nclab
More Than Coding

KAREL JR ANSWER KEYS FOR STUDENT JOURNALS

REVISED OCTOBER 6, 2016

TABLE OF CONTENTS

UNIT 1 SECTIONS 1-5

SECTION 1

MANUAL MODE, BASIC FEATURES

PAGE 1 – 5

KEYWORD (COMMAND)	KEYBOARD KEY	SCREEN BUTTON	FUNCTION
`go`	Up arrow ↑		Move Karel one step forward
`right`	Right arrow →		Karel turns to his right
`left`	Left arrow ←		Karel turns to his left
`Get`	Shift key		Karel picks up the object on the square he occupies.
`put`	Control key		Karel puts an object into a container on the square he occupies.

FACTOR	WHAT IT MEASURES	WHY IT IS IMPORTANT
Number of operations	Everything Karel does is an operation: go, left, right, get, put.	More operations means a longer program. A program with a lot of operations is also difficult to understand and troubleshoot.
Number of steps	The number of squares Karel travels to get to his destination	Planning the shortest path will make the program more efficient and save time.
Amount of time	The run time of the program	Programs that take a long time use more computer resources such as processing and memory. They tie up network connections. They may not work well with other programs that need a fast response time. They are boring to the user who has to wait.

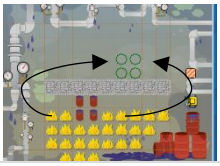
SECTION 1

PAGE 1 - 6

Many of these levels restricted the number of steps you could take. Did you plan ahead, or just keep trying until you were successful? How can you plan the number of steps to stay less than or equal to the maximum allowed?

Answers will vary – reflect student experience. Planning examples – read instructions, count squares, visualize the path before running the program, discuss alternatives with a partner

Discuss at least two different pathways through the maze to complete Level 1.6 "Fire!" Is there any advantage to using one rather than the other?



Answers will vary. Sample: After picking up the four drums, Karel could continue around the west side of the barrier, or turn back and go around the east side. Either way is 44 operations, so it makes no difference. (unless a student can prove fewer operations with their method).

What pattern was needed to complete Level 1.7 "Flowers" within 13 steps? Would you have chosen this pattern without the fences to guide you?

Answers will vary (in the past, the game did not have the fences. Students rarely used this pattern first and ended up using too many steps.)

SECTION 2

BASIC COMMANDS AND SYNTAX

PAGE 1 - 9

INCORRECT CODE	CORRECTED	WHAT ERRORS WERE MADE?

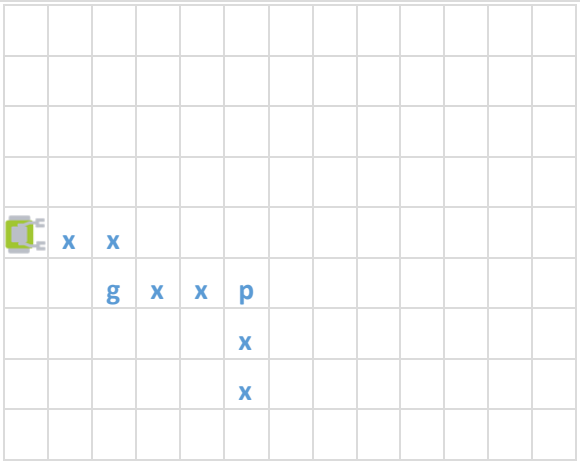
`go go get`
`go`
`right`
`put`
	`go`
	`go`
	`get`
	`go`
	`right`
	`put`
		More than one command was written on the same line

`go`
`  get`
`go`
`  left`
`go`
`go`
	`go`
	`get`
	`go`
	`left`
	`go`
	`go`
		Indentations have meaning in Karel. A new command line should not be indented.

Write the code needed for Karel to move forward two steps, turn right, move forward one step, pick up an object, turn left, move forward three steps, put the object into a container, turn right, move two steps. What will this look like when you run the program? How many operations are there? _13_____ *How many steps did Karel take?* ___8_____

Program

```
go
go
right
go
get
left
go
go
go
put
right
go
go
```



SECTION 2

PAGE 1 - 10

What happens when you give a robot a command that is not correct? Give an example.

- You may get an error message when you try to run the program.
  Example: misspell go, get this error message: "gv is not a known command. Line 2: gv"
- Karel might crash into a wall. Example: writing right instead of left.
- You may get a message at the end of the program such as "Not All Objects Collected" Example: not including a get command.

Think of a simple procedure you do every day, like putting your books away, eating lunch, getting dressed. How could you write code for such a procedure using `go`, `left`, `right`, `get`, `put`*?*

Answers will vary. Student should describe what the code is doing.

In Karel, what are SHIFT-ENTER and the eraser button used for?

Shift-Enter creates a space to enter a line of code above the line you are on. The Eraser button **(note: this has been changed to the two green arrows which restore the original default code – will be revised in the next edition)** erases all the code.

SECTION 3

REPEAT LOOPS

PAGE 1 - 13

Let's review vocabulary. Match each term to the correct definition.

TERM/DEFINITION
1. h
2. g
3. b
4. d
5. f
6. c
7. e
8. a

Looking for patterns in programs takes study and planning. What repeated patterns do you see in this picture?

Answers will vary:

Repeated patterns include the skewers themselves; the different combinations of meat and vegetables; the way the grill is made.

SECTION 3

PAGE 1 - 14

What syntax do you use when writing `repeat` loops?

Indent the body of the loop by two spaces.

Cooking often requires repeated procedures: putting cookie dough in rows on a baking sheet is one example. Can you think of others?

Answers will vary.

```
go          vs.          repeat 2          Both use two lines. When would a repeat 2 loop be useful?
go                         go
```

The loop is useful when the set of commands in the body takes more than one line.

SECTION 4

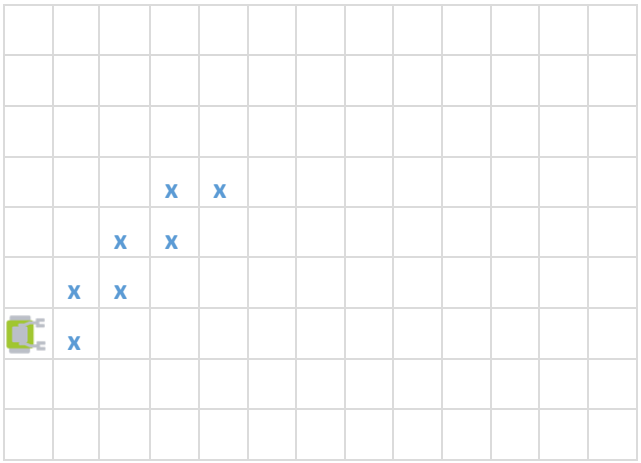
REPEAT LOOPS EMBEDDED IN PROGRAMS

PAGE 1 - 17

Compare these two examples with different indentations. Draw Karel's path for each one.

