Starlogo Idioms:

Definition: an idiom is
(1) a sequence of words which forms a whole unit of meaning,
(2) a manner of speaking that is natural to native speakers of a language

In Starlogo, an idiom is a action that can be described easily in a few word but coding the behavior takes several commands.

In this document, the following format will be used:

Observer/Turtle/Patch What type of idiom	code Commands that make up the action	example(s) Where it can be seen in use
the idiom description in English	Commands that make up the action	example.slogo

For example,

To setup a new program
 	
Usually a setup procedure clears the canvas of patches, turtles, and graphics left by the previous run, then creates the environment and the agents.
 	
Agents and the environment are then created and initialized prior to running a new simulation.
 	
Running the simulation usually involves setting the agents and/or environment into action.
	to setup
	  clearall
	  create-environment
	  create-agents
	end
	
	to create-environment
	  ; set up the environment
	end
	
	to create-agents
	  ; set up the agents
	end
	
	to go
	  ; tell agents and environment what to do
	end
		idioms-newrun.slogo

OBSERVER IDIOMS		

To create turtles (using crt)
 	
The observer procedure setup instructs the observer to create 10 turtles.
	to setup1
	  ca
	  crt 10
	end
		Idioms-create.slogo

To create turtles (using create-and-do)
 	
The observer procedure setup instructs the observer to create 10 turtles then ask each of the 10 turtles to set its color to red and set its shape to turtle-shape.
	to setup2
	  ca
	  create-and-do 10 [
	  setcolor red
	  setshape turtle-shape
	  ]
	end
		Idioms-create.slogo

To create turtles (using sprout)	*See "To create turtles" in the **Patch Idioms** section*	
To create turtles (using hatch)	*See "To create turtles" in the **Turtle Idioms** section*	

To make a terrain
 	
Create 10 hill-making turtles
Ask each turtle to do the following:
 	
Set its position to a random x and y value
Set their color to dark brown
Repeat 45 times:
  Color the patch it is on dark brown
  Turn a random direction
  Take a step forward
 	
Kill all turtles (they are no longer needed)
 	
Ask each patch dark brown patch to become a peak.
  Repeat 45 times:
  diffuse the height of each patch over its neighboring patches.
	patches-own [height]
	
	to make-terrain
	  crt 10 ; turtles create the foundations for hills
	  ask-turtles
	  [
	  setxy (random screen-height) (random screen-width)
	  setc 33
	  repeat 45
	  [stamp color
	  seth random 360
	  fd 1]
	  ]
	  ask-turtles [die] ; end of hill foundation making
	
	  ask-patches ; creates slopes for hills through difusion
	  [if pc = 33 [set height ((random 50) + 25)]]
	  repeat 45
	  [diffuse height .9]
	  ask-patches [scale-pc green height 0 50]
	end
		Idioms-terrain.slogo

To distribute trees to cover a percentage of		Idioms-

the canvas Multiply the number of patches on the canvas by the density divided by 100 to get the number of trees. Repeat for each of the trees; Create a tree Set its breed to trees Set its shape to tree shape Place it on the canvas (but not on top of another tree)	breeds [trees] ;this is needed in the turtle procedure area ; also must define a shape called tree-shape to distribute-trees ; uses a slider "density" ranging from 0 –100 repeat ((density / 100) * screen-height * screen-width) [create-and-do 1 [setbreed trees ;setshape tree_shape ; must have this shape place-uniquely ; see turtle idiom "place-uniquely"]] end	distribute.slogo
To setup a new run Usually a setup procedure clears the canvas of patches, turtles, and graphics left by the previous run, creates the environment and the agents. Agents and the environment are then created and initialized prior to running a new simulation. Running the simulation usually involves setting the agents in action.		Idioms-newrun.slogo

Turtles:

To kill another turtle
 	
Turtles must have a variable "killed?" that is either set to false or true.
 	
Initially all turtles are alive so "killed?" is set to false.
 	
Using the kill procedure, the "killed?" variable of the selected turtle is set to true.
 	
