



KAREL JR 4  
**STUDENT JOURNAL**

REVISED APRIL 19, 2017

NAME	
DATE STARTED	DATE COMPLETED
SCHOOL, CLASS, PERIOD	



## TABLE OF CONTENTS:

WELCOME TO YOUR JOURNAL	4
SECTION 16: USING GPSX AND GPSY SENSORS, SYMBOLS == != < >	5
SECTION 17: USING BOOLEAN VALUES TRUE AND FALSE	9
SECTION 18: USING THE FUNCTION RANDINT()	13
SECTION 19: EMPTY AND NON-EMPTY LISTS	17
SECTION 20: WORKING WITH LISTS	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 4. In Karel 3, you learned how to write more complex programs. If you needed a set of commands that could be used more than once in a program, you defined it. You learned a few new tricks on how to improve and evaluate your code. Finally, you became familiar with variables and functions.**

**In Karel 4, you work with a wide range of specific keywords, variables, and functions. You can test for position in the maze, whether a situation is true or false, use random values, and .... you are starting to use Python lists!**

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 Karel Jr 1, 2, and 3.**

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 vocabulary from Karel Jr 1 -4**, 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 16: USING GPSX AND GPSY SENSORS, SYMBOLS == != &lt; &gt;

In this section, you learn how to use the `gpsx` sensor to determine Karel's horizontal position in the maze, and use the `gpsy` sensor to determine Karel's elevation in the maze. You also use the symbols `==`, `!=`, `<` and `>`. You know that `gpsx` is 0 in the left-most column and 14 on the right-most one, `gpsy` is 0 in the bottom row and 11 in the top one. The keyword `and` ensures that conditions are satisfied at the same time, and the keyword `or` makes sure that at least one condition is satisfied. Parentheses should be used for expressions such as `(gpsx == 7)`, `(gpsy < 3)`.

Match the symbols to their meanings:

<code>==</code>	is greater than
<code>&lt;</code>	is not equal to
<code>!=</code>	is equal to
<code>&gt;</code>	is less than

It is important to understand the difference between the Karel `gpsx`, `gpsy` values and an `xy` coordinate plane. In Karel, the **squares** on bottom row are `gpsy == 0`; the squares in the left column are `gpsx == 0`. So the values do not represent a point on the grid lines as in an `xy` coordinate system.

Remember that `gpsx` and `gpsy` can be used to show a range of values, not just one position.

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

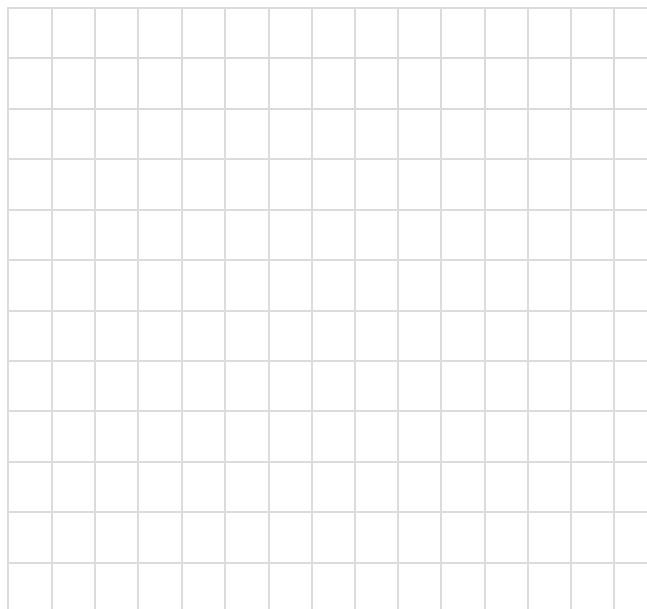
1. `(gpsy > 3)and(gpsx == 4)`

2. `(gpsy != 6)and(gpsx < 1)`

3. `(gpsx == 14)and(gpsy == 11)`

4. `(gpsy == 9)`

5. `(gpsx == 6)or(gpsx == 8)`



**SECTION 16 NOTES**

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

**QUESTIONS**

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?

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 (don't worry about how to get him over the mountain).

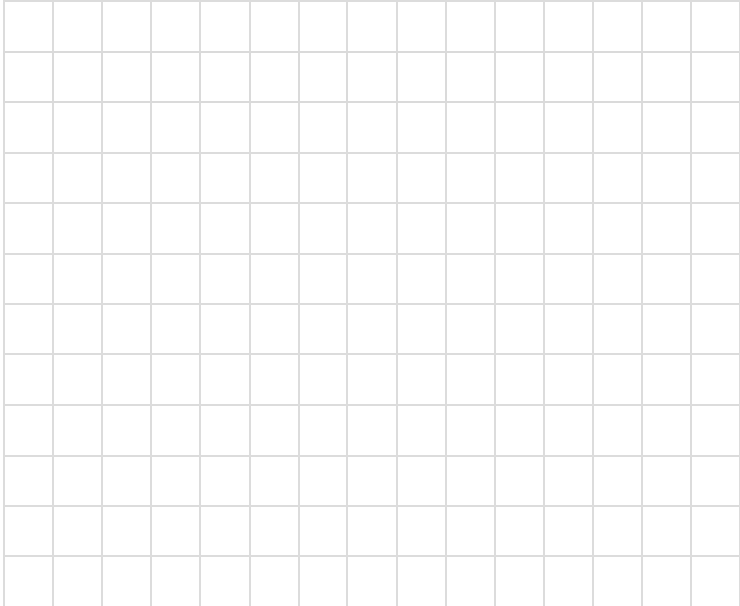
Think of the best way to use `gpsx`, `gpsy`.