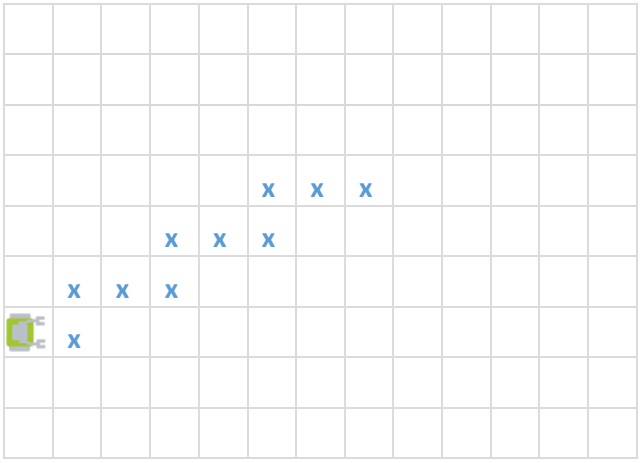
Program 1

```
repeat 3
  go
  left
  go
  right
go
```



Program 2

```
repeat 3
  go
  left
  go
  right
  go
```



Write a general rule for indentation with loops. Explain how to start the loop, and how to end it.

Sample answer: The repeat command is written as repeat x, where x is the number of times the commands will be repeated. The body of commands to be repeated is indented by 2 spaces for each line of command. The body ends when the indentation stops.

SECTION 4

PAGE 1 - 18

In 4.7, there were several possible solutions. The program can be written in 17, 16, or 15 lines.

Compare the solutions (discuss with a friend).

See solution manual for examples of 19, 16, and 15 line solutions. Answers will vary.

What is being done to reduce the number of lines?

More repeat loops reduce the number of lines.

Is the program more effective when it is shorter?

Answers will vary: Has 3 more operations; run time is about the same. So the speed is about the same. However, programs with repeat loops are easier to understand and edit.

SECTION 5

MULTIPLE LOOPS AND NESTED LOOPS

PAGE 1 - 21

Mark the indentation errors in the following programs.

Program 1 Separate command followed by a nested loop	Program 2 Nested loop followed by two separate commands	Program 3 Separate loops	Program 4 First, nested loops Then, a separate loop

`go`
`repeat 3`
`go`
`repeat 4`
`go`
`left`
`go`
`get`
`right`
`go`
	`repeat 2`
	`go`
	`left`
	`repeat 6`
	`go`
	`right`
	`go`
		`repeat 4`
		`go`
		`left`
		`go`
		`right`
		`go`
		`repeat 5`
		`go`
		`get`
			`repeat 6`
			`go`
			`left`
			`repeat 4`
			`go`
			`get`
			`repeat 10`
			`go`

In Program 1, line 3 needs 1 more indent space. Lines 8, 9, and 10 should be indented 2 spaces from the nested repeat 4.

In Program 2, Lines 2 and 3 should be indented 2 more spaces. Lines 5 should be indented 2 spaces right of repeat 6. Line 6 and 7 should not be indented.

In Program 3, repeat 5 should not be indented.

In Program 4, line 3 should be indented 2 spaces. Lines 5 and 6 should be indented 2 more spaces. Line 7 should not be indented. Line 8 should be indented 2 spaces.

Nested patterns occur in nature, art, engineering; anything that is made up of patterns within patterns. Describe the nested loops in this picture.

Sample Answer:

Each large black mark on the dial is followed by 4 small marks. Those small marks could be a repeat loop of 4 within the larger repeat loop of 12 large black marks.

SECTION 5

PAGE 1 - 22

How are multiplication and division similar to nested loops?

Answers will vary. The outer loops are a **multiple** of what is being created in the inner loop. Example: packing 12 crayons into a box (inner loop), then putting 12 boxes of crayons in a carton (outer loop).

A gardener plants a row of corn. She is planning 10 plants for the row. She puts three seeds in each hole, hoping that at least one will germinate and grow into a plant (in real life, these would be spaced closely, but for now, take a step each time you "plant" a seed). She spaces the holes apart by two steps. Use `go, left, right, put` to write a program to do this task.

```
repeat 10
  repeat 3
    go
    put
    go
  go
```

(students may include a couple of `go` commands to put spaces between the groups of seeds)

`left` and `right` are not needed for one row. They would be used to turn and move to the next row.

UNIT 2 SECTIONS 6-10

SECTION 6

IF CONDITIONS

PAGE 2 - 5

Time for a vocabulary check. Match each term to the correct definition.

TERM/DEFINITION
1. c
2. e
3. f
4. b
5. a
6. d

SENSOR TYPE	ACTION USUALLY TAKEN	IS KAREL IN THE SQUARE OR IN FRONT OF THE SQUARE WHEN HE DETECTS THE SENSOR?	EXAMPLES
Object	get	In the square	Picks up a pumpkin
Container	put	In the square	Puts the pumpkin in a basket
Obstacle	Avoid by moving around or along the obstacle	In front of the sensor	Detects a wall. Turns to the left.

SECTION 6

PAGE 2 - 6

For a sensor word to be blue-colored it must meet two conditions. What are they?

The sensor word must exist in the Karel library. It must be spelled correctly.

Give two examples of how conditions are used in Section 6.

Answers may vary. Sample: 6.5 If Karel senses a poisonous plant, then he goes around it to the right.

Compare conditional loops to `repeat` loops. When would you choose one over the other?

Repeat loops are used when a fixed definite number of repetitions is needed and known in advance. Conditional loops are used when Karel will only execute a set of operations under certain conditions, and we do not know whether those conditions exist.

SECTION 7

IF/ELSE CONDTIONS; NORTH SENSOR

PAGE 2 - 9

Think of `if/else` as a set of two choices depending on the presence or absence of a sensor. Think of studying for a math test. If you don't understand a type of problem, you will practice it; if you do understand it, you will choose a different type of problem to work on.

Describe two other real life conditions that branch into two choices:

Answers will vary. `Else` branches require a different set of actions based on the absence of the `if` condition.

The North Star has been used in navigation for thousands of years. By knowing where North is, we can angle off to any other direction. Karel uses `north` in a similar manner, but he is only allowed to turn 90 degrees at a time by using `left` or `right`. Fill in the table to describe how to end up with East, South, and West by turning right or left. Karel starts out by facing North.

