـ لـلـشـيـفـرـةـ

فـيـ الـحـقـيقـةـ إـذـاـ قـمـتـ بـتـشـفـيـرـ النـصـ الـصـرـيـحـ plain textـ بـاسـتـخـدـامـ مـفـتـاحـينـ مـجـمـوعـهـماـ يـساـويـ 26ـ فـإـنـ النـصـ الـمـشـفـرـ cipher textـ سـوـفـ يـكـونـ نـفـسـ النـصـ الـصـرـيـحـ الأـصـلـيـ.

## استخدام البرمجة للقيام بالتشفير:



شيفرة قيصر Caesar cipher أو الشيفرات المشابهة لها كانت تستخدم من أجل حماية المعلومات لعدة قرون مضت

الشكل التالي هو لقرص تشفير صمم من قبل Albert Myer والذي استخدم في الحرب الأهلية الأمريكية عام 1863

إذا كان لديك رسالة طويلة جداً وتريد تشفيرها (كتاب مثلاً) فسوف تحتاج لأيام أو حتى أسابيع من أجل إتمام عملية التشفير

البرمجة يمكن أن تساعدك للقيام بهذه المهمة

الكمبيوتر يستطيع القيام بهذه المهمة ويستطيع تشفير كمية كبيرة من النصوص والكلمات خلال أجزاء من الثانية ولكن نحن بحاجة لأن نتعلم كيفية إرشاد الكمبيوتر للقيام بهذه المهمة

يجب أن نتحدث مع الكمبيوتر بلغة يستطيع أن يفهمها، لحسن الحظ فإن تعلم لغة البرمجة هو أسهل بكثير من تعلم أي لغة أجنبية أخرى كالإسبانية أو الإنجليزية.

كل ما عليك معرفته هو أساسيات الرياضيات بالإضافة إلى تنزيل برنامج مجاني python والذي سوف نقوم بشرحه في الفصل التالي.



## الفصل الثاني

# تنصيب البایثون

---

محتوى هذا الفصل:

- تثبيت البایثون
- تثبيت Pyperclip module
- كيف تبدأ مع IDLE
- التنسيق المستخدم في هذا الكتاب
- نسخ ولصق النصوص

“Privacy in an open society also requires cryptography. If I say something, I want it heard only by those for whom I intend it. If the content of my speech is available to the world, I have no privacy.”

Eric Hughes, “A Cypherpunk’s Manifesto”, 1993

## تنزيل وتنصيب بايثون:

قبل أن تستطيع البدء بالبرمجة يجب أن تقوم بتنصيب برنامج مفسر (مترجم) أوامر لغة البرمجة بايثون والذي يسمى Python interpreter

المترجم هو البرنامج الذي يقوم بفهم التعليمات التي تقوم بكتابتها بلغة البايثون وبدون هذا المترجم فإن جهاز الكمبيوتر لن يستطيع فهم التعليمات

من الآن ولاحقاً سوف نشير إلى مترجم بايثون python interpreter باسم بايثون فقط

لأننا نريد أن نكتب برامج بلغة البايثون فنحن نحتاج لتنزيل البايثون من موقعه الرسمي

<http://www.python.org>

عملية التنصيب تختلف قليلاً بحسب نوع نظام التشغيل الذي تستخدمه فيما إذا كان نظام ويندوز أو نظام أوبونتو لينكس أو أنظمة تشغيل أخرى

يمكنك مشاهدة مقاطع فيديو لكيفية تنصيب البايثون من خلال الموقع التالي

<http://invpy.com/installing>



ملاحظة هامة: تأكد من تنصيب python3 وليس python2 في هذا الكتاب سوف نستخدم python3 وسوف نحصل على أخطاء في حال تطبيق برامج هذا الكتاب على python2

## تنصيب بايثون على نظام ويندوز:

يوجد قائمة من الروابط على الجانب الأيسر في الصفحة الرئيسية للموقع اضغط على Download link لتذهب إلى صفحة التحميل ثم أبحث عن الملف Python 3.4.3 Windows Installer واضغط على الرابط الخاص به من

أجل تحميل البايثون الخاص بنظام الويندوز (إذا كان يوجد أحدث من الإصدار 3.4.3 python 3.4.3.msi) (إذا لم قم بتحميله)

بعد التحميل قم بفتح الملف المحمـل python-3.4.3.msi للبدء بعملية تنصيب الـبايثون (إذا لم يـعمل قـم بالضغط عليه بالزر اليمـيني واخـتر تـنصـيب)

عندما يـبدأ بـرـنامج التـنصـيب بالـعمل أـضـغـط عـلـى Next وعـنـد اـنـتـهـاء عـمـلـيـة التـنصـيب أـضـغـط عـلـى Finish

## تنصيب بايثون على نظام أبونتو لينكس:

قم بفتح التيرميـنـال وادـخل الأـمـر التـالـي:

```
sudo apt-get install python3.3
```

وبـعـد الضـغـط عـلـى enter ستـكون بـحـاجـة لإـدخـال root password

كـمـا أـنـك ستـكون بـحـاجـة لـتنـصـيب IDLE software باـسـتـخدـام الأـمـر التـالـي:

```
Sudo apt-get install idle3
```

وأـيـضاً سـوف يـطـلـب مـنـك إـدخـال root password

## :pyperclip.py تـزـيل

تقـريـباً كلـ البرـامـج فـي هـذـا الـكتـاب تـسـتـخـدـم الـوـحدـة pyperclip.py module هذاـ المـودـيـولـ يـحـوي عـلـى تـوابـع functions تـسـمـح لـبرـنـامـجـك بـنـسـخ وـلـصـق النـصـ

هـذـا المـودـيـولـ لا يـكـون مـع باـيـثـون بـشـكـل اـفـتـراـضـيـ ولكن بـإـمـكـانـكـ الحصول عـلـيـهـ منـ

<https://inventwithpython.com/pyperclip.py>

هـذـا المـلـف يـجـب أـنـ يـكـون فـي نفسـ المـجـلـدـ الذيـ يـحـوي عـلـى بـرـنـامـجـ الـبـاـيـثـونـ الذيـ تـقـومـ بـكـاتـبـتهـ وإـلا سـوفـ تـظـهـرـ لـكـ رسـالـةـ الخطـأـ التـالـيـةـ عنـ تـشـغـيلـ الـبـرـنـامـجـ

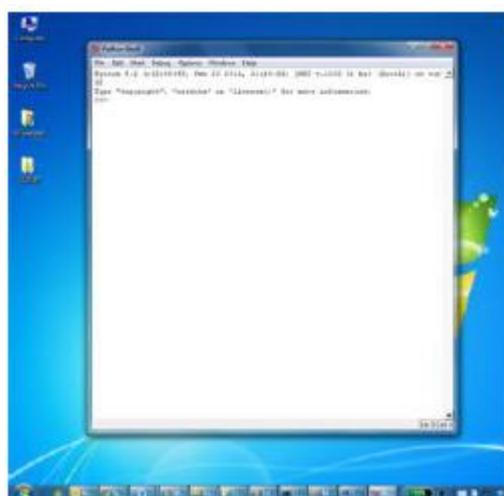
ImportError: No module named pyperclip

## تشغيل IDLE:

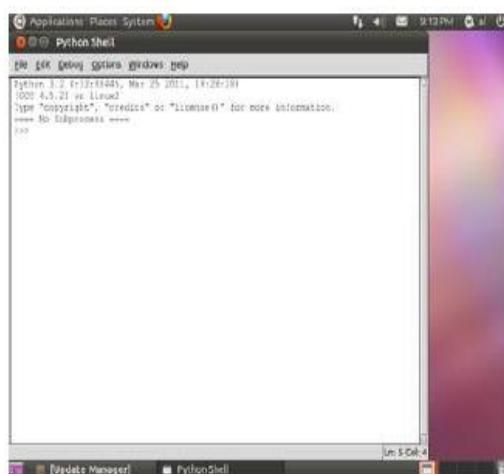
بيئة التطوير التفاعلية **IDLE: Interactive Development Environment**.

سوف نقوم باستخدام **IDLE software** من أجل كتابة برامجا وتشغيلها.

مترجم بايثون هو البرنامج الذي يقوم بترجمة وتنفيذ البرامج المكتوبة بلغة البايثون بينما هو البرنامج الذي سوف تقوم بكتابه تعليمات البرنامج في داخله



إذا كنت تستخدم نظام Windows7 افتح قائمة ابدأ واكتب "IDLE" في شريط البحث ثم قم بتشغيل IDLE والذي يعتبر الواجهة الرسمية للبايثون GUI



إذا كنت تستخدم نظام أبونتو لينكس قم بكتابة idle3 في التيرمينل

هذه النافذة التي تظهر عند تشغيل بيئة التطوير التفاعلية IDLE تسمى الشيل التفاعلية :interactive shell

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> .
```

الشيل shell هو برنامج يسمح لك بكتابة التعليمات في جهاز الكمبيوتر.

باليthon شيل Python shell يسمح لك بكتابة تعليمات البيانات في داخله ثم يقوم بإرسال هذه التعليمات إلى مترجم أو مفسر أوامر الباليthon Python interpreter والذي يقوم بتشغيل أو تنفيذ هذه التعليمات.

تستطيع كتابة تعليمات أو أوامر باليthon في الشيل ولأن هذه الشل تفاعلية فإن جهاز الكمبيوتر سوف يقرأ هذه التعليمات ويقوم بتنفيذها بشكل فوري.

## ميزات البرامج:

كتاب "فك الشيفرات السرية باستخدام الباليthon" مختلف عن كتب البرمجة الأخرى لأنه يركز على الكود البرمجي source code للبرامج الكاملة.

بدل من أن يعلمك مبادئ البرمجة فهو يساعدك على كتابة برامجك الخاصة.

هذا الكتاب يظهر لك برماج كاملة ثم يتم شرح كيف تعمل هذه البرامح.

عندما تقرأ هذا الكتاب قم بكتابة الكود البرمجي للبرامح المشرحة في هذا الكتاب بنفس الترتيب أو يمكنك تحميل ملفات الكود البرمجي للبرامح هذا الكتاب من الموقع:

<http://invpy.com/hackingsource>

البايثون هي لغة برمجة مفروعة وسهلة ومن خلال قراءة الفصول الأولى من هذا الكتاب يمكنك فهم العمل الذي سوف يقوم الكود بتنفيذه، إذا كنت قد انتقلت مباشرةً لقراءة الفصول القادمة وشعرت بالضياع حاول العودة للفصول السابقة

## أرقام الأسطر والمسافات:

عندما تقوم بإدخال الكود البرمجي بنفسك لا تقم بكتابة أرقام الأسطر التي تظهر في بداية كل سطر

مثلاً إذا وجدت الكود التالي:

```
1. number = random.randint(1, 20)
2. spam = 42
3. print('Hello world!')
```

فلا تقم بكتابة الأرقام الموجودة في بداية كل سطر ولا المسافات الموجودة بعد هذه الأرقام فقط قم بكتابة الأوامر كما في الشكل التالي:

```
number = random.randint(1, 20)
spam = 42
print('Hello world!')
```

هذه الأرقام فقط تستخدم في الشرح من أجل الإشارة إلى أسطر معينة في الكود وهي ليست جزء من الكود البرمجي الفعلي.

بالإضافة لذلك تأكد من إدخال الكود البرمجي بشكل مطابق تماماً للكود الموجود في هذا الكتاب وهذا يتضمن حالة الأحرف كبيرة أو صغيرة لأن البايثون هي لغة حساسة لحالة الأحرف

مثلاً في بايثون HELLO تختلف عن hello

لاحظ أن بعض الأسطر لا تبدأ من حافة الجانب الأيسر للصفحة ولكنها تبدأ بعد مسافة معينة 4 or 8 فراغات، كن متأكد من إدخال الرقم الصحيحة للمسافة في بداية كل سطر(كل الأحرف في IDLE لها نفس العرض، يمكنك معرفة عدد المسافات من خلال معرفة عدد الأحرف في السطر الأعلى من السطر الذي تنظر إليه)

مثلاً يمكنك أن ترى أن السطر الثاني متقدم بمسافة أربع خطوات لأن حرف (Whil) في السطر الأعلى هي أربعة أحرف

السطر الثالث متقدم بمسافة أربع خطوات عن السطر الثاني لأن (if n) هي أربع خطوات

```
while spam < 10:  
    if number == 42:  
        print('Hello')
```

## إلتلاف النص:

بعض الأسطر في الكود طويلة جداً ولا تتسع في سطر واحد فإن النص سوف يلف إلى السطر الثاني، عندما تقوم بكتابة هذه الأسطر في ملف نصي قم بإدخال النص بسطر واحد بدون أن تقوم بالنزول إلى السطر الثاني

يمكنك معرفة متى يبدأ السطر من خلا النظر إلى رقم السطر في الجانب الأيسر كما يظهر في المثال التالي الذي يحوي على سطرين فقط

```
1. print('This is the first line! xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
xxxxxxxxxxxxxx')  
2. print('This is the second line!')
```

## تخطيط البرنامج بشكل أون لاين:

تخطيط البرنامج أو إنشاء رسم تصوري له يظهر الكود في نفس الطريقة التي سيقوم الكمبيوتر بتتنفسه

بإمكانك زيارة الموقع التالي <http://invpy.com/hackingtraces> لرؤية مخطط كل برنامج في هذا الكتاب.

هذا الموقع يحوي على ملاحظات في كل خطوة وهي تشرح العمل الذي يقوم البرنامج بتنفيذها وهذا يساعدك على فهم البرامج بشكل أفضل.

## فحص الكود باستخدام أداة :Diff

طريقة سهلة لتعلم البايثون هي كتابة الكود البرمجي لهذه البرامج ولكن يمكن أن تكتب إحدى التعليمات بشكل خاطئ سهواً وهذا يؤدي إلى خطأ في تنفيذ البرنامج يمكنك نسخ النص البرمجي الذي قمت بكتابته ولصقه في online diff tool الموجودة في موقع الكتاب، هذه الأداة تظهر أي اختلاف بين الكود البرمجي في الكتاب والكود البرمجي الذي قمت أنت بكتابته

وهذه الطريقة تسهل عملية إيجاد أي خطأ كتابي في برنامجك  
[هذه الأداة موجودة في](http://invpy.com/hackingdiff)

فيديو تعليمي يشرح طريقة استخدام هذه الأداة موجود في  
<http://invpy.com/hackingvideos>

## نسخ ولصق النص:

عملية نسخ ولص النص مفيدة جداً وخاصة لأن العديد من النصوص التي سيتم تشفيرها وفك تشفيرها هي نصوص طويلة وبدلاً من كتابتها يمكنك نسخ النص من هذا الكتاب أو من النسخة الإلكترونية الأصلية لهذا الكتاب ولصقه في IDLE

يمكن رؤية كيفية القيام بهذه العملية من خلال <http://invpy.com/copypaste>

## روابط معلومات أخرى:

يوجد العديد من المصادر التي يمكن أن تساعدك في تعلم التشفير والبرمجة ولكن لن تكون  
بحاجة لتعلم كل الأمور الآن ولكن في بعض الأحيان أنت بحاجة لمعرفة هذه التفاصيل  
والشروط الإضافية، لذلك تم إضافة رابط معلومات إضافية للمساعدة على فهم بعض  
محتويات هذا الكتاب

رغم أن هذا الكتاب لا يحوي معلومات وشروط خطيرة رجاءً لا تستخدم محتوى هذا الكتاب بشكل سيء

البرمجة والتشفير:

البرمجة والتشفير مهاراتان منفصلتان ولكن تعلمها معاً هو أمر مفيد لأن الكمبيوتر يستطيع القيام بالتشفيير بشكل أسرع بكثير من الإنسان

**مثلاً:** التالي هو النص الكامل لرواية وليم شكسبير "روميو وجولييت" وهو مشفر بشيفرة استبدال

## بسیطة





إذا حاولت القيام بتنشيف هذا النص بشكل يدوي سوف تحتاج لأسابيع من العمل لإنجاز هذه المهمة ومن المحتمل أن تقع ببعض الأخطاء

حجاز الكنديه تستطيع تشفير هذا النص، أو فاكي تشفيره بشكاله كما، وبأقل من ثانتين

ولكنك بحاجة لمعرفة كيفية القيام ببرمجة الكمبيوتر للقيام بعملية التشفير وهذا هو محتوى هذا الكتاب.

إذا كنت تستطيع البرمجة فإنك تستطيع أيضاً فك شيفرة النصوص المشفرة التي شفرها آشخاص آخرون من أجل الحفاظ على أسرارهم تعلم البرمجة يمكنك أن تتعلم كيف تصبح هاكر.



## الفصل الثالث

# الشيل التفاعلية

---

محتوى هذا الفصل:

- الأرقام الصحيحة والأرقام الحقيقة
- العبارات الجبرية
- القيم
- العمليات
- حساب العبارات الجبرية
- تخزين القيم في المتغيرات
- إعادة الكتابة في المتغيرات

قبل أن نبدأ بكتابة برنامج التشفير يجب أن نتعلم أولاً بعض أساسيات البرمجة، هذه الأساسيات هي القيم والعمليات والعبارات الجبرية والمتغيرات.

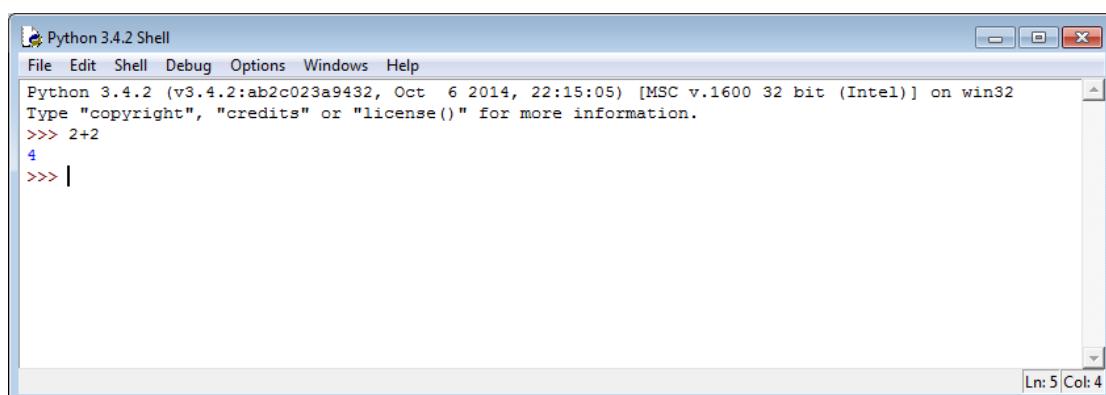
لنبدأ بتعلم كيفية استخدام شيل بايثون التفاعلية

يجب أن تستخدم جهاز كمبيوتر أثناء قراءة هذا الكتاب من أجل القيام بكتابة أكواد الأمثلة لترى بنفسك نتيجة تنفيذ هذه الأكواد.

## بعض أساسيات الرياضيات:

ابداً بفتح IDLE

يمكنك أن ترى الشيل التفاعلية والمؤشر >>> الذي يسمى prompt الشيل التفاعلية يمكن أن تعمل كآلية حاسبة أكتب 2+2 واضغط انتر، الإجابة ستكون 4



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> |
```

2+2 ليس برنامج ولكنه تعليمة مفردة

إشارة الجمع + تخبر الكمبيوتر بأن يقوم بجمع الرقمين 2 and 2

من أجل القيام بعملية الطرح استخدم إشارة الطرح ' - '

من أجل ضرب الأعداد استخدم الرمز ' \* '

ومن أجل القسم الرمز ' / '

المعامل	العملية
+	الجمع
-	الطرح
*	الضرب
/	القسمة

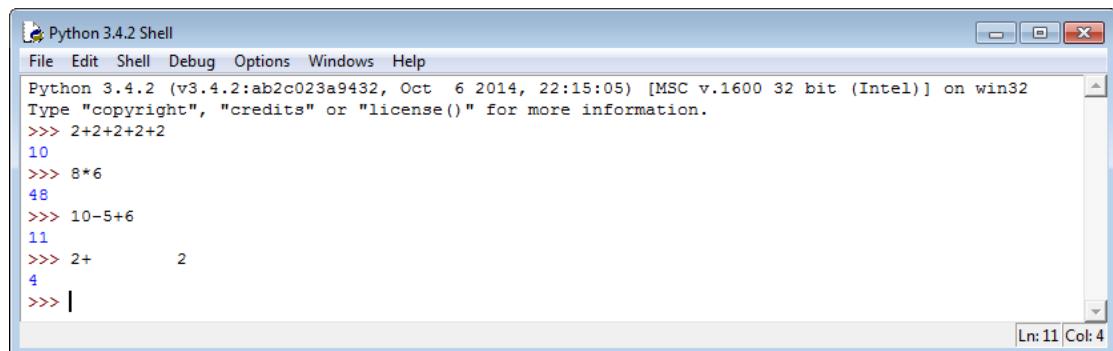
الرموز / \* + تسمى معاملات لأنها تخبر جهاز الكمبيوتر ليقوم بتنفيذ عملية على الأرقام أو القيم التي تكون حولها

## الأعداد الصحيحة والأعداد الحقيقية:

في البرمجة كل الأرقام مثل 0 and 99 and 4 تسمى أعداد صحيحة  
 الأرقام التي تحوي على فاصلة عشرية مثل 3.5 and 42.1 and 5.0 تسمى أعداد حقيقة  
 في بايثون الرقم 5 هو عدد صحيح  
 أما الرقم 5.0 هو رقم حقيقي

## العبارة الجبرية:

حاول كتابة العبارات الجبرية في الشيل واضغط انتر



```

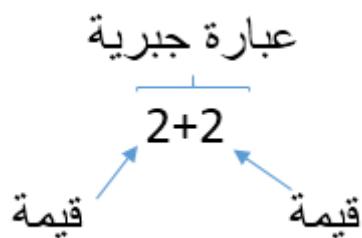
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2+      2
4
>>> |

```

هذه المسائل الرياضية تسمى عبارات جبرية

جهاز الكمبيوتر قادر على حل الملايين من هذه المسائل خلال ثوانٍ قليلة  
 العبارة الجبرية تكون مكونة من أكثر من قيمة (أرقام) ومتصلة مع بعضها بواسطة العماملات  
 (العمليات الرياضية)

ويمكن أن تحوي على أي مسافة بين الأعداد والإشارات ولكن كن متأكد دائماً من أن تبدأ السطر من أوله



## ترتيب العمليات:

يمكن أنك تتذكر ترتيب العمليات من دروس الرياضيات في المدرسة  
 الضرب والقسمة لها أولوية على الجمع والطرح

الబایتون يتبع هذه القاعدة فعندما يرى البيايثون `+ * and` فهو يقوم بتنفيذ عملية الضرب أولاً  
 جرب المثال التالي في الشيل التفاعلي

```

>>> 2 + 4 * 3 + 1
15
>>>
    
```

لأن عملية الضرب تنفذ أولاً

$$2 + 4 * 3 + 1 = 2 + 12 + 1 = 15$$

كما يمكن استخدام الأقواس من أجل تغيير أولوية العمليات، بحيث يتم تنفيذ ما داخل الأقواس أولاً، جرب المثال التالي:

```
>>> (2 + 4) * (3 + 1)
24
>>>
```

## حساب العبارات الجبرية:

عن الانتهاء من حل أو حساب العبارة الجبرية  $5 + 10$  والحصول على القيمة 15 نقوم بأن العبارة الجبرية تم حسابها أو إيجاد نتيجتها

عملية حساب العبارة الجبرية هي تحويل العبارة إلى قيمة وحيدة نهائية  
العبارة الجبرية  $5 + 10 + 2 + 3$  لها نفس القيمة النهائية وهي 15

## الأخطاء:

يمكن أن تقوم بكتابة كود أو تعليمات خاطئة وعندما فإن الباليثون ببساطة سوف يخبرك بوجود خطأ ويجب عليك مراجعة الكود وتصليحه والمحاولة مرة ثانية.

مالم تكن تمتلك خبرة في البرمجة لن تتمكن من فهم رسائل الخطأ ولكن يمكنك دائمًا البحث بواسطة Google عن نص رسالة الخطأ لتجد صفحة الويب الخاصة بهذا الخطأ.

كما يمكنك أيضًا الذهاب إلى <http://invpy.com/errors> لرؤية قائمة برسائل الأخطاء الشائعة في لغة الباليثون.

## A: المجموعة، الفصل 3، أمثلة عملية:

يمكنك إيجاد أمثلة عملية في <http://invpy.com/hackingpractice3A>

## أنواع البيانات:

أنواع البيانات هي تصنيفات مثل عدد صحيح أو عدد حقيقي أو سلسلة نصية أو قائمة.  
لكل قيمة نوع بيانات محدد.

مثلاً القيمة 42 هي عدد صحيح `integer` ويرمز له بشكل مختصر `int`  
أما القيمة 7.5 فهي عدد حقيقي `float` ويرمز لها بنفس الاسم  
وهناك أنواع أخرى من البيانات سوف نتعرف عليها مثل النوع `string` (السلسلة النصية)  
الآن فقط تذكر أنه لكل قيمة نوع بيانات معين.

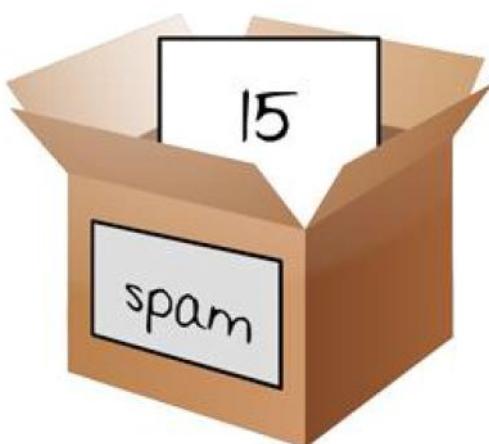
وتذكر أن 42 هو عدد صحيح `int` أما 42.0 فهو عدد حقيقي `float`.

## تخزين القيم في المتغيرات:

في برامجنا سنكون بحاجة لحفظ القيم التي نقوم بحسابها، يمكننا تخزين القيم في المتغيرات  
المتغير هو كالصندوق الذي يمكن أن تحفظ القيمة في داخله ويتم ذلك باستخدام إشارة المساواة  
"=" بين اسم المتغير والقيمة المراد تخزينها

المثال التالي هو حفظ القيمة 15 في المتغير ذو الاسم `spam`

```
>>> spam = 15  
>>>
```



هذه العملية تقوم بحجز مكان في الذاكرة باسم `spam` وتقوم بحفظ القيمة 15 داخل هذه الحجرة الذاكرة.

لسهولة الفهم يمكنك ان تخيل بأن الحجرة الذاكرة هي صندوق وله اسم `spam` وفي داخله القيمة 15.

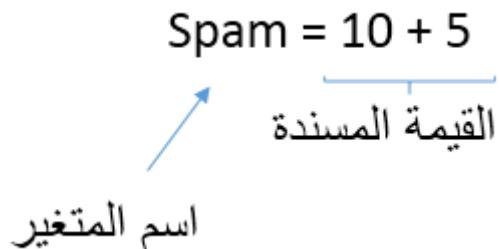
اسم المتغير `spam` يمكن أن يكون أي اسم أو أي حرف أنت تختاره (باستثناء بعض الكلمات الممحوزة)

بعد كتابة اسم المتغير وإسناد القيمة المراد حفظها والضغط على انتر فإنك لن ترى أي شيء فقط سوف تنتقل إلى سطر جديد (ما لم يكن هناك رسالة خطأ)

وعندها يمكنك أن تفترض بأن عملية اسناد القيمة إلى المتغير تمت بنجاح.

ويمكنك بعدها استدعاء هذا القيمة ببساطة من خلال كتابة اسم المتغير في الشيل التفاعلية

```
>>> spam = 15  
>>> spam  
15  
>>>
```



المتغير يقوم بتخزين او حفظ قيمة واحدة فقط وليس عبارة جبرية

مثلاً إذا تم اسناد العبارة الجبرية  $10 + 5$  إلى المتغير `spam`

فإن القيمة التي سوف تخزن هي 15

بعد القيام بعملية تخزين او حفظ القيمة 15 داخل المتغير `spam` يمكن القيام بعمليات مثل

```
>>> spam = 15
>>> spam + 5
20
>>>
```

هذه العملية تقوم بإضافة القيمة 5 إلى القيمة الموجودة داخل المتغير `spam`

## إعادة الكتابة في المتغيرات:

يمكن تغيير القيمة الموجودة ضمن المتغير من خلال عملية إسناد جديدة كما في المثال التالي:

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
>>>
```

في أول مرة تم إدخال `5 + spam` ظهرت قيمة النتيجة 20

لأن القيمة المخزنة في المتغير `spam` كانت 15

بعد عملية الإسناد الجديدة `3 = spam` تم حفظ القيمة الجديدة 3 ضمن المتغير `spam`

وبعد تنفيذ الأمر `5 + spam` ظهرت النتيجة 8 لأن  $8 = 3 + 5$

كما يمكن تغيير القيمة الموجودة في المتغير كما في المثال التالي:

```
>>> spam = 15
>>> spam = spam + 5
20
>>>
```

العبارة `5 + spam = spam` تخبر الكمبيوتر بأن القيمة الجديدة للمتغير `spam` هي القيمة القديمة لـ `spam` مضافاً إليها العدد 5

تذكر دائمًاً عندما يكون اسم المتغير في الجانب الأيسر لإشارة المساواة فهذا يعني أنه سيتم اسناد قيمة جديدة في داخله أما عندما يكون اسم المتغير موجود في الطرف الأيمن لإشارة المساواة فهذا يعني أنه سوف يتم استخدام القيمة الموجودة داخل هذا المتغير

يمكن أن نقوم بزيادة قيمة المتغير أكثر من مرة كما في المثال التالي:

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
>>>
```

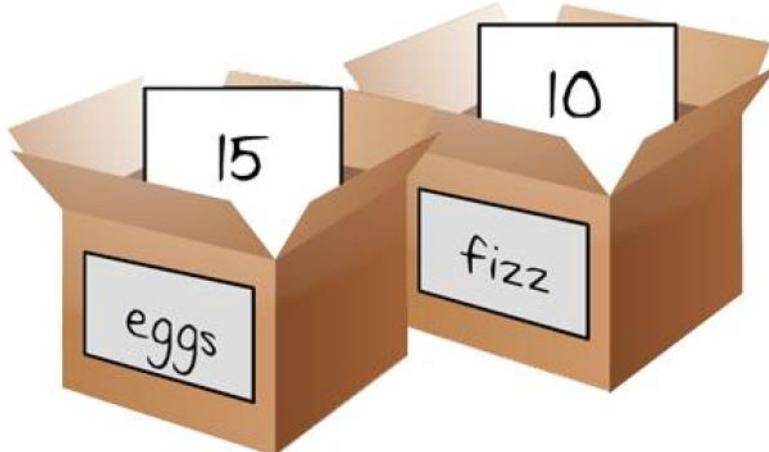
## استخدام أكثر من متغير:

يمكن أن يحوي البرنامج على أكثر من متغير في المثال التالي سوف نستخدم متغيرين مختلفين eggs and fizz (اسم المتغير يمكن أن يكون أي حرف أو أي اسم أنت تختاره باستثناء بعض الكلمات المحجوزة في لغة البايثون)

```
>>> fizz = 10
>>> eggs = 15
```

المتغير fizz يحوي القيمة 10 في داخله

المتغير eggs يحوي القيمة 15 في داخله



في المثال التالي سوف نقوم بإسناد قيمة جديدة للمتغير spam

```
>>> fizz = 10
>>> eggs = 15
>>> spam = fizz + eggs
>>> spam
25
>>>
```

القيمة الجديدة للمتغير spam أصبحت 25 لأن  $10 + 15 = 25$

## أسماء المتغيرات:

الكمبيوتر لا يهتم باسم المتغير (يمكنك اختيار أي حرف أو أي اسم باستثناء بعض الكلمات المحجوزة) ولكن يجب أن تستخدم أسماء للمتغيرات تعكس أو توضح نوع البيانات التي سيتم حفظها في هذا المتغير وهذا من أجل تسهيل فهم العمل الذي سوف يقوم البرنامج بتنفيذه.

يمكن أن تقوم بتسمية المتغير ب asssnsnskkllfdsfdf or monkey والكمبيوتر سوف يقوم بتشغيل البرنامج بشكل طبيعي.

أسماء المتغيرات (مثل أي شيء آخر في لغة بايثون) في حساسة لحالة الأحرف (كبيرة أو صغيرة)، مثل المتغير spam مختلف عن SPAM or Spam

أنها فكرة سيئة بأن تقوم باستخدام نفس الاسم للمتغيرات ولكن باختلاف حالة الأحرف

إذا قمت ب تخزين الاسم الأول في المتغير name والاسم الثاني في NAME فسوف يكون الأمر مربك أو مشوش عندما تقوم بمراجعة الكود البرمجي بعد فترة من كتابته ولكن تقنياً فإن البرنامج سوف يعمل بدون وجود أي تعليمة خطأ syntax errors ولكن سوف يعمل بشكل غير صحيح وهذا النوع من الأخطاء يسمى bug المبرمجون لا يقومون فقط بكتابة البرنامج بل يقومون أيضاً بتصليح الأخطاء fixing bugs

## حالة الجمل:

طريقة أخرى للمساعدة عندما تكون بحاجة لحفظ قيمة نصية مثل ما الذي تناولته في الأقطار كاسم متغير

whatihadforbreakfast اسهل للقراءة من whatIHadForBreakfast هذا يسمى حالة الجمل camel case

وتحتم العملية باستخدام الأحرف الصغيرة والكبيرة لتسهيل قراءة الجملة وهو أمر اختياري

## الخلاصة – متى سوف نبدأ عمليات فك الشيفرات:

قريباً ولكن قبل أن نستطيع فك الشيفرات يجب أن نتعلم أساسيات البرمجة ولن تكون بحاجة لتعلم الكثير قبل البدء بكتابة برامج فك التشفير ولكن بقي فصل واحد في البرمجة يجب أن نغطيه قبل أن ننتقل للشيفرات.

في هذا الفصل تعرفت على أساسيات البرمجة بلغة البايثون وذلك باستخدام الشيل التفاعلية كما تعرفت على أنواع المتغيرات والعمليات الرياضية وسوف تتعلم عن هذه الأمور بشكل أكثر من خلال الفصول القادمة.