In the turtle-go procedure, each turtle checks to see if its "killed?" variable has been set to true. If so, it dies. Then it checks to see if another turtle is on the same patch. If so, it kills the other turtle.
 	
Can you figure out why turtles die in pairs?
	turtles-own [killed?]
	
	to setup
	  setkilled? false
	end
	
	to kill-turtle :another
	  setkilled?-of :another true
	end
	
	to turtle-go
	  walk-randomly ; see idioms-walk-randomly
	  if killed? [die]
	  if count-turtles-here < 2 [stop]
	  kill-turtle one-of-turtles-here
	end
		idioms-kill-turtle.slogo

To place turtles randomly on the screen Set the x coordinate of the turtle to be a random value between 0 and (screen-width – 1). Set the y coordinate of the turtle to be a random value between 0 and (screen-height – 1). n.b. The center of the canvas is 0,0 so can you figure out why this works correctly?	to place-randomly setxy random screen-width random screen-height end	idioms-place-randomly.slogo
To orient turtles randomly Set the turtle's heading to be some random heading between 0 and 359 degrees where a heading of 0 is towards the top of the screen (North).	to orient-turtles-randomly setheading random 360 end	idioms-orient-randomly.slogo

To orient turtle outward from screen center Set the turtle's heading to an angle given by using ycor as the numerator xcor as the denominator in the equation: Arctangent (opposite/adjacent) = angle n.b. the center of the screen has coordinates (0,0) and an angle 0 is North rather than East.	to set-heading-outward setheading 90 - (atan ycor xcor) end	idioms-orient-outward.slogo
To make turtles walk randomly To make a turtle walk randomly we change its heading before it walks forward a step. Random 90 returns a number between 0 – 89 for the right turn and the left turn. To show all the possible values for a right turn followed by a left turn you can form a 2D matrix with 0-89 as row values and 0-89 as column values. Fill in the values and what do you find? Are the values equally distributed or did you find a weighted distribution?	to walk-randomly rt random 90 lt random 90 fd 1 end	idioms-walk-randomly.slogo
To avoid a patch of a certain color To avoid a red patch, check to see if the patch in front of the turtle is red. If so, change the turtles heading and move forward one step, else just take a step.	to avoid-red-patch ifelse pc-ahead = red [rt random 360] [fd 1] end	idioms-avoid-red-patch.slogo

To follow another turtle
 	
To follow a turtle with ID number passed in the variable ":leader", set the turtle's heading to point to the position of the leader (given by the leader's current x and y coordinates). Then take a step.
	to follow :leader
	  setheading towards xcor-of :leader ycor-of :leader
	  fd 1
	end
		Idioms-follow.slogo

To run away from another turtle
 	
To run away from a turtle with ID number passed in the variable ":another", simply set the turtle's heading to be the opposite of the direction towards ":another".
	to runaway :another
	  setheading ((towards xcor-of :another ycor-of :another) + 180
	  fd 1
	end
		Idioms-runaway.slogo

To place turtles randomly without overlapping another turtle
**uses recursion*
 	
To place a turtle on a randomly selected patch where no other turtle resides, first move to a randomly selected patch. Then see if there is only one turtle at this location. If so, then the turtle stays, else the turtle must try another patch until it finds one where it is the lone turtle. The stop command exits the current procedure.
 	
Sequence:
Jump to a new patch
  I'm not the only one here
  Jump to a new patch
  I'm not the only one here
  Jump to a new patch
  I am the only one here!
  Stop -> exit this search
	to place-uniquely
	  setxy random screen-width random screen-height
	  if count-turtles-here = 1 [stop]
	  place-uniquely
	end
		idioms-place-uniquely.slogo

To use breeds
 	
Place this declaration at the top of the Observer or Turtle command center. It tells Starlogo that your turtles to have two different breeds, `dogs` and `cats`.
 	
The create-breed command creates the specified number of turtles of a breed.
 	
