

Mission 7: Personal Billboard

Student Workbook





Mission 7: Personal Billboard ✓

Show images that match your current mood, and learn about Python lists.

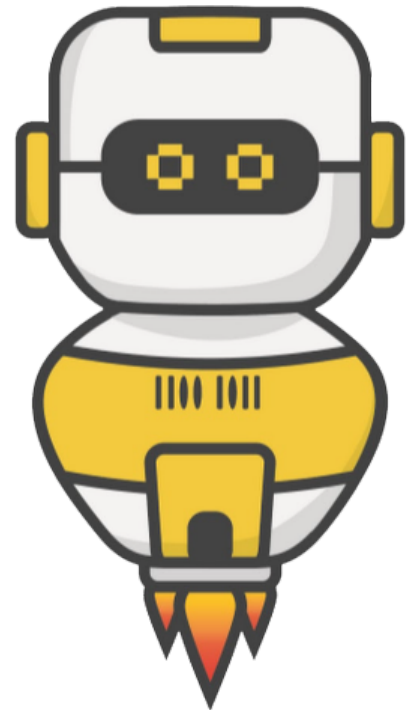
Let's go to the next level

Now that your programs can keep going with a loop, and you can control input with the buttons, it is time to handle more data.

Have you seen wearable devices – things like hats, belts or shirts that display a message or image? You can create one with CodeX!

Go to the Mission 7 Log and fill out the Pre-Mission preparation.

- If you could show what you like or your mood by displaying something, what would you display? (example: a color, an image, a slogan, etc.)
- What type of clothing would you display your message on?




Mission 7: Personal Billboard

In this project you'll use the CodeX display and buttons to make a *billboard* that shows others how you're feeling, a fun picture, or a short message.

On battery power, you could make the CodeX into a *wearable* electronic **badge** or a **portable sign** for a wall or desk!



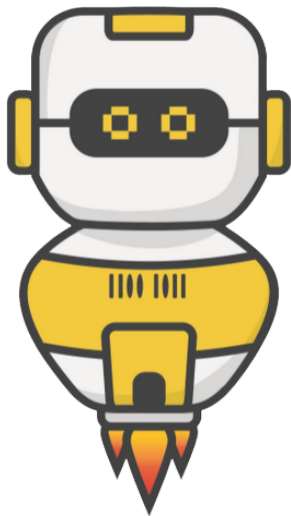
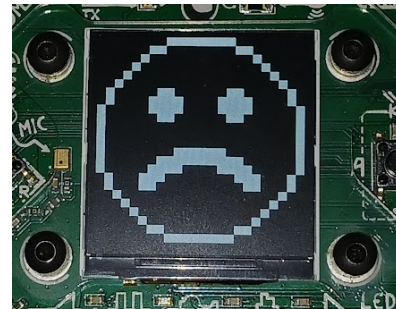
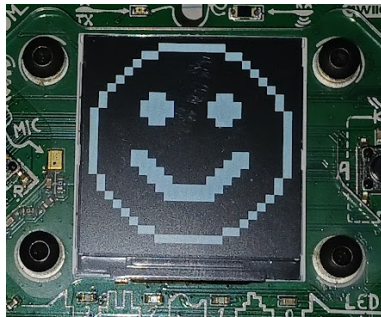
Mission 7: Get started

- Go to <https://sim.firialabs.com/> and log in.
- Go to Mission 7 
- Click **NEXT** and start Mission 7.

Objective #1: Image selector

The CodeX has several built-in images. You have used them since Mission 2. You learned about using buttons for input in Mission 6.