في الفصل التالي سوف نتعلم كيفية كتابة برنامج يحوي على أكثر من تعليةمة والذي يمكن أن نستخدمه لتنفيذ هذه التعليمية لأكثر من مرة سوف نتعرف على بعض الأساسيات وسوف تقوم بكتابة أول برنامج خاص بك.



## كتابة البرامج

---

محتوى هذا الفصل:

- السلاسل النصية
- تكرار السلاسل
- استخدام IDLE لكتابة الكود البرمجي
- حفظ وتشغيل البرامج في IDLE
- التابع print()
- التابع input()
- التعليقات

اكتفينا من استخدام الرياضيات والأعداد الصحيحة الآن

البايثون هو أكثر من أن يكون آلة حاسبة، في هذا الفصل سوف نتعلم كيفية تخزين نص داخل المتغيرات، وجمع أكثر من نص مع بعض وعرض النص على الشاشة وسوف تقوم بكتابة برنامجك الأول الذي سوف يقوم بالترحيب بالمستخدم باستخدام النص Hello World ويسمح للمستخدم بكتابة اسمه.

## السلسل النصية: Strings

في بايثون سوف نعمل مع قيمة نصية تسمى string

كل برامج التشفير وفك التشفير سوف تتعامل مع سلاسل نصية كنص صريح أو نص مشفر.

النص الصريح والنص المشفر سيتم تمثيلهم في برامجنا باستخدام القيمة النصية.

يمكن تخزين القيم النصية داخل المتغيرات كما في حالة تخزين الأعداد الصحيحة والحقيقة ولكن في حالة القيم النصية يجب أن نضع القيمة النصية داخل علامات التنصيص ( ' )

جرب المثال التالي في الشيل التفاعلية:

```
>>> spam = 'hello'  
>>>
```

علامة التنصيص ليست جزء من القيمة النصية ومن خلالها يعرف البايثون أن نوع القيمة المراد تخزينها هي سلسلة نصية.

إذا قمت بكتابة اسم المتغير spam في الشيل فسوف يتم عرض المحتوى المخزن في المتغير 'hello' وهو القيمة النصية spam

```
>>> spam = 'hello'  
>>> spam  
'hello'  
>>>
```

السلسلة النصية يمكن أن تكون مكونة من أكثر من كلمة ويمكن أن تحوي على أرقام وأحرف ورموز مختلفة

```
>>> 'hello'  
'hello'  
>>> 'Hi there!'  
'Hi there!'  
>>> 'KITTENS'  
'KITTENS'  
>>> ''  
''  
>>> '7 apples, 14 oranges, 3 lemons'  
'7 apples, 14 oranges, 3 lemons'  
>>> 'Anything not pertaining to elephants is irrelephant.'  
'Anything not pertaining to elephants is irrelephant.'  
>>> '0*#%&0cfY0*&gfc%Y0*&%3yc8r2'  
'0*#%&0cfY0*&gfc%Y0*&%3yc8r2'
```

لاحظ أن السلسلة النصية '' هي سلسلة نصية فارعة ولا تحوي على أي حرف بين علامتي التنصيص وهي تسمى **blank string or empty string**

## جمع السلاسل:

يمكن أن تقوم بجمع أكثر من سلسلة نصية باستخدام إشارة الجمع '+' كما في المثال التالي:

```
>>> 'Hello' + 'World!'  
'HelloWorld!'  
>>>
```

من أجل ترك فراغ أو مسافة بين الكلمتين "Hello" and "World" يمكنك إضافة مسافة بعد كلمة 'Hello' وقبل إشارة التنصيص ( )

```
>>> 'Hello ' + 'World!'  
'Hello World!'  
>>>
```

تذكر أن بإثون سوف يقوم بجمع محتوى السلاسل كما هي بالضبط، إذا كنت تريد مسافة في نتيجة الجمع فيجب عليك ترك مسافة في السلسلة النصية الأصلية.

القيمة النصية 'Hello' يتم تخزينها في متغير من نوع **string** عملية الجمع يجب أن تكون بين متغيرين أو بين قيمتين لها نفس النوع ولا يمكن جمع متغيرين أو قيمتين من نوعين مختلفين.

مثلاً لا يمكن جمع متغير من نوع عدد صحيح int مع متغير من نوع سلسلة نصية string

وعند القيام بمثل هذه العملية سوف تظهر رسالة خطأ كما في المثال التالي:

```
>>> 'Hello' + 45
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    'Hello' + 45
TypeError: Can't convert 'int' object to str implicitly
>>>
```

## تكرار السلسلة النصية باستخدام المعامل (\*) :

يمكنك أيضاً استخدام المعامل '\*' بين سلسلة نصية وعدد صحيح من أجل تكرار السلسلة النصية عدد من المرات بحسب قيمة العدد الصحيح.

جرب المثال التالي في الشيل التفاعلية:

```
>>> 'Hello' * 3
'HelloHelloHello'
>>> spam='abcdef'
>>>
>>> 'Hello' * 3
'HelloHelloHello'
>>> spam = 'ABCDEF'
>>> spam = spam * 3
>>> spam
'ABCDEFABCDEFABCDEF'
>>> spam = spam * 2
>>> spam
'ABCDEFABCDEFABCDEFABCDEFABCDEFABCDEF'
```

المعامل '\*' يمكن أن يكون بين متغيرين أو قيمتين من نوع عدد صحيح integer (عندما تكون عملية ضرب للأعداد) أو يمكن أن يكون بين قيمة نصية وعدد صحيح (عندما تكون عملية تكرار السلسلة النصية) ولكنه لا يمكن أن يكون بين قيمتين نصيتين وفي هذه الحالة سوف تظهر رسالة خطأ كما في المثال التالي:

```
>>> 'Hello' * 'world'
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    'Hello' * 'world'
TypeError: can't multiply sequence by non-int of type 'str'
>>>
```

إذا في لغة بایثون يمكن جمع السلاسل النصية باستخدام المعامل '+ ولكن لا يمكن ضرب السلاسل النصية بعضها باستخدام المعامل '\*'.

## طباعة القيم باستخدام التابع `:print()`

يوجد نوع آخر من التعليمات في لغة بایثون وهو استدعاء التوابع `call function` مثلًا يمكنك استدعاء التابع `print()` والذي يقوم بطباعة القيم التي تمرر إليه كما في المثال التالي:

```
>>> print('Hello')
Hello
>>> print(42)
42
>>>
```

التابع (كما في المثال السابق `print()`) يملك كود معين ليقوم بتنفيذ مهمة معينة مثل طباعة القيمة التي تمرر له على الشاشة.

يوجد عدة أنواع مختلفة من التوابع في لغة بایثون وعند استدعاء التابع فسوف يتم تنفيذ الكود البرمجي الموجود في جسم التابع.

في المثال السابق استخدمنا التابع `print()` لطباعة القيمة التي تمرر له على الشاشة يمكن تمرير أكثر من قيمة للتابع `print()` كما في المثال التالي:

```
>>> spam = 'Ali'
>>> print('Hello, ' + spam)
Hello, Ali
>>>
```

في بعض الحالات تكون بحاجة لاستخدام بعض الرموز مثل علامة التنصيص المفردة (') كجزء من السلسة النصية ولكن يمكن أن نحصل على رسالة خطأ لأن بایثون يعتقد أن إشارة

التنصيص المفردة هذه هي للدلالة على انتهاء قيمة السلسة النصية والأحرف أو الكلمات الموجودة بعض إشارة التنصيص هذه سوف تكون عبارة عن كود غير مفهوم

جرب المثال التالي في الشيل التفاعلي:

```
>>> print('Ali's cat is name 3nter.')
SyntaxError: invalid syntax
>>>
```

من أجل استخدام علامة التنصيص المفردة ( ' ) يتم ذلك من خلال كتابة ( \' )  
وунدها سوف يفهم البائثون أن إشارة التنصيص هذه هي جزء من النص وليس للدلالة على  
نهاية النص

جرب المثال التالي:

```
>>> print('Ali\'s cat is name 3nter')
Ali's cat is name 3nter
>>>
```

وبشكل مماثل يمكن استخدام عدد من الرموز بعد \ من أجل القيام بطباعة أمر معين كما يظهر  
في الجدول التالي:

نتيجة التنفيذ	الرمز
يطبع \	\\
يطبع '	'
يطبع "	"
ينتقل لسطر جديد	\n
يترك مسافة tab	\t

لاحظ في المثال التالي عند استخدام \ وبعدها الحرف t ترك مسافة tab رغم أن \ والحروف t  
هما جزء من النص لذلك انتبه لهذه العملية

```
>>> print('He flew away in a green \teal helicopter.')
He flew away in a green          eal helicopter.
>>> |
```

من أجل طباعة \| وبعدها حرف t بدون ترك مسافة tab يتم ذلك باستخدام \\

كما في المثال التالي:

```
>>> print('He flew away in a green \\teal helicopter.')
He flew away in a green \teal helicopter.
>>> |
```

## علامات التنصيص المفردة والمزدوجة:

يمكن أن تكون السلسلة النصية بين علامتي تنصيص مزدوجة " "

```
>>> print('Hello world')
Hello world
>>> print("Hello world")
Hello world
>>> |
```

ولكن لا يمكن الجمع بين علامة تنصيص مفردة وعلامة تنصيص مزدوجة وفي هذه الحالة سوف تظهر رسالة خطأ

```
>>> print('Hello world') |
```

SyntaxError: EOL while scanning string literal  
>>>

أنا أفضل استخدام علامة تنصيص واحد وهذا أسهل لأن علامة التنصيص المزدوجة بحاجة لأن تضغط على زر shift

## الفهرسة :indexing

برامج فاك التشفير سوف تحتاج للحصول على حرف واحد من السلسلة النصية ويتم ذلك باستخدام قوسين [ ] بعد اسم المتغير من النوع النصي وكتابة رقم الحرف داخل القوسين (كالتعامل مع المصفوفات) هذا الرقم يخبر البايثون بموضع الحرف داخل السلسلة النصية

- الرقم 0 يشير إلى الحرف الأول
- الرقم 1 يشير إلى الحرف الثاني
- الرقم 2 يشير إلى الحرف الثالث وهكذا

جرب المثال التالي:

```
>>> spam = 'Hello'  
>>> spam[0]  
'H'  
>>> spam[1]  
'e'  
>>> spam[2]  
'l'  
>>> |
```

تذكر دائماً أن عملية الفهرسة أو العنونة تبدأ بالرقم 0 وليس الرقم 1

string: ' 

H	e	l	l	o
---	---	---	---	---

 ',  
indexes: 0 1 2 3 4

عملية الفهرسة يمكن أن تتم بطريقة أخرى كما في المثال التالي:

```
>>> 'Syria'[2]  
'r'  
>>>
```

أو بالشكل التالي:

```
>>> var = 'Syria'[2]
>>> var
'r'
>>>
```

إذا قمت بإدخال رقم أكبر من عدد أحرف السلسلة النصية فسوف يرد بـPython رسالة الخطأ  
"string index out of range"

جرب المثال التالي:

```
>>> 'Syria'[10]
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    'Syria'[10]
IndexError: string index out of range
>>>
```

## الفهرسة السلبية:

هذا النوع من الفهرسة يبدأ من نهاية السلسلة النصية ثم يعود إلى بدايتها

- الرقم 1- يشير إلى آخر حرف في السلسلة النصية
- الرقم 2- يشير إلى الحرف ما قبل الأخير في السلسلة النصية وهكذا

جرب المثال التالي:

```
>>> 'Syria'[-1]
'a'
>>> 'Syria'[-2]
'i'
>>> 'Syria'[-3]
'r'
>>> 'Syria'[-4]
'y'
>>> 'Syria'[-5]
'S'
>>>
```

## التقطيع:

إذا كنت تريد الحصول على أكثر من حرف من السلسلة النصية، يمكنك استخدام التقطيع بدل الفهرسة وهذه العملية تتم أيضاً باستخدام قوسين [ ] ولكن داخل القوسين يوجد رقمين وبينهما نقطتان ' : ' للإشارة إلى بداية ونهاية مجال التقطيع

جرب المثال التالي:

```
>>> 'Syria'[0:3]
'Syr'
>>>
```

السلسلة المقطعة تبدأ من الرقم الأول داخل القوسين وتنتهي مع الرقم الثاني الموجود داخل القوسين وتتضمن كل الأحرف الموجودة بينهما

جرب المثال التالي:

```
>>> 'Hello world!'[0:5]
'Hello'
>>> 'Hello world!'[6:12]
'world!'
>>> 'Hello world!'[-6:-1]
'world'
>>> 'Hello world!'[6:12][2]
'r'
>>>
```

لاحظ عن استخدام 'Hello world!'[6:12][2] فسيتم اختيار الحرف الثالث ضمن المجال [6:12]

وبشكل مختلف عن الفهرسة فإن عملية الاقطاع لن تظهر رسالة خطأ في حال استخدام أكبر من طول السلسلة النصية وبدل ذلك سوف تقوم بإعادة أكبر طول ممكن

جرب المثال التالي:

```
>>> 'Syria'[0:999]
'Syria'
>>> 'Syria'[2:999]
'ria'
>>> 'Syria'[100:200]
''
>>>
```

في حال ترك مكان الرقم الأول فارغ فإن باليثون وبشكل أتوماتيكي سوف يفترض أن هذا الرقم هو 0

جرب المثال التالي:

```
>>> 'Syria'[:3]
'Syr'
>>> 'Syria'[0:3]
'Syr'
>>>
```

في حال كان مكان الرقم الثاني فارغ فإن باليثون وبشكل أتوماتيكي يفترض أنك تريد بقية السلسلة

جرب المثال التالي:

```
>>> 'Syria'[2:]
'ria'
>>>
```

الاقطاع هو عملية بسيطة من أجل الحصول على سلسلة نصية فرعية ولكن هذه السلسلة الفرعية تبقى مثل أي سلسلة أخرى

جرب المثال التالي:

```
>>> myName = 'Jameel Huseen Tawelh'
>>> myName[-6:]
'Tawelh'
>>> myName[:13]
'Jameel Huseen'
>>>
```

## كتابة برنامج في محرر النصوص في IDLE:

### IDLE's File Editor

لحتى الآن نحن نقوم بكتابية تعليمة واحدة في كل مرة في الشيل التفاعليه وعند كتابة برنامج نحن بحاجة لكتابه وتنفيذ عدد من التعليمات.

لنبدأ بكتابة برنامجنا الأول

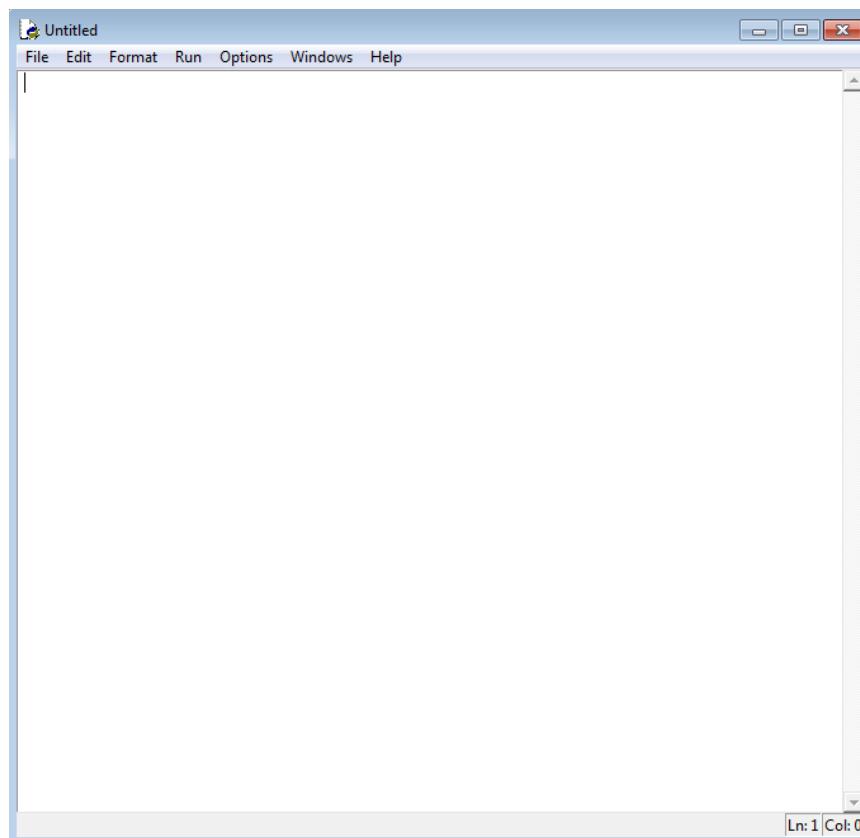
اسم البرنامج (السوفت وير) software الذي يؤمن الشيل التفاعليه هو

IDLE: Interactive Development Environment

IDLE يملك جزء آخر بجانب الشيل التفاعليه وهو محرر النصوص file editor

في القسم العلوي من نافذة الشيل اضغط على File > New File

سوف تظهر نافذة جديدة فارغة من أجل كتابة البرنامج في داخلها، هذه النافذة هي محرر النصوص file editor



يمكنك دائمًا التمييز بين محرر النصوص والشيل التفاعلية من خلال <>> التي تظهر دائمًا في الشيل التفاعلية

## :Hello World!

الطريقة التقليدية لتعليم البرمجة للمبتدئين هي كتابة برنامج بسيطة يقوم بطباعة النص

"Hello world"

سوف نقوم بكتابه هذا البرنامج الآن

قم بإدخال النص التالي في نافذة محرر النصوص، هذا النص هو الكود البرمجي للبرنامج لأنه يحوي على تعليمات بلغة البيايثون والتي سوف يقوم البيايثون بتنفيذها

## ال코드 البرمجي لـHello world

```
1. # This program says hello and asks for my  
name.  
2. print('Hello world!')  
3. print('What is your name?')  
4. myName = input()  
5. print('It is good to meet you, ' + myName)
```

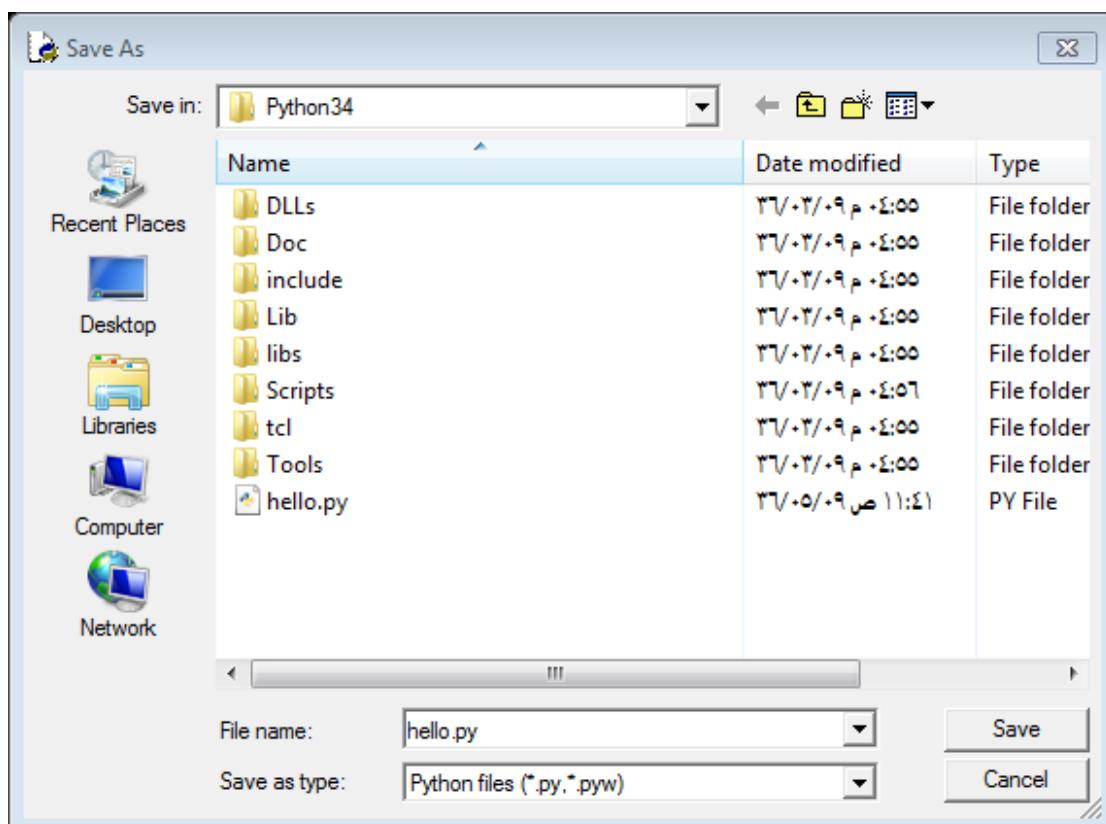
```
#This program says hello and asks for my name.
print('Hello world!')
print('What is your name?')
myName = input()
print('It is good to meet you, ' + myName)
```

Ln: 1 Col: 0

## حفظ البرنامج:

بعد أن تنتهي من ادخال الكود البرمجي قم بحفظه من خلال

عند ظهور نافذة الحفظ ادخل الاسم `hello.py` واختر



يجب أن تقوم بحفظ البرنامج في كل مرة تقوم بالتعديل على الكود البرمجي ويتم ذلك باستخدام

Ctrl + s

يمكنك مشاهدة فيديو تعليمي ل كيفية استخدام محرر النصوص من موقع الكتاب

<http://invpy.com/hackingvideos>

## تشغيل البرنامج:

الآن حان وقت تشغيل برنامجك الأول

أختار Run > Run Module أو فقط قم بالضغط على F5

برنامتك يجب أن يبدأ بالعمل في نافذة الشيل

```
Hello world!
What is your name?
Jameel Huseen Tawelh
It is good to meet you, Jameel Huseen Tawelh
>>>
```

عندما يقوم البرنامج بسؤالك عن اسمك قم بإدخال اسمك واضغط انتر

مبروك لقد أصبحت الآن مبرمج و قمت بكتابة برنامجك الأول.



## تشغيل البرامج التي تم حفظها:

أغلق نافذة محرر النصوص من خلال الضغط على X في الزاوية العليا من أجل إعادة فتح البرنامج الذي تم حفظه اختر File > Open واختر الملف الذي تريد فتحه

## كيف يعمل برنامج "hello world" :

كل سطر تم إدخاله هو عبارة عن تعليمية تخبر باليثون بما يجب عليه القيام به

برنامج الكمبيوتر هو عبارة عن وصفة تخبر الكمبيوتر قم بهذه الخطوة ثم قم بهذه الخطوة وهكذا إلى أن تصل إلى النهاية وبعد تنفيذ أول تعليمية في البرنامج ينتقل لتنفيذ التعليمية التالية وهكذا وتسمى هذه العملية تنفيذ البرنامج execution

تنفيذ البرنامج يبدأ من أول سطر في كود البرنامج ثم ينتقل إلى الأسفل.

البرنامج يمكن أن يتجاوز عدد من التعليمات ويقوم بتنفيذ تعليمات لاحقة وهذا ما سنعرف عليه في الفصل القادم.

لنبدأ بشرح تعليمات برنامجنا الأول

## التعليقات:

```
1. # This program says hello and asks for my name.
```

hello.py

هذا السطر يسمى تعليق comment

التعليقات لا تكتب من أجل الكمبيوتر بل تكتب من أجل المبرمج جهاز الكمبيوتر يتغافل هذه التعليقات وهي تكتب فقط من أجل تذكر المبرمج بالمهمة التي سيقوم البرنامج بتنفيذها.

أي نص يأتي بعد الإشارة # يكون عبارة عن تعليق (يستخدم لتسهيل قراءة الكود البرمجي) وسوف يتم تجاهله عند تنفيذ البرنامج.

## التابع :function

لقد قمنا باستخدام التابع print() والأمر الجميل في هذا التابع أنك بحاجة فقط لمعرفة كيفية استخدامه، هذا التابع يقوم بعرض النص الممرر إليه على الشاشة ولكنك لسه بحاجة لمعرفة كيف تتم هذه العملية، الآن يكفي فقط أن تتعلم كيفية استخدام هذا التابع.

استدعاء التابع function call هو الجزء من الكود الذي يخبر البرنامج أن يقوم بتنفيذ الكود الموجود في جسم التابع المستدعى ومن ثم العودة لتنفيذ باقي تعليمات البرنامج.

التابع () print() يقوم بأخذ القيمة الممررة إليه (التي يتم كتابتها داخل القوسين () ) ويقوم بعرضها على الشاشة.

لأننا نريد طباعة Hello world! على الشاشة لذلك قمنا بكتابة هذه العبارة داخل الأقواس الخاصة بالتابع print

## التابع :print()

```
2. print('Hello world!')  
3. print('What is your name?')
```

hello.py

لقد قمنا باستدعاء التابع print() ليقوم بطباعة السلسلة التي يتم تمريرها داخل الأقواس () هذا التابع يقوم بطباعة القيمة الممرره له على الشاشة.

## التابع :input()

```
4. myName = input()
```

hello.py

في السطر الرابع من الكود البرمجي قمنا بإسناد قيمة التابع input() إلى المتغير myName عندما يتم استدعاء التابع input() فإن البرنامج ينتظر المستخدم إلى أن يقوم بإدخال نص معين ويضغط انتر.

في هذا البرنامج فإن المستخدم يقوم بإدخال اسمه (سلسلة نصية) والتي سوف يتم حفظها في المتغير myName

التابع input() يعيد القيمة التي يقوم المستخدم بإدخالها أثناء استدعاء التابع input() لا يتم تمرير أي قيمة داخل القوسين () وهذا التابع يقوم فقط بإعادة القيمة التي يقوم المستخدم بإدخالها

```
5. print('It is good to meet you, ' + myName)
```

في السطر الخامس قمنا باستدعاء التابع `print()` مع استخدام معامل الجم ( + ) من أجل دمج السلسلة النصية 'It is good to meet you' مع اسم المستخدم الذي يقوم بإدخاله.

تذكر أن جهاز الكمبيوتر يقوم فقط بتنفيذ الأوامر التي يطلبها البرنامج منه.

البرنامج السابق مبرمج لكي يتطلب من المستخدم أن يقوم بإدخاله اسمه ويسمح له بإدخال قيمة نصية ثم يقول `Hello world` ويظهر السلسلة النصية التي قام المستخدم بإدخالها.

الكمبيوتر والrogram لا يهتمون بالقيمة النصية (الاسم) التي يقوم المستخدم بإدخالها إن كانت اسم المستخدم الحقيقي أو أي اسم آخر أو حتى أي كلمات أخرى ويتم عرض السلسلة النصية تماماً كما أدخلها المستخدم.

## الخلاصة:

كتابة البرامج هي طريقة للتحدث مع الكمبيوتر، في هذا الفصل قمنا بجمع عدة تعليمات من لغة البايثون من أجل كتابة برنامج كامل يقوم بسؤال المستخدم عن اسمه ثم يرحب به

كل البرامج القادمة في هذا الكتاب سوف تكون أكثر تعقيد ولكن لا تقلق لأننا سنقوم بشرح هذه البرامج سطراً سطراً ويمكنك دائماً إدخال التعليمات في الشيل التفاعلية لترى العمل الذي تقوم به قبل استخدامها في البرنامج الكامل.

الآن سوف نبدأ ببرограмنا الأول لفك تشفير الشيفرة العكسية `reverse cipher`



# الفصل الخامس

## الشيفرة العسكرية

---

محتوى هذا الفصل:

- التابع (len)
- حلقة while
- البيانات من النوع البوليني Boolean (المنطقي)
- معاملات المقارنة
- الأوامر الشرطية

“Every man is surrounded by a neighborhood of voluntary spies.”

Jane Austen

## الشيفرة العكسية :Reverse Cipher

الشيفرة العكسية تقوم بتشифير الرسالة من خلال طباعة أحرف الرسالة بشكل عكسي.

مثلاً الرسالة "Hello world!" يتم تشفيرها بالشكل التالي "olleH dlrow !" من أجل فك التشفير يمكن عكس الرسالة المشفر من أجل الحصول على الرسالة الأصلية.

في هذه الشيفرة فإن خطوات التشفير وفك التشفير هي نفسها.

الشيفرة العكسية هي شيفرة ضعيفة جداً ويمكن كشفها من خلال النظر فقط إلى النص المشفر سوف نستخدم هذه الشيفرة لأنها سهلة الشرح وسيكون من السهل كتابة برنامجنا الأول لفك التشفير لهذه الشيفرة.

### ال코드 البرمجي لبرنامج الشيفرة العكسية :

في IDLE اختر file > New File وقم بكتابة الكود التالي ثم قم حفظه باسم reverseCipher.py

(لا تكتب أرقام الأسطر في الكود البرمجي)

#### Source code for reverseCipher.py

```
1. # Reverse Cipher
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. message = 'Three can keep a secret, if two of them are dead.'
5. translated = ''
6.
7. i = len(message) - 1
8. while i >= 0:
9.     translated = translated + message[i]
10.    i = i - 1
11.
12. print(translated)
```

```
# Reverse Cipher
# http://inventwithpython.com/hacking (BSD Licensed)

message = '.daed era meht fo owt fi ,terces a peek nac eerhT'
translated = ''

i = len(message) - 1
while i >= 0:
    translated = translated + message[i]
    i = i - 1

print(translated)
```

Ln: 2 Col: 0

عند تنفيذ هذا البرنامج ستكون النتيجة كالتالي:

```
>>>
.daed era meht fo wot fi ,terces a peek nac eerhT
>>>
```

من أجل فك تشفير هذه الرسالة

'.daed era meht fo owt fi ,terces a peek nac eerhT'

قم بنسخ النص المشفر ولصقه في السطر الرابع ليتم حفظه في المتغير message

وتأكد من وجود علامات التنصيص في بداية ونهاية هذا النص

```
4. message = '.daed era meht fo owt fi ,terces a peek nac eerhT' reverseCipher.py
```

الآن وعند تشغيل هذا البرنامج سنحصل على النص الأصلي بعد القيام بعملية فك التشفير

```
>>>
Three can keep a secret, if two of them are dead.
>>>
```

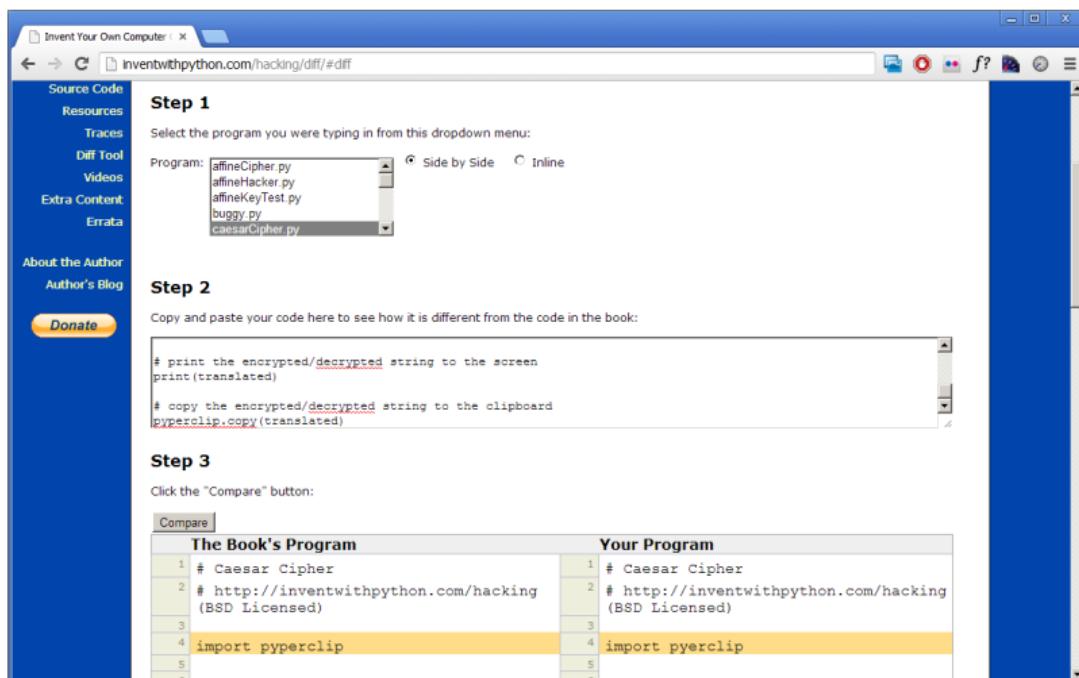
## فحص الكود البرمجي بـأداة Diff بشكل أون لاين:

رغم أنه يمكنك أن تقوم بنسخ ولصق الكود البرمجي أو تحميله من موقع الكتاب على الانترنت ولكن من المفيد أن تقوم بكتابه هذه البرامج بنفسك وهذا سوف يعطيك فكرة أفضل عن الكود في هذه البرامج ولكن في بعض الحالات سوف تقع بعفيض الأخطاء عندما تقوم بكتابه الكود البرمجي بنفسك.

من أجل مقارنة الكود المكتوب في هذا الكتاب مع الكود الذي قمت بكتابته بنفسك يمكنك ذلك من خلال نسخ الكود الذي قمت بكتابته ولصقه في الحقل النصي في المتصفح في الصفحة

<http://invpy.com/hackingdiff>

ثم الضغط على زر Compare عندها سوف تقوم الأداة diff بإظهار أي اختلاف بين الكود الذي قمت أنت بكتابته وال kod الأصلي الموجود في الكتاب وهي طريقة سهلة من أجل ايجاد الأخطاء الكتابية في الكود والتي تسبب ظهور رسائل خطأ



## كيف يعمل هذا البرنامج:

reverseCipher.py

```
1. # Reverse Cipher  
2. # http://inventwithpython.com/hacking (BSD Licensed)
```

أول سطرين في البرنامج هما عبارة عن تعليقات، السطر الأول يشرح المهمة التي يقوم بها البرنامج أما السطر الثاني فهو عنوان الموقع الذي يمكنك إيجاد هذا البرنامج فيه.