To have different breeds take different actions, check to see what breed is and then take the corresponding action.
	breeds [dogs cats]
	
	to create-dogs-and-cats
	  create-dogs 10
	  create-cats 10
	end
	
	to move-pet
	  if breed = cats [fd 1]
	  if breed = dogs [fd 2]
	end
		Idioms-breeds.slogo

To use hidden turtles to alter the environment
 	
Declare that there is a breed called a grassmaker.
 	
Create one grassmaker and have it hide itself.
 	
In the go procedure, the grassmaker walks around randomly (see the turtle idiom *"To walk randomly")* and changes the patch it is on to green.
	breeds [grassmaker]
	
	to create-agents
	  create-grassmaker-and-do 1
	  [hideturtle]
	end
	
	to go
	  ask-grassmaker
	  [walk-randomly
	  stamp green]
	end
	
	to walk-randomly
	  rt random 90
	  lt random 90
	  fd 1
	end
		idioms-hidden-turtle.slogo

To accelerate / decelerate a turtle
 	
First at the top of the turtle procedures window declare that turtles have a variable called "speed".
 	
When the turtle is told to move, it takes "speed" steps forward.
 	
To accelerate a turtle, increase the value of "speed"
 	
To decelerate a turtle, decrease the value of "speed"
	turtles-own [speed]
	
	to move
	  fd speed
	end
	
	to accelerate
	  setspeed speed + 1
	end
	
	to decelerate
	  setspeed speed – 1
	end
		idioms-speed.slogo

To control the number of births
 	
In this example, we stop populating the canvas once there is one turtle for each patch. We check to see if there are fewer turtles than patches. If so, turtles give birth to new turtles.
	to turtles-go
	  if count-turtles < (screen-width * screen-height)
	  [give-birth]
	end
	
	to give-birth
	  hatch
	  [setage 0]
	end
		idioms-control-births.slogo

To grab a partner based on breed
 	
First at the top of the turtle procedures window declare that there are two kinds of turtles: dogs and cats. Also declare that the turtles have a variable called "owned?", and a variable called owner.
 	
To grab a dog, set it's owned? variable to true, and set it's owner to your id.
	breeds [dogs humans]
	turtles-own [owned? owner]
	
	to adopt-a-dog
	  grab one-of-dogs-here
	  [setowned?-of partner true
	  setowner–of partner who]
	end
		idioms-grab-partner-of-breed.slogo

To create groups of different breeds, each a certain percentage of the population
 	
Create two sliders "%dogs" and "%cats" with values 0 – 100 and a slider called "numpets".
 	
Check to see that %dogs + %cats = 100
	breeds [dogs cats]
	
	to create-agents
	  ifelse (%dogs + %cats = 100)
	  [
	  create-dogs-and-do ((numpets) * (%dogs / 100))
	  [setcolor red place-uniquely]
	  create-cats-and-do ((numpets) * (%cats / 100))
	  [setcolor blue place-uniquely]
	  ]
	  [
	  print "error: %dogs + %cats must equal 100"
	  ]
	end
		idioms-percent-population.slogo

To measure the distance to another turtle	globals [distance-between] to find-distance-to :another setdistance-between (distance (xcor-of :another) (ycor-of :another)) end	idioms-distance-between.slogo

To find nearest turtle In turtles-go, all turtles walk around randomly. Then if it is a male, find the distance to the female and store it in the turtle variable "distance-from-female". If it is a female, find out which male is closest and store it in the global variable "nearest".	globals [nearest] breeds [female male] …… turtles-own [distance-from-female] to find-nearest :another setnearest who-min-of-turtles [distance-from-female] end to find-distance setdistance-from-female (distance (xcor-of 0) (ycor-of 0)) end to turtles-go walk-randomly ifelse breed = female [find-nearest 0] [find-distance] end	idioms-find-nearest.slogo
To find the number of turtles in my immediate neighborhood. Sets num-neighbors to be the number of turtles inside a circle of radius 2 (minus oneself) centered at the current position of the turtle named ":id".	globals [num-neighbors] to find-num-neighbors :id setnum-neighbors (count-turtles-with [(distance xcor-of :id ycor-of :id) < 2] – 1) end	idioms-count-neighbors.slogo

