Connecting to GitHub

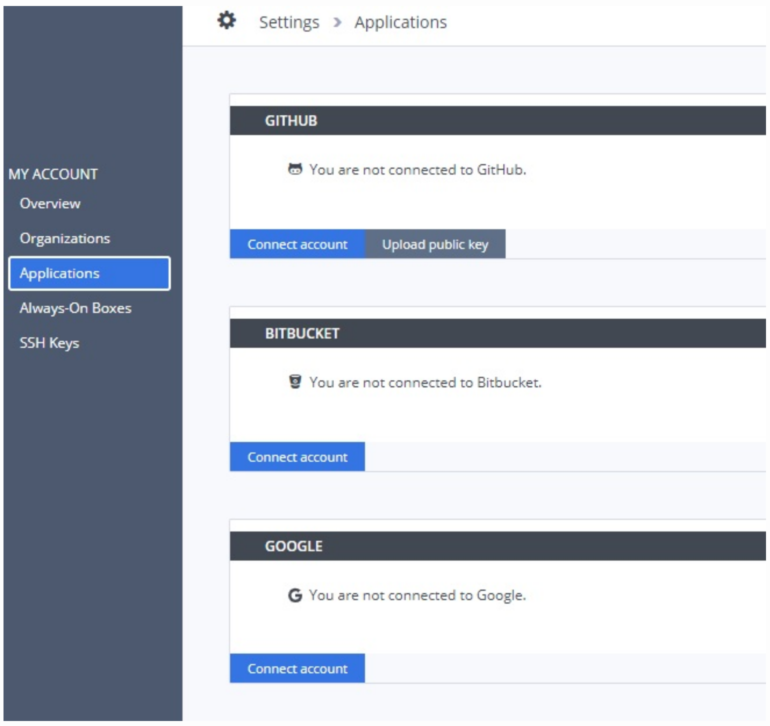
Connecting to GitHub

We are going to build a Python package that you can upload to your GitHub repository. If you do not yet have an account, please create one now. We are going to clone a repository that will contain the code for your Python package.

Connecting GitHub and Codio

You need to connect GitHub to your Codio account. This only needs to be done one time.

- In your Codio account, click on your username
- Click on **Applications**



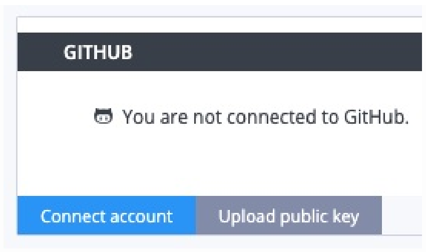
You are not connected to GitHub.

You are not connected to Bitbucket.

You are not connected to Google.

The image depicts all of the services you can connect to your Codio account. GitHub is the topmost option.

- Under GitHub, click on **Connect account**



You are not connected to GitHub.

The image depicts the buttons associated with connecting your GitHub account to Codio. The first button says to connect GitHub. The second button says to upload a public key.

- You will be using an SSH connection, so you need to click on **Upload public key**

Fork the Repository

- Go to the pixel_art repository. This repository is the starting point for your project.
- Click on the "Fork" button in the top-right corner.
- In your forked repository, click the green "Code" button.
- Copy the SSH information. It should look something like this:

```
git@github.com:<your_github_username>/pixel_art.git
```

info

Important

If you do not use the SSH information, you will have to provide your username and Personal Access Token (PAT) to GitHub each time you push or pull from the repository. See this documentation for setting up a PAT for your GitHub account.

In the Terminal

- Clone the repository by using the `git clone` command and the copied SSH information. Your command should look something like this:

```
git clone git@github.com:<your_github_username>/pixel_art.git
```

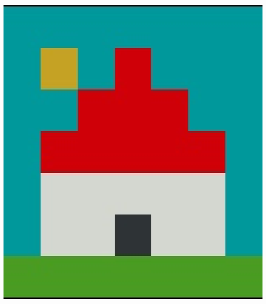
- You should see a `pixel_art` directory appear in the file tree.

