הגדרת נהלים ושימוש חוזר בלוקים

איור 21-1.

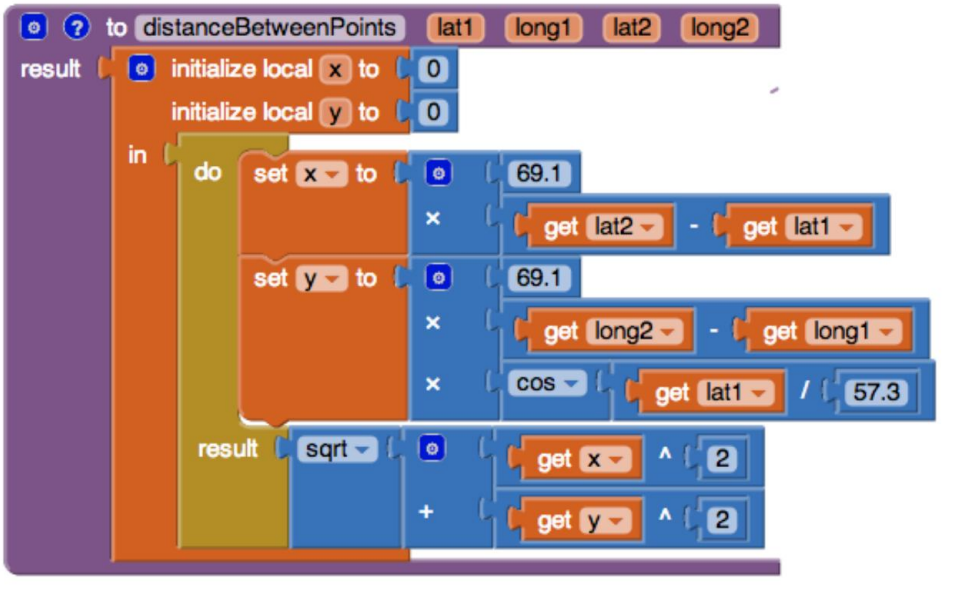
שפות תכנות כגון App Inventorמספקות סט בסיסי של פונקציונליות מובנית -במקרה של App Inventor, סט בסיס של בלוקים. שפות תכנות מספקות גם דרך להרחיב את הפונקציונליות על ידי הוספת פונקציות (בלוקים) חדשות לשפה. Inventor, App-באתה עושה זאת על ידי הגדרת נהלים - המכונים רצפים של בלוקים -שהאפליקציה שלך יכולה לקרוא להם בדיוק כפי שהיא קוראת לחסימות המוגדרות מראש של App Inventor. כפי שתראו בפרק זה, היכולת ליצור הפשטות כאלה חשובה מאוד לפתרון בעיות מורכבות, שהיא אבן היסוד של בניית אפליקציות מושכות באמת.

כשהורים אומרים לילדם, "לך לצחצח שיניים לפני השינה", הם באמת מתכוונים, "קח את מברשת השיניים ומשחת השיניים שלך מהארון, סחט קצת משחת שיניים על המברשת, סובב את המברשת על כל שן למשך 10שניות (חה!) ," וכן הלאה. "צחצח שיניים" הוא הפשטה: שם מוכר לרצף של הוראות ברמה נמוכה יותר. במקרה זה, ההורים מבקשים מהילד לבצע את ההוראות שכולם הסכימו שמשמעותן "לצחצח שיניים".

בתכנות, אתה יכול ליצור רצפים בעלי שם כאלה של הוראות. שפות תכנות מסוימות מכונות אותן פונקציות או תת-תוכניות. App Inventor-בהם נקראים פרוצדורות. פרוצדורה היא רצף שם של בלוקים שניתן להתקשר אליהם מכל מקום באפליקציה.

איור 21-11הוא דוגמה לנוהל שמעריך את המרחק, במיילים, בין שתי קואורדינטות GPSשאתה שולח אליו.

1בלוקים אלה מוצגים עם כניסות מוטבעות, מה שמקטין את רוחב הבלוקים. אתה יכול ללחוץ לחיצה ימנית על בלוקים כדי לעבור בין קלט "מוטבע" ו"חיצוני".



איור 21-2. נוהל לחישוב המרחק בין נקודות

אל תדאג יותר מדי לגבי הפנימיות של הליך זה עדיין; כל מה שאתה צריך מבין כרגע שהליכים כמו זה מאפשרים לך להרחיב את השפה שבה אתה מעצב ובונה תוכניות. אם כל הורה היה צריך להסביר את השלבים של "לצחצח שיניים" לילד שלו או שלה בכל לילה, ייתכן שהילד הזה לא יגיע לכיתה ה'. זה הרבה יותר יעיל להגיד "צחצח שיניים", וכולם יכולים להמשיך ללכת לישון בשעה סבירה.

באופן דומה, לאחר שתגדיר את הפרוצדורה מרחק בין נקודות, תוכל להתעלם את הפרטים של איך זה עובד ופשוט מתייחסים (קוראים) לשם הפרוצדורה בעת עיצוב או קידוד של אפליקציה גדולה יותר. סוג זה של הפשטה הוא המפתח לפתרון בעיות גדולות ומאפשר לנו לפרק פרויקט תוכנה גדול לנתחי קוד ניתנים לניהול.

