

# mediaComp

## A multimedia library for Python 3

---

Department of Computer Science, Ball State University, Muncie, IN USA

Document based on Release v0.4.7

October 28, 2025

---

### About this document

This document provides a brief description of the signature and use of each function provided in the mediaComp library. It is organized into sections, based on the purpose of each function, with individual functions listed alphabetically within each section.

- [File & I/O functions](#)
- [Color functions](#)
- [Picture functions](#)
- [Shape functions](#)
- [Sound functions](#)

A [table of contents](#) (which lists all functions and methods) with clickable links is provided. Changes to functions and methods are documented in [Appendix A](#).

### Library description

MediaComp is a free and open-source multimedia library for Python 3 which enables the easy manipulation of images and sounds. It utilizes popular libraries to provide an abstraction of manipulating sounds, images, and colors, by abstracting the complexity into easy-to-use function calls.

### Library history

MediaComp is a conversion of the multimedia library originally developed and released by Mark Guzdial & Barbara Ericson for use with their *Introduction to Computing and Programming in Python: A multimedia Approach* book (last edition ISBN 978-0-13-402554-4). It is based on Python 3, whereas the Guzdial/Ericson (G/E) version was based on Jython 2.7. Our implementation is based on Gordon College's [IES4py](#) conversion, which implemented a subset of the original G/E multimedia library

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

features so the library could be used in Python 3 code. Their goal was, and ours is, to provide the pedagogical assets of the G/E multimedia library without requiring Jython or needing to use the JES IDE. While mediaComp is still a subset of the G/E implementation, it provides added functionality when compared to JES4py. We have implemented a few changes, when compared to the original implementation of the G/E library. These are documented in [Appendix B](#).

MediaComp is a conversion of only the media computation multimedia library, and does not include [JES](#), the Jython Environment for Students, which is an educational IDE used in the Media Computation curriculum developed by Guzdial and Ericson. More details on the curriculum are available at <http://www.mediacomputation.org/>.

## Documentation syntax conventions

The following visual conventions are used in this document.

### Function signatures

- Optional parameters and punctuation are enclosed in brackets [ ].

### Description

- When referencing a parameter, the parameter is shown in *italics*.

### Example code

- Example code is displayed in `Courier` font.

# Table of Contents

About this document.....	1
Library description .....	1
Library history .....	1
Documentation syntax conventions .....	2
Table of Contents .....	3
File & I/O Functions .....	6
getMediaFolder([filename]).....	6
pickAFile() .....	6
pickAFolder() .....	7
requestInteger(message) .....	7
requestIntegerInRange(message) .....	7
requestNumber(message) .....	8
requestString(message).....	8
setLibFolder([directory]).....	9
setMediaFolder([directory]).....	9
showError(message) .....	10
showInformation(message).....	10
showWarning(message):.....	10
Color Functions .....	12
Default colors .....	12
distance(color1, color2).....	12
getBlue(pixel)  getGreen(pixel)  getRed(pixel).....	13
getColor(pixel).....	13
getColorWrapAround() .....	13
makeColor(red[, green, blue]) .....	14
makeDarker(color) .....	14
makeLighter(color).....	15
pickAColor().....	15
setAllPixelsToAColor(picture, color).....	15
setBlue(pixel, value)  setGreen(pixel, value)  setRed(pixel, value) .....	16
setColor(pixel, color).....	16

setColorWrapAround(flag).....	17
Picture Functions.....	18
addText(picture, xpos, ypos, text[, color]) .....	18
copyInto(smallPicture, bigPicture, startX, startY) .....	18
duplicatePicture(picture).....	19
getHeight(picture).....	19
getPixelAt(picture, xpos, ypos) .....	20
getPixels(picture) .....	20
getWidth(picture) .....	20
getX(pixel)   getY(pixel) .....	21
makeEmptyPicture(width, height[, color]).....	21
makePicture(path).....	22
pictureTool(picture).....	22
show(picture).....	23
writePictureTo(picture, path).....	23
Shape Functions .....	24
addArc(picture, startX, startY, width, height, start, angle[, color]).....	24
addArcFilled(picture, startX, startY, width, height, start, angle[, color]) .....	24
addLine(picture, startX, startY, endX, endY[, color]) .....	24
addOval(picture, startX, startY, width, height[, color]).....	25
addOvalFilled(picture, startX, startY, width, height[, color]) .....	25
addRect(picture, startX, startY, width, height[, color]).....	25
addRectFilled(picture, startX, startY, width, height[, color]) .....	25
Sound Functions.....	27
duplicateSound(sound).....	27
getDuration(sound).....	27
getNumSamples(sound) .....	27
getSampleAt(sound, index) .....	28
getSamples(sound) .....	28
getSampleValue(sample).....	29
getSampleValueAt(sound, index).....	29
getSamplingRate(sound).....	29

