Type this in the new window:

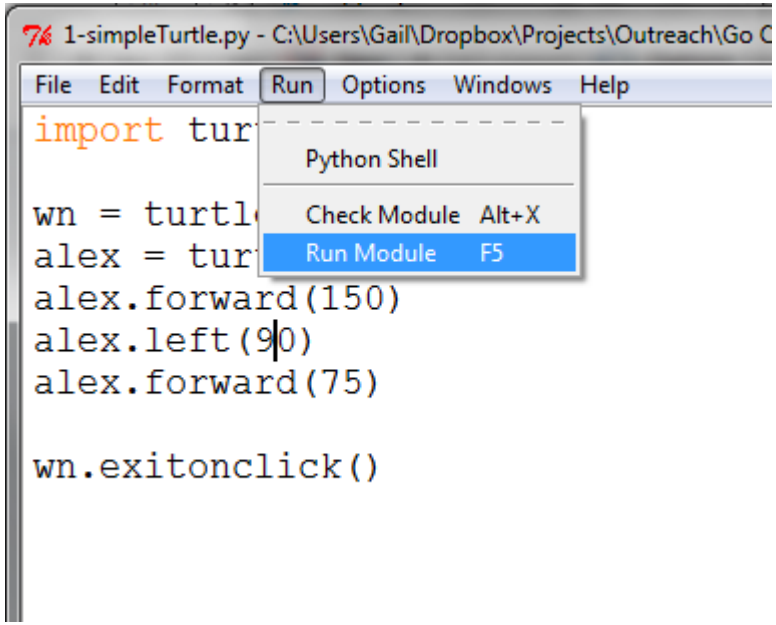
```
import turtle

wn = turtle.Screen()

alex = turtle.Turtle()
alex.forward(150)
alex.left(90)
alex.forward(75)

wn.exitonclick()
```

