الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة الشهيد الشيخ العربي التبسي – تبسة
كلية العلوم الاقتصادية، العلوم
التجارية وعلوم التسيير
قسـم: العلوم الاقتصادية

مطبوعة في مقياس:

اساسيات البرمجة بالبايثون 1

Python Programming Basic 1

(دروس و نماذج مقترحة)

موجهة لطلبة السنة الاولى ليسانس اقتصاد دولي عن بعد
وطلبة السنة أولى جذع مشترك
(قسم العلوم الاقتصادية)
وجميع الطلبة في جميع التخصصات المعنيين بتعلم البرمجة بالبايثون

إعداد الدكتور:

حميدان ربيع

2026/2025

بسم الله الرحمن الرحيم



الفهرس

مقدمة	02
المحور الاول: من التفكير إلى الكود رحلة الخوارزميات	03
المحور الثاني: مدخل الى البرمجة بلغة Python	09
المحور الثالث: كتابة أول برنامج بسيط بلغةPython	14
المحور الرابع: المتغيرات و العمليات الحسابية عليها	17
المحور الخامس:التحكم في تدفق البرنامج (جمل الشرط)	23
المحور السادس:الحلقات التكرارية (loops)	28
المحور السابع: الدوال في بايثون(Functions)	33
المحور الثامن: التعامل مع النصوص في بايثون (String)	37
المحور التاسع: القوائم (Lists) والصفوف (Tuples)	41
المحور العاشر:القواميس (Dictionaries) والمجموعات (Sets)	46
المحور الحادي عشر: البرمجة كائنية التوجه (OOP)	53
المحور الثاني عشر: الاستعانة بأدوات الذكاء الاصطناعي لتوليد الأكواد	57
المحور الثالث عشر: التعامل مع الملفات في بايثون	60
المحور الرابع عشر: مشاريع مقترحة للاحتراف	65
خاتمة	79
قائمة المراجع	81

مقدمة:

تأتي هذه المطبوعة البيداغوجية لتسد فجوة معرفية متنامية في عصر الرقمنة والتحول التكنولوجي، حيث لم تعد البرمجة حكراً على المختصين في علوم الحاسوب، بل أصبحت أداةً أساسية تُعزز القدرات التحليلية وتُنمّي التفكير المنطقي لدى دارسي مختلف التخصصات.

وإن كان المخاطَب الأول بهذه المطبوعة هو طالب السنة الأولى في تخصص الاقتصاد الدولي بنظام التعليم عن بُعد، فإنها في جوهرها تُخاطب كلَّ من يرغب في ولوج عالم البرمجة بلغة بايثون، بغض النظر عن مستواه الأكاديمي أو تخصصه. فلغة بايثون، بفضل بساطتها وسهولة تعلمها وقوة مكتباتها، تُعدُّ اللغة المثالية للانطلاق في رحلة البرمجة.

تراعي هذه المطبوعة خصوصية طالب الاقتصاد الدولي الذي يحتاج إلى أدوات برمجية تساعده في معالجة البيانات الاقتصادية، وأتمتة المهام المتكررة، كما تناسب أي متعلم يبحث عن تأسيس متين في مفاهيم البرمجة الأساسية.

وقد صُممت المطبوعة لتُقرأ في بيئة التعلم عن بُعد، حيث يعتمد الطالب على نفسه في استيعاب المفاهيم، وذلك من خلال تبسيط النظريات، وتقديم أمثلة تطبيقية متدرجة، وترقيم الأكواد البرمجية بشكل واضح، مع إتاحة فرصة التجربة والخطأ.

تغطي المطبوعة المحاور التالية بشكل متسلسل ومنطقي: بدءاً بمدخل تمهيدي إلى البرمجة بلغة بايثون، ثم كتابة أول برنامج بسيط، فالانتقال إلى التعامل مع المتغيرات والعمليات الحسابية، ثم النصوص، فالقوائم والصفوف، فالقواميس والمجموعات، مروراً بهندسة الأوامر وتطبيقاتها، وانتهاءً بالتعامل مع الملفات. هذا التسلسل يُمكن المتعلم من بناء معرفته تراكميا و بشكل قوي ، حيث يعتمد كل محور على ما سبقه.

راجين من الله التوفيق، أن تكون هذه المطبوعة خير عون لكل طالب يسعى إلى امتلاك ناصية البرمجة، وأن تفتح له آفاقاً واسعة في عالم التكنولوجيا والاقتصاد الرقمي.

المحور الأول: من التفكير إلى الكود رحلة الخوارزميات

قبل الشروع في كتابة السطر البرمجي الأول بلغة "بايثون"، من الضروري إدراك حقيقة جوهرية؛ وهي أن البرمجة ليست مجرد إتقان لقواعد اللغة، بل هي في أصلها فن صياغة الحلول المنطقية للمشكلات . ولكي نتمكن من تحويل أفكارنا إلى تعليمات يفهمها الحاسوب، لا بد من تبني منهجية منظمة للتفكير، وهو ما يجسده مفهوم الخوارزميات.(Algorithms)

1. أصل كلمة خوارزمية:

إن كلمة خوارزمية مشتقة من إسم العالم العربي الجليل محمد بن موسى الخوارزمي الذي عاش في بغداد من سنة 780 الى 847 م في عصر الخليفة المأمون , وقد برع هذا العالم في الرياضيات والفلك, وترك بصمات في التراث الحضاري العالمي، فقد وضع الخوارزمي مبادئ علم الجبر وألف كتاب" الجبر و المقابلة "وأعطى الجبر اسمه حتى اصبحت كلمة الجبر موجودة في جميع اللغات تقريباً.

وفي تلك الأونة انطلق اسم الخوارزميات Algorithms على جداول الضرب والقسمه والحساب العشري, وظل هذا الاسم متداولا في أوروبا مدة قرون حتى تطور مؤخرا ليحمل مدلولا جديداً مرتبطاً بالبرمجة (وسام، 2019).

2. اهمية الخوارزمية:

إن أهم مرحلة في حل مسألة ما بإستخدام الحاسوب هي المرحلة المتعلقة بإيجاد خطة الحل ، يجب أن تكون هذه الخطة قابلة للتنفيذ من قبل الآلة ، وقابلة للتوصيف على وجه لا يدعو الى اللبس أو التأويل، يطلق اسم الخوارزمية على هذه الخطة (وسام، 2019).

3. تعريف الخوارزمية:

مجموعة الخطوات المتسلسلة والمحددة التي تؤدي إلى حل مسألة معينة والوصول إلى نتائج محددة اعتبارًا من معطيات ابتدائية (وسام، 2019).

4. أنواع الخوارزميات:

1.4. خوارزميات حسابية :تهتم بالمسائل الرياضية (حل معادلة من الدرجة الأولى)

2.4. خوارزميات غير حسابية :لا تهتم بالمسائل الرياضية ولكنها تحتاج إلى حل منطقي (طريقة التدقيق الإملائي لنص ما، اتخاذ قرار بالذهاب إلى مكان ما وتحديد الطريق الأمثل للوصول إليه)

سنهتم في هذا المقرر بالخوارزميات الحسابية فقط.

5. طرق التعبير عن الخوارزمية:

- ✓ الطريقة الكلامية : كتابة الخوارزمييات على شكل خطوات باستخدام اللغة المتداولة كاللغة العربية أو الانجليزية.
- ✓ الطريقة الرمزية: كتابة الخوارزميات باستخدام الرموز.

- ✓ الطريقة التدفقية : كتابة الخوارزميات باستخدام المخططات البيانية(المخططات التدفقية) (نسرين، 2017).

مثال توضحي:

أكتب الخوارزمية التي تعطي نتيجة حل التعبير الرياضي الآتي باستخدام اللغة المتداولة (الطريقة الكلامية):

$$Y = (x^2 + 7)/(x + 2)$$

علما بأن x معلومة

الحل:

يمكن التعبير عن الخوارزمية باللغة المتداولة(العربية)على الشكل الآتي:

الخطوة الأولى: أقرأ(أدخل)قيمة المتحول x . الخطوة الثانية :احسب المقام a= (x+2)

الخطوة الثالثة :إذا كان المقام مساويا للصفر اطبع " المسألة ليس لها حل " الخطوة الرابعة :وإلا احسب البسط $b=(x^2+7)$

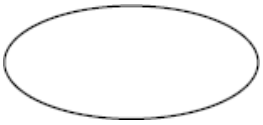
الخطوة الخامسة:احسب قيمة y : y=b/a الخطوة السادسة :اطبع (أكتب) قيمة y الخطوة السابعة :توقف

المخطط التدفقي : يعتبر المخطط التدفقي تمثيل مصور لتحليل المشكلة (نسرين، 2017) يعتمد على الرموز االصطالحية التالية:

a) **لتحديد بداية ونهاية الخوارزمية نستخدم الشكل**



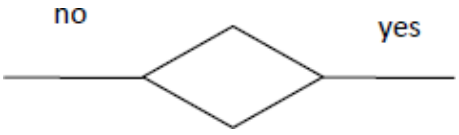
b) **عمليات الإدخال والإخراج نستخدم الشكل:**

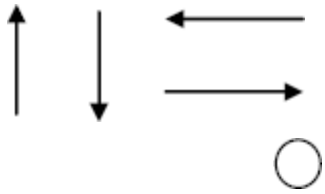


c) **عملية المعالجة نستخدم الشكل:**



d) **عملية الشرط(القرار)نستخدم الشكل:**





e) لمعرفة اتجاه الخوارزمية نستخدم:

f) نقطة توصيل وربط

تمرين 1: اكتب الخوارزمية الكلامية والرمزية والمخطط التدفقي لإيجاد مساحة ومحيط المستطيل ؟

الحل:

الخوارزمية الرمزية:

المدخلات : x , y

المعالجة: s=y*x

m=(y+x) *2

المخرجات: m , s

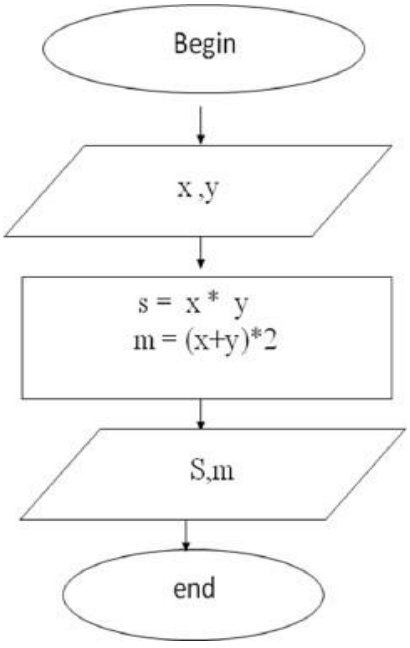
الخوارزمية الكلامية :

المدخلات : الطول والعرض

المعالجة : المساحة(S) =الطول x العرض المحيط(m)= (الطول +العرض)x 2

المخرجات : المساحة والمحيط

المخطط التدفقي:



تمرين 2 : على نمط المثال السابق اكتب الخوارزمية الكلامية و الرمزية والمخطط التدفقي لإيجاد مساحة ومحيط الدائرة؟

الحل:

الخوارزمية الكلامية :

1- المدخلات: نصف القطر (r)

2- المعالجة : المساحة(s) = π x نصف القطر للتربيع المحيط(m) =2xنصف القطرx p

3- المخرجات: المساحة والمحيط لدائرة

الخوارزمية الرمزية:

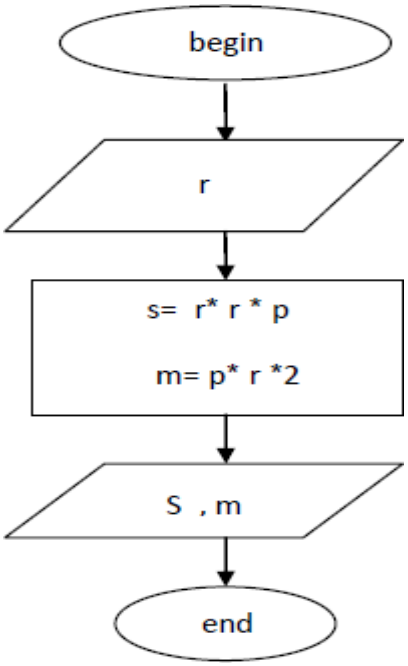
المدخلات: r

المعالجة : s=p*r*r

m=p*r*2

المخرجات :s,m

المخطط التدفقي:



تمرين3: اكتب الخوارزمية الرمزية والمخطط التدفقي لحل المعادلة : $a X + b = 0$

مناقشا جميع الحالات الممكنة ل a,b؟

الحل:

الخوارزمية الرمزية:

- أدخل(اقرأ) : a , b
- إذا كان ($a=0$, $b\neq 0$) نجد : $0x+b=0$ أي $b=0$

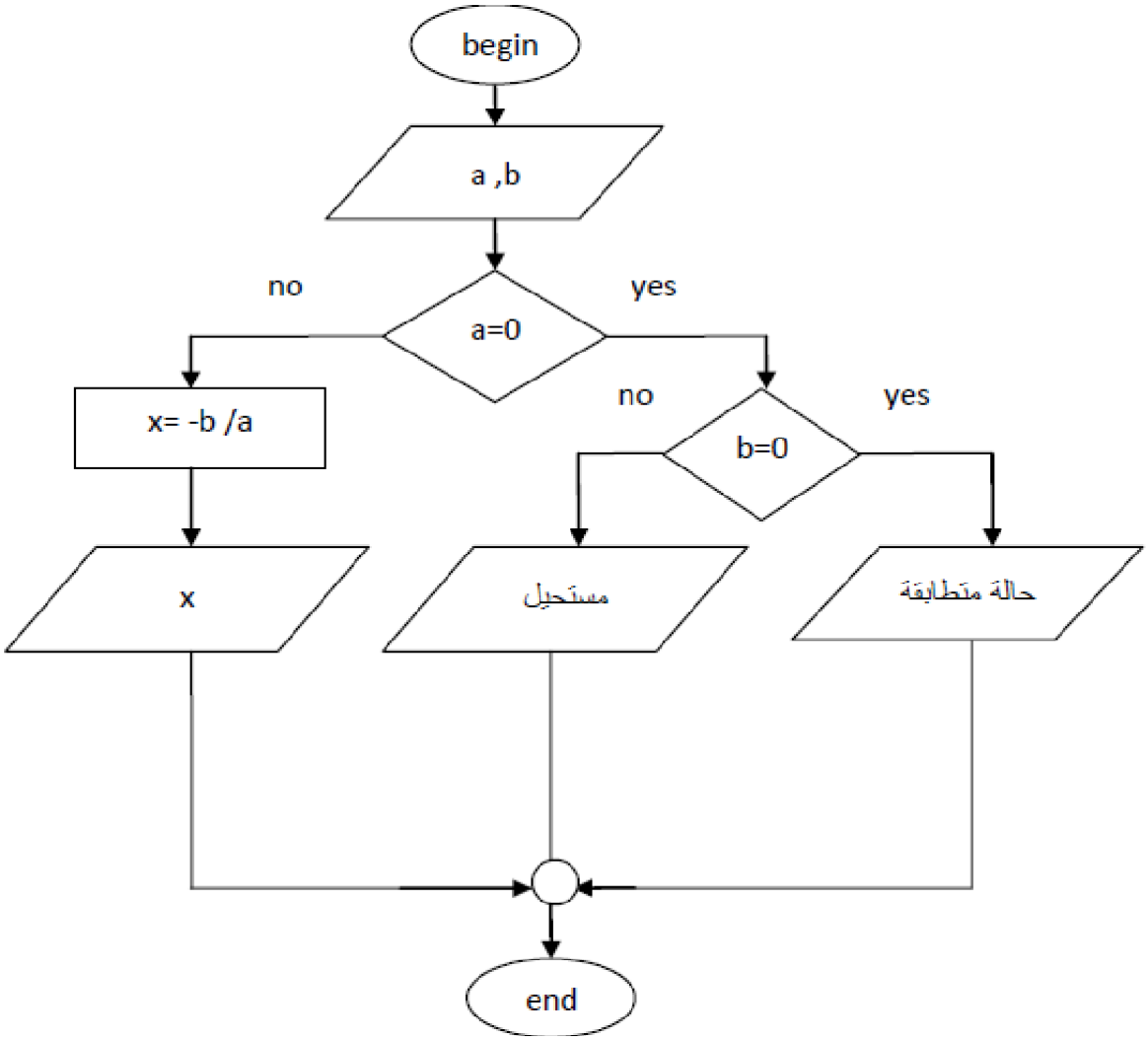
  أطبع (أكتب) : "مستحيل الحل" .
- إذا كان ($b=0$ نجد : $0x+0=0$
  $a=0$),

  أطبع (أكتب) : "حالة متطابقة" .

  إذا كان ($a\neq 0$) : $x=-b/a$

  أطبع قيمة x

المخطط التدفقي:



بعد معرفة طريقة حل أي المشكلة و كتابة الخوارزمية المناسبة لها يمكننا بعد ذلك كتابتها بأي لغة برمجة مثل ...java, c++, dart لان الحاسوب لا يفهم الخوارزمية في شكلها التدفقي او الرمزي بل يجب ترجمتها لاحد لغات البرمجة المعروفة ولعل أهمها في زمننا هذا لغة البايثون (Python) التي ستكون مجال الدراسة و التعلم في هذه المطبوعة البيداغوجية.

المحور الثاني: مدخل الى البرمجة بلغة Python

تعتبر البرمجة بشكل عام أداةً أساسية في العصر الرقمي، حيث تساهم في أتمتة المهام وتحليل البيانات وحل المشكلات المعقدة بكفاءة. وتتميز لغة بايثون ببساطتها وقوتها، مما جعلها من أكثر اللغات انتشاراً في مجالات متعددة كالذكاء الاصطناعي وتحليل البيانات وتطوير الويب. كما تلعب البرمجة دوراً حيوياً في العلوم الاقتصادية، إذ تساعد في بناء النماذج التنبؤية، وتحليل الأسواق المالية، واتخاذ القرارات المستندة إلى البيانات. ولتنفيذ هذه المهام، تتوفر بيئات عمل متنوعة مثلPyCharm ، وVS Code، و Jupyter Notebook التي تسهل كتابة وتصحيح الأكواد. في هذا المحور، سنستعرض أساسيات البرمجة بشكل عام.

1. علم البرمجة

علم البرمجة هو الطريقة التي نستخدمها لإعطاء أوامر وتعليمات دقيقة للحاسوب حتى يقوم بأداء مهام معينة (نسرين، 2017).

- هذه الأوامر تُكتب باستخدام لغات البرمجة مثل Java, Python, C, C++
- الكمبيوتر في الأصل لا يفهم إلا لغة الآلة (0 و 1).
- لذلك نكتب الكود بلغة برمجة قريبة من البشر، ثم تُترجم إلى لغة الآلة ليتم تنفيذها.

2. الهدف من علم البرمجة

- حل المشكلات :تحويل مشكلة حقيقية إلى خطوات منطقية قابلة للتنفيذ بالحاسوب.
- الأتمتة :جعل الحاسوب يقوم بمهام متكررة بدل الإنسان.
- الإبداع والابتكار :تصميم تطبيقات، مواقع، ألعاب، أنظمة ذكية.

3. مكونات علم البرمجة

1. المشكلة :(Problem) شيء نريد حله (مثلاً: حساب معدل طالب).
2. الخوارزمية :(Algorithm) سلسلة خطوات مرتبة لحل المشكلة.
    - مثال:
        a) إدخال علامتين.
        b) جمعهما.
        c) قسمة المجموع على 2.

d) عرض النتيجة.

3. اللغة :(Programming Language) الوسيلة التي نكتب بها الحل.
4. المترجم/المفسر :(Compiler/Interpreter) يحول الكود من لغة برمجة إلى لغة الآلة.

4. مجالات علم البرمجة

- تطوير الويب (مواقع وتطبيقات).
- تطوير التطبيقات (هواتف وحواسيب).
- الألعاب.
- الذكاء الاصطناعي.
- تحليل البيانات.
- الأمن السيبراني .
- الأتمتة والتحكم.

5. لماذا نتعلم البرمجة؟

- لأنها لغة العصر الرقمي.
- تساعد على التفكير المنطقي وحل المشكلات.
- تفتح مجالات عمل واسعة.
- تعطيك قوة الإبداع وصناعة أشياء مفيدة بنفسك.

6. أهمية لغة بايثون في الاقتصاد

a) تحليل البيانات الاقتصادية (Economic data analysis)

- الاقتصاد يعتمد بشكل كبير على الأرقام والإحصائيات.
- بايثون توفر مكتبات قوية مثل:
  - Pandas → لتنظيم ومعالجة الجداول.
  - NumPy → للحسابات الرياضية والإحصائية.
  - Matplotlib / Seaborn → لعرض الرسوم البيانية.

تساعد الاقتصاديين على فهم اتجاهات السوق، التضخم، البطالة، النمو الاقتصادي...

b) النمذجة الاقتصادية (Economic Modeling)

- بايثون يمكن بناء نماذج رياضية للتنبؤ بسلوك الاقتصاد.

- مثلاً:
  - توقع أسعار النفط.
  - محاكاة العرض والطلب.
  - حساب الناتج المحلي الإجمالي في سيناريوهات مختلفة.

c) التحليل المالي (Financial Analysis)

- تُستخدم في تحليل الأسهم، العملات، والسندات.
- حساب مؤشرات مثل: المتوسطات المتحركة، المخاطر، العوائد.
- بناء أنظمة Trading Algorithms للتداول الآلي في الأسواق المالية.

d) الاقتصاد الكلي والجزئي (Macroeconomics and Microeconomics)

- في الاقتصاد الكلي: تحليل السياسات الحكومية وأثرها على البطالة والتضخم.
- في الاقتصاد الجزئي: دراسة سلوك المستهلك والشركات عبر البيانات.

e) الذكاء الاصطناعي والتعلم الآلي (AI & Machine Learning)

- بايثون هي اللغة الأولى للذكاء الاصطناعي.
- الاقتصاديون يستعملونها لتوقع:
  - اتجاهات السوق.
  - قرارات المستهلك.
  - احتمالية إفلاس شركة أو بنك.

f) سهولة التعلم والتطبيق

- بايثون سهلة مقارنة بلغات أخرى مثل R أو C++.
- هذا يسمح لطلبة الاقتصاد (حتى لو ما عندهم خلفية قوية في الحاسوب) باستعمالها لفهم وتحليل مشاكل اقتصادية حقيقية.

الخلاصة : بايثون أصبحت أداة أساسية في الاقتصاد الحديث لأنها تربط بين البيانات الضخمة (Big Data) والتحليل الرياضي والتنبؤات الذكية.

7. بيئة التطوير المدمجة (IDE)

a) التعريف

تُعد بيئة التطوير المتكاملة (Integrated Development Environment) بمثابة "المختبر الشامل" للمبرمج؛ وهي تطبيق برمجي صُمم خصيصاً لدمج مجموعة واسعة من الأدوات والوظائف الأساسية في واجهة مستخدم واحدة. تهدف هذه البيئة إلى تحسين كفاءة المبرمج عبر توفير كل ما يحتاجه لإنشاء، واختبار، وتصحيح البرمجيات في مكان واحد.فبدلاً من تشتت المبرمج بين استخدام محرر نصوص بسيط (مثل Notepad) لتعديل الكود، ثم الانتقال إلى أدوات خارجية أخرى لتشغيله أو فحصه، يقوم الـ IDE بدمج هذه المهام معاً (Nelli, 2015).

b) مكونات الـ IDE

عادةً أي IDE يحتوي على:

- محرر نصوص → (Code Editor) المكان اللي نكتب فيه الكود.
- مترجم/مفسر → (Compiler/Interpreter) يحول الكود للغة الآلة وينفذه.
- مصحح أخطاء → (Debugger) يساعدك تكتشفي الأخطاء وتصلحيها.
- إكمال تلقائي → (Auto-completion) يقترحلك الكود عند الكتابة.
- إدارة الملفات والمشاريع →لتنظيم الملفات الكبيرة.

c) أشهر بيئات العمل لبايثون

- IDLE: بيئة بسيطة مدمجة مع بايثون (مناسبة للمبتدئين).
- PyCharm: بيئة قوية جداً ومستخدمة في المشاريع الكبيرة.
- Jupyter Notebook: مميزة لتحليل البيانات والبحث العلمي (تعرض كود + نتائج + شرح في نفس الصفحة).
- VS Code: محرر خفيف ومرن يدعم بايثون مع إضافات.

d) اهمية بيئة العمل المدمجة (IDE)

- يسرّع البرمجة.
- يقلل من الأخطاء.
- يساعد على فهم الكود وتنظيم المشاريع.
- مناسب للتعلم وللاحتراف.

في ختام هذا المحور، نكون قد تعرفنا على مفهوم البرمجة وأهميتها المتزايدة في عصر التحول الرقمي، حيث لم تعد حكراً على المختصين بل أصبحت أداةً ضرورية في مختلف المجالات. كما استعرضنا المجالات الواسعة التي تغطيها لغة بايثون، بدءاً من تحليل البيانات وحتى الذكاء الاصطناعي، مع تسليط الضوء على قيمتها الخاصة في العلوم الاقتصادية، مثل نمذجة الأسواق والتنبؤ بالمؤشرات المالية. وتطرقنا أيضاً إلى بيئات العمل المتعددة التي تسهل كتابة وتنفيذ أكواد بايثون، مما يساعد على اختيار الأنسب حسب الحاجة. وبعد هذا الإطار النظري، ننتقل في المحور التالي إلى الجانب التطبيقي العملي، حيث سنعرض أول برنامج بلغة بايثون لنبدأ رحلة البرمجة خطوة بخطوة.

المحور الثالث :كتابة أول برنامج بسيط بلغةPython

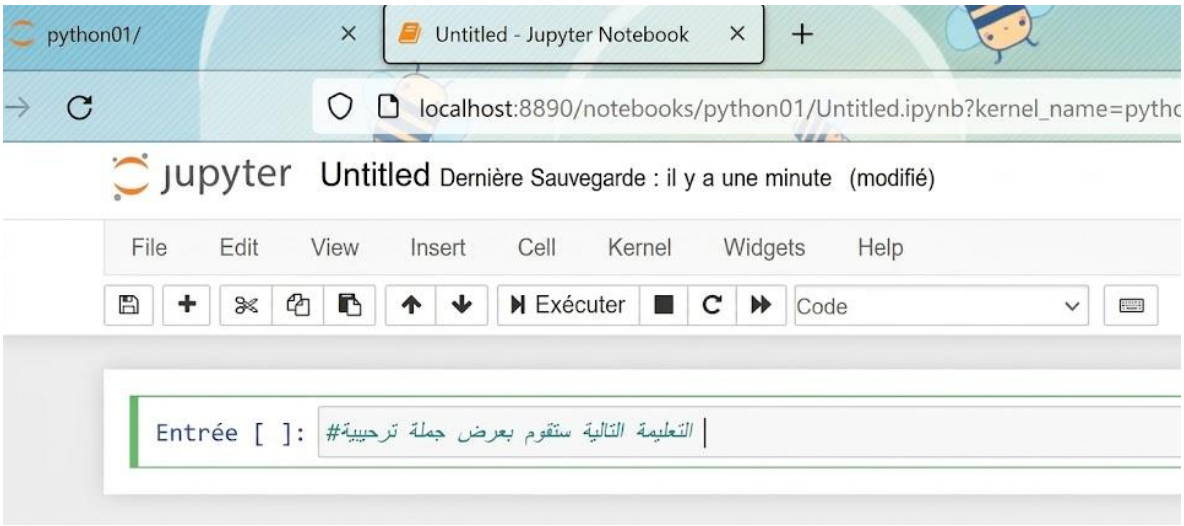
سنتناول في هذا المحور منهجية التطوير المتكاملة باستخدام لغة بايثون، حيث سنمضي معاً في رحلة تطبيقية لبناء برنامج متكامل خطوة بخطوة، مع استعراض كافة الركائز الأساسية التي تضمن إتقان هذه اللغة والتمكن من أدواتها.

