تعرُّف الوحدة

تُعدُّ البرامج الحاسوبية من صميم المنظمات والأعمال الحديثة، فهي ضرورية لتقديم المنتجات والخدمات ومساعدة المنظمات على الاستجابة لبيئة الأعمال التي تتغير باستمرار. إن دراسة هذه الوحدة ستنقلك من مستخدم لبرامج الحاسوب إلى مطور لبرامج الحاسوب يمكنه تصميم وبرمجة حلول لمجموعة متنوعة من المشكلات. وستتعلم استخدام مهارات التفكير الحاسوبي لتحليل المشكلات، وتحديد الأنماط، وتفكيك المهام المعقدة إلى أجزاء أصغر وأسهل في الحل. تتمحور البرمجة بشكل عام حول حل المشكلات، وستقوم هذه الوحدة بصقل مهاراتك التحليلية ومهارات حل المشكلات استعدادًا للحياة العملية أو لمواصلة المراحل الدراسية.

كيفية إجراء التقييم

التقييم

سيُجرى لك التقييم من خلال سلسلة من المهام التي يحددها معلمك.

ستُقيَّم هذه الوحدة داخليًّا عبر سلسلة من المهام التي سيحددها معلمك. وستجد طوال دراستك لهذه الوحدة أنشطة تمرين تقييمي ستساعدك على العمل حتى تصل إلى تقييمك، علمًا بأن إنجاز هذه الأنشطة لا يعني أنك قد حققت درجة محددة، بل يعني أنك أجريت أبحاثًا أو تحضيرات مفيدة ستكون ذات صلة بمهمتك النهائية.

لكي تتمكن من إنجاز المهام المحددة في واجباتك، من المهم التأكد من أنك قد استوفيت جميع معايير درجة النجاح. ويمكنك القيام بذلك بالعمل على الواجبات التي تُكلَّف بها.

إذا كنت تسعى إلى تحقيق درجة التفوق أو الامتياز، يجب عليك أيضًا التأكد من عرض المعلومات في واجبك بالطريقة التي تتطلبها معايير التقييم ذات الصلة. فعلى سبيل المثال، تتطلب درجة التفوق أن تُحلِّل الموضوع وتعرض مبرراته، بينما تتطلب درجة الامتياز أن تُقيّمه.

ستتألف الواجبات التي يحددها معلمك من عدد من المهام المصممة لاستيفاء المعايير الواردة في جدول الصفحة الآتية. من المحتمل أن تكون المهمة الأولى مهمة مكتوبة تعتمد على البحث وتتطلب منك شرح مهارات التفكير الحاسوبي ومبادئ برمجة الحاسوب، بينما ستتضمن المهمة الثانية أنشطة عملية مثل:

- تصميم حل برمجي لتلبية متطلبات العميل
- تطوير حل برمجي لتلبية متطلبات العميل.

معايير التقييم

يوضح لك هذا الجدول ما يجب عليك القيام به من أجل الحصول على درجة النجاح أو التفوق أو الامتياز.

النجاح	التفوق	الامتياز
نتاج التعلم أ دراسة مهارات التفكير الحاسوبي ومبادئ البرمجة الحاسوبية		
A.P1 شرح كيفية تطبيق مهارات التفكير الحاسوبي في إيجاد الحلول التي يمكن تفسيرها إلى تطبيقات برمجية. **تمرين تقييمي 4.1** **A.P2** شرح كيفية تطبيق مبادئ البرمجة الحاسوبية بلغات مختلفة لإنتاج تطبيقات برمجية. **تمرين تقييمي 4.1** **A.P3** شرح كيفية استخدام مبادئ تصميم البرامج لإنتاج تطبيقات برمجية عالية الجودة تلبي احتياجات المستخدمين. **تمرين تقييمي 4.1**	**A.M1** تحليل كيفية تأثير مهارات التفكير الحاسوبي في تصميم البرامج وجودة التطبيقات البرمجية المنتجة. **تمرين تقييمي 4.1**	**A.D1** تقييم كيفية تأثير مهارات التفكير الحاسوبي في تصميم البرامج وجودة التطبيقات البرمجية المنتجة. **تمرين تقييمي 4.1**
نتاج التعلم ب تصميم حل برمجي لتلبية متطلبات العميل		
B.P4 إنتاج تصميم لبرنامج حاسوبي لتلبية متطلبات عميل **تمرين تقييمي 4.2** **B.P5** مراجعة التصميم مع الآخرين لتحديد وتوجيه التحسين على الحل المقترح **تمرين تقييمي 4.2**	**B.M2** تبرير قرارات التصميم، مع توضيح كيف سيؤدي التصميم إلى حل فعال. **تمرين تقييمي 4.2**	**BC.D2** تقييم التصميم والبرنامج الحاسوبي المحسن مقابل متطلبات العميل. **تمرين تقييمي 4.2**
نتاج التعلم ج تطوير حل برمجي لتلبية متطلبات العميل		
C.P6 إنتاج برنامج حاسوبي يلبي متطلبات العميل **تمرين تقييمي 4.2** **C.P7** مراجعة مدى تلبية البرنامج الحاسوبي لمتطلبات العميل. **تمرين تقييمي 4.2**	**C.M3** تحسين البرنامج الحاسوبي لتلبية متطلبات العميل. **تمرين تقييمي 4.2**	**BC.D3** إظهار المسؤولية الفردية والإبداع والإدارة الذاتية الفعالة في تصميم وتطوير ومراجعة البرنامج الحاسوبي. **تمرين تقييمي 4.2**

بدء النشاط

يحل المبرمجون ذوو الخبرة العديد من المشكلات قبل أن يلمسوا لوحات مفاتيح حواسيبهم. اكتب قائمة بالمهام والأسئلة التي تعتقد أن المبرمج سيفكر فيها عند تصميم وبناء برنامج حاسوبي. في نهاية هذه الوحدة، راجع القائمة التي أعددتها واكتشف ما إذا فاتتك أي فكرة، مثل المهام أو الأسئلة المحددة.

نتاجات التعلم

ستتعلم في هذه الوحدة:

أ} دراسة مهارات التفكير الحاسوبي ومبادئ البرمجة الحاسوبية

ب} تصميم حل برمجي لتلبية متطلبات العميل

ج} تطوير حل برمجي لتلبية متطلبات العميل.

أ} دراسة مهارات التفكير الحاسوبي ومبادئ البرمجة الحاسوبية

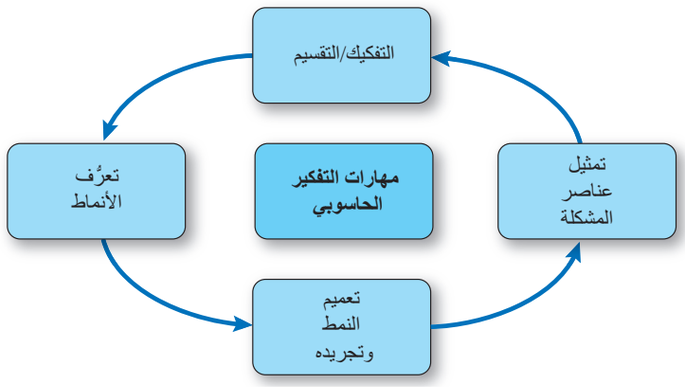
البرمجة ليست مجرد مسألة تعلم كيفية استخدام لغة البرمجة التي تحظى بشعبية أو طلب كبير في الوقت الحالي، بل هي تعلم كيفية حل المشكلات من خلال التفكير بطريقة منطقية وفهم ما هي لغة البرمجة، وما يمكنها فعله وكيفية استخدامها.

المهارات

- مهارات التحليل واتخاذ القرار

مهارات التفكير الحاسوبي

تعتمد برمجة الحاسوب الناجحة على ممارستك لمهارات التفكير الحاسوبي، وتساعدك هذه المهارات على فحص المشكلة وتحليلها بشكل منهجي وتحديد الحلول المحتملة التي يمكنك تطويرها لتصبح تطبيقات برمجية فاعلة. يمكن فهم مهارات التفكير الحاسوبي على أنها أربع خطوات منفصلة ولكنها مترابطة، كما هو موضح في الشكل 4.1.



الشكل 4.1 مهارات التفكير الحاسوبي

التفكيك

التفكيك هو عملية تقسيم الأفكار المعقدة إلى أجزاء أصغر وأسهل. أحيانًا قد يُطلق على هذه العملية التحليل إلى عوامل. بشكل عام، فإن المشكلات التي لا يتم تفكيكها تكون أكثر صعوبة في الحل. فتقسيم مشكلة كبيرة إلى عدد من المشكلات الأصغر غالبًا ما يحسن فرص النجاح، لأنه يسمح لك بالتركيز على شيء واحد في كل مرة، حتى تتمكن من دراسة تفاصيله عن كثب.

يستخدم الجميع عملية التفكيك كل يوم، غالبًا دون إدراك. على سبيل المثال، تتضمن عملية إعداد وجبة عائلية ما يأتي:

1 اختيار وصفة مناسبة لاتباعها

2 حساب الكميات الصحيحة من مكونات الوصفة وعدد أفراد الأسرة

3 جمع المكونات المناسبة

4 تحضير المكونات

5 طهي المكونات بالترتيب الصحيح

6 طهي المكونات بالطرق الصحيحة

7 طهي المكونات للمدد الصحيحة

8 تجميع الوجبة

9 وضع الوجبة في أطباق جاهزة للأكل.

بهذه الطريقة، يمكن تقسيم مشكلة أو مهمة واحدة (إعداد وجبة عائلية) إلى تسع مهام فرعية على الأقل، ويمكن تفكيك كل منها بشكل أكبر، إذا لزم الأمر، حتى تصبح الخطوات المطلوبة لحل كل مهمة سهلة الفهم نسبيًا. في البرمجة، تتضمن عملية التفكيك المراحل الأربع الآتية:

تحديد ووصف المشكلات والعمليات

في هذه المرحلة، ستقوم بسرد المشكلات والعمليات بشكل موجز، باستخدام لغة تتناسب مع مصدر المشكلة. على سبيل المثال، إذا كنت تتعامل مع مشكلة مالية، يجب عليك استخدام مصطلحات دقيقة خاصة بالقطاع المالي. هذا يعني أنك بحاجة إلى أن تكون على دراية باللغة التقنية المستخدمة في قطاع الأعمال المرتبط بالمشكلة.

تقسيم المشكلات والعمليات إلى خطوات متميزة

في هذه المرحلة، ستقوم بتفكيك المشكلات والعمليات المعقدة إلى خطوات منفصلة يمكن إعادة تجميعها بشكل صحيح عند وضعها معًا. لا يوجد حد محدد لعدد الخطوات المدرجة أو عدد المستويات التي يمكنك تفكيكها. سوف تستمر ببساطة في تفكيك كل خطوة حتى تصل إلى مستوى مقبول من الفهم. على سبيل المثال، يتم تقسيم مشكلة حساب صافي أجر شخص ما (الراتب بعد خصم الضرائب والخصومات الأخرى) إلى عدة خطوات في الشكل 4.2 الوارد في الصفحة الآتية.

وصف المشكلات والعمليات في مجموعة من الخطوات المنظمة

في هذه المرحلة، ستقوم بتوثيق المشكلات والعمليات التي قمت بتفكيكها إلى مجموعة من الخطوات المنظمة. يجب أن يكون هذا واضحًا بما يكفي لتتبعه أنت أو الآخرون.

إيصال أهم سمات المشكلات والعمليات للآخرين

في هذه المرحلة، ستناقش المشكلات والعمليات مع الآخرين. قد يشمل ذلك مبرمجين آخرين أو العميل. بمجرد أن تقوم بتفكيك مشكلة معقدة، يصبح من الممكن البدء في النظر إلى الخطوات المتضمنة لمعرفة ما إذا كانت هناك أي أنماط متكررة.

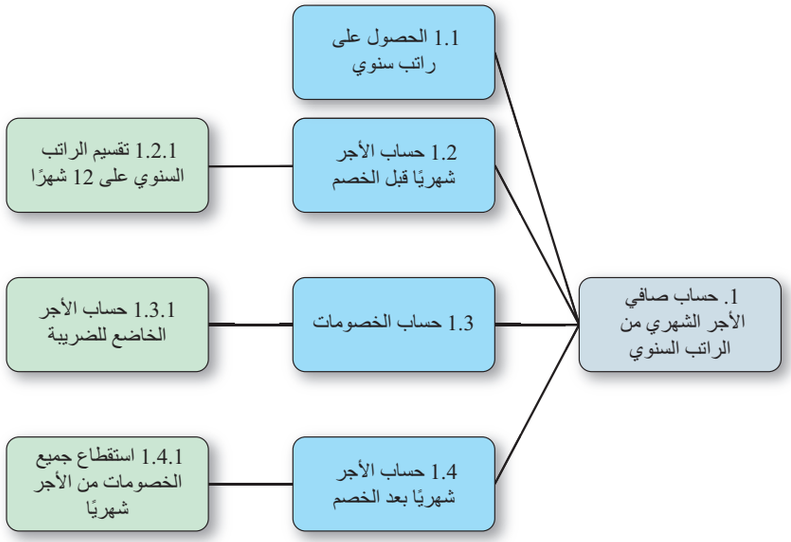
المهارات

- مهارات الإدارة الذاتية والتخطيط
- القدرة على العمل بطريقة قانونية وأخلاقية

فكِّر مليًّا

بصفتك مبرمجًا، يجب أن تكون مهارات الاتصال الخاصة بك مرنة. ستحتاج إلى تعديل التسليم الخاص بك لتلبية لتلبية الاحتياجات المختلفة للمتلقين. على سبيل المثال، سيفهم المبرمجون **المصطلحات** الفنية بينما قد لا يفهم العميل ذلك. من ناحية أخرى، قد يقدر العميل استخدام اللغة الخاصة بقطاع الأعمال، ولكن قد لا يقدر المبرمجون ذلك.

يجب أن تكون قادرًا على توصيل المشكلة للآخرين، لأن وجود فهم واضح للمشكلة أمر ضروري لنجاحك النهائي في حلها.



الشكل 4.2 الرسوم البيانية، بدلًا من النص، هي أفضل طريقة لتوثيق العمليات

تعرُّف الأنماط

تعرُّف الأنماط هو القدرة على رؤية السمات المتكررة داخل المشكلة نفسها وبين المشكلات المختلفة. على سبيل المثال، قد تنطوي مشكلة جديدة على سمات مشابهة لسمات مشكلات أخرى تم مواجهتها وحلها سابقًا. تعرف هذه الأنماط المتكررة يمكن أن يجعل حل المشكلات أسهل بكثير، حيث يمكن أن يوفر نقطة انطلاق جيدة.

تعرُّف الأنماط هو عملية تعتمد على خمس خطوات رئيسة.

1 تحديد العناصر أو السمات المشتركة في المشكلات أو الأنظمة. ويشمل ذلك:
- فحص المشكلات أو الأنظمة
- سرد العناصر أو السمات الموجودة في كل مشكلة أو نظام
- تسليط الضوء على العناصر الموجودة في أماكن متعددة
- تعرُّف تلك العناصر أو الميزات كأنماط.

2 تحديد وتفسير الفروق الشائعة بين العمليات أو المشكلات. ويشمل ذلك:
- فحص المشكلات والعمليات
- سرد العناصر أو الميزات الموجودة في كل مشكلة أو نظام
- تسليط الضوء على ما تنفرد بها كل واحدة
- تعرُّف تلك العناصر أو السمات كاختلافات.

3 تحديد العناصر الفردية داخل المشكلات. ويشمل ذلك:
- دراسة المشكلات لتحديد المدخلات والعمليات (بما في ذلك الاختيارات والتكرارات) والمخرجات الموجودة.

4 وصف الأنماط التي تم تحديدها.

5 عمل تنبؤات بناءً على الأنماط المحددة:
- لكل نمط محدد، حدد كيف يمكن استخدامه في المستقبل أو كيف يظهر في مواقف مشابهة.

المصطلح الرئيس

المصطلحات – الكلمات أو العبارات التي تستخدمها مجموعة معينة فقط والتي يجد الأشخاص خارج تلك المجموعة صعوبة في فهمها.

تعميم النمط وتجريده

يحدث تعميم النمط عندما يمكن تحديد العلاقات بين الأنماط ويمكن استخلاص استنتاجات بسيطة. على سبيل المثال، يمكن تحديد الأنماط حتى عندما لا يكون واضحًا من البداية أن هناك العديد من التشابهات.

في وسائل النقل المختلفة (على سبيل المثال السيارات والحافلات والشاحنات)، يمكننا رؤية عناصر متشابهة تتكرر في أنماط التصميم. تشتمل كل مركبة على أربع عجلات ومحورين وهيكل وآلية توجيه وما إلى ذلك. من أجل حل المشكلات بفعالية، يجب أن تنظر إلى وراء ما هو واضح (في هذه الحالة، الاختلافات المادية بين أشكال النقل الثلاثة). بدلًا من ذلك، حاول تحديد العناصر المشتركة والعلاقات بين هذه العناصر.

هناك جزءان في هذه المرحلة من التفكير الحسابي.

تحديد المعلومات المطلوبة لحل مشكلة محددة: ويُمكنك تحقيق ذلك من خلال معرفة:

- ما المعلومات التي تحتاج إليها
- أسباب احتياجك لهذه المعلومات ("الأساس المنطقي")
- التنسيق الذي يجب تقديم هذه المعلومات به
- مدى سرعة الحاجة إلى هذه المعلومات لمنع تأخير الحل.

ثانيًا، تصفية المعلومات غير المطلوبة لحل مشكلة محددة. يمكنك تحقيق ذلك عن طريق:

- معرفة المعلومات غير المطلوبة، لأنها ستكون مصدر إلهاء
- معرفة (وتبرير) سبب استبعادك لهذه المعلومات.

تمثيل عناصر المشكلة

لتمثيل أجزاء مشكلة أو نظام بشكل عام، تحتاج إلى تحديد:

- **المتغيرات** – تشير إلى القيم التي قد تتغير في المشكلة أو النظام، وعادةً ما يتم إدخالها من قبل المستخدم أو كنتيجة لعملية حسابية مطلوبة
- **الثوابت** – تشير إلى القيم التي لا تتغير كثيرًا أو تظل ثابتة لفترة معقولة في المشكلة أو النظام (على سبيل المثال، معدل الضريبة الأساسي هو 20 في المائة)
- **العمليات الرئيسة** – تشير إلى العمليات التي تعتبر بالغة الأهمية لفهم مشكلة ما أو كيفية عمل النظام
- **عمليات متكررة** – تشير إلى العمليات التي تحدث عدة مرات داخل مشكلة
- **المدخلات** – تشير إلى القيم التي تم إدخالها في النظام، بما في ذلك الوحدات المستخدمة، وربما أي قيم أو نطاقات صالحة (على سبيل المثال، عندما يكون الجنس هو "M" للذكور أو "F" للإناث، أو حيث يجب أن يكون سعر المنزل بين 20,000 دولار و2,000,000 دولار)
- **المخرجات** – تشير إلى المعلومات المقدمة للمستخدم بتنسيق مطلوب، يتم تحديده عادةً من قبل العميل كجزء من متطلباته.

في الحوسبة، التجريد هو مفهوم يتم من خلاله تقسيم الأنظمة إلى طبقات مختلفة، حيث تخفي كل طبقة تعقيد الطبقة الموجودة تحتها. وهذا يسمح للمبرمج باستخدام سمة دون الحاجة إلى معرفة كيفية عملها بالضبط: يتم ببساطة 'تجريد' الآليات غير ذات الصلة والمعقدة أو إزالتها.

استخدامات تطبيقات البرامج

تطبيقات البرمجيات (غالبًا ما تُسمى 'التطبيقات' أو 'البرامج') هي برامج تم تطويرها لتنفيذ مهام محددة، وحل مشاكل معينة وتلبية احتياجات المستخدم المحددة. هناك العديد من الطرق المختلفة لتصنيف التطبيقات: بحسب نوع ترخيص البرنامج (مجاني أو تجاري أو غير ذلك)، بحسب منصة الحاسوب (المكتبي أو المحمول أو غير ذلك) أو بحسب الاستخدام. الجدول 4.1 في الصفحة الآتية يوضح بعض تطبيقات البرمجيات المصنفة بحسب الاستخدام الشائع، بما في ذلك **آثار** استخدامها.

مناقشة

في مجموعات صغيرة أو أزواج، فكر في عملية يومية مثل حساب تكلفة تزيين الغرفة بالطلاء الجديد. حاول تحديد المتغيرات والثوابت والعمليات الرئيسة والمدخلات والمخرجات المتضمنة.

المصطلح الرئيس

الآثار – الآثار المحتملة لشيء ما.

الجدول 4.1 استخدامات وآثار تطبيقات البرامج

فئة الاستخدام	الاحتياجات التي تم الوفاء بها	أمثلة	الآثار
الألعاب	ألعاب الفيديو التي يمكن استخدامها للترفيه أو التعليم أو المساعدة في التعافي بعد الصدمة.	أكتيفيجن كول أوف ديوتي بايووير ماس إيفكت يوبيسوفت أساسنز كريد بوب كاب بيغل مايكروسوفت ماينكرافت	العزلة الاجتماعية المحتملة مشاكل صحية (لها تأثير في ممارسة الرياضة البدنية) الصحة النفسية (كأداة للاسترخاء) إدمان محتمل
الترفيه	التطبيقات التي تساعد المستخدمين على الاسترخاء والاستمتاع بأشكال مختلفة من الوسائط، مثل الموسيقى أو الفيديو أو الكتب من خلال التنزيل أو النشر أو كليهما.	أبل آيتونز ميكروسوفت ميديا بلاير تطبيق أمازون كيندل	العزلة الاجتماعية المحتملة المشاهدة/الاستماع بحسب الحاجة (استهلاك أكثر مرونة)
الإنتاجية	تطبيقات مثل جداول البيانات وقواعد البيانات ومعالجات النصوص وبرامج العروض التقديمية التي تساعد العمال على إنجاز المهام بشكل أكثر كفاءة، عادةً عند العمل في الأدوار الإدارية.	مايكروسوفت أوفيس أباتشي أوبن أوفيس أدوبي® كريتيف سويت جوجل درايف/تطبيقات للعمل	تحسين الإنتاجية مهارات العمل الجديدة أفكار جديدة وتقنيات حل المشكلات كفاءة أكبر تكلفة مخفضة
تخزين المعلومات وإدارتها	التطبيقات المستخدمة لتخزين وإدارة المعلومات بأمان (منع الفقدان) وتمكين الاسترجاع السريع، عادة عبر الإنترنت.	دروب بوكس جوجل درايف آبل آي كلاود	تقليل مخاطر فقدان البيانات تكرار البيانات الوصول المرن إلى البيانات
مهام متكررة أو خطيرة	التطبيقات المستخدمة لأتمتة المعدات في البيئات التي تشكل خطرًا على الناس أو التي تحل محل المهام اليدوية الرتيبة.	صناعة الطاقة الشاحنات ذاتية القيادة (التعدين) تصنيع السيارات المعالجة الكيميائية	تقليل المخاطر الجسدية للفرد المزيد من الرضا الوظيفي (وظائف أقل تكرارًا وملًا) مهارات العمل الجديدة تحسين الإنتاجية تكلفة مخفضة
وسائل التواصل الاجتماعي	تطبيقات مصممة لتوصيلك بأشخاص آخرين، للمساعدة في التواصل ومشاركة الأفكار والأحداث والصور ومقاطع الفيديو.	X (المعروفة سابقًا باسم تويتر) فيسبوك سناب شات الواتساب بنتريست. إنستغرام ووردبريس بلوجر	التغييرات في مهارات التواصل القدرة على مناقشة المشكلات ومشاركتها مع الآخرين القدرة على تبادل الأفكار مع الآخرين المخاطر الأمنية المخاطر التي يتعرض لها القصر والبالغون الضعفاء إدمان محتمل تحسين المساهمة في القضايا الهامة

المصطلح الرئيس

النظام الثنائي – نظام أرقام يستخدم فقط الرقمين 0 و1 لتكوين الأرقام (المعروف أيضًا باسم "الأساس 2"). على سبيل المثال، "5" في النظام الثنائي هي 101 (1 × 4 + 0 × 2 + 1 × 1). تحتوي دوائر الحاسوب على حالات "تشغيل" و"إيقاف" يمكن استخدامها لتمثيل قيم 0 و1 الثنائية.

تطبيق النظرية

حدد عشرة تطبيقات برمجية ربما قمت بتثبيتها على الحاسوب المنزلي أو الجهاز اللوحي أو الهاتف الذكي.

- ما فئة الاستخدام التي تنتمي إليها هذه التطبيقات؟
- ما المشكلات التي يحلونها؟
- ما الآثار المترتبة على استخدام هذا النوع من البرامج؟

ميزات وخصائص لغات البرمجة

تم تطوير المئات من لغات البرمجة المختلفة منذ منتصف القرن العشرين. تمت كتابة البرامج الأصلية بلغة الآلة (**الثنائية**) التي تخبر معالج الحاسوب بما يجب القيام به بالضبط، ولكن ثبت أن هذا يستغرق وقتًا طويلًا ويتطلب الكثير من المهارات. على الرغم من السرعة والكفاءة في التنفيذ، إلا أن مثل هذه البرامج استغرقت وقتًا طويلًا نسبيًا لإنشائها وتم اعتبار التطبيقات المعقدة مهمة كبيرة.

مع مرور الوقت، جعلت **اللغات الجديدة ذات المستوى المنخفض والمستوى المرتفع** البرمجة عملية أكثر قابلية للفهم. ونتيجة لذلك، قللت وقت إنتاج التطبيقات وبرامج الأنظمة الأكثر تعقيدًا. يقارن الجدول 4.2 بين نوعين مختلفين من كود البرنامج المستخدم لإخراج شاشة الرسالة نفسها، في البداية باستخدام لغة الآلة منخفضة المستوى ثم باستخدام اللغة عالية المستوى، C++.

الجدول 4.2 إخراج 'طالب BTEC' باستخدام كود البرنامج منخفض وعالي المستوى

لغة منخفضة المستوى	لغة عالية المستوى

طالب BTEC في كود الآلة Intel x86 (موضح بالنظام الست عشري):
`B4 09`
`BA 09 01`
`CD 21`
`CD 20`
`42 54 45 43 20 73 74 75 64 65`
`6E 74 24`
	"طالب BTEC" في كود مصدر C++:
	`cout << "BTEC student";`

بينما يمكننا على الأرجح قراءة كود C++ بشكل أكثر راحة، يمكن للحاسوب معالجة المكافئ منخفض المستوى بسهولة أكبر بكثير حيث أن تعليمات كود الآلة تتحدث مباشرة إلى **وحدة المعالجة المركزية (CPU)**. وبالمقارنة، يجب ترجمة كود C++ بنجاح إلى لغة الآلة باستخدام **برنامج التحويل** قبل أن يتم تنفيذه بواسطة وحدة المعالجة المركزية.

