Recursion is fantastic…

And is often “handy”…

What’s Up Next…

- Loop structures: **`for`** and **`while`**
- Writing some “bigger” programs
    - Secret Sharing (cryptography)
    - Games (Nim, Mastermind)
    - Data compression

Loops!

Mystery 1

I love a good mystery!

```
def leppard(input_string):
    output_string = ''
    for symbol in input_string:
        if symbol == 'o':
            output_string = output_string + 'ooo'
        else:
            output_string = output_string + symbol
    return output_string
```

```
>>> leppard(“hello”)
```

```
>>> leppard(“hello to you”)
```

Mystery 2

I love a good mystery!

```
vowels = ['a', 'e', 'i', 'o', 'u']

def spamify(word):
    for i in range(len(word)):
        if word[i] not in vowels:
            return word[0:i] + "spam" + word[i+1:]
    return word
```

What’s range?

```
>>> spamify(“oui”)

>>> spamify(“hello”)

>>> spamify(“aardvark”)
```

for

```
for <variable> in <iterable>:
    Do stuff!


for symbol in “blahblahblah”:
    print(symbol)

for element in [1, 2, 3, 4]: …

for index in range(42): …
```

Three uses of for!

I’d like to see four uses of three!

while

```
while <condition>:
    Do stuff!

i = 0
while i < 100:
    print(i)
    i += 1

sum = 0
i = 0
while i < 10:
    sum = sum + i
    i += 1
print(sum)
```

Write equivalent for-loops.

Draw flow charts.

Using for

```
def mapSqr(L):
    '''
    Assume L is a list.  Return map(sqr, L).
    '''
```

Move over Playstation!

```
num = int(input(“Give me a number: ”))
string = input(“Give me a string: ”)
```

```
import random

def play():
    print('Welcome!')
    secret = random.randint(1, 100)
    num_guesses = 0
    user_guess = 0
    while user_guess != secret:
        user_guess = int(input('Enter your guess: '))
        num_guesses += 1
        if user_guess == secret:
            print('You got it in', num_guesses, 'guess(es)!')
        elif user_guess > secret:
            print('Too high')
        else:
            print('Too low')
    print('Thanks for playing.')

play()
```

Printing strings, numbers, etc.

Move over Playstation!

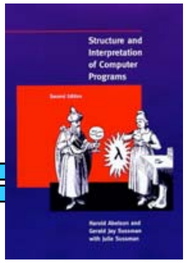
Can you spot the difference?

```python
import random

def play():
    print('Welcome!')
    secret = random.randint(1, 100)
    num_guesses = 0
    user_guess = 0
    while True:
        user_guess = int(input('Enter your guess: '))
        num_guesses += 1
        if user_guess == secret:
            print('You got it in', num_guesses, 'guess(es)!')
            break
        elif user_guess > secret:
            print('Too high')
        else:
            print('Too low')
    print('Thanks for playing.')

play()
```

Good Design


Structure and Interpretation of Computer Programs
Second Edition
Harold Abelson and Gerald Jay Sussman with Julie Sussman

Programs must be written for people to read, and only incidentally for machines to execute. - Abelson and Sussman

1. Design your program “on paper” first. Identify the separate logical parts and the input/output for each parts.
2. Once your design is established, write the function “signatures” (function name, inputs) and docstrings.
3. Fill in the code for a function, **test that function carefully, and proceed only when you are convinced that the function works correctly.**
4. Use descriptive function and variable names (how about **`x, stuff, florg, jimbob`** ?).
5. Don’t replicate functionality.
6. Keep your code readable and use comments to help! `# Here’s one now!`
7. Avoid global variables unless absolutely necessary! Instead, pass each function just what it needs.
8. Use recursion and functional constructs (e.g. map, reduce, filter, lambda) where appropriate.

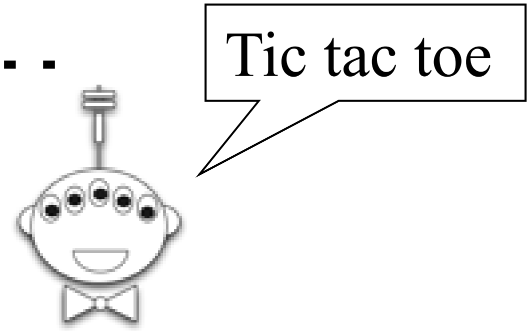
Exercises

Implement factorial, using a for-loop.

Use a loop to implement fib, where

fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1)+fib(n-2)

An Example…



Objective: Write a tic-tac-toe program that lets two human players play and stops when a player has won.