getSound(sample) .....	30
makeEmptySound(numSamples[, samplingRate]) .....	30
makeEmptySoundBySeconds(duration[, samplingRate]).....	31
makeSound(path).....	31
play(sound[, isBlocking]) .....	32
playNote(note, duration[, intensity]) .....	32
setSampleValue(sample, value).....	33
setSampleValueAt(sound, index, value).....	33
soundTool(sound).....	34
stopPlaying(sound).....	34
writeSoundTo(sound, path).....	34
Appendix A:.....	36
Changes since previous mediaComp versions.....	36
Release v0.4.7 Changes.....	36
blockingPlay(sound) .....	36
play(sound).....	36
Appendix B:.....	37
Changes to the original Guzdial/Ericson library .....	37
addLibPath([directory]) and setLibPath([directory]) .....	37
blockingPlay(sound) .....	37
explore(picture) .....	37
explore(sound) .....	37
getLength(sound).....	37
getMediaPath([filename]) .....	37
getPixel(picture, xpos, ypos) .....	37
makeBrighter(color) .....	37
play(sound).....	38
setMediaPath([directory]) .....	38
setMediaPath([directory]) .....	38

## File & I/O Functions

### getMediaFolder([filename])

Provides the full path (including the filename) to the file you specify by prefixing the path to the Media Folder. If no filename is given, only the path to the Media Folder will be returned. Usually used with the **setMediaFolder** function to set the place where you keep your media.

**filename:** the name of the file you want (optional)

**returns:** the full path to the media folder (and filename, if provided)

Example:

```
def getFullPathToMarysSong():
    setMediaFolder("C:\")
    return getMediaFolder("MarysSong.wav")
```

This sets the media folder to C:\ and will return the full path of MarysSong.wav ('C:\MarysSong.wav').

```
def getAndSetMediaFolder():
    setMediaFolder("C:\")
    return getMediaFolder()
```

This sets the media folder to C:\ and will then return it ('C:\').

### pickAFile()

Opens a file chooser dialog box to let the user pick a file and returns the complete path name as a string. Takes no input.

.

**returns:** the string path to the file chosen in the dialog box

Example:

```
def openAnyPicture():
    file = pickAFile()
    return makePicture(file)
```

This example opens a file chooser dialog box, then the user picks a file, and the defined function returns it as a picture object, by using the **makePicture** function.

## pickAFolder()

Opens a file chooser dialog box to let the user pick a folder and returns the complete path name as a string. Takes no input.

**returns:** the string path to the folder chosen in the dialog box

Example:

```
def showFolderContents():
    folder = pickAFolder()
    import os
    print(os.listdir(folder))
```

This opens a file chooser dialog, then the user picks a folder, and the function prints the contents of that folder.

---

## requestInteger(message)

This will open a dialog box to allow the user to input any integer. The value of *message* will be displayed in the dialog box. It will not accept the user's response unless it is an integer.

**message:** the message to display to the user in the dialog box

**returns:** the input integer

Example:

```
def getAge():
    age = requestInteger("What is your age?")
    return age
```

This will open a dialog box asking the user's age and then return the provided value.

---

## requestIntegerInRange(message)

This will open a dialog box to allow the user to input an integer. The value of *message* will be displayed in the dialog box. It will not accept the user's response unless it is an integer within the specified range, inclusive of end points.

**message:** the message to display to the user in the dialog box

**returns:** the input integer

Example:

```
def getGuess() :
    guess= requestIntegerInRange("What is your guess?", 1, 100)
    return guess
```

This will open a dialog box asking for the user's guess, forcing it to be between 1 and 100 inclusive, and then return the provided value.

---

## requestNumber(message)

This will open a dialog box to allow the user to input any numeric value. The value of *message* will be displayed in the dialog box. It will not accept the user's response unless it is an integer or float number.

**message:** the message to display to the user in the dialog box

**returns:** the input value as a float

Example:

```
def getGPA() :
    gpa = requestInteger("What is your GPA?")
    return gpa
```

This will open a dialog box, asking for the user's GPA, and then return the provided value.

---

## requestString(message)

This will open a dialog box to allow the user to input any string. The value of *message* will be displayed in the dialog box.

**message:** the message to display to the user in the dialog box

**returns:** the input string

Example:

```
def printName() :
    name = requestString("What is your name?")
    print("Hello " + name + " !")
```

This will open a dialog box, ask the user's name, and then print it back out.



---

## setLibFolder([directory])

Adds a directory where it will look for modules when an import statement is performed. If a directory is provided, it adds that directory. If none is provided, a file chooser will open for you to select a directory to be added.

**directory:** a string path to a directory. (optional)

**returns:** the full path specified

Example:

```
def addMediaFolderAsLibFolder() :  
    setLibFolder(getMediaFolder())
```

This will allow you to import .py files from the current media folder.

---

## setMediaFolder([directory])

Establishes the directory in which to search to find a file. You can omit the directory. If you do, it will open a file chooser dialog box to let you select a directory.