استخدامات وتطبيقات اللغات عالية ومنخفضة المستوى

بنية الحاسوب معقدة إلى حد كبير. اللغات البرمجية منخفضة المستوى مثل لغة الآلة (التي غالبًا ما تُكتب بالنظام الثنائي، وهو نظام أساسه 2، أو النظام السداسي عشري، وهو نظام أساسه 16) أو لغة التجميع، تخفي القليل جدًا من هذه التعقيدات عن المبرمج. لكي يتمكن المبرمج من برمجة الحاسوب على هذا المستوى، يجب أن يكون على دراية جيدة بهيكلية المعالج.

وبالمقارنة، تستخدم اللغات عالية المستوى مثل .NET Microsoft Visual Basic وC++ التجريد لإخفاء تعقيدات البنية من المبرمج. في هذه اللغات، قد يُترجم أمر واحد إلى مئات التعليمات المعقدة ذات المستوى المنخفض التي يمكن للمعالج فهمها.

غالبية لغات البرمجة التجارية المستخدمة في العالم اليوم عالية المستوى. هذا لأن اللغات عالية المستوى تعمل على:

1. تحسين إنتاجية المبرمجين عند كتابة كود البرنامج
2. تحسين قابلية قراءة الكود
3. إنتاج كود يسهل **تصحيحه**
4. السماح باستخدام أدوات تطوير البرامج الأكثر مرونة
5. إنتاج كود يمكن **نقله**.

المهارات

- مهارات التحليل واتخاذ القرار
- مهارات الإدارة الذاتية والتخطيط
- القدرة على العمل بطريقة قانونية وأخلاقية

المصطلحات الرئيسة

اللغات ذات المستوى المنخفض والمستوى المرتفع – في البرمجة، يشير المصطلحان "منخفض" و"مرتفع" إلى موضع اللغة في إطار فهمها بواسطة الحاسوب (على سبيل المثال، ثنائية المستوى منخفضة المستوى) وفهمها من قبل شخص (على سبيل المثال، اللغات الطبيعية مثل الإنجليزية والعربية والتايلاندية والهولندية عالية المستوى).

وحدة المعالجة المركزية (CPU) – "الدماغ" المركزي للحاسوب. عادةً ما يتحكم في موارد الحاسوب ومدخلاته ومخرجاته، والأهم من ذلك، تعليمات المعالجة والبيانات التي يتم جلبها من ذاكرة الوصول العشوائي (RAM).

برنامج التحويل – برنامج خاص يترجم كود البرنامج المكتوب بلغة عالية المستوى إلى تعليمات ثنائية يمكن لوحدة المعالجة المركزية معالجتها.

تصحيح الأخطاء – عملية تحديد خطأ (أو خلل) في كود البرنامج وإزالته.

منقول – مكتوب باستخدام بنية حاسوبية واحدة، ولكن تم تجميعه للاستخدام على جهاز آخر. يُعرف هذا أيضًا باسم التجميع المتقاطع.

بحث

يُنسب مصطلح "الخطأ" بشكل شائع إلى رائدة الحاسوب غريس هوبر التي عثرت في عام 1945 على فراشة أصبحت محاصرة بين نقاط الاتصال في مفتاح الترحيل بالحاسوب. بمجرد إزالة "الفراشة"، عمل الحاسوب مرة أخرى. لكن في الواقع، هذا المصطلح أقدم من هذا. قم ببعض الأبحاث لمعرفة ما إذا كان بإمكانك تحديد أصلها.

توفر اللغات ذات المستوى المنخفض التحكم النهائي في أجهزة الحاسوب. ومع ذلك، فهي تستغرق وقتًا طويلًا كما أنها معقدة الاستخدام. في المقابل، فإن اللغات عالية المستوى تلخص صعوبات التحدث إلى أجهزة الحاسوب مباشرة وتوفر فرصًا أكثر سرعة لتطوير البرامج. ومع ذلك، غالبًا ما يكون البرنامج النهائي أقل كفاءة وأبطأ سرعة وأكبر حجمًا مما قد يكون عليه إذا تم إنتاجه بلغة منخفضة المستوى.

نماذج البرمجة

أدت أنواع مختلفة من المشكلات إلى ظهور أنماط برمجة مختلفة. يهدف كل نمط، يُعرف بالنموذج، إلى حل مشكلة بطريقة مختلفة، غالبًا لتلبية احتياجات المستخدم المختلفة. يوضح الجدول 4.3 الأنواع الأكثر شيوعًا لنماذج البرمجة.

يمكن أن تنتمي بعض لغات البرمجة إلى نماذج متعددة. على سبيل المثال، يشيع استخدام "لغة روبي" بشكل كلغة برمجة نصية، ولكنها تحتوي أيضًا على العديد من الميزات كائنية التوجه (OO). تعتمد لغة مايكروسوف فيجوال بيسك دوت نت، والتي يصنفها الكثيرون على أنها لغة تعتمد على الأحداث، بشكل كبير أيضًا على الفئات (مثل الأزرار والنماذج ومربعات الحوار) التي تمثل جوهر نظام تشغيل Microsoft Windows. وبالتالي، يمكن اعتبارها أيضًا لغة موجهة للكائنات.

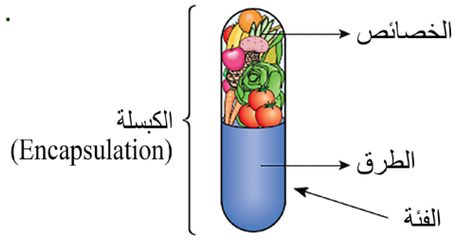
الجدول 4.3 نماذج لغة البرمجة المختلفة

نماذج البرمجة	لغات البرمجة	الميزات والخصائص
البرمجة الإجرائية	بيسك، سي، فورتران	غالبًا ما تكون اللغات الإجرائية هي الأولى التي يتعلمها المبرمج. وغالبًا ما تُعتبر أداة للأغراض العامة وتستخدم لإنشاء العديد من أنواع التطبيقات المختلفة. تتم كتابتها عادةً في مجموعة من الخطوات المحددة جيدًا والتي تحل مشكلة محددة، على سبيل المثال إجراء عملية حسابية بسيطة في لغة سي، كما هو موضح في الشكل 4.3. إذا أصبحت الخطوات معقدة أو طويلة أو متكررة، فقد يختار المبرمج تقسيم الخطوات إلى إجراءات منفصلة، ولكل منها غرض واحد يمكن استخدامه عدة مرات. يمكن أيضًا استخدام مصطلح "ضروري" لوصف هذا النوع من اللغة، لكن اللغات الحتمية تميل إلى الاعتماد بشكل أقل على الإجراءات.

```c
#include <stdio.h>
#include <string.h>

int main ()
{
  int a;
  int b;
  int c;

  puts ("Enter first number");
  scanf("%d", &a);
  puts ("Enter second number");
  scanf("%d", &b);

  c = a + b;
  printf("%d + %d = %d",a,b,c);

  return 0;
 }
```

الشكل 4.3 كود برنامج لغة سي يعرض مجموع رقمين تم إدخالهما

الجدول 4.3 *متابعة*

نماذج البرمجة	لغات البرمجة	الميزات والخصائص
كائنية التوجه (OO)	• C++ • Microsoft C# • Oracle Java™ • PHP • Ada • Python • Ruby	البرمجة كائنية التوجه هي نهج حديث شائع للبرمجة. تعتمد اللغات كائنية التوجه على مفاهيم "الفئات" و"الكائنات" لحل مشاكل العالم الحقيقي. ونظرًا لهذا النهج، يتم استخدامها بشكل شائع لتصميم ألعاب الفيديو ومواقع التجارة الإلكترونية وأنظمة قواعد البيانات وواجهات المستخدم. في اللغات كائنية التوجه ، يتم إنشاء الكائنات من فئات يتم تصميمها عادةً على غرار "أشياء" في العالم الحقيقي مثل العملاء والحسابات المصرفية والمنتجات والطلبات وما إلى ذلك. تعمل كل فئة كخريطة برمجية، حيث تقوم بتغليف (أو احتواء) حالة الشيء (بياناته أو خصائصه) وسلوكياته (وظائفه أو طرقه) في كود البرنامج راجع (الشكل 4.4). يقوم المبرمج بإنشاء تفاعلات محددة بين الكائنات المختلفة من أجل حل المشكلة. نظرًا لوجود كل فئة على حدة، يمكن تعديلها أو تكييفها بسهولة لتعكس التغييرات التي تحدث في العالم الحقيقي دون أن يكون لها تأثير سلبي في الحل بأكمله. وهذا يجعل اللغات كائنية التوجه جذابة للغاية للمطورين عندما يفكرون في متطلبات معالجة الصيانة المستمرة.  **الشكل 4.4** الكبسلة
القائمة على الأحداث Event-driven (ED)	• مايكروسوفت فيجوال بيسك .نت، بايثون، روبي، جافا سكريبت	هذا نموذج شائع لتطوير التطبيقات الرسومية التي تستجيب للأحداث التي تم إنشاؤها إما عن طريق النظام (مثل ساعة النظام) وإما عن طريق المستخدم (مثل النقر بالفأرة). عادةً ما تعمل البرامج التي تعتمد على الأحداث بشكل غير متسلسل مع المستخدمين القادرين على تحديد العمليات التي يريدون تنفيذها بدلًا من اتباع المدخلات المحددة مسبقًا لبنية برنامج أكثر صرامة. يركز المطورون عادةً على برمجة معالجات الأحداث، وهي الشفرة التي تحدد الإجراءات التي يجب تنفيذها عند تشغيل حدث معين عبر مراقب. المراقب هو عملية تنتظر حدوث حدث معين. على سبيل المثال، إذا تم النقر فوق خيار قائمة ملف->فتح، فسيتم عرض مربع حوار فتح ملف.

الآلة	• لغة الآلة
	• لغة التجميع (يشار إليها أحيانًا بشكل غير رسمي باسم "المجمع")
		هذه هي لغات البرمجة ذات المستوى الأدنى والتي توفر التحكم في الأجهزة الأساسية للآلة. ويجب كتابة كود الآلة لعائلة وحدة معالجة مركزية معينة، مثل Intel x86 (32 بت) أو x64 (64 بت). لا يمكن تشغيله بسهولة على منصات مختلفة دون التحويل إلى تعليمات وحدة المعالجة المركزية للنظام الأساسي الجديد أو محاكاة تعليمات وحدة المعالجة المركزية الأصلية.
		وتستخدم لغة التجميع سلسلة من أساليب الاستذكار الملائمة للأشخاص لتمثيل تعليمات وحدة المعالجة المركزية الأساسية. أما الوسائل المساعدة للذاكرة، فهي وسائل تساعد الناس على تذكر المفاهيم المعقدة، عادة من خلال التمثيل البصري أو الأقوال أو القوافي التي يسهل تذكرها. في لغة التجميع، تعمل الوسائل المساعدة للذاكرة على تحسين قابلية القراءة وزيادة الإنتاجية. على سبيل المثال، عند إضافة 4 + 2 في لغة التجميع:
		• **mov** يعني نقل، **add** يعني **إضافة** و**int** يعني مقاطعة (int 20 ينهي البرنامج)
		• **ax** و**bx** عبارة عن سجلات (مناطق ذاكرة عالية السرعة) داخل وحدة المعالجة المركزية.
		`mov ax, 0004`
		`mov bx, 0002`
		`add ax, bx`
		`int 20`
		تأتي الوسائل المساعدة للذاكرة هذه في شكل **أكواد تشغيل ومعاملات**. تصف أكواد التشغيل العملية التي يتم تنفيذها والمعاملات هي القيم التي تتم معالجتها بواسطة العملية.
		في لغة الآلة، يبدو هذا الكود نفسه معبرًا عنه بالنظام الست عشري هكذا:
		`B8 04 00`
		`BB 02 00`
		`01 D8`
		`CD 20`

الجدول 4.3 *متابعة*

نماذج البرمجة	لغات البرمجة	الميزات والخصائص
		من أجل التنفيذ، يجب ترجمة لغة التجميع إلى لغة الآلة باستخدام برنامج خاص يسمى المترجم. غالبًا ما تُستخدم لغات الآلة عندما تكون السرعة أمرًا حيويًا أو عندما يكون الوصول منخفض المستوى إلى أجهزة الحاسوب أو الاتصال بالإلكترونيات المتصلة أمرًا مهمًا. على سبيل المثال، يمكن استخدامه في آلات البيع وشاشات العرض الأمامية وألعاب الفيديو.

لغات الترميز	• HTML (لغة ترميز النص التشعبي)
	• XML (لغة الترميز القابلة للتوسعة)
		الترميز هو شكل من أشكال اللغة المستخدمة لتحديد تنسيق محتوى المستند بطريقة منظمة باستخدام علامات خاصة. على سبيل المثال، يتم استخدام <P> في HTML للإشارة إلى بداية ونهاية الفقرة، كما هو موضح في الشكل 4.5.
		`<p>A new paragraph</p>`
		الشكل 4.5
		• يمكن استخدام لغات الترميز الأخرى لتمثيل هياكل البيانات المعقدة بطريقة محايدة للمنصة. على سبيل المثال، إدخالات من مكتبة رموز وجهات المطارات الدولية، كما هو موضح في الشكل 4.6.
		• غالبًا ما يتم استخدام XML كتنسيق مفضل لنقل البيانات بين أنظمة الحاسوب والتطبيقات المختلفة، مثل عند تصدير واستيراد البيانات بين أنظمة قواعد البيانات العلائقية التي عادة ما تكون غير متوافقة.
		`<AIRPORT>`
		`<COUNTRY>Thailand</COUNTRY>`
		`<CAPITAL>Bangkok</CAPITAL>`
		`<CODE>BKK</CODE>`
		`</AIRPORT>`
		`<AIRPORT>`
		`<COUNTRY>Pakistan</COUNTRY>`
		`<CAPITAL>Islamabad</CAPITAL>`
		`<CODE>ISB</CODE>`
		`</AIRPORT>`
		الشكل 4.6 يُستخدَم الترميز لإظهار بنية البيانات

البرمجة النصية	• Perl:
	• جافا سكريبت
	• Ruby:
	• PHP
		يتم استخدام أنواع مختلفة من لغات البرمجة لأغراض مختلفة.
		يتم استخدام JavaScript في تصميم الويب لأتمتة العمليات وإضافة ميزات تفاعلية إلى صفحات الويب.
		PHP هي لغة برمجة نصية شائعة على جانب الخادم تستخدم لإنشاء تطبيقات معقدة عبر الإنترنت تستخدم في التجارة الإلكترونية.
		غالبًا ما يتم استخدام Perl وRuby لأتمتة عمليات النظام على الحاسوب عن طريق ربط وتنفيذ المهام التي ربما تم تشغيلها في الأصل بشكل منفصل من قبل المستخدم. على سبيل المثال، يعرض الشكل 4.7 برنامج Ruby لعرض أول 255 بايت من ملف محدد على الشاشة.
		`#!/usr/bin/ruby`
		`filename = ARGV[0]`
		`aFile = File.new(filename,"r")`
		`if aFile`
		`content = aFile.sysread(255)`
		`puts content`
		`else`
		`puts "Unable to open file!"`
		`end`
		الشكل 4.7 نموذج برنامج روبي

مقارنة وتباين لغات البرمجة

هناك العديد من العوامل التي يجب مقارنتها وموازنتها عند النظر في لغات البرمجة المختلفة. وتشمل هذه المتطلبات (مثل الأجهزة والبرامج والأجهزة الخاصة) والأداء وسهولة التطوير.

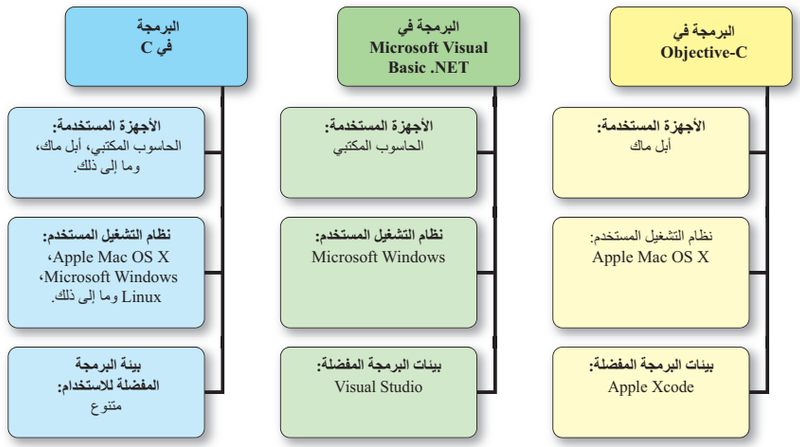
الأجهزة والبرامج اللازمة لتشغيل البرنامج وتطويره

تتطلب بعض لغات البرمجة أجهزة وبرامج محددة لتشغيل البرنامج وتطويره. المتطلبات المفضلة للغات البرمجة الشهيرة موضحة في الشكل 4.8.

تعتبر بعض لغات البرمجة، مثل C، "متعددة المنصات". هذا يعني أنها مدعومة على العديد من المجموعات المختلفة من الأجهزة وأنظمة التشغيل. من المحتمل أن يفسر هذا سبب استمرار شعبية لغة C بين المبرمجين التجاريين، على الرغم من حقيقة أنها صدرت لأول مرة في عام 1972.

مناقشة

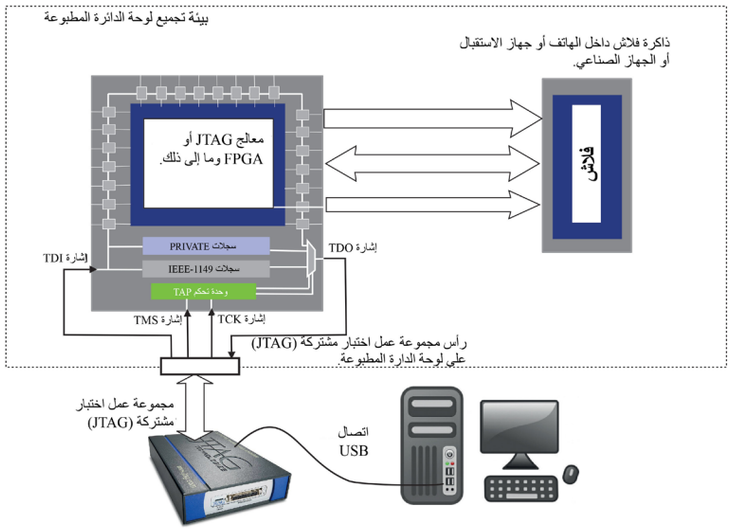
في مجموعات صغيرة أو أزواج، ناقش لماذا تعتقد أنك قد تحتاج إلى مقارنة لغات البرمجة ولماذا من المهم أن يكون لديك مجموعة متنوعة من اللغات المختلفة. اكتب الأسباب التي يمكنك التفكير فيها. تابع القراءة الآن وشاهد عدد الأسباب التي حددتها بشكل صحيح.



الشكل 4.8 متطلبات الأجهزة والبرامج المتباينة للغات البرمجة المحددة

الأجهزة الخاصة المطلوبة

عند برمجة حلول للأجهزة الخارجية، من الشائع جدًا توصيل حاسوب مكتبي بجهاز خاص، عادةً واجهة البرمجة التسلسلية (SPI) أو مجموعة عمل الاختبار المشتركة (JTAG). عادة ما يتم توصيل هذا الجهاز الخاص بجهاز حاسوب مكتبي عبر ناقل تسلسلي عالمي (USB) أو كبل تسلسلي أو متوازي. يسمح للمطور بإجراء عملية تسمى البرمجة داخل النظام (ISP). تسمح البرمجة داخل النظام (ISP) بإعادة برمجة الأجهزة دون إزالتها من دائرتها الأصلية. على سبيل المثال، الاستخدام الشائع للبرمجة داخل النظام (ISP) هو إعادة برمجة الهاتف المحمول لإصلاحه أو فتح ميزاته، كما هو موضح في الشكل 4.9 في الصفحة الآتية.



الشكل 4.9 مجموعة عمل اختبار مشتركة (JTAG) تربط بين هاتف محمول وجهاز حاسوب

الأداء

تتباين لغات البرمجة في كفاءة أدائها. عادةً ما يتم قياس الأداء بناءً على قدرة اللغة على تنفيذ الخوارزميات المعقدة مقارنةً بالوقت. على سبيل المثال، تستفيد بعض لغات البرمجة بشكل أفضل من وحدة المعالجة المركزية من خلال وجود مترجمين أكثر كفاءة يقومون بإنشاء كود آلة محسّن.

تقوم بعض لغات البرمجة بإدارة ذاكرة الوصول العشوائي بشكل أكثر كفاءة من غيرها من خلال اتخاذ تدابير صارمة **لجمع البيانات المهملة**. جمع البيانات المهملة ليس عملية إلزامية وبعض اللغات، مثل C، تتطلب من المبرمج أن يتذكر إلغاء تخصيص الذاكرة التي لم تعد قيد الاستخدام يدويًا. قد يؤدي جمع البيانات المهملة بشكل مكثف إلى تسريع اللغة ولكنه ينطوي على خطر حدوث "تسرب" للذاكرة.

مجالات التطبيق المفضلة

بعض لغات البرمجة مناسبة بشكل أفضل لبعض مجالات التطبيق.

تعتبر لغة C لغة للأغراض العامة، ولكنها فعالة بشكل خاص للتحكم في الأجهزة والإلكترونيات الخارجية بسبب التحكم المنخفض المستوى في نظام الحاسوب.

يتم استخدام Java لإنشاء تطبيقات لصفحات الويب وتطبيقات Android للأجهزة المحمولة. يعني نهج "الكتابة مرة واحدة والتشغيل في أي مكان" (WORA) أنه مدمج في العديد من الأشكال المختلفة لأجهزة الترفيه المنزلي، مثل أجهزة استقبال التلفزيون ومشغلات بلو راي.

تُستخدم لغة PHP لإنشاء تطبيقات خادم للأنشطة التجارية في مجال التجارة الإلكترونية، والتجارة عبر الإنترنت، وغيرها.

تُستخدم لغة C# لإنشاء ألعاب الفيديو لنظام Microsoft Windows وأجهزة الألعاب الشهيرة، وذلك بفضل سهولة استخدامها مع مجموعة أدوات Microsoft XNA، التي تُعتبر مجموعة شائعة من الأدوات المستخدمة في تطوير وإدارة ألعاب الفيديو.

يتم استخدام Objective-C لإنشاء تطبيقات الجوال لأجهزة نظام التشغيل iOS الخاص بأجهزة أبل، مثل iPad وiPhone وما إلى ذلك.

وقت التطوير

يعتبر وقت التطوير اعتبارًا مهمًا عند اختيار نموذج البرمجة. على سبيل المثال، كما رأينا، البرامج المكتوبة باستخدام لغة الآلة ولغة التجميع تكون فعّالة وتُنفَّذ بسرعة كبيرة، ولكن وقت التطوير عند

المصطلح الرئيس

جمع البيانات المهملة – عملية تلقائية تحاول استعادة ذاكرة الوصول العشوائي المحجوزة بواسطة برنامج لتخزين البيانات، على سبيل المثال معرف (متغير أو كائن) لم تعد هناك حاجة إليه.

استخدام هذه اللغات قد يكون أطول كثيرًا مقارنة بالعمل مع اللغات عالية المستوى في بيئات التطوير المتكاملة الحديثة.

في البرمجة التجارية، يتم ربط الوقت مباشرة بالمال، لذلك هناك تركيز قوي على كتابة كود موثوق مع الحد الأدنى من الأخطاء في أسرع وقت ممكن. ونتيجة لذلك، فإن لغات البرمجة التي تدعمها أدوات التطوير الغنية بالميزات التي تدعي تقليل وقت التطوير شائعة.

ومع ذلك، قد يكون من الصعب قياس وقت التطوير. هذا لأنه من الصعب تحديد **المقاييس** المقبولة والاتفاق عليها داخل المجال. على سبيل المثال، يمكنك اعتبار عدد أسطر الكود (LOC) المكتوبة في الساعة مقياسًا مقبولًا، ولكن هذا قد لا يأخذ في الاعتبار عدد الأخطاء ذات الصلة التي يتم إنشاؤها. بالإضافة إلى ذلك، إذا علم المبرمج أن إنتاجيته تقاس بعدد أسطر الكود (LOC) الذي ينشئه في الساعة، فقد يشجعه ذلك على كتابة كود **مطول** بشكل مفرط.

المصطلحات الرئيسة

المقياس – شكل متفق عليه للقياس يتيح المقارنة والتقييم.
مطول – استخدام كلمات أو أكواد أكثر من اللازم.

سهولة التطوير

كما ذكرنا سابقًا، فإن بعض لغات البرمجة أسهل في الاستخدام من غيرها، لأن بعض اللغات تقدم لمطور البرمجيات أدوات متقدمة يمكن أن تقدم تلميحات أو كود برنامج يتم إنشاؤه تلقائيًا. على سبيل المثال، يقوم Microsoft Visual Basic .NET بتحويل نماذج الإدخال التي تم إنشاؤها باستخدام وظيفة السحب والإفلات إلى كود برنامج تلقائيًا.

يمكن أن تؤثر جودة الأدوات الأخرى المتاحة داخل محرر لغة البرمجة أيضًا على سهولة التطوير وإنتاجية المبرمج. تشمل هذه الأدوات أنظمة المساعدة، وتمييز بناء الجملة وأدوات تصحيح الأخطاء.

- تسليط الضوء على بناء الجملة هو ميزة في العديد من أدوات تحرير لغات البرمجة. حيث يعرض هياكل برمجية مختلفة بألوان محددة. على سبيل المثال، قد يظهر التعليق باللون الأخضر، وسلسلة النص باللون الأحمر وهكذا. يجد معظم المطورين أن الكود الناتج أسهل في القراءة والفهم وتصحيح الأخطاء.
- توفر أدوات تصحيح الأخطاء للمبرمج مجموعة من الميزات التي تساعدهم في تحديد الأخطاء في الكود وإزالتها. تشمل أدوات تصحيح الأخطاء الشائعة التتبع والمراقبة ونقاط التوقف.

