PIC16A Final Exam, Problem 2 (45 points)

Spring 2021

Academic Integrity Statement

As a matter of Departmental policy, **we are required to give you a 0** unless you **type your name** after the following statement:

> *I certify on my honor that I have neither given nor received any help, or used any non-permitted resources, while completing this evaluation.*

- David Nguyen

Permitted Resources

This exam is open-book, open-notes, open-internet, open-everything. The only thing you are not allowed to do is ask another human being for help, with the exception of me. Examples:

1. **Permitted**: Checking course notes and videos; browsing Google or StackOverflow; consulting previous Campuswire posts.
2. **Not permitted**: Requesting help on Chegg; publicly posting on Campuswire (messages to me are ok); privately asking classmates for help; posting on StackOverflow.

If you encounter any situation in which you're not 100% sure what's permitted, **just ask**!

Partial Credit

Let us give you partial credit! If you're stuck on a problem and just can't get your code to run:

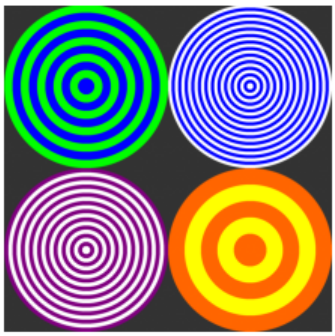
First, **breathe**. Then, do any or all of the following:

1. Write down everything relevant that you know about the problem, as comments where your code would go.
2. If you have non-functioning code that demonstrates some correct ideas, indicate that and keep it (commented out).
3. Write down pseudocode (written instructions) outlining your solution approach.

In brief, even if you can't quite get your code to work, you can still **show us what you know.**

Introduction

In this problem, you'll use object-oriented programming and Numpy techniques to create simple graphics with bullseyes, like this one:



Example output from this problem.

A Note on Loops

There is precisely **one** place in this entire problem in which a loop (such as a `for` -loop, a `while` -loop, or a list comprehension) is completely appropriate. This place is **Part D**. I'll tell you when we get there. With that one exception, you should avoid the use of loops whenever possible. Solutions that use loops elsewhere will receive partial credit.

Concision and Style

Remember, *concision and style* are two of the criteria on which we assess your solutions. You should aim to write code that is as short, simple, and readable as possible. My own solution for this problem requires 21 lines excluding comments. Longer solutions are ok, as long as they don't perform redundant computation and appropriately make use of Numpy operations.

Comments and Docstrings

Comments and docstrings are not required in any part of this problem. However,they may help us give you partial credit, so they are recommended unless you're feeling very confident.

Input Checking

- **It is not necessary to perform any input checking** in this problem -- you can assume that the user will supply inputs with the correct data types, shapes, etc.

Image Appearance

It's ok for your images to look a little different from mine. For example, your image might appear slightly jagged or blocky, depending on the size of your background `Canvas`. That's ok! As long as your image clearly demonstrates correct code, you'll receive full credit.

Part A

You don't have to write any code here, just run the block below.

In [1]:
```
# run this to get started
import numpy as np
from matplotlib import pyplot as plt
```

You did it!

Part B (10 points)

Create a class `Canvas`, and implement two methods.

- The `__init__()` method should take two arguments other than `self`, `background` and `n`.
  - The `background` is expected to be a 1d Numpy array of length 3, representing an RGB color. For example, `black = np.array([0,0,0])` and `purple = np.array([0.5, 0.5, 0])`.
  - At this stage, the `__init__` method should create an instance variable `self.im`, a Numpy array of shape `(n, n, 3)`. This array should be constructed so that `self.im[i,j] == background` for each value of `i` and `j`.
- The `show()` method should take no arguments (except for `self`), and simply display `self.im` using the `plt.imshow()` function.

For example, the code

```
purple = np.array([0.5, 0.0, 0.5])
C = Canvas(purple, 2001) # 2001 x 2001 pixels
C.show()
```

should display a purple square, like this one:

Example output.

