Simple Repetition

CSC 1200 - Principles of Computing

Overview

- Packages
  - Turtle Commands
- Simple Repetition
- For Loop
- Encapsulation
- Generalization

Turtle Package

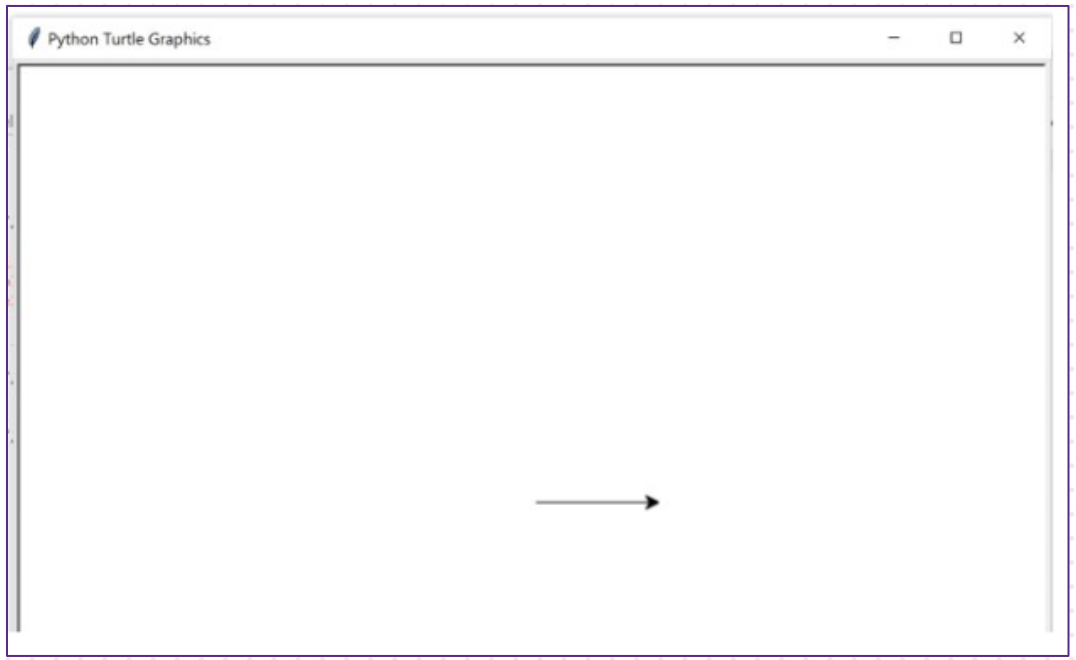
- A **package** is a collection of prewritten modules (and often variables) that are available for a programmer to use in his/her programs so that the programmer doesn't have to implement that functionality.
- For example, we have already used the math package so that we would not have to implement sin, cos, tan, log, or define the variable pi for ourselves.

- A Python graphics package called turtle is available for use. It provides the programmer with the ability to draw lines by steering a "turtle" (basically your cursor) around the screen.
  - This package is based on a program called "Logo" that has been around forever.

A Simple Turtle Program

```
import turtle

turtle.forward( 90 )
```

- Just like the math package, in order to be able to use the turtle package, we must use the keyword import in order to make the contents of the package available to the program.
- The forward function moves the “turtle” forward 90 units, leaving behind a visualization of the trail that it traveled along.
- We can string together various turtle commands to draw shapes (of varying complexity).

