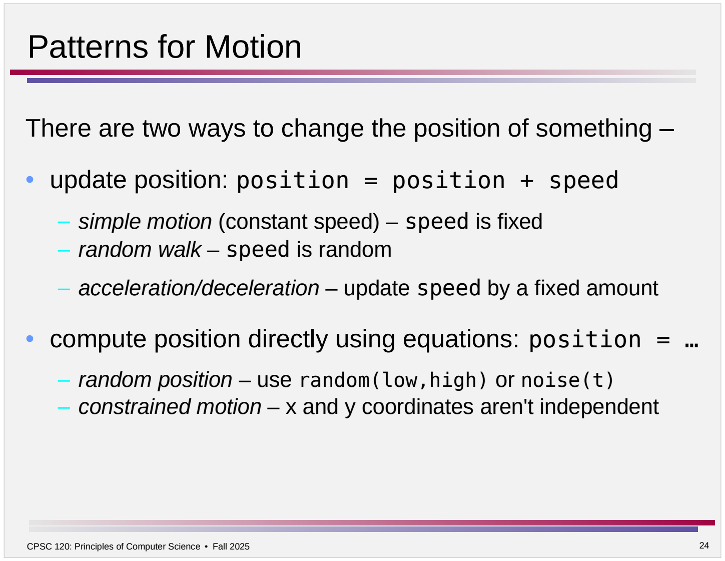
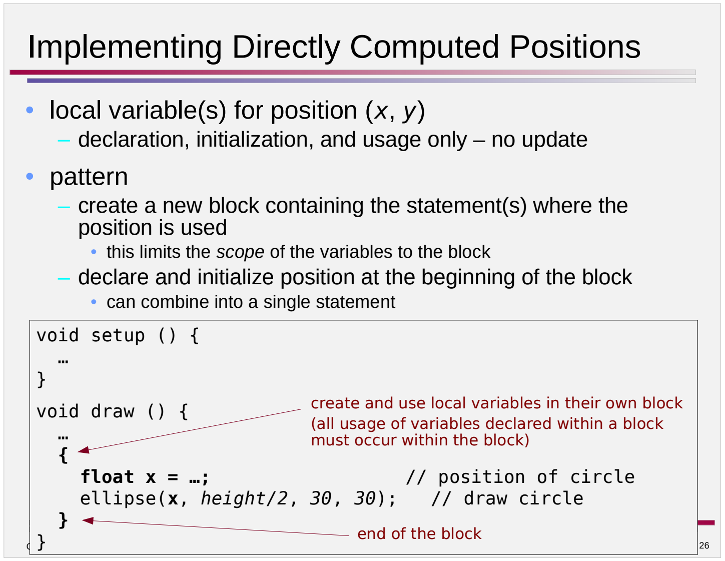


Patterns for Motion

There are two ways to change the position of something –

- update position: `position = position + speed`
  - *simple motion* (constant speed) – `speed` is fixed
  - *random walk* – `speed` is random
  - *acceleration/deceleration* – update `speed` by a fixed amount
- compute position directly using equations: `position = …`
  - *random position* – use `random(low,high)` or `noise(t)`
  - *constrained motion* – x and y coordinates aren't independent



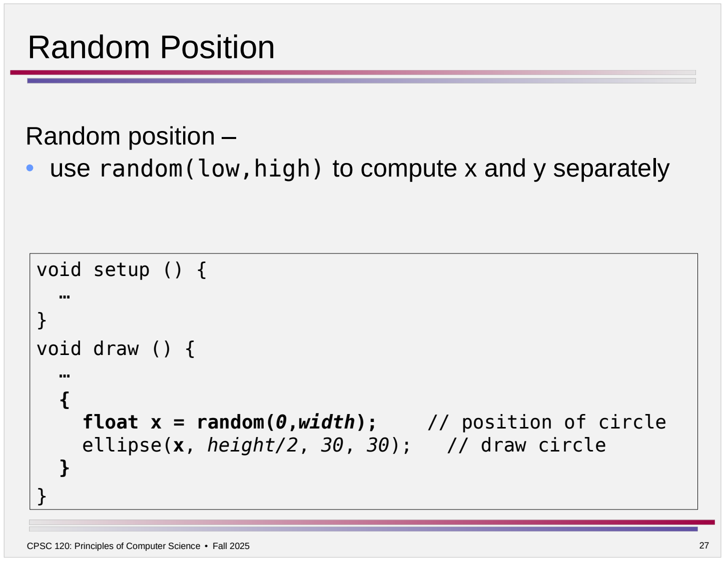
Implementing Directly Computed Positions

- local variable(s) for position (*x*, *y*)
  - declaration, initialization, and usage only – no update
- pattern
  - create a new block containing the statement(s) where the position is used
    - this limits the *scope* of the variables to the block
  - declare and initialize position at the beginning of the block
    - can combine into a single statement

```
void setup () {
  …
}
void draw () {
  …
  {
    float x = …;                    // position of circle
    ellipse(x, height/2, 30, 30);   // draw circle
  }
}
```

create and use local variables in their own block (all usage of variables declared within a block must occur within the block)