- Start this project by writing code that will:
 - Display the HAPPY face when BTN_L is pressed
 - Display the SAD face when BTN_R is pressed



DO THIS:

- Start a new file named **Billboard**
- Import codex
- Use a **while True:** loop
- Show pics.HAPPY if BTN_L was pressed
- Show pics.SAD if BTN_R was pressed
 - Use CodeTrek if you need help

```
from codex import *  
  
while True:  
    if buttons.was_pressed(BTN_L):  
        display.show(pics.HAPPY)  
  
    if #.. fill in the rest of the code
```

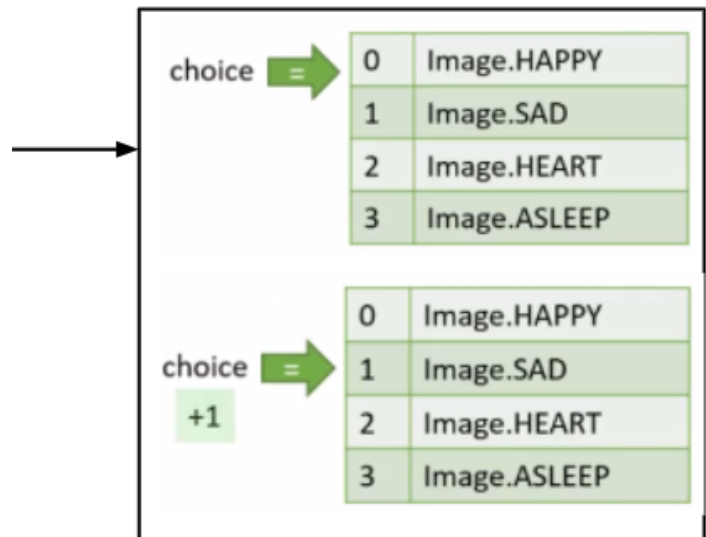
Objective #2: Select more images

You will use the CodeX to display your mood, so you need more than two pictures!

- You will still use the LEFT and RIGHT buttons to scroll through the pictures
- So you need some way to keep track of which picture to display
- You will use the variable **choice** to keep track of which image to display, and update **choice** with the buttons

You can use a number to keep track of the images like this:

A number like this is called an **index**. It is like using your finger to point to the image!



Objective #2: Select more images

To compare a number to a specific value, use `==`

- `choice == 1`

Use this comparison in an if statement to display an image

- Use an if statement for each picture
- You will have 4 additional if statements
- Use HAPPY, SAD, and two more pictures

Concept: Comparison operator

`=` assigns a value
`count = 1`

`==` compares two values to see if they are the same
`if choice == 1`

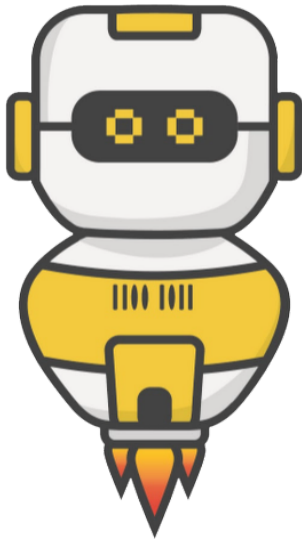
Comparison operators:

Greater than `>`
Less than `<`
Greater than or equal to `>=`
Less than or equal to `<=`
Equal to `==`
Not equal to `!=`

Built-in images you can use:

- `pics.HEART`
- `pics.HEART_SMALL`
- `pics.MUSIC`
- `pics.HAPPY`
- `pics.SAD`
- `pics.SURPRISED`
- `pics.ASLEEP`
- `pics.TARGET`
- `pics.TSHIRT`
- `pics.PLANE`
- `pics.HOUSE`
- `pics.TIARA`

Objective #2: Select more images

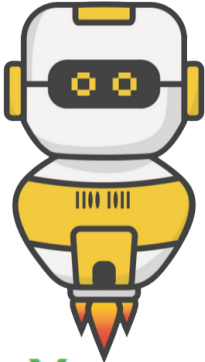


DO THIS:

- Go to your Mission Log and answer the questions about index and comparison operators
- Define the variable choice and assign it the value 0
- Write an if statement to display HAPPY (if choice == 0:)
- Write an if statement to display SAD (if choice == 1:)
- Write an if statement to display another pic (if choice == 2:)
- Write an if statement to display another pic (if choice == 3:)
- Change the **if buttons.was_pressed(BTN_R)** code to increment choice (**choice = choice + 1**)

Try to do the code on your own, and then check your work with the next page.

Objective #2: Select more images



Your code should look like this:

The last two pictures will be the ones you chose.