To make a turtle take an action if a threshold is reached. The turtles randomly run around the screen until the run out of energy (energy = 0). Once run out of energy they become inactive until they restore their energy by resting for a period of time.	turtles-own [energy active?] to turtle-go ifelse (energy > 0) and active? [set energy energy - 1 wiggle] [ifelse (energy < 100) [set active? false set energy energy + 1] [set active? true]]	idioms-threshold.slogo

	display-energy end	
To scale turtle color based on variable. Creates generally happy and sad (red and blue) turtles. When a turtle encounters a sad turtle, it gets sadder. When it encounters a happy turtle, it gets happier. The shade of red or blue indicates the degree of happiness or sadness.	turtles-own [emotion-level my-color] to display-emotion scale-color my-color emotion-level –45 45 end	idioms-emotion.slogo
To change turtle shape based on state. Turtles and rabbits run around the screen a various speeds. Rabbits run the fastest and as they run out off energy the turn into turtles and move more slowly. Once energy hits 0 the turtles rest until they turn back into rabbits.	turtles-own [energy] to change-shape ifelse (energy < 30) [setshape turtle-shape] [setshape rabbit-shape] end	idioms-shape.slogo

To bounce off a wall.
 	
Turtles are enclosed in a box and bounce off the walls by adjusting their headings if the encounter colored patches.
	to check-wall
	  if pc-ahead = red ; SOUTH wall
	  [
	  ifelse (heading < 180)
	  [setheading 180 - heading]
	  [setheading 540 - heading]
	  check-wall]
	if pc-ahead = sky ; EAST and WEST walls
	  [
	  setheading 360 - heading
	  check-wall]
	if pc-ahead = lime ; NORTH wall
	  [
	  ifelse (heading < 180)
	  [setheading 180 - heading]
	  [setheading 540 - heading]
	  check-wall]
	end
		idioms-walls.slogo

To collide
 	
Turtles run around the screen at a certain speed and collide with each other. Momentum is conserved when the collide.
	turtles-own [speed energy totalenergy]
	
	to collide
	  grab one-of-turtles-here
	  [seth random 360
	  seth-of partner random 360
	
	  setenergy (totalenergy * (((random 99) + 1) / 100))
	  setenergy-of partner totalenergy - energy
	  setspeed (sqrt (energy / 500))
	  setspeed-of partner (sqrt ((energy-of partner)/ 500))
	  check-wall
	  fd 1]
	end
		idioms-collision.slogo

To eat and gain energy	to eat-grass ;turn the patch to black and increase energy if pc = green [stamp black setenergy energy + 1] end	idioms-eat.slogo

To walk and lose energy	to walk
	  rt random 50
	  lt random 50
	  fd 1
	  setenergy energy - 0.25
	end
		idioms-eat-and-walk.slogo

To reproduce under certain circumstances Hatch-threshold set by slider	to reproduce if energy > hatch-threshold [setenergy energy / 2 hatch []] end	idioms-eat-walk-reproduce.slogo

To get sick some percent of the time	turtles-own [sick?]
	
	to infect
	  rt random 100
	  lt random 100
	  fd 1
	  if sick? and ((random 100) < 90)
	  [setsick?-at 0 0 true]
	end
		idioms-infect.slogo

To maintain a long term relationship with another turtle	turtles-own [buddy] to get-a-buddy :myfriend setbuddy :myfriend end	idioms-buddy.slogo

To create a population with randomly distributed age	Turtles-own [age] ; in turtle procedure window
	
	To setup
	  Create-and-do 100
	  [setage random 100]
	end
		idioms-random-age.slogo

To move based on wind direction	Globals [wind?]
	
	to get-blown-by-wind
	  if wind? ; use value of wind_direction set on slider
	  [setheading wind_direction + ((random 30) – 15)
	  fd 1
	  ]
	end
		idioms-wind.slogo

