Module 1. An Introduction to Frequency and Period

Sound is produced by the back-and-forth motion of a vibrating object. In the 19th century, mechanical mechanisms were developed to capture and reproduce this motion. In the 20th century, electronic devices were invented to amplify sound. In the 21st century, computers are used to record and reproduce sound digitally.

This module provides an introduction to ways in which a digital computer can be used to capture and reproduce sound. The sound laboratories and experiments that follow use an educational computing language, Snap*!*, developed at the University of California, Berkeley.

Topic 1.1 Securing a Snap! Account

A free Snap*!* account can be obtained from the University of California, Berkeley web site:

https://Snap.berkeley.edu/Snap

Secure a Snap*!* account before continuing. The Snap*!* reference manual is available here:

https://Snap.berkeley.edu/Snap/help/SnapManual.pdf

Section II of the reference manual provides information about securing a Snap*!* account and saving projects. Review this section before continuing to the explorations that follow below.

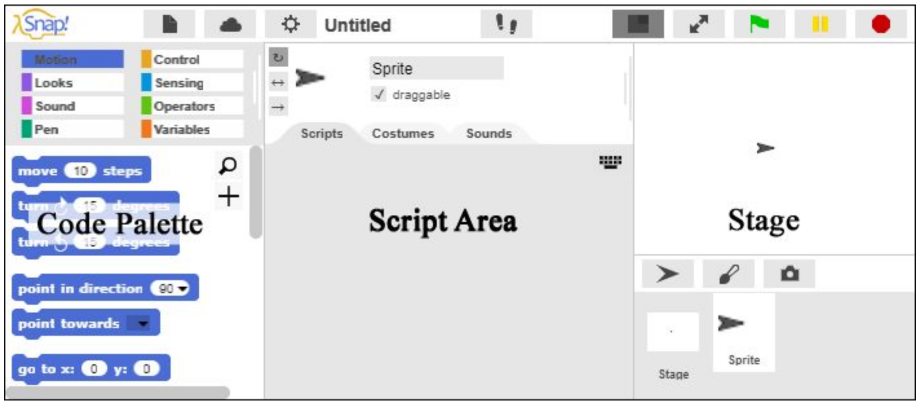
Snap*!* help forums maintained at the University of California, Berkeley, are available here:

https://forum.Snap.berkeley.edu/

Assistance with questions that are not addressed in the reference manual can be obtained through the Snap*!* forum.

Topic 1.2 The Snap! Workspace

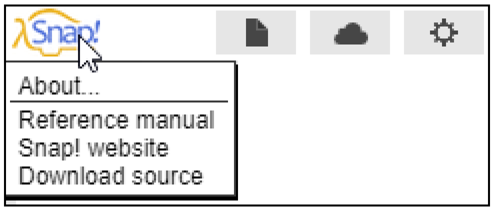
In a typical Snap*!* session, blocks of code are dragged from the *Code Block Palette* on the left-hand side of the screen to a *Script Area* in the middle of the screen. Blocks of code are snapped together to create scripts. Clicking a group of code blocks causes the script to run, performing an action. Often these actions involve movement of sprites on a *Stage* at the right-hand side of the screen.



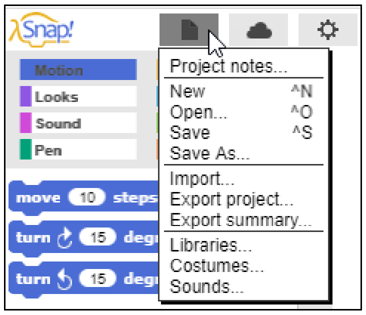
A *sprite corral* beneath the stage indicates which sprite is currently selected. Each sprite has its own separate script space.