BTN_L isn't changed

BTN_R increments choice

```
from codex import *

choice = 0

while True:
    if choice == 0:
        display.show(pics.HAPPY)

    if choice == 1:
        display.show(pics.SAD)

    if choice == 2:
        display.show(pics.SURPRISED)

    if choice == 3:
        display.show(pics.ASLEEP)

    if buttons.was_pressed(BTN_L):
        display.show(pics.HAPPY)

    if buttons.was_pressed(BTN_R):
        choice = choice + 1
```


Objective #3: Scroll both directions

In Mission 6, you learned about increment and decrement

- Increment:
 - Increase the value of a variable by a set amount
 - Example: `num = num + 1`
- Decrement:
 - Decrease the value of a variable by a set amount
 - Example: `num = num - 1`

You will change the code for `BTN_L` to decrement choice so you can scroll the opposite way.

Another awesome feature of the debugger is that you can watch your variables and track their values while the code is running.

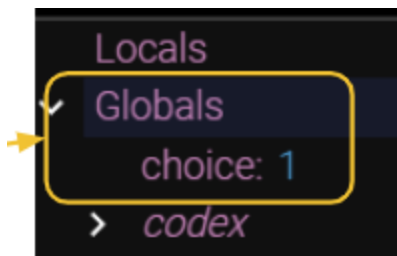
- Start the debugger



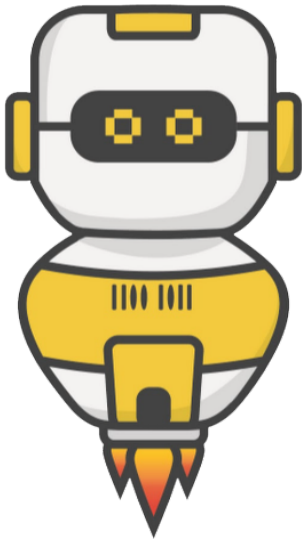
- Open the console panel



- Watch the variables as you step through the code

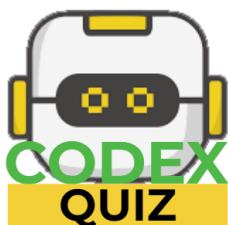
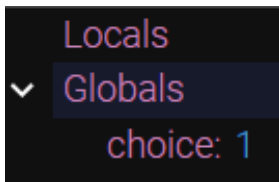
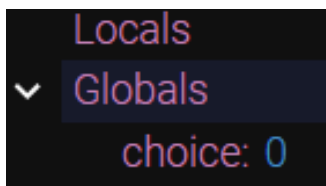
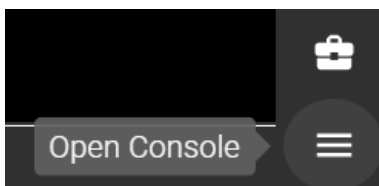


Objective #3: Scroll both directions



DO THIS:

- Go to your Mission Log and review “increment” and “decrement” from Mission 6
- Change the code for BTN_L to decrement **choice** by 1
- Start the debugger
- Open the console panel
- Use the **Step In** button to run the code.
 - Click several times, and then press BTN_R. Check the value of choice.
 - Click several more times, and then press either BTN_R or BTN_L. Check the value of choice.
 - Continue as long as you want, until you understand the code.
 - Then STOP the code.



Mission Quiz: Billboard checkpoint

Test your skills by **taking the quiz.**

Objective #4: Wrap around

You probably noticed that if you keep pressing BTN_R, it stops at the last image.

- The value of **choice** keeps increasing, but the image stays the same.
- Also, pressing BTN_L many times keeps the first image on the screen.
- The value of **choice** decreases, but the image stays the same.
- There are no if statements for **choice == 4** or **choice == -1**
- So the last image displayed remains on the screen



Can you improve the program and avoid this problem?

Objective #4: Wrap around

Instead of adding more images or **if statements**, make the value of **choice** wrap-around to the first value.

