Graphics and Computer Vision

Question 1

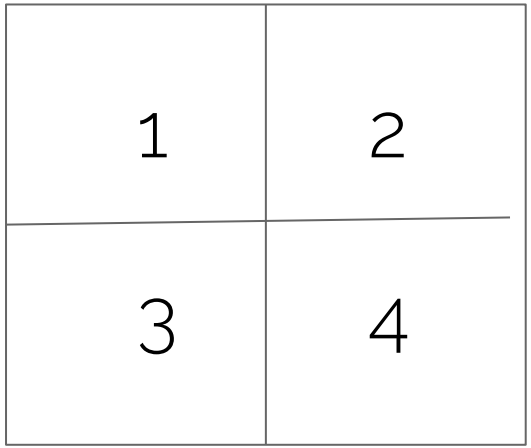
What **colour** would a pixel with RGB value [0, 0, 50] appear to be?

a. Green
b. Blue
c. Red

Is it **dark** or **light**?
a. Dark
b. Light
c. Almost exactly half-way

Question 2

How would we represent the following image as a list?
(Assume the numbers represent specific colours, like [52, 235, 50]



a. `[ 1, 2, 3, 4 ]`
b. `[ [1, 2], [3, 4] ]`
c. `[ [1, 3], [2, 4] ]`
d. `[ [1], [2], [3], [4] ]`

Question 3

If an image is represented by the list:

```
[ [[10,20,30], [15,20,200]],
  [[25,0,98], [5,0,0]],
  [[60,200,0], [0,0,200]] ]
```

1. How many pixels does it have?	A) 2	B) 3	C) 6	D) 18
2. How many rows does it have (height?)	A) 2	B) 3	C) 6	D) 18
3. How many columns does it have (width?)	A) 2	B) 3	C) 6	D) 18
4. How many pixels are essentially blue?	A) 0	B) 1	C) 2	D) 3
5. How many pixels do not have any red at all?	A) 0	B) 1	C) 2	D) 3

```python
# Check if a pixel is green

# Import custom module for image processing
import cmpt120image

def is_green(img, row, col):
    """
    Detects if a pixel is green
    Inputs: img - 2D list of RGB values
            row - row index of the pixel
            col - column index of the pixel
    Returns: True if green; False otherwise
    """

    selected_pixel = kid[0][0]
    r = selected_pixel[0]
    g = selected_pixel[1]
    b = selected_pixel[2]

    low_red    = r < 30
    high_green = g > 255 - 30
    low_blue   = b < 30
    return low_red and high_green and low_blue


# Load images
kid = cmpt120image.get_image('images/kid-green.jpg')

# Call our function to check if a pixel is green
print(is_green(kid, 0, 0))
```

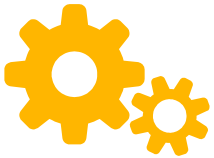
```
True
>
```

Recall Checking a pixel colour

`img[0][0]`

Image Processing

This lesson



- Review of nested for loops
- Creation of your own module

https://docs.python.org/3/tutorial/modules.html

Nested Loops

Draw on Image

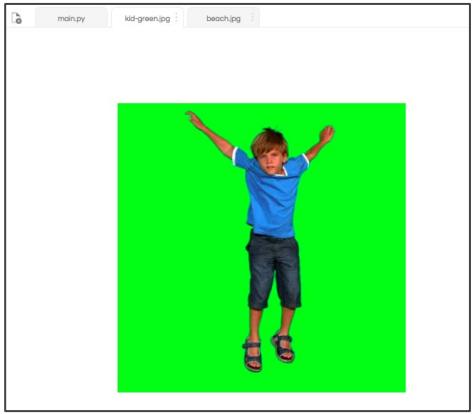
- Let's write some code to draw a black square in the top-left of an image.

Exploring images

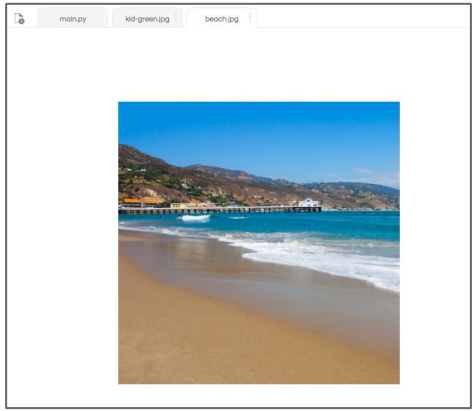
```python
# Draw a black square on an image.

# Import custom module for image processing
import cmpt120image

def is_green(img, row, col):
    """
    Detects if an pixel's RGB value combines to gre
    Input:  img - 2D list of RGB values
            row - row of the pixel
            col - col of the pixel
    Returns: True if green, False otherwise
    """
    my_pixel = img[row][col]
    r = my_pixel[0]
    g = my_pixel[1]
    b = my_pixel[2]
    return r < 60 and g >= 205 and b < 60

# Main program
# ----------------------------------------------

# Load image
kid = cmpt120image.getImage('ExWeek10/kid-green.jpg')

# Show before
cmpt120image.showImage(kid, "Before Change")
print(f"Is (10,20) green? {is_green(kid, 10, 20)}")
input("Press enter to continue...")

# Draw black square in top left
SIZE = 100
for row in range(SIZE):
    for col in range(SIZE):
        kid[row][col] = [0,0,0]

# Show after
cmpt120image.showImage(kid, "After Change")
print(f"Is (10,20) green? {is_green(kid, 10, 20)}")
input("Press enter to continue...")

# Save
cmpt120image.saveImage(kid, "ExWeek10/kid_with_sq.jpg")
```