فكِّر مليًّا

بغض النظر عن لغة البرمجة أو النظام الأساسي أو أدوات التطوير التي تستخدمها، ستحتاج إلى تطوير وإظهار السلوك المهني المناسب لصناعة البرمجة. وتشمل هذه السلوكيات:

- أن تكون منضبطًا - يجب عليك تحديد أهداف التنمية ذات الصلة والعمل وفقًا لجداول زمنية واقعية
- احترام الآخرين وآرائهم، خاصة عندما يقدمون لك ملاحظات أساسية حول برنامجك ومدى ملاءمته عند تقييمه مقابل احتياجات المستخدم الأصلية
- تقييم النتائج لمساعدتك على تبرير توصياتك وقراراتك، خاصة في ما يتعلق بتصميم الحل الخاص بك أو تنفيذه
- مقارنة الأهداف المحددة ونتائجها النهائية ومقارنتها، حيث سيساعدك ذلك على فهم وتحسين أدائك كمطور ومحلل للمشكلات
- العمل مع الآخرين كجزء من فريق - هذا مهم بشكل خاص عند المساهمة في مهمة أكبر لحل المشكلات تشمل الآخرين. يجب عليك تقديم الدعم لزملائك ومساعدتهم على إدارة عبء العمل المشترك والاستجابة بشكل إيجابي وبنّاء للاعتراضات والنزاعات والمساعدة في إدارة توقعات عميلك.

بحث

يمكنك البحث واستكشاف لغات برمجة مختلفة عبر الإنترنت من خلال استخدام بيئات التطوير الافتراضية. تتيح لك هذه البيئات تحديد لغة معينة وإدخال بعض الأكواد الأساسية للبرنامج ثم تنفيذها.

البُنى والتقنيات وتنفيذها بلغات مختلفة

بناءات لغات البرمجة هي اللبنات الأساسية المستخدمة لإنشاء برنامج. ستستخدمها بطرق مختلفة وتطبق تقنيات متنوعة لحل المشكلات التي تُعطى لك. شيء واحد ستكتشفه في وقت مبكر جدًا هو أن العديد من البنى والتقنيات شائعة عبر عدد من اللغات، وهو ما يعني إمكانية الاستفادة من معرفتك بلغة في استخدام لغات أخرى.

تم استخدام عينات من الأكواد من عدد من لغات البرمجة الشهيرة المختلفة لمساعدتك في مقارنة وتباين بنياتها وتقنياتها. ومع ذلك، تم كتابة عينات الأكواد الأكبر في هذه الوحدة بلغة Microsoft C#، وهي لغة شائعة جدًا في مجال تطوير البرمجيات. مع تقدم مهاراتك، ستجد أنه من السهل نسبيًا تحويل هذه العينات البرمجية إلى لغة هدف أخرى.

كلمات الأوامر

مفهوم الكلمات المحجوزة (أو الأوامر) هو في صميم معظم لغات البرمجة. لا يمكن للمبرمج استخدام هذه الكلمات المحجوزة لتسمية الأشياء (انظر المعرفات). بدلًا من ذلك، يتم استخدامها لأمر إجراء معين، مثل 'فتح ملف'، 'مسح الشاشة' أو 'وضع نص على الشاشة'. يمكن أن تختلف كمية الكلمات المحجوزة والغرض منها اختلافًا كبيرًا بين لغات البرمجة المختلفة، ولكن عندما تصبح أكثر خبرة كمطور، ستبدأ في تحديد المفاهيم المتكررة.

المعرفات

المعرف هو اسم يسهل على المبرمجين استخدامه ويمثل كمية مخزنة في ذاكرة الوصول العشوائي للحاسوب. تُستخدم أنواع عديدة من المعرفات في البرامج، ولكن بعض الأنواع الأكثر شيوعًا هي:

- ثابت - هذا معرف يمثل قيمة **لن** تتغير في أثناء تشغيل البرنامج
- متغير - هذا معرف يمثل قيمة **قد** تتغير في أثناء تشغيل البرنامج.

على سبيل المثال، عندما يقوم شخص بتسجيل الدخول إلى برنامج، يتم عادةً تخزين اسمه وكلمة المرور كمتغيرات، حيث ستكون هذه التفاصيل مختلفة لكل مستخدم. ومع ذلك، يجب على البرنامج الذي يحسب سعر تلفزيون جديد في ماكينة تسجيل النقد الإلكترونية في المتجر أن يضمن إضافة معدل أي ضريبة مبيعات حالية إلى السعر. تختلف ضرائب المبيعات في البلدان حول العالم، ولكن الطريقة التي تُدار بها في البرنامج هي نفسها. لن تتغير هذه القيمة مع كل عملية بيع، لذا يمكن تعيينها كقيمة ثابتة.

من أجل إنشاء (أو إعلان) ثابت أو متغير، يجب عليك تزويد لغة البرمجة بشيئين:

- اسم
- نوع بيانات.

يجب عليك دائمًا اختيار اسم معقول وهادف. على سبيل المثال، سيكون اسم المتغير الجيد لتخزين اسم المستخدم هو 'username'. لا يمكن استخدام بعض الأسماء لأنها محجوزة من قبل اللغة لاستخدام معين، عادة لأنها كلمة أمر. تختلف هذه الكلمات المحجوزة بين لغات البرمجة المختلفة، لذا كن حذرًا.

موضوعات ذات صلة

يتم عرض قائمة الكلمات المحجوزة لـ Java في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

هناك العديد من اصطلاحات التسمية المختلفة المستخدمة في البرمجة. CamelCase أو snake_case هما ربما الأكثر شيوعًا في البرمجة الحديثة، على سبيل المثال إذا أردنا تخزين عمر المستخدم بالسنوات (كعدد صحيح أو عدد كامل) في C#:

الجمل المتصلة		
snake_case	int user_age;	كل الكلمات بحروف صغيرة. يتم ربط الكلمات برمز تسطير سفلي.
	int userAge;	الكلمة الأولى بحروف صغيرة. يتم كتابة الرمز الأول من كل كلمة تالية برمز حرف إنجليزي كبير.

تحتوي معظم لغات البرمجة أيضًا على مفهوم المتغيرات المحلية والعالمية. أسهل طريقة للتفكير في هذا هي تذكر أن المتغيرات العالمية يمكن استخدامها في أي مكان في كود البرنامج الخاص بك. المتغيرات المحلية تقتصر على الاستخدام في كتلة الكود (مثل الدالة) التي تم التصريح بها فيها.

أنواع البيانات

تدعم جميع لغات البرمجة تقريبًا مفهوم أنواع البيانات. يتم استخدام نوع البيانات لتحديد نوع القيمة التي يمكن للمتغير أو الثابت تخزينها، والعمليات التي يمكن إجراؤها عليها وسلوكها داخل البرنامج. تقدم معظم لغات البرمجة العديد من أنواع البيانات المختلفة للمبرمج لاستخدامها.

الأنواع الأكثر شيوعًا من البيانات المستخدمة هي:

- رمز – هذا يخزن رمزًا واحدًا (الرمز هو رمز أبجدي أو رقم أو مساحة فارغة أو علامة ترقيم)
- سلسلة – يخزن مجموعة من الرموز
- عدد صحيح – يخزن رقمًا كاملًا (أي رقم دون منازل عشرية)
- حقيقي (يُطلق عليه أحيانًا النقطة العائمة) – يخزن هذا رقمًا يحتوي على منازل عشرية
- **منطقية (Boolean)** – هذه قيمة منطقية تخزن إما true (1) وإما false (0)

نظرًا لأن هذه الأنواع من البيانات شائعة جدًا، فمن الممكن مقارنة تنفيذها في لغات البرمجة المختلفة، كما هو موضح في الجدول 4.4 في الصفحة الآتية.

توصف بعض اللغات بأنها ذات كتابة قوية وأخرى بأنها ذات كتابة ضعيفة (أو فضفاضة). هذا يُحدث فرقًا جوهريًا في رسمية لغة البرمجة. تتطلب لغات البرمجة ذات النوع القوي أن تكون البيانات وأنواع البيانات متسقة عند استخدامها.

على سبيل المثال، في جافا (Java)، إذا تم إعلان متغير كعدد صحيح، فيجب استخدامه فقط لتخزين المتغيرات الصحيحة، وإلا قد تحدث أخطاء. هذا لأن جافا لغة ذات نوعية قوية. ومع ذلك، فإن لغة C غير صارمة في تحديد الأنواع: حيث ستقوم بتحويل البيانات تلقائيًا إلى النوع الصحيح الذي يتوقعه المتغير دون الحاجة إلى أن يضيف المبرمج أي كود محدد للقيام بذلك. هذه ميزة مناسبة، لكنها يمكن أن تؤدي إلى نتائج غير متوقعة.

الجمل

الجمل هي الجانب الأساسي للعديد من لغات البرمجة. إنها تحدد الإجراءات الأساسية التي يمكن تنفيذها وغالبًا ما يجمعون بين عدد من تراكيب اللغة.

تظهر أمثلة شائعة لأنواع الجمل الأساسية مع مكافئات لغوية مباشرة في الجدول 4.5 في الصفحة الآتية. هناك العديد من أوجه التشابه بينهما. على سبيل المثال، كلها **حساسة لحالة الأحرف**، ولكن فقط بعضها يتطلب فواصل منقوطة لتحديد نهاية الجملة.

موضوعات ذات صلة

ينطبق مفهوم المتغيرات المحلية والعالمية أيضًا على تطوير تطبيقات الأجهزة المحمولة، كما هو موضح في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

نصائح

كن حذرًا عند التعامل مع أنواع البيانات. فكر دائمًا في البيانات التي تريد تخزينها ونوع البيانات الأنسب للاستخدام. يمكن أن يؤدي الخطأ في نوع البيانات إلى مشاكل مثل اقتطاع المنازل العشرية والتقريب غير الصحيح، وكلاهما قد يكون كارثيًا في تطبيق يتعامل مع المال.

المصطلحات الرئيسة

منطقية (Boolean) – شكل من أشكال البيانات المنطقية سُمِيت على اسم عالم الرياضيات في القرن التاسع عشر، جورج بول.

حساس لحالة الأحرف – عندما تتعرف لغة برمجة على الفرق بين رموز الأحرف الإنجليزية الكبيرة والصغيرة، مثل "a" و"A". على سبيل المثال، إذا كانت كلمة الأمر برمز من رموز أحرف إنجليزية صغيرة، فسيحدث خطأ إذا تمت كتابتها بشكل غير متوقع برموز أحرف إنجليزية كبيرة. تفضل معظم لغات البرمجة الحديثة رموز الأحرف الإنجليزية الصغيرة.

الجدول 4.4 إعلانات المتغيرات باستخدام أنواع بيانات مختلفة في لغات برمجة مختلفة

	تنفيذ لغة البرمجة في...		
ما يجب الإعلان عنه...	C	C++	Microsoft Visual Basic .NET
Initial (**character**)	char initial;	char initial;	Dim initial As Char
Username (**string**)	char username[20];	string username;	Dim username As String
Age (**integer**)	int age;	int age;	Dim age As Integer
Price (**real**)	float price;	float price;	Dim price As Double
Valid (**Boolean**)	Not available	bool valid;	Dim valid As Boolean

الجدول 4.5 أنواع الجمل، أغراضها وتعبيراتها في لغات مختلفة

نوع الجملة	الغرض		
التعيين (Assignment)	تخزين قيمة في معرف (متغير أو ثابت).		
	Microsoft C#	**C++**	**Ruby:**
	userAge = 17;	userAge = 17;	userAge = 17
المدخلات (Input)	السماح بإدخال المستخدم، عادة من لوحة المفاتيح أو ملف.		
	Microsoft C#	**C++**	**Ruby:**
	userName = Console.ReadLine();	cin >> userName;	userName = gets
المخرجات (Output)	إنشاء مخرجات للمستخدم، عادةً إلى الشاشة أو الطابعة أو مكبر الصوت أو الملف وما إلى ذلك.		
	Microsoft C#	**C++**	**Ruby:**
	Console.WriteLine("Hello");	cout << "Hello";	puts "Hello"

المصطلحات الرئيسة

التسلسل - إجراء تلو الآخر، لا شيء مفقود، لا شيء متكرر.

الاختيار - الإجراءات المختارة بناءً على الحالة المقدمة والتي يتم تقييمها على أنها صحيحة أو خاطئة.

التكرار - عملية متكررة، عادة ما يتم القيام بشيء ما بشكل متكرر حتى تقترب من الحل قدر الإمكان.

موضوعات ذات صلة

يشبه كود C# للاختيارات والتكرارات إلى حد كبير أمثلة Java الموجودة في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

بُنى التحكم

عادةً ما يتم بناء الخوارزميات التي تتحكم في البرامج باستخدام مزيج من ثلاثة كتل بناء برمجية أساسية تُعرف بهياكل التحكم. هياكل التحكم هذه هي **التسلسل** و**الاختيار** و**التكرار**.

العوامل المنطقية

العوامل هي رموز خاصة تستخدم لأداء مهام خاصة في البرنامج. تعمل العمليات المنطقية مثل "And" و"Or" و"Not" وما إلى ذلك وفقًا لمبادئ منطقية، ويتم استخدامها لدمج الشروط في عبارات 'if' والحلقات المختلفة. على سبيل المثال، يظهر العامل المنطقي 'And' (&&) في مثال حالة الجملة 'if' الموضح في الكود الزائف في الشكل 4.10 في الصفحة الآتية. في هذا المثال، سيحصل العميل على شحن مجاني إذا كانت قيمة طلبه أكثر من مبلغ محدد **و**إذا كان عنوانه يبعد أقل من 25 كيلومترًا. **يجب** أن يكون كلا الجزأين من الشرط صحيحين للتأهل للشحن المجاني.

يجب توخي الحذر عند الانتقال بين اللغات. على سبيل المثال، في بعض اللغات (مثل Visual Basic .NET) يتم استخدام علامة '=' الواحدة لاختبار المساواة، بينما في لغات أخرى (PHP، C#، C++، C، Java) يتم استخدام '==' بدلًا من ذلك. عليك استغراق وقتًا في فحص الأنواع المختلفة من العوامل المتاحة في لغة البرمجة المستهدفة.

```
IF customerOrder > 50 and deliveryDistance < 26 THEN
    Output confirmation of discount message
```

الشكل 4.10 عينة من الكود الزائف

الروتينات الفرعية والوظائف والإجراءات

الروتينات الفرعية والدوال والإجراءات هي مصطلحات تُستخدم في البرمجة الإجرائية، حيث يتم تقسيم الكود إلى عدد من الوحدات المختلفة. قد يُطلق على كل وحدة اسم روتين فرعي أو دالة أو إجراء بحسب لغة البرمجة المستخدمة. كل وحدة مسؤولة عن أداء مهمة واحدة وعادة ما تكون بين 5 و50 سطرًا من الكود في الطول.

يوضح كود C# الموضح في الشكل 4.11 استخدام دالة معرفة من قبل المبرمج لحساب مساحة الدائرة عند إعطاء نصف قطر محدد.

```
class Program
{
    const float PI = 3.14f;

    static void Main(string[] args)
    {
        float radius;
        float area;

        radius = 10.0f;
        area = calcAreaCircle(radius);  // function call

        Console.WriteLine("Area of circle is {0}", area);
        Console.ReadKey();
    }

    //function declaration
    public static float calcAreaCircle(float radius)
    {
        return PI * radius * radius;
    }
}
```

الشكل 4.11 كود لحساب مساحة الدائرة بمعلومية نصف القطر

يعني استخدام هذه الوحدات أن الكود يميل إلى أن يكون أسهل في الكتابة والقراءة وتصحيح الأخطاء. عادةً ما يمكن إعادة استخدام الكود المكتوب بهذه الطريقة من خلال حلول متعددة ويسمح بتقسيم تطبيق واحد والعمل عليه من قبل عدة مبرمجين في الوقت نفسه، ما يعني أن وقت التطوير يمكن تقليله.

هيكل البيانات

هيكل البيانات هو تقنية برمجة تُستخدم لجمع وتنظيم عناصر البيانات في هيكل رسمي، ما يساعد على معالجة البيانات بكفاءة. على الرغم من أن توافر هياكل بيانات معينة يختلف بين لغات البرمجة المختلفة، إلا أن العديد منها شائع جدًا. تشمل هياكل البيانات الشائعة هذه السلسلة، والمصفوفة (أحادية وثنائية الأبعاد)، والمكدس، والانتظار والسجل.

موضوعات ذات صلة

يتم توفير قائمة مفصلة لمشغلات Java الشائعة (بما في ذلك الأنواع المنطقية) في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

نصائح

تحتوي جميع لغات البرمجة تقريبًا على دالات مكتبة يمكن للمبرمجين استخدامها عند حل المشكلات المعقدة. تسمح دالات المكتبة للمبرمج بأداء المهام الشائعة ولكن الحاسمة على البيانات، مثل تنسيق مظهرها أو العثور على طول السلسلة أو مربع الرقم. يمكنك أيضًا تنزيل وتثبيت دالات الطرف الثالث من المواقع الإلكترونية ذات السمعة الطيبة لتوسيع لغات البرمجة.

أحيانًا، يمكن البرمجة بشكل أكثر كفاءة من خلال اختيار واستخدام هياكل بيانات محددة، خاصةً عندما يتم دمج ذلك مع هياكل التحكم في التكرار (الحلقات). يصبح مطورو البرمجيات على دراية بهياكل البيانات المختلفة في أثناء تعلمهم لغات البرمجة المختلفة، ويرد في ما يأتي بعض هياكل البيانات الأكثر شيوعًا.

النص (أو السلسلة)

هذه بنية بيانات تُستخدم لتخزين مجموعة من الرموز إذ يتطلب الرمز الواحد على الأقل بايت واحد من ذاكرة الوصول العشوائي. قد تكون السلاسل ذات 'طول ثابت' (على سبيل المثال، تحتوي فقط على عشرة رموز). بدلًا من ذلك، يمكنهم استخدام رمز "terminator" خاص لتحديد نهايتهم. قد يعني هذا أن سلسلة تتطلب تضمين رمز إضافي، ولكنه يسمح لها بأن تكون ذات أطوال مرنة.

يمكن الوصول إلى كل رمز فردي في سلسلة باستخدام الفهرس الموضعي الخاص به. على سبيل المثال، في الجدول 4.6، الكوكب[2] هو 'r'. لاحظ أن فهرس الموضع الأول هو دائمًا 0.

الجدول 4.6 لكل رمز في سلسلة يوجد فهرس موضعي

البيئة					
0	1	**2**	3	4	5
ه	أ	**r**	t	h	#

يتم توضيح هذا المثال في سلاسل C# في الشكل 4.12.

```
string Planet;
Planet = "Earth";

Console.WriteLine("Whole string is {0}", Planet);
Console.WriteLine("3rd character is {0}", Planet[2]);
```

الشكل 4.12 العمل مع سلاسل C#

تستخدم بعض اللغات مصفوفة ثابتة أحادية البعد من الرموز لمحاكاة سلسلة نصية، بدلًا من استخدام نوع بيانات سلسلة نصية محدد. ومع ذلك، فإن استخدامها متشابه جدًا.

تُستخدم السلاسل عادةً لتخزين النصوص البسيطة التي يدخلها المستخدم، مثل اسم المستخدم. قد تُستخدم السلاسل أيضًا لإدخال نصوص أكثر تعقيدًا للمعالجة، مثل محتوى البريد الإلكتروني أو الرسائل الفورية أو التغريدات. يُستخدم مصطلح 'substring' لوصف جزء من سلسلة أكبر، على سبيل المثال 'Ear' هو جزء من 'Earth'.

مصفوفة (أحادية البعد)

في العادة، هذا هيكل بيانات ثابت، ما يعني أنها ذات حجم ثابت. ومع ذلك، فإن لغات البرمجة الحديثة تكون عادةً أكثر مرونة وتسمح بتغيير حجم المصفوفة.

عادةً ما يمكن للمصفوفة تخزين نوع واحد فقط من البيانات، ويُقبل أي نوع من البيانات، مثل الأعداد الصحيحة والرموز والقيم المنطقية.

إذا أردنا تخزين سبع درجات حرارة قصوى يومية بالدرجة المئوية لمحطة الطقس المحلية، فسنقوم بإنشاء مصفوفة من سبعة أرقام عشرية، كما هو موضح في الجدول 4.7.

الجدول 4.7 مصفوفة أحادية البعد من سبعة أرقام عشرية

Temperature						
0	1	2	3	4	5	6
12.50	10.45	12.30	14.60	17.70	11.20	12.50

يبدو أن هذا يشبه السلسلة. في الواقع، يمكنك اعتبار السلسلة مصفوفة من الرموز، وكما هو الحال مع السلسلة، من الممكن الوصول إلى العناصر الفردية أو العناصر في المصفوفة باستخدام الفهرس المطلوب. على سبيل المثال، Temperature [4] هي 17.70. يوضح الشكل 4.13 إنشاء هذه المصفوفة البسيطة أحادية البعد والوصول إليها في C#.

```
//create the array of decimal temperatures
float[] temperature = new float[7];

//initialise each element
temperature[0] = 12.50f;
temperature[1] = 10.45f;
temperature[2] = 12.30f;
temperature[3] = 14.60f;
temperature[4] = 17.70f;
temperature[5] = 11.20f;
temperature[6] = 12.50f;

//select Thursday's and output it
Console.WriteLine("Temperature on Thursday is {0} degrees C", temperature[4]);

//wait for keypress to continue
Console.ReadLine();
```

الشكل 4.13 مصفوفة أحادية الأبعاد في C#

مصفوفة (ثنائية الأبعاد)

مصفوفة ثنائية الأبعاد تشبه المصفوفة أحادية الأبعاد، لكنها يمكن أن تخزن عدة صفوف من البيانات. تخيل أننا ما زلنا نريد تخزين سبع درجات حرارة قصوى يومية بالدرجة المئوية لمحطة طقس محلية، ولكننا نحتاج إلى القيام بذلك على مدار ثلاثة أسابيع متتالية. في هذه الحالة، سنقوم بإنشاء مصفوفة ثنائية الأبعاد مكونة من سبعة في ثلاثة أرقام عشرية، كما هو موضح في الجدول 4.8.

الجدول 4.8 مصفوفة ثنائية الأبعاد

Temperature							
	0	**1**	**2**	**3**	**4**	**5**	**6**
0	12.50	10.45	12.30	14.60	17.70	11.20	12.50
1	12.22	11.00	10.00	20.00	21.00	14.00	15.50
2	13.00	14.00	14.50	14.60	12.30	14.00	14.60

كما كان من قبل، من الممكن الوصول إلى العناصر الفردية أو العناصر في المصفوفة باستخدام كلا المؤشرين، على سبيل المثال، Temperature [1][4] هي القيمة الموجودة حيث يلتقي العمود 4 والصف 1: 21.00.

من الممكن استخدام أوامر أعلى للأبعاد. على سبيل المثال، يمكن استخدام مصفوفة ثلاثية الأبعاد لتخزين 21 درجة حرارة قمنا بتخزينها لكل من محطات الطقس الخمس المختلفة.

يمكن تقسيم المصفوفات ودمجها. في بعض اللغات، يُطلق على هذا اسم تفجير المصفوفات ودمج المصفوفات. في اللغات التي يتم فيها تخزين السلاسل بوصفها صفوفًا من الرموز، فإن استخراج السلاسل الفرعية ودمج السلاسل المختلفة هي استخدامات عملية لهذه الأنواع من العمليات. راجع معالجة السلسلة في الصفحة 27 لمزيد من المعلومات حول استخراج السلسلة الفرعية وتسلسل السلسلة.

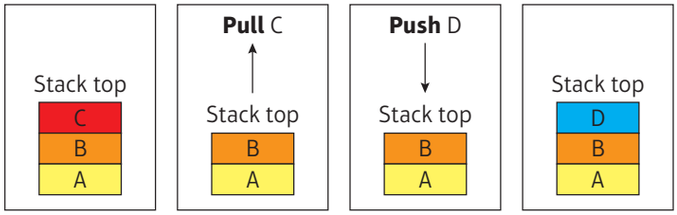
نصائح

قد يختلف الترتيب الفعلي للفهارس بين لغات البرمجة المختلفة. القاعدة العامة هي أن ترتيب الوصول إلى العنصر سيعكس ترتيب الإعلان الخاص بالمصفوفة، أي لا يمكنك الوصول إلى عنصر في Temperature [4][1] لأن هذه المؤشرات لا تعكس ترتيب الإعلان الخاص بالمصفوفة.

المكدس (STACK)

يُعرف المكدس بأنه هيكل بيانات يعتمد على طريقة **LIFO**. لديها عمليتان أساسيتان: الدفع والسحب (يُعرف أحيانًا بالدفع والإخراج).

المكدسات جزء أساسي من نظام تشغيل أي منصة حاسوبية، وهي أداة شائعة يستخدمها المبرمج عند تطوير الحلول. يتم عرض المكدسات بشكل عمودي، كما هو موضح في الشكل 4.14.



الشكل 4.14 عمليات المكدس

يمكنك رؤية عملية المكدس في C# في الشكل {4.15}.

```
using System;
using System.Collections;

namespace stack
{
    class Program
    {
        static void Main(string[] args)
        {
            char data;                  //single item of stack data
            Stack st = new Stack();     //our stack

            //push onto stack
            st.Push('A');
            st.Push('B');
            st.Push('C');

            //show the stack
            Console.WriteLine("Current stack: ");
            foreach (char ch in st)
            {
                Console.WriteLine(ch);
            }

            //remove from stack top - it should be "C"
            data = (char)st.Pop();
            Console.WriteLine("The popped value: {0}", data);

            //push onto stack again
            st.Push('D');

            //show the changed stack
            Console.WriteLine("Current stack: ");
            foreach (char ch in st)
            {
                Console.WriteLine(ch);
            }

            //wait for keypress to continue
```

الشكل 4.15 عملية المكدس في C#

المصطلح الرئيس

LIFO - يعني هذا "آخر من يدخل، أول من يخرج" ويصف كيفية معالجة البيانات في بعض هياكل البيانات. هذا يعني أن العنصر الأخير من البيانات الذي تم الضغط عليه هو أيضًا العنصر الأول من البيانات الذي قد يتم سحبه مرة أخرى.

```
            //wait for keypress to continue
            Console.ReadKey();
        }
    }
}
```

الشكل 4.15 *متابعة*

على عكس المصفوفة، حيث يمكن الوصول إلى العناصر بأي ترتيب، يمكن الوصول إلى المكدس فقط بطريقة آخر ما يدخل، أول ما يخرج. يمكن أن يكون هذا مفيدًا بشكل خاص عند معالجة **الخوارزميات المتكررة**. غالبًا ما يتم تضمين المكدسات في مكتبة لغة البرمجة، مع جميع الأساليب اللازمة لمعالجتها.