- Use an **if statement** to know when to wrap around.
- Use a comparison operator.

```
if buttons.was_pressed(BTN_R):  
    choice = choice + 1  
    if choice > 3:  
        choice = 0
```

NOTE: you are assigning a value, so use = and not ==

You can have an if statement inside an if statement -- just be careful with the indenting

The second if statement causes the value of choice to wrap-around, and start over.

- The last index is 3
- The first index is 0

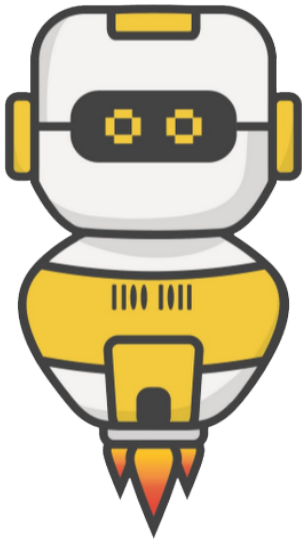
```
if buttons.was_pressed(BTN_R):  
    choice = choice + 1  
    if choice > 3:  
        choice = 0
```

What will the if statement look like to wrap-around BTN_L?

- The value of choice will need to be the LAST index if less than 0.

0	Image.HAPPY
1	Image.SAD
2	Image.HEART
3	Image.ASLEEP

Objective #4: Wrap around



DO THIS:

- Go to your Mission Log and write down what you think the code should look like to wrap-around the value of choice in BTN_L. Indent the heartbeat code

Modify your code

- Add an if statement to BTN_R so the value of choice wraps around
- Add an if statement to BTN_L so the value of choice wraps around
- Test your code
- Then stop the code

```
from codex import *

choice = 0

while True:
    if choice == 0:
        display.show(pics.HAPPY)

    if choice == 1:
        display.show(pics.SAD)

    if choice == 2:
        display.show(pics.SURPRISED)

    if choice == 3:
        display.show(pics.ASLEEP)

    if buttons.was_pressed(BTN_L):
        choice = choice - 1
        if choice < 0:
            # Add your code here

    if buttons.was_pressed(BTN_R):
        choice = choice + 1
        if choice > 3:
            choice = 0
```

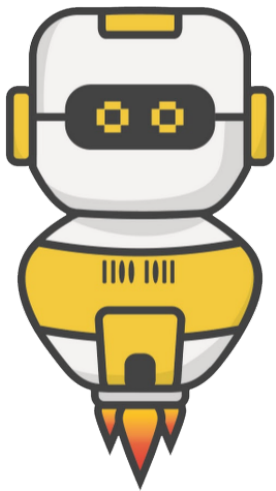
Objective #5: Image list

Four pictures is nice, but what if you want to add more?


That is a lot of typing!

- Every new image needs an if statement
- Your code can get very long very quickly!

Instead, you can make a list!



DO THIS:

- Click on  `list` in the instructions panel
- Go to your Mission Log and answer the questions about `list`

Objective #5: Image list

- A list is a type!
- Now you know six data types:
 - Integer
 - CodeX image
 - String
 - Boolean
 - Float
 - List



A note about a list and the index of each item

- The order of the items in the list is important
- Each item has an index (number) assigned
- The first index is always 0
- The last index is always 1 less than the number of items

INDEX	ITEM
0	HAPPY
1	SAD
2	SURPRISED
3	ASLEEP
4	TIARA
5	PLANE

NOTE: This list has 6 items, so the index is 0, 1, 2, 3, 4, and 5

Objective #5: Image list

Things you can do with a list:

- Create a list (use [])

```
my_list = [pics.HAPPY, pics.SAD, pics.SURPRISED, pics.ASLEEP, pics.TIARA]
```

- Access an item in the list (use [])

```
my_image = my_list[1]
```

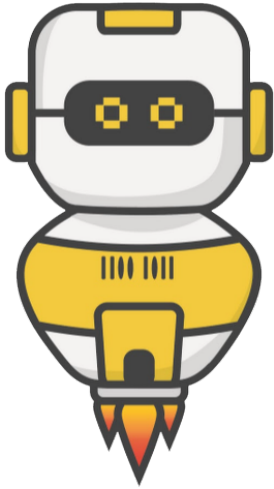
```
my_image = pics.SAD
```

```
my_image = my_list[choice]
```

```
my_image = whatever image  
is at the current  
value of choice
```

