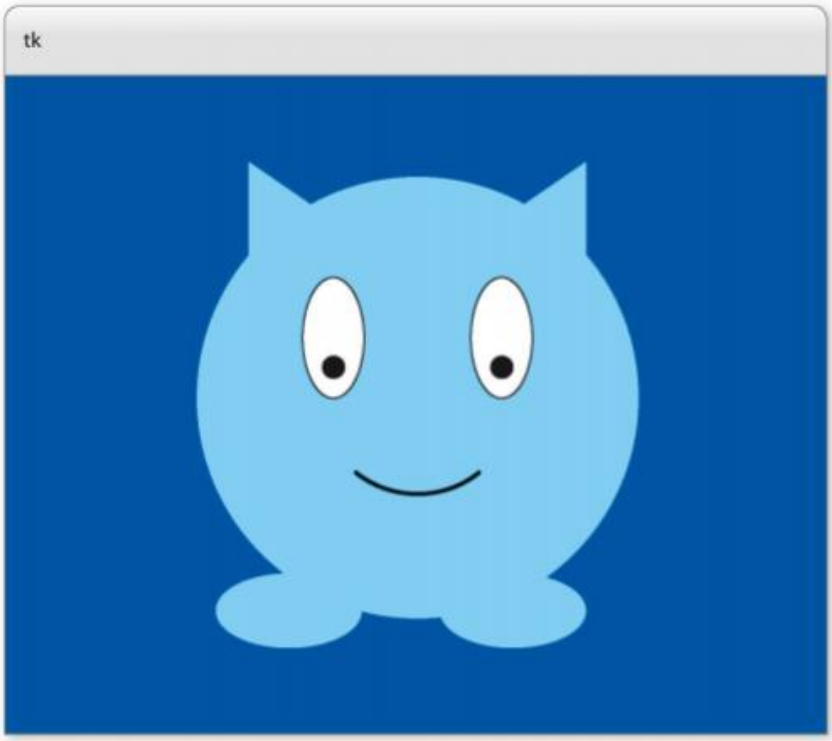
Screen Pet

Have you ever wished you had a pet to keep you company while doing your homework on your computer? In this project, you'll create a pet that "lives" in a corner of your computer screen. It will keep you busy, because you'll need to look after your pet to keep it happy.

What happens

When you start the program, Screen Pet will sit there, with a little smile on its face, blinking at you. Your cute, sky-blue companion will change its expression from normal (below) to happy, cheeky, or sad, depending on how you interact with it on the screen. But don't worry, it's friendly – it won't bite if it gets bored!



Screen Pet appears in a Tkinter window.

△ **Happy face**
If you "stroke it" with the mouse-pointer, Screen Pet beams and blushes.

△ **Cheeky face**
If you double-click on it to "tickle" it, the cheeky pet sticks out its tongue.

△ **Sad face**
If you ignore it, Screen Pet will become sad. Stroking it will cheer it up again.

How it works

Running `Tkinter`'s `root.mainloop()` function sets up a `while` loop that keeps checking for input from the user. The loop keeps going until you close the main `Tkinter` window. This is also how you were able to make a GUI (graphical user interface) that reacted to a user clicking on a button or entering text in Ask the Expert.

▷ **Mainloop animation**
You can also animate images in a `Tkinter` window using the `root.mainloop()` function. By telling it to run functions that change the image at set times, you can make Screen Pet appear to move by itself.

LINGO

Event-driven program

Screen Pet is an event-driven program, which means that the things it does and the order it does them in depend on input from the user. The program looks for inputs, such as key-presses and mouse-clicks, then calls a different function to handle each one. Word-processing programs, video games, and drawing programs are all examples of event-driven programs.

▽ **Screen Pet flowchart**
The flowchart shows the sequence of actions and decisions, and how user inputs affect them. The program runs in an endless loop. It uses an ever-changing happiness variable to keep track of the pet's mood.

Draw your Screen Pet

Let's get started. First you need to create the window where your Screen Pet will live. Then you'll write some code to draw the pet on the screen.

1 Create a new file

Open IDLE. Go to the File menu and select New File, then save the file as "screen_pet.py".

2 Add the Tkinter module

You need to import parts of Python's `Tkinter` module at the start of your program. Type this code to bring in `Tkinter` and open a window where your Screen Pet will live.