black_square.py

Our goal: Combine images



+



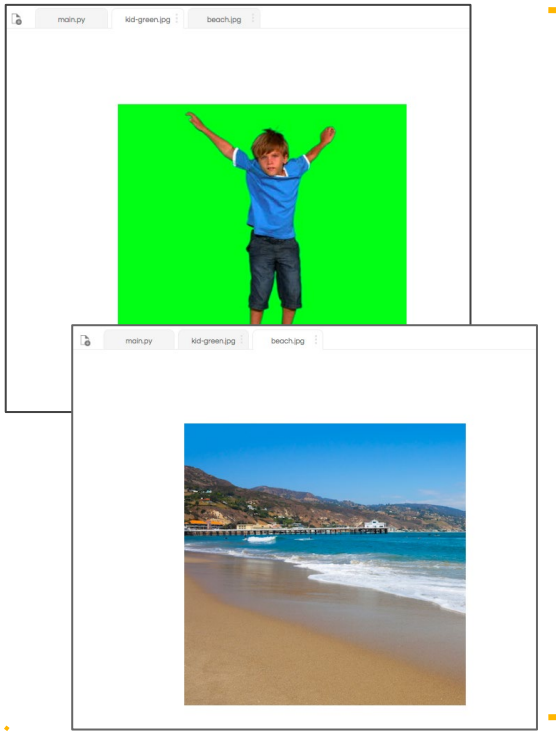
Combining images!

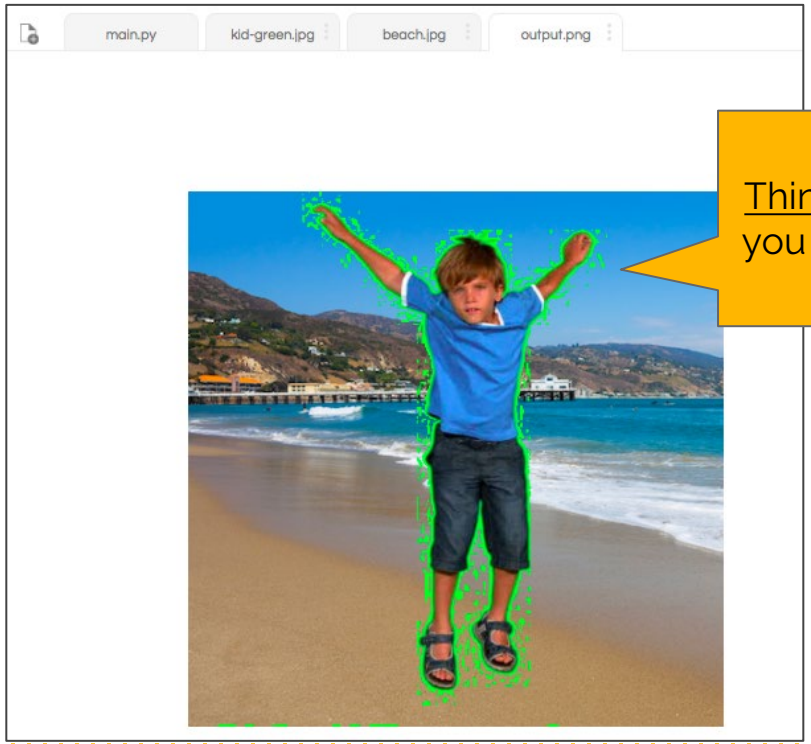
```python
# Combine images

# Import custom module for image processing
import cmpt120image

def is_green(img, row, col):
    """
    Detects if an pixel's RGB value combines to greer
    Input:  img - 2D list of RGB values
            row - row of the pixel
            col - col of the pixel
    Returns: True if green, False otherwise
    """
    TOL = 30
    my_pixel = img[row][col]
    r = my_pixel[0]
    g = my_pixel[1]
    b = my_pixel[2]
    return r < TOL and g >= 255 - TOL and b < TOL

```

Nested loops are useful for traversing 2D tables

```python
# Main program
# ------------------------------------------------

# Load images
beach = cmpt120image.get_image('images/beach.jpg')
kid = cmpt120image.get_image('images/kid-green.jpg')

# Merge images
height = len(kid)
width = len(kid[0])
for row in range(height):
    for col in range(width):
        if is_green(kid, row, col):
            kid[row][col] = beach[row][col]

cmpt120image.show_image(kid, "MERGED!")
cmpt120image.wait_for_escape()
```

Combining images!