1. التعليقات في بايثون (comments in python)

تعتبر التعليقات مهمة جدا في أي لغة برمجة حيث انها تستعمل لشرح عمل تعليمة معينة او جزء من البرنامج مما يسهل قراءته فيما بعد قصد اجراء أي تعديلات او تحسينات مستقبلية على البرنامج .ففي لغة بايثون نقوم بوضع العلامة # في بداية السطر الذي يمثل التعليق .حيث لا يقوم البرنامج بتنفيذ هذا السطر فقط هو كتابة مساعدة للمبرمج لفهم او توضيح مهمة كل جزء من البرنامج .كما يمكن كتابة التعليق في أي مكان في البرنامج.

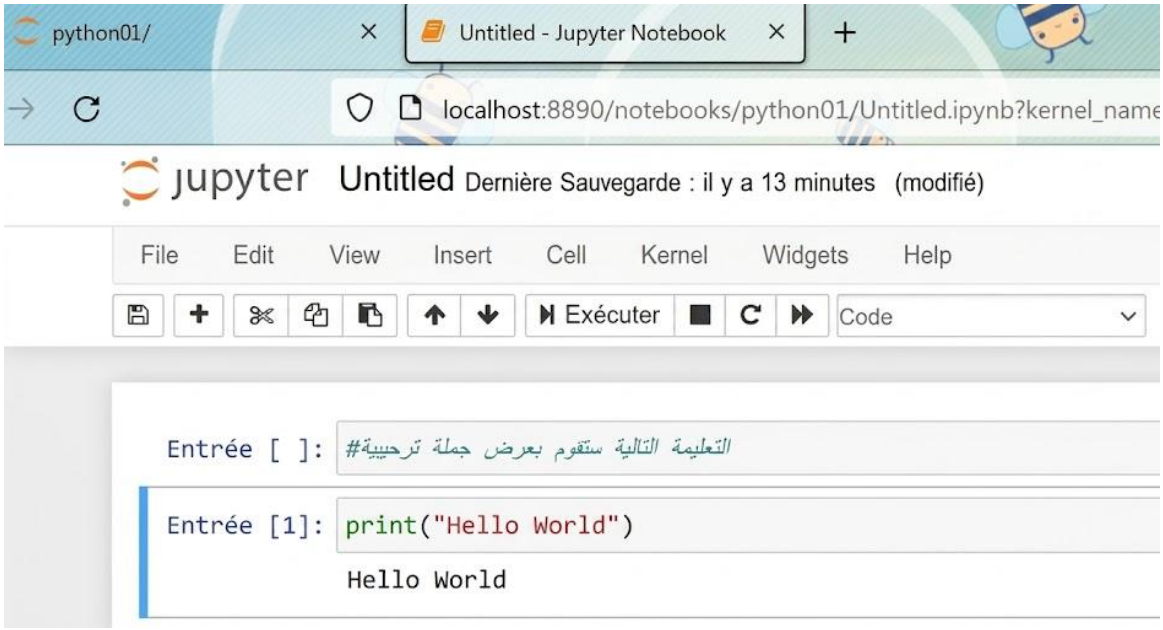
مثال	ترحيبية جملة بعرض ستقوم التالية التعليمة #

في برنامج jupyter notbook تكون التعليقات كما يلي



2. طباعة النصوص (Output)

لطباعة النصوص و عرض النتائج على الشاشة نستعمل التعليمةprint مثال عند طباعة نص Hello Worldنكتب print("Hello World") ثم تنفيذ البرنامج عن طريق زر Exécuter او بالضغط على الزرين من لوحة المفاتيح. Ctrl + Entrée



3. المتغييرات (Variables)

المتغير مكان نخزن فيه قيمة معينة مثل عدد ,نص.....

فمثلا

نوع المتغير	اسمه بالفرنسية	اسمه بالانجليزية	رمزه	مثال
نص	Caractère	String	Str	name="mohamed"
عدد صحيح	Entier	Integer	Int	a=5
عدد عشري	Réel	Real	Float	b=3.14
عدد ناطق	Booléen	Boolean	bool	c=True

حيث يعرف المتغير باسم يتكون هذا الاسم من حروف و ارقام فلا يمكن بداية كتابة اسم المتغير بعدد او رمز خاص ما عدا المطة التي تكون مع السطر مثل. x_ كما يمكن لاسم المتغير ان يحمل رمز المطة بين الحروف .كذلك يكون اسم المتغير ذو دلالة فمثلا الاجر الشهري (Month Salary) يمكن كتابة المتغير بهذا الشكل Month_Salary حتى يكون اسم المتغير ذو دلالة.

كما يمكن إعطاء قيمة مباشرة للمتغير في البرنامج ثم طابعتها من خلال الامر print او الطلب من المستخدم ادخال قيمة للمتغير عن طريق الامرinput فتتم الكتابة عن طريق كتاية اسم المتغير متبوعا بعلامة = ثم الامر input و كتابة جملة تساعد المستخدم في ادخال قيمة المتغير بين قوسين و بين مزوجتية

مثل: name=input("Entrée votre nom: ")

مثال 1 انشاء المتغييرات و طباعتها:

```
Entrée [3]: # اعطاء قيم لكل من المتغيرات name, a, b et c
            name=("Mohamed") #name variable str
            a=5              #a variable int
            b=3.14           #b variable float
            c=True           #c variable Boo
```

```
Entrée [4]: # طباعة كل متغير
            print(name)
            print(a)
            print(b)
            print(c)
```

```
Mohamed
5
3.14
True
```

مثال 2 ادخال قيمة المتغير عن طريق الامرinput

```
Entrée [*]: # ادخال قيمة المتغير name
            name=input("Entrér votre nom: ")
```

```
Entrér votre nom:
```

```
Entrée [*]: # ادخال قيمة المتغير name
            name=input("Entrér votre nom: ")
```

```
Entrér votre nom: Rabie
```

```
Entrée [5]: # إدخال قيمة المتغير name
            name=input("Entrér votre nom: ")
```

```
Entrér votre nom: Rabie
```

```
Entrée [6]: print(name)
```

```
Rabie
```

المحور الرابع:المتغيرات و العمليات الحسابية عليها

بعد ان قمنا بالتعرض في الدروس السابقة الى انواع المتغيرات والتي تكون اما اعداد صحيحة او عشرية او ناطقة (integer, float, boolean) وقد تكون نصوص .(string) في هذا الدرس سوف نقوم بتوضيح اشهر العمليات بالنسبة للمتغيرات العددية.

```
# أنواع البيانات الأساسية
name = "أحمد"#      "أحمد (String)
age = 25            # عدد صحيح (Integer)
height = 175.5      # عدد عشري (Float)
is_student = True # قيمة منطقية (Boolean)
```

امثلة عن المتغييرات:

```
# تعريف متغيرات متعددة
x = 10
y = 20
z = 30

# تعريف متغيرات في سطر واحد
a, b, c = 5, 10, 15

# إعادة تعيين قيمة متغير
counter = 0
counter = counter + 1  # 1 تصبح القيمة
```

1. العمليات الحسابية (Arithmetic Operations) العمليات الأساسية:

في هذه الفقرة سنقوم بعرض مختلف العمليات الحسابية الأساسية مثل الجمع ,الطرح ,الضرب ,القسمة و القسمة الصحيحة و كذلك باقي القسمة و الاس.

```
# الجمع
result1 = 10 + 5      # 15

# الطرح
result2 = 20 - 8      # 12

# الضرب
result3 = 6 * 7       # 42

# القسمة العادية
result4 = 15 / 4      # 3.75

# القسمة الصحيحة
result5 = 15 // 4     # 3

# باقي القسمة
result6 = 15 % 4      # 3

# الأُسّ
result7 = 2 ** 3      # 8
```

امثلة عملية:

```
# حساب متوسط ثلاثة أعداد
num1 = 85
num2 = 90
num3 = 78
average = (num1 + num2 + num3) / 3
print(f"المتوسط: {average}")  # 84.333 :المتوسط

# حساب مساحة مستطيل
length = 12
width = 8
area = length * width
print(f"مساحة المستطيل: {area}")  # 96 :مساحة المستطيل

# تحويل درجة الحرارة
celsius = 25
fahrenheit = (celsius * 9/5) + 32
print(f"25 مئوية = {fahrenheit} فهرنهايت")  # 25 مئوية = 77.0 فهرنهايت
```

العمليات المختصرة:

```
x = 10

x += 5    # تعادل x = x + 5
x -= 3    # تعادل x = x - 3
x *= 2    # تعادل x = x * 2
x /= 4    # تعادل x = x / 4

print(x)  # 6.0
```

2. أولوية العمليات (Operational Priority):

الأولوية في العمليات الرياضية تكون للاقواس ثم الاس و بعده الضرب و القسمة على حد سواء ثم يأتي الجمع و الطرح كذلك لهما نفس الأولوية

مثال:

```
# الأولوية: الأس ← الضرب والقسمة ← الجمع والطرح
result1 = 10 + 5 * 2    # 20 (وليس 30)
result2 = (10 + 5) * 2  # 30
result3 = 2 ** 3 * 2    # 16
```

3. مكتبة الرياضيات (math)

هي مكتبة من اشهر المكتبات في بايثون حيث تحوي العديد من الدوال الجاهزة و القابلة للاستعمال المباشر مثل دالة الجذر التربيعي و دالة القيمة المطلقة و الدوال المثلثية و غيرها وفيما يلي سنقوم بعرض اشهر هذه الدوال (Nelli, 2015).

الدوال الرياضية الأساسية (Basic mathematical functions)

✓ الجذر التربيعي sqrt()

```
import math
x = 16
result = math.sqrt(x)
print(result) # 4.0
y = 2
result2 = math.sqrt(y)
print(result2) # 1.41
```

✓ دالة الاس pow()

```
base = 2
exponent = 3
result = math.pow(base, exponent)
print(f"{base} أس  {exponent} = {result}") # 8.0
# ** العادي المشغل مع مقارنة
print(2 ** 3) # 8
print(math.pow(2, 3)) # 8.0
```

✓ دالة القيمة المطلقة fabs()

```
num = -5.7
result = math.fabs(num)
print(f" القيمة المطلقة ل {num} هي  {result}") # 5.7
```

✓ دوال التقريب للاعلى و الاسفل

- **التقريب للاعلى ceil()**

```
x = 3.2
result = math.ceil(x)
print(f" تقريب  {x} للأعلى  = {result}") # 4
y = 5.8
print(f" تقريب  {y} للأعلى  = {math.ceil(y)}") # 6
```

- **التقريب للاسفل floor()**

```
x = 3.9
result = math.floor(x)
print(f"تقريب {x} للأسفل = {result}") # 3
y = 7.1
print(f"تقريب {y} للأسفل = {math.floor(y)}") # 7
```

✓ **الثوابت الرياضية مثل pi - π**

```
pi = math.pi
print(f"قيمة π = {pi}") # 3.141592653589793
print(f"قيمة π مقربة = {pi:.2f}") # 3.14
# حساب مساحة دائرة
radius = 5
area = math.pi * math.pow(radius, 2)
print(f"مساحة دائرة نصف قطرها {radius} = {area:.2f}") # 78.54
```

✓ **دوال التحميع و التوليد**

- **دالة العاملية factorial()**

```
n = 5
result = math.factorial(n)
print(f"{n}! = {result}") # 120
# مثال: عدد الطرق لترتيب 3 كتب
books = 3
arrangements = math.factorial(books)
print(f"عدد طرق ترتيب {books} كتب = {arrangements}") # 6
```

- **دالة القاسم المشترك الأكبر gcd()**

```
a = 48
b = 18
result = math.gcd(a, b)
print(f"القاسم المشترك الأكبر ل {a} و {b} هو {result}") # 6
```

مثال اقتصادي مبسط

حساب الربح و الخسارة في مشروع صغير

```
import math
# حساب صافي الربح لمشروع بيع العصائر
print("=== حساب أرباح مشروع العصائر ===")
# المدخلات الأساسية
```

```
الكوب_سعر = 5 # دج
للكوب_التكلفة = 2 # دج
اليومي_الأكواب_عدد = 100
# الأساسية الحسابات
اليومي_الأكواب_عدد * الكوب_سعر = اليومي_الإيراد
اليومي_الأكواب_عدد * للكوب_التكلفة = اليومية_التكلفة
اليومية_التكلفة - اليومي_الإيراد = اليومي_الربح
print(f"اليومي الإيراد : {اليومي_الإيراد} دج")
print(f"اليومية التكلفة : {اليومية_التكلفة} دج")
6
print(f"اليومي الربح : {اليومي_الربح} دج")
# شهرية حسابات
شهريا_العمل_أيام = 26
شهريا_العمل_أيام * اليومي_الربح = الشهري_الربح
print(f"\nالشهري الربح : {الشهري_الربح} دج")
# الربح نسبة حساب
الربح_نسبة = ((الكوب_سعر - للكوب_التكلفة) / للكوب_التكلفة) * 100
print(f"كوب لكل الربح نسبة : {الربح_نسبة :.1f}%")
# للتقريب math دوال استخدام
print(f"لأعلى مقرب الشهري الربح : {math.ceil(الشهري_الربح)} دج")
print(f"لأسفل مقرب الشهري الربح : {math.floor(الشهري_الربح)} دج")
# الشهرية التكاليف لتغطية اللازمة الأكواب عدد حساب
الشهرية_التكاليف = 3000 # ورواتب إيجار
للتغطية_الأكواب_عدد = الشهرية_التكاليف / (الكوب_سعر - للكوب_التكلفة)
print(f"الشهرية التكاليف لتغطية اللازمة الأكواب عدد: {math.ceil(للتغطية_الأكواب_عدد)} كوب")
```

المحور الخامس :التحكم في تدفق البرنامج (جمل الشرط)

1. التحكم في تدفق البرنامج

يُقصد بـ التحكم في التدفق القدرة على تحديد المسار الذي يسلكه البرنامج أثناء التنفيذ. ففي الحالة الافتراضية، تقوم لغة "بايثون" بتنفيذ الأسطر البرمجية بشكل متسلسل)من الأعلى إلى الأسفل). ومع ذلك، تتطلب البرامج الذكية نوعاً من "المنطق" لتغيير هذا الترتيب بناءً على معطيات معينة. فبدلاً من التنفيذ الأعمى لكل الأسطر، تمنحنا أدوات التحكم القدرة على جعل البرنامج يتخذ قرارات لحظية. وهنا يأتي دور جمل الشرط (VanderPlas, 2016).

