

Learn to Code with Small Basic

Getting Small Basic

- Go to the Windows app store and search for “Small Basic” and click the Install button *or*
- Go to www.smallbasic.com and click the download button



Tips & Tricks

#1 Pro-Tip: BE LAZY!

- Start typing and hit tab or enter to complete the code
- Use the up and down arrow buttons to scroll through the methods
- Look at the explanations on the right-hand side

Lesson 1: The Basics

If you're new to programming, you might hear some new things—we're going to define them here for you!

Today, you're going to write your own computer program. A **program** is a set of instructions that tells a computer what to do. There's a lot of different types of programs, from apps on phones to what makes your computer run!

Another word you might hear is **algorithm**. It sounds complicated, but you probably already know what it means. It's just a process—imagine if you had to teach a robot how to braid hair or put on a coat. How would you break it down into smaller pieces?

We might use the term **object-oriented programming** too. This describes how the code works. An **object** is a package of **operations** and **properties**. For example, an object is like a dog. The breed, size or color of the dog is a property and an operation would be when you tell it to sit, bark or fetch. Small Basic helps you with this by color coding, as you're about to see!

Lesson 2: Hello, Turtle

Let's start with a very simple program! Type the following code in Small Basic:

```
Turtle.Show()
```

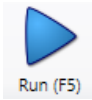


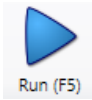
Click the button at the top. What does it do?

Lesson 3: Let's Give Turtle Something to Do

We have a little turtle now, but before we do more with it let's make the background a little less boring. We're going to use an image of a maze, and we're going to put it on the screen at the top left (that's what the 0s will do). Make sure you use " instead of two ' symbols!

```
GraphicsWindow.DrawImage("https://aka.ms/mz1", 0, 0)
```




Press  and see the a maze for the turtle to run through! Let's teleport the turtle to the top left of the window, so we can start at the beginning of the maze. You know how we used 0s to make the image start at the top left corner? We're going to do something similar with the turtle, but we're going to set it down a little bit *away* from the top left corner. Let's use 50 instead of 0. We're going to make sure the turtle is 50 pixels away from the left edge:

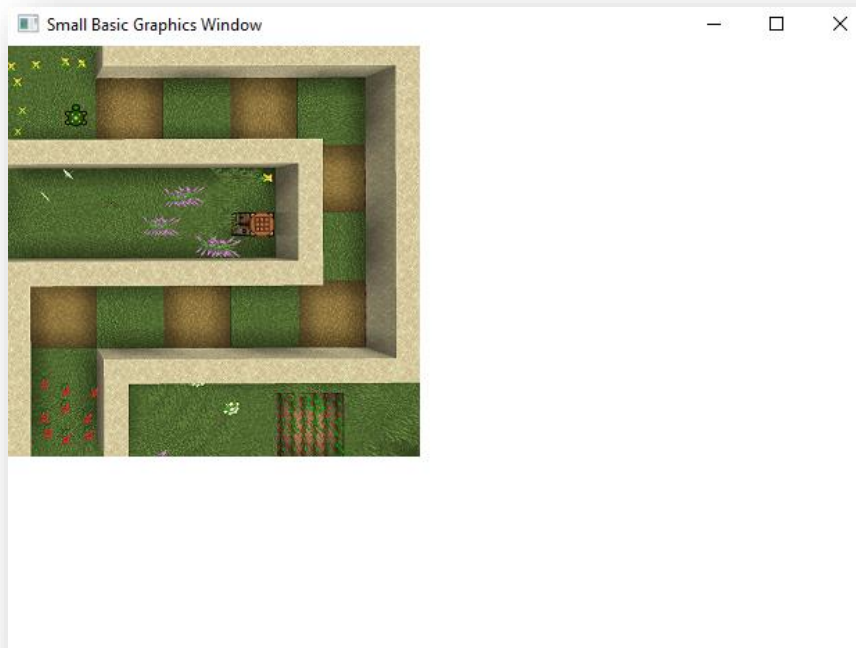
```
Turtle.X = 50
```

and 50 pixels away from the top:

```
Turtle.Y = 50
```



Press  to make sure the turtle is where you want it to be!



Lesson 4: Get Moving!