Topic 1.3 Snap*!* Menus

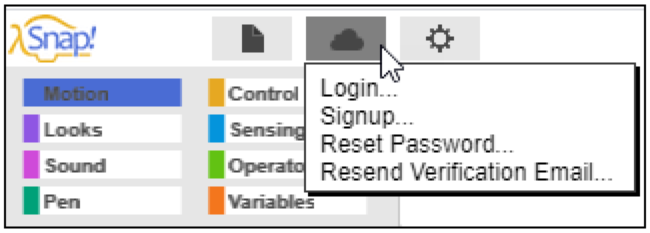
There are several menus that can be accessed in the top left-hand corner of the Snap*!* screen. The Snap*!* icon in the top left-hand corner can be used to access the Snap*!* reference manual.



The *File* menu to the right of the Snap*!* icon can be used to save projects and open projects that have been previously saved. This menu can also be used to access costumes for sprites, sounds, and libraries of additional Snap*!* code blocks.



The *Login* menu is to the right of the *File* menu.



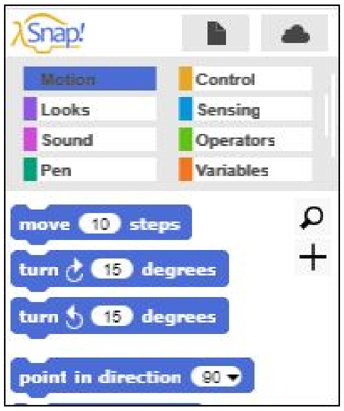
A user must be logged into Snap*!* in order to save projects. Therefore it is a good idea to log in to Snap*!* at the beginning of every session.

A *Settings* menu is to the right of the *Login* menu. This menu provides a number of options for customizing Snap*!* For example, the *Zoom Blocks* option can be used to increase the size of the code blocks so that they can be more easily viewed in presentations.

Exploration 3.1 If you have not already, log into Snap*!* Explore some of the options available under the different menu settings.

Topic 1.4 The Code Block Palette

Scripts are created in Snap*!* by snapping blocks of code together (such as the **Move 10 Steps** code block in the illustration). The types of *code blocks* available are displayed in a *Code Block Palette* at the top left-hand side of the screen. For example, the *Motion* code blocks are currently highlighted in the palette below. Other categories of code blocks include *Looks*, *Sound*, *Pen*, *Control*, *Sensing*, *Operators*, and *Variables*. Each category is a different color (e.g.. *Motion* code blocks are blue). Click on the different categories (Motion, Looks, etc.) to access the code blocks associated with that category.



The *Motion* code blocks direct the movement of sprites (actors that can move about the stage on the right-hand side of the screen.) The *Looks* code blocks control the appearance of sprites. The *Sound* code blocks are used to play sounds. The *Pen* code blocks control the color and thickness of the turtle's pen. The *Control* code blocks provide control structures such as the **Repeat** command. The *Sensing* code blocks are used to sense the status of Snap*!* objects and monitor external inputs such as the keyboard and the microphone. The *Operators* code blocks provide mathematical and logical functions. The *Variables* palette is used to create and modify variables.

Exploration 1.4 The Code Block Palette

Click on each of the categories in the *Code Block Palette* to get a sense of the types of commands that are found under each category.

Topic 1.5 A Digital Clock

Sound is produced by the back-and-forth motion of a vibrating object. The rate at which the back-and-forth motion occurs is an important characteristic of sound. Two closely related terms used to describe this characteristic are *frequency* and *period*.

Frequency. The rate at which a back-and-forth motion occurs over a given amount of time. The number of back-and-forth events that occur within a given time period is known as frequency.

Period. The duration of time for a single back-and-forth event is known as its *period*.

These terms are used to characterize sound, but are also used to define other non-auditory events. The motion of the rotating flywheel of an engine can be described in terms of *revolutions per minute*. For example, the flywheel might be described as rotating at a rate of 60 revolutions per minute (rpm). In this instance, the frequency of motion could be described as *60 rpm*. If the frequency is 60 rpm, this means that one complete rotation would occur in one second. Therefore, the period in which a single rotation occurs could be described as *1 second*. In this example, frequency and period are related. If the frequency increases, the period in which one rotation occurs will decrease.

