ENG HASAN HAJJAR

نبذة عن صاحب الكتاب

المهندس حسن حجار، حاصل على درجة البكالوريوس في الهندسة المعلوماتية من جامعة حلب في عام 2022 م، حيث تشترك تلك الدرجة مع معرفته الواسعة في مجال تطوير البرمجيات وتكنولوجيا المعلومات.

يُعتبر المهندس حجار قياديًا في مجاله، حيث قام بالعمل في عدة شركات برمجية ومشاريع مستقلة منذ كان طالباً جامعياً في المرحلة الأولى. يتميز بالمرونة والقدرة على التكيف مع تحديات البرمجة المعقدة، ويقدم حلاً فعّالًا وموثوقًا في معظم المشاريع التي يشارك فيها.

في عام 2024 م، يتابع المهندس حجار دراسته في برنامج الماجستير بتخصص الهندسة الطبية والحيوية، حيث ينعكس اهتمامه بالابتكار وتقنيات التطور في مجال الرعاية الصحية والطبية.

بجانب عمله الأكاديمي والمهني، يشارك حسن خبرته مع مجتمع البرمجة من خلال تدريس اللغات البرمجية والمواضيع ذات الصلة. يعمل أيضاً كاستشاري برمجي، حيث يقدم حلاً مخصصًا ومبتكرًا لتحسين أداء البرمجيات وتلبية احتياجات العملاء.

بفضل خلفيته الشاملة وإلمامه العميق بمجالات هندسة المعلومات وتكنولوجيا المعلومات، يعد المهندس حسن حجار مصدرًا قيمًا للمعرفة والاستشارات في عالم البرمجة والتقنية.

ENG HASAN HAJJAR

الفصل الأول

مقدمة

أولاً: مقدمة عن البايثون:

بايثون (بالإنجليزية: Python) هي لغة برمجة، عالية المستوى سهلة التعلم مفتوحة المصدر قابلة للتوسيع، تعتمد أسلوب البرمجة الكائنية (OOP). لغة بايثون هي لغة مُفسَّرة، ومُتعدِدة الاستخدامات، وتستخدم بشكل واسع في العديد من المجالات، كبناء البرامج المستقلة باستخدام الواجهات الرسومية وفي تطبيقات الويب. بشكل عام، يمكن استخدام بايثون لعمل البرامج البسيطة للمبتدئين، ولإنجاز المشاريع الضخمة في الوقت نفسه. غالباً ما يُنصح المبتدؤون في ميدان البرمجة بتعلم هذه اللغة لأنها من بين أسرع اللغات البرمجية تعلماً.

ثانياً: لمحة تاريخية عن البايثون:

طُوِّرت بايثون في معهد الرياضيات والمعلوماتية الهولندي (CWI) في مدينة أمستردام على يد جايدو فان روسم في أواخر ثمانينات القرن العشرين، وكان أول إعلان عنها في عام 1991م. كُتبت

نواة اللغة باستعمال لغة سي. أطلق روسم الاسم «بايثون» على لغته تعبيرًا عن إعجابه بفِرقَة مسرحية هزلية شهيرة من بريطانيا، كانت تطلق على نفسها اسم مونتي بايثون.

تتميز بايثون بمجتمعها النشط، كما أن لها الكثير من المكتبات البرمجية ذات الأغراض الخاصة التي برمجها أشخاص من ذلك المجتمع. مثلاً، هناك مكتبة باي جايم التي توفر مجموعة من الدوال من أجل برمجة الألعاب. يمكن لبايثون أيضاً التعامل مع العديد من أنواع قواعد البيانات مثل mysql وغير ذلك.

تدعم بايثون أنماط برمجة متعددة هي التوجيه الكائني، البرمجية غرضية التوجه والبرمجة الوظيفية. تُستخدم بايثون عادةً مثل العديد من لغات البرمجة الديناميكية كلغة برمجة نصية. بايثون لديها نموذج مفتوح للتطوير، قائم على مجتمع بايثون البرمجي ومدعوم من مؤسسة برمجيات بايثون.

ثالثاً: استخدامات لغة البايثون:

من الاستخدامات:

- تطوير الويب (من جانب الخادم).
- تطوير البرمجيات.
- الرياضيات.
- البرمجة النصية للنظام.

ماذا يمكن أن تفعل بايثون؟

- يمكن استخدام بايثون على الخادم لإنشاء تطبيقات الويب.
- يمكن استخدام Python جنبًا إلى جنب مع البرامج لإنشاء مهام سير العمل.
- يمكن لبيثون الاتصال بأنظمة قواعد البيانات. ويمكنه أيضًا قراءة الملفات وتعديلها.
- يمكن استخدام بايثون للتعامل مع البيانات الضخمة وإجراء العمليات الحسابية المعقدة.

- يمكن استخدام بايثون للنماذج الأولية السريعة، أو لتطوير البرمجيات الجاهزة للإنتاج.
- لماذا بايثون؟
- تعمل لغة Python على منصات مختلفة (Windows، وMac، وLinux، و Raspberry Pi، وما إلى ذلك).
- لدى Python بناء جملة بسيط مشابه للغة الإنجليزية.
- لدى بايثون بناء جملة يسمح للمطورين بكتابة برامج ذات أسطر أقل من بعض لغات البرمجة الأخرى.
- تعمل بايثون على نظام مترجم فوري، مما يعني أنه يمكن تنفيذ التعليمات البرمجية بمجرد كتابتها. وهذا يعني أن النماذج الأولية يمكن أن تكون سريعة جدًا.
- يمكن التعامل مع بايثون بطريقة إجرائية، أو بطريقة موجهة للكائنات، أو بطريقة وظيفية.

انتهى الفصل

ENG HASAN HAJJAR

الفصل الثاني

مقارنة البايثون مع لغات البرمجة المعاصرة

أولاً : ميّزات البايثون:

لغة البرمجة Python تتميز بعدة ميزات هامة تجعلها شديدة الشعبية ومفيدة لمجموعة واسعة من التطبيقات. أقدم لك بعض الميزات الهامة التي تميز Python عن باقي لغات البرمجة:

1. قابلية القراءة والكتابة:

- Python تتمتع ببنية بسيطة وقواعد صارمة، مما يجعل الشفرة سهلة القراءة والكتابة. هذا يساعد في تسهيل التعاون بين المطورين وفهم الشفرة حتى للأشخاص الذين لا يعرفون Python بشكل كبير.

2. تعدد الاستخدامات:

- يمكن استخدام Python في مجموعة واسعة من التطبيقات بما في ذلك تطوير الويب، والحوسبة العلمية، والتحليل البياني، والذكاء الاصطناعي، وتطوير تطبيقات السحابة، وأكثر من ذلك.

3. مجتمع نشط ومكتبات غنية:

- يتمتع Python بمجتمع مطورين كبير ونشط، مما يعني وجود العديد من المكتبات والإطارات الجاهزة التي تسهل على المطورين إنشاء تطبيقاتهم بسرعة وكفاءة.

4. توافق مع معايير صناعية:

- Python يلتزم بمعايير صناعية قياسية، مما يجعلها مناسبة للاستخدام في العديد من الصناعات والتطبيقات التجارية.

5. البرمجة الديناميكية:

- Python تعتبر لغة ديناميكية، مما يعني أن المتغيرات لا تحتاج إلى تعريف أنواعها مسبقًا، ويمكن تغيير نوع المتغير أثناء تشغيل البرنامج. هذا يجعل البرمجة أكثر مرونة وسهولة.

6. التوجيه الكائني:

- Python تدعم التوجيه الكائني (OOP)، مما يسمح للمطورين بتنظيم الشفرة بشكل أفضل وإعادة استخدام الأكواد بشكل فعال.

7. الواجهات البسيطة:

- Python يسهل تعلمه واستخدامه، وهو مناسب للمبتدئين بسبب بنيته البسيطة والتي تشبه اللغة الإنجليزية.

8. ميزة التكامل:

- Python يتكامل بشكل جيد مع لغات أخرى مثل C و C++، ويوفر واجهات برمجة تطبيقات (API) للتفاعل مع العديد من التقنيات والخدمات.

9. متعدد المنصات:

- Python يمكن تشغيله على معظم أنظمة التشغيل الرئيسية (Windows، Linux، macOS)، مما يعني أن التطبيقات المكتوبة بـ Python قابلة للتشغيل على مختلف البيئات.

10. التطوير السريع:

- Python يسمح بتطوير سريع وفعال، حيث يمكن للمطورين بناء تطبيقاتهم بسرعة باستخدام المكتبات الجاهزة والإطارات.

تلك هي بعض الميزات الهامة التي تجعل لغة البرمجة Python مميزة وشائعة في مجال تطوير البرمجيات.

ثانياً : البايثون واللغات الأخرى:

لغة البرمجة Python تختلف عن العديد من لغات البرمجة الأخرى، مثل C++, HTML, JavaScript، و Java، من حيث الصفات والميزات. فيما يلي بعض الفروق البارزة بين لغة البرمجة Python وبعض اللغات الأخرى:

1. التركيز على القراءة والكتابة:

- Python تعتبر من لغات البرمجة سهلة القراءة والكتابة، مما يجعلها مثالية للمبتدئين وتسهل على المطورين فهم الشفرة.

2. الكفاءة في التطوير:

- Python تعزز فعالية التطوير بفضل خصائصها الديناميكية وسهولة التفاعل مع الكود، مما يقلل من وقت التطوير بشكل عام.

3. النمط التفاعلي:

- Python يقدم مفسرًا تفاعليًا يتيح للمستخدمين تجربة الأوامر فوريًا، وهو أمر مفيد خلال عملية التطوير والاختبار.

4. النموذج الديناميي:

- Python هي لغة برمجة ديناميكية، حيث لا يلزم تعريف النوع مسبقًا، مما يسهل على المطورين كتابة الشفرة بشكل أسرع.

5. الإدارة الآلية للذاكرة:

- Python يتمتع بإدارة ذاكرة آلية، مما يقلل من الحاجة إلى التحكم اليدوي في الذاكرة مقارنة بلغات مثل C++.

6. الوسوم والتنسيق:

- HTML وتقنيات الويب الأخرى (مثل JavaScript وCSS) تستخدم وسوم لتحديد هيكل الصفحة وتنسيقها، بينما Python يستخدم تنسيق الهوية البيانية لتنظيم الشفرة.

7. البرمجة الشيئية:

- C++ وJava تشجعان على البرمجة الشيئية، في حين أن Python تدعم البرمجة الشيئية وتسمح أيضًا بأسلوب برمجيات أخرى مثل البرمجة الإجرائية.

8. الأداء:

- لغات مثل C++ توفر أداءً عالي المستوى، بينما Python تتمتع بأداء معقول ولكن ليس بنفس مستوى الأداء الذي يقدمه C++.