This line imports the parts of the `Tkinter` module that you'll need in this project.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas
root = Tk()
```

This line starts `Tkinter` and opens a window.

3 Make a new canvas

In the window, make a dark blue canvas called "c", on which you'll draw your pet. Add this code after the line that opens the `Tkinter` window. These four lines of new code are the start of the main part of your program.

The canvas will be 400 pixels wide and 400 pixels high.

The background colour will be dark blue.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas
root = Tk()
c = Canvas(root, width=400, height=400)
c.configure(bg='dark blue', highlightthickness=0)
c.pack()
root.mainloop()
```

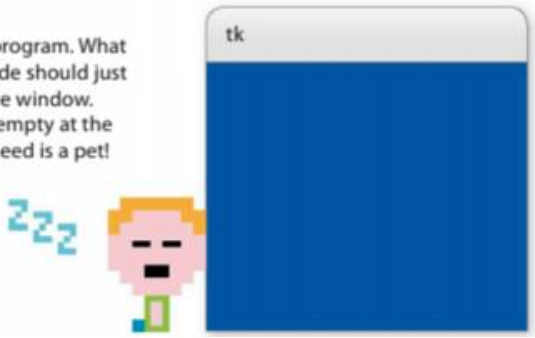
This command arranges things within the `Tkinter` window.

Any commands that start with `c.` relate to the canvas.

This line starts the function that looks out for input events, such as mouse-clicks.

4 Run it

Now try running the program. What do you notice? The code should just show a plain, dark-blue window. It looks a bit dull and empty at the moment – what you need is a pet!



Don't forget to save your work.

5 Get drawing

To draw your pet, add these instructions above the last two lines of code. There's a separate command for each body part. The numbers, called coordinates, tell `Tkinter` what to draw and where to draw it.

Storing the body colour in the variable `c.body_color` means you don't have to keep typing in `'SkyBlue1'`.

```
c.configure(bg='dark blue', highlightthickness=0)
c.body_color = 'SkyBlue1'
body = c.create_oval(35, 20, 365, 350, outline=c.body_color, fill=c.body_color)
ear_left = c.create_polygon(75, 80, 75, 10, 165, 70, outline=c.body_color, fill=c.body_color)
ear_right = c.create_polygon(255, 45, 325, 10, 320, 70, outline=c.body_color, \
                             fill=c.body_color)
foot_left = c.create_oval(65, 320, 145, 360, outline=c.body_color, fill= c.body_color)
foot_right = c.create_oval(250, 320, 330, 360, outline=c.body_color, fill= c.body_color)

eye_left = c.create_oval(130, 110, 160, 170, outline='black', fill='white')
pupil_left = c.create_oval(140, 145, 150, 155, outline='black', fill='black')
eye_right = c.create_oval(230, 110, 260, 170, outline='black', fill='white')
pupil_right = c.create_oval(240, 145, 250, 155, outline='black', fill='black')

mouth_normal = c.create_line(170, 250, 200, 272, 230, 250, smooth=1, width=2, state=NORMAL)

c.pack()
```

In the code, "left" and "right" refer to the left and right of the window as you look at it.

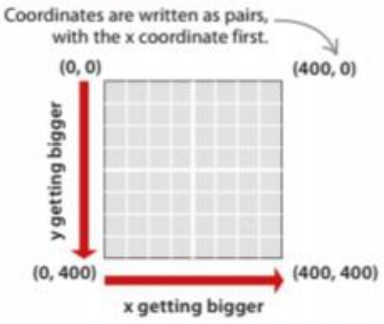
These pairs of coordinates define the start, mid-point, and end of the mouth.

The mouth is a smooth line, 2 pixels wide.

EXPERT TIPS

Tkinter coordinates

The drawing instructions use x and y coordinates. In `Tkinter`, the x coordinates start at 0 on the left and increase as you move across the window, until they reach 400 on the far right. The y coordinates start at 0 at the top. They get bigger as you move down, until they reach 400 at the bottom.



6 Run it again

Run the program again and you should see Screen Pet sitting in the middle of the `Tkinter` window.

Blinking pet

Your Screen Pet looks cute, but it's not doing anything! Let's write some code to get it blinking. You'll need to create two functions: one to open and shut the eyes, the other to tell them how long to stay open and shut for.

To blink, the eyes fill with sky blue and the pupils disappear

7 Open and close the eyes

Create this function, `toggle_eyes()`, at the top of your file, under the first line of code. It makes the eyes look closed by hiding the pupils and filling the eyes with the same colour as the body. It also switches the eyes between being open and closed.