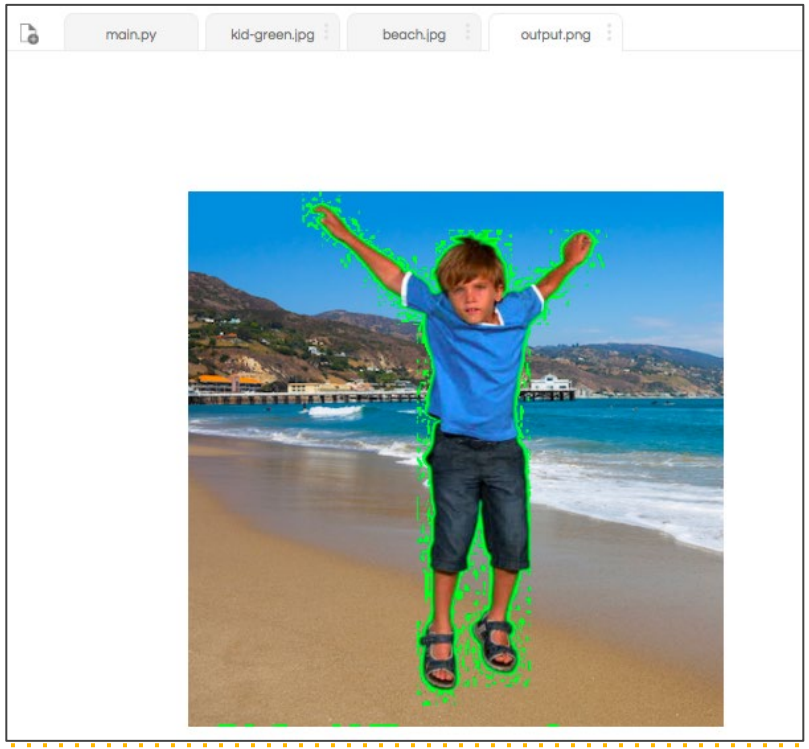


Think: How might you improve this?

Aim for excellence

Modify the code to make the image combination *seamless*. Trust me, it's doable :)

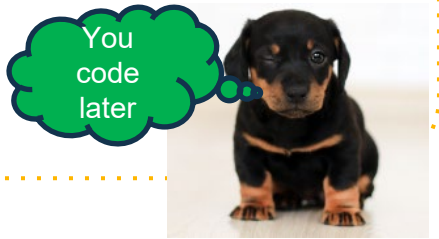
Bonus: Try a different background! Just make sure your source background image is 450x450 pixels in size.



New challenge

- Write a python program which reads in an image and **turns all the bright green pixels to black**

- Challenges for later
  - Can you change green to white?
  - Can you change to a random gray?
  - Can you change green into horizontal white/black lines?

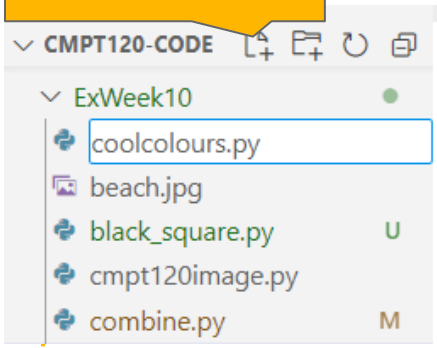


Cleanup yer code

Make your own module

https://docs.python.org/3/tutorial/modules.html

Create a new file.



A **module's name** is its **filename** with the .py removed.

ExWeek10 > coolcolours.py > ...

```
1   # Cool Colours module
2
3   def is_green(img, row, col):
4       """
5       Detects if an pixel's RGB value combines to green
6       Input:  img - 2D list of RGB values
7               row - row of the pixel
8               col - col of the pixel
9       Returns: True if green, False otherwise
10      """
11      TOL = 30
12      my_pixel = img[row][col]
13      r = my_pixel[0]
14      g = my_pixel[1]
15      b = my_pixel[2]
16      return r < TOL and g >= 255 - TOL and b < TOL
17
18  def get_colour_name(img, row, col):
19      """
20      Returns the name of the colour at a pixel
21      Input:  img - 2D list of RGB values
22              row - row of the pixel
23              col - col of the pixel
24      Returns: True if green, False otherwise
25      """
26      if is_green(img, row, col):
27          return "green"
28      else:
29          return "other"
```

Paste your **function definitions** in here, and remove them from your main program.