9. المجتمع والدعم:

- Python لديها مجتمع كبير ونشط من المطورين، مما يعزز من توفر مكتبات وأدوات مفيدة.

10. الاستخدامات:

- Python يستخدم على نطاق واسع في مجالات مثل التطوير الويب والذكاء الاصطناعي، بينما C++ يستخدم في تطوير البرامج ذات الأداء العالي مثل الألعاب.

تلك هي بعض الفروق الرئيسية بين Python وبعض لغات البرمجة الشهيرة الأخرى. يجب على المطور اختيار اللغة التي تناسب احتياجات مشروعه بناءً على المتطلبات والأهداف المحددة.

2-1- البايثون وال C++ :

هناك العديد من الفروق بين لغة البرمجة Python ولغة السي بلس بلس (C++). فيما يلي بعض الفروق الرئيسية بينهما:

1. النمط البرمجي:

- Python هي لغة برمجة تفسيرية (interpreted) وتدعم البرمجة الديناميكية، حيث يتم تحليل وتنفيذ الشفرة خطوة بخطوة. بينما C++ هي لغة برمجة تجميعية (compiled) وتدعم البرمجة الثابتة وتحتاج إلى ترجمة قبل تنفيذها.

2. إدارة الذاكرة:

- في C++، يتحكم المبرمج تمامًا في إدارة الذاكرة، مما يعني أنه يجب عليه القيام بعمليات تخصيص وتحرير الذاكرة يدويًا. بينما في Python، يتمتع جار التخزين الضمني (garbage collector) بالقدرة على إدارة تلك العمليات تلقائيًا، مما يقلل من خطأ الإدارة اليدوية للذاكرة.

3. الكتابة والقراءة:

- Python تُعتبر أكثر سهولة في الكتابة والقراءة مقارنة بـ C++، حيث تتيح للمبرمجين كتابة كميات أقل من الشفرة لتحقيق نفس الوظائف.

4. التوجيه الكائني:

- كلاهما يدعم مفاهيم البرمجة الشيئية (OOP)، ولكن الدعم في C++ قوي ومبني بشكل أساسي حول التوجيه الكائني، بينما يُعتبر التوجيه الكائني في Python جزءًا من اللغة ولكن ليس بنفس القوة.

5. الأداء:

- C++ عادةً ما تكون أداؤها أفضل من Python، خاصة في تطبيقات تتطلب أداءً عاليًا، مثل الألعاب أو البرامج ذات الحسابات الكبيرة.

6. استخدامات مختلفة:

- C++ تستخدم على نطاق واسع في تطبيقات النظم، البرمجة المضمنة، وتطوير الألعاب، بينما يستخدم Python بشكل شائع في التطوير الويب، الحوسبة العلمية، والتطبيقات الخفيفة.

7. مكتبات وإطارات العمل:

- Python تتمتع بمجموعة واسعة من المكتبات والإطارات الجاهزة التي تساعد في تسريع عملية التطوير، بينما C++ يتطلب في بعض الأحيان كتابة المزيد من الشفرة لتحقيق نفس الغرض.

8. سهولة التعلم:

- Python عادةً ما تكون أسهل للتعلم والاستفادة منها للمبتدئين مقارنة بـ C++، حيث تقدم بنية بسيطة وصفرية.

يُلاحظ أن اختيار اللغة يعتمد على احتياجات المشروع وتفضيلات المبرمج، وكل لغة لها ميزاتها واستخداماتها الملائمة.

2-2- البايثون و ال JavaScript :

هناك العديد من الفروق بين لغة البرمجة Python ولغة الجافا سكريبت (JavaScript). فيما يلي بعض الفروق الرئيسية بينهما:

1. الاستخدام:

- Python غالبًا ما يُستخدم في مجالات مثل التطوير الويب (خاصة باستخدام إطار العمل Django)، الحوسبة العلمية، والتطبيقات العامة. بينما يُستخدم JavaScript أساسًا لبرمجة تفاعل المستخدم في صفحات الويب وتطوير الواجهات الرسومية.

2. الموقع الذي يتم تنفيذ الكود فيه:

- Python يُعتبر لغة خادم (Server-side language) حيث يتم تنفيذ الكود على الخادم. في المقابل، JavaScript هو لغة عميل (Client-side language) تنفذ على متصفح الويب لديك.

3. النموذج البرمجي:

- Python تدعم البرمجة الشيئية (OOP) والبرمجة الإجرائية، بينما JavaScript تدعم أيضًا البرمجة الشيئية وتعتبر أساسًا للبرمجة الشيئية في سياق تطوير الويب.

4. تحكم الذاكرة:

- Python تتمتع بإدارة ذاكرة آلية وجار التخزين الضمني، مما يساعد على تقليل أخطاء الإدارة اليدوية للذاكرة. في حين أن JavaScript يعتمد على جميع التحكم في الذاكرة على متصفح الويب والمحرك الجافا سكريبت الذي يستخدمه.

5. التنفيذ:

- Python يحتاج إلى مفسر (interpreter) لتنفيذ الشفرة، بينما يتم ترجمة وتنفيذ شفرة JavaScript مباشرة على متصفح الويب.

6. مكتبات وإطارات العمل:

- Python تحظى بمجموعة كبيرة من المكتبات والإطارات الجاهزة مثل Django و Flask. JavaScript يستفيد من العديد من المكتبات والإطارات لتطوير واجهات المستخدم وتفاعل المستخدم، مثل React وAngular.

7. التعلم والبداية:

- Python تُعتبر عادةً مثالية للمبتدئين بسبب بنيتها البسيطة وسهولة فهمها. JavaScript أيضًا يعتبر متاحًا للمبتدئين ويُستخدم على نطاق واسع في تطوير الويب.

8. الاستخدام خارج المتصفح:

- Python يمكن استخدامه في تطبيقات السطح المكتبي والحوسبة العلمية وأغراض أخرى، بينما يُستخدم JavaScript بشكل أساسي داخل المتصفح ولكن أصبح أيضًا يُستخدم في مجالات مثل تطوير تطبيقات الخوادم باستخدام Node.js.

تلك هي بعض الفروق الرئيسية بين لغة البرمجة Python ولغة الجافا سكريبت. يتعلم المطورون استخدام اللغات المناسبة وفقًا لاحتياجات مشروعهم وسياق العمل.

2-3- البايثون والجافا :

هناك العديد من الفروق بين لغة البرمجة Python ولغة Java. فيما يلي بعض الفروق الرئيسية بينهما:

1. النمط البرمجي:

- Python تدعم البرمجة الشيئية (OOP) والبرمجة الإجرائية، بينما Java تُركز أساسًا على البرمجة الشيئية.

2. إدارة الذاكرة:

- في Python، يتمتع "Automatic Garbage Collector" جار التخزين الضمني بالقدرة على إدارة تخصيص وتحرير الذاكرة تلقائيًا ، بينما في Java يتم تحكم المبرمج في تلك العمليات.

شرح متعمق:

في لغة البرمجة Python، يتم تنظيف الذاكرة تلقائيًا باستخدام آلية تسمى "جار التخزين الضمني" أو "garbage collector". الذاكرة الضمنية تتيح للبرنامج تخصيص مساحة في الذاكرة لتخزين المتغيرات والكائنات أثناء تشغيل البرنامج، وعندما لا يكون هناك حاجة لهذه المتغيرات أو الكائنات، يقوم جار التخزين الضمني بتحرير الذاكرة المخصصة لها.

الفائدة الرئيسية هنا هي أن المبرمج لا يحتاج إلى القلق بشكل مباشر حول تحرير الذاكرة بنفسه. في Python، يمكن للمبرمج كتابة الشفرة دون الحاجة إلى دعم الذاكرة بطرق تفصيلية، وذلك لأن جار التخزين الضمني يدير هذه العملية بشكل تلقائي.

بالمقابل، في لغة البرمجة Java، يُكلِّف المبرمج بشكل أكبر بإدارة الذاكرة. يجب عليه أن يتأكد من تحرير الذاكرة التي تم تخصيصها يدويًا بمرور الوقت. هذا يتطلب اهتمامًا دقيقًا من المبرمج لتتبع الكائنات التي تحتاج إلى تحرير الذاكرة وتحديدها. هذه العملية تتسبب في إمكانية حدوث أخطاء متعلقة بالذاكرة مثل تسريب الذاكرة إذا لم يتم التعامل معها بشكل صحيح.

3. الأداء:

- Java عادةً ما تكون أداؤها أفضل من Python، خاصة في التطبيقات الكبيرة والمعقدة. يُعتبر Java أكثر فعالية في الأداء وأداءً على المدى الطويل.

4. نظام التشغيل:

- Java تعتبر من لغات البرمجة المتعددة المنصات (cross-platform)، مما يعني أن التطبيقات المكتوبة في Java يمكن تشغيلها على أنظمة تشغيل مختلفة دون الحاجة إلى إعادة كتابة الشفرة. Python أيضًا تدعم هذه الميزة بشكل جزئي، ولكن Java تفوز في هذا السياق.

5. تنفيذ الشفرة:

- Python تعتبر لغة مفسرة (interpreted) حيث يتم تنفيذ الشفرة خطوة بخطوة. في المقابل، Java تعتبر لغة مترجمة (compiled) حيث يتم ترجمة الشفرة إلى لغة بينية تعتبر محمولة بين الأنظمة.

6. الكتابة والقراءة:

- Python تعتبر أكثر سهولة في الكتابة والقراءة مقارنة بـ Java، حيث تقدم بنية بسيطة وقواعد أقل صرامة.

7. التعامل مع الخطأ:

- Java يتطلب التعامل الصريح مع استثناءات البرمجة (exceptions)، بينما Python يتيح التعامل مع الأخطاء بطريقة أكثر تساهلاً.

8. استخدامات مختلفة:

- Java تُستخدم على نطاق واسع في تطبيقات المؤسسات، تطوير الواجهات الرسومية، وتطبيقات الأعمال. بينما Python تُستخدم بشكل شائع في تطوير الويب، الحوسبة العلمية، وتطوير البرمجيات السريعة.

9. مكتبات وإطارات العمل:

- كلاهما يتمتع بمجموعة واسعة من المكتبات والإطارات الجاهزة، ولكن Python تُشهد على وجود مجتمع كبير من المطورين يسهم في إنشاء وصيانة مكتبات وإطارات قوية.

تلك هي بعض الفروق الرئيسية بين لغة البرمجة Python ولغة Java. يجب على المبرمجين اختيار اللغة التي تناسب احتياجات مشروعهم والسياق الذي يعملون فيه.