Let's make that turtle move through the maze! We can use these commands to make the turtle move:

<code>Turtle.Move(100)</code>	This moves the turtle in the direction that it's facing. Change 100 to however far you want the turtle to go!
<code>Turtle.TurnRight()</code>	This makes the turtle turn to the right
<code>Turtle.TurnLeft()</code>	This makes the turtle turn to the left

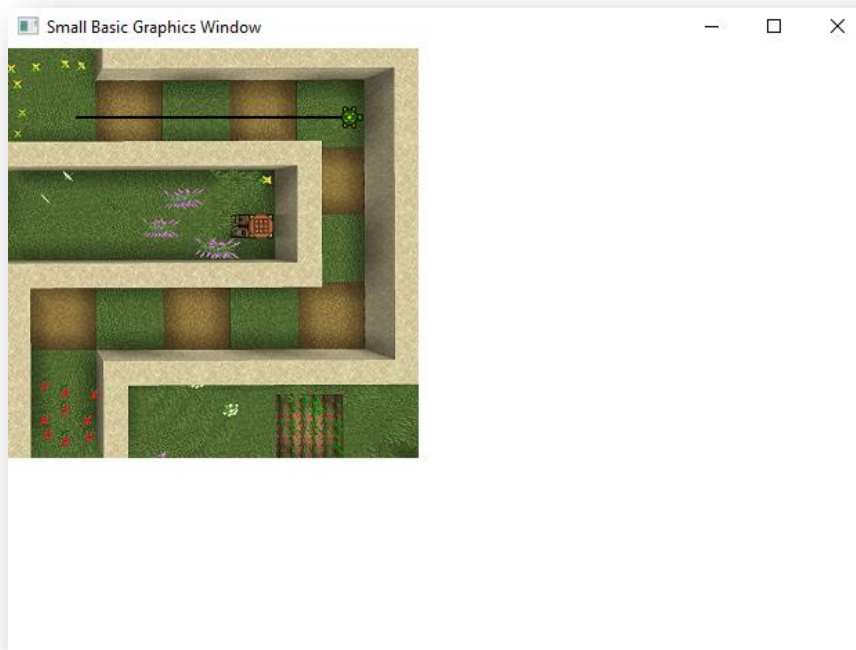
Let's start out by making the turtle turn in the correct direction and start moving! Type the following code in Small Basic:

```
Turtle.TurnRight()  
Turtle.Move(100)
```



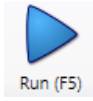
Click the `Run (F5)` button at the top and see what the turtle does.

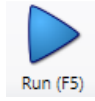
Oh no, it didn't go far enough! Can you change how far the turtle moves until it looks like this?



Great! Now the turtle is ready to go down the next path. What does it have to do now? It has to turn before it can keep going! Add another line of code at the bottom:

```
Turtle.TurnRight()
```



And then run it with the  button to make sure the turtle is facing the correct direction. Keep going- make the turtle move through the whole maze until it reaches the red flowers at the bottom left!

✍️ CHALLENGE: Can you make the turtle do a celebration dance at the end?

Want to do something a little different? Here are some things you can try! Where do you think you would put them in the code?

Change the Color of the Turtle's Trail:

```
GraphicsWindow.PenColor = "Blue"
```

Change Turtle's Trail to a Random Color:

```
GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
```

Change Turtle's Speed:

```
Turtle.Speed = 7
```

Lesson 5: We're Smarter than a Turtle

The turtle made it through the maze, but it took a lot of time to write out the exact path it was going to take. Instead of planning the route ahead of time, why don't we get the turtle to turn or move when we press an arrow key? That way, we can solve all sorts of mazes without extra work!

First, we need to tell the program two things:

- 1) What kind of user input to listen to (do we want to do something when the mouse moves? When a key is pressed? When the mouse clicks?)
- 2) What to do when we get that input

We're going to do something any time a keyboard key is pressed down. Any time that happens, we want to run a **subroutine** (it's like a little program within our bigger program) that will handle what we do with that keyboard key.