قائمة الانتظار (FIFO)

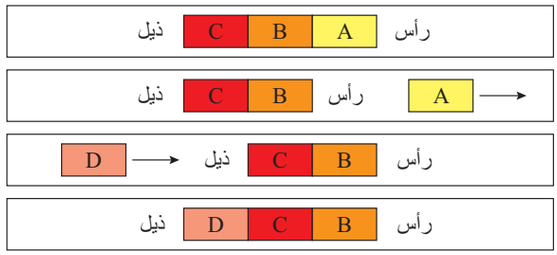
على عكس المكدس، تُعرف قائمة الانتظار البسيطة باسم بنية بيانات **FIFO** ولديها عمليتان أساسيتان:

- إضافة (إلى قائمة الانتظار) أو 'إدراج'
- إزالة (من بداية قائمة الانتظار) أو 'إلغاء قائمة الانتظار'.

فقط البيانات الموجودة في مقدمة قائمة الانتظار يمكن الوصول إليها وإزالتها.

مثل المكدسات، تُعتبر قوائم الانتظار جزءًا حيويًا من نظام تشغيل أي منصة حاسوبية. على سبيل المثال، قد تكون على دراية بمفهوم قائمة انتظار الطابعة. بالنسبة للمبرمج، فهي طريقة ممتازة لإدارة معالجة المهام.

يتم عرض قوائم الانتظار بشكل أفقي كما هو موضح في الشكل 4.16.



الشكل 4.16 عمليات قوائم الانتظار

الشكل 4.17 يوضح إنشاء والوصول إلى قائمة انتظار بسيطة في C#.

```
using System;
using System.Collections;
using System.Linq;
using System.Text;

namespace queue
{
    class Program
    {
        static void Main(string[] args)
        {
            char data;              //single item of queue data
            Queue q = new Queue();  //our queue
```

الشكل 4.17 قائمة انتظار بسيطة C#

المصطلحات الرئيسة

الخوارزميات المتكررة - جزء من كود البرمجة الذي ينفذ نفسه بشكل متكرر حتى يصل إلى حالة نهائية حيث يمكن إرجاع النتيجة المحسوبة.

FIFO - هذا يعني "أول من يدخل، أول من يخرج". هذا يعني أن العنصر الأول من البيانات المضافة هو أيضًا العنصر الأول من البيانات الذي يمكن إزالته.

```
            //add to queue
            q.Enqueue('A');
            q.Enqueue('B');
            q.Enqueue('C');

            //show the queue
            Console.WriteLine("Current queue: ");
            foreach (char ch in q)
            {
                Console.Write(ch + " ");
            }

            //remove from head - it should be 'A'
            data = (char)q.Dequeue();
            Console.WriteLine("The removed value: {0}", data);

            //add to queue again
            q.Enqueue('D');

            //show the changed queue
            Console.WriteLine("Current queue: ");
            foreach (char ch in q)
            {
                Console.Write(ch + " ");
            }

            //wait for keypress to continue
            Console.ReadKey();
        }
    }
}
```

الشكل 4.17 *متابعة*

السجل (أو الهيكل)

السجل هو مفهوم مشابه للمصفوفة. ومع ذلك، فإنه يختلف لأنه يمكنه تخزين مزيج من أنواع البيانات داخل هيكله. على سبيل المثال، يمكن استخدامه لتخزين التفاصيل، كما هو موضح في الجدول 4.9.

الجدول 4.9 سجل تفاصيل الطلاب

الطالب				
StudentID	الاسم الأول:	اللقب	العمر	مسجل
2001	بريستون	مايلا	21	صواب

في هذا المثال، يتم تخزين 'رقم تعريف الطالب' و'عمره' كأعداد صحيحة، و'الاسم الأول' و'اسم العائلة' كسلاسل نصية و'مسجل' كقيمة منطقية. يظهر تنفيذ بسيط لهيكل سجل في C# في الشكل 4.18 في الصفحة الآتية.

```
using System;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    public class testStructure
    {
        //create the record structure
        struct Student
        {
            public int StudentID;
            public string Surname;
            public string Forename;
            public int Age;
            public bool Enrolled;
        };

        public static void Main(string[] args)
        {
            Student mystudent;    //declare mystudent of type Student structure

            //initialise the elements in the record structure
            mystudent.StudentID = 2001;
            mystudent.Surname = "Preston";
            mystudent.Forename = "Myla";
            mystudent.Age = 21;
            mystudent.Enrolled = true;

            //print the data in the record structure
            Console.WriteLine("ID        : {0}", mystudent.StudentID);
            Console.WriteLine("Surname   : {0}", mystudent.Surname);
            Console.WriteLine("Firstname : {0}", mystudent.Forename);
            Console.WriteLine("Age       : {0}", mystudent.Age);
            Console.WriteLine("Enrolled  : {0}", mystudent.Enrolled);

            //wait for a keypress to continue
            Console.ReadKey();
        }
    }
}
```

الشكل 4.18 بنية سجل أو هيكل (سجل) بسيط في C# باستخدام 'struct'

يمكنك أيضًا إنشاء مصفوفة من الهياكل لتخزين مثيلات متعددة من سجل ما. لهذا السبب، يمكن اعتبار السجل (أو "البنية") كأساس لتخزين السجلات البسيطة في جدول قاعدة بيانات أو ملف بيانات ثنائي.

معالجة السلاسل

على الرغم من أن السلاسل هي ببساطة مجموعة من الرموز، إلا أنها قد تحتاج إلى معالجة. يُعرف هذا العمل عادةً بين المبرمجين بمعالجة السلاسل. غالبًا ما يحتاج المبرمجون إلى تنفيذ عمليات معينة على السلاسل النصية ومعظم لغات البرمجة عالية المستوى تحتوي على دوال مكتبية متخصصة للمساعدة. في ما يأتي بعض المهام الشائعة، مع أمثلة عملية على كود C# توضح تنفيذها. وظائف السلاسل النصية المماثلة موجودة في العديد من لغات البرمجة الأخرى.

إيجاد طول سلسلة نصية بعدد الرموز

في الشكل 4.19، يتم استخدام خاصية الطول لكائن السلسلة النصية 'month' للوصول إلى عدد الرموز في السلسلة.

```
string month = "January";
int length;

length = month.Length;
Console.WriteLine("{0} is {1} characters long", month, length);
```

الشكل 4.19 في هذا المثال، السلسلة هي الكلمة 'January'

فحص الرموز الفردية

في الشكل 4.20، سيكون الناتج هو 'الرمز 3 من كلمة January هو u'، لأن 'u' هو الرمز الموجود في الموضع 3.

```
string month = "January";
char singleLetter;
int whichLetter = 3;

singleLetter = month[whichLetter];
Console.WriteLine("Character {0} of {1} is {2}", whichLetter, month, singleLetter);
```

الشكل 4.20 السلسلة تبدأ برمز 'J' في الموضع 0

استخراج سلسلة فرعية

في الشكل 4.21، سيكون الناتج هو 'Substring is ua'. هذا هو الناتج لأننا نبدأ الاستخراج عند الموضع 3 ('u')، ونريد استخراج الرمزين الآتيين، بما في ذلك الأول ('ua').

```
string month = "January";
string subString;
int startPos = 3;
int howMany = 2;

subString = month.Substring(startPos, howMany);
Console.WriteLine("Substring is {0}", subString);
```

الشكل 4.21 نموذج للسلسلة الفرعية

دمج سلسلتين (Concatenating two strings)

الدمج (Concatenation) هو عملية ضم الأشياء معًا، ويُستخدم في البرمجة لوصف عملية ضم سلاسل أقصر معًا لتشكيل سلاسل أطول، على سبيل المثال "Hello" و"World!" لتشكيل "Hello World!".

في الشكل 4.22 في الصفحة الآتية، نقوم بدمج سلسلتين: 'January' و'has 31 days'. الناتج سيكون 'Joined string is January has 31 days'. هذا هو الناتج لأننا نستخدم عامل الدمج '+' لربط السلسلتين معًا. تتوفر تقنيات أخرى في C#، ولكن ربما يكون هذا هو الأبسط في الاستخدام.

```
string month = "January";
string saying = " has 31 days.";
string combined;

combined = month + saying;
Console.WriteLine("Joined string is {0}", combined);
```

الشكل 4.22 دمج سلسلتين

معالجة الملفات

تخزين البيانات هو جانب رئيس من البرمجة الحديثة. القدرة على فتح ملفات البيانات للقراءة (أي الحصول على البيانات من ملف) والكتابة (أي إرسال البيانات إلى ملف) في لغة البرمجة أمر أساسي. لكي تستخدم ملفًا، يجب فتحه، وعندما تنتهي يجب إغلاقه. تتطلب بعض لغات البرمجة أوامر خاصة لتنفيذ هذه العمليات. على سبيل المثال، تستخدم C الدالة ()fopen و()fclose.

شكل بسيط من معالجة الملفات هو القدرة على قراءة البيانات من ASCII أو ملف نصي. يمكن تحقيق ذلك في Microsoft C# باستخدام بعض التركيبات البسيطة، كما هو موضح في الشكل 4.23.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //exception handling
            try
            {
                // attempt to read the named file
                using (StreamReader file = new StreamReader("c:/customer.txt"))
                {
                    string line; //stores a single line of text read from the file

                    //loop which continues to read a line of text until
                    //end of file (EOF) is reached
                    while ((line = file.ReadLine()) != null)
                    {
                        //output the line of text read
                        Console.WriteLine(line);
                    }
                }
            }
            catch (Exception e)
            {
                //show user the correct error message.
                Console.WriteLine("Sorry, could not read the file.");
                Console.WriteLine(e.Message);
            }
            Console.ReadKey();
        }
    }
}
```

الشكل 4.23 معالجة ملفات C# توضح قراءة ملف ASCII/نص

المصطلحات الرئيسة

واجهة المستخدم الرسومية (GUI) – واجهة مستخدم حديثة تتكون من النوافذ والرموز التي يتم التحكم فيها عن طريق الفأرة والمؤشر. ومن الأمثلة على ذلك Microsoft Windows 10 أو Apple OS X.

واجهة سطر الأوامر (CLI) – نمط قديم لواجهة المستخدم القائمة على النص والتي ما تزال قيد الاستخدام. يتفاعل المستخدمون مع الحاسوب عبر الأوامر التي يتم إدخالها من لوحة المفاتيح. ومن الأمثلة على ذلك موجه أوامر Microsoft أو وحدة Linux®.

في الشكل 4.23، يتم استخدام حلقة 'while' بسيطة بلغة C# للتكرار عبر كل سطر من الملف النصي المحدد. عند قراءة السطر النصي، يتم إخراجه على الشاشة. يتم استخدام معالجة الاستثناءات لضمان عدم تعطل البرنامج بشكل قاتل إذا تعذر العثور على الملف النصي المحدد.

تدعم معظم لغات البرمجة الحديثة (مثل C# Microsoft وPHP وRuby وما إلى ذلك) أيضًا الاتصال بقواعد البيانات العلائقية مثل Microsoft SQL وMySQL. هذا يدعم معالجة البيانات الأكثر تعقيدًا.

معالجة الأحداث

معالجة الأحداث – تصف عملية استخدام دالة مكتوبة خصيصًا لتنفيذ إجراءات محددة عند تفعيل حدث من قبل المستخدم أو النظام. ومن أمثلة ذلك إخراج نتيجة عملية حسابية عند الضغط على زر في نموذج شاشة، أو عرض رسالة تأكيد عند تغيير دقة الشاشة.

توجد الأحداث بشكل أكثر شيوعًا في التطبيقات القائمة على النماذج، مثل تلك التي تستخدم عناصر **واجهة المستخدم الرسومية (GUI)** للسماح للمستخدم بالتفاعل مع التطبيق. من الممكن إنشاء تطبيقات بسيطة لمعالجة الأحداث تعمل في بيئة وحدة التحكم أو **واجهة سطر الأوامر (CLI)**، كما هو موضح في الشكل 4.24.

```
//Publisher - defines the event and the delegate
public class myClock
{
    //Defines the delegate, needed to define an event inside a class
    public delegate void SecondHandler(myClock clock, EventArgs evt);
    //Defines an event based on the delegate
    public event SecondHandler second;
    public EventArgs evt = null;

    public void Start()
    {
        //never stop!
        while (true)
        {
            //1000 milliseconds is 1 second
            System.Threading.Thread.Sleep(1000);
            if (second != null)
            {
                //raise the event (each second)
                second(this, evt);
            }
        }
    }
}

//Subscriber - accepts the event and provides the event handler (showClock)
public class Listener
{
    public void Subscribe(myClock clock)
    {
        clock.second += new myClock.SecondHandler(showClock);
    }
```

الشكل 4.24 استخدام أحداث وحدة التحكم في C# لإنشاء ساعة في الوقت الفعلي