- التنفيذ الشرطي :تنفيذ كتلة برمجية معينة فقط في حال تحقق شرط منطقي محدد.
- المسارات البديلة :توجيه البرنامج لتنفيذ أمر مغاير تماماً إذا لم يتحقق ذلك الشرط.

2. جمل الشرط (if statements)

- **الشكل العام:**

```
if condition:
    # code to execute if the condition is True
```

مثال:

```
age = 18

if age >= 18:
    print("You are an adult.")
```

النتيجة:

```
You are an adult.
```

لان الشرط age>=18 صحيح

3. جملة if...else

تُستخدم لما نريد تنفيذ أمر إذا كان الشرط صحيح و أمر آخر إذا كان خطأ.

```
if condition:
    # code if condition is True
else:
    # code if condition is False
```

مثال:

```
age = 16

if age >= 18:
    print("You are an adult.")
else:
    print("You are underage.")
```

النتيجة:

```
You are underage.
```

4. جملة if...elif...else

تُستخدم لما نريد اختبار أكثر من شرط.

```
if condition1:
    # code
elif condition2:
    # code
else:
    # code
```

مثال:

```
grade = 85

if grade >= 90:
    print("Excellent")
elif grade >= 75:
    print("Good")
else:
    print("Needs improvement")
```

النتيجة:

```
Good
```

5. جملة شرطية داخل أخرى(Nested if)

يمكنك وضع if داخل if.

مثال:

```
age = 20
student = True

if age >= 18:
    if student:
        print("Adult student")
    else:
        print("Adult non-student")
else:
    print("Minor")
```

التبيجة:

```
Adult student
```

ملاحظات مهمة:

a) المسافات (Indentation) ضرورية جدًا في بايثون، لأنها تحدد أين يبدأ وينتهي كل جزء من الكود.
عادة 4 مسافات أو(Tab)

b) الشروط يمكن أن تحتوي على:

- مقارنات

العملية	الرمز
تساوي	==
اكبر او تساوي	>=
اكبر تماما	>
اصغر او تساوي	<=
اصغر تماما	<
لا تساوي	!=

- العمليات المنطقية:

تستعمل لربط العديد من الشروط

العملية	الرمز
الربط ب "و"	and
الربط ب "أو"	or
النفي	not

مثال:

```python
age = 19
has_id = True

if age >= 18 and has_id:
    print("You can enter")
```

التمرين: تحديد نوع الرقم

اكتب برنامج بلغة Python يقوم بما يلي:

1. يطلب من المستخدم إدخال رقم.
2. يتحقق:
   - إذا كان الرقم **موجب** ← يطبع: "الرقم موجب"
   - إذا كان الرقم **سالب** ← يطبع: "الرقم سالب"
   - وإذا كان الرقم **صفر** ← يطبع: "الرقم هو صفر"

الحل:

```python
# طلب إدخال رقم من المستخدم
number = float(input("أدخل رقمًا: "))

# استخدام الشروط لتحديد نوع الرقم
if number > 0:
    print("الرقم موجب")
elif number < 0:
    print("الرقم سالب")
else:
    print("الرقم هو صفر")
```

المحور السادس :الحلقات التكرارية (loops)

1. الحلقات التكرارية

تُعد الحلقات التكرارية (loops) واحدة من أقوى البنى المنطقية في البرمجة، وهي تقنية تُستخدم لتنفيذ كتلة من الأوامر البرمجية بشكل متكرر وتلقائي طالما أن هناك شرطاً معيناً يتحقق. الهدف الأساسي منها هو أتمتة العمليات المتشابهة وتقليل الجهد البشري في كتابة الكود (Hilpisch, 2018).

مثلاً:
بدل ما نكتب:

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

نستطيع ان نستخدم حلقة واحدة لتكرار هذا السطر أربع مرات

2. أنواع الحلقات في بايثون:

في Python عندنا نوعين أساسيين من الحلقات:

a) **حلقة for**

b) **حلقة while**

أولاً: حلقة for

تُستخدم للتكرار عدد معين من المرات (عند معرفة عدد التكرارات مسبقًا).

الصيغة العامة:

```
for variable in sequence:
    # code to repeat
```

الـ **sequence** ممكن تكون:

- قائمة (list)

- سلسلة نصية (string)
- مدى أرقام (range)

مثال 1:

```
for i in range(5):
    print("Hello Wiss!")
```

النتائج:

```
Hello Wiss!
Hello Wiss!
Hello Wiss!
Hello Wiss!
Hello Wiss!
```

لأن range (5) يعني الأرقام: 0, 1, 2, 3, 4 (أي خمس مرات)
مثال 2:

```
for i in range(1, 6):
    print("Number:", i)
```

النتائج:

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

ثانياً: حلقة while

تُستخدم للتكرار طالما أن شرطًا معينًا صحيح (True)
وتتوقف عندما يصبح الشرط خاطئًا(False).

الصيغة العامة:

```
while condition:
    # code to repeat
```

مثال:

```
count = 1

while count <= 5:
    print("This is number", count)
    count += 1  # زيادة العداد
```

النتائج:

```
This is number 1
This is number 2
This is number 3
This is number 4
This is number 5
```

ملاحظة مهمة:

تحدث الحلقة اللانهائية (Infinite Loops) عندما يفشل البرنامج في الوصول إلى "شرط التوقف (Exit Condition)"، مما يؤدي إلى استمرار تنفيذ كتلة الأوامر داخل الحلقة بشكل مستمر ودون انقطاع. من الناحية المنطقية، يعني هذا أن الشرط الذي يحكم الحلقة يظل دائماً "صحيحاً (True)"، ولا توجد آلية داخلية لتغيير هذه الحالة الا التوقيف القصري للجهاز (Hilpisch, 2018).

مثال خاطئ:

```
count = 1
while count <= 5:
    print(count)
# نسينا count += 1 ❌
```

3. التحكم داخل الحلقات

a) لإنهاء الحلقة قبل اكتمالها (break).

مثال:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

النتائج:

```
0
1
2
3
4
```

b) لتخطي تكرار معين وإكمال الباقي (continue).

مثال:

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

النتائج:

```
0
1
3
4
```

تمرين:

حساب المتوسط

نريد برنامجًا بلغة **Python** يقوم بما يلي

1. يطلب من المستخدم إدخال **عدد العلامات.(n)**
2. يطلب من المستخدم إدخال **العلامات واحدة تلو الأخرى.**
3. يحسب **المجموع** و**المتوسط** في النهاية.
4. يطبع النتيجة للمستخدم.

الحل:

```
# طلب عدد العلامات من المستخدم
n = int(input("كم عدد العلامات التي تريد حساب متوسطها؟ "))

# تهيئة المتغيرات
total = 0  # مجموع العلامات

# حلقة لتكرار الإدخال
for i in range(n):
    mark = float(input(f"أدخل العلامة رقم {i+1}: "))
    total += mark  # جمع العلامات

# حساب المتوسط
average = total / n

# طباعة النتيجة
print("مجموع العلامات هو:", total)
print("المتوسط هو:", average)
```

المحور السابع: الدوال في بايثون (Functions)

1. تعريف الدالة

تُعرف الدالة بأنها وحدة برمجية مستقلة، تضم مجموعة من الأوامر والتعليمات المترابطة التي صُممت معاً لأداء مهمة محددة بدقة. يمكن تشبيه الدالة بـ "آلة" داخل المصنع الكبير؛ لها مدخلات محددة، تقوم بمعالجة معينة، ثم تعطي مخرجات أو نتائج واضحة. الميزة الجوهرية للدوال هي مبدأ إعادة الاستخدام (Reusability). فبدلاً من كتابة الخطوات نفسها عدة مرات في أجزاء مختلفة من البرنامج، يتم تغليف هذه الخطوات داخل "دالة" باسم فريد، ومن ثم استدعاؤها كلما دعت الحاجة. فالدالة اذا هي وسيلة المبرمج الذكي لتحويل الكود من نصوص مكررة ومشتتة إلى هيكل منظم، فعال، وسهل التطوير. (Downey, 2014).

2. طريقة إنشاء دالة في Python

الصيغة العامة:

```
def function_name(parameters):
    # code to execute
```

def كلمة مفتاحية لإنشاء دالة.

function_name اسم الدالة (من اختيارك).

parameters (اختياري) قيم تدخل إلى الدالة.

return تُستخدم لإرجاع نتيجة (اختياري).

مثال 1: دالة بسيطة بدون معاملات (Parameters)

```
def greet():
    print("Hello Wiss! 🌸")

# استدعاء الدالة
greet()
```

النتائج:

```
Hello Wiss! 🌸
```

مثال 2 : دالة تأخذ معاملات (Parameters)

```
def greet(name):
    print("Hello", name + "!")

greet("Wiss")
```

النتائج:

```
Hello Wiss!
```

مثال 3: دالة تُرجع قيمة(return)

```
def add(a, b):
    return a + b

result = add(5, 3)
print("المجموع هو:", result)
```

النتائج:

```
المجموع هو: 8
```

3. دالة مع شرط (if)

يمكن وضع الشروط داخل الدوال.

مثال:

```
def check_age(age):
    if age >= 18:
        return "مسموح بالدخول ✅"
    else:
        return "ممنوع الدخول ❌"

print(check_age(16))
```

النتائج:

```
❌ ممنوع الدخول
```

4. دالة تستعمل داخل حلقة (loop + function)

مثال:

```python
def square(x):
    return x * x

for i in range(1, 6):
    print("مربع", i, "هو:", square(i))
```

النتائج:

```
مربع 1 هو: 1
مربع 2 هو: 4
مربع 3 هو: 9
مربع 4 هو: 16
مربع 5 هو: 25
```

5. أهمية الدوال

لماذا نستخدم الدوال؟

a) تجعل الكود منظمًا وسهل القراءة.

b) تقلل من تكرار الكود.

c) تسهّل إصلاح الأخطاء وتعديل البرنامج.

d) تساعد على إعادة استخدام الكود في مشاريع أخرى.

تمرين:

برنامج لحساب متوسط ثلاث علامات باستخدام دالة

```
def average(a, b, c):
    avg = (a + b + c) / 3
    return avg

# إدخال القيم من المستخدم
x = float(input("أدخل العلامة الأولى: "))
y = float(input("أدخل العلامة الثانية: "))
z = float(input("أدخل العلامة الثالثة: "))

result = average(x, y, z)
print("المتوسط هو:", result)
```

النتائج إذا كانت العلامات(15، 14، 16):

```
المتوسط هو: 15.0
```

المحور الثامن: التعامل مع النصوص في بايثون (String)

1. تعريف النصوص (Strings)

النص (String) هو نوع من أنواع البيانات الأساسية في بايثون، ويُستخدم لتمثيل البيانات النصية. تقنياً، يُعتبر النص عبارة عن سلسلة مرتبة من الرموز (Characters)التي قد تشمل حروفاً، أرقاماً، رموزاً خاصة، أو حتى مسافات فارغة (McKinney, 2017).

```
# Single quotes
name = 'John'
# Double quotes
message = "Hello World"
# Triple quotes for multi-line
text = """This is a
multi-line string"""
```

2. العمليات الأساسية على النصوص

- الدمج (Concatenation)

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print(full_name)  # Output: John Doe
```

- التكرار (Repetition)

```
word = "Hello"
repeated = word * 3
print(repeated)  # Output: HelloHelloHello
```

3. الفهرسة والتقطيع (Indexing and Slicing)

- الفهرسة (Indexing)

```
text = "Python"
print(text[0])   # Output: P
print(text[-1])  # Output: n (last character)
```

- التقطيع (Slicing)

```
text = "Programming"
print(text[0:4])    # Output: Prog
print(text[3:])     # Output: gramming
print(text[:6])     # Output: Progra
print(text[::2])    # Output: Pormig (every second character)
```

4. الدوال الشائعة للنصوص

- طول النص (Length)

```
text = "Hello"
print(len(text))  # Output: 5
```

- التحويل الى حروف كبيرة / صغيرة (Converting to uppercase/lowercase letters)

```
text = "Python Programming"
print(text.upper())      # Output: PYTHON PROGRAMMING
print(text.lower())      # Output: python programming
print(text.title())      # Output: Python Programming
```

- البحث و الاستبدال

```
text = "I love Python programming"
print(text.find("Python"))     # Output: 7
print(text.replace("Python", "Java"))  # Output: I love Java programming
```

- التقسيم (Spliting)

```
sentence = "Hello,World,Python"
words = sentence.split(",")
print(words)  # Output: ['Hello', 'World', 'Python']
```

- الدمج (Joining)

```
words = ['Python', 'is', 'awesome']
sentence = " ".join(words)
print(sentence)  # Output: Python is awesome
```

5. فحص النصوص

```
text = "Python123"
print(text.isalpha())      # False (not all alphabetic)
print(text.isalnum())      # True (alphanumeric)
print(text.isdigit())      # False
print(text.startswith("P")) # True
print(text.endswith("3"))   # True
```

6. التنسيق (String formatting)

- الموصى بها (f-string)

```
name = "Alice"
age = 25
print(f"My name is {name} and I'm {age} years old")
# Output: My name is Alice and I'm 25 years old
```

- format() method

```
text = "My name is {} and I'm {} years old".format("Bob", 30)
print(text)  # Output: My name is Bob and I'm 30 years old
```

7. إزالة المسافات (Removing spaces)

```
text = "   Hello World   "
print(text.strip())     # Output: "Hello World"
print(text.lstrip())    # Output: "Hello World   "
print(text.rstrip())    # Output: "   Hello World"
```

8. أمثلة عملية