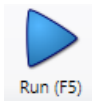
Here's how we do it:

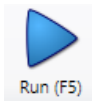
```
GraphicsWindow.KeyDown = WhenKeyPressed
```

```
Sub WhenKeyPressed
EndSub
```

Any code between **Sub** and **EndSub** is our **subroutine**, and it'll run any time a keyboard key is pressed down. It doesn't do anything yet so let's start by making the turtle move forward when we press the "Up" arrow. To do this we want to add some code in the subroutine to check which key was pressed, and then we want to do what the user wants that key to do. So, for example, if the key was the "Up" arrow, we want the turtle to move forward.

```
Sub WhenKeyPressed
  If GraphicsWindow.LastKey = "Up" Then
    Turtle.Move(25)
  EndIf
EndSub
```




Run this with the  button to see if you have it all working! (If your turtle is moving because of the code you wrote from steps 1-4, you can delete that code now).

Great! Any time the "Up" arrow is pressed, the turtle moves forward a bit! Now we should allow the user to make the turtle turn using the left and right arrows. Can you figure out how to do that?

```
Sub WhenKeyPressed
  If GraphicsWindow.LastKey = "Up" Then
    Turtle.Move(25)
  EndIf
  If GraphicsWindow.LastKey = "Right" Then
    What do we put here?
  EndIf
  If GraphicsWindow.LastKey = "Left" Then
    What do we put here?
  EndIf
EndSub
```



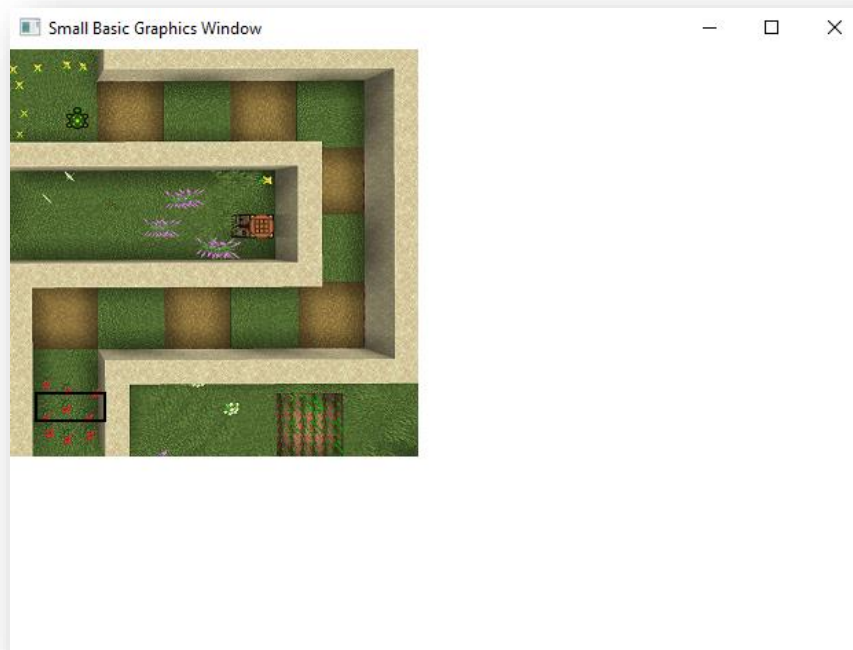
Once you have that completed, press the  button again and finish the maze yourself!

✂ CHALLENGE: Write your name with the turtle!

Lesson 6: Winner, Winner!

Now we can easily guide the turtle to the end, but wouldn't it be nice if our hard work was rewarded? Small Basic will allow us to draw a box on the screen and then we can detect if the turtle is inside it. We can use that to detect when we've "won" the maze!

First, let's draw a rectangle. We want it to be at $x=20$, $y=240$, and we want the height to be 40 and width to be 50. How do we do this? Use the auto-complete to figure out what method on `GraphicsWindow` to use, and then read the right-side panel to put the **parameters** (the numbers inside the parenthesis) in the correct order. After you're done, the screen should look like this:



🔪 CHALLENGE: Make the finish box a different color!

That black box is our finish! Now we need to check if the turtle is ever inside it. The top left corner of the box is at $x=20$, $y=250$ so we can figure out where the other corners are by adding the height or width to that. Here's what that code would look like. Where would you put this?