DIRECTION WE WANT KAREL TO FACE	NUMBER OF **LEFT** TURNS NEEDED	NUMBER OF **RIGHT** TURNS NEEDED
East	3	1
South	2	2
West	1	3

SECTION 7

PAGE 2 - 10

Describe two `if/else` conditions from the Section 7 levels. What are the conditions and what are the outcomes? Pick two that you might use yourself when designing a maze.

Answers will vary. Sample: 7.5: Karel needs to place a spider on all the marks. Some marks are in his path, and some marks are to the left. If there is a mark, he places a spider on it. Else, he goes left, places the spider on that mark, turns around, goes back to the path, turns left and moves forward.

The games are starting to combine different kinds of loops. What would you use to solve the following situations?

Karel follows a wall 14 units long, checking for snakes. When he finds one, he goes around it. If he doesn't, he moves forward.

Note: Another solution would be to go to the right of the snake. In that case, all the right and left turns are opposite. Some students might take this a step further and build in a nested if condition to check for a snake in the square after the first snake, before returning to the main path.

```
repeat 14
  if snake
    left
    go
    right
    go
    go
    right
    go
    left
  else
    go
```

Karel is traveling east. He can't move forward unless he is facing east.

```
repeat 3
  if not north
    left
right
```

Karel must pick up twenty computer chips. Each time he picks one up, he turns left and goes one step. Otherwise, he goes forward one step.

```
repeat 11
 if chip
   get
   left
   go
 else
   go
```

SECTION 8

EMPTY SENSOR;

AND, NOT, OR KEYWORDS

PAGE 2 - 13

Check your understanding of how these keywords operate by completing the table.

LOGICAL OPERATOR KEYWORD	DESCRIBE THE CONDITION	EXAMPLE answers will vary
`not`	condition must not be met	`if not empty` `  put`
`and`	both or all of the conditions must be met	`if cart and (not empty)` `  put`
`or`	means that one (or a set of conditions within parentheses) of two or more conditions must be met	`if nugget or jewel` `  get`

`empty` refers to Karel's pocket, which may contain different amounts of an object. A common real-life problem is, of course, how much money you have in your wallet, or on your bank card, when you go shopping. Think of two other examples of checking a "not empty" condition.

ITEMS IN THE "POCKET"	WHAT HAPPENS IF THE POCKET IS NOT EMPTY? YOU CAN ALSO WRITE AN "ELSE" FOR WHEN IT IS EMPTY.
	Answers will vary.
Example: Sports drinks in beverage fridge	If the fridge is not empty, you can get a sports drink Else, you need to go to the store to buy more sports drinks.

SECTION 8

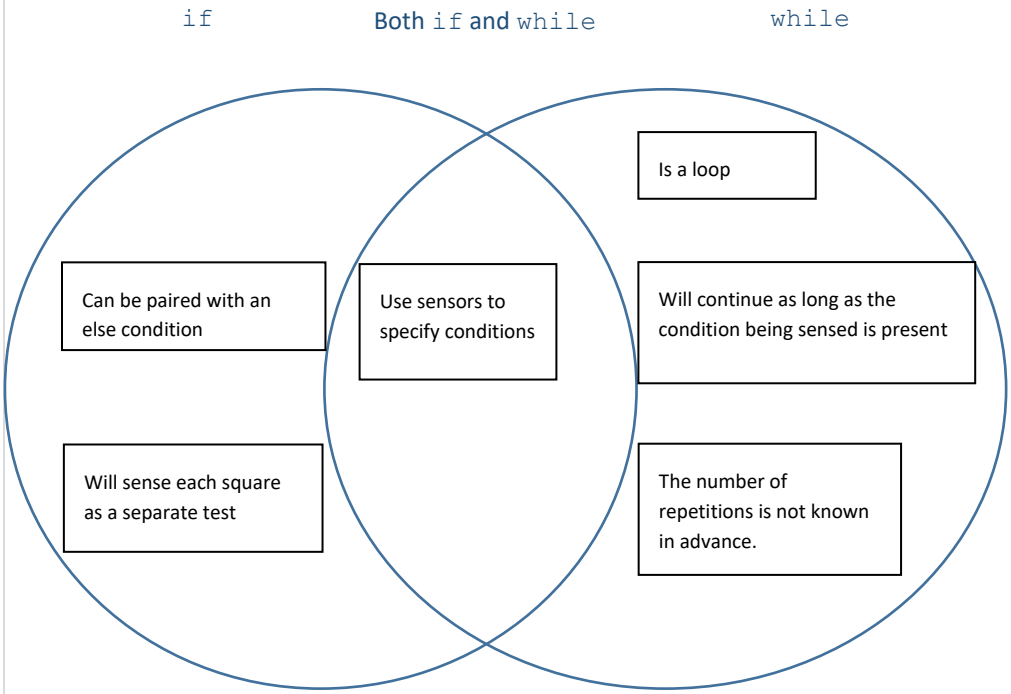
PAGE 2 - 14

What logical operator or operators would you use for the following conditions? Write out an expression using the operators and parentheses if needed.

Mix chemical A and B together but do not mix them with C. The mixture will explode if you do.	**Sample Answers (accept all reasonable answers)** A and B not C (A and B) not C
I can work Wednesday or Thursday next week and Tuesday the following week.	Wednext or Thurnext and Tuesfollow (Wed or Thurs)Next and (Tues)Follow
I will not eat pizza with onions or anchovies.	Not pizza (onions or anchovies)
Make up your own example using logical operators.	Answers will vary.

SECTION 9
WHILE LOOP
PAGE 2 - 17

Place these statements in the correct part of the Venn diagram.



SECTION 9
PAGE 2 - 18

`while not home` is a commonly used `while` loop. Explain what that means.

Sample: Karel will continue his operations until he senses the home sensor. Then he will stop and end the program.

Another common loop is `while wall` or the opposite: `while not wall`. What were these loops used for in this Section? ***Note: This question will be corrected in the next revision. While wall and while not wall are not used in this section. The keyword acid is used instead of wall, so acid could be substituted. While not acid is never used.)***

Sample: In Section 9.5 Karel continued along the wall of acid (while acid) until he no longer sensed the acid. Then he went on through.

Your SUPER INSPECTOR 9000 is checking a fence for damage. Each board must be examined carefully. The robot won't stop until the whole fence has been checked. What kinds of conditions will you use to program the robot? Will you use `if`, `while`, or both?

The fence is a wall, so the robot could use a while wall or while fence type command. If could be used to note which sections are damaged.

SECTION 10

WHILE LOOPS AND CONDITIONS

PAGE 2 - 21

There are certain patterns in mazes that are require a specific set of commands. Review 10.1, 10.2, and 10.4 and write down the commands needed to navigate spirals, steps, and the perimeter of a square.