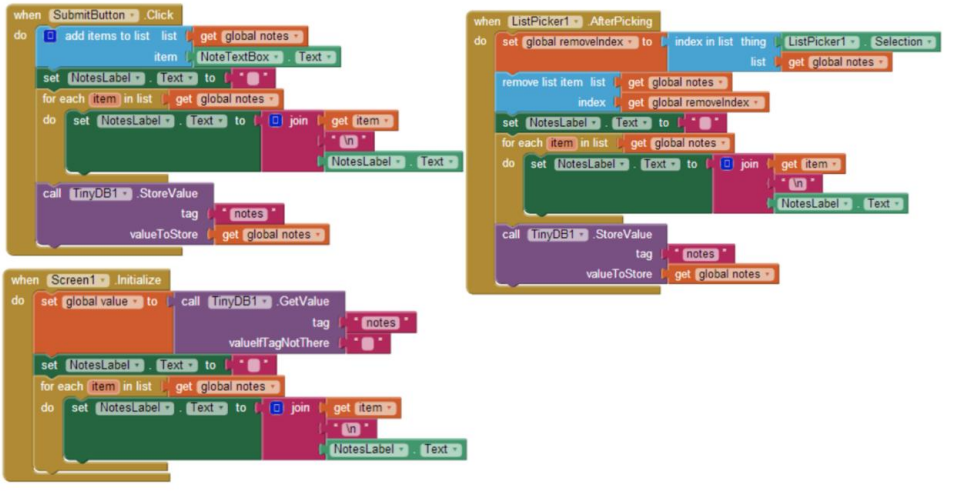
נהלים גם עוזרים להפחית שגיאות מכיוון שהם מבטלים יתירות בקוד שלך. עם נהלים, אתה יכול לשים גוש קוד במקום אחד ואז לקרוא לו ממקומות שונים ברחבי האפליקציה שלך. לכן, אם אתה בונה אפליקציה שצריכה לדעת את המרחק המינימלי בין המיקום הנוכחי שלך ל-01 נקודות אחרות, אינך צריך שיהיו לך 10עותקים של הבלוקים המוצגים באיור 21-1. במקום זאת, אתה פשוט מגדיר את נוהל ה- distanceBetweenPointsואז קורא לו בכל פעם שאתה צריך. האלטרנטיבה -העתקה והדבקה של בלוקים -תלויה בקוד הרבה יותר (זכור את הדיון מפרק 19)וכתוצאה מכך, מועדת לשגיאות, כי כאשר אתה מבצע שינוי, אתה צריך למצוא את כל שאר העותקים של אותם בלוקים ולשנות כל אחד מהם. אחד באותו אופן. תאר לעצמך שאתה מנסה למצוא את 5עד 10המקומות שבהם הדבקת א

גוש קוד מסוים באפליקציה עם 1,000שורות או בלוקים! פרוצדורה מאפשרת לך במקום זאת לכלול בלוקים במקום אחד, ואז לקרוא לו פעמים רבות.

נהלים גם עוזרים לך לבנות ספריית קוד שניתן לעשות בה שימוש חוזר ברבים אפליקציות. גם כאשר בונים אפליקציה למטרה מאוד ספציפית, מתכנתים מנוסים תמיד חושבים על דרכים ליצור את הקוד בצורה כזו שתוכל לעשות בו שימוש חוזר באפליקציות אחרות. חלק מהמתכנתים אף פעם לא יוצרים אפליקציות, אלא מתמקדים אך ורק בבניית ספריות קוד לשימוש חוזר עבור מתכנתים אחרים לשימוש באפליקציות שלהם!

ביטול יתירות

הבלוקים באיור 2-21הם מאפליקציית Note Taker. תסתכל על הבלוקים וראה אם אתה יכול לזהות את המיותרים.



איור 21-3. אפליקציית הערות עם קוד מיותר

הבלוקים המיותרים הם אלה הכוללים aעבור כל בלוק (למעשה עבור כל אחד והבלוקים המקוננים שלו והסט NotesLabel.Textשמעליו). בשלושתם עבור כל מקרים, תפקידו של הבלוק הוא להציג את רשימת ההערות. באפליקציה זו, התנהגות זו צריכה להתרחש בשלושה מטפלי אירועים: כאשר פריט חדש מתווסף, כאשר פריט מוסר, וכאשר הרשימה נטענת ממסד הנתונים עם השקת האפליקציה.

כאשר מתכנתים מנוסים רואים יתירות כזו, פעמון נדלק בראשם, כנראה עוד לפני שהם העתיקו והדביקו את הבלוקים מלכתחילה.

הם יודעים שעדיף לכלול יתירות כזו בהליך, גם כדי

להפוך את התוכנית למובנת יותר וכך יהיה הרבה יותר קל לבצע שינויים מאוחר יותר.

בהתאם לכך, מתכנת מנוסה ייצור נוהל, יעביר עותק
של הבלוקים המיותרים לתוכו, ולאחר מכן קרא להליך משלושת המקומות המכילים את
הבלוקים המיותרים. האפליקציה לא תתנהג בצורה שונה, אבל היא תהיה קלה יותר לתחזוקה
וקל יותר למתכנתים אחרים לעבוד איתם. ארגון מחדש של קוד (בלוק) כזה נקרא refactoring.

