



KAREL UNIT 5
STUDENT JOURNAL

REVISED APRIL 19, 2017

NAME	
DATE STARTED	DATE COMPLETED
SCHOOL, CLASS, PERIOD	

TABLE OF CONTENTS:

WELCOME TO YOUR JOURNAL	4
SECTION 21: PROBABILITY	5
SECTION 22: RECURSION	9
SECTION 23: RECURSION II	13
SECTION 24: ADVANCED SKILLS	17
SECTION 25: CHALLENGES	21
REVIEW YOUR PROGRESS	25
LIST OF BASIC COMMANDS AND KEYWORDS	26
LIST OF KEY VOCABULARY	28
FILE LOG: GAMES I HAVE CREATED	33
DESIGN TEMPLATE	34
NOTES	35

General Website: <https://nclab.com/>

Karel Gallery: <https://nclab.com/karel-gallery/>

Desktop (needs login information): <https://desktop.nclab.com/>

Keep your name and password in a safe place.

WELCOME TO YOUR JOURNAL

Welcome to Karel 5, the last unit in the Karel course. You will be learning and applying advanced logical thinking and programming skills. You are almost at the finish line – it is worth crossing!

As a reminder, this journal empowers you to:

1. Remember better.
2. Create your own reference book.
3. Make connections.
4. Keep a record of your work.
5. Use your notes to collaborate with others.

This journal is set up the same way as the previous Karel Journals.

The journal is divided into sections that match those in the on-line course. Each section has:

1. One or two **review pages** with questions or activities to help you remember what you have learned. There will always be an open box for your own notes.
2. A **bulletin board** where you can post real life examples: paste in pictures, sticky notes, and scribbles. There are ideas and suggestions at the top of each bulletin board.
3. A **planning page** for your end-of-section project.

The back of the journal contains a **glossary with all vocabulary from Karel course**, a **record page** for your files, and a **design template** that can be copied to work on games.

As always, remember to slow down, journal, talk with people, and sketch ideas. We hope you will develop a deeper understanding of what coding is all about, and discover the thrill of having a computer or machine carry out a program that you have written.

Happy coding!

SECTION 21: PROBABILITY

In this section, you learn how to use the function `rand` to create True or False with 50-50 probability. You use the function `rand` in conditions and while loops, and in maze algorithms. You know that 50-50 probability means that the two events are equally probable, and that `rand` and `rand` yields 25-75 probability, which means that the former event is three times less probable than the latter.

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
<code>rand</code>		The probability is 50/50 because
<code>rand</code> and <code>rand</code>		The probability is 25/75 because

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?

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)?

SECTION 21 NOTES

Use this space to write your own notes, questions, and problems.

QUESTIONS

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				
	Color 2				
25/75	Color 1				
	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.

SECTION 21 BULLETIN BOARD

This is a page to post ideas, pictures, sticky notes, drawings

The `rand` command is used differently depending on programming language and application. In some cases, it generates a random number (like Karel's `randint()`); in others it is used to control probability, as it does here. Research how `rand` is used.

SECTION 21 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use rand to control some choice that Karel needs to make.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:	Maze Sketch (12 rows x 15 columns)	
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		

SECTION 22: RECURSION

In this section, you learn how to use recursion, which is a command or function that calls itself. You know that recursion is suitable for tasks that can easily be reduced in size, that the recursive call must be placed in a stopping condition, and that failure to use a stopping condition easily turns recursion into an infinite loop.

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?)
<code>def walk</code>	
<code>if not home</code>	
<code>if shield</code>	
<code>get</code>	
<code>go</code>	
<code>walk</code>	
<code>return</code>	
<code>walk</code>	

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

SECTION 22 NOTES

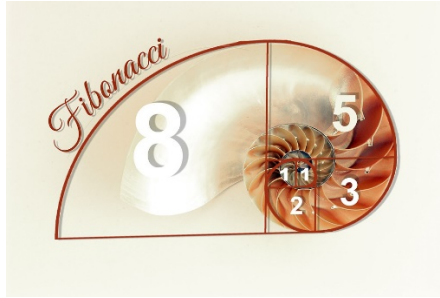
Use this space to write your own notes, questions, and problems.