## SECTION 16 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>The comparison to real GPS systems is obvious, although there are differences. Karel's coordinates can also be thought of as elevations on a topographic map, or stairways, or shelves. How could you apply your understanding of these values when coding real problems?</i>	

## SECTION 16 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 the sensors** `gpsx` **and/or** `gpsy`. 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 17: USING BOOLEAN VALUES TRUE AND FALSE

In this section, you learn how to use Boolean (logical) values `True` and `False`, store them in Boolean or logical variables, return Boolean values from Boolean functions, and use Boolean variables in conditions and `while` loops. You know that Karel's sensors such as `wall`, `nugget`, `mark`, `empty`, `north` etc. are Boolean functions. With Boolean variables you can do logical operations such as `and` or `or`. The symbol `=` is used to assign a value to a variable, and for mathematical equality ("is equal to") the symbol `==` is used. The result of a comparison such as `a == b` is either `True` or `False`.

This is a good time to review local and global variables. They are used extensively in this section.

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

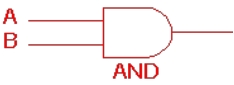
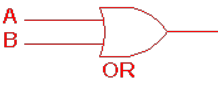
Level	Variable starts as	Condition Test	Outcome (True)	Outcome (False)
17.1	not used on this level	sensor keyword	Karel finds the sensor and prints "sensor:" True	Karel does not find the sensor (he either finds an empty cell or another sensor) and prints "sensor:" False
17.2		<code>if snake sn = true</code>		
17.3	<code>nug = false</code>			
17.4			Karel finds a bottle in every square. <code>if success (is met)</code> , prints "The row was complete!"	
17.5				<code>else (success is not met)</code> ; prints "One or more bottles is missing".
17.6		<code>while not home fo=(fo or map)</code>		
17.7				

## SECTION 17 NOTES

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


## QUESTIONS

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
<b>AND GATE</b> 	A = True (On) B = True (On)	
	A = True (On) B = False (Off)	
	A = False (Off) B = True (On)	
	A = False (Off) B = False (Off)	
<b>OR GATE</b> 	A = True (On) B = True (On)	
	A = True (On) B = False (Off)	
	A = False (Off) B = True (On)	
	A = False (Off) B = False (Off)	

There are other types of gates, too: NOT, NAND, NOR, XOR, and XNOR! A typical computer microprocessor has hundreds of millions of gates in it.

## SECTION 17 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Look up Boolean operations on line. You will be amazed at how many combinations can trigger a true or false result.</i>	<i>You can restrict your search to Boolean operations in Python if you want to see what the code looks like.</i>

## SECTION 17 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 variable based on a true or false condition. 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)																																																																																																																																																																																					
	<table border="1"> <tbody> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </tbody> </table>																																																																																																																																																																																					
Storyline:																																																																																																																																																																																						
Karel's goals:																																																																																																																																																																																						
Number of steps:	Keywords:																																																																																																																																																																																					
	Required (must use):																																																																																																																																																																																					
Number of operations:	Forbidden (can't use):																																																																																																																																																																																					
Any special challenges:																																																																																																																																																																																						

## SECTION 18: USING THE FUNCTION RANDINT()

In this section, you learn how to generate random integers using the function `randint()`, make Karel repeat something a random number of times, calculate the maximum and the minimum of a given set of numbers. You know that the function `randint(6)` can be used to simulate rolling dice.

`randint()` was used to simulate a game of chance in 18.1 and 18.2, and to build columns of random height in 18.3 and 18.4. In 18.5 to 18.7, you learned to write a function to determine maximum and minimum values of those columns.

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

Conditions	Code
Rolling “snake eyes” 	
Rolling a 7 on a dodecahedral die 	

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?


## SECTION 18 NOTES

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


## QUESTIONS

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.				7.			
3.				8.			
4.				9.			
5.				10.			
Score (Correct/Total)				Did you pass?			

Is this a good application for randomness? Explain.