الرخصة BSD Licensed تعني أن هذا البرنامج هو مجاني ويمكن نسخه والتعديل عليه من قبل أي شخص طالما أن هذا البرنامج سوف ينسب إلى كاتبه الأصلي (في هذه الحالة النسبة إلى موقع الكتاب (<http://inventwithpython.com/hacking>)

نص رخصة BSD: Berkeley Software Distribution يمكن أن تجده في

<http://invpy.com/bsd>

reverseCipher.py

```
4. message = 'Three can keep a secret, if two of them are dead.'
```

السطر الرابع يقوم بتخزين السلسلة النصية التي نريد تشفيرها في متغير باسم message في كل مرة نريد فك تشفير أو تشفير أي سلسلة نصية جديدة فقط نقوم بكتابه هذه السلسلة بشكل مباشر في السطر الرابع في الكود البرمجي (في هذا البرنامج لم نستخدم التابع input() وبدل ذلك يجب على المستخدم كتابة الرسالة التي يريد تشفيرها أو فك تشفيرها بشكل مباشر في الكود البرمجي)

وفي كل مرة يجب تغيير الكود البرمجي من أجل تشفير أو فك تشفير رسالة جديدة

reverseCipher.py

```
5. translated = ''
```

قمنا بتعريف متغير جديد باسم translated وهو يحوي حالياً على سلسلة نصية فارغة (translated) تذكر أن تعريف السلسلة النصية الفارغة يتم باستخدام زوج من علامات التنصيص المفردة وليس علامات التنصيص المزدوجة

## التابع :len()

```
7. i = len(message) - 1
```

reverseCipher.py

السطر السادس هو سطر فارغ وبايثن سوف يتجاوزه، في السطر التالي من الكود قمنا بإسناد قيمة للمتغير *i*

القيمة المسندة هي العبارة الجبرية 1 – len(message)

أول جزء من العبارة الجبرية هو len(message)

وهو استدعاء التابع len، هذا التابع يقبل القيم النصية كمدخلات تمرر له أثناء عملية الاستدعاء ثم يقوم بإعادة قيمة عدبية (عدد صحيح) تخبرنا بعدد الأحرف الموجودة في السلسلة النصية الممررة لهذا التابع (طول السلسلة النصية).

في البرنامج السابق قمنا بتمرير المتغير message إلى التابع len()

سوف يخبرنا بعدد أحرف السلسلة النصية المخزنة في المتغير message

جرب المثال التالي:

```
>>> len('Hello')
5
>>> len('')
0
>>> spam = 'Al'
>>> len(spam)
2
>>> len('Hello' + ' ' + 'world!')
12
>>>
```

من القيمة التي يعيدها التابع `len()` نعرف أن السلسلة النصية 'Hello' تملك 5 أحرف وأن السلسلة النصية الفارغة تملك 0 حرف وإذا قمت بتخزين السلسلة النصية 'Al' في متغير ومن ثم قمنا باستدعاء التابع `len()` ومررنا سم المتغير لهذا التابع فسوف يقوم التابع بإعادة القيمة 2 إذا مررنا العبارة '!Hello' + 'world!' إلى التابع `len()` فسوف يعيد القيمة 12 لأن المسافة الفارغة وعلامة التعجب يتم اعتبارهم كأحرف السطر السابع في البرنامج يقوم بإيجاد عدد أحرف السلسلة النصية المخزنة في المتغير `message` ويطرح منها العدد واحد ثم يقوم بتخزين العدد الناتج في المتغير `i` العدد المخزن في المتغير `i` يستخدم كفهرسة من أجل الإشارة للحرف الأخير في السلسلة النصية `message`.

## الحلقة :while

```
reverseCipher.py  
8. while i >= 0:
```

الحلقة `while` مكونة من أربعة أجزاء:

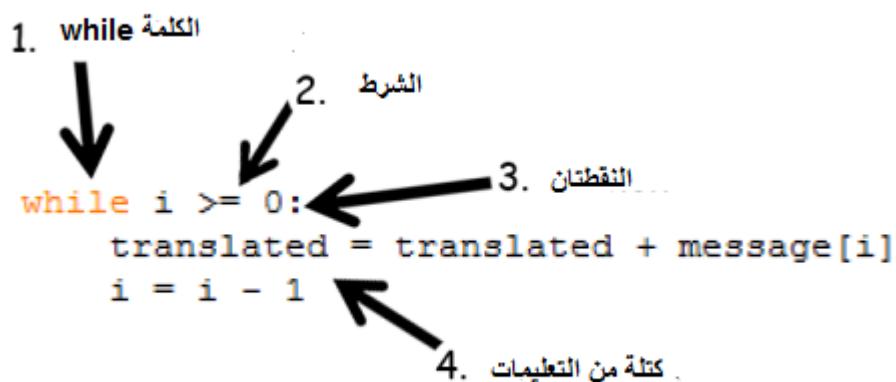
١. الكلمة `while`

٢. عبارة جبرية وتسمى الشرط والتي تحسب قيمة منطقية (قيمة بوليانية)

(سيتم شرحها في الفصل القادم) `True or False`

٣. النقطتان :

٤. كتلة من التعليمات (كما في السطرين 9 and 10)



لتتمكن من فهم الحلقة while يجب أن تفهم المتحوّلات المنطقية (البوليانية) وعبارات المقارنة

### البيانات من النوع المنطقي (البولاني):

هذا النوع من البيانات يحوي فقط على قيمتين صحيحة أو خطأ True or False

وهي حساسة لحالة الأحرف، يجب أن تبدأ دائمًا بحرف كبير T and F وبباقي الأحرف صغيرة

وهي ليست قيم نصية ولا يجب أن تضعها بين علامات التنصيص وهي تستخدم مع معاملات المقارنة (سوف تشرح لاحقاً في هذا الفصل)

مثل أي نوع من البيانات الأخرى فإن القيم البوليانية يمكن تخزينها ضمن المتغيرات

جرب المثال التالي:

```

>>> spam = True
>>> spam
True
>>> spam = False
>>> spam
False
>>>
    
```

## معاملات المقارنة:

في السطر الثامن من البرنامج، انظر إلى العبارة بعد الكلمة while

```
8. while i >= 0:
```

reverseCipher.py

> = ؛ تحيى على قيمتين (القيمة الموجودة في المتغير a والقيمة 0) ومعامل المقارنة = < = الذي يلفظ أكبر أو يساوي.

معاملات المقارنة تظهر بالجدول التالي:

العلمية	الإشارة
أصغر من	<
أكبر من	>
أصغر أو يساوي	<=
أكبر أو يساوي	>=
يساوي	==
لا يساوي	!=

ادخل العبارات التالية في الشيل التفاعلية وشاهد النتيجة المنطقية (البوليانية) التي يتم حسابها

```
>>> 0 < 6
True
>>> 6 < 0
False
>>> 50 < 10.5
False
>>> 10.5 < 11.3
True
>>> 10 < 10
False
>>>
```

العبارة  $6 < 0$  تعيّد القيمة المنطقية (البوليانية) **True** لأن العدد 0 هو أصغر من العدد 6

العبارة  $0 < 6$  تعيّد القيمة المنطقية **False** لأن العدد 6 ليس أصغر من العدد 0

العبارة  $10.5 < 50$  تعيّد القيمة المنطقية **False** لأن العدد 50 ليس أصغر من العدد 10.5

العبارة  $10.5 < 11.3$  تعيّد القيمة **True** لأن العدد 10.5 هو أصغر من العدد 11.3

العبارة  $10 < 10$  تعيّد القيمة **False** لأن العدد 10 ليس أصغر من نفسه

جرب العبارات الشرطية التالية:

```
>>> 10 <= 20
True
>>> 10 <= 10
True
>>> 10 >= 20
False
>>> 20 >= 20
True
>>>
```

```
>>> 10 == 10
True
>>> 10 == 11
False
>>> 10 != 10
False
>>> 10 != 11
True
>>> 'Hello' == 'Hello'
True
>>> 'Hello' == 'Good bye'
False
```

```
>>> 'Hello' == 'HELLO'  
False  
>>> 'Good bye' != 'Hello'  
True  
>>>
```

## انتبه للفرق بين = و ==

إشارة = تستخدم لإسناد قيمة معينة لمتغير أما == تستخدم للسؤال عن تساوي قيمتين

إذا كنت تريد سؤال بایثون عن تساوي قيمتين استخدم ==

إذا كنت تريد من البایثون أن يقوم بإسناد قيمة إلى متغير استخدم =

الأعداد والسلسل النصية هي دائماً قيم غير متساوية، جرب المثال التالي:

```
>>> 42 == 'Hello'  
False  
>>> 42 == '42'  
False  
>>> 10 == 10.0  
True  
>>>
```

لاحظ أن '42' أو أي قيمة رقمية أخرى عندما يتم كتابتها داخل علامات التنصيص تصبح سلسلة نصية.

## الشروط:

الشرط هو العبارة التي تكتب بعد تعليمات while or if

(التعليمية if سوف نتعرف عليها لاحقاً في فصل قادم)

الشروط عادةً تحوي على معاملات المقارنة ولكن الشروط تبقى عبارات جبرية.

## كتلة التعليمات:

كتلة التعليمات هي عبارة عن سطر أو أكثر من الكود البرمجي تكون مجموعة مع بعضها بنفس الفراغ (الهامش) في بداية كل سطر

يمكنك معرفة بداية الكتلة ونهايتها من خلال النظر إلى المسافة في بداية الأسطر الكتلة تبدأ عندما يكون السطر قد ازبح بمقدار أربع مسافات عن الحافة اليسرى وكل الأسطر اللاحقة التي تكون مزاحة بمقدار أربع مسافات هي جزء من الكتلة.

عندما يكون السطر مزاح بمقدار أربع مسافات أخرى (مزاح بمقدار ثمانية عن البداية) فهذا يعني أن كتلة جديدة من التعليمات قد بدأت داخل الكتلة الأولى

1. codecodecodecodecodecodecode	لا يحوي على مسافة بدائية #
2. ....codecodecodecodecodecodecode	يحوي على مسافة بمقدار أربع فراغات #
3. ....codecodecodecodecodecode	يحوي على مسافة بمقدار أربع فراغات #
4. .......	يحوي على مسافة بمقدار ثمانية #
5. ....codecodecodecodecode	فراغات
6.	يحوي على مسافة بمقدار أربع فراغات #
7. ....codecodecodecodecode	يحوي على مسافة بمقدار أربع فراغات #
8. codecodecodecodecodecodecode#	لا يحوي على مسافة بدائية #

- السطر الأول لا يحوي على أي مسافة بدائية
- السطر الثاني يحوي على مسافة بدائية بقدرها أربع فراغات وهذا يعني أن كتلة جديدة قد بدأت.
- السطر الثالث يحوي أيضاً على مسافة بدائية بقدرها أربعة فراغات وهذا يعني أن الكتلة السابقة مستمرة.
- السطر الرابع يحوي على مسافة بدائية بقدرها ثمانية فراغات وهذا يعني أن كتلة ثانية قد بدأت داخل الكتلة الأولى.
- السطر الخامس يحوي على مسافة بدائية بقدرها أربعة فراغات وهذا يعني أن الكتلة الثانية قد انتهت وأن الكتلة الأولى مستمرة.
- السطر السادس هو فارغ وسوف يتتجاهله البايثون.

- السطر السابع يحوي على مسافة بدائية مقدارها أربعة فراغات وهذا يعني أنه استمرار لكتلة التي بدأت في السطر الثاني.
- السطر الثامن لا يحوي على مسافة بدائية وهذا يعني أن الكتلة التي بدأت في السطر الثاني قد انتهت في السطر السابع.

(قيمة المسافة أربعة فراغات ليست قيمة إجبارية ويمكن أن تكون أي قيمة أخرى ولكن جرت العادة أن تكون هذه المسافة عبارة عن أربعة فراغات)

## تعليمات الحلقة: while

```
reverseCipher.py
8. while i >= 0:
9.     translated = translated + message[i]
10.    i = i - 1
11.
12. print(translated)
```

التعليمية `while` في السطر الثامن تخبر البياثون أن يقوم أولاً بفحص الشرط (`i >= 0`) إذا كانت نتيجة هذا الشرط هي `True` (الشرط متحقق) فإن البرنامج سوف يقوم بتنفيذ التعليمات الموجودة داخل الكتلة التي تلي تعليمية `while` (الكتلة هي السطرين 9 and 10)

أما إذا كانت نتيجة تنفيذ الشرط هي `False` (الشرط غير متحقق) فإن البرنامج سوف يتتجاوز الكود الموجود في الكتلة التالية ويقفز لتنفيذ أول سطر من الكود الذي يلي الكتلة (السطر 12)

إذا كان الشرط متحقق فإن البرنامج سوف يبدأ بتنفيذ التعليمات الموجودة في الكتلة وعندما ينتهي من تنفيذ آخر تعليمية في هذه الكتلة فسوف يعود إلى حلقة `while` مرة ثانية ليرى نتيجة الشرط مرة ثانية وإذا بقي الشرط متحقق فإن البرنامج سوف يقوم بتنفيذ التعليمات الموجودة داخل الكتلة مرة ثانية ثم يعود إلى حلقة `while` ويفحص حالة الشرط مرة أخرى فإذا أصبح الشرط غير متحقق عندما فإن البرنامج سوف يتتجاوز أساطير الكتلة ويقفز لتنفيذ أول سطر بعد الكتلة.

العبارة `0 >= i` تعني: طالما أن قيمة المتغير `i` هي أكبر أو تساوي الصفر، استمر في تنفيذ التعليمات الموجودة في الكتلة التالية.

## ازدياد السلسلة:

تذكر أنه في السطر السابع قد تم تعين القيمة الأولية للمتغير `i` وهي طول السلسلة النصية المخزنة في المتغير `message` منقوص منها واحد وأن الحلقة `while` في السطر الثامن تستمرة في تنفيذ التعليمات الموجودة داخل الكتلة التي تليها إلى أن يصبح الشرط `0 <= i < len(message)` غير متحقق `False`

```
reverseCipher.py
7. i = len(message) - 1
8. while i >= 0:
9.     translated = translated + message[i]
10.    i = i - 1
11.
12. print(translated)
```

كتلة التعليمات التي تلي `while` تحتوي على السطرين 9 and 10 السطر 9 يقوم بإسناد قيمة جديدة للمتغير `translated` ، هذه القيمة تزداد في كل مرة بالحرف ذو قيمة الفهرسة `i` من السلسلة النصية المخزنة في المتغير `message` وبهذه الطريقة فإن قيمة السلسلة المخزنة في المتغير `translated` تزداد إلى أن تصبح السلسلة النصية الكاملة الغير مشفرة.

في السطر العاشر يتم اسناد قيمة جديدة للمتغير `i` وهذه القيمة هي القيمة الحالية لهذا المتغير منقوصاً منها واحد.

السطر 12 لا يحوي على مسافة بدائية وهذا يخبر باليثون بأن كتلة التعليمات التي تلي `while` قد انتهت وبدلاً من أن يقوم البرنامج بتنفيذ السطر 12 فهو يعود إلى السطر 8 ليقوم بفحص حالة الشرط مرة ثانية، إذا كان الشرط متحقق فسوف يقوم بتنفيذ التعليمات الموجودة داخل الكتلة مرة ثانية (السطرين 10 and 9) ويستمر بتنفيذهما إلى أن يصبح الشرط غير متحقق `False` (عندما تصبح قيمة `i` أكبر من الصفر) وفي هذه الحالة فإن البرنامج سوف يقوم بتنفيذ أول سطر بعد الكتلة وهو السطر 12

المتغير `i` يبدأ بقيمة الفهرسة لأخر حرف في السلسلة النصية المخزنة في المتغير `message`.

المتغير translated يبدأ بقيمة سلسلة فارغة ثم (في داخل الحلقة) يأخذ هذا المتغير قيمة message[i] (والتي هي الحرف الأخير من السلسلة النصية الموجودة في المتغير message) حيث أن المتغير i يحوي على قيمة الفهرسة الخاصة بآخر حرف في السلسلة.

ثم يتم أننا نقص قيمة المتغير i بمقدار واحد وهذا يعني أن message[i] أصبحت تشير إلى الحرف ما قبل الأخير في السلسلة النصية ويتم إضافته إلى قيمة المتغير translated.

قيمة المتغير i تستمر بالتناقص إلى أن يصبح message[i] هو أول حرف في السلسلة ويتم إضافته كآخر حرف في السلسلة المخزنة في المتغير translated.

وهذا يجعل القيمة المخزنة في المتغير translated هي عكس القيمة الموجودة في المتغير message وعندما تصبح قيمة المتغير  $i = -1$  وعندما يصبح الشرط غير متحقق وسوف يقف البرامج لتنفيذ السطر 12

```
12. print(translated)
```

reverseCipher.py

في نهاية البرنامج (السطر 12) قمنا بطباعة محتوى المتغير translated وهي السلسلة النصية ('.daed era meht fo owt fi ,terces a peek nac eerhT')

إذا لم تفهم كيف يقوم هذا البرنامج بعكس السلسلة النصية حاول أن تقوم بإضافة سطر جديد داخل الحلقة while وهو:

```
print(i, message[i], translated)
```

ليصبح الكود البرمجي كما في الشكل التالي:

```
8. while i >= 0:  
9.     translated = translated + message[i]  
10.    print(i, message[i], translated)  
11.    i = i - 1  
12.  
13. print(translated)
```

reverseCipher.py

التابع (`print()`) سوف يقوم بطباعة ثلاثة قيم:

١. قيمة المتغير `i`

٢. `message[i]`

٣. قيمة المتغير `translated`

وذلك في كل مرة يتم فيها تنفيذ التعليمات داخل الكتلة في الحلقة `while`

هذا سوف يشرح خطوات تكرار الحلقة

الآن يمكنك تشغيل البرنامج و مشاهدة كيف تتغير قيمة المتغير `translated`

```
>>>
48 . .
47 d .d
46 a .da
45 e .dae
44 d .daed
43 .daed
42 e .daed e
41 r .daed er
40 a .daed era
39 .daed era
38 m .daed era m
37 e .daed era me
36 h .daed era meh
35 t .daed era meht
34 .daed era meht
33 f .daed era meht f
32 o .daed era meht fo
31 .daed era meht fo
30 o .daed era meht fo o
29 w .daed era meht fo ow
28 t .daed era meht fo owt
27 .daed era meht fo owt
26 f .daed era meht fo owt f
25 i .daed era meht fo owt fi
24 .daed era meht fo owt fi
23 , .daed era meht fo owt fi ,
22 t .daed era meht fo owt fi ,t
21 e .daed era meht fo owt fi ,te
20 r .daed era meht fo owt fi ,ter
19 c .daed era meht fo owt fi ,terc
18 e .daed era meht fo owt fi ,terce
17 s .daed era meht fo owt fi ,terces
16 .daed era meht fo owt fi ,terces
15 a .daed era meht fo owt fi ,terces a
```

```

14 .daed era meht fo owt fi ,terces a
13 p .daed era meht fo owt fi ,terces a p
12 e .daed era meht fo owt fi ,terces a pe
11 e .daed era meht fo owt fi ,terces a pee
10 k .daed era meht fo owt fi ,terces a peek
9 .daed era meht fo owt fi ,terces a peek
8 n .daed era meht fo owt fi ,terces a peek n
7 a .daed era meht fo owt fi ,terces a peek na
6 c .daed era meht fo owt fi ,terces a peek nac
5 .daed era meht fo owt fi ,terces a peek nac
4 e .daed era meht fo owt fi ,terces a peek nac e
3 e .daed era meht fo owt fi ,terces a peek nac ee
2 r .daed era meht fo owt fi ,terces a peek nac eer
1 h .daed era meht fo owt fi ,terces a peek nac eerh
0 T .daed era meht fo owt fi ,terces a peek nac eerhT
.daed era meht fo owt fi ,terces a peek nac eerhT
>>>

```

السطر الأول الذي يحوي على " . . 48 " يظهر قيمة message[i] and translated

: حيث أن:

i = 48

message[i] = . ( وهي آخر حرف في السلسلة النصية )

translated = .

بعدها يتم إنفاص قيمة المتغير | بقدر واحد ليصبح:

i = 47

message[i] = d

translated = .d

وتستمر هذه العملية إلى أن يصبح

i = 0

message[i] = T

translated = .daed era meht fo owt fi ,terces a peek nac eerhT

## شرح البرنامج خطوة بخطوة:

سوف نقوم بشرح البرنامج خطوة بخطوة بنفس الطريقة التي يقوم بها مترجم البايثون بالتنفيذ.

مترجم البايثون يبدأ بتنفيذ البرنامج من الأعلى من أول سطر ثم ينتقل إلى الأسفل ليقوم بتنفيذ التعليمات التالية.

الأسطر الفارغة والتعليقات سوف يتم تجاهلها.

الجدول التالي يظهر شرح لكل سطر بنفس الطريقة التي يقوم مترجم بايثون بتنفيذ هذا البرنامج:

```
reverseCipher.py
1. # Reverse Cipher
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. message = 'Three can keep a secret, if two of them are dead.'
5. translated = ''
6.
7. i = len(message) - 1
8. while i >= 0:
9.     translated = translated + message[i]
10.    i = i - 1
11.
12. print(translated)
```

الخطوة ١	السطر ١	تعليق وسيتم تجاهله
الخطوة ٢	السطر ٢	تعليق وسيتم تجاهله
الخطوة ٣	السطر ٣	تخزين السلسلة النصية
		'Three can keep a secret, if two of them are

dead.'		
في المتغير message		
تخزين السلسلة النصية الفارغة '' في المتغير translated	الخطوة ٤ السطر ٥	
$i = len(message) - 1 = 48$ يتم تخزين القيمة 48 في المتغير i	الخطوة ٥ السطر ٧	
شرط الحلقة while ( $i \geq 0$ ) محق والبرنامج سوف يقوم بتنفيذ التعليمات داخل كتلة التعليمات التالية	الخطوة ٦ السطر ٨	
يتم حساب [i]translated + message والتي تساوي القيمة النصية '. و يتم تخزين هذه القيمة في المتغير translated	الخطوة ٧ السطر ٩	
يتم حساب $i - 1$ والتي تساوي 47 ويتم تخزين هذه القيمة في المتغير i	الخطوة ٨ السطر ١٠	
عندما ينتهي البرنامج من تنفيذ آخر تعليمية في الكتلة يعود إلى بداية الحلقة ليقوم بإعادة فحص حالة الشرط مرة ثانية والشرط محق للمرة الثانية والبرنامج سوف يقوم بتنفيذ التعليمات داخل الكتلة للمرة الثانية	الخطوة ٩ السطر ٨	
يتم حساب [i]translated + message والتي تساوي 'd.'. ويتم تخزين هذه السلسلة النصية في المتغير translated	الخطوة ١٠ السطر ٩	
يتم حساب $i - 1$ والتي تساوي 46 وسيتم حفظ هذه القيمة في المتغير i	الخطوة ١١ السطر ١٠	
يتم العودة لفحص حالة الشرط وهو مازال محق وينتقل البرنامج لتنفيذ التعليمات داخل الكتلة مرة أخرى	الخطوة ١٢ السطر ٨	
يستمر تنفيذ التعليمات داخل الكتلة إلى أن تصبح قيمة المتغير i = 0 وقيمة المتغير translated = '.daed era meht fo owt fi ,terces	الخطوة ١٣ إلى الخطوة ١٤٩	

a peek nac eerh'		
يتم إعادة فحص الشرط وهو محقق $0 >= 0$	السطر ٨	الخطوة ١٥٠
يتم حساب $[i] = \text{translated} + \text{message}$ والتي تساوي	السطر ٩	الخطوة ١٥١
'daed era meht foowt fi ,terces a peek nac eerhT'		
ويتاك حفظ هذه السلسلة النصية في المتغير <code>translated</code>		
يتم حساب $i = 1$ والتي تساوي ١ - ويتم تخزين هذه القيمة في المتغير <code>i</code>	السطر ١٠	الخطوة ١٥٢
الشرط $0 <= 1$ غير متحقق والبرنامج سوف يتجاوز التعليمات داخل الكتلة وسوف ينتقل لتنفيذ السطر 12	السطر ٨	الخطوة ١٥٣
تصبح قيمة المتغير <code>translated</code> هي السلسلة النصية .daed era meht fo owt fi , terces a peek nac eerhT'	السطر ١٢	الخطوة ١٥٤
ويتم تمرير استدعاء التابع <code>print()</code> وتمرير هذه القيمة له من أجل أن يقوم بطباعتها على الشاشة		
لا يوجد أي سطر بعد السطر 12 لذلك يتوقف البرنامج		

## استخدم التابع `input()` في برمجنا:

البرامج في هذا الكتاب مصممة لتقوم بتشифر أو فك تشفير الرسالة أو السلسلة النصية التي يتم كتابتها بشكل مباشر في الكود البرمجي، يمكن تعديل هذه الخاصية من خلال استدعاء التابع `input()` والذي يسمح للمستخدم بإدخال السلسلة النصية التي يرغب بفك تشفيرها، هذا التابع يظهر المؤشر للمستخدم ويتوقف عن تنفيذ البرنامج إلى أن يقوم المستخدم بإدخال القيمة النصية والضغط على انتر.

يمكنك تغيير السطر الرابع في البرنامج ليصبح كالتالي:

```
reverseCipher.py
```

```
4. message = input('Enter message: ')
```

الآن وبعد تنفيذ البرنامج سوف يظهر المؤشر `prompt` على الشاشة وينتظر إلى أن يقوم المستخدم بإدخال الرسالة والضغط على انتر الرسالة التي يقوم المستخدم بإدخالها يجب أن تكون سلسلة نصية وسيتم حفظها في المتغير `message`

```
Enter message: Hello world!
!dlrow olleH
>>>
```

## الخلاصة:

الآن وبعد أن تعلمنا كيفية التعامل مع النصوص يمكننا البدء بكتابة البرامج التي يمكن أن تتفاعل مع المستخدم.

السلسلات النصية هي نوع من البيانات والذي سوف نستخدمه في برمجنا ويمكننا استخدام المعامل '`+`' من أجل جمع السلسلات النصية مع بعضها البعض، كما يمكننا استخدام الفهرسة والاقتطاع من أجل خلق سلسلة جديدة كجزء من سلسلة نصية أخرى.

**التابع (len):** يتم تمرير سلسلة نصية له ويقوم بإعادة رقم صحيح يدل على عدد الأحرف في هذه السلسة النصية.

نوع البيانات المنطقي (البوليفاني) له فقط قيمتين: True and False . عاملات المقارنة = < , > , == , != تستخدم لمقارنة قيمتين وتظهر الناتج على شكل قيمة منطقية (بوليفانية).

الشروط هي عبارات جبرية تستخدم في عدة أنواع مختلفة من التعليمات. الحلقة while تستمرة بتنفيذ التعليمات داخل الكتلة التي تليها طالما أن الشرط محقق True . الكتلة هي مجموعة من الأسطر التي لها نفس الإزاحة (المسافة البدائية) ويمكن أن تحوي في داخلها على كتل جديدة.

الأجزاء المتبعة في البرنامج هو البداية بمتغير يحوي على سلسلة نصية فارغة ومن ثم زيادة هذه السلسلة حرف بحرف إلى أن نحصل على السلسلة المطلوبة.



## الفصل السادس

# شیفرة قیصر

---

محتوى هذا الفصل:

- التعليمة import
- الثوابت constants
- الطريقة upper() :method
- حلقة for
- تعليمات if, elif and else
- المعاملات in and not in
- الطريقة fine()

“BIG BROTHER IS WATCHING YOU.”

“1984” by George Orwell

## تجهيز البرنامج:

في الفصل الأول تعلمنا عدة طرق من أجل القيام بتشифر قيصر، الآن سوف نستخدم برنامج من أجل القيام بهذه المهمة.

الشيفرة العكسية دائماً تقوم بالتشифر بنفس الطريقة ولكن تشيفر قيصر يستخدم المفاتيح keys حيث يتم تشيفر الرسالة بطرق مختلفة بالإضافة على قيمة المفتاح المستخدم.

المفتاح في شيفرة قيصر هو عدد صحيح ضمن المجال 0 to 25.

إذا عرف محل الشيفرة cryptanalyst أن التشيف المستخدم هو تشيفر قيصر فهذا غير كافي ويجب أن يعرف قيمة المفتاح المستخدم في عملية التشيف من أجل كسر أو فك هذه الشيفرة.

## ال kod البرمجي لشيفرة قيصر:

قم بكتابة الكود التالي في محرر النصوص واحفظه باسم caesarCipher.py واضغط F5 من أجل تنفيذ البرنامج

يجب أن تقوم أولاً بتحميل الوحدة pyperclip.py module ووضع هذا الملف في نفس المجلد الموجود فيه الملف caesarCipher.py

يمكنك تحميل هذا الموديول من <http://invpy.com/pypreclip.py> (عملية التحميل هي عبارة عن نسخ الكود الموجود في هذه صفحة الويب السابقة ولصقه وحفظه في محرر (pyperclip.py باسم النصوص

## Source code for caesarCipher.py

```
1. # Caesar Cipher
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. import pyperclip
5.
6. # the string to be encrypted/decrypted
7. message = 'This is my secret message.'
8.
9. # the encryption/decryption key
10. key = 13
11.
12. # tells the program to encrypt or decrypt
13. mode = 'encrypt' # set to 'encrypt' or 'decrypt'
14.
15. # every possible symbol that can be encrypted
16. LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
17.
18. # stores the encrypted/decrypted form of the message
19. translated = ''
20.
21. # capitalize the string in message
22. message = message.upper()
23.
24. # run the encryption/decryption code on each symbol in the message string
25. for symbol in message:
26.     if symbol in LETTERS:
27.         # get the encrypted (or decrypted) number for this symbol
28.         num = LETTERS.find(symbol) # get the number of the symbol
29.         if mode == 'encrypt':
30.             num = num + key
31.         elif mode == 'decrypt':
32.             num = num - key
33.
34.         # handle the wrap-around if num is larger than the length of
35.         # LETTERS or less than 0
36.         if num >= len(LETTERS):
37.             num = num - len(LETTERS)
38.         elif num < 0:
39.             num = num + len(LETTERS)
40.
41.         # add encrypted/decrypted number's symbol at the end of translated
42.         translated = translated + LETTERS[num]
43.
44.     else:
45.         # just add the symbol without encrypting/decrypting
46.         translated = translated + symbol
47.
48. # print the encrypted/decrypted string to the screen
49. print(translated)
50.
51. # copy the encrypted/decrypted string to the clipboard
52. pyperclip.copy(translated)
```

عند تشغيل هذا البرنامج سوف تظهر النتيجة التالية:

```
>>>  
GUVF VF ZL FRPERG ZRFFNTR.  
>>> .
```

تم تشفير السلسلة النصية 'This is my secret message.' بشيفرة قيصر باستخدام المفتاح

13

هذا البرنامج عندما يعمل يقوم بشكل أوتوماتيكي بنسخ النص المشفر ويمكنك لصقه في ملف نصي أو في رسالة ايميل.

بهذه الطريقة يمكنك بسهولة أخذ النص المشفر إلى خارج البرنامج وإرساله إلى شخص آخر من أجل فك التشفير قم بلصق النص المشفر في السطر 7 ليتم اسنادها إلى المتغير message ثم قم بتغيير السطر 13 ليصبح mode = 'decrypt'

```
caesarCipher.py  
6. # the string to be encrypted/decrypted  
7. message = 'GUVF VF ZL FRPERG ZRFFNTR.'  
8.  
9. # the encryption/decryption key  
10. key = 13  
11.  
12. # tells the program to encrypt or decrypt  
13. mode = 'decrypt' # set to 'encrypt' or 'decrypt'
```

الآن وعند تنفيذ هذا البرنامج سوف نحصل على النتيجة التالية:

```
>>>  
THIS IS MY SECRET MESSAGE.  
>>> .
```

إذا حصلت على رسالة الخطأ التالية:

```
Traceback (most recent call last):  
  File "C:/Python34/caesarCipherde.py", line 4, in <module>  
    import pyperclip  
ImportError: No module named 'pyperclip'  
>>> .
```

هذا يعني أنك لم تقم بوضع الملف `pyperclip.py` في المجلد الصحيح (نفس المجلد الذي قمت بحفظ البرنامج بداخله)

إذا استمر هذا الخطأ ولم تتمكن من إصلاحه فقط قم بحذف السطرين 4 and 52 من الكود  
هذا سوف يخلص الكود من التعليمات التي تعتمد على `pyperclip module`  
الغاية من `pyperclip module` هو نسخ النص المشفر بشكل أوتوماتيكي بعد تنفيذ البرنامج.

## كيف يعمل هذا البرنامج:

سنقوم بشرح كل سطر من كود هذا البرنامج

## استيراد الوحدات باستخدام التعليمة `:import`

```
caesarCipher.py
1. # Caesar Cipher
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. import pyperclip
```

في السطر الرابع استخدمنا التعليمة `import` في البایتون يحوي على العديد من التوابع المبنية في داخله، بعض التوابع موجودة في برامج مفصلة تسمى الموديول (الوحدة) `Module` وهي عبارة عن برنامج بایthon يحوي على تابع إضافية والتي يمكنك أن تستخدمها في برامجك.