2-4- البايثون وال php :

هناك العديد من الفروق بين لغة البرمجة Python ولغة PHP. فيما يلي بعض الفروق الرئيسية بينهما:

1. الاستخدام الرئيسي:

- Python غالبًا ما يُستخدم في تطوير البرمجيات العامة، الحوسبة العلمية، وتطبيقات الويب. بينما PHP أُنشِئت أصلاً لتطوير تطبيقات الويب وتفاعلها مع قواعد البيانات.

2. النموذج البرمجي:

- Python تدعم البرمجة الشيئية (OOP) والبرمجة الإجرائية، بينما PHP تركز أساسًا على البرمجة الإجرائية، ولكن تدعم أيضًا البرمجة الشيئية.

3. الأداء:

- عمومًا، PHP يتميز بأداء جيد في تطوير تطبيقات الويب والصفحات الديناميكية. Python أيضًا يُستخدم في تطوير الويب، ولكن PHP يكون غالبًا أكثر فعالية في هذا السياق.

4. تفاعل مع قواعد البيانات:

- PHP تم تصميمها بشكل أساسي للتفاعل مع قواعد البيانات، وهي متكاملة بشكل أفضل مع MySQL وقواعد البيانات الأخرى. Python أيضًا يدعم تفاعل ممتاز مع قواعد البيانات، ويستخدم بشكل شائع مع SQLite و PostgreSQL.

5. الكتابة والقراءة:

- Python تُعتبر أكثر سهولة في الكتابة والقراءة مقارنة بـ PHP، وتقدم بنية بسيطة وصفرية.

6. إدارة الذاكرة:

- PHP تقوم بإدارة الذاكرة تلقائيًا بشكل أفضل من Python، خاصة في سياق تطوير تطبيقات الويب.

7. نظام التشغيل:

- Python تعتبر من لغات البرمجة المتعددة المنصات (cross-platform)، مما يعني أن تطبيقات Python يمكن تشغيلها على أنظمة متعددة دون تعديل. PHP أيضًا تعمل على معظم خوادم الويب، ولكن يجب تكوينها بشكل مناسب لتشغيلها على نظام معين.

8. المكتبات والإطارات:

- كلاهما يحظون بدعم قوي من المكتبات والإطارات الجاهزة. Python يشتهر بإطارات مثل Django و Flask، في حين أن PHP يستفيد من إطارات مثل Laravel و Symfony.

9. التوجيه الكائني:

- Python تعتبر لغة مفتوحة ومرنة تدعم التوجيه الكائني بفعالية، بينما PHP تشجع على التركيز على البرمجة الإجرائية.

10. التعلم والمجتمع:

- Python غالبًا ما يعتبر مثاليًا للمبتدئين بسبب بنيته البسيطة وسهولة فهمه. PHP أيضًا يمكن أن يكون مناسبًا للمبتدئين، خاصة في سياق تطوير تطبيقات الويب.

هذه بعض الفروق الرئيسية بين Python و PHP. يتوقف اختيار اللغة على احتياجات المشروع والتفضيلات الشخصية للمطور.

2-5- البايثون وال C# :

هناك العديد من الفروق بين لغة البرمجة Python ولغة C#. فيما يلي بعض الفروق الرئيسية بينهما:

1. النمط البرمجي:

- Python تدعم البرمجة الشيئية (OOP) والبرمجة الإجرائية، بينما C# تركز بشكل أساسي على البرمجة الشيئية.

2. النظام النموذجي:

- C# تستند إلى نظام النموذج الصارم (strongly-typed)، حيث يجب تحديد نوع كل متغير بوضوح، بينما Python تستخدم نموذج النموذج الضعيف (dynamically-typed)، مما يعني أنه يمكن تعيين نوع المتغير تلقائيًا دون الحاجة إلى تعيينه.

3. البيئة والمنصات:

- C# تعتبر جزءًا من بيئة تطوير Microsoft (Microsoft ecosystem) وتدعم بشكل أساسي نظام التشغيل Windows. بينما Python يمكن تشغيله على أنظمة التشغيل المختلفة بما في ذلك Windows وLinux وMac OS.

4. مجتمع المطورين والمكتبات:

- Python تتمتع بمجتمع كبير ونشط من المطورين، مما يعني توفر مكتبات وإطارات واسعة النطاق. C# أيضًا لديه مجتمع قوي ويستفيد من تكامله مع منصة .NET، مما يوفر مجموعة كبيرة من المكتبات والأدوات.

5. أداء التطبيقات:

- عمومًا، C# يُعتبر أكثر كفاءة في الأداء من Python، خاصة في تطبيقات الويب الكبيرة والمعقدة وتطبيقات سطح المكتب.

6. التطوير المتكامل:

- C# يتمتع بدعم متكامل للتطوير مع Visual Studio، بينما Python يمكن تطويره باستخدام محررات النص العادية وبيئات تطوير متكاملة مثل PyCharm وVS Code.

7. الاستخدامات الرئيسية:

- C# يستخدم على نطاق واسع في تطوير تطبيقات Windows، تطبيقات سطح المكتب، وتطبيقات الألعاب، بينما Python يُستخدم بشكل شائع في تطوير الويب، الحوسبة العلمية، والتطبيقات العامة.

8. المنصات العابرة:

- Python تستخدم على نطاق واسع في تطوير التطبيقات عبر المنصات، بينما C# يمكن أيضًا استخدامه في هذا السياق من خلال منصة .NET.

تلك هي بعض الفروق الرئيسية بين Python وC#. يجب على المطور اختيار اللغة التي تتناسب مع احتياجات مشروعه والتفضيلات الشخصية.

ثالثاً : الأشخاص القادرين على تعلم البايثون:

لغة البرمجة Python تعتبر مناسبة لشريحة واسعة من الأشخاص، ويمكن لمجموعة متنوعة من الأفراد التعلم واستخدام Python بشكل فعّال. إليك قائمة بالأشخاص الذين يمكن أن يستفيدوا من تعلم لغة البرمجة Python:

1. المبتدئين:

- Python تُعتبر إحدى أفضل لغات البرمجة للمبتدئين. بفضل بنيتها البسيطة وسهولة قراءة الشفرة، يمكن للمبتدئين التعلم بسرعة وفهم مفاهيم البرمجة.

2. المطورين:

- سواء كانوا مطورين ويب، مطورين حوسبة علمية، أو مطورين تطبيقات، يستخدم العديد من المحترفين Python في أعمالهم اليومية.

3. الطلاب:

- Python تُستخدم على نطاق واسع في المؤسسات التعليمية والجامعات. الطلاب يمكن أن يستخدموا Python في مشاريعهم وأعمالهم الدراسية.

4. الباحثين العلميين:

- يستخدم العديد من الباحثين العلميين Python في تحليل البيانات، والمحاكاة، والحوسبة العلمية بسبب العديد من المكتبات المتقدمة المتاحة.

5. المحللين البيانيين:

- Python يعتبر أداة فعّالة لتحليل البيانات والإحصائيات، ويستخدمه المحللون البيانيون لاستخراج الأنماط والتحليلات من البيانات.

6. المهندسين:

- يمكن للمهندسين استخدام Python في تطوير البرمجيات والتحكم في الأجهزة، وذلك بفضل إمكانياتها في التفاعل مع الأجهزة الإلكترونية.

7. مطوري الذكاء الاصطناعي والتعلم الآلي:

- Python تعتبر لغة محبوبة بين مطوري الذكاء الاصطناعي والتعلم الآلي، حيث توفر العديد من المكتبات المتخصصة مثل TensorFlow وPyTorch.

8. المطورين العاملين في مجال التطوير الويب:

- Python تُستخدم بشكل شائع في تطوير الويب، سواء كان ذلك باستخدام إطارات ومكتبات مثل Django وFlask، أو في تطوير الجزء الخلفي للتطبيقات.

9. مطوري الألعاب:

- Python يمكن أن يكون له دور في تطوير الألعاب، سواء كان ذلك في البرمجة الرئيسية أو في إنشاء أدوات وسكربتات لدعم عملية التطوير.

بشكل عام، Python تقدم بيئة تعلم مفتوحة وميسرة لمجموعة واسعة من الأفراد، من المبتدئين إلى المحترفين، ومن مختلف المجالات الفنية.

انتهى الفصل

ENG HASAN HAJJAR

الفصل الثالث

بيئة الباثيون

أولاً: بناء جملة بايثون:

تم تصميم بايثون لسهولة القراءة، ولها بعض أوجه التشابه مع اللغة الإنجليزية مع تأثير الرياضيات.

تستخدم بايثون أسطرًا جديدة لإكمال الأمر، على عكس لغات البرمجة الأخرى التي غالبًا ما تستخدم الفواصل المنقوطة أو الأقواس.

تعتمد بايثون على المسافة البادئة، باستخدام المسافة البيضاء، لتحديد النطاق؛ مثل نطاق الحلقات والوظائف والفئات. غالبًا ما تستخدم لغات البرمجة الأخرى الأقواس المتعرجة لهذا الغرض.

مثال:

```
print("Hello, World!")
```

يطبع العبارة `Hello, World!`.

ثانياً: تثبيت بايثون :

سيتم تثبيت لغة python بالفعل على العديد من أجهزة الكمبيوتر الشخصية وأجهزة Mac.