INDEX	ITEM
0	HAPPY
1	SAD
2	SURPRISED
3	ASLEEP
4	TIARA

Objective #5: Image list



DO THIS:

- Add a list to your code
 - Use the same four images
- Change the code to access the list
 - Add two lines of code to access the list using choice for the index
 - Delete the four if statements that displayed the images

Leave the if statements for BTN_L and BTN_R

```
from codex import *

choice = 0
my_list = [pics.HAPPY, pics.SAD, pics.SURPRISED, pics.ASLEEP]

while True:
    my_image = my_list[choice]
    display.show(my_image)

    if buttons.was_pressed(BTN_L):
        choice = choice - 1
        if choice < 0:
            choice = 3

    if buttons.was_pressed(BTN_R):
        choice = choice + 1
        if choice > 3:
            choice = 0
```

Objective #6: No magic numbers

- With four images in your list, the index numbers are
 - 0, 1, 2, 3
- You use these numbers for wrap-around

```
if choice < 0:  
    choice = 3
```

```
if choice > 3:  
    choice = 0
```

- If you added another image, the last index would be 4, not 3.
- You would have to change 3 to 4 everywhere in the code!
- These literals are called “magic numbers”

- Magic numbers make the code harder to maintain, and harder to read and understand.
- The magic number in this program is the last index of the list
- So ...
- Use a built-in function!

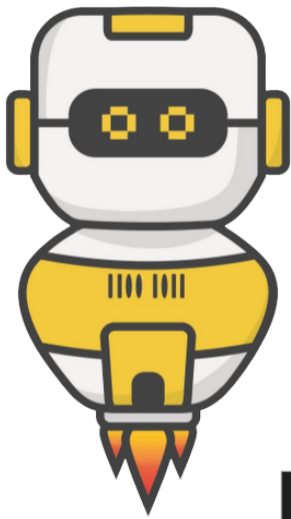
```
len(my_list)
```

This code will give the length of the list, which is the number of items in the list.

- **Remember:** the last index is always one less than the number of items

```
LAST_INDEX = len(my_list) - 1
```

Objective #6: No magic numbers



Now you can add more images

DO THIS:

- Add another image to your list
 - A list of images is on slide 9
- Create a variable for **LAST_INDEX**

You can choose the image you want to add

```
choice = 0
my_list = [pics.HAPPY, pics.SAD, pics.SURPRISED, pics.ASLEEP, pics.TIARA]
LAST_INDEX = len(my_list) - 1
```

- Use the **LAST_INDEX** variable in the code:

```
if buttons.was_pressed(BTN_L):
    choice = choice - 1
    if choice < 0:
        choice = LAST_INDEX

if buttons.was_pressed(BTN_R):
    choice = choice + 1
    if choice > LAST_INDEX:
        choice = 0
```



Mission Quiz: List len

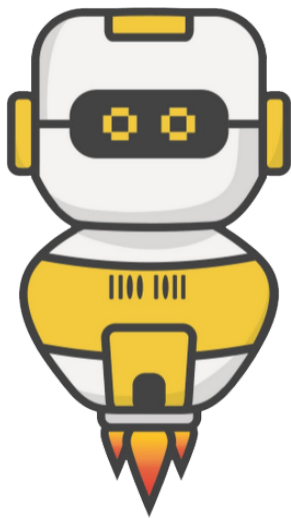
Test your skills by taking the quiz.

Objective #7: Text time!

Images are expressive ... but text can say so much more!