في هذا البرنامج قمنا باستيراد الوحدة (الموديول) `pyperclip` من أجل أن نستطيع استدعاء التابع `pyperclip.copy()`

تعليق الاستيراد `import` مكونة من الكلمة `import` وتليها اسم الوحدة المراد استيرادها.  
في السطر الرابع قمنا باستيراد `pyperclip module` والذي يحوي على عدد من التوابع المتعلقة بنسخ ولصق النصوص من وإلى الذاكرة

caesarCipher.py

```
6. # the string to be encrypted/decrypted
7. message = 'This is my secret message.'
8.
9. # the encryption/decryption key
10. key = 13
11.
12. # tells the program to encrypt or decrypt
13. mode = 'encrypt' # set to 'encrypt' or 'decrypt'
```

في السطور السابقة قمنا بإعداد ثلاثة متغيرات:  
المتغير `message` يحوي على السلسلة النصية التي سيتم تشفيرها أو فك تشفيرها.  
المتغير `key` يحوي على عدد صحيح وهو مفتاح التشفير `encryption key`  
المتغير `mode` يحوي على إحدى القيمتين '`encrypt`' (والتي تخبر البرنامج بالقيام بتشفير  
الرسالة) أو '`decrypt`' (والتي تخبر البرنامج بالقيام بفك تشفير الرسالة)

## الثوابت:

caesarCipher.py

```
15. # every possible symbol that can be encrypted
16. LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

نحن بحاجة أيضاً إلى سلسلة نصية تحوي على كل الأحرف الأبجدية الكبيرة وبشكل مرتب.  
قمنا بإسناد السلسلة النصية التي تحوي على كل الأحرف الأبجدية الكبيرة إلى المتغير  
`LETTERS` وهذه السلسلة تحوي على كل الأحرف التي يمكن أن يتم تشفيرها  
في برنامجاً هذه المجموعة من الأحرف تسمى مجموعة الرموز `symbol set`  
في نهاية هذا الفصل سوف تتعلم كيف تقوم بتوسيع هذه المجموعة لتحوي على رموز بالإضافة  
للحراف.

## الطريقة : upper()

```
caesarCipher.py
18. # stores the encrypted/decrypted form of the message
19. translated = ''
20.
21. # capitalize the string in message
22. message = message.upper()
```

في السطر 19 يم حفظ سلسلة نصية فارغة في المتغير `translated` بشكل مشابه لبرنامج الشيفرة العكسية، وعند نهاية تنفيذ البرنامج فإن المتغير `translated` سوف يحوي على كامل السلسلة النصية المشفرة أو الغير مشفرة ولكن الآن يبدأ بسلسلة نصية فارغة.

في السطر 22 قمنا بإسناد قيمة للمتغير `message` ، هذه القيمة هي ناتج تطبيق الطريقة `upper()` على القيمة المحفوظة في المتغير `message`.

الطريقة `method` هي مثل التابع `function` (باستثناء أنها تلحق بقيم غير محتواه في الوحدات `non-module value` كما في السطر 22 حيث أن المتغير يحوي على قيمة) اسم الطريقة هو `upper()` ويتم استدعائها وتطبيقاتها على القيمة النصية المحفوظة في المتغير `message`

التابع `module` ليس طريقة `method` لأنه موجود في الموديول `pyperclip` ولكن `pyperclip.copy()` هي وحدة `module` والتي تم استيرادها في السطر الرابع، لذلك فإن `(copy)` هو ليس طريقة `pyperclip module` وهو تابع لأنه موجود داخل `module` إذا كان هذا الأمر مربك لك فيمكنك دائمًا تسمية الطرق والتوابع باسم تابع.

معظم أنواع البيانات (مثل السلسل النصية) لها طرق `methods`.

السلسل النصية تملك طرق `upper()` and `lower()` والتي تقوم بتحويل أحرف السلسلة النصية إلى أحرف صغيرة أو كبيرة.

جرب المثال التالي:

```
>>> 'Hello world!'.upper()
'HELLO WORLD!'
>>> 'Hello world!'.lower()
'hello world!'
>>>
```

```
>>> fizz = 'Hello world!'
>>> fizz.upper()
'HELLO WORLD!'
>>> fizz
'Hello world!'
>>>
```

## حلقة :for

```
caesarCipher.py
24. # run the encryption/decryption code on each symbol in the message string
25. for symbol in message:
```

حلقة `for` تستخدم من أجل تكرار أو الدوران عبر قائمة أو سلسلة نصية وبشكل مختلف عن حلقة `while` التي تحوي على شرط فإن حلقة `for` تحوي على 6 أجزاء وهي:

١. الكلمة `for`
٢. اسم المتغير
٣. الكلمة `in`
٤. القيمة النصية أو المتغير الذي يحوي على قيمة نصية
٥. النقطتان :
٦. كتلة من التعليمات



في كل مرة يتم فيها تنفيذ البرنامج فإنه سوف يقوم بتكرار تنفيذ كتلة التعليمات داخل الحلقة.

جرب المثال التالي في الشيل التفاعليه

```
>>> for letter in 'Syria':
       print(letter)

S
Y
r
i
a
>>>
```

## حلقة while المكافأة لحلقة for

حلقة while تشبه حلقة for

يمكنك كتابة حلقة while تقوم بنفس العمل الذي تقوم به الحلقة for من خلال إضافة بعض التعليمات كما في المثال التالي:

```

>>> i = 0
>>> while i < len('Syria'):
    letter = 'Syria'[i]
    print('The letter is ' + letter)
    i = i + 1

The letter is S
The letter is y
The letter is r
The letter is i
The letter is a
>>>

```

لاحظ أن حلقة while السابقة تقوم بنفس العمل الذي تقوم به حلقة for ولكن حلقة for هي أبسط وأسهل قبل أن تستطيع فهم الأسطر 26 to 32 من برنامج شيفرة قيصر يجب أن تتعلم أولًا التعليمات find() والمعاملات if, elif and else

## التعليمية :if:

التعليمية if تقرأ (إذا كان الشرط محقق True قم بتنفيذ كتلة التعليمات التالية وإذا لم يكن الشرط متحقق False تجاوز كتلة التعليمات) افتح محرر النصوص واكتب البرنامج التالي ثم قم بحفظه باسم password.py واضغط F5 من أجل تنفيذه

### Source code for password.py

```

1. print('What is the password?')
2. password = input()
3. if password == 'rosebud':
4.     print('Access granted.')
5. if password != 'rosebud':
6.     print('Access denied.')
7. print('Done.')

```

عندما يتم تنفيذ السطر الذي يحوي على password = input() فإن البرنامج يتوقف وينتظر المستخدم ليقوم بعملية الإدخال، المستخدم يستطيع أن يدخل أي شيء يردده والقيمة المدخلة سوف يتم حفظها في المتغير password

إذا كانت القيمة المدخلة هي 'password == 'rosebud' فإن نتيجة العبارة ستكون True (الشرط متحقق) وعندها سيقوم البرنامج بتنفيذ الكتلة وطباعة العبارة 'Access granted.'

أما إذا كانت نتيجة العبارة 'password == 'rosebud' هي False (الشرط غير متحقق) فإن البرنامج لن ينفذ كتلة التعليمات التالية وينتقل إلى الشرط التالي 'password != 'rosebud' والذي ستكون نتيجته True (متحقق) وعندها سوف يقوم البرنامج بتنفيذ كتلة التعليمات التالية ويقوم بطباعة 'Access denied' أما إذا كانت نتيجة هذا الشرط هي False (غير متحقق) فإن البرنامج سوف يتجاوز كتلة التعليمات.

## التعليمية :else

عادةً نريد فحص حالة الشرط وتنفيذ كتلة من التعليمات إذا كان الشرط متحقق وكتلة أخرى من التعليمات إذا كان الشرط غير متحقق.

التعليمية else يمكن أن تستخدم بعد كتلة التعليمات الخاصة بـ if وسوف يتم تنفيذ كتلة التعليمات الخاصة بـ else في حال كان الشرط الخاص بـ if غير متحقق.  
(إذا كان الشرط متحقق نفذ هذه الكتلة أو إذا لم يكن متحقق نفذ الكتلة التالية)  
اكتب الكود التالي وقم حفظه باسم password2.py ولاحظ أنه يقوم بنفس العمل الذي يقوم به البرنامج الأول password.py ولكن في هذا البرنامج قمنا باستخدام if and else

### Source code for password2.py

```
1. print('What is the password?')
2. password = input()
3. if password == 'rosebud':
4.     print('Access granted.')
5. else:
6.     print('Access denied.')
7. print('Done.')
```

## التعليمية :elif

التعليمية elif هي اختصار ل if  
(إذا كان الشرط متحقق نفذ كتلة التعليمات وإذا لم يكن متحقق افحص حالة الشرط الثاني إذا كان متحقق وإلا قم بتنفيذ آخر كتلة تعليمات)

اكتب البرنامج التالي في محرر النصوص وقم بحفظه باسم num.py

```
num = 12
if num < 10:
    print('الرقم مكون من خانة واحدة')
elif 9 < num < 100:
    print('الرقم مكون من خانتين')
elif 99 < num < 1000 :

    print('الرقم مكون من ثلاثة خانات')
else:
    print('الرقم مكون من أكثر من ثلاثة خانات')
```

ولأن الرقم هو 12 فسوف تكون نتيجة تنفيذ البرنامج هي:

```
>>>
الرقم مكون من خانتين
>>>
```

لاحظ أنه يتم تنفيذ كتلة واحدة فقط.

## :in and not in المعاملات

العبارة التي تحوي على سلسلتين نصيتين مرتبطتين بالمعامل in سوف تعيد True إذا كانت السلسلة الأولى موجودة في السلسلة الثانية وإلا فسوف تعيد False.

انتبه أن المعاملات in , not in هي حساسة لحالة الأحرف.

جرب المثال التالي:

```
>>> 'hello' in 'hello world!'
True
>>> 'ello' in 'hello world!'
True
>>> 'HELLO' in 'hello world!'
False
>>> 'HELLO' in 'HELLO world!'
True
>>> '' in 'Hello'
True
>>> '' in ''
True
>>> 'D' in 'ABCDEF'
True
>>>
```

المعامل not in هو عكس المعامل in

جرب المثال التالي:

```
>>> 'hello' not in 'hello world!'
False
>>> 'ello' not in 'hello world!'
False
>>> 'HELLO' not in 'hello world!'
True
>>> 'Hello' not in 'HELLO world!'
True
>>> '' not in 'Hello'
False
>>> '' not in ''
False
>>> 'D' not in 'ABCDEF'
False
>>>
```

استخدام العبارات التي تحوي على المعاملات in , not in هي أمر مفید جداً في الشرط الذي

يلی تعليمة if

انتبه إلى أن كلمة in المستخدمة في تعليمة for (من أجل تحديد مجال الحلقة) ليست نفس

المعامل in المستخدم هنا

## الطريقة :`find()`

`string method` `find()` هي طريقة خاصة بالسلسل النصية

حيث يتم تمرير حرف لهذه الطريقة وتقوم هي بإعادة عدد صحيح هو دليل الفهرسة الذي يشير إلى مكان تواجد هذا الحرف داخل السلسلة النصية.

جرب المثال التالي:

```
>>> 'hello'.find('o')
4
>>> 'hello'.find('h')
0
>>> 'hello'.find('e')
1
>>> fizz = 'hello'
>>> fizz.find('h')
0
>>>
```

إذا تم تمرير حرف غير موجود في السلسلة النصية تقوم هذه الطريقة بإعادة القيمة -1

الطريقة `find()` هي حساسة لحالة الأحرف.

جرب المثال التالي:

```
>>> 'hello'.find('x')
-1
>>> 'hello'.find('H')
-1
>>>
```

القيمة التي يتم تمريرها للطريقة `find()` يمكن أن تكون أكثر من حرف وسوف تعيد هذه الطريقة قيمة الفهرسة التي تدل على مكان أول حرف في السلسلة، جرب المثال التالي:

```

>>> 'hello'.find('x')
-1
>>> 'hello'.find('H')
-1
>>>
>>> 'hello'.find('ello')
1
>>> 'hello'.find('lo')
3
>>> 'hello hello'.find('e')
1
>>>

```

## العودة إلى كود البرنامج:

الآن وبعد أن فهمت التعليمات if, elif, else والمعاملات in, not in والطريقة find()

أصبح من السهل عليك فهم كيف يعمل برنامج شифرة قيصر

```

caesarCipher.py
26.     if symbol in LETTERS:
27.         # get the encrypted (or decrypted) number for this symbol
28.         num = LETTERS.find(symbol) # get the number of the symbol

```

إذا كان الحرف الموجود في المتغير symbol (المتغير الذي تستخدمه حلقة for) موجودة في السلسلة النصية المحفوظة في المتغير LETTERS (أي إذا كان هذا الحرف هو حرف كبير) فإن نتيجة العبارة symbol in LETTERS ستكون True (الشرط متحقق)

تذكر أننا في السطر 22 قمنا بتحويل أحرف الرسالة إلى أحرف كبيرة باستخدام التعليمية

```
message = message.upper()
```

لذلك فإن القيمة symbol لا يمكن أن تكون حرف صغير

الحالة الوحيدة التي يكون فيها هذا الشرط غير متحقق False هي عندما تكون قيمة symbol هي علامة ترقيم أو حرف مثل '?' or '4'

نريد أن نفحص قيمة symbol إذا كانت حرف كبير لأن برمجنا سوف يقوم بتشифر أو فك تشفير الأحرف الكبيرة فقط، وأي حرف أو رموز أخرى سوف تضاف إلى القيمة النصية المحفوظة في المتغير translator بدون تشفير أو فك تشفير.

كتلة تعليمات جديدة تبدأ بعد تعليمية if في السطر 26 وإذا نظرت إلى باقي البرنامج سوف تلاحظ أن كتلة التعليمية هذه تستمر إلى السطر 42 حيث توجد التعليمية else والتي هي جزء من التعليمية if الموجودة في السطر 26

```
29.     if mode == 'encrypt':  
30.         num = num + key  
31.     elif mode == 'decrypt':  
32.         num = num - key
```

caesarCipher.py

الآن وبعد أن عرفنا رقم الحرف الموجود في المتغير num يمكننا تشفيره أو فك تشفيره من خلال إضافة أو طرح قيمة مفتاح التشفير من رقم هذا الحرف.

في شيفرة قيصر يتم إضافة قيمة مفتاح التشفير إلى رقم الحرف من أجل التشفير ويتم إنقاص قيمة مفتاح التشفير من رقم الحرف من أجل عملية فك التشفير.

المتغير mode يحوي على سلسلة نصية تخبر البرنامج أن يقوم بعملية التشفير أو فك التشفير فإذا كانت قيمة هذا المتغير هي 'encrypt' فإن الشرط في السطر 29 سيكون محق وسوف يقوم البرنامج بتنفيذ السطر 30 (وسوف يتجاوز كتلة التعليمات الخاصة بالتعليمية elif)

أما إذا كانت قيمة هذا المتغير هي أي قيمة أخرى غير 'encrypt' فسوف يكون الشرط في السطر 29 غير محق False وعندما سوف يقوم البرنامج بتجاوز السطر 30 ويقوم بفحص حالة السطر الخاص بتعليمه elif في السطر 31

في هذه الطريقة يعرف البرنامج فيما إذا كان يجب عليه أن يقوم بعملية التشفير (إضافة قيمة مفتاح التشفير) أو أن يقوم بعملية فك التشفير (طرح قيمة مفتاح التشفير)

```

34.     # handle the wrap-around if num is larger than the length of
35.     # LETTERS or less than 0
36.     if num >= len(LETTERS):
37.         num = num - len(LETTERS)
38.     elif num < 0:
39.         num = num + len(LETTERS)

```

تذكر عندما قمنا بتنفيذ تشفير قيصر باستخدام الورقة والقلم في الحالات التي كانت نتيجة جمع قيمة المفتاح مع العدد الخاص بالحرف المراد تشفيره وكانت النتيجة أكبر من 26 أو في حالة فك التشفير عند القيام بعملية الطرح وكانت النتيجة أصغر من 0 ، في هذه الحالات كنا نقوم بجمع أو طرح القيمة 26 إلى أو من النتيجة من أجل الحصول على القيمة الصحيحة وهذه العملية تتم في الأسطر من 39 to 36 من البرنامج.

إذا كانت قيمة num أكبر أو تساوي 26 فإن الشرط في السطر 36 سيكون محققا True وسوف يتم تنفيذ السطر 37 (ويتم تجاوز التعليمية elif في السطر 38)

ومن ناحية أخرى فإن بايثون يقوم بفحص إذا كانت قيمة num أصغر من 0 فإذا كان هذا الشرط محققا فسوف يتم تنفيذ السطر 39

في شيفرة قيصر يتم إضافة أو طرح القيمة 26 لأن عدد الأحرف في اللغة الإنجليزية هو 25 حرفا.

لاحظ أنه بدلاً من استخدام القيمة 26 بشكل مباشر قمنا باستدعاء التابع len(LETTERS) والذي سوف يقوم بإعادة القيمة 26

يمكننا تعديل القيمة المحفوظة في المتغير LETTERS من أجل تشفير أو فك تشفير أحرف غير كبيرة وهذا ما سيتم شرحه في نهاية هذا الفصل

```

41.     # add encrypted/decrypted number's symbol at the end of translated
42.     translated = translated + LETTERS[num]

```

الآن وبعد أن تم تعديل قيمة العدد الموجود في المتغير `num` وأصبح قيمة الفهرسة أو دليل الحرف المراد تشفيره أو فك تشفيره، نريد إضافة الأحرف المشفرة أو غير مشفرة إلى نهاية السلسلة النصية المحفوظة في المتغير `translated`

```
caesarCipher.py  
44.     else:  
45.         # just add the symbol without encrypting/decrypting  
46.         translated = translated + symbol
```

في السطر 44 التعليمة `else` تبعد مسافة بدائية مقدارها أربعة فراغات وهذا يدل على أنها استمرار للتعليقة `if` الموجودة في السطر 26

إذا كانت الشرط الخاص بتعليقه `if` في السطر 26 غير محقق فإن كتلة التعليمات السابقة سوف يتم تجاوزها والانتقال إلى تنفيذ التعليمة الخاصة ب `else` في السطر 46

(السطر 45 يتم تجاوزه لأنه عبارة عن تعليق)

كتلة التعليمات الخاصة ب `else` هي عبارة عن سطر واحد حيث يتم إضافة القيمة المحفوظة في المتغير `symbol` إلى القيمة الموجودة في المتغير `translated`

الرموز أو القيم الغير حرفية مثل الفراغ '' أو النقطة '.' يتم إضافتها إلى `translated` بدون أي تشفير أو فك تشفير.

## عرض ونسخ النص المشفر أو الغير مشفر:

```
caesarCipher.py  
48. # print the encrypted/decrypted string to the screen  
49. print(translated)  
50.  
51. # copy the encrypted/decrypted string to the clipboard  
52. pyperclip.copy(translated)
```

السطر 49 لا يحوي على مسافة بدائية وهذا يعني أن كتلة التعليمات السابقة قد انتهت في السطر 49 تم استدعاء التابع `print()` من أجل أن يقوم بطباعة قيمة المتغير `translated`

في السطر 52 تم استدعاء التابع الموجود في الوحدة `pyperclip module` والمسمي `(copy())` وحيث يتم تمرير قيمه له ليتم نسخها إلى الذاكرة لنتمكن من لصقها في مستند نصي أو في أي مكان آخر.

إذا قمت باستدعاء التابع بالشكل التالي `(copy(translated))` بدلاً من الشكل `(copy)` فسوف تظهر رسالة خطأ `pyperclip.copy(translated)`

```
>>> copy('Hello')
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in <module>
    copy('Hello')
NameError: name 'copy' is not defined
>>>
```

وفي حال نسيت استدعاء الوحدة `import pyperclip` فلن تتمكن من استدعاء التابع `(pyperclip.copy())` وسوف تحصل على رسالة خطأ.

هذا هو برنامج شيفرة قيصر، عندما تريد تشغيله لاحظ أن جهاز الكمبيوتر يمكنه تنفيذ كامل البرنامج وشفير السلسلة النصية بأجزاء من الثانية، حتى ولو كتبت رسالة طويلة جداً داخل المتغير `message` فإن جهاز الكمبيوتر قادر على تشفيرها أو فك تشفيرها بأقل من ثانية.

الآن أصبح بإمكانك تشفير أو فك تشفير أي رسالة بشيفرة قيصر ونسخها ولصقها في أي مكان أو أي ملف أو ارسالها عبر الايميل.

## تشفير القيم الغير حرفية:

برنامجه شيفرة قيصر الذي قمنا بكتابته لا يستطيع تشفير القيم الغير حرفية.

إذا كنت تريدين تشفير السلسلة النصية التالية 'The password is 31337' باستخدام مفتاح التشفير 20 فسوف تتم عملية التشفير وسوف تحصل على السلسلة المشفرة التالية

'Dro zkccgybn sc 31337' هذه الرسالة المشفرة لا تحافظ على سرية الرسالة ويمكنا تعديل البرنامج السابق من أجل تشفير القيم الغير حرفية.

إذا قمت بتعديل السلسلة النصية المحفوظة في المتغير LETTERS لتحوي على قيم غير الأحرف الكبيرة وذلك لأن السطر 26 يحوي على الشرط symbol in LETTERS والذي يجب أن يكون محققاً.

بعد تعديل الكود سوف تعرف السبب من استخدام len(LETTERS) بدلاً من كتابة 26 بشكل مباشر.

تعديل الكود يتم من خلال تعديل السطر 16 ليصبح يحوي على رموز وأرقام وجعل السطر 22 عبارة عن تعليق عن طريق إضافة الرمز # في بداية هذا السطر.

يمكنك الحصول على الكود الجديد للبرنامج من <http://invpy.com/caeserCipher2.py>

لقد قمت أنا بالتعديل على البرنامج لتتمكن من ادخال النص وقيمة المفتاح ونوع العملية اثناء تنفيذ البرنامج وذلك من خلال تعديل الأسطر التالية لتصبح كالتالي:

7. message = input('input text : ' , )

10. key = input('input key 0 - 26 : ' , )

30. num = num + int(key)

32. num = num - int(key)

ليصبح الكود البرمجي بشكل كامل بعد التعديل كالتالي:

```

# شفرة قيصر

import pyperclip
message = input('input text: ')
key = input('input key 0 - 26 :')
mode = input('choose encrypt or decrypt: ')
LETTERS = '!#$%&\'()*+,-./0123456789:@ABCDEFGHIJKLMNOPQRSTUVWXYZ'
translated = ''
for symbol in message:
    if symbol in LETTERS:
        num = LETTERS.find(symbol)
        if mode == 'encrypt':
            num = num + int(key)
        elif mode == 'decrypt':
            num = num - int(key)
        if num >= len(LETTERS):
            num = num - len(LETTERS)
        elif num < 0:
            num = num + len(LETTERS)
        translated = translated + LETTERS[num]
    else:
        translated = translated + symbol
print(translated)
pyperclip.copy(translated)

```

هذا الكود الأخير وبعد التعديل غير موجود في موقع الكتاب باللغة الأجنبية وسوف أقوم بإرفاقه مع النسخة العربية

الآن وبعد التعديل يتم تنفيذ البرنامج بالشكل التالي من أجل القيام بعملية التشفير

```

>>>
input text: syria
input key 0 - 26 :2
choose encrypt or decrypt: encrypt
u{tkc

```

من أجل القيام بعملية فك التشفير

```

input text: u{tkc
input key 0 - 26 :2
choose encrypt or decrypt: decrypt
syria
>>>

```

عند التنفيذ يطلب البرنامج من المستخدم إدخال النص وبعد إدخال النص والضغط على انتر يطلب البرنامج من المستخدم إدخال مفتاح تشفير له قيمة من 26 - 2 وبعد إدخال قيمة مفتاح التشفير والضغط على انتر يطلب البرنامج تحديد نوع العملية (تشفيه أو فك تشفير) وبهذا الشكل يمكن استخدام البرنامج للقيام بعمليات تشفير وفك تشفير لنصوص مختلفة وباختيار قيم مختلفة لمفتاح التشفير بشكل سهل وبدون التعديل على الكود البرمجي في كل مرة.

## الخلاصة:

الآن لقد تعلمت مبادئ البرمجة الأساسية وأصبحت قادراً على كتابة برنامج يقوم بالتشفيه وفك التشفير والأهم أنك أصبحت قادراً على فهم كيف يعمل كود البرنامج.

الوحدة `module` هي برنامج بلغة الباليتون يحوي على توابع مفيدة يمكنك استخدامها.

من أجل استخدام هذه التوابع يجب عليك أو لاً استيراد الوحدة باستخدام التعليمية `import .` استدعاء التابع الموجودة داخل الوحدة يتم من خلال كتابة اسم الوحدة مسبوقاً بنقطة قبل اسم التابع كما في المثال التالي:

الطريقة `method` هي عبارة عن تابع يلحق بقيمة معينة من نوع معين من البيانات.

الطريقتين (`upper()` and `lower()`) هما طريقتين نصيتين `string methods` ويقومان بإعادة أحرف كبيرة أو أحرف صغيرة للسلسلة النصية التي تقوم باستدعائهما.

الطريقة (`find()`) هي أيضاً طريقة نصية `string method` والتي تقوم بإعادة قيمة عدد صحيح يدل على عنوان الفهرسة لمكان وجود الحرف أو القيمة الممررة إليها في السلسلة النصية.

حلقة `for` هي حلقة تقوم بتكرار الأحرف الموجودة داخل السلسلة وتقوم بتنفيذ التعليمات داخل الكتلة التالية من أجل كل دورة.

التعليمات `if, elif, else` تقوم بتنفيذ كتلة من التعليمات بالاعتماد على حالة الشرط إذا كان محق أو غير محق.

المعاملان `in, not in` يستخدمان من أجل معرفة وجود أو عدم وجود حرف أو سلسلة نصية داخل سلسلة نصية أخرى ويعيدان إحدى القيمتين `True or False`.

في الفصل القادم سوف نستخدم هذه المهارات من أجل كتابة برنامج يقوم بفك شифرة قيصر وقراءة النص الصريح لرسائل مشفرة بشيفرة قيصر.



## الفصل السابع

# فك شيفرة قيصر باستخدام

## تقنية القوة الغاشمة

---

محتوى هذا الفصل:

- قول شانون المأثور وقاعدة كيرشوف
- تكنية القوة الغاشمة brute-force
- التابع range()
- تنسيق السلسل النصية

## فأك الشيفرات :Hacking Ciphers

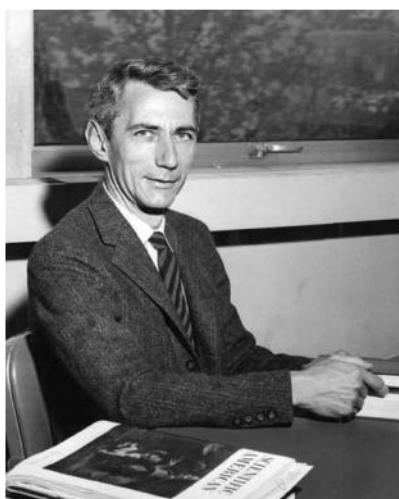
يمكن فأك شيفرة قيصر باستخدام تقنية تحليل التشفير: القوة الغاشمة brute-force لأن البرنامج الذي سنقوم بكتابته قادر على كسر تشفير قيصر فلا تستخدم هذه الشيفرة في تشفير معلوماتك السرية.



Auguste Kerckhoffs January 19,  
1835 - August 9, 1903

"A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."

قاعدة كيرشوف (نسبة لعالم التشفير أجوست كيرشوف) والتي تقول: "نظام التشفير يجب أن يبقى سرياً حتى ولو عرف كل شخص هذا النظام، باستثناء المفتاح الذي يكون معرف من قبل الناس"



Claude Shannon  
April 30, 1916 - February 24, 2001  
"The enemy knows the system."

قول مؤثر لعالم الرياضيات كلويد شانون:  
"العدو يعرف النظام"

## **هجوم القوة الغاشمة:**

لا يوجد أحد يستطيع منع محل الشифرات من تخمين قيمة مفتاح التشفير ، وفك تشفير النص المشفر باستخدام هذا المفتاح وإذا لم يكن هذا المفتاح هو المفتاح الصحيح سوف يحاول استخدام قيمة أخرى ويستمر باستخدام قيم أخرى حتى الوصول إلى القيمة الصحيحة.

التقنية التي يتم من خلالها استخدام كل قيم الممكنة لمفتاح التشفير تسمى هجوم القوة الغاشمة brute-force attack وهذا الهجوم ليس هجوم معقد وهو غير فعال في كل الشيفرات ولكنه فعال ضد شيفرة قيصر.

## **النص البرمجي لبرنامج فك شيفرة قيصر:**

افتح نافذة محرر النصوص من خلال File > New File وقم بكتابة الكود التالي ثم قم بحفظه باسم caesarHacker.py واضغط F5 من أجل تنفيذ هذا البرنامج

يجب أن تقوم بتحميل الوحدة (الموديول) pyperclip.py module ووضعها في نفس المجلد الذي يحوي على الملف caesarHacker.py

يمكنك الحصول على هذه الوحدة من <http://invpy.com/pyperclip.py>

## Source code for caesarHacker.py

```
1. # Caesar Cipher Hacker
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. message = 'GUVF VF ZL FRPERG ZRFFNTR.'
5. LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
6.
7. # loop through every possible key
8. for key in range(len(LETTERS)):
9.
10.    # It is important to set translated to the blank string so that the
11.    # previous iteration's value for translated is cleared.
12.    translated = ''
13.
14.    # The rest of the program is the same as the original Caesar program:
15.
16.    # run the encryption/decryption code on each symbol in the message
17.    for symbol in message:
18.        if symbol in LETTERS:
19.            num = LETTERS.find(symbol) # get the number of the symbol
20.            num = num - key
21.
22.            # handle the wrap-around if num is 26 or larger or less than 0
23.            if num < 0:
24.                num = num + len(LETTERS)
25.
26.            # add number's symbol at the end of translated
27.            translated = translated + LETTERS[num]
28.
29.        else:
30.            # just add the symbol without encrypting/decrypting
31.            translated = translated + symbol
32.
33.    # display the current key being tested, along with its decryption
34.    print('Key #%s: %s' % (key, translated))
```

و عند تشغيل هذا البرنامج سوف نحصل على النتيجة التالية:

```
Key #0: GUVF VF ZL FRPERG ZRFFNTR.  
Key #1: FTUE UE YK EQODQF YQEEMSQ.  
Key #2: ESTD TD XJ DPNCPE XPDDLKP.  
Key #3: DRSC SC WI COMBOD WOCCCKQO.  
Key #4: CQRB RB VH BNLANC VNBBJPN.  
Key #5: BPQA QA UG AMKZMB UMAAIOM.  
Key #6: AOPZ PZ TF ZLJYLA TLZZHNL.  
Key #7: ZNOY OY SE YKIXKZ SKYYGMK.  
Key #8: YMNX NX RD XJHWJY RJXXFLJ.  
Key #9: XLMW MW QC WIGVIX QIWWEKI.  
Key #10: WKLV LV PB VHFUHW PHVVDJH.  
Key #11: VJKU KU OA UGETGV OGUUCIG.  
Key #12: UIJT JT NZ TFDSFU NFTTBHF.  
Key #13: THIS IS MY SECRET MESSAGE.  
Key #14: SGHR HR LX RDBQDS LDRRZFD.  
Key #15: RFGQ GQ KW QCAPCR KCQQYEC.  
Key #16: QEFP FP JV PBZOBQ JBPPXDB.  
Key #17: PDEO EO IU OAYNAP IAOOWCA.  
Key #18: OCDN DN HT NZXMZO HZNNVBZ.  
Key #19: NBCM CM GS MYWLYN GYMMUAY.  
Key #20: MABL BL FR LXVKXM FXLLTZX.  
Key #21: LZAK AK EQ KWUJWL EWKKSYW.  
Key #22: KYZJ ZJ DP JVTIVK DVJJRXV.  
Key #23: JXYI YI CO IUSHUJ CUIIQWU.  
Key #24: IWXH XH BN HTRGTI BTHHPVT.  
Key #25: HVWG WG AM GSQFSH ASGGOUS.  
>>>
```

لاحظ أن نتيجة فك التشفير باستخدام المفتاح 13 هي العبارة المفهومة وبالتالي فإن مفتاح التشفير المستخدم في عملية التشفير يجب أن يكون 13

## كيف يعمل هذا البرنامج:

```
1. # Caesar Cipher Hacker  
2. # http://inventwithpython.com/hacking (BSD Licensed)  
3.  
4. message = 'GUVF VF ZL FRPERG ZRFFNTR.'  
5. LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

caesarHacker.py

هذا البرنامج يبدأ بخلق متغير باسم `message` والذي يحوي على النص المشفر المراد فك تشفيره.

المتغير `LETTERS` يحوي على كل الأحرف التي يمكن أن يتم تشفيرها باستخدام الشيفرة وهذا المتغير يجب أن يحوي على نفس القيم التي يحويها المتغير `LETTERS` الموجود في برنامج شيفرة قيصر الذي استخدم من أجل القيام بعملية التشفير وإلا فإن برنامج فك الشيفرة لن يعمل بشكل صحيح.

## التابع `:range()`

```
7. # loop through every possible key  
8. for key in range(len(LETTERS)):
```

caesarHacker.py

السطر 8 يحوي على حلقة `for` والتي تقوم بالدوران عبر القيم التي يعيدها التابع `(range())` التابع `(range())` يأخذ قيمة عدد صحيح ويعيد قيم لمجال هذه القيمة وقيم هذا المجال يمكن أن تستخدم في حلقة `for` من أجل تكرار الحلقة عدد معين من المرات

جرب المثال التالي:

```
>>> for i in range(4):  
    print(i)  
  
0  
1  
2  
3  
>>>
```

```

>>> for i in range(4):
    print('Hello')

Hello
Hello
Hello
Hello
>>>

```

في السطر 8 حلقة for تقوم بضبط قيمة المفتاح ضمن المجال من 0 – 26 (ولكن لا تتضمن 26) بدل من أن نقوم بكتابة القيمة 26 بشكل مباشر في كود البرنامج، قمنا باستخدام القيمة المعادة من التابع (التابع LETTERS و السبب في ذلك هو في حال إضافة قيم جديدة أو تعديل القيم الموجودة في المتغير LETTERS سيؤدي البرنامج لعمل بشكل صحيح (عد إلى تشفير القيم الغير حرفية في الفصل السابق لمعرفة السبب)

عندما يتم تنفيذ البرنامج أول مرة فهو يذهب عبر الحلقة ويتم ضبط قيمة المفتاح key = 0 و يتم فك تشفير الرسالة باستخدام المفتاح 0 (الកود الموجود ضمن حلقة for هو الذي يقوم بعملية فك التشفير) و عند تنفيذ الدورة الثانية للحلقة for في السطر الثامن فسوف يتم ضبط قيمة المفتاح key = 1 و يتم فك تشفير الرسالة بالمفتاح 1 و تستمر هذه العملية إلى أن نصل إلى قيمة المفتاح 1

26

يمكنك تمرير عددين صحيحين للتابع range() بدل قيمة واحدة، حيث أن القيمة الأولى هي بداية المجال والقيمة الثانية هي لنهاية المجال ويتم الفصل بين القيمتين بفاصلة ،

جرب المثال التالي:

```

>>> for i in range(2, 6):
    print(i)

2
3
4
5
>>>

```

## العودة إلى كود البرنامج:

```
caesarHacker.py
7. # loop through every possible key
8. for key in range(len(LETTERS)):
9.
10.    # It is important to set translated to the blank string so that the
11.    # previous iteration's value for translated is cleared.
12.    translated = ''
```

في السطر 12 تم إسناد قيمة سلسلة نصية فارغة للمتغير `translated` وفي الأسطر القادمة سوف يتم إضافة النص بعد عملية فك التشفير إلى المتغير `translated`

```
caesarHacker.py
14. # The rest of the program is the same as the original Caesar program:
15.
16. # run the encryption/decryption code on each symbol in the message
17. for symbol in message:
18.     if symbol in LETTERS:
19.         num = LETTERS.find(symbol) # get the number of the symbol
```

من السطرين 31 – 17 استخدمنا نفس الكود الذي استخدمناه في برنامج شифرة فيصر في الفصل السابق ولكنه مختلف قليلاً لأن هذا الكود يقوم بعملية فك التشفير فقط بدلاً من التشفير وفك التشفير

في السطر 17 قمنا بتعريف حلقة `for` تقوم بالمرور أو الدوران على كل رمز من السلسلة النصية المشفرة الموجودة في المتغير `message`

في السطر 18 يتم فحص كل رمز فيما إذا كان حرف كبير (إذا كان موجود ضمن المتغير `LETTERS` والذي يحوي على أحرف كبيرة فقط) إذا كان هذا محق فنتم فك تشفير هذا الرمز.