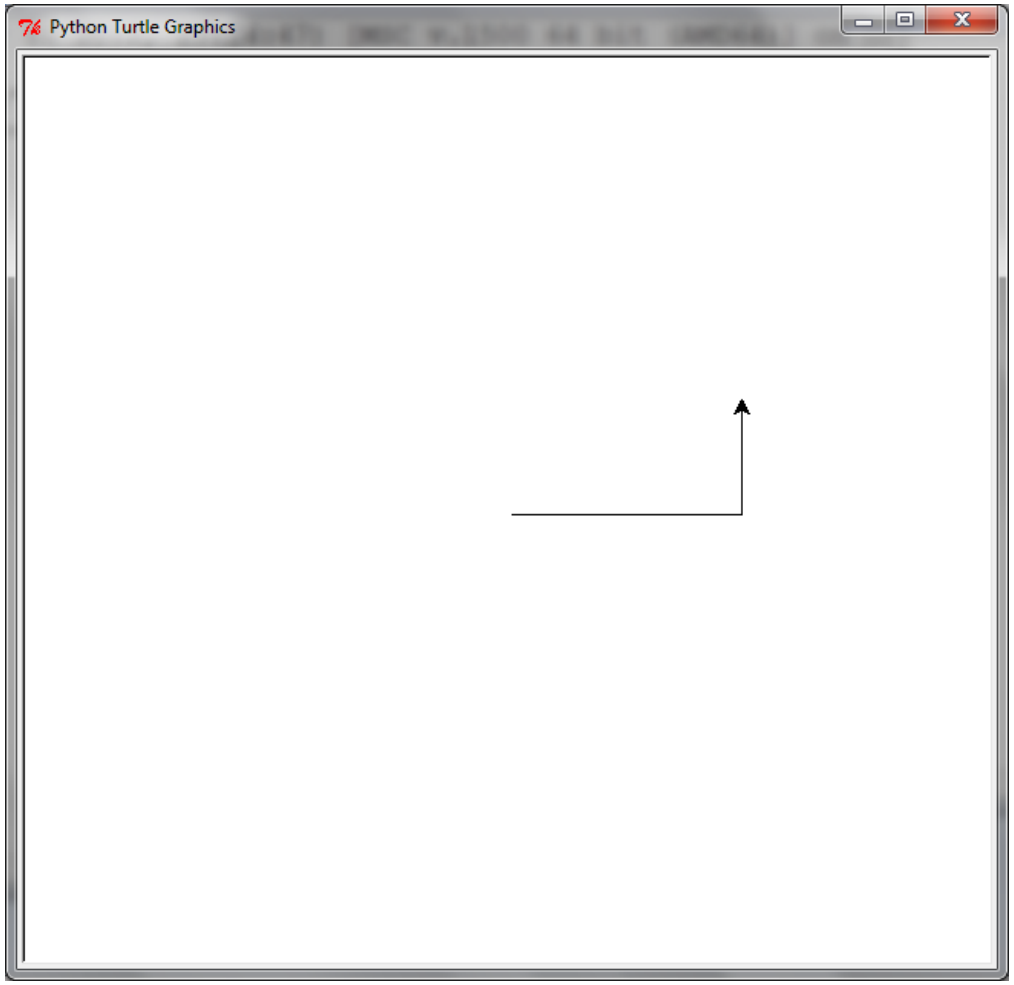
Save, Save As

Note: make sure you add `.py` to the end of your file!

(And don't name the file turtle.py)

Run, Run Module




Python Turtle Graphics

Woah you just ran your first script!

```
import turtle

wn = turtle.Screen()

alex = turtle.Turtle()
alex.forward(150)
alex.left(90)
alex.forward(75)

wn.exitonclick()
```

Tell Python you want to use Turtle Graphics in your program

Create a new window to draw with the turtle on; refer to the window from now on as `wn`

```
import turtle

wn = turtle.Screen()

alex = turtle.Turtle()
alex.forward(150)
alex.left(90)
alex.forward(75)

wn.exitonclick()
```

```
import turtle

wn = turtle.Scr

alex = turtle.Turtle()
alex.forward(150)
alex.left(90)
alex.forward(75)

wn.exitonclick()
```

Ask Turtle Graphics to create a new `Turtle` to draw with; call it `alex`

```
import turtle

wn = turtle.Screen()

alex = turtle.Turtl
alex.forward(150)
alex.left(90)
alex.forward(75)

wn.exitonclick()
```

Ask alex to go forward, turn left, and go forward again, drawing while she moves

```
import turtle

wn = turtle.Screen()

alex = turtle.Turtle()
alex.forward(150)
alex.left(90)
alex.forward(75)

wn.exitonclick()
```

Tell the program to exit when someone clicks on the window we named `wn`

Try changing the numbers in `alex`'s movement code, or even add new movements.

Can you get `alex` to draw a square?

How about a pentagon?

REPETITION

One way to draw a pentagon...

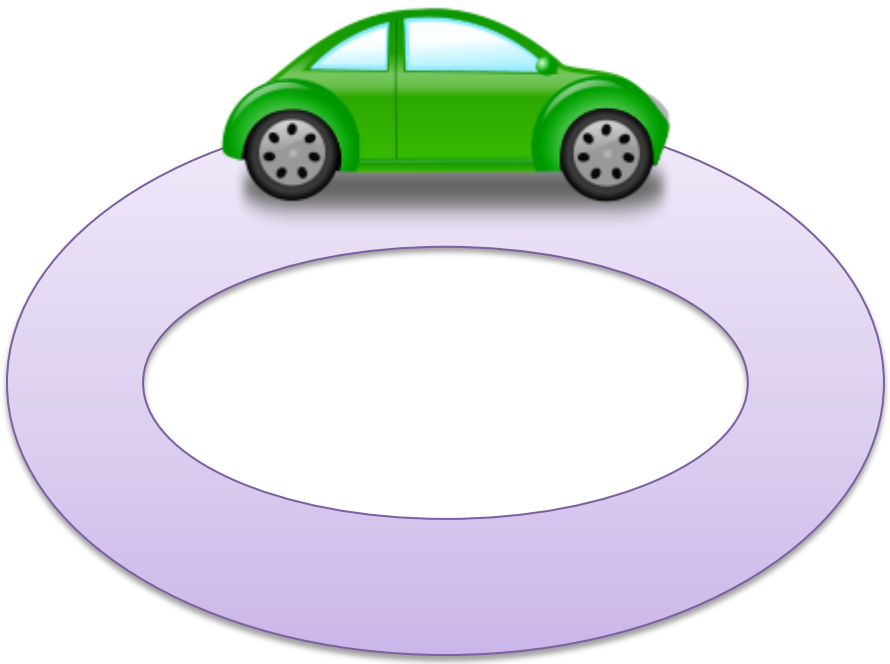
```
alex.forward(100)
alex.left(72)
alex.forward(100)
alex.left(72)
alex.forward(100)
alex.left(72)
alex.forward(100)
alex.left(72)
alex.forward(100)
alex.left(72)
```

One way to draw a pentagon...

```
alex.forward(100)
alex.left(72)
alex.forward(100)
alex.left(72)
alex.forward(100)
alex.left(72)
```

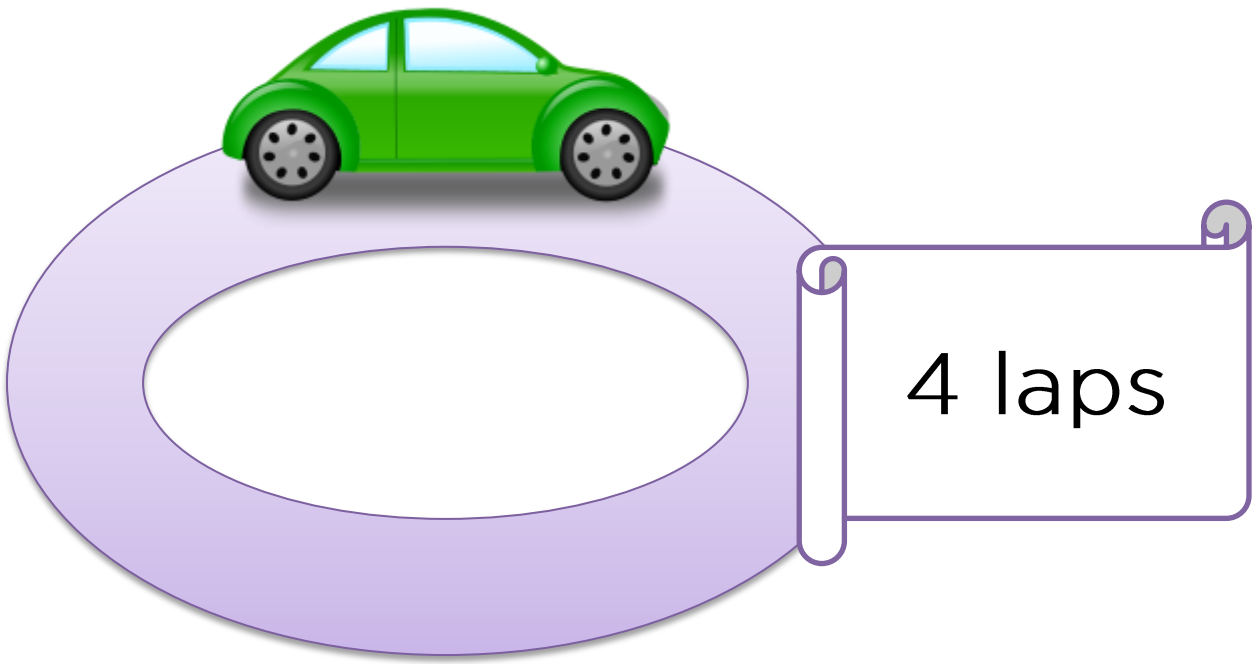
Can we avoid writing the same lines of code over and over?

