Chapter 32

FUNCTIONS

Functions are blocks of code that you can use to build a bigger program. You can add more premade functions by importing **modules**.

A kind of library

The words **function** and **procedure** mean almost the same thing. When talking about Python, we refer to function (some people might say "procedure").

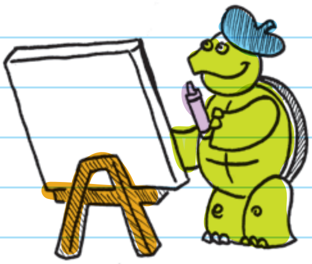
TURTLE GRAPHICS

The **TURTLE MODULE** is filled with functions that allow you to treat the screen like a drawing board and use the turtle icon to draw on it.

A **library** is a collection of prewritten functions and code that can be imported into a project.

Importing the Module

You can import the turtle module using the **import** command. Start with the word "from," then enter the module name, and then enter "import," which means you want to import from the named module (in this case, "turtle").



You can name which specific functions you want to import from a module. To import all the functions, end the import command with "*".

Import command key word

```
from turtle import *
```

Means "all"

Module name

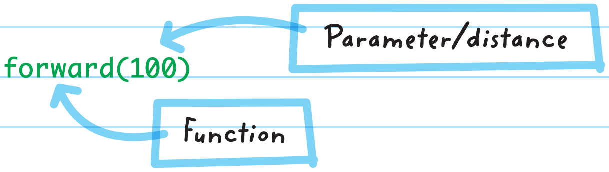
This means import all the functions from the turtle module.

Moving the Turtle

After importing the turtle module, you can use its functions to move the turtle around the screen. The turtle starts out facing to the right. You can make the turtle move using the forward and backward functions.

The default "turtle" is actually just a triangle that points in the direction it's facing.

FOR EXAMPLE, you can make the turtle move forward 100 **PIXELS** on the screen with the command:

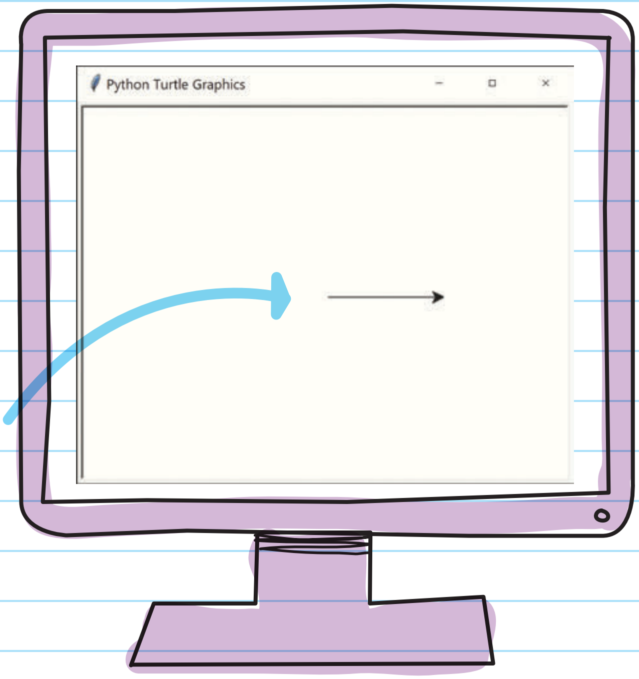


```
forward(100)
```

This is what the code looks like:

```
from turtle import *
forward(100)
```

The command will open a new window and draw a line from left to right that's 100 pixels long (the triangle at the end of the line is the turtle):



PIXEL

A **pixel** (from **picture element**) is a point of light on a screen. The small points make up images on a computer display.

The **left**( ) and **right**( ) functions turn the turtle left or right based on the direction it is facing on the screen. Both of the functions require a parameter for the number of degrees they turn.