في السطر 19 يتم تحديد مكان توضع الرمز في المتغير `LETTERS` باستخدام الطريقة `find()` ويتم تخزين القيمة التي تدل على مكان هذا الرمز في المتغير `num`

```
caesarHacker.py
20.     num = num - key
21.
22.     # handle the wrap-around if num is 26 or larger or less than 0
23.     if num < 0:
24.         num = num + len(LETTERS)
```

في السطر 20 يتم طرح قيمة المفتاح من قيمة num (في شифرة قيصر يتم الطرح في حالة فك التشفير) وفي حالة كانت النتيجة أصغر من الصفر يتم إضافة القيمة 26 (القيمة المعادلة من التابع (len(LETTERS)

```
caesarHacker.py
26.     # add number's symbol at the end of translated
27.     translated = translated + LETTERS[num]
```

الآن وبعد أن تم تعديل قيمة num يتم حساب قيمة LETTERS[num] والتي هي الحرف أو الرمز بعد عملية فك التشفير

في السطر 27 يتم إضافة هذا الرمز إلى السلسلة النصية الموجودة في المتغير translated

```
caesarHacker.py
29.     else:
30.         # just add the symbol without encrypting/decrypting
31.         translated = translated + symbol
```

إذا كان الشرط في السطر 18 غير محقق أي أن الرمز غير موجود في LETTERS (ليس حرف كبير) فلن نطبق عليه عملية فك التشفير ويتم إضافته كما هو إلى القيمة النصية الموجودة في المتغير translated

التعليمية else هي خاصة بالتعليمية if في السطر 18

## تنسيق السلسلة النصية:

```
caesarHacker.py
33.     # display the current key being tested, along with its decryption
34.     print('Key #%s: %s' % (key, translated))
```

في السطر 34 قمنا باستدعاء التابع `print()` ليقوم بطباعة عدة أسطر بحسب عدد مرات تكرار الحلقة `for` في السطر 8

القيم التي تم تمريرها إلى التابع `print()` لم نستخدمها من قبل، هذه القيم تقوم بتنسيق السلسلة النصية.

القيمة 5% تستخدم من أجل وضع سلسلة نصية بجانب سلسلة نصية أخرى، حيث يتم استبدال 5% الأولى بقيمة المتغير الأول واستبدال 5% الثانية بقيمة المتغير الثاني

جرب المثال التالي:

```
>>> 'Hello %s!' % ('world')
'Hello world!'
>>> 'Hello' + ' world' + '!'
'Hello world!'
>>> 'The %s ate the %s that ate %s.' % ('dog', 'cat', 'rat')
'The dog ate the cat that ate rat.'
>>>
```

تنسيق النصوص بهذه الطريقة هو أسهل من عملية جمع النصوص باستخدام المعامل '+' إحدى ميزات تنسيق النص هذه هي إمكانية إضافة قيم غير نصية (أرقام مثلاً) إلى السلسلة النصية وهذا الأمر غير ممكن باستخدام المعامل '+' حيث لا يمكن جمع قيمة نصية مع قيمة رقمية وسوف تحصل على رسالة خطأ، جرب المثال التالي:

```
>>> '%s had %s pies.' % ('Alice', 42)
'Apple had 42 pies.'
>>> 'Alice' + ' had ' + 42 + ' pies.'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    'Alice' + ' had ' + 42 + ' pies.'
TypeError: Can't convert 'int' object to str implicitly
>>>
```

في السطر 34 من كود البرنامج استخدمنا خاصية تنسق النص من أجل خلق سلسلة نصية تحوي على قيمتي المتغيرين `key`, `translated` لأن المتغير `key` يحوي على قيمة عدبية (عدد صحيح) أما المتغير `translated` يحوي على سلسلة نصية لذلك استخدمنا خاصية التنسيق السابقة.

## الخلاصة:

الضعف في شيفرة قيصر هو وجود عدد محدود من المفاتيح التي يمكن أن تستخدم في عملية التشفير وفك التشفير

أي جهاز كمبيوتر يمكنه بسهولة القيام بعملية فك التشفير باستخدام ال 26 مفتاح المحتملين ثم يقوم محل الشيفرة بالنظر إلى الرسائل الناتجة من عملية فك التشفير ومعرفة الرسالة الحقيقية. من أجل جعل رسائل أكثر سرية يجب أن تستخدم عدد أكبر من مفاتيح التشفير.

شيفرة التحويل في الفصل القادم يمكن أن تؤمن هذا لـك.



## التشفيير باستخدام شيفرة

### التحويل

---

محتوى هذا الفصل:

- خلق تابع باستخدام التعليمة `def`
- التابع `(main()`
- البارامترات
- المجالات العامة والمجالات المحلية والمتغيرات العامة والمتغيرات المحلية
- التابع `(list()`
- المعاملات `(+=, -=, *=, /=)`
- الطريقة النصية `(join())`
- إعادة القيم والتعليق `return`
- المتغير الخاص `_name`

شيفرة قيسر ليست شيفرة آمنة لأنه يمكن فكها باستخدام تقنية القوة الغاشمة من خلال المرور على كل القيم الممكنة الـ 26 لفتح التشفير.

شيفرة التحويل تملك عدد أكبر من مفاتيح التشفير الممكنة وهذا يجعل هجوم القوة الغاشمة أكثر صعوبة.

## التشفيير باستخدام شيفرة التحويل:

### Encryption with the Transposition Cipher

بدلاً من استبدال الحروف بحروف أخرى فإن شيفرة التحويل تقوم بدمج رموز في نص الرسالة من أجل جعل الرسالة الأصلية غير قابلة للقراءة.

'Common sense is not so common.' قبل البدء بكتابة الكود، لنقم بتشفيير الرسالة

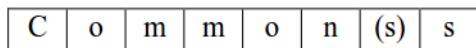
باستخدام الورقة والقلم وبشكل يتضمن الفراغات وعلامات الترقيم، هذا يعني أن الرسالة مكونة من 30 حرفاً.

سوف نستخدم الرقم 8 كمفتاح.

أول خطوة هي رسم عدد من الصناديق مساوي لقيمة المفتاح (ثمانية صناديق في هذا المثال)



ثاني خطوة هي كتابة الرسالة المراد تشفيرها في داخل الصناديق، كل حرف في صندوق تذكر أن المسافة الفارغة تعتبر حرف (في هذا الكتاب قمنا بكتابته (S) للإشارة إلى المسافة الفارغة لكي لا تبقى صناديق فارغة



نحن نملك فقط 8 صناديق ولكن الرسالة المراد تشفيرها مكونة من 30 حرفاً لذلك سوف نقوم برسم ثمانية صناديق أخرى تحت الصناديق الأولى ونقوم بكتابة بقية الأحرف في داخلها

1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>
C	o	m	m	o	n	(s)	s
e	n	s	e	(s)	i	s	(s)
n	o	t	(s)	s	o	(s)	c
o	m	m	o	n	.		

قمنا بتظليل آخر صندوقين من أجل تجاهلهم.

النص المشفر هو الأحرف التي تقرأ من الأعلى إلى الأسفل لكل عمود

'o' , 'n' , 'C' , 'e' هذه الأحرف من العمود الأول بعدها ننتقل لأحرف العمود الثاني ثم الثالث وهكذا حتى نهاية العواميد

النص المشفر هو قراءة الأحرف من الأعلى إلى الأسفل ثم الانتقال إلى العمود الثاني

'o' , 'n' , 'o' , 'm'

ويتم تجاهل الصناديق المظللة.

النص المشفر هو 'Cenoonommstmme oo snnio s s c'

وهو ممزوج بشكل كاف لمنع أي شخص من معرفة نص الرسالة الأصلية من خلال النظر إلى الرسالة المشفرة.

### خطوات عملية التشفير:

١. تحديد عدد أحرف الرسالة و اختيار قيمة مفتاح التشفير
٢. رسم عدد من الصناديق مساوي لقيمة مفتاح التشفير
٣. ملئ الصناديق بأحرف الرسالة من اليسار إلى اليمين
٤. عند امتلاء الصناديق وعدم انتهاء أحرف الرسالة يتم إضافة سطر جديد من الصناديق
٥. تظليل الصناديق الفارغة في نهاية الصناديق
٦. كتابة الرسالة المشفرة من خلال قراءة الأحرف من الأعلى إلى الأسفل و عند الانتهاء من أول عمود ننتقل إلى العمود المجاور ويتم تجاهل الصناديق المظللة

## برنامج التشفير باستخدام شيفرة التحويل:

التشفير باستخدام الورق والقلم يتطلب الكثير من الجهد ويمكن أن نقع بأخطاء كتابية أثناء عملية التشفير.

لنبدأ بكتابه برنامج يقوم بعملية التشفير باستخدام شيفرة التحويل ( برنامج فك التشفير سوف يتم شرحه في الفصل القادم )

عند استخدام هذا البرنامج سوف تظهر مشكلة بسيطة، إذا كانت المسافات الفارغة في نهاية النص المشفر فلن نتمكن من رؤية هذه المسافات أو لن نتمكن من معرفة عدد هذه المسافات ومن أجل حل هذه المشكلة فإن البرنامج سوف يضيف الرمز | في نهاية النص المشفر (الرمز موجود في لوحة المفاتيح فوق زر انتر)

مثلاً:

```
Hello| # There are no spaces at the end of the message.  
Hello | # There is one space at the end of the message.  
Hello  | # There are two spaces at the end of the message.
```

## ال kod البرمجي لبرنامج الشifer باستخدام شيفرة التحويل:

قم بفتح ملف جديد من خلال File > New File ثم قم بكتابة الكود التالي في داخله ثم قم بحفظه باسم transpositionEncrypt.py واضغط F5 من أجل تنفيذ هذا البرنامج

يجب أن تقوم أولاً بوضع الوحدة pyperclip.py في نفس المجلد الذي سوف تقوم بحفظ البرنامج في داخله، يمكنك الحصول على pyperclip.py module من

<http://invpy.com/pyperclip.py>



## Source code for transpositionEncrypt.py

```
1. # Transposition Cipher Encryption
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. import pyperclip
5.
6. def main():
7.     myMessage = 'Common sense is not so common.'
8.     myKey = 8
9.
10.    ciphertext = encryptMessage(myKey, myMessage)
11.
12.    # Print the encrypted string in ciphertext to the screen, with
13.    # a | (called "pipe" character) after it in case there are spaces at
14.    # the end of the encrypted message.
15.    print(ciphertext + '|')
16.
17.    # Copy the encrypted string in ciphertext to the clipboard.
18.    pyperclip.copy(ciphertext)
19.
20.
21. def encryptMessage(key, message):
22.     # Each string in ciphertext represents a column in the grid.
23.     ciphertext = [''] * key
24.
25.     # Loop through each column in ciphertext.
26.     for col in range(key):
27.         pointer = col
28.
29.         # Keep looping until pointer goes past the length of the message.
30.         while pointer < len(message):
31.             # Place the character at pointer in message at the end of the
32.             # current column in the ciphertext list.
33.             ciphertext[col] += message[pointer]
34.
35.             # move pointer over
36.             pointer += key
37.
38.     # Convert the ciphertext list into a single string value and return it.
39.     return ''.join(ciphertext)
40.
41.
42. # If transpositionEncrypt.py is run (instead of imported as a module) call
43. # the main() function.
44. if __name__ == '__main__':
45.     main()
```

نتيجة تنفيذ هذا البرنامج هي:

```
Cenoonommstmmme oo snnio. s s c|  
>>>
```

النص المشفر (بدون الرمز | ) تم نسخه إلى الذاكرة ويمكنه لصقه في أي ملف نصي أو في رسالة إيميل لنقوم بإرساله إلى أي شخص تريده.

إذا كنت تري تشفير رسالة مختلفة أو استخدام مفتاح تشفير مختلف قم بتعديل القيم المنسنة للمتغيرات 7 and 8 في السطرين myMessage and myKey

## كيف يعمل هذا البرنامج:

```
transpositionEncrypt.py  
1. # Transposition Cipher Encryption  
2. # http://inventwithpython.com/hacking (BSD Licensed)  
3.  
4. import pyperclip
```

في برنامج التشفير باستخدام شيفرة التحويل وبشكل مماثل لبرنامج شيفرة قيصر سيتم نسخ النص المشفر إلى الذاكرة لذلك قمنا باستيراد الوحدة pyperclip لنتمكن من استدعاء التابع pyperclip.copy()

## :def خلقتابع باستخدام التعليمية

```
transpositionEncrypt.py  
6. def main():  
7.     myMessage = 'Common sense is not so common.'  
8.     myKey = 8
```

التابع (مثل التابع `print()`) هو برنامج صغير داخل البرنامج الأصلي عندما يتم استدعاء التابع يقوم البرنامج بتنفيذ التعليمات الموجودة داخل التابع ثم يعود لتنفيذ الكود في السطر الذي يلي استدعاء التابع.

يمكنك خلق تابع باستخدام التعليمة `def` كما في السطر 6

التعليمة `def` في السطر 6 لا تقوم باستدعاء التابع المسمى `(main)` ولكن تقوم بخلق تابع جديد اسمه `(main)` وهذه التعليمة هي اختصار للكلمة `defining` (تعريف).

بعد خلق التابع يمكننا استدعائه لاحقاً في برنامجك وعند الاستدعاء فإن البرنامج سوف ينتقل لتنفيذ الكود الموجود داخل هذا التابع (كتلة التعليمات التي تلي التعليمة `def`)

افتح نافذة جديدة لمحرر النصوص واكتب الكود التالي:

#### Source code for helloFunction.py

```
1. def hello():
2.     print('Hello!')
3.     total = 42 + 1
4.     print('42 plus 1 is %s' % (total))
5. print('Start!')
6. hello()
7. print('Call it again.')
8. hello()
```

قم بحفظه باسم `helloFunction.py` ثم اضغط F5 من أجل التنفيذ وستكون النتيجة كالتالي:

```
>>>
Start!
Hello
42 plus 1 is 43
Call it again.
Hello
42 plus 1 is 43
>>>
```

عندما يعمل هذا البرنامج فهو يبدأ التنفيذ من الأعلى، في أول سطر التعليمة `def` التي تقوم بتعريف تابع باسم `(hello)` ثم يقوم البرنامج بتجاوز تنفيذ كتلة التعليمات التي تلي التعليمة `def`

ويقوم بتنفيذ التعليمية `print('Start!')` وهذا سبب ظهور الكلمة 'Start!' في أول سطر من نتيجة تنفيذ البرنامج.

في السطر الذي يلي التعليمية `print('Start!')` يتم استدعاء التابع `hello` وعندما فإن البرنامج سوف ينتقل لتنفيذ الكود الموجود في كتلة التعليمات الخاصة بهذا التابع.

هذا التابع يقوم بطباعة الكلمة 'Hello' ويقوم بجمع  $1 + 42$  ويطبع النتيجة على الشاشة.

عندما ينتهي البرنامج من تنفيذ التعليمات الخاصة بالتابع فسوف يعود لتنفيذ ما تبقى من الكود الموجود في السطر الذي يلي تعليمية استدعاء التابع وهو `print('Call it again.')` وبعد تنفيذ هذه التعليمية يتم استدعاء التابع `hello` مرة ثانية وعندما سوف ينتقل البرنامج لتنفيذ الكود الخاص بالتابع مرة ثانية.

## التابع (`main()` في البرنامج:

```
transpositionEncrypt.py
6. def main():
7.     myMessage = 'Common sense is not so common.'
8.     myKey = 8
```

في برنامج هذا الكتاب يوجد تابع باسم `main()` والذي عند استدعاءه يقوم بتشغيل البرنامج. السبب سوف يتم شرحه في نهاية هذا الفصل ولكن الآن يجب أن تعرف أن التابع `(main)` في البرنامج في هذا الكتاب دائماً يتم استدعاءه بعد أن تعمل البرنامج.

في السطرين 8 and 7 تم تعريف المتغيرين `myMessage` and `myKey` حيث سيتم تخزين النص الصريح (النص الغير مشفر) ومفتاح التشفير في هذين المتغيرين

```
transpositionEncrypt.py
10. ciphertext = encryptMessage(myKey, myMessage)
```

ال kod الذي سوف يقوم بعملية التشفير قمنا بوضعه داخل تابع يتم تعريفه في السطر 21 باسم `encryptMessage()`

هذا التابع سوف يمرر له قيمتين:

- قيمة عدديّة (عدد صحيح) وهي قيمة المفتاح
- وقيمة نصيّة وهي الرسالة المراد تشفيرها

عند استدعاء هذا التابع يتم الفصل بين القيم الممررة إليه باستخدام فاصلة ' , '

القيمة التي يعيدها التابع `encryptMessage()` هي قيمة نصيّة وهي النص المشفر.

كود هذا التابع سوف يتم شرحه لاحقاً في هذا الفصل.

القيم التي يعيدها هذا التابع سوف يتم تخزينها في المتغير `ciphertext`

```
transpositionEncrypt.py
12.     # Print the encrypted string in ciphertext to the screen, with
13.     # a | (called "pipe" character) after it in case there are spaces at
14.     # the end of the encrypted message.
15.     print(ciphertext + '|')
16.
17.     # Copy the encrypted string in ciphertext to the clipboard.
18.     pyperclip.copy(ciphertext)
```

الرسالة المشفر سوف يتم طباعتها على الشاشة في السطر 15 ويتم نسخها إلى الذاكرة في السطر 15

البرنامج يقوم بطباعة الرمز ' | ' في نهاية الرسالة المشفرة لكي يتمكن المستخدم من معرفة إذا كان يوجد فراغات في نهاية السلسلة المشفرة.

السطر 18 هو آخر سطر في جسم التابع `main()` وبعد تنفيذه سوف يعود البرنامج لتنفيذ السطر الذي يليه سطر استدعاء هذا التابع.

استدعاء التابع `main()` هو في السطر 45 في نهاية البرنامج لذلك بعد الانتهاء من تنفيذ آخر سطر من التعليمات الخاصة بهذا التابع سوف ينتهي البرنامج.

## البارامترات:

```
21. def encryptMessage(key, message):
```

transpositionEncrypt.py

التابع الذي يقوم بعملية التشفير `encryptMessage()` يأخذ بارامترین.

**البارامترات:** هي متغيرات تحوي على قيم يتم تمريرها إلى التابع عند استدعاءه.

البارامترات يتم حذفها بشكل أوتوماتيكي بعد تنفيذ التابع.

عندما يتم استدعاء التابع `encryptMessage()` في السطر 10 ، يتم تمرير قيمتين له (`myKey` and `myMessage`)

هذه القيم تسند إلى البارامترات `key` and `message` (التي يمكن أن تراها في السطر 21)

**البارامتر:** هو اسم المتغير الموجود بين القوسين ( ) الخاصين بالتابع.

البایثون سوف يظهر رسالة خطأ إذا قمت بتمرير قيم أكثر أو أقل من عدد البارامترات التي يملكها التابع، جرب المثال التالي:

```
>>> len('hello', 'world')
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    len('hello', 'world')
TypeError: len() takes exactly one argument (2 given)
>>> len()
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    len()
TypeError: len() takes exactly one argument (0 given)
>>>
```

في التابع `len()` يتم تمرير قيمة واحدة فقط وسوف تحصل على رسالة خطأ في حال تمرير أكثر من قيمة أو عدم تمرير أي قيمة.

## تغيير البارامترات الموجودة في داخل التابع فقط:

انظر إلى البرنامج التالي الذي يقوم بتعريف ثم استدعاء التابع `func()`

```
def func(param):
    param = 42
spam = 'Hello'
func(spam)
print(spam)
```

عند تشغيل هذا البرنامج فإن التابع `print()` الذي تم استدعاءه في السطر الأخير سوف يقوم  
بطباعة 'Hello' وليس 42

```
>>>
Hello
>>>
```

عندما يتم استدعاء التابع `func()` ويتم تمرير المتغير `spam` له فإن المتغير `spam` لن يكون له القيمة 42 وبدل ذلك ستكون له القيمة الموجودة في المتغير `param` وأي تغيير في قيمة `param` لن تغير قيمة `spam` (يوجد استثناء لهذه القاعدة، عندما نقوم بتمرير قيمة لقائمة `list` أو قاموس `dictionary` والتي سوف يتم شرحها في الفصل 10 )

من الضروري أن نفهم أن المتغير الذي يتم وضعه بين القوسين عند تعريف التابع يسمى باراميتر وأن القيمة التي يتم تمريرها للتابع عند استدعائه يتم نسخها إلى الباراميتر، فإذا تغير الباراميتر فإن المتغير الذي يحوي على القيمة التي يتم تمريرها للتابع لن يتغير.

## المتغيرات في المجال المحلي والمجال العام:

يمكن أن تكون متقاضي لماذا قمنا باستخدام البارامترتين `key` and `message` رغم أننا نملك متغيرين `myKey` and `myMessage` في التابع `main()`

السبب هو أن المتغيران myKey and myMessage هي متغيرات محلية  
ولا يمكن استخدامها خارج التابع main()

في كل مرة يتم فيها استدعاء التابع، يتم خلق مجال محلي local scope

المتغيرات التي يتم خلقها أثناء استدعاء التابع تكون موجودة داخل هذا المجال المحلي.

البارامترات دائمًا تكون موجودة في المجال المحلي.

عندما يقوم التابع بإعادة قيمة فإن المجال المحلي يتم تدميره والمتغيرات المحلية يتم نسيانها.

المتغير في المجال المحلي يبقى منفصل عن المتغير في المجال العام (الموجود خارج جسم التابع) حتى ولو كان لهذين المتغيرين نفس الاسم.

المتغيرات التي يتم خلقها خارج التابع تكون موجودة في المجال العام global scope

عندما ينتهي البرنامج من التنفيذ فيتم تدمير المجال العام وكل المتغيرات الموجودة في البرنامج تُنسى (كل المتغيرات في برنامج الشبيرة العكسية وبرنامج شبيرة قبصر هي متغيرات عامة)

## التعليمية global:

إذا كنت تريد لمتغير موجود داخل جسم التابع أن يكون متغير عام قم باستخدام التعليمية global  
جانب اسم التابع في أول سطر بعد التعليمية def

التالي هو قواعد المتغيرات العامة والمحلية:

١. المتغيرات خارج التابع هي دائمًا متغيرات عامة
٢. إذا لم يتم تعين قيمة لمتغير داخل أي تابع فهو متغير عام
٣. إذا كان المتغير داخل التابع فلا يمكن استخدامه في تعليمات عامة ويتم استخدامه فقط كمتغير محلي
٤. إذا تم استخدام التعليمية global مع متغير محلي داخل تابع فيصبح متغير عام

أكتب البرنامج التالي في محرر النصوص واحفظه باسم scope.py واضغط F5 للتنفيذ:

#### Source code for scope.py

```
1. spam = 42
2.
3. def eggs():
4.     spam = 99 # spam in this function is local
5.     print('In eggs():', spam)
6.
7. def ham():
8.     print('In ham():', spam) # spam in this function is global
9.
10. def bacon():
11.     global spam # spam in this function is global
12.     print('In bacon():', spam)
13.     spam = 0
14.
15. def CRASH():
16.     print(spam) # spam in this function is local
17.     spam = 0
18.
19. print(spam)
20. eggs()
21. print(spam)
22. ham()
23. print(spam)
24. bacon()
25. print(spam)
26. CRASH()
```

نتيجة التنفيذ ستكون كالتالي:

```
42
In eggs():  99
42
In ham() :  42
42
In bacon() :  42
0
Traceback (most recent call last):
  File "C:/Python34/scope.py", line 21, in <module>
    CRASH()
  File "C:/Python34/scope.py", line 12, in CRASH
    print(spam)
UnboundLocalError: local variable 'spam' referenced before assignment
```

عند استخدام المتغير spam في الأسطر 19, 21, 23, 25, 1 فهو خارج كل التوابع فهو متغير عام.

في التابع () eggs تم إسناد القيمة 99 للمتغير spam لذلك فإن البيثون سوف يعتبر هذا المتغير كمتغير محلي وهو مختلف بشكل عن المتغير العام الذي له نفس الاسم spam

اسناد القيمة 99 إلى المتغير المحلي spam لا تؤثر على قيمة المتغير العام spam (المتغيران مختلفان تماماً ولكن لهما نفس الاسم).

المتغير spam في التابع () ham لم يتم اسناد أي قيمة له داخل هذا التابع لذلك فهو متغير عام.

المتغير spam في التابع () bacon يستخدم التعليمة global لذلك فهو متغير عام وعملية اسناد القيمة 0 في السطر 13 سوف تغير قيمة المتغير العام spam = spam

المتغير spam في التابع () CRASH() تم اسناد قيمة له ولا يتم استخدام التعليمة global معه لذلك فهو متغير محلي، ولكن انتبه أنه تم استخدامه في التابع () print (في السطر 16) قبل أن يتم اسناد قيمة له (السطر 17) وهذا هو سبب ظهور رسالة الخطأ عند استدعاء التابع () CRASH()

```
UnboundLocalError: local variable 'spam' referenced before assignment
```

يمكن أن يكون أمر مشوش في حال استخدام نفس الاسم لمتغيرين محلي وعام لذلك يفضل استخدام أسماء مختلفة.

## القائمة :List

```
transpositionEncrypt.py  
22.     # Each string in ciphertext represents a column in the grid.  
23.     ciphertext = [''] * key
```

في السطر 23 قمنا باستخدام نوع جديد من البيانات يسمى القائمة list

القائمة يمكن أن تحوي على عدة قيم، وبشكل مماثل للسلسلة النصية التي تبدأ وتنتهي بعلامة التنصيص فإن القائمة تبدأ وتنتهي بالقوسین [ ] ويتم وضع القيم المراد حفظها داخل هذين القوسين ويتم الفصل بين هذه القيم باستخدام الفاصلة ' , '

جرب المثال التالي في الشل التفاعلية:

```
>>> animals = ['cat', 'dog', 'horse']
>>> animals
['cat', 'dog', 'horse']
>>>
```

المتغير `animals` يحوي على قائمة من القيم وهذه القيم هي ثلاثة قيم نصية كل قيمة داخل القائمة تسمى مادة `item`. القوائم مفيدة جداً عندما نريد تخزين الكثير من القيم ولا نريد خلق متغير لكل قيمة. العديد من العمليات يمكن أن تعمل مع القوائم مثل الفهرسة `indexing` والاقطاع `slicing` دليل الفهرسة يشير إلى الماده داخل القائمه.

جرب المثال التالي:

```
>>> animals = ['cat', 'dog', 'horse']
>>> animals[0]
'cat'
>>> animals[1]
'dog'
>>> animals[2]
'horse'
>>> animals[1:2]
['dog']
>>> animals[0:2]
['cat', 'dog']
>>>
```

تذكر أن دليل الفهرسة يبدأ بالقيمة **0** وليس القيمة **1**

يمكن استخدام الحلقة `for` من أجل المرور على كل القيم الموجودة في القائمه

جرب المثال التالي:

```
>>> for spam in ['cat', 'dog', 'horse']:
    print ('I like ' + spam)

I like cat
I like dog
I like horse
>>>
```

## استخدام التابع (list) في مجال القائمة:

إذا كنت تريد قائمة من القيم التي تزداد بمقدار معين في كل قيمة يمكنك بناء هذه القائمة باستخدام الحلقة for كما في المثال التالي:

```
>>> myList = []
>>> for i in range(10):
    myList = myList + [i]

>>> myList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

يمكن القيام بهذه العملية بشكل مباشر من خلال تمرير التابع (range) إلى التابع (list)

```
>>> myList = list(range(10))
>>> myList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

التابع (list) يمكنه أيضاً تحويل سلسلة نصية إلى قائمة بحيث تكون المواد أو القيم داخل القائمة هي أحرف هذه السلسلة، كما في المثال التالي:

```
>>> myList = list('Hello world!')
>>> myList
['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '!']
>>>
```

لن نستخدم التابع (list) في مجال القوائم أو مع السلاسل النصية في هذا البرنامج ولكن سيتم ذلك في فصل لاحق في هذا الكتاب.

## إعادة تعيين الموارد في القائمة:

المواد داخل القائمة يمكن أن يتم تعديلها، جرب المثال التالي:

```
>>> animals = ['cat', 'dog', 'horse']
>>> animals
['cat', 'dog', 'horse']
>>> animals[2] = 9999
>>> animals
['cat', 'dog', 9999]
>>>
```

## إعادة تعيين الأحرف في السلسلة النصية:

يمكننا تعديل الموارد في القوائم ولكن لا يمكننا إعادة تعيين حرف في سلسلة نصية وفي حال تنفيذ ذلك سوف نحصل على رسالة خطأ

جرب المثال التالي:

```
>>> 'Hello world!'[6] = 'x'
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in <module>
    'Hello world!'[6] = 'x'
TypeError: 'str' object does not support item assignment
>>>
```

من أجل تغيير حرف في سلسلة نصية يتم استخدام الاقطاع

جرب المثال التالي:

```
>>> spam = 'Hello world!'
>>> spam = spam[:6] + 'x' + spam[7:]
>>> spam
'Hello xorld!'
```

## قائمة بمجموعة من القوائم:

القائمة يمكن أن تحوي في داخلها على قوائم أخرى

جرب المثال التالي:

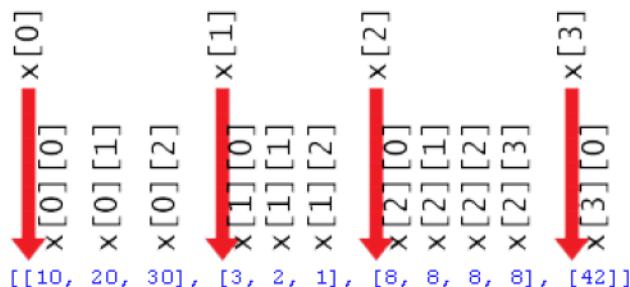
```
>>> spam = [['dog', 'cat'], [1, 2, 3]]  
>>> spam[0]  
['dog', 'cat']  
>>> spam[0][0]  
'dog'  
>>> spam[0][1]  
'cat'  
>>> spam[1][0]  
1  
>>> spam[1][1]  
2  
>>>
```

كما يمكن استخدام مجموعة أخرى من الفهرسة في حال كانت القيم هس سلاسل نصية

جرب المثال التالي:

```
>>> spam = [['dog', 'cat'], [1, 2, 3]]  
>>> spam[0][1][1]  
'a'  
>>>
```

الشكل التالي يظهر الفهرسة لقائمة اسمها x تحوي على عدد من القوائم في داخلها



## استخدام التابع () len() والمعامل in مع القوائم:

نستخدم التابع () len() لمعرفة عدد الأحرف في سلسلة نصية (طول السلسلة). التابع () len() يمكن أن يعمل أيضاً مع القوائم ويعيد عدد صحيح هو عدد الموارد الموجودة داخل القائمة.

جرب المثال التالي:

```
>>> animals = ['cat', 'dog', 'horse']
>>> len(animals)
3
>>>
```

المعامل in يستخدم من أجل معرفة فيما إذا كانت سلسلة نصية ما موجودة في داخل سلسلة نصية أخرى.

المعامل in يعمل أيضاً مع القوائم.

جرب المثال التالي:

```
>>> animals = ['cat', 'dog', 'horse']
>>> 'cat' in animals
True
>>> 'dog' in animals
True
>>> 'anteat' in animals
False
>>>
```

كما أن علامات التنصيص الفارغة كانت تعبر عن سلسلة نصية فارغة فإن القوسين [ ] الفارغين يمثلان قائمة فارغة.

جرب المثال التالي:

```
>>> animals = []
>>> len(animals)
0
>>>
```

## جمع وتكرار القوائم باستخدام المعاملات \* + and :

بشكل مماثل لاستخدام المعاملات \* + مع السلسل النصية فإن هذه المعاملات يمكن أن تعمل مع القوائم.

جرب المثال التالي:

```
>>> 'hello ' + 'world'  
'hello world'  
>>> 'hello ' * 5  
'hello hello hello hello hello '  
>>>
```

المثال السابق هو للتذكير باستخدام المعاملات \* + مع السلسل النصية

يمكنك استخدام المعاملات \* + مع القوائم بشكل مشابه للمثال السابق

## خوارزمية شيفرة التحويل:

يجب أن نقوم بتحويل الخطوات التي قمنا بها باستخدام الورقة والقلم إلى كود بلغة البايثون

- النص المراد تشفيره 'Common sense is not common.'
- مفتاح التشفير 8

إذا قمت بكتابة أحرف هذا النص داخل الصناديق سوف تحصل على الشكل التالي:

C	o	m	m	o	n	(s)	s
e	n	s	e	(s)	i	s	(s)
n	o	t	(s)	s	o	(s)	c
o	m	m	o	n	.		

دليل الفهرسة لكل حرف في الصناديق (تذكر أن الفهرسة تبدأ من الرقم 0 وليس من الرقم 1)

C 0	o 1	m 2	m 3	o 4	n 5	(s) 6	s 7
e 8	n 9	s 10	e 11	(s) 12	i 13	s 14	(s) 15
n 16	o 17	t 18	(s) 19	s 20	o 21	(s) 22	c 23
o 24	m 25	m 26	o 27	n 28	.		

