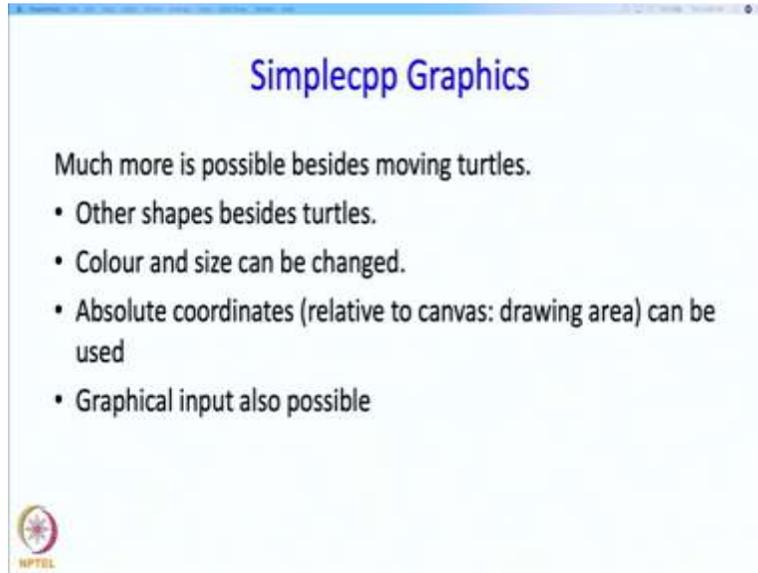


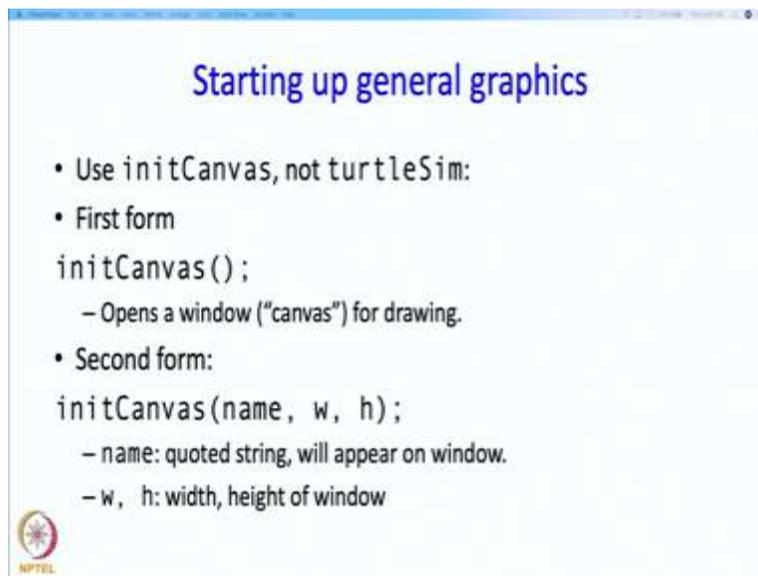
**An Introduction to Programming through C++**  
**Professor Abhiram G. Ranade**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Bombay**  
**Lecture No. 05**  
**Simplecpp Graphics**

(Refer Slide Time: 0:41)



Hello and welcome to the third lecture of the second week of the course on an introduction to programming through C++ for NPTEL. The topic of this lecture sequence is Simplecpp Graphics, so in simplecpp graphics a lot more is possible besides moving turtles. So you can have other shapes also besides turtles. You can change the colour and size, you can have absolute coordinates, so relative to canvas and graphical input is also possible.

(Refer Slide Time: 0:56)



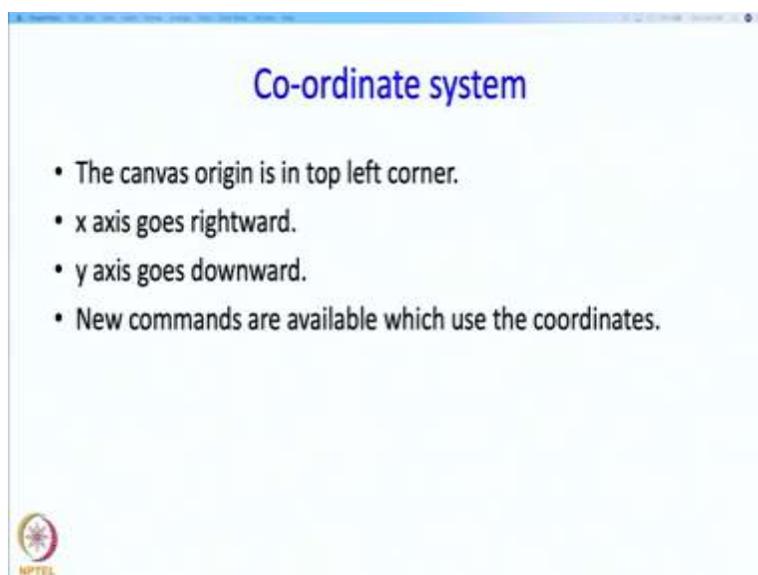
### Starting up general graphics

- Use `initCanvas`, not `turtleSim`:
- First form  
`initCanvas()`;  
– Opens a window (“canvas”) for drawing.
- Second form:  
`initCanvas(name, w, h)`;  
– name: quoted string, will appear on window.  
– w, h: width, height of window



So to start up this more general graphics, you have to use `initCanvas` and not `turtlesim`. So the first form of `initCanvas` is, `initCanvas()` without any arguments and this will just open a window or the canvas for drawing. And the second form, you can give a name to that window and you can also say how wide that window is going to be and how high it is going to be, okay so the name should be quoted string and `w` and `h` should be numbers.