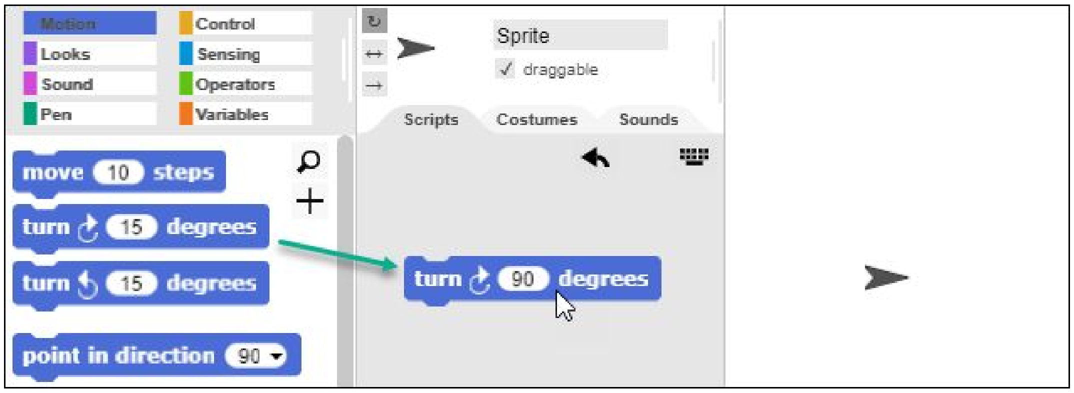
The concepts of frequency and period can be applied to the back-and-forth vibration of an object that produces sound in a similar manner. The time scale for vibration that produces sound is typically *events per second* rather than events per minute, however.

A digital clock is at the heart of every digital computer. The digital clock is used to control the timing and rate at which computer instructions are executed. It can also be used to create a digital timer that can be used to time events.

Topic 1.6 Rotating the Turtle

The examples that follow make use of the digital clock to illustrate ways in which code blocks from several different code block palettes are used. In this example a timer will be constructed that will keep track of seconds as the turtle rotates. The user will be able to stop the timer (and the turtle) by pressing the space bar.

The code blocks in the *Motion Palette* are blue. Drag the **Turn Right** code block into the script area and enter 90 degrees as the input to this code block. Then click the code block. Each time the code block is clicked, the screen turtle should turn right 90 degrees.