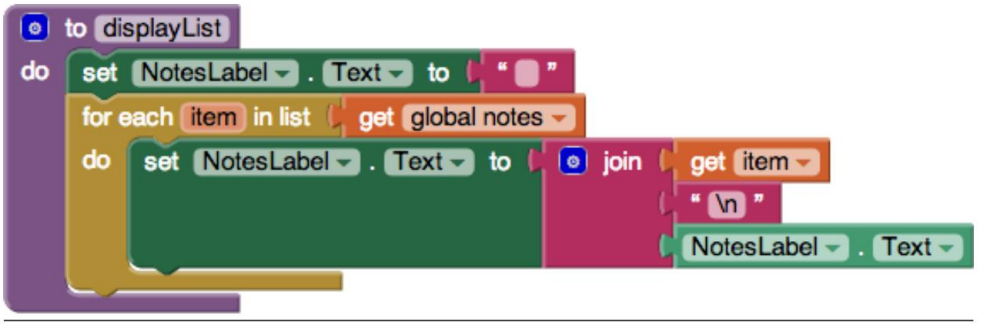
הגדרת נוהל

בואו נבנה נוהל לעשות את העבודה של בלוקי הקוד המיותרים מאיור 21-2. ב-App Inventor,
אתה מגדיר הליך באופן דומה לאופן שבו אתה מגדיר משתנים.
מהמגירה של נהלים, גרור החוצה או בלוק להליך או בלוק תוצאת להליך . השתמש באחרון אם
ההליך שלך אמור לחשב ערך כלשהו ולהחזיר אותו (נדון בגישה זו מעט מאוחר יותר בפרק).

לאחר גרירת גוש אל נוהל , תוכל לשנות את שם ברירת המחדל שלו על ידי לחיצה על המילה
"פרוצדורה" והקלדת שם חדש. הבלוקים המיותרים שברצונך לשחזר מבצעים את עבודת הצגת
הרשימה, אז נקרא להליך displayList, המוצג באיור 21-3.

איור 21-4. לחץ על "הליך" כדי לתת שם להליך שלך

השלב הבא הוא הוספת הבלוקים בתוך ההליך. במקרה זה, אנו משתמשים בלוקים שכבר
קיימים, אז נגרור את אחד מהבלוקים המיותרים המקוריים ממטפל האירועים שלו ונמקם אותו
בתוך הבלוק to displayList, כפי שמוצג באיור 21-4.

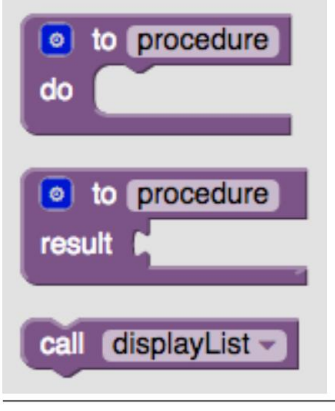


איור .21-5הליך displayListמקפל את הקוד המיותר

קריאה לנוהל

נהלים, כמו displayListו"צחצוח שיניים", הם ישויות עם פוטנציאל לבצע משימה. עם זאת, הם יבצעו את המשימה הזו רק אם הם יידרשו לעשות זאת. עד כה, יצרנו נוהל אך לא קראנו לו. לקרוא להליך פירושו להפעיל אותו, או לגרום לו לקרות.

בכאשר אתה מגדיר נוהל, App Inventor, הוספת חסימת שיחה אוטומטית למגירת הנהלים כפי שמוצג באיור .21-5



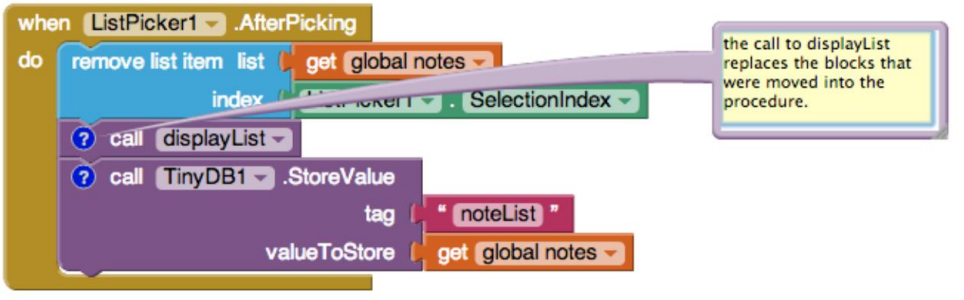
איור .21-6חסימת שיחה מופיעה במגירה של נהלים כאשר אתה מגדיר נוהל

כבר השתמשת בחסימות שיחות כדי להתקשר לפונקציות המוגדרות מראש של App Inventor, כגון Ball.MoveToו- .Texting.SendMessageכאשר אתה מגדיר נוהל, בעצם יצרת בלוק משלך; הרחבת את שפת ממציא האפליקציות.

באמצעות חסימת השיחה החדשה , אתה יכול להפעיל את היצירה שלך.

לדוגמא של אפליקציית Note Taker, תגרור שלושה בלוקים של שיחות displayList ולהשתמש בהם כדי להחליף את הקוד המיותר בשלושת מטפלי האירועים. למשל, ה

יש לשנות את המטפל באירוע ListPicker1.AfterPicking(למחיקת הערה) כפי שמוצג באיור 21-6.



איור 21-7. שימוש בקריאה displayListכדי להפעיל את החסימות כעת בהליך

מונה התוכניות

כדי להבין כיצד פועל בלוק השיחה , חשבו על אפליקציה כבעלת מצביע שעובר דרך הבלוקים שמבצעים פונקציות. במדעי המחשב, מצביע זה נקרא מונה התוכניות.

כאשר מונה התוכנית מבצע את החסימות בתוך מטפל באירועים והוא מגיע לבלוק קריאה , הוא קופץ לפרוצדורה ומבצע את החסימות שבו. כאשר ההליך מסתיים, מונה התוכנית קופץ חזרה למיקומו הקודם (גוש השיחה) וממשיך משם. אז, עבור הדוגמה של Note Taker, בלוק הסר פריט רשימה מבוצע; לאחר מכן מונה התוכנית קופץ לנוהל displayListומבצע את הבלוקים בהליך זה (הגדרת NotesLabel.Textלטקסט הריק, וה- עבור כל אחד מהם); ולבסוף מונה התוכנית חוזר לבצע את בלוק TinyDB1.StoreValue .