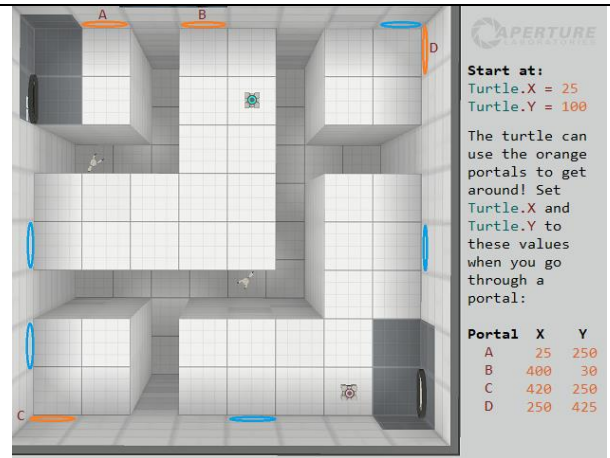
```
If Turtle.X > 20 And Turtle.X < 20+50 And Turtle.Y > 240 And Turtle.Y < 240+40 Then  
    GraphicsWindow.ShowMessage("You won!", "Congrats!")  
EndIf
```


Lesson 7: Challenge Time

Now that you beat that maze, why not try something a little harder? Or make your own!



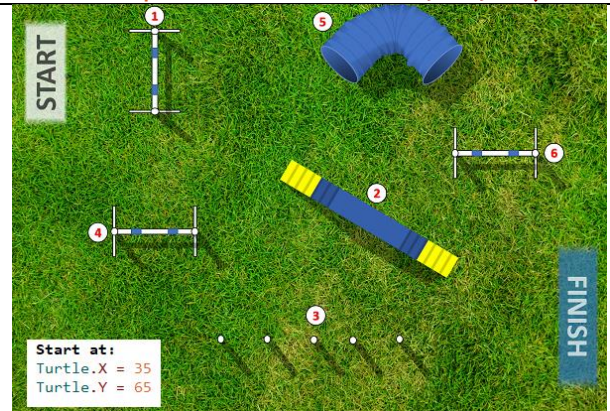
```
GraphicsWindow.DrawImage(
  "https://aka.ms/mz2", 0, 0)
```



```
GraphicsWindow.DrawImage(
  "https://aka.ms/mz3", 0, 0)
```




```
GraphicsWindow.DrawImage(
  "https://aka.ms/mz4", 0, 0)
```

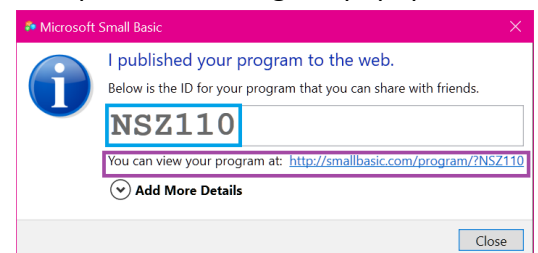


```
GraphicsWindow.DrawImage(
  "https://aka.ms/mz5", 0, 0)
```

Lesson 8: Share your code!


In Small Basic, you can export a program so that other people with Small Basic can look at the code, run it and edit it. Simply click the  Publish button in the top menu. You'll get a popup that looks like this!

The special code in blue is what you can share with your friends who also use Small Basic. The link in purple is what you can share with friends and family who don't write code; you can email or text it to them!



Lesson 9: Learn from others!


In Small Basic, you can import someone else's code to work on it and fix a problem, make it better, or just customize it for fun! It's also a good way to learn how others write code and learn from their examples. We're going to import in a Tetris game and then play with the code.

Click  and type in **"tetris"** (all lowercase). The Tetris code appears!

Let's run it and see what it looks like! The instructions are on the right. The Left and Right arrows move it, Up rotates, and Down drops your piece. Do you see the title that says "Small Basic Tetris"? Let's change that!

- Scroll up to the top of your code. On line 17 we see:
`GraphicsWindow.Title = "Small Basic Tetris"`
- Let's add more to the title:
`GraphicsWindow.Title = "Small Basic Tetris - Small Basic Rocks!"`

Now, run the code to see your changes!

For some fun work at home, import Asteroids! Click  and type in **"asteroids"**. Play it!

✈ CHALLENGE: What changes can you make to Tetris and Asteroids?