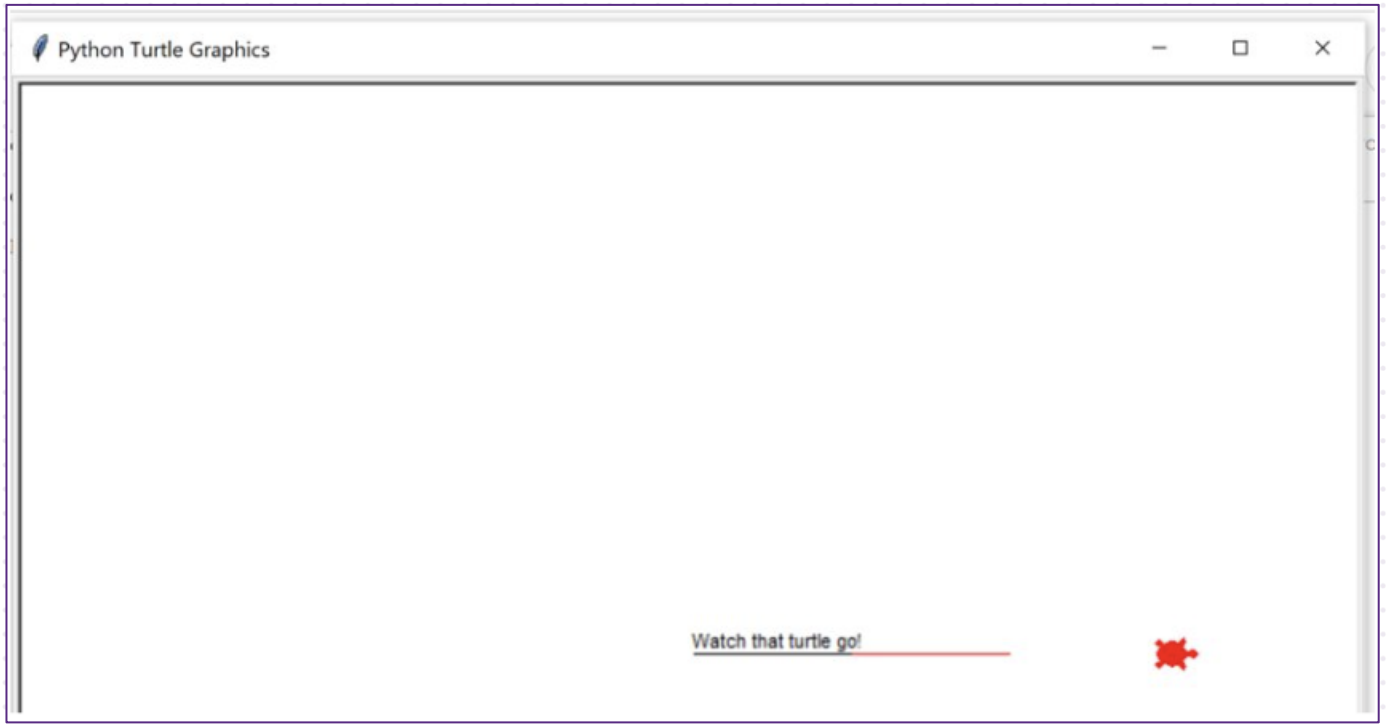


Some Basic Turtle Commands

forward	speed	reset
backward	pendown	clear
left	penup	write
right	pensize	showturtle
goto	color	hideturtle
setx	pencolor	shape
sety		