end of the block

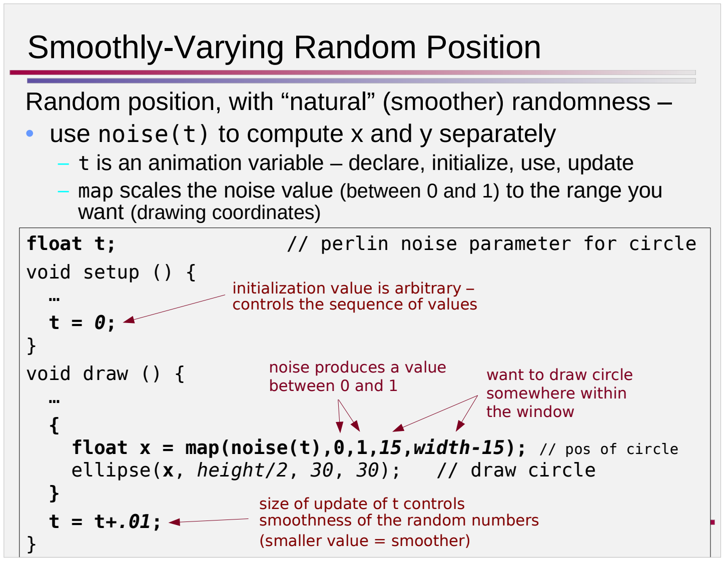


Random Position

Random position –

- use `random(low,high)` to compute x and y separately

```
void setup () {
  …
}
void draw () {
  …
  {
    float x = random(0,width);    // position of circle
    ellipse(x, height/2, 30, 30);   // draw circle
  }
}
```



Smoothly-Varying Random Position

Random position, with "natural" (smoother) randomness –

- use `noise(t)` to compute x and y separately
  - `t` is an animation variable – declare, initialize, use, update
  - `map` scales the noise value (between 0 and 1) to the range you want (drawing coordinates)

```
float t;                  // perlin noise parameter for circle
void setup () {
  …
  t = 0;
}
void draw () {
  …
  {
    float x = map(noise(t),0,1,15,width-15); // pos of circle
    ellipse(x, height/2, 30, 30);   // draw circle
  }
  t = t+.01;
}
```

initialization value is arbitrary – controls the sequence of values

noise produces a value between 0 and 1

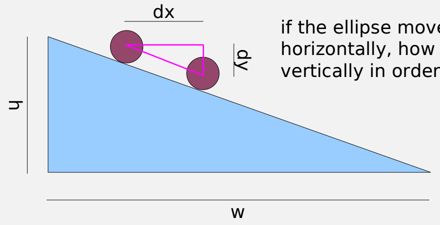
want to draw circle somewhere within the window

size of update of t controls smoothness of the random numbers (smaller value = smoother)

Constrained Motion – Ramps

On a ramp, the object's position is constrained to be along a particular line.

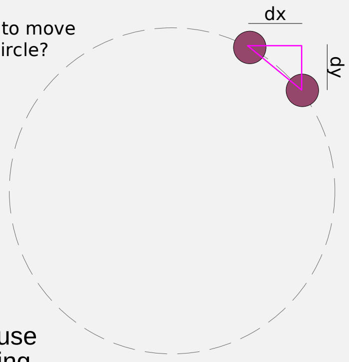
- `xspeed` and `yspeed` aren't independent – there must be a certain relationship between the amount of change in the x coordinate and in the y coordinate



if the ellipse moves distance dx horizontally, how far does it have to move vertically in order to stay on the ramp?

Constrained Motion – Circular Motion

if the ellipse moves distance dx horizontally, how far does it have to move vertically in order to stay on the circle?



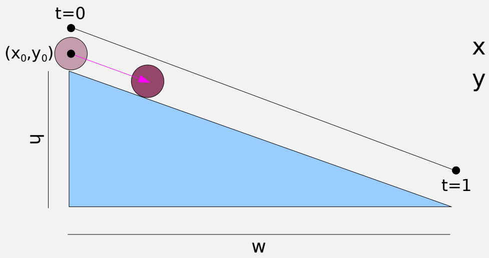
This question is harder for circles than for ramps, because there's not a fixed ratio relating dy to dx.

Parametric Equations

The *parametric equations* for a curve express the coordinates of the points on the curve in terms of functions of an independent *parameter*.

```
x = f(t)
y = f(t)
```

Parametric Equations – Ramps

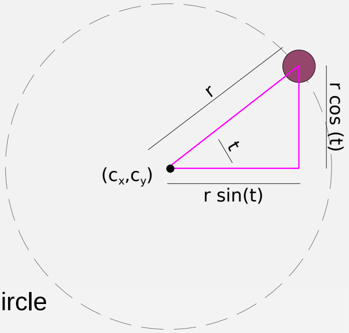


$x = x_0 + t\,w$
$y = y_0 + t\,h$

- w, h are the dimensions of the ramp
- (x_0,y_0) is determined by the position of the ramp and the size of the object on the ramp
- t is the parameter – increasing t moves along the ramp
  - 0 ≤ t ≤ 1 (other values are legal, but are beyond the span of the ramp)

Parametric Equations – Circular Motion

$$x = c_x + r \cos t$$
$$y = c_y + r \sin t$$



- (c_x,c_y) is the center of the circle
- r is the radius of the circle
- t is the parameter – increasing t moves around the circle
  - t is in radians, so 0 ≤ t ≤ 2*`PI` covers one revolution

Implementing Constrained Motion

- animation variable *t*
  - declare, initialize (usually to 0), use (to determine position), update (size of update controls speed)
- local variable(s) for position
  - compute position using parametric equations

```
float t;                 // position parameter for circle
void setup () {
  …
  t = 0;
}
void draw () {
  …
  {
    float x = x0 + t*w;      // position of circle
    float y = y0 + t*h;
    ellipse(x, y, 30, 30);   // draw circle
  }
  t = t+.01;
}
```

0 starts at the beginning of the ramp – use a different value to start farther along

ramp equations

(substitute your own values for x_0, y_0, w, h – or your own equations for a different motion path)

size of update of t controls speed