(Refer Slide Time: 1:32)



### Co-ordinate system

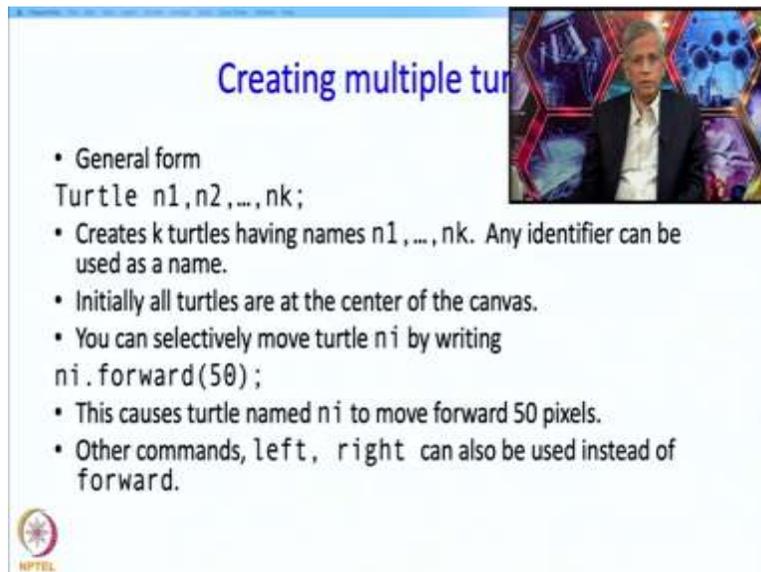
- The canvas origin is in top left corner.
- x axis goes rightward.
- y axis goes downward.
- New commands are available which use the coordinates.



Now, the drawing area or the canvas has a coordinate system. So the canvas origin is in the top left corner, the x axis goes rightwards as you are normally accustomed to, the y axis on the other

hand goes downward rather than upward as you might be more commonly using in math. Then the commands that you use, may not all use the coordinate system but there will be some commands which will use the coordinate system.

(Refer Slide Time: 2:06)



**Creating multiple turtles**

- General form  
`Turtle n1,n2,...,nk;`
- Creates  $k$  turtles having names  $n_1, \dots, n_k$ . Any identifier can be used as a name.
- Initially all turtles are at the center of the canvas.
- You can selectively move turtle  $n_i$  by writing  `$n_i$ .forward(50);`
- This causes turtle named  $n_i$  to move forward 50 pixels.
- Other commands, `left`, `right` can also be used instead of `forward`.



Okay, so how do you create other objects? Well, let us start with how do you create many turtles, rather than use just the one that is given to us. So the general form is to say something like turtle  $n_1, n_2, n_k$  where  $n_1, n_2, n_k$  are the names that you are going to give to the turtles that you are going to create. So this is going to create  $k$  turtles having names  $n_1$  through  $n_k$  and of course any identifier can be used.

Initially all the turtles are at the centre of the canvas and all are facing right, like our single turtle but that single turtle which you got by default is no longer going to be there if you used `initCanvas`. You can selectively move the turtle whose name is  $n_i$  by writing  `$n_i$ .forward(50)` or whatever rather than just `forward(50)`. So since, you have several turtles you have to say which turtle you are talking to. And others command like `left`, `right` can also be used instead of `forward`.

(Refer Slide Time: 3:16)

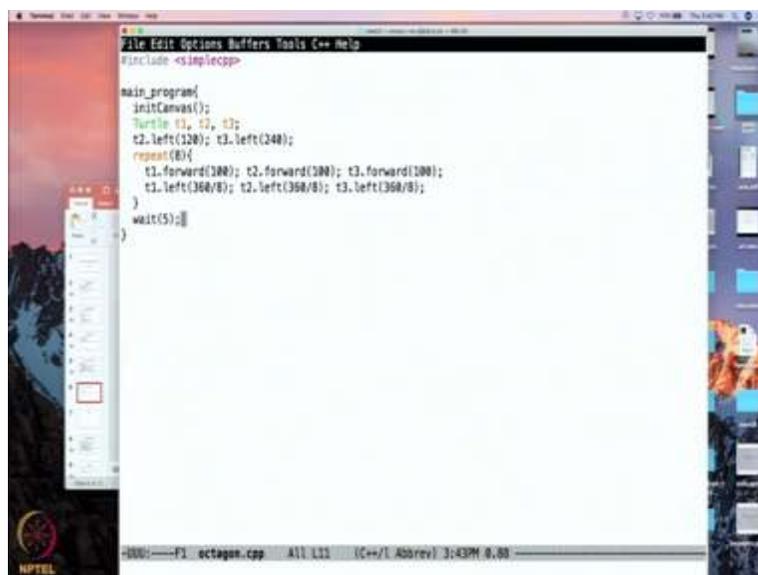
```
Drawing 3 octagons using 3 turtles moving
simultaneously

main_program{
  initCanvas();
  Turtle t1, t2, t3;
  t2.left(120); t3.left(240);
  repeat(8){
    t1.forward(100); t2.forward(100); t3.forward(100);
    t1.left(360/8); t2.left(360/8); t3.left(360/8);
  }
}
```



Okay, so just to give you a sense of what we can do, here is a programme and in it we have the first statement is just this `initCanvas` statement so the canvas is created, then we are going to create three turtles and as I said they are going to be at the centre and facing right, but now we are going to tell different things to the turtles. So the first, `t1` will leave as it is, turtle `t2` we'll ask to turn left by 120 degrees, `t3` we'll ask to turn left by 240 degrees.