```
    //Event Handler - shows the clock in the top-left corner of the screen
    private void showClock(myClock clock, EventArgs evt)
    {
        Console.CursorLeft = 1;
        Console.CursorTop = 1;
        DateTime now = DateTime.Now;
        Console.WriteLine(now);
    }
}
//class used to test the event-driven clock
class Test
{
    static void Main()
    {
        //create a new clock object
        myClock clock = new myClock();
        //create a new listener object
        Listener myEventListener = new Listener();
        //link the event and the handler
        myEventListener.Subscribe(clock);
        //start the clock ticking...
        clock.Start();
    }
}
```

الشكل 4.24 *متابعة*

في هذا المثال، يتم استخدام حدث لمدة ثانية واحدة لتوفير ساعة في الوقت الفعلي على شاشة وحدة التحكم. هذا مثال لنموذج الناشر-المشترك في C#. يحدد الناشر الحدث والمفوض. بمجرد أن يقوم الناشر باستدعاء أو تفعيل الحدث، يتم إخطار الكائنات الأخرى في البرنامج مثل المشترك. ثم يقبل المشترك الحدث ويوفر معالج حدث مناسب. في هذه الحالة، معالج الحدث المناسب هو دالة تعرض الساعة في الزاوية العلوية اليسرى من الشاشة.

توثيق الكود

يجب دائمًا توثيق كود البرنامج وفقًا لمعايير جيدة. سيؤدي القيام بذلك إلى:

- جعل الكود أكثر قابلية للقراءة
- مساعدة المبرمجين الآخرين على فهم الكود
- المساعدة عندما يقوم المبرمجون بتصحيح الكود
- المساعدة في صيانة كود البرنامج، خاصة إذا كان ذلك سيتم بواسطة مطور آخر.

يمكن للمبرمجين استخدام تقنيات مختلفة لتوثيق كودهم. تشمل هذه التقنيات:

- المعرفات ذاتية التوثيق - وهذا يعني استخدام أسماء ذات معنى تشرح الغرض (وأحيانًا نوع البيانات) للقيمة التي يمثلها المعرف
- تعليقات السطر الواحد - يُقصد بها تعليقات قصيرة تشرح الغرض من سطر من الكود
- تعليقات متعددة الأسطر - يُقصد بها تعليقات أطول تشرح الغرض من قسم أكثر تعقيدًا من الكود أو تمهيد لقسم منفصل من الكود، مثل دالة أو إجراء أو فئة.
- استخدام الثوابت - يُقصد به استبدال مثيلات متعددة من القيم "الرمزية" الثابتة (الرقمية والنصية وما إلى ذلك) المنتشرة في جميع أنحاء البرنامج بمعرفات مسماة يمكن تعديلها عند نقطة واحدة في كود البرنامج.

موضوعات ذات صلة

تلعب معالجة الأحداث القائمة على النماذج دورًا مهمًا في تطوير تطبيقات الأجهزة المحمولة، كما هو موضح في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

نصائح

يمكن أن تختلف قواعد إنشاء أسماء المعرفات بشكل كبير بين لغات البرمجة. يمكن لبعض المعرفات استخدام رموز معينة فقط (مثل عدم وجود مسافات) أو لا يمكن أن تبدأ برموز معينة (مثل الأرقام من 0 إلى 9). بالإضافة إلى ذلك، غالبًا ما يتم تقديم المشورة الأسلوبية، على سبيل المثال لا تختصر أو تستخدم الاختصارات عند تسمية المعرفات.

يوضح الجدول 4.10 هذه التقنيات الأكثر شيوعًا في C#.

الجدول 4.10 توثيق الكود في Microsoft C#

مفهوم التوثيق	Microsoft C#
المعرفات ذاتية التوثيق	`int userAge;`
تعليقات سطر واحد	`// stores user's age`

تعليقات متعددة الأسطر	`/*`
	`This section of code validates the user's age when entered.`
	`*/`

استخدام الثوابت	`const double PI = 3.14159;`

موضوعات ذات صلة

تتوافر أيضًا الأدوات التي تنشئ وثائق HTML مباشرةً من التعليقات الموضوعة في كود البرمجة. تمت مناقشة هذا بمزيد من التفصيل في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

المهارات

- مهارات التحليل واتخاذ القرار

بحث

يوصي العديد من ناشري البرامج بمعايير لتسمية المعرفات. على سبيل المثال، توفر Microsoft وثائق عبر الإنترنت تحتوي على نصائح وإرشادات حول هذه المشكلات الفنية من أجل تشجيع ممارسات البرمجة الجيدة. قم بزيارة شبكة المطورين الخاصة بهم لمعرفة اصطلاحات تسمية المعرفات الموصى بها.

مبادئ المنطق المطبقة على تصميم البرنامج

في الحوسبة، المنطق هو مجموعة المبادئ الموجودة أسفل العناصر المختلفة للبرنامج أو النظام التي تسمح للبرنامج أو النظام بالعمل. يستفيد تصميم البرنامج الناجح من تطبيق المنطق البسيط. هذا أكثر أهمية من القدرة على التفكير بلغة برمجة معينة. إن القدرة على التفكير المنطقي ستساعدك على معالجة البيانات بشكل صحيح، وتشكيل شروط معقدة تمكنك من اختبار البيانات بشكل مناسب وبناء خوارزميات قوية لحل حتى أكثر المشكلات تعقيدًا.

تشمل مبادئ المنطق المستخدمة في الحوسبة المنطق الرياضي والتكرارات والمنطق الافتراضي واستخدام المجموعات. وتساعد هذه المفاهيم المبرمجين على فهم المشكلات بشكل كامل وبناء المنطق الذي يتم دمجه في برامجهم.

المنطق الرياضي

يمكن تطبيق المنطق الرياضي لمساعدة المطورين على فهم ووصف الحقائق الأساسية في مشكلة ما وعلاقاتها ببعضها البعض بدقة. على سبيل المثال، الاستدلال هو العملية المستخدمة للوصول إلى استنتاج منطقي من مقدمات (مواقف) يُعتقد أو يُعرف أنها صحيحة. يمكن رؤية ذلك في المثال الآتي:

- الفرضية: جميع اللحوم تأتي من الحيوانات. هذا صحيح.
- الفرضية: يأتي لحم الضأن من الأغنام. هذا صحيح.
- الاستدلال: لذلك يأتي لحم الضأن من الحيوانات.

هذا الاستنتاج صحيح لأن الأغنام حيوانات. الاستدلالات الخاطئة تُسمى مغالطات.

يعد مفهوم الاستدلال واستخدامه أمرًا مهمًا عند استخدام لغات البرمجة مثل Prolog ("البرمجة بالمنطق") لبناء قواعد معرفية بسيطة يمكن الاستعلام عنها للكشف عن حقائق جديدة.

يمكن أيضًا مراعاة المفاهيم المنطقية الأخرى. على سبيل المثال، يحاول الاتساق معرفة ما إذا كانت الجملة (ونقيضها) صحيحة من أجل تحديد ما إذا كانت هناك تناقضات في الجملة.

التكرار

يحدث التكرار عندما يتم تطبيق إجراء حسابي على نتيجة تطبيق سابق.

الاكتمال

الاكتمال هو مفهوم منطقي ينص على أنه يمكنك "إثبات" أي شيء صحيح. في ما يأتي تأكيد بسيط يستخدم الرموز الآتية:

- Σ (الرمز اليوناني سيجما)
- Φ (الرمز اليوناني فاي)
- ⊣ (رمز البرهان) يعني 'يثبت' أو 'ينتج'.

كتابة التأكيد 'Σ ⊣ Φ' تعني 'من Σ، أعلم أن Φ'. ببساطة أكثر، يعني أن Σ 'يثبت' Φ.

هناك العديد من الأشكال المختلفة للاكتمال، مثل الوظيفي والدلالي. في البرمجة، يمكن استخدام الاكتمال لإثبات ما إذا كانت الخوارزميات ستعمل بنجاح وفقًا لقواعد وأهداف ومنطق معين.

جداول الحقيقة

يمكنك التحقق من المنطق الخوارزمي البسيط من خلال استخدام جداول الحقيقة. على سبيل المثال، لا يمكن للعميل تسجيل الدخول (L) إلى النظام إلا إذا كان لديه اسم مستخدم صالح (U) وكلمة مرور (P)، ولم يتم حظره (O) خلال آخر 30 دقيقة.

في جدول الحقيقة لهذا المثال، يُستخدَم 1 لتمثيل 'true'، بينما '0' يعني false. الرمز '.' يعني 'و' المنطقية. الشريط الموجود أعلى "O" يعني "لا" منطقيًا. هذا يعني أنه يمكننا التعبير عن هذه العبارة المنطقية كالآتي: $L = (U.P).\bar{O}$ أو 'تسجيل الدخول الناجح = اسم المستخدم وكلمة المرور وعدم الإغلاق'. بعد تقييم التعبير، يمكننا إثبات التركيبة الوحيدة الصالحة، كما هو موضح في الجدول 4.11. يمكن تحويل التعبير الناتج إلى أي لغة برمجة مستهدفة، حيث أن منطقه الأساسي مثبت بواسطة جدول الحقيقة.

الجدول 4.11 جدول الحقيقة الذي يوضح التوليفات الممكنة والترتيب الصحيح الوحيد (الصف بالخط العريض)

U	P	O	L $(U.P).\bar{O}$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	**1**	**0**	**1**
1	1	1	0

المنطق الاقتراحي

يعتمد المنطق الافتراضي على المفهوم البسيط القائل بأنه يمكن اعتبار الجملة إما صحيحة وإما خاطئة، ولكنها ليست صحيحة وخاطئة. فكر في هذه الأمثلة.

- الشمس حارة.
- اثنان زائد ثلاثة يساوي ستة.

كلتا الجملتين عبارة عن افتراضات ولديهما قيمة حقيقية إما "صحيحة" وإما "خاطئة". المثال الأول صحيح لأن الشمس بالفعل حارة. المثال الثاني هو اقتراح خاطئ، لأن اثنين زائد ثلاثة يساوي خمسة، وليس ستة. هذه الأنواع من الجمل البسيطة تُعرف بالاقتراحات الأساسية.

يمكن إنشاء مقترحات أكثر تعقيدًا باستخدام الروابط. هناك خمسة روابط أساسية. يحتوي كل منها على شكل كلمة ورمز، كما هو موضح في الرسوم البيانية أدناه.

∨	∧	¬
أو	و	غير
↔		→
IF_AND_ONLY_IF		IF_THEN (أو ضمنيًا)

تسمح لنا هذه الرموز الأساسية للربط بتمثيل تعبيرات معقدة بطريقة رياضية للغاية.

على سبيل المثال، يمكن التعبير عن الجملة، 'إذا كان لدي مال (**M**oney) أو تذكرة (**T**icket)، يمكنني مشاهدة فيلم' (**F**ilm) كما يأتي: 'F → (M ∨ T)'. بعبارة أخرى: F صحيح إذا كانت M أو T صحيحة.

في مثال آخر، يمكن التعبير عن عبارة "إذا لم يكن لدي مال (**M**oney)، فعند استحقاق دفعة مستحقة مباشرة (**D**irect **D**ebit)، يجب أن تحدث غرامة مصرفية (**B**ank **F**ine)" على النحو الآتي: '¬ M → (DD → BF)'. بعبارة أخرى: إذا كانت M خاطئة، فإن كان DD صحيحًا، يجب أن يكون BF صحيحًا.

لأن مثل هذه الجمل المنطقية البسيطة تستخدم تدوينًا رياضيًّا رسميًّا، فمن الممكن اختبار صحتها قبل إنتاج أي كود برمجي. من الممكن أيضًا استخدام المنطق الاقتراحي لاشتقاق معرفة جديدة من الحقائق الموجودة.

يُستخدم تعديل إضافي يسمى المنطق الديناميكي المقترح propositional dynamic logic (PDL) لعرض وظيفة الخوارزميات. تم تصميم المنطق الديناميكي المقترح لعرض حالات وأحداث الأنظمة الديناميكية (مثل البرامج العاملة) بطريقة رياضية سهلة. إحدى أهم ميزاته هي قدرته على التحقق مما إذا كانت الخوارزميات المكتملة يمكن أن تكتمل بالحالة المطلوبة: أي حساب النتيجة الصحيحة.

استخدام المجموعات

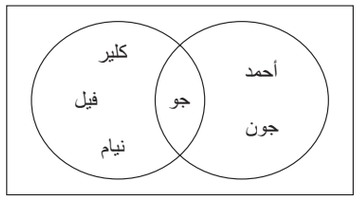
يمكن استخدام **المجموعات** للمساعدة في تنظيم البيانات وتحديد العلاقات المتبادلة بين مجموعات البيانات المختلفة. وهذا يسمح لك بالبحث بسهولة و**تصفية** البيانات المعقدة المحتملة. تحتوي بعض لغات البرمجة على أنواع بيانات تدعم وظائف نمط المجموعة، ويمكن استخدام هذا المنطق لتكوين أذونات المستخدم.

على سبيل المثال، قد يكون لديك مجموعتان من الأشخاص بحقوق وصول مختلفة في برنامج معين. المجموعة A هي مجموعة المسؤولين، والتي تحتوي على جون، أحمد وجو. يمكن التعبير عن هذا على أنه المجموعة أ = {جون، أحمد، جو}. المجموعة B هي المجموعة المالية التي تضم كلير وجو ونيام وفيل. يمكن التعبير عن هذا على أنه المجموعة ب = {كلير، جو، نيامه، فيل}. يمكننا تمثيل هذه المجموعات كمخطط فين، كما هو موضح في الشكل 4.25 في الصفحة الآتية.

المصطلحات الرئيسة

المجموعة – مجموعة من الكائنات المميزة. يمكن أن تحتوي المجموعات على أي شيء (مثل الأسماء والأرقام والألوان أو الحروف الأبجدية) وقد تتكون من العديد من الأعضاء المختلفين.

التصفية – تضمين قيم معينة أو استبعادها عند تشغيل البحث.



الشكل 4.25 مخطط فين يوضح مجموعتين من الناس

هناك وظيفة معينة في البرنامج لا يمكن الوصول إليها إلا من قبل المسؤولين (أعضاء مجموعة المسؤولين) الذين يعملون في قسم المالية (أعضاء مجموعة المالية). بالنظر إلى الشكل 4.25، يمكننا بسهولة أن نرى أن المستخدم الوحيد الذي يوجد في كلا المجموعتين هو جو. هذا يعني أن جو فقط لديه حق الوصول إلى هذه الوظيفة.

جودة التطبيقات البرمجية

عند مواجهة المشكلة نفسها، يمكن للمبرمجين المختلفين التوصل إلى حلول مختلفة تمامًا. يحدث هذا عادةً لأن حل المشكلات هو عملية شخصية للغاية ومعظم لغات البرمجة تتمتع بمرونة كافية للسماح بمجموعة متنوعة من الحلول لمشكلة واحدة. ومع ذلك، فإن التصميم الفعلي وتنفيذ الحل البرمجي له تأثير مباشر في جودته العامة. يتم الحكم على جودة الحل البرمجي من خلال تقييم كفاءته أو أدائه، وقابليته للصيانة، وقابليته للنقل، وموثوقيته، وقوته، وسهولة استخدامه.

الكفاءة أو الأداء

عادةً ما يتم تقييم أداء التطبيق من خلال قياس موارد النظام التي يستهلكها البرنامج. على سبيل المثال، دورات ساعة وحدة المعالجة المركزية ووقت المعالج وذاكرة الوصول العشوائي المخصصة ومعدل قراءة وسائط التخزين والكتابة إليها. قد تكون العملية المعقدة تتطلب الكثير من وحدة المعالجة المركزية. قد يكون من الممكن مراجعة كود البرنامج ليكون أكثر اقتصادية، ما قد يحسن الأداء (يعمل بشكل أسرع) ويستخدم موارد أقل (مثل ذاكرة الوصول العشوائي وما إلى ذلك). هذا النوع من المراجعة شائع في البرامج التي تعتمد على السرعة، مثل برمجة ألعاب الفيديو (حيث يمكن أن تتأثر سرعة إطارات الشاشة بسبب كتابة الأكواد غير الفعالة) أو التحدث إلى الأجهزة الخارجية في الأنظمة الآنية حيث يمكن أن تكون مشاكل التوقيت حاسمة، على سبيل المثال أنظمة التحكم في حركة المرور أو أنظمة توجيه الصواريخ.

قابلية الصيانة

قابلية الصيانة هي السهولة التي يمكن بها تعديل البرنامج من قبل المطورين الحاليين أو المستقبليين من أجل إجراء صيانة **تصحيحية** أو **تحسينية** أو **تكيفية**. على سبيل المثال، سيكون من الأسهل تعديل أو تغيير الكود المكتوب بطريقة قابلة للصيانة، وهذا سيوفر الوقت والمال في التطوير. هناك دائمًا مقايضة يجب القيام بها بين كتابة الكود بكفاءة لتقليل تكاليف التطوير الأولية والنظر في المدة التي ستستغرقها الصيانة المستقبلية.

قابلية النقل

قابلية النقل هي مقياس لعدد منصات الحاسوب المختلفة التي يمكن أن يستهدفها كود المصدر، مثل الأجهزة وأنظمة التشغيل وما إلى ذلك. بعض لغات البرمجة، مثل لغة C، تتميز بقابليتها العالية للنقل، حيث تحتوي على مترجمات تقوم بتحويل الكود الحالي إلى كود الآلة المستهدف على العديد من المنصات المختلفة. على سبيل المثال، قد يقتصر تنفيذ برنامج مكتوب باستخدام Microsoft Visual Basic .NET على نظام تشغيل Microsoft، بينما يمكن نقل البرامج المكتوبة بلغة C لاستخدامها في أنظمة تشغيل Microsoft أو Linux مع تغييرات قليلة في الأكواد الخاصة بالمنصة.

المهارات

- مهارات التحليل واتخاذ القرار

مناقشة

في مجموعات صغيرة، شارك تجاربك في مواجهة مشاكل مع تطبيقات البرامج غير الموثوق بها. هل تعتقد أن هذه المشكلات كان من الممكن منعها؟ لماذا تعتقد أن هذه المشكلات لم تُحل قبل إصدار التطبيق؟

المصطلحات الرئيسة

الصيانة التصحيحية – إصلاح خطأ أو خلل تم تحديده.

الصيانة التحسينية – إجراء تحسين على برنامج يعزز من أدائه.

الصيانة التكيفية – إجراء تعديلات على البرنامج، عن طريق إضافة الوظائف أو تغييرها أو إزالتها لتعكس الاحتياجات المتغيرة.

الموثوقية

يتم تحديد موثوقية التطبيق من خلال الدقة العامة والاتساق في مخرجاته عبر عمليات تشغيل متعددة. الاتساق مهم بشكل خاص في اختبار الموثوقية. قد يحسب البرنامج الإجابات بدقة كبيرة، ولكن إذا لم يتمكن من تكرار ذلك في كل مرة يتم تشغيله، فإنه لا يعتبر موثوقًا. هذا أكثر أهمية في البيئات التي تعتمد فيها سلامة الأشخاص على موثوقية البرنامج، مثل أنظمة السلامة في محطة الطاقة أو اكتشاف الاصطدام في أنظمة التحكم في حركة الطائرات.

المتانة

المتانة هي مقياس لجودة كود البرنامج، خاصة عندما يتم اختبار البرنامج لضمان أن البيانات المتطرفة والخاطئة يمكن معالجتها دون التسبب في تعطل البرنامج.

قابلية الاستخدام

قابلية الاستخدام هي مقياس لمدى سهولة استخدام التطبيق. إذا تم اتباع مبادئ التصميم الذي يركز على المستخدم User-centred Design (UCD)، فسيشارك المستخدمون طوال عملية التصميم. سيساعد تقييمهم المستمر على تحسين التصميم عدة مرات قبل قبوله بالكامل. تسمح التقنيات الشائعة، مثل التطوير السريع للتطبيقات Rapid Application Development (RAD)، للمطور بتغيير مخططات التخطيط والتدفق والتفاعل للتطبيق بسرعة لتلقي تعليقات واقعية على الواجهة من المستخدم. على الرغم من أن هذا الشكل من النماذج الأولية السريعة يفتقر عادةً إلى الوظائف الكاملة للمنتج النهائي، إلا أنه يوفر للمستخدم المستهدف 'مظهرًا وإحساسًا' واقعيًا إلى حد ما للتطبيق في وقت مبكر من عملية التطوير.

تتضمن قابلية الاستخدام مدخلات البرنامج والتنقل ومعايير الإخراج. يمكن أن يكون للمظهر العام للتطبيق أيضًا تأثير في هذا. على سبيل المثال، قد تتأثر سهولة الاستخدام إذا كان اختيار الألوان أو الخطوط سيئًا. يجب أيضًا مراعاة إمكانية الوصول للمستخدمين ذوي الإعاقات أو الاحتياجات الخاصة عند التفكير في سهولة الاستخدام، خاصة للمستخدمين الذين يعانون من إعاقات جسدية أو بصرية أو سمعية والذين قد يستخدمون **التقنيات المساعدة**. يمكن للدراسة الأوسع للتفاعل بين الإنسان والحاسوب Human-computer Interaction (HCI) أيضًا أن تساعد في تصميم واجهات التطبيق، بما في ذلك كيفية تأثير سيكولوجية المستخدم، وخاصة سلوكه، على استخدامه للتطبيق.

مناقشة

لقد رأينا بالفعل بضعة أمثلة حيث تكون الموثوقية أمرًا بالغ الأهمية. ناقش مع زميل لك واكتب ثلاثة أمثلة أخرى للحالات التي يجب أن يكون فيها البرنامج موثوقًا تمامًا.

المصطلح الرئيس

التقنيات المساعدة – الأجهزة أو البرامج المصممة لمساعدة المستخدمين الذين يعانون من إعاقة معينة أو احتياجات خاصة، مثل قارئات الشاشة للمستخدمين ذوي الإعاقات البصرية أو صعوبات في التعلم.

وقفة للتفكير هل يمكنك شرح نتاج التعلم؟ ما العناصر التي وجدتها أسهل؟

تلميح دون النظر إلى نص الدراسة هذا، قم بعمل قائمة بميزات مهارات التفكير الحاسوبي ومبادئ برمجة الحاسوب.

توسيع الأفق كيف يؤثر نوع لغة البرمجة في تطوير حل برمجي؟

تمرين تقييمي 4.1 A.P1, A.P2, A.P3, A.M1, A.D1

تطلق مدرسة محلية حملة للترويج للبرمجة. لقد طُلب منك كتابة منشور مدونة مناسب للطلاب يشرح ويعزز تطبيق مهارات التفكير الحاسوبي عند حل المشكلات وتطوير تطبيقات البرامج.

لجعل منشور المدونة مفيدًا قدر الإمكان لجمهورك المستهدف، يجب عليك شرح مبادئ برمجة الحاسوب، خاصة كيفية استخدام اللغات المختلفة لإنشاء حلول. يجب عليك أيضًا مناقشة المبادئ الأساسية لتصميم البرامج وكيفية تطبيقها لإنتاج البرامج عالية الجودة التي يحتاج إليها جمهورك المستهدف ويستخدمها يوميًا.

تمرين تقييمي 4.1 *متابعة*

اختتم منشور مدونتك بتحليل وتقييم تأثير التفكير الحسابي في تصميم البرمجيات وجودة البرمجيات المنتجة.

في أثناء كتابة منشور مدونتك، فكر بعناية في جمهورك المستهدف ومستوى معرفتهم بالبرمجة. فكر في كيفية شرح المفاهيم التقنية لجمهور قد يحتوي على كل من الخبراء والمبتدئين.

التخطيط

- ما المهمة؟ ما المطلوب مني فعله؟
- ما مدى الثقة التي أشعر بها في قدرتي على إنجاز هذه المهمة؟
- هل هناك أي جوانب قد أواجه صعوبة فيها؟
- هل أستخدم لغة مناسبة للفئة المستهدفة؟
- هل أفهم الفرق بين التحليل والتقييم؟

التنفيذ

- أعرف ما أفعله وما أريد تحقيقه.
- يمكنني تحديد أين أخطأت وتعديل تفكيري أو نهجي لإعادة نفسي إلى المسار الصحيح.

المراجعة

- أستطيع شرح المهمة وكيف تعاملت معها.
- يمكنني شرح كيف سأتعامل مع العناصر الأكثر صعوبة بشكل مختلف في المرة القادمة (أي ما سأفعله بشكل مختلف).

ب تصميم حل برمجي لتلبية متطلبات العميل

المهارات

- اختيار أدوات وأنظمة تكنولوجيا المعلومات المناسبة لتطوير حلول البرمجيات
- مهارات الإدارة الذاتية والتخطيط

النقطة الأساسية لأي حل برمجي هي أنه يلبي متطلبات العميل. إذا لم تحقق ذلك بالفعل أو لم تتمكن من تحقيقه، فلا يهم مدى جاذبية الحل أو سهولة استخدامه أو فعاليته: فأنت لم تحل بالفعل مشكلة عميلك.

تأكد من أنك تفهم حقًا متطلبات عميلك قبل أن تبدأ. بمجرد أن تبدأ في تصميم وتطوير الحل، يجب عليك أيضًا العودة إلى متطلبات العميل الأصلية على فترات منتظمة. سيساعدك هذا في الحفاظ على تطوير البرمجيات الخاصة بك على المسار الصحيح لتحقيق النجاح.

دورة حياة تطوير البرمجيات

المصطلح الرئيس

النموذج المفاهيمي – طريقة لتنظيم الأفكار والمفاهيم بطريقة منطقية. غالبًا ما تمثل النماذج المفاهيمية الأفكار المعنية بطريقة مرئية توضح العلاقات بينها بطريقة بسيطة يسهل فهمها.

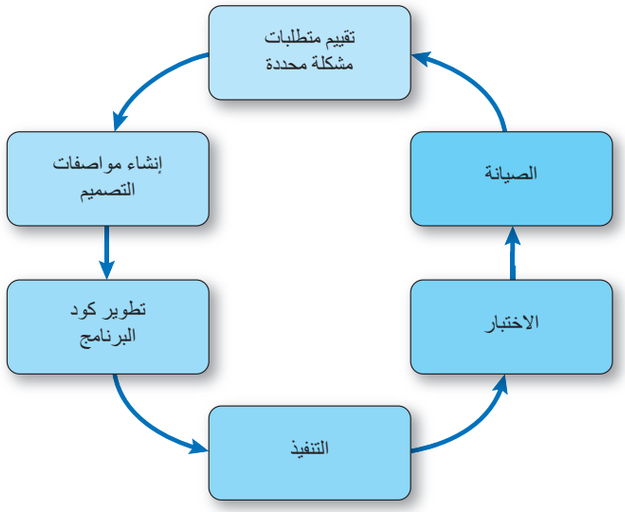
تصميم حل برمجي هو عملية دورية ذات مراحل محددة بوضوح. على الرغم من وجود العديد من الإصدارات المختلفة لدورة حياة تطوير البرمجيات وتغير الأسماء، إلا أن معظم المراحل شائعة وتتبع الترتيب المنطقي نفسه.

دورة حياة تطوير البرمجيات (SDLC) هي **نموذج مفاهيمي**. إنه يصف المراحل المستخدمة لإدارة إنشاء حل برمجي من بدايته إلى صيانته المستمرة وتقاعده النهائي أو استبداله. يتم عرض دورة حياة تطوير البرمجيات النموذجية في الشكل 4.26 في الصفحة الآتية.

المصطلحات الرئيسة

حالة الاستخدام – قائمة بالإجراءات أو الأحداث المحددة التي تحدث بين المستخدم والبرنامج. تشمل حالات الاستخدام المحتملة التي تحدث عندما يحاول العميل سحب النقود من جهاز الصراف الآلي "رفض البطاقة" و"الرقم السري صحيح" و"الرقم السري غير صحيح" و"ابتلاع البطاقة" و"صرف النقود" و"عدم توافر نقود".

جدول التتبع – جدول يتتبع المدخلات والعمليات والمخرجات لكل حالة استخدام. يتضمن نتيجة متوقعة (ما يجب أن يحدث) ونتيجة فعلية (نتيجة البرنامج)، والتي يمكن مقارنتها ودراستها لتحديد النتائج غير المتوقعة.



الشكل 4.26 نموذج مفاهيمي لدورة حياة تطوير البرمجيات – هناك تداخل كبير بين تطوير كود البرنامج والتنفيذ

توضح العملية الآتية خطوة بخطوة كيفية تطبيق هذه الخطوات الست في تطوير البرمجيات.

خطوة بخطوة: دورة حياة تطوير البرمجيات (SDLC)

6 خطوات

1 تقييم متطلبات مشكلة محددة

يجب أن تفهم تمامًا متطلبات العميل لحل مشكلة محددة. إذا لم يكن لديك فهم واضح لما يجب تحقيقه، فسيكون من الصعب أو حتى المستحيل إنشاء حل عملي.

عادةً ما يتم استخلاص متطلبات حل المشكلة المحددة من موجز العميل. عندما تكون التفاصيل غير كافية أو تكون هناك حاجة إلى توضيح، يجب عليك التحقيق لفهم ما يريد العميل تحقيقه من البرنامج. قد يتضمن ذلك عددًا من تقنيات التحقيق المختلفة بما في ذلك:

- الاستجواب المباشر للمستخدمين أو الموظفين حول المهام التي يؤدونها وأنماط عملهم وأي صعوبات يواجهونها في وظائفهم
- مراقبة العملية اليدوية التي تتم حوسبتها، مثل ملء نموذج استفسار العميل أو حساب تكلفة الطلب
- فحص المستندات اليدوية التي تحتاج إلى حوسبة. قد يشمل ذلك نماذج جمع البيانات مثل نماذج الطلبات والجداول الزمنية أو قوائم الأسهم والمخرجات مثل التقارير أو الرسوم البيانية.

بمجرد تحديد المتطلبات بوضوح، من الممكن تحديد ما يكون ضمن نطاق المشروع وما لا يكون ضمن النطاق. من هذا، يمكنك إنشاء مواصفات التصميم.

2 إنشاء مواصفات التصميم

يجب أن تتضمن مواصفات التصميم الكاملة العناصر الآتية.

- نطاق المشروع: هذا هو ما يغطيه الحل المقترح الخاص بك ويمكن أيضًا تسميته "مجال المشكلة". يساعد تحديد نطاق المشروع مقدمًا مطور البرامج على البقاء على المسار الصحيح وتجنب "زحف الميزات" - وهذا يعني إضافة ميزات غير مطلوبة في البداية، والتي من المحتمل أن تؤخر تنفيذ الحل.

خطوة بخطوة: دورة حياة تطوير البرمجيات (SDLC) *متابعة*

6 خطوات

- المدخلات: هي القيم التي يتم إدخالها في البرنامج وكيف سيتم إدخال هذه القيم (على سبيل المثال يدويًا أو تلقائيًا من قاعدة بيانات).
- المخرجات: هي المعلومات التي تم إنشاؤها بواسطة البرنامج. يجب عليك تضمين تفاصيل حول تنسيقها وتخطيطها.
- واجهة المستخدم: هي الطريقة التي سيتفاعل بها المستخدم مع البرنامج (مثل أنظمة القوائم واستخدام لوحة المفاتيح والفأرة والتحكم الصوتي وأنظمة الألوان والتخطيط).
- العمليات والخوارزميات: تشمل العمليات الحسابية التي يتم إجراؤها من أجل توليد المخرجات المرغوبة من البيانات التي يتم إدخالها والتحقق من صحة البيانات وما إلى ذلك.
- الجدول الزمني: هذا هو الوقت الذي ستستغرقه العملية، بما في ذلك الأهداف التي يجب إنجازها خلال مدة المشروع. يجب أن تتضمن أيضًا "المعالم" المتفق عليها، وهي نقاط محددة مسبقًا في العملية التي يمكن للمطور من خلالها إثبات مدى التقدم الذي تم إحرازه.

3 تطوير كود البرنامج

يجب كتابة كود البرنامج باللغة التي اخترتها، على الرغم من أن العميل قد يحتاج أحيانًا إلى لغة معينة. سيعتمد الاختيار على مواصفات التصميم والمتطلبات المحددة للحل. كما رأيت بالفعل، هناك العديد من العوامل المختلفة التي ستؤثر في اختيارك للغة البرمجة.

يجب أن توفر المدخلات والمخرجات والعمليات والخوارزميات وواجهة المستخدم الموثقة المطلوبة لمطور البرامج رؤية واضحة جدًا لماهية البرنامج وكيف يجب أن يعمل. ويجب على المطور أخذ مواصفات التصميم واستخدام لغة البرمجة المختارة بأكبر قدر ممكن من الكفاءة لإنشاء كود برنامج العمل، باستخدام الميزات الأكثر ملاءمة للغة المحددة.

في البيئة التجارية، يتم تحقيق ذلك عادةً باستخدام خطوات منفصلة، تُعرف باسم "نقاط القصة". هذه تقسم الأجزاء المختلفة من البرنامج الذي يتم العمل عليه إلى ساعات عمل، حيث تعادل ساعة واحدة عادةً نقطة قصة واحدة. هذا يعني أن الميزات الأكثر تعقيدًا ستعادل المزيد من نقاط القصة.

ينتج مطورو البرامج التجارية عمومًا كود البرنامج على أساس 40/60، ما يعني أنهم يقضون 60 في المائة من وقتهم في كتابة الكود و40 في المائة في التخطيط وتصحيح الأخطاء والاختبار.

4 التنفيذ

يتضمن التنفيذ:

- اختيار لغة البرمجة الأكثر ملاءمة (إذا لم يكن العميل قد اختارها بالفعل)
- اختيار بيئة التطوير - أداة (أدوات) البرنامج المستخدمة لبناء البرنامج ونظام التشغيل وأحيانًا نوع الأجهزة
- كتابة أكواد الحل من مواصفات التصميم
- تصحيح الكود للتأكد من تحديد الأخطاء البسيطة وإزالتها قبل بدء الاختبار الرسمي.

5 الاختبار

يعد الاختبار عملية أساسية تضمن أن أي برنامج يلبي تمامًا متطلبات العميل ويعمل بطريقة دقيقة وموثوقة وقوية.

هناك نوعان شائعان من الاختبارات: اختبار الصندوق الأبيض واختبار الصندوق الأسود.

- عادةً ما يتم إجراء اختبار المربع الأبيض بواسطة المطور الذي أنتج البرنامج. يتضمن تتبع **حالات الاستخدام** من خلال منطق كود البرنامج وإكمال **جداول التتبع**.
- يتم إجراء اختبار الصندوق الأسود بواسطة مستخدم (أو مختبر داخلي ليس لديه وصول إلى الكود) وفقًا لحالة الاستخدام. لا يتعرض المستخدم لكود البرنامج ولا يحتاج إلى معرفة كيفية عمل البرنامج. أنت، بصفتك مطورًا، تهتم فقط بالنتيجة التي يحصل عليها المستخدم.

يتيح الاختبار تحسين البرنامج (جعله أكثر دقة أو تحديدًا) وتحسينه (جعله يعمل بشكل أسرع أو أكثر كفاءة).

خطوة بخطوة: دورة حياة تطوير البرمجيات (SDLC) *متابعة*

6 خطوات

6 الصيانة

- الصيانة هي عملية مستمرة، ويؤدي ذلك إلى تصحيح كود البرنامج بناءً على الاختبار و/أو ملاحظات المستخدم.
- قد تعمل الصيانة على تكييف الحل الأصلي لتلبية احتياجات العملاء المتغيرة. على سبيل المثال، إذا كان العميل يتطلب في الأصل المعلومات المالية بعملة واحدة، وأصبح الآن يحتاج إليها بعدة عملات، فسيكون ذلك تعديلًا على الكود الحالي.
- وبدلًا من ذلك، قد تعمل الصيانة على توسيع الحل من خلال تضمين وظائف إضافية. على سبيل المثال، إذا طلب العميل خيارات قائمة جديدة تمامًا لتلبية متطلبات جديدة تمامًا، فسيكون هذا توسعًا لتضمين وظائف إضافية.

المهارات

- مهارات التحليل واتخاذ القرار
- التواصل الكتابي الرسمي
- اختيار أدوات وأنظمة تكنولوجيا المعلومات المناسبة لتطوير حلول البرمجيات

نصائح

تذكر أنه قد يكون هناك أكثر من حل واحد يجب مراعاته لمشكلة واحدة.

المصطلحات الرئيسة

القيود – القيود المفروضة على شيء ما. تشمل القيود في سياق البرمجة ميزات لغات البرمجة والمهارات التقنية للمطور والمنصات التي تدعمها لغة البرمجة وما إلى ذلك.

السياق – الإعداد أو الظروف المحيطة بشيء ما. في تصميم حلول برمجية، سيتضمن السياق تفاصيل مثل الخلفية التاريخية للمشكلة.

قد تتكرر العملية الكاملة لدورة حياة تطوير البرمجيات مرارًا وتكرارًا. أداء أي حل مبرمج سيتراجع في النهاية بمرور الوقت مع تغير احتياجات العميل أو ظروف العمل. يمكن أن تؤدي الصيانة النشطة إلى إطالة عمر البرنامج، ولكن في النهاية، ستتم إعادة تنفيذ دورة حياة البرنامج وسيتعين تقييم متطلبات الحل المحسن الجديد.

تصميم حلول برمجية

تصميم أو مواصفات حلول البرمجيات هو وثيقة رسمية يتم إنجازها من قبل المطورين (بمساهمة من العملاء و/أو المستخدمين المستهدفين) قبل بدء البرمجة الفعلية. يجب أن يتضمن تحليلًا كاملًا لعملية حل المشكلة التي تم اتباعها وجميع الحلول المقترحة. إذا كُتبت بشكل صحيح، يجب أن توفر تفاصيل كافية للمبرمج الذي ليس لديه دراية بالمشكلة لبناء التطبيق المطلوب باستخدام الحل المفضل.

جمل تعريف المشكلة

جمل تعريف المشكلة هي عنصر أساسي في تصميم حلول البرمجيات. إنها أوصاف واضحة للمشكلات الموجودة، والأشخاص المتأثرين بالمشكلات و**القيود** التي قد تؤثر في الحل. عادةً ما يتم إنشاؤها في أثناء التحقيق الأولي والمناقشة مع العملاء من خلال طرح الأسئلة الخمسة: من، ماذا، متى، أين ولماذا؟ فهي تساعد المطورين على فهم المشكلة ونطاقها وقيودها. وبهذا تساعد المطورين على التركيز عند حل المشكلة.

أسئلة 'ماذا؟' و'متى؟' و'أين؟' ستساعدك في إنشاء ملخص كامل للمشكلة التي يجب حلها وتعقيدها، بالإضافة إلى فوائد وقيود أي حلول مقترحة.

- ما **سياق** هذه المشكلة؟
- ما طبيعة المشكلة التي يجب حلها؟
- ما حدود أو نطاق المشكلة؟
- ما المتطلبات التي يحددها العميل؟
- ما القيود التي تحد من الحل الذي يمكن تطويره؟
- ما فوائد حل هذه المشكلة؟
- ما تأثير عدم حل هذه المشكلة؟
- ما الغرض من الحل المطلوب؟
- ما تعقيدات المشكلة التي نحتاج إلى حلها؟
- ما طبيعة التفاعل بين المستخدمين المستهدفين والحل؟
- ما مستوى التعاون الذي سيحصل عليه مصممو البرمجيات من المستخدمين والعملاء الحاليين عند حل المشكلة؟

- ما الموارد المتاحة لحل هذه المشكلة؟
- متى تحدث المشكلة؟ هل يمكن عزل توقيتها؟ هل هناك نمط يمكن تعرُّفه؟
- متى يجب تشغيل هذا الحل؟
- أين تحدث المشكلة؟ هل يمكن عزل موقعها؟ هل هناك نمط يمكن تعرُّفه؟
- أين الحل الذي سيتم نشره؟

ستساعد الأسئلة من نوع 'من؟' في تحديد المستخدمين المستهدفين لحل برمجي وطبيعة التفاعل الذي سيكون لدى المستخدمين مع الحل البرمجي.

- من المتأثر بهذه المشكلة؟
- من يريد حل هذه المشكلة؟

ستساعد الأسئلة من نوع 'لماذا؟' في تحديد فوائد الحل البرمجي بشكل أكبر.

- لماذا تحدث هذه المشكلة؟ هل يمكن عزل سببها؟ هل هناك نمط يمكن تعرُّفه؟
- لماذا يجب حل هذه المشكلة؟

مثال عملي: كتابة جملة تعريف المشكلة باستخدام سؤال "ماذا؟"

تاجر مواد البناء لديه مشكلة في تكامل أنظمة المتجر والأنظمة عبر الإنترنت. يسأل المطور العميل، "ما طبيعة المشكلة التي يجب حلها؟"

هذه هي جملة تعريف المشكلة الذي ينشئه المطور بناءً على إجابات العميل على سؤال "ماذا؟" سؤال:

قد يكون لدى العميل حسابان تجاريان مختلفان يمكنه استخدامهما داخل فرع المتجر لشراء البضائع على الائتمان. ومع ذلك، عندما يزور متجرنا على الإنترنت، يمكنه فقط استخدام النوع الجديد من الحسابات – ببساطة لا يتم تعرف أنواع الحسابات القديمة وقد تلقينا شكاوى في هذا الصدد. نحتاج إلى السماح للعميل بتحديد الحساب الذي يريد استخدامه لشراء بضائعنا عن طريق اختيار الحساب الصحيح من القائمة المتاحة على الشاشة.

جملة تعريف المشكلة هذه واضحة ومختصرة، وتصف المشكلة الدقيقة التي يجب حلها باستخدام المصطلحات الصحيحة لقطاع العميل.

تطبيق النظرية

باستخدام المثال العملي، اكتب جمل تعريف المشكلة التي تجيب على هذه الأسئلة.

1 ما فوائد حل هذه المشكلة؟

2 ما تأثير عدم حل هذه المشكلة؟

ميزات البرنامج

يعد وصف ميزات حل البرنامج المقترح جزءًا أساسيًا من تصميم الحل. يتضمن ذلك مهام البرنامج الرئيسة، وأي تخزين مطلوب للبيانات والمدخلات والمخرجات المطلوبة.

المهام الرئيسة للبرنامج وتنسيقات الإدخال والإخراج

عند النظر في قيم الإدخال والإخراج المطلوبة، يجب عليك أيضًا التفكير في التنسيقات المطلوبة. على سبيل المثال، تم إطلاع أحد المطورين على إنشاء برنامج بسيط للتحويل بين ثلاث عملات اعتمادًا على اختيار

المستخدم. على سبيل المثال أي ثلاث عملات من الروبية أو اليورو أو الدرهم أو البات أو الليرة أو الريال أو الدينار. قام المطور بإنشاء تمثيل مرئي للمدخلات المطلوبة والعمليات والمخرجات وتخزين البيانات، كما هو موضح في الشكل 4.27 (العملات النموذجية هنا هي الروبية واليورو والدينار).

المُدخل المطلوب	العمليات المطلوبة
قيمة الروبية (9.99، رقم عشري بخانتين عشريتين) اختيار العملة المستهدفة (حرف واحد - تم التحقق من صحته إلى D أو E)	قيمة الروبية المدخلة اختيار عملة المدخلات التحقق من صحة الاختيار إلى D أو E تحويل الروبية إلى الدرهم أو اليورو، بحسب اختيار المستخدم مُخرج المبلغ المحول
المخرج المطلوب	**تخزين البيانات المطلوب**
قيمة العملة المحولة إما إلى الدرهم أو اليورو بحسب اختيار المستخدم (9.99، رقم عشري بخانتين عشريتين)	لا يوجد

الشكل 4.27 مخطط رباعي يوضح المدخلات والمخرجات والعمليات

قام المطور بتحديد جميع المدخلات والمخرجات وذكر صيغها، خاصة عدد الأماكن العشرية التي يجب استخدامها. كما قام بتضمين تفاصيل أي تحقق مطلوب.

نصائح

عند تحديد التنسيقات، من المعتاد استخدام 9 لرقم عددي وA لرمز أبجدي وX لأي رمز.

الرسوم التوضيحية التخطيطية

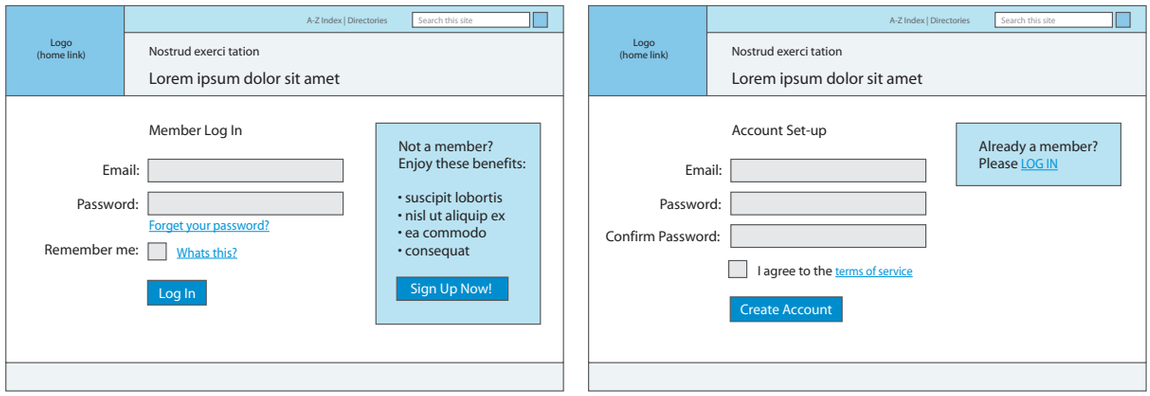
سيحتوي تصميم الحلول بالتأكيد على عدد من المخططات المختلفة، أو الرسوم التوضيحية التخطيطية. هناك ثلاثة أنواع شائعة يمكن تضمينها، يركز كل منها على جوانب من **تجربة المستخدم (UX)**. وهي كالآتي:

- تخطيطات الشاشة، توضح كيفية تنظيم العناصر على "الصفحة" الافتراضية
- واجهات المستخدم، والتي توضح كيفية تفاعل المستخدم مع التطبيق
- عناصر التنقل، توضح كيف سينتقل المستخدم بين "الصفحات" الافتراضية المختلفة.

المصطلح الرئيس

تجربة المستخدم (UX) – مقياس لكيفية تفاعل المستخدم مع البرنامج ورضاه عند استخدامه.

يمكن رسم هذه الرسوم التوضيحية (انظر الشكل 4.28) على الورق أو إنشاؤها إلكترونيًا باستخدام أدوات تصميم متخصصة، والعديد منها متاح عبر الإنترنت.



الشكل 4.28 تخطيطان محتملان للشاشة لتطبيق واحد

الجدول 4.12 مزايا وعيوب الرسوم البيانية التوضيحية

المزايا	العيوب
يمكن تسريع عملية الإنشاء عندما يتم إنشاء الرسوم التوضيحية باستخدام أدوات التصميم الإلكترونية.	قد تتطلب أدوات التصميم الإلكتروني التسجيل أو الشراء.
يمكن أن تكون سهلة الإنشاء وسريعة التغيير، خاصة إذا تم إنشاؤها إلكترونيًا.	يمكن أن يكون مستهلكًا للوقت إذا كان يجب تكرارها من خلال عدد من مراجعات التصميم.
السماح للمصمم بإرشاد المستخدمين المحتملين عبر مسار التطبيق. قد يساعد ذلك في تحديد مواضع الارتباك أو الصعوبة في أثناء تحسين التصميم.	يمكن أن يكون من الصعب تكرار المظهر على الشاشة بدقة.
السماح للعميل بمشاهدة النماذج الأولية للتجربة المقصودة للمستخدم قبل بدء كتابة الكود.	
تحسين مشاركة العملاء من خلال إشراكهم في عملية التصميم. هذا يعني أن التطبيق النهائي لديه أفضل فرصة ممكنة لتلبية متطلباتهم بالكامل.	
السماح للمصممين بتجربة أفكار مختلفة والحصول على تعليقات حول أفكارهم.	
توفير وقت التطوير، حيث لا يضطر المطورون إلى تعديل كود البرنامج بشكل متكرر، خاصة وأن التعديلات قد تؤدي إلى حدوث أخطاء.	

الخوارزميات ومراحل المعالجة

الخوارزمية هي مجموعة من التعليمات التي يتم اتباعها لحل مشكلة أو تنفيذ مرحلة معينة من المعالجة في الحل الشامل، مثل التحقق من صحة مدخلات المستخدم. قد تتكون تطبيقات البرامج من العديد من الخوارزميات المختلفة، التي يتم تنفيذها في كود البرنامج باستخدام مزيج من العديد من تركيبات البرمجة والوظائف والإجراءات المختلفة.

لأنها قد تكون صعبة في التواصل، يمكن تمثيل الخوارزميات باستخدام عدد من أدوات التصميم المختلفة، بعضها يقدم توضيحًا بيانيًا.

الأدوات الثلاثة الأكثر شيوعًا هي الكود الزائف والمخططات الانسيابية ومخططات الأحداث أو الجداول.

موضوعات ذات صلة

تمت مناقشة تخطيطات الشاشة والتنقل لتطبيقات الأجهزة المحمولة (تقنية تسمى التخطيط الشبكي) في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

الكود الزائف

الكود الوهمي هو مخطط غير رسمي للخوارزمية، ويُعبّر عنه بلغة طبيعية. يمكن تحويله إلى لغة البرمجة المستهدفة. على سبيل المثال، يوضح الشكل 4.29 عملية تحقق بسيطة من عمر مستخدم يتراوح بين 18 و60. يجب ألا يحتوي الكود الوهمي على أي أوامر أو بناء جملة للغات البرمجة.

موضوعات ذات صلة

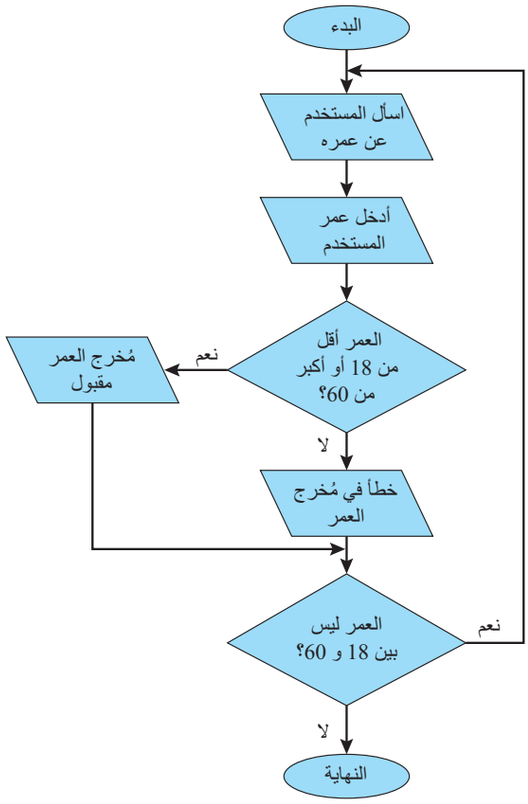
تمت مناقشة الخوارزميات المستخدمة لتصميم تطبيقات الأجهزة المحمولة في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