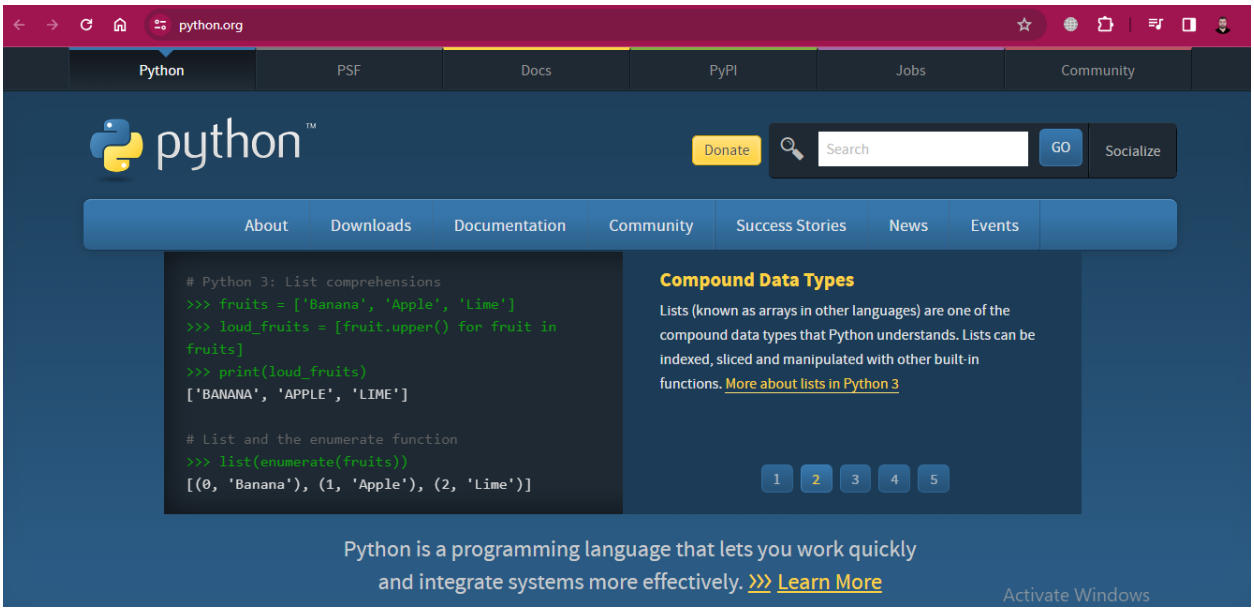
للتحقق مما إذا كان لديك python مثبتًا على جهاز كمبيوتر يعمل بنظام Windows، ابحث في شريط البداية عن Python أو قم بتشغيل ما يلي في سطر الأوامر (cmd.exe):

```
C:\Users\Your Name>python -version
```

للتحقق مما إذا كان لديك python مثبتًا على نظام Linux أو Mac، ثم على Linux افتح سطر الأوامر أو على Mac افتح Terminal واكتب:

```
python -version
```

إذا وجدت أنه ليس لديك لغة Python مثبتة على جهاز الكمبيوتر الخاص بك، فيمكنك تنزيلها مجانًا من الموقع التالي: https://www.python.org/



الشكل (1) – موقع البايثون الأساسي

ثالثاً: بداية سريعة :

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

طريقة تشغيل ملف بايثون هي كالتالي في سطر الأوامر:

```
C:\Users\Your Name>python helloworld.py
```

حيث "helloworld.py" هو اسم ملف python الخاص بك.

لنكتب أول ملف بايثون لدينا، يسمى helloworld.py، والذي يمكن إجراؤه باستخدام أي محرر نصوص.

helloworld.py

```
print("Hello, World!")
```

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

```
C:\Users\Your Name>python helloworld.py
```

يجب أن يكون الناتج كما يلي:

```
Hello, World!
```

تهانينا، لقد قمت بكتابة وتنفيذ أول برنامج بايثون لك.

ENG HASAN HAJJAR

رابعاً: سطر أوامر بايثون:

لاختبار كمية قصيرة من التعليمات البرمجية في بايثون، في بعض الأحيان يكون من الأسرع والأسهل عدم كتابة التعليمات البرمجية في ملف. أصبح هذا ممكنًا لأنه يمكن تشغيل Python كسطر أوامر بحد ذاته.

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

```
C:\Users\Your Name>python
```

أو، إذا لم يعمل الأمر"python" ، يمكنك تجربة:"py"

```
C:\Users\Your Name>py
```

من هناك يمكنك كتابة أي لغة بايثون، بما في ذلك مثالنا helloworld الذي سبق ذكره في البرنامج التعليمي:

```
C:\Users\Your Name>python
```

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

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

```
>>> print("Hello, World!")
```

والتي سوف تكتب "مرحبا بالعالم!" في سطر الأوامر:

```
C:\Users\Your Name>python
```

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

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

>>> print("Hello, World!")

Hello, World!
```

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

```
exit()
```

خامساً: تنفيذ بناء جملة بايثون Python Syntax :

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

```
>>> print("Hello, World!")

Hello, World!
```

أو عن طريق إنشاء ملف python على الخادم، باستخدام ملحق الملف.py ، وتشغيله في سطر الأوامر:

```
C:\Users\Your Name>python myfile.py
```

سادساً: مسافة بادئة بايثون :

تشير المسافة البادئة إلى المسافات في بداية سطر التعليمات البرمجية.

بينما في لغات البرمجة الأخرى تكون المسافة البادئة في التعليمات البرمجية مخصصة لسهولة القراءة فقط، فإن المسافة البادئة في بايثون مهمة جدًا.

تستخدم بايثون المسافة البادئة للإشارة إلى كتلة من التعليمات البرمجية.

مثال:

```
if 5 > 2:
  print("Five is greater than two!")
```

- سوف تعطيك بايثون خطأً إذا تخطيت المسافة البادئة.

مثال:

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

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

مثال:

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

يجب عليك استخدام نفس عدد المسافات في نفس كتلة التعليمات البرمجية، وإلا فسوف تعطيك بايثون خطأ:

مثال:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

هنا سيعطي خطأ.

انتهى الفصل

ENG HASAN HAJJAR

الفصل الرابع

التعليقات في الباثيون

أولاً: مقدمة:

تتمتع Python بإمكانية التعليق لغرض التوثيق داخل التعليمات البرمجية.

تبدأ التعليقات بـ #، وستعرض بايثون بقية السطر كتعليق:

مثال:

```
#This is a comment.
print("Hello, World!")
```

- يمكن استخدام التعليقات لشرح كود بايثون.
- يمكن استخدام التعليقات لجعل التعليمات البرمجية أكثر قابلية للقراءة.
- يمكن استخدام التعليقات لمنع التنفيذ عند اختبار التعليمات البرمجية.

ثانياً: إنشاء تعليق:

تبدأ التعليقات بـ #، وسوف تتجاهلها بايثون:

مثال:

```
#This is a comment
print("Hello, World!")
```

يمكن وضع التعليقات في نهاية السطر، وستتجاهل بايثون بقية السطر:

مثال:

```
print("Hello, World!") #This is a comment
```

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

مثال:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

ثالثاً: تعليقات متعددة الأسطر:

ليس لدى بايثون حقًا صيغة للتعليقات متعددة الأسطر.

لإضافة تعليق متعدد الأسطر، يمكنك إدراج # لكل سطر:

مثال:

```
#This is a comment
#written in
```

```
#more than just one line
print("Hello, World!")
```

أو، ليس تمامًا كما هو مقصود، يمكنك استخدام سلسلة متعددة الأسطر.

بما أن بايثون ستتجاهل القيم الحرفية للسلسلة التي لم يتم تعيينها لمتغير، فيمكنك إضافة سلسلة متعددة الأسطر (علامات الاقتباس الثلاثية) في التعليمات البرمجية الخاصة بك، ووضع تعليقك بداخلها:

مثال:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

طالما لم يتم تعيين السلسلة لمتغير، ستقرأ بايثون الكود، لكنها تتجاهله بعد ذلك، وتكون قد قمت بعمل تعليق متعدد الأسطر.

انتهى الفصل

ENG HASAN HAJJAR

الفصل الخامس

المتغيرات في الباثيون

أولاً: مقدمة:

في بايثون، يتم إنشاء المتغيرات عندما تقوم بتعيين قيمة لها:

مثال:

```
x = 5
y = "Hello, World!"
```

- ليس لدى بايثون أمر للإعلان عن متغير.

- المتغيرات عبارة عن حاويات لتخزين قيم البيانات.
- يتم إنشاء المتغير في اللحظة التي تقوم فيها بتعيين قيمة له لأول مرة.

مثال:

```
x = 5
y = "John"
print(x)
print(y)
```

لا يلزم التصريح عن المتغيرات بأي نوع معين، بل ويمكن تغيير نوعها بعد تعيينها.

مثال:

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

ثانياً: التحويل القسري Casting :

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

مثال:

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

ثالثاً: معرفة نوع المتغير:

يمكنك الحصول على نوع بيانات المتغير باستخدام الدالة type().

مثال:

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

سوف تتعلم المزيد عن أنواع البيانات والإرسال لاحقًا في هذا الكتاب.

رابعاً: اقتباسات مفردة أم مزدوجة :

يمكن الإعلان عن متغيرات السلسلة إما باستخدام علامات الاقتباس المفردة أو المزدوجة:

مثال:

```
x = "John"
# is the same as
x = 'John'
```

خامساً: حساسية اللغة للمتحولات :

أسماء المتغيرات حساسة لحالة الأحرف.

مثال:

```
a = 4
A = "Sally"
#A will not overwrite a
```

ENG HASAN HAJJAR

سادساً: أسماء المتغيرات :

يمكن أن يكون للمتغير اسم قصير (مثل x وy) أو اسم أكثر وصفًا (العمر، اسم السيارة، الحجم الإجمالي).

6-1- قواعد متغيرات بايثون:

- يجب أن يبدأ اسم المتغير بحرف أو بشرطة سفلية ولا يمكن أن يبدأ اسم المتغير برقم.
- **يمكن أن يحتوي اسم المتغير فقط على أحرف أبجدية رقمية وشرطات سفلية (A-z، 0-9، و _).**
- أسماء المتغيرات حساسة لحالة الأحرف (العمر والعمر والعمر هي ثلاثة متغيرات مختلفة).
  - لا يمكن أن يكون اسم المتغير أيًا من كلمات Python الأساسية.

مثال:

أسماء متحولات صحيحة:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

مثال:

أسماء متحولات غير صحيحة:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

تذكر أن أسماء المتغيرات حساسة لحالة الأحرف

6-2- أسماء متغيرة متعددة الكلمات :

قد يكون من الصعب قراءة الأسماء المتغيرة التي تحتوي على أكثر من كلمة واحدة. هناك العديد من التقنيات التي يمكنك استخدامها لجعلها أكثر قابلية للقراءة:

6-2-1- حالة الجمل :

كل كلمة، باستثناء الأولى، تبدأ بحرف كبير:

```
myVariableName = "John"
```

6-2-2- قضية باسكال:

تبدأ كل كلمة بحرف كبير:

```
MyVariableName = "John"
```

6-2-3- حالة الثعبان:

يتم فصل كل كلمة بحرف سفلي:

```
my_variable_name = "John"
```

سابعاً: الإسناد لمتغيرات بايثون:

7-1- - تعيين قيم متعددة:

العديد من القيم لمتغيرات متعددة. تتيح لك لغة Python تعيين قيم لمتغيرات متعددة في سطر واحد.

مثال:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

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

7-2- تعيين قيمة واحدة لمتغيرات متعددة:

يمكنك تعيين نفس القيمة لمتغيرات متعددة في سطر واحد:

مثال:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

7-3- فك المجموعة:

إذا كان لديك مجموعة من القيم في قائمة، في Tuple وما إلى ذلك يسمح لك Python باستخراج القيم إلى متغيرات. وهذا ما يسمى التفريغ.

مثال:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
```

```
print(y)
print(z)
```

سنتعرف على المزيد حول التفريغ في فصل لاحق.

ثامناً: طباعة (إخراج) المتغيرات:

تُستخدم الدالة print() غالبًا لإخراج المتغيرات.