```python
# مثال 1: معالجة بيانات المستخدم
username = "  user123  "
clean_username = username.strip().lower()
print(clean_username)  # Output: user123

# مثال 2: تحليل عنوان URL
url = "https://www.example.com/page"
if url.startswith("https"):
    print("Secure connection")

# مثال 3: عد الكلمات
sentence = "Python is a powerful programming language"
word_count = len(sentence.split())
print(f"Number of words: {word_count}")  # Output: Number of words: 6

# مثال 4: عكس النص
text = "Python"
reversed_text = text[::-1]
print(reversed_text)  # Output: nohtyP
```

المحور التاسع: القوائم (Lists) والصفوف (Tuples)

1. القوائم (Lists)

a) تعريف القوائم

تُعد القائمة (List) واحدة من أكثر هياكل البيانات مرونة وشيوعاً في لغة بايثون. وهي عبارة عن حاوية منظمة تُستخدم لتخزين مجموعة من العناصر من أي نوع (رقمي, نصي او ناطق) في تسلسل محدد، ويمكن تغييرها بعد الإنشاء ، حيث يتم الوصول إلى كل عنصر عبر رقم تعريفي يُعرف بـالمؤشر (Index) (Müller & Guido, 2016).

b) انشاء القوائم (Creating lists)

```
# إنشاء قائمة
fruits = ["apple", "banana", "cherry"]
numbers = [1, 2, 3, 4, 5]
mixed_list = [1, "hello", 3.14, True]
```

c) الوصول الى العناصر في القائمة (Indexing)

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])   # Output: apple
print(fruits[-1])  # Output: cherry (آخر عنصر)
```

d) التقطيع (Slicing)

```
numbers = [0, 1, 2, 3, 4, 5]
print(numbers[1:4])   # Output: [1, 2, 3]
print(numbers[:3])    # Output: [0, 1, 2]
print(numbers[3:])    # Output: [3, 4, 5]
print(numbers[::2])   # Output: [0, 2, 4] (2 خطوة)
```

e) اظافة العناصر (Adding elements)

```
fruits = ["apple", "banana"]
fruits.append("cherry")        # إضافة عنصر واحد في النهاية
print(fruits)                  # Output: ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")     # إدراج عنصر في موضع محدد
print(fruits)                  # Output: ['apple', 'orange', 'banana', 'cherry']

fruits.extend(["grape", "mango"])  # إضافة عدة عناصر
print(fruits)                  # Output: ['apple', 'orange', 'banana', 'cherry', 'grape', 'mang
o']
```

f) حذف العناصر (Delete elements)

```
fruits = ["apple", "banana", "cherry", "banana"]
fruits.remove("banana")        # يحذف أول occurrence للقيمة
print(fruits)                  # Output: ['apple', 'cherry', 'banana']

popped = fruits.pop()          # يحذف آخر عنصر ويرجع قيمته
print(popped)                  # Output: banana
print(fruits)                  # Output: ['apple', 'cherry']

del fruits[0]                  # يحذف العنصر في الفهرس 0
print(fruits)                  # Output: ['cherry']

fruits.clear()                 # يفرغ القائمة
print(fruits)                  # Output: []
```

g) دوال أخرى

```
numbers = [1, 2, 3, 2, 4, 2]
print(len(numbers))            # Output: 6 (الطول)
print(numbers.count(2))        # Output: 3 (عدد مرات تكرار 2)
print(numbers.index(3))        # Output: 2 (فهرس أول ظهور للقيمة 3)

numbers.sort()                 # ترتيب تصاعدي
print(numbers)                 # Output: [1, 2, 2, 2, 3, 4]

numbers.reverse()              # عكس القائمة
print(numbers)                 # Output: [4, 3, 2, 2, 2, 1]

new_list = numbers.copy()      # نسخ القائمة
```

2. الصفوف (Tuples)

a) تعريف الصفوف

تُعد الصفوف (Tuples) أحد هياكل البيانات الأساسية في بايثون، وهي عبارة عن متتاليات منظمة تُستخدم لتخزين مجموعة من العناصر. بالرغم من تشابهها الظاهري مع "القوائم" في القدرة على حفظ أنواع مختلفة من البيانات، إلا أنها تختلف عنها في فلسفة التصميم والخصائص الوظيفية. تتميز الصفوف بأنها كائنات "ثابتة (Immutable) "؛ وهذا يعني أنه بمجرد إنشاء الصف وتحديد عناصره، يُغلق الكائن تقنياً ولا يمكن إجراء أي تعديلات عليه (سواء بالحذف، أو الإضافة، أو تغيير قيم العناصر) (Müller & Guido, 2016).

b) انشاء صف (Creating tuples)

```
# إنشاء صف
fruits = ("apple", "banana", "cherry")
numbers = (1, 2, 3, 4, 5)
single_element = (1,)          # ملاحظة: يجب وضع فاصلة للعنصر الواحد
```

c) الوصول الى العناصر في الصف (Indexing)

```
fruits = ("apple", "banana", "cherry")
print(fruits[0])   # Output: apple
print(fruits[-1])  # Output: cherry
```

d) التقطيع (Slicing)

```
numbers = (0, 1, 2, 3, 4, 5)
print(numbers[1:4])   # Output: (1, 2, 3)
print(numbers[:3])    # Output: (0, 1, 2)
```

e) دوال مفيدة للصفوف

```
fruits = ("apple", "banana", "cherry", "banana")
print(len(fruits))              # Output: 4
print(fruits.count("banana"))   # Output: 2
print(fruits.index("cherry"))   # Output: 2
```

3. الفرق الاساسية بين القوائم و الصفوف

الميزة	القوائم(Lists)	الصفوف(Tuples)
التعديل	متغيرة(mutable)	غير متغيرة(immutable)
السرعة	أبطأ	أسرع في التكرار
الاستخدام	تخزين بيانات قابلة للتغيير	تخزين بيانات ثابتة
المساحة	تشغل مساحة أكبر	تشغل مساحة أقل
الأمان	أقل أماناً (قابلة للتعديل)	أكثر أماناً (ثابتة)

تمرين:

إدارة قائمة مهام

```
# برنامج إدارة المهام

tasks = []

while True:
    print("\n1. Add task")
    print("2. View tasks")
    print("3. Remove task")
    print("4. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        task = input("Enter task: ")
        tasks.append(task)
        print("Task added successfully!")

    elif choice == "2":
        if tasks:
            print("\nYour tasks:")
            for i, task in enumerate(tasks, 1):
                print(f"{i}. {task}")
        else:
            print("No tasks found!")

    elif choice == "3":
        if tasks:
            print("\nYour tasks:")
            for i, task in enumerate(tasks, 1):
                print(f"{i}. {task}")
            task_num = int(input("Enter task number to remove: "))
            if 1 <= task_num <= len(tasks):
                removed = tasks.pop(task_num - 1)
                print(f"Task '{removed}' removed!")
            else:
```

```
            print("Invalid task number!")
        else:
            print("No tasks to remove!")

    elif: choice == "4"
        print("Goodbye!")
        break

    else:
        print("Invalid choice!")
```

المحور العاشر:القواميس (Dictionaries) والمجموعات (Sets)

1. القاموس (Dictionaries)

a) تعريف القاموس

يُعد القاموس في لغة بايثون هيكلاً بيانياً متطوراً يعتمد على مفهوم "الارتباط .(Mapping) "على عكس القوائم التي تعتمد على الترتيب الرقمي، يعتمد القاموس على نظام الثنائيات(Key-Value) ، حيث يتم ربط كل "مفتاح" بقيمة معينة، مما يجعله وسيلة فائقة السرعة للبحث عن البيانات واسترجاعها.

تُعرف القواميس باستخدام الأقواس المتعرجة { } وتخضع للقواعد التنظيمية التالية:

- ✓ **المفاتيح** :(Keys) تعمل كمعرفات فريدة للوصول إلى البيانات. ويشترط في المفتاح أن يكون غير قابل للتعديل (Immutable) مثل النصوص أو الأرقام، ولا يمكن تكراره داخل القاموس الواحد لضمان دقة الوصول.
- ✓ **القيم** :(Values) هي المحتوى المرتبط بالمفتاح. وتتميز بمرونة مطلقة؛ حيث يمكن أن تكون من أي نوع (نصوص، أرقام، قوائم، أو حتى قواميس أخرى)، كما يُسمح بتكرار القيم ذاتها لمفاتيح مختلفة.
- ✓ **القابلية للتعديل** :(Mutability) القاموس كائن مرن، حيث يمكن للمبرمج إضافة مفاتيح جديدة، تحديث قيم حالية، أو حذف مدخلات بعد إنشاء القاموس.

كما تُستخدم القواميس كأداة مثالية لتمثيل البيانات ذات الصلة الوثيقة، مثل سجلات الموظفين (حيث يكون "رقم الموظف" هو المفتاح و"بياناته" هي القيمة) أو إعدادات البرامج. وبفضل هيكليتها، توفر القواميس أداءً عالياً جداً في عمليات البحث مقارنة بهياكل البيانات الخطية.

الخلاصة :القاموس هو "الفهرس الذكي" في بايثون؛ فهو ينظم البيانات بطريقة منطقية تجعل من عملية الوصول إلى المعلومة أمراً فورياً ودقيقاً بغض النظر عن حجم البيانات (Müller & Guido, 2016).

b) انشاء قاموس (Creating a dictionary)

```
# Empty dictionary
my_dict = {}
# Or
my_dict = dict()

# Dictionary with initial values
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

print(person)  # Output: {'name': 'John', 'age': 30, 'city': 'New York'}
```

c) الوصول الى العناصر (Indexing)

```
# Using keys
print(person["name"])   # Output: John
print(person.get("age")) # Output: 30

# If key doesn't exist, get() returns None or a default value
print(person.get("country"))        # Output: None
print(person.get("country", "USA")) # Output: USA (default value)
```

d) تعديل القاموس (Dictionary modification)

```
# Adding a new key-value pair
person["job"] = "Engineer"
print(person)  # Output: {'name': 'John', 'age': 30, 'city': 'New York', 'job': 'Engineer'}

# Updating an existing key
person["age"] = 31
print(person["age"])  # Output: 31

# Removing a key-value pair
del person["city"]
print(person)  # Output: {'name': 'John', 'age': 31, 'job': 'Engineer'}

# Or using pop()
age = person.pop("age")
print(age)     # Output: 31
print(person)  # Output: {'name': 'John', 'job': 'Engineer'}
```

e) دول مهمة للقواميس

```
person = {"name": "John", "age": 30, "city": "New York"}

# Get all keys
print(person.keys())   # Output: dict_keys(['name', 'age', 'city'])

# Get all values
print(person.values()) # Output: dict_values(['John', 30, 'New York'])

# Get all key-value pairs as tuples
print(person.items())  # Output: dict_items([('name', 'John'), ('age', 30), ('city', 'New Yor
k')])

# Length of dictionary
print(len(person))     # Output: 3

# Check if key exists
print("name" in person) # Output: True
```

f) التكرار على القواميس

```
person = {"name": "John", "age": 30, "city": "New York"}

# Iterate over keys
for key in person:
    print(key, person[key])

# Iterate over keys and values
for key, value in person.items():
    print(f"{key}: {value}")
```

2. المجموعات (Sets)

تُعرف المجموعة في لغة بايثون بأنها حاوية برمجية تُستخدم لتخزين مجموعة من العناصر، وتستمد خصائصها من مفهوم "المجموعات" في الرياضيات. هي هيكل بيانات يركز بشكل أساسي على العلاقات المنطقية بين العناصر بدلاً من ترتيبها (Müller & Guido, 2016).

a) انشاء المجموعة (Creating sets)

```
# Empty set
my_set = set()

# Set with initial values
fruits = {"apple", "banana", "cherry"}
print(fruits)  # Output: {'banana', 'apple', 'cherry'} (order may vary)

# Note: To create a set with one element, use set() because {} creates an empty dictionary
```

b) إضافة و إزالة العناصر (Adding and removing elements)

```
fruits = {"apple", "banana"}

# Add one element
fruits.add("orange")
print(fruits)  # Output: {'banana', 'orange', 'apple'}

# Add multiple elements
fruits.update(["grape", "mango"])
print(fruits)  # Output: {'banana', 'orange', 'apple', 'grape', 'mango'}

# Remove an element (raises error if not found)
fruits.remove("banana")
print(fruits)  # Output: {'orange', 'apple', 'grape', 'mango'}
```

```
# Remove an element (does not raise error if not found)
fruits.discard("banana")  # No error even though 'banana' is not present

# Pop a random element (since sets are unordered)
popped = fruits.pop()
print(popped)  # Output: random element
```

c) العمليات على المجموعات

(الاتحاد "|" . التقاطع "&" . الفرق "-". العناصر غير المكررة "^")

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Union (all elements from both sets)
print(A | B)  # Output: {1, 2, 3, 4, 5, 6}
print(A.union(B))  # Same

# Intersection (common elements)
print(A & B)  # Output: {3, 4}
print(A.intersection(B))  # Same

# Difference (elements in A but not in B)
print(A - B)  # Output: {1, 2}
print(A.difference(B))  # Same

# Symmetric difference (elements in A or B but not in both)
print(A ^ B)  # Output: {1, 2, 5, 6}
print(A.symmetric_difference(B))  # Same
```

d) دوال مهمة في المجموعات

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

# Length
print(len(numbers))  # Output: 3

# Check if element exists
print(2 in numbers)  # Output: True

# Clear the set
numbers.clear()
print(numbers)  # Output: set()
```

3. الفروق الرئيسية بين القواميس و المجموعات:

المجموعات	القواميس	الميزة
غير مرتب	محفوظ	الترتيب
غير مسموح	مسموح في القيم فقط	التكرار
عناصر فردية	مفتاح-قيمة	الهيكل
عمليات رياضية، إزالة تكرارات	تخزين بيانات منظمة	الاستخدام

تمرين 1: استخدام القواميس لحساب تكرار الكلمات

```
text = "apple banana apple cherry banana apple"
words = text.split()

word_count = {}
for word in words:
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)  # Output: {'apple': 3, 'banana': 2, 'cherry': 1}
```

تمرين 2: إزالة التكرارات باستخدام المجموعات

```
numbers = [1, 2, 2, 3, 4, 4, 5]
unique_numbers = set(numbers)
print(unique_numbers)  # Output: {1, 2, 3, 4, 5}

# Convert back to list if needed
unique_list = list(unique_numbers)
print(unique_list)  # Output: [1, 2, 3, 4, 5] (order may vary)
```

تمرين 3: نظام إدارة الطلاب باستخدام القواميس

```
students = {
    "1001": {"name": "Alice", "grade": "A"},
    "1002": {"name": "Bob", "grade": "B"},
    "1003": {"name": "Charlie", "grade": "A"}
}

def add_student(student_id, name, grade):
    students[student_id] = {"name": name, "grade": grade}

def update_grade(student_id, grade):
    if student_id in students:
        students[student_id]["grade"] = grade
    else:
        print("Student not found")

def get_student_info(student_id):
    return students.get(student_id, "Student not found")

# Test the functions
add_student("1004", "Diana", "B")
update_grade("1001", "A+")
print(get_student_info("1001"))
```

المحور الحادي عشر: البرمجة كائنية التوجه (OOP)

1. تعريف البرمجة كائنية التوجه

البرمجة كائنية التوجه (Object-Oriented Programming) هي طريقة لتنظيم الكود تجعله أكثر تنظيماً وسهولة في الصيانة والتطوير. بدلاً من كتابة أوامر متتالية، نقوم بإنشاء "كائنات" تمثل أشياء في العالم الحقيقي (Müller & Guido, 2016).