من هذه الصناديق يمكننا أن نرى:

العمود الأول له قيم الفهرسة 0, 8, 16, 24 وهي للأحرف o

العمود الثاني له قيم الفهرسة 1, 9, 17, 25 وهي للأحرف m

ويمكن أن نستنتج بأن العمود رقم n سوف يكون له قيم الفهرسة  $0+n, 8+n, 16+n, 24+n$

C $0+0=0$	o $1+0=1$	m $2+0=2$	m $3+0=3$	o $4+0=4$	n $5+0=5$	(s) $6+0=6$	s $7+0=7$
e $0+8=8$	n $1+8=9$	s $2+8=10$	e $3+8=11$	(s) $4+8=12$	i $5+8=13$	s $6+8=14$	(s) $7+8=15$
n $0+16=16$	o $1+16=17$	t $2+16=18$	(s) $3+16=19$	s $4+16=20$	o $5+16=21$	(s) $6+16=22$	c $7+16=23$
o $0+24=24$	m $1+24=25$	m $2+24=26$	o $3+24=27$	n $4+24=28$	.	5+24=29	

لماذا اخترنا الأرقام ؟ 0, 8, 16, 24

هذه الأرقام حصلنا عليها من إضافة قيمة المفتاح (الرقم 8 في هذا المثال) ابتداءً من الرقم 0

$$0+8=8, \quad 8+8=16, \quad 16+8=24$$

ولكننا لم نستخدم هذه القيمة لأنها أكبر من عدد أحرف الرسالة ولهذا السبب توقفنا

عند الرقم 24

إذا تخيلت قائمة تحوي على ثمانية سلاسل نصية بحيث أن كل سلسلة مكونة من الأحرف الموجودة في كل عمود فإن قيمة هذه القائمة ستكون كالتالي:

```
[ 'Ceno', 'onom', 'mstm', 'me o', 'o sn', 'nio.', ' s ', 's c' ]
```

بهذه الطريقة يمكننا محاكات الصناديق بקוד بلغة بايثون

سنقوم بخلق قائمة فارغة والتي تحوي على عدد من السلالس النصية الفارغة مساوي لقيمة مفتاح التشفير لأن كل سلسلة نصية سوف تمثل عمود

(في هذا المثال القائمة سوف تحوي على 8 سلالس نصية لأن مفتاح التشفير هو 8)

```
transpositionEncrypt.py  
22.     # Each string in ciphertext represents a column in the grid.  
23.     ciphertext = [''] * key
```

المتغير `ciphertext` هو قائمة تحوي على عدد من القيمة النصية

كل سلسلة نصية في هذه القائمة تمثل عمود، حيث أن :

`ciphertext[0]` هو العمود الأول من اليسار

`ciphertext[1]` هو العمود الثاني وهكذا

السلالس النصية سوف تحوي على كل الأحرف الموجودة داخل كل عمود

0	1	2	3	4	5	6	7
C	o	m	m	o	n	(s)	s
e	n	s	e	(s)	i	s	(s)
n	o	t	(s)	s	o	(s)	c
o	m	m	o	n	.		

المتغير `ciphertext` الخاص بهذه الشبكة سيكون كالتالي:

```
>>> ciphertext = ['Ceno', 'onom', 'mstm', 'me o', 'nio.', ' s ', 's c']  
>>> ciphertext[0]  
'Ceno'  
>>> ciphertext[1]  
'onom'  
>>>
```

أول خطوة لخلق هذه القائمة هي خلق عدة سلاسل نصية فارغة بعد الأعمدة (قيمة مفتاح التشفير)، يمكننا استخدام عملية التكرار من أجل تكرار سلسلة نصية فارغة عدد من المرات بحسب قيمة مفتاح التشفير.

وهذه ما يتم حسابه في السطر 23

```
transpositionEncrypt.py
25.     # Loop through each column in ciphertext.
26.     for col in range(key):
27.         pointer = col
```

الخطوة التالية هي إضافة نص إلى كل سلسلة نصية في القائمة `ciphertext`.  
الحلقة `for` في السطر 26 تقوم بالدوران على كل عمود حيث أن المتغير `col` سوف يملك قيم  
تستخدم في الفهرسة في `ciphertext`.  
المتغير `col` سوف يبدأ بالقيمة 0 من أجل أول دورة للحلقة `for` ثم القيمة 1 من أجل ثاني دورة  
للحلقة `for` وهكذا.

وبهذه الطريقة فإن `ciphertext[col]` سوف تمثل السلسلة النصية الموجودة في العمود ذو  
الرقم `col`.

أما المتغير `pointer` فسوف يستخدم في الفهرسة للقيمة النصية المحفوظة في المتغير  
، وفي كل دورة من حلقة `for` فإن قيمة `pointer` سوف تبدأ بنفس قيمة `col` (كما  
في السطر 27)

## معاملات الزيادة:

من أجل اسناد قيمة جديدة لمتغير بالاعتماد على قيمة المتغير الحالية نقوم باستخدام معاملات الزيادة كما في المثال التالي:

```
>>> spam = 40
>>> spam = spam + 2
>>> print(spam)
42
>>>
```

يمكن القيام بهذه العملية من خلال استخدام المعامل `=+`

جرب المثال التالي:

```
>>> spam = 40
>>> spam += 2
>>> print(spam)
42
>>> spam = 'Hello'
>>> spam += ' world!'
>>> print(spam)
Hello world!
>>>
>>> spam = ['dog']
>>> spam += ['cat']
>>> print(spam)
['dog', 'cat']
>>>
```

التعليمية `= 2` تعطي نفس نتيجة التعليمية `spam = spam + 2` ولكن التعليمية الأولى هي مختصرة.

المعامل `= +` يتم استخدامه لإضافة أعداد صحيحة مع بعضها أو لإضافة سلاسل نصية مع بعضها أو حتى لإضافة مواد أو قيم لقوائم.

الجدول التالي يظهر بعض المعاملات والعمليات المكافئة لها:

العملية المكافحة	المعامل
$\text{Spam} = \text{spam} + 2$	$\text{Spam} += 2$
$\text{Spam} = \text{spam} - 2$	$\text{Spam} -= 2$
$\text{Spam} = \text{spam} * 2$	$\text{Spam} *= 2$
$\text{Spam} = \text{spam} / 2$	$\text{Spam} /= 2$

## العودة إلى كود البرنامج:

```

29.     # Keep looping until pointer goes past the length of the message.
30.     while pointer < len(message):
31.         # Place the character at pointer in message at the end of the
32.         # current column in the ciphertext list.
33.         ciphertext[col] += message[pointer]
34.
35.         # move pointer over
36.         pointer += key

```

في داخل الحلقة `for` التي تبدأ في السطر 26 قمنا بتعريف حلقة `while` والتي تبدأ في السطر 30 ، من أجل كل عمود نريد المرور على المتغير `message` ونلقط كل حرف دليله هو من مضاعفات قيمة مفتاح التشفير .

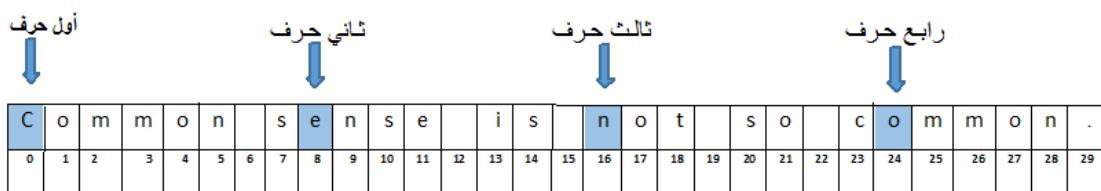
في السطر 27 ومن أجل أول دورة للحلقة `for` كانت قيمة `pointer` هي 0 طالما أن قيمة `pointer` هي أصغر من طول الرسالة، نريد أن يتم إضافة الحرف `[message[pointer]]` (الحرف الذي دليل الفهرسة الخاص به هو `pointer` وهو في هذه الحالة الحرف الأول من الرسالة لأن دليل الفهرسة هو 0) إلى نهاية السلسلة النصية ذات الرقم `col` داخل القائمة `ciphertext`

ثم نقوم بإضافة 8 (قيمة مفتاح التشفير) إلى قيمة `pointer` في كل مرة تتكرر فيها الحلقة (السطر 36)

- في أول دورة للحلقة حصلنا على الحرف `message[0]`

- في ثاني دورة للحلقة حصلنا على الحرف message[8]
- في ثالث دورة للحلقة حصلنا على الحرف message[16]
- في رابع دورة للحلقة حصلنا على الحرف message[24]

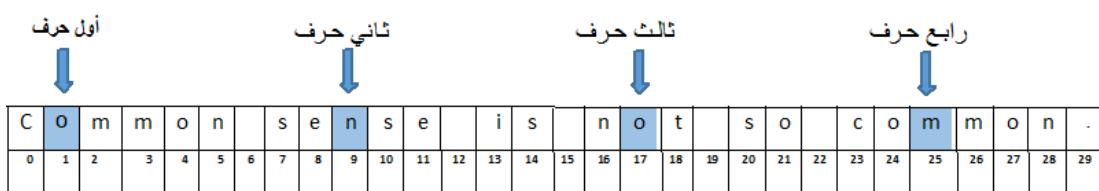
ويتم إضافة هذه الأحرف إلى نهاية السلسلة النصية ciphertext[col] (بما أن  $col = 0$  من أجل أول دورة للحلقة هذا يعني ciphertext[0])



الشكل السابق يظهر كل أحرف الرسالة مع دليل الفهرسة الخاص بكل حرف كما يظهر الأحرف التي تم اختيارها في أو دورة لحلقة for وعند جمع هذه الأحرف مع بعضها سوف نحصل على 'Ceno' وهي السلسلة النصية الأولى في القائمة ciphertext

```
>>> ciphertext = ['Ceno', 'onom', 'mstm', 'me o', 'nio.', ' s ', 's c']
>>> ciphertext[0]
'Ceno'
```

في الدورة الثانية لحلقة for تصبح قيمة المتغير  $col = 1$  (بدلاً من 0) وتصبح قيمة المتغير  $pointer = 1$  والآن وعند إضافة قيمة المفتاح (8) إلى pointer ضمن حلقة while في السطر 30 فسوف نحصل على قيم الفهرسة 1, 9, 17, 25



message[1], message[9], message[17], message[25]

هذه الأحرف هي مكونات السلسلة النصية الثانية في القائمة `ciphertext` وهي  
(وهو العمود الثاني) `ciphertext[1] = 'onom'`

عندما تنتهي حلقة `for` بعد أن تمر على كل العواميد سوف تصبح قيمة القائمة

```
ciphertext = ['Ceno', 'onom', 'me o', 'o sn', 'nio.', ' s', 's c']
```

سوف نستخدم الطريقة النصية `join()` من `string method` إلى سلسلة نصية واحدة.

## الطريقة النصية `:join()`

الطريقة `join()` تم استخدامها في السطر 39 من كود البرنامج.

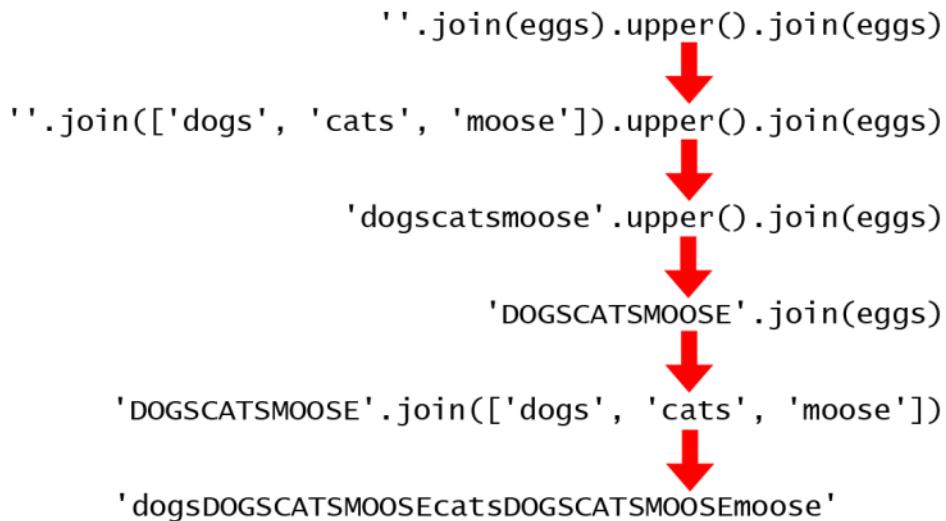
هذه الطريقة تأخذ قائمة من السلاسل النصية وتعيد سلسلة نصية وحيدة هي عبارة عن مجموع هذه السلاسل مع بعضها البعض.

استدعاء هذه الطريقة يجب أن يكون بين السلاسل والقائمة (سوف نستخدم سلسلة نصية فارغة من أجل الاستدعاء)

جرب المثال التالي:

```
>>> eggs = ['dogs', 'cats', 'moose']
>>> ''.join(eggs)
'dogscatsmoose'
>>> ' '.join(eggs)
'dogs cats moose'
>>> 'XYZ'.join(eggs)
'dogsXYZcatsXYZmoose'
>>> ''.join(eggs).upper().join(eggs)
'dogsDOGSCATSMSOOSEcatsDOGSCATSMSOOSEmoose'
>>>
```

مراحل تنفيذ العبارة الأخيرة `".join(eggs).upper().join(eggs)`



## إعادة القيم والتعليمات :return

```

38.     # Convert the ciphertext list into a single string value and return it.
39.     return ''.join(ciphertext)

```

استخدامنا للطريقة `(join)` هو استخدام بسيط وليس معقد كما في المثال السابق.

قمنا باستدعاء الطريقة `(join)` على سلسلة نصية فارغة ومررنا لها المتغير `ciphertext` ليتم جمع السلسل النصية الموجودة في هذا المتغير مع بعضها البعض (بدون أي فراغات بينها)

التابع أو الطريقة دائمًا يتم استدعائهما من أجل حساب قيمة وهذه القيمة تسمى القيمة المعادة (القيمة التي يعيدها التابع أو الطريقة) `return value`

قمنا بخلق التابع باستخدام التعليمة `def` كما قمنا باستخدام التعليمة `return` من أجل أخباره بالقيمة التي يجب عليه إعادتها.

التعليمة `return` تتبع باسم المتغير المراد إعادة قيمته أو يمكن أن تتبع بعبارة جبرية بدلاً من اسم المتغير وفي هذه الحالة يتم إعادة القيمة بعد حساب العبارة الجبرية.

افتح نافذة جديدة في محرر النصوص واتكتب الكود التالي وافظه باسم `addNumbers.py` واضغط F5 للتنفيذ:

### Source code for addNumbers.py

```
1. def addNumbers(a, b):
2.     return a + b
3.
4. spam = addNumbers(2, 40)
5. print(spam)
```

عند تنفيذ هذا البرنامج سوف نحصل على النتيجة التالية:

```
42
>>>
```

التابع addNumbers(2, 42) سوف يقوم بحساب القيمة 42 والتعليمية return سوف تقوم بإعادة هذه القيمة

### العودة إلى الكود:

```
38.     # Convert the ciphertext list into a single string value and return it.
39.     return ''.join(ciphertext)
```

التابع encryptMessage() يستخدم التعليمية return من أجل إعادة القيمة النصية التي تم تشكيلها من خلال دمج السلسل النصية الموجودة في القائمة ciphertext. القيمة المعادة هي النص المشفر.

### المتغير الخاص `__name__`

```
42. # If transpositionEncrypt.py is run (instead of imported as a module) call
43. # the main() function.
44. if __name__ == '__main__':
45.     main()
```

يمكننا تشغيل برنامجنا في الوحدة `module` باستخدام اسم المتغير الخاص `_name` عندما يعمل برنامج البايثون هناك متغير خاص `_name` (خطين قبل الاسم وخطين بعده) الذي يتم إسناد القيمة النصية '`main`' له وذلك قبل أول سطر من برنامجك.

في نهاية ملف السكريبت `script file` وبعد كل تعليمية `def` نقوم بكتابة بعض التعليمات التي تقوم بفحص فيما إذا كانت قيمة `_name` هي القيمة النصية '`main`' وإذا كان ذلك محققاً نقوم باستدعاء التابع.

التعليمية `if` في السطر 44 يتم تفزيذها كأول سطر من الكود عند الضغط على `F5` (طبعاً بعد تنفيذ تعليمية `import` في السطر 4 والتعليمية `def` في السطرين 6 and 21)

السبب لكتابتنا لهذا الكود بهذه الطريقة لأن البايثون يقوم بإسناد القيمة '`main`' إلى المتغير `_name` عند تشغيل البرنامج أو يقوم بإسناد القيمة '`'transpositionEncryp'` إلى المتغير `_name` إذا تم استيراد هذا البرنامج من قبل برامج أخرى.

بهذه الطريقة يستطيع البايثون معرفة فيما إذا كان هذه البرنامج بدأ العمل كبرنامج مستقل أو تم استيراده من قبل برامج أخرى كما في الوحدات `module`

بشكل مماثل للطريقة التي يقوم برنامجنا باستيراد الوحدة `pyperclip` من أجل أن نتمكن من استدعاء التوابع الموجودة في داخلها فهناك برامج يمكن أن تقوم باستيراد برنامجنا من خلال التعليمية `import transpositionEncrypt.py` `encryptMessage()`

عندما يتم تنفيذ التعليمية `import` فإن البايثون يقوم بالنظر إلى الملف (الوحدة) من `module` من خلال اللامقة '`.py`'. في نهاية اسم الملف (وهذه هو سبب أن التعليمية `import pyperclip` تقوم باستيراد الملف `(pyperclip.py)`

عندما يقوم البايثون باستيراد ملف فسوف يتم اسناد اسم هذا الملف إلى المتغير `_name` وثم يبدأ البرنامج بالعمل.

عندما يتم استيراد برنامجنا transpositionEncrypt.py نريد لكل تعليمات def أن تعمل (من أجل تعریف التابع encryptMessage() الذي نريد استدعاءه) ولكننا لا نريد استدعاء التابع main لأنه يقوم بتنفيذ كود التشفیر للسلسلة النصية

8 'Common sense is not so common.' باستخدام المفتاح

وهذه هو سبب وضعنا الكود داخلتابع ( التابع (main) ثم قمنا بإضافة آخر سطرين في البرنامج من أجل استدعاء هذا التابع، عندما نقوم بهذه الأمور فإن برماجنا يمكن أن يعمل كبرنامجه مستقل ويمكن أن يعمل أيضاً كوحدة module لبرنامجه آخر.

## حجم المفتاح وطول الرسالة:

لاحظ مَا يَحْدُثُ عِنْدَمَا يَكُونُ طُولُ الرِّسْلَةِ أَقْلَى مِنْ ضُعْفِيَّ المَفْتَاحِ

C o m m o n (s) s e n s e (s) i s (s) n o t (s) s o (s) c o  
m m o n .

عند استخدام المفتاح 25 لتشифر الرسالة 'Common sense is not so common.' فسوف نحصل على الرسالة المشفرة 'Cmommomno.n sense is not so co' يوجد جزء من الرسالة لم يتم تشفيره !

هذا يحدث عندما يكون حجم المفتاح أكبر من ضعفي طول الرسالة وفي هذه الحالة سوف يوجد على الأقل حرف وحيد في العامود وسوف يتم عرضه في الرسالة المشفرة كما هو في الرسالة الأصلية قبل عملية التشفير.

وبسبب ذلك فإن المفتاح في شيفرة التحويل يجب أن يكون أقل من نصف طول الرسالة المراد تشفيرها.

لتحمّل كلما كانت الرسالة أطول يوجد عدد أكبر من مفاتيح التشفير الممكنة لتشفيّرها باستخدام شيفرة التحويل.

## الخلاصة:

في هذا الفصل تعلمنا المزيد من أساسيات البرمجة بالإضافة إلى كتابة برنامج لشيفرة التحويل والتي تعتبر أكثر أماناً من شيفرة فيصر.

تعرفنا على التوابع والقوائم ومعاملات الزيادة، تذكر دائماً من أجل أن تفهم أي سطر من كود البرنامج فقط قم بحسابه خطوط بخطوة بنفس الطريقة التي يقوم بها الباليثون.

أصبح بإمكاننا ترتيب الكود الخاص بنا ضمن مجموعات تسمى التوابع `functions` والتي يتم خلقها باستخدام التعليمة `def` والقيم التي تكون بين أقواس التابع عند تعريفه تسمى البارامترات.

البارامترات هي متغيرات محلية وهي مختلفة عن المتغيرات الموجودة خارج جسم التابع والتي تسمى متغيرات عامة.

تذكر دائماً أن المتغيرات المحلية مختلفة عن المتغيرات العامة حتى ولو كان لها نفس الاسم القوائم هي نوع من المتغيرات والتي يمكن أن يخزن فيها عدة قيم أخرى ويمكن أن يتم تخزين قائمة ضمن قائمة أخرى.

العديد من العمليات التي يمكن القيام بها مع السلسل النصية (الفهرسة، التقطيع، التابع `(len)`) يمكن أيضاً استخدامها مع القوائم.

يمكن استخدام معاملات الزيادة من أجل اختصار التعليمات.

الطريقة النصية `(join)` تقوم بجمع عدة سلاسل نصية موجودة ضمن قائمة وتعيد سلسلة نصية وحيدة.

إذا لم تفهم مبادئ البرمجة في هذه الفصل بشكل جيد رجاءً قم بالعودة إلى بداية الفصل وأعد القراءة مرة ثانية.

في الفصل القادم سوف نتحدث عن فك التشفير في شيفرة التحويل.



## الفصل التاسع

### فك التشفير في شيفرة التحويل

#### التحول

---

محتوى هذا الفصل:

- عملية فك التشفير في شيفرة التحويل
- التوابع (math.ceil(), math.floor(), and round)
- المعاملات المنطقية (البوليانية) and , or
- جداول الحقيقة

“When stopping a terrorist attack or seeking to recover a kidnapped child, encountering encryption may mean the difference between success and catastrophic failures.”

Attorney General Janet Reno, September 1999

شكل مختلف عن شифرة قيسار فإن عملية فك التشفير في شифرة التحويل هو عملية صعبة في هذا الفصل سوف نقوم بكتابه البرنامج transpositionDecrypt.py والذي يقوم بعملية فك التشفير

## عملية فك التشفير في شيفرة التحويل على الورق:

لفترض أننا قمنا بإرسال النص المشفر 'Cenoonommstmme oo snnio. s s c' إلى صديق والذي يعرف مفتاح التشفير السري المستخدم في عملية التشفير وهو الرقم 8 أول خطوة يقوم بها هذا الصديق من أجل فك تشفير هذه الرسالة هو حساب عدد الصناديق التي يجب أن يقوم برسمها من أجل القيام بهذه المهمة يجب أن يقوم بتقسيم طول الرسالة المشفرة على قيمة المفتاح وتقرير النتيجة للعدد الصحيح التالي

في هذا المثال طول الرسالة المشفرة هو 30 حرفاً (نفس طول النص الصريح) و مفتاح التشفير هو الرقم 8 وبالتالي  $30/8 = 3.75$

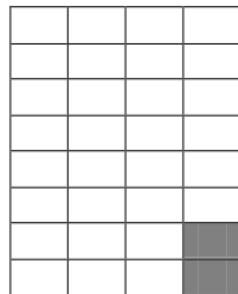
يتم تقرير النتيجة إلى العدد الصحيح التالي وهو 4 ، هذا يعني أنه يجب رسم شبكة من الصناديق تحتوي على أربع عواميد (الرقم الذي قمنا بحسابه) وثمانية أسطر (قيمة المفتاح)


انتبه في حالة كانت نتيجة التقسيم هو عدد صحيح فنأخذ العدد الناتج كما هو دون القيام بأي عملية تقرير

مثلاً  $30/5 = 6.0$  في هذه الحالة نأخذ الرقم 6 ولا نقوم بأي عملية تقرير الخطوة التالية هي حساب عدد الصناديق في أقصى اليمين والتي يجب تظليلها

عدد الصناديق الكلي هو 32 وطول الرسالة المشفرة هو 30 وبالتالي  $2 = 30 - 32$

يجب تظليل صندوقين في الجانب الأيمن



ثم تبدأ عملية ملئ الصناديق بالأحرف، كل حرف في صندوق حيث يتم البدء من الزاوية العلوية  
اليسرى ثم باتجاه اليمين كما في عملية التشفير

الرسالة المشفرة هي 'Cenoonomstmme oo snnio. s s c' لذلك يكون :

محتوى أول سطر هو 'Ceno'

محتوى ثاني سطر هو 'onom' وهذا

الفراغات يتم تمثيلها ب (s)

C	e	n	o
o	n	o	m
m	s	t	m
m	e	(s)	o
o	(s)	s	n
n	i	o	.
(s)	s	(s)	
s	(s)	c	

صديقنا الذي استلم هذه الرسالة المشفرة سوف يرها إذا قام بقراءة النص بشكل عمودي من كل  
عمود وسوف يحصل على النص الصريح لهذه الرسالة وهو

'Common sense is not so common.'

خطوات عملية فك التشفير:

- حساب عدد الأعمدة ويتم ذلك من خلال تقسيم طول الرسالة على المفتاح وتقرير  
النتيجة إلى العدد الصحيح التالي

٢. رسم شبكة من الصناديق بحيث يكون عدد الأعمدة هو العدد الذي قمنا بحسابه في الخطوة الأولى وعدد الأسطر هو نفس قيمة المفتاح
٣. حساب عدد الصناديق التي يجب تظليلها من خلال طرح عدد أحرف الرسالة من عدد الصناديق الكلي
٤. تظليل الصناديق التي تم حساب عددها في الخطوة الثالثة في الزاوية اليمنى السفلى
٥. ملي الصناديق بأحرف الرسالة المشفرة بدأً من أول سطر ومن اليسار إلى اليمين ويتم تجاهل الصناديق المظللة
٦. الحصول على النص الصريح من خلال البدء بالقراءة من العمود الأيسر ومن الأعلى إلى الأسفل ثم الانتقال إلى العمود التالي
- في حال استخدام مفتاح مختلف سوف يتم رسم عدد خاطئ من العواميد وحتى لو اتبعت باقي الخطوات بشكل صحيح فلن نحصل على النص الصريح إلا إذا استخدمت قيمة المفتاح الصحيحة

## برنامج فك التشفير لشيفرة التحويل:

افتح نافذة جديدة في محرر النصوص وأكتب الكود التالي ثم احفظه باسم  
transpositionDecrypt.py

## كود برنامج فك التشفير لشيفرة التحويل:

افتح نافذة جديدة من محرر النصوص File > New File واترك الكود التالي ثم قم بحفظه باسم  
transpositionDecrypt.py واضغط F5 من أجل التنفيذ

يجب أن تضع الملف (الوحدة) pyperclip.py في نفس المجلد الذي سوف تحفظه به ملف  
transpositionDecrypt.py

يمكنك الحصول على pyperclip.py من <http://invpy.com/pyperclip.py>

## Source code for transpositionDecrypt.py

```
1. # Transposition Cipher Decryption
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. import math, pyperclip
5.
6. def main():
7.     myMessage = 'Cenoonommstmme oo snnio. s s c'
8.     myKey = 8
9.
10.    plaintext = decryptMessage(myKey, myMessage)
11.
12.    # Print with a | (called "pipe" character) after it in case
13.    # there are spaces at the end of the decrypted message.
14.    print(plaintext + '|')
15.
16.    pyperclip.copy(plaintext)
17.
18.
19. def decryptMessage(key, message):
20.     # The transposition decrypt function will simulate the "columns" and
21.     # "rows" of the grid that the plaintext is written on by using a list
22.     # of strings. First, we need to calculate a few values.
23.
24.     # The number of "columns" in our transposition grid:
25.     numOfColumns = math.ceil(len(message) / key)
26.     # The number of "rows" in our grid will need:
27.     numRows = key
28.     # The number of "shaded boxes" in the last "column" of the grid:
29.     numOfShadedBoxes = (numOfColumns * numRows) - len(message)
30.
31.     # Each string in plaintext represents a column in the grid.

32.     plaintext = [''] * numOfColumns
33.
34.     # The col and row variables point to where in the grid the next
35.     # character in the encrypted message will go.
36.     col = 0
37.     row = 0
38.
39.     for symbol in message:
40.         plaintext[col] += symbol
41.         col += 1 # point to next column
42.
43.         # If there are no more columns OR we're at a shaded box, go back to
44.         # the first column and the next row.
45.         if (col == numOfColumns) or (col == numOfColumns - 1 and row >=
numOfRows - numOfShadedBoxes):
46.             col = 0
47.             row += 1
48.
49.     return ''.join(plaintext)
50.
51.
52. # If transpositionDecrypt.py is run (instead of imported as a module) call
53. # the main() function.
54. if __name__ == '__main__':
55.     main()
```

نتيجة تنفيذ هذا البرنامج هي:

```
Common sense is not so common.|  
>>>
```

إذا كنت تريده فك تشفير رسالة مختلفة أو القيام بعملية فك التشفير باستخدام قيمة مفتاح مختلفة يمكنك تغيير قيم المتغيران myMessage, myKey في السطرين 5 and 6

## كيف يعمل هذا البرنامج:

```
transpositionDecrypt.py  
1. # Transposition Cipher Decryption  
2. # http://inventwithpython.com/hacking (BSD Licensed)  
3.  
4. import math, pyperclip  
5.  
6. def main():  
7.     myMessage = 'Cenoonommstmme oo snnio. s s c'  
8.     myKey = 8  
9.  
10.    plaintext = decryptMessage(myKey, myMessage)  
11.  
12.    # Print with a | (called "pipe" character) after it in case  
13.    # there are spaces at the end of the decrypted message.  
14.    print(plaintext + '|')  
15.  
16.    pyperclip.copy(plaintext)
```

أول جزء من هذا البرنامج شبيه جداً بالجزء الأول من برنامج transpositionEncrypt.py

الوحدة pyperclip والوحدة math يتم استيرادهم باستخدام التعليمية import

يتم خلق التابع main() والذي يحوي على بارامترین myMessage, myKey ثم يتم استدعاء decryptMessage()

القيمة التي يعيدها هذا التابع هي النص الصريح والتي يتم بعدها طباعتها على الشاشة (بعد الرسالة يتم طباعة الرمز '|') ويتم نسخ هذا النص إلى الذاكرة

```
19. def decryptMessage(key, message):
```

انظر إلى خطوات عملية فك التشفير في بداية هذا الفصل  
 مثلاً إذا كنت تريد فك تشفير الرسالة 'Cenoonomstmme oo snnio. s s c' (المكونة من 30 حرفاً) باستخدام المفتاح 8 فيجب عليك رسم وملئ الصناديق كما في الشكل التالي:

C	e	n	o
o	n	o	m
m	s	t	m
m	e	(s)	o
o	(s)	s	n
n	i	o	.
(s)	s	(s)	
s	(s)	c	

التابع (`decryptMessage()`) يقوم بخطوات عملية فك التشفير

## التابع (`:math.ceil`, `math.floor()`, `round()`)

عندما تقوم بتقسيم الأرقام باستخدام معامل القسمة '`/`' فإن النتيجة التي سيتم إعادتها هي عدد حقيقي ذو فاصلة عشرية floating point number حتى ولو كانت نتيجة القسم عدد بدون باقي.

مثلاً:  $21 / 7 = 3.0$  وليس 3

```
>>> 21 / 7
3.0
```

وفي حال كانت نتيجة القسمة تحوي على باقي ستظهر النتيجة رقم مع فاصلة عشرية كما في المثال التالي:

$22 / 5 = 4.4$

```
>>> 22 / 5
4.4
```

إذا حصلت على نتيجة  $4 / 5 = 0.4$  هذا يعني أنك تستخدم Python 2 وليس Python 3

اذهب إلى <http://python.org> وقم بتحميل وتنصيب Python 3

إذا أردت تقرير أو تدوير الرقم إلى أقرب عدد صحيح يمكنك استخدام التابع `round()`

جرب المثال التالي:

```
>>> round(4.2)
4
>>> round(4.5)
4
>>> round(4.9)
5
>>> round(5.0)
5
>>> round(22 / 5)
4
>>>
```

إذا كنت تريد التقرير إلى العدد التالي يمكنك استخدام التابع `math.ceil()` والذي هو اختصار الكلمة 'ceiling' والتي تعني سقف أو حد أعلى

إذا كنت تريد التقرير إلى العدد الأقل يمكنك استخدام التابع `math.floor()`

هذه التوابع موجودة في الوحدة `math module` لذلك يجب أن تقوم باستيراد هذه الوحدة لكي تستطيع استدعاء هذه التوابع.

جرب المثال التالي:

```
>>> import math
>>> math.floor(4.0)
4
>>> math.floor(4.2)
4
>>> math.floor(4.9)
4
>>> math.ceil(4.0)
4
>>> math.ceil(4.2)
5
>>> math.ceil(4.9)
5
>>>
```

التابع `(math.ceil)` يقوم بالخطوة الأولى من خطوات عملية فك التشفير لشيفرة التحويل.

```
transpositionDecrypt.py
19. def decryptMessage(key, message):
20.     # The transposition decrypt function will simulate the "columns" and
21.     # "rows" of the grid that the plaintext is written on by using a list
22.     # of strings. First, we need to calculate a few values.
23.
24.     # The number of "columns" in our transposition grid:
25.     numOfColumns = math.ceil(len(message) / key)
26.     # The number of "rows" in our grid will need:
27.     numOfRows = key
28.     # The number of "shaded boxes" in the last "column" of the grid:
29.     numOfShadedBoxes = (numOfColumns * numOfRows) - len(message)
```

في السطر 25 يتم حساب عدد الأعمدة (الخطوة الأولى في عملية فك التشفير) من خلال قسم طول الرسالة `len(message)` على قيمة المفتاح ثم يتم تمرير النتيجة إلى التابع `(math.ceil)` . ويتم حفظ النتيجة النهائية في المتغير `.numOfColumns`

في السطر 27 يتم حساب عدد الأعمدة (الخطوة الثانية في عملية فك التشفير) وهو عدد صحيح مساوي لقيمة المفتاح ويتم حفظ هذه القيمة في المتغير `.numOfRows` .

في السطر 29 يتم حساب عدد الصناديق المظللة (الخطوة الثالثة في عملية فك التشفير) وهو عدد الصناديق الكلي (عدد الأسطر ضرب عدد الأعمدة) منقوصاً منه طول الرسالة.

```
transpositionDecrypt.py
31.     # Each string in plaintext represents a column in the grid.
32.     plaintext = [''] * numOfColumns
```

المتغير `plaintext` هو قائمة تحوي على عدد من السلسلات النصية وهذه السلسلات تبدأ فارغة ثم يتم ملئ كل سلسلة بأحرف كل عمود

استخدام عملية الضرب سوف يؤدي لتكرار السلسلة النصية الفراغة بحسب عدد الأعمدة.