Functions:

main(): Welcomes user, plays a game, asks if we want to play again
welcome(): Prints the welcome message
playGame(): Maintains a board and plays one game
getMove(board, player): Queries the player (1 or 2) for her/his move and changes the board accordingly
printBoard(board): Takes a board as input and displays it
gameOver(board): Evaluates a board to see if game over

```
'''
Tic-tac-toe by Ran Libeskind-Hadas
Modified by Brian Borowski, 10/28/2014
Updated to Python 3 on 01/23/2016
'''
debug = False

def main():
    '''This is the main function for the tic-tac-toe game'''
    welcome()
    while True:
        if debug: print('About to enter playGame()')
        playGame()
        response = input('Would you like to play again? (y or n): ').strip()
        if not response in ['y', 'Y', 'yes', 'Yes', 'Yup', 'si', 'oui', 'youbetcha']:
            print('Bye!')
            return

def welcome():
    '''Prints the welcome message for the game.
       We might also print the rules for the game and any other
       information that the user might need to know.'''
    print('Welcome to tic-tac-toe!')

def playGame():
    '''Play one game of tic-tac-toe'''
    if debug:  print('Entering the playGame() function')
    board = [ [' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]
    player = 1
    print('The board looks like this:')
    printBoard(board)
    while not gameOver(board):
        getMove(board, player)
        if player == 1: player = 2
        else: player = 1
        print('The board looks like this:')
        printBoard(board)
```

```python
def gameOver(board):
    '''Returns False if the game is NOT over.  Otherwise, prints a message
       indicating which player has won and then returns True indicating that the
       game is over.'''
    if debug:  print('Entering the gameOver function')
    winner = getWinner(board)
    if winner == '1':
        print('Player 1 wins!')
        return True
    if winner == '2':
        print('Player 2 wins!')
        return True
    if boardFull(board):
        print('Tie.')
        return True
    return False

def getMove(board, player):
    '''Takes the board and the current player (1 or 2) as input.
        Asks the player for her/his move.  If it's a legitimate move,
        the change is made to the board.  Otherwise, the player
        is queried again until a valid move is provided.'''
    print('Player ' + str(player) + '\'s turn')
    while True:
        row = int(input('Enter the row: ').strip())
        column = int(input('Enter the column: ').strip())
        if row < 0 or row > 2 or column < 0 or column > 2:
            print('That\'s not a valid location on the board! Try again.')
        elif board[row][column] != ' ':
            print('That cell is already taken! Try again.')
        else:
            board[row][column] = str(player)
            break
```

```python
def printBoard(board):
    if debug:  print('Entering the printBoard() function')
```

Try to implement this function.

```python
def boardFull(board):
    if debug:  print('Entering the boardFull() function')
```

And this one too!

A board should be printed as follows:

```
   |   |
-----------
   | 1 | 1
-----------
   |   | 2
```

```python
def printBoard(board):
    if debug:  print('Entering the printBoard() function')
    for row in range(0, 3):
        print(' ', end='')
        for column in range(0, 3):
            print(board[row][column], end=' ')
            if column < 2: print('|', end=' ')
        print()  # CAUSES A LINEBREAK!
        if row < 2: print('-' * 11)

def boardFull(board):
    if debug:  print('Entering the boardFull() function')
    for row in range(3):
        for col in range(3):
            if board[row][col] == ' ':
                return False
    return True
```

```
def getWinner(board):
    if debug:  print('Entering the getWinner() function')
    # Check rows
    for row in range(3):
        val = board[row][0]
        if val != ' ':
            col = 1
            while col < 3:
                if board[row][col] != val:
                    break
                col += 1
            if col == 3:
                return val
    # Check columns
    for col in range(3):
        val = board[0][col]
        if val != ' ':
            row = 1
            while row < 3:
                if board[row][col] != val:
                    break
                row += 1
            if row == 3:
                return val
# Check major diagonal
    val = board[0][0]
    if val != ' ':
        index = 1
        while index < 3:
            if board[index][index] != val:
                break;
            index += 1
        if index == 3:
            return val
    # Check minor diagonal
    val = board[0][2]
    if val != ' ':
        index = 1
        while index < 2:
            if board[index][3 - index - 1] != val:
                break;
            index += 1
        if index == 3:
            return val
    return ' '
```

```
if __name__ == '__main__':
    main()
```

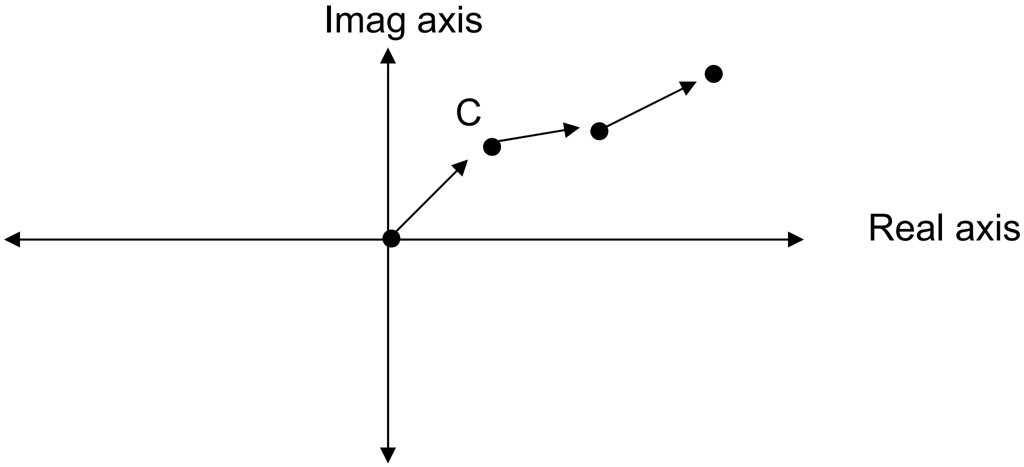
Lab Problem: The Mandelbrot Set

Consider some complex number C

$z_0 = 0$

$z_{n+1} = z_n^2 + C$

For which values of C does this *not* diverge?