First the code checks the eyes' current colour: white is open, blue is closed.

This line sets the eyes' `new_color` to the opposite value.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas

def toggle_eyes():
    current_color = c.itemcget(eye_left, 'fill')
    new_color = c.body_color if current_color == 'white' else 'white'
    current_state = c.itemcget(pupil_left, 'state')
    new_state = NORMAL if current_state == HIDDEN else HIDDEN
    c.itemconfigure(pupil_left, state=new_state)
    c.itemconfigure(pupil_right, state=new_state)
    c.itemconfigure(eye_left, fill=new_color)
    c.itemconfigure(eye_right, fill=new_color)
```

Now the code checks if the current state of the pupils is `NORMAL` (visible) or `HIDDEN` (not visible).

These lines change the visibility of the pupils.

This line sets the pupils' `new_state` to the opposite value.

These lines change the eyes' fill colour.

LINGO

Toggling

Switching between two states is known as "toggling". So you "toggle" the lights in your house when you switch them on and off. The blinking code switches, or toggles, between Screen Pet's eyes being open and closed. If the eyes are closed when you run it, they'll change to being open. If they're open, they'll change to being closed.

8 **Realistic blinking**

The eyes need to close only briefly and stay open for a while between blinks. Add this function, **`blink()`**, under the code you typed in Step 7. It blinks the eyes for a quarter of a second (250 milliseconds), then finishes with a command that tells **`mainloop()`** to call it again after 3 seconds (3,000 milliseconds).

```
    c.itemconfigure(eye_right, fill=new_color)

def blink():
    toggle_eyes()
    root.after(250, toggle_eyes)
    root.after(3000, blink)

root = Tk()
```

Close the eyes.

Wait 250 milliseconds, then open the eyes.

Wait 3,000 milliseconds, then blink again.

9 **Animate!**

Put this line in the main part of your program, just above the last line. Now run the program. Your pet will come to life after 1 second (1,000 milliseconds) and sit there blinking until you close the window.

```
root.after(1000, blink)
root.mainloop()
```

Wait 1,000 milliseconds, then start blinking.

Changing moods

Screen Pet looks quite happy just now, with its little smile, but let's cheer it up even more. We'll give it a bigger, beaming smile and bright, rosy cheeks.

10 **Make a happy face**

Add this code to the part of the program that draws Screen Pet, after the line that creates the "normal" mouth. As well as a happy mouth and pink cheeks, it also draws a sad mouth. They will all remain hidden for now.

Create a happy mouth.

Create a sad mouth.

```
mouth_normal = c.create_line(170, 250,200, 272, 230, 250, smooth=1, width=2, state=NORMAL)
mouth_happy = c.create_line(170, 250, 200, 282, 230, 250, smooth=1, width=2, state=HIDDEN)
mouth_sad = c.create_line(170, 250, 200, 232, 230, 250, smooth=1, width=2, state=HIDDEN)

cheek_left = c.create_oval(70, 180, 120, 230, outline='pink', fill='pink', state=HIDDEN)
cheek_right = c.create_oval(280, 180, 330, 230, outline='pink', fill='pink', state=HIDDEN)

c.pack()
```

These lines create pink, blushing cheeks.

11 Show the happy face

Next, create a function called **show_happy()** to reveal the happy expression when you move the mouse-pointer over Screen Pet as if you were stroking it. Type this code beneath the **blink()** function you added in Step 8.

LINGO

Event handler

The function **show_happy()** is an event handler. This means it's only called when a particular event happens, so that it can deal with it. In your code, stroking your pet calls **show_happy()**. In real life, you might call a "mop the floor" function to handle a "spill drink" event!

The **if** line checks to see if the mouse-pointer is over the pet.

event.x and **event.y** are the coordinates of the mouse-pointer.