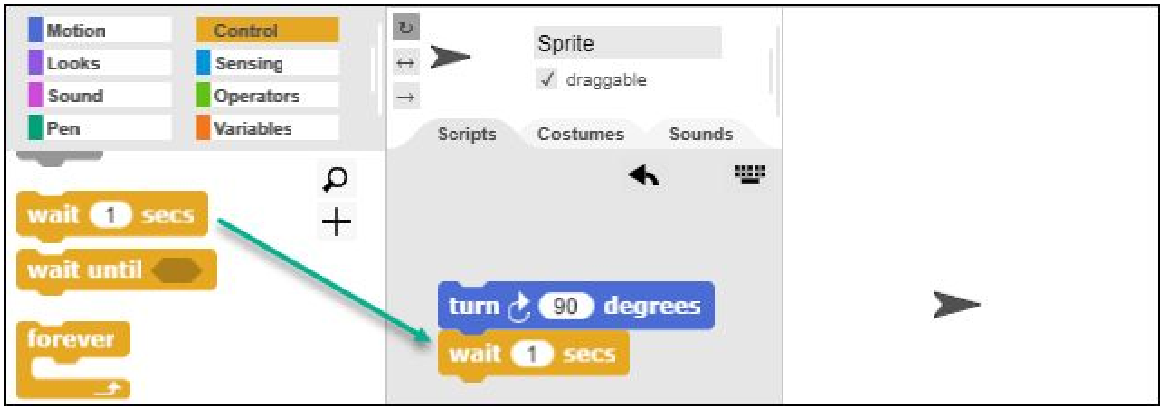


Exploration 1.6 Rotating the Turtle

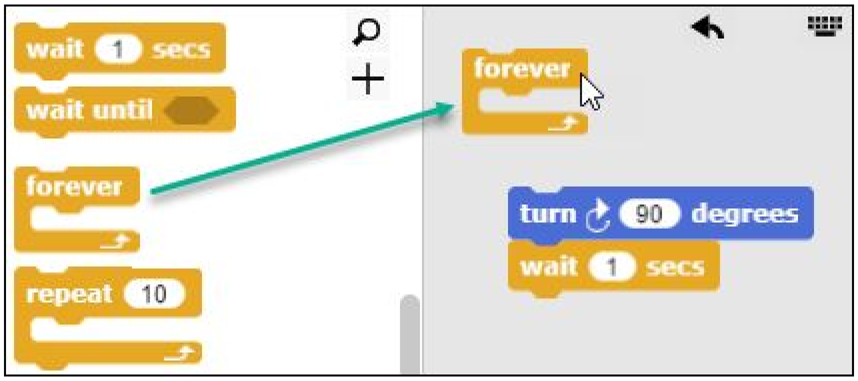
Explore some of the other code blocks in the *Motion Palette*. What does the **Move** code block do?

Topic 1.7 Controlling the Timing of the Turtle's Rotation

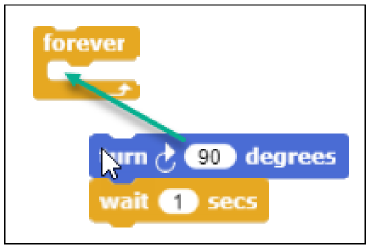
Go the *Control Palette* and drag a **Wait** block onto the script area. Snap it into place beneath the **Turn** block.


Motion
Looks
Sound
Pen
Control
Sensing
Operators
Variables
Sprite
draggable
Scripts
Costumes
Sounds
wait 1 secs
wait until
forever
turn 90 degrees
wait 1 secs

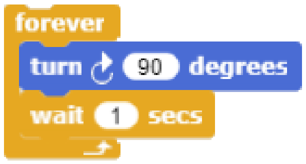
Then drag a **Forever** block from the *Control Palette* into the script area.



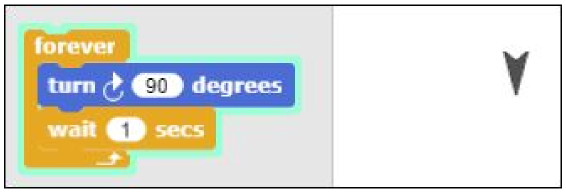
Drag the combined **Turn** and **Wait** blocks into the open space within the C-shaped **Forever** block.



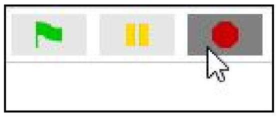
The combined blocks should look like this.



Click the combined group of code blocks. (A group of code blocks that have been snapped together in this way is referred to as a *script*.) When the script is executed, an outline or halo will surround the script to indicate that the program is running. The screen turtle should turn right by 90 degrees one time per second. (**Forever** means that the turtle will continue to turn until the user stops the script.)



Click the red *Stop* button in the top right-hand corner of the screen to stop the script (or click the highlighted script).

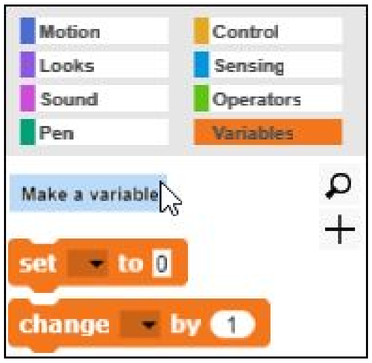


Exploration: Controlling the Timing of the Turtle's Rotation

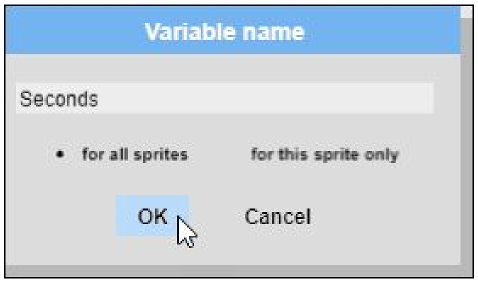
Explore some of the other uses of variables. How could a timer be created that counts down the seconds?