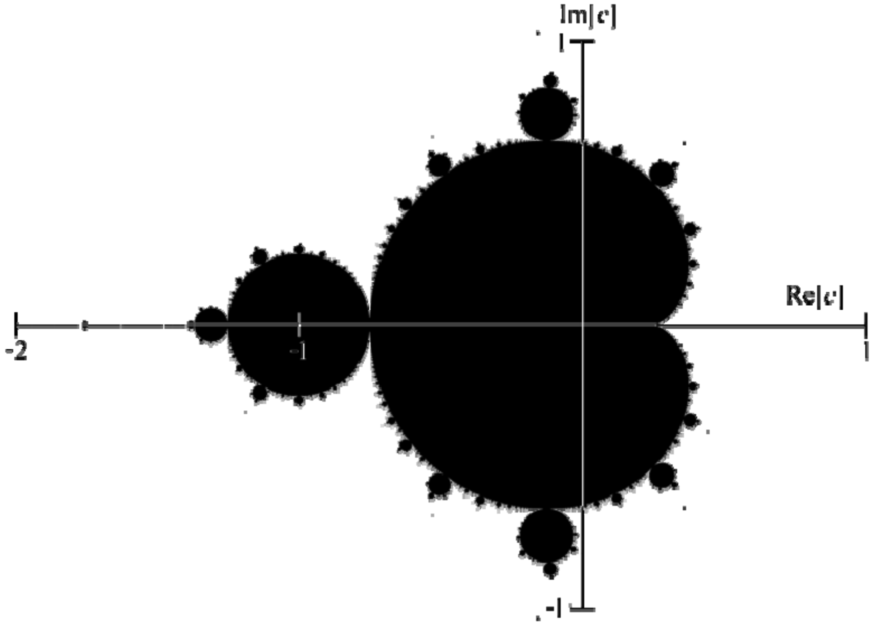
Lab Problem: The Mandelbrot Set

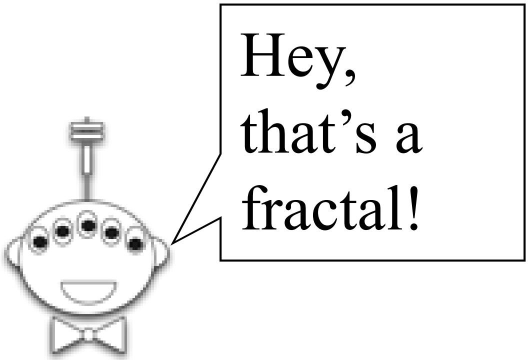
Consider some complex number C

$z_0 = 0$

$z_{n+1} = z_n^2 + C$

For which values of C does this *not* diverge?