You are now ready to start the project.

Lab - Pixel Art

Pixel Art Package

Pixel art is any piece of art built with the deliberate placement of pixels to create a larger image. The image below is an example of simple pixel art.



The image depicts a house, sun, and grass done in the style of pixel art.

We are going to create `pixel_art`, a Python package that allows users to create pixel art and then print it to the terminal. The finished package will include the following items:

- `examples.py` - module that has several pre-made examples the user can display
- `pixels.py` - module that is the interface for building pixel art
- `__init__.py` - initialization file for the package
- `setup.py` - metadata for the package including dependencies
- `pixel_art` wheel - built distribution we will install to test the package
- Post to GitHub - keep a copy of the package in your own GitHub repository

Initializing the Package

Let's start by initializing the two modules for the `pixel_art` package. Copy this code into the `__init__.py` file.

```
import pixel_art.examples, pixel_art.pixels
```

Users can simply import `pixel_art` and both modules will be ready to go.

Pixel Art Module 1

Setup

Our pixel art package depends on the `colorama` package, which allows you to change the foreground, background, and text colors in the terminal. The “pixels” are nothing more than two spaces (`'  '`) with a background color. Using a virtual environment, install `colorama` in the terminal.

```
cd pixel_art/pixel_art
python3 -m venv pa-env
source pa-env/bin/activate
python3 -m pip install colorama
```

Start by importing from the `colorama` package. In particular, we want `init` to initialize our pixel art and `Back` which lets us set the background color. By setting `autoreset` to true, the colors of the terminal will return to normal after the pixel art image is displayed.

```
from colorama import init, Back, Style

init(autoreset=True)
```

Next, we need to create our “pixels”. Colorama allows for only eight colors. We are going to use them all. Create a variable that corresponds to each color. Set its value to `Back` and the color name (the color name should be all caps) concatenated to a string with two spaces.

```
BLACK = Back.BLACK + '  '
RED = Back.RED + '  '
GREEN = Back.GREEN + '  '
YELLOW = Back.YELLOW + '  '
BLUE = Back.BLUE + '  '
MAGENTA = Back.MAGENTA + '  '
CYAN = Back.CYAN + '  '
WHITE = Back.WHITE + '  '
```

Users are going to pass a string representation of the color they want. So we need a way to convert the string into the Colorama objects we defined above. Use a dictionary that has strings for each color as the key and their

Colorama object as the value.

```
COLORS_DICT = {
  'black' : BLACK,
  'red' : RED,
  'green' : GREEN,
  'yellow' : YELLOW,
  'blue' : BLUE,
  'magenta' : MAGENTA,
  'cyan' : CYAN,
  'white' : WHITE
}
```

Image Class

The `Image` class is the main vehicle for creating pixel art. Create a constructor that takes a width and height for the pixel art image. If no arguments are passed, default to 7 and 7. The `width` and `height` attributes are assigned their respective parameters. The `img` attribute takes its value from the `init_list` method.

```
class Image():
  def __init__(self, w=7, h=7):
    self.width = w
    self.height = h
    self.img = self.init_list()
```

The pixel art is nothing more than a two-dimensional list. Each element in the list will be a “pixel”. In `init_list()`, create an empty list. Create a loop that runs as many times as the `height` attribute. Append an empty list to `lst` each time the loop runs. Return `lst` which becomes the initial value for the `img` attribute.

```
  def init_list(self):
    lst = []
    for row in range(self.height):
      lst.append([])
    return lst
```

Pixel Art Module 2

Displaying Pixels

Before continuing, change into the `pixel_art` directory and activate the virtual environment.

```
cd pixel_art/pixel_art
source pa-env/bin/activate
```