QUESTIONS

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:

SECTION 22 BULLETIN BOARD

<p><i>This is a page to post ideas, pictures, sticky notes, drawings</i></p>	<p><i>There are many examples in mathematics of recursion – a process that keeps calling itself to infinity – unless we stop it of course. Fibonacci numbers are a good example. This series was developed in 1202 by an Italian mathematician of the same name, who observed how rabbits reproduced. The Fibonacci numbers are defined by:</i></p> $F_n = F_{n-1} + F_{n-2}, \text{ with } F_0 = 0 \text{ and } F_1 = 1$ <p><i>They are sometimes represented as a triangle, or geometrically as a spiral pattern. Research other recursive patterns.</i></p>
	

SECTION 22 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use at least one defined recursive function. Print the results in the main program.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:	Maze Sketch (12 rows x 15 columns)	
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		

SECTION 23: RECURSION II

In this section, you review and practice previous sections: how to use stopping conditions in recursion, how to make the recursive call from inside a stopping condition, how to split complex tasks into simpler ones, how to use inequalities, how to get the length of a list, how to increase and decrease values, and how to pop items from lists.

The key to recursions is finding the stopping condition that ends the pattern being detected by the recursion.

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

23.1	
23.2	
23.3 (bounty)	
23.4	
23.5	
23.6	
23.7 (eat)	

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

15.7	23.7

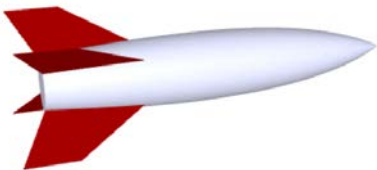
SECTION 23 NOTES

Use this space to write your own notes, questions, and problems.

QUESTIONS

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.



SECTION 23 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Recursion vs. loops: when should you use recursion?</i>	<i>Research recursion (especially Python, since you know some of the code) to find out what people have to say about its strengths and limitations.</i>

SECTION 23 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Build a program that uses one of the specialized recursions learned in Section 23.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:	Maze Sketch (12 rows x 15 columns)	
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		

SECTION 24: ADVANCED SKILLS

In this section, you practice all your skills from previous sections in more complex tasks.

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

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

Level	Skills
24.1	
24.2	
24.3	
24.4	
24.5	
24.6	
24.7	

SECTION 24 NOTES

Use this space to write your own notes, questions, and problems.

QUESTIONS

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



SECTION 24 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>You have learned how to code in a simplified version of Python. Python itself is only one of many programming languages used today.</i>	<i>One classic way that programmers use to compare the syntax of different languages is to write a program that will print "Hello World!" Search "Hello World!" on the Internet to see for yourself.</i>

SECTION 24 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use at least two advanced skills that you have learned in Karel 5.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:		Maze Sketch (12 rows x 15 columns)
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		

SECTION 25 NOTES

Use this space to write your own notes, questions, and problems.

QUESTIONS

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.

SECTION 25 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>The Eight Queens is an example of a classic logic puzzle.</i>	<i>Search the Internet for logic and math puzzles. Can you find the code to solve them?</i>

SECTION 25 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **This is your final game for the course. Research some classic logic puzzles or games and see if you can reproduce them with Karel.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:	Maze Sketch (12 rows x 15 columns)	
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		

REVIEW YOUR PROGRESS

This is the final Section of Karel Unit 5. Reflect on what you have learned so far.

Rate yourself C, B, or A:

- C if you could use this skill any time and could coach someone else;
- B if you have a good understanding but need more practice, and
- A if you feel that you are unsure of yourself and need teaching or coaching.