(Refer Slide Time: 4:14)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

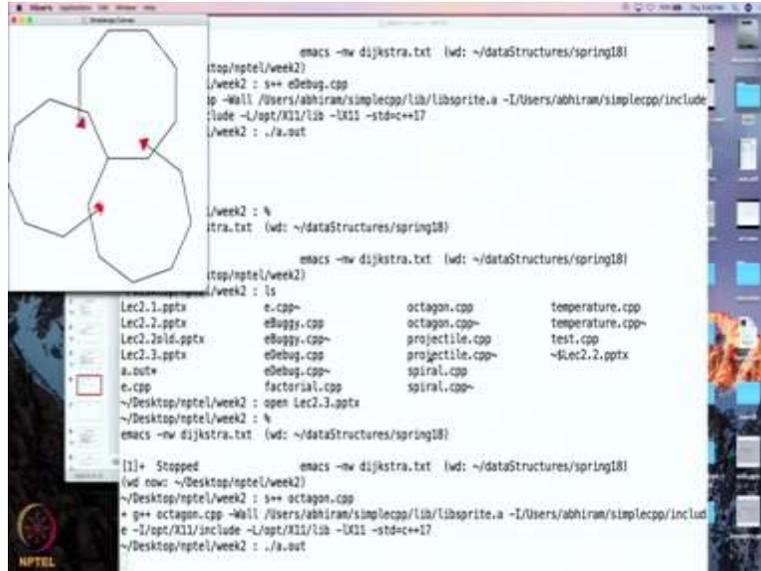
main_program{
  initCanvas();
  Turtle t1, t2, t3;
  t2.left(120); t3.left(240);
  repeat(8){
    t1.forward(100); t2.forward(100); t3.forward(100);
    t1.left(360/8); t2.left(360/8); t3.left(360/8);
  }
  wait(5);
}
```

—F1 octagon.cpp All L11 (C++/1 Abbrev) 3:43PM 8,88

And now we are going to get the turtles to move, so all of them are going to move forward and all of them are going to turn left by 360 by 8 degrees, so each one of them is going to be drawing

an octagon, okay. So let's see what happens, okay so the program that we wrote and let us now execute it.

(Refer Slide Time: 4:19)



So we will compile it first, so let us execute it. See that? each turtle was moving as we commanded. The turtle went out but again came back in as soon as its coordinates became consistent with the material of the canvas okay, so we just did this.

(Refer Slide Time: 4:46)

### Creating graphical objects

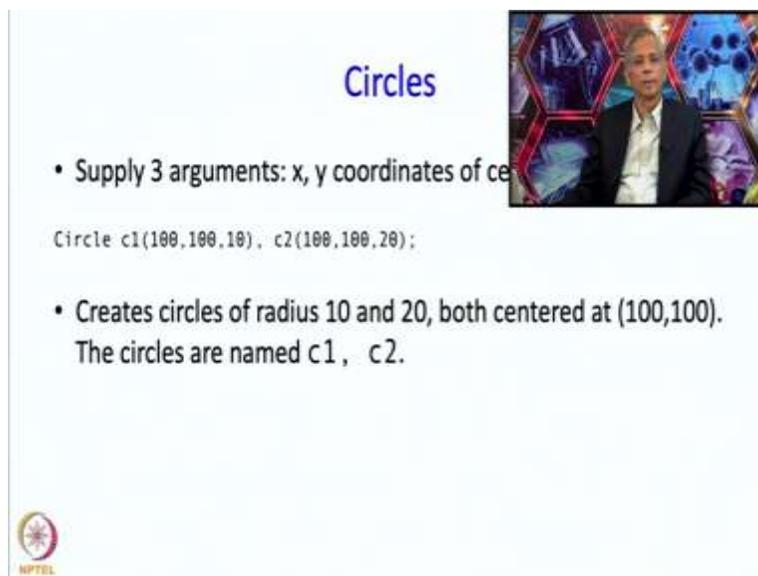
- Basic form:  
`shape-type name(arguments);`
- Creates graphical object of type `shape-type`. Possible types:
  - Circle, Rectangle, Line, Text
- `name` : Name given to created object.
- Created object can be manipulated by writing `name.forward(100);` etc.
- Multiple objects of same type can be created by giving more comma separated names with arguments.
- Pens of non turtle shapes are up by default.



Now, we are going to turn to creating other graphical objects, in general to create a general object, you do something like what you do to create a variable. You are going to give the type, this time the shape type and then the name along with the arguments. So the shapes that you can create are circle, rectangle, line and text, so the name that we have in this basic form is the name