Add the following methods to the `Image` class. The `display()` method is going to draw everything to the terminal. Iterate over the `img` attribute. Take each inner-list (`row`) and pass it to the `display_row()` method.

```
  def display(self):
    for row in self.img:
      self.display_row(row)
```

In Python, if you print a list it will print the square brackets with each element separated by a comma. We want just the pixels. Moreover, we want each pixel in the list to appear next to one another. As you iterate over the list, print the pixel **without** the newline character (`end=''`). After the loop runs, print a newline character. This way you are ready for the next row of pixels.

```
  def display_row(self, row):
    for pixel in row:
      print(pixel, end='')
    print()
```

Converting Colors

The `convert_color()` method takes a string that represents a color and returns the associated Colorama object. If the string is not found in `COLORS_DICT` then return a black pixel. `convert_color()` is a helper method for adding pixels to the image.

```
  def convert_color(self, str_clr):
    return COLORS_DICT.get(str_clr, BLACK)
```

Adding Pixels

The last method in the `Image` class is `add_row()`. This method takes an integer as the first parameter. This represents the row of pixels (first row, second row, etc.). After the integer comes an indefinite number of strings. That is why `*args` is used as the parameter; we don't know how many parameters there will be. If the `width` attribute is 7, then there should be eight parameters. The first is the integer designating the row, followed by seven strings for the colors of the pixels.

If the user wants to add a row of pixels all with the same color, it would not be a good user experience if we made them type the same color again and again. If there are only two elements in `args` then the user passed an integer and a string. Use a comprehension to create a Colorama object with `convert_color()`. The comprehension should iterate as many times as the `width` attribute.

```
  def add_row(self, *args):
    if len(args) == 2:
      colors = [self.convert_color(args[1]) for i in
        range(self.width)]
      self.img[args[0]] = colors
```

If more than two parameters are passed, separate all of the colors from the integer by slicing `args` from the first element to the end of the list. Using a comprehension, transform each color string into its Colorama object with the `convert_color()` method. Finally, use the integer (`args[0]`) to update the row with the list of colors.

```
    else:
      color_params = args[1:]
      colors = [self.convert_color(arg) for arg in color_params]
      self.img[args[0]] = colors
```

▼ **Code**

Your code should look like this:

```
from colorama import init, Back, Style

init(autoreset=True)
```

```
BLACK = Back.BLACK + '  '
RED = Back.RED + '  '
GREEN = Back.GREEN + '  '
YELLOW = Back.YELLOW + '  '
BLUE = Back.BLUE + '  '
MAGENTA = Back.MAGENTA + '  '
CYAN = Back.CYAN + '  '
WHITE = Back.WHITE + '  '

COLORS_DICT = {
  'black' : BLACK,
  'red' : RED,
  'green' : GREEN,
  'yellow' : YELLOW,
  'blue' : BLUE,
  'magenta' : MAGENTA,
  'cyan' : CYAN,
  'white' : WHITE
}

class Image():
  def __init__(self, w=7, h=7):
    self.width = w
    self.height = h
    self.img = self.init_list()

  def init_list(self):
    lst = []
    for row in range(self.height):
      lst.append([])
    return lst

  def display(self):
    for row in self.img:
      self.display_row(row)

  def display_row(self, row):
    for pixel in row:
      print(pixel, end='')
    print()

  def convert_color(self, str_clr):
    return COLORS_DICT.get(str_clr, BLACK)

  def add_row(self, *args):
    if len(args) == 2:
      colors = [self.convert_color(args[1]) for i in
        range(self.width)]
      self.img[args[0]] = colors
    else:
      color_params = args[1:]
      colors = [self.convert_color(arg) for arg in color_params]
      self.img[args[0]] = colors
```

Examples Module

Setup

The purpose of the `examples` module is to provide the user with pre-made examples so they get an idea of what is possible with the `pixel_art` package. Start by changing into the `pixel_art` directory and activating the virtual environment.

```
cd pixel_art/pixel_art
source pa-env/bin/activate
```