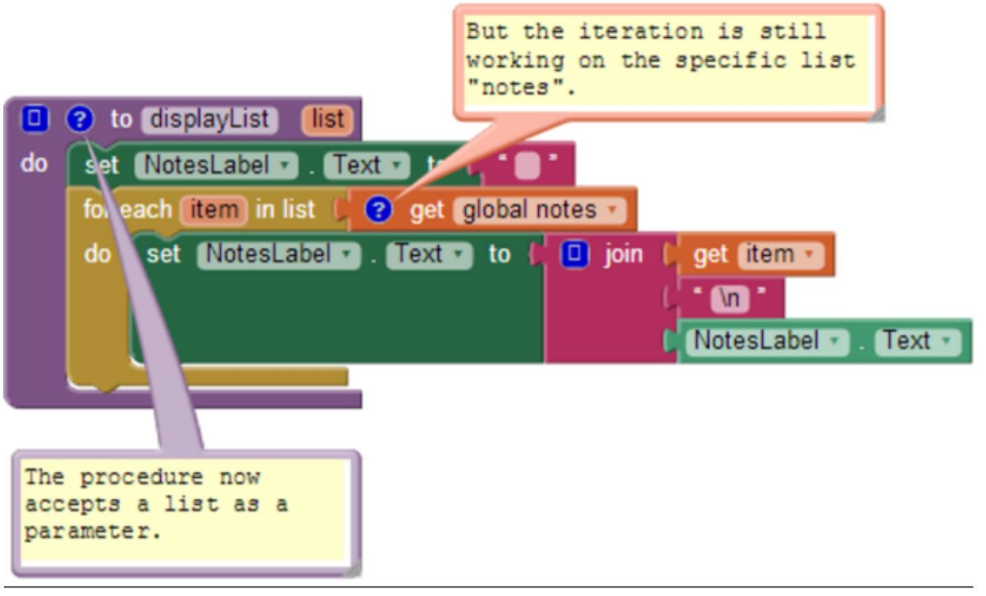
הוספת פרמטרים לנוהל שלך

הליך displayListמאפשר להחזיר קוד מיותר למקום אחד. קל יותר להבין את האפליקציה מכיוון שאתה יכול לקרוא את מטפלי האירועים ברמה גבוהה ובאופן כללי להתעלם מהפרטים של אופן הצגת רשימה. זה גם מועיל מכיוון שאתה עשוי להחליט לשנות את אופן הצגת הרשימה, וההליך מאפשר לך לבצע שינוי כזה במקום אחד (במקום שלושה).

עם זאת, לנוהל displayListיש מגבלות מבחינת התועלת הכללית שלו. ההליך פועל רק עבור רשימה ספציפית (הערות) ומציג רשימה זו בתווית ספציפית (NotesLabel). לא יכולת להשתמש בו כדי להציג רשימת נתונים שונה - לדוגמה, רשימה של משתמשי האפליקציה - מכיוון שהיא מוגדרת באופן ספציפי מדי.

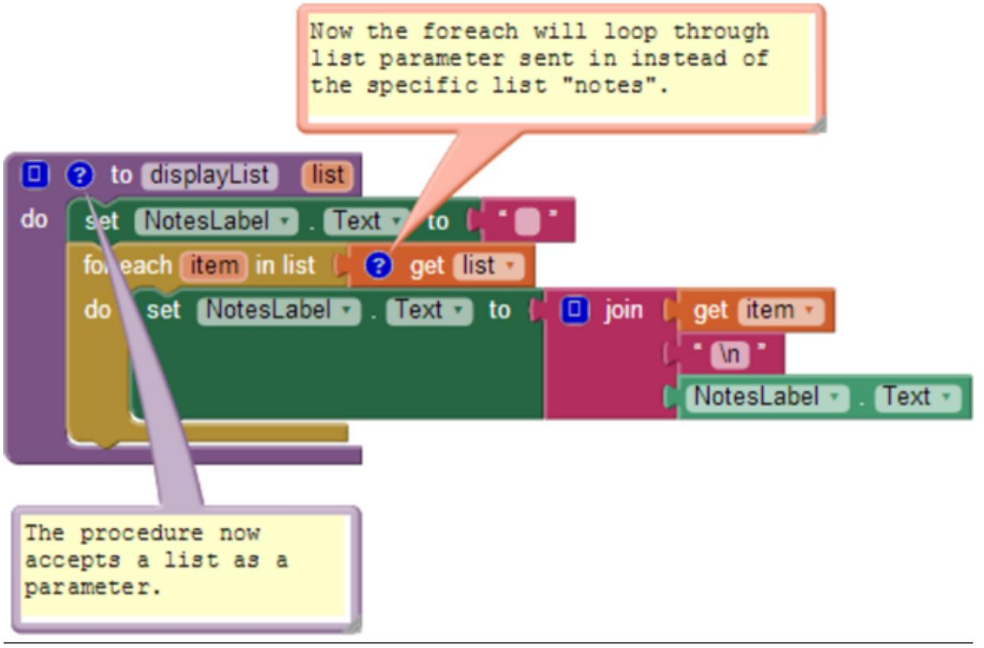
App Inventorושפות אחרות מספקות מנגנון שנקרא פרמטרים להפיכת נהלים לשימוש כלליים יותר. פרמטרים כוללים את המידע שהליך צריך כדי לבצע את עבודתו. הם מספקים את המפרטים של אופן ביצוע ההליך. בדוגמה שלנו לצחצוח שיניים לפני השינה, תוכל להגדיר "סוג משחת שיניים" ו"זמן צחצוח" כפרמטרים של הליך "צחצוח שיניים".