To walk uphill / downhill	patches-own [height]
	turtles-own [maxheight maxhead]
	
	to walk-uphill
	  setmaxheight height
	  repeat 4 ;compare current height to heights around you
	  [
	  if (height-towards 0 1) > maxheight
	  [setmaxheight (height-towards 0 1)
	  setmaxhead heading]
	  rt 90
	  ]
	  seth maxhead ;turn toward highest nearby peak
	  fd 1
	end
		idioms-uphill.slogo

To age	 Turtles-own [age] to age	idioms-age.slogo

	seth random 360 fd 1 setage age + age / 100 scale-color white age 100 0 end	
To random death	To random-death ifelse (random 100) < chance-of-dying [die] [seth random 360 fd 1] end	idioms-random-death.slogo
To make only a certain percentage of turtles perform an action	Turtles-own [behavior] to create-agents ; often invoked by pushing a button let [:cnt 0] repeat num-turtles [create-and-do 1 [ifelse :cnt < percentage / 100 * num-turtles [set behavior 0 setc blue] [set behavior 1 setc red] setxy random screen-width random screen-height] set :cnt :cnt + 1] end to go fd 1 if behavior = 1 [seth random 360] end	idioms-percent-behavior.slogo

To move to an empty patch in the neighborhood	to move-to-empty-spot
	  let [:currentdir ((random 8) * 45)]
	  repeat 8
	  [seth :currentdir
	  if (count-turtles-towards 0 1) < 1 ;found an empty spot
	  [
	
	  ]
	  set :currentdir :currentdir + 45
	  ]
	end
		idioms-move-to-empty.slogo

To find and alter a turtle in the neighborhood.	to find-and-alter
	  let [:currentdir ((random 8) * 45)]
	  repeat 8
	  [seth :currentdir
	  if ((count-turtles-towards 0 1) > 0) [
	
	  [setc-of partner yellow]
	  stop
	  ]
	  set :currentdir :currentdir - 45
	  ]
	end
		idioms-alter-turtle.slogo

To give birth
In this example, a turtle gives birth to a number of offspring after a gestational period. At birth each offspring has age set to 0, a random heading, and is placed a small distance away from the mother.
	to lay-eggs
	  setc red
	  wait gestation-time
	  repeat num-offspring
	  [
	  hatch [
	  setage 0
	  setc blue
	  seth random 360
	  jump random 3
	  ]
	  setc blue
	  set age 0
	end
		idioms-birth.slogo

To make a turtle change another turtle in the distance	setshape-of partner happy-face	idioms-change-turtle.slogo

Patches

To disperse a chemical on patches	patches-own [chemical] to create-environment ask-patches [setchemical random 100] diffuse chemical 0.99 ask-patches [scale-pc blue chemical 0 100] end to evaporate ask-patches [set chemical chemical * .9 scale-pc blue chemical 0 100] end	idioms-chemical.slogo
To save current background and restore	patches-own [old-patch-color] to restore-graphics ask-patches [setpc old-patch-color] end to store-graphics ask-patches [setold-patch-color pc] end to create-graphics cg create-and-do 1[repeat 20 [seth random 360 pd fd 10 pu jump random 5]] end	idioms-background.slogo
To have patches display the amount of a substance contained there	Patches-own [food] to see-how-much-food ask-patches [scale-pc green food 0 100]	Idioms-patches-display-amount.slogo

	end	

To create proportions of types of patches	;---
	; observer procedures
	
	to setup
	  ca
	  crt 1
	end
	
	to create-different-types-of-patches
	  ifelse (%red + %blue) > 100
	  [
	  print "error: % red + %blue must be less than or equal to 100"
	  stop
	  ]
	  [
	  let [:numpatches (screen-height * screen-width)]
	  let [:numredpatches (:numpatches * (%red / 100))]
	  repeat :numredpatches [ask-turtles [placeUniqueRed]]
	  let [:numbluepatches (:numpatches * (%blue / 100))]
	  repeat :numbluepatches [ask-turtles [placeUniqueBlue]]
	  ]
	end
	