FOR EXAMPLE, to turn 90 degrees to the left, use:

Parameter/degree

```
left(90)
```

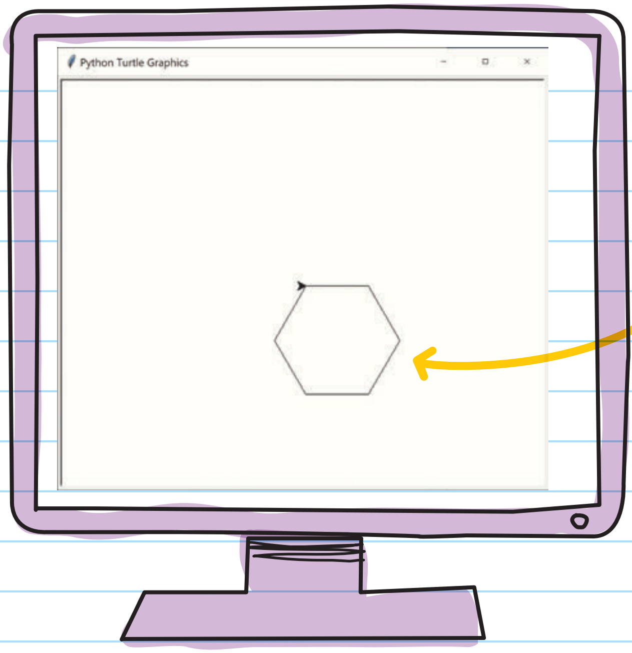
Function

You can use the turtle functions along with other parts of Python.

FOR EXAMPLE, you could use loops to make some interesting art. A *for* loop that repeats 6 times along with **forward** and **right** functions from the turtle module can be used to draw a hexagon:

```
from turtle import *
for j in range(6):
        forward(70)
        right(60)
```

This directive tells the turtle to repeat the following code 6 times: move forward 70 pixels, then turn right 60 degrees.

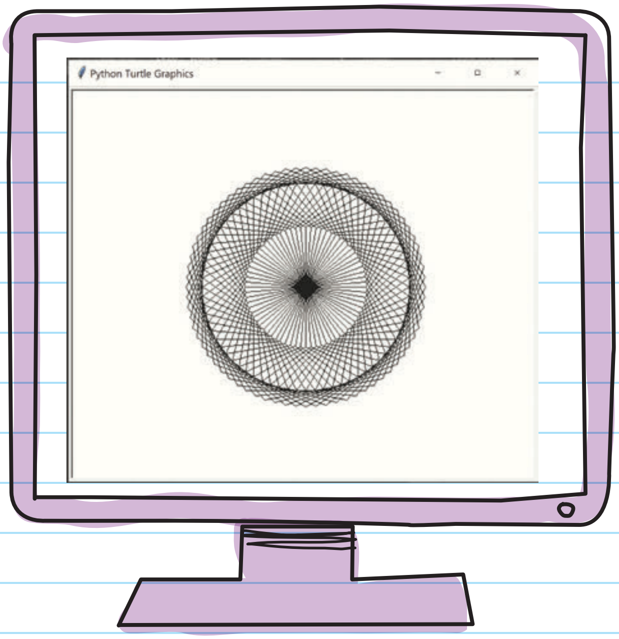


You can also nest the hexagon loop inside another loop that rotates by 5 degrees after each hexagon is drawn.

By repeating the outer loop 72 times, you can draw 72 hexagons, each 5 degrees rotated to the right.

```
from turtle import *
for i in range(72)
      for j in range(6):
                forward(70)
                right(60)
      right(5)
```

Here's what the drawing looks like after running the program:



ADDITIONAL TURTLE ART FUNCTIONS

There are many more functions in the turtle module that can be used to create fun art.

FUNCTION	DESCRIPTION
forward()	Moves the turtle forward; parameter is number of pixels to move
backward()	Moves the turtle backward; parameter is number of pixels to move
right()	Turns the turtle clockwise; parameter is number of degrees to turn
left()	Turns the turtle counterclockwise; parameter is number of degrees to turn
penup()	Picks up the turtle's pen (ends tracing the movements of the turtle); no parameters

FUNCTION	DESCRIPTION
pendown( )	Puts down the turtle's pen (begins to trace the movements of the turtle); no parameters
pencolor( )	Changes the color of the turtle's pen; parameters are named colors
heading( )	Returns the current heading—useful if you need to know which way the turtle is facing; no parameters
position( )	Returns the current *x* and *y* position—useful if you need to know where on the screen the turtle is; no parameters
goto( )	Moves the turtle to a specific position; parameters are *x*-axis and *y*-axis coordinates
fillcolor( )	Changes the color the turtle will use to fill a polygon; parameter types are the same as pencolor( )
begin_fill( )	Remembers the starting point for a filled polygon—used before a shape you want to fill is drawn; no parameters
end_fill( )	Closes the polygon and fills with the current fill color—used after a shape you want to fill has been drawn; no parameters
dot( )	Draws a dot at the current position; no parameters
stamp( )	Stamps the image of the turtle shape on the screen wherever the turtle is; no parameters
shape( )	Changes the shape of the turtle; parameters are "arrow," "classic," "turtle," "circle," or "square"

Even though some functions don't use parameters, you still need to include the parentheses ( ).