**directory:** The directory you want to set as the media folder. (optional)

**returns:** The full path to the media folder just specified.

Examples:

```
def openBobsSong() :  
    setMediaFolder()  
    return makeSound("BobsSong.wav")
```

This will let the user set a folder in which to look for files, open the file "BobsSong.wav" in the selected folder, and return a sound object.

```
def openMarysSong() :  
    setMediaFolder("C:\\music")  
    return makeSound("MarysSong.wav")
```

This sets the folder to look in "C:\\music", opens the file "MarysSong.wav" in that folder, and returns a sound object.

---

## showError(message)

Displays a message dialog box to the user, showing the value of *message*. An “X” icon is shown to the left of *message*, and “Error” is used as the dialog box title.

**message:** the message to show to the user

**returns:** nothing

Example:

```
showError("You must provide a response.")
```

This will show the provided message in a dialog box with an “X” icon.”

---

## showInformation(message)

Displays a message dialog box to the user, showing the value of *message*. An “i” icon is shown to the left of *message*, and “Information” is used as the dialog box title.

**message:** the message to show to the user

**returns:** nothing

Example:

```
def showIntroduction():  
    path = getMediaFolder("introText.txt")  
    string = readFile(path)  
    showInformation(string)
```

This will show the contents of the introText.txt file in a dialog box with an “i” icon.”

---

## showWarning(message):

Displays a message dialog box to the user, showing the value of *message*. An “!” icon is shown to the left of *message*, and “Warning” is used as the dialog box title.

**message:** the message to show to the user

**returns:** nothing

Example:

```
showWarning("This is your last chance.")
```

This will show the provided message in a dialog box with an “!” icon.”

---

# Color Functions

## Default colors

The following colors may be used without needing to define them. Note they do need to be used as shown, i.e. in all lower-case letters. The RGB value of each color is provided.

### COLOR: RGB

- black: 0, 0, 0
  - blue: 0, 0, 255
  - cyan: 0, 255, 255
  - darkGray: 64, 64, 64
  - gray: 128, 128, 128
  - green: 0, 255, 0
  - lightGray: 192, 192, 192
  - magenta: 255, 0, 255
  - orange: 255, 200, 0
  - pink: 255, 175, 175
  - red: 255, 0, 0
  - white: 255, 255, 255
  - yellow: 255, 255, 0
- 

## distance(color1, color2)

Takes two Color objects and returns a single number representing the distance between the colors. The red, green, and blue values of the colors are taken as a point in (x, y, z) space, and the Cartesian distance is computed. The smaller the returned value, the closer the two colors are.

**color1:** the first color you want compared

**color2:** the second color you want compared

**returns:** a floating-point number representing the Cartesian distance between the colors

Example:

```
def showDistRedAndBlue():
    red = makeColor(255, 0, 0)
    blue = makeColor(0, 0, 255)
    print(distance(red, blue))
```

This will print the distance between red and blue, 360.62445840513925.

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

---

## getBlue(pixel)   getGreen(pixel)   getRed(pixel)

Takes a Pixel object and returns the value (between 0 and 255) of the amount of redness / greenness / blueness in that pixel.

**pixel:** the pixel you want to get the amount of red/green/blue from

**returns:** the red / green / blue value of the pixel

Example:

```
def getHalfRed(pixel):  
    red = getRed(pixel)  
    return red // 2
```

This takes in a pixel and returns the amount of red in that pixel divided by two.

---

## getColor(pixel)

Takes a Pixel and returns the Color object of that pixel.

**pixel:** the pixel you want to extract the color from

**returns:** the color of the pixel

Example:

```
def getDistanceFromRed(pixel):  
    red = makeColor(255, 0, 0)  
    color = getColor(pixel)  
    return distance(red, color)
```

This takes in a pixel and returns that pixel's color's distance from red.

---

## getColorWrapAround()

Takes no input, and returns the current value of **ColorWrapAround**. If it is true, color values will wrap-around ( $356 \bmod 256 = 100$ ); if **false**, color values lower than 0 will be forced to 0 and higher than 255 forced to 255. Default is **false**.

**returns:** a Boolean (1/true or 0/false) for the current value of **ColorWrapAround**

Example:

```
def setColorWithoutWrap(pixel, r, g, b):
    oldWrapVal = getColorWrapAround()
    setColorWrapAround(0)
    setColor(pixel, makeColor(r, g, b))
    setColorWrapAround(oldWrapVal)
```

This will temporarily disable **colorWrapAround** (if enabled), change the pixel's color, and then restore the **colorWrapAround** to the initial value.

---

## makeColor(red[, green, blue])

Takes three integer inputs for the red, green, and blue components (in order) and returns a color object. If *green* and *blue* are omitted, the *red* value is used as the intensity of a gray color.

**red:** the amount of red you want in the color (or a Color object you want to duplicate)

**green:** the amount of green you want in the color (optional)

**blue:** the amount of blue you want in the picture (optional)

**returns:** the color object made from the inputs

Examples:

```
def makeCyan():
    return makeColor(0, 255, 255)
```

This will return the color cyan.

```
def makeMiddleGry():
    return makeColor(127)
```

This will return a medium gray color.