2. المفاهيم الأساسية

- **الفئات (Classes) والكائنات(Objects)**
- **الفئة (Class)**: مثل النموذج أو التصميم الذي يحدد الخصائص والسلوكيات
- **الكائن (Object)**: نسخة فعلية مبنية على الفئة

التشبيه البسيط

- **الفئة (Class)** مثل قالب الكعك يحدد الشكل والمكونات
- **الكائن (Object)** مثل قطعة الكعك نسخة فعلية من القالب

مثال 01:

```
# تعريف فئة
class Employee:
    pass

# إنشاء كائن من الفئة
employee1 = Employee()
employee2 = Employee()
```

مثال 02:

- تعريف الفئة (Class)

```
# الفئة - مثل التصميم أو النموذج
class Dog:
    # المُهيئ - ينفذ عند إنشاء كل كائن جديد
    def __init__(self, name, color):
        # الخصائص - تميز كل كلب
        self.name = name
        self.color = color
        self.age = 0

    # الطرق - الأفعال التي يستطيع الكلب القيام بها
    def bark(self):
        print(f"{self.name} هوو هوو :يقول!")

    def celebrate_birthday(self):
        self.age += 1
        print(f"عيد ميلاد سعيد! {self.name} أصبح عمره {self.age} سنة")
```

- انشاء الكائنات (Objects)

```
# إنشاء كائنات - كل كلب هو كائن منفصل
dog1 = Dog("ريكس" ,"بني")
dog2 = Dog("لونا" ,"أسود")
dog3 = Dog("ماكس" ,"أبيض")
```

- استخدام الكائنات

```
print(f"اسم الكلب الأول: {dog1.name}")  # ريكس
print(f"لون الكلب الثاني: {dog2.color}")  # أسود

# كل كائن ينفذ الطرق بشكل مستقل
dog1.bark()  # ريكس يقول: هوو هوو!
dog2.bark()  # لونا يقول: هوو هوو!

# كل كائن يحتفظ ببياناته الخاصة
dog1.celebrate_birthday()  # ريكس أصبح عمره 1 سنة
dog1.celebrate_birthday()  # ريكس أصبح عمره 2 سنة
dog2.celebrate_birthday()  # لونا أصبح عمره 1 سنة

print(f"عمر ريكس: {dog1.age}")  # 2
print(f"عمر لونا: {dog2.age}")  # 1
```

a) الخصائص و الطرق

- **الخصائص (Attributes)**: بيانات تميز الكائن (مثل الاسم، الراتب)
- **الطرق (Methods)**: وظائف يقوم بها الكائن (مثل حساب المكافأة)

مثال تطبيقي: نموذج شركة مالية

لننشئ نظاماً بسيطاً لإدارة الموظفين في شركة مالية

```
class Employee:
    # المُهيئ (Constructor) - ينفذ تلقائياً عند إنشاء كائن جديد
    def __init__(self, name, salary, department):
        # الخصائص (Attributes)
        self.name = name
        self.salary = salary
        self.department = department
        self.bonus = 0

    # الطرق (Methods)
    def calculate_bonus(self, performance_rating):
        """حساب المكافأة بناءً على تقييم الأداء"""
        if performance_rating == "ممتاز":
            self.bonus = self.salary * 0.2  # 20% من الراتب
        elif performance_rating == "جيد جداً":
            self.bonus = self.salary * 0.1  # 10% من الراتب
        else:
            self.bonus = self.salary * 0.05  # 5% من الراتب

        return self.bonus

    def get_total_salary(self):
        """حساب الراتب الإجمالي (الراتب + المكافأة)"""
        return self.salary + self.bonus

    def display_info(self):
        """عرض معلومات الموظف"""
        print(f"الاسم: {self.name}")
        print(f"القسم: {self.department}")
        print(f"الراتب الأساسي: {self.salary}")
        print(f"المكافأة: {self.bonus}")
        print(f"الراتب الإجمالي: {self.get_total_salary()}")
        print("-" * 30)

# إنشاء كائنات (موظفين)
employee1 = Employee("أحمد محمد", 5000, "المحاسبة")
```

```
employee2 = Employee("فاطمة عبدالله", 6000, "التسويق")
employee3 = Employee("خالد إبراهيم", 7000, "التطوير")

# استخدام الطرق
employee1.calculate_bonus("ممتاز")
employee2.calculate_bonus("جيد جداً")
employee3.calculate_bonus("مقبول")

# عرض المعلومات
print("=== معلومات الموظفين ===")
employee1.display_info()
employee2.display_info()
employee3.display_info()
```

المحور الثاني عشر: الاستعانة بأدوات الذكاء الاصطناعي لتوليد الأكواد

1. أدوات الذكاء الاصطناعي في البرمجة

تمثل أدوات الذكاء الاصطناعي التوليدي (Generative AI) ، مثل ChatGPTو GitHub Copilot، ثورة تقنية في مجال هندسة البرمجيات. فهي لا تعمل مجرد محركات بحث متطورة، بل تعمل كـ "مساعد مبرمج (Pair Programmer) "ذكي، يساهم في سد الفجوة بين الفكرة المنطقية والتنفيذ البرمجي الفعلي. حيث تعتمد هذه الأدوات على نماذج لغوية ضخمة تم تدريبها على مليارات الأسطر البرمجية، مما يمنحها القدرة على:

- تحليل السياق :فهم المقصد من الطلبات المكتوبة باللغة الطبيعية وتحويلها إلى كود برمجي دقيق.
- التوليد التلقائي :(Code Generation) صياغة دوال كاملة أو اقتراح الأسطر التالية للكود بناءً على السياق الحالي، مما يقلل من المهام التكرارية المملة.
- تصحيح وتحسين الكود :اكتشاف الثغرات المنطقية واقتراح حلول بديلة أكثر كفاءة وأماناً.

مما يؤدي الى:

- تعزيز الإنتاجية :تمكين المبرمجين من التركيز على تصميم الأنظمة وحل المشكلات المعقدة بدلاً من الانشغال بالتفاصيل الإنشائية البسيطة.
- تسريع عملية التعلم :توفير شرح فوري للكود المعقد ومساعدة المبتدئين في فهم المفاهيم البرمجية عبر أمثلة عملية حية.
- رفع جودة المخرجات :تقليل الأخطاء البشرية الشائعة من خلال اقتراح أفضل الممارسات البرمجية المعتمدة عالمياً.

الخلاصة :إن أدوات الذكاء الاصطناعي ليست بديلاً عن المبرمج، بل هي قوة مضاعفة لقدراته؛ فهي تحول عملية البرمجة من مجرد كتابة تقنية إلى عملية إبداعية تعتمد على التفكير المنطقي والقيادة الذكية للأدوات (Müller & Guido, 2016).

2. امثلة عن استخدام الذكاء الاصطناعي في البرمجة؟

a) كتابة كود جديد

مثل ان تطلب من احد ادوات الذكاء الاصطناعي كتابة كود معين كما هو موضح في الصورة التالية:

```python
# يمكنك طلب: "اكتب دالة في بايثون لجمع عددين"
def add_numbers(a, b):
    return a + b

# أو: "اكتب دالة لحساب متوسط قائمة"
def calculate_average(numbers):
    if len(numbers) == 0:
        return 0
    return sum(numbers) / len(numbers)
```

b) تصحيح الاخطاء

مثال

```python
# ❌ كود به خطأ
def divide_numbers(x, y):
    return x / y  # خطأ إذا كانت y = 0

# ✅ "اطلب من الذكاء الاصطناعي: "صحح هذا الكود ليتعامل مع القسمة على صفر
def divide_numbers(x, y):
    if y == 0:
        return "خطأ: لا يمكن القسمة على صفر"
    return x / y
```

c) شرح كود معقد

مثال:

```python
# كود معقد يمكن طلب شرحه
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# "الذكاء الاصطناعي يشرح: "هذه دالة لحساب متتالية فيبوناتشي باستخدام العودية
```

d) مثال تطبيقي واقعي

سأطلب الان من برنامج الذكاء الاصطناعي Deepseek ان يقوم بإنشاء برنامج بلغة بايثون لإنشاء الة حاسبة بسيطة

```
# "اطلب: "اصنع آلة حاسبة بسيطة في بايثون"
def calculator():
    print("آلة حاسبة بسيطة")
    print("1. جمع")
    print("2. طرح")
    print("3. ضرب")
    print("4. قسمة")

    choice = input("4/3/2/1) اختر العملية): ")

    num1 = float(input("أدخل الرقم الأول: "))
    num2 = float(input("أدخل الرقم الثاني: "))

    if choice == '1':
        print(f"الناتج: {num1 + num2}")
    elif choice == '2':
        print(f"الناتج: {num1 - num2}")
    elif choice == '3':
        print(f"الناتج: {num1 * num2}")
    elif choice == '4':
        if num2 != 0:
            print(f"الناتج: {num1 / num2}")
        else:
            print("خطأ: لا يمكن القسمة على صفر")
    else:
        print("اختيار غير صحيح")

# calculator()
```

المحور الثالث عشر: التعامل مع الملفات في بايثون

يُعد التعامل مع الملفات الركيزة الأساسية لضمان "ديمومة البيانات" (Data Persistence). فبينما تُفقد البيانات المخزنة في الذاكرة المؤقتة (RAM) بمجرد إغلاق البرنامج، تتيح تقنيات التعامل مع الملفات حفظ المعلومات في وسائط التخزين الثابتة، مما يسمح باستعادتها، ومعالجتها، ومشاركتها في أي وقت لاحق.

العمليات الأساسية على الملفات: فتح، قراءة، وكتابة

a) فتح الملفات (Opening files)

```
# الطريقة الأساسية
file = open("data.txt", "r")  # فتح للقراءة
file.close()  # إغلاق الملف

# الطريقة الموصى بها (تغلق الملف تلقائياً)
with open("data.txt", "r") as file:
    # التعامل مع الملف هنا
    pass
```

b) أوضاع فتح الملفات

الوصف	الوضع
قراءة فقط	"r"
كتابة (تمسح المحتوى القديم)	"w"
إضافة (للنهاية دون مسح القديم)	"a"
قراءة وكتابة	"r+"

أمثلة أساسية

مثال 1: كتابة ملف نصي

```
# كتابة بيانات إلى ملف
with open("economic_data.txt", "w", encoding="utf-8") as file:
    file.write("البيانات الاقتصادية\n")
    file.write("==================\n")
    file.write("الناتج المحلي: 500 مليار\n")
    file.write("معدل التضخم: 2.5%\n")
    file.write("سعر الصرف: 3.75 ر.ق للدولار\n")

print("تم حفظ البيانات بنجاح!")
```

مثال 2: قراءة ملف نصي

```
# قراءة الملف كاملاً
with open("economic_data.txt", "r", encoding="utf-8") as file:
    content = file.read()
    print(content)

print("\n" + "="*40 + "\n")

# قراءة الملف سطراً سطراً
with open("economic_data.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())  # strip() يزيل المسافات الزائدة
```

مثال 3: إضافة بيانات إلى ملف

```
# إضافة بيانات جديدة دون مسح القديمة
with open("economic_data.txt", "a", encoding="utf-8") as file:
    file.write("\n")  # سطر فارغ للفصل
    file.write("بيانات جديدة:\n")
    file.write("5.2 :معدل البطالة%\n")
    file.write("200 :احتياطي النقد الأجنبي مليار\n")

print("تمت إضافة البيانات الجديدة!")
```

تمرين شامل يمكن نسخة في محرر البايثون و تنفيذه و الاستفادة منه.