Make sure to run the test block after your class definition in order to demonstrate that your code works.

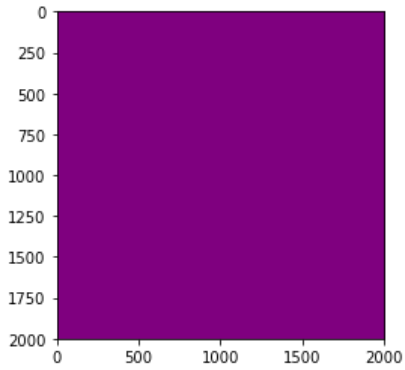
Notes

- You can use `ax.axis("off")` to remove the axis ticks and borders, although that's not required for this problem.
- For me, the easiest way to create `self.im` was to create an array of appropriate dimensions using `np.zeros()`, and then populate it using array broadcasting.
- This problem can be solved using a loop for **partial** credit.

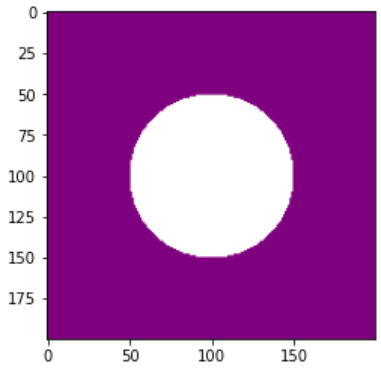
In [44]:
```
#imports
import numpy as np
from matplotlib import pyplot as plt
```

In [278]:
```
# define your Canvas class here
# solutions to Parts B, C, and D should all be in this cell
class Canvas:
    def __init__(self, background, n):
        #error checks
        if type(background) is not np.ndarray:
            raise TypeError('"background" needs to be a numpy array ')
        if (background.shape != (3,)) or (len(background) != 3):
            raise ValueError('"background" must be 1d array with length 3')
        if type(n) not in [int, float]:
            raise TypeError('"n" needs to be an int or float')

        #setting variables
        self.background = background
        self.n = n

        #get im to right size
        self.im = np.ones([self.n,1,1]) * np.ones([1,self.n,1]) * np.ones([1,1,3])
        #get im to right color spec
        self.im = self.im * self.background

    def show(self):
        '''
        displays the plot
        '''
        fig, ax = plt.subplots(1)
        ax.imshow(self.im)
        #ax.axis("off")
        pass

    def add_disk(self, centroid, radius, color):
        '''
        creates a circle with given color radius at given point for center
        '''
        thing = np.arange(self.n)
        thing = thing.reshape((1, self.n, 1))
        thing2 = np.ones([self.n,1,1]) * thing
        thing2 = thing2 * np.ones([1,1,3])

        counter = 0
        for i in range(len(thing2)):
            for j in range(len(thing2[i])):
                h = i - centroid[0]
                v = j - centroid[1]
                r = h**2 + v**2
                if r < radius**2:
                    self.im[i][j] = color

    def add_bullseye(self, centroid, radius, color, color2, bandwidth):
        radii = []
        radii.append(radius)

        marker = True
        check = radius
        while(marker):
            check = check - bandwidth
            if check <= 0:
                marker = False
            radii.append(check)

        counter = 0
        for r in radii:
            if counter%2 == 0:
                self.add_disk(centroid, r, color)
                counter += 1
            elif counter%2 == 1:
                self.add_disk(centroid, r, color2)
                counter += 1

        pass
```

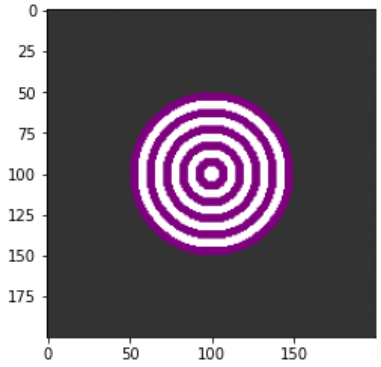
In [279]:
```
#part b
# test code: run but do not modify
purple = np.array([0.5, 0, 0.5])
C = Canvas(purple, 2001) # 2001 x 2001 pixels
C.show()
```