אתה מגדיר פרמטרים להליך על ידי לחיצה על הסמל הכחול בפינה השמאלית העליונה של הגדרת הנוהל. עבור הליך , displayListנגדיר פרמטר בשם "רשימה", כפי שמוצג באיור .21-7



איור .21-8ההליך מקבל כעת רשימה כפרמטר

אפילו כשהפרמטר מוגדר, הבלוקים עדיין מתייחסים ישירות לרשימה הספציפית הערות (הוא מחובר לחריץ "ברשימה" של עבור כל אחד מהם). מכיוון שאנו רוצים שהפרוצדורה תשתמש ברשימה שאנו שולחים כפרמטר, אנו מחליפים את ההפניה להערות גלובליות בהפניה לקבלת רשימה, כפי שהודגם באיור .21-8



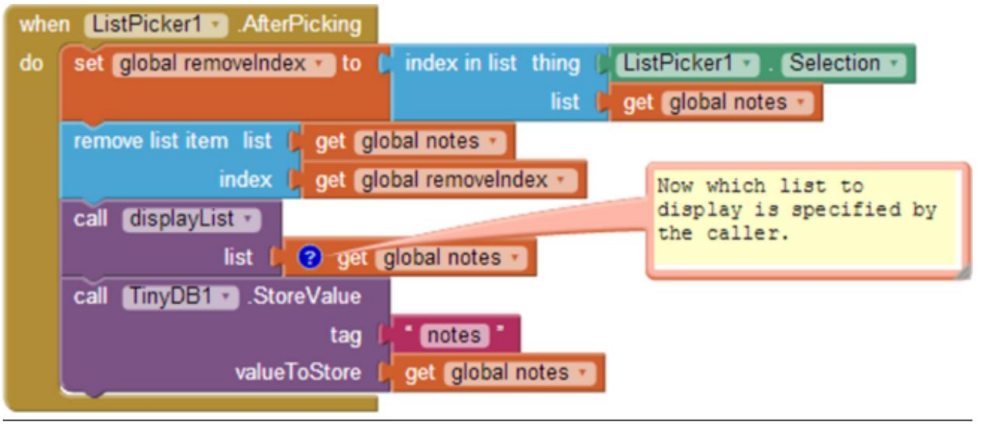
איור 21-9. כעת עבור כל אחד ישתמשו ברשימה שנשלחה

הגרסה החדשה של הנוהל היא יותר כללית: קריאות ל- displayListיכולות כעת שלח לו רשימה כלשהי, ו- displayListיציג אותה. כאשר אתה מוסיף פרמטר לפרוצדורה, App Inventorמכניס אוטומטית שקע מתאים בבלוק השיחה .
לכן, כאשר רשימת הפרמטרים מתווספת ל- displayList, בלוקי הקריאה ל- displayListנראים כמו איור 21-9.

איור 21-10. קריאה displayList-למחייבת כעת לציין איזו רשימה להציג

רשימת הפרמטרים בתוך הגדרת הפרוצדורה נקראת פרמטר פורמלי.
השקע המתאים בתוך בלוק השיחה נקרא פרמטר ממשי. כאשר אתה קורא לנוהל ממקום כלשהו באפליקציה, עליך לספק פרמטר בפועל עבור כל פרמטר פורמלי של ההליך. אתה עושה זאת על ידי זרימת כל השקעים בשיחה.

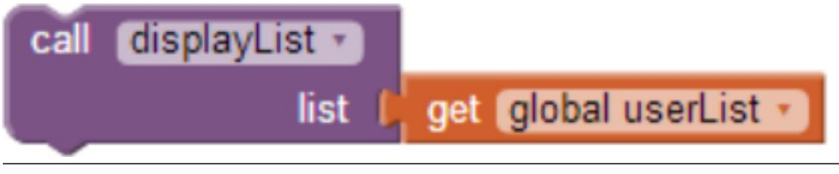
עבור האפליקציה Note Taker, אתה מוסיף הפניה לקבלת הערות גלובליות כפרמטר בפועל. איור 21-10 מראה כיצד יש לשנות את ListPicker.AfterSelection .



איור 21-11. קריאה displayList-לעם הערות שנשלחו כפרמטר בפועל

כעת, כאשר נקרא displayList, הערות הרשימה נשלחות אל הפרוצדורה וממוקמות ברשימת הפרמטרים. מונה התוכנית ממשיך לבצע את הבלוקים בהליך, תוך התייחסות לרשימת הפרמטרים אך באמת עובד עם המשתנה הערות.

בגלל הפרמטר, אתה יכול כעת להשתמש בהליך displayList עם כל רשימה, לא רק הערות. לדוגמה, אם אפליקציית Note Taker שותפה בין רשימת משתמשים וברצונך להציג את רשימת המשתמשים, תוכל להתקשר displayList-לולשלוח לה את userList, כפי שמוצג באיור 21-11.



איור 21-12. כעת ניתן להשתמש בהליך displayList כדי להציג כל רשימה, לא רק הערות

החזרת ערכים מהליך

עדיין יש בעיה אחת בהליך displayList מבחינת התועלת הכללית שלו - האם אתה יכול להבין מה זה? כפי שזה כתוב כרגע, זה יכול להציג כל רשימה של נתונים, אבל זה תמיד יציג את הנתונים האלה בתווית NotesLabel. מה אם תרצה שהרשימה תוצג באובייקט ממשק משתמש אחר (למשל, הייתה לך תווית שונה להצגת ה- userList)?