SPIRAL	STEPS (STAIRCASES AND SHELVES)	PERIMETER OF A SQUARE

`while not home`
`  go`
`  if wall`
`  left`
`  if key`
`  get`
 	
This is a spiral because Karel always turns left when he senses a wall. He checks every square.
	`while wall`
	`  if key`
	`  get`
	`  left`
	`  go`
	`  right`
	`  go`
	
	This is a set of steps: Karel zigzags.
	
	If Karel is filling shelves, he will need to turn around - for example: left, go, right, right, go, left, go
		`repeat 4`
		`  while not wall`
		`  go`
		`  if key`
		`  get`
		`  left`
		
		The perimeter of the square or rectangle includes 4 sides, so we use a repeat 4 loop. Within the loop, Karel will keep moving forward until he reaches the wall and turns left.

However, "real world" mazes aren't so regular. Write down the code explained in 10.6, which will work for navigating any maze.

IRREGULAR AND RANDOM MAZES

Code from 10.6
 	
`while not home`
`  if wall`
`  left`
`  if wall`
`  right`
`  right`
`  if key`
`  get`
`  go`
 	
This code will work for any maze because Karel checks for walls in any position. Example: Karel turns left. He senses another wall, he turns around and then goes forward.

SECTION 10

PAGE 2 - 22

Home robots: could they really do our chores? Pick one of these and come up with a plan. Don't try to write the code (it would be very long!), but try to think of tasks that would lend themselves to repeat loops, if conditions, and while loops. ***Answers may vary. Sample:***

Do the laundry	Repeat loop for the washer, dryer, fold, put away pattern. If conditions – example if whites, add bleach. While conditions: while laundry, do not take a shower.
Clean the bedroom	
Shop for groceries	
Pull weeds in the garden	
Recycle	

UNIT 3 SECTIONS 11-15

SECTION 11
USING THE KEYWORD DEF
PAGE 3 - 5

There are key advantages to using defined commands. Find some examples of the following advantages. These may be from the levels in this Section, or your own experience.

The program requires less lines of code, once the definition has been created.

Answers will vary. Example: 11.2 uses only 46 lines of code compared to 11.1 which uses 108 lines, because the defined command star takes care of the pattern Karel uses to collect chips from each star. Students can also use the example of the `waterbox` defined command in 11.5/11.6.

It is easier to fix problems within the defined command, rather than searching through the program and fixing several lines.

Answers will vary. Example: Students may refer to the way they solved 11.5 waterbox, before implementing it in 11.6 as a defined command. Or 11.7: each row and turn direction is a defined command. It is easy to fix errors in one of these components rather than finding and fixing the individual lines of code repeated several times in a program.

Write an example of the def keyword in action. What does it do? Make notes on the syntax and logic.

Code	**Syntax – what should I remember to do when using `def`?**	**What does the defined command do in the program?**
Examples will vary		

`def scoot`
`  right`
`  go`
`  go`
`  left`
	Defined command itself:
	Written before main program.
	First line is not indented. Lines in the body of the defined command are indented similar to repeat loops.
	Calling command in main program: just write the defined command name `scoot` as part of the program. It can be included as a single line command, or part of a loop.
		This command moves Karel from one star to the next.

SECTION 11
PAGE 3 - 6

You are starting to use simple blocks of programming to build complex routines. Explain how this process works in either 11.3/11.4 (using `def star`), or 11.5/11.6 (using `def waterbox`).

Sample: If a program includes repeated sets of commands, those sets of commands can be written as define commands which are used as needed. `star` is used every time Karel comes up to a star-shaped group of chips, which are all the same pattern. `Waterbox` is used every time Karel comes up to a box containing water bottles. The boxes are all the same size.

11.7 uses three defined commands to create repeated actions. Notice how comments are used as headings and explanations for each defined command, and for the main program. Practice writing a comment and a defined command. Remember to start the comment with the # symbol to show that it is just a text string and not part of the program itself.

Answers will vary. Example of one defined command used in program:

```
#Defined command to climb one step
def climb
  left
  go
  right
  go
#Climb 10 steps
repeat 10
  climb
```

SECTION 12

USING DEFINED COMMANDS; ADVANCED MAZE SKILLS

PAGE 3 - 9

In 12.1 to 12.3, you practiced a simple task that was repeated in a larger program. What was the simple task, and how was it used?

In 12.1, `place6` is created to put six bags of popcorn in a row. This command is used in 12.2 – repeated twice for the north and south rows, and once each for the east and west columns. In 12.3, the defined command is modified from `put` to `get` to collect the bags of popcorn.

The defined command `move` is a useful one for irregular mazes. For your own reference, write out the two versions of `move` that Karel uses to follow an irregular path; one for following the wall on his left, and the other for following it on his right. Include indentations so that it is obvious which actions are within the while loop and which are not.

FOLLOW WALL TO THE LEFT	FOLLOW WALL TO THE RIGHT

`def move`
`  left`
`  while`
`  wall`
`  right`
`  go`
	`def move`
	`  right`
	`  while`
	`wall`
	`  left`
	`  go`

SECTION 12

PAGE 3-10

George has developed a design that will be used to print Halloween themed fabric.

Here is the pattern, using `pumpkin` and `eye`.

Draw a pattern that could be based on this design.

Any sketch that makes multiple use of the basic pattern is acceptable. Example:

Write a defined command `design` that would make this pattern.

```
def design
  repeat 4
    put
    go
    left  (or right)
```

Note: Karel must have alternating pumpkins and eyes in his pocket in order for this program to work. The pocket can be loaded by collecting alternating pumpkins and eyes from rows before starting the design.

Write a program that would call a defined command `design` to make this pattern.

To make a pattern like this, a defined command can be created to move Karel over to start the next design, and a third defined command can move him into position for the next row

```
design
move4
design
nextrow
design
move4
design
```

SECTION 13

COMPARING PROGRAMS; SOLVING COMPLEX PROBLEMS

PAGE 3-13

Compare the two programs in 13.1 and 13.2. Write down the number of lines, the number of operations, and the time it takes to run. Add other observations. Decide which one you prefer and explain why using your code checking criteria (reliability, speed, ease of use, limitations).

	13.1	13.2
Number of lines	9 (not including comment lines)	13
Number of operations	503	243
Time to run program	9:16 (this will vary)	3:41
Observation:	13.1 has less lines, but runs twice as many operations and takes 3 x as long as 13.2	
Observation:	13.1 would be the only way to solve a problem with an irregular path, but since we have straight columns, we can use a go command until we hit the end wall, which is much faster. We collect all the pearls in a straight line and return in a straight line.	