Topic 1.8 Tracking Elapsed Seconds

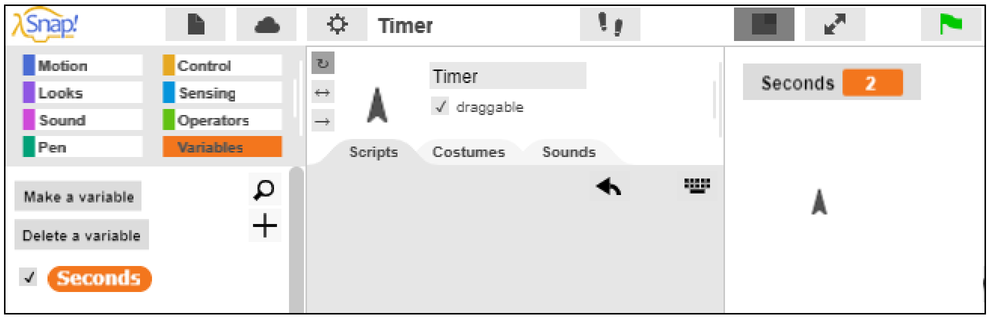
Variables provide a way to keep track of minutes and seconds as the timer counts. To create a *Seconds* variable to track seconds, select the *Variables Palette* and click the *Make a Variable* button.



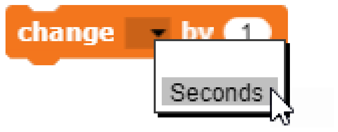
Enter the variable name *Seconds* in the dialog box that appears. (Leave the default option of "for all sprites" selected.) This variable will be used to track elapsed seconds.



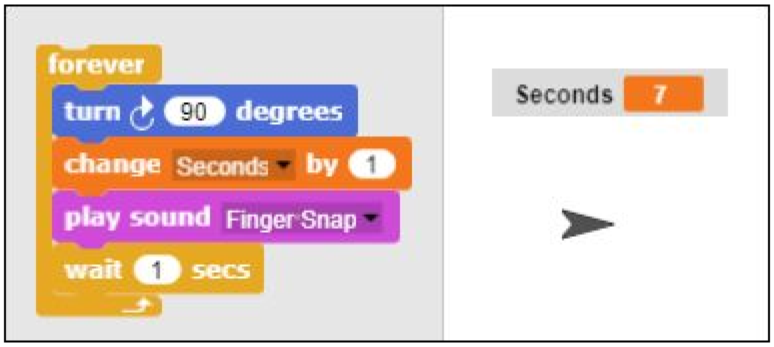
A variable watcher *Seconds* will appear on the stage. All of the variables that have been created are listed at the top of the *Variables Palette.* When the checkbox beside each variable is selected, the *Variable Watcher* will appear on the stage. The variable watcher can be used to monitor the status of the variable.



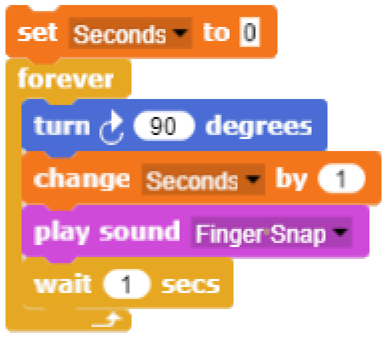
To count seconds, the seconds variable will need to increase the count each time a second elapses. Drag the code block **Change __ by 1** into the script area. Select the variable *Seconds* from its drop-down menu.