to be given to the created object and you can move the created object forward or in general manipulate it by writing name dot whatever operation. So if you write name.forward(100) then you will be moving that object forward by 100 pixels, so each object will have a current direction in which it is facing and that, and forward will cause that object to move in current direction.

And we already saw this that multiple objects of the same type can be created by giving comma separated names with arguments. And all these objects also have pens, but unlike turtles whose pens are down by default these other objects have their pens up, so they will not draw by default.

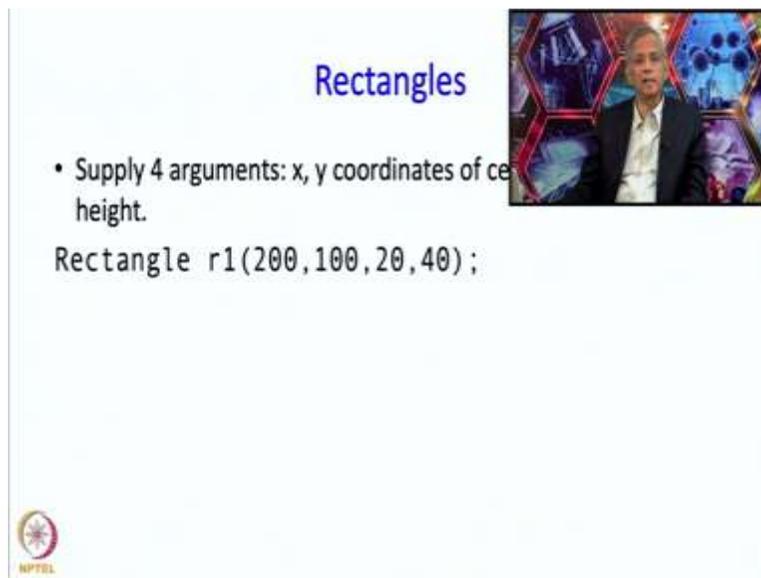
(Refer Slide Time: 6:10)



The slide is titled "Circles" in blue text. It features a video inset in the top right corner showing a man in a suit speaking. The main content consists of two bullet points and a code snippet. The first bullet point states: "Supply 3 arguments: x, y coordinates of centre and radius". The code snippet is: `Circle c1(100,100,10), c2(100,100,20);`. The second bullet point states: "Creates circles of radius 10 and 20, both centered at (100,100). The circles are named c1, c2." In the bottom left corner, there is a small circular logo with the text "NPTEL" below it.

Okay, so how do you create circles? So we are going to supply 3 arguments, the first two are the x, y coordinates of the centre and then the radius. So for example, if I write `Circle c1(100,100,10)` that gives me one circle at centre (100, 100) of radius 10, the second `c2(100,100,20)` will give me another circle at the same centre but with radius 20, so these two will be concentric circles.

(Refer Slide Time: 6:41)



**Rectangles**

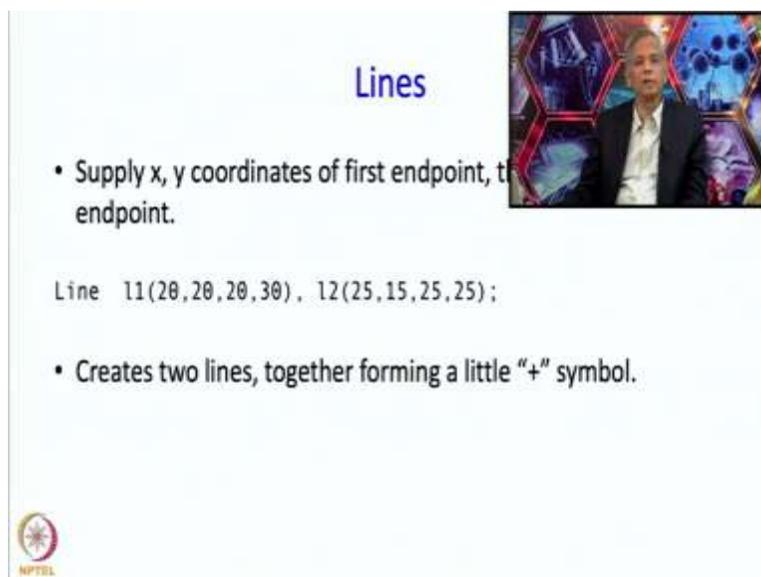
- Supply 4 arguments: x, y coordinates of centre and width and height.