FUNCTIONS

If you're using the same piece of code over and over in Python, then it's best to make your own function. When you make a new function, it's called **DEFINING A FUNCTION**.

To define a function, use this format:

Start with "def" for "define"

```
def functionName():
    print("hello")
```

Name the function whatever you want (using the same rules as naming variables), add parentheses, end with colon

Indent and add your code

To **CALL A FUNCTION** is to use a function that is already defined. When you call a function, Python finds the function definition and runs the code found in the function body.

You can only call a function after it has been defined. To call a function, use this format:

FUNCTION NAME

```
functionName()
```

PARENTHESES

I AM GOING TO CALL YOU THOR.
COME, THOR! COME WHEN I CALL YOU!

Whenever you call a function, the program jumps back to where the function was defined, runs all the code in the body of the function, then goes back to where the program left off when the function was called.

The **body** of the function is the indented part. The body is the code that will run when the function is called.

```
def functionName():
    ●●● ●● ●●●
    ●●● ●● ●●●

●●● ●● ●●●
●●● ●● ●●●
functionName()
●●● ●● ●●●
●●● ●● ●●●
```

Here's an example of a function that says "Hello, World!":

Defines function

```
def hello():
    print("Hello, World!")
hello()
```

Body of function, code that will run

Calling the hello function

This will print: `Hello, World!`

PARAMETERS AND RETURN VALUES

Parameters and variables are similar because they are both used to store information. However, a **PARAMETER** is different because it cannot be used outside a function. A parameter is data that is provided as input from a user. Functions can use parameters only within the body of the function. That's because a parameter in a function is only recognized within the function itself.

FOR EXAMPLE, you can create a function that converts meters to feet. Inside the convert function, you can multiply the meters parameter by 3.281 (the number of feet in 1 meter) to get how many feet are in the specified number of meters.

Function definition

Parameter name

```
def convert(meters):
    feet = meters * 3.281
    return feet
```

Use the parameter like a variable.

Call the function and include the value of the parameter, in this case 1.

```
convert(1)
```

If you tried to use the meters parameter outside of the function definition, you'd get an error:

```
def convert(meters):
    feet = meters * 3.281
    print(feet)
```

Function definition ends here.

```
convert(3)
print(meters)
```

"Meters" used outside the function definition results in an error.

```
NameError: name "meters" is not defined
```

The **RETURN VALUE** of a function is the information that you can pass from the function back out to the main program. This is the function's output. To pass information out of the function, write "return" followed by the output data.

Return value = function output

```
def convert (meters):
    feet = meters * 3.281
    return feet
```

This will return the value of feet each time the function runs.

With the convert function defined, you could call the function many times and get a different return value each time.

convert(5)	returns 16.405 (5 * 3.281)
convert(234)	returns 767.754 (234 * 3.281)
convert(5,234)	returns 17,172.754 (5.234 * 3.281).

To print the return value, use the **print( )** function with the function call inside it:

$$\texttt{print(}\underbrace{\texttt{convert(1)}}_{\text{Function call}}\texttt{)}$$