	;---
	; turtle procedures
	
	to placeUniqueRed
	  if pc = black [stamp red stop]
	  setxy random screen-width random screen-height
	  placeUniqueRed
	end
	
	to placeUniqueBlue
	  if pc = black [stamp blue stop]
	  setxy random screen-height random screen-width
	  placeUniqueBlue
	end
		idioms-%ofdifferentpatches.slogo

To randomly populate the world with trees	to make-trees :count	idioms-trees.slogo

(as patches)	if :count > 0 [ifelse pc = black [stamp green make-trees :count - 1] [setxy random screen-width random screen-height make-trees :count]] end	
To distribute patch color randomly	to color-patches :count if :count > 0 [ifelse pc = black [stamp blue color-patches :count - 1] [setxy random screen-width random screen-height color-patches :count]] end	idioms-distribute-patch-color-randomly.slogo
To scale a color	patches-own [scale-value] to create-environment ask-patches [set scale-value (random 100)] ask-patches [scale-pc red scale-value 0 100] end	idioms-scale-pc.slogo
To color a region	to create-environment ask-patches [if (abs xcor) < (abs X) and (abs ycor) < (abs Y) [setpc red]] end	idioms-color-region.slogo
To draw a function	to place-on-function setxy random screen-height random screen-width if (count-turtles-here != 1) or (int ycor) = (int xcor ^ 2) [place-on-function] end	idioms-draw-function.slogo

Other:

To roll of dice, probabilistic event Creates 100 red turtles and places them randomly on the screen. For every turtle there is a 30% chance it will turn blue. (Think of this as rolling a 100-sided die with sides numbered 0 - 99, if it comes up with the number less than	to setup ca create-and-do 100 [setxy random screen-width random screen-height setcolor red] end to turtle-go if (random 100) < 30	idioms-rolldice.slogo

30, which it should roughly 30% of the time, turn the turtle blue.)	[setcolor blue] end	
To kill three out of ten turtles Create 10 turtles (with ids 0 - 9) and place them randomly on the screen. Repeat the kill_1 procedure 3 times Kill_1 selects a random id between 0 – 9, if the turtle with that id is still alive, kill it, if the turtle with that id is already dead, recursively call kill_1.	to setup ca create-and-do 10 [setxy random screen-width random screen-height] end to kill_3_of_10 repeat 3 [kill_1] end to kill_1 let [:id_to_die random 10] ifelse (alive?-of :id_to_die) = true [kill :id_to_die] [kill_1] end	idioms-kill_3_of_10.slogo
To create a loop counter Uses a global variable to keep track of the number of times the main loop of the program called “observer-go” is executed.	Globals [loopcount] to setup setloopcount 0 end to observer-go setloopcount loopcount + 1 wait 1 end	idioms-loopcount.slogo
To create a timer	Globals [time] to clock every 1 [set time (time + 1)] end	idioms-create-timer.slogo
To output data to the output window	Globals [loopcount numrandomdeaths numsuicides] to setup ca createteens	

	set loopcount 0 set numrandomdeaths 0 set numsuicides 0 outputsettings end to outputsettings ; saved from slider values type "number of teens " print numteens type "number of counselors: " print numcouns type "odds of events: " print eventodds end to observerGo ask-turtles [go] set loopcount loopcount + 1 output-data if loopcount = 1000 [stopall] end to output-data type loopcount type ", " type numrandomdeaths type ", " print numsuicides end	
To output data to a file	Open output window in Starlogo (On the menubar go to Windows and select "Output Window" Write data to the output window (see above). Once the program stops, save the output file as a text file using "Save Output As" in the Output window. Open the text file in excel using "," as a delimiter. You should now see your data in columns and rows.	
To keep track of multiple runs	Name the output data file to contain the variable settings	
To use recursion		As in Teen World
To import a picture as a background	`import-picture-name c:\pussycat.jpg`	

Unclear what to make of these:

To make a turtle to end a behavior
 	
*what was the point of this?
	to check-patch-color-once
	  if pc-ahead = red [lt 90 stop]
	  if pc-ahead = blue [rt 90 stop]
	  fd 1
	end	