مثال:

```
x = "Python is awesome"
print(x)
```

في الدالة print()، يمكنك إخراج متغيرات متعددة، مفصولة بفاصلة:

مثال:

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

يمكنك أيضًا استخدام عامل التشغيل + لإخراج متغيرات متعددة:

مثال:

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

لاحظ حرف المسافة بعد "Python " و"is "، فبدونهما ستكون النتيجة "Pythonisawesome".

بالنسبة للأرقام، يعمل الحرف + كعامل رياضي:

مثال:

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

في الدالة print()، عندما تحاول دمج سلسلة ورقم باستخدام عامل التشغيل +، ستعطيك Python خطأً:

مثال:

```
x = 5
y = "John"
print(x + y)
```

أفضل طريقة لإخراج متغيرات متعددة في الدالة print() هي الفصل بينها بفواصل، والتي تدعم أنواعًا مختلفة من البيانات:

مثال:

```
x = 5
y = "John"
print(x, y)
```

الخرج:

```
5John
```

ENG HASAN HAJJAR

تاسعاً: المتغيرات العامة:

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

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

مثال:

قم بإنشاء متغير خارج الدالة، واستخدمه داخل الدالة

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

الخرج:

```
Python is awesome
```

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

مثال:

قم بإنشاء متغير داخل دالة بنفس اسم المتغير العام

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)
```

```
myfunc()

print("Python is " + x)
```

الخرج:

```
Python is fantastic

Python is awesome
```

تاسعاً: الكلمة المحجوزة global:

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

لإنشاء متغير عام داخل دالة، يمكنك استخدام الكلمة الأساسية العالمية.

مثال:

إذا كنت تستخدم الكلمة الأساسية العمومية، فإن المتغير ينتمي إلى النطاق العام:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

الخرج:

```
Python is fantastic
```

استخدم أيضًا الكلمة الأساسية العامة إذا كنت تريد تغيير متغير عام داخل دالة.

مثال:

لتغيير قيمة متغير عام داخل دالة، قم بالإشارة إلى المتغير باستخدام الكلمة الأساسية العامة:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

الخرج:

```
Python is fantastic
```

انتهى الفصل

ENG HASAN HAJJAR

الفصل السادس

أنواع البيانات في الباثيون

أولاً:أنواع البيانات المضمنة:

في البرمجة، نوع البيانات هو مفهوم مهم. يمكن للمتغيرات تخزين بيانات من أنواع مختلفة، ويمكن للأنواع المختلفة القيام بأشياء مختلفة.

تحتوي لغة Python على أنواع البيانات التالية المضمنة افتراضيًا، في هذه الفئات:

Text Type:	`str`	**نوع النص**
Numeric Types:	`int`, `float`, `complex`	**الأنواع الرقمية**
Sequence Types:	`list`, `tuple`, `range`	**أنواع التسلسل**
Mapping Type:	`dict`	**نوع التعيين**
Set Types:	`set`, `frozenset`	**أنواع المجموعة**
Boolean Type:	`bool`	**النوع المنطقي**
Binary Types:	`bytes`, `bytearray`, `memoryview`	**الأنواع الثنائية**
None Type:	`NoneType`	**لا يوجد نوع**

جدول (1) الأنواع للبيانات في البايثون

ثانياً: الحصول على نوع البيانات:

يمكنك الحصول على نوع البيانات لأي كائن باستخدام الدالة type() :

مثال:

اطبع نوع بيانات المتغير x:

```
x = 5
print(type(x))
```

ثالثاً: تحديد نوع البيانات:

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

أمثلة:

```
x = "Hello World"
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
Hello World
<class 'str'>
```

أمثلة:

```
x = 20
#display x:
print(x)
```

```
#display the data type of x:

print(type(x))
```

الخرج:

```
20
<class 'int'>
```

أمثلة:

```
x = 20.5

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
20.5
<class 'float'>
```

أمثلة:

```
x = 1j

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
1j
<class 'complex'>
```

أمثلة:

```
x = ["apple", "banana", "cherry"]

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
['apple', 'banana', 'cherry']
<class 'list'>
```

أمثلة:

```
x = ["apple", "banana", "cherry"]

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
['apple', 'banana', 'cherry']
<class 'list'>
```

أمثلة:

```
x = ("apple", "banana", "cherry")

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
('apple', 'banana', 'cherry')
<class 'tuple'>
```

أمثلة:

```
x = range(6)
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
range(0, 6)
<class 'range'>
```

أمثلة:

```
x = {"name" : "John", "age" : 36}
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
{'name': 'John', 'age': 36}
<class 'dict'>
```

أمثلة:

```
x = {"apple", "banana", "cherry"}
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
{'cherry', 'apple', 'banana'}
<class 'set'>
```

أمثلة:

```
x = frozenset({"apple", "banana", "cherry"})
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
frozenset({'cherry', 'apple', 'banana'})
<class ‘frozenset’>
```

أمثلة:

```
x = True
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
True
<class 'bool'>
```

أمثلة:

```
x = b"Hello"
#display x:
print(x)
#display the data type of x:
print(type(x))
```

الخرج:

```
b'Hello'
<class 'bytes'>
```

أمثلة:

```
x = bytearray(5)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
bytearray(b'\x00\x00\x00\x00\x00')
<class 'bytearray'>
```

أمثلة:

```
x = memoryview(bytes(5))

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
<memory at 0x01368FA0>
<class 'memoryview'>
```

أمثلة:

```
x = None

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

None

<class 'NoneType'>

ثالثاً: ضبط نوع البيانات المحدد:

إذا كنت تريد تحديد نوع البيانات، يمكنك استخدام وظائف المنشئ التالية:

أمثلة:

```
x = str("Hello World")

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
Hello World
<class 'str'>
```

أمثلة:

```
x = int(20)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
20
<class 'int'>
```

أمثلة:

```
x = float(20.5)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
20.5
<class 'float'>
```

أمثلة:

```
x = complex(1j)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
1j
<class 'complex'>
```

أمثلة:

```
x = list(("apple", "banana", "cherry"))

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
['apple', 'banana', 'cherry']
<class 'list'>
```

أمثلة:

```
x = tuple(("apple", "banana", "cherry"))

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
('apple', 'banana', 'cherry')
<class 'tuple'>
```

أمثلة:

```
x = range(6)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
range(0, 6)
<class 'range'>
```

أمثلة:

```
x = dict(name="John", age=36)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
{'name': 'John', 'age': 36}
<class 'dict'>
```

أمثلة:

```
x = set(("apple", "banana", "cherry"))

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
{'apple', 'cherry', 'banana'}
<class 'set'>
```

أمثلة:

```
x = frozenset(("apple", "banana", "cherry"))

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
frozenset({'apple', 'banana', 'cherry'})
<class 'frozenset'>
```

أمثلة:

```
x = bool(5)

#display x:
```

```
print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
True
<class 'bool'>
```

أمثلة:

```
x = bytes(5)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
b'\x00\x00\x00\x00\x00'
<class 'bytes'>
```

أمثلة:

```
x = bytearray(5)

#display x:

print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
bytearray(b'\x00\x00\x00\x00\x00')
<class 'bytearray'>
```

أمثلة:

```
x = memoryview(bytes(5))

#display x:
```

```
print(x)

#display the data type of x:

print(type(x))
```

الخرج:

```
<memory at 0x0368AFA0>
<class 'memoryview'>
```

انتهى الفصل

ENG HASAN HAJJAR

الفصل السابع

الأعداد في الباثيون

7-1- مقدمة :

هناك ثلاثة أنواع رقمية في بايثون:

- int
- float
- complex

يتم إنشاء متغيرات الأنواع الرقمية عندما تقوم بتعيين قيمة لها:

مثال:

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

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

مثال:

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

7-2- الأعداد الصحيحة int :

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

مثال:

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

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

3-7- الأعداد العشرية float :

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

مثال:

```
x = 1.10
y = 1.0
z = -35.59

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

العدد العشري float : يمكن أيضًا أن يكون أرقامًا علمية بحرف "e" للإشارة إلى قوة الرقم 10.

مثال:

```
x = 35e3
y = 12E4
z = -87.7e100

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

7-4- الأعداد العقدية أو المركبة complex :

تتم كتابة الأعداد المركبة بحرف "j" باعتباره الجزء التخيلي:

مثال:

```
x = 3+5j
y = 5j
z = -5j

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

7-5- التحويل للأنواع casting :

يمكنك التحويل من نوع إلى آخر باستخدام الطرق int() و float() و complex():

مثال:

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

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
```

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

- ملاحظة: لا يمكنك تحويل الأرقام المركبة إلى نوع أرقام آخر.

7-6- الرقم عشوائي :

ليس لدى بايثون دالة لإنشاء أرقام عشوائية، لكن بايثون لديها وحدة (module) تدعى random والتي يمكن استخدامها لإنشاء أرقام عشوائية:

مثال:

```
import random

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

7-8- تحديد نوع متغير:

قد تكون هناك أوقات تريد فيها تحديد نوع لمتغير. بايثون هي لغة موجهة للكائنات، وبالتالي فهي تستخدم الفئات لتحديد أنواع البيانات، بما في ذلك أنواعها الأساسية. لذلك يتم إجراء عملية تحديد النوع في بايثون باستخدام وظائف:

Int()- إنشاء رقم صحيح من عدد صحيح حرفي، أو عدد عشري حرفي (عن طريق إزالة جميع الكسور العشرية) ، أو سلسلة حرفية (بشرط أن تمثل السلسلة عددًا صحيحًا)

Float()- ينشئ رقمًا عائمًا من عدد صحيح حرفي أو عدد عشري حرفي أو سلسلة حرفية (بشرط أن تمثل السلسلة عددًا عشريًا أو عددًا صحيحًا)

Str()- إنشاء سلسلة من مجموعة واسعة من أنواع البيانات، بما في ذلك السلاسل والأعداد الصحيحة والحرفية العائمة

مثال:

Integers:

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

مثال:

Floats:

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

مثال:

Strings:

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

انتهى الفصل

ENG HASAN HAJJAR

الفصل السابع

السلاسل في الباثيون

7-1- مقدمة :

السلاسل النصية في لغة بايثون محاطة بعلامات اقتباس مفردة أو علامات اقتباس مزدوجة.

يمكنك عرض سلسلة حرفية باستخدام الدالة print():

مثال:

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

7-2- إسناد سلسلة إلى متغير:

يتم تعيين سلسلة إلى متغير باستخدام اسم المتغير متبوعًا بعلامة يساوي والسلسلة:

مثال:

```
a = "Hello"
print(a)
```

7-3- السلاسل متعددة الأسطر :

- يمكنك تعيين سلسلة متعددة الأسطر لمتغير باستخدام ثلاث علامات اقتباس:

مثال:

You can use three double quotes:

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

أو ثلاثة علامات اقتباس واحدة:

مثال:

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

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

4-7- السلاسل هي المصفوفات:

مثل العديد من لغات البرمجة الشائعة الأخرى، السلاسل النصية في بايثون عبارة عن صفائف من البايتات تمثل أحرف يونيكود.

ومع ذلك، لا تحتوي لغة Python على نوع بيانات للأحرف، فالحرف الواحد هو ببساطة سلسلة بطول 1.

يمكن استخدام الأقواس المربعة للوصول إلى عناصر السلسلة.

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

```
a = "Hello, World!"
print(a[1])
```

7-5- حلقات مع السلسلة:

نظرًا لأن السلاسل عبارة عن مصفوفات، فيمكننا تكرار الأحرف الموجودة في سلسلة باستخدام حلقة for.

مثال: قم بالمرور على الحروف الموجودة في كلمة "banana":

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

7-6- طول السلسلة:

للحصول على طول السلسلة، استخدم الدالة len().

مثال:طباعة طول السلسلة a :

```
a = "Hello, World!"
print(len(a))
```

7-7- التحقق من السلسلة:

- للتحقق من وجود عبارة أو حرف معين في سلسلة، يمكننا استخدام الكلمة الأساسية **in**.

مثال: تحقق مما إذا كانت كلمة "free" موجودة في النص التالي:

```
"The best things in life are free!"
```

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

- للتحقق من عدم وجود عبارة أو حرف معين في سلسلة ما، يمكننا استخدام الكلمة الأساسية **not in**.

مثال: تحقق مما إذا كانت كلمة " expensive" غير موجودة في النص التالي:

```
"The best things in life are free!"
```

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

- استخدم مفهوم التحقق من السلاسل في عبارة if:

مثال: اطبع فقط في حالة عدم وجود كلمة " expensive":

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

7-8- تقطيع السلاسل:

يمكنك إرجاع نطاق من الأحرف باستخدام بناء سلسلة جزئية.

حدد دليل البداية ودليل النهاية، مفصولين بنقطتين، لإرجاع جزء من السلسلة.

مثال: اطبع الأحرف من الموضع 2 إلى الموضع 5 (غير متضمن):

```
b = "Hello, World!"
print(b[2:5])
```

- ملاحظة: الحرف الأول يحتوي على فهرس 0.

7-8-1- سلسلة جزئية من البداية:

من خلال ترك دليل البداية فارغ، سيبدأ المجال عند المحرف الأول:

مثال: اطبع المحارف من البداية إلى الموضع 5 (غير متضمنة):

```
b = "Hello, World!"
print(b[:5])
```

7-8-2- سلسلة جزئية حتى النهاية:

من خلال ترك فهرس النهاية فارغ، سينتقل المجال إلى النهاية:

مثال: اطبع المحارف من الموضع 2، وحتى النهاية:

```
b = "Hello, World!"
print(b[2:])
```

7-8-3- الفهرسة السلبية:

استخدم الفهارس السالبة لبدء الشريحة من نهاية السلسلة:

مثال: اطبع المحارف:

من: "o" في "World!" (الموضع -5)

إلى،: "d" في "World!" (الموضع -2):

```
b = "Hello, World!"
print(b[-5:-2])
```

7-9- التعديل على السلاسل:

لدى بايثون مجموعة من الأساليب المضمنة التي يمكنك استخدامها على السلاسل.

7-9-1- الأحرف الكبيرة:

مثال: تقوم الطريقة Upper() بإرجاع السلسلة المكتوبة بأحرف كبيرة:

```
a = "Hello, World!"
print(a.upper())
```

7-9-2- الأحرف الصغيرة:

مثال: تقوم الطريقة lower() بإرجاع السلسلة بأحرف صغيرة:

```
a = "Hello, World!"
print(a.lower())
```

7-9-3- إزالة المسافة البيضاء:

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

مثال: يقوم التابع strip() بإزالة أي مسافة بيضاء (فارغة)من البداية أو النهاية:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

ENG HASAN HAJJAR

7-9-4- استبدال السلسلة:

يستبدل تابع `replace()` سلسلة بسلسلة أخرى:

مثال:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

7-9-5- تقسيم السلسلة:

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

مثال: تقوم الطريقة `split()` بتقسيم السلسلة إلى سلاسل فرعية إذا وجدت مثيلات للفاصل:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

سنتعرف على المزيد حول القوائم في فصل قوائم بايثون.

7-10- دمج السلاسل:

لربط سلسلتين أو دمجهما، يمكنك استخدام عامل +.

مثال: دمج المتغير `a` مع المتغير `b` في المتغير `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

ولإضافة مسافة بينهما، أضف " ":

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

11-7- تنسيق السلاسل:

كما تعلمنا في فصل متغيرات بايثون، لا يمكننا الجمع بين السلاسل والأرقام.

مثال: سيعطي خطأ

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

ولكن يمكننا الجمع بين السلاسل والأرقام باستخدام تابع format()!

يأخذ التابع format() الوسائط التي تم تمريرها، وينسقها، ويضعها في السلسلة حيث تكون العناصر النائبة {} هي:

مثال: استخدم طريقة format() لإدراج الأرقام في السلاسل:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

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

مثال:

```
quantity = 3
itemno = 567
```

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

يمكنك استخدام أرقام الفهرس {0} للتأكد من وضع الوسائط في العناصر النائبة الصحيحة:

مثال:

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

12-7- إدراج أحرف غير قانونية في سلسلة:

لإدراج أحرف غير قانونية في سلسلة، استخدم حرف هروب.

حرف الهروب هو شرطة مائلة عكسية \ متبوعة بالحرف الذي تريد إدراجه.

مثال على الحرف غير القانوني هو علامة الاقتباس المزدوجة داخل سلسلة محاطة بعلامات اقتباس مزدوجة:

مثال: سوف تحصل على خطأ إذا استخدمت علامات الاقتباس المزدوجة داخل سلسلة محاطة بعلامات اقتباس مزدوجة:

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

لإصلاح هذه المشكلة، استخدم حرف الهروب \":

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

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

- أحرف الهروب الأخرى المستخدمة في بايثون:

- \' Single Quote
- \\ Backslash
- \n New Line
- \t Tab

13-7- طرق السلسلة:

لدى بايثون مجموعة من الطرق المضمنة التي يمكنك استخدامها على السلاسل.

ملاحظة: تقوم كافة طرق السلسلة بإرجاع قيم جديدة. لا يغيرون السلسلة الأصلية.

الوصف	الطريقة

تحويل الحرف الأول إلى أحرف كبيرة
مثال:
`txt = "hello, and welcome to my world."`
`x = txt.capitalize()`
`print (x)`
القاعدة العامة :
`string.capitalize()`
مثال 2:
	capitalize()

```
txt = "python is FUN!"
x = txt.capitalize()
print (x)
```

casefold() تحويل السلسلة إلى حالة صغيرة

مثال:

```
txt = "Hello, And Welcome To My World!"
x = txt.casefold()
print(x)
```

التعريف والاستخدام:

تقوم طريقة casefold() بإرجاع سلسلة حيث تكون جميع الأحرف صغيرة.

تشبه هذه الطريقة الطريقة Lower()، لكن طريقة casefold()أقوى ، مما يعني أنها ستحول المزيد من الأحرف إلى أحرف صغيرة، وستجد المزيد من التطابقات عند مقارنة سلسلتين ويتم تحويل كليهما باستخدام casefold()طريقة.

القاعدة العامة:

```
string.casefold()
```

center() إرجاع سلسلة مركزية

مثال: اطبع كلمة "banana" بمساحة 20 حرفًا، مع وضع كلمة "banana" في المنتصف:

```
txt = "banana"
x = txt.center(20)
print(x)
```

القاعدة العامة :

```
string.center(length, character)
```

length اجباري. طول السلسلة التي تم إرجاعها

Character اختياري. الحرف لملء المساحة المفقودة على كل جانب.
الافتراضي هو " " (مسافة)

مثال : استخدام الحرف "0" كحرف الحشو:

```
txt = "banana"
x = txt.center(20, "O")
print(x)
```

count() إرجاع عدد المرات التي تحدث فيها قيمة محددة في سلسلة

مثال: إرجاع عدد المرات التي تظهر فيها القيمة "apple" في السلسلة:

```
txt = "I love apples, apple are my favorite fruit"
x = txt.count("apple")
print(x)
```

القاعدة العامة :

```
string.count(value, start, end)
```

value اجباري. سلسلة. السلسلة المراد البحث عنها

start اختياريًا. عدد صحيح. الموقف لبدء البحث. الافتراضي هو 0

end اختيارية. عدد صحيح. الموقف من إنهاء البحث. الافتراضي هو نهاية السلسلة

مثال: البحث من الموضع 10 إلى 24:

```
txt = "I love apples, apple are my favorite fruit"
```

```
x = txt.count("apple", 10, 24)
print(x)
```

encode()	إرجاع نسخة مشفرة من السلسلة.
endswith()	يُرجع true إذا انتهت السلسلة بالقيمة المحددة **مثال:** سيرجع true

```
txt = "Hello, welcome to my world."
x = txt.endswith(".")
print(x)
```

القاعدة للتابع:

string.endswith(*value, start, end*)

value مطلوبة. القيمة المراد التحقق من انتهاء السلسلة بها

start اختيارية. عدد صحيح يحدد عند أي موضع لبدء البحث

end اختيارية. عدد صحيح يحدد الموضع الذي سيتم إنهاء البحث عنده

مثال:تحقق مما إذا كان الموضع من 5 إلى 11 ينتهي بعبارة "my world.".

```
txt = "Hello, welcome to my world."
x = txt.endswith("my world.", 5, 11)
print(x)
```

expandtabs()	يضبط حجم علامة التبويب للسلسلة

ENG HASAN HAJJAR

find() يبحث في السلسلة عن قيمة محددة ويعيد الموضع الذي تم العثور عليه فيه

مثال:

أين توجد في النص كلمة "welcome" ؟:

```
txt = "Hello, welcome to my world."

x = txt.find("welcome")

print(x)
```

التعريف والاستخدام:

تعثر طريقة find() على التواجد الأول للقيمة المحددة.

تقوم طريقة find() بإرجاع -1 إذا لم يتم العثور على القيمة.

طريقة find() هي تقريبًا نفس طريقة index() والفرق الوحيد هو أن طريقة index() تثير استثناءً إذا لم يتم العثور على القيمة.

القاعدة العامة :

```
string.find(value, start, end)
```

value إجباري. القيمة للبحث عنها

start اختياريًا. من أين تبدأ البحث. الافتراضي هو 0

end اختيارية. حيث لإنهاء البحث. الافتراضي هو نهاية السلسلة

مثال: أين يوجد في النص أول ظهور للحرف "e" ؟:

```
txt = "Hello, welcome to my world."
x = txt.find("e")
print(x)
```

مثال: أين يوجد في النص أول ظهور للحرف "e" عندما تبحث فقط بين الموضع 5 و 10؟:

```
txt = "Hello, welcome to my world."
x = txt.find("e", 5, 10)
print(x)
```

مثال:

إذا لم يتم العثور على القيمة، فإن طريقة find() ترجع -1، لكن طريقة index() ستثير استثناءً:

```
txt = "Hello, welcome to my world."

print(txt.find("q"))
print(txt.index("q"))
```

format()	تنسيق القيم المحددة في سلسلة
format_map()	تنسيق القيم المحددة في سلسلة
index()	يبحث في السلسلة عن قيمة محددة ويعيد الموضع الذي تم العثور عليه فيه

مثال:

أين توجد في النص كلمة "welcome"؟:

```
txt = "Hello, welcome to my world."
x = txt.index("welcome")
print(x)
```

التعريف والاستخدام:

تعثر طريقة `index()` على التواجد الأول للقيمة المحددة.

تثير طريقة `index()` استثناءً إذا لم يتم العثور على القيمة.

طريقة `index()` هي تقريبًا نفس طريقة `find()` والفرق الوحيد هو أن طريقة `find()` ترجع -1 إذا لم يتم العثور على القيمة.

القاعدة:

```
string.index(value, start, end)
```

value اجبارية. القيمة للبحث عنها

start اختياريًا. من أين تبدأ البحث. الافتراضي هو 0

end اختيارية. حيث لإنهاء البحث. الافتراضي هو نهاية السلسلة

مثال:

أين يوجد في النص أول ظهور للحرف "e" ؟:

```
txt = "Hello, welcome to my world."
x = txt.index("e")
print(x)
```

مثال:

أين يوجد في النص أول ظهور للحرف "e" عندما تبحث فقط بين الموضع 5 و 10؟:

```
txt = "Hello, welcome to my world."
x = txt.index("e", 5, 10)
print(x)
```

مثال:

إذا لم يتم العثور على القيمة، فإن طريقة find() ترجع -1، لكن طريقة index() ستثير استثناءً:

```
txt = "Hello, welcome to my world."
print(txt.find("q"))
print(txt.index("q"))
```

isalnum()	إرجاع True إذا كانت جميع الأحرف في السلسلة عبارة عن حروف أبجدية رقمية
isalpha()	إرجاع True إذا كانت جميع الأحرف في السلسلة بالأحرف الأبجدية
isascii()	إرجاع True إذا كانت جميع الأحرف في السلسلة عبارة عن أحرفascii
isdecimal()	إرجاع True إذا كانت جميع الأحرف في السلسلة عبارة عن أرقام عشرية
isdigit()	إرجاع True إذا كانت جميع الأحرف في السلسلة عبارة عن أرقام
isidentifier()	إرجاع True إذا كانت السلسلة عبارة عن معرف
islower()	إرجاع True إذا كانت كافة الأحرف في السلسلة صغيرة

مثال:

تحقق مما إذا كانت جميع الأحرف الموجودة في النص مكتوبة بأحرف صغيرة:

```
txt = "hello world!"

x = txt.islower()

print(x)
```

التعريف والاستخدام:

تقوم الدالة islower() بإرجاع True إذا كانت جميع الأحرف مكتوبة بأحرف صغيرة، وإلا فستُرجع False.

لا يتم التحقق من الأرقام والرموز والمسافات، فقط الحروف الأبجدية.

القاعدة العامة:

```
string.islower()
```

مثال:

تحقق مما إذا كانت جميع الأحرف الموجودة في النصوص مكتوبة بأحرف صغيرة:

```
a = "Hello world!"
b = "hello 123"
c = "mynameisPeter"

print(a.islower())
print(b.islower())
print(c.islower())
```

isnumeric()	إرجاع صحيح إذا كانت كافة الأحرف في السلسلة رقمية
isprintable()	إرجاع True إذا كانت كافة الأحرف الموجودة في السلسلة قابلة للطباعة

isspace()	إرجاع True إذا كانت جميع الأحرف في السلسلة عبارة عن مسافات بيضاء
istitle()	تُرجع القيمة True إذا كانت السلسلة تتبع قواعد العنوان
isupper()	تُرجع القيمة True إذا كانت جميع الأحرف في السلسلة مكتوبة بأحرف كبيرة
join()	يربط عناصر التكرار حتى نهاية السلسلة

مثال:

قم بربط جميع العناصر الموجودة في tuple في سلسلة، باستخدام حرف التجزئة كفاصل:

```
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print(x)
```

الخرج:

```
John#Peter#Vicky
```

التعريف والاستخدام :

تأخذ طريقة join() جميع العناصر الموجودة في كائن قابل للتكرار وتجمعها في سلسلة واحدة.

يجب تحديد سلسلة كفاصل.

القاعدة العامة:

`string.join(iterable)`
 	
`iterable` اجباري. أي كائن قابل للتكرار حيث تكون كافة القيم التي تم إرجاعها عبارة عن سلاسل
 	
مثال:
 	
قم بربط جميع العناصر الموجودة في القاموس في سلسلة، باستخدام الكلمة "TEST" كفاصل:
 	
`myDict = {"name": "John", "country": "Norway"}`
`mySeparator = "TEST"`
`x = mySeparator.join(myDict)`
`print(x)`
 	
الخرج:
 	
nameTESTcountry	

إرجاع نسخة مضبوطة على اليسار من السلسلة	ljust()
تحويل سلسلة إلى حالة صغيرة	lower()
إرجاع نسخة القطع اليسرى من السلسلة **مثال:** إزالة المسافات على يسار السلسلة:	lstrip()

```
txt = "     banana     "
x = txt.lstrip()
print("of all fruits", x, "is my favorite")
```

الخرج:

of all fruits banana   is my favorite

التعريف والاستخدام

تقوم الطريقة `lstrip()` بإزالة أي أحرف بادئة (المسافة هي الحرف البادئ الافتراضي المراد إزالته)

القاعدة العامة:

```
string.lstrip(characters)
```

`characters` اختيارية. مجموعة من الأحرف المراد إزالتها كأحرف بادئة

مثال:

إزالة الأحرف الرائدة:

```
txt = ",,,,,ssaaww.....banana"
x = txt.lstrip(",.asw")
print(x)
```

الخرج:

banana

partition() تقوم بإرجاع tuple حيث يتم تقسيم السلسلة إلى ثلاثة أجزاء

مثال:

ابحث عن كلمة "bananas " وأرجع tuple يحتوي على ثلاثة عناصر:

1- كل شيء قبل "المحددة"
2- " المحددة "
3- كل شيء بعد " المحددة "

```
txt = "I could eat bananas all day"
x = txt.partition("bananas")
print(x)
```

الخرج:

('I could eat ', 'bananas', ' all day')

التعريف والاستخدام :

يبحث تابع partition() عن سلسلة محددة، ويقسم السلسلة إلى tuple يحتوي على ثلاثة عناصر.

يحتوي العنصر الأول على الجزء الموجود قبل السلسلة المحددة.

يحتوي العنصر الثاني على السلسلة المحددة.

العنصر الثالث يحتوي على الجزء بعد السلسلة.

ملاحظة: تبحث هذه الطريقة عن التواجد الأول للسلسلة المحددة.

القاعدة العامة:

```
string.partition(value)
```

value مطلوبة. السلسلة المطلوب البحث عنها

مثال:

إذا لم يتم العثور على القيمة المحددة، فإن تابع partition() يُرجع tuple يحتوي على:

1 - السلسلة بأكملها، 2 - سلسلة فارغة، 3 - سلسلة فارغة:

```
txt = "I could eat bananas all day"
x = txt.partition("apples")
print(x)
```

الخرج:

```
(' ,' ,'I could eat bananas all day')
```

replace() إرجاع سلسلة حيث يتم استبدال قيمة محددة بقيمة محددة

مثال: استبدال كلمة "banana":

```
txt = "I like bananas"
x = txt.replace("bananas", "apples")
print(x)
```

التعريف والاستخدام:

تستبدل طريقة replace() عبارة محددة بعبارة أخرى محددة.

ملاحظة: سيتم استبدال كافة تكرارات العبارة المحددة، إذا لم يتم تحديد أي شيء آخر.

القاعدة العامة:

```
string.replace(oldvalue, newvalue, count)
```

oldvalue اجباري السلسلة المطلوب البحث عنها

newvalue اجباري السلسلة المراد استبدال القيمة القديمة بها

count اختياري. رقم يحدد عدد تكرارات القيمة القديمة التي تريد استبدالها.

مثال:

استبدال كل تكرار لكلمة "one" :

```
txt = "one one was a race horse, two two was
one too."
x = txt.replace("one", "three")
print(x)
```

مثال:

يستعاض عن أول ظهورين لكلمة "one" :

```
txt = "one one was a race horse, two two was
one too."
x = txt.replace("one", "three", 2)
print(x)
```

rfind() يبحث في السلسلة عن قيمة محددة ويعيد آخر موضع تم العثور عليه فيه

مثال: أين يوجد في النص آخر تواجد للسلسلة "casa" ؟:

```
txt = "Mi casa, su casa."
x = txt.rfind("casa")
print(x)
```

التعريف والاستخدام

تبحث طريقة rfind() عن التواجد الأخير للقيمة المحددة.

تقوم طريقة rfind() بإرجاع -1 إذا لم يتم العثور على القيمة.

طريقة rfind() هي تقريبًا نفس طريقة rindex().

القاعدة العامة:

```
string.rfind(value, start, end)
```

value مطلوبة. القيمة للبحث عن

start اختياريًا. من أين تبدأ البحث. الافتراضي هو 0

end اختيارية. حيث لإنهاء البحث. الافتراضي هو نهاية السلسلة

مثال: أين يوجد في النص آخر ظهور للحرف "e" ؟:

```
txt = "Hello, welcome to my world."
x = txt.rfind("e")
print(x)
```

مثال: أين يوجد في النص آخر ظهور للحرف "e" عندما تبحث فقط بين الموضع 5 و10؟:

```
txt = "Hello, welcome to my world."
x = txt.rfind("e", 5, 10)
print(x)
```

مثال: إذا لم يتم العثور على القيمة، فإن طريقة rfind() ترجع -1، لكن طريقة rindex() ستثير استثناءً:

```
txt = "Hello, welcome to my world."
print(txt.rfind("q"))
print(txt.rindex("q"))
```

يبحث في السلسلة عن قيمة محددة ويعيد آخر موضع تم العثور عليه فيه	rindex()

إرجاع نسخة صحيحة ومضبوطة من السلسلة	rjust()

تقوم بإرجاع tuple حيث يتم تقسيم السلسلة إلى ثلاثة أجزاء	rpartition()

مثال: ابحث عن آخر تواجد لكلمة "bananas"، وقم بإرجاع صف يحتوي على ثلاثة عناصر:

1- كل شيء قبل "المباراة"

2- "المباراة"

3- كل شيء بعد "المباراة"

```
txt = "I could eat bananas all day, bananas are my favorite fruit"
x = txt.rpartition("bananas")
print(x)
```

الخرج:

('I could eat bananas all day, ', 'bananas', ' are my favorite fruit')

التعريف والاستخدام:

يبحث الأسلوب `rpartition()` عن آخر تواجد لسلسلة محددة، ويقسم السلسلة إلى صف يحتوي على ثلاثة عناصر.

يحتوي العنصر الأول على الجزء الموجود قبل السلسلة المحددة.

يحتوي العنصر الثاني على السلسلة المحددة.

العنصر الثالث يحتوي على الجزء بعد السلسلة.

القاعدة العامة:

string.rpartition(*value*)

value *مطلوبة*. السلسلة المطلوب البحث عنها

مثال: إذا لم يتم العثور على القيمة المحددة، تقوم طريقة rpartition() بإرجاع صف يحتوي على: 1 - سلسلة فارغة، 2 - سلسلة فارغة، 3 - السلسلة بأكملها:

```
txt = "I could eat bananas all day, bananas
are my favorite fruit"
x = txt.rpartition("apples")
print(x)
```

rsplit()	يقسم السلسلة عند الفاصل المحدد، ويعيد قائمة list
rstrip()	إرجاع نسخة القطع الصحيحة من السلسلة

مثال: قم بإزالة أي مسافات بيضاء في نهاية السلسلة:

```
txt = "     banana     "
x = txt.rstrip()
print("of all fruits", x, "is my favorite")
```

الخرج:

of all fruits   banana is my favorite

القاعدة العامة:

string.rstrip(*characters*)

characters اختيارية. مجموعة من الأحرف المطلوب إزالتها كأحرف لاحقة

مثال: قم بإزالة الأحرف اللاحقة إذا كانت عبارة عن فاصلات أو نقاط أو s أو q أو w:

```
txt = "banana,,,,,ssqqqww....."
x = txt.rstrip(",.qsw")
print(x)
```

split()

يقسم السلسلة عند الفاصل المحدد، ويعيد قائمة

مثال: قم بتقسيم سلسلة إلى قائمة حيث تكون كل كلمة عبارة عن عنصر قائمة:

```
txt = "welcome to the jungle"
x = txt.split()
print(x)
```

الخرج:

```
['welcome', 'to', 'the', 'jungle']
```

التعريف والاستخدام:

تقوم طريقة Split() بتقسيم السلسلة إلى قائمة.

يمكنك تحديد الفاصل، الفاصل الافتراضي هو أي مسافة بيضاء.

ملاحظة: عند تحديد maxsplit، ستحتوي القائمة على العدد المحدد من العناصر بالإضافة إلى عنصر واحد.

القاعدة العامة:

```
string.split(separator, maxsplit)
```

separator اختياري. يحدد الفاصل الذي سيتم استخدامه عند تقسيم السلسلة. بشكل افتراضي، أي مسافة بيضاء هي فاصل

maxsplit اختياري. يحدد عدد الانقسامات التي يجب القيام بها. القيمة الافتراضية هي -1، وهي "كافة الأحداث"

مثال: قم بتقسيم السلسلة باستخدام الفاصلة، متبوعة بمسافة، كفاصل:

```
txt = "hello, my name is Peter, I am 26 years old"
x = txt.split(", ")
print(x)
```

الخرج:

['hello', 'my name is Peter', 'I am 26 years old']

مثال: استخدم حرف التجزئة كفاصل:

```
txt = "apple#banana#cherry#orange"
x = txt.split("#")
print(x)
```

الخرج:

['apple', 'banana', 'cherry', 'orange']

مثال: قم بتقسيم السلسلة إلى قائمة تحتوي على عنصرين كحد أقصى:

```
txt = "apple#banana#cherry#orange"
# setting the maxsplit parameter to 1, will return a list with 2 elements!
x = txt.split("#", 1)
print(x)
```

الخرج:

['apple', 'banana#cherry#orange']

splitlines() يقسم السلسلة عند فواصل الأسطر ويعيد القائمة

مثال: قم بتقسيم سلسلة إلى قائمة حيث يكون كل سطر عنصرًا في القائمة:

```
txt = "Thank you for the music\nWelcome to the jungle"

x = txt.splitlines()

print(x)
```

الخرج:

['Thank you for the music', 'Welcome to the jungle']

القاعدة العامة:

string.splitlines(*keeplinebreaks*)

keeplinebreaks اختياري. يحدد ما إذا كان يجب تضمين فواصل الأسطر (صحيح) أم لا (خطأ). القيمة الافتراضية خاطئة

مثال: قم بتقسيم السلسلة، ولكن احتفظ بفواصل الأسطر:

```
txt = "Thank you for the music\nWelcome to the jungle"
x = txt.splitlines(True)
print(x)
```

startswith() يُرجع True إذا كانت السلسلة تبدأ بالقيمة المحددة

مثال: تحقق مما إذا كانت السلسلة تبدأ بـ "Hello":

```
txt = "Hello, welcome to my world."

x = txt.startswith("Hello")
```

```
print(x)
```

التعريف والاستخدام:

تُرجع الدالة `startswith()` ال True إذا كانت السلسلة تبدأ بالقيمة المحددة، وإلا فستُرجع False.

القاعدة العامة:

```
string.startswith(value, start, end)
```

`value` مطلوبة. القيمة المطلوب التحقق مما إذا كانت السلسلة تبدأ بها

`start` اختياريًا. عدد صحيح يحدد عند أي موضع لبدء البحث

`end` اختيارية. عدد صحيح يحدد الموضع الذي سيتم إنهاء البحث عنده

مثال: تحقق مما إذا كان الموضع من 7 إلى 20 يبدأ بالحرف "wel":

```
txt = "Hello, welcome to my world."
x = txt.startswith("wel", 7, 20)
print(x)
```

strip() إرجاع نسخة مشذبة مقطعة من السلسلة

مثال: إزالة المسافات في بداية ونهاية السلسلة:

```
txt = "     banana     "
x = txt.strip()
print("of all fruits", x, "is my favorite")
```

التعريف والاستخدام:

يقوم التابع strip() بإزالة أي مسافات بيضاء بادئة وزائدة.

البادئة تعني في بداية السلسلة، واللاحقة تعني في النهاية.

يمكنك تحديد الحرف (الحروف) المراد إزالتها، وإذا لم يكن الأمر كذلك، فستتم إزالة أي مسافات بيضاء.

القاعدة العامة :

```
string.strip(characters)
```

characters اختيارية. مجموعة من الأحرف المراد إزالتها كأحرف بادئة/لاحقة

مثال: إزالة الأحرف البادئة والزائدة:

```
txt = ",,,,,rrttgg.....banana....rrr"
x = txt.strip(",.grt")
print(x)
```

الخرج:

```
banana
```

swapcase()

يتم تبديل الحالات، حيث تصبح الأحرف الصغيرة كبيرة والعكس صحيح

مثال: جعل الحروف الصغيرة كبيرة والحروف الكبيرة صغيرة:

```
txt = "Hello My Name Is PETER"
x = txt.swapcase()
print(x)
```

التعريف والاستخدام:

تقوم طريقة `swapcase()` بإرجاع سلسلة حيث تكون كافة الأحرف الكبيرة صغيرة والعكس صحيح.

القاعدة العامة:

```
string.swapcase()
```

title()	تحويل الحرف الأول من كل كلمة إلى أحرف كبيرة

مثال: اجعل الحرف الأول في كل كلمة كبيرًا:

```
txt = "Welcome to my world"
x = txt.title()
print(x)
```

التعريف والاستخدام

تقوم طريقة title() بإرجاع سلسلة حيث يكون الحرف الأول في كل كلمة بأحرف كبيرة. مثل رأس أو عنوان.

إذا كانت الكلمة تحتوي على رقم أو رمز، فسيتم تحويل الحرف الأول بعد ذلك إلى أحرف كبيرة.

القاعدة العامة:

```
string.title()
```

translate()	إرجاع سلسلة مترجمة

مثال: استبدل أي أحرف "S" بحرف "P":

```
#use a dictionary with ascii codes to replace
83 (S) with 80 (P):
mydict = {83:  80}
txt = "Hello Sam!"
print(txt.translate(mydict))
```

الخرج:

Hello Pam!

التعريف والاستخدام

تقوم طريقة () Translator بإرجاع سلسلة حيث يتم استبدال بعض الأحرف المحددة بالحرف الموصوف في القاموس، أو في جدول التعيين.

استخدم طريقة () maketrans لإنشاء جدول التعيين.

إذا لم يتم تحديد حرف في القاموس/الجدول، فلن يتم استبدال الحرف.

إذا كنت تستخدم قاموسًا، فيجب عليك استخدام رموز ascii بدلاً من الأحرف.

```
string.translate(table)
```

`table` مطلوب. إما قاموس، أو جدول تعيين يصف كيفية إجراء الاستبدال

مثال: استخدم جدول التعيين لاستبدال "S" بـ "P":

```
txt = "Hello Sam!"
mytable = str.maketrans("S", "P")
print(txt.translate(mytable))
```

مثال: استخدم جدول التعيين لاستبدال العديد من الأحرف:

```
txt = "Hi Sam!"
x = "mSa"
y = "eJo"
mytable = str.maketrans(x, y)
print(txt.translate(mytable))
```

مثال: تصف المعلمة الثالثة في جدول التعيين الأحرف التي تريد إزالتها من السلسلة:

```
txt = "Good night Sam!"
x = "mSa"
y = "eJo"
z = "odnght"
mytable = str.maketrans(x, y, z)
print(txt.translate(mytable))
```

مثال: نفس المثال المذكور أعلاه، ولكن باستخدام القاموس بدلاً من جدول التعيين:

```
txt = "Good night Sam!"
mydict = {109: 101, 83: 74, 97: 111, 111:
None, 100: None, 110: None, 103: None, 104:
None, 116: None}
print(txt.translate(mydict))
```

تحويل سلسلة إلى حالة كبيرة	upper()
يملأ السلسلة بعدد محدد من القيم 0 في البداية	zfill()

ENG HASAN HAJJAR

انتهى الفصل