In [285]:
```
#part c
purple = np.array([.5, 0, .5])
white  = np.array([1, 1, 1])

C = Canvas(purple, 200)
center = (100, 100)
C.add_disk(center, 50, white)
C.show()
```



In [286]:
```
#part d
# test code: run but do not modify
purple = np.array([0.5, 0.0, 0.5])
white  = np.array([1.0, 1.0, 1.0])
grey   = np.array([0.2, 0.2, 0.2])

C = Canvas(background = grey, n = 200)
C.add_bullseye((100, 100), 50, purple, white, bandwidth = 5)
C.show()
```



Part C (15 points)

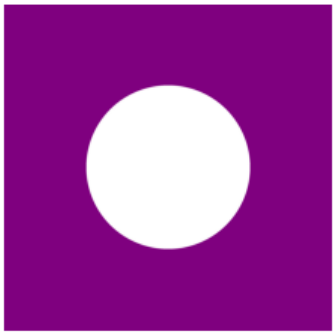
Modifying your class above (that is, not copy/pasting code), implement a method called `add_disk(centroid, radius, color)`, which draws a colored disk with specified `radius`, centered at `centroid`, of the specified color.

- `centroid` may be assumed to be a tuple, list, or Numpy array of the form `(x,y)`, where `x` gives the horizontal coordinate of the disk's center and `y` gives the vertical coordinate.
- All points within distance `radius` of the `centroid` should be filled in with `color`.

For example, the code

```
purple = np.array([.5, 0, .5])
white  = np.array([1, 1, 1])

C = Canvas(background = purple, n = 2001)
C.add_disk((1001, 1001), 500, white)
C.show()
```

should produce the following image:



Example output.

Run the test code supplied below to demonstrate that your code is working.

Math Note

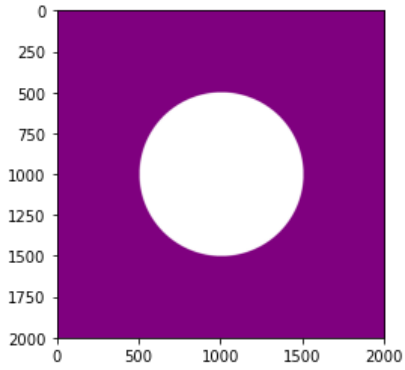
Recall that the (open) disk of radius r with centroid (x_0, y_0) is the set of all points (x, y) satisfying the formula

$$(x - x_0)^2 + (y - y_0)^2 < r^2 \ .$$

Programming Notes

- The function `np.meshgrid` is a useful way to represent the horizontal and vertical coordinates. If you take this approach, you should ensure that this function is called only **once** even if your user calls `self.add_disk()` multiple times.
- This problem can be solved using a loop for **partial** credit.
- If you do take the loop-based approach for partial credit, you may need to reduce `n`, the size of the `Canvas`, in the examples below, as your code might be slower.

In [299]:
```
# test code: run but do not modify
purple = np.array([.5, 0, .5])
white  = np.array([1, 1, 1])

C = Canvas(background = purple, n = 2001)
C.add_disk((1001, 1001), 500, white)
C.show()
```



Part D (15 points)

Modify your code from Part B (still no copy/paste), write a method called `add_bullseye(centroid, radius, color1, color2, bandwidth)` . This method should create a bullseye pattern consisting of concentric circles with alternating colors. Each circle should have thickness equal to `bandwidth` , and the radius of the entire pattern should be equal to `radius` . For example:

```
purple = np.array([0.5, 0.0, 0.5])
white  = np.array([1.0, 1.0, 1.0])
grey   = np.array([0.2, 0.2, 0.2])

C = Canvas(background = grey, n = 2001)
C.add_bullseye((1001, 1001), 500, purple, white, bandwidth = 50)
C.show()
```

Example output.