```
root.after(3000, blink)

def show_happy(event):
    if (20 <= event.x <= 350) and (20 <= event.y <= 350):
        c.itemconfigure(cheek_left, state=NORMAL)
        c.itemconfigure(cheek_right, state=NORMAL)
        c.itemconfigure(mouth_happy, state=NORMAL)
        c.itemconfigure(mouth_normal, state=HIDDEN)
        c.itemconfigure(mouth_sad, state=HIDDEN)
    return
```

Show the pink cheeks.

Show the happy mouth.

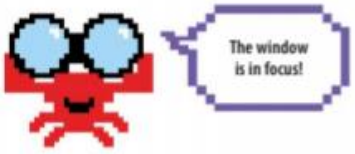
Hide the normal mouth.

Hide the sad mouth.

EXPERT TIPS

Focus

Tkinter won't be able to spot you moving the mouse-pointer over the window to stroke Screen Pet unless the window is "in focus". You can get it in focus by clicking once anywhere in the window.



12 Happy moves

When the program starts, Screen Pet blinks without you doing anything. But to get it to look happy when it's being stroked, you need to tell it what event to look out for. **Tkinter** calls the mouse-pointer moving over its window a **<Motion>** event. You need to link this to the handler function by using **Tkinter**'s **bind()** command. Add this line to the main part of your program. Then run the code and stroke the pet to try it out.

```
c.pack()

c.bind('<Motion>', show_happy)

root.after(1000, blink)
root.mainloop()
```

This command links the moving mouse-pointer to the happy face.

13 Hide the happy face

You only want Screen Pet to look really happy when you're actually stroking it. Add a new function, **hide_happy()**, below the code for **show_happy()**. This new code will set Screen Pet's expression back to normal.

Don't forget to save your work.

```
def hide_happy(event):
    c.itemconfigure(cheek_left, state=HIDDEN)
    c.itemconfigure(cheek_right, state=HIDDEN)
    c.itemconfigure(mouth_happy, state=HIDDEN)
    c.itemconfigure(mouth_normal, state=NORMAL)
    c.itemconfigure(mouth_sad, state=HIDDEN)
    return
```

Hide the pink cheeks.

Hide the happy mouth.

Show the normal mouth.

Hide the sad mouth.

14 Call the function

Type this line to call **hide_happy()** when the mouse-pointer leaves the window. It links **Tkinter**'s **<Leave>** event to **hide_happy()**. Now test your code.

```
c.bind('<Motion>', show_happy)
c.bind('<Leave>', hide_happy)

root.after(1000, blink)
```

What a cheek!

So far, your pet has been very well behaved. Let's give it a cheeky personality! You can add some code that will make Screen Pet stick its tongue out and cross its eyes when you tickle it by double-clicking on it.

15 Draw the tongue

Add these lines to the code that draws Screen Pet, under the line that creates the sad mouth. The program will draw the tongue in two parts, a rectangle and an oval.

```
mouth_sad = c.create_line(170, 250, 200, 232, 230, 250, smooth=1, width=2, state=HIDDEN)
tongue_main = c.create_rectangle(170, 250, 230, 290, outline='red', fill='red', state=HIDDEN)
tongue_tip = c.create_oval(170, 285, 230, 300, outline='red', fill='red', state=HIDDEN)

cheek_left = c.create_oval(70, 180, 120, 230, outline='pink', fill='pink', state=HIDDEN)
```

16 Set up flags

Add two flag variables to the code to keep track of whether Screen Pet's eyes are crossed or its tongue is out. Type them just above the line that tells Screen Pet to start blinking, which you added to the main part of the code in Step 9.

```
c.eyes_crossed = False
c.tongue_out = False


root.after(1000, blink)
```

These are the flag variables for the pupils and the tongue.

17 Toggle the tongue

This function toggles Screen Pet's tongue between being out and in. Put the code shown below above the **show_happy()** function that you created in Step 11.

```
def toggle_tongue():
    if not c.tongue_out:
        c.itemconfigure(tongue_tip, state=NORMAL)
        c.itemconfigure(tongue_main, state=NORMAL)
        c.tongue_out = True
    else:
        c.itemconfigure(tongue_tip, state=HIDDEN)
        c.itemconfigure(tongue_main, state=HIDDEN)
        c.tongue_out = False

def show_happy(event):
```

The code checks to to see if the tongue is out already.

If the tongue isn't out, these lines make it visible.

This line sets a flag variable saying the tongue is now out.

The tongue is already out (**else**).

This line sets a flag variable saying the tongue isn't out.

These lines hide the tongue again.

EXPERT TIPS

Using flag variables

Flag variables help you keep track of something in your program that can be in one of two states. When you change the state, you update the flag. The "Engaged / Vacant" sign on a toilet door is a flag – you set it to "Engaged" when you lock the door and back to "Vacant" when you unlock it.

```
    root.after(3000, blink)


def toggle_pupils():
    if not c.eyes_crossed:
        c.move(pupil_left, 10, -5)
        c.move(pupil_right, -10, -5)
        c.eyes_crossed = True
    else:
        c.move(pupil_left, -10, 5)
        c.move(pupil_right, 10, 5)
        c.eyes_crossed = False
```

The code checks to see if the eyes are crossed already.

If the pupils aren't crossed, this line moves them in.

This line sets a flag variable saying the eyes are crossed.

The eyes are already crossed (`else`).

These lines move the pupils back to normal.

This line sets a flag saying the eyes aren't crossed.

18 Toggle the pupils

For the cross-eyed look, the pupils need to point inwards. This `toggle_pupils()` function will switch Screen Pet's pupils between pointing inwards and looking normal. Type it below the `blink()` function you added in Step 8.

19 Coordinate the cheekiness

Now create a function to get Screen Pet to stick its tongue out and cross its eyes at the same time. Type this code under the `toggle_tongue()` function you added in Step 17. Use the `root.after()` function to make Screen Pet go back to normal after 1 second (1,000 milliseconds), like you did in `blink()`.

Don't forget to save your work.

```
def cheeky(event):
    toggle_tongue()
    toggle_pupils()
    hide_happy(event)
    root.after(1000, toggle_tongue)
    root.after(1000, toggle_pupils)
    return
```

Stick the tongue out.

Cross the pupils.

Hide the happy face.

Put the tongue back in after 1,000 milliseconds.

Uncross the pupils after 1,000 milliseconds.

20 Link double-clicks to cheekiness

To trigger Screen Pet's cheeky expression, link any double-click event to the `cheeky()` function. Put this new line just below the line you added in Step 14 to hide Screen Pet's happy face. Run the code and double-click to see the cheekiness!

```
c.bind('<Motion>', show_happy)
c.bind('<Leave>', hide_happy)
c.bind('<Double-1>', cheeky)
```

`<Double-1>` is `Tkinter`'s name for a double-click in the window with the mouse.

Sad pet

Finally, make Screen Pet notice if you don't pay any attention to it. After nearly a minute without being stroked, your poor, neglected pet will show its sad face!

21 **Set up a happiness level**

Put this line of code just above the flag variables you added to the main part of the program in Step 16. It creates a happiness level for Screen Pet and sets the level at 10 when you run the program and draw the pet.

```
c.happy_level = 10
c.eyes_crossed = False
```

Screen Pet starts with a happiness level of 10.

22 **Create a new command**

Type this line below the command you added in Step 9 that starts Screen Pet blinking. It tells `mainloop()` to call the function `sad()`, which you'll add in Step 23, after 5 seconds (5,000 milliseconds).

```
root.after(1000, blink)
root.after(5000, sad)
root.mainloop()
```

23 **Write a sad function**

Add this function, `sad()`, beneath `hide_happy()`. It checks to see if `c.happy_level` is 0 yet. If it is, it changes Screen Pet's expression to a sad one. If it's not, it subtracts 1 from `c.happy_level`. Like `blink()`, it reminds `mainloop()` to call it again after 5 seconds.

```
def sad():
    if c.happy_level == 0:
        c.itemconfigure(mouth_happy, state=HIDDEN)
        c.itemconfigure(mouth_normal, state=HIDDEN)
        c.itemconfigure(mouth_sad, state=NORMAL)
    else:
        c.happy_level -= 1
    root.after(5000, sad)
```

This line checks to see if the value of `c.happy_level` is 0.

If `c.happy_level` equals 0, the code hides the happy and normal expressions.

This line sets Screen Pet's expression to sad.

The value of `c.happy_level` is greater than 0 (`else`).

Subtract 1 from the value of `c.happy_level`.

Call `sad()` again after 5,000 milliseconds.

24 **Cheer up, Screen Pet!**

Is there any way to stop Screen Pet from getting sad? Or cheer it up when it's miserable? Luckily there is – you just click into its window and stroke it. Add this line of code to the **show_happy()** function you wrote in Step 11. Now the function will reset the value of the variable **c.happy_level** back to 10 and make Screen Pet show its happy face again. Run the code to see your pet get sad, then cheer it up by stroking it.

Don't forget to save your work.

```
    c.itemconfigure(mouth_normal, state = HIDDEN)
    c.itemconfigure(mouth_sad, state = HIDDEN)
    c.happy_level = 10
return
```

This line puts the happiness level back up to 10.

Hacks and tweaks

Is Screen Pet your ideal pet now? If not, you can change the way it behaves or add some extra features! Here are a few ideas for personalizing your Screen Pet.

Be friendly, not cheeky

Maybe you'd rather not have a cheeky pet? Get Screen Pet to give you a friendly wink instead of making a rude face when you double-click on it.

EXPERT TIPS

Extra happiness

It might be distracting if you have to keep stroking and tickling Screen Pet while you're doing your homework. To make it sad less often, set the value of **c.happy_level** to a higher number at the start.

Increase this number.

```
c.happy_level = 10
c.eyes_crossed = False
```

1 Add this function underneath the **blink()** function. It's similar to the **blink()** code, but it will only toggle one eye.

```
def toggle_left_eye():
    current_color = c.itemcget(eye_left, 'fill')
    new_color = c.body_color if current_color == 'white'   else 'white'
    current_state = c.itemcget(pupil_left, 'state')
    new_state = NORMAL if current_state == HIDDEN else HIDDEN
    c.itemconfigure(pupil_left, state=new_state)
    c.itemconfigure(eye_left, fill=new_color)
```

2 The next function closes and opens the left eye once to make Screen Pet wink. Type it below `toggle_left_eye()`.

```
def wink(event):
    toggle_left_eye()
    root.after(250, toggle_left_eye)
```

3 Remember to change the command that binds the double-click event (`<Double-1>`) to `wink()` instead of `cheeky()` in the main part of the program.

```
c.bind('<Double-1>', wink)
```

Change `cheeky` to `wink` here.

Rainbow pets

It's easy to make Screen Pet a different colour by changing the value of `c.body_color`. If you can't decide what colour to choose, you can add a function that keeps changing Screen Pet's colour nonstop!

1 First add a line to import Python's `random` module. Put it under the line that loads the project's `Tkinter` features.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas
import random
```

2 Now type a new function, `change_color()`, just above the main part of the code. It picks a new value for `c.body_color` from the list `pet_colors`. Then it redraws Screen Pet's body using the new colour. Because it uses `random.choice`, you can never be sure what colour the pet will be next!

List of possible colours for Screen Pet

This line chooses another colour from the list at random.

```
def change_color():
    pet_colors = ['SkyBlue1', 'tomato', 'yellow', 'purple', 'green', 'orange']
    c.body_color = random.choice(pet_colors)]
    c.itemconfigure(body, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(ear_left, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(ear_right, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(foot_left, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(foot_right, outline=c.body_color, fill=c.body_color)
    root.after(5000, change_color)
```

These lines set Screen Pet's body, feet, and ears to the new colour.

The program calls `change_color()` again after 5,000 milliseconds (5 seconds).

3 Finally, add this just above the last line in the main part of the program to get `mainloop()` to call `change_color()` 5 seconds (5,000 milliseconds) after the program starts.

```
root.after(5000, change_color)
```

Your pet will begin changing colour 5 seconds after the program starts.

4 You might want to alter the values in the code so that Screen Pet changes colour less rapidly. You could also change the colours in the list to ones you like better, or add extra colours.

Feed me!

Pets need food, as well as stroking and tickling. Can you figure out ways to feed your pet and keep it healthy?

A growing Screen Pet needs plenty of healthy food to eat!

1 Perhaps try adding a "Feed me!" button to Screen Pet's window and a `feed()` function that's called when you click the button.

2 You could even make Screen Pet grow if you click "Feed me!" a certain number of times. This line of code makes its body bigger.

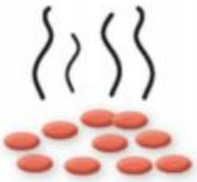
This code reshapes the oval that makes up Screen Pet's body.

```
body = c.create_oval(15, 20, 395, 350, outline=c.body_color, fill=c.body_color)
```

3 Then try writing some code so that your pet's body shrinks back to its original size again if it doesn't get enough food.

▷ **Clean that up!**

The problem with feeding Screen Pet is that it will need to poo as well! Write some code that makes it poo a while after you feed it. Then add a "Clean up" button. Clicking "Clean up" should call a handler function that removes the poo.



EXPERT TIPS

A bigger window

If you add buttons or other extra features to Screen Pet's window, it might get a bit crowded and uncomfortable for your pet. If so, you can enlarge the `Tkinter` window. To do this, change the values for width and height in the command that creates the canvas at the start of the main program.