Loops



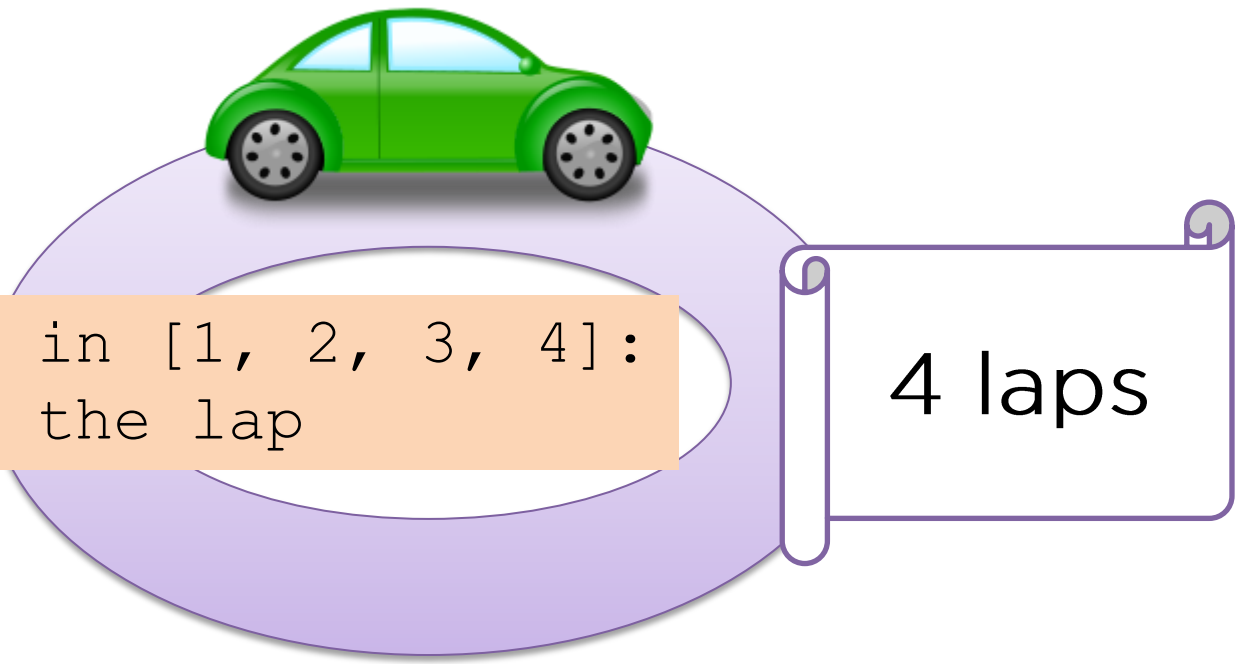
Drive the same track multiple times

for loop



Drive the same track exactly four times

for loop




```
for lapNum in [1, 2, 3, 4]:
    # drive the lap
```

Drive the same track exactly four times

Using a for loop to draw a pentagon

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

Using a for loop to draw a agon

This gives a name to the lap numbers as we "drive" around (first it will be 1, then 2, ...)

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

Using a for loop to draw a pen

This is a list representing the lap numbers.

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

Using a for loop to draw a pentag

The colon says we're ready to specify how to drive each lap

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

Using a for loop to draw a pentagon

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

We use indentation to show what code belongs inside the for loop

Using a for loop to draw a pentagon

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

Using a for loop to draw a pentagon

```
for sideNum in [1, 2, 3, 4, 5]:
    alex.forward(100)
    alex.left(72)
```

This is the code that will run each lap (5 times in this case)

Shortcut: range

```
for sideNum in range(5):
    alex.forward(100)
    alex.left(72)
```

Shortcut: range

This produces the list [0,1,2,3,4]

```
for sideNum in range(5):
    alex.forward(100)
    alex.left(72)
```

Shortcut: range

```
for sideNum in range(5):
    alex.forward(100)
    alex.left(72)
```

Important:
We still have 5 laps, we're just counting them from 0 instead of 1

Try drawing a hexagon instead!

How many lines of code did you have to change?

What other cool shapes or designs can you make?

Google “Turtle 3.4”

PlanetB | Syntax Highlight | www.planetb.ca/projects/ | 24.1. turtle — Turtle graph

Python Software Foundation [US] https://docs.python.org/3/library/turtle.html

Apps | GDI Links | BEFORE 5/27/14 | Cool Mac Stuff | Introduction to Ther | Building a Fire Pit: Ti | install me please!

Python » 3.4.2 Documentation » The Python Standard Library » 24. Program Frameworks » previous | next | modules | index

Table Of Contents

24.1. turtle — Turtle graphics

24.1.1. Introduction

Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966.

Imagine a robotic turtle starting at (0, 0) in the x-y plane. After an `import turtle`, give it the command `turtle.forward(15)`, and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves. Give it the command `turtle.right(25)`, and it rotates in-place 25 degrees clockwise.

By combining together these and similar commands, intricate shapes and pictures can easily be drawn.

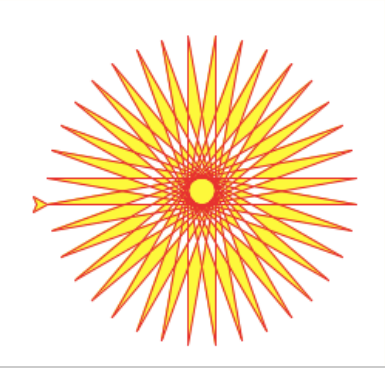
The `turtle` module is an extended reimplementation of the same-named module from the Python standard distribution up to version Python 2.5.

It tries to keep the merits of the old turtle module and to be (nearly) 100% compatible with it. This means in the first place to enable the learning programmer to use all the commands, classes and methods interactively when using the module from within IDLE run with the `-n` switch.

The turtle module provides turtle graphics primitives, in both object-oriented and procedure-oriented ways. Because it uses `tkinter` for the underlying graphics, it needs a version of Python

Turtle star

Turtle can draw intricate shapes using programs that repeat simple moves.



Okay– I'm cutting you all loose in the API for 10 minutes!

See how many new functions you can teach yourself.

If you get a function to work and you see that your neighbor is paused, share the function you found!

Try these commands – experiment and see what designs you can make!

```
alex.shape("turtle")
alex.reset()
alex.shape("square")
alex.penup()
alex.pendown()
alex.up()
alex.backward(someNumber)
alex.color("red")
alex.pensize(someNumber)
alex.stamp()
alex.circle(someNumber)
```

Type this code and see what it does...

```
for aColor in ["red", "blue", "yellow",
               "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Using a color variable in a loop

This variable will change every lap

```
for aColor in ["red", "blue", "yellow",
               "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Using a color variable in a loop

Instead of referring to a lap with a number, this time we'll use a color

```
for aColor in ["red", "blue", "yellow",
               "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Using a color variable in a loop

The for loop will have 5 laps since we have to go through each color one at a time

```
for aColor in ["red", "blue", "yellow",
               "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Using a color variable in a loop

A word in quotes is called a string – it is just text, not a variable

```
for aColor in ["red", "blue", "yellow",
               "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Using a color variable in a loop

```
for aColor in ["red", "blue", "yellow",
               "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Since the value in the `aColor` box changes each lap, we set a new color to draw with each time

Defining our own routine

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

Defining our own routine

Indicates we want to start our routine (aka “function”) definition

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

Defining our own routine

Our routine will be called
`drawSquare`

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

Defining our own routine

Routine names are followed by brackets

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

Defining our own routine

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

Indentation indicates what code to run when we run the routine (i.e. “call the function”)