- You can use a string variable to create a message or slogan
- Remember: a string data type uses quotation marks: “..”
 - `my_message = “Meh”`
 - `my_message = “Having a great day”`
- You also include a string message in your list
 - `display.show(my_message)` will display the text string

```
my_list = ["Ahoy", pics.HAPPY, pics.SAD, pics.SURPRISED, pics.ASLEEP, pics.TIARA]
```



DO THIS:

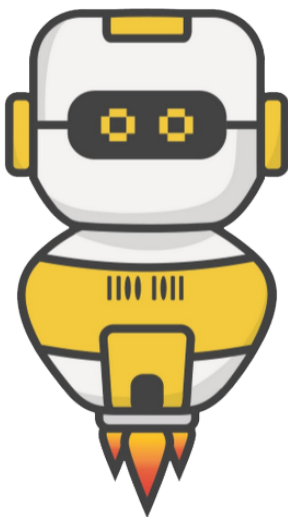
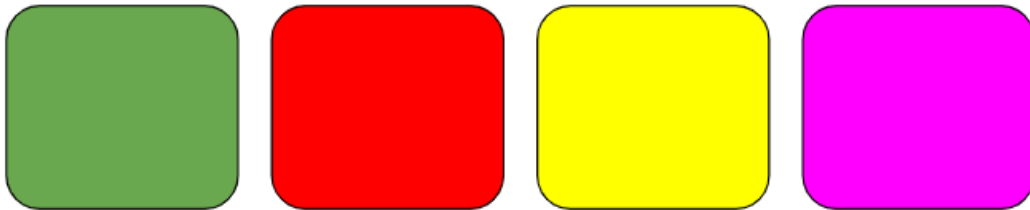
- Add a text string to your list
 - **OPTIONAL:** Your list can look like this to make it easier to read.

```
choice = 0
my_list = ["Ahoy",
           pics.HAPPY,
           pics.SAD,
           pics.SURPRISED,
           pics.ASLEEP,
           pics.TIARA]
LAST_INDEX = len(my_list) - 1
```

Objective #8: Green with envy

What if you're neither HAPPY nor SAD? ...and text just isn't describing you?

- Sometimes you just need a *color*.
- Maybe you are GREEN with envy!
- Wouldn't it be cool to fill the display with a color?
- Try it out!



DO THIS:

- Add GREEN to the list
- Run the program
- Get an error?
- Find out why in the next objective

```
choice = 0
my_list = [GREEN,
           "Ahoy",
           pics.HAPPY,
           pics.SAD,
           pics.SURPRISED,
           pics.ASLEEP,
           pics.TIARA]
LAST_INDEX = len(my_list) - 1
```

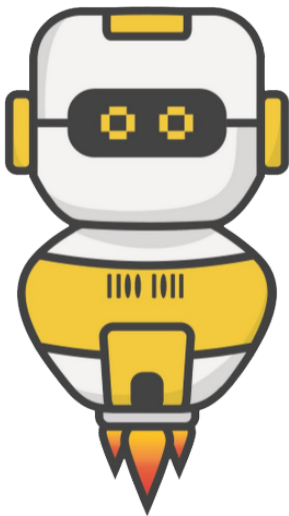
✘ Billboard 1 of 1 problem

TypeError: Show requires either a bitmap or a string

Objective #9: Fill 'er up

GREEN isn't an image or a string. What type is it?

- Colors in the `codex` library are actually tuples!
- A **tuple** is like a list that can't be changed.
- CodeX color tuples have three integer values: `(red, green, blue)`
- You learned about RGB values in Mission 3
- What do you think the tuple for GREEN is?



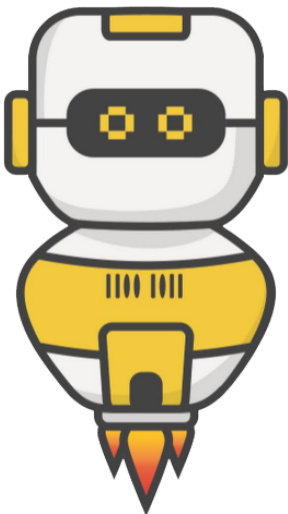
DO THIS:

- Go to the Mission Log and write your guess for the RGB tuple of GREEN

Objective #9: Fill 'er up

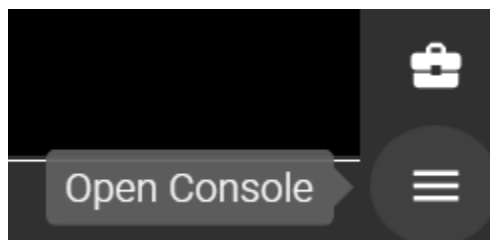
`display.show()` doesn't work with colors, but `display.fill()` does!