In the **top** panel, import the `pixels` module as `pixels` so we can reference the color constants and the `Image` class for our examples.

```
import pixels
```

Example Colors

We need to convey the available colors to the user. We want a sample of the color as well as the string needed to invoke it. The `colors` function does not have any parameters. In an f-string, print each color constant from the `pixels` module followed by the color's name in string form. The `{pixels.Back.REST}` resets the background color so only the pixel is colored. The rest of the output should be in the normal terminal.

```python
def colors():
  print(f"{pixels.BLACK}{pixels.Back.RESET} - 'black'")
  print(f"{pixels.RED}{pixels.Back.RESET} - 'red'")
  print(f"{pixels.GREEN}{pixels.Back.RESET} - 'green'")
  print(f"{pixels.YELLOW}{pixels.Back.RESET} - 'yellow'")
  print(f"{pixels.BLUE}{pixels.Back.RESET} - 'blue'")
  print(f"{pixels.MAGENTA}{pixels.Back.RESET} - 'magenta'")
  print(f"{pixels.CYAN}{pixels.Back.RESET} - 'cyan'")
  print(f"{pixels.WHITE}{pixels.Back.RESET} - 'white'")
```

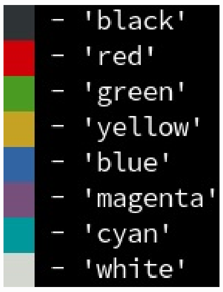
Add the code below to the testing section.

```
# test your code here
if __name__ == '__main__':
  colors()
```

Run the `examples` module to test the `colors` function in the **bottom** panel.

```
python3 examples.py
```

You should see the following output:



```
 - 'black'
 - 'red'
 - 'green'
 - 'yellow'
 - 'blue'
 - 'magenta'
 - 'cyan'
 - 'white'
```

The image depicts a sample pixel in the colors black, red, green, yellow, blue, magenta, cyan and white. Next to each pixel is the name of the color in quotes.

Example House

The default size of a pixel art image is seven pixels by seven pixels. The `house` function provides an example of pixel art with the default dimensions. Instantiate an `Image` object and add each row of pixels. Use the `display()` method to draw the image to the terminal

```
def house():
  img = pixels.Image()
  img.add_row(0, 'cyan')
  img.add_row(1, 'cyan', 'yellow', 'cyan', 'red', 'cyan',
        'cyan', 'cyan')
  img.add_row(2, 'cyan', 'cyan', 'red', 'red', 'red', 'cyan',
        'cyan')
  img.add_row(3, 'cyan', 'red', 'red', 'red', 'red', 'red',
        'cyan')
  img.add_row(4, 'cyan', 'white', 'white', 'white', 'white',
        'white', 'cyan')
  img.add_row(5, 'cyan', 'white', 'white', 'black', 'white',
        'white', 'cyan')
  img.add_row(6, 'green')
  img.display()
```

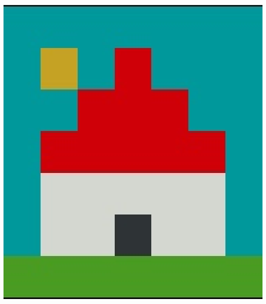
Add the code below to the testing section.

```
# test your code here
if __name__ == '__main__':
  house()
```

Run the `examples` module to test the `house` function in the **bottom** panel.

```
python3 examples.py
```

You should see the following output:



The image depicts a house, sun, and grass done in the style of pixel art.