Modified Program

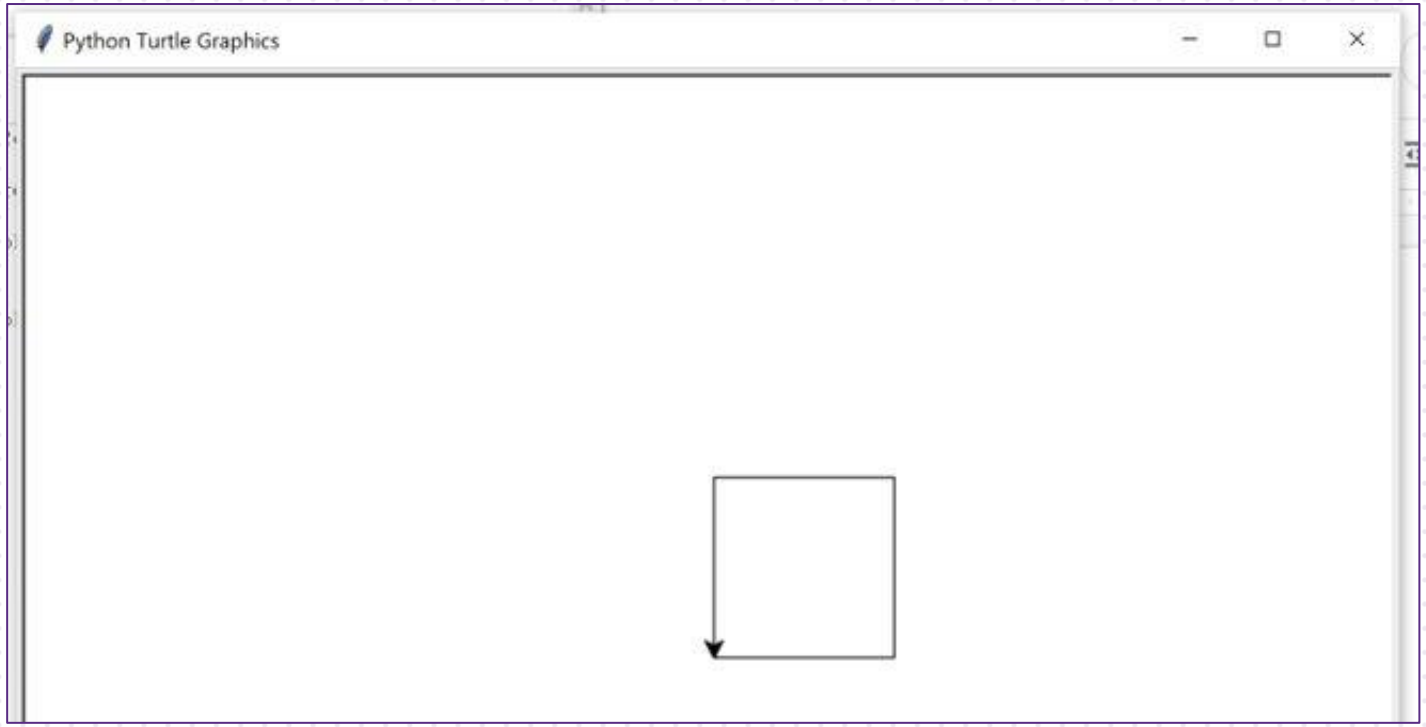
```
from turtle import *

write("Watch that turtle go!")
forward( 90 )

color( "red" )
shape( "turtle" )

forward( 90 )

penup()
forward( 90 )
```



Simple Repetition

- Suppose we wanted to draw a square. We could do the following:

```
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
```



- What if we had to draw an octagon? A dodecagon? An icosagon? Oh my!
- Whenever we repeat the same statements many times, we should use a programming construct called a **loop**.

For Loop

General (basic) syntax:

for target in list:

<tab>statements

Range

The range(num) function returns a sequence of numbers starting with 0 (default), incrementing by 1 (default), and ending with num.

Example:

```
for num in range(4):

    print( num )
```

```
>>> print( range(4) )
range(0, 4)
>>> for num in range(4):
...     print( num )
...
...
0
1
2
3
>>>
```

Use For Loop to Draw Square

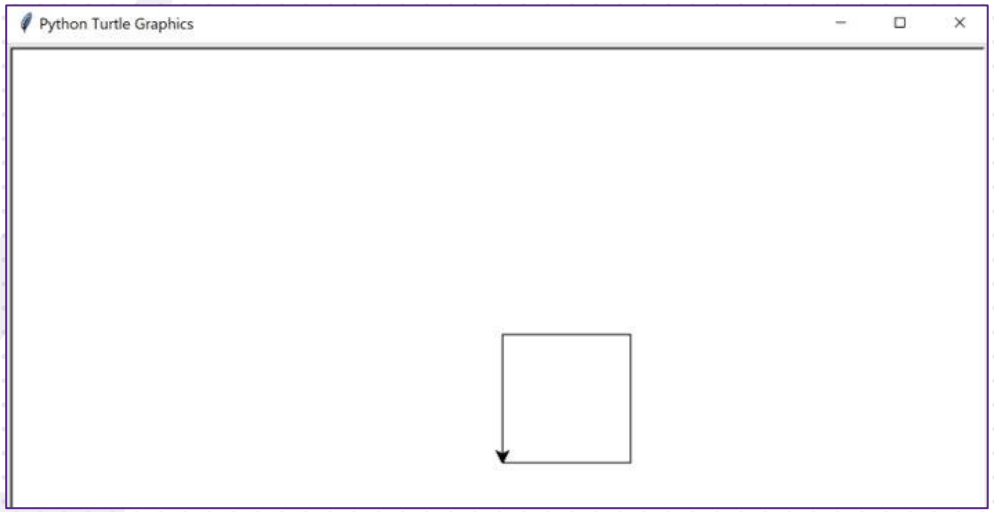
```
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
```