- You just have to know when to use `display.show()` and when to use `display.fill()`
- You need to check for the **type**
- You can use the console panel to help you



DO THIS:

- Open the console panel. You can type commands directly into the console.



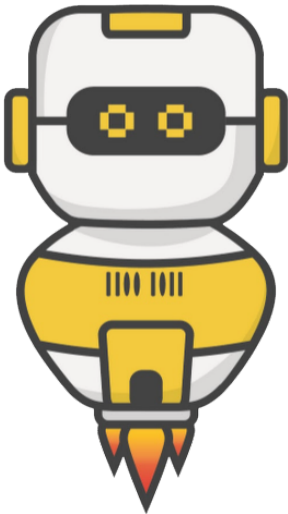
- Check the type of several values:
 - `type(7)` -> 'int'
 - `type(1.15)`
 - `type(True)`
 - `type([1, 2, 3])`
- The type is shown like this:
- Now get the type of a color
 - `type((0, 255, 0))`

```
>>> type(7)
<class 'int'>
>>> type(1.15)
```

Objective #9: Fill 'er up

- The type of a color is 'tuple'
- You can use this information in your code
- If the type is 'tuple', use `display.fill()`.
Else use `display.show()`

```
>>> type(7)
<class 'int'>
>>> type(1.15)
<class 'float'>
>>> type(True)
<class 'bool'>
>>> type([1, 2, 3])
<class 'list'>
>>> type((0, 255, 0))
→ <class 'tuple'>
>>>
```

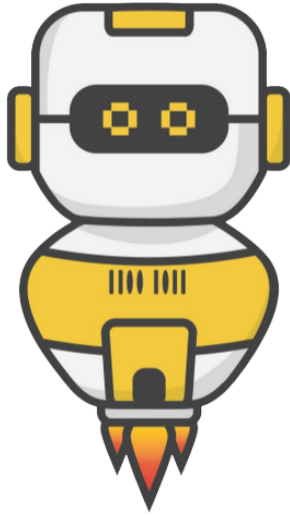


DO THIS:

- Add an if statement to the code that compares the current `my_image` to a tuple.
- If it is, use `display.fill()`.
- Else use `display.show()`
- Run the code. You should get colors, text and images!

```
while True:
    my_image = my_list[choice]
    if type(my_image) == tuple:
        display.fill(my_image)
    else:
        display.show(my_image)
```


Objective #9: Fill 'er up



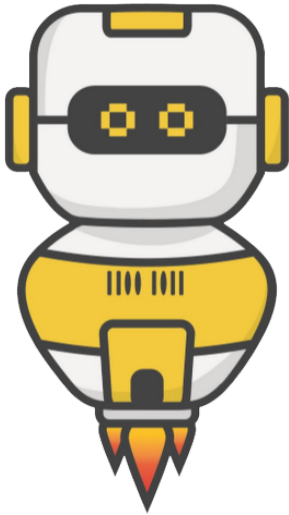
DO THIS:

- Add more colors, text or images to your list.
- Run the code.
- No matter how many items you have, the code should work without making any other changes.
- Pretty cool, Right!
- Now you can display your mood by stopping on the color, text, or image that represents you.

```
my_list = [GREEN,  
           "Ahoy",  
           pics.HAPPY,  
           pics.SAD,  
           RED,  
           pics.SURPRISED,  
           "Having a great day",  
           pics.ASLEEP,  
           PINK,  
           pics.TIARA,  
           "Meh",  
           pics.TARGET]
```

Mission Complete

You have completed the seventh mission.



Do this:

- Read your “Completed Mission” message
- Complete your Mission 7 Log
 - Post-Mission Reflection
- Get ready for your next mission!

Wait! Before you go ... Clear the CodeX

Go to FILE -- BROWSE FILES

Select the “Clear” file and open it

Run the program to clear the CodeX

Okay. Now you can go.