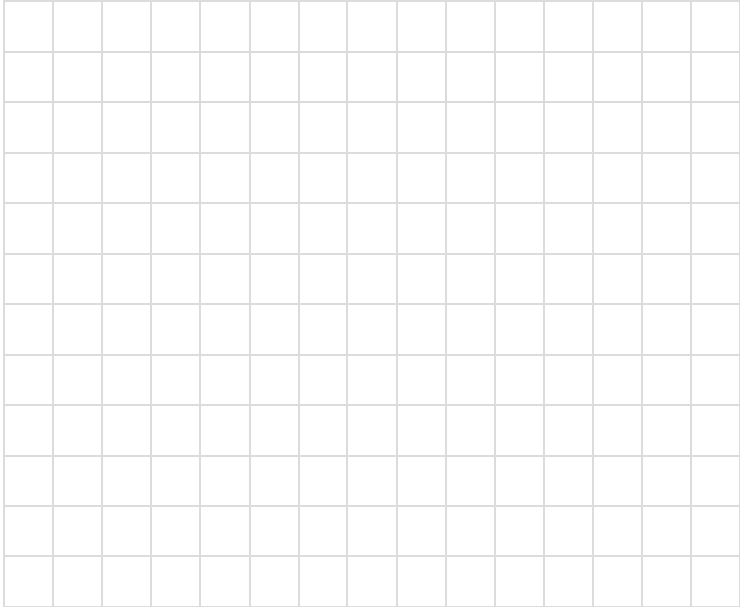
--

## SECTION 18 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Look at images (photographs, animations, CGI, impressionist paintings). What random patterns do you see?</i>	<i>Look up Wolfram Rule 30.</i>

## SECTION 18 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 `randint()` and tests for either a maximum or a minimum.** 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 19: EMPTY AND NON-EMPTY LISTS

In this section, you learn how to create empty and non-empty lists, append items to a list using `append()`, go through list items one at a time, and get the length of a list `L` using `len(L)`. You know that lists are like variables, but they can hold multiple values.

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

Code	Meaning
<code>A = [1, 3, 5, 7, 9]</code>	
<code>for x in L   print "Map found at:", x</code>	
<code>Y = [ ]</code>	
<code>C.append (x)</code>	
<code>len(m)</code>	

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

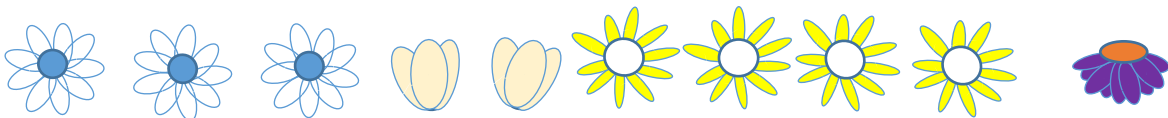
Start with an <b>empty list</b> .	
In order, <b>add</b> three erasers, one pencil sharpener, two pencils, one pen	
Parse the list using a For loop to print out each item.	
Find the length of the list.	

## SECTION 19 NOTES

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

## QUESTIONS

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.




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

## SECTION 19 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Think of lists that we make every day: not just "to do" lists, or grocery lists.</i>	<i>Think of situations where you are counting, mapping, or recording data.</i>

## SECTION 19 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. **Create an empty list and append items to it.** 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 20: WORKING WITH LISTS

In this section, you learn how to remove and return the last item of a list using `pop()`, remove and return the first item of a list using `pop(0)`, get the length of a list using `len()`, use the for loop to go through lists one item at a time, and merge lists. You know that list items can be numbers, Boolean variables, and even text strings. Lists can contain other lists, such as `[gpsx, gpsy]` pairs.

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

Code	Meaning
<code>b = A.pop( )</code>	
<code>if orchid o.append([gpsx, gpsy])</code>	
<code>repeat 4 la = X.pop()</code>	
<code>n = L.pop(0)</code>	
<code>R.append(R2.pop(0))</code>	

Build commands for Karel. We are making a map of coin locations, using a list `m`.

Start with an empty list.	
While not home, move forward. If there is a coin, add True to the list. Otherwise add False.	
If there is a wall, turn left. If there is a wall, turn right twice.	
Print the list.	

## SECTION 20 NOTES

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


## QUESTIONS

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.

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)

--

## SECTION 20 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Python lists are powerful. Think of how these can be applied in real life.</i>	<i>Think of adding and removing items, merging information, and using different types of sensors.</i>

## SECTION 20 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. **Create two sections to the game. In the first section, you will create a list that will be used in the second section.** 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 Jr 4. 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 gpx and gpsy coordinates to locate, get or put items; set conditions with coordinates (==, !=, <, >)			
Use Boolean operators True and False to determine outcomes			
Use random integers to play a game or create a pattern.			
Find the maximum or minimum value of a set.			
Create, append and remove items from a list.			
Use for loops based on a list.			
Merge lists.			

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

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

## LIST OF BASIC COMMANDS AND KEYWORDS FROM KAREL JR 1, 2, 3, AND 4

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.

### 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 FROM KAREL JR 1, 2, 3 AND 4 (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`



## FILE LOG: GAMES I HAVE CREATED

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



## NOTES