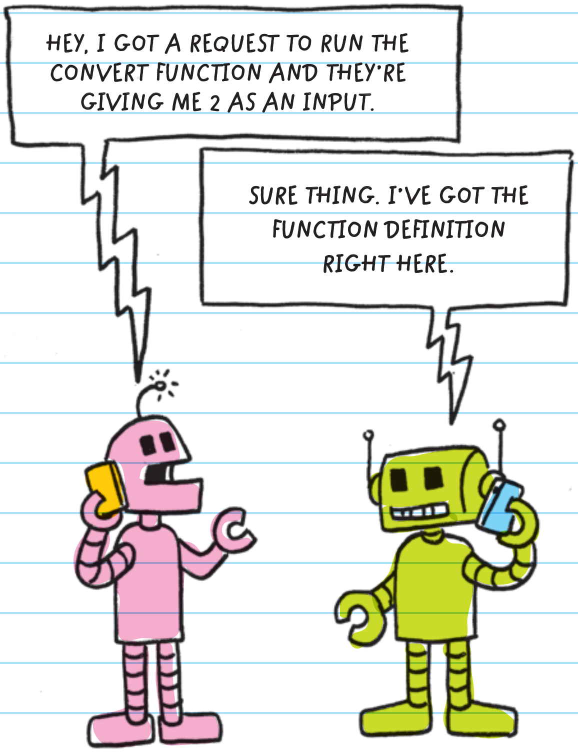
The program will print: `3.281`

You can also add text around the function call to give an explanation of the code, like this:

```
print("3 meters = ", convert(3), "feet")
```

The program will print:

```
3 meters =  9.843 feet
```


HEY, I GOT A REQUEST TO RUN THE CONVERT FUNCTION AND THEY'RE GIVING ME 2 AS AN INPUT.
SURE THING. I'VE GOT THE FUNCTION DEFINITION RIGHT HERE.

You can repeat a small chunk of code while using a parameter to slightly change it for each use. For example, in a word game program, instead of typing "Please enter a noun" or "Please enter an adjective" repeatedly, you could make a function to do it for you:

Creates a new variable called "words" and assigns it a blank list

Defines a new function called "prompt"

```
words = []

def prompt (kind):

    ask = "Please enter " + kind + ": "

    word = input(ask)

    words.append(word)
```

The kind of word the user will be asked to enter

The user is asked to enter a type of word.

The variable "word" is assigned the value of what the user enters.

The value for "word" is added to the list called "words".

```
prompt("an adjective")
prompt("a nationality")
prompt("a person")
prompt("a plural noun")
prompt("an adjective")
prompt("a plural noun")
```

The type of word the user will enter

The "prompt" function can be called many times, asking the user for different types of words by changing the parameters for each function call.

The word list, created by the user's input, is inserted into the word game:

```
print("Computers were invented by a", words[0], words[1],
"engineer named", words[2], ". To make a computer, you need
to take a lot of", words[3], ", melt them down, and make",
words[4], words[5], ".")
```

The output for this program is a completed story that uses the user's words to fill in key details.

CHECK YOUR KNOWLEDGE

1. How do you import all the functions in the turtle module into a Python program?

2. Write out the two lines of code you would need to move the turtle 45 pixels and turn right 30 degrees (assuming you've already imported the turtle module).

3. What does it mean to call a function? What code would you write to call a function called "BopIt" with no input information?

4. When you create a new function, you need to ______ it.

5. You can pass information into a function using ______.

6. You can pass information out of a function to the main program by using ______.

7. Explain what's wrong with the program below:

```
def distance (laps):
    meters = laps * 100
    return meters
print(laps)
```

8. In order to use a function after you've defined it, you need to ________ it. This is done with ________.

9. Write the return value for each of the following functions:

NAME	CODE	RETURNS

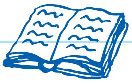
A.	`name = "Max"`
	`def hello_you(person):`
	`    sentence = "Hello " + person`
	`    return sentence`
	
	`hello_you("Max")`	

B.	`def plotter(x, y):`
	`    instructions = "Plot a course through " + str(x) + " and " + str(y)`
	`    return instructions`
	
	`plotter(3, 5)`	

MORE QUESTIONS

NAME	CODE	RETURNS
C.	`def absolute_value(num):` `    if num >= 0:` `        return num` `    else:` `        return num * -1` `absolute_value(-4)`	
D.	`def favorite(category, thing):` `    sentence = "My favorite " +` `    category + " is the " + thing` `    return sentence` `favorite("snake", "Python")`	

CHECK YOUR ANSWERS



1. From turtle import *

2. forward(45)
   right(30)

3. When you call a function, you are using the name of the function to tell the program to run the code that corresponds to that function name.

   The code would look like: BopIt()

4. Define

5. Parameters

6. Return

7. The "laps" parameter is used outside the function definition.

8. call, parentheses

9.

A.	Hello Max
B.	Plot a course through 3 and 5
C.	4
D.	My favorite snake is the Python