---

## makeDarker(color)

Takes a color object and returns a slightly darker version of the original color.

**color:** the color object you want to darken

**returns:** the new, darker color

Example:

```
def makeMuchDarker(color):
    return makeDarker(makeDarker(makeDarker(color)))
```

Takes in a color object and returns a much darker version of it by calling **makeDarker** three times.

---

## makeLighter(color)

Takes a color object and returns a slightly lighter version of the original color.

**color:** the color object you want to lighten

**returns:** the new, lighter color

Example:

```
def makeMuchLighter(color):
    return makeLighter(makeLighter(makeLighter(color)))
```

Takes in a color object and returns a much lighter version of it by calling **makeLighter** three times.

---

## pickAColor()

Opens a color chooser dialog box to let the user pick a color and returns it. Takes no input.

**returns:** the color object chosen in the dialog box

Example:

```
def makeColoredPicture():
    color = pickAColor()
    return makeEmptyPicture(100, 100, color)
```

This opens a color selector dialog box, the user picks a color, and the function returns a 100 x 100 picture of the chosen color.

---

## setAllPixelsToAColor(picture, color)

Modifies the whole image so that every pixel in that image is the given color.

**picture:** the picture to change the pixels of

**color:** the color to set each pixel to

**returns:** nothing

Example:

```
def makeColoredPicture(picture):
    color = pickAColor()
    setAllPixelsToAColor(picture, color)
```

This accepts a picture, opens a color selector dialog box, the user picks a color, and the function sets all pixels in the picture to the chosen color.

---

## setBlue(pixel, value) setGreen(pixel, value) setRed(pixel, value)

Takes in a Pixel object and a value (between 0 and 255) and sets the corresponding color component value of that pixel to the provided value.

**pixel:** the pixel you want to set the color component value of

**value:** a number (0 - 255) for the new color component value of the pixel

Example:

```
def zeroRed(pixel):
    setRed(pixel, 0)
```

This will take in a pixel and set its amount of red to zero.

---

## setColor(pixel, color)

Takes in a pixel and a color, and sets the pixel to the given color.

**pixel:** the pixel you want to set the color of

**color:** the color you want to set the pixel to

**returns:** nothing

Example:

```
def makeMoreBlue(pixel):
    myBlue = getBlue(pixel) + 60
    newColor = makeColor(getRed(pixel), getGreen(pixel), myBlue)
    setColor(pixel, newColor)
```

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>



This will take in a pixel and increase its level of blue by 60.

---

## setColorWrapAround(flag)

Takes a Boolean as input. If *flag* is **true**, color values will wrap-around ( $356 \bmod 256 = 100$ ); if **false**, color values lower than 0 will be forced to 0 and higher than 255 forced to 255. Default is **false**.

**flag:** a Boolean (1/true or 0/false) for the new **ColorWrapAround** value

**returns:** nothing

Example:

```
def setColorWithoutWrap(pixel, r, g, b):  
    oldWrapVal = getColorWrapAround()  
    setColorWrapAround(0)  
    setColor(pixel, makeColor(r, g, b))  
    setColorWrapAround(oldWrapVal)
```

This will temporarily disable **colorWrapAround** (if enabled), change the pixel's color, and then restore the **colorWrapAround** to the initial value.

---

## Picture Functions

### `addText(picture, xpos, ypos, text[, color])`

Takes a picture, an x position and a y position (two numbers), and some text as a string, which will get drawn into the picture, in the specified color. Default color is black.

**picture:** the picture you want to add the text to

**xpos:** the x-coordinate where you want to start writing the text

**ypos:** the y-coordinate where you want to start writing the text

**text:** a string containing the text you want written

**color:** the color you want to draw in (optional)

**returns:** nothing

Examples:

```
def addDate(picture):
    str = "Today is the first day of the rest of your life."
    addText(picture, 0, 0, str)
```

This takes in a picture and adds a black date stamp to the upper left corner.

```
def addHappyBirthday(picture):
    str = "Happy Birthday!!!"
    addText(picture, 0, 0, str, orange)
```

This takes a picture and adds an orange Happy Birthday message to the upper left corner.

### `copyInto(smallPicture, bigPicture, startX, startY)`

Takes two pictures, an x position and a y position as input, and modifies *bigPicture* by copying as much of *smallPicture* as will fit into it, starting at the *startX*, *startY* position in the destination picture.

**smallPicture:** the picture to paste into the big picture

**bigPicture:** the picture to be modified

**startX:** the X coordinate of where to place the small picture on the big one

**startY:** the Y coordinate of where to place the small picture on the big one

**returns:** nothing

Example:

```
def addBorder(picture, borderWidth, borderHeight, borderColor):
    width = getWidth(picture) + 2 * borderWidth
    height = getHeight(picture) + 2 * borderHeight
    canvas = makeEmptyPicture(width, height, borderColor)
    copyInto(picture, canvas, borderWidth, borderHeight)
    return canvas
```

This will take a picture and return a new picture which is the original plus a border of specified size and color.