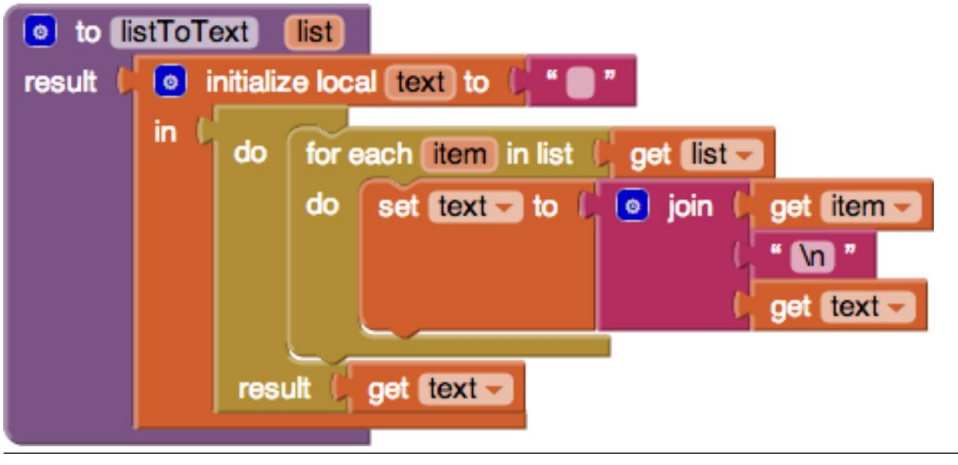
פתרון אחד הוא להגות מחדש את הפרוצדורה ולשנות את תפקידו מהצגת רשימה בתווית מסוימת לפשוט החזרת אובייקט טקסט שאתה יכול

להציג בכל מקום. לשם כך, אתה משתמש בבלוק תוצאת פרוצדורה , המתואר באיור 21-12, במקום בלוק הפרוצדורה .

איור 21-13. חסימת תוצאת ההליך

אתה תבחין שבהשוואה לחסום ההליך , תוצאת ההליך לבלוק יש שקע נוסף בתחתית. אתה מציב משתנה במשבצת הזו והוא מוחזר למתקשר. לכן, בדיוק כפי שהמתקשר יכול לשלוח נתונים לפרוצדורה עם פרמטר, פרוצדורה יכולה לשלוח נתונים בחזרה עם ערך החזרה.

איור 21-13 מציג את הגרסה המחודשת של ההליך הקודם, הפעם באמצעות בלוק תוצאת פרוצדורה . שים לב שמכיוון שההליך עושה כעת עבודה שונה, השם שלו משתנה מ- displayList ל- listToText.



איור 21-14. listToText מחזיר אובייקט טקסט שהמתקשר יכול למקם בכל תווית

בבלוקים המוצגים באיור 21-13, טקסט משתנה מקומי מוגדר להחזיק את הנתונים כאשר ההליך חוזר על כל פריט ברשימה. טקסט מאותחל כמשתנה מקומי, במקום משתנה גלובלי, מכיוון שהוא משמש רק בהליך זה.

משתנה טקסט זה מחליף את רכיב NotesLabel הספציפי מדי שהיה בשימוש בגרסת displayList של הליך זה. כאשר עבור כל אחד מסתיים, הטקסט המשתנה מכיל את פריטי הרשימה, כאשר כל פריט מופרד על ידי תו שורה חדשה, \n (למשל, "item1\nitem2\item3"). משתנה טקסט זה מחובר לאחר מכן לשקע ערך ההחזרה.

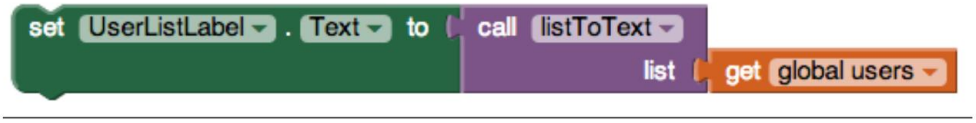
כאשר תוצאת הליך מוגדרת, בלוקי השיחה המתאימים לה נראים שונים
מאשר אלה עבור הליך. השווה את הקריאה ל- listToTextעם הקריאה ל- displayList
באיור .21-14

איור .21-15הקריאה מימין מחזירה ערך ולכן חייבת להיות מחוברת למשהו

ההבדל הוא שלרשימת השיחהtxeToT יש תקע בצד שמאל. הסיבה לכך היא שכאשר
הקריאה מבוצעת, ההליך יעבור את המשימה שלו ואז יחזיר ערך לבלוק הקריאה . ערך ההחזר
הזה חייב להיות מחובר למשהו.
במקרה זה, המתקשרים ל- displayListיכולים לחבר את ערך ההחזר הזה לכל תווית שהם
רוצים. עבור הדוגמה של ,Note Takerשלושת מטפלי האירועים שצריכים להציג רשימה
יתקשרו לפרוצדורה, כפי שמוצג באיור .21-15

איור .21-16המרת הערות הרשימה לטקסט והצגתן NotesLabel-ב

הנקודה החשובה כאן היא שמכיוון שההליך הוא גנרי לחלוטין ו
אינו מתייחס לרשימות או תוויות כלשהן באופן ספציפי, חלק אחר של האפליקציה יכול
להשתמש בו כדי להציג כל רשימה בתווית כלשהי, כפי שמוצג באיור .21-16



איור .21-17ההליך אינו קשור עוד לרכיב תווית מסוים

שימוש חוזר בלוקים בין אפליקציות

אין צורך להגביל שימוש חוזר בלוקי קוד באמצעות נהלים לאפליקציה אחת. ישנם נהלים רבים,
כגון ,listToTextשבהם תוכל להשתמש כמעט בכל אפליקציה שתיצור. בפועל, ארגונים וקהילות
תכנות בונים ספריות קוד של נהלים עבור תחומי העניין שלהם.