SKILL OR CONCEPT	C	B	A
Use probability (rand, rand and rand) to solve a game			
Understand the pros and cons of recursion			
Write recursive loops with stopping conditions			
Use advanced recursive techniques (nested recursions, etc.)			
Use advanced list techniques.			
Break large programs into components; test components			
Plan and solve classic logic problems			

Now, set some learning goals based on your self-evaluation. Don't worry if you aren't an expert yet!

RETAKE CERTAIN LEVELS FIND A COACH REVIEW AND DISCUSS NOTES PRACTICE	PRACTICE REVIEW AND DISCUSS NOTES CREATE	READY FOR THE NEXT COURSE CREATE COACH

LIST OF BASIC COMMANDS AND KEYWORDS FROM KAREL UNITS 1-5

Command words: `go`, `left`, `right`, `get`, `put`

Directional commands (`go`, `left`, `right`) are always from the robot's point of view.

`go` advances the robot one step.

`left` turns the robot to its left.

`right` turns the robot to its right.

Retrieving and placing objects (`get`, `put`)

`get` picks up an object

`put` places an object

`randint(n)` randomly selects a number between 1 and n.

Loops

`repeat x`, where x = the number of times the command is to be repeated.

`while x`, where x = a defined condition

`for x in L`, where x is an item in a list L

Conditions

`if x`, where x = a defined condition (this may be a single sensor word or a more complex set of conditions using sensor words and operators)

`else` may follow an `if` condition to provide the alternative course of action

Keywords that are Logical Operators

`not` the condition is that the sensor is `not` present

`and` the condition needs all of the sensors joined by `and` to be present

`or` the condition needs one of the sensors joined by `or` to be present

`true`, `false` the condition exists, or doesn't exist

Important Sensor Words (Karel senses objects and containers when he is in the same square.)

`empty` Karel's pocket is empty

`north` Karel is facing `north`, or the top of the maze grid.

`wall` any obstacle is a type of `wall`. Karel senses it in the square in front of him.

`home` the `home` square. Karel sense it when he is in that square.

`gpsx`, `gpsy` sense Karel's location on the grid in the horizontal and vertical directions.

Defined commands

`def` `def` begins a defined command, which is a set of commands that will be called in the main program.

Working with functions and variables

`inc(n)` tells the program to increase the value of `n`. The default increment is 1. To increase by a different amount, write the function as `inc(n, x)`, where `x` is either a value or a variable that will increase `n`.

Example:

`inc(n, 2)` increases the variable `n` by 2 each time

`inc(total, r)` increases the variable `total` by the variable `r` each time.

`dec(n)` tells the program to decrease the value of `n`. The default is -1. To decrease by a different amount, use the same rules as for increasing.

`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.

Example:

`Print "Placed one bottle."` will print "Placed one bottle."

`Print (n) "bottles remain."` will print "36 bottles remain." (if `n=36`)

`return n` ends the function, returning a final value for the variable.

`rand`: a function that creates True or False with 50-50 probability. Calling a `rand` function is like tossing a coin.

`rand and rand`: the combined function creates a 25/75 probability

Python Lists

`L=[]` Empty list

`L=[1, 3, 2, 8]`

Non-empty list

`L.append(x)` Add an item `x` to a list `L`

`L.append([gpsx, gpsy])` Add a pair of items to a list

`pop`: removes an item from a list and assigns it to a variable.

`la = L.pop()` removes the last item and assigns it to variable `la`


`fi = L.pop(0)` removes the first item and assigns it to variable `fi`


LIST OF KEY VOCABULARY WORDS FROM KAREL UNITS 1-5 (IN ORDER OF APPEARANCE)

Command words: `go`, `left`, `right`, `get`, `put`. These words tell Karel what to do.

Home is the destination square, marked by red diagonal stripes which change to green when Karel approaches the square. The word `home` is also used in conjunction with commands.

Max may refer to maximum number of steps, operations, or programming lines.

Steps are the number of squares that Karel moves. The shoe icon  counts the number of steps.