---

## duplicatePicture(picture)

Takes a picture as input and returns a new picture object with the same image as the original.

**pic:** the picture that you want to duplicate

**returns:** a new picture object with the same image as the original

Example:

```
file = pickAFile()
mypic = makePicture(file)
copy_of_mypic = duplicatePicture(mypic)
```

This opens a file chooser, makes a Picture object using the chosen file, and then creates a second Picture object by copying the first one.

---

## getHeight(picture)

Takes a picture as input and returns its height in the number of pixels top-to-bottom in the picture.

**picture:** the picture you want to get the height of

**returns:** the height of the picture

Example:

```
def howTall(picture)
    height = getHeight(picture)
    print("The picture is " + str(height) + " pixels tall.")
```

This takes a picture and prints a descriptive message about its height.

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

---

## getPixelAt(picture, xpos, ypos)

Takes a picture, an x position and a y position (two numbers), and returns the Pixel object at that point in the picture.

**picture:** The picture you want to get the pixel from

**xpos:** the x-coordinate of the pixel you want

**ypos:** the y-coordinate of the pixel you want

**returns:** a Pixel object

Example:

```
def getUpperLeftPixel(picture)
    return getPixelAt(picture, 0, 0)
```

This will take in a picture and return the pixel in the upper left-hand corner, (0, 0).

---

## getPixels(picture)

Takes a picture as input and returns the sequence of Pixel objects in the picture.

**picture:** The picture you want to get the pixels from

**returns:** A list of all the pixels in the picture

Example:

```
def getTenthPixel(picture):
    pixels = getPixels(picture)
    return pixels[9]
```

This takes a picture and returns the 10<sup>th</sup> pixel in that picture.

---

## getWidth(picture)

Takes a picture as input and returns its width in the number of pixels left-to-right in the picture.

**picture:** The picture you want to get the width of

**returns:** The width of the picture

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Example:

```
def howWide(picture)
    width = getWidth(picture)
    print("The picture is " + str(width) + " pixels wide.")
```

This takes in a picture and prints a descriptive message about its width.

---

## getX(pixel)    getY(pixel)

Takes in a Pixel object and returns the x or y position of where that pixel is in the picture.

**pixel:** the pixel you want to find the x-coordinate or y-coordinate of

**returns:** the x-coordinate or y-coordinate of the pixel

Example:

```
def getHalfX(pixel):
    pos = getX(pixel)
    return pos // 2
```

This will take in a pixel and return half of its x-coordinate.

---

## makeEmptyPicture(width, height[, color])

Makes a new "empty" picture and returns it to you. The width and height must be between 0 and 10,000. Default color, if not provided, is white.

**width:** the width of the empty picture

**height:** the height of the empty picture

**color:** background color of the empty picture (optional)

**returns:** a new Picture object with all the pixels set to the specified color

Examples:

```
def makeEmptySquare(edge):
    return makeEmptyPicture(edge, edge)
```

This will take in an edge length and call **makeEmptyPicture** to return an empty white picture with all sides the same length.

```
def makeEmptyBlueSquare(edge):
    return makeEmptyPicture(edge, edge, blue)
```

This will take in an edge length and call **makeEmptyPicture** to return an empty blue picture with all sides the same length.

---

## makePicture(path)

Takes a filename (with path) as input, reads the file, and creates a Picture object from it.

**path:** the name and path of the file you want to open as a picture

**returns:** the picture object.

Example:

```
def makePictureSelector():
    file = pickAFile()
    return makePicture(file)
```

This function will open a file selector box and then return the Picture object made from that file.

---

## pictureTool(picture)

When given a Picture object, it displays it as an image. Detailed information about x and y coordinates and pixel color is provided. There is also a zoom option if it's too small or large.

**picture:** The Picture object you want to view.

**returns:** nothing

Example:

```
def exploreAPicture():
    file = pickAFile()
    pic = makePicture(file)
    pictureTool(pic)
```

This creates a Picture object from a file and opens it in the picture tool.

---

## show(picture)

Displays the provided Picture object on screen as an image.

**picture:** the Picture object you want to display

**returns:** nothing

Example:

```
def openShow():
    file = pickAFile()
    pic = makePicture(file)
    show(pic)
```

This will let the user choose which file to be shown and show it.

---

## writePictureTo(picture, path)

Takes a Picture object and a file name and path (string) as input, then writes the picture to the file as a JPEG, PNG, or BMP. (Be sure to end the filename in ".jpg" or ".png" or ".bmp" for the operating system to understand it well.)

**picture:** the picture you want to be written out to a file

**path:** the file name and path to the file you want the picture written to

**returns:** nothing

Example:

```
def writeTempPic(picture):
    file = r"C:\Temp\temp.jpg"
    writePictureTo(picture, file)
```

This takes in a picture and writes it to C:\Temp\temp.jpg.