```
Rectangle r1(200,100,20,40);
```

NPTTEL

I can make rectangles, so I can make a rectangle called r1 and the first two numbers are going to be interpreted as the x, y coordinates of the centre. So in this case (200,100) is the centre of the rectangle and the width is 20 and the height is 40.

(Refer Slide Time: 7:01)



**Lines**

- Supply x, y coordinates of first endpoint, then x, y coordinates of second endpoint.

```
Line l1(20,20,20,30), l2(25,15,25,25);
```

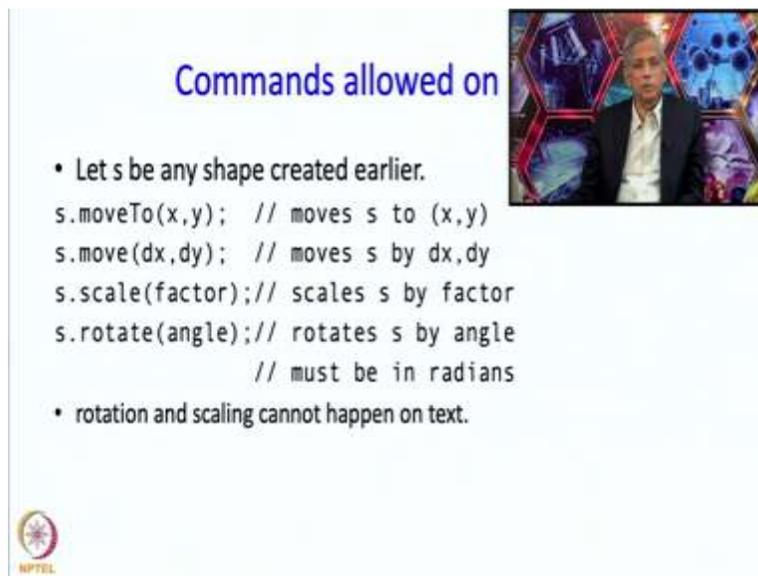
- Creates two lines, together forming a little "+" symbol.

NPTTEL

I can create lines and I just have to supply the coordinates of the first endpoint and then the coordinates of the second endpoint. So for example, here I have a line starting at (20, 20) going to (20, 30) So (20,20) the x coordinate remains the same, the y coordinate goes from 20 to 30, so



(Refer Slide Time: 9:19)



**Commands allowed on**

- Let  $s$  be any shape created earlier.

```
s.moveTo(x,y); // moves s to (x,y)
s.move(dx,dy); // moves s by dx,dy
s.scale(factor); // scales s by factor
s.rotate(angle); // rotates s by angle
                  // must be in radians
```

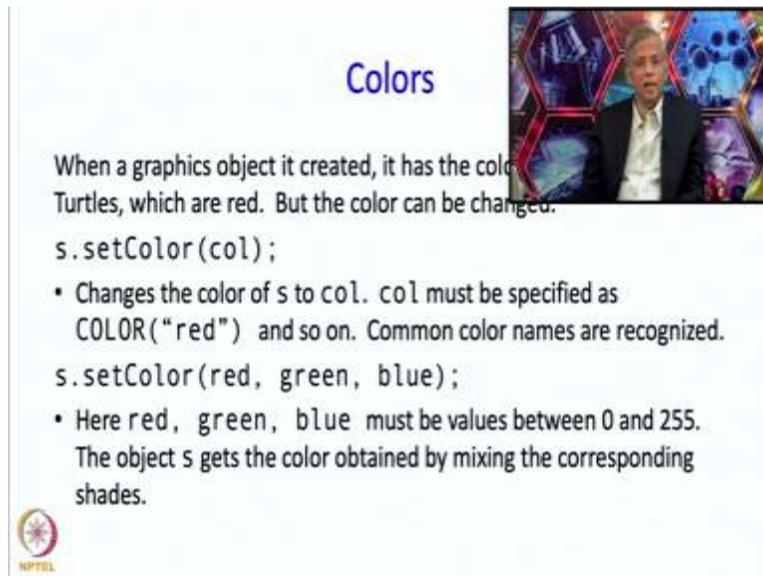
- rotation and scaling cannot happen on text.



Okay so, there are some commands allowed on shapes so for example I can say move shape  $s$  to coordinates  $x, y$ , so  $x, y$  are absolute coordinates over here, the shape which was created at some other point has moved to coordinate  $x, y$ . So by the way when I say move a shape, what moves is its centre, so the centre of the circle moves, the centre of the rectangle moves and for lines also the centre of the line moves.

I can move by giving a relative displacement, so if I say  $s.move(dx, dy)$  then its original position was  $x_1, y_1$  then the new position is  $x_1$  plus  $dx$  and  $y_1$  plus  $dy$ , okay. I can scale a shape, so if I say  $s.scale(2)$  then that shape will become twice as big, its centre will remain where it was earlier, so around that centre it will become twice as big. I can rotate it and it rotates again around the centre so here I specify the angle, okay and this angle must be in radians, okay so it is, and radians is actually the more common measure on the computer just as it is more common measure in scientific work. Now rotation and scaling, as of today cannot happen on text, but there are, there are efforts to change that.

(Refer Slide Time: 11:02)



**Colors**

When a graphics object is created, it has the color black. Turtles, which are red. But the color can be changed.

```
s.setColor(col);
```

- Changes the color of `s` to `col`. `col` must be specified as `COLOR("red")` and so on. Common color names are recognized.

```
s.setColor(red, green, blue);
```

- Here `red`, `green`, `blue` must be values between 0 and 255. The object `s` gets the color obtained by mixing the corresponding shades.



When a graphic subject is created it has the colour black except for turtles which are red, but the colour can be changed. So you can write if `s` is the graphic subject or `s` is the shape that you created then you can change its colour by writing `s.setColor(col)` so notice that the spelling over here is `c-o-l-o-r` sort of paying homage to the American spelling which was used in the logo language from which we said this turtle graphic has been inspired. So this will change the color of `s` to `col`, `col` must be specified as this text color and in parentheses the actual name of the colour red or whatever so common colour names are recognised so you can just specify them.

Another way is to specify the colour to whatever fine shade you want and as you might know colours on a computer are obtained by mixing the primary colours red, green and blue, so you specify what fraction or what amount of red you want, what amount of green you want, what amount of blue you want and you get their mixture. And these numbers red, green and blue must be between 0 and 255. Let me just observe that if you make all of them 255, you will get the colour white, and if you make all of them 0 you will get the colour black.

(Refer Slide Time: 12:36)

### Filling Rectangles and Circles