I prefer 13.2 because the conditions permit a faster and more efficient program. It is reliable in this column configuration. The program is a little longer than 13.1, but still short and easy to understand and use. (if 13.1, it would be because the program could follow any wall, not just a straight one. It is reliable under a wider range of conditions.)

Testing and optimizing a small component before using it in a larger program is an important concept. Think of all the systems used to build a car. A car is made up of a drive train (mostly engine and transmission), chassis, body, wheels, axles and steering, electrical systems, electronics, and so forth. Pick one of these systems, or a small part of it such as a steering wheel. Use reliability, speed (or efficiency), ease of use, and limitations as criteria. What would you look for in testing a design of this system or part?

SECTION 13

PAGE 3-14

Answers will vary. Look for references to reliability, speed, ease of use, limitations.

SECTION 14

VARIABLES AND FUNCTIONS; INC(), DEC(), AND PRINT

PAGE 3-17

Using the word list, fill the blanks in the following definitions.

Variable: in terms of programming, variable is the **name** and **value** of something that will be recorded in **memory**. The **counting** variable is used in Section 14. The **initial** value of this variable is set: for example: n = 0.

`inc(n)` tells the program to **increase** the value of n. The default increment is **1**.

`dec(n)` tells the program to **decrease** the value of n. The default is **-1**.

`print(n)` tells the program to print the **final** value of n after the program has ended. Text **strings** can be printed out on their own or as part of a command. The text is always enclosed in **quotation** marks.

What were the inc(n) or dec(n) functions used for in the Section 14 levels?

LEVEL	USE OF INC(N) OR DEC(N)
14.1 to 14.3	Increases the number of maps found (if map)
14.4, 14.5	Increases the number of breaks found (if not wall)
14.6, 14.7	Decreases the number of bottles in Karel's pocket (when he puts the bottle on the shelf)

SECTION 14

PAGE 3-18

Write the following statements as print commands. The first one is done for you.

Karel has used n coins.	`print` "Karel has used" `(n)` "coins."
n fence sections are damaged.	`print (n)` "sections of fence are damaged."
The total number of pages is n.	`print` "The total number of pages is" `(n)` "."

All businesses must keep track of items they buy and sell in their inventories. Programs use counting variables to keep track of the increasing and decreasing amounts of each item. It is helpful to write alerts into the program, so that the purchaser knows when to order replacement stock. Write two lines of code that will print out "Order light bulbs, item #10765" if the number of light bulbs is 15.

```
if n = 15
  print "Order light bulbs, item #10765."
```

You are completely out of coral bracelets, item #35-672. Write a print message for your website store program which tells the customer that the item is not available. Write it in cheerful language that makes them want to continue browsing your store.

Example (answers may vary):

```
if Item35672 = 0
print "We are sorry, item #35-672 is sold out.  We invite you to continue
browsing our fine and unique collection.  For more information on availability
of the sold out item, please contact us."
```

SECTION 15

VARIABLES AND FUNCTIONS, INC(), DEC(), RETURN, LOCAL, GLOBAL

PAGE 3-21

Here is some sample code. Underline the local variables, and circle the global variables.

```
def column

# Count pearls:
 c = 0
 while pearl
   right
   go
   left
   inc(c)
 go
 return c

# Main program:
result = column
print "There are", result, "pearls!"
```

Why can't we print the variable c?

We cannot print the variable c because it was created within the defined function `column`. It can't be used outside of that function.

What is another way to write this program so that we can print c?

(This method is NOT recommended)

If we set the value of the variable c in the main program before calling the function column, then it is a global variable and can be used to print c.

SECTION 15

PAGE 3-22

In this Section, you are able to increase or decrease a variable by more than one: in effect, multiplying or dividing for each operation on the variable. Give examples of how you would code the following. **Answers may vary. These are sample answers.**

Total sales of bicycles at $285.00 each (note: "endofthemonth" and "bicyclesold" are not keywords in Karel.)

```
def bikesales
  b=0
  while not [endofthemonth]
    if  [bicyclesold)
      inc(b,285)
  return b
result=bikesales

print("The total value of bike sales is $", result)
```

Students are given pencils from a box of 500. Each student gets 3 pencils.

```
p=500
s=0
while p>2
  dec(p,3)
  inc(s,1)
print (s, "students received three pencils each.
There are", p, "pencils left.")
```

Note: since 3 pencils are being handed out at a time, it makes sense to set the limit as p>2

Find out how many tomato plants are in each row. Then report on the number of tomato plants in all the rows.

Solution is similar to 15.7 (see solution manual)

- Define a command to find the leftmost edge of the row.
- Count the number of plants in a row (n)
- Count the number of rows (r) incrementing r by n each row to get the total number of plants.

UNIT 4 SECTIONS 16-20

SECTION 16

USING GPSX AND GPSY SENSORS, SYMBOLS

== != < >

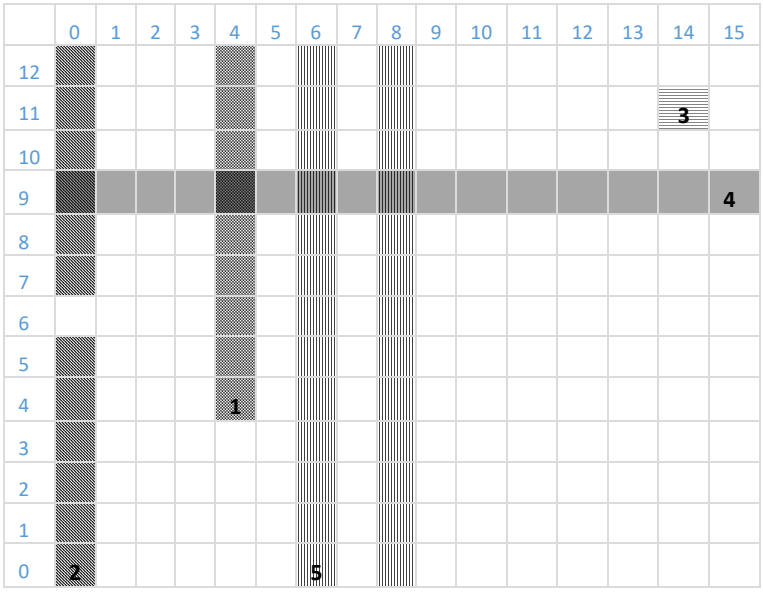
PAGE 4-5

Match the symbols to their meanings:

==	is equal to
<	is less than
=!	is not equal to
>	is greater than

Using different colors or shading, mark and number the location of the following expressions on the grid.