```
Do
   Ask user for their age
   Input user's age
   If age is less than 18 or greater than 60 then
      Output age error
   Else
      Output age accepted
   Endif
While age is not between 18 and 60.
```

الشكل 4.29 التحقق البسيط من عمر المستخدم

المخططات الانسيابية (Flowcharts)

المخطط الانسيابي هو تمثيل بياني للخوارزمية، يوضح إجراءاتها ومنطقها من خلال مجموعة من الرموز الموحدة. يعرض الشكل 4.30 الخوارزمية نفسها الواردة في الشكل 4.29 في الصفحة السابقة، ولكن يتم تمثيلها في مخطط انسيابي.



يعرض الشكل 4.30 التحقق من عمر المستخدم معبر عنه كمخطط انسيابي

مخططات الأحداث

مخططات الأحداث أو جداول الأحداث تُستخدم لتسجيل الأحداث في جدول بسيط. لكل حدث، يتم تضمين التفاصيل الآتية في الجدول:

- اسم الحاوية التي يوجد بها الكائن المستجيب (مثل Form1)
- اسم الكائن الذي يستجيب للحدث (مثل Button1)
- الحدث الذي يتم الرد عليه (مثل النقر)
- معالج الحدث الذي يتم تشغيله بواسطة المراقب (مثل button1Action()).

يمكن أن تعمل جداول الأحداث كقائمة تحقق مفيدة عند بناء مكونات واجهة المستخدم الرسومية لتطبيق.

بحث

على الرغم من أنه يمكن إنشاء المخططات الانسيابية يدويًا، إلا أن هناك العديد من الأدوات المجانية عبر الإنترنت التي تدعم إنشائها بطريقة "السحب والإسقاط" سهلة الاستخدام.

هياكل البيانات

يجب أن يوضح أي حل برمجي الهياكل المختلفة التي يمكن استخدامها كجزء من الخوارزمية. وهي تختلف عن تخزين البيانات حيث يتم تخزين هياكل البيانات في ذاكرة الوصول العشوائي في أثناء تنفيذ التطبيق.

تشمل هياكل البيانات الشائعة المصفوفات (أحادية البعد، ثنائية البعد، وثلاثية البعد)، الطوابير، المكدسات والسجلات.

تخزين البيانات

تصميم أي حل برمجي يتضمن التعامل مع استمرارية وجود البيانات. يُقال إن تخزين بيانات الحاسوب في ذاكرة الوصول العشوائي (RAM) "متقلب" أو غير مستقر لأنه يُفقد عند فصل الطاقة، على سبيل المثال عند إيقاف تشغيل الحاسوب.

هذا يعني أنه، بينما لا بأس من تخزين البيانات في ذاكرة الوصول العشوائي في أثناء تشغيل البرنامج، تحتاج إلى شكل أكثر ديمومة (غير متطاير) من تخزين البيانات لتمكين البيانات من الوجود بين استخدامات البرنامج، خاصة عند إزالة الطاقة. تشمل أشكال تخزين البيانات غير المتطايرة التخزين المغناطيسي، مثل القرص الصلب، أو محرك فلاش USB.

أكثر أشكال تخزين البيانات غير المتطايرة شيوعًا هو ملف البيانات. وهي خاصية مدعومة من قبل معظم لغات البرمجة. على الرغم من أن ملف البيانات يخزن البيانات المستخدمة من قبل البرنامج على وسائط غير متقلبة، إلا أنه لا يخزن تعليمات البرنامج نفسه. يتم الاحتفاظ بهذه في ملفات منفصلة.

قد توجد ملفات البيانات في العديد من التنسيقات المختلفة. تشمل الأمثلة الشائعة:

- الرمز القياسي الأمريكي لتبادل المعلومات (ASCII) أو النص
- ثنائي (حيث يتم تخزين الأرقام كقيمة ثنائية نقية وليس بوصفها أكواد رموز ASCII فردية)
- csv: قيمة مفصولة بفاصلة.
- XML (لغة الترميز القابلة للتوسع).

يوضح الجدول 4.13 كيف يمكن تحديد بيانات العميل نفسها في ASCII وCSV وXML للمقارنة.

الجدول 4.13 مقارنة بين تنسيقات الملفات المختلفة المستخدمة لتخزين البيانات البسيطة

تنسيق ملف البيانات	مثال على تخزين البيانات	ملاحظات

ASCII	`CustomerAcc  Lastname Firstname Location`
	`000000001  Jones  Alex  London`
	`000000003  Willis  Claire  Swindon`
		يتم استخدام الصف الأول للعناوين. يتم استخدام علامة تبويب لفصل كل قيمة. يفصل سطر جديد بيانات كل عميل.

CSV	`CustomerAcc,Lastname,Firstname,Location`
	`000000001,Jones,Alex,London`
	`000000003,Willis,Claire,Swindon`
		يتم استخدام الصف الأول للعناوين ويتم فصل كل قيمة بفاصلة. يفصل سطر جديد بيانات كل عميل.

XML	`<?xml version="1.0">`
	`<Customer>`
	`<CustomerAcc>000000001</CustomerAcc>`
	`<Lastname>Jones</Lastname>`
	`<Firstname>Alex</Firstname>`
	`<Location>London</Location>`
	`</Customer>`
	`<Customer>`
	`<CustomerAcc>000000003</CustomerAcc>`
	`<Lastname>Willis</Lastname>`
	`<Firstname>Claire</Firstname>`
	`<Location>Swindon</Location>`
	`</Customer>`
		تبدأ وثيقة XML بعلامة XML توضح إصدارها. يتم 'حظر' كل عميل أو فصله عن الآخرين باستخدام زوج من علامات البدء والنهاية للعميل. يتم وضع كل جزء من البيانات في زوج من العلامات الافتتاحية والختامية التي تصف السمة، على سبيل المثال CustomerAcc.

غالبًا ما يتأثر اختيار تنسيق ملف البيانات باستخدامه. على سبيل المثال، إذا كان من المقرر استيراد ملف بيانات إلى تطبيق جدول بيانات (مثل Microsoft Excel)، فإن CSV هو خيار شائع. يمكن استخدام ملفات البيانات لأغراض عديدة، مثل تخزين سجلات المستخدمين، أو معلومات تكوين البرنامج أو تفاصيل ترخيصه.

يتطلب تصميم حل برمجي منك النظر في تخزين البيانات بالطرق الآتية:

- ما متطلبات تخزين البيانات لحل المشكلة (مثل بيانات العملاء وبيانات المنتجات)؟
- ماذا يجب أن تسمى الملفات؟
- كيف سيتم الوصول إلى البيانات (مثل القراءة، الكتابة أو **الإلحاق**)؟
- إذا تم تحديث البيانات أو حذفها وما إلى ذلك، كيف سيحدث ذلك؟
- ما تنسيق الملف الذي سيتم استخدامه (مثل ASCII أو CSV أو XML)؟
- أين يجب تخزين الملف (مثل اسم الوسائط والمجلد)؟

قد تتطلب الحلول الأكثر تعقيدًا العديد من ملفات البيانات لكي تعمل بشكل صحيح. في بعض الحالات، خاصة عند البحث عن الوظائف وعندما تكون هناك حاجة إلى علاقات بيانات معقدة، قد يكون من الأنسب استخدام قاعدة بيانات علائقية بدلًا من ملفات البيانات.

المصطلح الرئيس

الإلحاق – الإضافة إلى نهاية شيء ما. في ملف بيانات، يعني إضافة بيانات جديدة إلى النهاية.

هياكل التحكم

تشمل هياكل التحكم التسلسلات والاختيارات والتكرارات. تحتوي خوارزمية التحقق الموضحة في الشكلين 4.29 (الصفحة 39) و4.30 (الصفحة 40) على جميع هياكل التحكم الثلاثة. يجب أن يُظهر الحل البرمجي بوضوح نطاق هياكل التحكم المختارة وكيفية استخدامها لحل المشكلات المعقدة كجزء من خوارزمية. المخططات الانسيابية والكود الزائف هي تقنيات جيدة لاستخدامها في تسليط الضوء على هذه التفاصيل.

نصائح

المصادقة لا تفحص ما إذا كانت البيانات قد تم إدخالها بدقة. هذه عملية منفصلة تسمى الاعتماد.

التحقق من صحة البيانات

ربما سمعت المثل القائل جودة المخرجات هي انعكاس لجودة المدخلات. هذا يصف القاعدة العامة بأن جودة المخرجات تعتمد بشكل مباشر على جودة المدخلات. إذا كانت المدخلات غير مفهومة، فسيكون الناتج أيضًا غير مفهوم. التحقق هو العملية التي تتحقق مما إذا كانت القيمة المدخلة منطقية ومعقولة قبل معالجتها.

استخدام مربعات الاختيار والأزرار وقوائم الاختيار عادة ما يحد من خيارات الإدخال إلى مدخلات معقولة. ومع ذلك، ما تزال معظم البرامج تتطلب التحقق للتعامل مع احتمال وجود مدخلات خاطئة من المستخدم، خاصة عند استخدام مدخلات لوحة المفاتيح التقليدية.

من المهم جدًا بناء قواعد التحقق في الحل، للتحقق مما إذا كانت المدخلات المختلفة منطقية وتمنع النتائج غير الدقيقة، أي **أخطاء وقت التشغيل** أو تعطل التطبيق القاتل.

هناك عدة أنواع مختلفة من التحقق.

المصطلح الرئيس

خطأ وقت التشغيل – مشكلة تحدث في أثناء استخدام التطبيق. تؤدي هذه الأخطاء إلى قفل التطبيق (رفض قبول إدخال المستخدم) أو تعطله (إنهاء المستخدم وإعادته إلى قائمة الجهاز أو سطح المكتب).

فحص النطاق

يُقيّم فحص النطاق ما إذا كانت البيانات المدخلة ضمن نطاق الحد الأدنى إلى الحد الأقصى الصالح. على سبيل المثال، إذا كان العميل مقيدًا بشراء ما يصل إلى عشرة من عنصر معين، فإن النطاق الصالح سيكون من 0 إلى 10. سيتم اعتبار الإدخال خارج هذا النطاق الشامل غير صالح. الشفرة النموذجية لبرنامج C# لإجراء هذا النوع من التحقق موضحة في الشكل 4.31 في الصفحة الآتية.

بالطبع، إدخال 0 سيشير إلى أن العميل لا يحاول شراء أي عدد من العنصر. ومع ذلك، فإنه ما زال إدخالًا صالحًا منطقيًا وسيمر عبر التحقق.

النطاقات لا يجب أن تقتصر على المدخلات العددية. على سبيل المثال، يمكن أن تمثل الرموز 'A' إلى 'F' نطاقًا صالحًا أيضًا.

```
const int MIN = 0;       //minimum value of the range
const int MAX = 10;      //maximum value of the range

string strNumber;        //string to temporarily store our input
int qtyValue;            //our inputted quantity

//perform loop while quantity is outside range
do
{
    //input quantity
    Console.WriteLine("Enter quantity between {0} and {1}:", MIN, MAX);
    strNumber = Console.ReadLine();
    qtyValue = int.Parse(strNumber);

    //check is quantity outside range
    if (qtyValue < MIN || qtyValue > MAX)
    {
        Console.WriteLine("Sorry, the quantity entered is outside the allowed range");
    }

} while (qtyValue < MIN || qtyValue > MAX);

Console.WriteLine("Your quantity of {0} is valid, thank you.",qtyValue);

//@TODO other things with the quantity...

//wait for keypress to continue
Console.ReadKey();
```

الشكل 4.31 روتين التحقق من صحة النطاق المحدد في C#

فحص الطول

فحص الطول يقيّم عدد الرموز التي تم إدخالها، وبعض المدخلات المعروفة جيدًا لها أطوال محدودة؛ على سبيل المثال:

- يتم تقييد رسائل خدمة الرسائل القصيرة (SMS) إلى 160 رمز
- تم تحديد التغريدات في الأصل بعدد 140 رمز (مشتقة من طول الرسائل النصية القصيرة ناقص 20 رمزًا لعنوان المستخدم الفريد).

الشكل 4.32 يظهر مقتطف C# يحد من إدخال الرموز إلى طول أقصى محدد مسبقًا بواسطة ثابت.