Defining our own routine

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

This code will never run until we ask it to

Running the routine

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)

drawSquare()
```

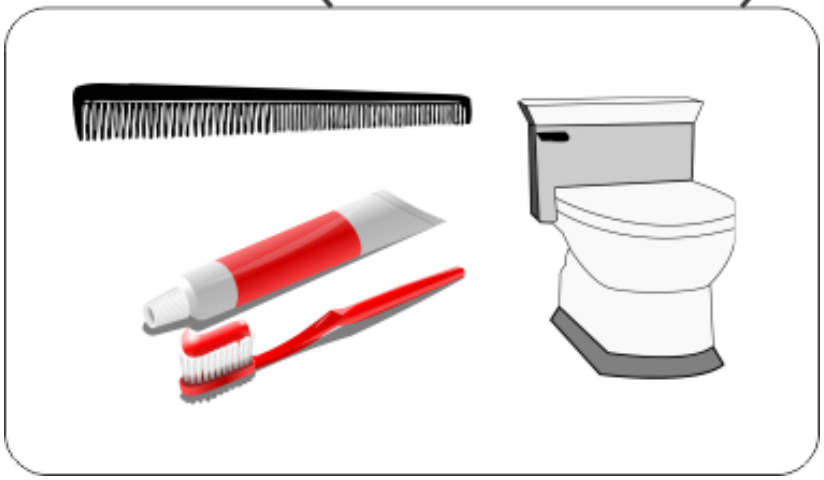
Running the routine

```
def drawSquare():
    alex.penup()
    alex.goto(50, 50)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)
```

```
drawSquare()
```

Run the routine (i.e. "call the function")

routine(doThisFirst)



Customizing the routine

```
def drawSquare(x, y):
    alex.penup()
    alex.goto(x, y)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)

drawSquare(50, 50)
drawSquare(200, 200)
```

Customizing the routine

Routine parameters: variables that will be filled when we run the routine

```
def drawSquare(x, y):
    alex.penup()
    alex.goto(x, y)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)

drawSquare(50, 50)
drawSquare(200, 200)
```

Customizing the routine

Parameters can be used inside the routine as if they were regular variables

```
def drawSq
    alex.p
    alex.goto(x, y)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)

drawSquare(50, 50)
drawSquare(200, 200)
```

Customizing the routine

```
def drawSquare(x, y):
    alex.penup()
    alex.goto(x, y)
    alex.pendown()
    for side in range(4):
        (50)
        0)

drawSquare(50, 50)
drawSquare(200, 200)
```

The `drawSquare` routine can now be called with specific values for the parameters

Customizing the routine

```
def drawSquare(x, y):
    alex.penup()
    alex.goto(x, y)
    alex.pendown()
    for side in range(4):
        alex.forward(50)
        alex.right(90)

drawSquare(50, 50)
drawSquare(200, 200)
```

When called a second time, all new values are used for the parameters

Let’s change it.

```
def drawSquare(x=100, y=200):
    Alex.penup()
    Alex.goto(x, y)
    Alex.pendown()
    for side in range(4):
        Alex.forward(50)
        Alex.right(90)
```

What changed inside of the method body?

[not including the last two lines drawSquare() and drawSquare(400, 400)]

New

```
def drawSquare(x=100, y=200):
    Alex.penup()
    Alex.goto(x, y)
    Alex.pendown()
    for side in range(4):
        Alex.forward(50)
        Alex.right(90)

drawSquare()
drawSquare(400, 400)
```

Old

```
def drawSquare(x, y):
    Alex.penup()
    Alex.goto(x, y)
    Alex.pendown()
    for side in range(4):
        Alex.forward(50)
        Alex.right(90)

drawSquare()
drawSquare(400, 400)
```

Old

Let's give our arguments some default values.

```
def drawSquare(x, y):
    Alex.penup()
    Alex.goto(x, y)
    Alex.pendown()
    for side in range(4):
        Alex.forward(50)
        Alex.right(90)
```

Which command works with the unchanged code above?

```
drawSquare()
drawSquare(400, 400)
```

New

```
def drawSquare(x=100, y=200):
    Alex.penup()
    Alex.goto(x, y)
    Alex.pendown()
    for side in range(4):
        Alex.forward(50)
        Alex.right(90)
```

Let's give our arguments some default values. This is the only change we need to make.

```
drawSquare()
drawSquare(400, 400)
```

Now which of these commands works with the new cod?