1. `(gpsy > 3)and(gpsx == 4)`
2. `(gpsy != 6)and(gpsx < 1)`
3. `(gpsx == 14)and(gpsy == 11)`
4. `(gpsy == 9)`
5. `(gpsx == 6)or(gpsx == 8)`



SECTION 16

PAGE 4-6

You need to pick up all your clothes and put them in the basket in the southwest corner of your room. You use the Clean-O-Matic robot to take care of this chore. How will you program the robot?

- Answers may vary. Ideas could include:
- Some way of traveling through the room, for example: row by row, column by column
- Conditions: if clothes/get
- Note: since Karel cannot place more than one item in a square, the "basket" could be enough squares in the southwest corner of the maze (say 9 x 9) to accommodate all the items.
- Use a gpsx/gpsy locator to get Karel to the corner.
- Karel could use a defined command similar to `waterbox` (Levels 11.5, 11.6) to place the items in the "basket".

Karel must retrieve three oxygen bottles left on the mountain and report their location.

He must deposit them at the "base camp" located on gpsy == 0, between gpsx == 12 and gpsx == 14.

Write the sections of code that will perform just these two tasks. Think of the best way to use gpsx, gpsy. (Answers may vary)

```
# retrieve oxygen bottles and report
location
if oxygen
  get
  print ("bottle found at", gpsx, gpsy)
```

```
#deposit bottles at base camp
if ((gpsx>11 and gpsx<15)and gpsy ==0)
  put
  go
```

SECTION 17

USING BOOLEAN VALUES TRUE AND FALSE

PAGE 4-9

How are Boolean values used in each level? Fill in the blanks in the following table.

Level	Variable starts as	Condition Test	Outcome (True)	Outcome (False)
17.1	not used on this level	sensor keyword	Karel finds the sensor and prints "sensor:" True	Karel does not find the sensor (he either finds an empty cell or another sensor) and prints "sensor:" False
17.2	`sn=false`	`if snake` `sn = true`	Karel prints that he has found snake.	Karel prints that he has found a spider. (note: all the mazes contain either a snake or a spider.)
17.3	`nug = false`	`if nugget` `nug = true`	Karel prints that he has found a gold nugget.	`Else`, Karel prints that he has found a gem.
17.4	`cr=true` `complete=true`	`cr=cr and bottle` (sensor words by themselves work as true/false Booleans. Cr is changed to (cr and bottle), so if a bottle is present, cr remains true. If there is no bottle cr changes to false.	Karel finds a bottle in every square. `if success` (is met), prints "The row was complete!"	`Else`, Karel prints "One or more bottles were missing!"
17.5	`b=true`	`b= b and bottle` (as above)	`if success` Print "There was a bottle in every corner."	`else` (`success` is not met); prints "One or more bottles is missing".

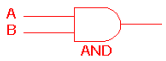
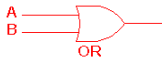
17.6	`fo = false`
	`found = false`
		`while not home`
		`fo=(fo or map)`
			`success = found`
			`if success`
			Karel prints "I found a map!"
				`Else`
				Karel prints "There was no map!"

17.7	`wa = false`	`wa = (wa or nugget)`	`success = walk`
			`if success`
			Karel prints "I found a nugget on the way!"
				`Else`
				Karel prints "I did not find a nugget!"

SECTION 17

PAGE 4-10

And/Or logic gate: electronic circuits are based on electrical signals that are either on or off. We can think of on as True and off as False. We can use a "truth table" to predict whether or not the output will be on or off. Complete the tables for the AND gate, and for the OR gate.

Type of Logic Gate	Input	Output
AND GATE 	A = True (On) B = True (On)	True
	A = True (On) B = False (Off)	False
	A = False (Off) B = True (On)	False
	A = False (Off) B = False (Off)	False
OR GATE 	A = True (On) B = True (On)	True
	A = True (On) B = False (Off)	True
	A = False (Off) B = True (On)	True
	A = False (Off) B = False (Off)	False

SECTION 18

USING THE FUNCTION RANDINT()

PAGE 4-13

Chance situations: How would you write a function for the following?

Conditions	Code (answers may vary in details such as names and print commands)

Rolling "snake eyes" 	`rn = 0`
	`die1 = 0`
	`die2 = 0`
	`while (die1 != 1) or (die2 != 1)`
	`  die1 = randint(6)`
	`  die2 = randint(6)`
	`  inc(rn)`
	`  print("Roll", rn, ": die1 =", die1, ": die2 =", die2)`

Rolling a 7 on a dodecahedral die 	`rn = 0`
	`die1 = 0`
	`while (die1 != 7)`
	`  die1 = randint(12)`
	`  inc(rn)`
	`  print("Roll", rn, ": die1 =", die1)`

Explain the procedure for finding the maximum height of the columns in 18.6. What are the limitations? What minor change is needed to find the minimum?

Answers may vary. Sample:

We set the initial value of the maximum (m) to 0. Once Karel runs the function column and gets a height, the height (as "column") is compared to the maximum. If it is greater than the maximum "if c>m", then m will be made equal to this new value "m=c". If not, then it will stay the same. Only a higher value will change it. At the end of the program, we should have the highest value of m, or maximum.

Limitations: example (students may find other limitations): the sensors used to detect the column (in this case light bulbs) must be consistent, present in every square of the column. The moment they run out, the program assumes that it has reached the top.

To change this to the minimum, we start with a maximum value, in this case m=7, and the column heights are compared to see if they are less. If they are, m is decreased.

SECTION 18

PAGE 4-14

It's your worst nightmare: you start a test, and can't remember anything! You will have "go random" and hope for the best. This is a multiple choice test, with a, b, c, or d as answers. Write those choices on scraps of paper to be drawn at random for each answer….. Write which answer you drew in the spaces below. The answer key is at the end of this section. Check your answers. Did guessing (random drawing) pass the test? Compare your results with those of another student.

Question	Random Answer	Actual Answer	Correct? Y/N	Question	Random Answer	Actual Answer	Correct? Y/N
1.				6.			
2.	Students gather and record data and sum at the bottom			7.			
3.				8.			
4.				9.			
5.				10.			
Score (Correct/Total)				Did you pass?			
Is this a good application for randomness? Explain.							
Answers will vary. Generally speaking, this is not a good application. Most runs will fail the test. Students should refer to their data to justify their response.							

SECTION 19 EMPTY AND NON-EMPTY LISTS PAGE 4-17

In the table below, explain the meaning of each line of code.