```
s.setFill(v);
```

- Allowed only when s is a Rectangle or a Circle.
- If v is true, then the interior of s is filled with its color.
- If v is false, then the interior is left white.



If you assign a colour to a shape, it only really changes the border, but if you want it to change the interior then you have to say, you have to fill the shape, so for this you have to say `s.setFill()`, so `setFill` you can set to true or false and true means that I want the interior filled and false means I want the interior left white. So this is allowed only when s is a circle or a rectangle and for others this does not really makes sense.

(Refer Slide Time: 13:13)

### Tracking a shape



- `s.getX()` returns the current x coordinate of the shape.
- Likewise `s.getY()` returns the current y coordinate of the shape.
- `s.getOrientation()` returns the current orientation, i.e. the angle through which s has been rotated so far.
- `s.getScale()` returns the current scale factor used for s.

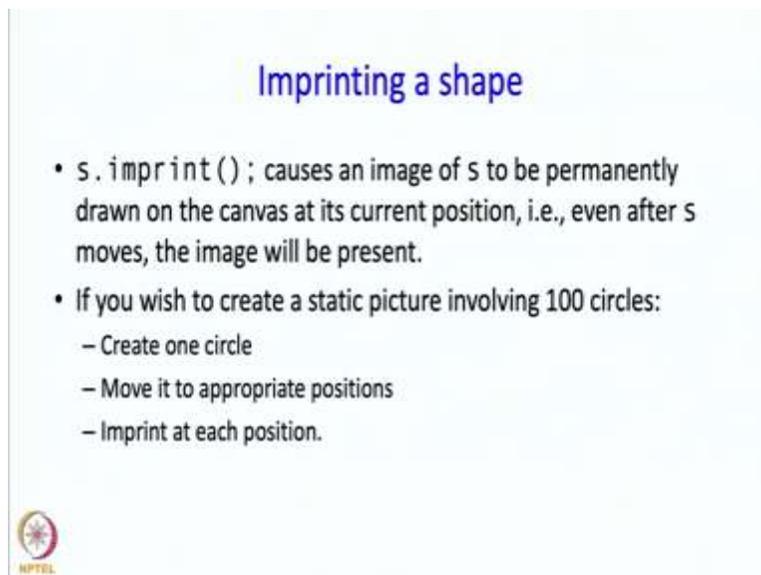


Once you have drawn a shape you can ask where is the shape currently and `s.getX()` will return the x coordinate of that shape or the x coordinate of the centre and similarly, `s.getY()` will give

you the y coordinate of the centre. Similarly, `s.getOrientation()` will return the current orientation, that is the angle through which `s` has been rotated so far. So remember that the originally all shapes start facing the right direction and so from that the orientation is measured.

`getScale()`, returns to you the current scale factor, so you might have done fair amount of scaling so the end result of that is return if you right this command. So this command evaluates to the current scale factor used for `s`. So you could have known that yourself because after all you were the one who issued the scale commands, but this is a way by which you do not really have to keep track of all things which you did.

(Refer Slide Time: 14:27)



### Imprinting a shape

- `s.imprint()`; causes an image of `s` to be permanently drawn on the canvas at its current position, i.e., even after `s` moves, the image will be present.
- If you wish to create a static picture involving 100 circles:
  - Create one circle
  - Move it to appropriate positions
  - Imprint at each position.



One useful operation is to imprint a shape, so `s.imprint()` will cause an image of `s` to be permanently drawn on the canvas at its current position. Okay so even after `s` moves, the image will be present. If you wish to create a static picture involving 100 circles or some other objects then it is better to follow this procedure, so you create one circle, you move it to the appropriate positions and you imprint the circle in those positions. Rather than that you could have also done, create 100 circles, but that usually is going to be cumbersome and that usually also takes more time because C++ will have to keep track of all those circles as it draws and things like that. So if you are, if you are not going to have circles which move then it is better to get them into your picture by imprinting them.

(Refer Slide Time: 15:39)

## Graphical input

`getClick()`

- causes the program to wait until the user clicks on the screen
- Then it returns the value  $65536x + y$ , where  $x, y$  are the coordinates of the cursor at the position of the click.
- Note that we can get back the coordinates by writing  

```
int w = getClick();  
int x = w/65536, y = w % 65536;
```
- This works because the coordinates are at most a few thousand, much smaller than 65536.
- You may just write  

```
getClick();
```



You also have graphical input and for this you have the command `getClick()`, so this will cause the program to wait until the user clicks on the screen. This is sort of like the `cin` command, the `cin` command causes the program to wait until the user types something, `getClick()` program causes to wait until the user clicks. This command returns the value 65536, which is 2 raised to 16 incidentally, times  $x$ , plus  $y$ , where  $x$  and  $y$  are the coordinates of the cursor position at the position of the click. So you can wait for the user to click and you can get the point where the user clicked.

So, how do you get the points? Well this is how, so you do not just issue the `getClick()` command, but you say `int w equals to getClick` so that expression 65536 times  $x$  plus  $y$  will get put into  $w$ . Now if you take the quotient that has to be your  $x$  coordinate and if you take the remainder that has to be the  $y$  coordinate, so this how you can discover the coordinates where the user clicked, okay. And this will work provided  $y$  is smaller than 65536, but of course the number of pixels in any device that you might have is likely to be way smaller than 65536 and therefore, this scheme works quite nicely. So you may also write just `getClick()` and not something equal to get clicked, in that case your program will just wait for a click to happen and whatever value that it receives by looking at the positions of the mouse cursor those are just going to get thrown away.

(Refer Slide Time: 17:41)

### Example

```
main_program{
  initCanvas("Projectile", 500, 500);
  int start = getClick();
  Circle p(start/65536, start % 65536, 5); // at click position
  p.penDown(); // let us see its path
  double vx=1, vy=-1, gravity=0.01;
  repeat(500){
    p.move(vx, vy);
    vy += gravity;
    wait(0.01);
  }
  getClick(); // wait for the user to click. Only then terminate.
}
// Will show a circle move as if thrown against gravity
// from the click position.
```



The image shows a slide with code and a video player. The code is a NetLogo-like program for a projectile simulation. The video player shows a man speaking and a whiteboard with a hand-drawn circle and an arrow pointing upwards and to the right, representing a projectile's path.

So let us do an example, so here is a main program, so we are creating a canvas and we are calling it projectile, so this is going to be the title that will appear on the window and I am going to have an integer `int start=getClick()`, so immediately after creating the canvas, I am going to wait for a click to happen. So I am going to create a circle next, but its x coordinate, the x coordinate of the centre is going to be start divided by 65536 or it is going to be that the x coordinate of the click position. And the y coordinate is going to be start mod 65536 or in other words the y coordinate of the click position, so this means that the circle is going to be centred at the point at which you clicked and its radius is going to be 5.

So, circles normally do not have their pen down, so we are going to make the pen go down and now we are going to do something fun, we are going to have velocities for this circle. Well there is no formal velocity as such but we will sort of make it appear as if there is a velocity, so we are going to say that the x velocity is 1, the y velocity is minus 1 and then there is gravitational force of 0.01. So notice that we have our vertical velocity is minus 1, so what is vertical velocity? So what is minus 1? So minus 1 nearly says that our object will want to move in the negative y direction, so what is the positive y direction? So the positive y direction is going downwards as we said at the beginning of the lecture. So the positive y direction is going downwards and so the negative y direction is going upwards. So initially, our object has been assigned velocity. So suppose we clicked at this point, then a circle will be created over here and unit velocity will be assigned in both directions. So our object will be disposed to move in this direction. Now, we will make the object move, so we will make the object move for 500 steps. So in each step the object will move by  $v_x$ ,  $v_y$ . However, in each step the velocity will also change, the velocity will change by 0.01 which is going to be the effect of gravity of course, this is just make believe but this says that the y velocity will increase by 0.01, remember that y velocity initially is negative so as it increases by 0.01 its velocity is going to be decreasing so it was going, it was moving very fast in the upward direction. It is going to slow down in the upward direction and eventually it is going to start moving fast in the downward direction.

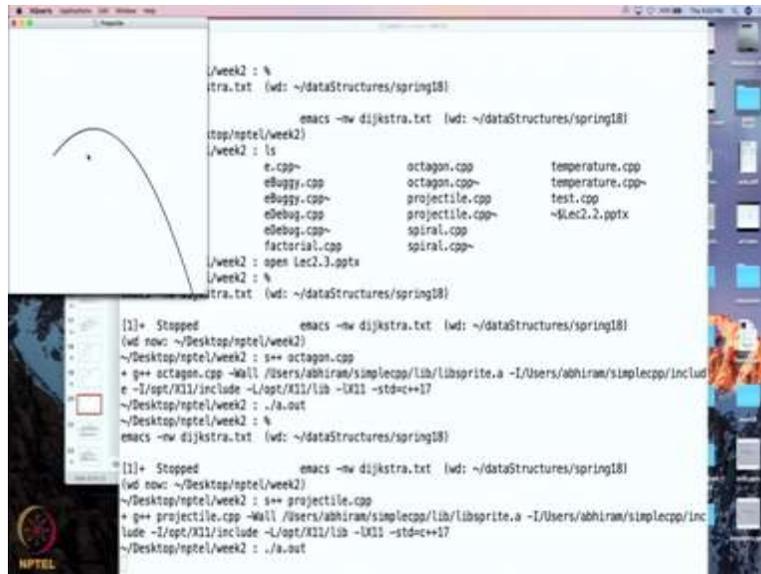
Notice that this is exactly what happens to a stone when you throw the stone in the air, initially it has a vertical upward velocity and gravity acts on it and gravity pulls the stone down and also its velocity faster and faster in the downward direction. So in order for you to see what is going on, we are going to wait for about 100th of a second, okay and that is it. So after the whole thing is finished we are going to wait for you to click so that you get a chance to see what has happened.

(Refer Slide Time: 21:28)

```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

main_program
initCanvas("Projectile", 500, 500);
int start = getClick();
Circle p(start/65536, start % 65536, 5); // at click position
p.pendown(); // let us see its path
double vx=1, vy=-1, gravity=0.01;
repeat(500){
    p.move(vx, vy);
    vy += gravity;
    wait(0.01);
}
getClick(); // wait for the user to click. Only then terminate.
// Will show a circle move as if thrown against gravity
// from the click position.
```