نظام حفظ وقراءة البيانات

```
class EconomicDataManager:
    def __init__(self, filename="economic_data.csv"):
        self.filename = filename
        self.data_list = []

    def add_data(self, economic_data):
        """إضافة بيانات اقتصادية جديدة"""
        self.data_list.append(economic_data)

    def save_to_file(self):
        """حفظ جميع البيانات إلى ملف"""
        try:
            with open(self.filename, "w", encoding="utf-8") as file:
                # كتابة رأس الملف
                file.write("country,gdp,inflation,unemployment,year\n")

                # كتابة البيانات
                for data in self.data_list:
                    file.write(data.to_string() + "\n")

            print(f"تم حفظ {len(self.data_list)} سجل في {self.filename}")
            return True

        except Exception as e:
            print(f"خطأ في الحفظ: {e}")
            return False

    def load_from_file(self):
        """قراءة البيانات من ملف"""
        try:
```

```
        with open(self.filename, "r", encoding="utf-8") as file:
            lines = file.readlines()

            # تخطي رأس الملف (السطر الأول)
            for line in lines[1:]:
                line = line.strip()
                if line: # إذا لم يكن السطر فارغاً
                    data = EconomicData.from_string(line)
                    self.data_list.append(data)

        print(f"تم تحميل {len(self.data_list)} سجل من {self.filename}")
        return True

    except FileNotFoundError:
        print("الملف غير موجود. سيتم إنشاؤه عند الحفظ.")
        return False
    except Exception as e:
        print(f"خطأ في القراءة: {e}")
        return False

def display_all_data(self):
    """عرض جميع البيانات"""
    if not self.data_list:
        print("لا توجد بيانات لعرضها")
        return

    print("=== جميع البيانات الاقتصادية ===")
    for data in self.data_list:
        data.display()

def search_by_country(self, country):
    """البحث عن بيانات بلد معين"""
    results = [data for data in self.data_list if data.country.lower() == country.lower()]

    if results:
        print(f"=== نتائج البحث عن {country} ===")
        for data in results:
            data.display()
    else:
        print(f"لا توجد بيانات للبلد {country}")

def get_statistics(self):
    """حساب إحصائيات عن البيانات"""
    if not self.data_list:
        print("لا توجد بيانات لحساب الإحصائيات")
        return

    gdp_values = [data.gdp for data in self.data_list]
    inflation_values = [data.inflation for data in self.data_list]
```

```
print("=== الإحصائيات الاقتصادية ===")
print(f"متوسط الناتج المحلي: {sum(gdp_values) / len(gdp_values):.2f} مليار")
print(f"أعلى ناتج محلي: {max(gdp_values)} مليار")
print(f"أقل ناتج محلي: {min(gdp_values)} مليار")
print(f"متوسط التضخم: {sum(inflation_values) / len(inflation_values):.2f}%")
```

مثال استخدام النظام

```
# إنشاء مدير البيانات
manager = EconomicDataManager()

# إنشاء بيانات اقتصادية عينة
data1 = EconomicData("السعودية", 800, 2.5, 5.8, 2023)
data2 = EconomicData("الإمارات", 450, 1.8, 3.2, 2023)
data3 = EconomicData("مصر", 350, 18.5, 7.1, 2023)
data4 = EconomicData("السعودية", 750, 3.1, 6.2, 2022)

# إضافة البيانات
manager.add_data(data1)
manager.add_data(data2)
manager.add_data(data3)
manager.add_data(data4)

# حفظ البيانات في ملف
manager.save_to_file()

# عرض البيانات
manager.display_all_data()

# البحث عن بيانات بلد معين
manager.search_by_country("السعودية")

# عرض الإحصائيات
manager.get_statistics()
```

المحور الرابع عشر: مشاريع مقترحة للاحتراف

مشروع 01:

برنامج تحليل مصاريف الأسرة الشهري

```
# محلل المصاريف الشهرية البسيط
class ExpenseAnalyzer:
    def __init__(self):
        self.expenses = {}
        self.monthly_income = 0

    def set_income(self, income):
        """تحديد الدخل الشهري"""
        self.monthly_income = income
        print(f"تم تحديد الدخل الشهري: {income} دج")

    def add_expense(self, category, amount):
        """إضافة مصروف جديد"""
        if category in self.expenses:
            self.expenses[category] += amount
        else:
            self.expenses[category] = amount
        print(f"تم إضافة {amount} دج لـ {category}")

    def calculate_total_expenses(self):
        """حساب إجمالي المصروفات"""
        total = 0
        for amount in self.expenses.values():
            total += amount
        return total

    def analyze_expenses(self):
        """تحليل المصروفات"""
        if self.monthly_income == 0:
            print("الرجاء تحديد الدخل الشهري أولاً")
            return

        total_expenses = self.calculate_total_expenses()
        savings = self.monthly_income - total_expenses
        savings_percentage = (savings / self.monthly_income) * 100

        print("\n" + "="*40)
        print("تحليل المصروفات الشهرية")
        print("="*40)

        print(f"الدخل الشهري: {self.monthly_income} دج")
```

```
        print(f"إجمالي المصروفات: {total_expenses} دج")
        print(f"المدخرات: {savings} دج")
        print(f"نسبة المدخرات: {savings_percentage:.1f}%")

        # تقييم الأداء المالي
        if savings_percentage >= 20:
            rating = "ممتاز"
        elif savings_percentage >= 10:
            rating = "جيد"
        elif savings_percentage >= 0:
            rating = "مقبول"
        else:
            rating = "في خطر"

        print(f"التقييم المالي: {rating}")

    def show_expense_breakdown(self):
        """عرض توزيع المصروفات"""
        if not self.expenses:
            print("لا توجد مصروفات مسجلة")
            return

        total = self.calculate_total_expenses()

        print("\nتفصيل المصروفات:")
        print("-" * 30)

        for category, amount in self.expenses.items():
            percentage = (amount / total) * 100
            print(f"{category}: {amount} دج ({percentage:.1f}%)")

    def find_largest_expense(self):
        """إيجاد أكبر مصروف"""
        if not self.expenses:
            print("لا توجد مصروفات")
            return

        max_category = max(self.expenses, key=self.expenses.get)
        max_amount = self.expenses[max_category]

        print(f"\nأكبر مصروف: {max_category} - {max_amount} دج")
        return max_category, max_amount

# مثال استخدام البرنامج
def main():
    analyzer = ExpenseAnalyzer()

    # تحديد الدخل
    analyzer.set_income(10000)
```

```
    # إضافة المصروفات
    analyzer.add_expense("إيجار", 3000)
    analyzer.add_expense("طعام", 2000)
    analyzer.add_expense("مواصلات", 800)
    analyzer.add_expense("تعليم", 1500)
    analyzer.add_expense("ترفيه", 500)
    analyzer.add_expense("فواتير", 700)

    # التحليل
    analyzer.analyze_expenses()
    analyzer.show_expense_breakdown()
    analyzer.find_largest_expense()

# تشغيل البرنامج
if __name__ == "__main__":
    main()
```

النتائج المتوقعة:

```
تم تحديد الدخل الشهري: 10000 دج
تم إضافة 3000 دج لـ إيجار
تم إضافة 2000 دج لـ طعام
تم إضافة 800 دج لـ مواصلات
تم إضافة 1500 دج لـ تعليم
تم إضافة 500 دج لـ ترفيه
تم إضافة 700 دج لـ فواتير

==========================================
تحليل المصروفات الشهرية
==========================================
الدخل الشهري: 10000 دج
إجمالي المصروفات: 8500 دج
المدخرات: 1500 دج
نسبة المدخرات: 15.0%
التقييم المالي: جيد

تفصيل المصروفات:
------------------------------
إيجار: 3000 دج (35.3%)
طعام: 2000 دج (23.5%)
مواصلات: 800 دج (9.4%)
تعليم: 1500 دج (17.6%)
ترفيه: 500 دج (5.9%)
فواتير: 700 دج (8.2%)

أكبر مصروف: إيجار - 3000 دج
```

مشروع 02:

مشروع تحليل البيانات الاقتصادية باستخدام أساسيات بايثون

مشروع تحليل بيانات مالية - محلل الاقتصاد البسيط

لنفترض أننا نحصل على بيانات الأسعار يومياً ونريد حساب:

- متوسط السعر
- أعلى وأقل سعر
- اتجاه السوق (صعودي أم هبوطي)
- تقييم أداء السهم

سنقوم بتخزين البيانات في قوائم ونستخدم الدوال والحلقات الشرطية للتحليل.

الخطوات:

1. إدخال البيانات: سندخل بيانات الأسعار كقائمة من القيم.
2. حساب المتوسط الحسابي للأسعار.
3. أعلى وأقل سعر.
4. حساب التغير اليومي والنسبة المئوية للتغير.
5. تحديد الاتجاه (صعودي إذا كان آخر سعر أعلى من المتوسط، وهبوطي إذا كان أقل).

لنبدأ ببرنامج بسيط ثم نطوره.

مثال على البيانات: [100, 102, 101, 105, 104, 108, 107]

نريد إخراج:

- متوسط السعر
- أعلى سعر
- أقل سعر
- التغير اليومي (آخر يوم مقارنة باليوم الأول)
- النسبة المئوية للتغير
- الاتجاه

سنقوم بكتابة دوال منفصلة لكل مهمة.

لن نستخدم أي مكتبات خارجية، فقط الأساسيات.

سأكتب البرنامج على شكل دالة رئيسية وعدة دوال مساعدة.

سأضيف أيضا إمكانية إدخال البيانات من المستخدم.