Example Notes

The final example is musical notes. This image is done with custom dimensions so the image is more intricate. Create the `notes` function, which has no parameters. Instantiate an `Image` object with a width of 16 pixels and a height of 14 pixels. Add each row of pixels, and use `display()` to draw it to the terminal.

```
def notes():
  img = pixels.Image(16, 14)
  img.add_row(0, 'magenta')
  img.add_row(1, 'magenta', 'magenta', 'magenta', 'magenta',
               'magenta', 'magenta', 'black', 'black',
               'black', 'black', 'black', 'black',
               'black', 'black', 'black', 'magenta')
  img.add_row(2, 'magenta', 'magenta', 'magenta', 'magenta',
               'magenta', 'black', 'white', 'white',
               'white', 'white', 'white', 'white',
               'white', 'white', 'black', 'magenta')
  img.add_row(3, 'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'white', 'white',
               'white', 'white', 'white', 'white',
               'white', 'white', 'black', 'magenta')
  img.add_row(4, 'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'black',
               'black', 'black', 'black', 'black',
               'black', 'white', 'black', 'magenta')
  img.add_row(5, 'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'magenta',
               'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'magenta')
  img.add_row(6, 'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'magenta',
               'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'magenta')
  img.add_row(7, 'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'magenta',
               'magenta', 'magenta', 'magenta', 'magenta',
               'black', 'white', 'black', 'magenta')
  img.add_row(8, 'magenta', 'magenta', 'black', 'black',
               'black', 'white', 'black', 'magenta',
               'magenta', 'magenta', 'black', 'black',
               'black', 'white', 'black', 'magenta')
  img.add_row(9, 'magenta', 'black', 'white', 'white',
               'white', 'white', 'black', 'magenta',
               'magenta', 'black', 'white', 'white',
               'white', 'white', 'black', 'magenta')
  img.add_row(10, 'magenta', 'black', 'white', 'white',
                  'white', 'white', 'black', 'magenta',
                  'magenta', 'black', 'white', 'white',
                  'white', 'white', 'black', 'magenta')
  img.add_row(11, 'magenta', 'black', 'white', 'white',
                  'white', 'white', 'black', 'magenta',
                  'magenta', 'black', 'white', 'white',
                  'white', 'white', 'black', 'magenta')
  img.add_row(12, 'magenta', 'magenta', 'black', 'black',
                  'black', 'black', 'magenta', 'magenta',
                  'magenta', 'magenta', 'black', 'black',
                  'black', 'black', 'magenta', 'magenta')
  img.add_row(13, 'magenta')
  img.display()
```

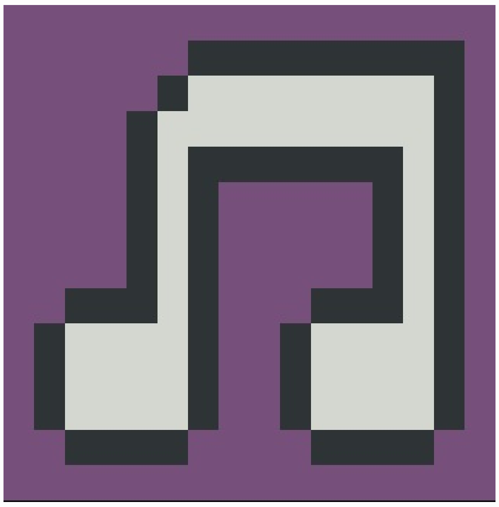
Add the code below to the testing section.

```
# test your code here
if __name__ == '__main__':
    notes()
```

Run the `examples` module to test the `notes` function in the **bottom** panel.

```
python3 examples.py
```

You should see the following output:



The image depicts a pixel art image that is 16 pixels by 14 pixels. Two, white musical notes are outlined in black and sit on a magenta background.