Drag the **Change Seconds by 1** block into the **Forever** block. Click the group of blocks to execute the script. The variable *Seconds* will increase by one each time the turtle turns.



The revised script now functions as a timer. The code block "**Set Seconds to** __" can be placed before the **Forever** block to ensure that the timer always begins with a starting value of 0. (This step is called *initialization* because it establishes the initial value of the variable.)

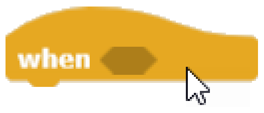


Exploration 1.8 Controlling the Timing of the Turtle's Rotation

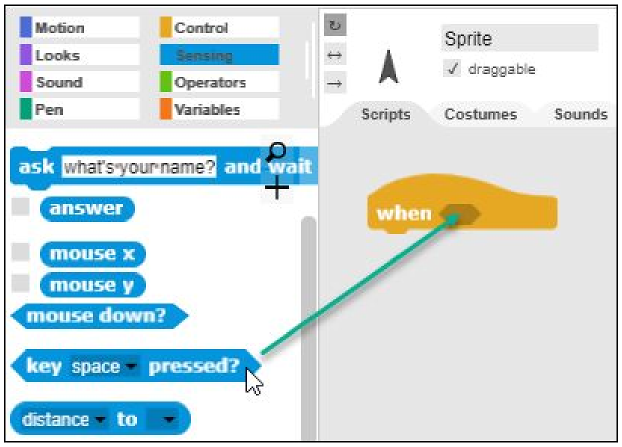
Explore some of the other uses of variables. How could a timer be created that counts down the seconds?

Topic 1.9 Stopping the Timer

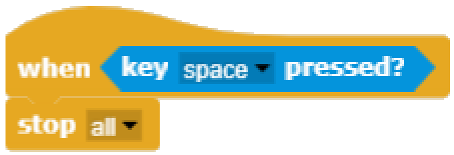
A second script can be added to stop the timer when the space bar is pressed. Begin by dragging the **When** code block from the **Control Palette** into the script area.



Select the *Sensing Palette* and drag the **Key Space Pressed** block into the hexagonal slot in the **When** block.

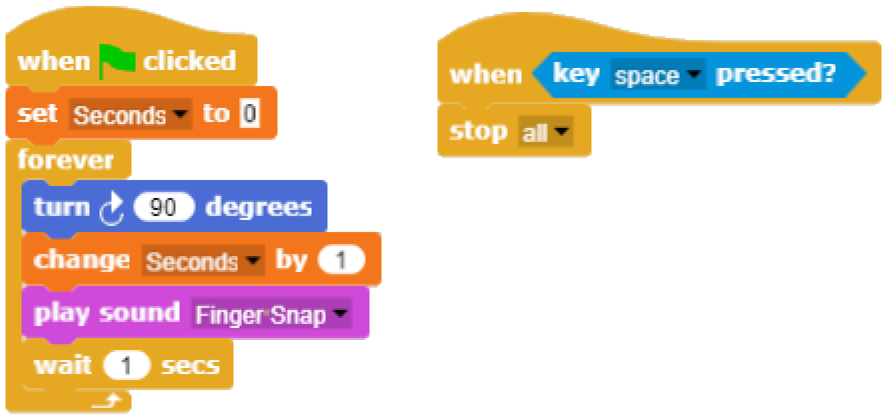


Then add the **Stop All** block from the *Control Palette*.



```
when key space pressed?
stop all
```

Finally, add the **Green Flag** block to the other script. The green flag block will enable the user to start the program by clicking the green flag in the upper right-hand corner of the screen. Pressing the space bar on the keyboard will stop the timer.



```
when clicked
set Seconds to 0
forever
    turn 90 degrees
    change Seconds by 1
    play sound Finger Snap
    wait 1 secs
```