Code	Meaning (answers may vary)
`A = [1, 3, 5, 7, 9]`	The list A equals a list of five odd numbers starting with 1
`for x in L` `  print "Map found at:", x`	For variable x in list L, print "Map found at" the value of x.
`Y = [ ]`	The list Y equals (is set as, defined as) an empty list
`C.append (x)`	Append variable x to list C.
`len(m)`	The length of list m (the total number of items in list m)

Build list commands for the following. We will call the list P.

Start with an **empty list.**	`P = [ ]`

In order, **add** three erasers, one pencil sharpener, two pencils, one pen	`P.append ("eraser")`
	`P.append ("eraser")`
	`P.append ("eraser")`
	`P.append ("pencil sharpener")`
	`P.append ("pencil")`
	`P.append ("pencil")`
	`P.append ("pen")`

Parse the list using a For loop to print out each item.	`for x in P` `  print x`
Find the length of the list.	`n = len(P)` `print(n)` (optional)

SECTION 19 PAGE 4-18

You need a bouquet of one of each variety of flowers. Create a list that tells your flower-picking robot how many steps to take to get to the next type of flower.

F = [1, 3, 2, 4]

Note: this is the default maze in Creative Suite. Students just have to add three lines to the code.

Write a program that will have Karel record the gpsy location of each key and print a list.

```
K=[]
while not home
  go
  if key
    get
    K.append(gpsy)
  if wall
    right
    if wall
      repeat 2
        left
print(K)
```

SECTION 20

WORKING WITH LISTS

PAGE 4-21

In the table below, explain the meaning of each line of code.

Code	Meaning
`b = A.pop( )`	Remove the last item on list A and assign it (return it) to variable b
`if orchid` `  o.append([gpsx, gpsy])`	If Karel finds an orchid, append the gpsx and gpsy coordinates to list o
`repeat 4` `  la = X.pop()`	Remove the last item 4 times from X and return it to la.
`n = L.pop(0)`	Remove the first item on list L and return it to n.
`R.append(R2.pop(0))`	Add the first item from list R2 to list R

*Build commands for Karel. We are making a map of coin locations, using a list **m**. Note: students can test this code on the default program in Creative Suite by substituting key for coin.*

Start with an empty list.	`m = [ ]`

While not home, move forward. If there is a coin, add True to the list. Otherwise add False.	`while not home`
	`  go`
	`  if coin`
	`  m.append(True)`
	`  else`
	`  m.append(False)`

If there is a wall, turn left.
If there is a wall, turn right twice.
	`  if wall`
	`  left`
	`  if wall`
	`  right`
	`  right`

Print the list.	`print(m)`

SECTION 20

PAGE 4-21

Here is a list of numbers. L=[2, 9, 6, 1, 0, 5, 5, 8, 10, 4, 3, 6, 8, 7, 20, 1]

Write a `for` loop that will test the values in L, and if they are less than 6, append them to an empty list K. Print out the results.	`L=[2,9,6,1,0,5,5,8,10,4,3,6,8,7,20,1]`
	`K=[]`
	`for x in L`
	`  if x<6`
	`  K.append(x)`
	`  print("Added",x,"to K")`
	`print("List K now contains",K)`

You manage a fast food restaurant. You stock your hamburger buns once a week. What would you need to write into a program that monitored and reported the hamburger bun inventory? (You do not need to write the code – just make a plan)

Answers will vary. Sample:

You will need some information on how many buns are used each week, and maybe more detailed information, such as how many per day, or if there are peak days to be aware of. You also need to know how long it takes to get new buns delivered. With that information, you can set a variable equal to your inventory, take out a bun each time it is used, then set alerts for reordering based on the remaining number of buns. The trigger number will depend on what you have learned from your data. This could become more sophisticated based on delivery schedules, peak days, etc.

UNIT 5 SECTIONS 21-25

SECTION 21 PROBABILITY PAGE 5-5

Complete the truth tables to show why `rand` is a 50/50 probability, and `rand and rand` is 25/75.

Function	Possible outcomes	Explanation for Probability (sample answers)
`rand`	`true`	The probability is 50/50 because 1 out of 2 possible outcomes is `true`, and 1 out of 2 outcomes is `false`.
	`false`	
`rand and rand`	`true + true = true`	The probability is 25/75 because 1 out of 4 possible outcomes is `true`, and 3 out of 4 outcomes are `false`. If `false` is an outcome of part of an `and` function, the outcome will be `false`. (note: see questions Section 17, page 4-10 for an illustration of and circuits)
	`true + false = false`	
	`false + true = false`	
	`false + false = true`	

Think of reasons to use probability in a game by answering these questions.

Karel uses `rand` to control his movements in Levels 21.5 to 21.7. What types of environment are best suited to random movement? Answers may vary, sample answer: Environments suited to random movement would include open spaces, especially with irregular boundaries and/or irregularly placed obstacles. These cannot be solved by either a wall-following algorithm (which would miss the open space), or one that uses columns or rows (which would not work in a simple form because of the irregular boundaries).

When would you use a 50/50 probability, and when would you bias the decision by using 25/75 (Look back to Levels 21.3, 21.4)? Answers may vary, sample answer: conditions dictate which way to go. 50/50: Simple yes/no decisions or outcomes that only have 2 choices, types of materials present (e.g. even distribution of two different objects).

25/75: The "correct" choice or outcome is one out of a range of possibilities; uneven distributions of objects; favored sorting of objects into one area over another.

PAGE 5-6

Classic probability exercises help to visualize how `rand` works. Put colored tiles in a bag (2 tiles of different colors for the 50/50, and 1 tile of one color, 3 tiles of another for the 25/75 draw) and draw them out. Tally the results for 10 draws, 25 draws and 100 draws and record the total counts in the chart. How close to 50/50 and 25/75 are you at each point?

Probability		10 trials	25 trials	100 trials	Comments
50/50	Color 1	Students record results in these columns			Summarize observations. Comment on how close each result matches the probability, and if the match improves with the number of trials (normally, more trials = a closer match)
	Color 2				
25/75	Color 1	Students record results in these columns			
	Color 2				

Karel is exploring a dark cave. Write a program that will have Karel move left or right until he finds a flashlight. Don't forget to check for walls.

This program will be similar to 21.5-21.7

```
while empty
  go
  if wall
    left
    if wall
      right
      right
  if flashlight
    get
```

SECTION 22 RECURSION

PAGE 5-9

Recursion can be a difficult concept to grasp. Explain the role of each component in the following program. Here are some vocabulary terms to help you:

custom command ***stopping condition*** ***recursive call*** ***main program***

Line	Purpose (what is happening on this line?)
`def walk`	Defines the **custom command** `walk`
`  if not home`	**Stopping condition** (`walk` will continue to call itself until Karel is home)
`    if shield`	Body condition
`      get`	Body command as a result of the condition
`    go`	Body command
`    walk`	**Recursive call** (starts the body of `walk` again)
`  return`	Ends walk and returns to main program.
`walk`	**Main program**

Why is `if not home` the stopping condition? How does this differ from a while loop that uses `while not home`?