```
const int MAX = 10;      //set maximum number of characters

string message;          //our message to input and process
int messageLength;       //our message's length in characters

//perform loop while string too long!
do
{
    //input string
    Console.WriteLine("Enter message (max {0} characters)", MAX);
    message = Console.ReadLine();

    messageLength = message.Length;

    //check its length
    if (messageLength > MAX)
    {
        Console.WriteLine("Sorry, your message of {0} characters is too long.",messageLength);
    }

} while (messageLength > MAX);
```

الشكل 4.32 روتين التحقق من صحة فحص الطول النموذجي في C#

فحص الوجود

فحص الوجود يقيّم ما إذا كانت البيانات موجودة - أي ما إذا كانت موجودة. على سبيل المثال، يجب على المستخدم إدخال ما إذا كان 'ذكر' أو 'أنثى' عند ملء نموذج تسجيل، ويضمن التحقق من التواجد أنهم لم يتركوا الإدخال فارغًا أو غير محدد.

تحقق من النوع

فحص النوع يقيّم ما إذا كانت البيانات المدخلة من النوع الصحيح، كما في الشكل 4.33. على سبيل المثال، إذا كان على المستخدم إدخال عمره، فإن ذلك يتطلب أن يكون الإدخال عددًا صحيحًا (رقمًا كاملًا). إذا لم يمنع المبرمج إدخال أنواع بيانات غير صحيحة، فقد يتسبب ذلك في حدوث خطأ قاتل في أثناء وقت التشغيل.

العمر	أ	(نوع البيانات هو رمز، هذا غير صالح)
العمر	16	(نوع البيانات هو عدد صحيح، هذا صَالِح)
العمر	16/09/01	(نوع البيانات هو تاريخ، هذا غير صالح)

الشكل 4.33 مثال على فحص النوع

فحص التنسيق

فحص التنسيق يقيّم ما إذا كانت البيانات المدخلة في التنسيق الصحيح، على سبيل المثال، يتحقق من أن سلسلة تحتوي على رمز منطقة بريدية في المملكة المتحدة تتبع التنسيق 'PO1 3AX'.

- PO هو المنطقة، مثل GL (غلوستر) - يجب أن تكون هذه الرموز أحرف إنجليزية كبيرة، أحرف أبجدية، أو رمز أو رمزين.
- 1 هو الحي، عادة ما يكون بين 1 و20 لكل منطقة - يجب أن يبلغ هذا الرمز ما يصل إلى رقمين.
- 3 هو القطاع، وعادة ما يغطي ما يصل إلى 3000 عنوان - يجب أن يكون هذا رقمًا واحدًا.
- AX هي الوحدة، وعادة ما تغطي ما يصل إلى 15 عنوانًا - يجب أن تكون هذه الرموز أحرف إنجليزية كبيرة، أحرف أبجدية، مكونة من رمزين.

سيتم تطبيق فحص التنسيق على رمز المنطقة البريدية وفقًا لهذه القواعد، كما هو موضح في الشكل 4.34.

الرمز البريدي؟	GL2 4TH	(الرمز البريدي صالح)
الرمز البريدي؟	W1A 1AA	(الرمز البريدي صالح)
الرمز البريدي؟	GL2 24TH	(الرمز البريدي غير صالح؛ يمكن أن يكون القطاع رقمًا واحدًا فقط)

الشكل 4.34 فحص التنسيق على رمز المنطقة البريدية

الرموز البريدية في الأردن تتكون من 5 أرقام، على سبيل المثال، وسط عمان هو 11110. في هذه الحالة، سيكون التحقق من التنسيق هو التأكد من أن الرمز يحتوي على خمسة أرقام ويبدأ بالرقم 1. وبالمثل، تحتوي الرموز البريدية لباكستان على خمسة أرقام، ولكن يمكن أن يكون الرقم الأول أي شيء بين 1 و9. أي رمز يبدأ بالرقم 0 سيكون غير صحيح حتى لو كان يتكون من 5 أرقام.

رقم التحقق (Check digit)

عادةً ما يكون رقم التحقق عبارة عن رمز واحد (عادةً رقم) مشتق من خوارزمية يتم تنفيذها على جزء من البيانات. تم تصميم الخوارزمية لإنشاء هذا الرقم المحدد فقط إذا كانت البيانات (مثل سلسلة من الرموز) تحتوي على تلك الرموز بالضبط وتم ترتيبها بهذا الترتيب المحدد. أي رمز غير صحيح أو تبديل لمواضع الرموز ينتج عنه قيمة رقم تحقق مختلفة ويفشل الاختبار. تُستخدم أرقام التحقق في الغالب لاكتشاف الأخطاء في القيم المدخلة مثل الباركود وأرقام الحسابات المصرفية وأكواد تسجيل البرامج.

رقم الكتاب الدولي المعياري ذو العشرة أرقام (ISBN-10) يستخدم رقم تحقق. على سبيل المثال، كان لإصدار سابق من هذا الكتاب رقم ISBN-10 هو 1846909287. يتم التعامل مع الرقم الأخير (7) كرقم تحقق. يمكنك تحديد ما إذا كان هذا الرمز صحيحًا باستخدام التقنية الموضحة في البيان المرحلي الآتي.
بعد يناير 2007، انتقلت الكتب إلى تنسيق رقم ISBN مكون من 13 رقمًا. الرقم الأخير في رقم ISBN المكون من 13 رقمًا هو أيضًا رقم التحقق.

خطوة بخطوة: التحقق من صحة الرقم

3 خطوات

1 اضرب كل رقم في أوزان موضعية أصغر وأصغر، كما هو موضح في الجدول 4.14.

الجدول 4.14 رقم التحقق للكتاب الدولي للطالب

الأرقام في رقم الكتاب الدولي ISBN	1	8	4	6	9	0	9	2	8	7
عامل الضرب	× 10	× 9	× 8	× 7	× 6	× 5	× 4	× 3	× 2	× 1
المجاميع الفرعية	**10**	**72**	**32**	**42**	**54**	**0**	**36**	**6**	**16**	**7**

2 أضف المجاميع الفرعية في جميع الأعمدة:

10 + 72 + 32 + 42 + 54 + 0 + 36 + 6 + 16 + 7 = 275

3 قم بإجراء **قسمة الباقي** على 11:

275 قسمة على 11 = 25 المتبقي 0

هل المتبقي 0؟ نعم، هناك، ما يعني أن رقم ISBN صالح. أي نتيجة أخرى تعني إدخال رقم ISBN غير صحيح.

تحدد هذه التقنية الأرقام غير الصحيحة أو المتبادلة في الكود. في المثال المعطى في خطوة بخطوة، يُعتبر الرقم الأخير (7) هو رقم التحقق.

المصطلح الرئيس

قسمة الباقي – إجراء عملية القسمة وإرجاع الباقي بدلًا من حساب الناتج العشري أو الكسري.

التهجئة

فحص الإملاء يقيم ما إذا كانت الكلمات المدخلة يمكن العثور عليها في قاموس إلكتروني - أي أنها كلمات صالحة.

معالجة الأخطاء وإعداد التقارير

العديد من لغات البرمجة الحديثة لديها ميزات بناء جملة مصممة للتعامل مع أخطاء وقت التشغيل عند حدوثها. إذا لم تكن لديهم هذه الميزات، فإن التطبيق سيتعطل أو يتجمد دون استجابة.

من الأساليب الشائعة لمعالجة الأخطاء استخدام تعبير "try... throw... catch" الذي يمكن العثور عليه في العديد من اللغات، بما في ذلك C ++ وMicrosoft C# وOracle Java وPHP. تعمل تقنية معالجة الأخطاء هذه عن طريق محاولة تنفيذ عملية ما، والتقاط أي أخطاء محتملة و(اختياريًا) إلقاء استثناء مناسب. يمنع هذا النهج التطبيق من فشل العملية بطريقة غير محكومة، حيث أن ذلك قد يتسبب في تعطل التطبيق بالكامل.

مثال على هذه التقنية في معالجة الأخطاء موضح في الشكل 4.35 في الصفحة الآتية من خلال توضيح مخاطر القسمة على الصفر. في هذا المثال، يتم وضع عملية القسمة داخل كتلة المحاولة، تحسبًا لإدخال المستخدم '0' (صفر) كرقم ثاني. تم القيام بذلك لأن قسمة رقم على صفر يمكن أن تولد خطأً خطيرًا على منصة الحاسوب وقد تؤدي إلى تعطل وقت التشغيل. باستخدام كتلة "try... catch"، يمكنك تجنب ذلك من خلال عرض الاستثناء الذي تم اكتشافه، والذي يعد في هذه الحالة خطأ "DividebyZero".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace test
{
    class Program
    {
        static void Main(string[] args)
        {
            float num1;        // first number
            float num2;        // second number
            float quotient;    // result of dividing first number by second
            string strNumber; // string to temporarily store our input

            quotient = 0;

            //get 1st number
            Console.WriteLine("Enter 1st number");
            strNumber = Console.ReadLine();
            num1 = float.Parse(strNumber);

            //get 2nd number
            Console.WriteLine("Enter 2nd number");
            strNumber = Console.ReadLine();
            num2 = float.Parse(strNumber);

            //try the division
            try
            {
                quotient = num1 / num2;
                Console.WriteLine("{0} / {1} = {2}", num1, num2, quotient);
            }
            catch (DivideByZeroException e)
            {
                //output the handled exception
                Console.WriteLine("Error exception caught was: {0}", e);
            }

            //wait for key press before finishing
            Console.ReadKey();
        }
    }
}
```

الشكل 4.35 مثال على معالجة الأخطاء في Microsoft C# باستخدام كتل try وcatch

الإبلاغ عن الأخطاء هو جانب مهم من تطوير البرمجيات. قد يتم توجيهها إلى المستخدم، من خلال إخباره بأنه ارتكب خطأ، أو نحو المطور، حتى يتمكن من فهم أين واجه البرنامج خطأ وطبيعة هذا الخطأ.

تشمل التقنيات الشائعة للإبلاغ عن الأخطاء:

- عرض رسالة خطأ على الشاشة و/أو رمز خطأ
- إلحاق تفاصيل الخطأ بملف سجل إلكتروني يمكن عرضه بشكل منفصل
- إرسال بريد إلكتروني إلى المطور يتضمن تفاصيل الخطأ.

اختيار اللغة

هناك عدد من العوامل المختلفة التي تؤثر في اختيار لغة البرمجة لمشروع تطوير البرمجيات.

تفضيل العميل

قد يعبر عميلك عن تفضيله للغة برمجة معينة.

الملاءمة

اعتمادًا على الطبيعة التقنية لمهمة تطوير البرمجيات، قد تكون لغات البرمجة المختلفة أكثر ملاءمة في كل حالة.

إذا كنت تقوم بإنشاء حل للتجارة الإلكترونية، فعادةً ما تستخدم PHP (PHP: معالج النص التشعبي المسبق) أو تقنيات ASP .NET من Microsoft. هذا لأن هذه لغات برمجة نصية من جانب الخادم. إنهم يقدمون الوظائف والميزات والمكتبات التي تمكن من التطوير السريع لهذا النوع من الحلول.

إذا كنت تقوم بإنشاء لعبة فيديو، يمكنك استخدام Microsoft C# ('C sharp'). تميل C# إلى أن تكون مفضلة بسبب إطار عمل .NET. الواسع XNA، الذي يمكّن من تطوير ألعاب الفيديو لنظام التشغيل Microsoft Windows وXbox وهواتف Windows المحمولة.

إذا كنت تقوم بإنشاء كود برمجي للتفاعل مع الأجهزة الإلكترونية، يمكنك استخدام لغة C. تستخدم هذه الأنظمة المدمجة قدرة C على الوصول إلى الأجهزة على مستوى 'الهيكل المادي' 'bare metal'، حيث يمكنها بدء وتوقيت ومراقبة الإشارات الإلكترونية بسهولة في الوقت الفعلي. لهذا السبب، يعد C خيارًا برمجيًّا شائعًا للتحكم في تفاعل Raspberry Pi أو Arduino مع الأجهزة الخارجية.

قابلية النقل

بعض لغات البرمجة أكثر قابلية للنقل من غيرها. وهذا يعني أن الكود البرمجي المكتوب يمكن ترجمته (تحويله إلى كود الآلة أو النظام الثنائي) ليُستخدم على العديد من وحدات المعالجة المركزية المختلفة. لغة C هي لغة برمجة ناضجة ولذلك يتوافر عدد لا يحصى من برامج التحويل البرمجي لترجمتها، بما في ذلك تلك التي تقوم بالتحويل المتبادل من منصة أجهزة إلى أخرى. على سبيل المثال، يمكن لبرنامج التحويل البرمجي للغة C على منصة Microsoft Windows (معالج X86_64) إنشاء كود آلة ليعمل على جهاز Android (**وحدة المعالجة المركزية ARM**)، مثل الهاتف المحمول أو الجهاز اللوحي، وغيرها.

قللت شعبية Java من بعض مخاوف قابلية النقل. هذا لأن كود Java، عند ترجمته إلى كود بايت "**مستقل بنائيًا**" عن المعالج، يمكن تشغيله في الآلات الافتراضية الخاصة للغة جافا (JVM). توجد هذه الأجهزة الافتراضية ضمن مجموعة واسعة من الأجهزة، من الهواتف المحمولة إلى أجهزة التلفاز.

قابلية الصيانة

على الرغم من أن الصيانة هي واحدة من المراحل المتأخرة في دورة حياة تطوير البرمجيات (SDLC)، إلا أنها ستؤثر في اختيار لغة البرمجة. من غير المحتمل أن لا تحتاج الحلول المبرمجة إلى صيانة أبدًا. الأسباب الأكثر شيوعًا للصيانة هي الأخطاء والتغييرات في احتياجات العميل أو المستخدم. يمكن أن تستغرق الصيانة وقتًا، لذلك من المهم أن تشجع لغة البرمجة المستهدفة على ممارسات المطور الجيدة مثل

نصائح

يجب عليك دائمًا التأكد من عدم عرض معلومات التصحيح على شاشة المستخدم. هذا لأنها قد تكشف عن طريق الخطأ بيانات حساسة أو الأعمال الداخلية للبرنامج. هذا مهم بشكل خاص عند تطوير التطبيقات عبر الإنترنت من أجل منع القرصنة والاحتيال المحتمل.

بحث

تحقق من سبب قيام العميل بتحديد لغة برمجة لاستخدامها عندما يقوم بإحاطة فريق التطوير بسير العمل.

المصطلحات الرئيسة

وحدة المعالجة المركزية ARM – عائلة من وحدات المعالجة المركزية ذات الكفاءة في استهلاك الطاقة التي أنشأتها شركة ARM، المعروفة سابقًا باسم آلات RISC المتقدمة، والتي تُستخدم في مجموعة متنوعة من الأجهزة الإلكترونية مثل الهواتف المحمولة والأجهزة اللوحية وأجهزة الألعاب المحمولة.

مستقل بنائيًا – كود غير مصمم للتشغيل على عائلة معينة من وحدات المعالجة المركزية ولكن بدلًا من ذلك، يتم تشغيله على جهاز افتراضي.

القابلية للقراءة والتوسع. ستؤدي هذه العوامل إلى كود برمجي أسهل في الصيانة، وهذا مفيد بشكل خاص إذا كان الشخص الذي يقوم بصيانة الحل البرمجي ليس المطور الأصلي.

التوسع (Extensibility)

التوسع يصف قدرة الحل على النمو (أو التوسيع) مع تغير احتياجات العميل أو المستخدمين بمرور الوقت. بعض الأساليب البرمجية، مثل البرمجة كائنية التوجه (OOP)، قابلة للتوسع بشكل خاص. يرجع هذا الأمر إلى أن طبيعتها القائمة على الفئات تعكس بشكل وثيق المواقف والعمليات والبيانات في العالم الحقيقي، ما يضمن أن يكون الكود سهل التكيف مع تغير الاحتياجات. لهذا السبب، فإن لغات البرمجة كائنية التوجه مثل C++ و C# تحظى بشعبية كبيرة.

الخبرة

ترتبط القدرة التقنية للمطور بمعرفته بلغة برمجة معينة. ما لم يتم توظيف مطورين خارجيين بعقود قصيرة الأجل لمدة المشروع، فإن اختيار اللغة مقيد بالخبرة المتاحة داخل الشركة.

غالبًا ما يتم تشجيع المطورين على توسيع مجموعة مهاراتهم وهذا يعني أنه قد يتم إرسالهم في برامج دراسية من قبل مديريهم كجزء من تطويرهم المهني المستمر (CPD). ومع ذلك، غالبًا ما تكون الشركات مستعدة لدفع أجر لمطوري العقود للحصول على الخبرة في التقنيات الجديدة أو المطلوبة إذا لم يكن لدى مطوريها الداخليين مجموعة المهارات المطلوبة لتنفيذ المشاريع الحرجة.

الوقت

يعتمد طول الوقت المستغرق في التطوير على تعقيد المشكلة واختيار لغة البرمجة والأدوات المستخدمة لحل المشكلة. تم تصميم معظم بيئات التطوير الحديثة لتحسين سرعة عمل المبرمج. هذا لأن تحسين الإنتاجية في القطاع التجاري مهم في خفض التكاليف.

التطوير السريع للتطبيقات (RAD) هو نهج لتطوير البرمجيات يمكّن المبرمجين من إنتاج الكود البرمجي بسرعة كبيرة. يتم تحقيق ذلك باستخدام لغات البرمجة التي تسمح للمبرمج بإنشاء **نماذج أولية** للتطبيق بسرعة. يمكن بعد ذلك تعديل هذه النماذج الأولية من خلال عملية تكرارية لاختبار العميل والمراجعة. يمكن أن يقلل ذلك لوقت المستغرق في تخطيط المشروع. يعتمد التطوير السريع للتطبيقات (RAD) أيضًا على استخدام مقتطفات الكود القابلة لإعادة الاستخدام وإنشاء واجهة مستخدم بالسحب والإفلات. هذا يعني أن لغات البرمجة التي تدعم التطوير السريع للتطبيقات (RAD)، مثل Visual Basic .NET من Microsoft، يمكن أن يكون لها تأثير مفيد في وقت التطوير.

الدعم

يمكن تنزيل العديد من لغات البرمجة وأدوات التطوير مجانًا من الإنترنت. وهذا يسمح للمطورين ببناء حلول تجارية بتكاليف قليلة. ومع ذلك، قد لا يكون هناك أي دعم فني لهذه اللغات والأدوات المجانية. بالإضافة إلى ذلك، قد لا يتم تحديثها أو صيانتها. لهذا السبب، من المهم عند اختيار لغة برمجة أن تعرف أن التكنولوجيا لن تتوقف أو تصبح غير مدعومة في المستقبل القريب. قد يعني هذا أن المطور قد يفضل شراء أداة تطوير تقدم دعمًا كاملًا من الشركة المصنعة، وتحديثات وإصلاحات متكررة، ومجتمعًا نشطًا من المستخدمين.

المصطلح الرئيس

النموذج الأولي – نموذج عملي للحل المطلوب أو مكونات الحل. قد لا يشتمل على جميع الوظائف ولكنه يسمح للعملاء والمستخدمين باختبار ومراجعة الحل المقترح. ثم يتم الاستفادة من التعقيبات في تحسين النموذج الأولي التالي وتستمر العملية حتى يصبح المنتج النهائي جاهزًا.

التكلفة

يمكن ربط تكلفة تطوير المشروع بعدد من العوامل. العديد من هذه العوامل مُرتبط بشكل مباشر أو غير مباشر باختيار لغة البرمجة. تشمل عوامل التكلفة القابلة للتحديد:

- أدوات التطوير
- المهارات البرمجية المطلوبة، إما من خلال تدريب الموظفين الحاليين وإما توظيف مطورين بعقود يمتلكون الخبرة اللازمة في اللغة
- التراخيص المطلوبة للغات أو الأدوات أو توزيع كود البرنامج التنفيذي
- سرعة التطوير (الوقت المستغرق لتطوير الحل البرمجي باللغة المختارة)
- سهولة الصيانة (احتمالية الأخطاء)
- سهولة التوسع

حساب تكلفة المشروع مهمة معقدة، واختيار لغة البرمجة يلعب دورًا رئيسًا في تقديرات التكلفة.

البرامج المحددة مسبقًا ومقتطفات الكود

في بعض الأحيان يكون من الممكن دمج برامج محددة مسبقًا أو مقتطفات من الأكواد الموجودة داخل الحل. يجب توثيقها كجزء من تصميم حلول برمجية حتى يعرف مطورو المستقبل ما يتعاملون معه. تقدم العديد من بيئات تطوير البرمجيات مقتطفات من الأكواد المكتوبة مسبقًا والتي يمكن الرجوع إليها أو ببساطة 'إسقاطها' في الحل في أثناء تطويره. بالإضافة إلى ذلك، يمكن أيضًا تنزيل شفرة الطرف الثالث ودمجها مع القليل من الصعوبة التقنية.

استخدام البرامج والرموز المسبقة له مزايا وعيوب، كما هو موضح في الجدول 4.15.

يجب على المطورين دائمًا التحقق من أن استخدامهم لبرنامج أو مقتطف كود محدد مسبقًا لا ينتهك شروط الاستخدام التي حددها المطور الأصلي. على سبيل المثال، قد يقدم المطور الكود الخاص به "مجانًا للاستخدام" طالما لم يتم استخدامه في حل مطور تجاريًا. إذا استخدم مطور آخر هذا الكود في منتج تجاري ينتجه، فسيعتبر تصرفه غير قانوني.

الجدول 4.15 مزايا وعيوب استخدام كود طرف ثالث في الحل

المزايا	العيوب
يوفر وقت التطوير.	قد يتسبب الكود في حدوث أخطاء محتملة إذا لم يتم اختباره بشكل مناسب.
يوفر المال (إذا كانت البرامج/الكود مجانية ولا تتطلب ترخيصًا إضافيًا).	قد لا يكون الكود قابلًا للتعديل (في بعض الحالات، قد يمنع المطور الأصلي التعديل بشكل صريح).
من المحتمل أن يتم اختبار الكود مسبقًا لذا يجب أن يكون خاليًا من الأخطاء.	قد يؤدي ذلك إلى مشاكل توافق غير متوقعة.
من المحتمل أن يتم كتابة الكود بطريقة فعالة وقابلة للصيانة.	قد لا يعمل الكود مع حل المطور إذا تم تحديث البرنامج/الشفرة المحددة مسبقًا.
قد يوفر الكود وظائف إضافية قد تكون مفيدة في المستقبل.	قد تحتوي البرامج المحددة مسبقًا على برامج ضارة، خاصة إذا تم تنزيلها من مصدر غير موثوق به.
	قد لا يكون هناك دعم من المطور الأصلي.
	قد يتم إيقاف الكود أو التخلي عنه.

فكِّر مليًّا

ينطوي تصميم أي تطبيق على كميات كبيرة من التخطيط، لا سيما من حيث التحقيق في متطلبات العميل وفهمها وتعقيدات لغة البرمجة المختارة.

يجب ألا يقتصر أي حل مقترح على سرد ميزات البرنامج التي تحتاج إلى مراعاتها فحسب، بل يجب أن يكون فحصًا شاملًا لجميع العوامل التي شكلت حلَّك، بما في ذلك التعليقات الواردة من العميل (إذا كنت قد شاركت تصميمات النماذج الأولية) وزملائك.

كما أن تسجيل النتائج بدقة وبشكل شامل خلال تصميم الحلول (بما في ذلك إعداد الأصول الأصلية) يعزز حل المشكلات من خلال توفير أساس متين لبدء التطوير الفعال.

الأصول الجاهزة والأصلية

عادةً ما تحتوي البرامج الحديثة على عرض تقديمي غني بالوسائط وتدمج مجموعة واسعة من الأصول عالية الجودة لتعزيز مظهرها وواجهة المستخدم والوظائف. قد تتضمن الأصول الرقمية النموذجية:

- الرسوم – مثل PNG، أو BMP أو JPEG
- الرسوم المتحركة – مثل HTML5، Adobe Flash SWF وملفات GIF المتحركة
- الصوت – مثل WAV، MP3
- فيديو – مثل AVI، MP4
- **أنواع الخط والخطوط** – مثل Arial وTimes New Roman وVerdana.

يجب تضمين الأصول المستخدمة من قبل البرنامج في تصميم حلول البرمجيات كمورد رئيس. يتم تجميعها بحسب الفئة ويتم سرد التفاصيل الآتية لكل أصل:

- أسماء الملفات (والمسارات)
- تنسيقات الملف
- أحجام الملفات
- الأبعاد/الدقة بالبكسل (للرسوم الرقمية)
- المدة (للصوت والفيديو الرقمي)
- إطارات (للرسوم المتحركة)
- إطارات في الثانية (إطارات في الثانية - للرسوم المتحركة والفيديو)
- ملاحظات حول الغرض/الاستخدام في البرنامج
- أي معلومات ترخيص مطلوبة، مثل حقوق الطبع والنشر الأصلية.

ملاحظات من الآخرين

يتمثل أحد الجوانب الرئيسة لعملية تصميم البرامج في تلقي الملاحظات من الأقران وعميلك. تصميم التطبيق هو عملية تكرارية - أي أنها تكرر عملية معينة عدة مرات حتى تقترب من الاكتمال. ويُمكن استخدام الملاحظات في تحسين عملية معاودة إجراء العملية.

جمع الملاحظات حول تخطيطات الشاشة، واجهة المستخدم، التنقل، الخوارزميات وما إلى ذلك سيساعدك على تحسين أفكارك. وسيتيح ذلك لك تحديد وإزالة جوانب التصميم التي لا تعمل بشكل جيد وتحديد الأفكار التي يحبها المستخدمون والاحتفاظ بها. سيساعدك أيضًا على اكتساب الثقة في قدراتك على حل المشكلات، ويشجعك على النظر في أفكار تصميم بديلة ويقوي قدرتك على اتخاذ القرارات وتبرير التغييرات للآخرين.

المصطلحات الرئيسة

نوع الخط – تصميم الأبجدية، أي الأشكال الفعلية للحروف والرموز. "Arial" و"Times New Roman" هما مثالان لنوع الخط.

الخط – كلمة تستخدم لوصف الملف الرقمي الذي يحتوي على الخط. على سبيل المثال، arial.ttf هو ملف خط TrueType. يستخدم العديد من الأشخاص مصطلحي 'نوع الخط' و'الخط' بالتبادل.

نصائح

عند استخدام أي أصل رقمي، يجب أن تقر بحقوق الطبع والنشر الخاصة به. في بعض الأحيان، قد تحتاج إلى طلب إذن رسمي لتضمين الأصل من مالك حقوق الطبع والنشر. قد يشمل ذلك دفع رسوم ترخيص لمالك حقوق الطبع والنشر. في المملكة المتحدة، يتم تعريف حقوق النشر بموجب قانون حقوق النشر والتصاميم وبراءات الاختراع لعام 1988.

فكِّر مليًّا

في سياق تطوير حل البرنامج الخاص بك، ستتلقى تعليقات من أقرانك ومستخدمي الاختبار والعميل. يمكن أن تكون وجهات نظر وآراء الآخرين مفيدة للغاية، لأنها عادة ما تكون غير متحيزة ويمكن أن توفر حكمًا محايدًا حول تصميم وتنفيذ برنامجك. هذا مهم بشكل خاص عند النظر في نقاط الضعف، وكذلك نقاط القوة.

تعلم كيفية تلقي الملاحظات بشكل إيجابي. سيضمن ذلك أن تستجيب لها بطريقة ناضجة وبناءة وأن تتمكن من استخدامها لتحسين المنتج النهائي وأن تكتسب مهارة في حل المشكلات.

موضوعات ذات صلة

تعد التعليقات أيضًا مكونًا رئيسًا للاختبار القوي لتطبيقات الأجهزة المحمولة، كما هو موضح في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

خطة الاختبار

يجب أن يكون الاختبار جزءًا من تصميم الحلول البرمجية. ويجب أن توضح خطة الاختبار كيف تخطط لاختبار برنامجك، خطوة بخطوة، وأن تتضمن العديد من حالات الاستخدام. كل حالة استخدام تروي قصة تفاعل المستخدم الناجح أو غير الناجح مع البرنامج. تشمل حالات الاستخدام تفاصيل:

- البيانات التجريبية المدخلة
- العمليات المنفذة
- ترتيب العمليات التي يتم تنفيذها
- استجابة المستخدم لمطالبات البرنامج.

عادةً ما توجد بيانات الاختبار في واحدة من ثلاث حالات ممكنة:

- النموذجي/العادي - البيانات ضمن النطاق المقبول الذي سيدخله المستخدم عادةً
- طرفي - بيانات غير محتملة عند حواف النطاق المقبول.
- خاطئ - البيانات التي لا يجب إدخالها (مثل النص بدلًا من الأرقام).

يجب اختيار بيانات الاختبار بشكل واقعي، حيثما كان ذلك مناسبًا. على سبيل المثال، يمكن أخذها من القيم التي يقدمها العميل، خاصة إذا كنت تقوم بأتمتة عملية يدوية، مثل معالجة الطلبات لعميلك.

موضوعات ذات صلة

لمزيد من المعلومات حول بيانات الاختبار، راجع **الوحدة 5: نمذجة البيانات**.

القيود الفنية والتصميمية

قد تكون مهاراتك في حل المشكلات محدودة بمجموعة متنوعة من القيود التقنية والتصميمية. يجب أن تنعكس هذه القيود في تصميم الحلول البرمجية وقد تشمل ما يأتي:

- الاتصال - ما الأجهزة أو اتصالات الشبكة المطلوبة لتطبيقك؟ هذا مهم بشكل خاص عند الحاجة إلى أجهزة خاصة (على سبيل المثال للإدخال أو التخزين أو الإخراج) أو إذا كان التطبيق يحتاج إلى اتصال شبكة نشط.
- تخزين الذاكرة - هل هناك متطلبات ذاكرة الوصول العشوائي (RAM) التي يجب ألا تتجاوزها البصمة الرقمية لبرنامجك؟
- لغات البرمجة - هل تم اختيار لغة البرمجة مسبقًا من قبل العميل؟ قد يحدث هذا إذا كان لديهم خبرة مع التطبيقات المكتوبة بلغات برمجة معينة.