בדרך כלל, שפות תכנות מספקות כלי ייבוא שדרכו ניתן לכלול קוד ספרייה בכל אפליקציה. ל-ppA Inventorעדיין אין כלי עזר כזה. הדרך היחידה לשתף נהלים היא ליצור אפליקציית ספרייה מיוחדת ולהתחיל בפיתוח אפליקציה חדשה על ידי שמירת עותק חדש של אותה אפליקציה ועבודה ממנה.

נוהל המרחק בין נקודות

עם הדוגמה של , displayList (listToText)אפיינו את הגדרת הפרוצדורה כדרך לחסל קוד מיותר: אתה מתחיל לכתוב קוד, מוצא יתירות תוך כדי, ומשנה את הקוד שלך כדי לחסל אותם. בדרך כלל, עם זאת, מפתח תוכנה או צוות יעצבו אפליקציה מההתחלה תוך מחשבה על נהלים וחלקים לשימוש חוזר. סוג זה של תכנון יכול לחסוך לך זמן משמעותי ככל שהפרויקט מתקדם.

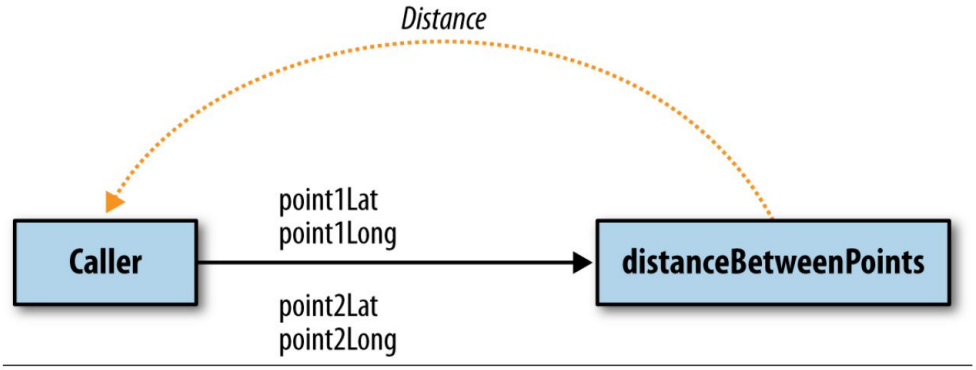
שקול אפליקציה כדי לקבוע את בית החולים המקומי הקרוב ביותר למיקומו הנוכחי של המשתמש -משהו שיהיה שימושי מאוד במקרה חירום. להלן תיאור עיצוב ברמה גבוהה של האפליקציה:

כאשר האפליקציה מופעלת, מצא את המרחק, במיילים, בין המיקום הנוכחי לבית החולים הראשון. ואז מצא אותו לבית החולים השני, וכן הלאה. כאשר יש לך את המרחקים, קבע את המרחק המינימלי והצג את הכתובת (ו/או מפה) למיקום זה.

מהתיאור הזה, האם אתה יכול לקבוע את ההליכים שהאפליקציה הזו צריכה? לעתים קרובות, הפעלים בתיאור כזה מרמזים על ההליכים שתזדקקו להם. חזרה בתיאור שלך, כפי שצוין עם ה-"וכן הלאה," הוא רמז נוסף. במקרה זה, איתור המרחק בין שתי נקודות וקביעת המינימום של מרחקים מסוימים הם שני הליכים הכרחיים.

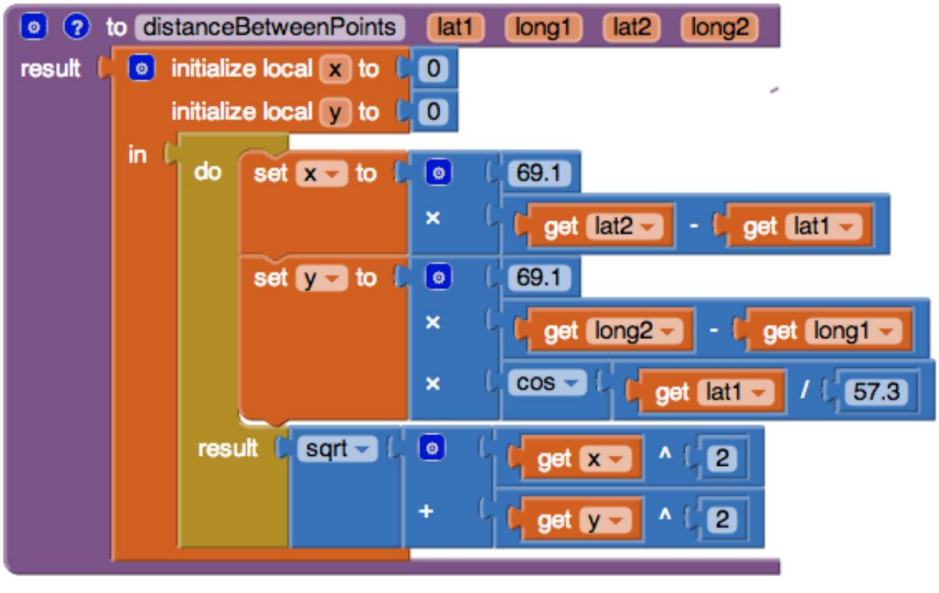
בוא נחשוב על העיצוב של הנוהל למציאת המרחק בין שתי נקודות, שנקרא ל- ,fne) distanceBetweenPointsאז מקוריות היא לא הצד החזק שלי). בעת תכנון פרוצדורה, צריך לקבוע את התשומות והפלטים שלו: הפרמטרים שהמתקשר ישלח לנוהל כדי שיבצע את עבודתו, ואת ערך התוצאה שהפרוצדורה תשלח בחזרה למתקשר. במקרה זה, המתקשר צריך לשלוח את קו הרוחב והאורך של שתי הנקודות להליך, כפי שמוצג באיור .21-17

תפקידו של הנוהל הוא להחזיר את המרחק, בקילומטרים.



איור 21-18. המתקשר שולח ארבעה פרמטרים קלט ומקבל מרחק

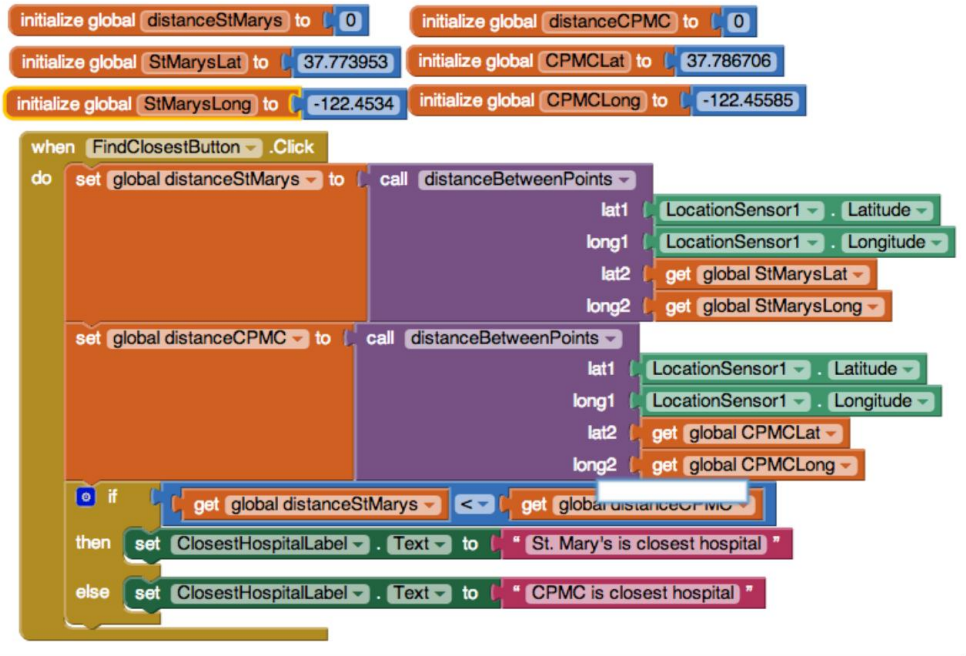
איור 21-18 מציג את ההליך שנתקלנו בו בתחילת הפרק,
שימוש בנוסחה לקירוב הקילומטראז' בין שתי קואורדינטות GPS.



איור 21-19. נוהל distanceBetweenPoints

איור 21-19 מציג בלוקים שמבצעים שתי קריאות לפרוצדורה, כל אחת מהן
מאתר את המרחק מהמיקום הנוכחי לבית חולים מסוים.
עבור השיחה הראשונה, הפרמטרים בפועל עבור הנקודה הראשונה הם הקריאות
הנוכחיות מ- LocationSensor, בעוד שהנקודה השנייה מוגדרת על ידי קואורדינטות ה-SPG
עבור בית החולים St. Mary's. הערך המתקבל ממוקם במשתנה distanceStMarys.
השיחה השנייה דומה אך במקום זאת משתמשת בנתונים עבור בית החולים CPMC עבור
הנקודה השנייה.

האפליקציה ממשיכה להשוות את שני המרחקים שהוחזרו כדי לקבוע איזה בית חולים הכי קרוב. אבל, אם היו יותר בתי חולים מעורבים, באמת היית צריך להשוות רשימה של מרחקים כדי למצוא את הקצר ביותר. ממה שלמדת, האם אתה יכול ליצור הליך בשם findMinimum שמקבל רשימת מספרים כפרמטר ומחזיר את האינדקס של המינימום?



איור 21-20. שתי קריאות לנוהל distanceBetweenPoints

סיכום

שפות תכנות כגון App Inventor מספקות סט בסיסי של פונקציונליות מובנית. באמצעות נהלים, ממציאי אפליקציות יכולים להרחיב את השפה הזו עם הפשטות חדשות. App Inventor לא מספק חסימה להצגת רשימה, אז אתה בונה אחת. צריך בלוק לחישוב המרחק בין קואורדינטות GPS?

אתה יכול ליצור משלך.

היכולת להגדיר בלוקים של פרוצדורות ברמה גבוהה יותר היא המפתח להנדסה גדולה, תוכנה הניתנת לתחזוקה ופתרון בעיות מורכבות מבלי להיות מוצף כל הזמן מכל הפרטים. נהלים מאפשרים לך להקיף בלוקי קוד ולתת שם לבלוקים האלה. בזמן שאתה מתכנת את ההליך, אתה מתמקד אך ורק בפרטים של אותם בלוקים. עם זאת, בתכנות שאר האפליקציה, כעת יש לך הפשטה -שם -שאתה יכול להתייחס אליו ברמה גבוהה.