Sample answer: `If not home` is used because each square must be tested independently before starting the recursion. `If not home` only tests that particular square, whereas a `while` loop continues the loop's body commands until Karel reaches the home square.

PAGE 5-10

Write a recursive function for one of the following situations, or make up your own.

- *Do pushups until your heartrate reaches 140 beats per minute. Then rest.*
- *Work in the garden until the temperature is 80 degrees F. Then go inside.*
- *Practice long division until you can divide a five digit number by a two digit number correctly. Then play a video game.*
- *Eat hot dogs at a contest until you are full. Then, please stop.*

Your turn:

Answers may vary. These are concept maps, rather than actual code. Answers should include a defined command, conditions, body commands, recursive call, return, and main program elements. Sample answer:

```
def routine
  if heartrate < 140
    pushup
    routine
  return
routine
rest
```

SECTION 23
RECURSION II
PAGE 5-13

Describe the recursion in each level. What stopped the recursion?

23.1	`robber` moves Karel up to the next level each time. `if wall` is the stopping `condition`. The recursion will stop when Karel does not detect a wall (in other words, he reaches the edge of the maze)
23.2	`row` moves Karel along a row (in this example, it is actually a column). `If candy` is the stopping condition. The recursion will stop when there is no candy in the square.
23.3 (bounty)	`bounty` is a recursion that contains the recursion `row`. It moves Karel along a row, collecting candy until no candy is detected, then the `row` recursion ends and Karel turns left. `row` will start again on the next row. The stopping condition for bounty is `if not home`. The recursion will end when Karel reaches the home square.
23.4	`sum` adds a set of numbers together that decrease by 1 each time. The stopping condition is `if n>0`, so the recursion will stop when the current value of `n` is zero.
23.5	`addlist` adds all the numbers in a list. Each time a number is added, it is removed from the list. `if len(T)>0` is the stopping condition. The recursion will stop when the list is empty.
23.6	`edge` moves Karel forward, getting a pie, until he does not detect a pie. The stopping condition is `if pie`.
23.7 (eat)	`eat` is a recursion that contains the recursion `edge`. It moves Karel along a edge, eating pie until no pie is detected, then the `edge` recursion ends and Karel turns around, goes back one step, and turns left. `edge` will start again on the next row. The stopping condition for bounty is `if pie`. The recursion will end when Karel no longer detects pies, even after he has turned around and repositioned himself.

In the final level (23.7), Karel solves an array by moving in a spiral pattern. Compare this way of solving arrays to the one in Level 15.7 Answers may vary. Sample answers:

15.7	**23.7**
Karel detects the left corner of the bottom row, then counts the items along that row. When he reaches the right corner, he turns and counts the number of rows, then multiplies to get a total. This program can be run with any sized array that is completely filled (no holes). Karel can be in any position along the row to start. This program is being run to count the number of units (crates).	Karel goes along a row. When he runs out of pie, he goes back to his last position and turns. Then he starts eating pies again. The program ends when there are no more pies. This program can also be run with an array of any size. It could also be run with a squared spiral, which 15.7 cannot do. It assumes Karel is starting in a fixed position. It does not count the number of pies, but this could be written into the program, incrementing a variable every time he eats a pie, and returning the total at the end.

PAGE 5-14

Write a program to solve the following problem, using recursion.

Oh no! The General has sent Sophia to the moon. Karel hurries to the launch pad, climbs into the rocket and gets ready to blast off. All he needs is the countdown. Write a recursion that will count down from 10 to 0 and print "Blast Off!" at the end.

```
def countdown
  # Stopping condition:
  if n > 0
    print (n, "…")
    dec(n)
    # Recursive call:
    countdown
  return

# Main program:
# Set n to 10:
n = 10
countdown
# Print Blastoff
print ("Blast Off!")
```

What skills did you practice on each level? Use the table to review each level.

SECTION 24

ADVANCED SKILLS

PAGE 5-17

Here are some terms that you can use.

gpsx, gpsy coordinates	Append and pop lists
while conditional loops	if/else conditions
Nested repeat loops	Complex tasks or patterns reduced to simpler ones
Information from one part of a puzzle used to solve another part.	Defined functions with counting variables

Students may find more than one skill in a level.

Skill	**Found in Levels:**
gpsx, gpsy coordinates	24.2, 24.5
while conditional loops	24.2 – 24.7 (note: 24.7 can be solved without while)
Nested repeat loops	24.1, 24.7 (note: 24.6 uses repeat loops that are not nested)
Information from one part of the puzzle used to solve another part	As defined commands: 24.2, 24.4, 24.5, 24.6. (As used in a list: 24.5, 24.7)
Append and pop lists	24.5 (append only), 24.7 (append and pop)
If/else conditions	24.5, (24.2, 24.3, 24.4, 24.6, and 24.7 use if but do not use else)
Complex tasks or patterns reduced to simpler ones	All levels
Defined functions with counting variables	24.2

PAGE 5-18

Traditional crafts and artwork contain many patterns. Suggest ways to write programs to create these examples.

Answers will vary. Since these patterns are regular, repeat 4 or for i in range (4) loops would be useful. Students' experience is mostly with repeat loops. The square border, outside dots, circle, inside dots, curved edge pattern, inside star and center dot must all be included in the program. The background could be assigned a black color, and the objects colored white, with the exception of the black inside star (unless this is just the edge of a shape)

Answers will vary. This time there is a pattern repeated 3 times. The pattern is a curved spiral. This could be created with a list that is decreased each time until the variable reaches a limiting value, or with steps repeated a specific number of times. The outer circle is a separate part of the program.

SECTION 25

CHALLENGES

PAGE 5-21

PAGE 5-22

Students use this section to solve problems and revisit puzzles. Grading is at teacher's discretion, and may include vocabulary usage, evidence of logical reasoning, details in reflections.

In this final section, you can practice coding complex tasks, including some classic logic problems.

Some of the solutions are tricky, so here is an extra note-taker page to help you work out your solutions.

If you have stuck with Karel all the way to the end of Unit 5 – Congratulations! You now have some great programming skills.

Pick a level at random from each of the previous 4 units. Erase the code, time yourself, and see how fast you can complete that level. Did you get it right the first time? Is your code elegant (simple, effective)? Record your results.

Answers will vary	