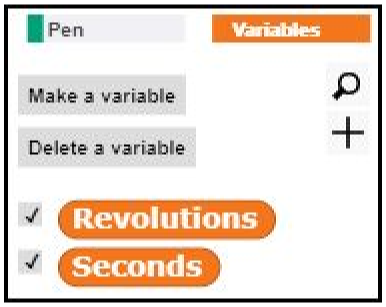
```
when key space pressed?
stop all
```

Exploration 1.9 Stopping the Timer

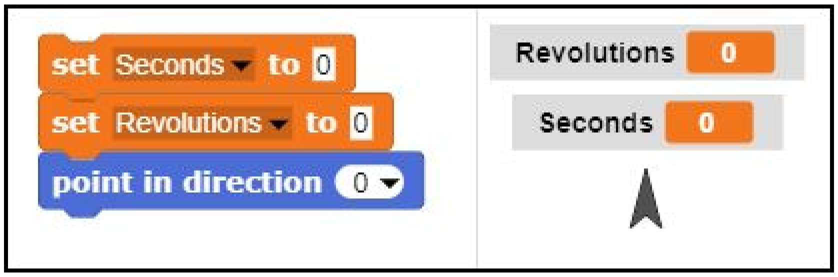
Explore some of the other code blocks in the *Sensing Palette*. What does the **Mouse Down** code block do? How could this code block be incorporated into the timer?

Topic 1.10 Counting Revolutions of a Rotating Turtle

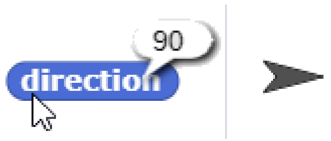
To count the number of revolutions of the turtle that occur within a given time period, add a *Revolutions* counter. Use the *Make a Variable* button in the *Variables Palette* to create a variable named *Revolutions.*



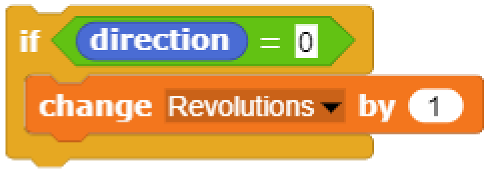
Begin by setting *Seconds* and *Revolutions* to an initial state of *zero*. Point the turtle straight up. This process of establishing the conditions for an initial state is known as *initialization*.


```
set Seconds to 0
set Revolutions to 0
point in direction 0
```

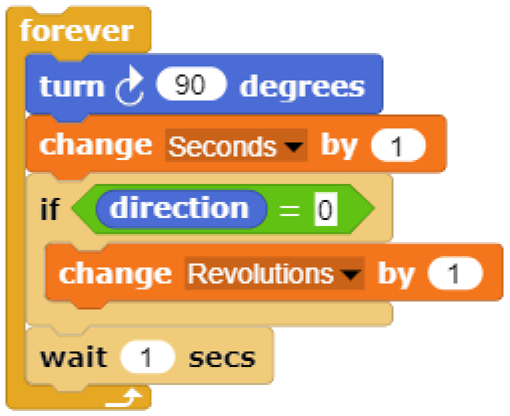
The **Direction** code block reports the direction in which the turtle is pointing. In the illustration below, the turtle is pointed to the right, at an angle of 90 degrees.



The **Direction** block can be combined with an **If** code block to increase the **Revolutions** counter by one each time turtle returns to a heading of 0 degrees after completing a full rotation. (The **Equals** code block used in this example is found under the green *Operators* palette.)