لنبدأ بالبرنامج:

```python
# مشروع نهاية الفصل: محلل البيانات الاقتصادية
# تم استخدام الأساسيات فقط دون استيراد مكتبات

class EconomicAnalyzer:
    def __init__(self):
        self.data = []
        self.countries = []

    def add_economic_data(self, country, gdp, inflation, unemployment, year):
        """إضافة بيانات اقتصادية جديدة"""
        data_point = {
            'country': country,
            'gdp': gdp,
            'inflation': inflation,
            'unemployment': unemployment,
            'year': year
        }
        self.data.append(data_point)

        if country not in self.countries:
            self.countries.append(country)

    def load_sample_data(self):
        """تحميل بيانات نموذجية للاختبار"""
        sample_data = [
            ('السعودية', 800, 2.5, 5.8, 2023),
            ('الإمارات', 450, 1.8, 3.2, 2023),
            ('مصر', 350, 18.5, 7.1, 2023),
            ('قطر', 200, 2.1, 0.8, 2023),
            ('الكويت', 150, 2.2, 2.1, 2023),
            ('السعودية', 750, 3.1, 6.2, 2022),
            ('الإمارات', 420, 2.0, 3.5, 2022),
            ('مصر', 320, 15.2, 7.8, 2022)
        ]

        for item in sample_data:
            self.add_economic_data(*item)

        print("تم تحميل البيانات النموذجية بنجاح!")

    def calculate_average(self, values):
        """حساب المتوسط الحسابي"""
        if len(values) == 0:
            return 0
        return sum(values) / len(values)

    def find_max_min(self, values):
```

```
    """إيجاد القيم القصوى والدنيا"""
    if len(values) == 0:
        return None, None
    return max(values), min(values)

def analyze_by_country(self, country):
    """تحليل بيانات بلد معين"""
    country_data = [d for d in self.data if d['country'] == country]

    if not country_data:
        print(f"لا توجد بيانات للبلد: {country}")
        return

    gdp_values = [d['gdp'] for d in country_data]
    inflation_values = [d['inflation'] for d in country_data]
    unemployment_values = [d['unemployment'] for d in country_data]
    years = [d['year'] for d in country_data]

    print(f"\n{'='*50}")
    print(f"تحليل البيانات الاقتصادية لـ: {country}")
    print(f"{'='*50}")

    # تحليل الناتج المحلي
    avg_gdp = self.calculate_average(gdp_values)
    max_gdp, min_gdp = self.find_max_min(gdp_values)
    max_gdp_year = years[gdp_values.index(max_gdp)]
    min_gdp_year = years[gdp_values.index(min_gdp)]

    print(f"الناتج المحلي الإجمالي:")
    print(f"  - المتوسط: {avg_gdp:.2f} مليار دولار")
    print(f"  - الأعلى: {max_gdp} مليار دولار (سنة {max_gdp_year})")
    print(f"  - الأدنى: {min_gdp} مليار دولار (سنة {min_gdp_year})")

    # تحليل التضخم
    avg_inflation = self.calculate_average(inflation_values)
    max_inflation, min_inflation = self.find_max_min(inflation_values)

    print(f"\nمعدل التضخم:")
    print(f"  - المتوسط: {avg_inflation:.2f}%")
    print(f"  - الأعلى: {max_inflation}%")
    print(f"  - الأدنى: {min_inflation}%")

    # تحليل البطالة
    avg_unemployment = self.calculate_average(unemployment_values)
    max_unemployment, min_unemployment = self.find_max_min(unemployment_values)

    print(f"\nمعدل البطالة:")
    print(f"  - المتوسط: {avg_unemployment:.2f}%")
    print(f"  - الأعلى: {max_unemployment}%")
```

```
        print(f"  - الأدنى: {min_unemployment}%")

        # تقييم الأداء
        self.evaluate_performance(country, avg_inflation, avg_unemployment)

    def analyze_all_countries(self):
        """تحليل شامل لجميع البلدان"""
        if not self.data:
            print("لا توجد بيانات للتحليل")
            return

        print(f"\n{'='*60}")
        print("تحليل شامل لجميع البيانات الاقتصادية")
        print(f"{'='*60}")

        # إحصائيات عامة
        all_gdp = [d['gdp'] for d in self.data]
        all_inflation = [d['inflation'] for d in self.data]
        all_unemployment = [d['unemployment'] for d in self.data]

        print("إحصائيات عامة:")
        print(f"  - متوسط الناتج المحلي: {self.calculate_average(all_gdp):.2f} مليار دولار")
        print(f"  - متوسط التضخم: {self.calculate_average(all_inflation):.2f}%")
        print(f"  - متوسط البطالة: {self.calculate_average(all_unemployment):.2f}%")

        # أفضل وأسوأ البلدان في كل مؤشر
        latest_data = self.get_latest_data()

        print("\nأفضل البلدان (آخر بيانات متاحة):")
        self.find_best_worst_countries(latest_data)

    def get_latest_data(self):
        """الحصول على أحدث بيانات لكل بلد"""
        latest_data = {}

        for item in self.data:
            country = item['country']
            year = item['year']

            if country not in latest_data or year > latest_data[country]['year']:
                latest_data[country] = item

        return latest_data

    def find_best_worst_countries(self, latest_data):
        """إيجاد أفضل وأسوأ البلدان في كل مؤشر"""
        # أفضل وأسوأ الناتج المحلي
        countries_by_gdp = sorted(latest_data.items(), key=lambda x: x[1]['gdp'], reverse=True)
```

```
        print(f" - أعلى ناتج محلي: {countries_by_gdp[0][0]} ({countries_by_gdp[0][1]['gdp']} مليار)")
        print(f" - أقل ناتج محلي: {countries_by_gdp[-1][0]} ({countries_by_gdp[-1][1]['gdp']} مليار)")

        # أفضل وأسوأ التضخم (الأقل هو الأفضل)
        countries_by_inflation = sorted(latest_data.items(), key=lambda x: x[1]['inflation'])
        print(f" - أفضل تضخم: {countries_by_inflation[0][0]} ({countries_by_inflation[0][1]['inflation']}%)")
        print(f" - أسوأ تضخم: {countries_by_inflation[-1][0]} ({countries_by_inflation[-1][1]['inflation']}%)")

        # أفضل وأسوأ البطالة (الأقل هو الأفضل)
        countries_by_unemployment = sorted(latest_data.items(), key=lambda x: x[1]['unemployment'])
        print(f" - أفضل بطالة: {countries_by_unemployment[0][0]} ({countries_by_unemployment[0][1]['unemployment']}%)")
        print(f" - أسوأ بطالة: {countries_by_unemployment[-1][0]} ({countries_by_unemployment[-1][1]['unemployment']}%)")

    def evaluate_performance(self, country, inflation, unemployment):
        """تقييم أداء البلد الاقتصادي"""
        print(f"\n تقييم الأداء الاقتصادي لـ {country}:")

        # تقييم التضخم
        if inflation < 3:
            inflation_rating = "ممتاز"
        elif inflation < 5:
            inflation_rating = "جيد"
        elif inflation < 10:
            inflation_rating = "متوسط"
        else:
            inflation_rating = "ضعيف"

        # تقييم البطالة
        if unemployment < 3:
            unemployment_rating = "ممتاز"
        elif unemployment < 5:
            unemployment_rating = "جيد"
        elif unemployment < 8:
            unemployment_rating = "متوسط"
        else:
            unemployment_rating = "ضعيف"

        print(f" - تقييم التضخم: {inflation_rating}")
        print(f" - تقييم البطالة: {unemployment_rating}")

        # التقييم العام
```

```
        if inflation_rating in ["ممتاز", "جيد"] and unemployment_rating in ["ممتاز", "جيد
"]:
            overall_rating = "ممتاز"
        elif inflation_rating in ["ممتاز", "جيد", "متوسط"] and unemployment_rating in ["
ممتاز", "جيد", "متوسط"]:
            overall_rating = "جيد"
        else:
            overall_rating = "يحتاج تحسين"

        print(f" - التقييم العام: {overall_rating}")

    def compare_two_countries(self, country1, country2):
        """مقارنة بين بلدين"""
        latest_data = self.get_latest_data()

        if country1 not in latest_data or country2 not in latest_data:
            print("احد البلدين غير موجود في البيانات")
            return

        data1 = latest_data[country1]
        data2 = latest_data[country2]

        print(f"\n{'='*50}")
        print(f" مقارنة بين {country1} و{country2}")
        print(f"{'='*50}")

        print(f"الناتج المحلي:")
        print(f" - {country1}: {data1['gdp']} مليار دولار")
        print(f" - {country2}: {data2['gdp']} مليار دولار")

        gdp_diff = data1['gdp'] - data2['gdp']
        if gdp_diff > 0:
            print(f" - الفرق: {country1} أعلى بمقدار {gdp_diff} مليار دولار")
        else:
            print(f" - الفرق: {country2} أعلى بمقدار {abs(gdp_diff)} مليار دولار")

        print(f"\nالتضخم:")
        print(f" - {country1}: {data1['inflation']}%")
        print(f" - {country2}: {data2['inflation']}%")

        inflation_diff = data1['inflation'] - data2['inflation']
        if inflation_diff < 0:
            print(f" - {country1} لديه تضخم أقل بمقدار {abs(inflation_diff):.1f}%")
        else:
            print(f" - {country2} لديه تضخم أقل بمقدار {inflation_diff:.1f}%")

        print(f"\nالبطالة:")
        print(f" - {country1}: {data1['unemployment']}%")
        print(f" - {country2}: {data2['unemployment']}%")
```

```
        unemployment_diff = data1['unemployment'] - data2['unemployment']
        if unemployment_diff < 0:
            print(f" - {country1} لديه بطالة أقل بمقدار {abs(unemployment_diff):.1f}%")
        else:
            print(f" - {country2} لديه بطالة أقل بمقدار {unemployment_diff:.1f}%")

    def save_report_to_file(self, filename="economic_report.txt"):
        """حفظ التقرير في ملف"""
        try:
            with open(filename, "w", encoding="utf-8") as file:
                file.write("تقرير التحليل الاقتصادي\n")
                file.write("=" * 40 + "\n\n")

                file.write(f"عدد البلدان: {len(self.countries)}\n")
                file.write(f"إجمالي البيانات: {len(self.data)}\n\n")

                latest_data = self.get_latest_data()

                file.write("أحدث البيانات المتاحة:\n")
                file.write("-" * 30 + "\n")

                for country, data in latest_data.items():
                    file.write(f"{country}:\n")
                    file.write(f" - الناتج المحلي: {data['gdp']} مليار دولار\n")
                    file.write(f" - التضخم: {data['inflation']}%\n")
                    file.write(f" - البطالة: {data['unemployment']}%\n")
                    file.write(f" - السنة: {data['year']}\n\n")

                # إحصائيات
                all_gdp = [d['gdp'] for d in self.data]
                all_inflation = [d['inflation'] for d in self.data]

                file.write("إحصائيات عامة:\n")
                file.write("-" * 30 + "\n")
                file.write(f"متوسط الناتج المحلي: {self.calculate_average(all_gdp):.2f} مليار دولار\n")
                file.write(f"متوسط التضخم: {self.calculate_average(all_inflation):.2f}%\n")

                max_gdp, min_gdp = self.find_max_min(all_gdp)
                file.write(f"أعلى ناتج محلي: {max_gdp} مليار دولار\n")
                file.write(f"أقل ناتج محلي: {min_gdp} مليار دولار\n")

            print(f"تم حفظ التقرير في ملف: {filename}")

        except Exception as e:
            print(f"حدث خطأ أثناء حفظ الملف: {e}")
```

```
def display_menu(self):
    """عرض القائمة الرئيسية"""
    print("\n" + "="*60)
    print("محلل البيانات الاقتصادية - القائمة الرئيسية")
    print("="*60)
    print("1. تحميل بيانات نموذجية")
    print("2. تحليل بلد معين")
    print("3. تحليل شامل لجميع البلدان")
    print("4. مقارنة بين بلدين")
    print("5. حفظ التقرير في ملف")
    print("6. إضافة بيانات جديدة")
    print("7. عرض جميع البلدان")
    print("0. الخروج")

def run(self):
    """تشغيل البرنامج"""
    print("مرحباً بك في محلل البيانات الاقتصادية!")

    while True:
        self.display_menu()
        choice = input("\nاختر الخيار المناسب (0-7): ").strip()

        if choice == "0":
            print("شكراً لاستخدامك البرنامج. إلى اللقاء!")
            break

        elif choice == "1":
            self.load_sample_data()

        elif choice == "2":
            if not self.data:
                print("لا توجد بيانات. قم بتحميل البيانات أولاً.")
                continue

            print("البلدان المتاحة:", ", ".join(self.countries))
            country = input("أدخل اسم البلد: ").strip()
            self.analyze_by_country(country)

        elif choice == "3":
            if not self.data:
                print("لا توجد بيانات. قم بتحميل البيانات أولاً.")
                continue
            self.analyze_all_countries()

        elif choice == "4":
            if not self.data:
                print("لا توجد بيانات. قم بتحميل البيانات أولاً.")
                continue
```

```
            print("البلدان المتاحة:", ", ".join(self.countries))
            country1 = input("أدخل اسم البلد الأول: ").strip()
            country2 = input("أدخل اسم البلد الثاني: ").strip()
            self.compare_two_countries(country1, country2)

        elif choice == "5":
            if not self.data:
                print("لا توجد بيانات. قم بتحميل البيانات أولاً.")
                continue

            filename = input("أدخل اسم الملف (اختياري): ").strip()
            if not filename:
                filename = "economic_report.txt"
            self.save_report_to_file(filename)

        elif choice == "6":
            country = input("أدخل اسم البلد: ").strip()
            try:
                gdp = float(input("الناتج المحلي (مليار دولار): "))
                inflation = float(input("معدل التضخم (%): "))
                unemployment = float(input("معدل البطالة (%): "))
                year = int(input("السنة: "))

                self.add_economic_data(country, gdp, inflation, unemployment, year)
                print("تم إضافة البيانات بنجاح!")
            except ValueError:
                print("خطأ: تأكد من إدخال أرقام صحيحة")

        elif choice == "7":
            if not self.data:
                print("لا توجد بيانات.")
            else:
                print("البلدان في النظام:", ", ".join(self.countries))
                print(f"إجمالي عدد البيانات: {len(self.data)}")

        else:
            print("خيار غير صحيح. حاول مرة أخرى.")

# تشغيل البرنامج
if __name__ == "__main__":
    analyzer = EconomicAnalyzer()
    analyzer.run()
```

ميزات المشروع:

1. **الأساسيات المستخدمة:**

- الفئات والكائنات(OOP)
- القوائم والقواميس
- الحلقات(for, while)
- الشروط(if, elif, else)
- الدوال(Functions)
- التعامل مع الملفات
- معالجة الأخطاء

2. **الوظائف الرئيسية:**

- تحليل بيانات اقتصادية لبلدان مختلفة
- مقارنة بين البلدان
- تقييم الأداء الاقتصادي
- حفظ التقارير في ملفات
- واجهة مستخدم تفاعلية

3. **المؤشرات الاقتصادية:**

- الناتج المحلي الإجمالي(GDP)
- معدل التضخم
- معدل البطالة

4. **التحليلات المقدمة:**

- المتوسطات الحسابية
- القيم القصوى والدنيا
- التقييم النسبي
- المقارنات

5. **كيفية الاستخدام:**

1. قم بتشغيل البرنامج
2. اختر "1" لتحميل بيانات نموذجية
3. استخدم الخيارات المختلفة للتحليل والمقارنة
4. اختر "5" لحفظ التقرير في ملف

هذا المشروع يوضح كيف يمكن استخدام أساسيات البرمجة لبناء تطبيق مفيد لتحليل البيانات الاقتصادية دون الحاجة لمكتبات خارجية!

خاتمة:

ختاما نرجوا ان نكون قد قطعنا معًا رحلة ممتعة ومثمرة في عالم البرمجة بلغة بايثون، بدأناها بالتعرف على مفهوم الخوارزميات كأساس للتفكير المنظم، ثم انتقلنا خطوة بخطوة من كتابة أول برنامج بسيط إلى التعامل مع المتغيرات، والنصوص، والقوائم، والقواميس، وهندسة الأوامر، وصولاً إلى التعامل مع الملفات.

لم تكن هذه المطبوعة مجرد مجموعة من الأكواد الجافة، بل محاولة لبناء عقلية برمجية لديك، عقلية تحلل المشكلات إلى خطوات صغيرة، وتفكر بطريقة منطقية، وتستطيع تحويل فكرة ما إلى تطبيق عملي.

ماذا أخذت معك؟

- لغة بايثون أصبحت الآن بين يديك أداةً تساعدك في معالجة البيانات، وتحليل المؤشرات الاقتصادية، وأتمتة الحسابات المتكررة.
- أصبحت تعرف كيف تخزن المعلومات، كيف تبحث فيها، كيف تتخذ قرارات ضمن برنامجك، وكيف تقرأ وتكتب الملفات.
- الأهم من كل ذلك: صرت تعلم أن البرمجة ليست صعبة، بل تحتاج فقط إلى ممارسة وصبر وتدرج.

نصيحة أخيرة لطالب الاقتصاد الدولي (ولكل مبتدئ)

لا تتوقف عند هذه المطبوعة. فالبرمجة كأي لغة جديدة، تتعلمها بالممارسة اليومية. جرب أن تكتب برامج صغيرة لحل مشكلات حقيقية تواجهها في دراستك الاقتصادية: احسب معدلات النمو، رتب بيانات التجارة الخارجية، ابحث عن أعلى قيمة لسعر صرف... كل هذه تطبيقات بسيطة لكنها ستعمق فهمك.

وتذكر دائمًا :أخطاء البرمجة ليست فشلاً، بل هي محطات تعلم .كل رسالة خطأ (Error) تخبرك بشيء جديد، وكل كود لا يعمل هو فرصة لفهم أعمق.

كلمة شكر وتقدير

شكرًا لك على وقتك وجهدك في قراءة هذه المطبوعة. فخورون بأنك اتخذت الخطوة الأولى في طريق البرمجة، واثقون أن ما تعلمته هنا سيكون نافعًا لك في مسيرتك الدراسية والمهنية، سواء اخترت التخصص في الاقتصاد الرقمي، أو تحليل البيانات، أو أي مجال آخر يحتاج إلى التفكير المنظم.

في النهاية، تذكر: ليست بايثون هي الهدف، بل ما يمكنك أن تصنعه بها.

نتمنى لك كل التوفيق، وإلى لقاء في مطبوعات متقدمة إن شاء الله.

"البرمجة ليست مجرد أداة تقنية، بل هي طريقة جديدة لرؤية العالم وحل مشكلاته".

قائمة المراجع

بن يحيى ثاني نسرين. (2017). *مدخل إلى الخوارزميات*. جامعة الجزائر 3.

مهيبل وسام. (2019). *الخوارزميات*. جامعة الجزائر 3.

McKinney, W. (2017). *Python for data analysis: Data wrangling with pandas, NumPy, and Jupyter* (2nd ed.). O'Reilly Media.

Downey, A. B. (2014). *Think stats: Exploratory data analysis in Python* (2nd ed.). O'Reilly Media.

VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data*. O'Reilly Media.

Hilpisch, Y. (2018). *Python for finance: Mastering data-driven finance* (2nd ed.). O'Reilly Media.

Müller, A. C., & Guido, S. (2016). *Introduction to machine learning with Python: A guide for* data scientists. O'Reilly Media.

Nelli, F. (2015). Python data analytics: Data analysis and science using PANDAs, Matplotlib and the Python Programming Language. Apress.