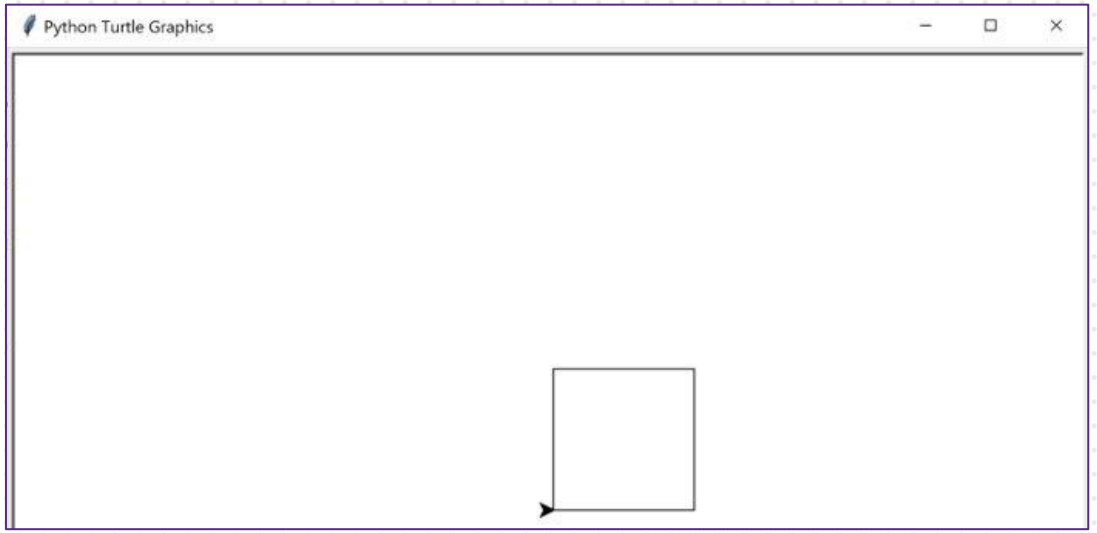
→

```
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
```

→

```
for num in range(4):
    forward(100)
    left(90)
```





Encapsulation

- The Python language and libraries provide many functions for us to use when we write programs (e.g. print(), math.sin())
- We don't need to know the details about how these work. We just want to use them.
- You can also write your own functions. The "main program" can just use these functions, and someone reading your main program doesn't need to know the details about how the function works.

- Wrapping up a piece of code in a function is called **encapsulation**.

Advantages of encapsulation:

- Increases readability
- Decreases code size (due to re-use)

Encapsulation Example

```
from turtle import *

def DrawSquare():
    for num in range( 4 ):
        forward( 100 )
        left( 90 )

#########  Main Program ############
DrawSquare()
exitonclick()
```

Generalization

- Adding a parameter to a function is an example of **generalization**.
- For example: The square we draw is always black. How could we make a generalized function that draws a square of any desired color?
- Need to pass in a parameter to the function that specifies the color of the square to be drawn.

```
def DrawColoredSquare(col):
    color( col )
    for num in range( 4 ):
        forward( 100 )
        left( 90 )
```

- Rainbow Square example program