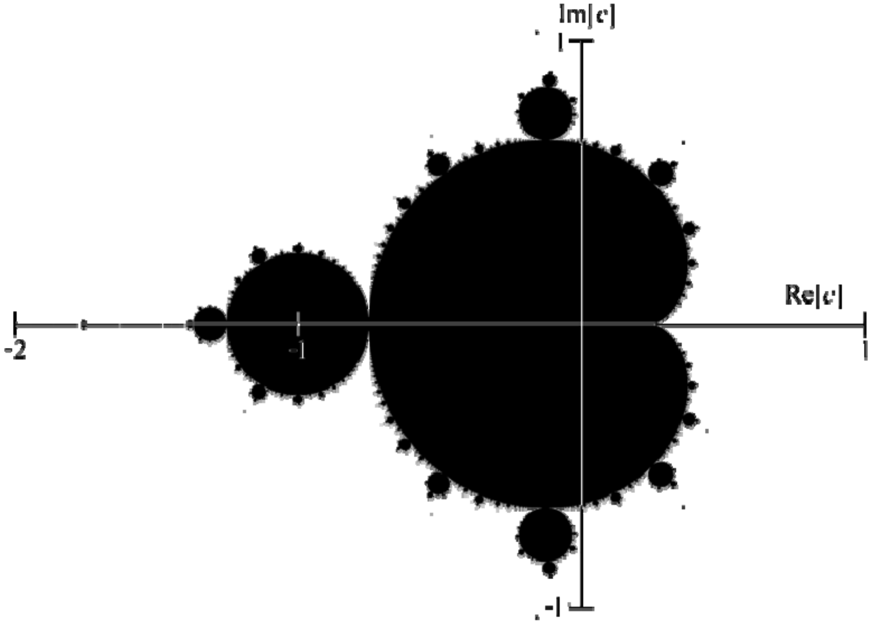
Lab Problem: The Mandelbrot Set

Consider some complex number C

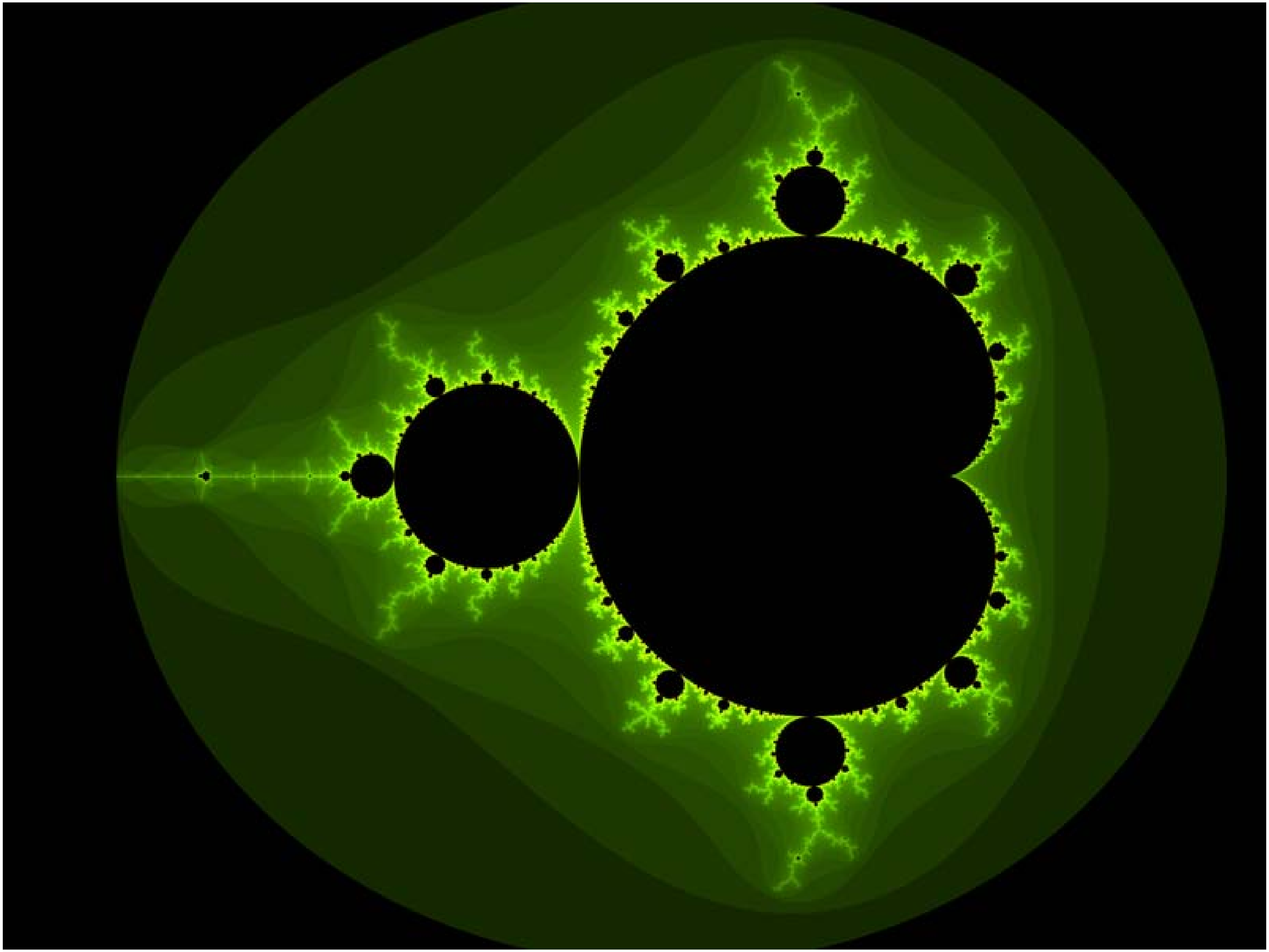
$z_0 = 0$

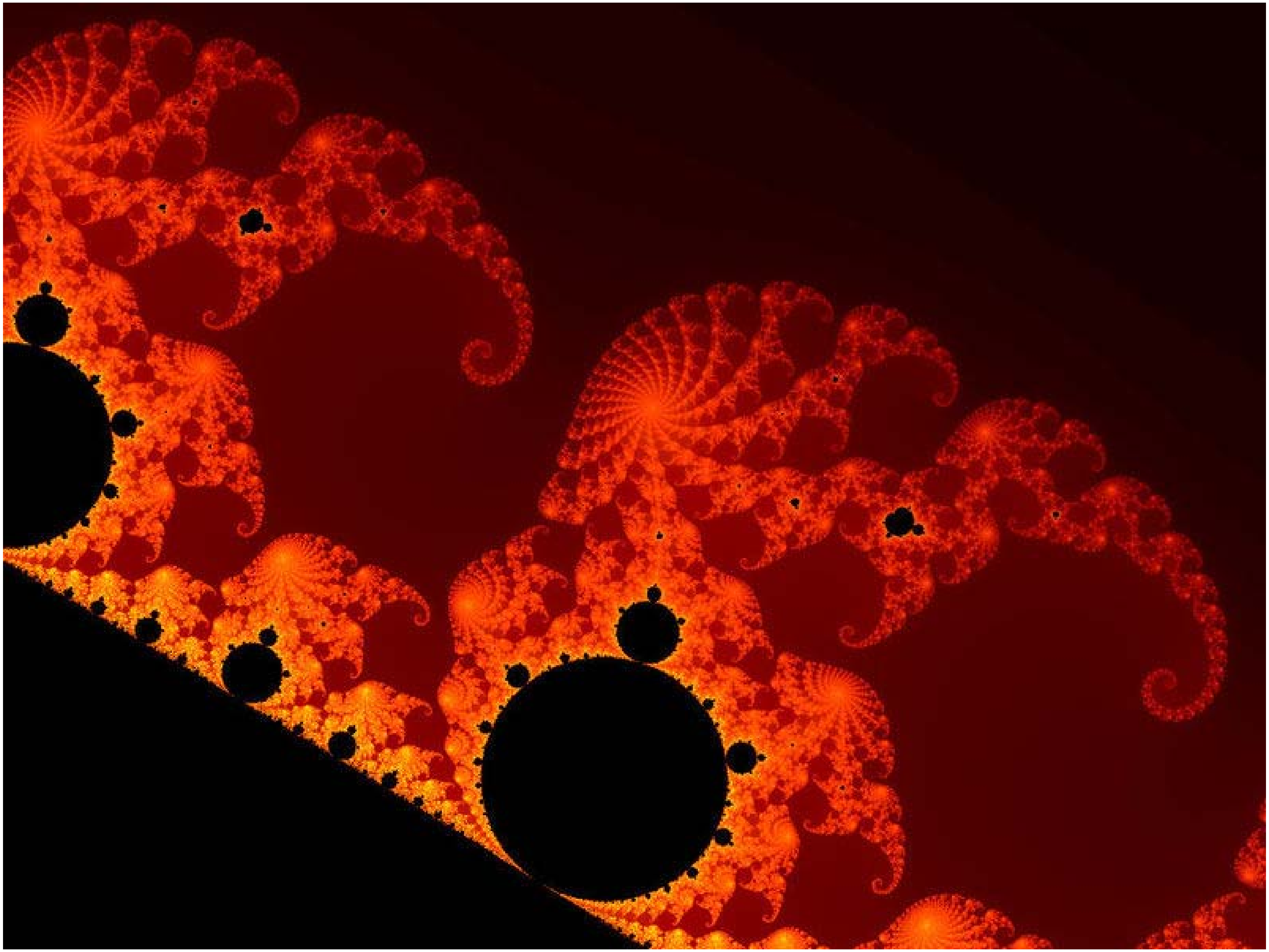
$z_{n+1} = z_n^2 + C$

For which values of C does this *not* diverge?



It is known that we can approximate the divergence test by seeing whether z_n exceeds 2.