فكِّر مليًّا

يوضح تصميمك للحلول قدرتك على نقل أفكارك ومقاصدك بدقة للآخرين من خلال الكلمة المكتوبة والمنطوقة. هذا صحيح سواء كنت تتواصل مع العميل عبر البريد الإلكتروني أو تكتب كودًا زائفًا أو تجمع ملاحظات المقابلة أو تنشئ مخططات انسيابية لتمثيل الخوارزميات المعقدة.

يجب أن يتفاعل استخدامك للنبرة واللغة بشكل مثمر مع العميل والمستخدمين المحتملين للبرنامج النهائي.

قد يُطلب منك تقديم الحل الخاص بك إلى العميل. سيكون عرضك التقديمي أكثر نجاحًا إذا كنت:

- تستخدم نبرة إيجابية وجذابة
- تختار مستوىً مناسبًا من اللغة التقنية التي يفهمها جمهورك المستهدف
- تجنب استخدام المصطلحات غير الضرورية.

وقفة للتفكير

هل يمكنك شرح نتاج التعلم هذا؟ ما العناصر التي وجدتها أسهل؟

تلميح دون النظر إلى نص هذه الدراسة، قم بإنشاء قائمة بسيطة بالأنشطة والوثائق التي تحتاج إلى تجميعها كجزء من تصميم الحلول البرمجية.

توسيع الأفق ما العوامل التي قد تؤثر في اختيار لغة البرمجة؟

ج { تطوير حل برمجي لتلبية متطلبات العميل

المهارات

- اختيار أدوات وأنظمة تكنولوجيا المعلومات المناسبة لتطوير حلول البرمجيات
- مهارات الإدارة الذاتية والتخطيط
- القدرة على العمل بطريقة قانونية وأخلاقية

موضوعات ذات صلة

لمزيد من المعلومات حول بيئات التطوير المتكاملة، انظر **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

بمجرد تصميم الحل البرمجي وتوثيقه ومراجعته والموافقة عليه، يحين الوقت لتنفيذ التصميم لإنتاج تطبيق يعمل. إذا كان تصميمك مفصلًا، يجب أن تجد أنه من السهل نسبيًا تحويله إلى كود مناسب في لغة البرمجة المستهدفة.

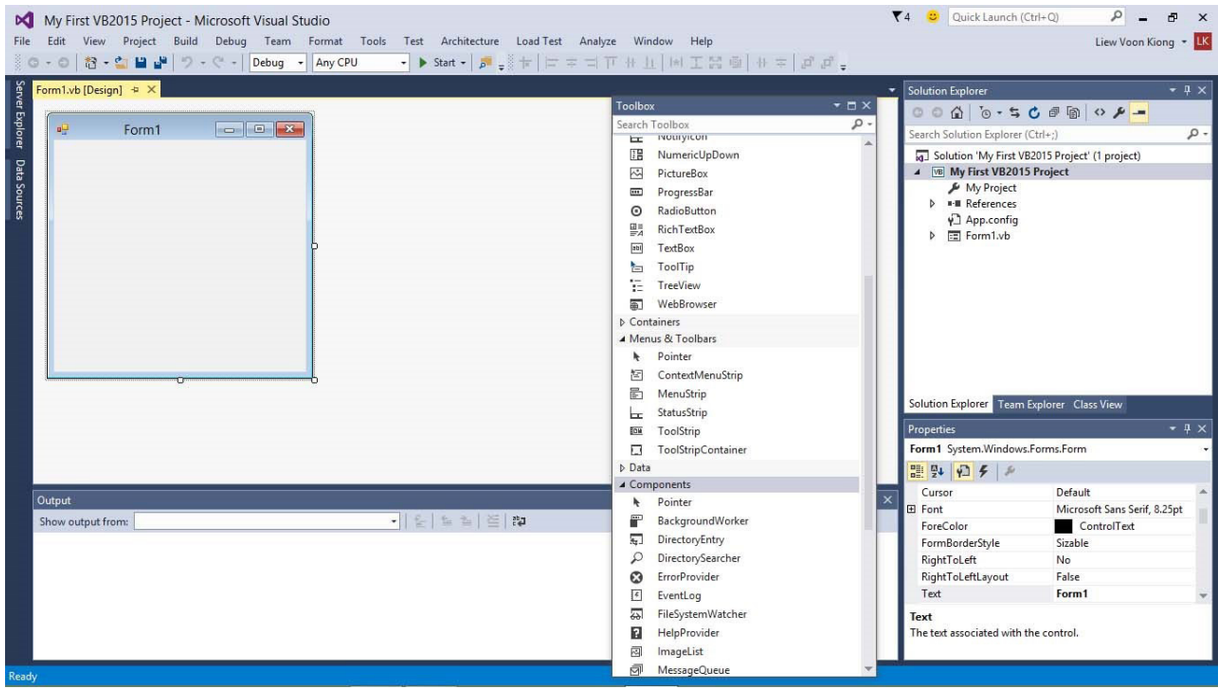
تطوير حلول البرمجيات

يمكن أن تستغرق عملية تطوير حلول البرمجيات وقتًا طويلًا. ومع ذلك، فقد أزالت أدوات التطوير الحديثة العديد من الصعوبات لكل من المبرمجين المبتدئين وذوي الخبرة. سيكون لديك العديد من الخيارات المتاحة لك عند تطوير تطبيقك. خطوتك الأولى هي اختيار بيئة التطوير الخاصة بك.

بيئة التطوير

تستخدم عملية تطوير البرمجيات بيئات تطوير متكاملة (IDEs). تساعد بيئات التطوير المتكاملة (IDEs) في تحسين الإنتاجية من خلال السماح للمطورين بإدارة المشاريع وتحرير وتجميع وتصحيح وتنفيذ الكود، كل ذلك من داخل مجموعة البرامج نفسها.

وأحيانًا، تكون بيئة التطوير المتكاملة (IDE) خاصة بلغة البرمجة المختارة. على سبيل المثال، يستخدم Microsoft Visual Basic .NET إصدارات Visual Studio التجارية أو إصدارات Visual Studio Express المجانية، كما هو موضح في الشكل 4.36.



الشكل 4.36 بيئة التطوير المتكاملة Microsoft Visual Basic .NET

لغات البرمجة الأخرى لا تحتوي على بيئة تطوير متكاملة محددة، وقد يؤثر ذلك في اختيار لغة البرمجة. يمكن تطوير اللغات الشائعة مثل C و++C وPHP باستخدام مجموعة متنوعة من بيئات التطوير. قد تتكون هذه البيئات من عدد من العناصر المتباينة. على سبيل المثال، يمكن تطوير PHP باستخدام أي من الإعدادات الآتية:

- نظام تشغيل لينكس، محرر النصوص نانو، خادم الويب أباتشي HTTPD مع وحدات PHP5 المثبتة
- نظام تشغيل Microsoft Windows، Notepad ++، Microsoft IIS Web Server مع تثبيت ثنائيات ويندوز PHP.

أي من هذه الحلول سيوفر بيئة مناسبة لمطور يعمل مع PHP.

الروتينات المكتبية والأكواد القياسية والروتين الفرعي التي ينشئها المستخدم

يجب أن توفر بيئة التطوير المختارة إمكانية الوصول إلى الروتينات المكتبية والأكواد القياسية والروتينات الفرعية المحددة من قبل المبرمج. يمكن استخدام ذلك لجعل البرنامج أكثر كفاءة.

تحتوي العديد من بيئات التطوير على ميزات لمساعدة المبرمجين على توليد الكود بسرعة أكبر. وتشمل هذه الميزات ما يأتي:

1. كود القالب (Boilerplate code) - يحدد 'الهيكل الرئيس' للكود القياسي الذي يعمل كنقطة انطلاق للمبرمجين لكتابة تطبيقاتهم.
2. الإكمال التلقائي للكود - النوافذ المنبثقة التي تقترح الكود الذي يمكن للمبرمج استخدامه لإنهاء سطر الكود الحالي، وغالبًا ما تقدم عددًا من الخيارات المختلفة، كاملة مع التعريفات والاستخدامات.
3. مقتطفات الكود - أقسام قصيرة من الكود التي يمكن إدراجها في برنامج من مكتبة من الروتينات المكتوبة مسبقًا، وغالبًا ما تتضمن المهام المستخدمة بشكل شائع.

كل هذه الميزات مصممة لتوفير الوقت للمبرمج وجعل عملية التطوير أكثر سهولة. بالإضافة إلى ذلك، توفر العديد من بيئات التطوير الوصول إلى موارد عبر الإنترنت تحتوي على دعم من طرف ثالث. على سبيل المثال، تحتوي MSDN (شبكة مطوري Microsoft) من Microsoft على مكتبة واسعة من الموارد والمنتديات النشطة.

اختبار الحلول البرمجية

بمجرد إزالة الأخطاء الواضحة، مثل أخطاء الصياغة، بنجاح من البرنامج باستخدام أدوات التصحيح المتاحة في بيئة البرمجة الخاصة بك، يمكن اختبار الحلول البرمجية. يتضمن ذلك اتباع عملية بسيطة خطوة بخطوة، باستخدام خطة اختبار وبيانات اختبار.

موضوعات ذات صلة

يتم مناقشة بيئات التطوير الشائعة لتطبيقات الهاتف المحمول في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

موضوعات ذات صلة

تُستخدم بيئات التطوير المتكاملة في جميع أنواع التطوير. على سبيل المثال، **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة** يفحص بيئات التطوير المتكاملة الشائعة المستخدمة لإنشاء تطبيقات الأجهزة المحمولة التي تعمل بنظامي التشغيل Apple iOS وAndroid ويوفر مثالًا عمليًا لإنشاء تطبيق برمجي لحاجة مستخدم محددة. لمزيد من المعلومات حول بيئات التطوير المتكاملة في سياقات مختلفة، انظر **الوحدة 6: تطوير المواقع الإلكترونية** و**الوحدة 8: تطوير ألعاب الحاسوب**.

مثال عملي: اختبار البرمجيات

الخطوة 1: تضع آنا خطة اختبار وتختار بيانات اختبار مناسبة، والتي تشمل بيانات نموذجية، ومتطرفة وخاطئة.

الخطوة 2: تُدخل آنا بيانات اختبار العينة الخاصة بها واختيارات المستخدم في جدول التتبع. ثم تسجل النتائج التي تتوقع رؤيتها.

الخطوة 3: تقوم آنا بإدخال بيانات اختبار العينة الخاصة بها في البرنامج المباشر وتسجيل النتائج الفعلية.

الخطوة 4: تُقارن آنا بين المخرجات المتوقعة والفعلية لترى ما إذا كانت مختلفة.

الخطوة 5: تقوم آنا بتحديد أي تناقضات وتعيد النظر في كود البرنامج لإصلاح الأخطاء الدلالية التي حددها الاختبار.

الخطوة 6: تُعيد آنا اختبار البرنامج لترى ما إذا كانت التغييرات التي أجرتها على الكود قد أصلحت الأخطاء. إذا لم تفعل ذلك، يجب عليها بعد ذلك تكرار العملية حتى يتم حل الأخطاء.

أنواع أخرى من الاختبارات ستشمل ما يأتي:

- اختبار التوافق - اختبار التطبيق في بيئات مختلفة، مثل أنظمة التشغيل المختلفة و/أو منصات الأجهزة المختلفة (حاسوب مكتبي، جهاز أبل ماك، هاتف محمول، جهاز اللوحي، وحدة تحكم في الألعاب).
- اختبار الاستقرار - المعروف أيضًا باسم اختبار التحميل أو التحمل؛ يتحقق من أن الحل البرمجي يعمل بشكل جيد باستمرار، مرة بعد مرة، دون مشاكل.
- الاختبار الوظيفي - أسلوب اختبار الصندوق الأسود الذي يضمن أن الحل البرمجي يلبي المتطلبات الوظيفية التي قدمها العميل في الأصل.

أخطاء بناء الجملة هي الأخطاء في الكود المصدري التي لا يستطيع مترجم لغة البرمجة تمريرها أو حلها، والتي يقوم المترجم بالإبلاغ عنها لتنبيه المطور. يحدث هذا عندما يكسر كود البرنامج قواعد أو بنية اللغة، على سبيل المثال عندما تُستخدم رموز غير صالحة أو تُكتب الكلمات المحجوزة بشكل غير صحيح.

الأخطاء الدلالية في الجمل غالبًا ما تكون أكثر صعوبة في تمييزها وإصلاحها من أخطاء بناء الجملة. هذا لأن تركيبتها جيدة، لذلك لا يتم الإبلاغ عنها من قبل المترجم. ومع ذلك، فهي "خطأ في المنطق أو المعنى"، ما يعني أنها لا تؤدي المهمة التي قصدتها.

تطوير تطبيقات البرامج وتنقيحها وتحسينها

بصفتك مطورًا، يمكنك استخدام عدد من الطرق لتطوير حل البرنامج وتحسينه. وتشمل الآتي:

- وضع تعليقات توضيحية على الكود، حيث سيسمح هذا للمطورين بإصلاح البرنامج أو تصحيحه وسيحسن من إمكانية صيانته
- تجميع البرنامج لمنصة أو بيئة معينة
- مراجعة جودة البرنامج من حيث الموثوقية وسهولة الاستخدام والكفاءة وقابلية الصيانة وقابلية النقل
- إجراء اختبار المستخدم والحصول على ملاحظات من المستخدمين حول تجربتهم
- الاستفادة من نتائج الاختبار والتغذية الراجعة في أي صيانة تحسينية أو تكيفية مخططة، على سبيل المثال استخدام الأخطاء التي تم العثور عليها في أثناء الاختبار الرسمي للكود المستهدف الذي يحتاج إلى إصلاح أو متابعة ملاحظات المستخدم لتحديد الجوانب التي تحتاج إلى تحسين
- توثيق أي تغييرات في التصميم والحل، مثل التغييرات في كود البرنامج، والمدخلات المطلوبة، والمخرجات المنسقة، وتخزين البيانات، وغيرها.

مراجعة الحلول البرمجية

يجب عليك دائمًا مراجعة وتقييم حلول البرمجيات الخاصة بك بمجرد تطويرها. قد تجد أن الأسئلة الآتية مفيدة عند مراجعة الحل الخاص بك:

- ما مدى ملاءمته للفئة المستهدفة والغرض المقصود؟
- هل يلبي الاحتياجات الأصلية للعميل؟
- ما مدى سهولة استخدامه؟
- هل الحل البرمجي ذو جودة عالية؟
  - أن يكون موثوقًا
  - قابل للاستخدام
  - فعّال في الأداء
  - سهل الصيانة
  - قابل للنقل إلى منصات أخرى؟
- هل تحتوي لغة البرمجة المختارة على قيود أضرت بالحل؟

موضوعات ذات صلة

لاختبار البرمجيات، تحتاج إلى خطة اختبار وبيانات اختبار، والتي تم مناقشتها سابقًا في هذه الوحدة. يتم أيضًا مناقشة اختبار البرمجيات في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**، و**الوحدة 13: اختبار البرمجيات**.

نصائح

إصلاح الأخطاء في إصدار من برنامج (غالبًا ما يُطلق عليه 'بناء') يمكن أن يؤدي عن غير قصد إلى تعطيل جوانب من البرنامج كانت تعمل قبل ذلك. تأكد دائمًا من اختبار البرنامج بالكامل في كل مرة يتم إصلاحه، بدلًا من اختبار الجوانب التي كانت التغييرات الأخيرة تهدف إلى إصلاحها فقط.

المهارات

- مهارات الإدارة الذاتية والتخطيط

موضوعات ذات صلة

انظر **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**، كمثال محدد على هذه الأساليب في تطوير تطبيقات الهاتف المحمول.

- هل كانت هناك أي قيود أخرى أثرت في الحل؟ وقد تتضمن:
  - الوقت المتاح للتطوير أو الاختبار
  - مهاراتك الشخصية ومعرفتك
  - مشاكل عند استخدام اللغات على منصات معينة.
- ما نقاط القوة والضعف في حل البرنامج؟
- كيف يمكنك تحسين الحل؟ فكر في هذا من حيث التحسينات التي ستجريها على المدى القصير والمتوسط والطويل.
- هل يمكن تحسين حل البرنامج من خلال:
  - تحسين المتانة، أي جعل البرنامج أكثر مرونة ضد أخطاء المستخدم والمدخلات السيئة وما إلى ذلك ومنعه من العمل بشكل متقطع أو التعطل
  - تحسين كفاءة الكود، أي جعل البرنامج ينفذ الخطوات بشكل أسرع أو يستخدم موارد أقل أو تزداد استجابته للمستخدم
  - توسيع الوظائف، أي تحسين تجربة المستخدم من خلال دمج خيارات جديدة أو التخصيص للمستخدم أو إضافة مهام جديدة للتطبيق لمعالجتها؟

المهارات والمعرفة والسلوكيات

عند تنفيذ حل جديد، ستحتاج إلى إظهار مهارات ومعارف وسلوكيات معينة. على سبيل المثال:

- التخطيط والتسجيل، بما في ذلك تحديد الأهداف ذات الصلة مع الجداول الزمنية، وكيف ومتى سيتم جمع التعليقات من الآخرين.
- مراجعة النتائج والاستجابة لها، بما في ذلك استخدام التعليقات من الآخرين، على سبيل المثال، التعليقات من متخصصي تكنولوجيا المعلومات والمستخدمين حول جودة البرنامج ومدى ملاءمته للمتطلبات الأصلية.
- عرض السلوكيات الشخصية وتأثيرها في النتائج، والتي تشمل الاحتراف، وآداب السلوك، ودعم الآخرين، والقيادة المناسبة في الوقت المناسب، والمساءلة والمسؤولية الفردية.
- تقييم النتائج للمساعدة في تقديم توصيات وقرارات مبررة عالية الجودة.
- مهارات الإعلام والاتصال، مثل القدرة على نقل المعنى المقصود، واستخدام اللهجة واللغة للاتصالات اللفظية والكتابية، والاستجابة بشكل بناء لمساهمات الآخرين.

موضوعات ذات صلة

يتم أيضًا مناقشة المراجعة بمزيد من التفصيل في **الوحدة 7: تطوير تطبيقات الأجهزة المحمولة**.

فكِّر مليًّا

التقييم هو مهارة أساسية ستطورها في عملك كمبرمج. خذ ما تعلمته من تقييم برامج الآخرين وأكوادهم وتمرن على تقييم برامجك الخاصة. قد تجد هذا صعبًا في البداية لأنك ستضطر إلى النظر إلى عملك كما لو كان عمل شخص آخر وتحديد نقاط ضعفه وكذلك نقاط قوته. ومع ذلك، فإن القيام بذلك سيحسن مهاراتك في البرمجة والأداء وسيطور قدرتك على تقديم توصيات مستنيرة وتبريرها. سيمنحك ذلك أيضًا مزيدًا من الثقة في اتخاذ القرارات حيث ستتمكن من تحليل أسبابك الخاصة لاتخاذ قرار معين أو التوصية به.

تمرين تقييمي 4.2 B.P4, B.P5, B.M2, C.P6, C.P7, C.M3, BC.D2, BC.D3

أنت مبرمج في فريق تطوير، وقد تم تقديمك للتو إلى عميل يريد من فريقك تطوير برنامج مخصص لتلبية احتياجات شركته المتغيرة.

العميل ليس لديه أفكار ثابتة حول كيفية حل المشكلة أو أي لغة برمجة يجب استخدامها. ومع ذلك، فقد قدم قائمة بالمدخلات المطلوبة، ووصفًا للمخرجات التي يجب إنشاؤها والإجراءات التي يجب أن يقوم بها التطبيق.

حقق في المشكلة وحلل خصائصها. عندما تعتقد أنك تفهمها بأفضل ما لديك من قدرة، قم بإنتاج تصميم شامل لبرنامج حاسوبي لتلبية متطلبات العميل.

تأكد من مراجعة التصميم مع الآخرين (بما في ذلك العميل) واستخدام ملاحظاتهم لتحسين الحل المقترح. يجب عليك تبرير جميع أحكام التصميم الخاصة بك، موضحًا أن الحل الذي قمت بإنشائه سيكون فعالًا ويلبي احتياجات العميل بالكامل.

بمجرد أن يتم توقيع التصميم من قبل العميل، سيطلب منك مدير الخط تنفيذ التصميم وتطوير الحل البرمجي.

تمرين تقييمي 4.2 *متابعة*

استخدم لغة برمجة مناسبة وميزات مناسبة لبيئة التطوير المختارة لإنتاج الكود المطلوب. بالإضافة إلى ذلك، تحتاج إلى اختيار روتينات المكتبة وتصميم الروتينات الفرعية لتحسين كفاءة برنامجك.

قم بإنشاء وتنفيذ خطة اختبار منطقية لاختبار برنامجك بشكل شامل. يجب عليك التحقق من وظائف الكود واستقراره وتوافقه. بمجرد اكتمال الاختبار، قم بتحسين الحل الخاص بك عن طريق توضيحه بشكل صحيح لتمكين الإصلاح الفعال وتصحيح الأخطاء.

راجع الكود الخاص بك من حيث الموثوقية وسهولة الاستخدام والكفاءة وقابلية الصيانة وقابلية النقل. استخدم ملاحظات المراجعة من المستخدمين لإجراء تغييرات على الحل وتوثيق الأساس المنطقي وراء كل تحسين.

أخيرًا، يجب عليك تقييم التصميم والبرنامج المحسن النهائي، ومقارنتهما مع متطلبات العميل لمعرفة ما إذا كنت قد لبيت احتياجاتهم الأصلية بالكامل.

يجب عليك إظهار المسؤولية الفردية والإبداع والإدارة الذاتية الفعالة في جميع مراحل مشروع تطوير البرمجيات.

التخطيط

- ما المهمة؟ ما المطلوب مني فعله؟ هل أفهم احتياجات العميل؟
- ما نقطة بدايتك في هذه المهمة؟ ما أهم شيء يجب معرفته؟
- ما مدى ثقتي في قدرتي على إنجاز هذه المهمة؟
- هل هناك أي مجالات قد أواجه صعوبة فيها، خاصة المجالات التقنية؟ إذا كان الأمر كذلك، فهل أعرف كيفية البحث عنها؟
- هل هناك أي موارد أو أمثلة عملية يمكن أن تساعدني في إنجاز هذه المهمة؟
- قبل أن أبدأ في كتابة الكود، إلى أي مدى الحل المقترح موصوف بشكل جيد في تصميم البرمجيات؟

التنفيذ

- أعرف ما أفعله وما أريد تحقيقه.
- سأستخدم هذا كفرصة لتجربة تقنيات جديدة وتحسين حل المشكلات.
- يمكنني استخدام التعليقات والمراجعات لتحديد متى أخطأت وتعديل تفكيري أو نهجي للعودة إلى المسار الصحيح.
- أعرف من أطلب منه ملاحظات حول الأفكار التي قمت بتوليدها.
- أنا مستعد لتحسين حلي بناءً على المتطلبات المحددة وملاحظات المستخدمين.

المراجعة

- يمكنني شرح المهمة وكيف تعاملت معها، مبررًا قراراتي في جميع المراحل.
- يمكنني تبرير التغييرات التي أجريتها بناءً على الملاحظات التي تلقيتها.
- أستطيع أن أشرح كيف سأتعامل مع العناصر الصعبة بشكل مختلف في المرة القادمة (أي ما سأفعله بشكل مختلف).

فكِّر في المستقبل

عبد الله خوري

مبرمج مبتدئ في فريق تطوير البرمجيات

لقد عملت في الفريق لمدة تزيد قليلًا عن عام. في ذلك الوقت، عملت على برامج مختلفة وقد تعلمت الكثير بالفعل. بعض المهام البرمجية التي نتعامل معها معقدة للغاية - لقد وجدت أنني لا أستطيع الجلوس أمام لوحة المفاتيح والبدء في كتابة كود البرنامج دون التفكير بشكل صحيح في الحل. عندما بدأت لأول مرة، شجعني مديري على رسم مخططات انسيابية لتبسيط المنطق التجاري المعقد إلى خطوات بسيطة. أجد أن هذه الطريقة ناجحة، وأعود إلى مخططاتي الانسيابية كثيرًا في أثناء عملي على الجزء الآتي من المشكلة.

عندما بدأت، كنت أعرف فقط Microsoft C#. كنت قلقًا عندما طُلب مني التطوير بلغات أخرى مثل جافا وPHP. بعد اللعب بهذه اللغات، اكتشفت أنها تحتوي على بناء جملة مشابه للغة C#، والعديد من مفاهيم C# التي كنت أعرفها كانت سهلة الترجمة. أعتقد أنه يمكنك القول أنني بدأت أتعرَّف الأنماط المتكررة!

عندما أحتاج إلى وصف البرمجة لأصدقائي، أقول لهم إنها تتعلق بشكل أساسي بحل المشكلات. أقضي الكثير من وقتي في تبسيط المشكلات المعقدة إلى مشكلات أبسط ومحاولة اكتشاف أي أنماط في المشكلات التي أتعامل معها.

تركيز مهاراتك

كن قادرًا على التكيف

هناك العديد من لغات البرمجة المختلفة المستخدمة في الحوسبة والصناعة تتطور باستمرار. من المهم حقًا أن تحافظ على مهاراتك وتحديث معارفك بقدر الإمكان. للحصول على وظيفة كمطور مبتدئ في فريق تطوير البرمجيات، يجب أن تكون قادرًا على تكييف نفسك ومهاراتك لتلبية احتياجات السوق ومواكبة الاتجاهات الحالية في البرمجة.

إليك بعض الأسئلة لتطرحها على نفسك لمساعدتك في القيام بذلك:

- ما لغات البرمجة المطلوبة حاليًا؟
- ما أنواع المهارات المطلوبة؟ هل هناك اهتمام خاص بأنظمة تشغيل معينة أو أنواع معينة من التطبيقات؟
- هل هناك أي مهارات أخرى متعلقة بالبرمجة مطلوبة، مثل معرفة تقنيات الويب أو قواعد البيانات العلائقية؟
- ما الخبرة والمعرفة بلغات البرمجة المختلفة المتوقعة من مطور مبتدئ؟ ما المهارات المذكورة في أوصاف وظائف تطوير البرمجيات والشواغر؟
- هل يمكنني نقل مهارات البرمجة الخاصة بي إلى لغات جديدة وبيئات تطوير مختلفة؟