---

## Shape Functions

`addArc(picture, startX, startY, width, height, start, angle[, color])`

`addArcFilled(picture, startX, startY, width, height, start, angle[, color])`

Takes a *picture*, (*x,y*) coordinates, *width*, *height*, two integer *angles*, and (optionally) a *color* as input. Adds an outline of an arc starting at (*x,y*) at an initial angle of *start* with the given *width* and *height*. The angle of the arc itself is *angle*, which is relative to *start*. Default color is black. The “filled” version draws a filled arc.

**picture:** the picture you want to draw the arc on  
**startX:** the x-coordinate of the center of the arc  
**startY:** the y-coordinate of the center of the arc  
**width:** the width of the arc **height:** the height of the arc  
**start:** the start angle of the arc in degrees  
**angle:** the angle of the arc relative to start in degrees  
**color:** the color you want to draw in (optional)  
**returns:** nothing

Examples:

```
def addBlueSemiCircle(picture, startX, startY, radius, start):
    addArc(picture, startX, startY, radius, radius, start, 180, blue)
```

This will take in a picture, start coordinates, a radius length, and a starting angle and call **addArc** to draw a blue arc with equal width and height.

`addLine(picture, startX, startY, endX, endY[, color])`

Takes a picture, a starting (*x, y*) position (two integers), and an ending (*x, y*) position (two integers), and (optionally) a color as input. Adds a line from the starting point to the ending point in the picture. Default color is black.

**picture:** the picture you want to draw the line on  
**startX:** the x position where you want the line to start  
**startY:** the y position where you want the line to start  
**endX:** the x position where you want the line to end  
**endY:** the y position where you want the line to end  
**color:** the color you want to draw in (optional)  
**returns:** nothing



Examples:

```
def drawDiagonal (picture):
    addLine (picture, 0, 15, 100, 115, blue)
```

This will take in a picture and draw a blue diagonal line on it from (0,15) to (100, 115)

---

## addOval (picture, startX, startY, width, height[, color])

## addOvalFilled (picture, startX, startY, width, height[, color])

Takes a picture, a starting (x, y) position (two integers), a width and height (two integers), and (optionally) a color as input. Adds an oval outline of the given dimensions using the (x, y) as the upper left corner of the bounding rectangle. Default color is black. The “filled” version draws a filled oval.

**picture:** the picture you want to draw the oval on

**startX:** the x-coordinate of the upper left corner of the bounding rectangle of the oval

**startY:** the y-coordinate of the upper left corner of the bounding rectangle of the oval

**width:** the width of the oval

**height:** the height of the oval

**color:** the color you want to draw in (optional)

**returns:** nothing

Examples:

```
def addBlueCircle (picture, startX, startY, radius):
    addOval (picture, startX, startY, radius, radius, blue)
```

This will take in a picture, start coordinates, and a radius length and call **addOval** to draw a blue oval with equal width and height.

---

## addRect (picture, startX, startY, width, height[, color])

## addRectFilled (picture, startX, startY, width, height[, color])

Takes a picture, a starting (x, y) position (two integers), a width and height (two integers), and (optionally) a color as input. Adds a rectangular outline of the specified dimensions using the (x, y) as the upper left corner. Default color is black. The “filled” version draws a filled rectangle.

**picture:** the picture you want to draw the rectangle on

**startX:** the x-coordinate of the upper left corner of the rectangle

**startY:** the y-coordinate of the upper left corner of the rectangle

**width:** the width of the rectangle

**height:** the height of the rectangle

**color:** the color you want to draw in (optional)

**returns:** nothing

Examples:

```
def addBlueSquare (picture, startX, startY, edge):  
    addRect (picture, startX, startY, edge, edge, blue)
```

This will take in a picture, start coordinates, and an edge length and call addRect to draw a blue rectangle with all sides the same length, that is, a square.

---

## Sound Functions

---

### duplicateSound(sound)

Takes a sound as input and returns a new Sound object with the same sample values as the original.

**sound:** the sound you want to duplicate

**returns:** a new Sound object with the same sample values as the original

Example:

```
file = pickAFile()
my_sound = makeSound(file)
copy_of_my_sound = duplicateSound(my_sound)
```

This opens a file chooser, makes a Sound object using the chosen file, and then creates a second Sound object by copying the first one.

---

### getDuration(sound)

Takes a sound as input and returns the number of seconds that sound lasts.

**sound:** the sound you want to find the length of (in seconds)

**returns:** the number of seconds the sound lasts

Example:

```
def songLength():
    sound = makeSound(r"C:\My Sounds\2secondsong.wav")
    print getDuration(sound)
```

This will print the number of seconds in the sound. For a song with 88,000 samples at 44kHz, it will print 2.0.

---

### getNumSamples(sound)

Takes a sound as input and returns the number of samples in that sound.

**sound:** the sound you want to find the length of (how many samples it has)

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

**returns:** the number of samples in sound

Example:

```
def songLength():
    sound = makeSound(r"C:\My Sounds\2secondsong.wav")
    print getNumSamples(sound)
```

This will print out the number of samples in the sound. For a 2 second song at 44kHz, it will print out 88,000.