Our **coolcolours** module contains 2 functions, **is_green()** and **get_colour_name()**

Let's make a module

How to use our module

ExWeek10 > my_test.py > ...

```
1   # Combine images using the coolcolours module
2
3   # Import custom module for image processing
4   import cmpt120image
5   import coolcolours
6
7   # Main program
8   # ------------------------------------------------
9
10  # Load images
11  beach = cmpt120image.getImage('ExWeek10/beach.jpg')
12  kid = cmpt120image.getImage('ExWeek10/kid-green.jpg')
13
14  # Merge images
15  height = len(kid)
16  width = len(kid[0])
17  for row in range(height):
18      for col in range(width):
19          if coolcolours.is_green(kid, row, col):
                kid[row][col] = beach[row][col]
21
22  cmpt120image.showImage(kid, "MERGED!")
23  input("Press enter to continue...")
```

Import our custom module

Use the functions we defined in our module, preceded by the **module name** and a **dot**.

my_test.py

Looking at cmpt120image.py

2024-7 > Week10 > previous > cmpt120image.py > ...

```
1   # cmpt120image.py
2   # Some helper functions to wrap the Pygame image functions
3   # CMPT 120; version Fall 2024 (modified by Brian Fraser; some code written with help of CoPilot)
4
5   import pathlib
6   import pygame
7   import numpy
8
9   def is_valid_pixels(pixels):
10      """
11      Input: pixels - 3d list of lists of RGB values (a height-by-width-by-3 list)
12      Returns: True if pixels is a valid 3d list of lists of RGB values, False otherwise
13      """
14      if type(pixels) != list or len(pixels) == 0:
15          return False
16      if type(pixels[0]) != list or len(pixels[0]) == 0:
17          return False
18      if type(pixels[0][0]) != list or len(pixels[0][0]) == 0:
19          return False
20      return True
```

Some functions (like is_valid_pixels(…)) are used just inside the module to check the data you pass as arguments.

Looking at cmpt120image.py

2024-7 > Week10 > previous > cmpt120image.py > ...

```
1   # cmpt120image.py
2   # Some helper functions to wrap the Pygame image functions
3   # CMPT 120; version Fall 2024 (modified by Brian Fraser; s
4
5   import pathlib
6   import pygame
7   import numpy
8
9   def is_valid_pixels(pixels):
10      """
11      Input: pixels - 3d list of lists of RGB values (a height-by-width-by-3 list)
12      Returns: True if pixels is a valid 3d list of lists of RGB values, False otherwise
13      """
14      if type(pixels) != list or len(pixels) == 0:
15          return False
16      if type(pixels[0]) != list or len(pixels[0]) == 0:
17          return False
18      if type(pixels[0][0]) != list or len(pixels[0][0]) == 0:
19          return False
20      return True
```

A **package** is a collection of **modules**.

pygame is a package containing the module **image**.

Looking at cmpt120image.py

```
22  def get_image(filename):
23      """
24      Input: filename - string containing image filename to open relative
25          to the folder of the current python file.
26      Returns: 3d list of lists (a height-by-width-by-3 list)
27      """
28      # Check argument types to help catch passing in the
29      # NOTE: If you are told there is an error on these
30      # means you are passing in the wrong type of argume
31      # Check your calling code carefully, using the debugger
32      assert type(filename) == str, "get_image(): `filenam
33
34      folder_of_code = pathlib.Path(__file__).par
35      full_name = folder_of_code / filename
36      image = pygame.image.load(full_name)
37
38      # do a transpose so its rows correspond to height of the image
39      return pygame.surfarray.array3d(image).transpose(1, 0, 2).tolis
40
41
42  def save_image(pixels, filename):
43      """
44      Input:  pixels - 3d list of lists of RGB values (a height-by-wi
45          filename - string containing filename to save image, relati
46                     folder of the current python file.
```

You can access the package's inner module using the dot operator.

We use **pygame**'s **image** module that has a **load** function to open the file, and get the image's 3d array representation.

We don't need to understand all the details, just know what the function does.

To hide away this complexity, we wrap this in a function called **get_image** which returns the 3d list of lists we have been using.

Let's review some concepts

What can nested loops be used for in the context of image processing?

What does **answer** contain after the code is run below?

```
def special(numbers):
  for x in numbers:
    for y in numbers:
      if x < y:
        return x+y

answer = special([3,1,2])
```

x will be assigned [3, 1, 2]

y will be assigned [3, 1, 2]

Options:

a) 1
b) 2
c) 3
d) 4