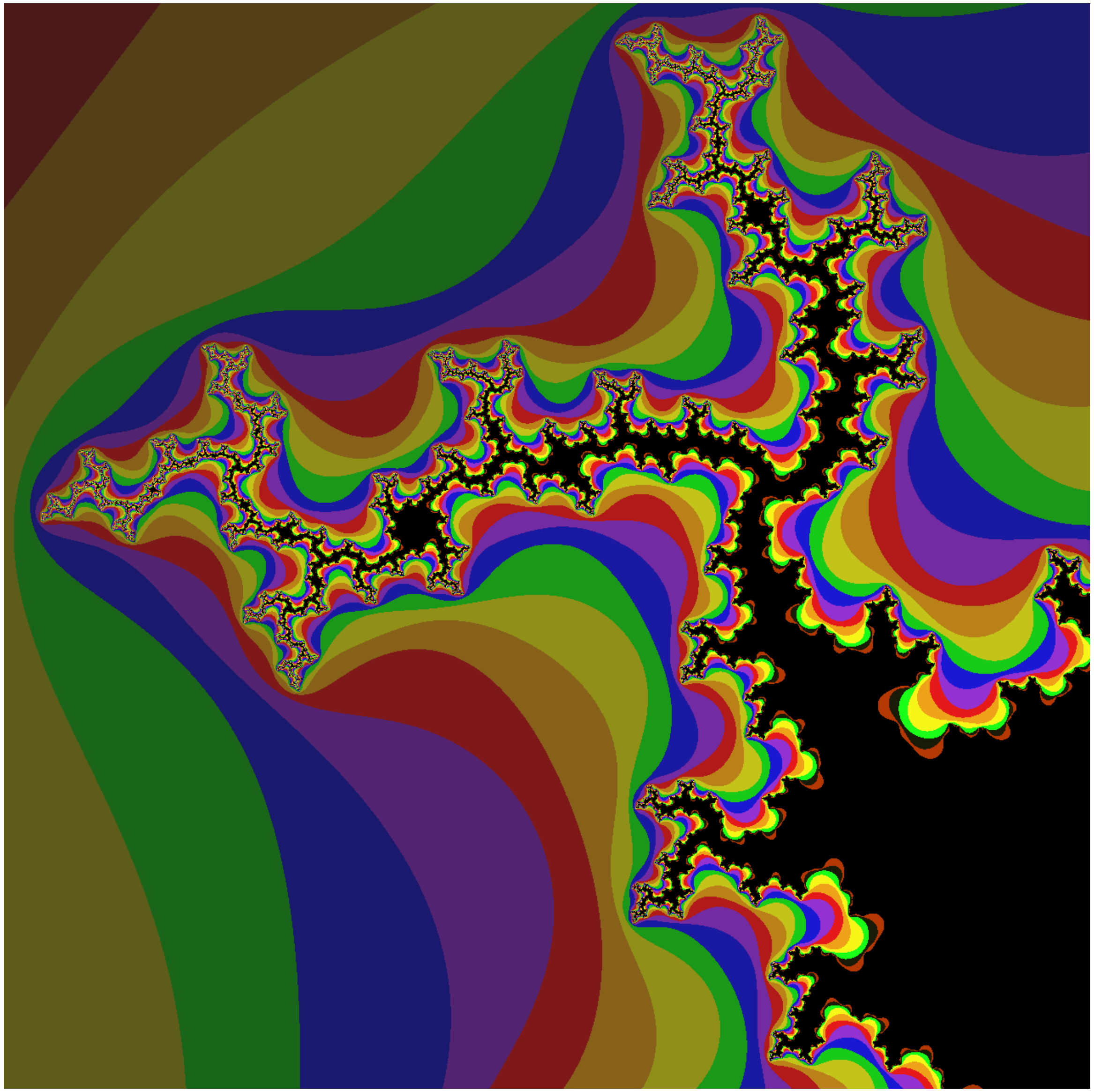


Image courtesy of Aaron Gable, CS 5 Black

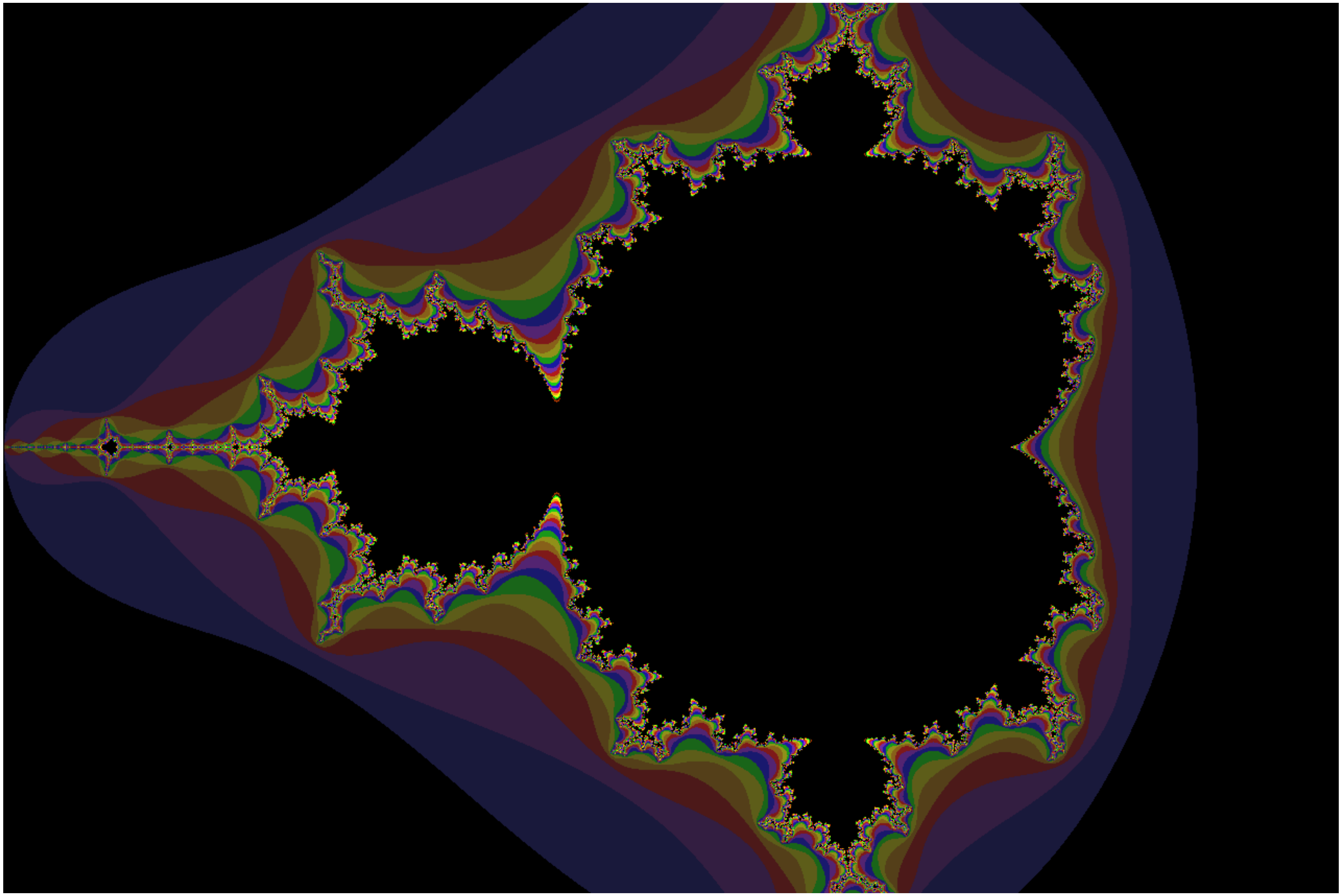
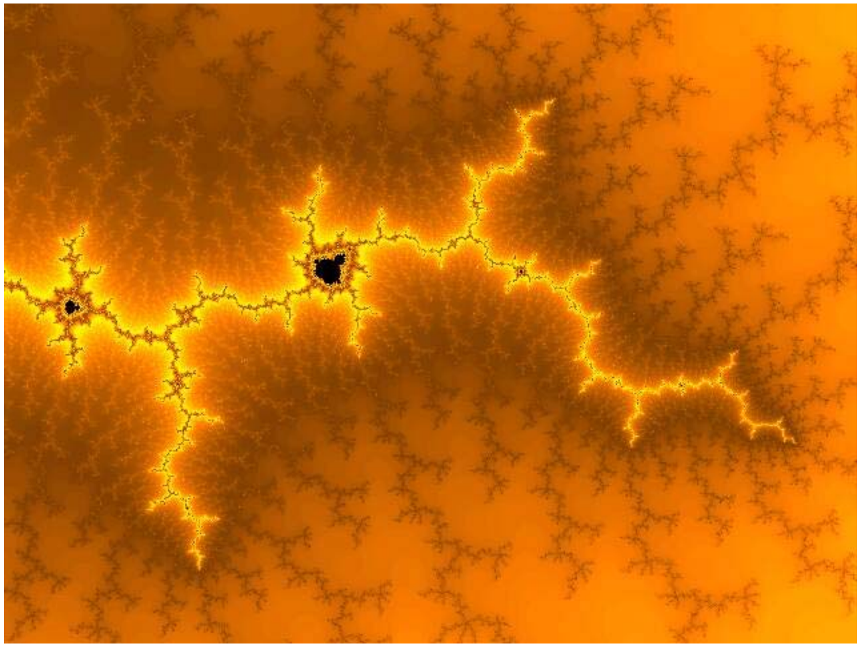
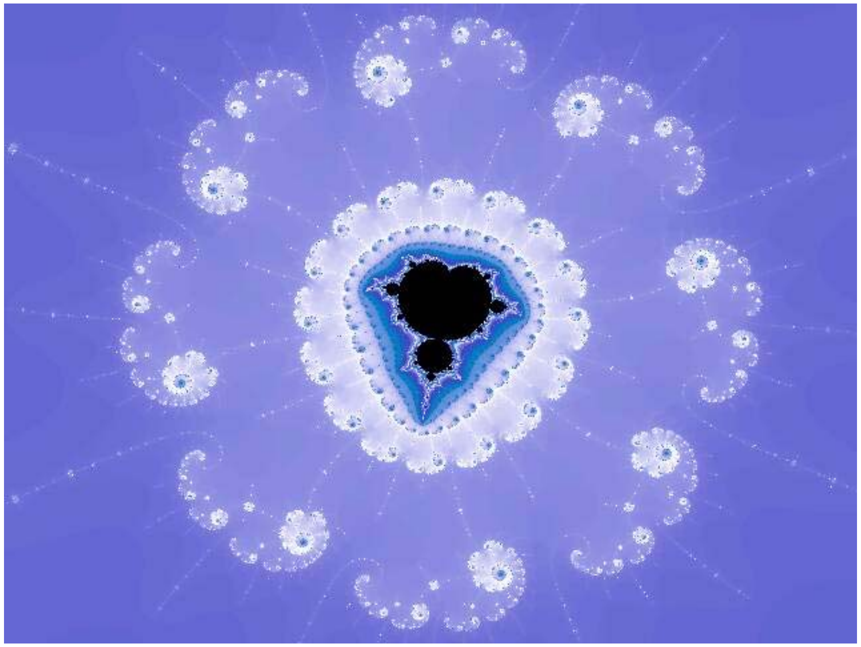
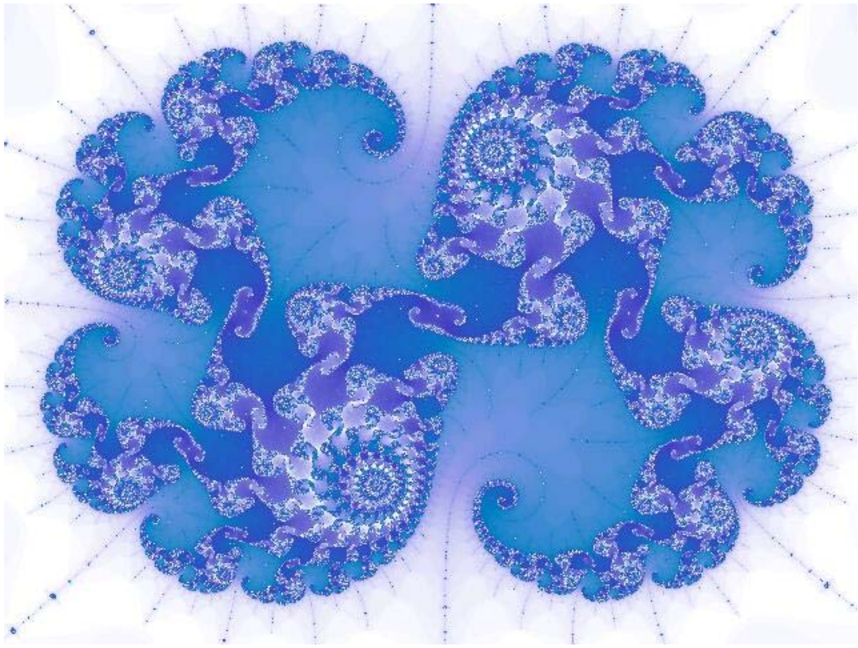
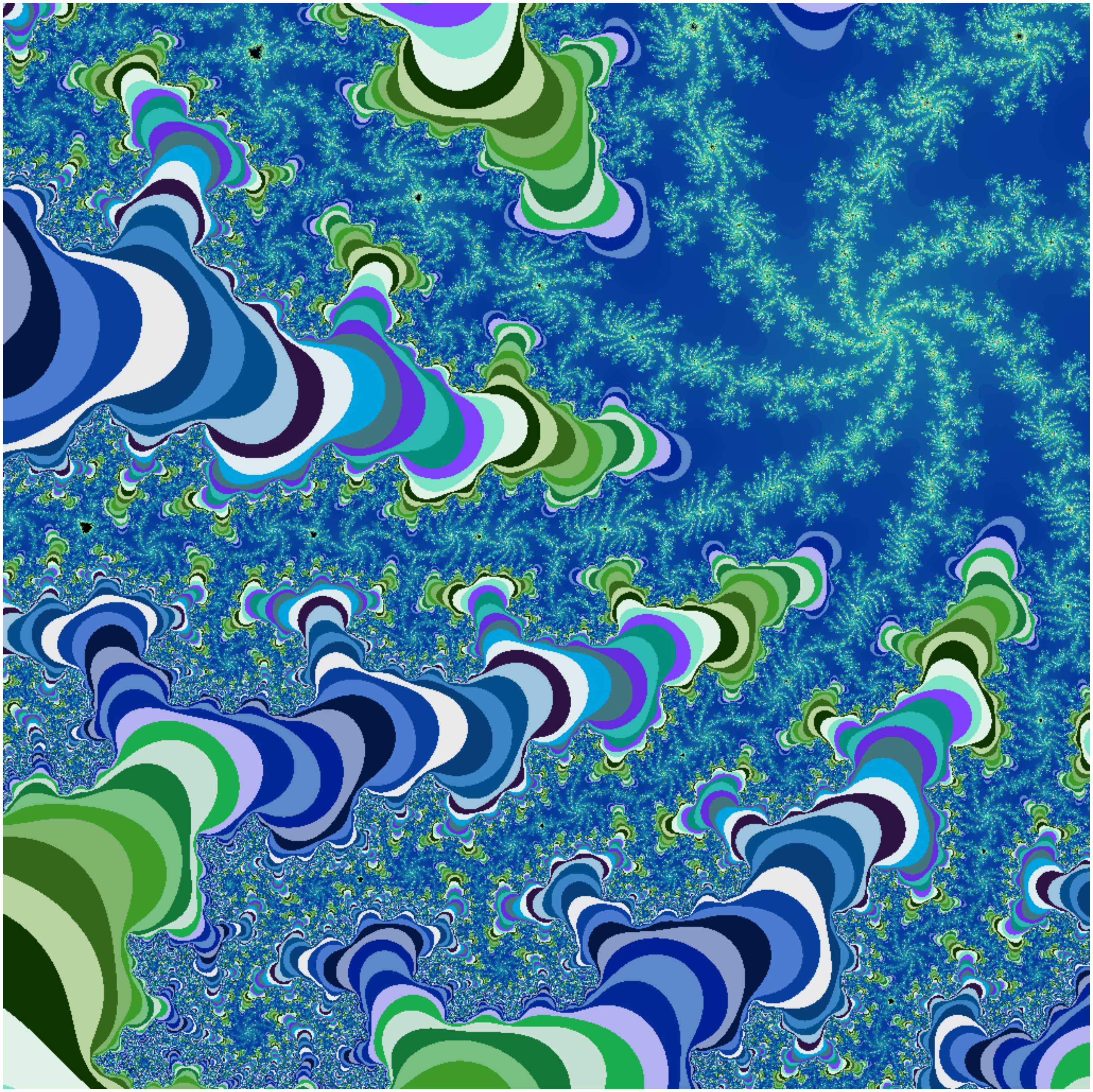


Image courtesy of Aaron Gable, CS 5 Black









2-D “Arrays”

```
>>> A = [ [0, 0, 0, 1], [1, 1, 0, 0], [0, 0, 0, 1] ]
>>> A = [ [0, 0, 0, 1],
          [1, 1, 0, 0],
          [0, 0, 0, 1] ]
>>> A[0][3]
???
```

Shallow Copy

```
>>> A = [1, 2, 3, 4]
>>> B = A
>>> B[0] = 42
>>> A[0]
???

def f():
    L = [1, 2, 3, 4]
    g(L)
    return L

def f(List):
    List[0] = 42
```

Deep Copy

```
def f():
    L = [1, 2, 3, 4]
    M = g(L)
    print(L)
    print(M)

def g(List):
    return map(lambda X: X+1, List)
```

Exercise

```
def f(L):
    '''Assume L is a list of at least 3 floats.
    Return a copy of L, changed as follows.
    Each element is the average of itself and the
    two adjacent elements.  But the first and last
    are unchanged.'''
```