---

## getSampleAt(sound, index)

Takes a sound and an index, and returns the Sample object at that point in the sound.

**sound:** The sound you want to get the sample from

**index:** the index of the sample you want

**returns:** a Sample object

Example:

```
def getFirstSample(sound):
    return getSampleAt(sound, 0)
```

This will take a sound and return the sample at index 0).

---

## getSamples(sound)

Takes a sound as input and returns the Samples in that sound.

**sound:** A Sound, the sound you want to extract the samples from

**returns:** A collection of all the samples in the sound

Example:

```
def printFirstTen(sound):
    samps = getSamples(sound)
    for num in range(0, 10):
        print samps[num]
```

This will take in a sound and print the first 10 samples in that sound

---

## getSampleValue(sample)

Takes a Sample object and returns its value (between -32,768 and 32,767).

**sample:** a sample of a sound

**returns:** the integer value of that sample

Example:

```
def sampleToInt(sample):  
    return getSampleValue(sample)
```

This will convert a sample object into an integer.

---

## getSampleValueAt(sound, index)

Takes a sound and an index (an integer), and returns the Sample object at that index.

**sound:** the sound you want to get the sample from

**index:** the index of the sample you want to get

**returns:** the sample object at that index

Example:

```
def getTenthSample(sound):  
    samp = getSampleValueAt(sound, 9)  
    return samp
```

This takes a sound, and returns the tenth sample in the sound.

---

## getSamplingRate(sound)

Takes a sound as input and returns the number representing the number of samples in each second for the sound.

**sound:** the sound you want to get the sampling rate from

**returns:** the integer value representing the number of samples per second

Example:

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

```
def getDoubleSamplingRate(sound):
    rate = getSamplingRate(sound)
    return (rate * 2)
```

This will take in a sound and return the sampling rate multiplied by 2.

---

## getSound(sample)

Takes a Sample object and returns the Sound that it belongs to.

**sample:** a sample belonging to a sound

**returns:** the sound the sample belongs to

Example:

```
def getSamplesFromAnySample(sample):
    sound = getSound(sample)
    return getSamples(sound)
```

This will take in a sample and return the list of samples from the original sound.

---

## makeEmptySound(numSamples[, samplingRate])

Takes one or two integers as input. Returns an empty Sound object with the given number of samples and (optionally) the given sampling rate. Default sampling rate is 22,050 samples/second. The resulting sound must not be longer than 600 seconds. Prints an error statement if *numSamples* or *samplingRate* are less than 0, or if  $(numSamples/samplingRate) > 600$ .

**numSamples:** the number of samples in the sound

**samplingRate:** the integer value representing the number of samples per second (optional)

**returns:** an empty sound with the given number of samples and sampling rate

Examples:

```
def make10SecondSound():
    return makeEmptySound(10 * 22050)
```

This will return an empty sound lasting 10 seconds, using the default sampling rate (22,050 bits/second).

```
def make10SecondSoundWithSamplingRate(samplingRate):
    return makeEmptySound(10 * samplingRate, samplingRate)
```

This will return an empty sound lasting 10 seconds, using the provided sampling rate.

---

## makeEmptySoundBySeconds(duration[, samplingRate])

Takes a floating-point number and optionally an integer as input. Returns an empty Sound object of the given duration and (optionally) the given sampling rate. Default rate is 22,050 bits/second. If the given arguments do not multiply to an integer, the number of samples is rounded up. Prints an error statement if *duration* or *samplingRate* are less than 0, or if duration > 600.

**duration:** the time in seconds for the duration of the sound

**samplingRate:** the integer value representing the number of samples per second of sound (optional)

**returns:** An Empty Sound.

Examples:

```
def make10SecondSoundWithSamplingRate(samplingRate):
    return makeEmptySoundBySecons(10, samplingRate)
```

This will return an empty sound lasting 10 seconds, using the provided sampling rate.

---

## makeSound(path)

Takes a filename as input, reads the file, and creates a sound from it.

**path:** a string path to a wav file

**returns:** the Sound object created from the file at the given path

Example:

```
def openAnySound():
    file = pickAFile()
    return makeSound(file)
```

This opens a file selector dialog box. The user picks a file, and the function returns it as a Sound object.

---

## play(sound[, isBlocking])

Plays a sound provided as input. If *isBlocking* is set to *True*, execution of the code will stop until the sound finishes playing. If *isBlocking* is not provided the default value is *False*, which means it will not stop execution of the code.

**sound:** the sound you want to be played

**isBlocking:** Boolean value; **True** means blocking (optional)

**returns:** nothing

Example:

```
def playAnySound():
    file = pickAFile()
    sound = makeSound(file)
    play(sound)
```

This will open a file selector dialog box, make a sound from the chosen file, and then play that sound.

Example:

```
def playSoundTwice():
    sound = makeSound(r"C:\My Sounds\preamble.wav")
    play(sound, True)
    play(sound)
```

This will play the preamble.wav sound twice, back-to-back, returning control to the code after starting to play it the second time.