Operations are anything that Karel does: move, turn, pick up or put down objects. The computer icon  counts the number of operations.

Objects are items placed in the maze. (The word “object” can have other connotations in programming that are not used here).

repeat is written on its own line as `repeat x`, where `x` = the number of times the command is to be repeated.

Body: the body contains the commands to be repeated. The commands are written on the lines following the `repeat` command, indented two spaces.

Loop: A set of commands repeated a given number of times.

Nested loop: A loop that is within another loop.

This is a good time to introduce some of the terms used in programming. Refer to the online textbook under Section 5 Programming for details.

Algorithm: a series of logical steps that leads to the solution of a task. Students may be familiar with algorithms used in operations such as subtraction and long division.

Logical error: a mistake in an algorithm. Planning helps reduce the number of errors.

Computer Program: An algorithm written using a programming language.

Syntax: the way a command line is written.

Syntax error: a mistake in spelling, operators, indentations, spaces

Sensor words: items from the Karel library, which can include collectible items (such as `orchid`), containers (such as `basket`), and obstacles (such as `wall`, `plant`). A word that is both in the library and correctly spelled will be blue-colored. Collectible and container items are sensed in the square that Karel occupies. Obstacles are sensed in the square in front of Karel.

if is written on its own line as `if x`, where `x` = a defined condition. In these lessons, predefined objects from the library are used as sensor words for the condition.

The body contains the commands to be followed if the `if` condition is met. The commands are written on the lines following the `if` condition, indented two spaces.

Condition (Section 8 in the textbook): tells the program what to look for and how to act. Conditions make decisions while the program is running and handle unexpected situations. The program may need to collect all the coins it finds, but may not know where the coins will be located. The `if` condition says: "Is there a coin? If there is a coin, get it." Conditions work like a switch. Note: because `if` conditions test each instance separately, they are NOT loops, even though they are written with a similar format.

Satisfy: in programming, satisfy means to meet the condition - the condition exists.

Aisle: a row or column with objects on either side

Sensor: the presence of something, such as a coin, used to create a condition.

`north`: "`if not north`" can be used to detect if Karel is facing north (the top of the maze), and can be used to reorient Karel to any direction, once he is facing north.

Key words `or`, `and`, `not`:

`or`, `and`, `not` are logical operators for the condition. In order to execute the command, `or` means that one (or a set of conditions within parentheses) of two or more conditions must be met, `and` means both or all of the conditions must be met, `not` means that condition must not be met.

`empty`: tells whether or not the robot has an object in its pocket. This creates a condition, either `if empty`, or `if not empty`

`while`: A `while` loop is a repeated set of commands that will continue as long as the condition being sensed is present. The number of repetitions is not known in advance. The `while` loop continues until the condition is no longer sensed. `while` loops use the same sensors as `if` conditions. A `while` loop is different because it continues until the condition is no longer sensed, whereas the `if` condition senses each square as a separate test.

Infinite loop: If a loop never senses when to end (the stopping condition), it can continue infinitely. Fortunately, most programs will time out if this happens. In Karel, programs can always be stopped manually if this happens.

Defined commands: `def` begins a defined command, which is a set of commands that will be called in the main program. Defined commands can be used as many times as needed in the

main program. This can reduce the number of lines needed, and also makes editing easier. If there is an error in the defined command, it can be fixed in one place.

Text string: words included in the program that are descriptive and not part of a command. Text strings are enclosed in quotation marks and are separated from command words by a comma.

Comment lines: lines of text strings, always starting with the # sign that describe what is happening in the program. Quotation marks are not needed in this case.

Variable: in terms of programming, variable is the **name** and **value** of something that will be recorded in **memory**. The **counting variable** is used to increase or decrease a value

Function: a defined command or set of commands based on a variable that returns a value. Functions `inc()` and `dec()` are used to increase or decrease a variable by a specified value.

Local variable: a variable created within a command or function. A local variable cannot be used outside of that particular command or function.