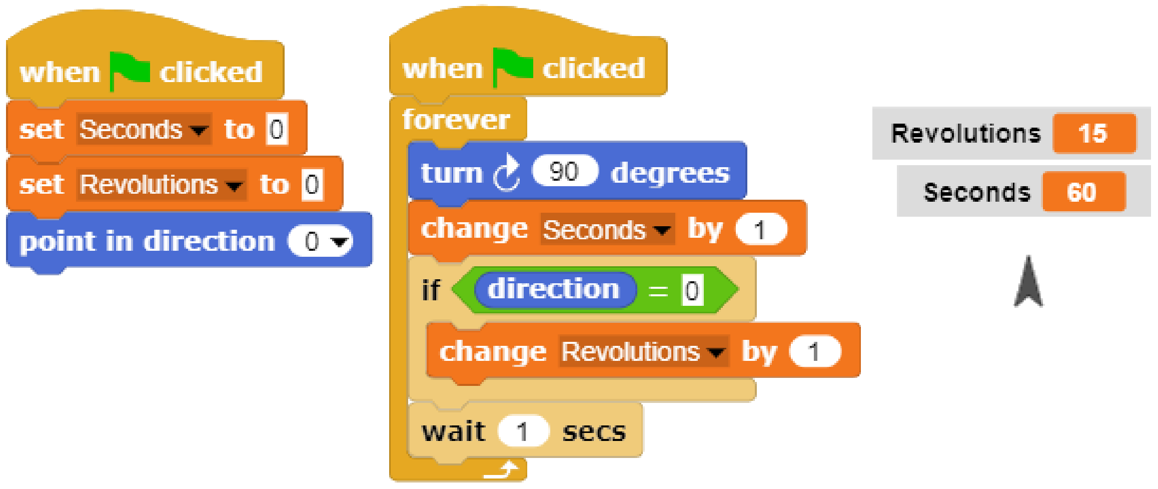

```
if direction = 0
    change Revolutions by 1
```

The **If** code block is placed within the **Forever** code block. Each time the turtle completes a rotation, the *Revolutions* counter is increased by one.



```
forever
  turn ↻ 90 degrees
  change Seconds by 1
  if direction = 0
    change Revolutions by 1
  wait 1 secs
```

The completed procedure initializes Seconds and Revolutions when the green flag is clicked. It also starts the loop that rotates the turtle and counts revolutions and seconds until the space bar is pressed.



```
when 🏳 clicked
set Seconds to 0
set Revolutions to 0
point in direction 0
```

```
when 🏳 clicked
forever
  turn ↻ 90 degrees
  change Seconds by 1
  if direction = 0
    change Revolutions by 1
  wait 1 secs
```

```
Revolutions 15
Seconds 60
```

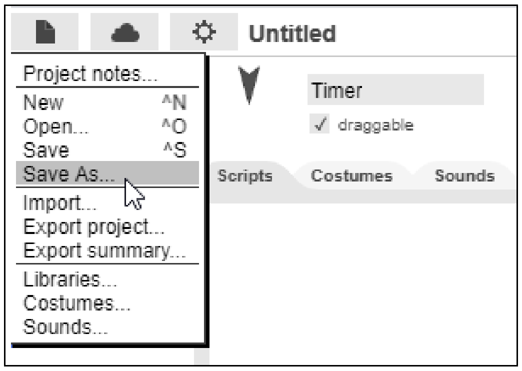
In this illustration, the turtle completed 15 revolutions in 60 seconds. The frequency of rotation, therefore, can be described as *15 revolutions per minute*. Dividing 15 revolutions into 60 seconds yields 4 seconds. The *period* of one rotation, therefore, is 4 seconds.

Exploration 1.10 Counting Revolutions of a Rotating Turtle.

Change the **Wait** time to 0.5 seconds. How many revolutions occur in one minute when this change in timing is made? How does this affect the period of time that it takes each revolution to occur?

Topic 1.11 Saving the Script

This is a good point to save the script. If the *Save* option (under the *File* menu) is used when the script is first saved, it will be given the default name of "Untitled." Since that name is not descriptive, use the *Save As* selection the first time that the script is saved.



This will produce a dialog box that allows the project to be given a name. In this instance, the project has been named, "Timer." The project can be saved on a local computer. The advantage of saving the project in your Snap*!* account in the cloud is that it can be later retrieved from any computer with an Internet connection. (You must be logged into Snap*!* in order to save projects.)

After the project is saved, the project name should replace the default name of "Untitled". After the project is saved and given a name, the *Save* option under the *File* menu can also be used to save future changes and revisions that may be made to the script.