Once we have tested all of the example images, we need to change the import statement so everything works when we convert it to a wheel. Your import statement should now look like this:

```
import pixel_art.pixels as pixels
```

▼ **Code**

Your code should look like this:

```
import pixel_art.pixels as pixels

def colors():
  print(f"{pixels.BLACK}{pixels.Back.RESET} - 'black'")
  print(f"{pixels.RED}{pixels.Back.RESET} - 'red'")
  print(f"{pixels.GREEN}{pixels.Back.RESET} - 'green'")
  print(f"{pixels.YELLOW}{pixels.Back.RESET} - 'yellow'")
  print(f"{pixels.BLUE}{pixels.Back.RESET} - 'blue'")
  print(f"{pixels.MAGENTA}{pixels.Back.RESET} - 'magenta'")
  print(f"{pixels.CYAN}{pixels.Back.RESET} - 'cyan'")
  print(f"{pixels.WHITE}{pixels.Back.RESET} - 'white'")

def house():
  img = pixels.Image()
  img.add_row(0, 'cyan')
```

```
    img.add_row(1, 'cyan', 'yellow', 'cyan', 'red', 'cyan',
          'cyan', 'cyan')
    img.add_row(2, 'cyan', 'cyan', 'red', 'red', 'red', 'cyan',
          'cyan')
    img.add_row(3, 'cyan', 'red', 'red', 'red', 'red', 'red',
          'cyan')
    img.add_row(4, 'cyan', 'white', 'white', 'white', 'white',
          'white', 'cyan')
    img.add_row(5, 'cyan', 'white', 'white', 'black', 'white',
          'white', 'cyan')
    img.add_row(6, 'green')
    img.display()


  def notes():
    img = pixels.Image(16, 14)
    img.add_row(0, 'magenta')
    img.add_row(1, 'magenta', 'magenta', 'magenta', 'magenta',
                   'magenta', 'magenta', 'black', 'black',
                   'black', 'black', 'black', 'black',
                   'black', 'black', 'black', 'magenta')
    img.add_row(2, 'magenta', 'magenta', 'magenta', 'magenta',
                   'magenta', 'black', 'white', 'white',
                   'white', 'white', 'white', 'white',
                   'white', 'white', 'black', 'magenta')
    img.add_row(3, 'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'white', 'white',
                   'white', 'white', 'white', 'white',
                   'white', 'white', 'black', 'magenta')
    img.add_row(4, 'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'black',
                   'black', 'black', 'black', 'black',
                   'black', 'white', 'black', 'magenta')
    img.add_row(5, 'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'magenta',
                   'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'magenta')
    img.add_row(6, 'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'magenta',
                   'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'magenta')
    img.add_row(7, 'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'magenta',
                   'magenta', 'magenta', 'magenta', 'magenta',
                   'black', 'white', 'black', 'magenta')
    img.add_row(8, 'magenta', 'magenta', 'black', 'black',
                   'black', 'white', 'black', 'magenta',
                   'magenta', 'magenta', 'black', 'black',
                   'black', 'white', 'black', 'magenta')
    img.add_row(9, 'magenta', 'black', 'white', 'white',
                   'white', 'white', 'black', 'magenta',
                   'magenta', 'black', 'white', 'white',
                   'white', 'white', 'black', 'magenta')
    img.add_row(10, 'magenta', 'black', 'white', 'white',
                    'white', 'white', 'black', 'magenta',
                    'magenta', 'black', 'white', 'white',
                    'white', 'white', 'black', 'magenta')
    img.add_row(11, 'magenta', 'black', 'white', 'white',
                    'white', 'white', 'black', 'magenta',
                    'magenta', 'black', 'white', 'white',
                    'white', 'white', 'black', 'magenta')
    img.add_row(12, 'magenta', 'magenta', 'black', 'black',
                    'black', 'black', 'magenta', 'magenta',
                    'magenta', 'magenta', 'black', 'black',
                    'black', 'black', 'magenta', 'magenta')
    img.add_row(13, 'magenta')
    img.display()
```

Creating a Wheel

Setup File

Before we can build the wheel for this package, we need to create a `setup.py` file. This one will be a bit different than the one we previously made. This is because `pixel_art` depends on the `coloroma` package. We want users to install `pixel_art` and have `pip` automatically install `colorama` at the same time.