## playNote(note, duration[, intensity])

Plays the requested note. Default intensity is 64.

**note:** the MIDI note number, from 0 to 127 (60 = Middle C) you want to be played

**duration:** the duration you want the note to be played in milliseconds

**intensity:** the intensity (a number between 0 and 127) you want the note to be played (optional)

**returns:** nothing

Example:

```
def playScale():
    dur = 1000
    playNote(60, dur)
    playNote(62, dur)
```



This will play two one-second notes with default intensity.

---

## setSampleValue(sample, value)

Takes a Sample object and a value (should be between -32,768 and 32,767), and sets the sample to that value.

**sample:** the Sound sample you want to change the value of

**value:** the value you want to set the sample to

**returns:** nothing

Example:

```
def setTenthSample(sound, value):
    samp = getSampleValueAt(sound, 9)
    setSampleValue(samp, value)
```

This takes a sound and an integer, and then sets the value of the tenth sample in the sound to the provided value.

---

## setSampleValueAt(sound, index, value)

Takes a sound, an index, and a value (should be between -32,768 and 32,767), and sets the value of the sample at the given index in the given sound to the given value.

**sound:** the sound you want to change a sample in

**index:** the index of the sample you want to set

**value:** the value you want to set the sample to

**returns:** nothing

Example:

```
def setTenthToTen(sound):
    setSampleValueAt(sound, 9, 10)
```

This takes in a sound and sets the value of the tenth sample to 10.

---

## soundTool(sound)

When given a Sound object, it displays the waveform representation of the sound. Detailed information about sample index and value is provided. There is also a zoom option if it is too small or large.

**sound:** The Sound object you want to view.

**returns:** nothing

Example:

```
def exploreASound():
    file = pickAFile()
    sound = makeSound(file)
    soundTool(sound)
```

This creates a Sound object from a selected file and opens it in the sound tool.

---

## stopPlaying(sound)

Stops a sound that is currently playing.

**sound:** the sound that you want to stop playing

**returns:** nothing

Example:

```
def playAndStop():
    sound = makeSound(r"C:\My Sounds\preamble.wav")
    play(sound)
    stopPlaying(sound)
```

This will start playing preamble.wav and stop it immediately after it starts.

---

## writeSoundTo(sound, path)

Takes a sound and a filename (a string) and writes the sound to that file as a WAV file. (Make sure that the filename ends in '.wav' if you want the operating system to treat it right.)

**sound:** the sound you want to write out to a file

**path:** the path to the file you want the sound written to

This work © 2024-2025 by David L. Largent is licensed under CC BY-NC-SA 4.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

**returns:** nothing

Example:

```
def writeTempSound(sound):  
    writeSoundTo(sound, 'C:\\temp\\temp.wav')
```

This takes in a sound and writes it out to a file called temp.wav in C:\temp\.

---

## Appendix A:

### Changes since previous mediaComp versions

The following functions/methods are those that have been renamed, modified, or eliminated in the *mediaComp* library since previous releases of the library.

#### Release v0.4.7 Changes

##### `blockingPlay(sound)`

Eliminated. Use the new optional parameter of the **play()** function to provide the same functionality.

##### `play(sound)`

Enhanced by adding an optional parameter (*isBlocking*) to provide the ability to have the function block execution until the sound has finished playing. This eliminated the need for **blockingPlay()**.

## Appendix B:

### Changes to the original Guzdial/Ericson library

The following functions/methods are those that have been renamed, modified, or eliminated in the *mediaComp* library that were part of the original Mark Guzdial and Barbara Ericson library, as updated by the Gordon College folk, which they published as *JES4py*.

`addLibPath([directory])` and `setLibPath([directory])`

Renamed for clarity. Use **setLibFolder([directory])** for the same functionality.

`blockingPlay(sound)`

Eliminated. Use the new optional parameter of the **`play()`** function to provide the same functionality.

`explore(picture)`

Eliminated in favor of the existing **pictureTool(picture)** that provides the same functionality.

`explore(sound)`

Eliminated in favor of the existing **soundTool(sound)** that provides the same functionality.

`getLength(sound)`

Eliminated in favor of the existing **getNumSamples(sound)** that provides the same functionality.

`getMediaPath([filename])`

Eliminated in favor of the existing **getMediaFolder([filename])** that provides the same functionality.

`getPixel(picture, xpos, ypos)`

Eliminated in favor of the existing **getPixelAt(picture, xpos, ypos)** that provides the same functionality.

`makeBrighter(color)`

Eliminated in favor of the existing **makeLighter(color)** that provides the same functionality.

`play(sound)`

Enhanced by adding an optional parameter (*isBlocking*) to provide the ability to have the function block execution until the sound has finished playing. This eliminated the need for **blockingPlay()**.

`setMediaPath([directory])`

Eliminated in favor of the existing **setMediaFolder([directory])** that provides the same functionality.

`setMediaPath([directory])`

Eliminated in favor of the existing **setMediaFolder([directory])** that provides the same functionality.