تذكر أنه عند استدعاء أي تابع سوف يتم خلق مجال محلي خاص به والمتغير `plaintext` موجود بداخل التابع `decryptMessage()` وهذا يعني أن المتغير `plaintext` هو متغير محلي وهو مختلف عن المتغير الذي له نفس الاسم `(main)` الموجود ضمن التابع `(main)`

شبكة الصناديق في مثالنا لها الشكل التالي:

C	e	n	o
o	n	o	m
m	s	t	m
m	e	(s)	o
o	(s)	s	n
n	i	o	.
(s)	s	(s)	
s	(s)	c	

المتغير `plaintext` هو قائمة تحوي على مجموعة من السلاسل النصية وكل سلسلة نصية مخصصة لعمود واحد وستكون قيمة المتغير `plaintext` كالتالي:

```
>>> plaintext = ['Common s' , 'ense is ' , 'not so c' , 'ommon.']
>>> plaintext[0]
'Common s'
>>>
```

يمكنا استدعاء الطريقة `join()` لتقوم بجمع هذه السلاسل النصية مع بعضها البعض لنجعل  
'Common sense is not so common.' على سلسلة نصية واحدة وهي

```
transpositionDecrypt.py
34.     # The col and row variables point to where in the grid the next
35.     # character in the encrypted message will go.
36.     col = 0
37.     row = 0
38.
39.     for symbol in message:
```

المتغيران `col` and `row` سوف يحددان السطر والعمود الذي سيكون فيه الحرف التالي  
تبدأ قيم المتغيران بالرقم 0 وفي السطر 39 نستخدم حلقة `for` من أجل المرور على أحرف  
الرسالة الموجودة في المتغير `message` وبداخل هذه الحلقة سوف يتم كتابة كود يقوم بتعديل  
قيم المتغيران `col` and `row` وسوف نستمر بهذه العملية إلى أن نحصل على السلسلة النصية  
الصحيحة التي يتم حفظها في المتغير من نوع قائمة `plaintext`

```
transpositionDecrypt.py
40.         plaintext[col] += symbol
41.         col += 1 # point to next column
```

أول خطوة في داخل حلقة `for` هي إضافة الرمز `symbol` إلى السلسلة النصية ذات قيمة  
الفهرسة `col` داخل القائمة `plaintext`

ثم نقوم بزيادة قيمة `co1` بمقدار واحد وذلك في السطر 41 وبهذه الطريقة سوف يتم إضافة باقي الرموز من خلال تكرار الحلقة

## المعاملات المنطقية :`and` , `or`

المعاملات المنطقية `or` , `and` و`while` تساعدنا في الشروط المعقّدة الخاصة بتعليمات

المعامل `and` يوضع بين عبارتين جبريتين ويعيد القيمة `True` إذا كانت كلا العبارتين الجبريتين محققتين.

المعامل `or` يوضع بين عبارتين جبريتين ويعيد القيمة `True` إذا كانت إحدى العبارتين محققة.

جرب المثال التالي في الشيل التفاعلي:

```
>>> 10 > 5 and 2 < 4
True
>>> 10 > 5 and 4 != 4
False
>>>
```

العملية الأولى تعيد القيمة `True` لأن كلا العبارتين بجانب المعامل `and` محققتين

`True and True = True`

العملية الثانية تعيد القيمة `False` لأن العبارة الأولى محققة والثانية غير محققة

`True and False = False`

جرب المثال التالي:

```
>>> 10 > 5 or 4 != 4
True
>>> 10 < 5 or 4 != 4
False
>>>
```

المعامل `or` يعيد القيمة `True` إذا كانت إحدى القيم على جانبه على الأقل هي `True` ويعيد `False` إذا كانت كلتا القيمتين على جانبيه هي `False`.

المعامل المنطقي `not` يقوم بعكس القيمة المنطقية

`not True = False`

`not False = True`

جرب المثال التالي:

```
>>> not 10 > 5
False
>>> not 10 < 5
True
>>> not False
True
>>> not not False
False
>>> not not not not not False
True
>>>
```

## جداول الحقيقة:

إذا نسيت كيفية عمل المعاملات المنطقية يمكن النظر إلى هـ الجداول والتي تسمى جداول الحقيقة:

---

جدول الحقيقة للمعامل `and`

A	and	B	النتيجة
True	and	True	<b>True</b>
True	and	False	<b>False</b>
False	and	True	<b>False</b>
False	and	False	<b>False</b>

جدول الحقيقة للمعامل **or**

A	or	B	النتيجة
True	or	True	<b>True</b>
True	or	False	<b>True</b>
False	or	True	<b>True</b>
False	or	False	<b>False</b>

جدول الحقيقة للمعامل **not**

not A	النتيجة
not True	False
not False	True

المعامل **and** يمكن أن يستخدم من أجل اختصار تعليمات **if**

جرب المثال التالي:

```
>>> if 10 > 5:  
...     if 2 < 4:  
...         print('Hello!')  
  
Hello!  
>>>  
>>> if 10 > 5 and 2 < 4:  
...     print('Hello!')  
  
Hello!  
>>>
```

كما يمكننا أيضاً استخدام المعامل **or** بدلاً من التعليمات **if , elif**

جرب المثال التالي:

```

>>> if 4 != 4:
    print('Hello!')
elif 10 > 5:
    print('Hello!')

Hello!
>>>
>>> if 4 != 4 or 10 > 5:
    print('Hello!')

Hello!
>>>

```

## ترتيب العمليات في المعاملات المنطقية:

كما في العمليات الرياضية فإن المعاملات المنطقية and, or, not لها ترتيب لتنفيذ العمليات

العملية ذات الأولوية العليا هي or ثم not ثم and

جرب المثال التالي:

```

>>> not False and True
True
>>> not (False and True)
True
>>>

```

## العودة إلى البرنامج:

```

transpositionDecrypt.py
43.     # If there are no more columns OR we're at a shaded box, go back to
44.     # the first column and the next row.
45.     if (col == numColumns) or (col == numColumns - 1 and row >=
numRows - numShadedBoxes):
46.         col = 0
47.         row += 1

```

يوجد حالتين نريد عندهما ضبط قيمة  $col = 0$

الحالة الأولى: إذا أزدانت قيمة `col` وأصبحت أكبر من آخر دليل فهرسة للقائمة `plaintext` وفي هذه الحالة فإن القيمة في المتغير `col` سوف تساوي `numOfColumns` (تذكر أن آخر دليل فهرسة للقائمة `plaintext` هو `numOfColumns`)

الحالة الثانية: إذا كان `col` له قيمة آخر دليل فهرسة و المتغير `row` كان يشير إلى الصناديق المظللة في آخر عمود

عندما فإن كامل شبكة فك التشفير مع دليل الفهرسة لكل حرف ستكون بالشكل التالي:

	0	1	2	3
0	C 0	e 1	n 2	o 3
1	o 4	n 5	o 6	m 7
2	m 8	s 9	t 10	m 11
3	m 12	e 13	(s) 14	o 15
4	o 16	(s) 17	s 18	n 19
5	n 20	i 21	o 22	.
6	(s) 24	s 25	(s) 26	
7	s 27	(s) 28	c 29	

كما ترى فإن الصناديق المظللة موجودة في آخر عمود (عندما يكون دليل الفهرسة هو 6 and 7 -1) وفي السطرين `numOfColumns` -1

قمنا باستخدام العبارة الجبرية `row >= numberOfRows - numShadedBoxes`

من أجل حساب دليل الفهرسة للأسطر التي تحوي على الصناديق المظللة

فإذا كانت هذه العبارة محققة `True` وكان `col = numOfColumns -1` عندما نقوم بإعادة ضبط قيمة `col` لتعود إلى القيمة 0

الشرط الخاص بتعليمه `if` هو

`(col == numOfColumns) or (col == numOfColumns -1 and row >= (numberOfRows - numShadedBoxes))`

إذا كان هذا الشرط محق سوف يقوم البرنامج بتنفيذ كتلة التعليمات التالية وهي تغير قيمة `col` ليعود لأول عمود من خلال إعادة تعين قيمته لتصبح مساوية لقيمة صفر وزيادة القيمة الموجودة في المتغير `row` بمقدار واحد

```
49.     return ''.join(plaintext)
```

transpositionDecrypt.py

عندما تنتهي حلقة `for` (التي بدأت في السطر 39) من الدوران على كل الأحرف الموجودة في المتغير `message` ستكون قيم السلسلة النصية الموجودة في القائمة `plaintext` هي أحرف السلسلة النصية بعد عملية فك التشفير (طبعاً إذا تم استخدام مفتاح التشفير الصحيح) السلسلة النصية الموجودة في القائمة `plaintext` يتم جمعها مع بعضها باستخدام الطريقة النصية `join()` والسلسلة النصية الناتجة هي القيمة التي سوف يعيدها التابع `decryptMessage()`

في هذا المثال فإن القائمة `plaintext` ستكون كالتالي:

```
Plaintext = ['Common s' , 'ense is ' , 'not so c' , 'ommon.]
```

والتعليمية `'.join(plaintext)` سوف تعيد السلسلة النصية التالية:

'Common sense is not so common.'

```
52. # If transpositionDecrypt.py is run (instead of imported as a module) call  
53. # the main() function.  
54. if __name__ == '__main__':  
55.     main()
```

transpositionDecrypt.py

أول سطر سوف يقوم البرنامج بتنفيذه بعد استيراد الوحدات وتنفيذ التعليمات `def` هو السطر 54 حيث نقوم بفحص فيما إذا كان هذا البرنامج يعمل بشكل مستقل أو تم استيراده من قبل برنامج آخر من خلال فحص قيمة المتغير الخاص `__name__` إذا كانت مساوية لقيمة `'__main__'` إذا كان هذا محق عندها يتم تنفيذ التابع `main()`

## الخلاصة:

من أجل برنامج فك التشفير قمنا بوضع معظم الكود داخل التابع (`decryptMessage()`)  
برامجنا يمكنها تشفير وفك تشفير الرسالة.'Common sense is not so common.'  
باستخدام المفتاح ذو القيمة 8 ولكن يجب أن نجرب رسائل مختلفة وقيمة مفتاح تشفير مختلفة  
ورؤية فيما إذا كانت النتيجة هي نفس الرسالة الأصلية  
يمكننا تغيير قيمة المتغيرات `key` , `message` في كل من البرنامجين  
`transpositionEncrypt.py` and `transpositionDecrypt.py`  
البرنامجين لمعرفة إذا كانوا يعملان أو لا ولكن بدل ذلك سنقوم بكتابة برنامج يقوم بهذه العملية  
بشكل اوتوماتيكي



## الفصل العاشر

# تشفير وفك تشفير الملفات

---

محتوى هذا الفصل:

- قراءة وكتابة الملفات
- التابع `open()`
- الطريقة `read()`
- الطريقة `close()`
- الطريقة `write()`
- التابع `os.path.exists()`
- الطريقة النصية `statwith()`
- الطريقة النصية `title()`
- الوحدة `time` والتابع `time()`

إلى الآن فإن برامجاً تعمل فقط مع رسائل قصيرة تقوم بكتابتها بشكل مباشر في الكود البرمجي  
للبرنامِج كسلسلة نصية

برنامِج التشفير في هذا الفصل سوف يستخدم شيفرة التحويل transposition cipher من أجل تشفير وفك تشفير ملفات كاملة والتي يمكن أن تحوي على ملايين الأحرف والكلمات.

## ملفات النص الصريح:

البرنامِج في هذا الفصل سوف يقوم بتشифر وفك تشفير ملفات تحوي على النص الصريح plain text files وهذا النوع من الملفات يحوي فقط على بيانات نصية وغالباً ما يكون امتداد هذه الملفات هو .txt.

الملفات التي تنتج من برامج المعالجة التي تسمح لك بتغيير الخط واللون وحجم النص هي ليست ملفات نص صريح.

يمكنك كتابة الملفات النصية في نظام التشغيل ويندوز باستخدام محرر النصوص مثل Notepad أو في نظام التشغيل لينكس باستخدام gedit أو باستخدام أي محرر نصوص آخر كما يمكنك استخدام IDLE كمحرر نصوص ومن خلاله تقوم بحفظ الملفات بأمتداد .txt. بدل من استخدام الامتداد .py.

يمكنك تحميل بعض الأمثلة لملفات نصية من موقع هذا الكتاب:

- <http://invpy.com/devilsdictionary.txt>
- <http://invpy.com/frankenstein.txt>
- <http://invpy.com/siddhartha.txt>
- <http://invpy.com/thetimemachine.txt>

بعد تحميل الملف النصي الموجود في الرابط <http://invpy.com/frankenstein.txt> قم بفتح الملف النصي من خلال برنامج محرر النصوص.

هذه الملف يحوي على أكثر من 78,000 كلمة وطبعاً هذا سوف يأخذ وقت طويل جداً من أجل كتابة هذه الكلمات في برنامج التشفير ولكن إذا كانت هذه الكلمات موجودة في ملف نصي فإن البرنامج يستطيع قراءة هذا الملف والقيام بعملية التشفير بأقل من ثانيتين.

إذا حصلت على رسالة خطأ مشابهه للرسالة التالية

```
"UnicodeDecodeError: 'charmap' codec can't decode byte 0x90 in  
position 148: character maps to <undefined>"
```

هذا بسبب أنك تقوم بتشغيل برنامج التشفير على ملف نص نصي غير صريح (مشفر)

non-plain text file

## ال הקוד البرمجي لبرنامج تشفير الملفات بشيفرة التحويل:

Source Code of the Transposition File Cipher Program

بشكل مشابه لبرنامج شيفرة التحويل فإن برنامج تشفير الملفات بشيفرة التحويل سوف يقوم باستيراد الملفات transpositionEncrypt.py and transpositionDecrypt.py لنتتمكن من استخدام التوابع encryptMessage() and decryptMessage() الموجودة في داخلهما.

وبهذه الطريقة لن نقوم بحاجة لإعادة كتابة الكود البرمجي الخاص بهذه التوابع في برنامجنا الجديد.

قم بفتح ملف جديد في محرر النصوص من خلال File > New File وقم بكتابة الكود التالي ثم قم بحفظه باسم transpositionFileCipher.py واضغط F5 من أجل تشغيل البرنامج.

يجب أن تقوم بتحميل الملف النصي Frankenstein.txt ووضعه في نفس المجلد الذي يحوي على ملف transpositionFileCipher.py

يمكنك تحميل هذا الملف النصي من <http://invpy.com/frankenstein.txt>

### Source code for transpositionFileCipher.py

```
1. # Transposition Cipher Encrypt/Decrypt File
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. import time, os, sys, transpositionEncrypt, transpositionDecrypt
5.
6. def main():
7.     inputFilename = 'frankenstein.txt'
8.     # BE CAREFUL! If a file with the outputFilename name already exists,
9.     # this program will overwrite that file.
10.    outputFilename = 'frankenstein.encrypted.txt'
11.    myKey = 10
12.    myMode = 'encrypt' # set to 'encrypt' or 'decrypt'
13.
14.    # If the input file does not exist, then the program terminates early.
15.    if not os.path.exists(inputFilename):
16.        print('The file %s does not exist. Quitting...' % (inputFilename))
17.        sys.exit()
18.
19.    # If the output file already exists, give the user a chance to quit.
20.    if os.path.exists(outputFilename):
21.        print('This will overwrite the file %s. (C)ontinue or (Q)uit?' % (outputFilename))
22.        response = input('> ')
23.        if not response.lower().startswith('c'):
24.            sys.exit()
25.
26.    # Read in the message from the input file
27.    fileObj = open(inputFilename)
28.    content = fileObj.read()
29.    fileObj.close()
30.
31.    print('%sing...' % (myMode.title()))
32.
33.    # Measure how long the encryption/decryption takes.
34.    startTime = time.time()

35.    if myMode == 'encrypt':
36.        translated = transpositionEncrypt.encryptMessage(myKey, content)
37.    elif myMode == 'decrypt':
38.        translated = transpositionDecrypt.decryptMessage(myKey, content)
39.    totalTime = round(time.time() - startTime, 2)
40.    print('%sion time: %s seconds' % (myMode.title(), totalTime))
41.
42.    # Write out the translated message to the output file.
43.    outputFileObj = open(outputFilename, 'w')
44.    outputFileObj.write(translated)
45.    outputFileObj.close()
46.
47.    print('Done %sing %s (%s characters).' % (myMode, inputFilename,
48. len(content)))
49.    print('Used file is %s.' % (myMode.title(), outputFilename))
50.
51. # If transpositionCipherFile.py is run (instead of imported as a module)
52. # call the main() function.
53. if __name__ == '__main__':
54.     main()
```

في المجلد الذي يحوي على الملفات Frankenstein.txt and transpositionFileCipher.py سوف يتم خلق ملف جديد بعد تنفيذ البرنامج وله الاسم Frankenstein.rncrypted.txt والذي يحوي على المحتوى المشفر للنص الموجود فيFrankenstein.txt

من أجل القيام بعملية فك التشفير قم بعجراء التعديلات التالية على الكود البرمجي (الكلمات المكتوبة بالخط العربي) ثم قم بتشغيل البرنامج مرة ثانية

```
transpositionFileCipher.py
7.     inputFilename = 'frankenstein.encrypted.txt'
8.     # BE CAREFUL! If a file with the outputFilename name already exists,
9.     # this program will overwrite that file.
10.    outputFilename = 'frankenstein.decrypted.txt'
11.    myKey = 10
12.    myMode = 'decrypt' # set to 'encrypt' or 'decrypt'
```

عندما تقوم بتشغيل البرنامج بعد إجراء هذا التعديل سوف يظهر في نفس المجلد ملف باسم Frankenstein.decrypted.txt والذي يحوي على النص الأصلي الموجود في الملف Frankenstein.txt

عندما تقوم بتشغيل برنامج التشفير السابق سوف تحصل على الخرج التالي:

```
Encrypting...
Encryption time: 1.21 seconds
Done encrypting frankenstein.txt (441034 characters).
Encrypted file is frankenstein.encrypted.txt.
```

الملف Frankenstein.encrypted.txt يتم خلقه في نفس المجلد الذي يحوي على ملف transpositionFileCipher.py

اذا قمت بفتح هذا الملف باستخدام محرر النصوص سوف ترى النص المشفر لمحتوى الملف Frankenstein.txt

يمكنك الآن إرسال هذا النص المشفر إلى أي شخص عبر الايميل ليقوم هو بعملية فك التشفير وقراءة نص الرسالة.

## القراءة من الملفات:

حتى الآن كل دخل نريد أن نعطيه للبرنامج هو دخل يقوم المستخدم بكتابته  
برامج باليثون يمكنها فتح وقراءة الملفات بشكل مباشر من القرص الصلب  
يوجد ثلاث خطوات من أجل قراءة محتوى أي ملف:

١. فتح الملف
٢. قراءة المحتوى عند طريق اسناده إلى متغير
٣. إغلاق الملف

## التابع (:file objects و open()

أول بارامتر يتم تمريره للتابع (`open`) هو اسم الملف المراد فتحه.  
إذا كان الملف في نفس المجلد الذي يحوي على برنامج الباليثون يكفي فقط كتابة اسم الملف أو  
يمكنك أيضاً كتابة المسار الذي يحوي على الملف مثل  
'c:\\\\Python32\\\\frankenstein.txt'  
أو '/usr/foobar/frankenstein.txt'  
التابع (`open`) يعيد قيمة من نوع البيانات "file object" ويوجد عدة طرق من أجل قراءة هذه  
القيمة أو الكتابة فيها أو إغلاق الملف.

## الطريقة (:File Object الخاصة ب read())

الطريقة (`read()`) تقوم بإعادة سلسلة نصية تحوي على كل النص الموجود داخل الملف،  
لفترض أن الملف النصي `spam.txt` يحوي على النص التالي "Hello world!" (يمكنك  
خلق هذا الملف بنفسك) وعند تشغيل الكود التالي في الشيل التفاعليه (هذا الكود يفترض أنك  
تعمل على نظام التشغيل ويندوز وأنك قمت بخلق ملف نصي باسم `spam.txt` في المسار \c:\

```
>>> fo = open("c:\\spam.txt.txt", 'r')
>>> content = fo.read()
>>> print(content)
```

نتيجة تنفيذ الكود السابق هي:

```
Hello World
>>>
```

إذا كان الملف النصي يحوي على أكثر من سطر، فإن السلسلة النصية التي المُعادة من الطريقة `read()` سوف تحوي على `\n` للدلالة على نهاية كل سطر

عندما تحاول طباعه سلسلة نصية بأكثر من سطر يمكنك استخدام `\n` كما في المثال التالي:

```
>>> print('Hello\\nworld')
Hello
world
>>>
```

إذا حصلت على رسالة الخطأ التالية:

```
"IOError: [Errno 2] No such file or directory"
```

قم بالتأكد من اسم الملف أو المسار الذي يحوي على هذا الملف.

## الطريقة `close()` الخاصة بـ `File Object`

بعد قراءة محتوى الملف وإسناده إلى متغير، يمكنك اعلام البايثون بأنك قد انتهيت من هذا الملف من خلال استدعاء الطريقة `close()`

```
>>> fo.close()
>>>
```

البايثون يقوم بشكل اوتوماتيكي بإغلاق أي ملف مفتوح عند انتهاء البرنامج الذي يستخدم هذا الملف وعندما تريد القراءة من هذا الملف مرة ثانية يجب أن تقوم بإغلاق الملف ومن ثم استدعاء التابع `open()` مرة ثانية.

الكود التالي هو من برنامج التشفير باستخدام شيفرة التحويل وهو يقوم بقراءة محتوى الملف الذي يكون اسمه مخزن في المتغير `inputFilename`

transpositionFileCipher.py

```
26.     # Read in the message from the input file
27.     fileObj = open(inputFilename)
28.     content = fileObj.read()
29.     fileObj.close()
```

## الكتابة في الملفات:

لقد قمنا بقراءة الملف الأصلي، الآن سوف نقوم بكتابة بكتابه النص المشفر في ملف آخر وذلك من خلال استخدام الطريقة `write()`

أو يمكنك القيام بنفس العملية من خلال فتح الملف باستخدام نمط الكتابة `'w'` بدلاً من نمط القراءة `'r'` كما في المثال التالي:

```
>>> fo = open("c:\\spam.txt", 'w')
>>>
```

بالإضافة إلى نمطي القراءة والكتابة يوجد أيضاً نمط الإضافة `"append"` وهو مشابة لنمط الكتابة باستثناء أنه في نمط الإضافة فإن أي سلسلة نصية يتم كتابتها في الناف سوف تضاف إلى نهاية محتوى الحالي للملف.

نمط الإضافة لا يقوم بالكتابة فوق المحتوى الحالي للملف.

من أجل فتح الملف في نمط الإضافة قم بكتابة `'a'` كبرامتر ثاني للطريقة `open()`

بشكل مماثل لنمط القراءة عندما كنا نقوم بكتابه `'r'` من أجل فتح الملف في نمط القراءة.

وفي حال عدم كتابة أي حرف كبرامتر ثاني في `(open())` ستام فتح الملف بنمط القراءة بشكل افتراضي.

## الطريقة `(file object) writr()` الخاصة بـ

يمكنك كتابة نص في داخل الملف من خلال استدعاء الطريقة `write()` method الخاصة بـ `file object`

ولكن يجب أن تقوم بفتح الملف أولاً بنمط الكتابة وإلا سوف تحصل على رسالة الخطأ التالية

```
"io.UnsupportedOperation: not readable"
```

الطريقة `(write()` تأخذ بaramتر واحد وهو السلسلة النصية التي تريد كتابتها داخل الملف السطور من 43 إلى 45 هي لفتح الملف في نمط الكتابة والكتابة في الملف ومن ثم إغلاق هذا الملف.

```
42.     # Write out the translated message to the output file.  
43.     outputFileObj = open(outputFilename, 'w')  
44.     outputFileObj.write(translated)  
45.     outputFileObj.close()
```

الآن وبعد أن تعرفنا على أساسيات القراءة والكتابة في الملفات لنعد إلى الكود البرمجي لبرنامج تشفير الملفات بشيفرة التحويل.

## كيف يعمل هذا البرنامج:

```
transpositionFileCipher.py  
1. # Transposition Cipher Encrypt/Decrypt File  
2. # http://inventwithpython.com/hacking (BSD Licensed)  
3.  
4. import time, os, sys, transpositionEncrypt, transpositionDecrypt  
5.  
  
6. def main():  
7.     inputFilename = 'frankenstein.txt'  
8.     # BE CAREFUL! If a file with the outputFilename name already exists,  
9.     # this program will overwrite that file.  
10.    outputFilename = 'frankenstein.encrypted.txt'  
11.    myKey = 10  
12.    myMode = 'encrypt' # set to 'encrypt' or 'decrypt'
```

في السطر الرابع استخدمنا التعليمة `import` من أجل استيراد البرامج `transpositionEncrypt.py` and `transpositionDecrypt.py` ومن أجل استيراد الموديولات `time, os` and `sys module` تعريف التابع `(main()` والذي سوف يقوم بتنفيذ كل العمليات في البرنامج عند استدعاءه.

المتغير `inputFilename` يحوي على سلسلة نصية وهي اسم الملف الذي سيتم القراءة النص منه ومن ثم يتم كتابة النص بعد عملية التشفير أو فك التشفير في الملف `outputFilename` شيفرة التحويل تستخدم عدد صحيح وهي قيمة مفتاح التشفير ويتم تخزينها في المتغير `myKey` إذا تم تخزين القيمة 'encrypt' في المتغير `myMode` فإن البرنامج سوف يقوم بتشифر محتوى الملف الموجود في `inputFilename` وإذا تم تخزين القيمة 'decrypt' في المتغير `myMode` فإن البرنامج سوف يقوم بفك تشفير محتوى الملف الموجود في `inputFilename`

## التابع (`:os.path.exists()`)

قراءة الملفات هو أمر طبيعي وغير مؤذٍ ولكن يجب أن تكون حذرين عند الكتابة في الملفات. إذا قمنا باستدعاء التابع ()`open` في نمط الكتابة وكان اسم الملف موجود مسبقاً فإن الملف الأول سيتم حذفه من أجل استخدام هذا الاسم للملف الثاني. ويمكن أن يتم بالصدفة حذف ملف مهم إذا قمنا بتمرير اسمه إلى التابع ()`open`. استخدام التابع ()`os.path.exists` يمكننا من فحص فيما إذا كان يوجد ملف بنفس الاسم. التابع ()`os.path.exists` يقبل بaramتر واحد وهو اسم الملف ويقوم بإعادة `True` إذا كان يوجد ملف بنفس الاسم و `False` إذا لم يكن يوجد ملف بنفس الاسم. التابع ()`os.path.exists` موجود في الموديول `path module` والتي هي موجودة في الموديول `os module` وعند استيراد الموديول `os` فإن الموديول `path` سيتم استيراده بشكل تلقائي أيضاً. جرب المثال التالي في الشيل التفاعلية:

```
>>> import os  
>>> os.path.exists('abcdef')  
False
```

```
>>> os.path.exists('C:\\Windows\\System32\\calc.exe')
True
>>>
```

بالتأكيد سوف تحصل على هذه النتيجة في حال كنت تستخدم البيايثون على نظام ويندوز لأن الملف calc.exe غير موجود في نظام لينكس.

```
transpositionFileCipher.py
14.     # If the input file does not exist, then the program terminates early.
15.     if not os.path.exists(inputFilename):
16.         print('The file %s does not exist. Quitting...' % (inputFilename))
17.         sys.exit()
```

قمنا باستخدام التابع os.path.exists() من أجل فحص فيما إذا كان اسم الملف الموجود في المتغير inputFilename موجود مسبقاً وإلا سيتم عرض رسالة للمستخدم والخروج كمن البرنامج.

## الطرق النصية :startswith() and endswith()

```
transpositionFileCipher.py
19.     # If the output file already exists, give the user a chance to quit.
20.     if os.path.exists(outputFilename):
21.         print('This will overwrite the file %. (C)ontinue or (Q)uit?' %
(outputFilename))
22.         response = input('> ')
23.         if not response.lower().startswith('c'):
24.             sys.exit()
```

إذا كان الملف الذي سوف يقوم البرنامج بالكتابة به موجود مسبقاً فسوف يطلب من المستخدم إدخال "C" في حال كان يريد المتابعة أو "Q" من أجل الخروج من البرنامج.

السلسلة النصية في المتغير response تقوم باستدعاء الطريقة lower() والسلسلة النصية المُعادلة من lower() تقوم باستدعاء الطريقة النصية startswith()

الطريقة النصية starswith() ستعيد True إذا كانت السلسلة النصية التي يتم تمريرها لها موجودة في بداية السلسلة

جرب المثال التالي في الشيل التفاعلية:

```
>>> 'hello'.startswith('h')
True
>>> 'hello world'.startswith('h')
True
>>> 'hello'.startswith('H')
False
>>> spam = 'Jameel'
>>> spam.startswith('Jam')
True
```

في السطر 23 إذا لم يقم المستخدم بكتابة 'C' , 'c' أو أي سلسلة نصية تبدأ بالحرف C سوف يتم استدعاء sys.exit() والخروج من البرنامج

المستخدم ليس مضطراً إلى أن يكتب 'Q' من أجل الخروج من البرنامج لأن أي سلسلة نصية يقوم بكتابتها ولا تبدأ بالحرف C سوف يتم استدعاء التابع sys.exit() من أجل الخروج من البرنامج.

الطريقة النصية endswith() تستخدم من أجل فحص فيما إذا كانت السلسلة النصية موجودة في نهاية سلسلة نصية أخرى

جرب المثال التالي في الشيل التفاعلية:

```
>>> 'Hello world'.endswith('world')
True
>>> 'Hello world'.endswith('Hello')
False
>>>
```

## الطريقة النصية :title()

بشكل مشابه للطرق النصية lower() and upper() والتي تعيد السلسلة النصية بحروف صغيرة أو كبيرة فإن الطريقة النصية title() تعيد السلسلة النصية أول حرف كبير من كل كلمة وباقى الأحرف صغيرة.

جرب المثال التالي في الشيل التفاعلية:

```
>>> 'hello'.title()
'Hello'
>>> 'HELLO'.title()
'Hello'
>>> 'hELLO'.title()
'Hello'
>>> 'hello world HOW ARE YOU'.title()
'Hello World How Are You'
>>>
```

```
26.     # Read in the message from the input file
27.     fileObj = open(inputFilename)
28.     content = fileObj.read()
29.     fileObj.close()
30.
31.     print('%sing...' % (myMode.title()))
```

transpositionFileCipher.py

في السطور 27 إلى 29 يتم فتح الملف الذي يكون اسمه موجود في المتغير `inputFilename` وقراءة محتوى هذا الملف وتخزينه في المتغير `content`

في السطر 31 يتم عرض رسالة تخبر المستخدم أن عملية التشفير أو فك الشفير سوف تبدأ ويجب أن يجوي المتغير `myMode` على السلسة النصية 'encrypt' or 'decrypt' والتي تقوم باستدعاء الطريقة النصية `title()` وسوف تظهر 'Encrypting...' or 'Decrypting...'.

## الوحدة `time` التابع (`time.time()`)

كل أجهزة الحاسب تملك ساعة من أجل تتبع التوقيت والتاريخ.

برنامنج البيانات يمكنه الوصول إلى هذه الساعة من خلال التابع (`time.time()`) التابع (`time`) موجود ضمن الموديول

التابع (`time.time()`) يعيد قيمة حقيقة `float value` لعدد الثواني من أول أيام الشهر الأول لعام 1970 هذه اللحظة تسمى عصر يونكس Unix Epoch

جرب المثال التالي:

```
>>> time.time()
1446316653.670954
>>> time.time()
1446316666.499506
>>>
```

القيمة الحقيقة التي يعيدها التابع `time.time()` غير مفيدة لتحديد قيمة الوقت الحالي ولكنها مفيدة من أجل المقارنة للفترة الزمنية بين استدعاء هذا التابع لأكثر من مرة ويمكن الاستفادة من هذا التابع من أجل تحديد الفترة الزمنية التي استمر بها البرنامج بالعمل.

```
transpositionFileCipher.py
33.     # Measure how long the encryption/decryption takes.
34.     startTime = time.time()
35.     if myMode == 'encrypt':
36.         translated = transpositionEncrypt.encryptMessage(myKey, content)
37.     elif myMode == 'decrypt':
38.         translated = transpositionDecrypt.decryptMessage(myKey, content)
39.     totalTime = round(time.time() - startTime, 2)
40.     print('%sion time: %s seconds' % (myMode.title(), totalTime))
```

نريد معرفة الفترة الزمنية التي استمرت بها عملية التشفير أو فك التشفير.  
في الأسطر 35 to 38 يتم استدعاء `encryptMessage()` or `decryptMessage()` وذلك بحسب القيمة المخزنة في المتغير `myMode` إن كانت '`encrypt`' or '`decrypt`'  
قبل تنفيذ هذا الكود قمنا باستدعاء التابع `time.time()` وقمنا بتخزين قيمة الوقت الحالة في المتغير `startTime`

في السطر 39 وبعد استدعاء تابع التشفير أو فك التشفير نقوم باستدعاء التابع `time.time()` مرة ثانية وقمنا بطرح القيمة المحفوظة في المتغير `startTime` من القيمة المعاادة من استدعاء التابع للمرة الثانية وهذا سوف يعطينا عدد الثواني بين استدعائين لهذا التابع.

القيمة المعاادة من حساب العبارة الجبرية `time.time() - startTime` يتم تمريرها إلى التابع `round()` والذي يقوم بتقرير القيمة الحقيقة إلى أقرب فاصلة عشرية وهذه القيمة يتم حفظها في المتغير `totalTime`

في السطر 40 يتم استدعاء التابع `print()` من أجل إظهار مدة الوقت التي استمرت به عملية التشفير أو فك التشفير.

```
transpositionFileCipher.py
42.     # Write out the translated message to the output file.
43.     outputFileObj = open(outputFilename, 'w')
44.     outputFileObj.write(translated)
45.     outputFileObj.close()
```

محتوى الملف الذي تم تشفيره أو فك تشفيره يتم حفظه في المتغير `translated` ولكن هذه القيمة يمكن أن تنسى عندما ينتهي البرنامج لذلك سوف نقوم بكتابة هذه السلسلة النصية في ملف وحفظه في القرص الصلب.

التعليمات في الأسطر 43 to 45 تقوم بفتح ملف جديد من (من خلال تمرير 'w' للتابع `write()` ومن ثم استدعاء الطريقة `(open()`)

```
transpositionFileCipher.py
47.     print('Done %sing %s (%s characters).' % (myMode, inputFilename,
len(content)))
48.     print('%sed file is %s.' % (myMode.title(), outputFilename))
```

```
49.
50.
51. # If transpositionCipherFile.py is run (instead of imported as a module)
52. # call the main() function.
53. if __name__ == '__main__':
54.     main()
```

بعد ذلك نقوم بطباعة بعض الرسائل للمستخدم تخبره أن العملية تمت وتخبره باسم الملف الذي تم الكتابة به.

السطر 48 هو آخر سطر في التابع `main()`

السطرين 53 and 54 يتم تنفيذهما بعد التعليمة `def` الموجودة في السطر 6 وسيتم استدعاء التابع `(main()` إذا تم تشغيل هذا البرنامج ولم يتم استيراده من قبل برنامج آخر (في الفصل الثامن قمنا بشرح القيمة الخاصة `__name__`)

## **الخلاصة:**

مبروك ...

باستخدام هذا البرنامج والتوابع يمكنك تشفير الملفات النصية الموجودة في القرص الصلب في جهازك ومهما كان حجمها.

يوجد العديد من قيم المفاتيح المحتملة من أجل القيام بهجوم القوة الغاشمة brute-force على الرسالة المشفرة بشيفرة التحويل.

ولكن إذا قمنا بكتابة برنامج يتعرف على اللغة الانجليزية (سلسلة من الكلمات المفهومة) يمكن لجهاز الكمبيوتر أن يقوم بفحص خرج آلاف محاولات فاك التشفير ليقرر ما هي قيمة المفتاح التي يتم بها فاك التشفير بشكل صحيح ونحصل على رسالة مفهومة باللغة الانجليزية.



## الفصل الحادي عشر

### كتابة برنامج لِاكتشاف

### اللغة الانجليزية

---

محتوى هذا الفصل:

- القواميس
- الطريقة `split()`
- القيمة `None`
- الأخطاء "Divide by Zero"
- التوابع `float()`, `int()` and `str()`
- الطريقة `append` الخاصة بالقوائم `List`
- البارامترات الأفتراضية
- حساب النسبة المئوية

الرسالة المشفرة باستخدام شيفرة التحويل يمكن أن يتم تشفيرها باستخدام الآلاف من مفاتيح التشفير.

جهاز الكمبيوتر يمكنه وبسهولة تطبيق هجوم القوة الغاشمة على هذه المفاتيح ولكن يجب عليك بعدها أن تنظر إلى آلاف النصوص بعد عملية فك التشفير لتجد النص الصحيح.

هذه مشكلة كبيرة في هجوم القوة الغاشمة من أجل كسر شيفرة التحويل.

عندما يقوم الكمبيوتر بفك تشفير الرسالة باستخدام قيمة مفتاح خاطئة فالنتيجة ستكون هي نص صريح غير مفهوم.

نحتاج لبرنامج يجعل الكمبيوتر قادر على إدراك فيما إذا النص الصريح هو نص غير مفهوم أو انه نص مفهوم باللغة الانجليزية.

وبهذه الطريقة إذا قام الكمبيوتر بفك تشفير الرسالة بالمفتاح الخطأ فهو سوف يعرف ان قيمة هذا المفتاح خاطئة وسوف ينتقل إلى قيمة مفتاح أخرى ويتابع عملية فك التشفير إلى أن يحصل على نص صريح ومفهوم باللغة الانجليزية عندها يتوقف ويقدم هذا المفتاح لمحل الشيفرة.

وعندها لن يكون على محل الشيفرة النظر إلى آلاف الرسائل بعد عملية فك التشفير.

## كيف يمكن للكمبيوتر أن يفهم اللغة الانجليزية:

لا يمكنه.

على الأقل ليس بنفس الطريقة التي يستطيع الإنسان أن يدرك بها الكمبيوتر لا يستطيع أن يفهم الرياضيات أو الشطرنج أو اللغات بل هو يقوم فقط بتنفيذ التعليمات.

ولكن هذه التعليمات يمكنها أن تحاكى السلوك المستخدم لحل مسائل الرياضيات أو لعب الشطرنج أو فهم اللغات.

ما هو التابع الذي يحتاج للقيام بهذه المهمة (لنسميها `isEnglish`) سنقوم بتمرير قيمة نصية لهذا التابع وهو يعيد `True` إذا كانت هذه السلسلة هي نص باللغة الانجليزية ويعيد `False` إذا كانت السلسلة النصية عبارة عن كلمات مكونة من أحرف عشوائية.

لأخذ بعض الأمثلة على نصوص باللغة الانجليزية ونصوص لكلمات عشوائية غير مفهومة.

```
Robots are your friends. Except for RX-686. She will try to eat you.
```

```
ai-pey e. xrx ne augur iirl6 Rtiyt fhubE6d hrSei t8..ow eo.telyoosEs t
```

أول أمر يمكن أن نلاحظه هو أن نص اللغة الانجليزية مكون من الكلمات التي يمكن أن تكون موجودة في المعجم أما النص العشوائي لا يحوي على كلمات موجودة في المعجم

إذا قمنا بتقسيم النص إلى كلمات باستخدام الطريقة النصية (`split`) عندها يمكننا البحث فيما إذا كانت هذه الكلمات موجودة في المعجم أو أنها كلمات عشوائية.

باستخدام الكود التالي:

```
>>> if word == 'aardvark' or word == 'abacus' or word == 'abandon' or .....
```

للقيام بهذه العملية بشكل اتوماتيكي سوف نستخدم ملف نصي يحوي على كلمات المعجم  
الانجليزية `dictionary file`

وسوف نقوم بكتابه تابع يقوم بفحص فيما إذا كانت هذه الكلمات موجودة في إحدى هذه الملفات.

عبارة عن ملف نصي كبير يحوي على عدد كبير من الكلمات  
`dictionary file` الانجليزية.

ليست كل الكلمات موجودة في "dictionary file" مثلاً يمكن أن يحوي النص على كلمات  
"RX-686"

كما يمكن أن يحوي النص العشوائي على بعض الكلمات الانجليزية الصحيحة  
 التابع الذي سوف نقوم بكتابته لن يكون سهلاً، إذا كانت معظم الكلمات هي كلمات انجليزية  
 صحيحة يمكننا أن القول أن هذا النص هو نص انجليزي صحيح وهذا يجعل احتمال أن يكون قد  
 تم فك التشفير باستخدام المفتاح الصحيح هو احتمال كبير.

يمكنك تحميل هذا الملف (والذي يحوي على أكثر من 45,000 كلمة ) من الرابط التالي:

<http://invpy.com/dictionary.txt>

التابع الذي سوف يقوم بكتابته `isEnglish()` سوف يقوم بتقسيم السلسلة النصية بعد عملية فك التشفير إلى كلمات ويقوم بفحص فيما إذا كانت هذه الكلمات هي كلمات باللغة الإنجليزية أم أنها كلمات عشوائية.

إذا كانت النسبة الأكبر من هذه الكلمات هي كلمات من اللغة الإنجليزية فهذا يعني أنه قد تم فك التشفير باستخدام مفتاح التشفير الصحيح.

## **:Detect English Module**

البرنامج `detectEnglish.py` الذي سوف يقوم بكتابته في هذا الفصل لا يستطيع العمل بمفرده وسوف يقوم باستيرداده في برنامج التشفير لنتمكن من استدعاء التابع `detectEnglish.isEnglish()`

وهذا هو سبب عدم وجود التابع `main()` داخل هذا البرنامج.

## **ال코드 البرمجي لوحدة اكتشاف اللغة الانجليزية:**

قم بفتح نافذة جديدة من محرر النصوص وقم بكتابة الكود التالي ثم قم بحفظه باسم `detectEnglish.py` ثم اضغط F5 من أجل تنفيذ البرنامج

## Source code for detectEnglish.py

```
1. # Detect English module
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. # To use, type this code:
5. # import detectEnglish
6. # detectEnglish.isEnglish(someString) # returns True or False
7. # (There must be a "dictionary.txt" file in this directory with all English
8. # words in it, one word per line. You can download this from
9. # http://invpy.com/dictionary.txt)
10. UPPERLETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
11. LETTERS_AND_SPACE = UPPERLETTERS + UPPERLETTERS.lower() + ' \t\n'
12.
13. def loadDictionary():
14.     dictionaryFile = open('dictionary.txt')
15.     englishWords = {}
16.     for word in dictionaryFile.read().split('\n'):
17.         englishWords[word] = None
18.     dictionaryFile.close()
19.     return englishWords
20.
21. ENGLISH_WORDS = loadDictionary()
22.
23.
24. def getEnglishCount(message):
25.     message = message.upper()
26.     message = removeNonLetters(message)
27.     possibleWords = message.split()

28.
29.     if possibleWords == []:
30.         return 0.0 # no words at all, so return 0.0
31.
32.     matches = 0
33.     for word in possibleWords:
34.         if word in ENGLISH_WORDS:
35.             matches += 1
36.     return float(matches) / len(possibleWords)
37.
38.
39. def removeNonLetters(message):
40.     lettersOnly = []
41.     for symbol in message:
42.         if symbol in LETTERS_AND_SPACE:
43.             lettersOnly.append(symbol)
44.     return ''.join(lettersOnly)
45.
46.
47. def isEnglish(message, wordPercentage=20, letterPercentage=85):
48.     # By default, 20% of the words must exist in the dictionary file, and
49.     # 85% of all the characters in the message must be letters or spaces
50.     # (not punctuation or numbers).
51.     wordsMatch = getEnglishCount(message) * 100 >= wordPercentage
52.     numLetters = len(removeNonLetters(message))
53.     messageLettersPercentage = float(numLetters) / len(message) * 100
54.     lettersMatch = messageLettersPercentage >= letterPercentage
55.     return wordsMatch and lettersMatch
```

## كيف يعمل هذا البرنامج:

detectEnglish.py

```
1. # Detect English module
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
4. # To use, type this code:
5. #     import detectEnglish
6. #     detectEnglish.isEnglish(someString) # returns True or False
7. # (There must be a "dictionary.txt" file in this directory with all English
8. # words in it, one word per line. You can download this from
9. # http://invpy.com/dictionary.txt)
```

التعليقات في بداية البرنامج تعطي إرشادات للمبرمج عن كيفية استخدام هذه الوحدة **module**

يجب أن تقوم بوضع الملف **dictionary.txt** في نفس المجلد الذي يحتوي على  
يجب أن تقوم بوضع الملف **detectEnglish.py** وإلا لن يعمل هذا البرنامج

يمكنك تحميل هذا الملف من الرابط <http://invpy.com/dictionary.txt>

detectEnglish.py

```
10. UPPERLETTERS = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
11. LETTERS_AND_SPACE = UPPERLETTERS + UPPERLETTERS.lower() + '\t\n'
```

في السطر 11 and 10 قمنا بإعداد بعض المتغيرات والتي تحوي على قيم ثابتة  
المتغير **UPPERLETTERS** يحتوي على الأحرف الكبيرة والمتغير  
**LETTERS\_AND\_SPACE** يحتوي على الأحرف الكبيرة والأحرف الصغيرة بالإضافة  
إلى المسافة الفارغة والمسافة التي يخلقها الزر **tab** (\t) والانتقال لسطر جديد (\n).

detectEnglish.py

```
13. def loadDictionary():
14.     dictionaryFile = open('dictionary.txt')
```

يجب أن تقوم بفتح الملف **dictionary** وحفظ محتواه في متغير كسلسلة نصية ليتمكن البايثون  
من التعامل معه.

في البداية سنقوم باستدعاء **open()** من أجل فتح هذا الملف

قبل التعرف كود التابع (loadDictionary) سوف نتعرف على نوع البيانات **dictionary** data type

## :Dictionary Data Type

هذا النوع من البيانات يحوي على قيم والتي يمكن أن تحوي على قيم متعددة أخرى، بشكل مشابه للقوائم lists

في نوع المتغير القائمة list كنا نستخدم قيمة عدد صحيح للفهرسة من أجل استدعاء القيم الموجودة في القائمة، مثل spam[42]

لكل مادة في المتغير من نوع القاموس dictionary يوجد مفتاح يستخدم لإستدعاء هذه المادة (القيم المحفوظة في داخل القوائم والقواميس تسمى مواد items)

المفتاح يمكن أن يكون عدد صحيح أو قيمة نصية مثل spam['hello'] or spam[42] القواميس تسمح لنا بتنظيم بيانات البرنامج بشكل أفضل من القوائم.

في القوائم نستخدم الأقواس المربعة [ ] أما في القواميس فنستخدم الأقواس الكبيرة { }

```
>>> emptyList = []
>>> emptedictionart = {}
```

المتغير من نوع قاموس تستخدم زوج من المفاتيح والتي يتم الفصل بينها باستخدام ' : ' ويتم الفصل بين المفاتيح المتعددة باستخدام ' , '

من أجل استدعاء قيمة من القاموس نستخدم الأقواس المربعة مع قيم المفاتيح المطلوبة، جرب المثال التالي في الشيل التفاعلية:

```
>>> spam = {'key1':'This is a value' , 'key2':42}
>>> spam['key1']
'This is a value'
>>> spam['key2']
42
>>>
```

جرب المثال التالي:

```
>>> spam = {'hello':42}
>>> eggs = spam
>>> eggs['hello'] = 99
>>> eggs
{'hello': 99}
>>> spam
{'hello': 99}
>>>
```

## تعديل وإضافة مواد إلى القاموس:

يمكنك إضافة أو تعديل القيم في القاموس من خلال الفهرسة، جرب المثال التالي:

```
>>> spam = {42:'hello'}
>>> print(spam[42])
hello
>>> spam[42] = 'goodbye'
>>> print(spam[42])
goodbye
>>>
```

بما أن القوائم يمكن أن تحوي على قوائم أخرى في داخلها فإن القواميس أيضاً يمكن أن تحوي على قواميس أخرى في داخلها، جرب المثال التالي:

```
>>> spam = {42:'hello'}
>>> print(spam[42])
hello
>>> spam[42] = 'goodbye'
>>> print(spam[42])
goodbye
>>>
>>> foo = {'fizz': {'name': 'Ali', 'age': 144}, 'moo':[ 'a', 'brown', 'cow' ] }
>>> foo['fizz']
{'name': 'Ali', 'age': 144}
>>> foo['fizz']['name']
'Ali'
>>> foo['moo']
['a', 'brown', 'cow']
>>> foo['moo'][1]
'brown'
>>>
```

## استخدام التابع len() مع القواميس:

التابع len() يخبرنا بعدد المواد الموجودة في القائمة أو عدد الأحرف الموجودة في سلسلة نصية وهو أيضاً يخبرنا بعدد المواد الموجودة في القاموس

جرب المثال التالي:

```
>>> spam = {}
>>> len(spam)
0
>>> spam['name'] = 'Ali'
>>> spam['age'] = 30
>>> len(spam)
2
>>>
```

## استخدام المعامل **in** في القواميس:

المعامل **in** يستخدم لمعرفة وجود قيمة مفتاح معين في القاموس (قيمة المفتاح وليس قيمة معينة)، جرب المثال التالي:

```
>>> eggs = {'foo': 'milk', 'bar': 'bread'}
>>> 'foo' in eggs
True
>>> 'blah blah' in eggs
False
>>> 'milk' in eggs
False
>>> 'bar' in eggs
True
>>> 'bread' in eggs
False
>>>
```

## استخدام حلقة **for** مع القواميس:

يمكننا المرور على كل المفاتيح باستخدام حلقة **for** ، جرب المثال التالي:

```
>>> spam = {'name': 'Ali' , 'age': 40}
>>> for k in spam:
    print(k)
    print(spam[k])
    print('=====')  
  
name
Ali
=====
age
40
=====
>>>
```

## الفرق بين القواميس والقوائم:

القواعد تشبه القوائم في عدة أمور ولكن يوجد بعض الاختلافات وهي:

١- المواد في القواميس لا تحوي على أي ترتيب، لا يوجد المادة الأولى والمادة الأخيرة كما في القوائم.

٢- القواميس لا يمكن جمعها باستخدام المعامل '+ ' إذا كنت تريد إضافة مواد جديدة إلى القاموس يجب عليك إدخال قيمة فهرسة لمفتاح جديد، مثلاً:

```
foo['a new key'] = 'a string'
```

٣- القوائم تتعامل فقط مع قيم الفهرسة التي تبدأ من الصفر إلى طول القائمة منقوص منه واحد أما القواميس فيمكنها التعامل مع أي قيمة لمفتاح الفهرسة  
إذا كان لديك قاموس يخزن القيم في المتغير `spam` فيمكنك تخزين قيمة في `spam[1]` or `spam[2]` دون الحاجة لتخزين قيمة في `spam[3]`

يجب أن تميز بين ملف القاموس الذي يحوي على كلمات المعجم والذي سوف نستخدمه في هذا البرنامج وبين المتغير من نوع قاموس في لغة البيايثون.

## إيجاد المواد في القواميس:

```
15.     englishWords = {}                                     detectEnglish.py
```

في التابع `loadDictionary()` سوف نقوم بتخزين كل الكلمات الموجودة في الملف النصي "dictionary value" (كلمات المعجم) في متغير من نوع قاموس "dictionary file" (python data type)

للأسف نفس الاسم (قاموس) يجب أن تميز بينهما.

سوف نستخدم قاموس من أجل تخزين القيم النصية لكل كلمة في ملف القاموس `dictionary file` والسبب من ذلك هو أن المعامل '`in`' يعمل في القواميس بشكل أسرع من القوائم تخيل أن لدينا القائمة والقاموس التاليين:

```
>>> listVal = ['spam', 'eggs', 'bacon']
>>> dictionaryVal = {'spam':0, 'eggs':0, 'bacon':0}
>>>
```

البايثون يستطيع تنفيذ التعليمات 'bacon' in dictionaryVal بشكل أسرع من التعليمات  
'bacon' in listVal

وفي الوحدة detectEnglish module سيكون لدينا آلاف المواد وسوف يتم تنفيذ التعليمات  
isEnglish(word) في الملايين من المرات عند استدعاء التابع word in ENGLISH\_WORDS

## الطريقة :split()

الطريقة النصية split() تقوم بإعادة قائمة لعدة سلاسل نصية

لمعرفة كيفية عمل هذه الطريقة جرب المثال التالي:

```
>>> 'I am a security engineer'.split()
['I', 'am', 'a', 'security', 'engineer']
>>>
```

كما يمكن استخدام هذه الطريقة بشكل مختلف كما في المثال التالي:

```
>>> 'helloxxxworldxxxhowxxxarexxxxyou?'.split('xxx')
['hello', 'world', 'how', 'are', 'you?']
>>>
```

```
16.     for word in dictionaryFile.read().split('\n'):
```

detectEnglish.py

في السطر 16 استخدمنا حلقة for لتقوم بضبط قيمة المتغير word لكل قيمة في القائمة التي  
ستعيدها التعليمية dictionaryFile.read().split('\n') ، لنقم بتجزئة هذه العملية

file object هو متغير يتم تخزينه في dictionaryFile

استدعاء الطريقة dictionaryFile.read() سوف يقوم بقراءة كامل الملف ويعيده على شكل  
سلسة نصية طويلة جداً.

سوف نقوم باستدعاء الطريقة split() من أجل الفصل بين هذه الكلمات.

وبالتالي فإن التعليمية `dictionaryFile.read().split("\n")` سوف تقوم بإعادة قائمة تحوي على الكلمات الموجودة في هذا الملف.

## القيمة :None

هي عبارة عن قيمة خاصة يمكن اسنادها لمتغير معين وهي القيمة الوحيدة لنوع البيانات `None` (بشكل مشابه لنوع البيانات المنطقي والذي يملك قيمتين: الصفر والواحد) نوع البيانات `None` يملك قيمة وحيدة وهي `None` من المفيد استخدام هذه القيمة عندما تحتاج لاستخدام قيمة غير موجودة والقيمة `None` تكتب بدون إشارات تنصيص والحرف الأول منها يجب أن يكون حرف كبير.

مثلاً: إذا كان لديك متغير له الاسم `quizAnswer` والذي سوف يحوي على إجابة المستخدم `True or False`

يمكنك ضبط القيمة `None` للمتغير `quizeAnswer` في حال لم يقم المستخدم بالإجابة على هذا السؤال.

استدعاء التابع التي لن تقوم بإعادة أي شئ (ينتهي التابع بدون استدعاء التعليمية `return`) يمكن أن تسند لها القيمة `None`

```
17.     englishWords[word] = None
```

في برنامجنا سوف نستخدم متغير من نوع قاموس `dictionary` باسم `englishWords` وسوف نستخدم المعامل `in` من أجل إيجاد المفاتيح في داخله.

لن نهتم للقيمة التي سوف تخزن في كل مفتاح، فقط سوف نستخدم القيمة `None`.

الحلقة `for` التي تبدأ في السطر 16 سوف تمر على كل كلمة في الملف النصي الذي يحوي على كلمات المقاموس والسطر 17 سوف نستخدم `word` كمفتاح في `englishWords` مع القيمة `None` التي ستكون مخزنة في كل مفتاح.

```
18.     dictionaryFile.close()
19.     return englishWords
```

detectEnglish.py

وبعد أن تنتهي الحلقة `for` فإن المتغير من نوع قاموس `englishWords` سوف يحوي على آلاف المفاتيح (الكلمات) وعندها سوف تقوم بـ إغلاق الملف النصي.

```
21. ENGLISH_WORDS = loadDictionary()
```

detectEnglish.py

في السطر 21 نستدعي التابع (`loadDictionary()`) ونقوم بتخزين القيمة التي يعيدها في المتغير `ENGLISH_WORDS`

```
24. def getEnglishCount(message):
25.     message = message.upper()
26.     message = removeNonLetters(message)
27.     possibleWords = message.split()
```

detectEnglish.py

التابع (`getEnglishCount()`) سوف يأخذ سلسلة نصية واحدة ويعيد قيمة عدد حقيقي للدلالة على عدد الكلمات التي تعتبر كلمات باللغة الانجليزية.

القيمة 0.0 تعني أنه لا يوجد أي كلمة انجليزية في المتغير `message`

أما القيمة 1.0 تعني أنه كل الكلمات في المتغير `message` هي كلمات انجليزية

التابع (`isEnglishCount()`) سوف يعيد قيمة بين 0.0 – 1.0 والتابع (`getEnglisgCount()`) سوف يستخدم هذه القيمة لتحديد فيما إذا كان النص بعد عملية فك التشفير هو نص باللغة الانجليزية أو لا.

في البداية يجب أن نقوم بخلق قائمة تحوي على الكلمات الموجودة في السلسلة النصية المخزنة في المتغير `message`.

في السطر 25 سوف نقوم بتحويل الكلمات إلى أحرف كبيرة.

في السطر 26 سوف نحذف القيم الغير حرفية من السلسلة النصية مثل الأرقام وعلامات الترقيم وذلك من خلال استدعاء التابع (`removeNonLetters()`) سوف نشرح طريقة عمل هذا التابع لاحقاً، وأخيراً سوف نستدعي الطريقة (`split()`) في السطر 27 من أجل القيام بـ تقسيم السلسلة النصية إلى كلمات وحفظها في المتغير `possibleWords`.

مثلاً إذا قمنا بتمرير السلسلة النصية 'Hello there. How are you?' إلى التابع possibleWords فسوف يعيد القيم التالية ويخزنها في المتغير getEnglishCount()

[ 'HELLO' , 'THERE' , 'HOW' , 'ARE' , 'YOU' ]

```
29.     if possibleWords == []:
30.         return 0.0 # no words at all, so return 0.0
```

detectEnglish.py

إذا كان المتغير message يحوي على أرقام مثل '12345' فسوف يقوم التابع removeNonLetters() بحذفها ويعيد سلسلة نصية فارغة وعندما سوف يتم استدعاء الطريقة split() على سلسلة نصية فارغة والتي سوف تعيد قائمة فارغة وسوف يتم إعادة القيمة 0.0

```
32.     matches = 0
33.     for word in possibleWords:
34.         if word in ENGLISH_WORDS:
35.             matches += 1
```

detectEnglish.py

القيمة الحقيقة التي يعيدها التابع getEnglishCount() ستكون بين القيمتين 0.0 – 1.0 في البداية سوف نقوم بتعيين القيمة 0 للمتغير matches ثم نستخدم حلقة for في السطر 33 من أجل أن تمر على كل كلمة موجودة في المتغير possibleWords لترى فيما إذا كانت هذه الكلمة موجودة في القاموس ENGLISH\_WORDS فإذا كانت موجودة سوف يتم زيادة قيمة المتغير matches بمقدار واحد.

عندما تنتهي حلقة for من عملها سوف يتم تخزين عدد الكلمات الانجليزية في المتغير matches

البرنامنج يعمل بالطريقة التالية: إذا كانت الكلمة موجودة في الملف النصي dictionary.txt فهـي كلمة انجليزية وإذا لم تكن موجودة فهي كلمة عشوائية.

## خطأ القسمة على صفر:

detectEnglish.py

36. return float(matches) / len(possibleWords)

إعادة قيمة حقيقة بين 0.0 - 1.0 تتم من خلال قسمة عدد الكلمات المكتشفة على أنها كلمات إنجليزية على عدد الكلمات الكلية.

هذه العملية تتم باستخدام المعامل " / " ، يجب أن تنتبه لكي لا نقوم بعملية القسمة على العدد صفر.

أبي رحمة الله قد علمني العبارة التالية: "القسمة على صفر هي كفر"  
الباقيون سوف يعيد رسالة خطأ عند محاولة القسمة على صفر، جرب المثال التالي:

```
>>> 42 / 0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    42 / 0
ZeroDivisionError: division by zero
>>>
```

الحالة الوحيدة التي يمكن أن تسبب هذه النتيجة هي عندما تكون نتيجة `len(possibleWords)` هي صفر

في كود البرنامج وفي السطر 29 قمنا بفحص القائمة `possibleWords` إذا كانت قائمة فارغة فسوف يتم تنفيذ التعليمية الموجودة في السطر 30 ولن نحصل على خطأ القسمة على صفر في السطر 36

## التابع :float() , int() and str()

detectEnglish.py

36. return float(matches) / len(possibleWords)

القيمة المخزنة في المتغير `matches` هي قيمة عدد صحيح `integer` قمنا بتمرير هذه القيمة إلى التابع (`float()`) والذي يعيد قيمة عدد حقيقي، جرب المثال التالي:

```
>>>  
>>> float(42)  
42.0  
>>> int(42.0)  
42  
>>> int("42")  
42  
>>> str(42)  
'42'  
>>> str(42.7)  
'42.7'  
>>>
```

التابع () `float()`, `int()`, and `str()` مفيدة في تحويل القيم بين أنواع المتغيرات

```
39. def removeNonLetters(message):  
40.     lettersOnly = []  
41.     for symbol in message:
```

`detectEnglish.py`

في التابع () `getEnglishCount()` قمنا باستدعاء التابع () `removeNonLetters()` من أجل إعادة السلسلة النصية بدون أرقام أو علامات ترقيم.

ال코드 في التابع () `removeNonLetters()` يبدأ بقائمة فارغة ثم يتم المرور على الأحرف الموجودة في المتغير `message`، إذا كانت الأحرف موجودة في السلسلة `LETTERS_AND_SPACE` فسوف يتم إضافتها إلى نهاية القائمة وإذا كانت المحارف هي أرقام أو علامات ترقيم فلن تكون موجودة في السلسلة `LETTERS_AND_SPACE` ولن يتم إضافتها إلى القائمة.

```
42.         if symbol in LETTERS_AND_SPACE:  
43.             lettersOnly.append(symbol)
```

`detectEnglish.py`

في السطر 42 قمنا بفحص فيما إذا كان كل حرف موجود في السلسلة `LETTERS_AND_SPACE` وإذا كان موجود سيتم إضافته إلى نهاية القائمة `append()` باستخدام الطريقة () `lettersOnly`

```
44.     return ''.join(lettersOnly)
```

`detectEnglish.py`

بعد أن تنتهي حلقة `for` الموجودة في السطر 41 من عملها سوف تحوي القائمة `lettersOnly` على الأحرف والفراغات فقط ومن أجل سلسلة نصية واحدة من هذه القائمة سوف نستدعي

الطريقة النصية `join` على سلسلة نصية فارغة والتي سوف تقوم بجمع كل السلسل النصية الموجودة في القائمة `lettersOnly` مع بعضها.

```
detectEnglish.py
47. def isEnglish(message, wordPercentage=20, letterPercentage=85):
48.     # By default, 20% of the words must exist in the dictionary file, and
49.     # 85% of all the characters in the message must be letters or spaces
50.     # (not punctuation or numbers).
```

التابع (`isEnglish`) يقبل سلسلة نصية كدخل له ويعيد قيمة منطقية من `True` or `False` من أجل الدلاله إذا كانت هذه السلسلة النصية هي نص باللغة الانجليزية أم أنها كلمات عشوائية.

في السطر 47 قمنا بتعريف التابع باستخدام ثلاثة برمترات.

البرامترات `wordPercentage` and `letterPercentage` تحوي على قيم افتراضية وإذا لم يقم التابع بتمرير قيم أخرى فسوف يتم استخدام هذه القيم بشكل افتراضي. إذا تم إستدعاء التابع `isEnglish()` مع برمتر واحد وهو السلسلة النصية فعندها ستكون قيم البرامترات الأخرى هي القيمة الافتراضية

`wordPercentage = 20`

`letterPercentage = 85`

الجدول التالي يظهر بعد طرق استدعاء هذا التابع باستخدام قيم مختلفة

Function Call	Equivalent To
<code>isEnglish('Hello')</code>	<code>isEnglish('Hello', 20, 85)</code>
<code>isEnglish('Hello', 50)</code>	<code>isEnglish('Hello', 50, 85)</code>
<code>isEnglish('Hello', 50, 60)</code>	<code>isEnglish('Hello', 50, 60)</code>
<code>isEnglish('Hello', letterPercentage=60)</code>	<code>isEnglish('Hello', 20, 60)</code>

عندما يتم استدعاء التابع (`isEnglish`) بدون البرامترتين الثاني والثالث فإن التابع سوف يتطلب أن يكون 20% من الكلمات الموجودة في المتغير `message` هي كلمات انجليزية (موجودة في الملف النصي الي يحوي على كلمات القاموس) و 85% من المحارف في المتغير `message` هي أحرف.

## حساب النسبة المئوية:

النسبة المئوية هي رقم بين 0 - 100

السلسة النصية التالية: "Hello cat Moose fsdkl ewpin" تحوي على خمس كلمات ولكن ثلاثة فقط من هذه الكلمات هي كلمات باللغة الانجليزية.

من أجل حساب النسبة المئوية لعدد الكلمات الانجليزية نقم بقسمة عدد الكلمات الانجليزية على عدد الكلمات الكلية ومن ثم نضرب النتيجة بالعدد 100

$$3 / 5 * 100 = 6$$

النسبة المئوية هي عدد بين 0% (لا يوجد أي كلمة انجليزية) و 100% (كل الكلمات هي كلمات انجليزية)

التابع (`isEnglish()`) سوف يحدد النص إذا كان نص باللغة الانجليزية أو لا إذا كان يحوي على الأقل 20% من الكلمات الانجليزية الموجودة في ملف القاموس في ملف القاموس و 85% من المحارف هي سلاسل نصية أو أحرف أو فراغات.

```
51.     wordsMatch = getEnglishCount(message) * 100 >= wordPercentage
```

في السطر 51 يتم حساب النسبة المئوية للكلمات الانجليزية في الرسالة من خلال تمرير المتغير `message` إلى التابع (`getEnglishCount()`) والذي يقوم بإعادة قيمة بين 0.0 and 1.0 ومن ثم يتم حساب النسبة المئوية بالإعتماد على هذه القيمة وذلك بالقيام بعملية الضرب بالعدد 100

إذا كان العدد الناتج أكبر أو يساوي قيمة البرامتي `wordPercentage` فسوف يتحقق الشرط ويتم حفظ القيمة `True` في المتغير `wordsMatch` وفي حال عدم تحقق الشرط سوف يتم حفظ القيمة `False` في المتغير `wordsMatch`

```
52.     numLetters = len(removeNonLetters(message))
53.     messageLettersPercentage = float(numLetters) / len(message) * 100
54.     lettersMatch = messageLettersPercentage >= letterPercentage
```

في الأسطر 52 to 54 يتم حساب المسبة المئوية للأحرف في الرسالة وذلك من أجل تحديد النسبة المئوية لكل من الأحرف والفراغات.

ويتم ذلك من خلال تقسيم عدد الأحرف على عدد المحارف الكلية (أقصد بكلمة محارف الأحرف والفراغات وعلامات الترقيم).

في السطر 52 قمنا باستدعاء التابع removeNonLetters(message) والذي سوف يعيد سلسلة نصية تحوي على الأحرف وعلامات الترقيم وبدون فراغات.

في السطر 53 يتم حساب النسبة المئوية من خلال تحويل قيمة numLetters إلى قيمة عدد حقيقي ومن ثم القسمة على عدد المحارف الكلية في الرسالة وضرب الناتج بالقيمة 100

في السطر 54 يتم فحص النسبة المئوية إذا كانت أكبر أو مساوية لقيمة letterPercentage وتخزين النتيجة المنطقية True or False في المتغير lettersMatch

```
55.     return wordsMatch and lettersMatch
```

detectEnglish.py

التابع (wordsMatch and lettersMatch and isEnglish()) يعيد True فقط إذا كان كل من المتغيران wordsMatch and lettersMatch يحويان القيمة True لذلك قمنا بتمرير هذين المتغيرين ضمن عبارة منطقية and لتنفيذ العملية المنطقية

إذا كانت قيمة كل من المتغيران wordMatch and letterMatch هي True فإن التابع isEnglish() سوف يعيد القيمة True وهذا يعني أن الرسالة هي عبارة عن نص باللغة الانجليزية.