Add the appropriate metadata about the package. For the dependencies, add `install_requires` to the function call. This should be a list of strings for all of the required packages. Even though there is only one dependency it still needs to be a list.

```
from setuptools import setup

setup(
   name='pixel_art',
   version='0.1',
   description='Package for drawing pixel art to the terminal.',
   author='Your name here',
   packages=['pixel_art'],
   install_requires = ['colorama']
)
```

Building the Wheel

Now that the setup file is done, we are going to change to the `package` directory and then build a pure-Python wheel for version 3 of Python. Enter the following commands into the **bottom** panel and press `ENTER`.

```
cd pixel_art
python3 setup.py sdist bdist_wheel
```

After the build is complete, change into the `pixel_art/dist` directory and list its contents. You should see a `.whl` file for our package.

```
cd dist
ls
```

Testing the Package

Install the Wheel

Even though the `pixel_art` wheel is not uploaded to PyPI or a GitHub repository, we can still install the local Python wheel. Start by creating a virtual environment for our test script. Enter the following commands in the terminal, which is in the **bottom** panel.

```
python3 -m venv smile
source smile/bin/activate
```

We are going to use `pip` to install `pixel_art` but we need to use the path to the wheel file. The wheel is found in the `dist` directory, which is inside the `pixel_art` directory.

```
python3 -m pip install pixel_art/dist/pixel_art-0.1-py3-none-
        any.whl
```

Once the installation is complete, we need to check one last thing. The last thing we need to check is `colorama` dependency. This package should be listed when we generate the list of required packages.

```
python3 -m pip freeze
```

You should see the `colorama` package listed as a dependency. When a user installs the `pixel_art` wheel they will also install `colorama` automatically. **Note**, your version number may be different than at the time of writing.

```
colorama==0.4.4
pixel-art==0.1
pkg-resources==0.0.0
```

▼ **Did you notice?**
In this case, the output from `freeze` is not being redirected to a `requirements.txt` file. We are just trying to verify that the `colorama` dependency is being recorded.

Create Some Pixel Art

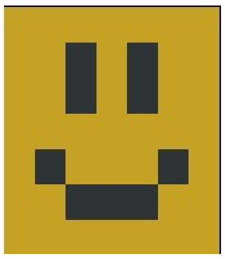
Using the newly installed package, let’s create some pixel art. The code below creates a seven by seven smiley face. Enter the following code in the **top** panel.

```
import pixel_art


smile = pixel_art.pixels.Image()
smile.add_row(0, 'yellow')
smile.add_row(1, 'yellow', 'yellow', 'black', 'yellow', 'black',
        'yellow', 'yellow')
smile.add_row(2, 'yellow', 'yellow', 'black', 'yellow', 'black',
        'yellow', 'yellow')
smile.add_row(3, 'yellow')
smile.add_row(4, 'yellow', 'black', 'yellow', 'yellow',
        'yellow', 'black', 'yellow')
smile.add_row(5, 'yellow', 'yellow', 'black', 'black', 'black',
        'yellow', 'yellow')
smile.add_row(6, 'yellow')
smile.display()
```

In the **bottom** panel, run your script. You should see the smiley face in the terminal.

```
python3 smile.py
```



The image depicts a yellow smiley face. The image is 7 pixels by 7 pixels. The eyes and mouth are black.

Posting to GitHub

Pushing to GitHub

Before continuing, you must push your work to GitHub. In the terminal:

- Commit your changes:

```
git add .
git commit -m "Create pixel_art package"
```

- Push to GitHub:

```
git push
```

Lab Challenge

Lab Challenge

Problem

The package `example_package` has two modules `mod1` and `mod2`. Modify the `__init__.py` file (top-left) so that `import example_package` imports the entire package.

Expected Output

Use the `test_package.py` file (bottom-left) to import `example_package` and call the `test_mod1` and `test_mod2` functions. The test file should have the following code:

```
import example_package

example_package.mod1.test_mod1()
example_package.mod2.test_mod2()
```

Clicking the button below will run the test script. You should see the following output:

```
Hello from module 1
Hello from module 2
```