Global variable: a variable created in the main program. A global variable cannot be used inside of a command or function.

Return: the `return` command ends the function, returning a final value for the variable.

Sensor: `gpsx`, `gpsy` use the grid coordinates to locate Karel (`gps` is "Global Positioning System"). `gpsx = 0`, `gpsy = 0` is the southeast corner square of the maze.

`gpsx` indicates the point along the horizontal x axis, measured in grid squares starting on the west (left) side.

`gpsy` indicates the point along the vertical y axis, measured in grid squares starting on the south (bottom) side.

`==` means "is equal to". For example, "`gpsx == 8`" means "The x coordinate position equals 8."

`!=` is a symbol that means "is not equal to". For example, "`gpsx != 7`" means "The x coordinate position is not equal to 7." This is useful when you want to carry out a task on every square except the ones flagged with `!=`. Make sure the two symbols are together with no spaces in between.

`<` and `>` serve the same function as in math. `gpsx < 4` would mean "All `gpsx` locations less than 4." `gpsy > 6` would mean "All `gpsy` locations greater than 6."

Expressions can be combined with all these symbols. For example: `(gpsx > 9) and (gpsy < 5)`

Boolean operator: a logical operator, for example: True or False.

True indicates that a condition is true.

False indicates that a condition is false (does not exist, for example).

Random: a random value is selected without regard to pattern, order, or reason. Each value within the set has an equal chance of being selected. A coin has an equal chance of landing heads or tails. A die has an equal chance of landing with 1, 2, 3, 4, 5 or 6 face up.

Randint: a command that selects a random integer. The command is written `randint(n)`, where `n` is an integer between 1 and `n`.

Maximum: the greatest value out of a set of values. The maximum is determined by a function that compares values.

Minimum: the least value out of a set of results. The minimum is also determined by a function.

List: A list is a set of items, enclosed in square brackets and separated by commas. For example:
`L = [2,2,8,3,4]`

`pop`: removes an item from a list and assigns it to a variable. Either the last item or the first item is removed. For example

```
la = L.pop()    removes the last item and assigns it to variable la
```

```
fi = L.pop(0)   removes the first item and assigns it to variable fi
```

Empty List: A list that does not contain any items, shown by empty square brackets. For example: `L = []`

Non-empty List: A list that contains items. For example: `L = [1,6,8,3]`

Append: Add items to a list. For example: `L.append(x)` , `L.append([gpsx, gpsy])`. Notice that two or more items must be enclosed in one set of parentheses.

Parse: Examine the items in a list. The items can be printed out as a line-by-line log of the list, using a For loop.

For loop: A for loop is able to iterate (repeat a function) for items in a list. It is indented the same way as other loops. For example, a for loop can print out a log of these items:

```
for x in L
    print "current list item:", x
```

resulting in

```
Current list item = 1
Current list item = 3
Current list item = 4
Current list item = 6
Current list item = 7
Current list item = 8
```

Length of a list: `len`

`pop`: removes an item from a list and assigns it to a variable. Either the last item or the first item is removed. For example

`la = L.pop()` removes the last item and assigns it to variable `la`

`fi = L.pop(0)` removes the first item and assigns it to variable `fi`

`rand`: a function that creates True or False with 50-50 probability. Calling a `rand` function is like tossing a coin.

`rand and rand`: the combined function creates a 25/75 probability

Recursion: a command or function that calls itself.

The recursion occurs within the body of the loop.

It must have a stopping condition. If not, it can turn into an infinite loop.

Stopping condition: a condition that ends a loop.

Infinite loop: a loop that theoretically could continue operating infinitely. Most programs have a timer that would eventually time out the loop.

FILE LOG: GAMES I HAVE CREATED

FILE NAME AND LOCATION	DATE	DESCRIPTION	NOTES
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			

DESIGN TEMPLATE

Game Title: _____ Date: _____ Author: _____

Story Ideas:**Maze elements:****Programming ideas:**

NOTES