```
-I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week2 : ./a.out
2
1: 1 T: 1 S: 1
1: 2 T: 1 S: 2
3
~/Desktop/npTEL/week2 : %
emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
[1]+ Stopped emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
(wd now: ~/Desktop/npTEL/week2)
~/Desktop/npTEL/week2 : ls
Lec2.1.pptx      e.cpp          octagon.cpp    temperature.cpp
Lec2.2.pptx      eBuggy.cpp    octagon.cpp    temperature.cpp~
Lec2.2old.pptx  eBuggy.cpp~   projectile.cpp  test.cpp
Lec2.3.pptx      eEbug.cpp      projectile.cpp~ ~Lec2.2.pptx
a.out*          eEbug.cpp~    spiral.cpp
e.cpp           factorial.cpp  spiral.cpp~
~/Desktop/npTEL/week2 : open Lec2.3.pptx
~/Desktop/npTEL/week2 : %
emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
[1]+ Stopped emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
(wd now: ~/Desktop/npTEL/week2)
~/Desktop/npTEL/week2 : s++ octagon.cpp
+ g++ octagon.cpp -Wall -I/Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/includ
e -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week2 : ./a.out
~/Desktop/npTEL/week2 : %
emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
[1]+ Stopped emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
(wd now: ~/Desktop/npTEL/week2)
~/Desktop/npTEL/week2 : s++ projectile.cpp
```



So alright, let us do a demo of this, so the program is called a projectile. So this is the program which I showed to you, okay. So let us, let us compile it and let us run it. So you can see that a window has been created, the name, projectile appears at the top and now the program is waiting for me to click on the screen. So suppose, I click on the screen you see a circle got created, sort of a ball and it is moving, it is moving against gravity, gravity is pulling it downwards and it is going to move for 500 steps. So it is going to move for 500 steps and in this case the 500 steps took it outside the canvas but that is okay. But once that entire thing has been drawn, it is waiting for me to click so let me click and that is the end of the program.

(Refer Slide Time: 22:53)

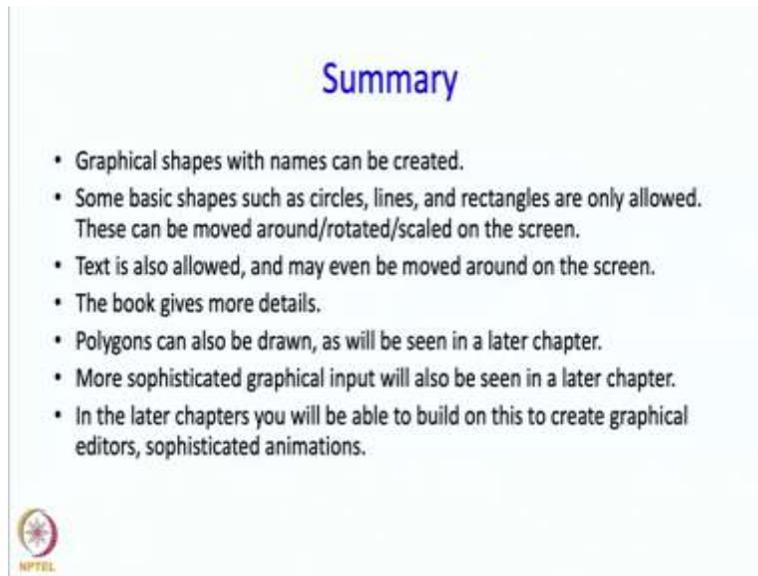
### Exercises

- Draw a plot of  $y = \sin(x)$ . Use imprinting to draw many small lines.
- Suppose a man is walking from the origin in the positive  $x$  direction. In a single time unit the man walks 3 pixels. A dog is at point  $(0,300)$ . At the beginning of each time unit, the dog turns towards the man and in the time unit walks 6 pixels. Show what happens in 50 time units.
- Create nice designs.
- Show the movement of a projectile having horizontal velocity  $v \cos(t)$  and vertical velocity  $v \sin(t)$  for fixed  $v$  and different  $t$ . You should be able to observe that when  $t$  is 45 degrees, the project goes the longest distance.



Okay, so you can do lots of interesting things with what I have told you, so for example you can draw a plot of arbitrary functions say  $y$  equals to  $\sin(x)$ , so, how do you do this? Well you have to decide where exactly that plot has to be drawn, so you will not supply the actual  $x$  coordinates and you may have to scale this whole thing a little bit, but I will let you think about it, but the one point that I must mention here is that it is a curve which you will have to approximate by lines. And those lines will be drawn by imprinting because there will be lots of small lines and you really do not want to create all those lines. Then there are a couple of other exercises and well you can just have fun drawing some nice designs or doing some nice animations.

(Refer Slide Time: 23:42)



The slide features a light blue background with the word "Summary" in a dark blue font at the top center. Below it is a bulleted list of seven items. In the bottom left corner, there is a circular logo with a star-like pattern and the text "NPTEL" underneath it.

## Summary

- Graphical shapes with names can be created.
- Some basic shapes such as circles, lines, and rectangles are only allowed. These can be moved around/rotated/scaled on the screen.
- Text is also allowed, and may even be moved around on the screen.
- The book gives more details.
- Polygons can also be drawn, as will be seen in a later chapter.
- More sophisticated graphical input will also be seen in a later chapter.
- In the later chapters you will be able to build on this to create graphical editors, sophisticated animations.

Okay so, to summarise, you can have graphical shapes with names and circles, lines are only allowed, but can move them, you can rotate them, change their colours. Text is also allowed and the book gives more details. Later on in the course we will see that polygons can also be drawn and later on we may see how dragging and things like that can also be done.

So these are basic things that you have learned in this lecture and you can already use them to create interesting drawings and animations and in fact even graphical editors, we will see that very soon. Thank you.