In this example, each of the bands is 50 pixels thick, and the entire pattern has radius 500.

Loops

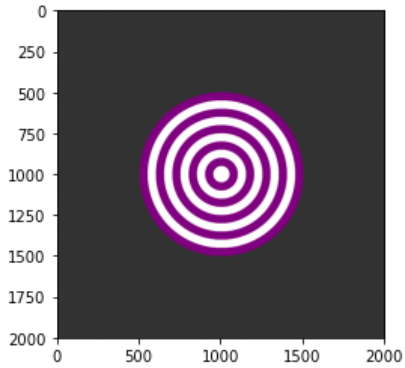
In this part, it would be appropriate to write **one** `for` - or `while` -loop.

Hint

You might wish to create a new cell and run the following code -- it could help you catch on to the right idea.

```
purple = np.array([0.5, 0.0, 0.5])
white  = np.array([1.0, 1.0, 1.0])
grey   = np.array([0.2, 0.2, 0.2])

C = Canvas(background = grey, n = 2001)
C.add_disk((1001, 1001), 500, purple)
C.add_disk((1001, 1001), 450, white)
C.show()
```

In [281]:
```
# test code: run but do not modify
purple = np.array([0.5, 0.0, 0.5])
white  = np.array([1.0, 1.0, 1.0])
grey   = np.array([0.2, 0.2, 0.2])

C = Canvas(background = grey, n = 2001)
C.add_bullseye((1001, 1001), 500, purple, white, bandwidth = 50)
C.show()
```



Part E (5 points)

Write a further demonstration of the correct functioning of your code by creating a new `Canvas` with at least three bullseyes drawn on it. You should demonstrate:

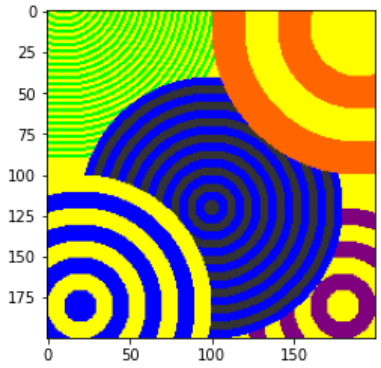
- At least three (3) different centroids.
- At least four (4) different colors.
- At least three (3) different values of the `bandwidth` parameter.

You're also welcome to vary the `radius` parameter, but this isn't required.

You're encouraged to be creative! Coordinate your colors, let your bullseyes partially intersect, etc. etc. But if you're not really feeling your artistic mojo today, it's ok to base your solution on the example shown at the very beginning of this problem, which satisfies all of the above criteria. I've predefined the colors I used for your convenience.

Provided that you've solved up to Part D correctly, no further modifications to your `Canvas` class are required.

In [298]:
```
# predefined colors -- feel free to add your faves!

purple = np.array([.5, 0, .5])
white  = np.array([1, 1, 1])
green  = np.array([0, 1, 0])
blue   = np.array([0, 0, 1])
orange = np.array([1, .4, 0])
yellow = np.array([1, 1, 0])
grey   = np.array([.2,.2,.2])

# write your demonstration here
C = Canvas(background = yellow, n = 200)
C.add_bullseye((-10, 10), 100, green, yellow, bandwidth = 2)
C.add_bullseye((180, 180), 60, purple, yellow, bandwidth = 10)
C.add_bullseye((120, 100 ), 80, blue, grey, bandwidth = 5)
C.add_bullseye((10, 190), 90, orange, yellow, bandwidth = 20)
C.add_bullseye((180, 20), 80, yellow, blue, bandwidth = 10)

# don't forget to show the image!
C.show()
```



In [ ]: